Deep Learning for NLP

O curso

Módulos, avaliação e projeto

_ Este curso (Aprendizado Profundo para Processamento de Linguagem Natural (NLP)) explora as técnicas mais avançadas para processar e compreender a linguagem humana usando redes neurais profundas. Os alunos irão adquirir conhecimento prático e teórico sobre como aplicar modelos de aprendizado profundo a uma gama de tarefas de NLP, desde classificação de texto até tradução automática e geração de linguagem natural.

- Este curso oferece uma imersão completa no mundo do Aprendizado Profundo aplicado ao Processamento de Linguagem Natural. Os alunos sairão do curso com habilidades práticas para aplicar modelos avançados de aprendizado profundo em uma variedade de cenários de NLP, além de um entendimento sólido das tendências e desafios atuais nesse campo dinâmico.

O curso

Módulos, avaliação e projeto

_ Nosso curso está dividido em 10 módulos, a saber:

1. Introdução ao Aprendizado Profundo e NLP (Habernal)
2. Representação de Texto e Modelos de Linguagem (Habernal)
3. Modelos Seq2Seq de Classificação de Texto (CNN, RNNs) (Stanford + StatQuest)
4. Atenção em Modelos Seq2Seq e Processamento de sequências com entradas longas (StatQuest)
5. Arquiteturas Transformer (StatQuest)
6. Pré-Treinamento e LLMs (Stanford)
7. Reinforcement Learning from Human Feddback (Stanford)
8. Modelos de Linguagem → Modelos de Mundo (Stanford)

O curso

Módulos, avaliação e projeto

_ Teremos as seguintes avaliações:

- Duas provas: 20 pontos cada
- Dois trabalhos práticos: 10 pontos cada
- Projeto de tema livre: 40 pontos

O curso

Módulos, avaliação e projeto

_ Sobre o projeto:

- Atividade que começou … agora!
- O tema é livre:
  - A escolha apropriada faz parte do projeto em si
  - Um tema apropriado deve levar em conta aspectos como sofisticação, relevância e potencial de resultados
  - O professor pode te orientar
- Não há entregáveis intermediários:
  - Apenas a entrega final, que ocorrerá próximo ao encerramento da disciplina
  - A entrega final compreende:
    - Um notebook com a implementação → reprodutibilidade
    - Um vídeo no Youtube mostrando sua capacidade de explicar e sintetizar o que foi feito, em cerca de 5 minutos

Deep Learning

Review

_ Basics

- Residual Connections
- Layer Normalization
- Batch Normalization
- Dropout
- Sigmoid, ReLU, GELU, Softmax
- MLPs, Convolutions

Deep Learning

Review

_ Basics

_ Losses

- Entropy
- Cross Entropy
- KL Divergence
- l2 Regularization

Deep Learning

Review

_ Basics

_ Losses

_ Optimizers

- Stochastic Gradient Descent
- Adam
- AdamW
- Learning Rate Schedules

Deep Learning

Review

_ Basics

_ Losses

_ Optimizers

_ Datasets

- Glue and Superglue
- Squad
- Librespeech
- HuggingFace Datasets

Deep Learning

Basics

_ Perceptrons

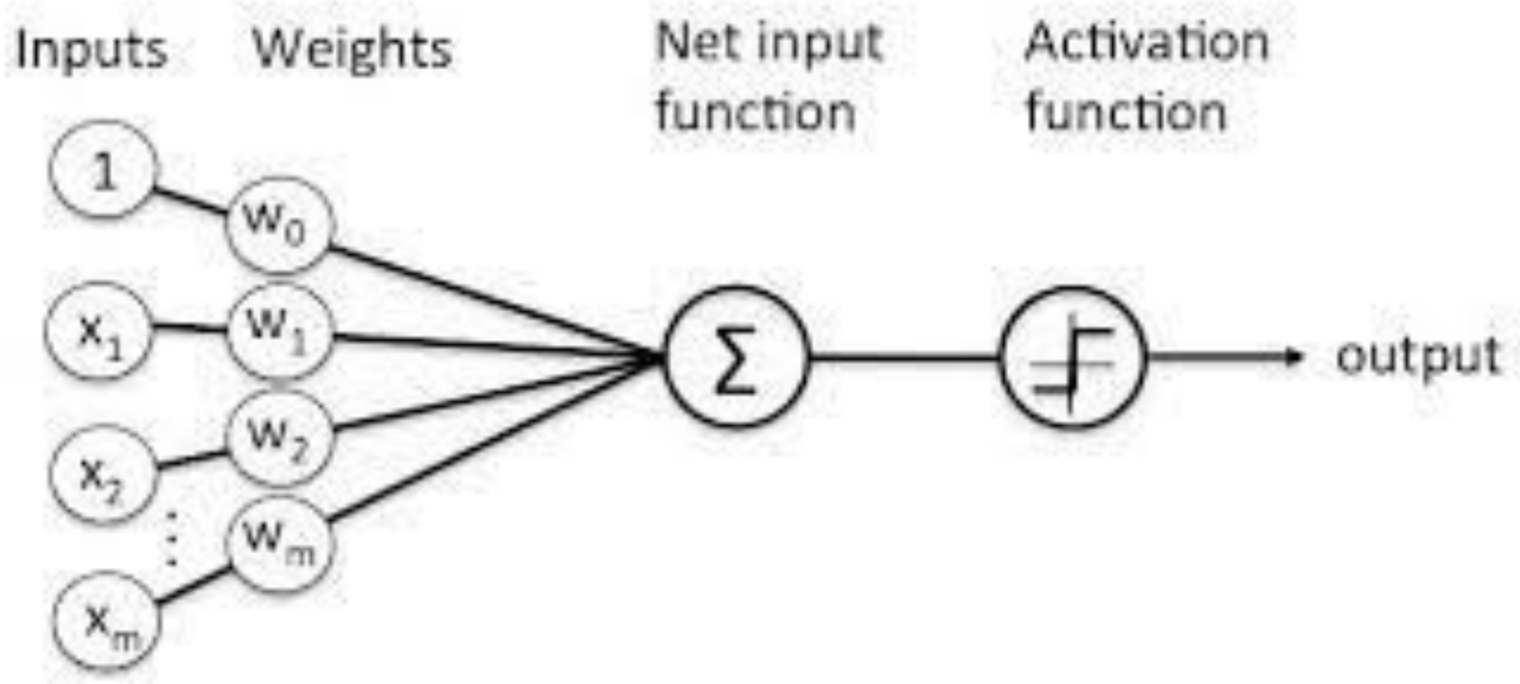
- A perceptron is a fundamental concept in machine learning. It consists of inputs, weights, a summation function, an activation function, and an output.
  - Inputs:
    - Inputs represent features or signals that the perceptron processes → $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, ..., $\boldsymbol{x_n}$.
  - Weights:
    - Each input is associated with a weight → $\boldsymbol{w_1}$, $\boldsymbol{w_2}$, ..., $\boldsymbol{w_n}$.
    - Weights determine the importance of each input in the computation.
  - Summation Function:
    - The weighted sum of inputs and weights is calculated → $(\boldsymbol{x_1} * \boldsymbol{w_1}) + (\boldsymbol{x_2} * \boldsymbol{w_2}) + \ldots + (\boldsymbol{x_n} * \boldsymbol{w_n})$.
  - Activation Function:
    - The sum is passed through an activation function → non-linearity and determines whether the perceptron should "fire" or not.

Deep Learning

Basics

_ Perceptrons

- A perceptron is a fundamental concept in machine learning. It consists of inputs, weights, a summation function, an activation function, and an output.
  - Output:
    - The output of the activation function serves as the perceptron's final output.
  - Learning
    - Perceptrons can be trained using algorithms like the Perceptron Learning Rule.
      - They adjust weights to better classify or predict data based on provided training examples.

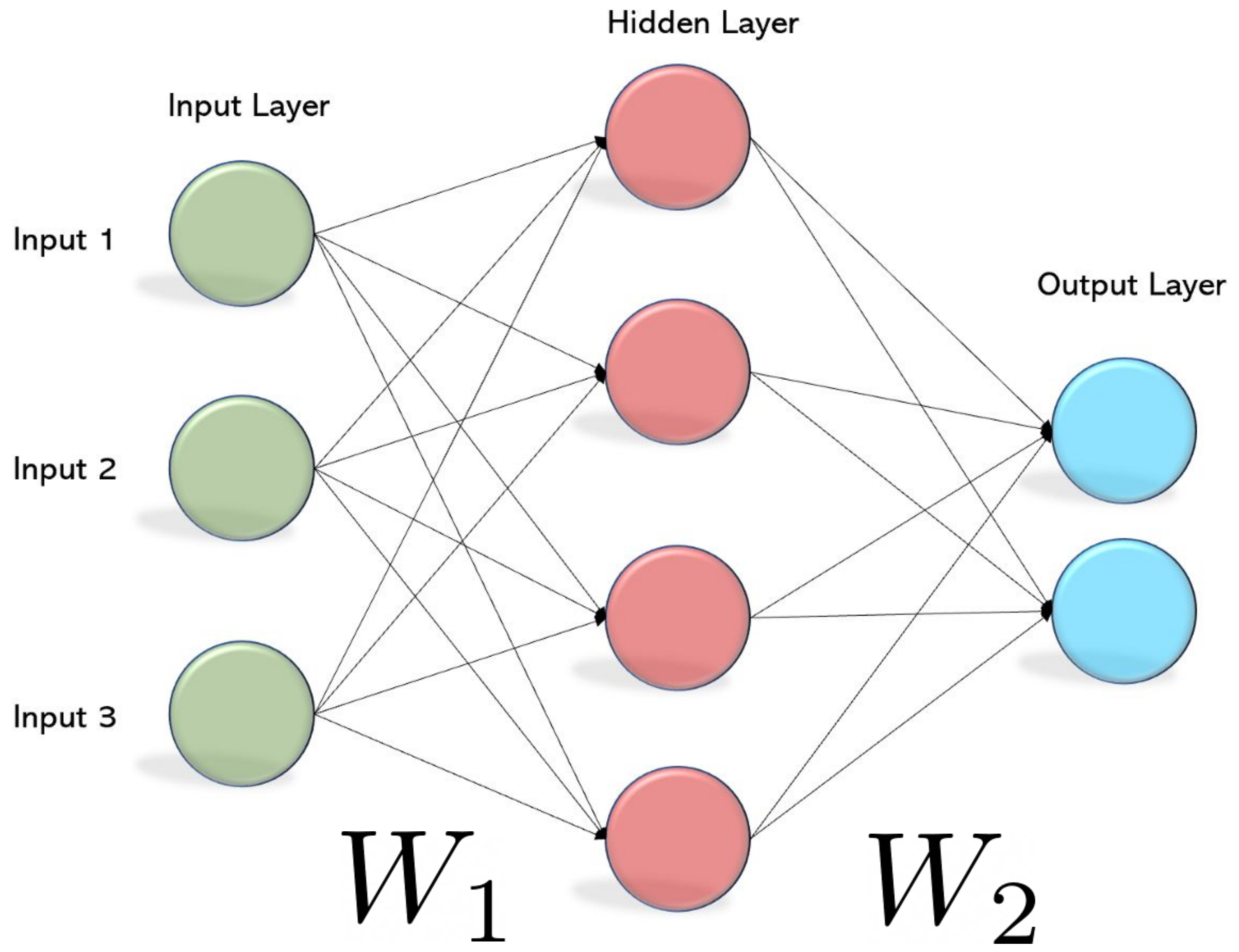


Deep Learning

Basics

_ Multilayer Perceptrons (MLPs)

- MLPs are usually weight matrices $\boldsymbol{W_i}$, activations ϕ and input $\boldsymbol{x}$ composed together



$$W_2\phi(W_1x)$$

Deep Learning

Basics

_ Softmax

- Converts outputs into a probability distribution

The softmax vector:
1. sums to one
2. nonnegative values

$$\mathrm{softmax} : \mathbb{R}^k \to (0,1)^k$$

$$\mathrm{softmax}_i(x_1, x_2, \ldots, x_k) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

  - Frequently used for classifying k outputs

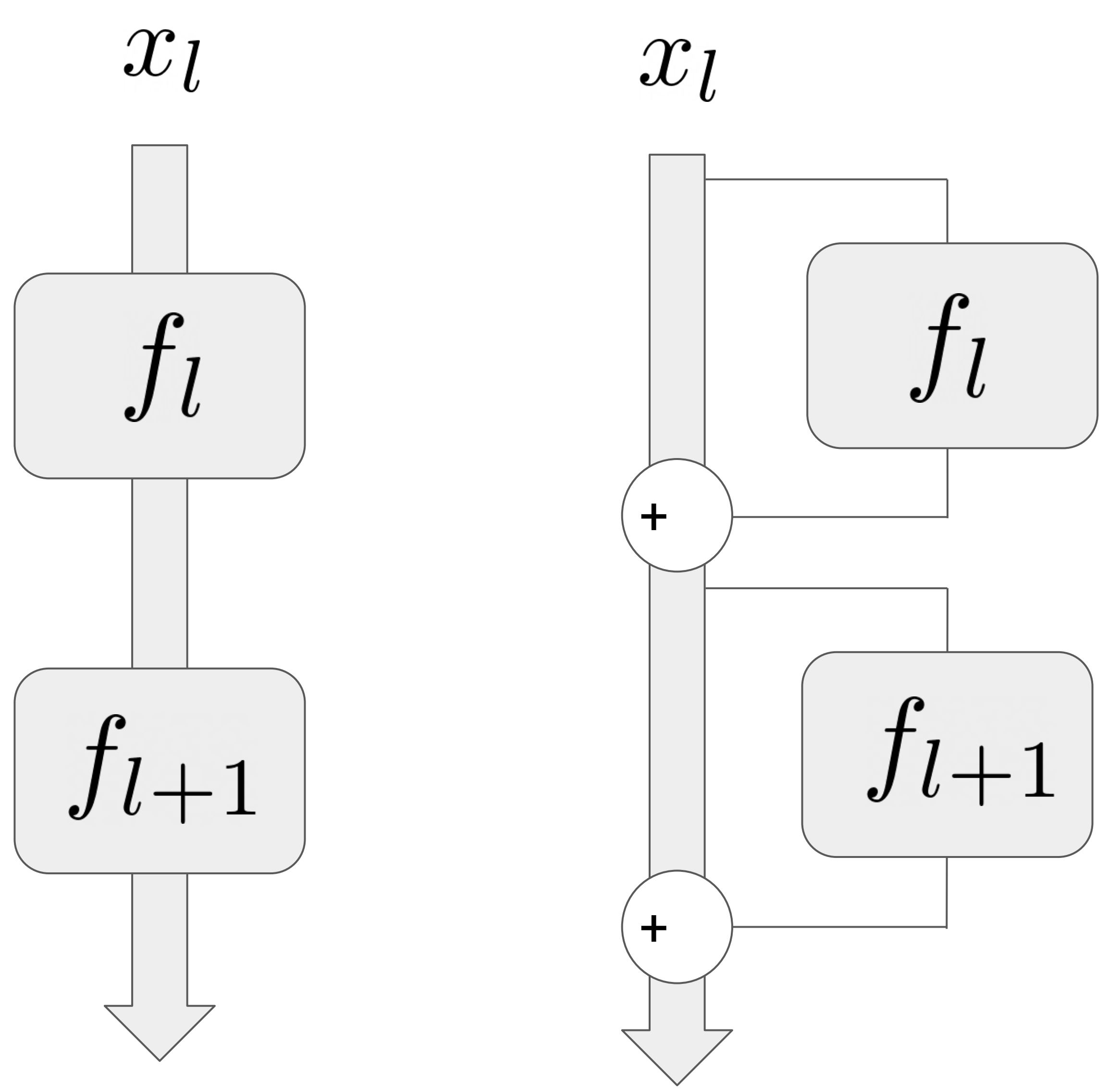
Deep Learning

Basics

_ Residual Connections

- Some neural network building blocks are useful irrespective of the problem setting, including residual connections
  - Let layer $\boldsymbol{l}$ be $\boldsymbol{f_l}$ and its input be $\boldsymbol{x_l}$
    - Usual layer outputs are $\mathbf{a^{(l)}} = \boldsymbol{f_l(x_l)}$
  - With residuals, $\mathbf{a^{(l)}} = \boldsymbol{f_l(x_l) + x_l}$

  - Residual connections appear to aid optimization



Deep Learning

Basics

_ Layer Normalization (1/2)

- Layer normalization reduces the chance of the feedforward signal's magnitude from blowing up or decaying
  - First assume $\boldsymbol{a^{(l)}}$ is in R^H is a vector representing the values of layer $\boldsymbol{l}$
  - Layer Norm will first standardize $\boldsymbol{a^{(l)}}$ by making it have 0 mean and a standard deviation of 1

$$\mu^{(l)} = \frac{1}{H}\sum_{i=1}^{H} a_i^{(l)} \qquad \sigma^{(l)} = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^{(l)} - \mu^{(l)})^2}$$

Deep Learning

Basics

_ Layer Normalization (2/2)

- Then Layer Normalizaton takes the standardized ***$a^{(l)}$*** and applies an affine transformation with a learned scale and shift vectors $\gamma, \beta \in \mathbb{R}^H$

learned parameters, does not depend on $a^{(l)}$

$$a_{\mathrm{LN}}^{(l)} = \frac{\gamma}{\sigma^{(l)}} \odot \left(a^{(l)} - \mu^{(l)}\right) + \beta$$

not learned parameters, depends on $a^{(l)}$

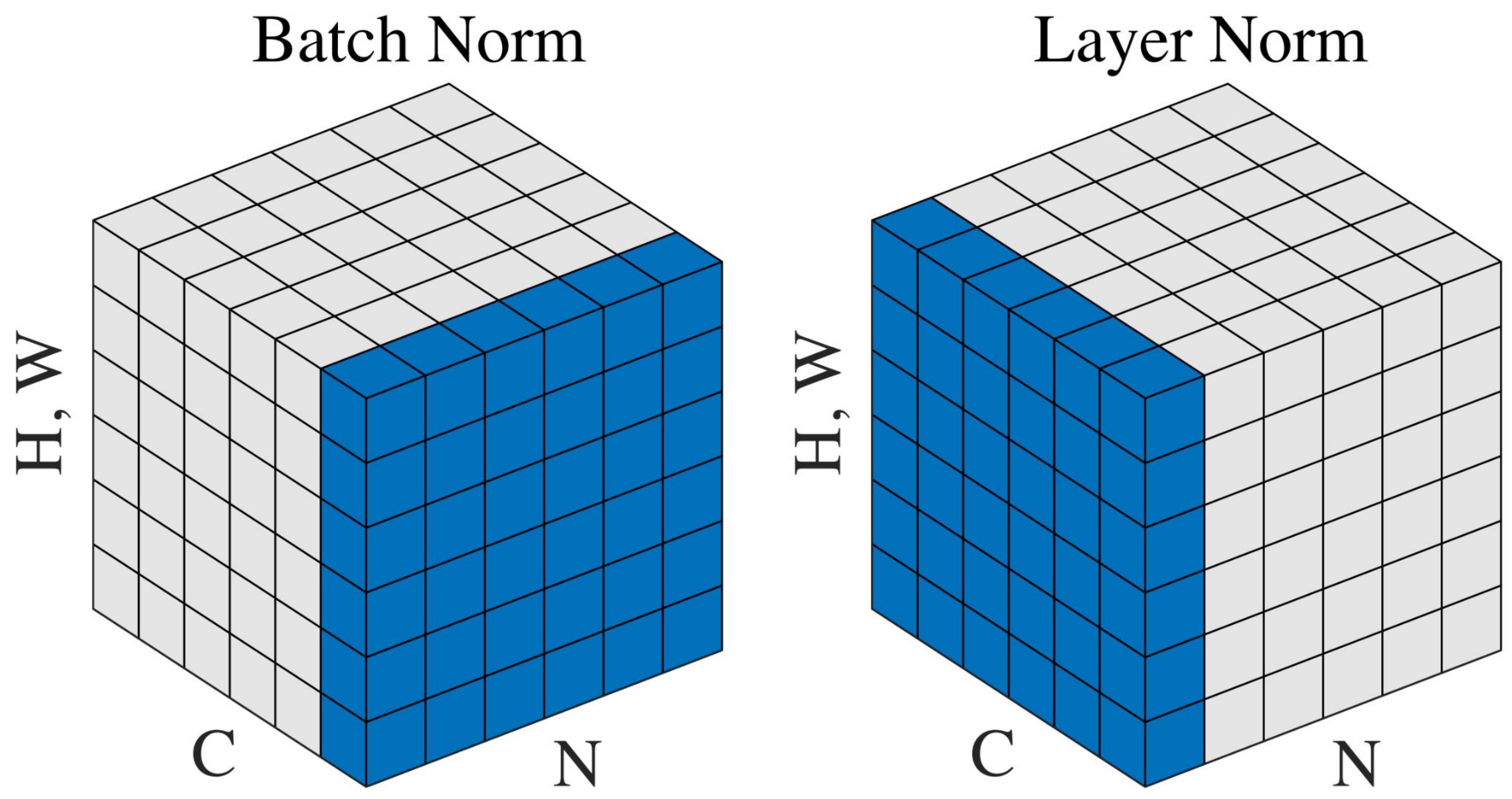
- Layer Normalization can be thought of as a method that aids optimization

Deep Learning

Basics

_ Batch Normalization (1/2)

- Batch normalization also reduces the chance of the feedforward signal's magnitude from blowing up or decaying
  - Batch Normalization is like Layer Normalization, except its mean and sigma are aggregated across examples in the batch, not activations



Deep Learning

Basics

_ Batch Normalization (2/2)

- Batch Normalization usually works well with sizable batch sizes greater than one, making it harder to use than Layer Normalization with very large models due to memory constraints
  - Both Layer and Batch Normalization make it easier to use larger learning rates
    - Researchers use Batch Normalization and Layer Normalization, but Layer Normalization is more common in recent architectures such as Transformers.

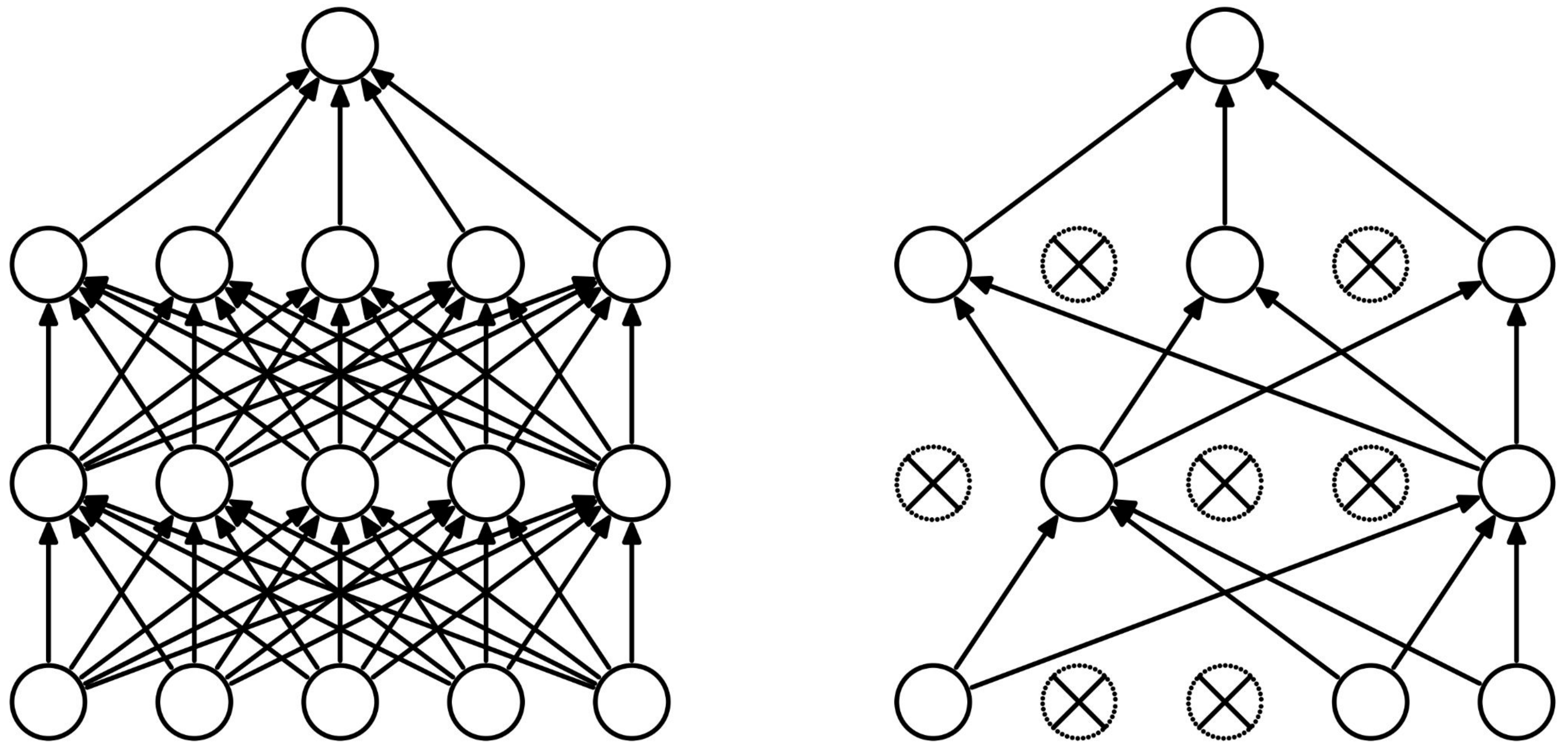
$$\mu_i^{(l)} = \frac{1}{B}\sum_{j=1}^{B} a_{i_j}^{(l)} \qquad \sigma_i^{(l)} = \sqrt{\frac{1}{B}\sum_{j=1}^{B}(a_{i_j}^{(l)} - \mu_i^{(l)})^2}$$

Deep Learning

Basics

_ Dropout

- Dropout randomly sets some activations to zero during training (often not during test time)
  - Assume ***b*** is a random Bernoulli mask vector
  - Then dropout elementwise multiplies the activations by ***b***
    - Dropout encourages redundant feature detectors



(a) Standard Neural Net (b) After applying dropout.

$$b \in \{0,1\}^H \quad b_i \sim \text{Bernoulli}(p)$$

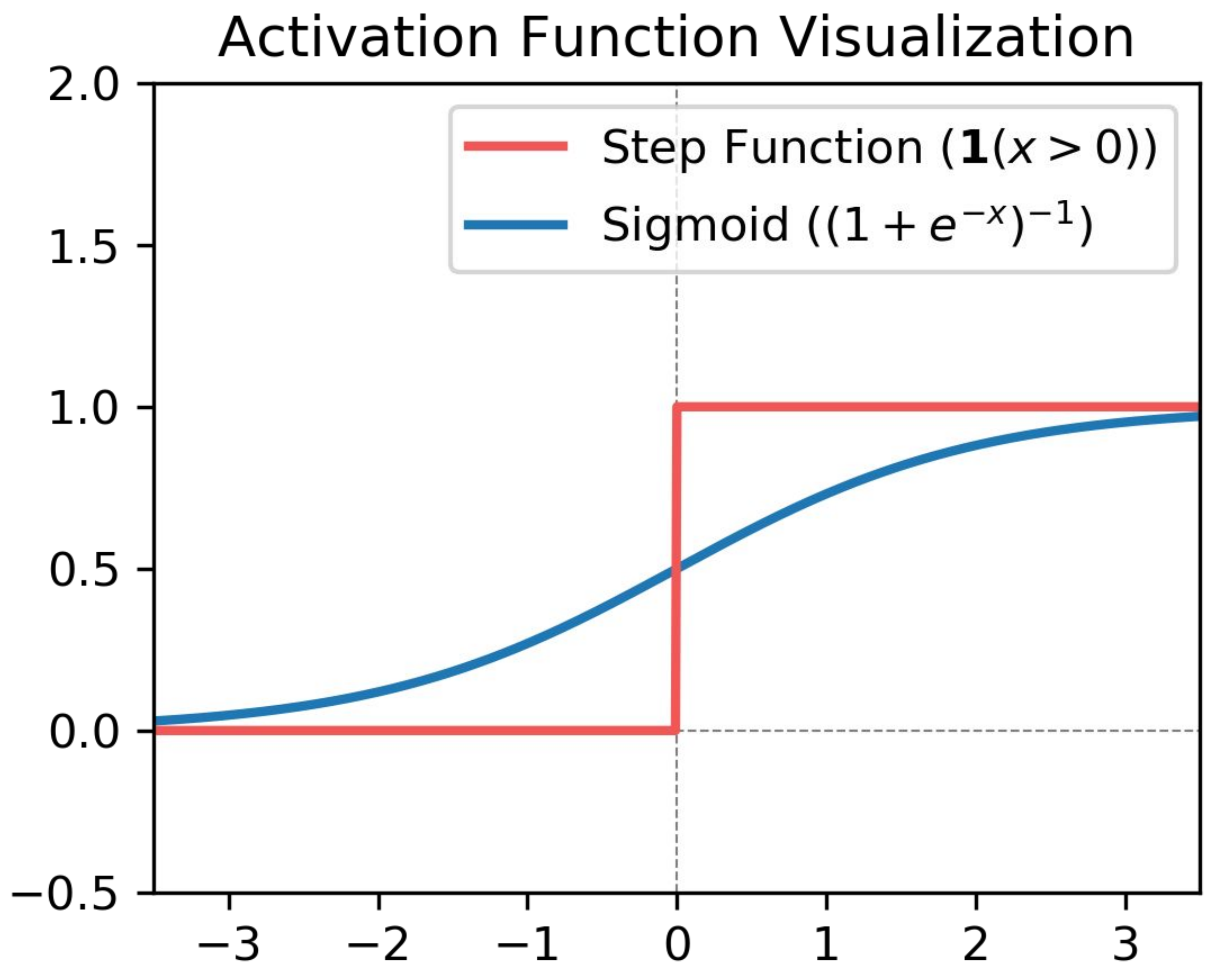
$$a^{(l)}_{\text{drop}} = a^{(l)} \odot b/(1-p)$$

Deep Learning

Basics

_ Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



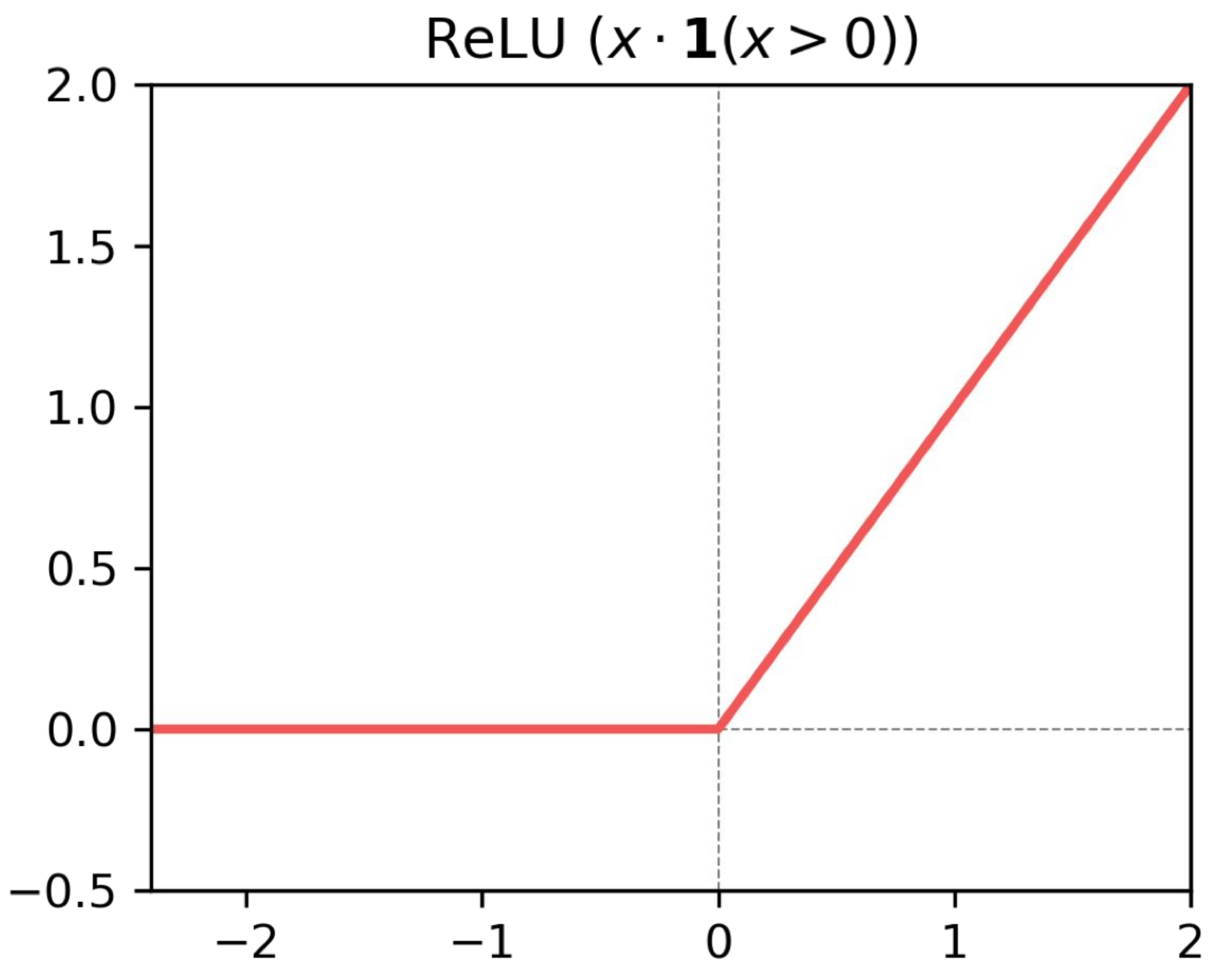
- If a feature is detected, then a neuron fires; a step function emulates firing, but is not differentiable
  - The sigmoid is differentiable and is a smooth approximation to a step function
    - The sigmoid function can be likened to a neuron firing probability
  - Sigmoids are used in LSTMs and as the output of probabilistic binary classifiers

Deep Learning

Basics

_ ReLU function

$$\texttt{ReLU}(x) = \max\{0, x\}$$



- ReLU stands for “Rectified Linear Unit”
  - Although not smooth, the ReLU is differentiable almost everywhere: $\texttt{ReLU}'(x) = \mathbf{1}(x > 0)$
    - Can be interpreted as gating inputs based on their sign:
      - If $\boldsymbol{x}$ is positive, let it through, otherwise, filter it

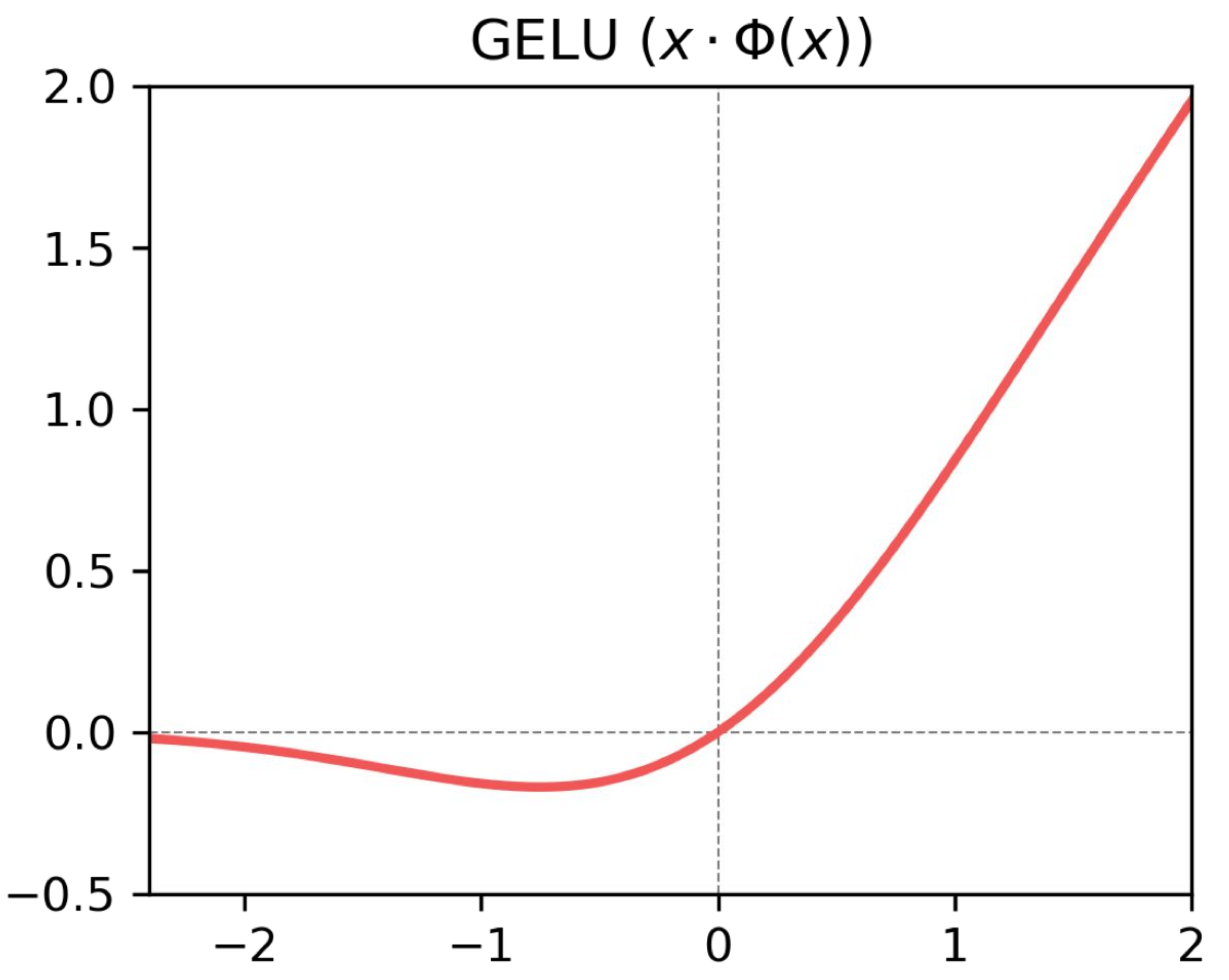
Deep Learning

Basics

_ GELU function

$$\mathtt{GELU}(x) = x \cdot \Phi(x)$$

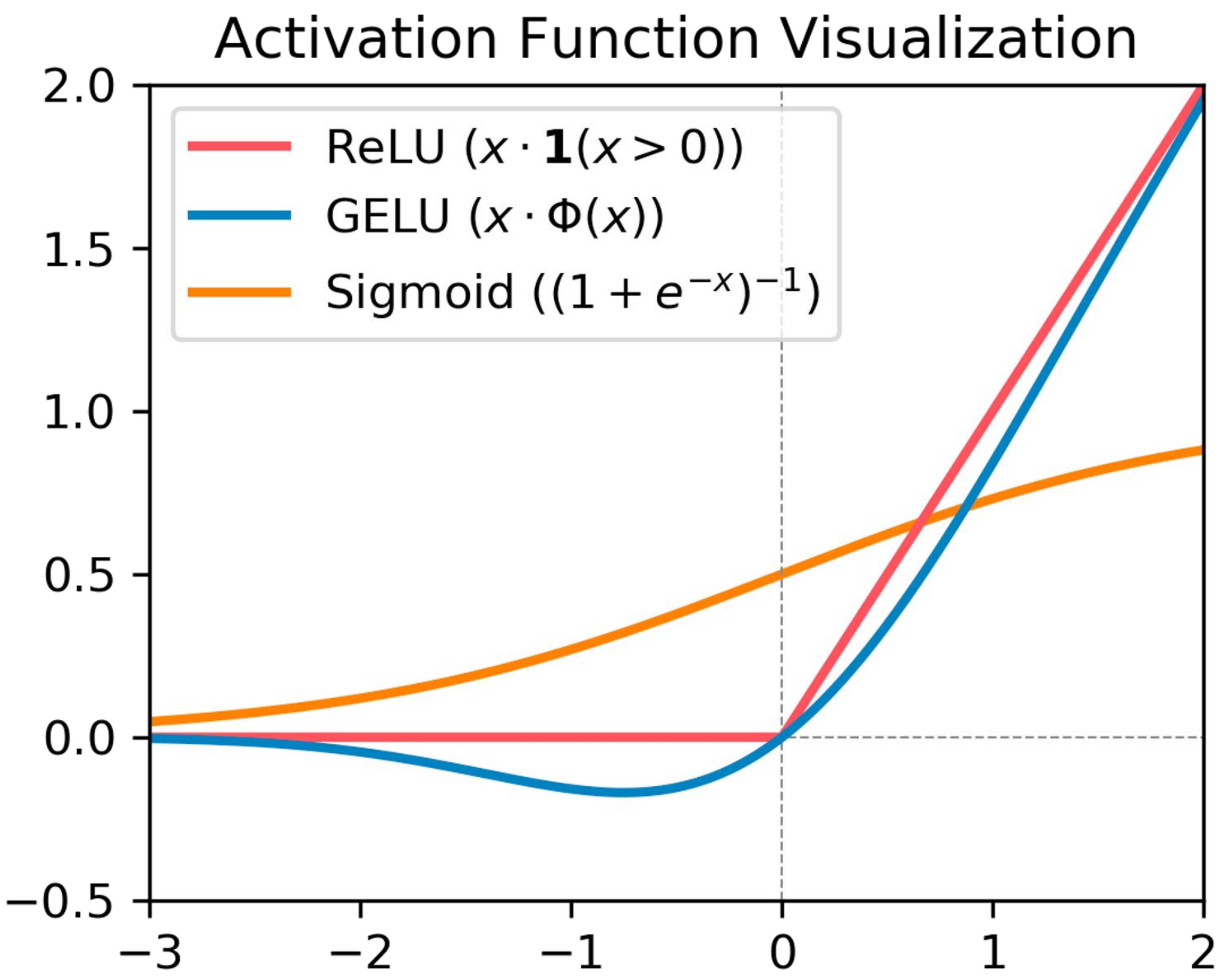
$$\Phi(x) = P(X \leq x), X \sim \mathcal{N}(0,1)$$



- GELU stands for "Gaussian Error Linear Unit"
  - GELU activations are designed to approximate the Gaussian Cumulative Distribution Function (CDF), which has the property of being smooth and continuous.
    - Gaussian distributions are commonly observed in natural data and processes, making GELU activations well-suited for a wide range of real-world tasks.

Deep Learning

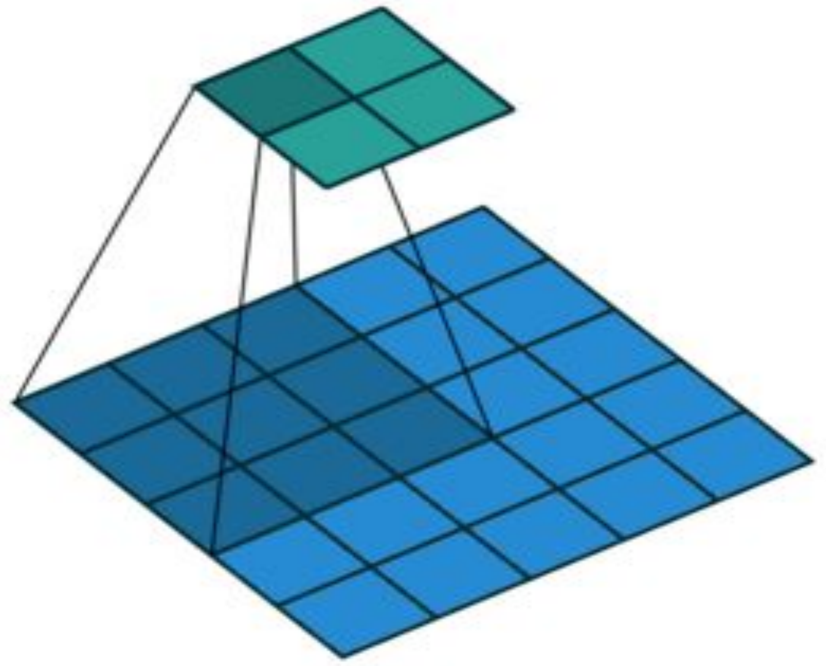
Basics

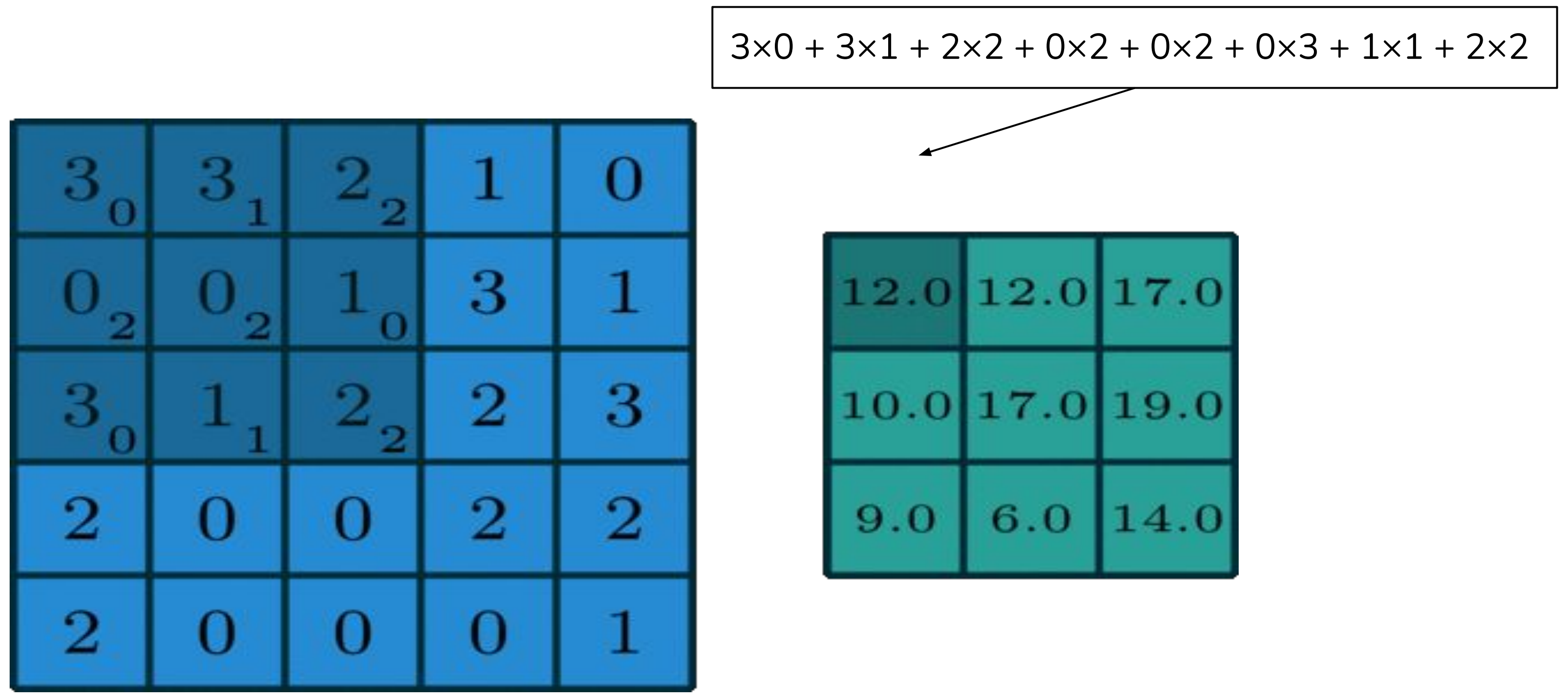


Deep Learning

Basics

_ Convolutions





- Since many useful data features may be local, we can move a sliding feature detector (“kernel”) across an input to help detect such features
  - Often used in hidden layers, convolutions use few parameters by repeatedly re-applying kernels across the whole input

Deep Learning

Losses

_ Minimum Description Length Principle (MDL)

- Imagine we want to encode sequences of As, Bs, and Cs
  - We can give higher probability events shorter descriptions, so that the coding scheme is more efficient
    - If we encoded A with 11 and C with 1, more probable events would require longer descriptions
- Notice the description length of symbol ***S*** in ***{A, B, C}*** is $-\log_2 P(S)$

- In Machine Learning, we often implicitly select the model that has the shortest description length (shortest encoding) of the data
  - With MDL, one can view learning as data compression

P(A) = 50%
P(B) = 25%
P(C) = 25%

A → 0
B → 10
C → 11

Deep Learning

Losses

_ Minimum Description Length Principle (MDL)

- It is a fundamental concept in machine learning that provides a criterion for selecting models based on their ability to represent data using concise descriptions.
  - The principle is rooted in the idea that simpler explanations are often preferable, as they can capture the underlying patterns in the data without overfitting.
    - In essence, the MDL principle suggests that the best model is the one that achieves a balance between accurately describing the data and minimizing the complexity of the model itself.

Deep Learning

Losses

_ Minimum Description Length Principle (MDL)

- Imagine you have a dataset and different models that can potentially explain the patterns in that data.
  - Each model has a certain complexity associated with it, which includes the number of parameters, rules, or components that the model uses.
  - According to the MDL principle, the best model is the one that produces the shortest description length for both the data and the model itself.
    - This balance between accurately describing the data and minimizing the model's complexity helps avoid overfitting and ensures that the model doesn't capture noise in the data
      - Overall, the MDL principle provides a framework for selecting models that are both effective in explaining data and simple enough to avoid overfitting. It encourages a balance between accuracy and complexity, promoting more interpretable and generalizable models.

Deep Learning

Losses

_ Entropy

- If the ith symbol has probability p_i and its encoding size is $-log\ p_i$ the expected code length is the entropy:

$$H(p) = \mathbb{E}_{X\sim P}[-\log p(X)] = -\sum_{i=1}^{k} p_i \log p_i$$

- Following the MDL, we can select models that minimize the entropy
  - Researchers often use the entropy as a loss for generative models
    - Entropy can also be thought of as measure of a random variable's randomness or uncertainty

Deep Learning

Losses

_ Cross-Entropy

- The cross entropy measures the difference between two distributions

$$H(p;q) = \mathbb{E}_{X\sim P}[\log q(X)] = -\sum_{i=1}^{k} p_i \log q_i$$

- The cross-entropy measures the average number of bits needed to encode events that occur following distribution *p*, if a coding scheme is used that is optimal for the probability distribution *q*
  - This is a loss for classifiers, which encode the conditional distribution Y | X

Deep Learning

Losses

_ KL Divergence

- The Kullback-Leibler Divergence measures the difference between two distributions

$$\mathrm{KL}[p\|q] = -\sum_{i=1}^{k} p_i \log \frac{q_i}{p_i}$$

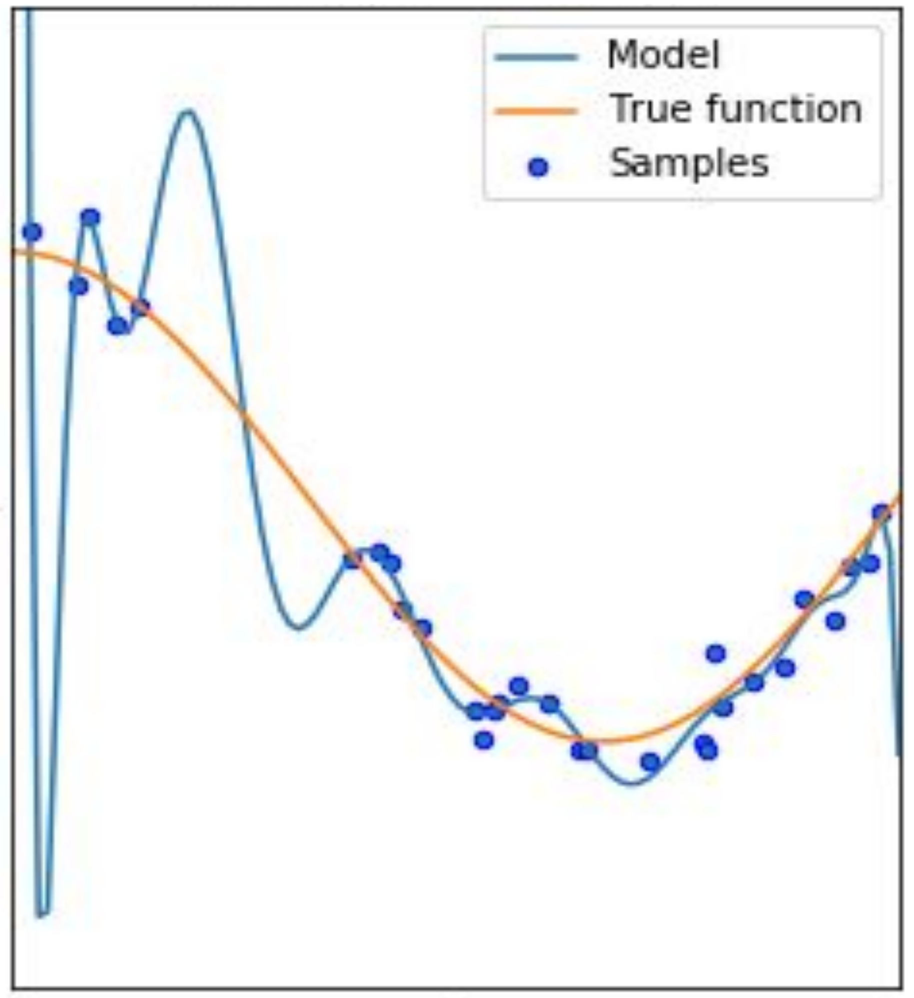
- If we encode messages with an optimal coding scheme for distribution ***q***, but the true distribution is actually ***p***, then each message requires, on average, an additional $\mathrm{KL}[p\|q]$ bits to be encoded compared to the optimal encoding.

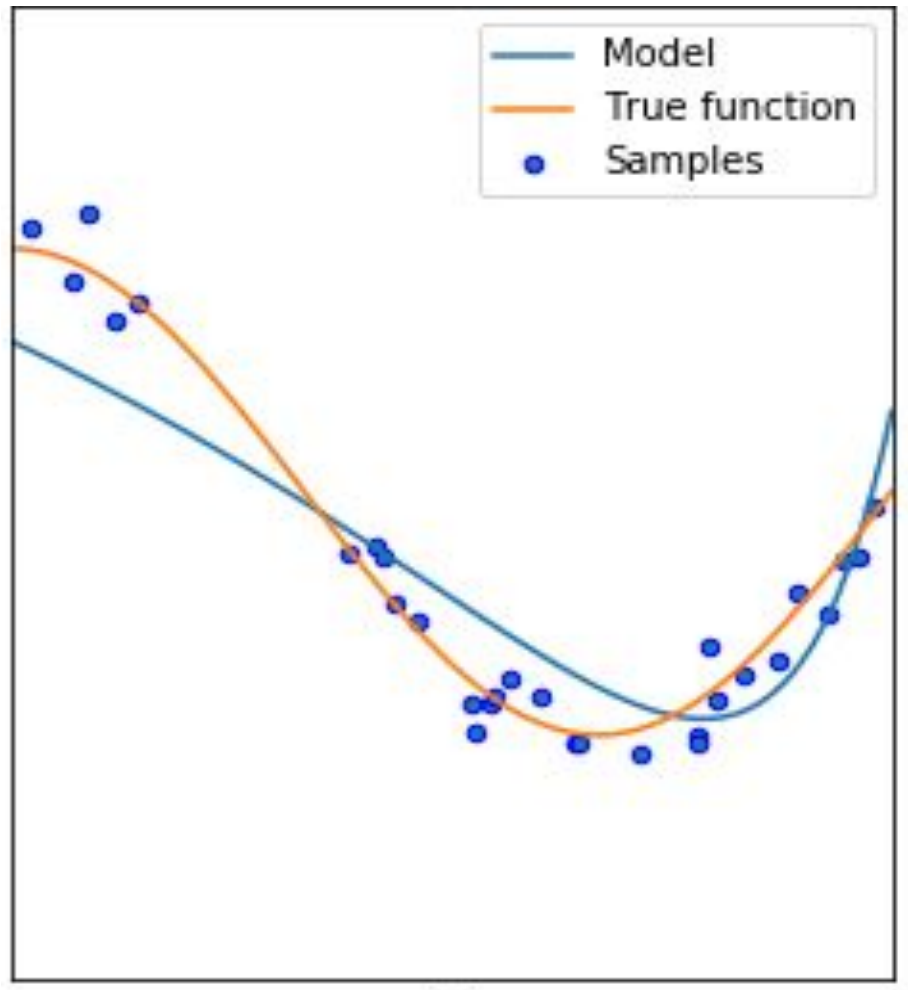
Deep Learning

Losses

_ l_2 Regularization

- Penalizes model complexity by adding the model parameter norm to the loss function
  - Regularization strength scaled by λ
  - A probabilistic interpretation is this incorporates a Gaussian prior over the parameters; the prior has mean 0 and variance inversely proportional to λ
- Can be interpreted as penalizing the bits required to encode the parameters (bigger norm → more bits)





Deep Learning

Optimizers

_ Stochastic Gradient Descent (SGD)

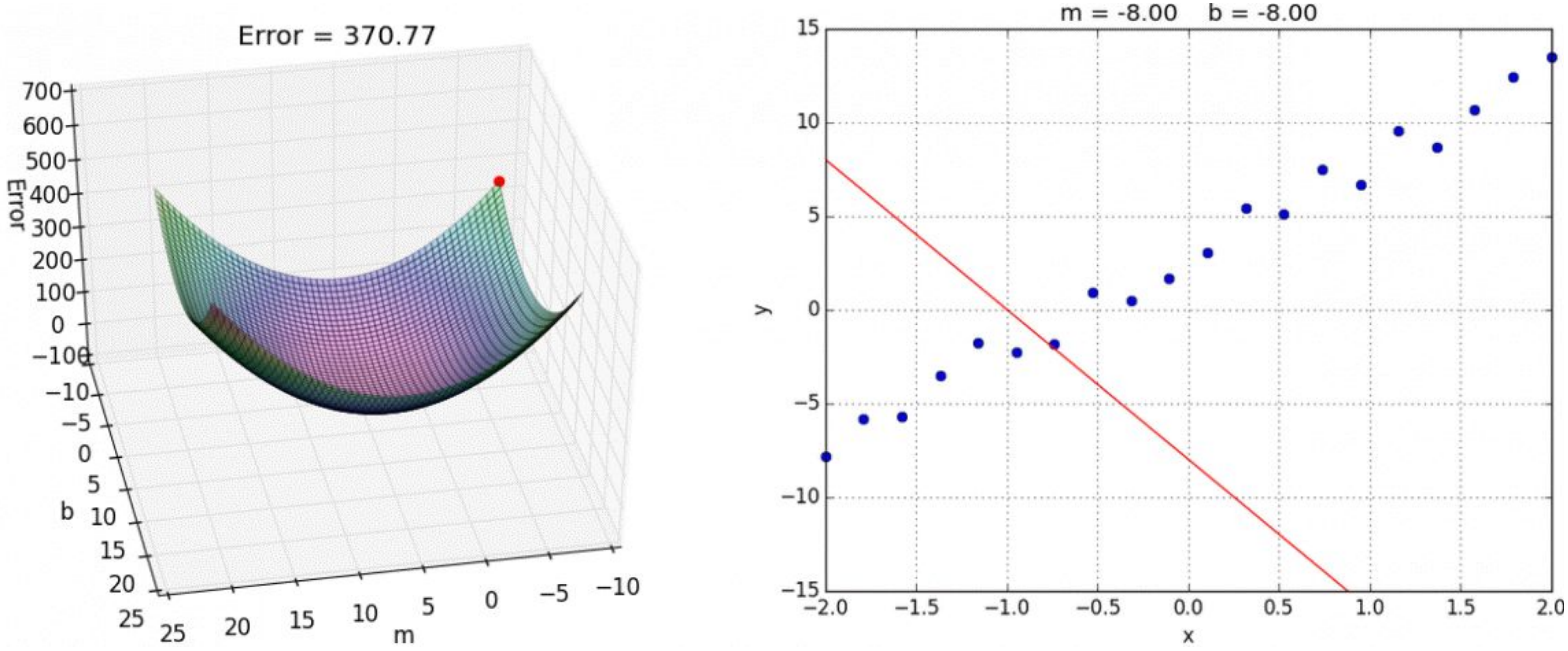
- To optimize the parameters θ using loss function $\mathcal{L}$, iteratively move in the direction of steepest descent with step size α

$$\theta_{k+1} = \theta_k - \alpha \nabla \mathcal{L}(\theta_k)$$

- Neural network optimizers are based around SGD but rarely use this exact formulation
  - Consequently networks are optimized with a local search method so models learn from many small incremental changes, not radical sudden changes nor hand-chosen parameters

Deep Learning

Optimizers



Deep Learning

Optimizers

_ SGD + Momentum

- To reduce gradient estimation noise during optimization, researchers often use momentum, which is equivalent to moving in the direction of an (exponential) moving average of the gradient

$$g_k = \nabla \mathcal{L}(\theta_k) + \mu g_{k-1}$$
$$\theta_{k+1} = \theta_k - \alpha g_k$$

μ is a fixed constant

- Gradients farther in the past have exponentially less weight, so old gradients die out and so that the optimizer can quickly adapt

$$g_k = \nabla_\theta \mathcal{L}(\theta_k) + \mu^1 \nabla_\theta \mathcal{L}(\theta_{k-1}) + \mu^2 \nabla_\theta \mathcal{L}(\theta_{k-2}) + \cdots$$

Deep Learning

Optimizers

_ Adam

- Basic idea is to combine momentum and a second moment adjustment

$m_k = (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_k) + \beta_1 m_{k-1}$ first moment estimate (momentum-like)

$v_k = (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_k))^2 + \beta_2 v_{k-1}$ second moment estimate (*roughly* squared length)

$m_0 = 0 \quad v_0 = 0$

$\hat{m}_k = \frac{m_k}{1-\beta_1^k} \qquad \hat{v}_k = \frac{v_k}{1-\beta_2^k}$

early on m and v values will be small, and this correction "blows them up" a bit for small *k*

Good default settings:

$$\theta_{k+1} = \theta_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

$\alpha = 0.001 \ \epsilon = 10^{-8} \ \beta_1 = 0.9 \ \beta_2 = 0.999$

small number to prevent division by zero

each dimension is divided by its magnitude

Deep Learning

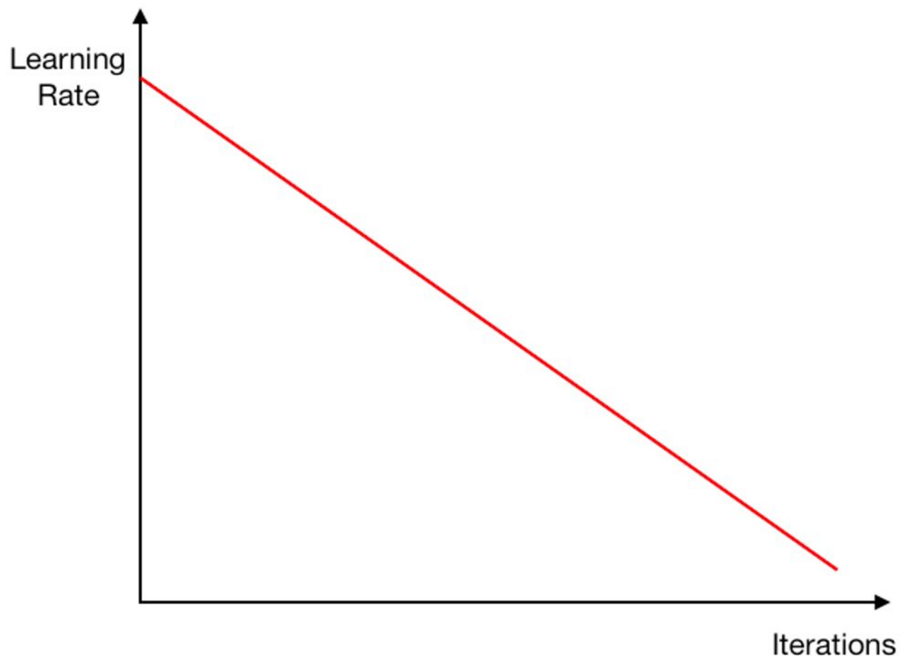
Optimizers

_ Learning Rate Schedules

- Learning rates are not always constant: often they decay following a schedule

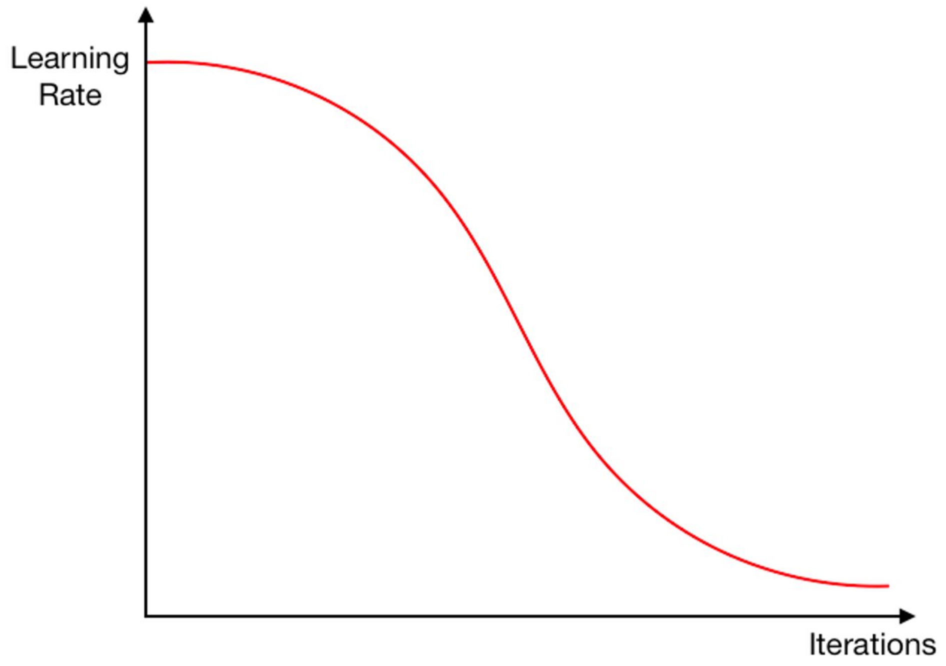
The Linear Decay schedule decays the learning rate a constant amount each iteration

$$\alpha_i = \alpha_{\text{initial}} \cdot \left(1 - \frac{i}{\texttt{max\_steps}}\right)$$



The Cosine Annealing schedule decays the learning rate proportional to a cosine function (from 0 to π)

$$\alpha_i = \alpha_{\text{initial}} \cdot 0.5 \cdot \left[1 + \cos\left(\pi \cdot \frac{i}{\texttt{max\_steps}}\right)\right]$$



NLP

Datasets

_ Common datasets:

- SST-2 and IMDb
  - NLP datasets for binary sentiment analysis of movie reviews
    - SST-2 contains pithy professional expert movie reviews, and IMDb contains full-length lay movie reviews

SST-2 Example

This is meticulously made, with every decision working to craft the stylistic, iconic film many of us hold so dear.

Positive

IMDb Example

Once again Mr. Costner has dragged out a movie for far longer than necessary. Aside from the terrific sea rescue sequences, of which there are very few I just did not care about any of the characters....

Negative

NLP

Datasets

_ Question Answering and Reading Comprehension:

- SQuAD (Stanford Question Answering Dataset)
- MS MARCO
- RACE (ReAding Comprehension from Examinations)
- NewsQA
- HotpotQA
- Natural Questions (NQ)

NLP

Datasets

_ Machine Translation:

- WMT (Workshop on Machine Translation) Datasets
- IWSLT (International Workshop on Spoken Language Translation) Datasets

NLP

Datasets

_ Text Classification:

- Reuters-21578: A dataset of news articles categorized into multiple classes.
- 20 Newsgroups: A dataset containing newsgroup documents categorized into 20 different topics.
- AG News
- Yelp Reviews
- Amazon Reviews
- Hate Speech and Offensive Language (HateEval)

NLP

Datasets

_ Named Entity Recognition (NER):

- CoNLL-2003: A dataset for NER with data from various sources like news articles.
- OntoNotes: A dataset with named entity annotations across multiple genres and domains.

NLP

Datasets

_ Semantic Textual Similarity:

- STS-B (Semantic Textual Similarity Benchmark): A dataset for measuring the similarity between sentences.
- ParaNMT-50M (Paraphrase Dataset)

NLP

Datasets

_ Language Generation:

- Wikipedia Text: Text from Wikipedia articles used for various generation tasks.
- ROCStories: A dataset for story generation tasks.
- Common Crawl
- WebText

NLP

Datasets

_ Dialogue and Language Understanding:

- ATIS (Airline Travel Information System): A dataset for spoken language understanding in the airline domain.
- MultiWOZ: A dataset for dialogue and natural language understanding in the task-oriented dialogue domain.
- DailyDialog
- Persona-Chat

NLP

Datasets

_ Summarization:

- CNN/DailyMail
- XSum (eXtreme Summarization)

NLP

Datasets

_ Language Understanding and Semantics:

- ROCStories
- ANLI (Adversarial NLI)
- BoolQ (Boolean Questions)

NLP

Datasets

_ Other Specialized Datasets:

- The Harry Potter Dataset
- COCO (Common Objects in Context)
- WordNet

NLP

Datasets

_ GLUE and SuperGLUE GLUE SuperGLUE

- These benchmarks aggregate NLP model performance over several tasks (e.g., sentiment analysis, natural language inference, etc.)
  - SuperGLUE is harder than GLUE, but state-of-the-art models have exceeded human-level performance
  - These benchmarks are used to show how well a pretrained NLP model performs across several downstream NLP tasks

NLP

Evaluation/Metrics

_ How to evaluate performance?

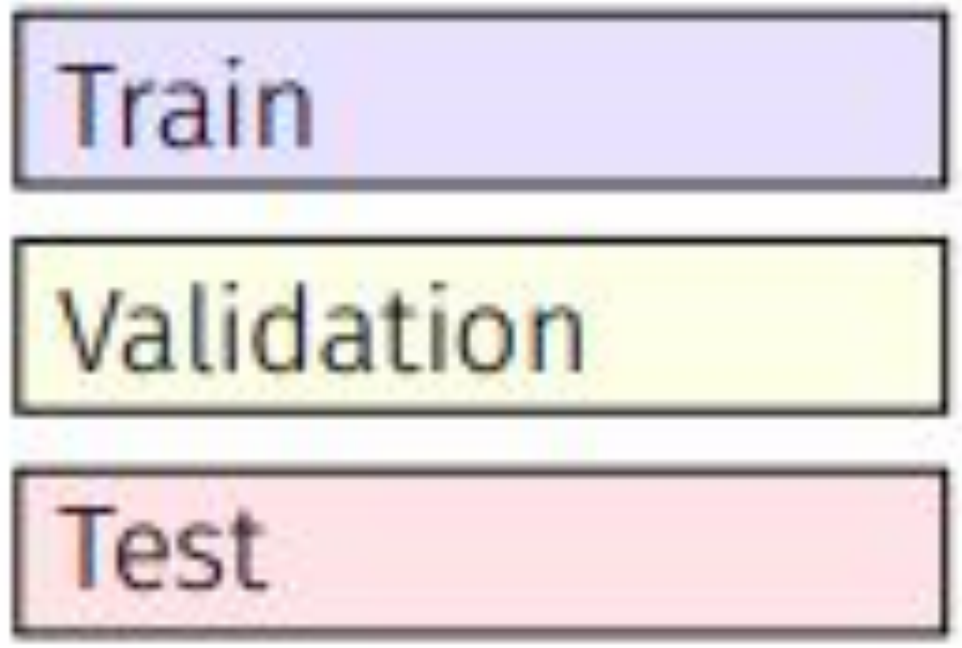
- Text classification
- Text generation
- Caveats of NLP benchmarking

NLP

Evaluation/Metrics

_ How to evaluate performance?

- Train/Dev/Test data splits
  - Training and Test data
  - Development (Validation) set used for optimizing hyper-parameters

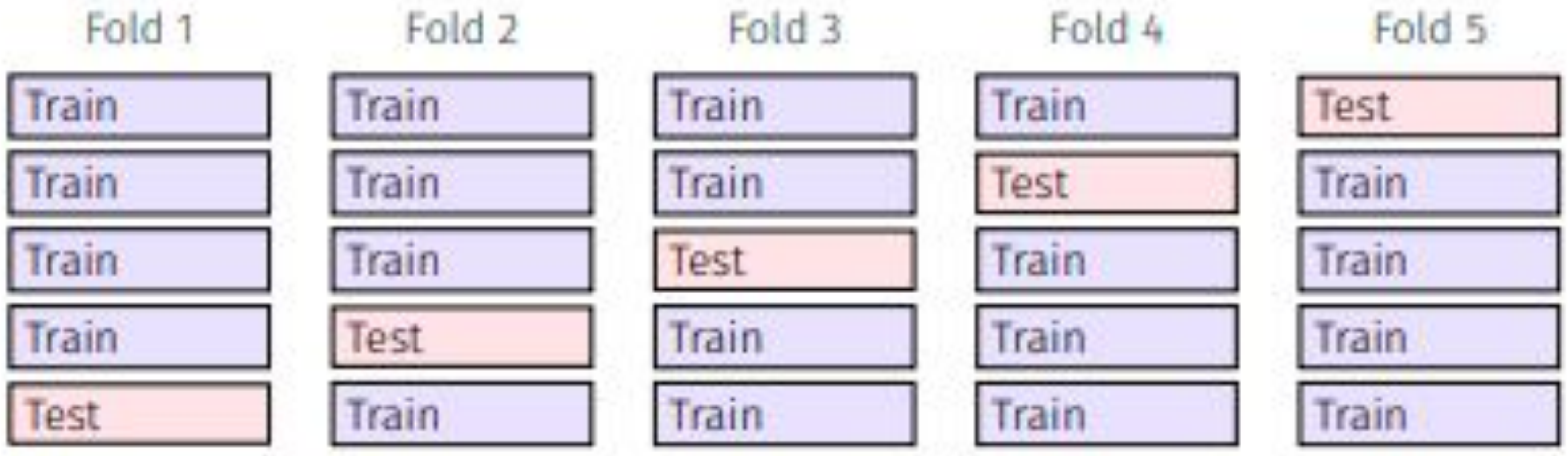


NLP

Evaluation/Metrics

_ How to evaluate performance?

- Cross-Validation
  - K-fold cross-validation
    - partitions the data into *K* chunks
      - *K - 1* of which form the training set
      - The last chunk serves as the validation set



NLP

Evaluation/Metrics

_ Evaluation of Text Classification

- Confusion matrix (binary case)
  - Two classes: Positive and Negative

Confusion matrix

	Pred. Negative	Pred. Positive
Act. Negative	True negative (TN)	False positive (FP)
Act. Positive	False negative (FN)	True positive (TP)

NLP

Evaluation/Metrics

_ Evaluation of Text Classification

- Accuracy of classifier ***f*** on test set ***T***:

$$\mathrm{Acc}_T(f) = \frac{1}{|T|}\sum_{i=1}^{|T|} I(f(x_i), y_i)$$

Example (Disease detection)

	Pred. Negative	Pred. Positive
Act. Negative	168	33
Act. Positive	48	37

  - 37 + 48 + 33 + 168 = 286 → Test set size $|T|$ = 286 $\mathrm{Acc}_T(f)$ = 1/286 x (37 + 168) = 0.7186

NLP

Evaluation/Metrics

_ Evaluation of Text Classification

- Precision, recall, F-1 score

Confusion matrix

	Pred. Negative	Pred. Positive
Act. Negative	True negative (TN)	False positive (FP)
Act. Positive	False negative (FN)	True positive (TP)

  - Precision (for class positive) = TP / (TP + FP)
  - Recall (for class positive) = TP / (TP + FN)
  - F-1 score (for class positive) = 2PR / (P + R)

NLP

Evaluation/Metrics

_ Evaluation of Text Classification

- Confusion matrix – multi-class
  - We can unambiguously compute Precision and Recall for each class
    - How to get the F-1 score for the complete test set across classes?
      - Macro-averaging (average of F-1 scores), or micro-averaging

true class: \ prediction:	money-fx	trade	interest	wheat	corn	grain
money-fx	95	0	10	0	0	0
trade	1	1	90	0	1	0
interest	13	0	0	0	0	0
wheat	0	0	1	34	3	7
corn	1	0	2	13	26	5
grain	0	0	2	14	5	10

NLP

Evaluation/Metrics

_ Evaluation of Text Generation

- BLEU (Bilingual Evaluation Understudy)
    - Among the first and most popular metrics proposed for automatic evaluation of MT systems
        - Precision-based metric that computes the *n-gram* overlap between the reference and the hypothesis
        - In particular, BLEU is the ratio of the number of overlapping *n-grams* to the total number of *n-grams* in the hypothesis.
    - Corpus-level metric
        - i.e., BLEU gives a score over the entire corpus (as opposed to scoring individual sentences)
    - Major drawbacks of BLEU:
        - it does not take recall into account
        - it only allows exact *n-gram* matching

NLP

Evaluation/Metrics

_ Evaluation of Text Generation

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
    - ROUGE metric includes a set of variants: ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S
        - ROUGE-N is similar to BLEU in counting the *n-gram* matches between the hypothesis and reference
            - however, it is a recall-based measure unlike BLEU which is precision-based
        - ROUGE-L measures the longest common subsequence (LCS) between a pair of sentences

NLP

Evaluation/Metrics

_ Caveats of NLP Benchmarks

- The ‘gold’ data paradigm might not always fit
  - The assumption of a ground truth makes sense when humans highly agree on the answer
    - “Does this image contain a bird?”
    - “Is ‘learn’ a verb?”
    - “What is the capital of Italy?”
  - This assumption often does not make sense, especially when language is involved
    - Questions determining a word sense
    - “Is this comment toxic?”
  - Human label variation impacts all steps of the traditional ML pipeline, and is an opportunity, not a problem

NLP

Evaluation/Metrics

_ Caveats of NLP Benchmarks

- Human annotators are biased
  - Datasets are often constructed using a small number of annotators, and humans are biased
    - Concerns about data diversity, especially when workers freely generate sentences
    - Models do not generalize well to examples from annotators that did not contribute to the training set

NLP

Evaluation/Metrics

_ In summary:

- Vast amount of tasks and datasets
    - Data quality matters
    - Understanding the data, annotators, task matters too
- Deep familiarity with common evaluation metrics is essential
    - Getting better scores is just a beginning of the story
    - Evaluating generation is an art

NLP

The challenges

_ Ambiguity and variability of human language

- Highly ambiguous
  - Example:
    - Compare “**I ate pizza with friends**” to “**I ate pizza with olives**”
- Highly variable
  - Example:
    - The core message of “**I ate pizza with friends**” can be expressed as “**friends and I shared some pizza**”
- Humans are great users of language, very poor at formally understanding and describing rules that govern language

NLP

The challenges

_ Supervised machine learning to save us …

- The best known set of methods for dealing with language data
  - Supervised machine learning algorithms
- Machine Learning attempts to infer patterns and regularities from a set of pre-annotated input–output pairs
  - Machine Learning excels at problem domains where:
    - a good set of rules is very hard to define …
    - but annotating the expected output for a given input is relatively simple.

NLP

The challenges

_ Language is very challenging

- Natural language exhibits properties that make it very challenging for Machine Learning
  - Discrete
  - Compositional
  - Sparse

NLP

The challenges

_ Language is symbolic and discrete

- Basic elements of written language: characters
  - Characters form words that denote objects, concepts, events, actions, and ideas
- Characters and words are discrete symbols
  - Words such as “hamburger” or “pizza” each evoke in us a certain mental representations
    - But they are distinct symbols, whose meaning is external to them, to be interpreted in our heads
    - No inherent relation between “hamburger” and “pizza” can be inferred from the symbols or letters themselves

NLP

The challenges

_ Characters and words are discrete symbols

- Compare that to concepts such as color (in machine vision), or acoustic signals — these concepts are continuous
  - Colorful image to gray-scale image using a simple mathematical operation
  - We can to compare two different colors based on inherent properties such as hue and intensity
- This cannot be easily done with words
  - There is no simple operation to move from the word “red” to the word “pink” without using a large lookup table or a dictionary

NLP

The challenges

_ Language is compositional

- Letters → words → phrases → sentences → documents
  - The meaning of a phrase can be larger than the meaning of the individual words, and follows a set of intricate rules
- To interpret a text, we need to work beyond the level of letters and words, and look at long sequences of words such as sentences, or even complete documents
  - Memory and context are essential

NLP

The challenges

_ Data sparseness

- Combinations of words to form meanings → ∞
  - We could never enumerate all possible valid sentences
- No clear way of generalizing from one sentence to another, or defining the similarity between sentences, that does not depend on their meaning which is unobserved to us
  - Challenging when learning from examples
  - Even with a huge example set we are very likely to observe events that never occurred in the example set and that are very different

Machine Learning in NLP

Simple algorithms

_ Supervised classification setup

- We often restrict ourselves to search over specific families of functions
  - e.g., the space of all linear functions with $\mathbf{d_{in}}$ inputs and $\mathbf{d_{out}}$ outputs
- By restricting to a specific hypothesis class, we are injecting the learner with inductive bias
  - A set of assumptions about the form of the desired solution

Machine Learning in NLP

Simple algorithms

_ High-dimensional linear functions

- Vector $\boldsymbol{x}$ is the input, matrix $\boldsymbol{W}$ and vector $\boldsymbol{b}$ are the parameters
  - Typically denoted $\boldsymbol{\Theta} = \boldsymbol{W}, \boldsymbol{b}$

→ Goal of learning

- Set the values of the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ such that the function behaves as intended on a collection of input values $\boldsymbol{x}_{1:k} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and the corresponding desired outputs $\boldsymbol{y}_{1:k} = \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k$

Machine Learning in NLP

Simple algorithms

_ Binary classification

- Our input is in the form of a natural language text
- Our labels are two categories
  - e.g., positive and negative
- Let’s start with the labels
  - Very easy: Just arbitrarily map the categories into 0 and 1 (e.g., negative = 0, positive = 1)

Machine Learning in NLP

Simple algorithms

_ How to transform text into a fixed-size vector of real numbers

- What is our setup:
  - $f(x) : R^d \rightarrow R$
  - $f(x) = x \cdot w + b$
- What we need:
  - $x \in R$
- What we have:
  - *"One of my favorite movies ever,The Shawshank Redemption is a modern day classic as it tells the story of two inmates who become friends and find solace over the years in which this movie takes place.Based on a Stephen King novel, …"*

Machine Learning in NLP

Simple algorithms

_ What is a “word”?

- A matter of debate among linguists, answer is not always clear
  - Very simplistic definition:
    - words are sequences of letters separated by whitespace
    - But: dog, dog?, dog., and dog) would be different words
      - Better: words separated by whitespace or punctuation
  - A process called tokenization splits text into tokens based on whitespace and punctuation
    - English: the job of the tokenizer is quite simple
    - Hebrew, Arabic: sometimes without whitespace
    - Chinese: no whitespaces at all

Machine Learning in NLP

Simple algorithms

Tokens

- Symbols cat and Cat have the same meaning, but are they the same word?
    - Something like New York, is it two words, or one?
- We distinguish between words and tokens
    - We refer to the output of a tokenizer as a token, and to the meaning-bearing units as words
- Keep in mind:
    - We use the term word very loosely, and take it to be interchangeable with token.

Machine Learning in NLP

Simple algorithms

Vocabulary

- We build a fix-sized static vocabulary
  - e.g., by tokenizing training data
  - Typical sizes:
    - 20,000 – 100,000 words
    - Each word has a unique fixed index:
      - V = (a_1 $abandon_2$ . . . cat_{852} . . . $zone_{2999}$ zoo_{3000})

Machine Learning in NLP

Simple algorithms

Averaged Bag of Words

$$\boldsymbol{x} = \frac{1}{|D|}\sum_{i=1}^{|D|} \boldsymbol{x}^{D[i]}$$

- $\mathbf{D_{[i]}}$
  - word in doc **D** at position i
  - $x^{D[i]}$
    - one-hot vector

Example: a cat sat → a, cat, sat

- V = $(a_1\ abandon_2 \ldots cat_{852} \ldots zone_{2999}\ zoo_{3000})$
  - a = $x^{D[1]} = (1_1\ 0_2\ 0_3 \ldots 0_{2999}\ 0_{3000})$
  - cat = $x^{D[2]} = (0_1 \ldots 1_{852} \ldots 0_{2999}\ 0_{3000})$
  - sat = $x^{D[3]} = (0_1 \ldots 1_{2179} \ldots 0_{2999}\ 0_{3000})$
    - x = $\mathbf{(0.33_1\ 0_2 \ldots 0_{851}\ 0.33_{852}\ 0_{853} \ldots 0.33_{2179} \ldots 0_{3000})}$

Machine Learning in NLP

Simple algorithms

_ Out-of-vocabulary (UNK) tokens

- Words in a language are very unevenly distributed (Zipf's law)
  - There is always a large 'tail' of rare words
  - When building the vocabulary, use the most frequent words, all others represented by an unknown token (UNK or OOV)
    - Example: most common 3,000 words and UNK
      - V = (a_1 $abandon_2$ . . . cat_{852} . . . $zone_{2999}$ zoo_{3000} UNK_{3001})

Machine Learning in NLP

Simple algorithms

- Subword units: Byte-pair encoding
    1. The words in the corpus are split into characters (marking original spaces with a special space character) — this is the initial vocabulary V
    2. The most frequent pair of characters is merged and added to V
    3. Repeat 2 for a fixed given number of times
    4. Each of these steps increases V by one, beyond the original inventory of single characters

- When done over large corpora with multiple languages and writing systems, BPE prevents OOV!

Machine Learning in NLP

Simple algorithms

_ Example

t h i s _ f a t _ c a t _ w i t h _ t h e _ h a t _ i s _ i n _ t h e _ c a v e _ o f _ t h e _ t h i n _ b a t

- Most frequent: t h (6 times), merge into a single token th

th i s _ f a t _ c a t _ w i th _ th e _ h a t _ i s _ i n _ th e _ c a v e _ o f _ th e _ th i n _ b a t

- Most frequent: a t (4 times), merge into a single token

th i s _ f at _ c at _ w i th _ th e _ h at _ i s _ i n _ th e _ c a v e _ o f _ th e _ th i n _ b at

Machine Learning in NLP

Simple algorithms

_ Example

th i s _ f at _ c at _ w i th _ th e _ h at _ i s _ i n _ th e _ c a v e _ o f _ th e _ th i n _ b at

- Most frequent: th e (3 times), merge into a single token

th i s _ f at _ c at _ w i th _ the _ h at _ i s _ i n _ the _ c a v e _ o f _ the _ th i n _ b at

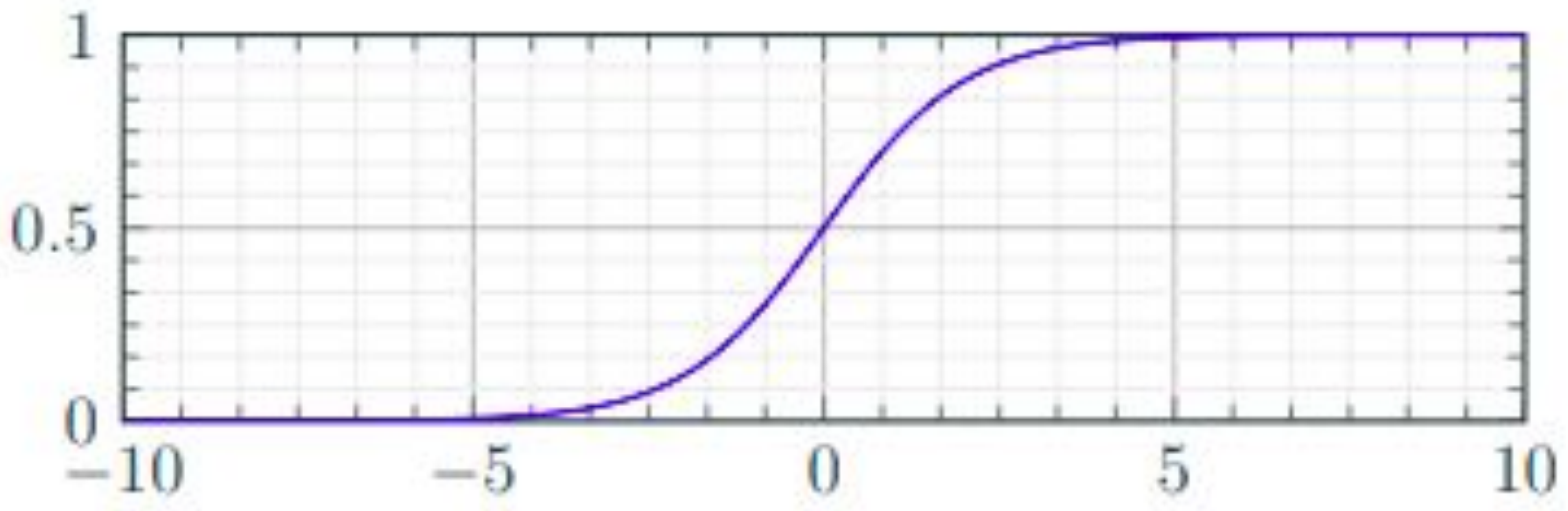
- V = {t, h, i, s, _, f, a, c, w, e, n, v, o, f, b, th, at, the}

Machine Learning in NLP

Simple algorithms

Log linear models

- We have this linear function:
  - $f(x) : R^{din} \rightarrow R$
  - $f(x) = xw+b = x_1w_1+. . .+x_{din}w_{din}+b$
    - $f(x)$ has unbounded range $(-\infty, +\infty)$
  - Non-linear mapping to [0, 1]
    - Each example's label is $\boldsymbol{y} \in \{0, 1\}$



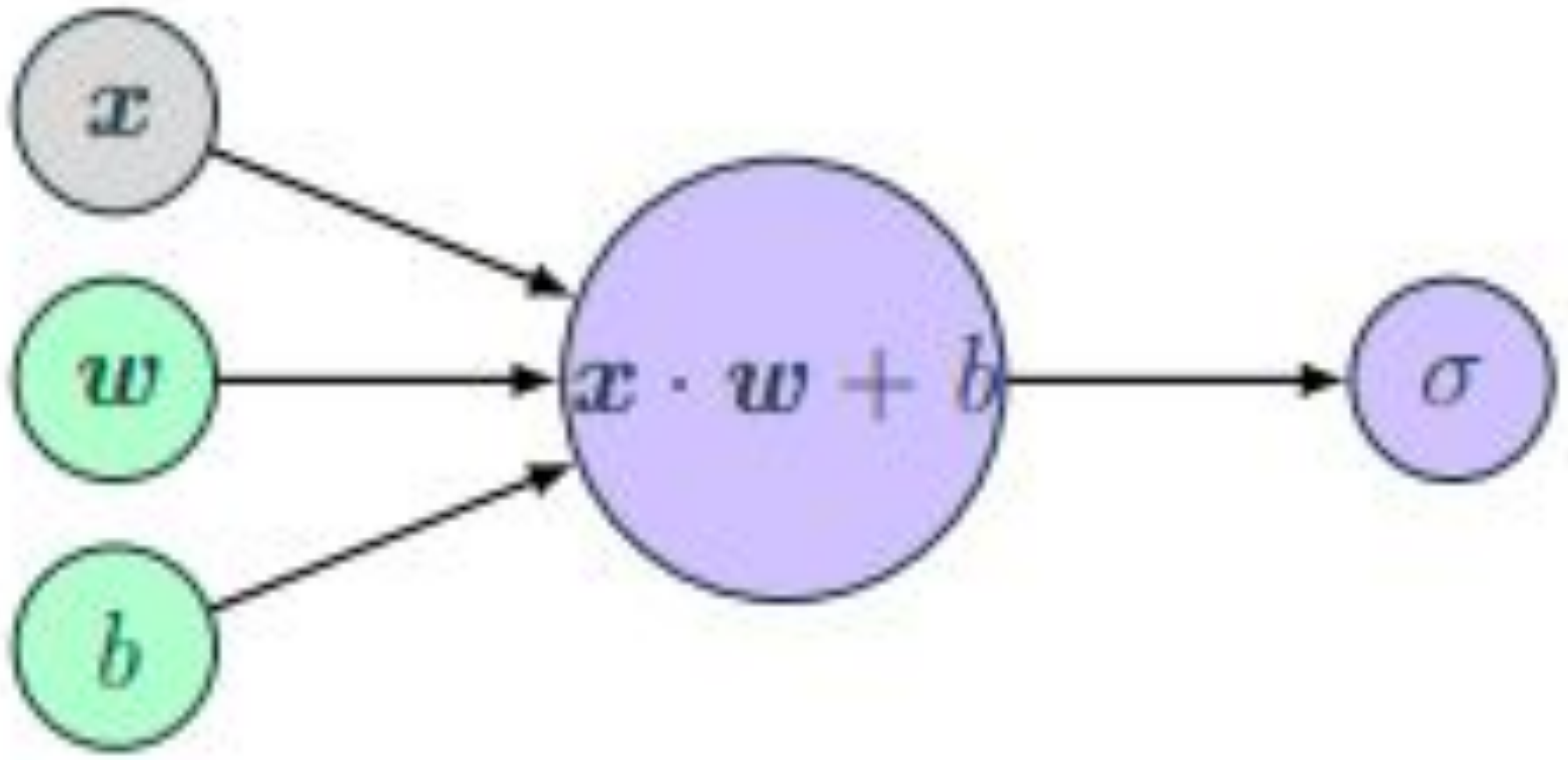
Machine Learning in NLP

Simple algorithms

_ Linear function through sigmoid — log-linear model

- Green circles are trainable parameters
- Gray circles are constant inputs

$$\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))}$$



Machine Learning in NLP

Simple algorithms

_ Decision rule of log-linear model

- Prediction = 1 if $\hat{\boldsymbol{y}}$ > 0.5
- Prediction = 0 otherwise

Natural interpretation:

- Conditional probability of prediction = 1 given the input $\boldsymbol{x}$

$$\sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 1|\boldsymbol{x})$$
$$1 - \sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 0|\boldsymbol{x})$$

Machine Learning in NLP

Simple algorithms

_ Binary Text Classification

- Finding the best model parameters
  - The loss function
    - Quantifies the loss suffered when predicting $\hat{y}$ while the true label is y for a single example.
      - $L(\hat{y}, y) : R^2 \rightarrow R$
    - Given a labeled training set ($x_{1:n}$, $y_{1:n}$), a per-instance loss function L and a parameterized function $f(x; \Theta)$ we define the corpus-wide loss with respect to the parameters Θ as the average loss over all training examples:

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

Machine Learning in NLP

Simple algorithms

_ Training as optimization

$$\mathcal{L}(\Theta) = \frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i;\Theta), y_i)$$

- The training examples are fixed, and the values of the parameters determine the loss
  - The goal of the training algorithm is to set the values of the parameters ***Θ***, such that the value of ***L*** is minimized

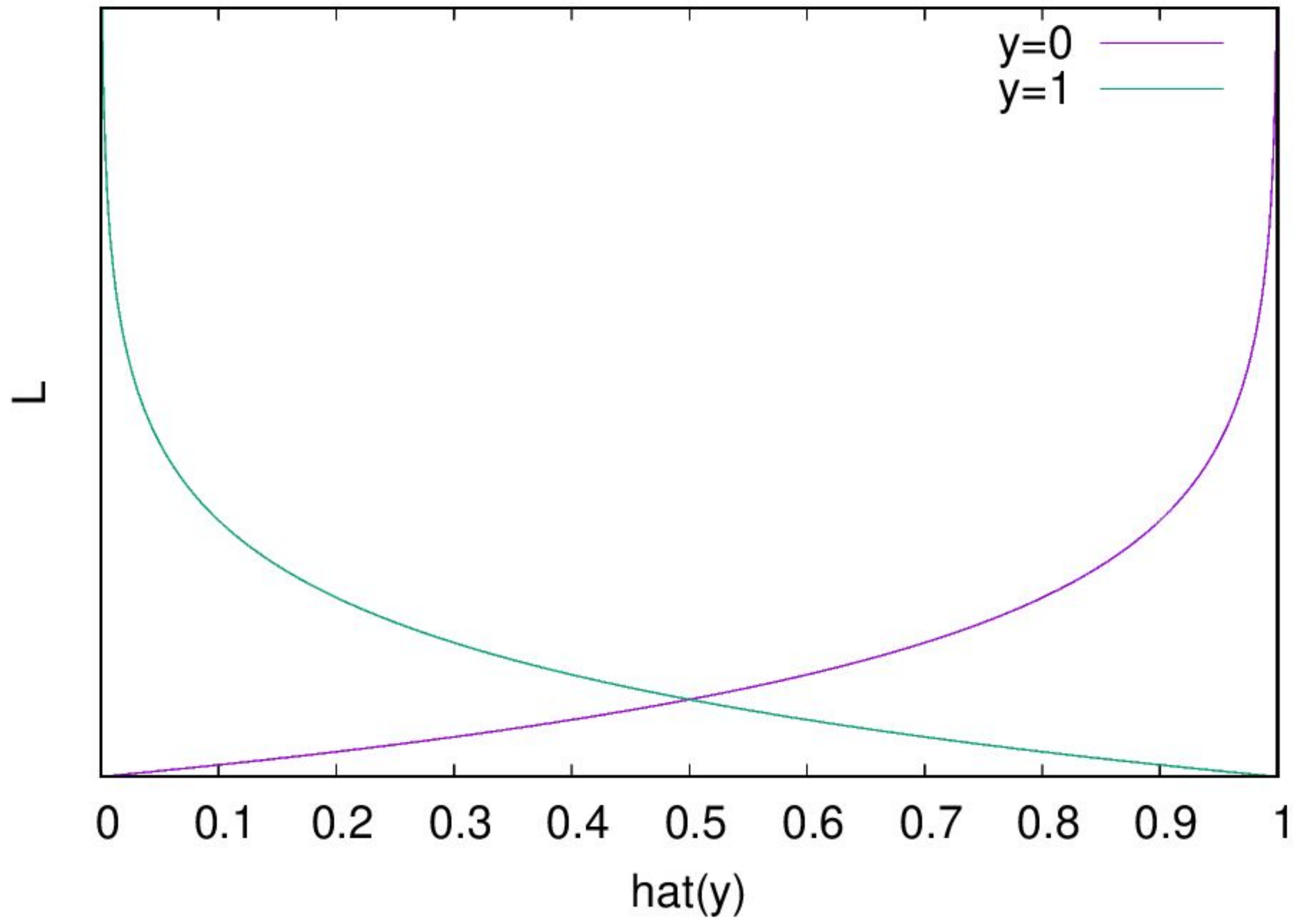
$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}(\Theta) = \underset{\Theta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i;\Theta), y_i)$$

Machine Learning in NLP

Simple algorithms

Training as optimization

- Binary cross-entropy loss (logistic loss)
  - $L_{logistic} = \boldsymbol{-y \log \hat{y} - (1 - y) \log(1 - \hat{y})}$

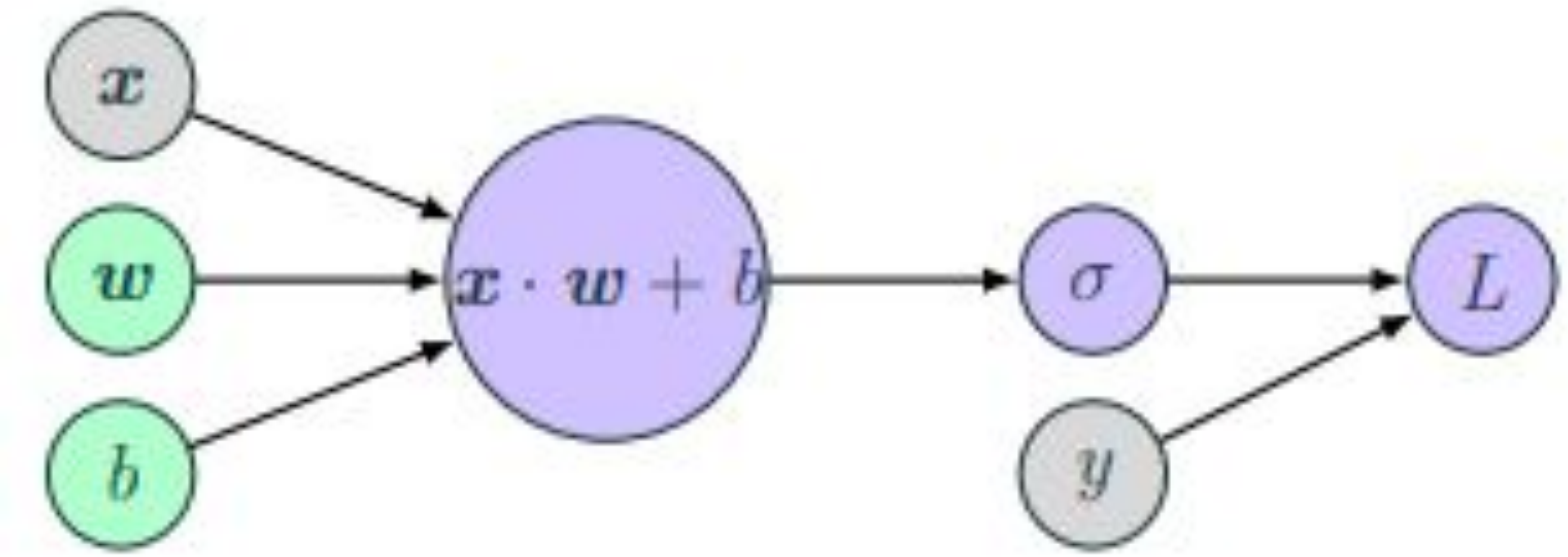


Machine Learning in NLP

Simple algorithms

_ Logistic regression

- Green circles are trainable parameters
- Gray circles are constant inputs



How can we minimize this function?

Machine Learning in NLP

Simple algorithms

_ Stochastic Gradient Descent

function SGD($\boldsymbol{f(x;\Theta)}, \boldsymbol{(x_1, \ldots, x_n)}, \boldsymbol{(y_1, \ldots, y_n)}, \boldsymbol{L}$)

while stopping criteria not met do

Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$

Compute the loss $\boldsymbol{L(f(x_i;\Theta), y_i)}$

$\boldsymbol{\hat{g}} \leftarrow$ gradient of $\boldsymbol{L(f(x_i;\Theta), y_i)}$ wrt. $\boldsymbol{\Theta}$

$\boldsymbol{\Theta \leftarrow \Theta - \eta\hat{g}}$

return $\boldsymbol{\Theta}$

Machine Learning in NLP

Simple algorithms

_ Minibatch Stochastic Gradient Descent

function mbSGD($\boldsymbol{f(x; \Theta), (x_1, \ldots, x_n), (y_1, \ldots, y_n)}$, $\boldsymbol{L}$)

while stopping criteria not met do

Sample $\boldsymbol{m}$ examples $\{\boldsymbol{(x_1, y_1), \ldots (x_m, y_m)}\}$

$\boldsymbol{\hat{g}} \leftarrow 0$

for i = 1 to $\boldsymbol{m}$ do

Compute the loss $\boldsymbol{L(f(x_i ; \Theta), y_i)}$

$\hat{g} \leftarrow \hat{g}$ + gradient of 1/m $\boldsymbol{L(f(x_i ; \Theta), y_i)}$ wrt. $\boldsymbol{\Theta}$

$\boldsymbol{\Theta \leftarrow \Theta - \eta \hat{g}}$

return $\boldsymbol{\Theta}$

Machine Learning in NLP

Simple algorithms

_ Properties of Minibatch Stochastic Gradient Descent

- The minibatch size can vary in size from $\boldsymbol{m} = 1$ to $\boldsymbol{m} = n$
  - Higher values provide better estimates of the corpus-wide gradients
  - Smaller values allow more updates and in turn faster convergence

Machine Learning in NLP

Simple algorithms

Log-linear multi-class classification

- So far we mapped our gold label $\boldsymbol{y} \in \{0, 1\}$
  - What if we classify into distinct categorical classes?
    - Categorical: There is no 'ordering'
      - Example: Classify the language of a document into 6 languages (En, Fr, De, It, Es, Other)
- Possible solution: Six weight vectors and biases
  - Consider for each language $\boldsymbol{l} \in$ {En, Fr, De, It, Es,Other}
    - Weight vector $\boldsymbol{w}^{\boldsymbol{l}}$ (e.g., w^{Fr})
    - Bias $\boldsymbol{b}^{\boldsymbol{l}}$ (e.g., b^{Fr})
      - We can predict the language resulting in the highest score

$$\hat{y} = f(\boldsymbol{x}) = \underset{\ell \in \{\text{En},\text{Fr},\text{De},\text{It},\text{Es},\text{Other}\}}{\operatorname{argmax}} \boldsymbol{x} \cdot \boldsymbol{w}^{\ell} + b^{\ell}$$

Machine Learning in NLP

Simple algorithms

Another solution

- We can re-arrange the $\boldsymbol{w} \in \boldsymbol{R}^{din}$ vectors into columns of a matrix $\boldsymbol{W} \in \boldsymbol{R}^{din \times 6}$ and $\boldsymbol{b} \in \boldsymbol{R}^{6}$, to get $\boldsymbol{f(x) = xW + b}$
  - Project the input $\boldsymbol{x}$ to an output $\boldsymbol{y}$
    - $\hat{y} = f(x) = xW + b$
    - Pick the element of $\hat{y}$ with the highest value prediction = $\hat{y}$ = $\text{argmax}_i\ \hat{y}[i]$

- What is $\hat{y}$?
  - Index of 1 in the one-hot:
    - For example, if $\hat{y} = 3$, then the document is in German De = $(0\ 0\ 1\ 0\ 0\ 0)$

Machine Learning in NLP

Simple algorithms

_ A representation of the input document

- Vector ***x*** is a document representation
    - Bag of words
        - For example (d_{in} = |V| dimensions, sparse)
        - Vector ŷ is more compact (only 6 dimensions)

	En	Fr	De	It	Es	Ot
a	•	•	•	•	•	•
at	•	•	•	•	•	•
...						
zoo	•	•	•	•	•	•

Machine Learning in NLP

Simple algorithms

_ A representation of the input document

- Vector ***x*** is a document representation
  - Bag of words
    - For example (d_{in} = |V| dimensions, sparse)
    - Vector ŷ is more compact (only 6 dimensions)

	En	Fr	De	It	Es	Ot
a	●	●	●	●	●	●
at	●	●	●	●	●	●
...						
zoo	●	●	●	●	●	●

- Representations are central to deep learning
  - One could argue that the main power of deep-learning is the ability to learn good representations

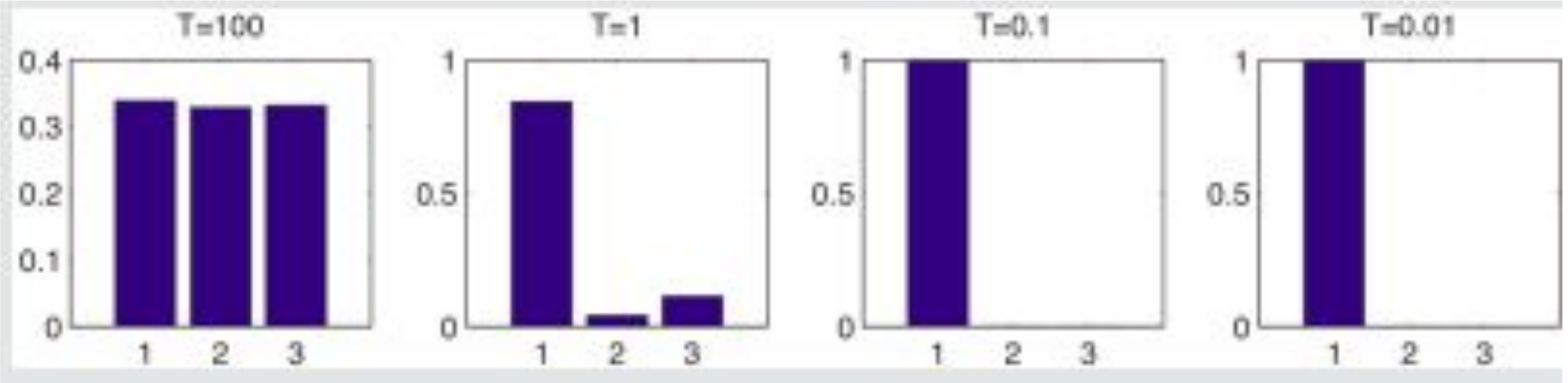
Machine Learning in NLP

Simple algorithms

_ Turning output vector into probabilities of classes

- Softmax
  - Applied element-wise, for each output value x[i]
  - Softmax can be smoothed with a 'temperature' ***T***
    - High temperature → uniform distribution
    - Low temperature → 'spiky' distribution, all mass on the largest element

$$\mathrm{softmax}(\boldsymbol{x}_{[i]};T) = \frac{\exp\left(\frac{\boldsymbol{x}_{[i]}}{T}\right)}{\sum_{k=1}^{K}\exp\left(\frac{\boldsymbol{x}_{[k]}}{T}\right)}$$



Machine Learning in NLP

Simple algorithms

_ Loss function for softmax

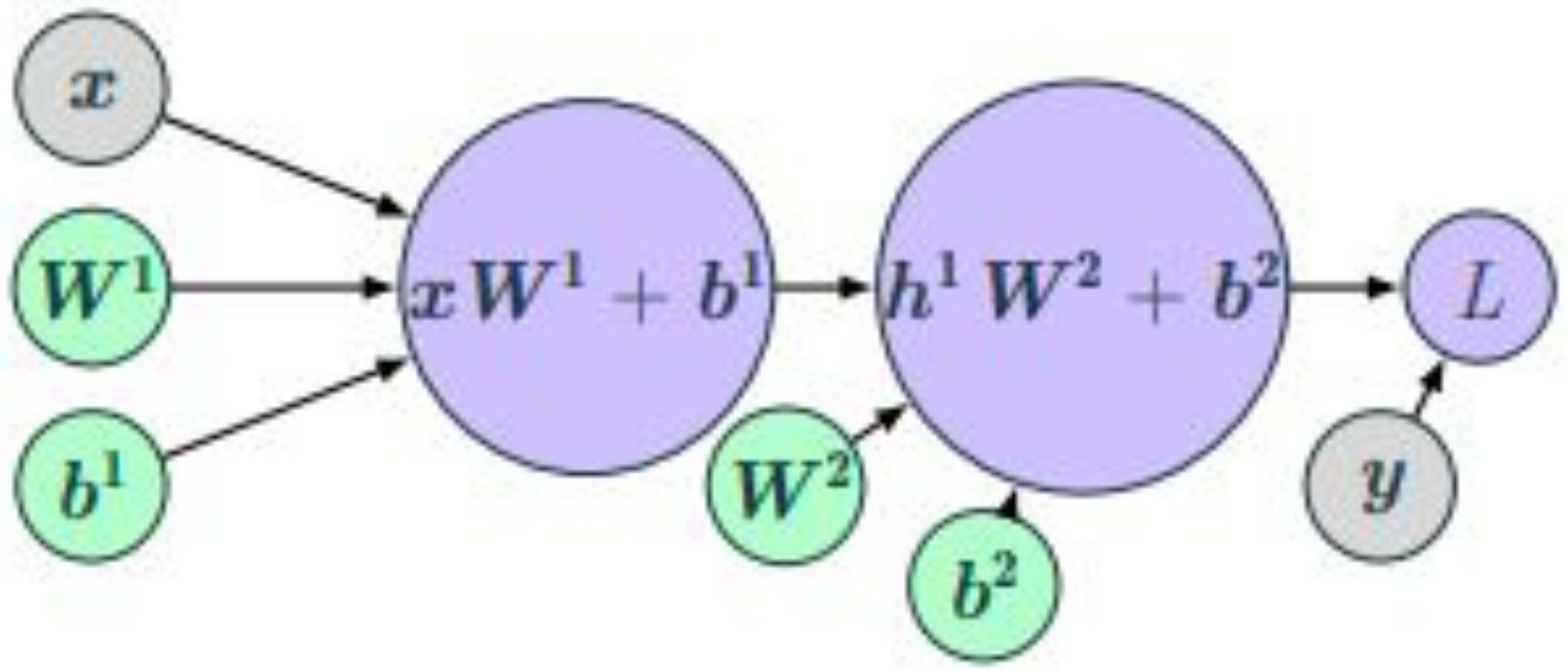
- Categorical cross-entropy loss (aka. negative log likelihood)
  - Vector representing the gold-standard categorical distribution over the labels
    - $y = (y_{[1]}, y_{[2]}, \ldots, y_{[K]})$
  - Output of softmax
    - $\hat{y} = (\hat{y}_{[1]}, \hat{y}_{[2]}, \ldots, \hat{y}_{[K]})$
      - That is, $\boldsymbol{\hat{y}_{[i]}} = \Pr(\boldsymbol{y=i|x})$
  - Cross-entropy loss
    - $L_{\text{cross-entropy}}(\hat{y}, y) = -\Sigma_{k=1}\, y_{[k]} \log(\hat{y}_{[k]})$

Machine Learning in NLP

Simple algorithms

_ Transformations and non-linearities

- Green circles are trainable parameters
- Gray circles are constant inputs

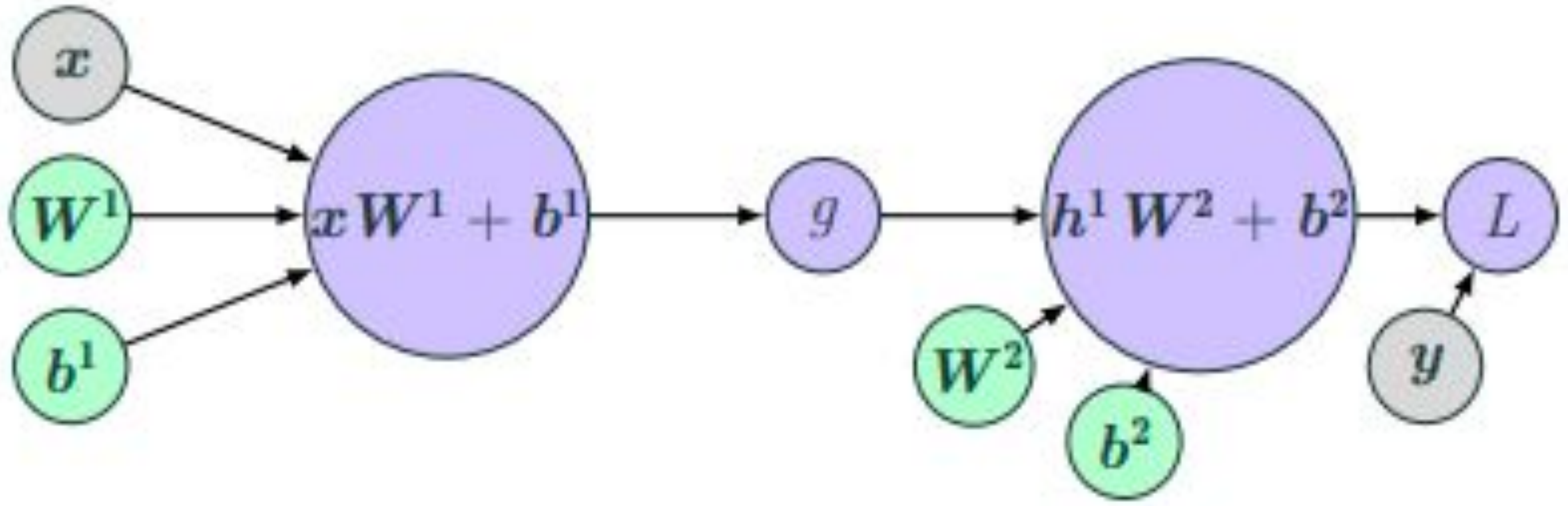


Machine Learning in NLP

Simple algorithms

_ Transformations and non-linearities

- Non-linear function ***g***
  - Rectified linear unit (ReLU) activation
    - a função de ativação ReLU e outras não-linearidades desempenham um papel crucial, permitindo que as redes profundas capturem relações complexas e não lineares nos dados e superem as limitações dos modelos lineares.



$$f(x) = g\,(xW^1 + b^1)\,W^2 + b^2$$

Language Models

Word embeddings

‘Classical’ Language models

- Assign a probability to sentences in a language
  - Example:
    - “What is the probability of seeing the sentence the lazy dog barked loudly?”
- Assigns a probability for the likelihood of given word (or a sequence of words) to follow a sequence of words
  - Example:
    - “What is the probability of seeing the word barked after the seeing sequence the lazy dog?

Language Models

Word embeddings

_ Language models formally

- Sequence of words $w_{1:n} = w_1 w_2 w_3 \ldots w_n$ estimate:
  - $Pr(w_{1:n}) = Pr(w_1, w_2, \ldots, w_n)$
- We factorize the joint probability into a product
  - One factorization is very useful: left-to-right
    - $Pr(w_{1:n}) = Pr(w_1 | \text{<s>}) \, Pr(w_2 | \text{<s>}, w_1) \, Pr(w_3 | \text{<s>} \, w_1 \, w_2) \ldots Pr(w_n | \text{<s>} \, w_1 \, w_2 \, w_3 \ldots w_{n-1})$

Language Models

Word embeddings

_ Simplifications in ‘classical’ language models

- Despite factorization:
  - The last term of $Pr(w_{1:n}) = Pr(w_1 | <s>) \, Pr(w_2 | <s>, w_1) \, Pr(w_3 | <s> \, w_1 \, w_2) \dots Pr(w_n | <s> \, w_1 \, w_2 \, w_3 \dots w_{n-1})$ still depends on all previous words of the sentence
- k-th order markov-assumption
  - The next word depends only on the last k words
    - $Pr(w_i | w_{1:i-1}) \approx Pr(w_i | w_{i-k:i-1})$

Language Models

Word embeddings

_ Estimating probabilities in 'classical' language models

- Maximum Likelihood Estimation (aka. counting and dividing)

$$\hat{P}_{\text{MLE}}(W_i = w | w_{i-k:i-1}) = \frac{\#(w_{i-k} \quad w_{i-k+1} \quad \dots \quad w_{i-1} \quad w)}{\#(w_{i-k} \quad w_{i-k+1} \quad \dots \quad w_{i-1})}$$

  - What if the denominator goes to 0?

Language Models

Word embeddings

_ Evaluating language models: Perplexity

- Trained LM tells us probability of ‘sentence’ ***s***: Pr(***s***)
  - Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: 1/n
  - Then the cross-entropy of our model is:

$$\sum_{i=1}^{n} \frac{1}{n} \log\left(\frac{1}{\Pr(s_i)}\right) = \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{\Pr(s_i)}\right) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)$$

- Perplexity of LM:
  - $2^{\text{cross-entropy}}$

Language Models

Word embeddings

_ Shortcomings of n-gram language models

- Long-range dependencies
  - To capture a dependency between the next word and the word 10 positions in the past, we need to see a relevant 11-gram in the text
- Lack of generalization across contexts
  - Having observed black car and blue car does not influence our estimates of the event red car if we haven’t see it before

Language Models

Word embeddings

_ Neural Language Models

- Let’s build a neural network
    - Input:
        - a k-gram of words $w_{1:k}$
    - Desired output:
        - a probability distribution over the vocabulary ***V*** for the next word w_{k+1}

Language Models

Word embeddings

_ Embedding layer

- If the input are symbolic categorical features
  - e.g., words from a closed vocabulary
- it is common to associate each possible feature value
  - i.e., each word in the vocabulary with a ***d***-dimensional vector for some **d**
- These vectors are also parameters of the model, and are trained jointly with the other parameters

Language Models

Word embeddings

Embedding layer: Lookup operation

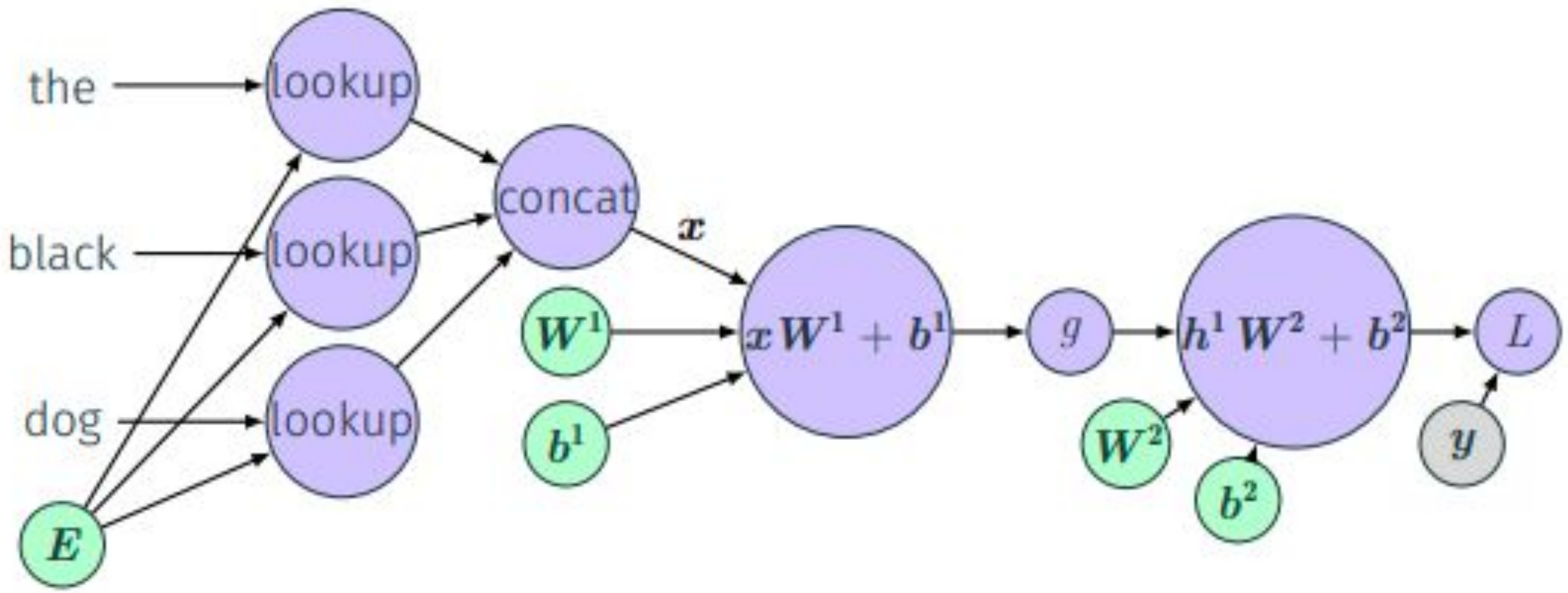
- The mapping from a symbolic feature value such as **word-number-48** to d-dimensional vectors is performed by an embedding layer (a lookup layer)
  - The parameters in an embedding layer are a matrix $W^{|V|\times d}$, each row corresponds to a different word in the vocabulary
- The lookup operation is then indexing v(w)
  - e.g., v(w) = v48 = E[48,:]

Language Models

Word embeddings

_ Example network concatenating 3 words as embeddings (d_w = 50)

- Each word $\in R^{|V|}$ (one hot), $\boldsymbol{E} \in R^{|V| \times 50}$, each lookup output $\in R^{50}$, concat output $x \in R^{150}$



Language Models

Word embeddings

_ Neural Language Models

- Let's build a neural network
  - Input:
    - a k-gram of words $w_{1:k}$
  - Desired output:
    - a probability distribution over the vocabulary ***V*** for the next word w_{k+1}
  - Each input word w_k is associated with an embedding vector $v(w) \in R^{dw}$ (dw — word embedding dimensionality)
  - Input vector ***x*** is a concatenation of k words
    - $x = [v(w_1); v(w_2); . . . ; v(w_k)]$

Language Models

Word embeddings

_ Neural Language Models

- MLP with one (or more) hidden layers
  - $v(w) = E_{w,:}$
  - $x = [v(w_1); v(w_2); \ldots ; v(w_k)]$
  - $h = g(xW^1 + b^1)$
  - $\hat{y} = Pr(W_i \mid w_{1:k}) = \text{softmax}(hW^2 + b^2)$

- Output dimension:
  - $\hat{y} \in R^{|V|}$

Language Models

Word embeddings

_ Neural Language Models

- Training neural LMs
  - Where to get training examples?
    - Training examples are simply word k-grams from an unlabeled corpus
      - Identities of the first k – 1 words are used as features
      - The last word is used as the target label for the classification
  - The model is trained using cross-entropy loss

Language Models

Word embeddings

_ Generating text with language models

- We can generate ("sample") random sentences from the model according to their probability

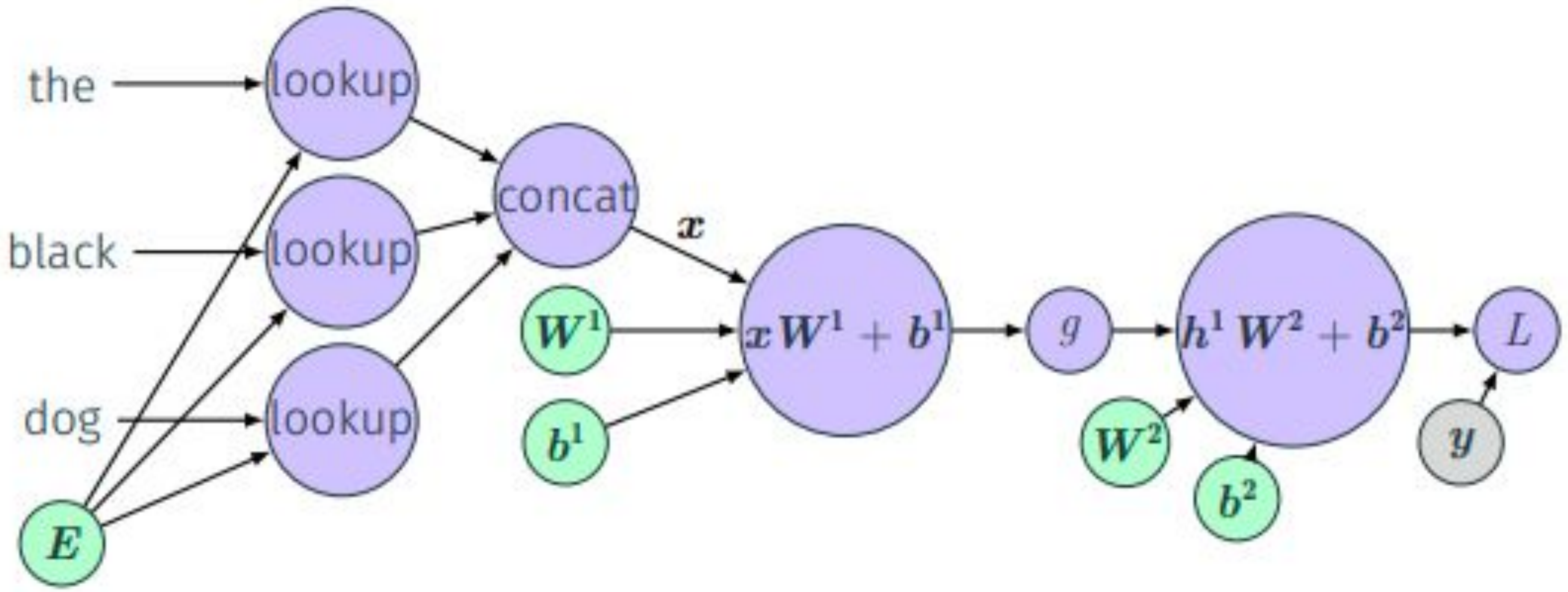
1. Predict a probability distribution over the vocabulary conditioned on the start symbol **<s>**
2. Draw a random word (the first word) according to the predicted distribution
3. Predict a probability distribution over the vocabulary conditioned on the start symbol and the first word
4. Draw a random word (the second word) according to the predicted distribution
5. Repeat until generated end-of-sentence symbol (or <EOS>)

Language Models

Word embeddings

_ Learned word representations as a by-product

- Each row of ***E*** learns a word representation
- Each column of W^2 learns a word representation



Language Models

Word embeddings

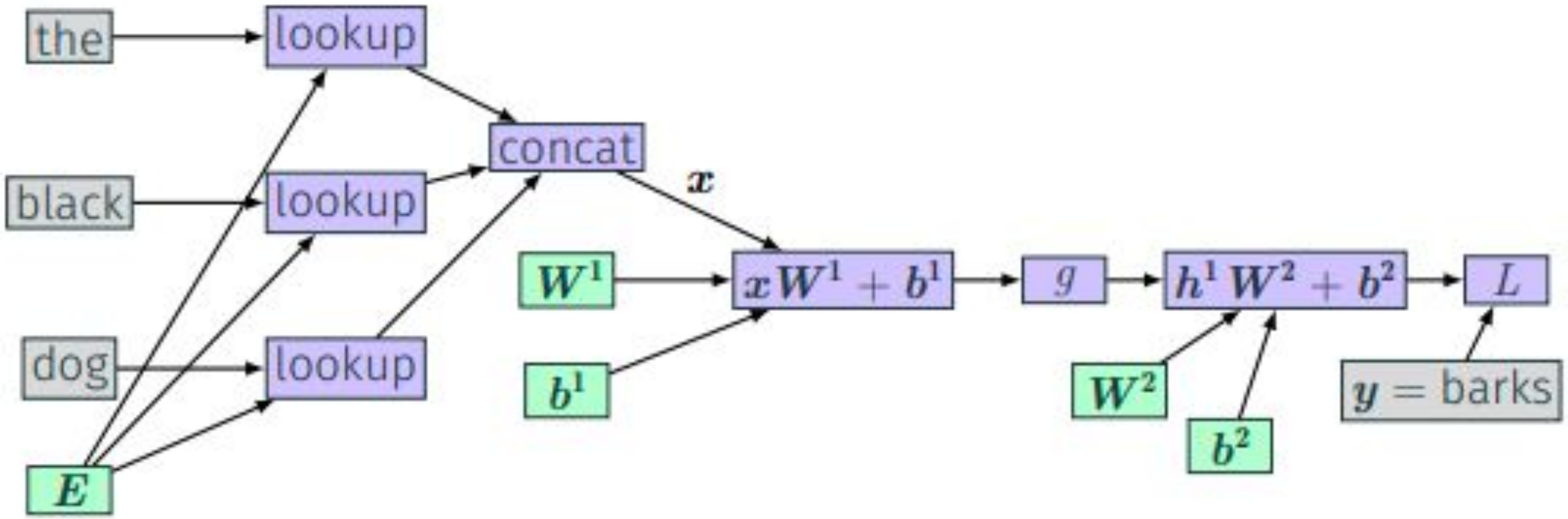
Word embeddings as pre-trained word representation

- Option A:
  - We can initialize the embeddings matrix ***E*** randomly and learn during our supervised task
- Option B:
  - Use pre-trained word embeddings from task for which we have a lot of data
    - Use self-supervised learning (create labeled data ‘for free’ using the next word prediction objective)
    - Learned word embedding matrix plugged into our supervised task
- Desired word embeddings properties: ‘Similar’ words have similar embeddings vectors

Language Models

Word embeddings

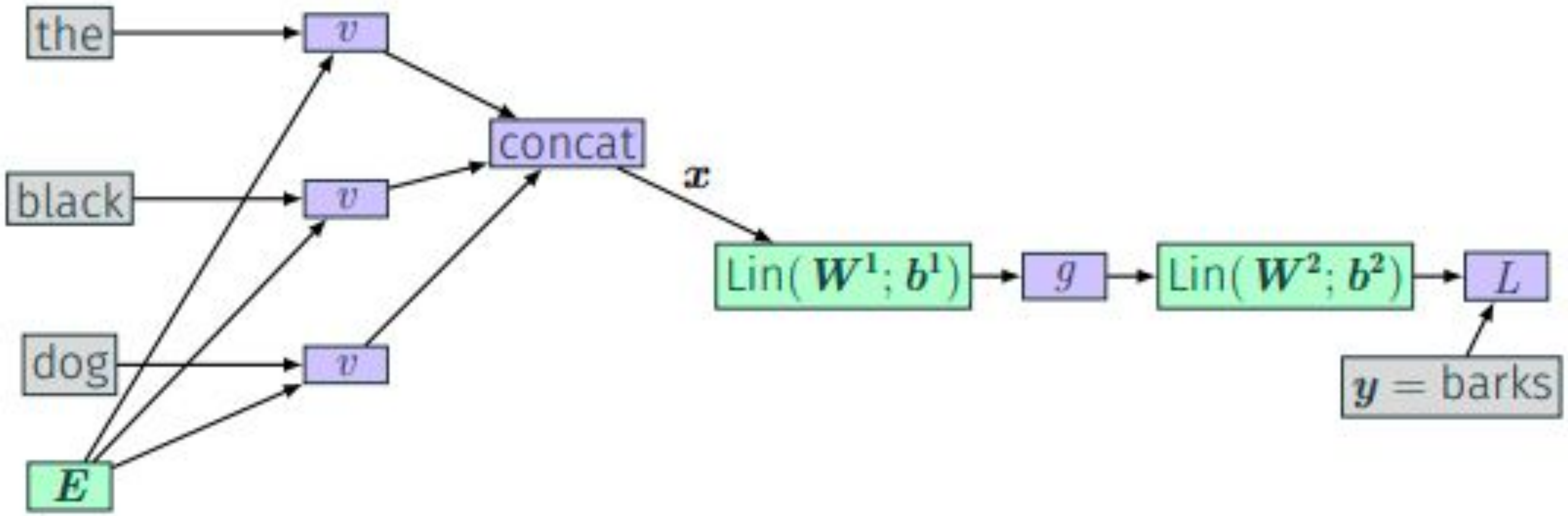
- Neural language model, context = 3 preceding words



Language Models

Word embeddings

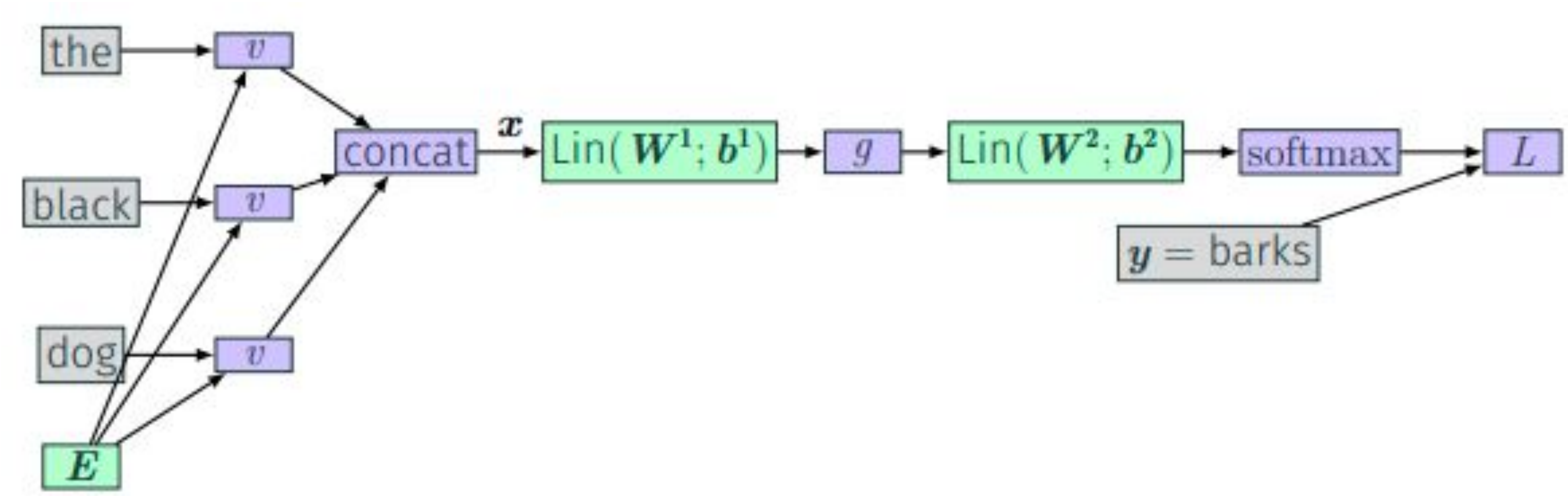
- Simplify notation: Lookup ***v***, Linear layers including parameters



Language Models

Word embeddings

- Softmax for actually predicting distribution over ***V***



Neural Language Models

Distributional hypothesis

_ Major drawbacks of one-hot encoding of words

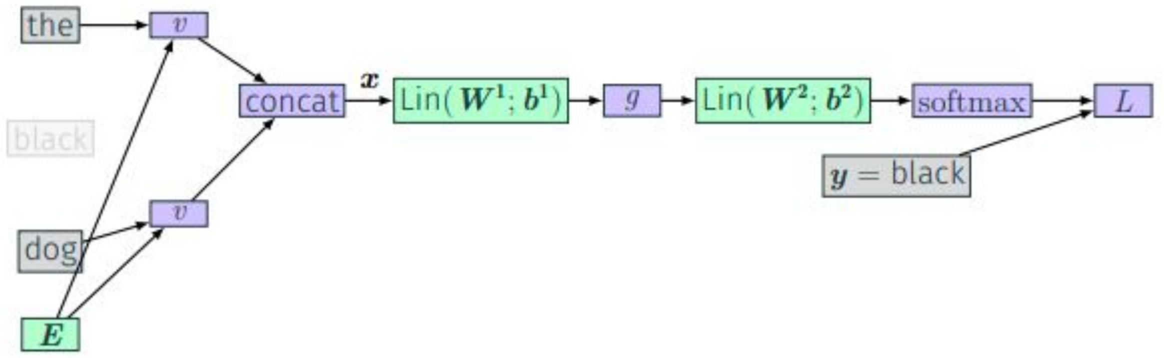
- Large, sparse representations
- No ‘semantic’ similarity, all words are equally ‘similar’
  - The distributional hypothesis states that words are similar if they appear in similar contexts
    - Intuitively, when we encounter a sentence with an unknown word such as the word **wampinuk**
      - “Marco saw a hairy little wampinuk crouching behind a tree”
    - We infer the meaning of the word based on the context in which it occurs

Neural Language Models

Word embeddings

Neural language model's goal:

- Predict probability distribution over ***V*** for the next word conditioned on the previous words
  - Side product:
    - Can learn useful word embeddings
      - What if we don't need probability distribution but just want to learn word embeddings?
        - We can relax our Markov assumption of 'look at k previous words only'
        - We can get rid of the costly normalization in softmax



Language Models

Word embeddings

Simplification 1:

- No need of a Markov property
  - For example, instead of modeling $\Pr(w_3|w_1, w_2, -)$, we model $\Pr(w_2|w_1, -, w_3)$

Language Models

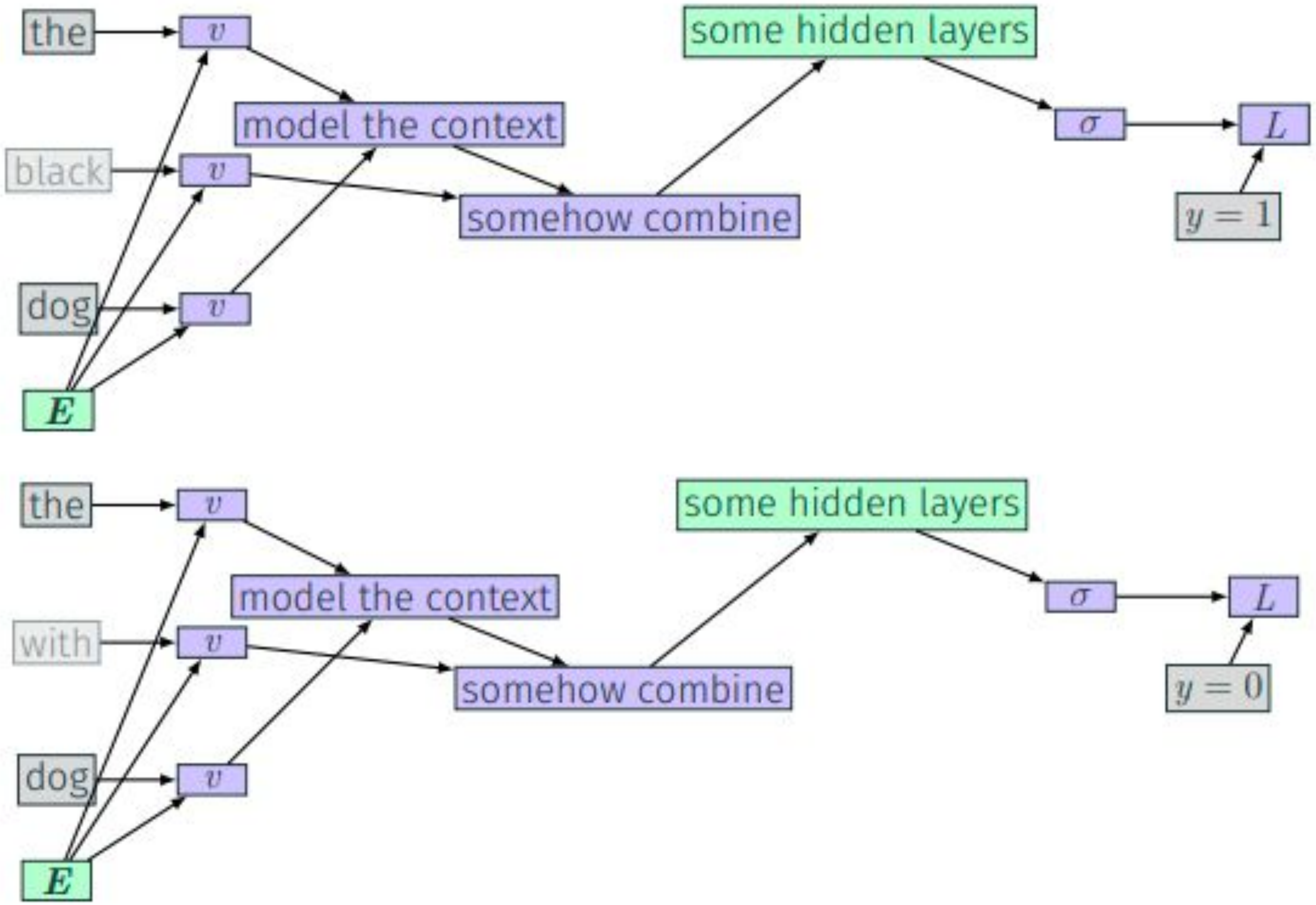
Word embeddings

_ Simplification 2:

- Give up the costly softmax probability distribution
  - Instead of predicting probability distribution, we just want to predict some score of context and target word
    - What could such a score be?
      - Prefer words in their true contexts (high score)
      - Penalize words in their ‘untrue’ contexts (low score)
- Instead of predicting probability distribution for the target word, we create an artificial binary task by randomly shuffling the target word ***w***
  - ***y = 1***, if it is a positive context
  - ***y = 0***, otherwise

Language Models

Word embeddings

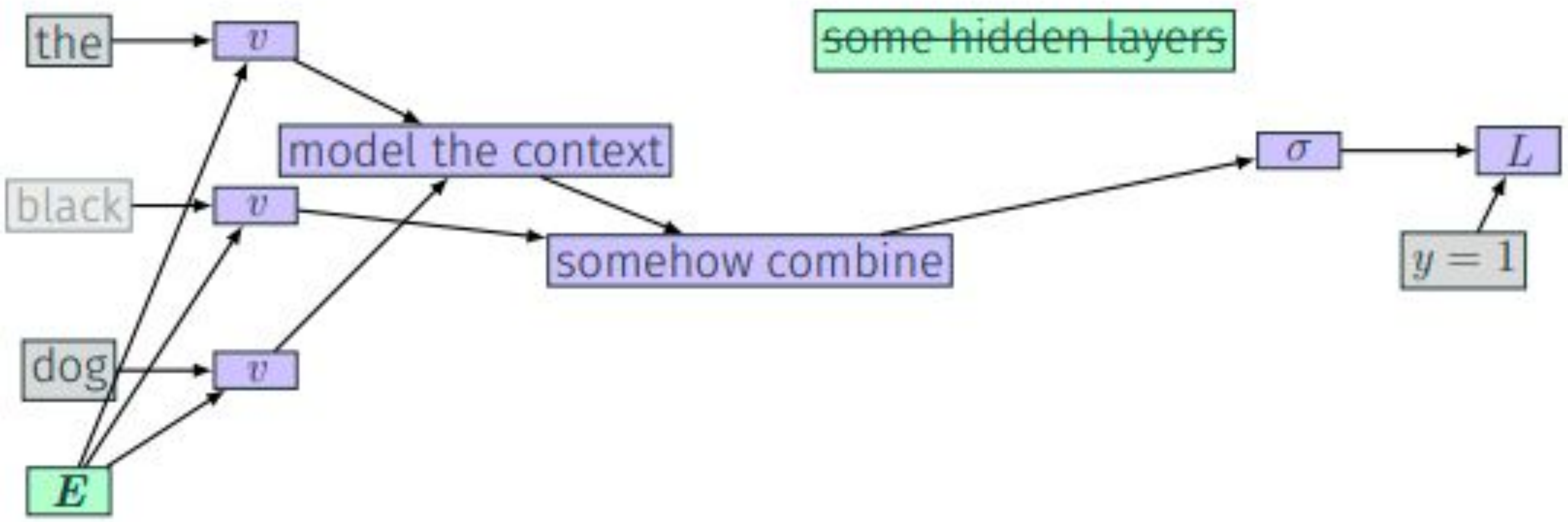


Language Models

Word embeddings

word2vec

- word2vec simplifies the neural LM by removing the hidden layer
  - So turning it into a log-linear model!

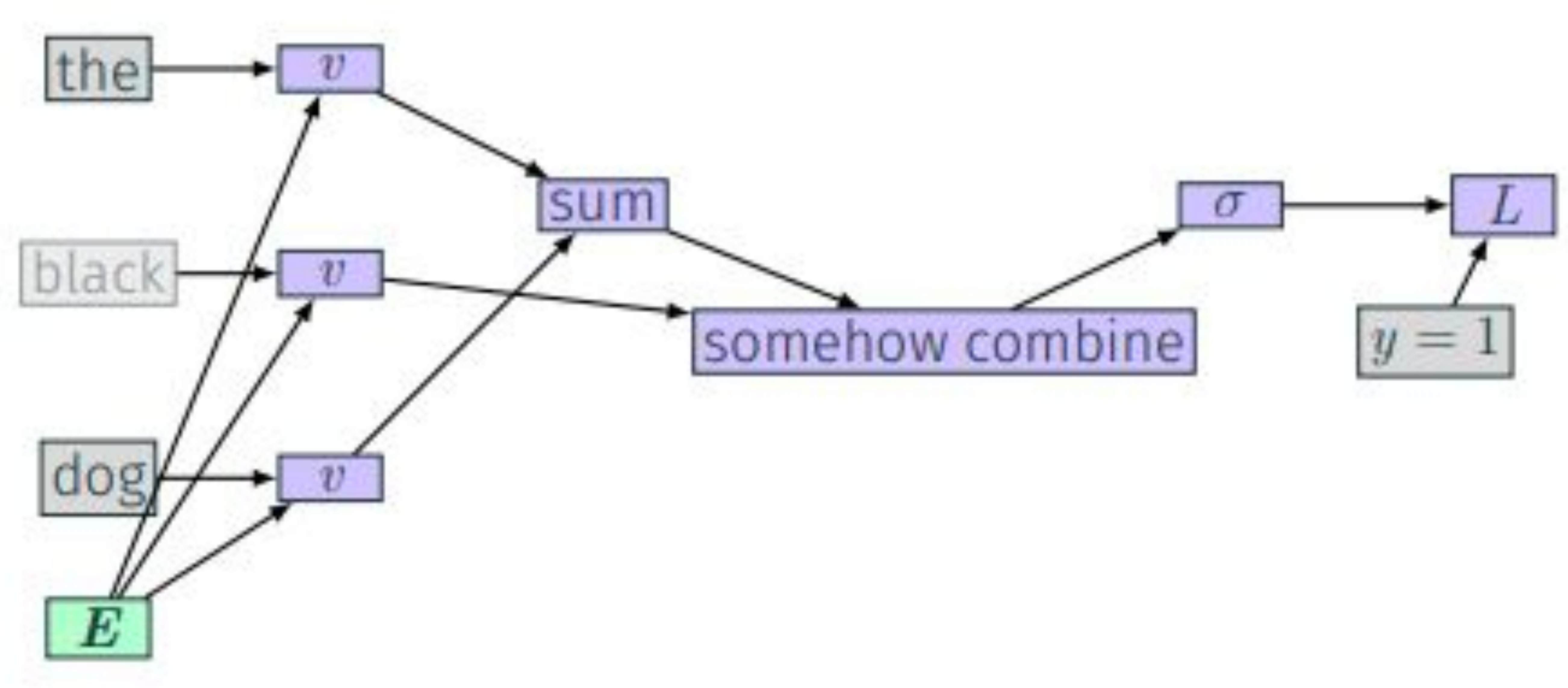


Language Models

Word embeddings

_ word2vec:

- How to model the context?
  - Continuous bag of words (CBOW) c = summation of the context vectors



Language Models

Word embeddings

_ word2vec: Learning useful word embeddings

- Train the network to distinguish ‘good’ word-context pairs from ‘bad’ ones
  - Create a set $\boldsymbol{D^+}$ of correct word-context pairs and set $\boldsymbol{D^-}$ of incorrect word-context pairs
  - The goal of the algorithm is to estimate the probability $\boldsymbol{Pr(D = 1|w, c)}$ that the word-context pair $\boldsymbol{w}$, $\boldsymbol{c}$ comes from the correct set $\boldsymbol{D^+}$
    - This should be high (1) for for pairs from $\boldsymbol{D^+}$ and low (0) for pairs from $\boldsymbol{D^-}$

Language Models

Word embeddings

- Maximize the log-likelihood of the data $D^+ \cup D^-$

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log \Pr(D=1|w,c) + \sum_{(w',c)\in \bar{D}} \log \Pr(D=0|w',c)$$

  - In word2vec Skip-Gram, for each correct word/context, sample ***k*** negative pairs into D^- , so D^- is k-times larger than D^+
    - ***k*** is a hyper-parameter

Language Models

Word embeddings

_ FastText embedding

- Popular word embedding models ignore the morphology of words, by assigning a distinct vector to each word
    - Limitation, especially for languages with large vocabularies and many rare words
        - Model each word as a bag of character n-grams
        - Each character n-gram has its own embedding
        - Word is represented as a sum of n-gram embeddings
    - Example:
        - Extract all the character n-grams for 3 ≤ n ≤ 6
            - eating → G_w = {<ea, eat, ati, tin, ing, ng>, <eat, eati, atin, ing>, <eati, eatin, ating, ting>, <eatin, eating, ating>}
                - v(eating) = summation of the vectors of all n-grams

Language Models

Word embeddings

_ Advantages of word2vec

- Pre-trained embeddings:
  - ‘Semantic’ input to any neural network instead of one-hot word encoding
  - Instance of transfer learning
    - pre-trained (self-trained) on an auxiliary task, plugged into a more complex model as pre-trained weights
      - Example: Represent a document as an average of its words’ embeddings (average bag-of-words through embeddings) for text classification
        - Semantic similarity, short document similarity, query expansion

Language Models

Word embeddings

_ Finding word analogies with word2vec

- 'Germany to Berlin is France to ?'
  - Solved by v(Berlin) – v(Germany) + v(France)
    - outputs vector ***x*** which is closest to Paris in the embeddings space (the closest row in ***E***)
- Find the queen
  - v(king) – v(man) + v(woman) ≈ v(queen)

Language Models

Word embeddings

_ Limitations of word2vec

- Definition of similarity
  - Completely operational: words are similar if used in similar contexts
- Antonyms
  - Words opposite of each other (buy—sell, hot—cold) tend to appear in similar contexts
    - Things that can be hot can also be cold, things that are bought are often sold
    - Models might tend to judge antonyms as very similar to each other

Language Models

Word embeddings

_ Limitations of word2vec

- Biases
  - Distributional methods reflect the usage patterns in the corpora on which they are based
    - The corpora reflect human biases in the real world (cultural or otherwise)
- Polysemy, context independent representation
  - Some words have obvious multiple senses
    - A bank may refer to a financial institution or to the side of a river
    - A star may an abstract shape, a celebrity, an astronomical entity
      - Using a single vector for all forms is problematic

First Assignment

Word embeddings

_ Intrinsic evaluation

- Objective:
  - Prepare data for modeling language
  - Learn a word2vec language model
  - Evaluate the model

First Assignment

Word embeddings

_ Intrinsic evaluation

- Code:
  - gensym (there are other implementations)
- Corpus:
  - https://mattmahoney.net/dc/text8.zip (there are other corpora)
- Pre-processing:
  - Punctuation, lower case, etc.
- Choices:
  - training sizes, window sizes, CBOW vs Skip-Gram etc.
- Evaluation:
  - Analogies using https://github.com/nicholas-leonard/word2vec/blob/master/questions-words.txt
    - input three words, pick the returned word, compute the distance to the correct word
    - Repeat and average.

First Assignment

Word embeddings

_ Help:

- Fórum de dúvidas: Guilherme Oliveira

Words → Sentences

Word Mover Distance

_ How to compare two sentences (or small documents)

- How can we capture when sentences/documents say the same thing using completely different words
  - WMD: a novel distance function between documents
    - The minimum amount of distance that the word embeddings of one document need to “travel” to reach the embeddings of the other document
      - The cost of traveling from word ***i*** to word ***j*** is:
        - **$c(i, j) = |x_i - x_j|^2$**

Words → Sentences

Word Mover Distance

_ Document distance

- Uses “travel cost” between two words to create a distance between two sentences/documents
  - We compute the cost of transforming each word ***i*** in document ***A*** in each word ***j*** in document ***B***

$$\min_{\mathbf{T}\geq 0} \sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j)$$

    - The minimum cumulative distance that all words in one document need to travel to exactly match the other document.

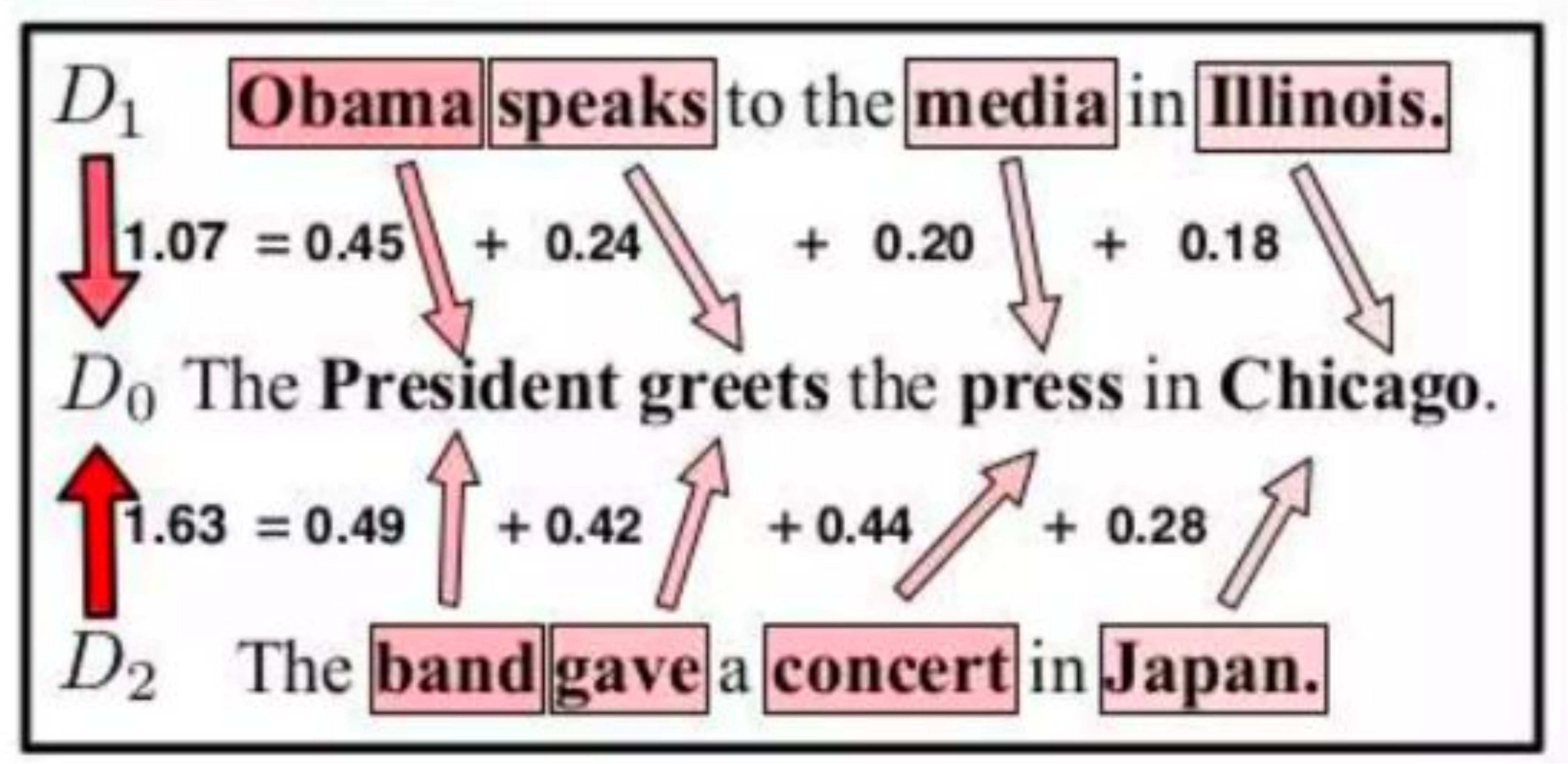
Words → Sentences

Word Mover Distance

_ Document distance

- Uses “travel cost” between two words to create a distance between two sentences/documents
    - We compute the cost of transforming each word ***i*** in document ***A*** in each word ***j*** in document ***B***

$$\min_{\mathbf{T}\geq 0} \sum_{i,j=1}^{n} \mathbf{T}_{ij} c(i,j)$$



Words → Sentences

Word Mover Distance

_ Computational complexity

- Best average time complexity: ***$O(p^3 \log p)$***, where ***p*** is the vocabulary size
  - For datasets with large vocabularies and/or large number of documents, solving the WMD optimal transportation problem becomes phroibitive

Words → Sentences

ConvNets

_ Main ConvNet idea:

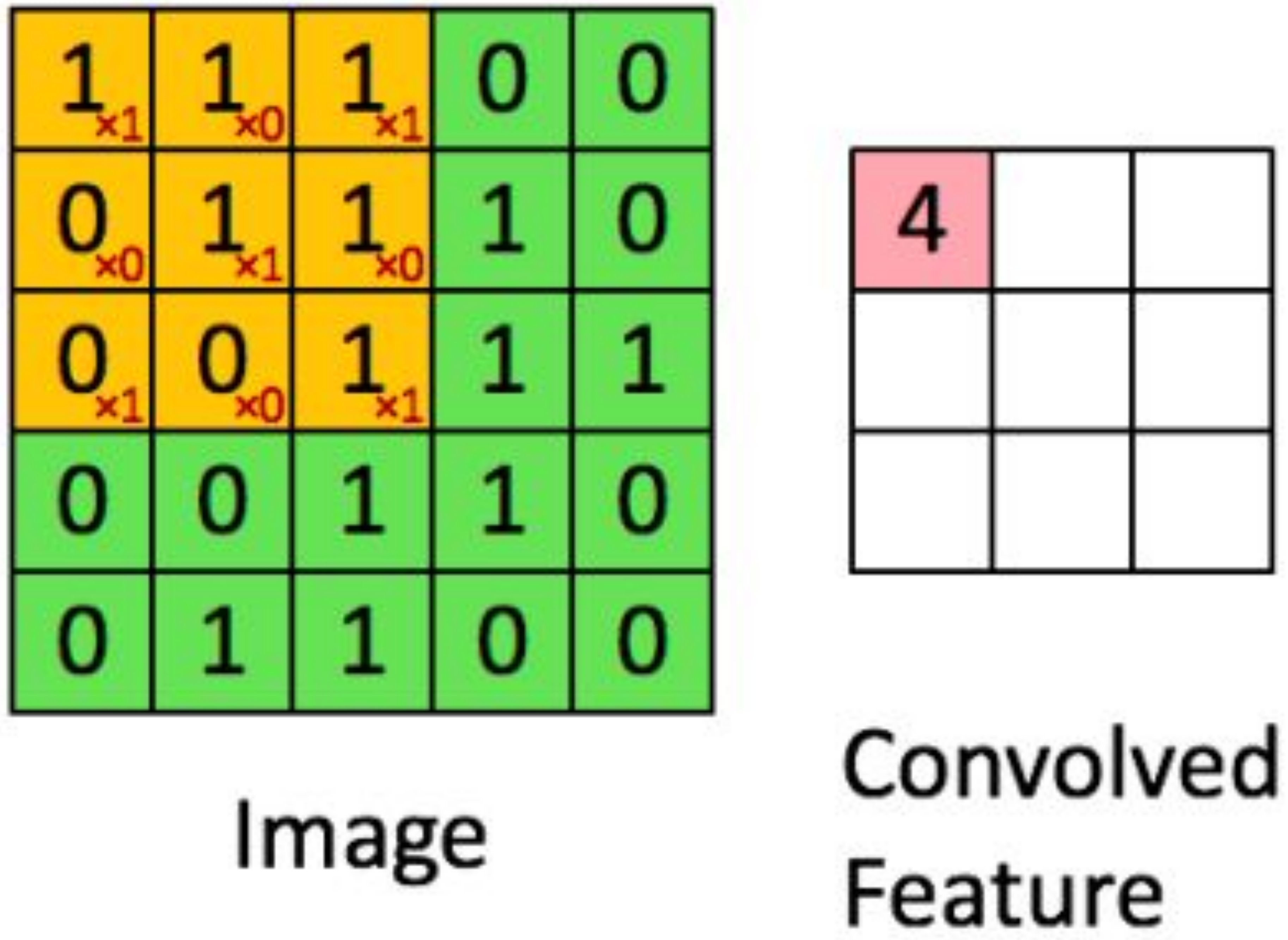
- What if we compute vectors for every possible word subsequence of a certain length?
    - Example: “tentative deal reached to keep government open”
        - computes vectors for:
            - tentative deal reached, deal reached to, reached to keep, to keep government, keep government open
            - Regardless of whether phrase is grammatical
        - Ok, not very linguistically or cognitively plausible
            - Then group them afterwards

Words → Sentences

ConvNets

_ What is convolution?

- Convolution is classically used to extract features from images
  - 2D example
    - Yellow color and red numbers show filter (=kernel) weights
    - Green shows input
    - Pink shows output



Words → Sentences

ConvNets

_ Text is 1D!

- Apply a filter (or kernel) of size 3

tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3

Words → Sentences

ConvNets

_ 1D convolution:

- Padding

ø	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
ø	0.0	0.0	0.0	0.0

3	1	2	−3
−1	2	1	−3
1	1	−1	1

ø,t,d	−0.6
t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3
g,o,ø	−0.5

Words → Sentences

ConvNets

- 3 channel 1D convolution with padding = 1

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

∅,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,∅	−0.5	−0.9	0.1

Words → Sentences

ConvNets

- conv1d, padded with max pooling over time

∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

3	1	2	−3
−1	2	1	−3
1	1	−1	1

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

max p	0.3	1.6	1.4

Words → Sentences

ConvNets

- conv1d, padded with ave pooling over time

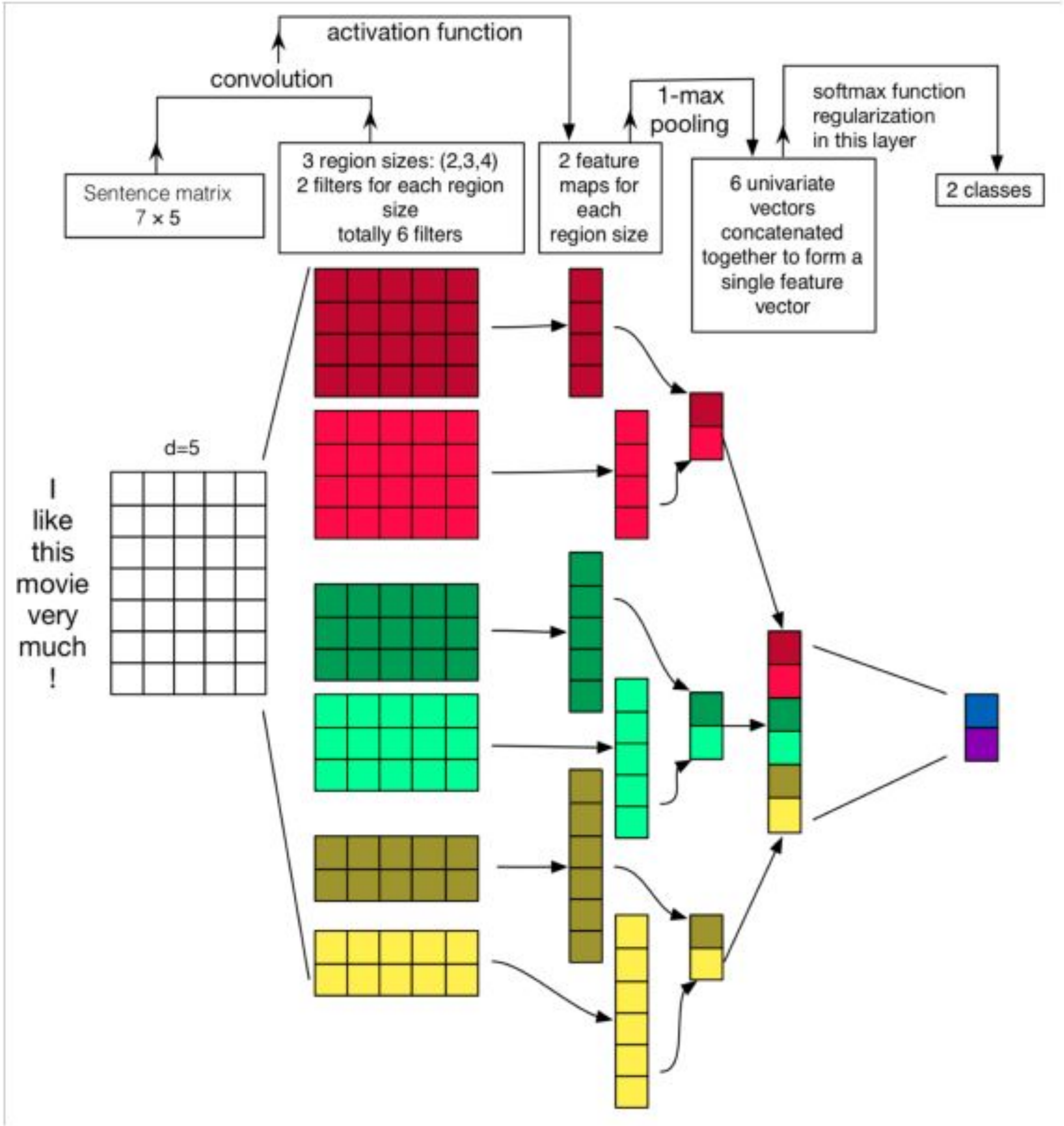
∅	0.0	0.0	0.0	0.0
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
∅	0.0	0.0	0.0	0.0

3	1	2	−3
−1	2	1	−3
1	1	−1	1

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1
ave p	−0.87	0.26	0.53

Words → Sentences

ConvNets



Words → Sentences

ConvNets

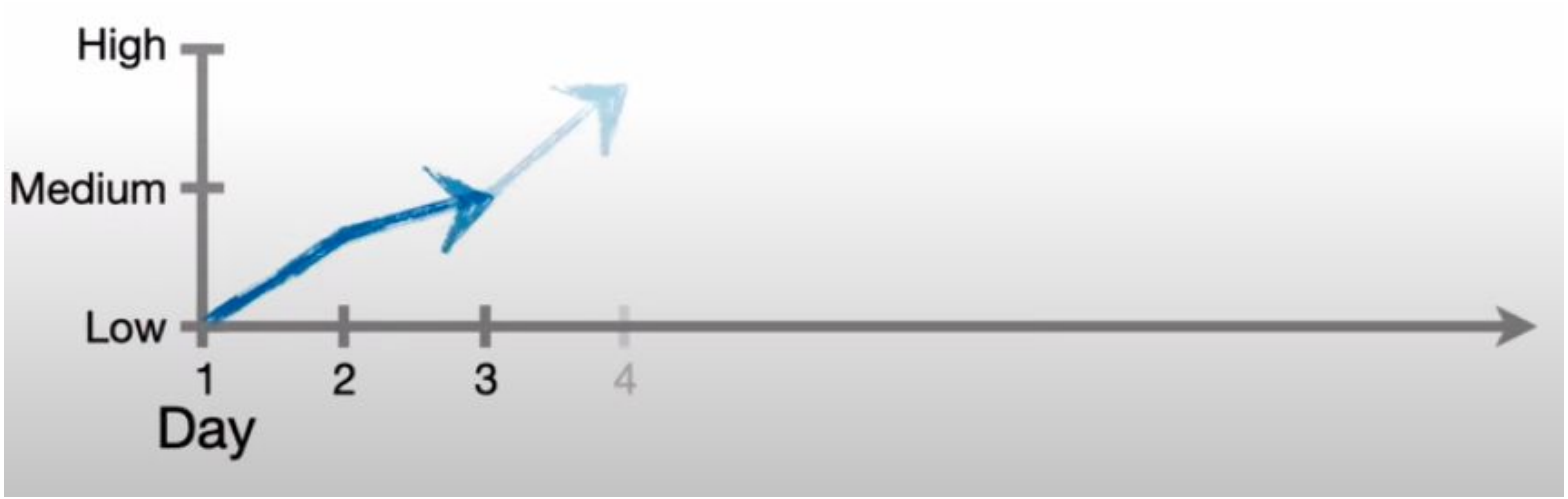
_ Training a ConvNet is a supervised task

- Therefore, it requires a supervised loss function
  - The loss function depends on the downstream application
    - Sentence classification, POS tagging, NER, Paraphrase detection, Translation etc.
  - Filters are weights!
    - Backpropagation updates filters in order to decrease the error
    - Dropout is essential

Memory

Recurrent Nets

- When we look at stock prices, they tend to change over time

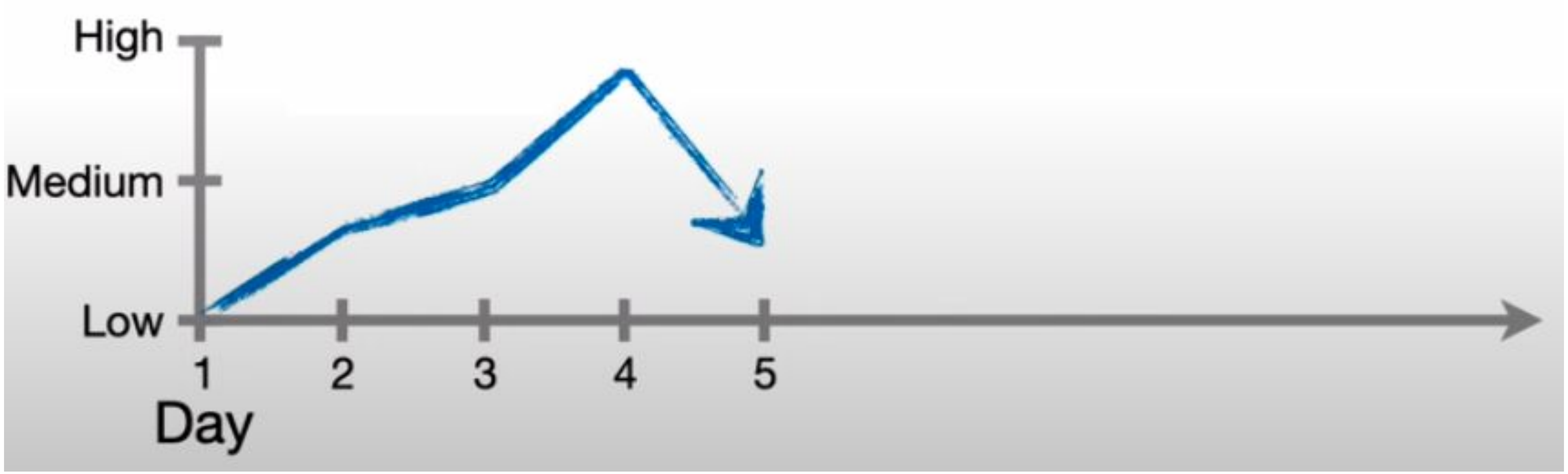


Memory

Recurrent Nets

_ When we look at stock prices, they tend to change over time

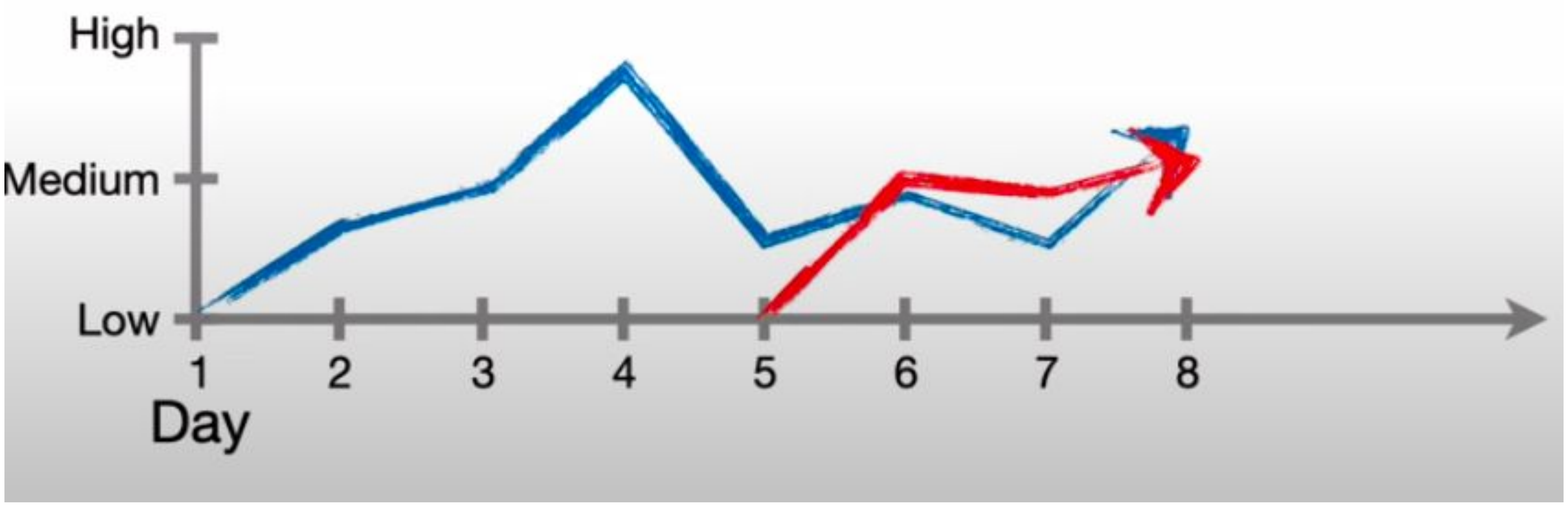
- For example the price for this stock went up for 4 days, before going down



Memory

Recurrent Nets

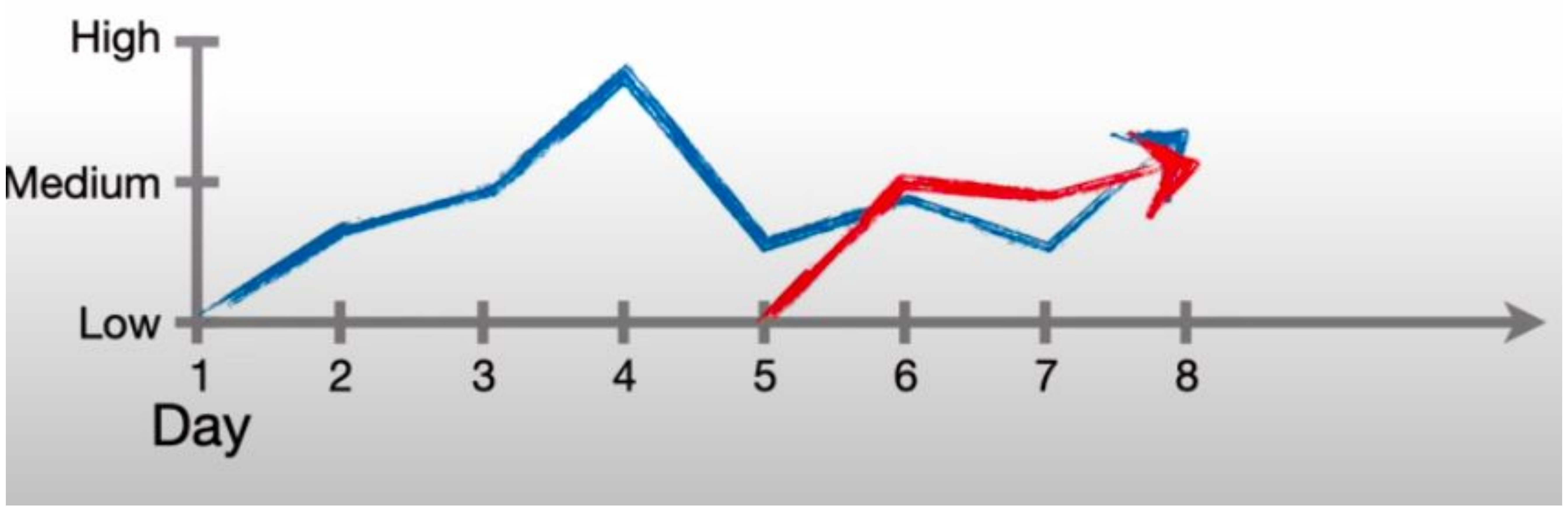
- When we look at stock prices, they tend to change over time
- Also, the longer the company has been traded on the stock market, the more data we will have for it



Memory

Recurrent Nets

- When we look at stock prices, they tend to change over time
- Also, the longer the company has been traded on the stock market, the more data we will have for it
  - So, we need a neural network that works with different amounts of data.
    - That is, how much data to remember?
    - This is very different from typical neural networks.

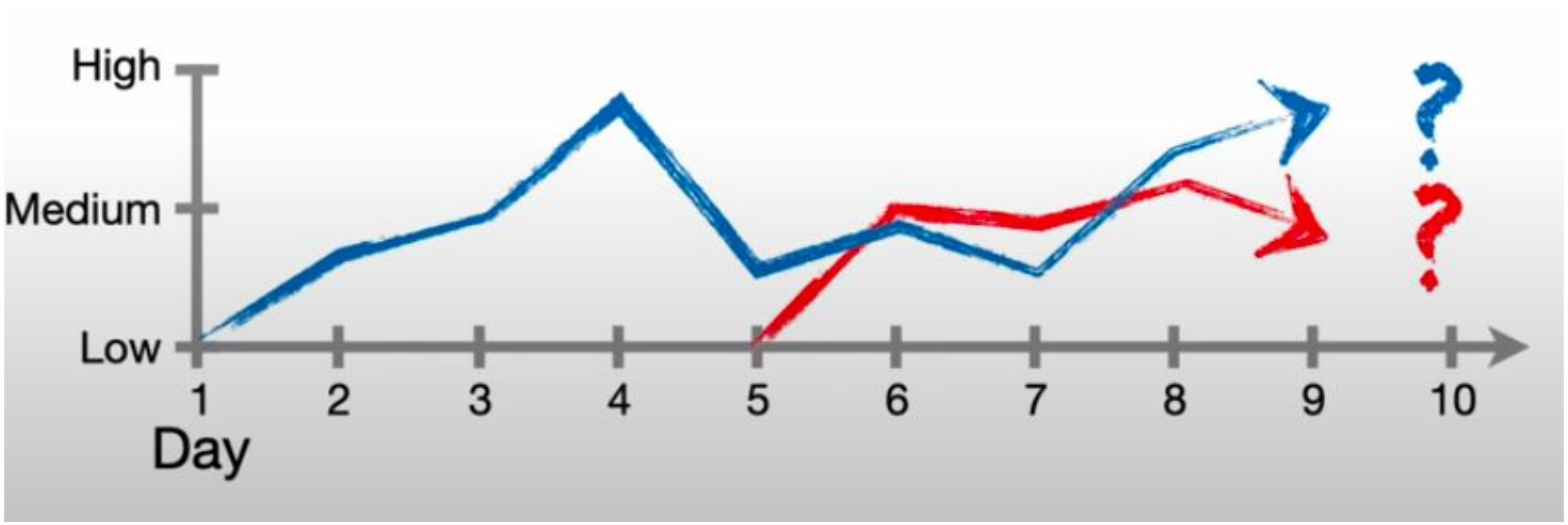


Memory

Recurrent Nets

_ When we look at stock prices, they tend to change over time

- Recurrent neural networks

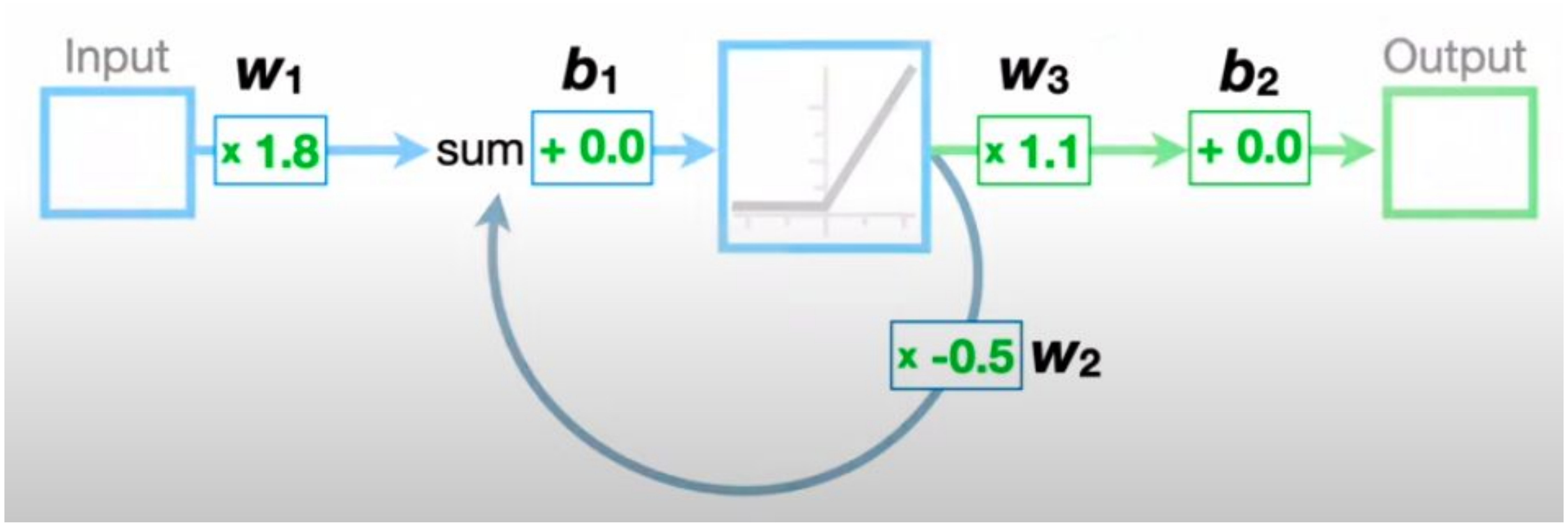


Memory

Recurrent Nets

_ Feedback loops

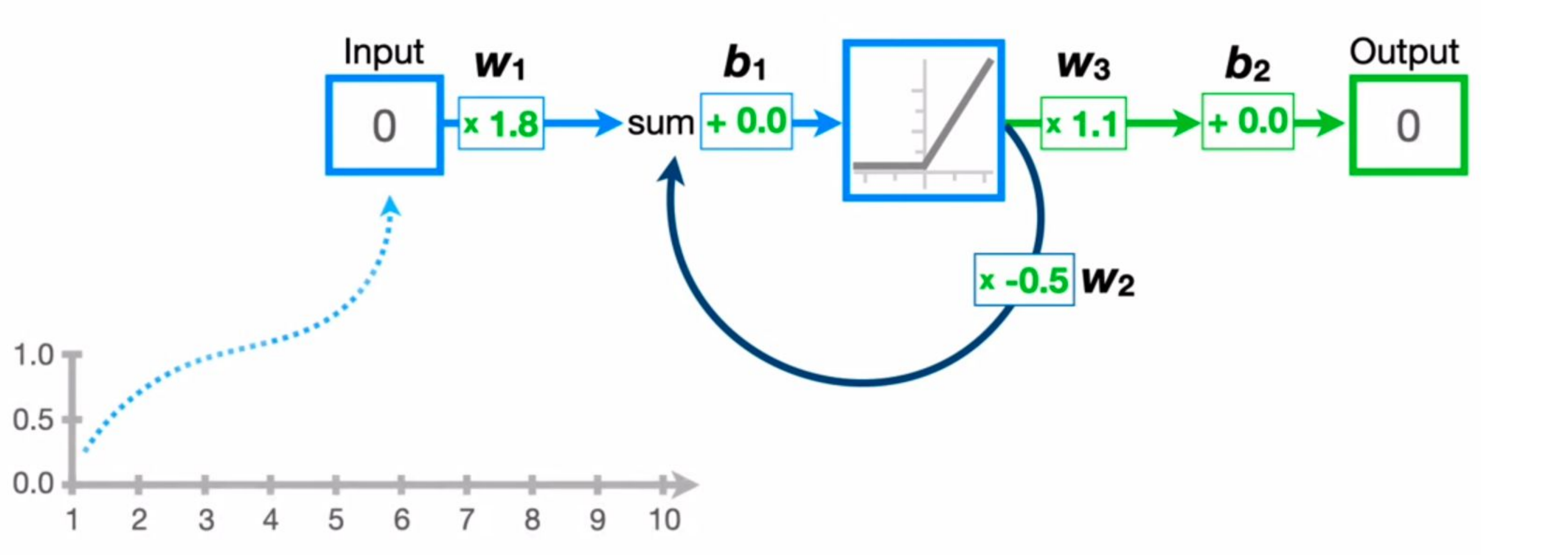
- Plus, weights, biases and activation functions



Memory

Recurrent Nets

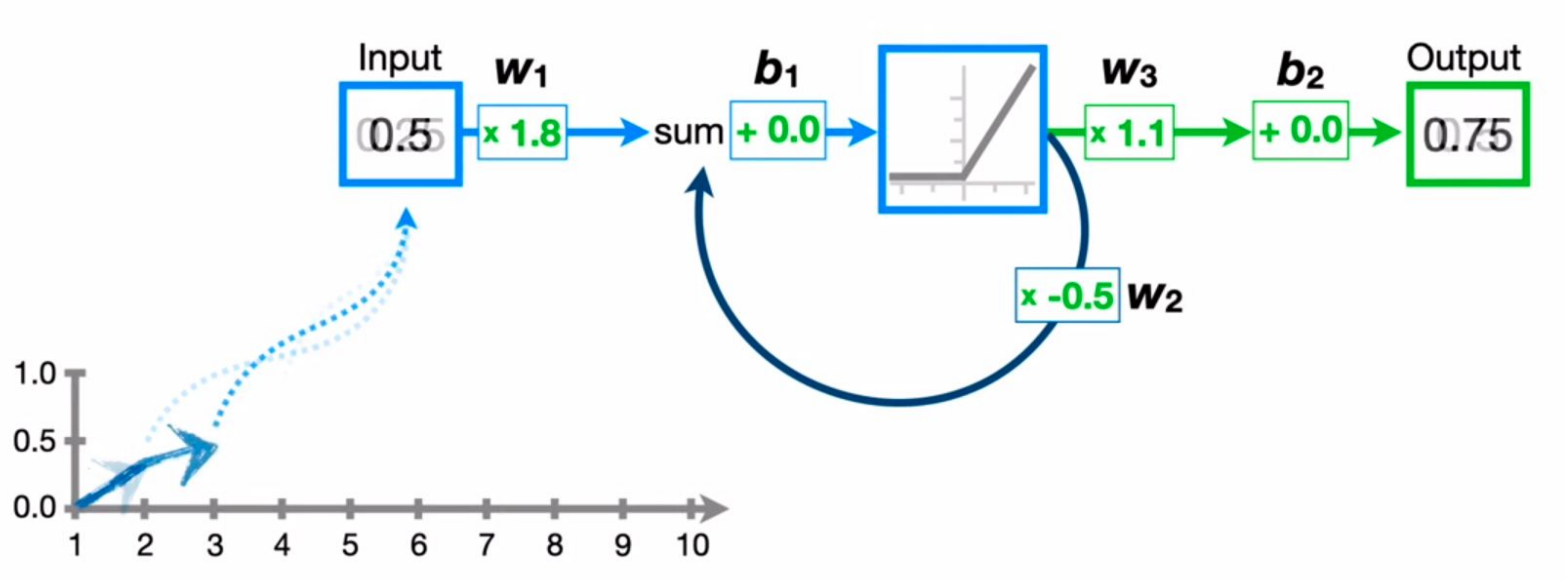
_ Feedback loops



Memory

Recurrent Nets

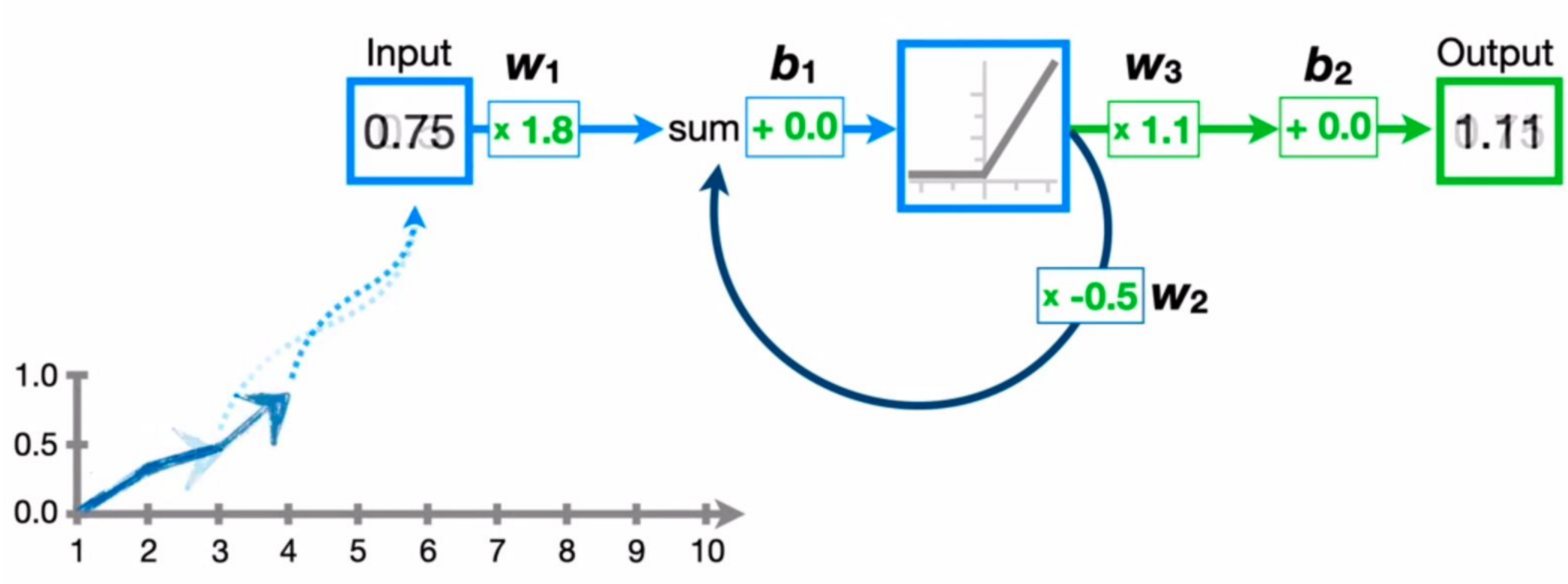
- Feedback loops



Memory

Recurrent Nets

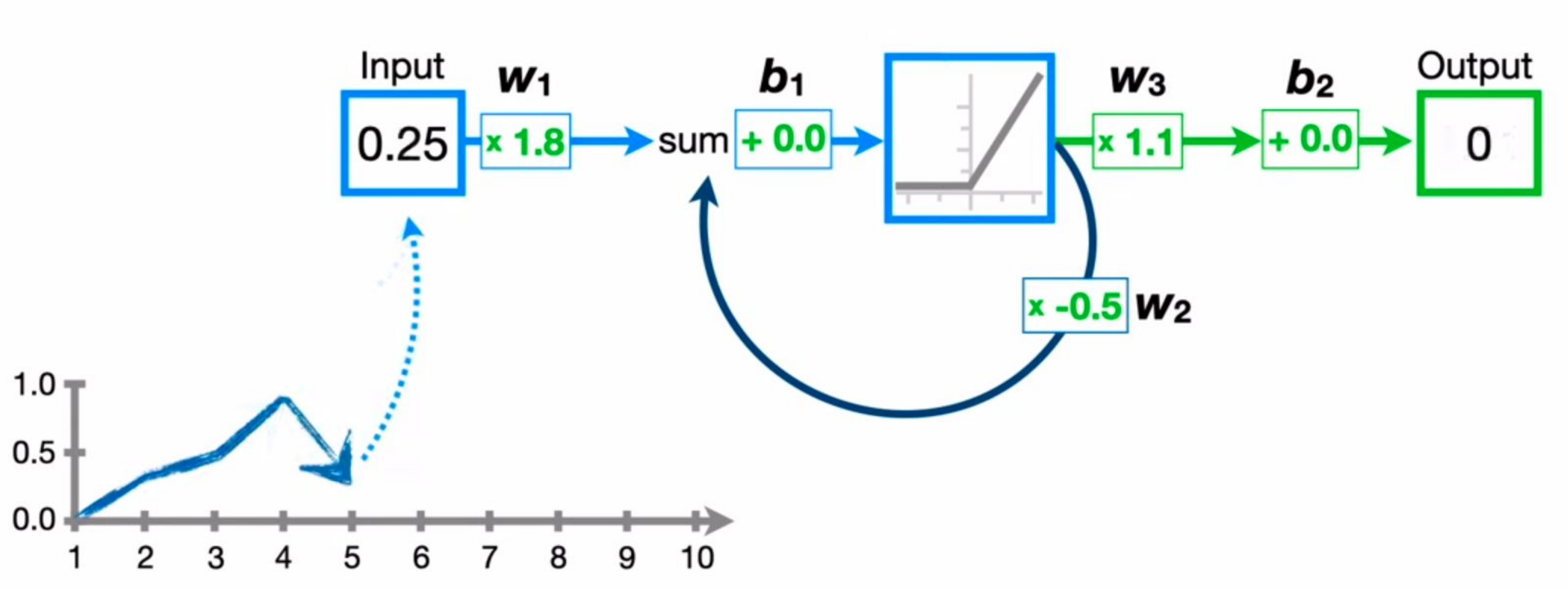
- Feedback loops



Memory

Recurrent Nets

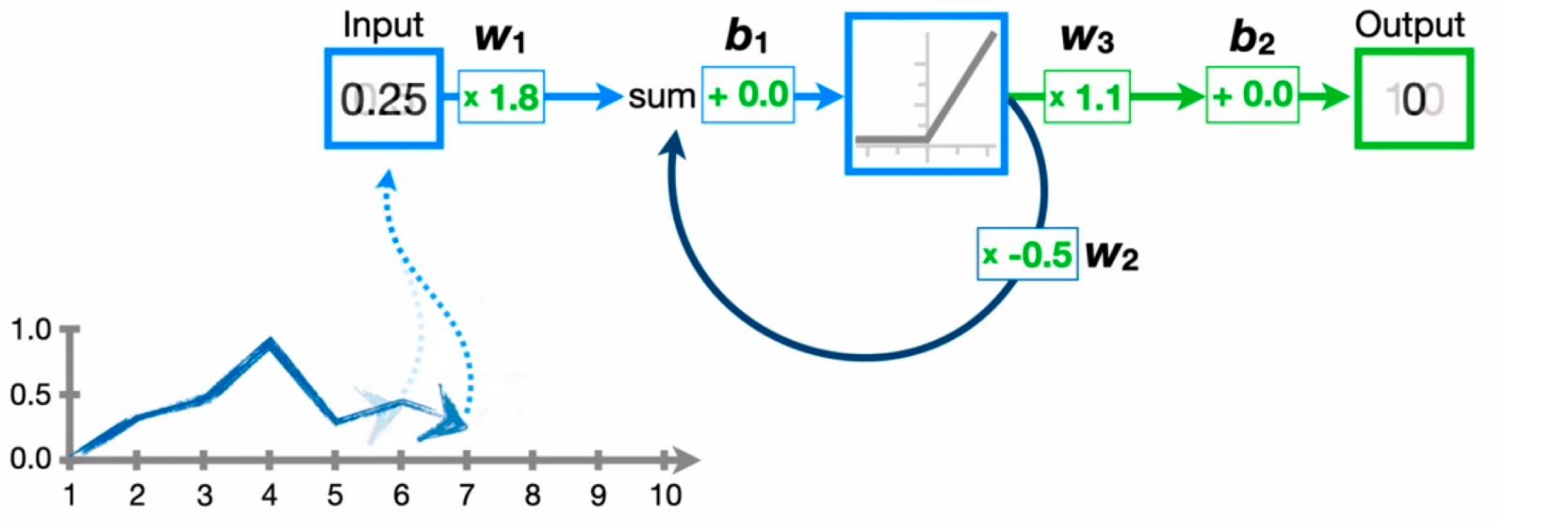
_ Feedback loops



Memory

Recurrent Nets

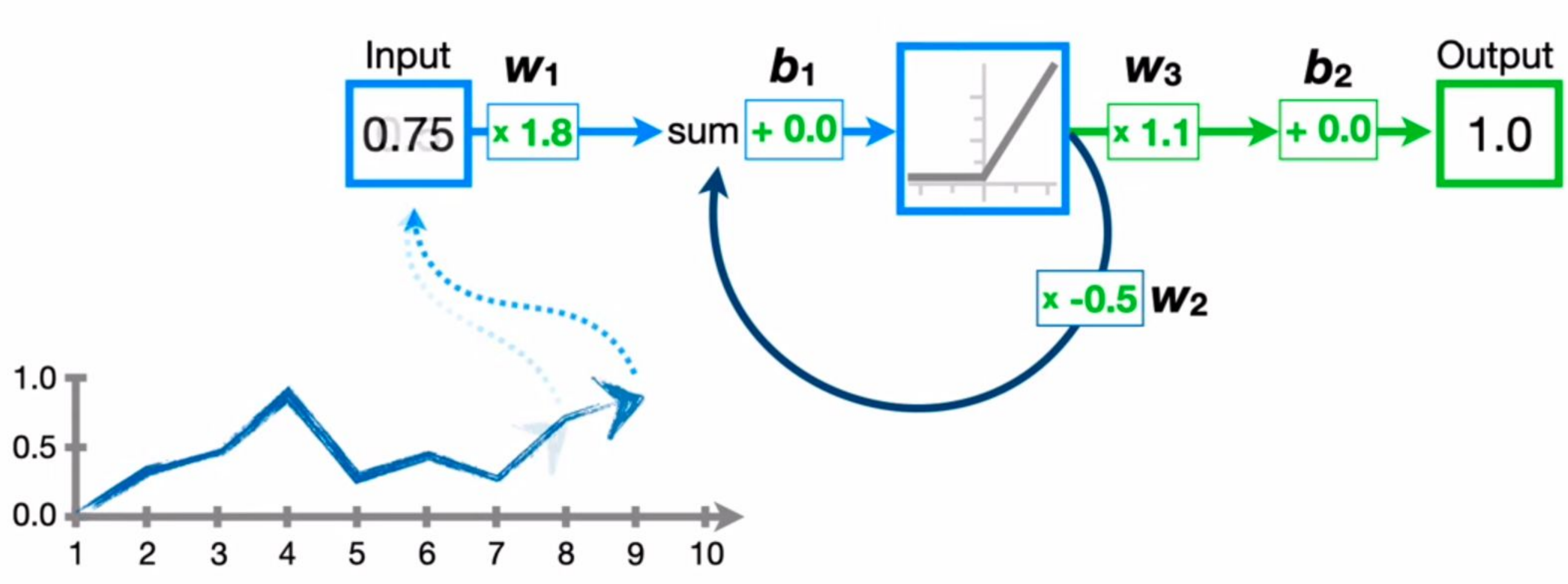
_ Feedback loops



Memory

Recurrent Nets

- Feedback loops

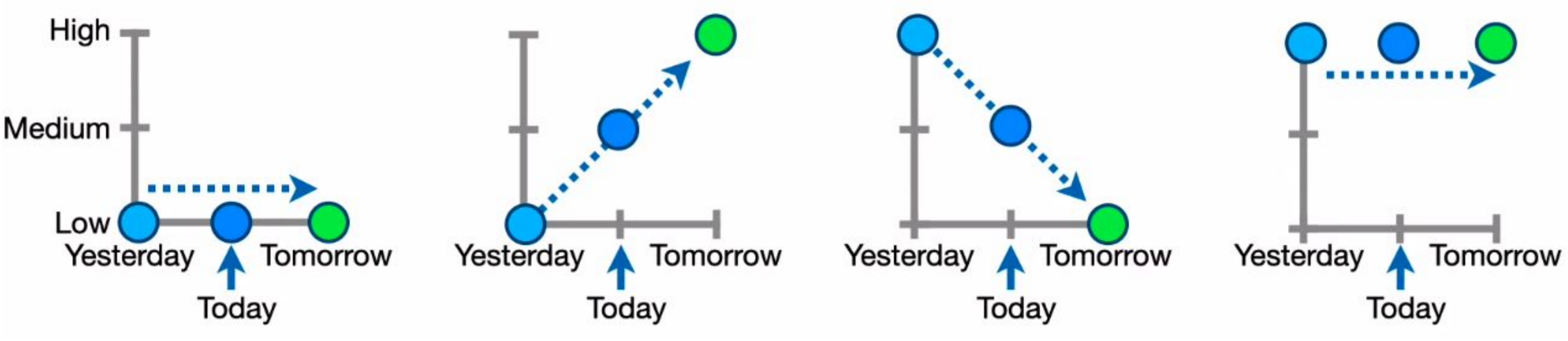


Memory

Recurrent Nets

_ How to use recurrent neural networks?

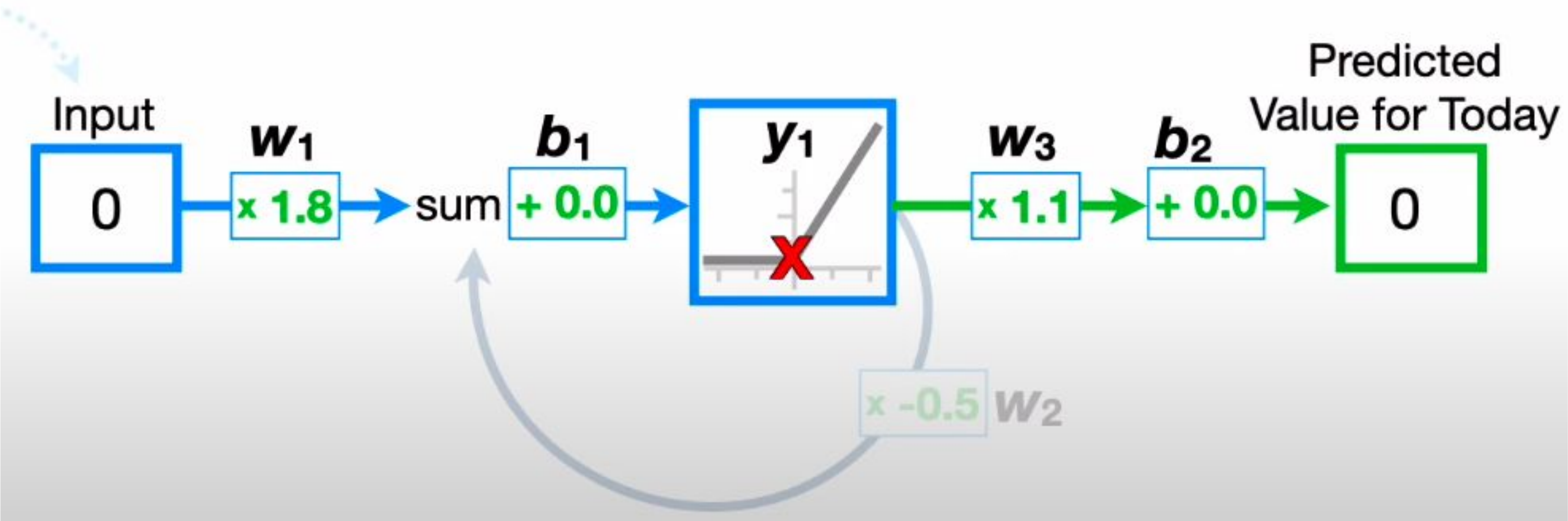
- Yesterday and today prices to predict tomorrow
  - Low = 0, Medium = 0.5, High = 1



Memory

Recurrent Nets

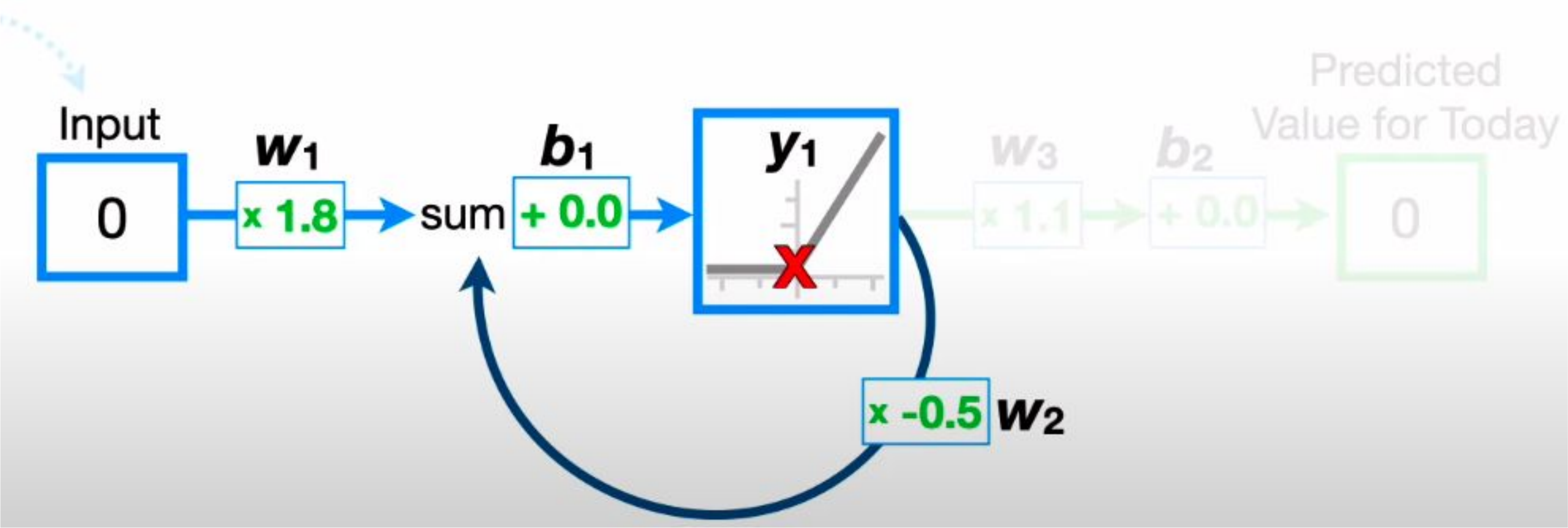
_ Feedback loops



Memory

Recurrent Nets

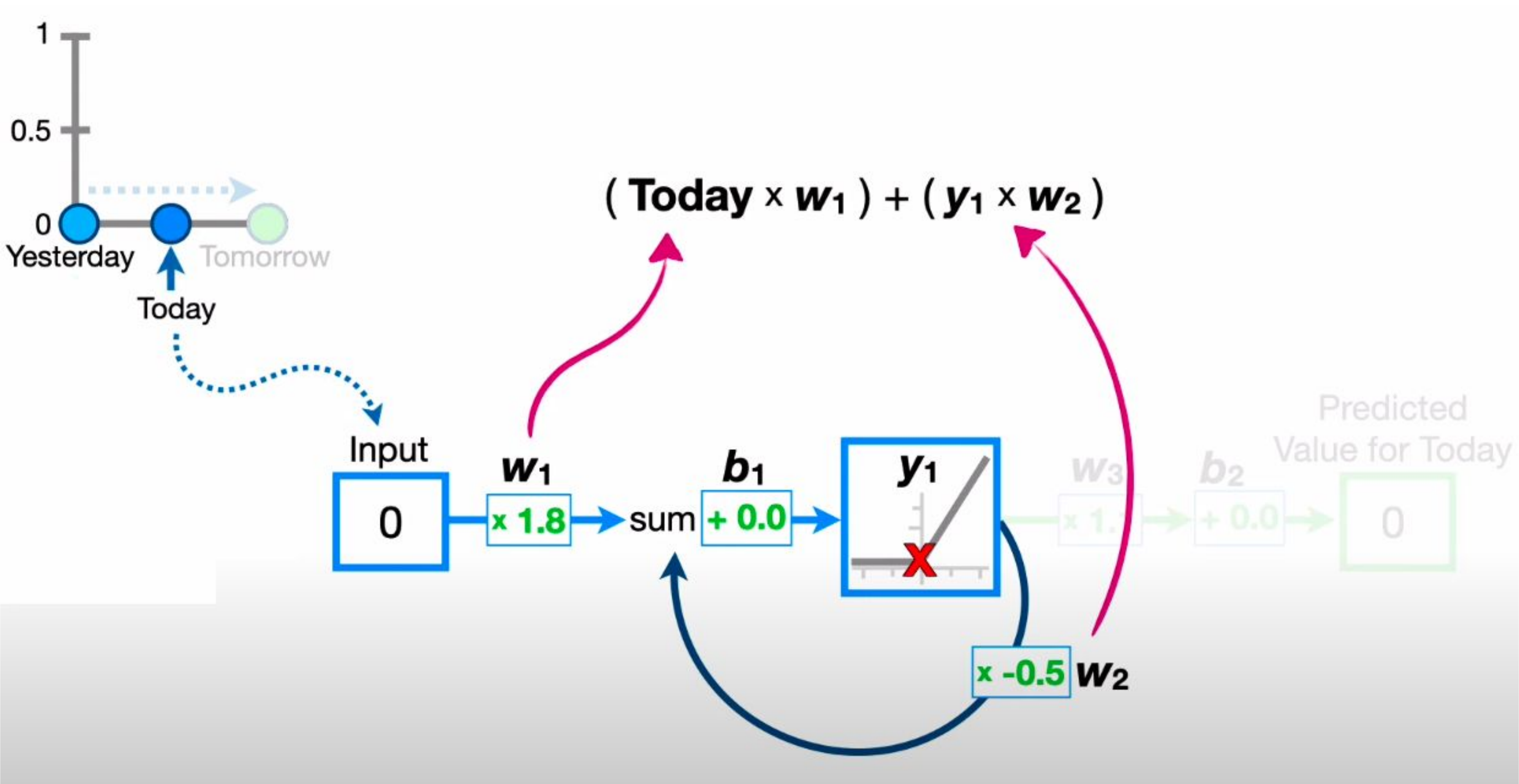
- Feedback loops



Memory

Recurrent Nets

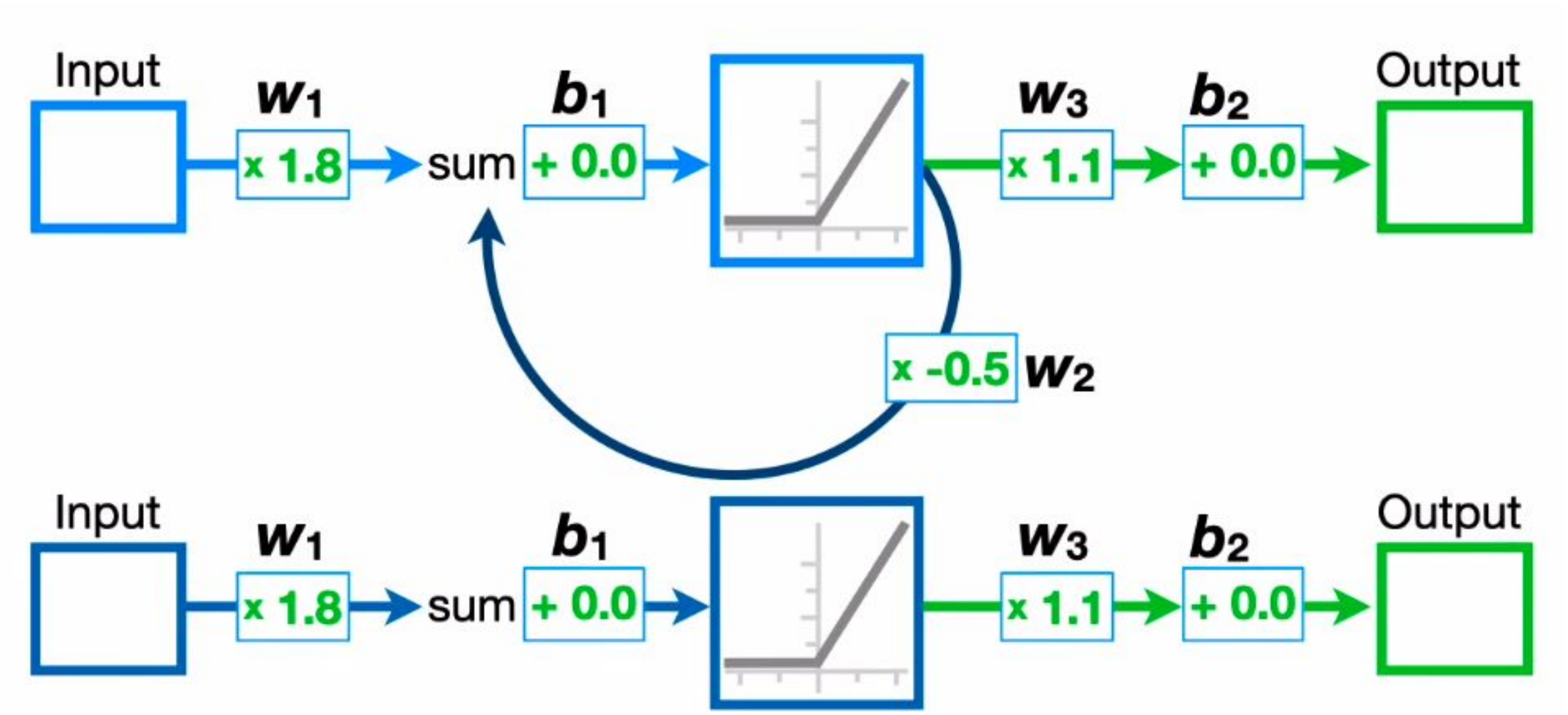
- Feedback loops



Memory

Recurrent Nets

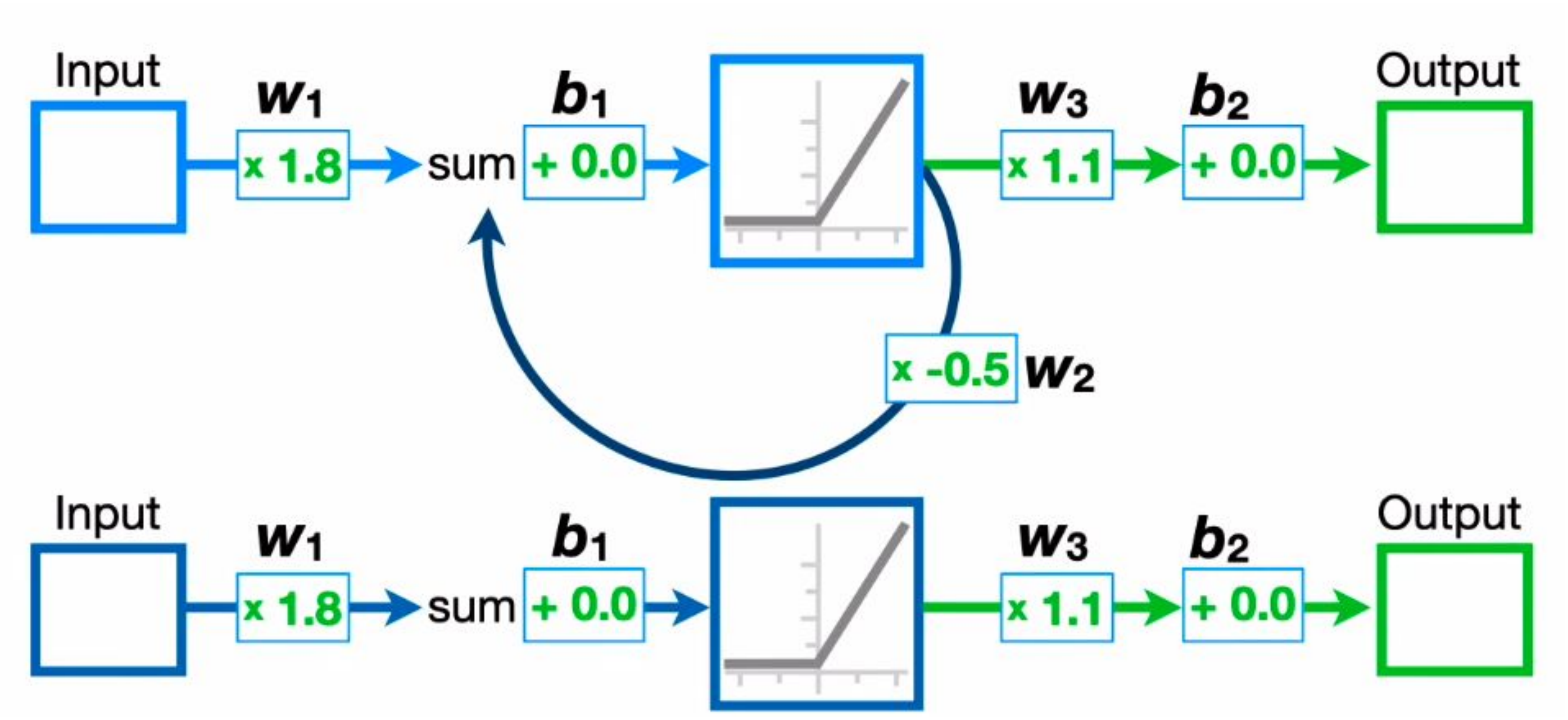
_ Unrolling

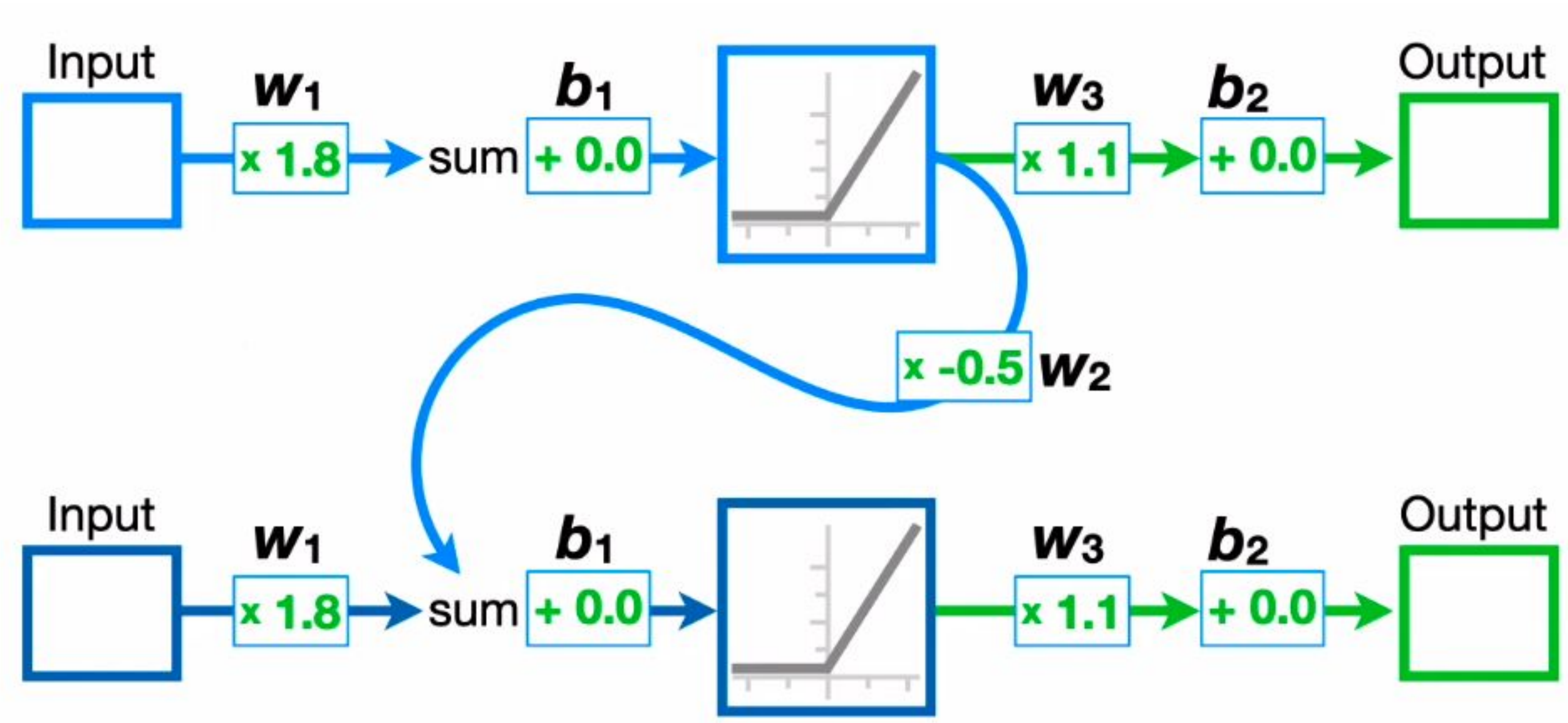


Memory

Recurrent Nets

Unrolling

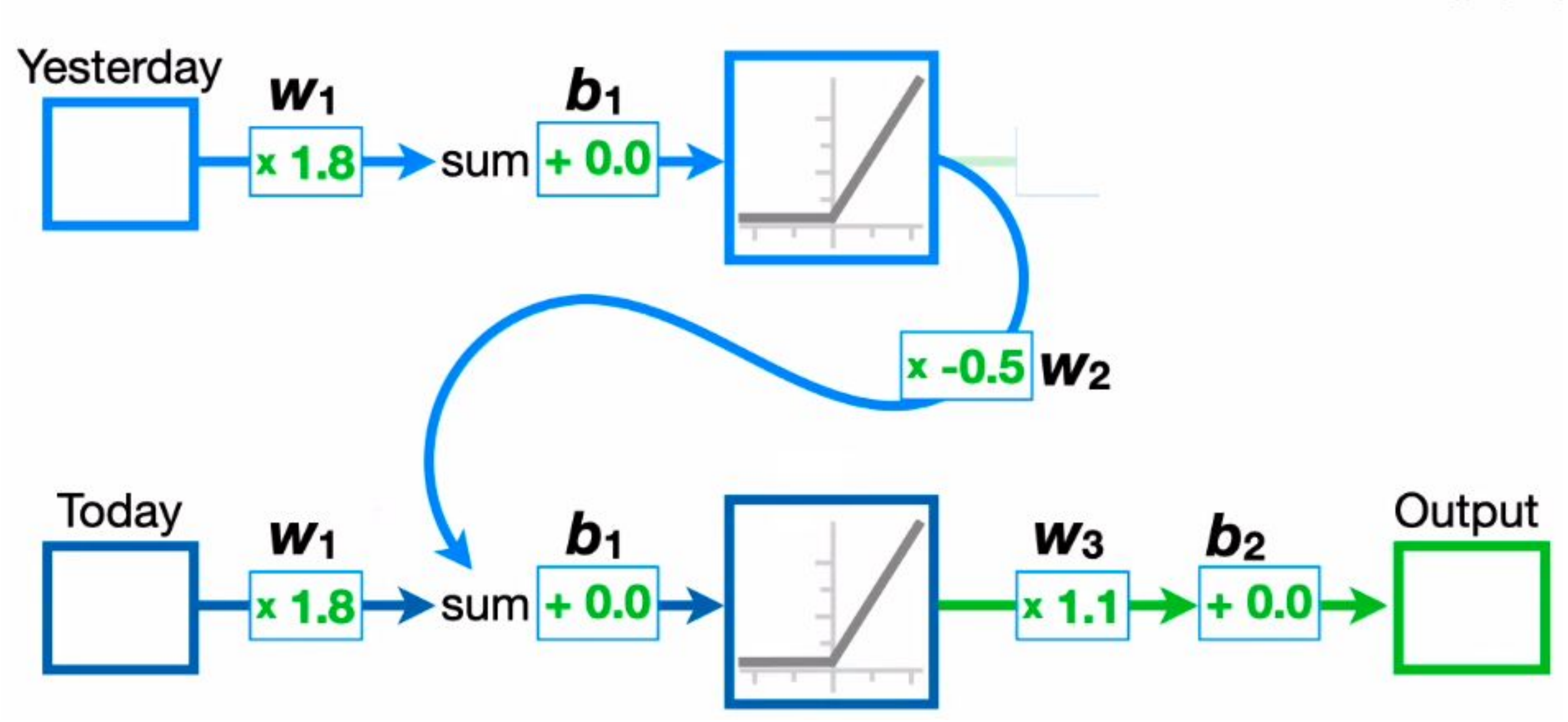




Memory

Recurrent Nets

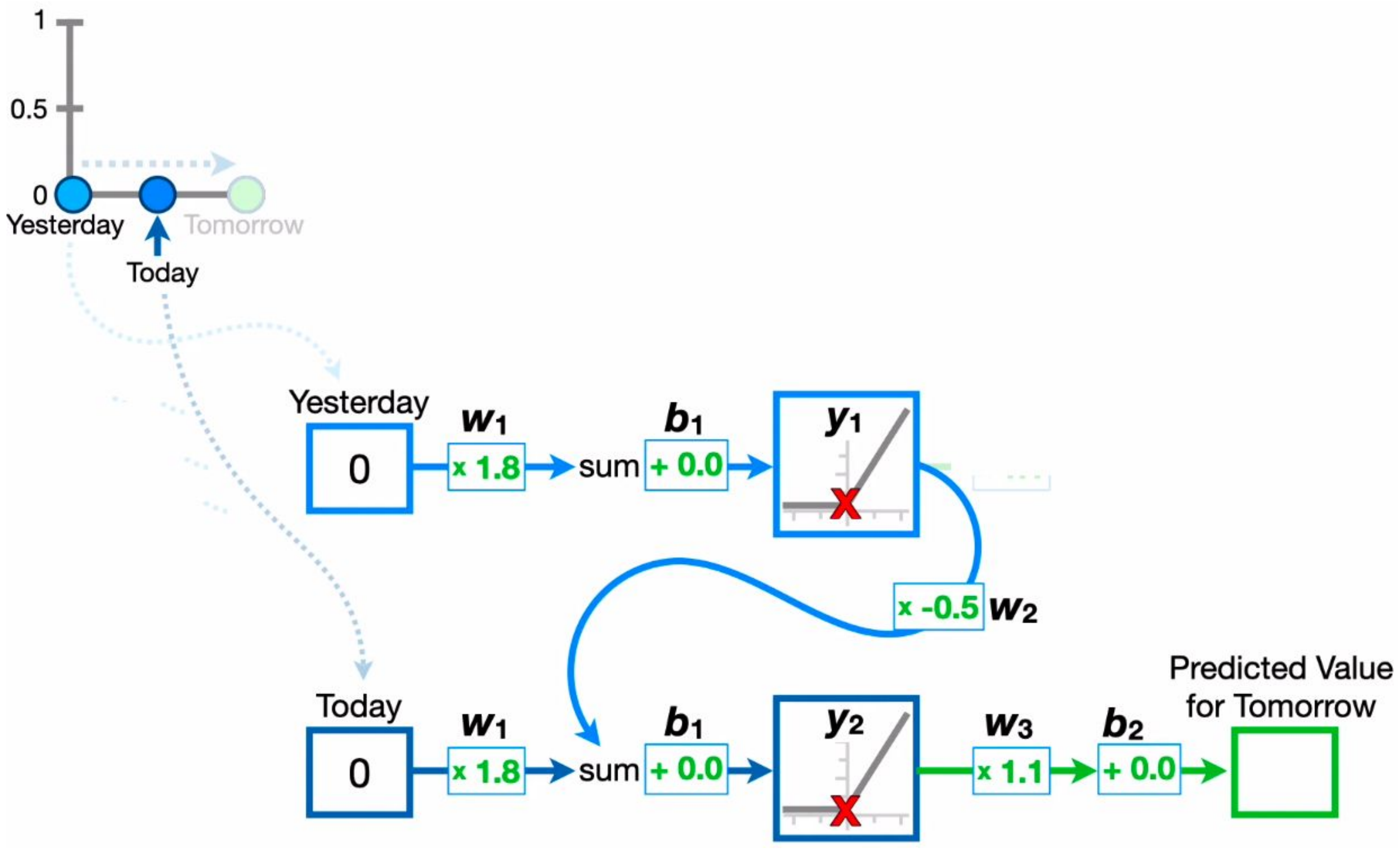
_ Unrolling



Memory

Recurrent Nets

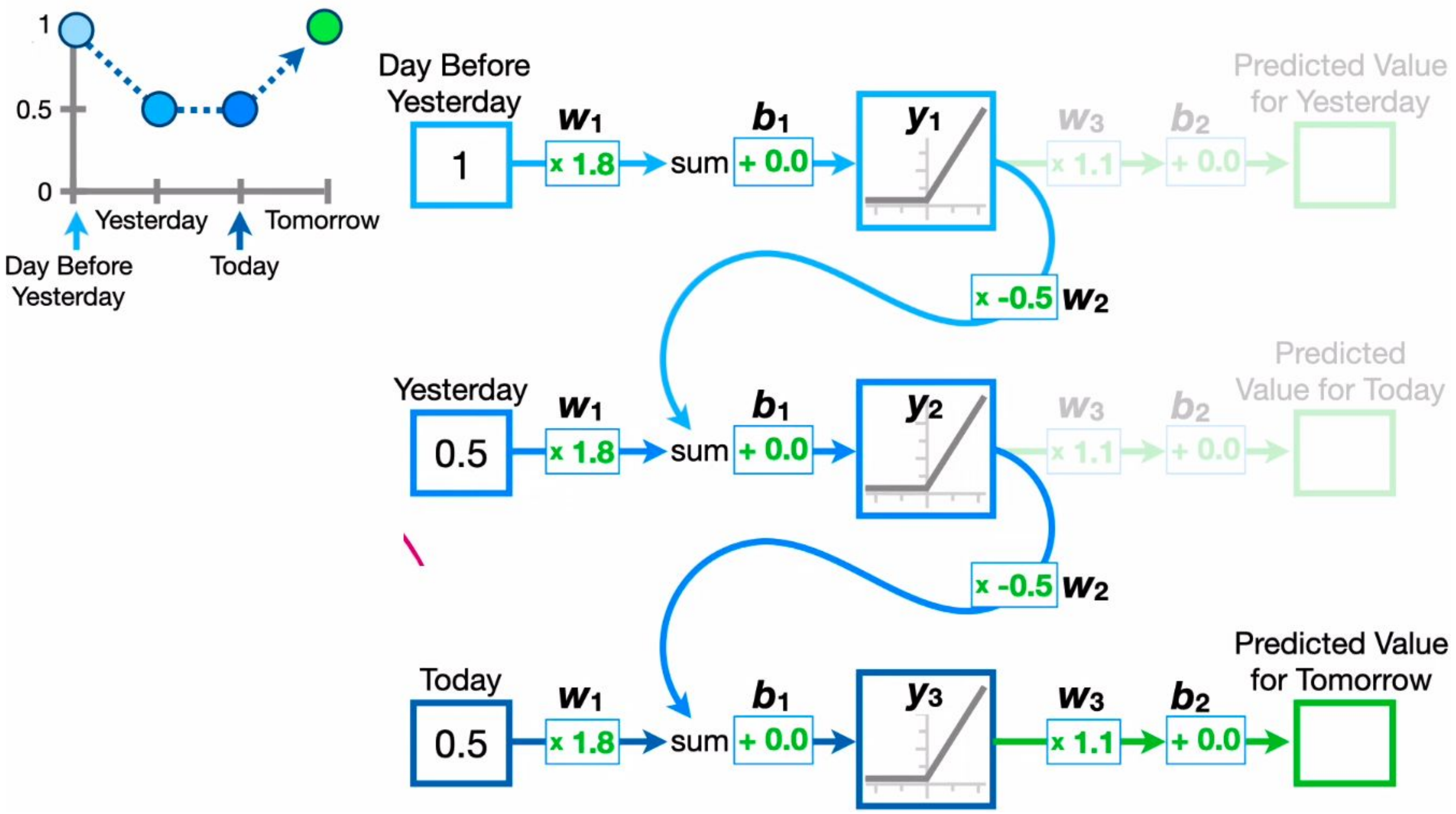
_ Unrolling



Memory

Recurrent Nets

- Unrolling many times

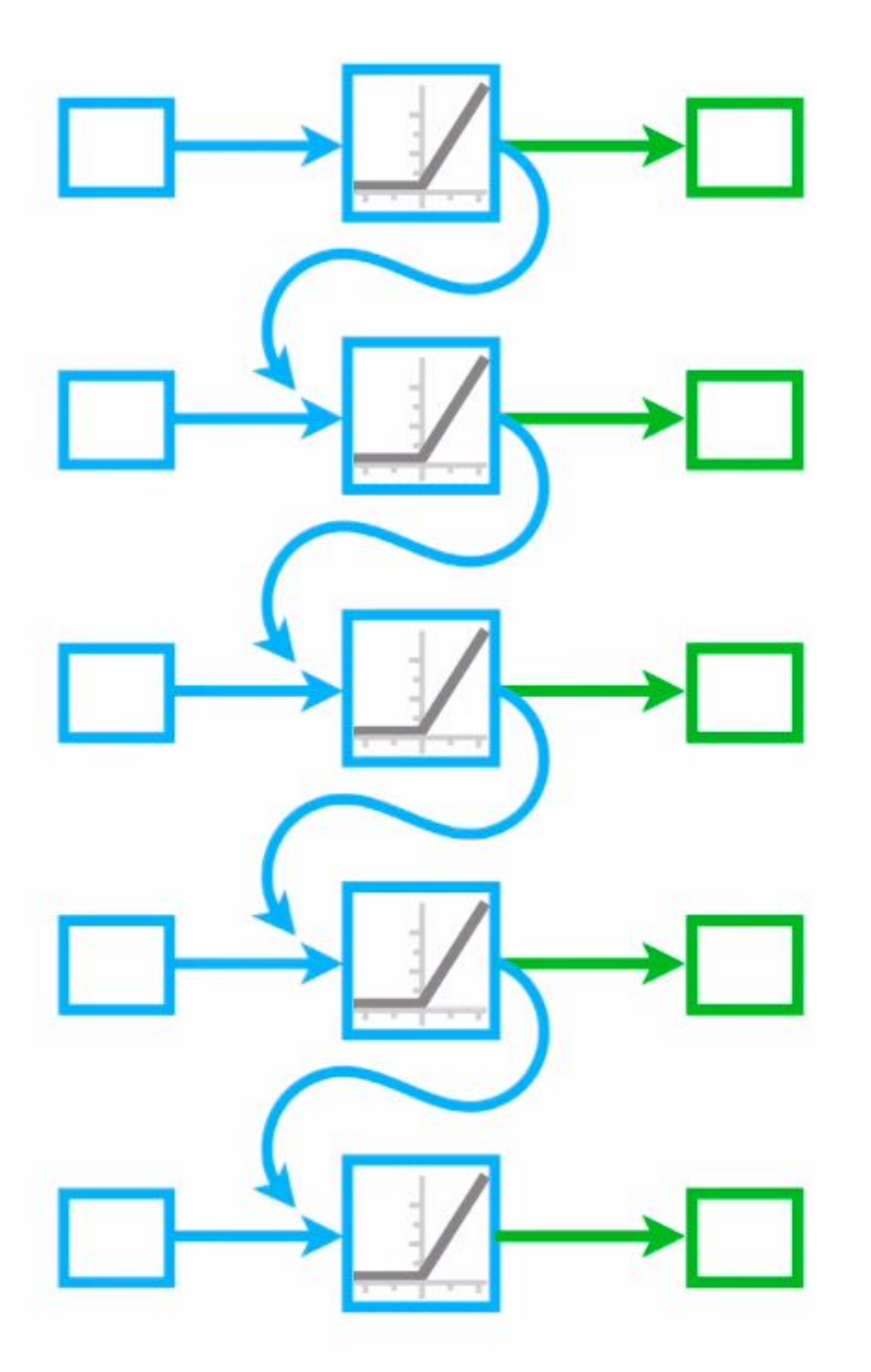


Memory

Recurrent Nets

_ Gradient Vanishing/Explosion

- Weights greater or smaller than 1



LSTM

long-Short Term Memory

_ We have two memories:

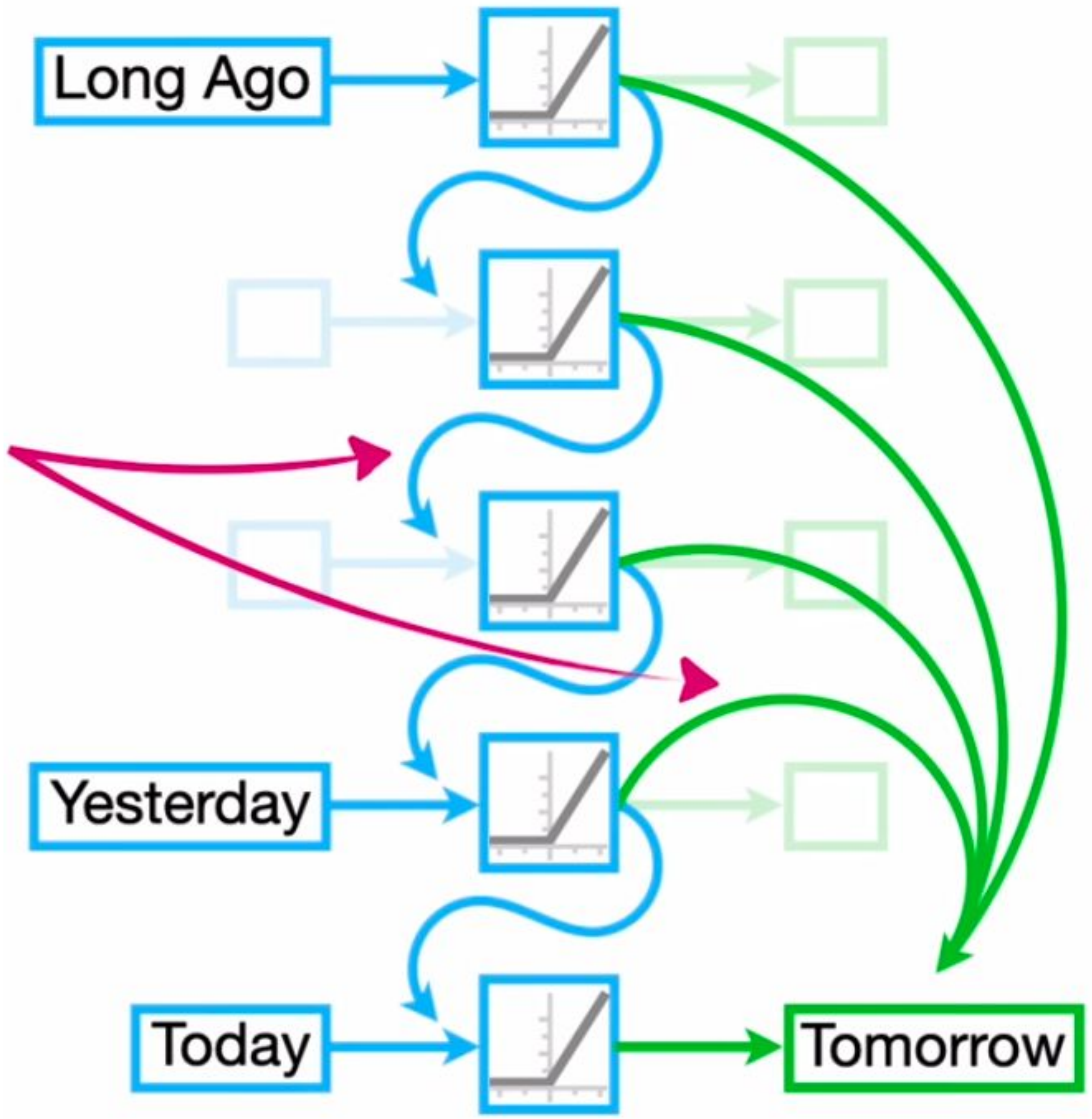
- Short memory:
    - Recent events are important
- Long memory:
    - Events that occurred longer ago must be memorized at some extent
- … but, typical RNNs have only one memory.

LSTM

long-Short Term Memory

We have two memories:

- How much of short memory should be incorporated?
- ... and how much of long memory should be discarded?
    - Weights are learned to control these memories.
        - Backpropagation!

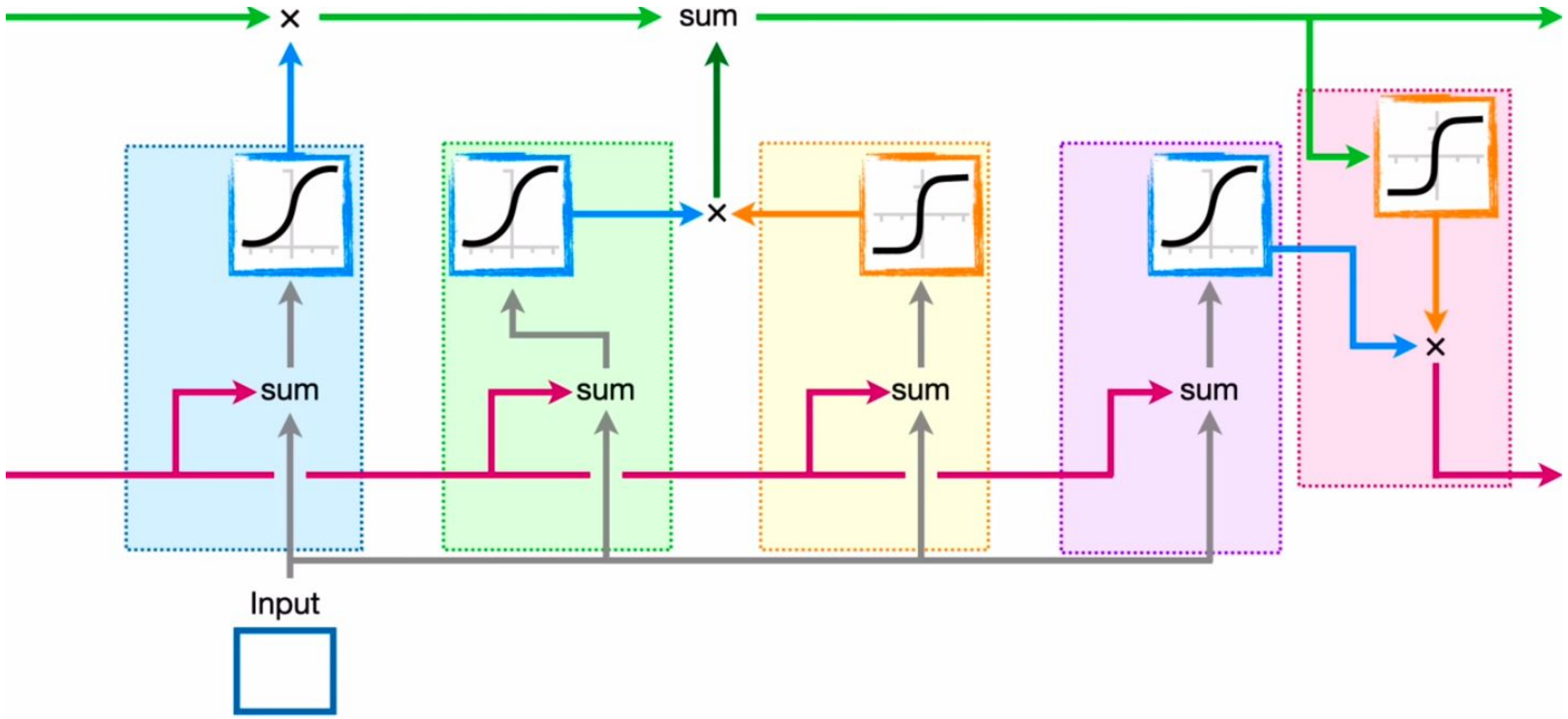


LSTM

long-Short Term Memory

_ We have two memories:

- The LSTM cell is complex!
    - Each cell has three components

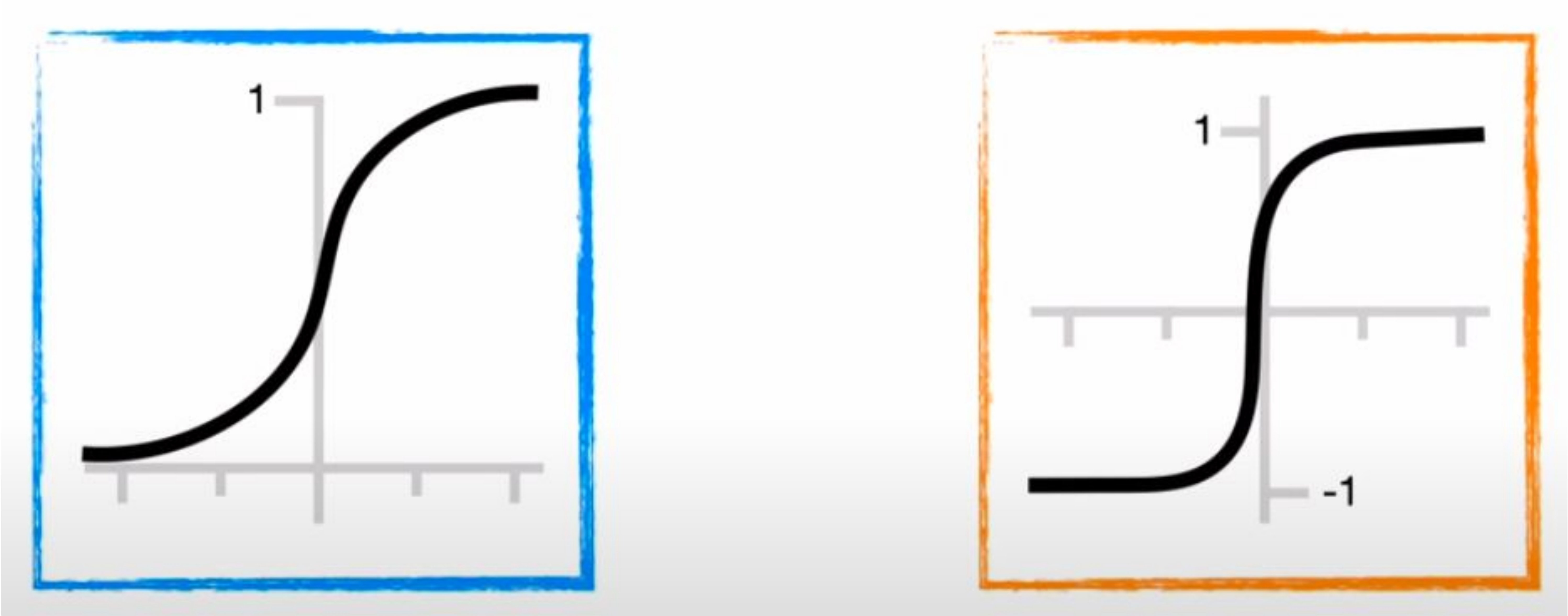


LSTM

long-Short Term Memory

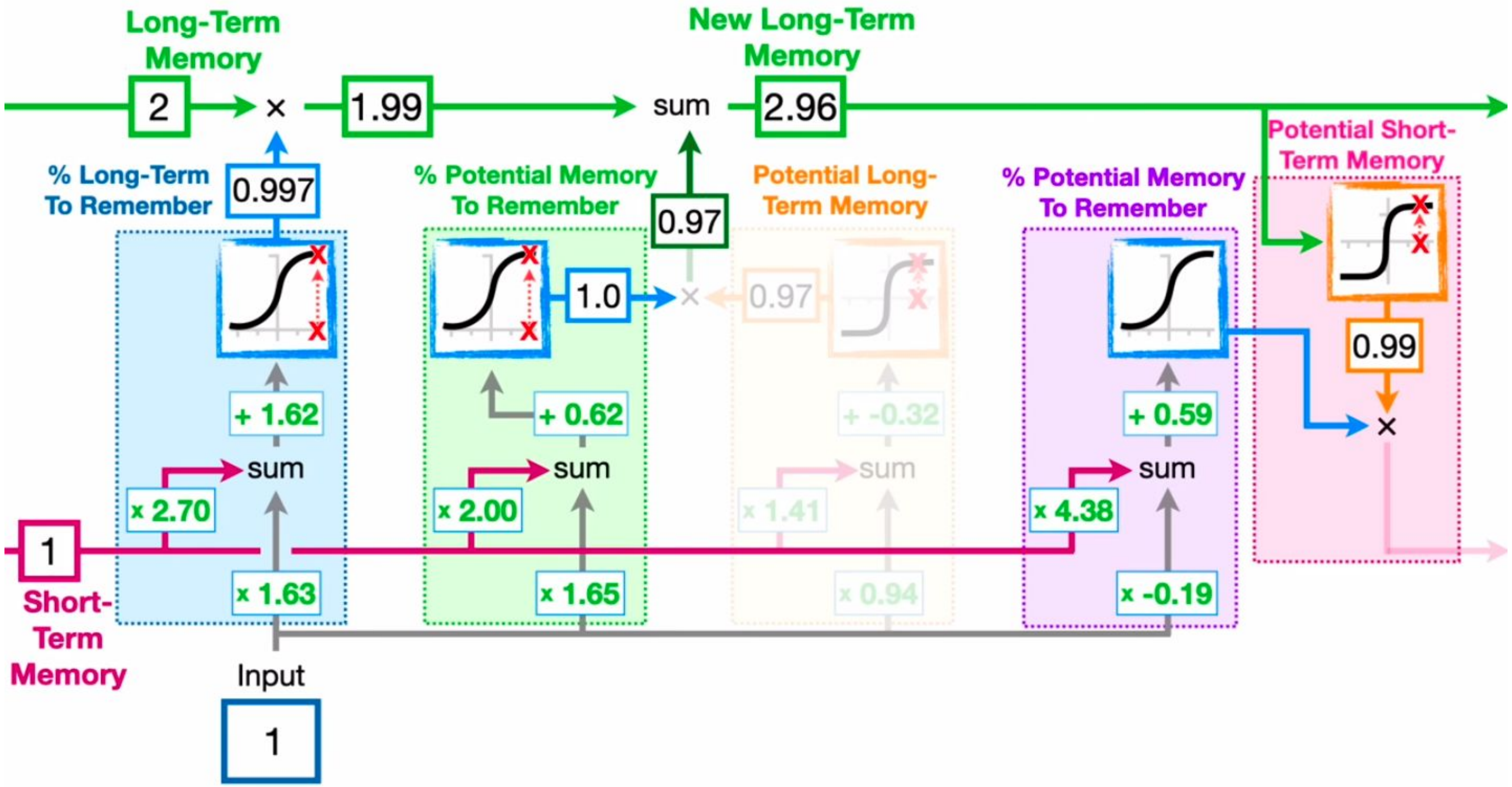
_ Two activation functions

- Sigmoid
  - Output between 0 and 1
- Tahn
  - Output between -1 and 1



LSTM

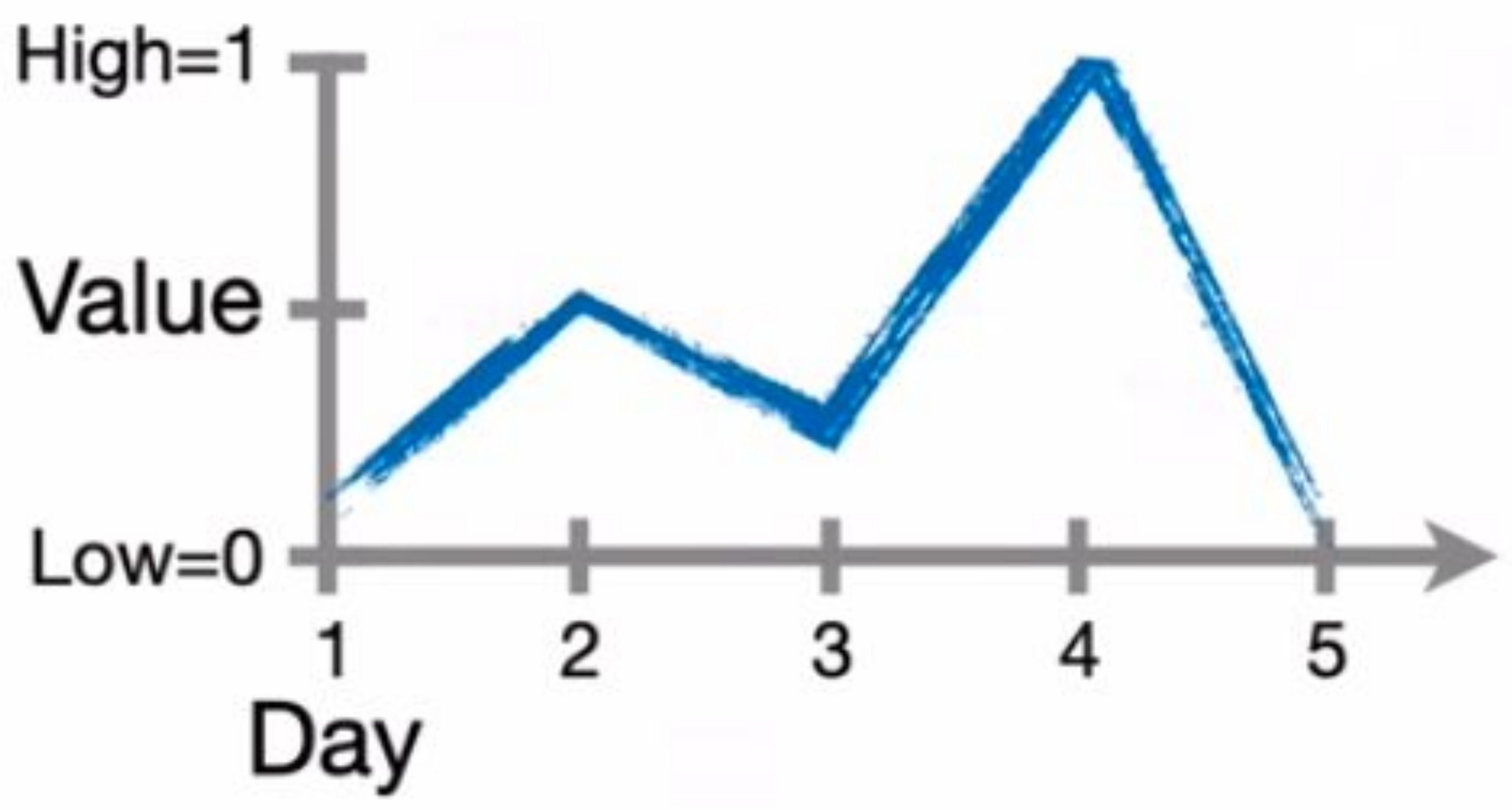
long-Short Term Memory

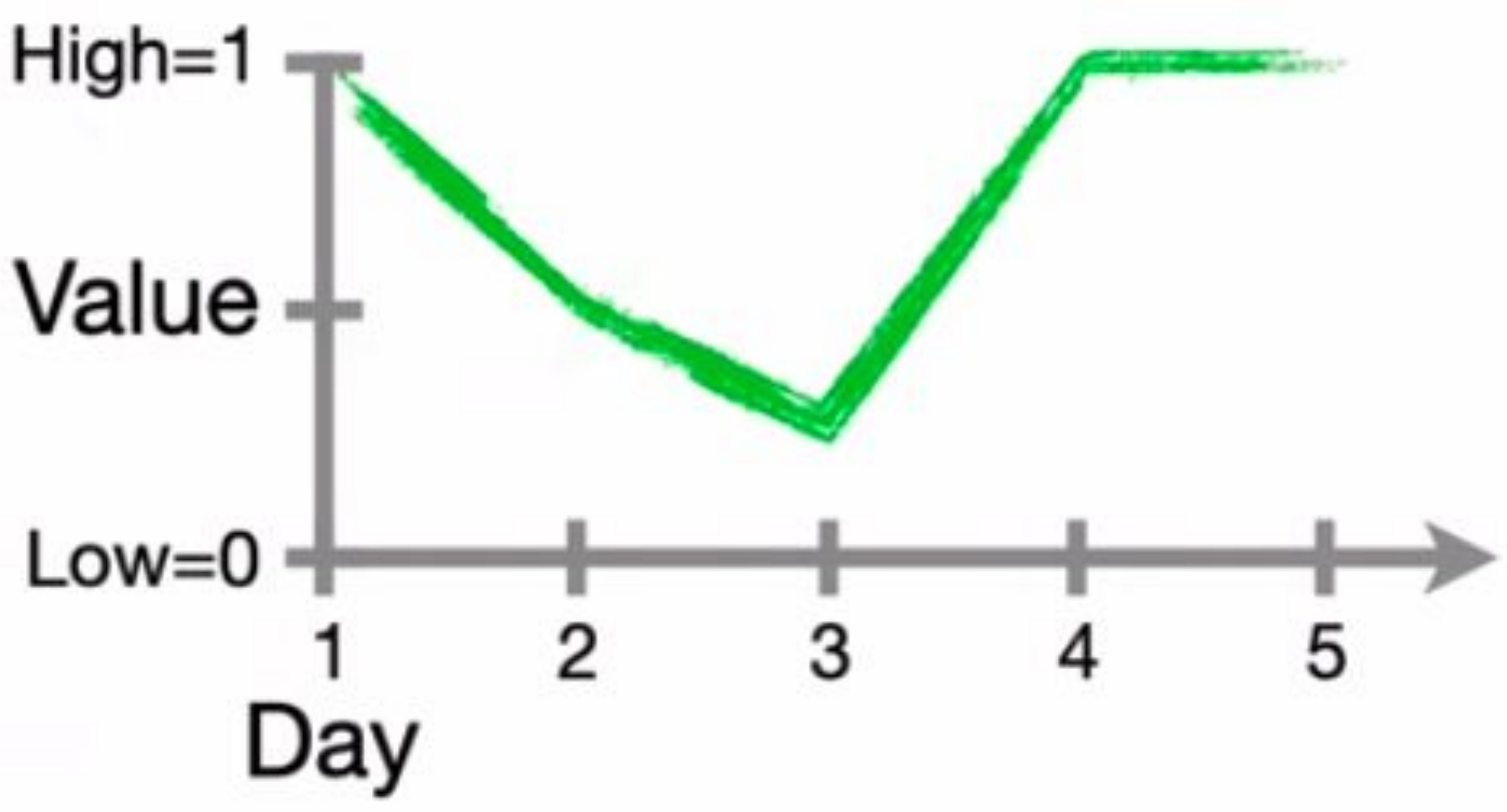


LSTM

long-Short Term Memory

_ Example

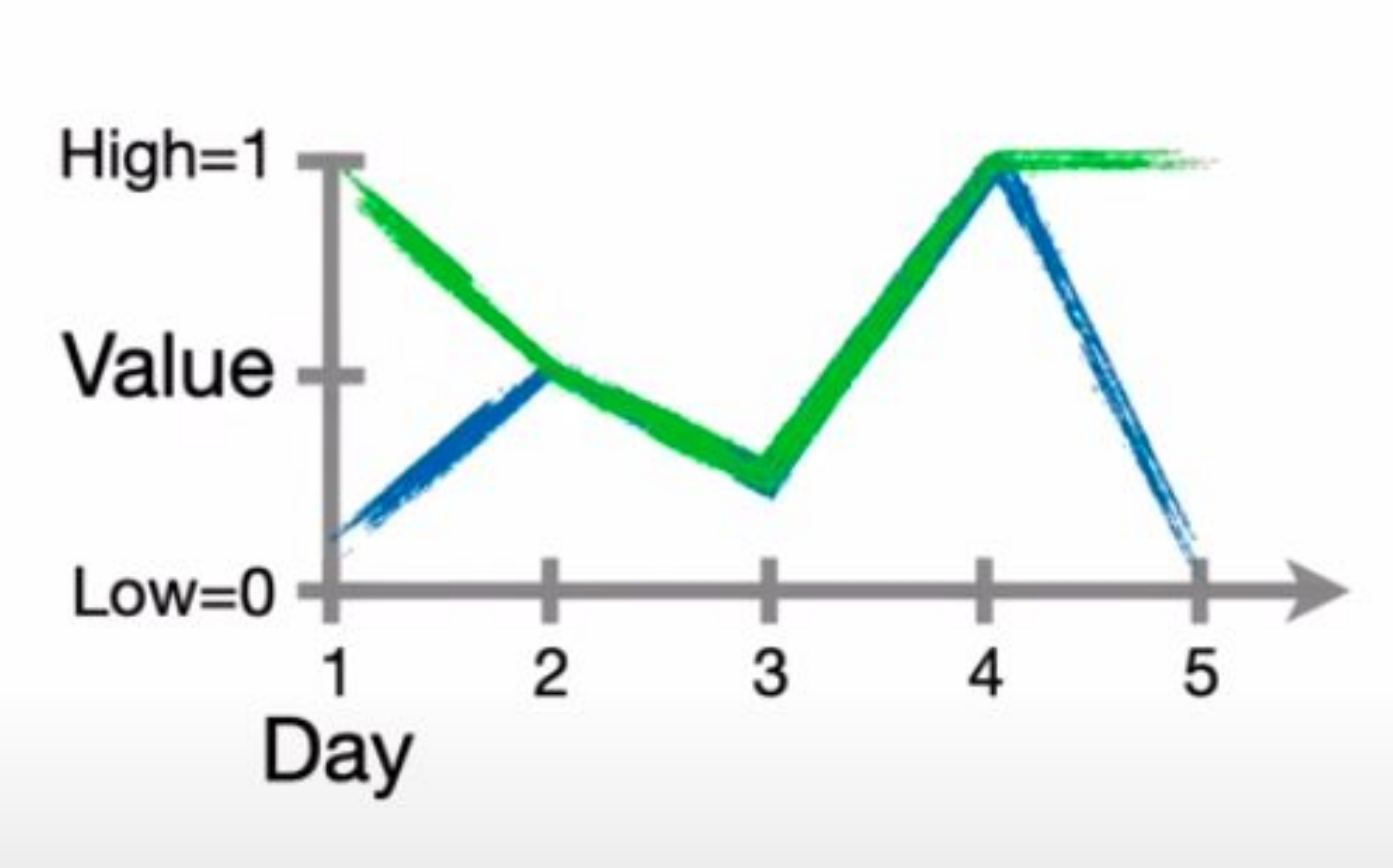




LSTM

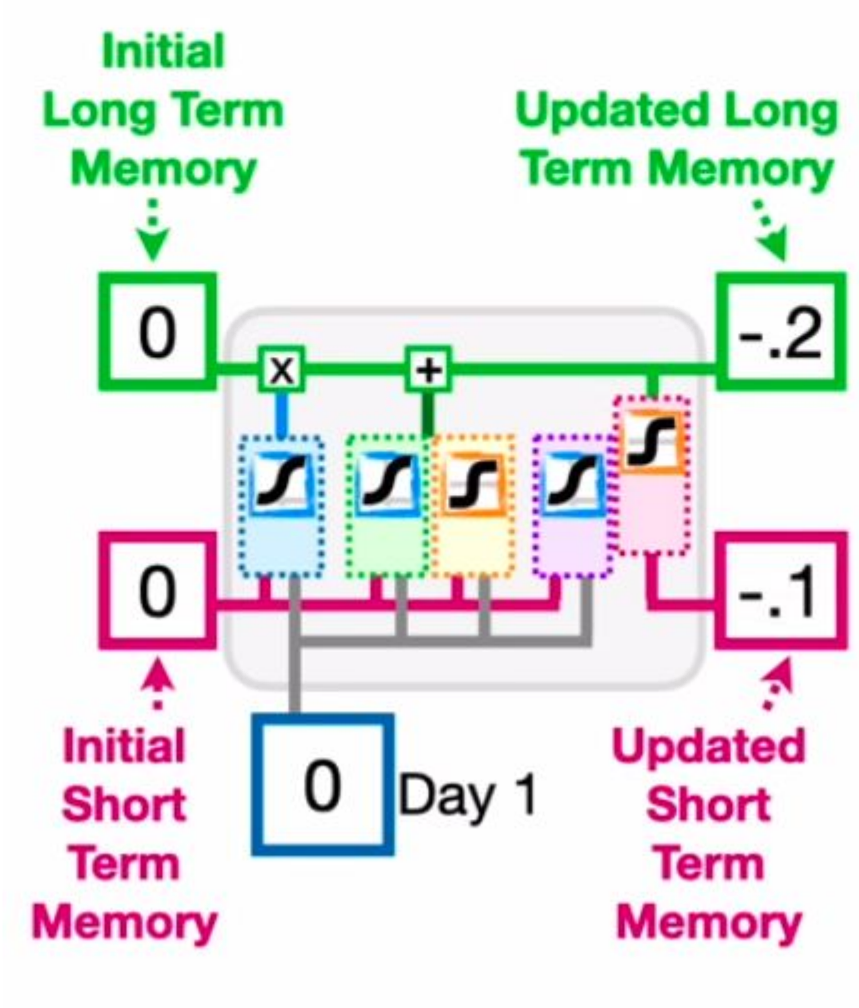
long-Short Term Memory

_ Example



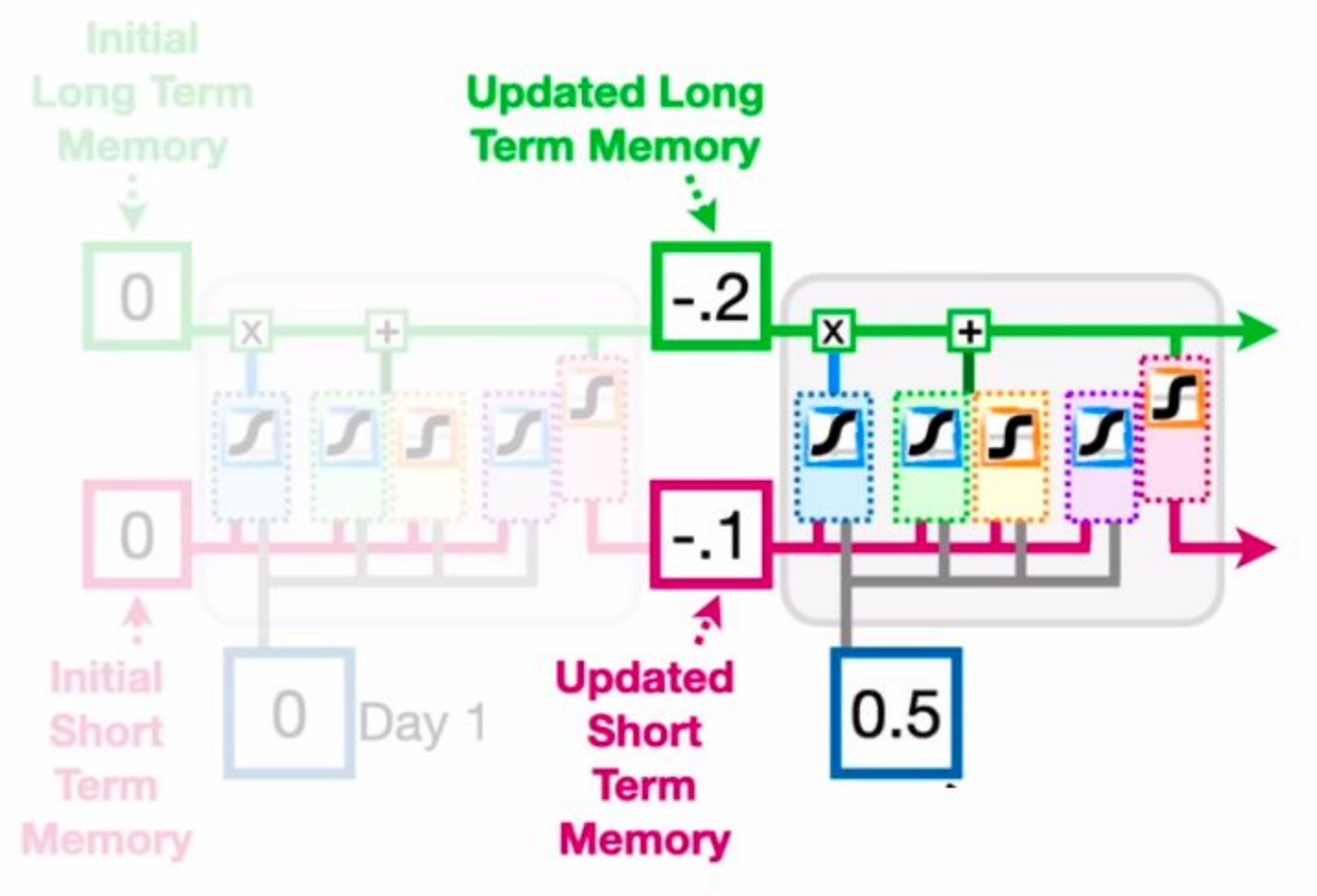
LSTM

long-Short Term Memory



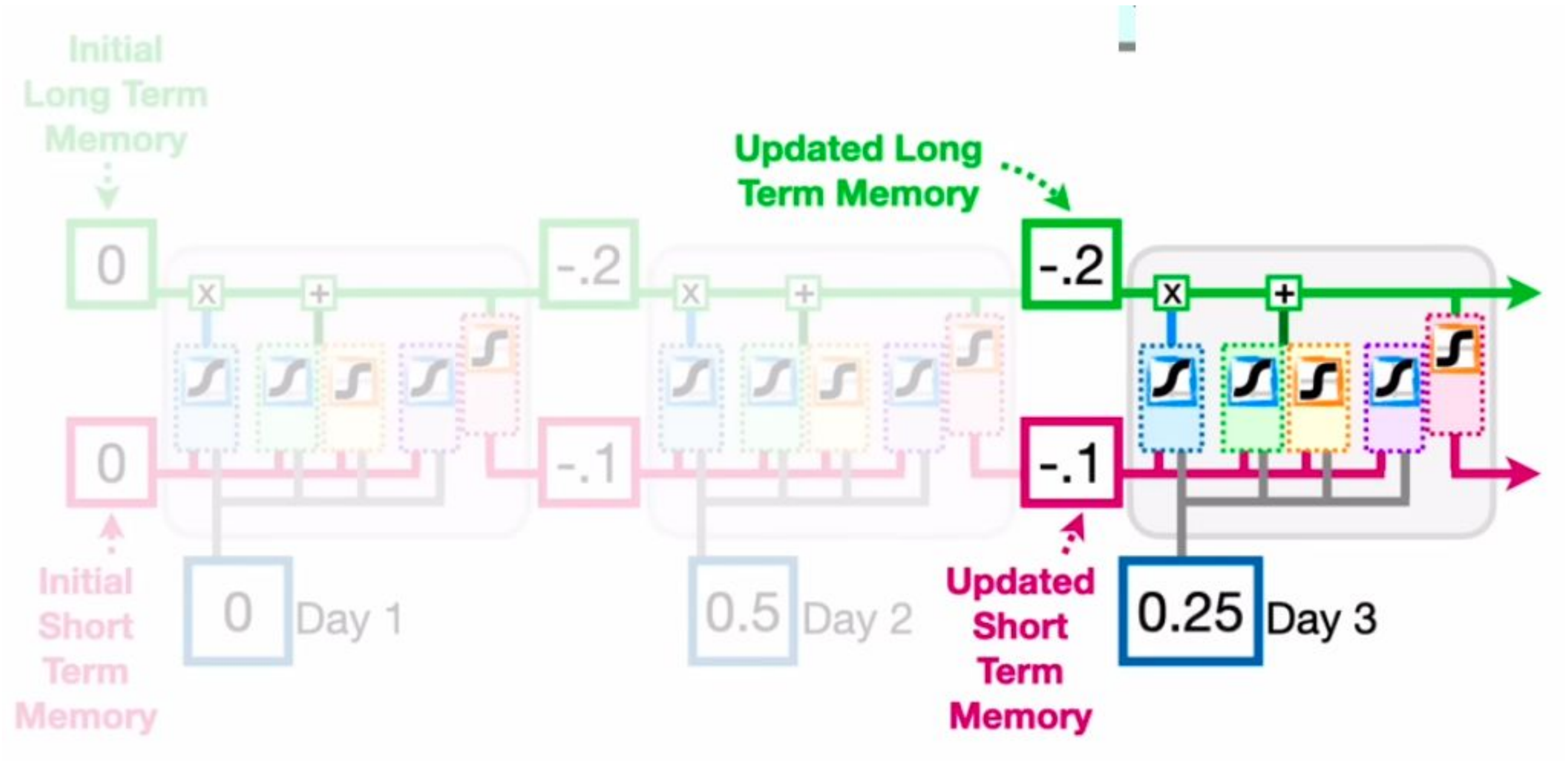
LSTM

long-Short Term Memory



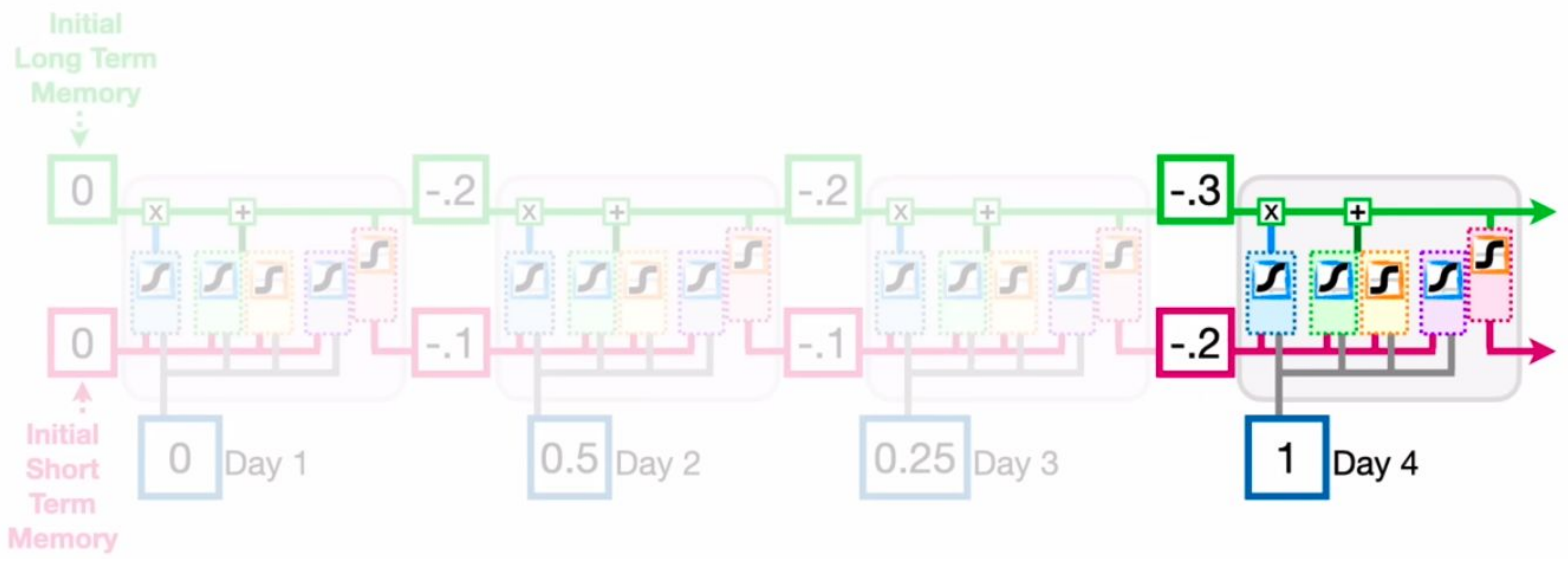
LSTM

long-Short Term Memory



LSTM

long-Short Term Memory



Encoder-Decoder

Seq2Seq Models

_ Sequences that need to be transformed in other sequences

- Translate one language to other
- Translate audio into text
- Sequence of amino acids to 3D structures

These problems are called sequence-to-sequence, or seq2seq

- We can solve these problems with an architecture called encoder-decoder

Encoder-Decoder

Seq2Seq Models

_ Challenges

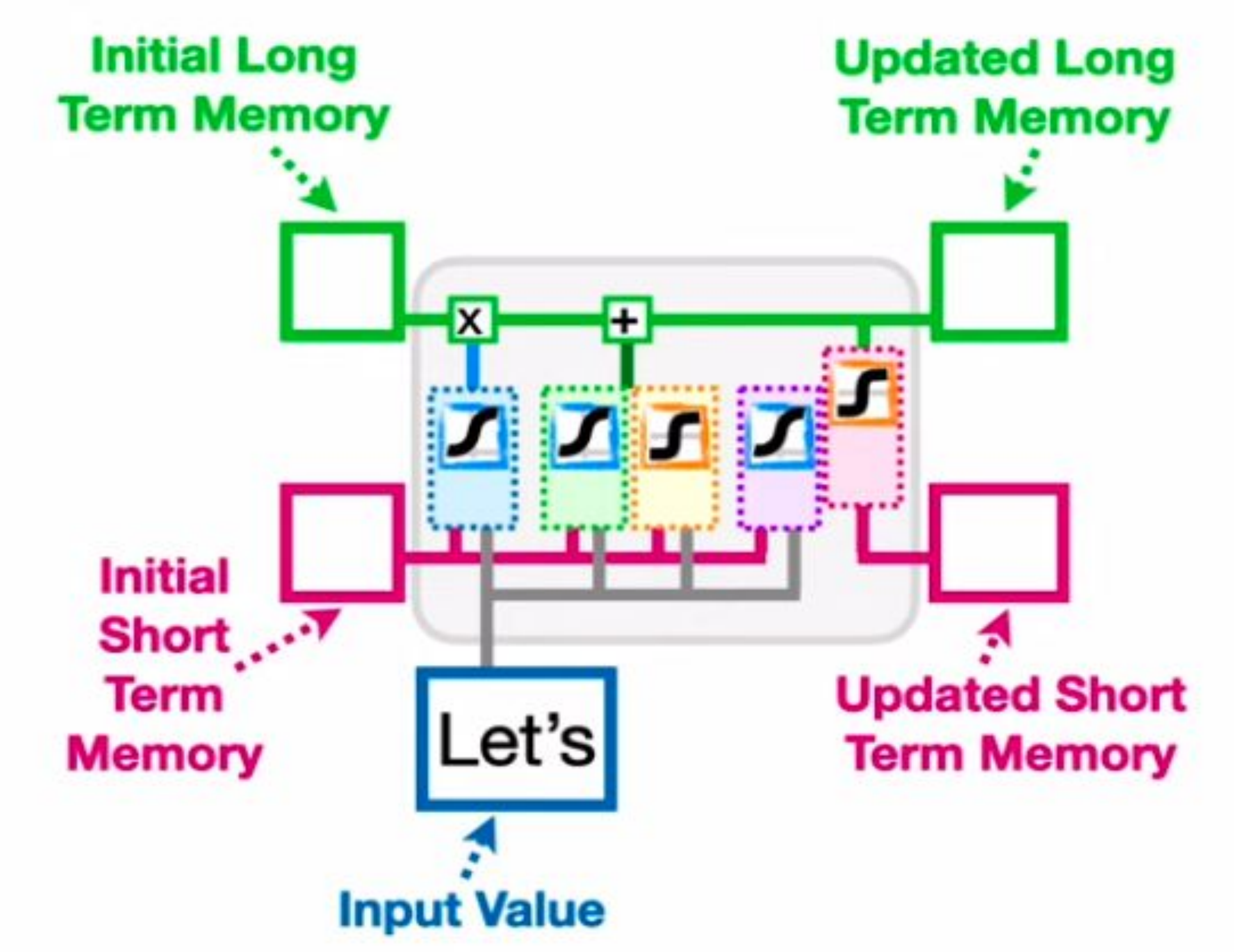
- The size of the input sequence may be shorter/larger than the size of the output sequence
    - Let's go → Vamos
        - The encoder-decoder must be able to handle variable input and variable output lengths.
        - LSTMs are good for dealing with variable input lengths.

Encoder-Decoder

Seq2Seq Models

_ LSTMs

- Remember: the input is a word embedding (one-hot encoding + fixed embedding layer)

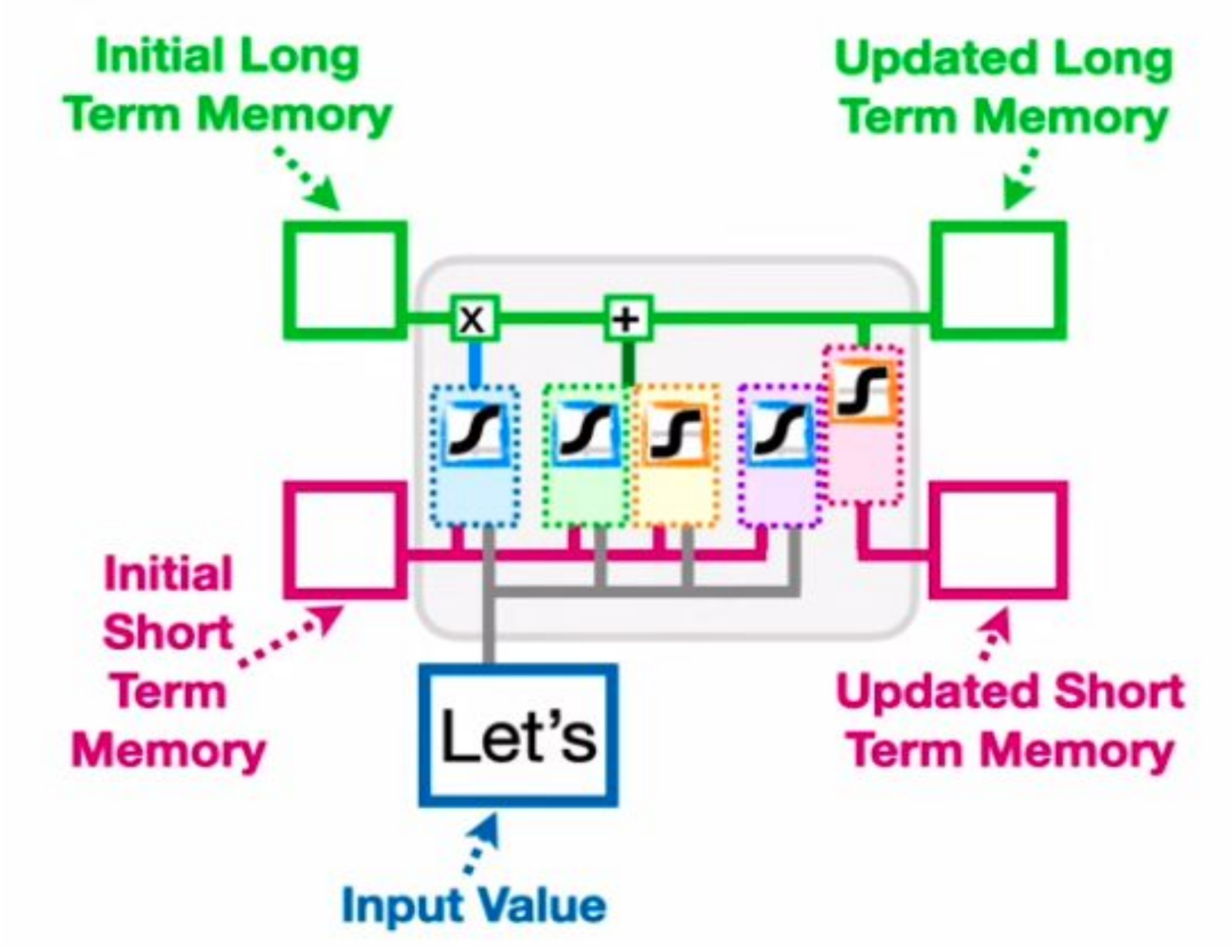


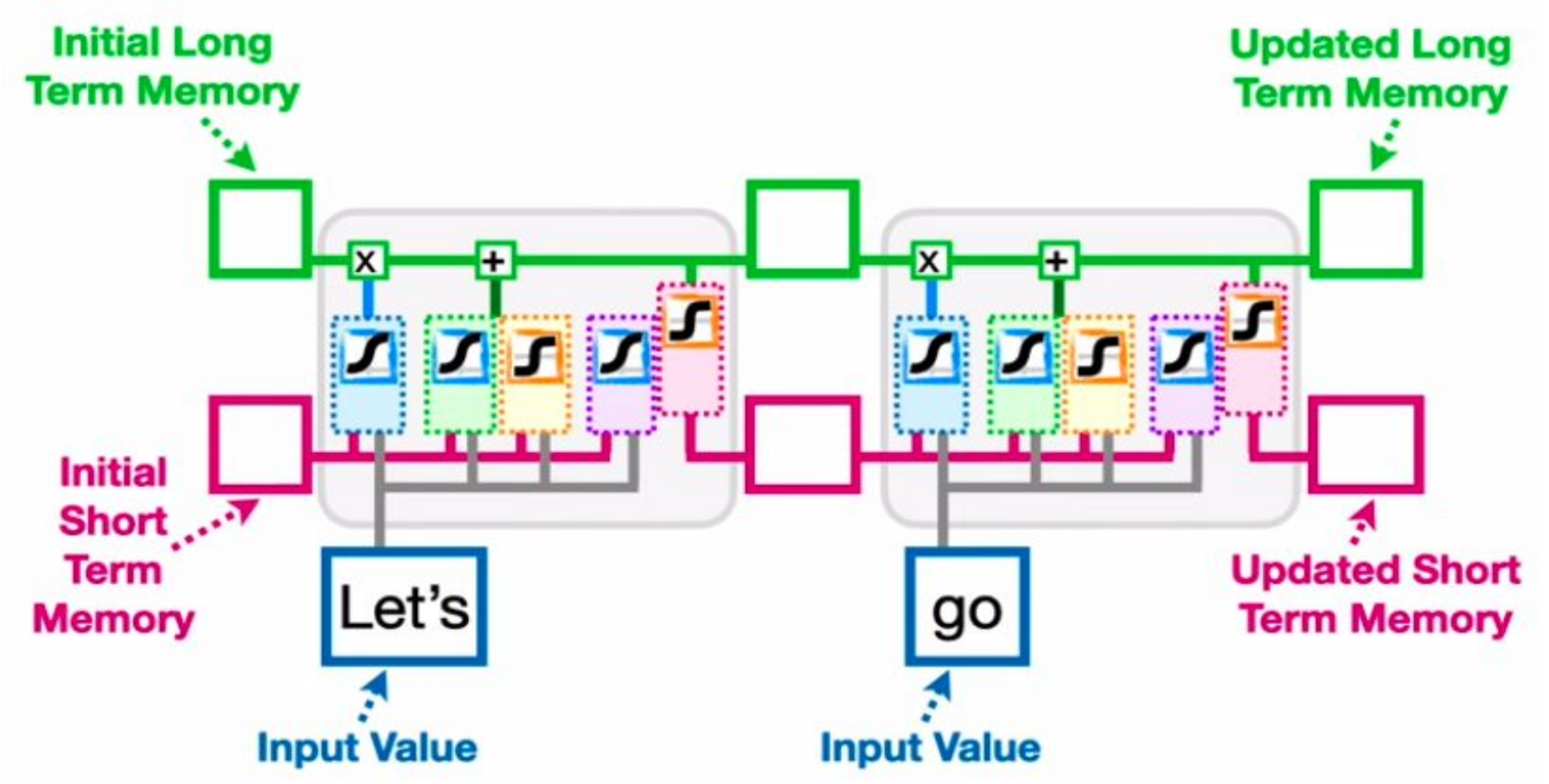
Encoder-Decoder

Seq2Seq Models

_ LSTMs

- Remember: the input is a word embedding (one-hot encoding + fixed embedding layer)

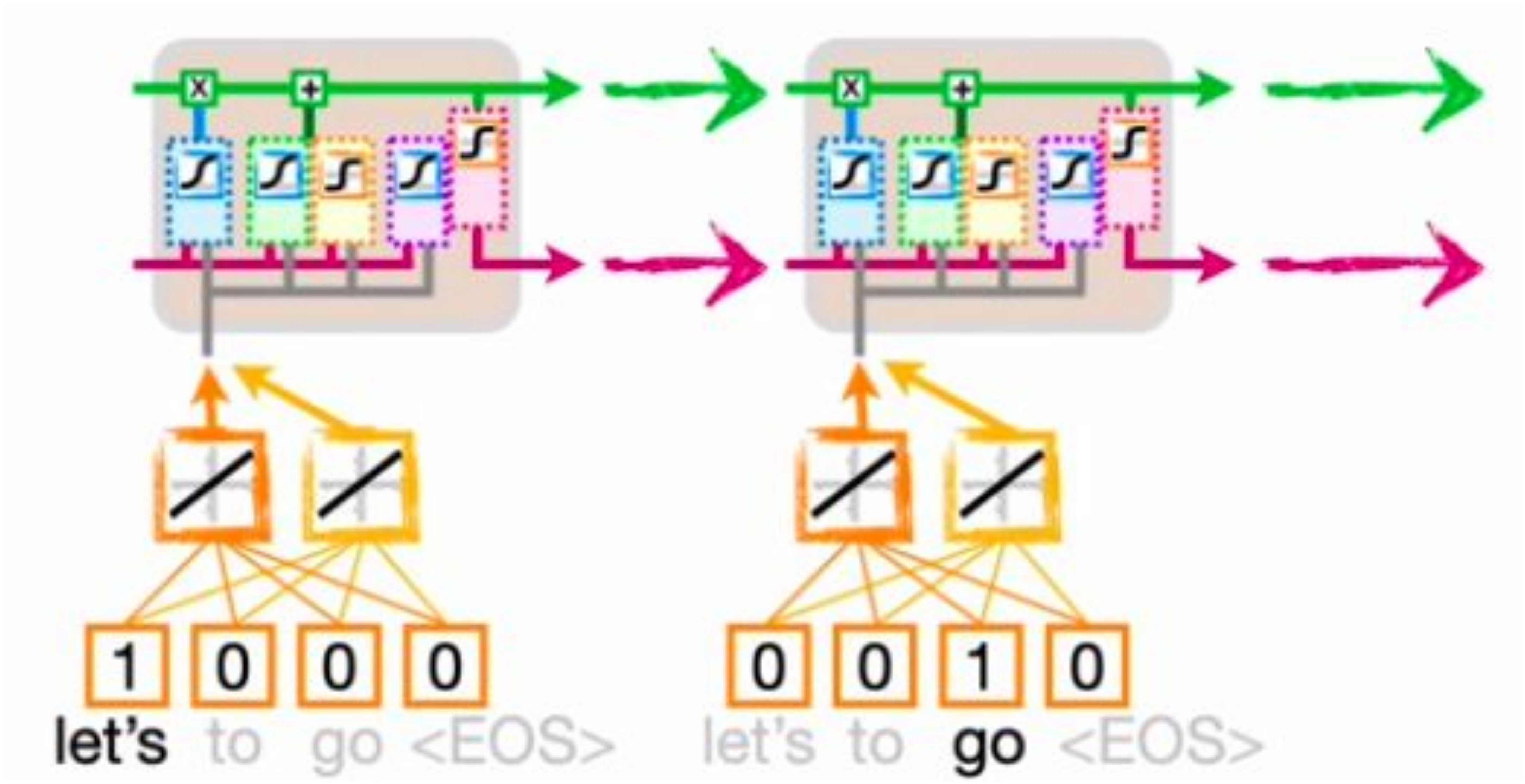




Encoder-Decoder

Seq2Seq Models

- Encoder: one layer

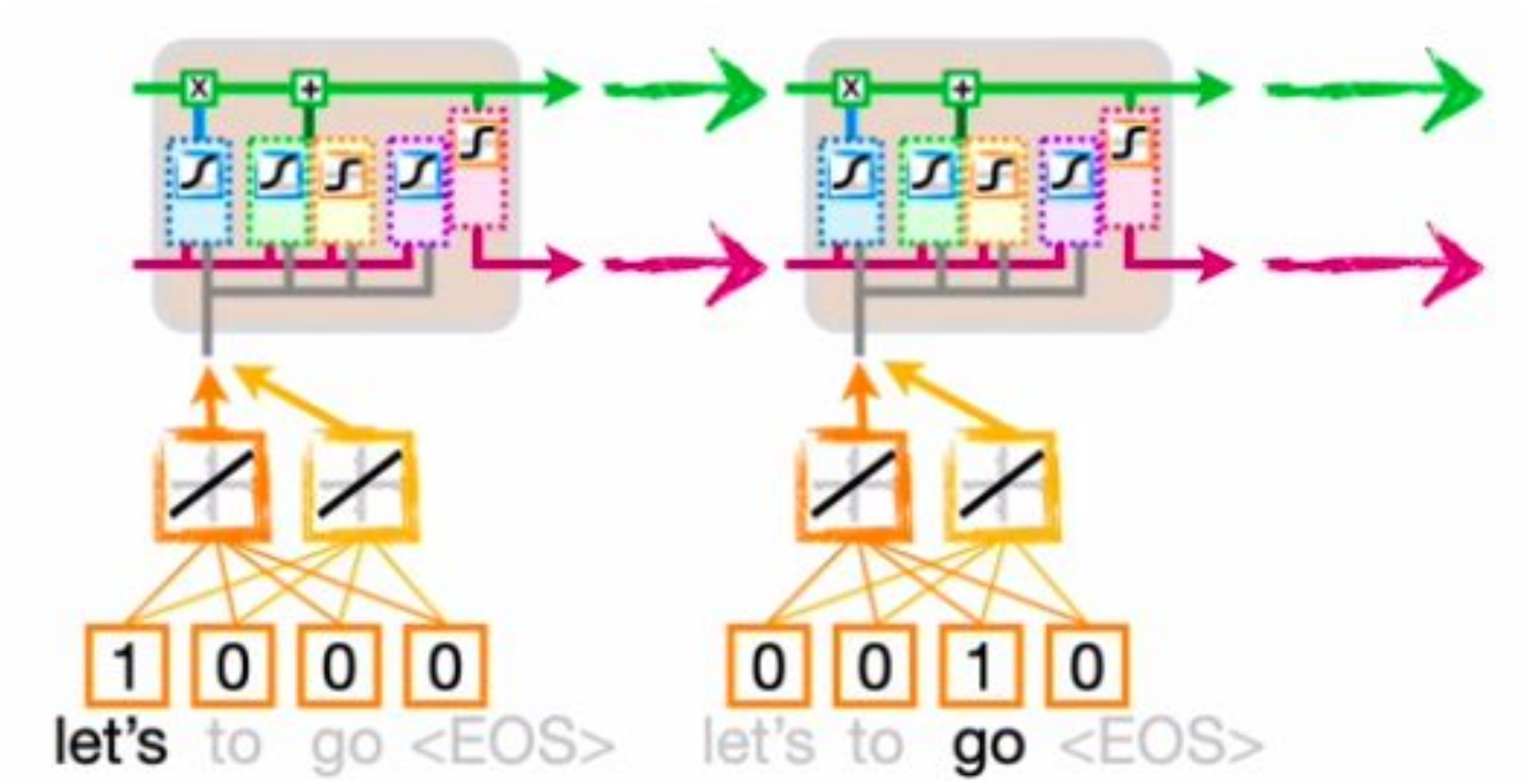


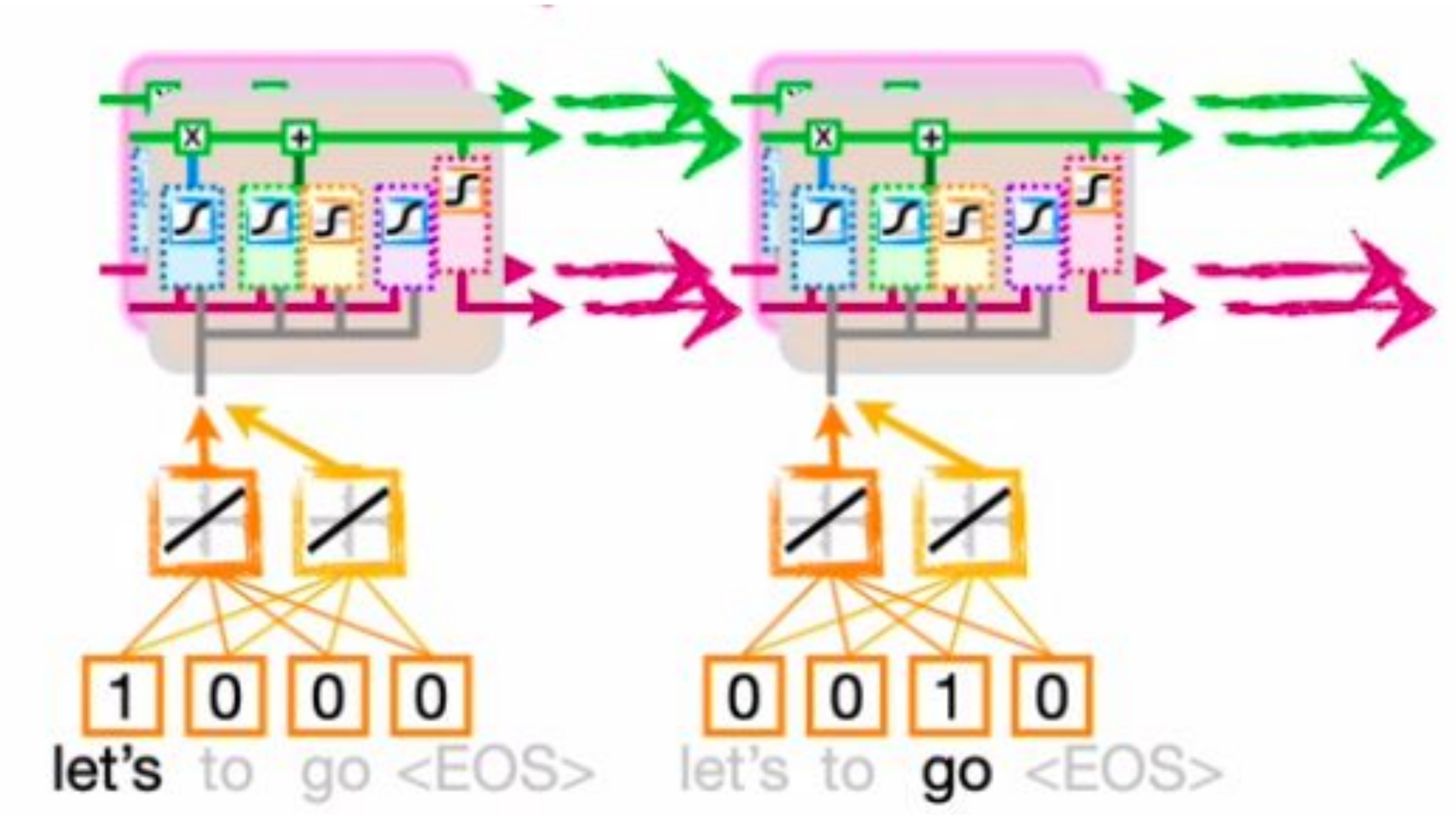
Encoder-Decoder

Seq2Seq Models

_ Encoder: one layer

- But we can have more than one cell

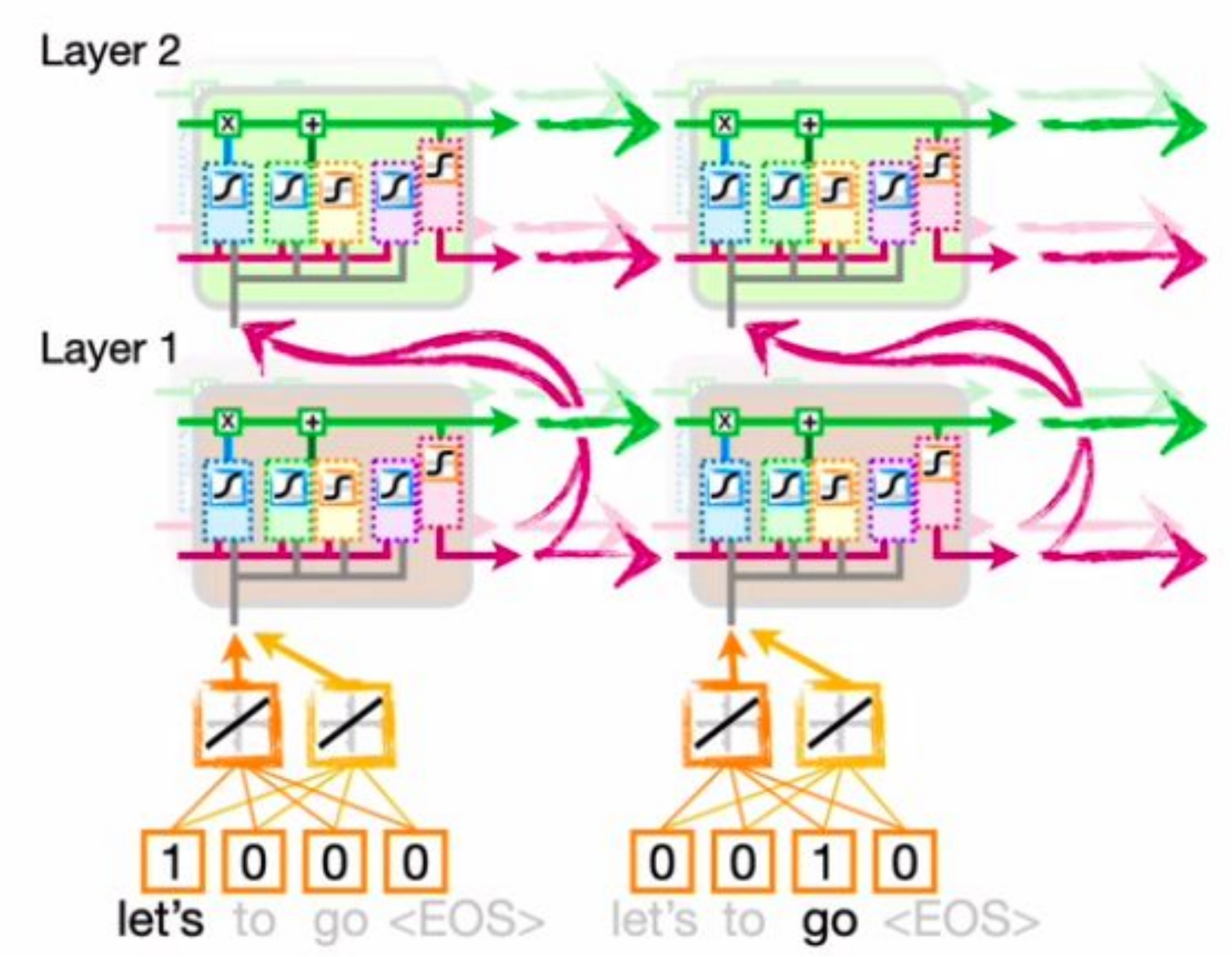




Encoder-Decoder

Seq2Seq Models

- Encoder: we can add more layers

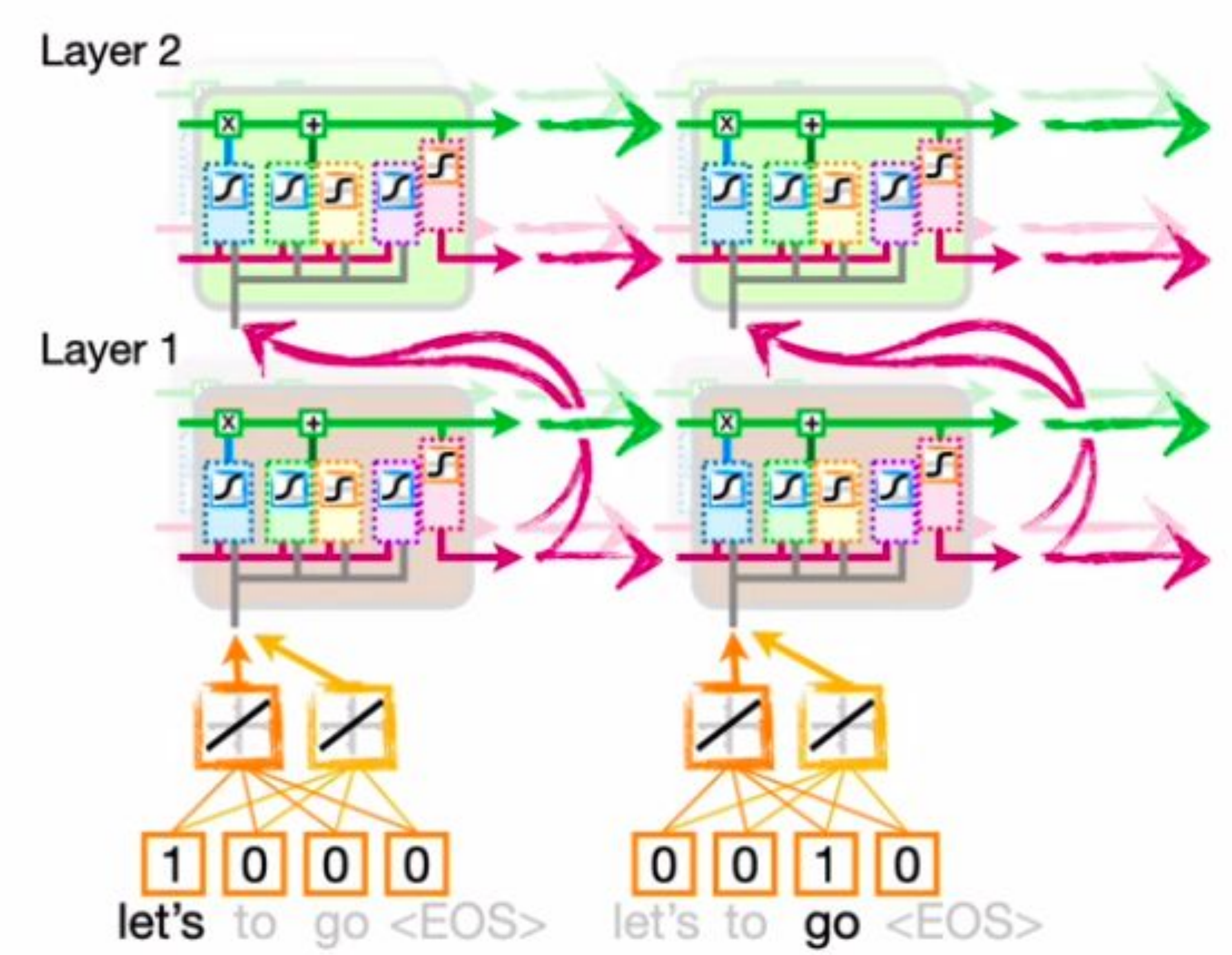


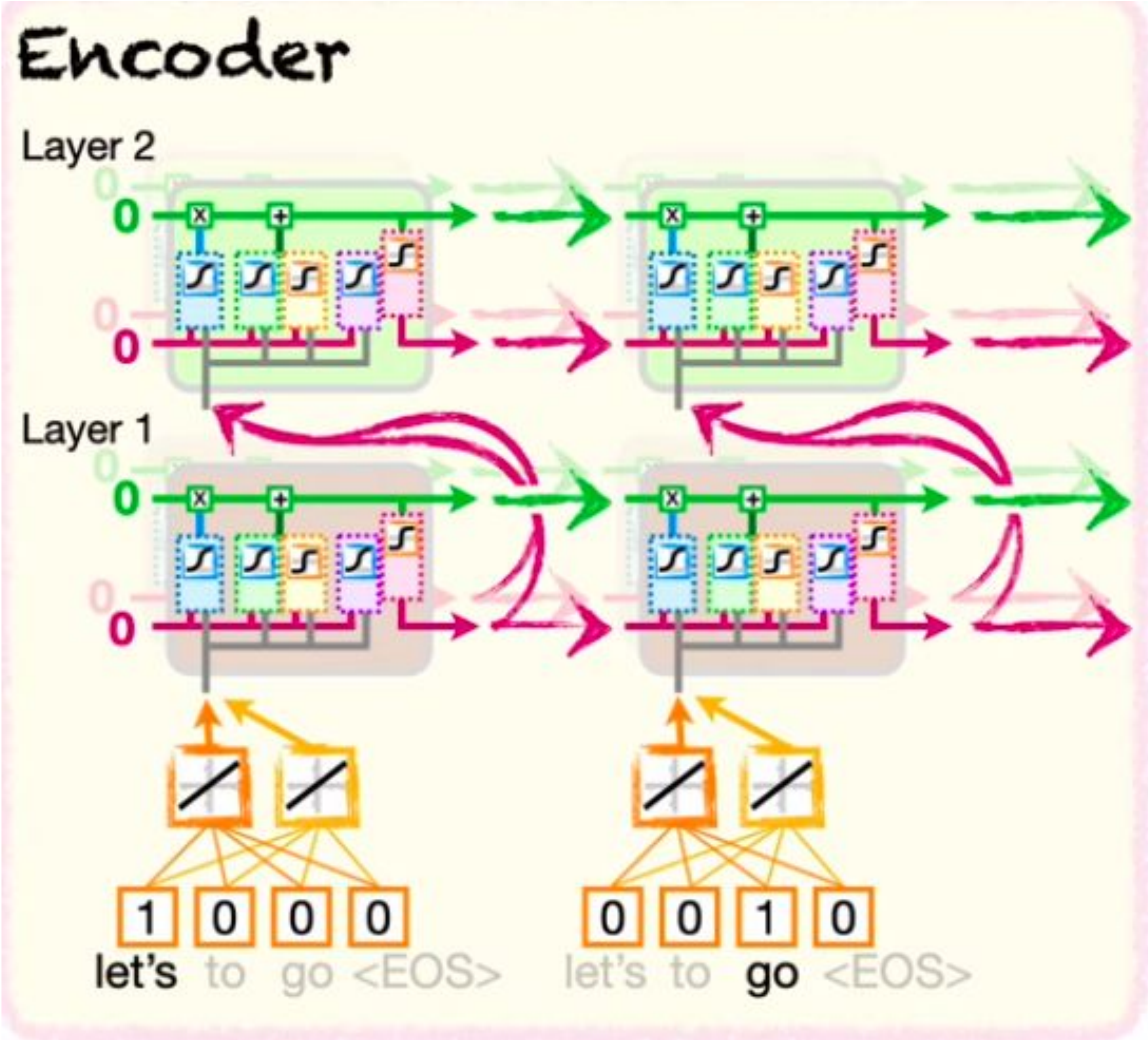
Encoder-Decoder

Seq2Seq Models

_ Encoder: we can add more layers

- And we have the encoder



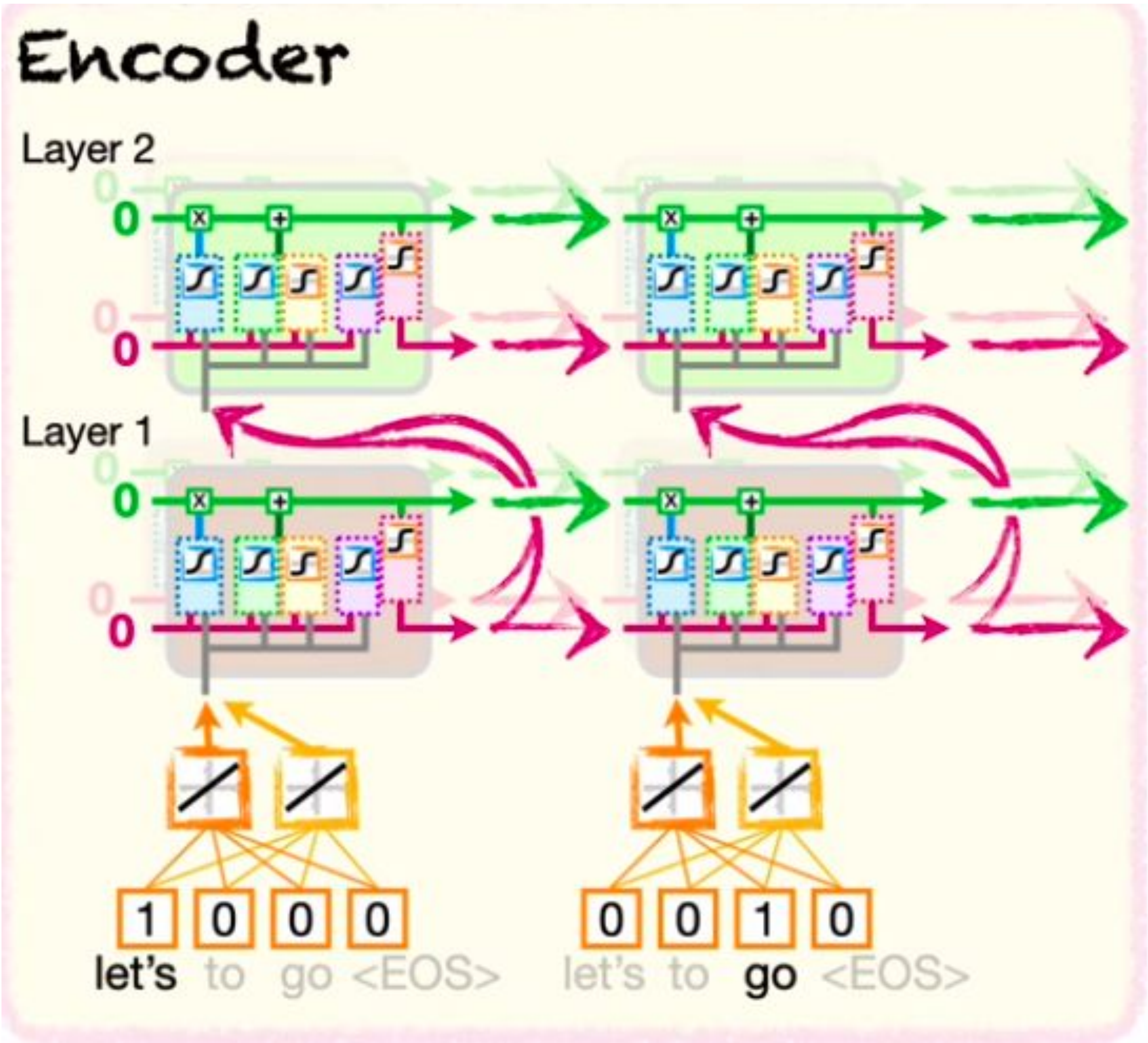


Encoder-Decoder

Seq2Seq Models

_ Encoder: we can add more layers

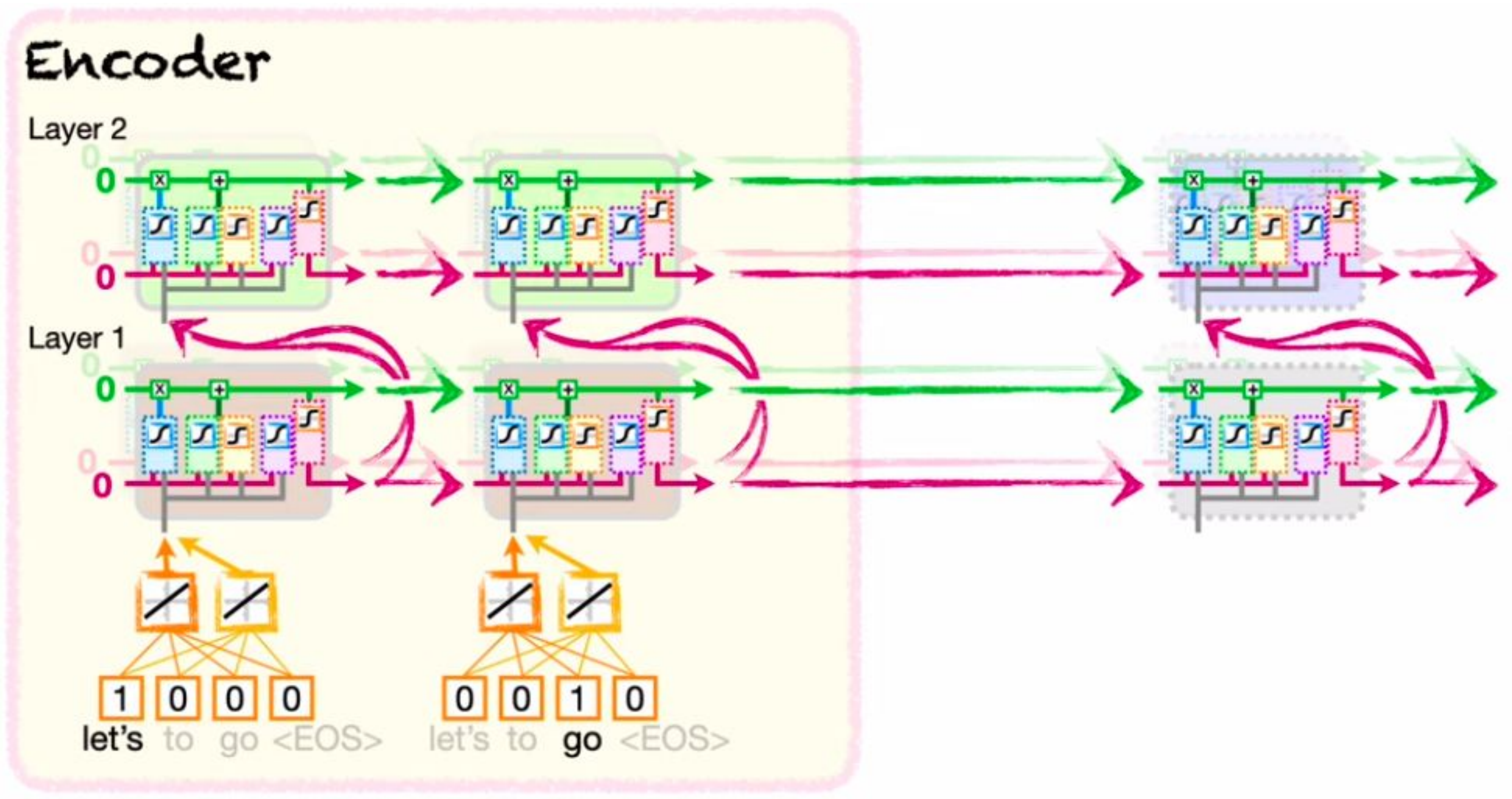
- And we have the encoder
  - We call the last long and short-term memories as **context vectors**
    - These vectors will initialize the memories in the decoder



Encoder-Decoder

Seq2Seq Models

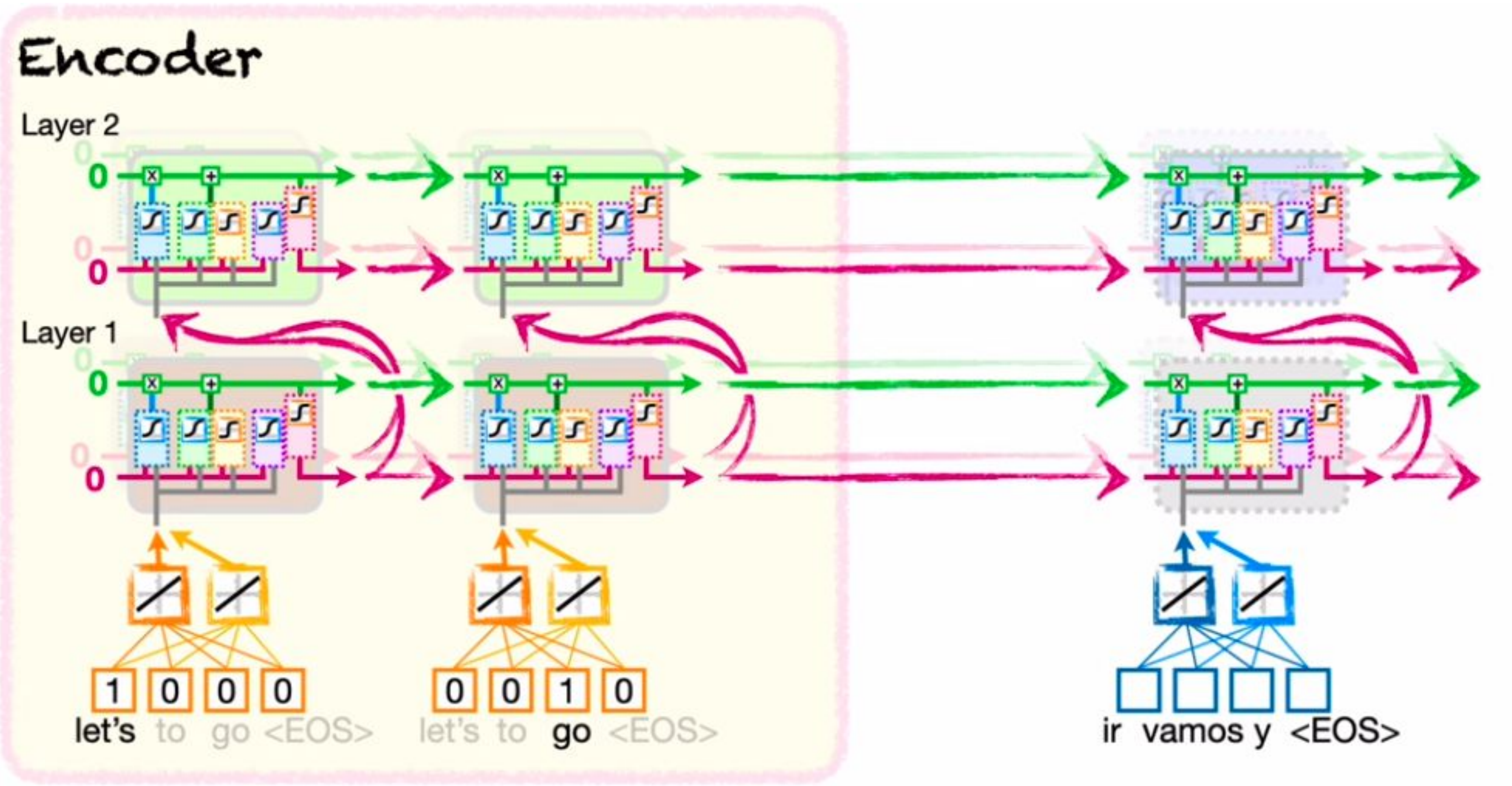
_ Decoder



Encoder-Decoder

Seq2Seq Models

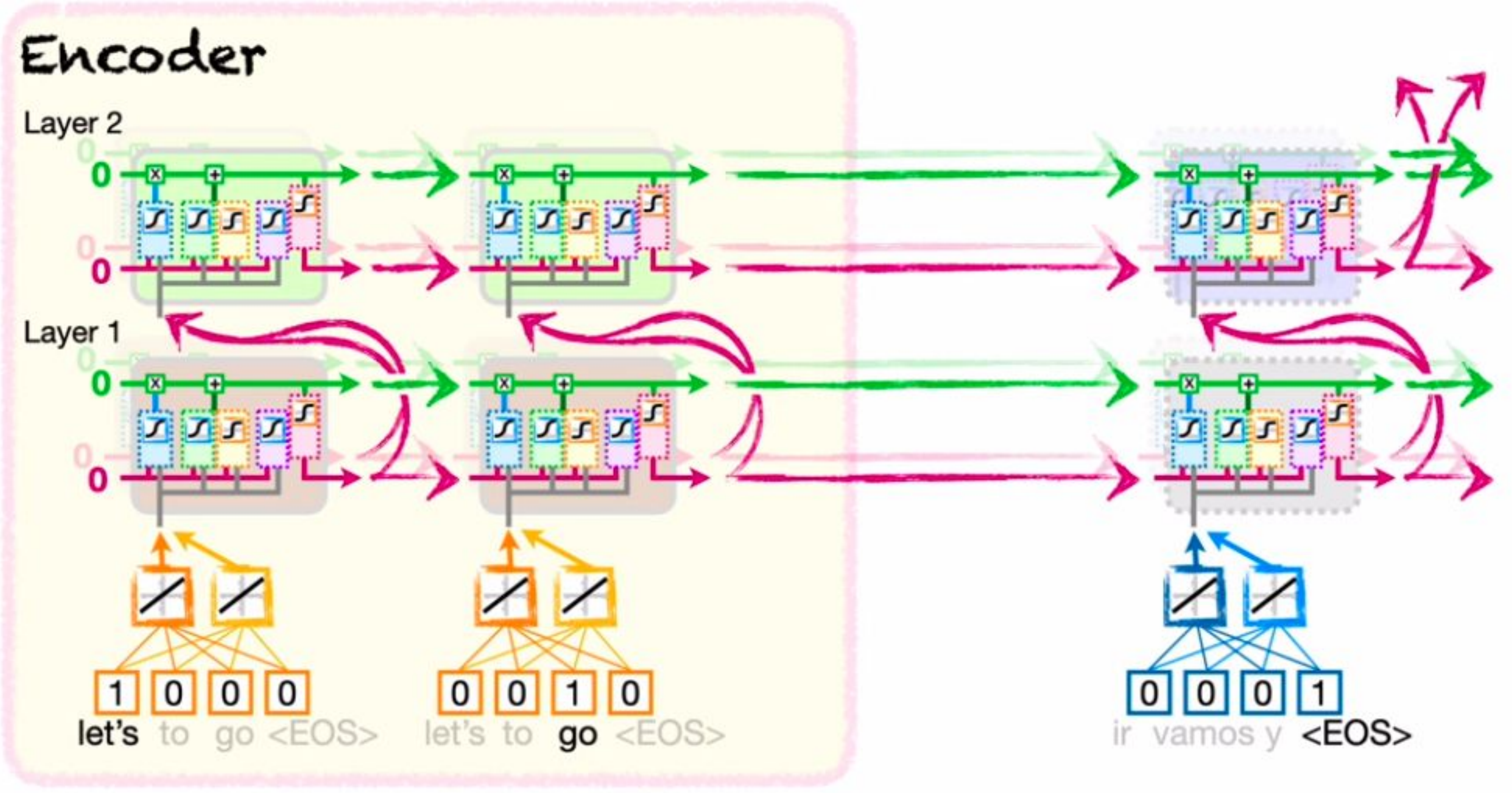
- Decoder



Encoder-Decoder

Seq2Seq Models

_ Decoder

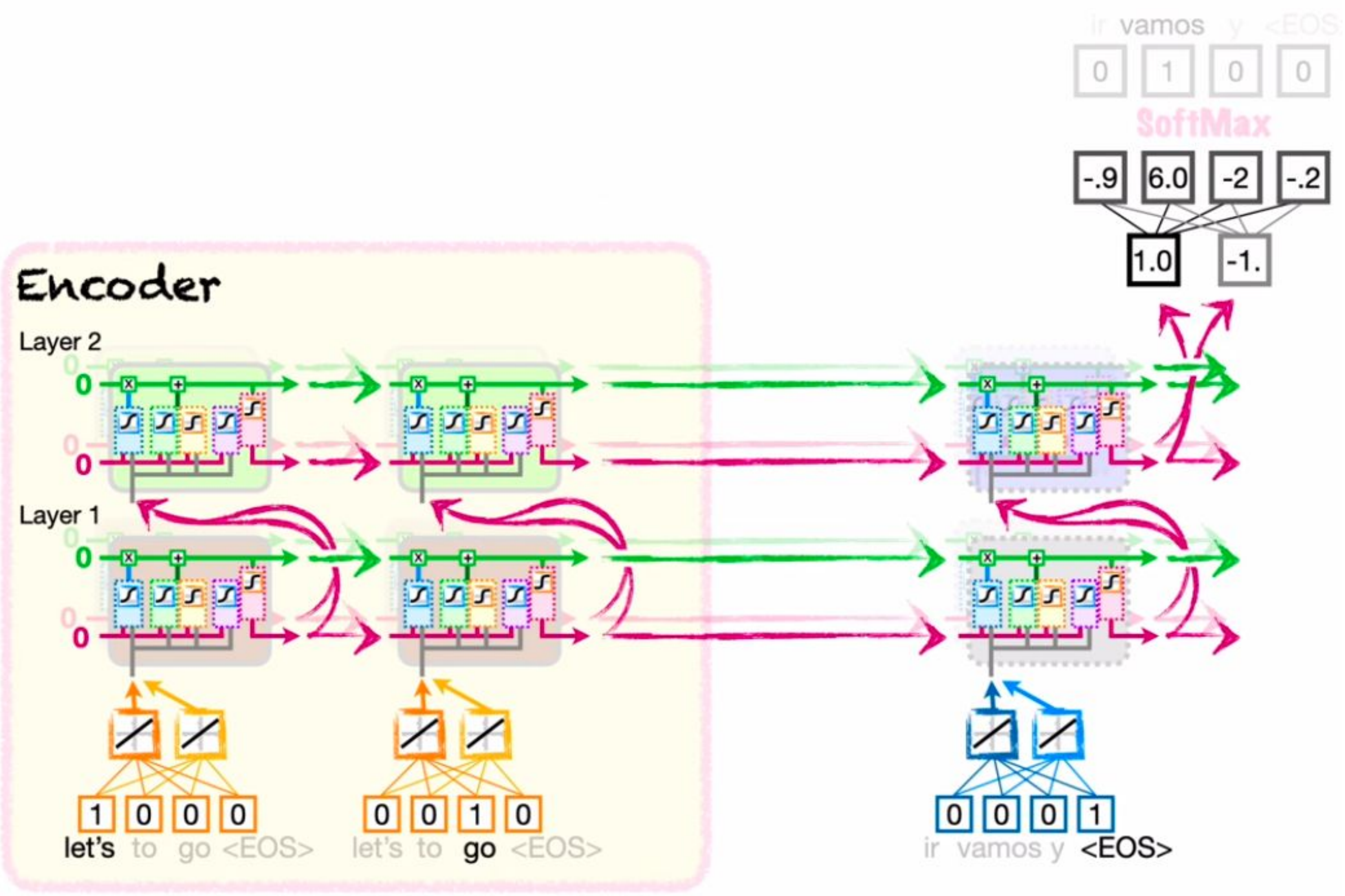


Encoder-Decoder

Seq2Seq Models

_ Decoder

- Fully-connected + Softmax

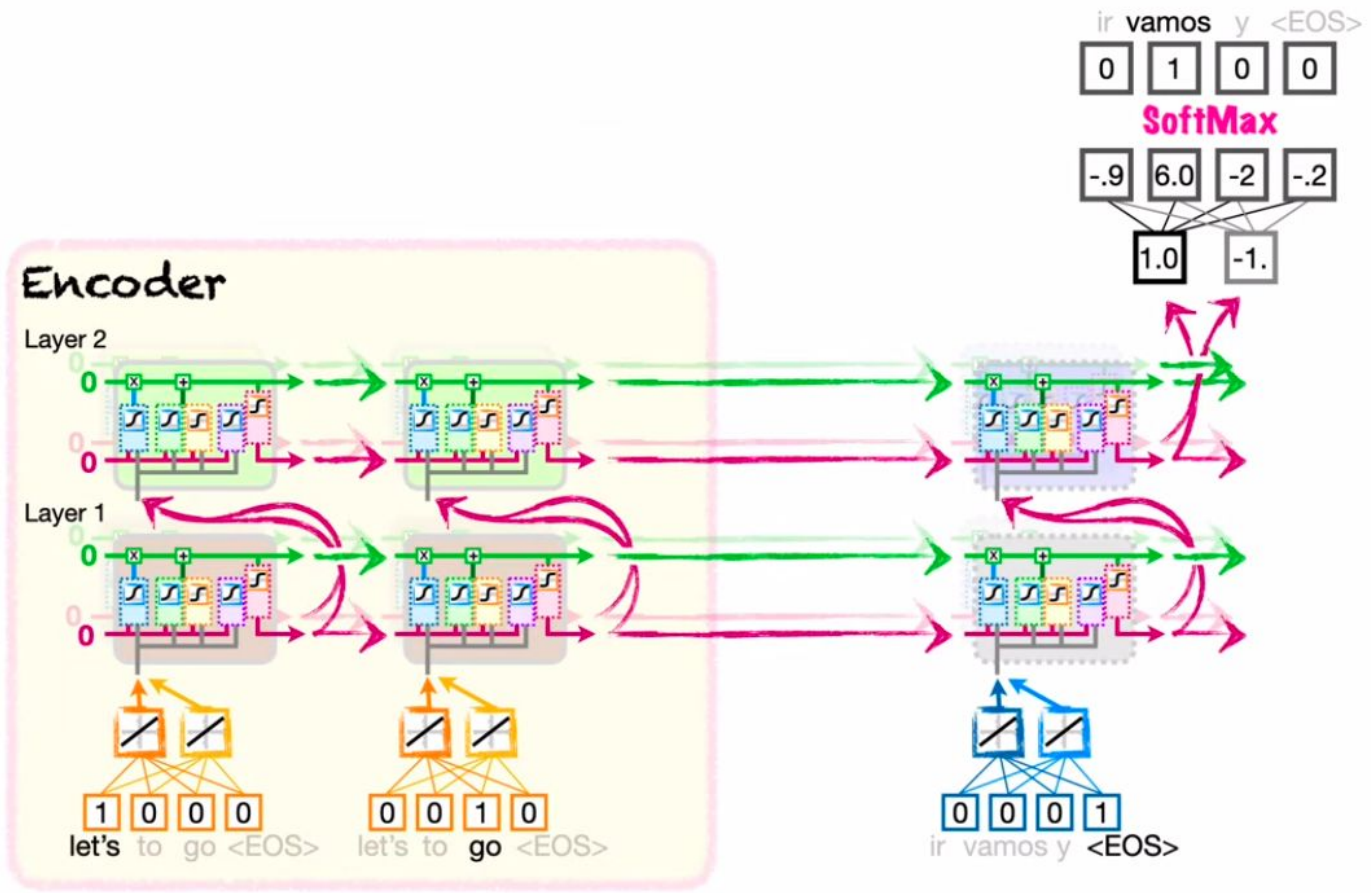


Encoder-Decoder

Seq2Seq Models

_ Decoder

- Fully-connected + Softmax

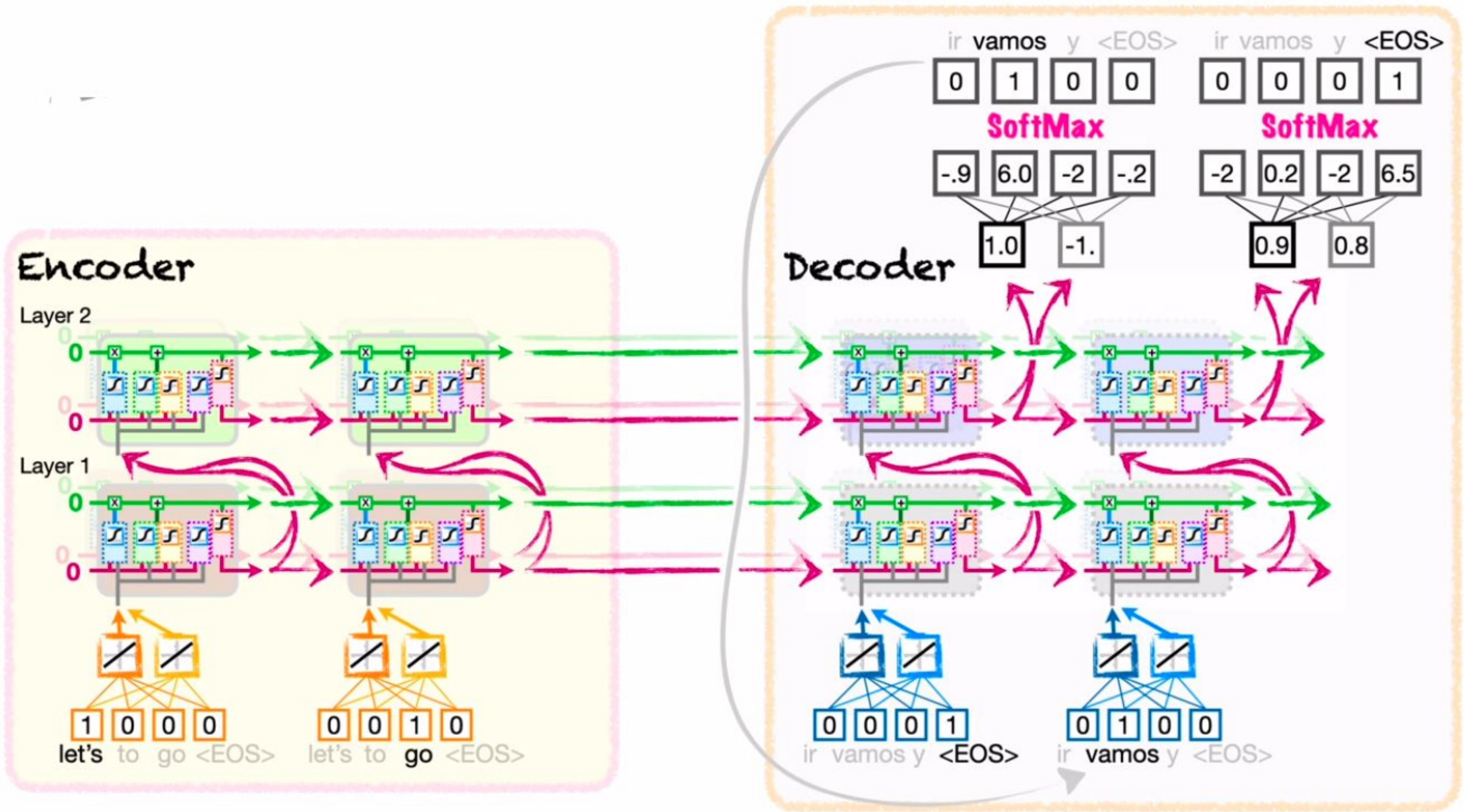


Encoder-Decoder

Seq2Seq Models

_ Decoder

- Fully-connected + Softmax
  - Weights → backpropagation



Encoder-Decoder

Seq2Seq Models

Decoder

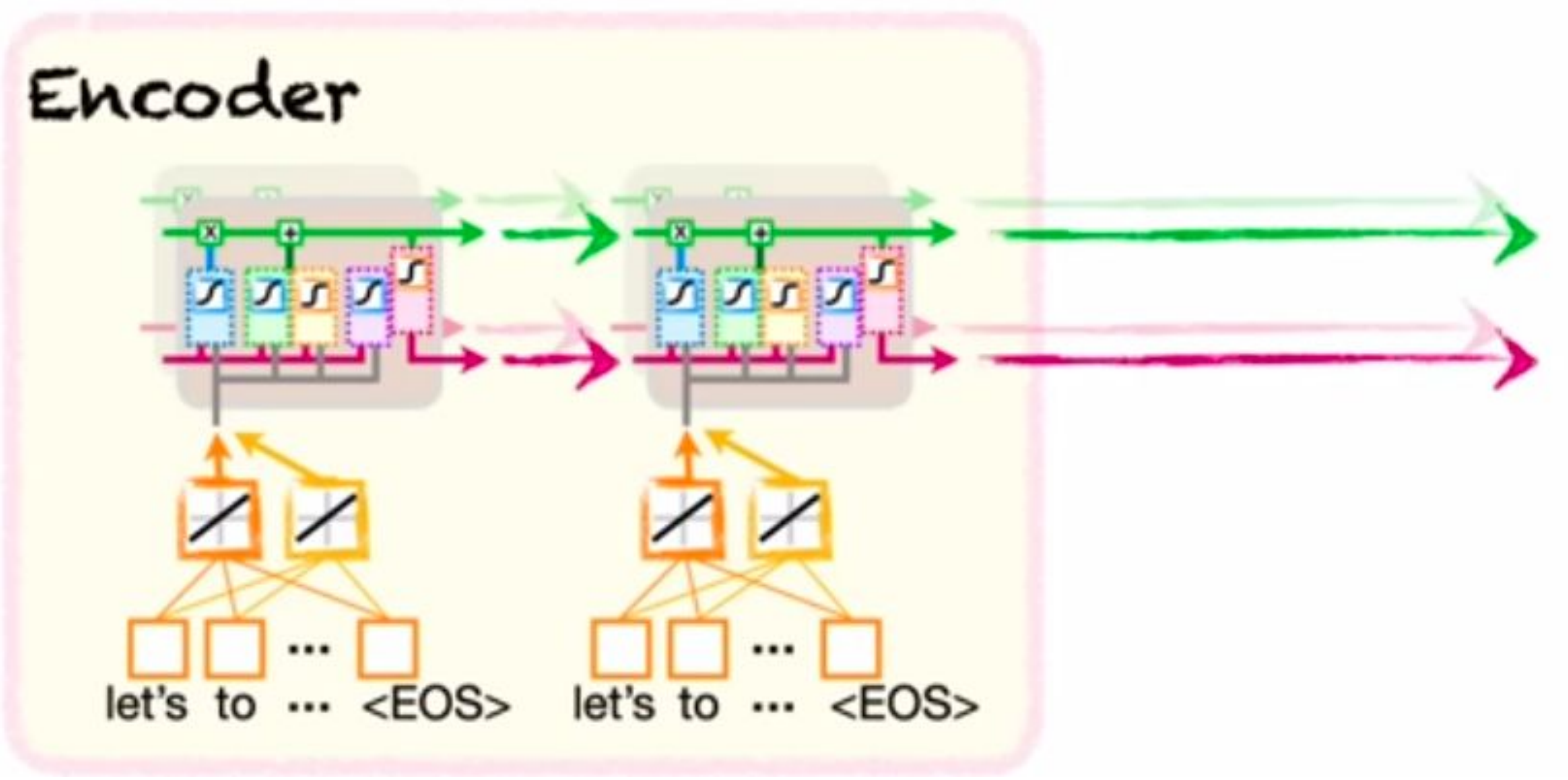
- Fully-connected + Softmax
  - Weights → backpropagation
    - During training, instead of using the predicted token as the next word, we always use the correct one.

Attention

Seq2Seq Models with Attention

_ The problem with long sentences

- Seq2Seq models work very well for very short sentences.
  - But their performance decreases drastically with long sentences.
  - Why this happens?
    - On a typical Seq2Seq model, the LSTM cell compresses the entire input sequence into a single context vector.

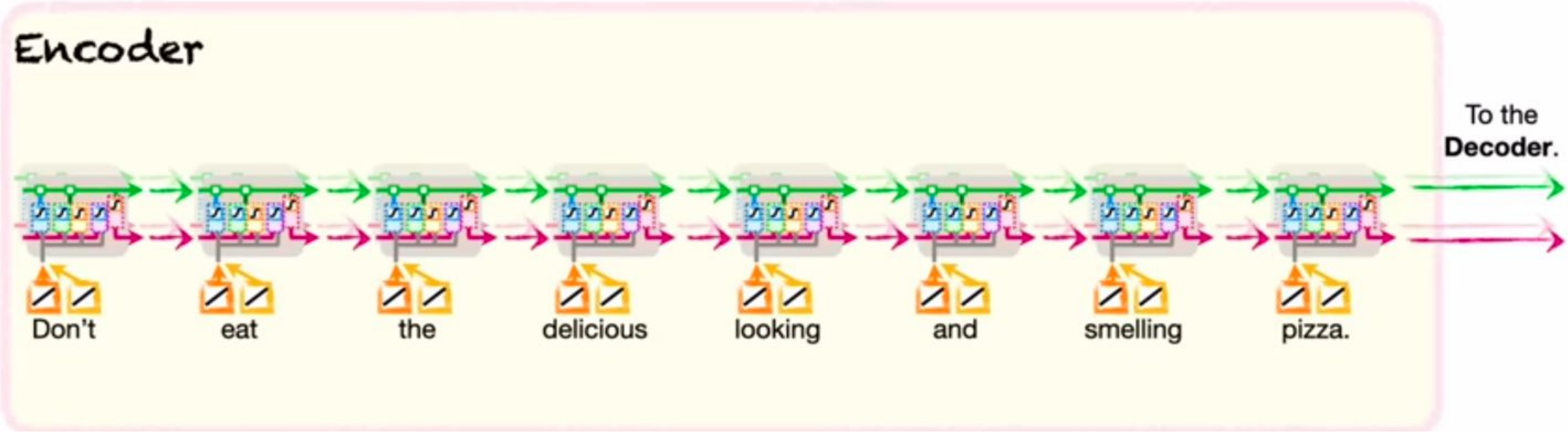


Attention

Seq2Seq Models with Attention

_ The problem with long sentences

- For longer input sentences, even with LSTMs, words that are input earlier are likely to be forgotten.

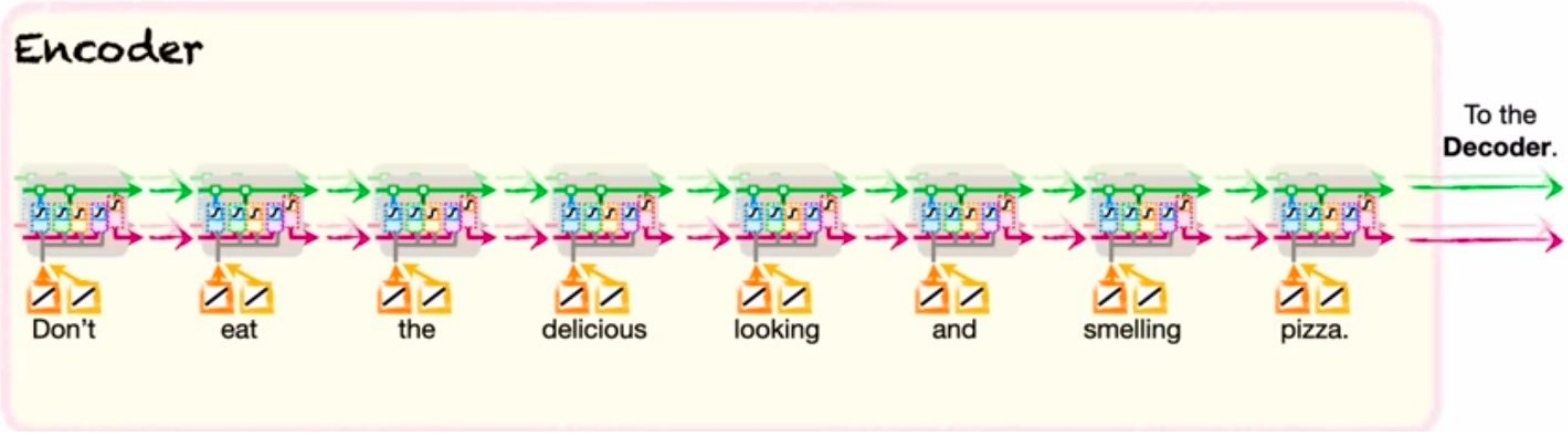


Attention

Seq2Seq Models with Attention

_ The problem with long sentences

- In this case, if we forget the first word, then the sentences would have completely opposite meanings.

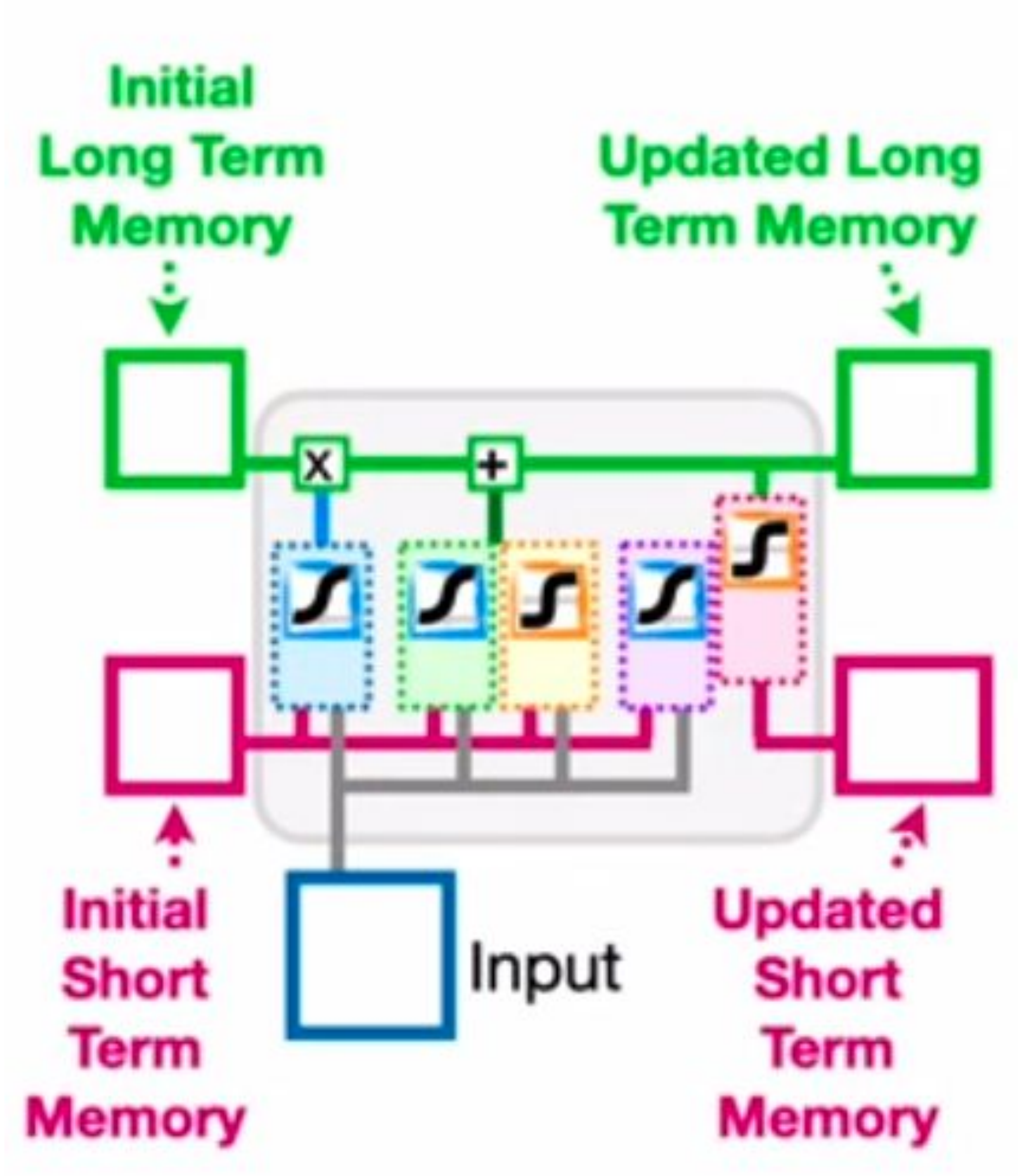


Attention

Seq2Seq Models with Attention

_ Remember the motivation for LSTMs.

- Typical recurrent neural networks have problems with long-term memory because they run both long and short-term memories through a same vector.
    - LSTMs, on the other hand, solve this problem by providing separate paths for long and short-term memories.

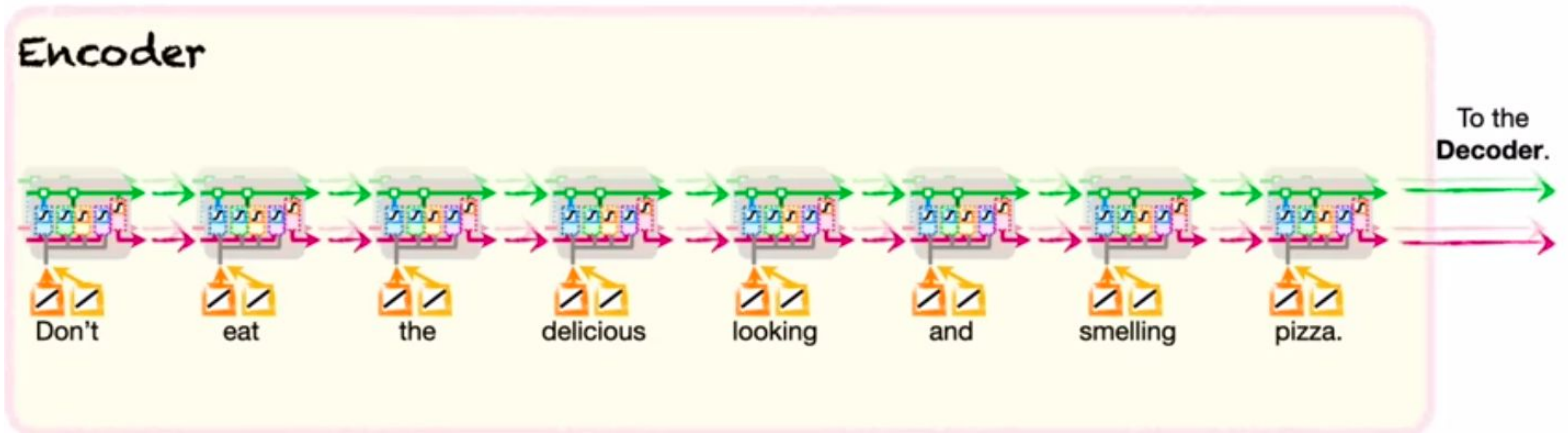


Attention

Seq2Seq Models with Attention

_ Remember the motivation for LSTMs.

- Even with separate memories, if we have a long input sentence, then booths paths have to carry a lot of information.
  - This means that a word at the start of a long sentence, can still get lost.

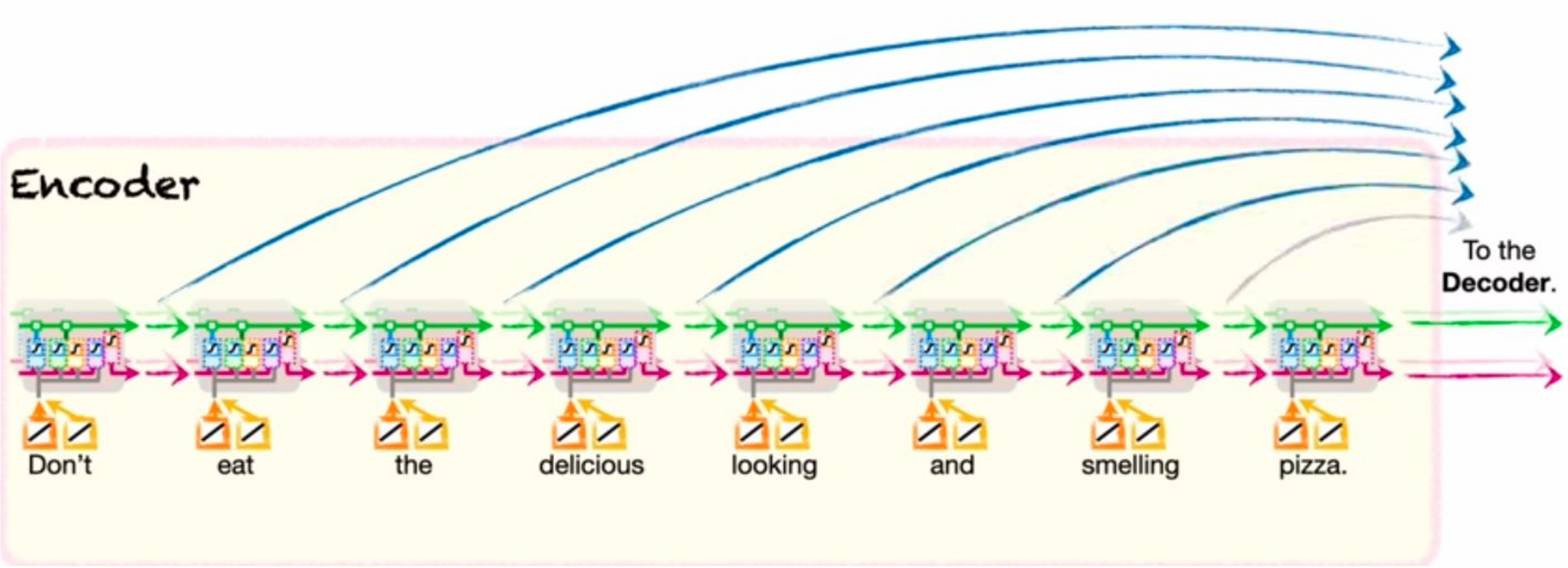


Attention

Seq2Seq Models with Attention

_ The intuition behind attention in neural networks.

- Add direct paths from the encoder to the decoder.
  - One per input word, so that each step of the decoder can directly access input words.



Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

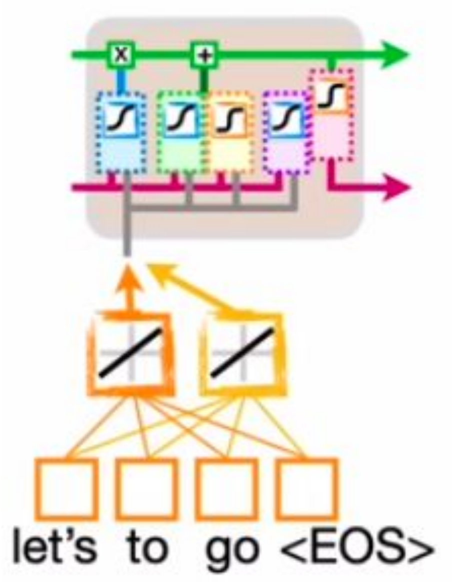
- We have an embedding layer

Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- We have an embedding layer
- And we add a LSTM cell

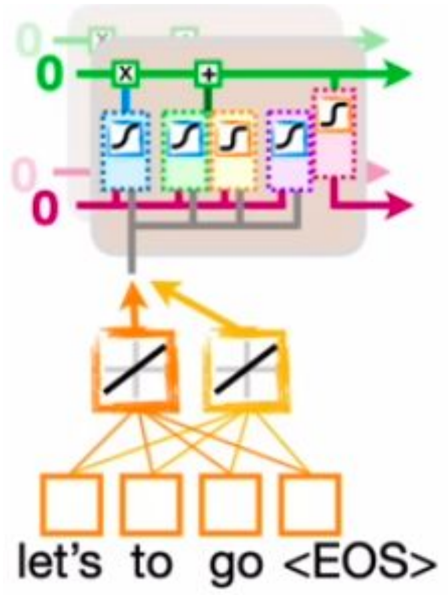


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- We have an embedding layer
- And we add a LSTM cell
  - Then we start the LSTM memories in the encoder with 0's

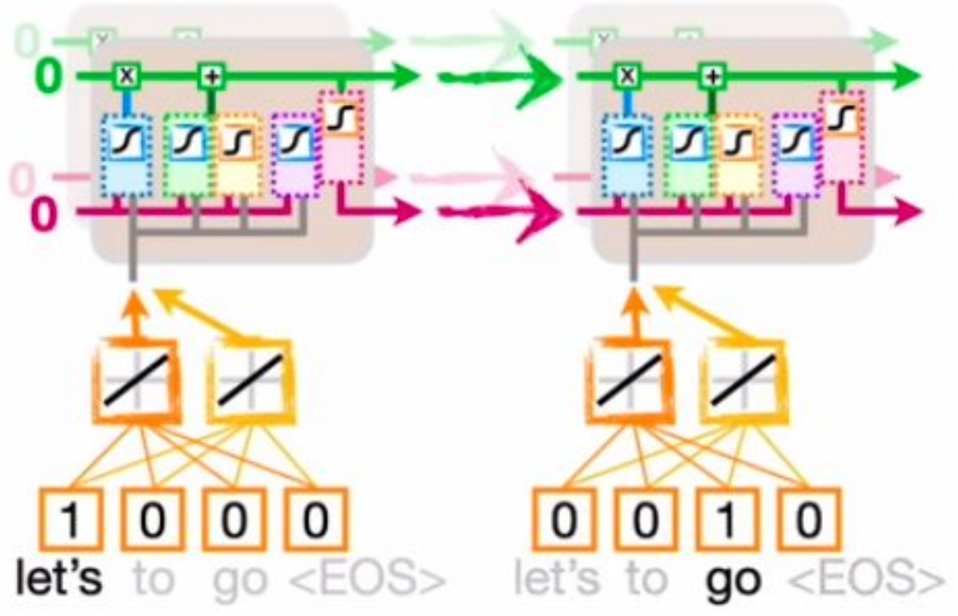


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- We have an embedding layer
- And we add a LSTM cell
    - Then we start the LSTM memories in the encoder with 0's
- Now, we add more context (more LSTM cells)
    - And we plug 1 in the embedding layer for the words in the sentence

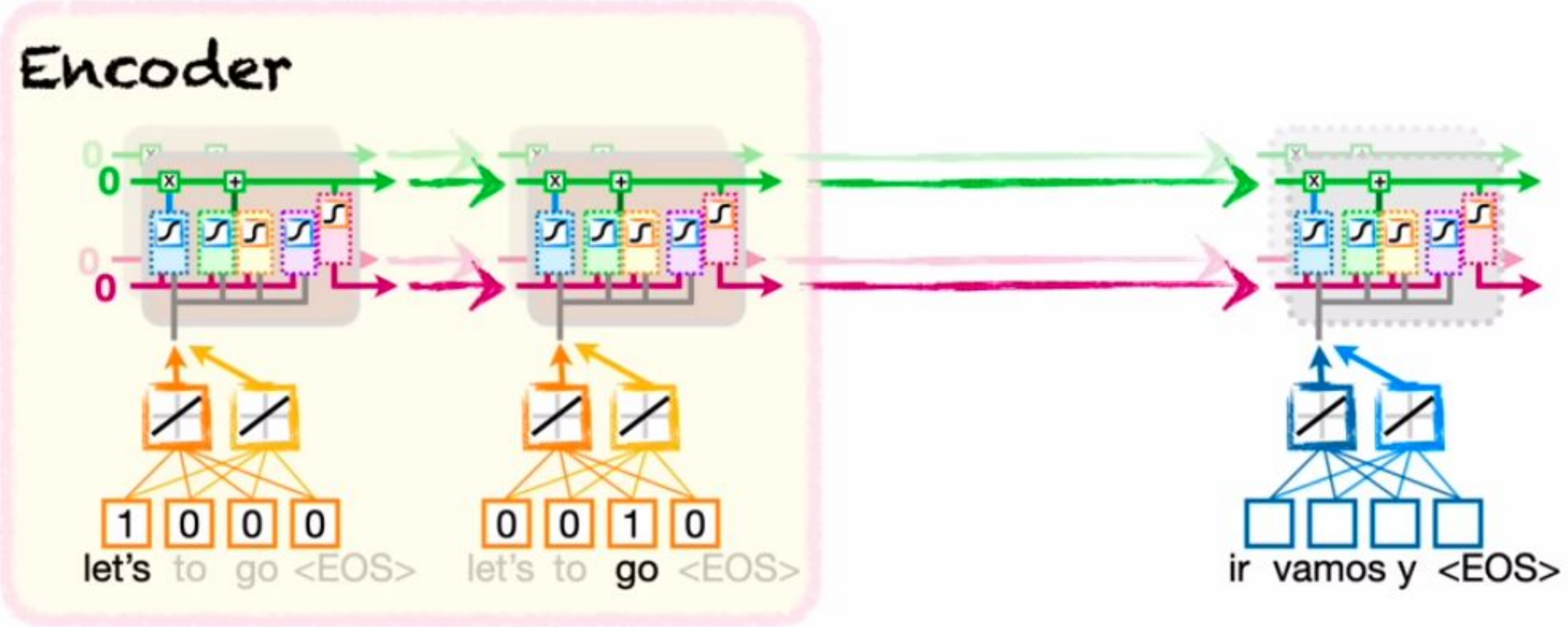


Attention

Seq2Seq Models with Attention

Encoder-Decoder with Attention

- We have an embedding layer
- And we add a LSTM cell
  - Then we start the LSTM memories in the encoder with 0’s
- Now, we add more context (more LSTM cells)
  - And we plug 1 in the embedding layer for the words in the sentence



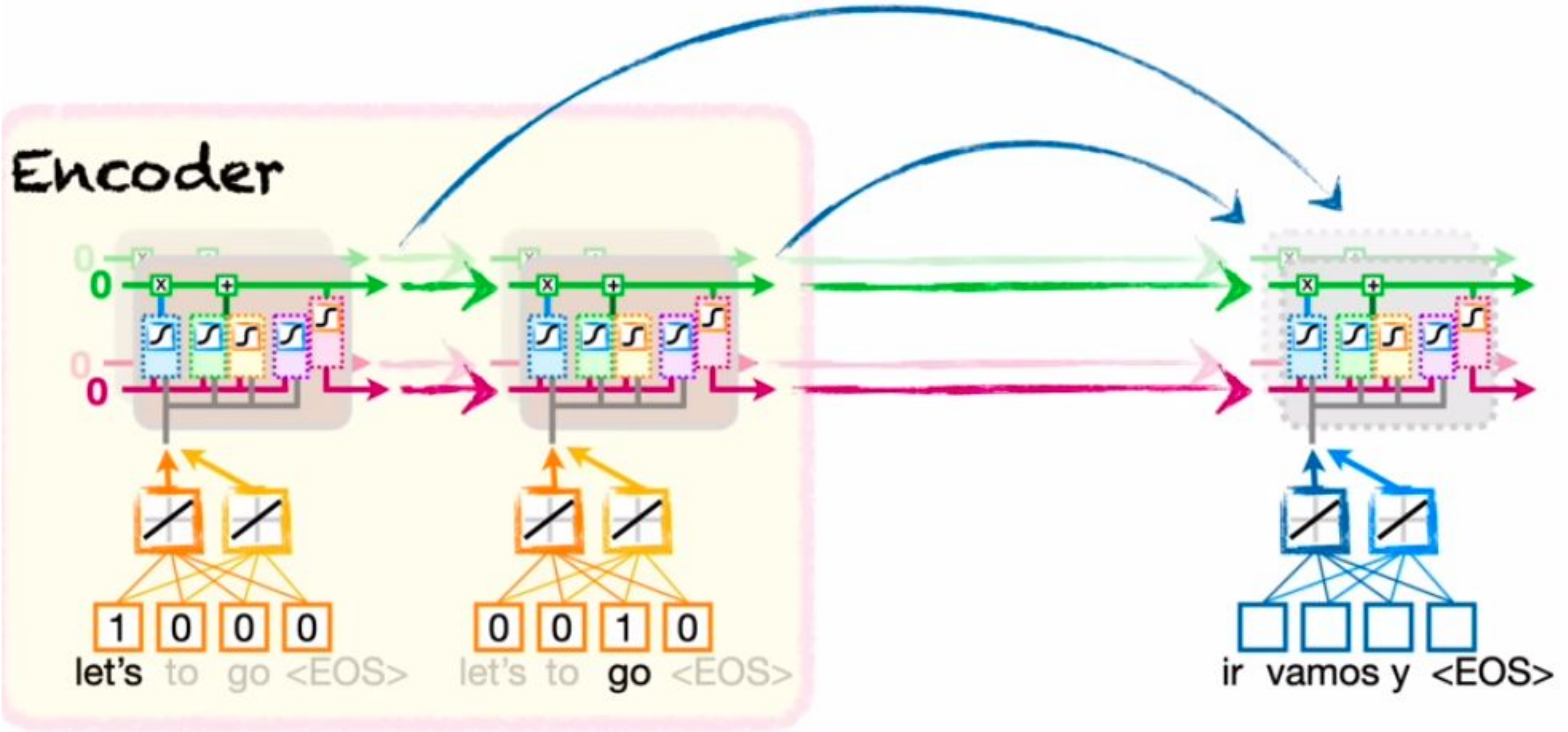
- The encoder is done!

Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Attention does not change the encoder
  - But, each step in the decoder must have direct access to the inputs.
    - How attention connects the decoder to each input in the encoder?

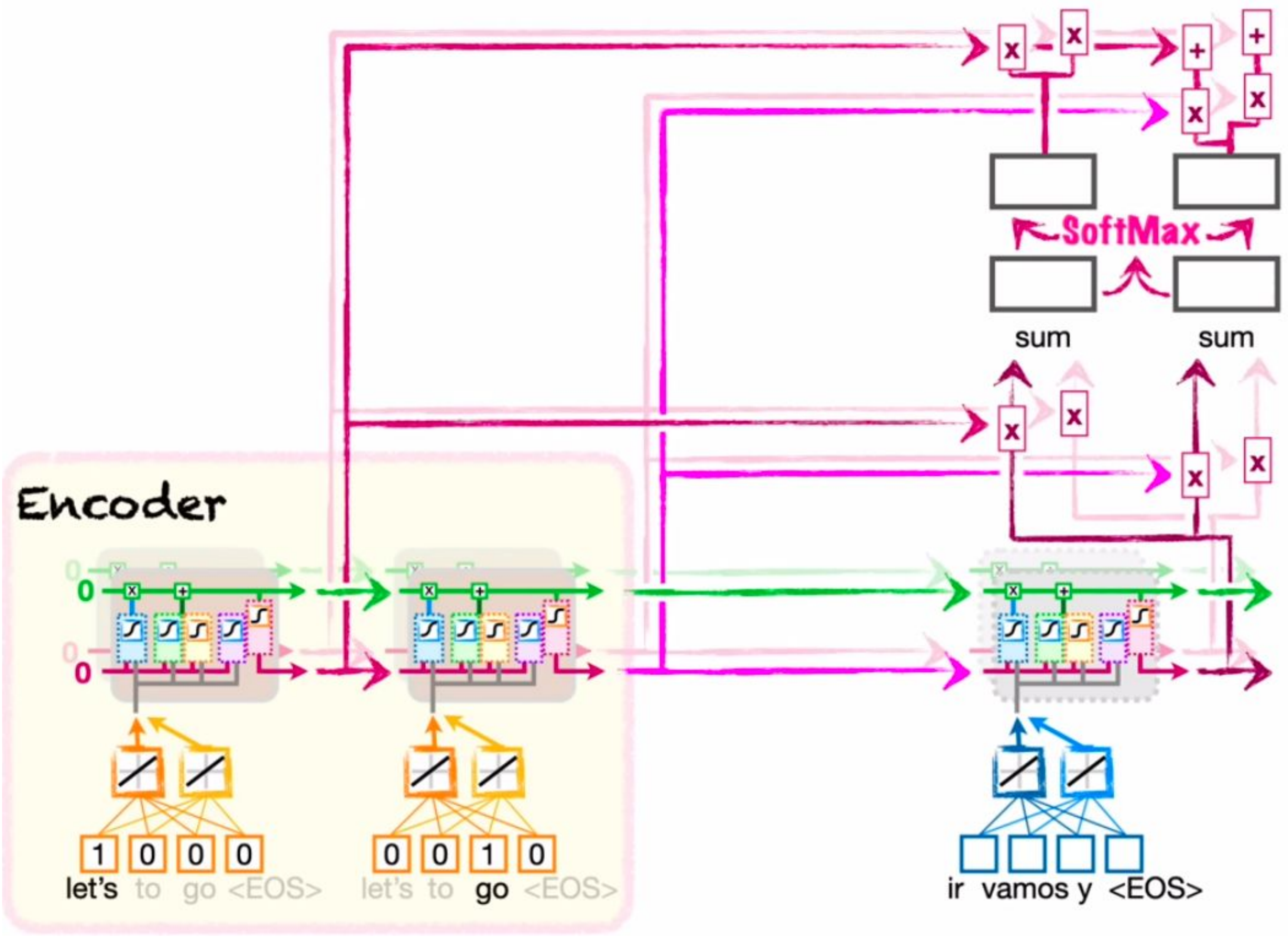


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- There are many forms of implementing Attention, and this is only one of them.

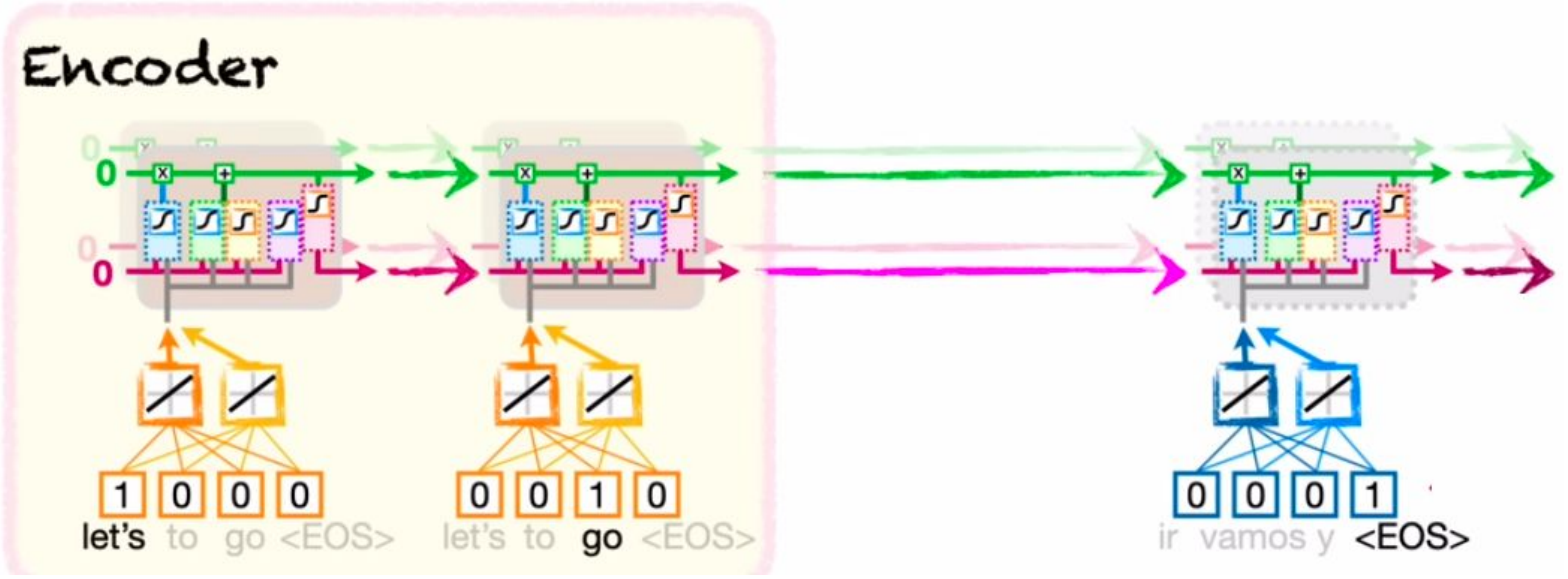


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Let's talk about the decoder.
  - Put 1 on the EOS symbol (EOS can be used as SOS also)

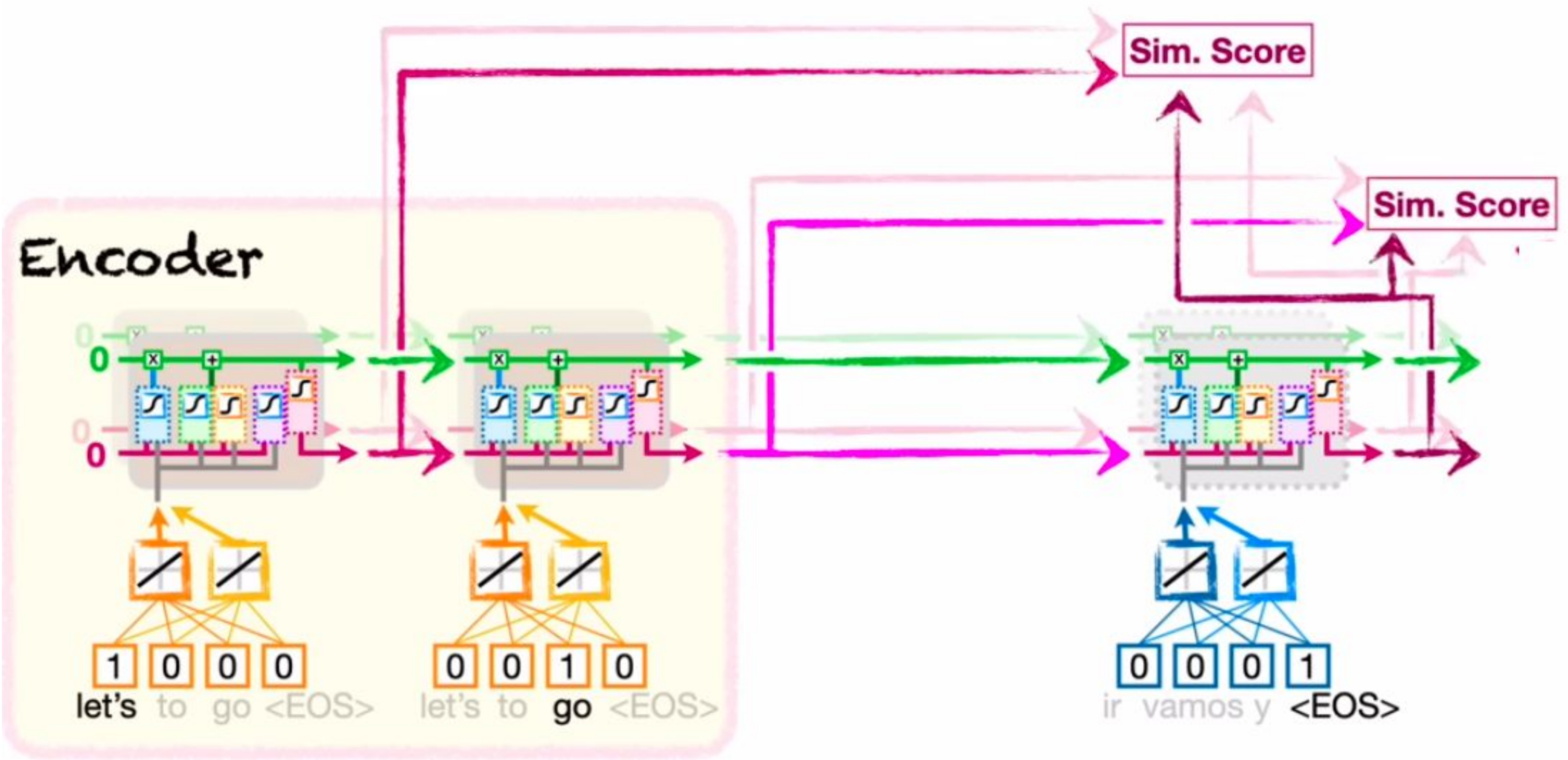


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- The first thing is to determine how similar the inputs from the encoder LSTMs are ate each step to the outputs from the decoder LSTMs.
    - In other words, we need a similarity score between the LSTM inputs (short-term memories) and each step in the decoder.



Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- There many ways to calculate the similarity between vectors.
  - Different attention mechanisms amploy different similarity measures.
    - We will assume the cosine similarity

$$\text{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Attention

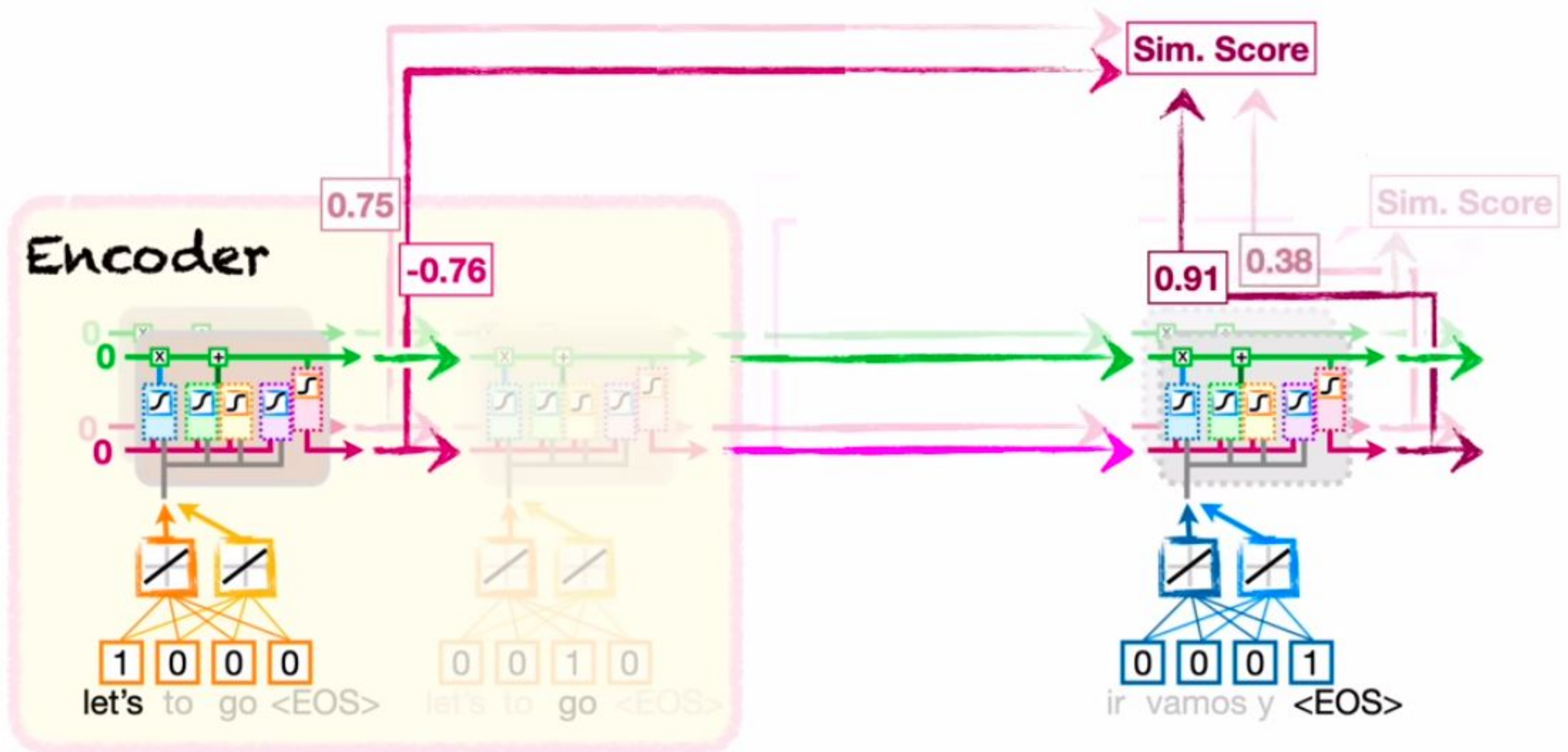
Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Let’s calculate the similarity between the input from the first step in the encoder (“let’s”) and the first output of the decoder (<EOS>).

LSTMs

	Cell #1	Cell #2
A = Encoder = Let’s →	-0.76	0.75
B = Decoder = <EOS> →	0.91	0.38

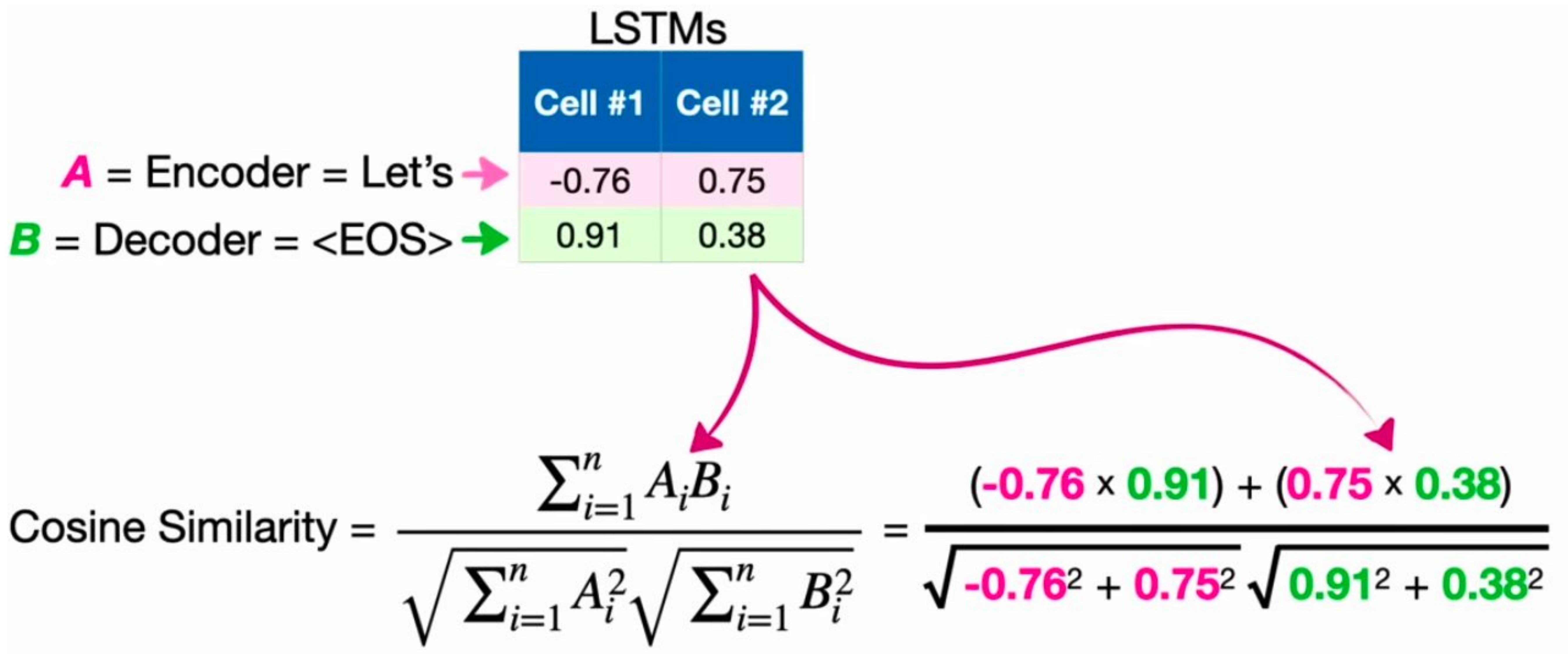


Attention

Seq2Seq Models with Attention

Encoder-Decoder with Attention

- Now we just plug the numbers into the equation for cosine similarity

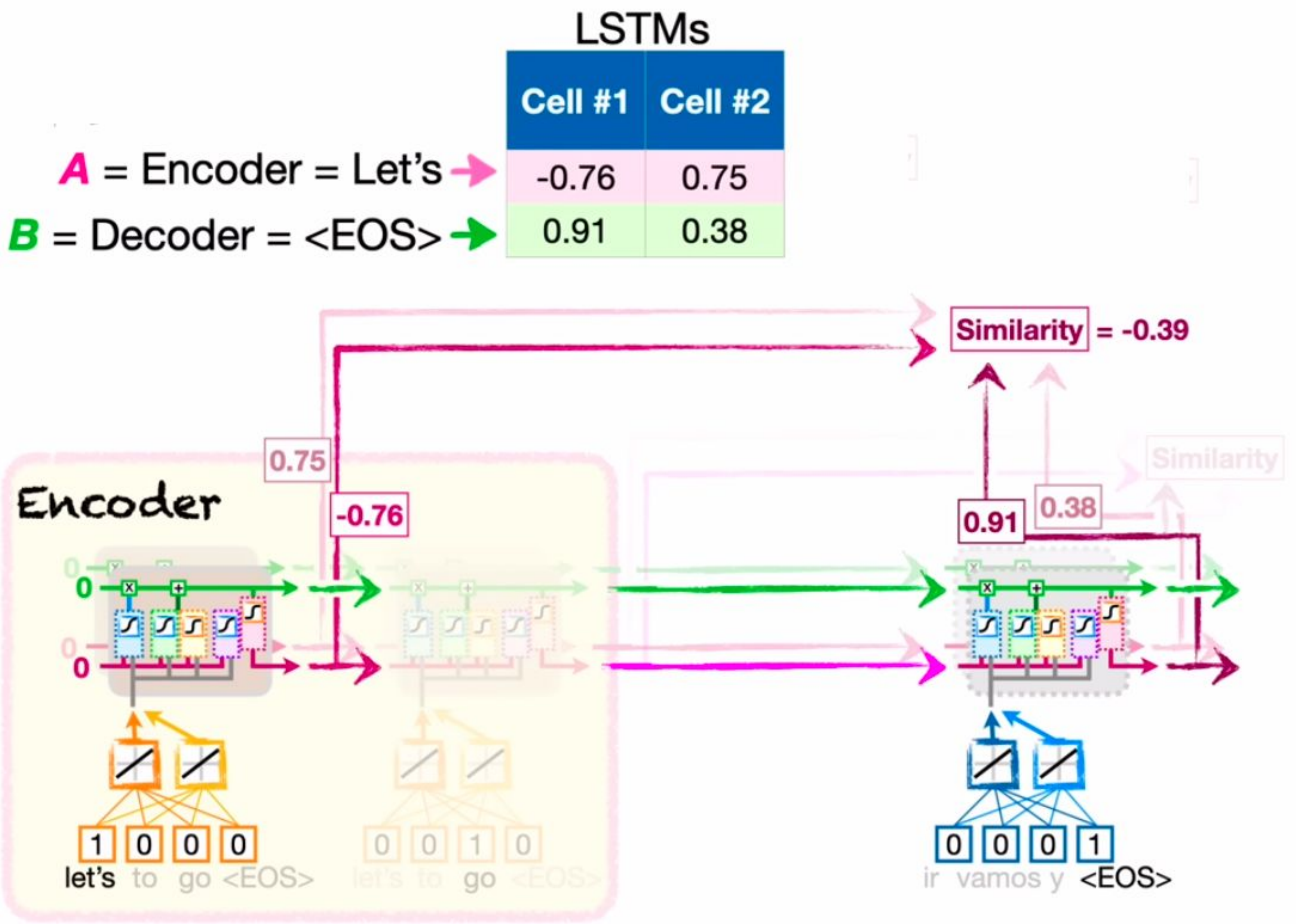


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- The result is -0.39

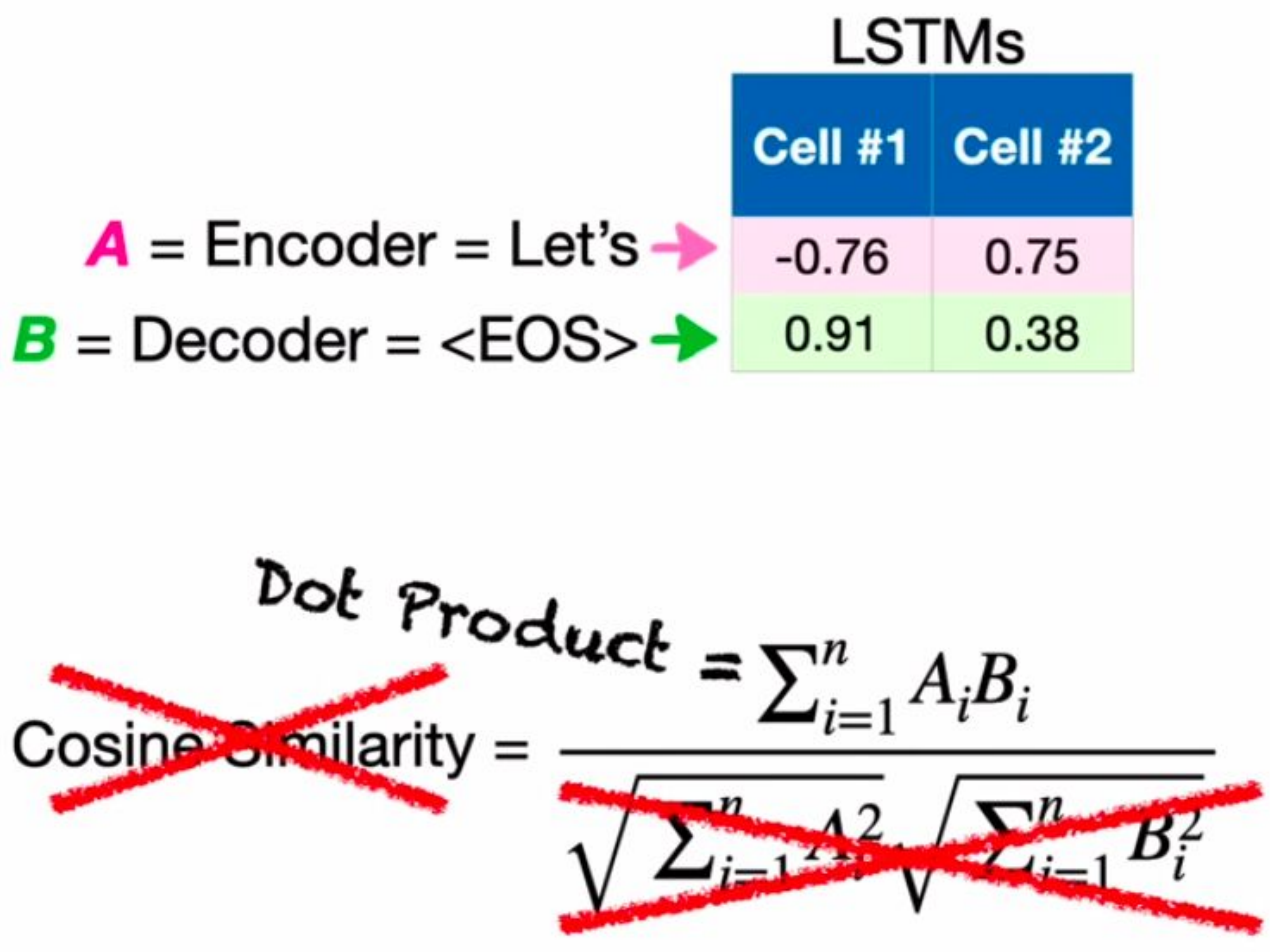


Attention

Seq2Seq Models with Attention

Encoder-Decoder with Attention

- A more common way to calculate the similarity for Attention is just calculate the numerator.
  - The dot product.
    - We do not need to scale the magnitude between -1 and 1.

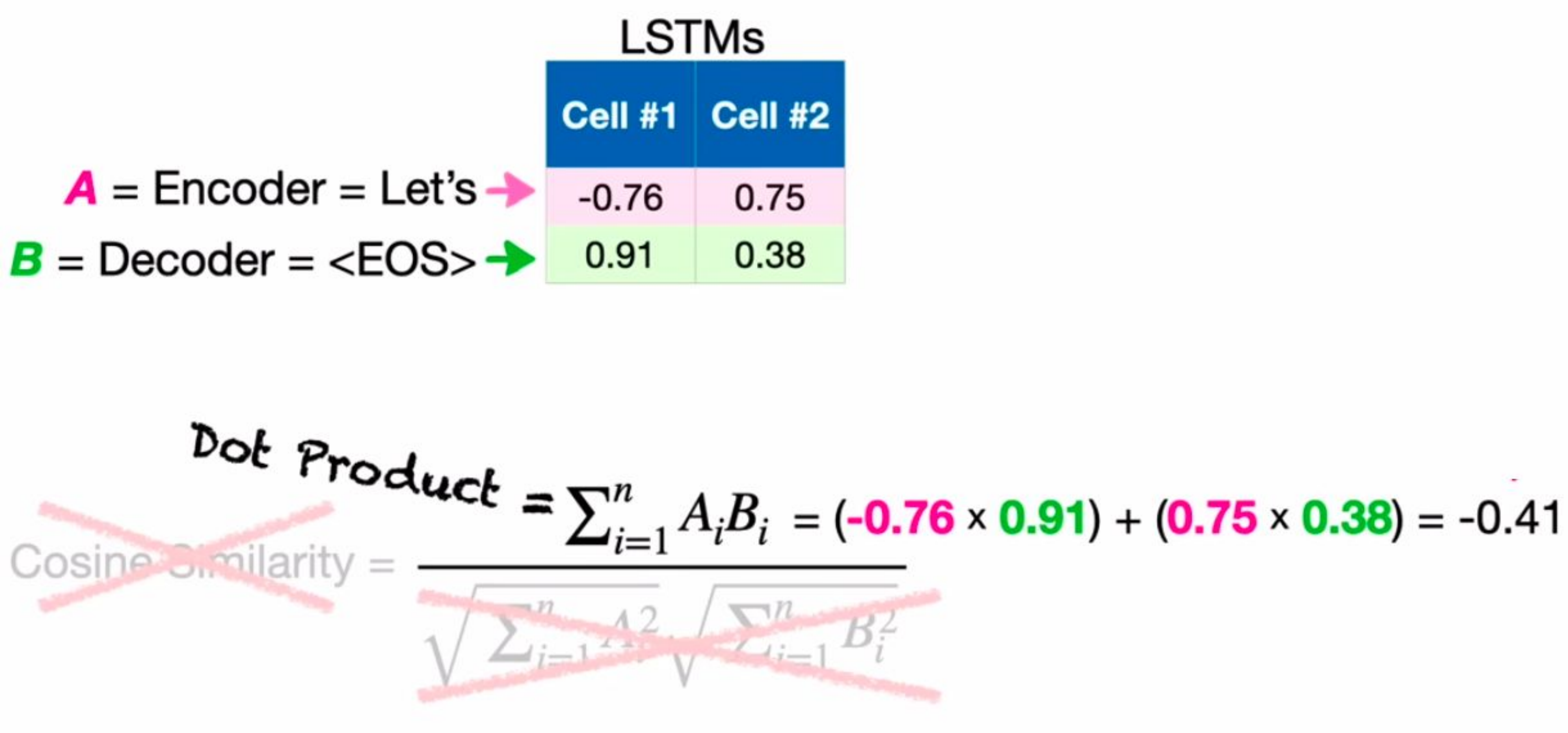


Attention

Seq2Seq Models with Attention

Encoder-Decoder with Attention

- A more common way to calculate the similarity for Attention is just calculate the numerator.
  - The dot product.
    - We do not need to scale the magnitude between -1 and 1.
      - Large positive numbers mean vectors are more similar than small positive numbers.
      - Large negative numbers mean vectors are more completely backwards than small negative numbers.

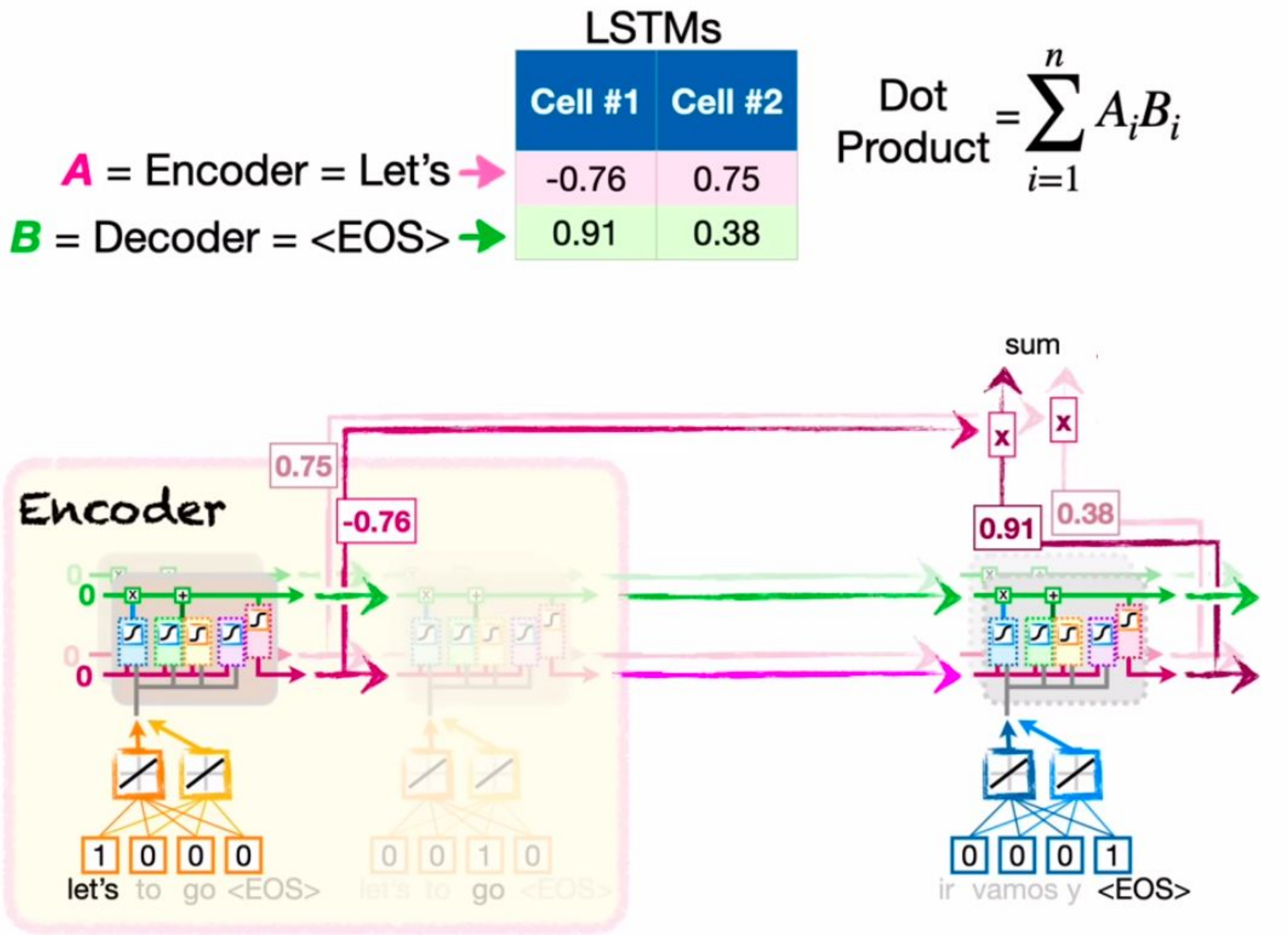


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Other advantage
  - Dot product is easier to implement in a network.

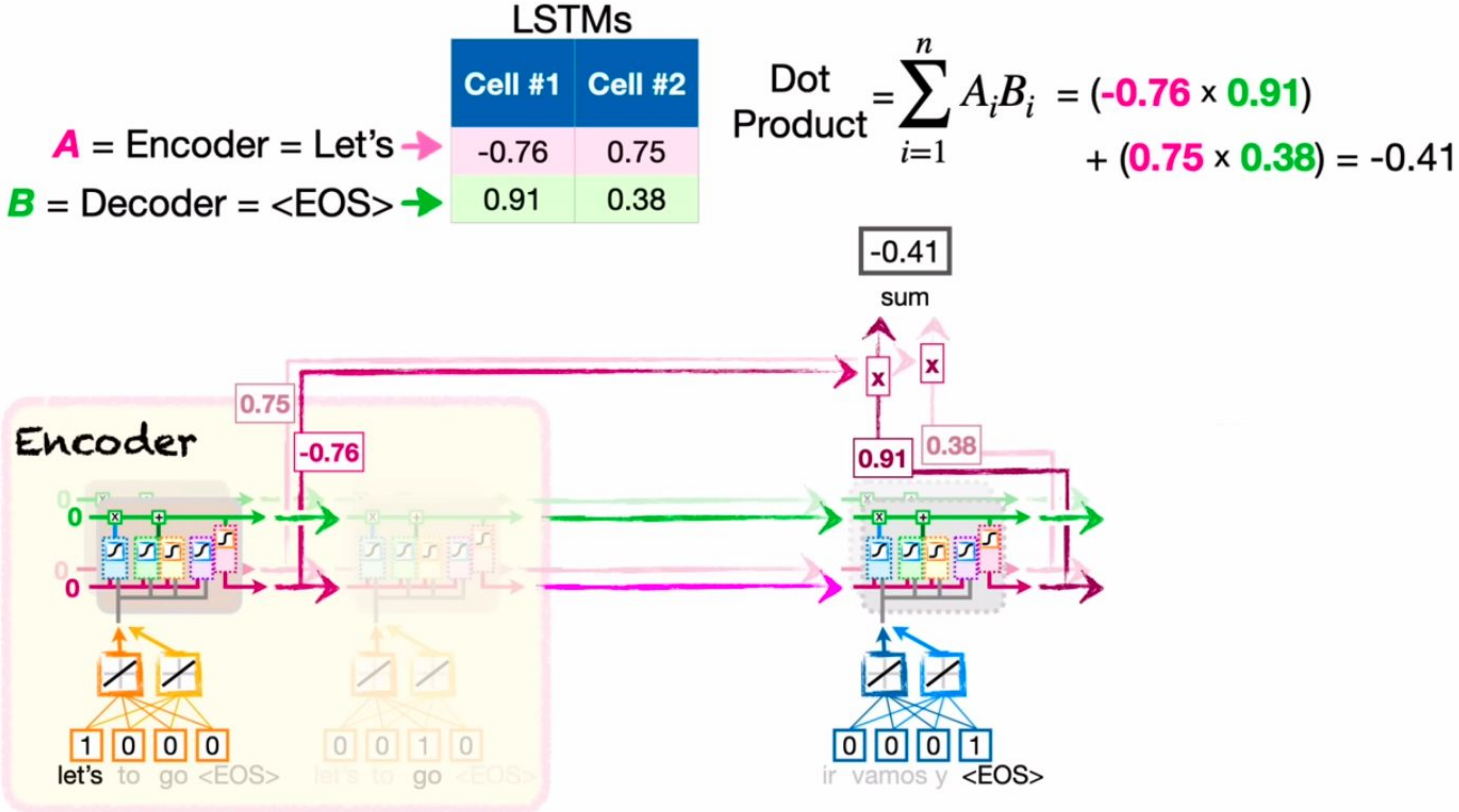


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Other advantage
  - Dot product is easier to implement in a network.

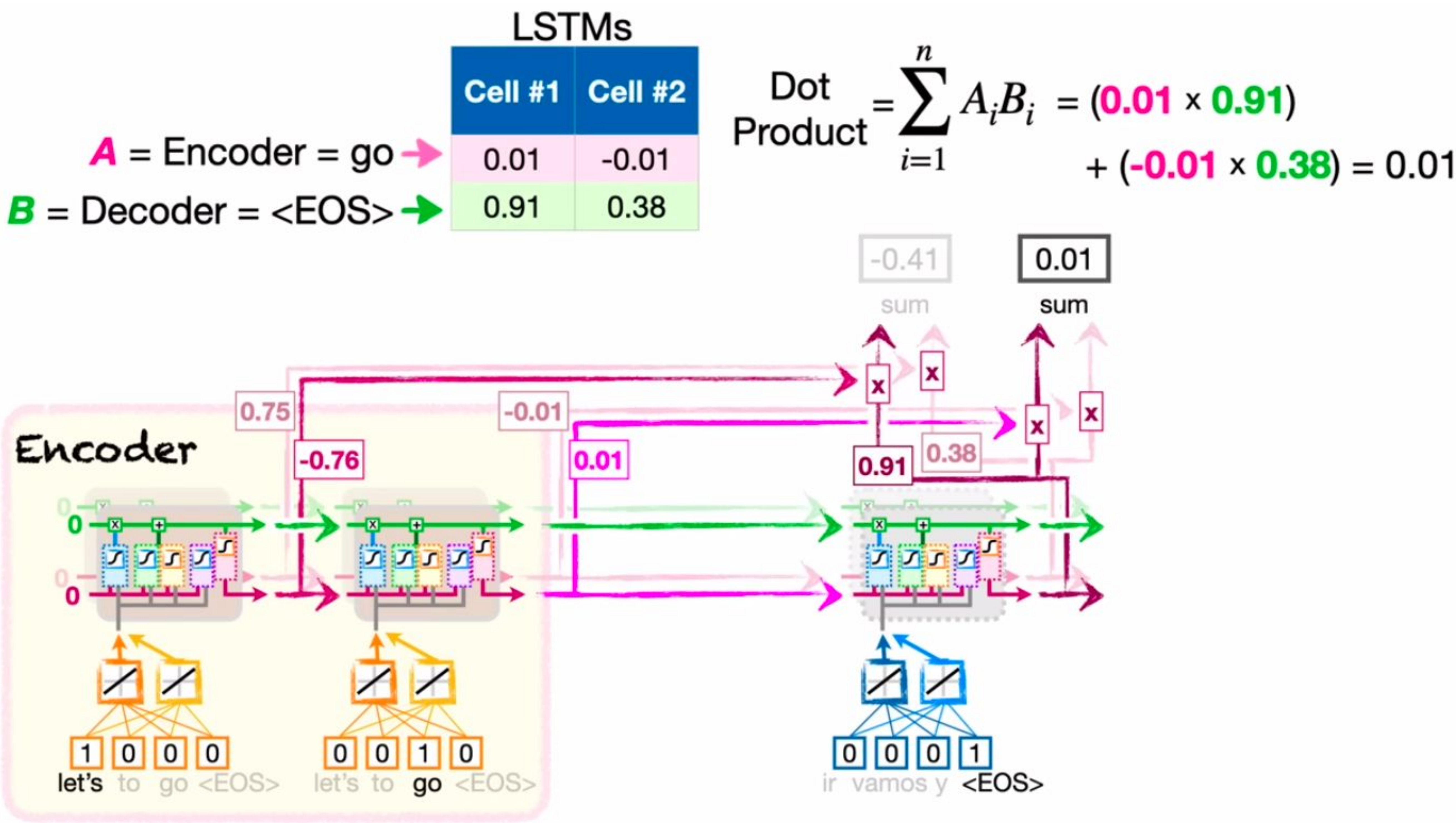


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Likewise, we can compute the similarity score between the second input of the encoder (“go”) with the first output of the decoder (“<EOS>”).

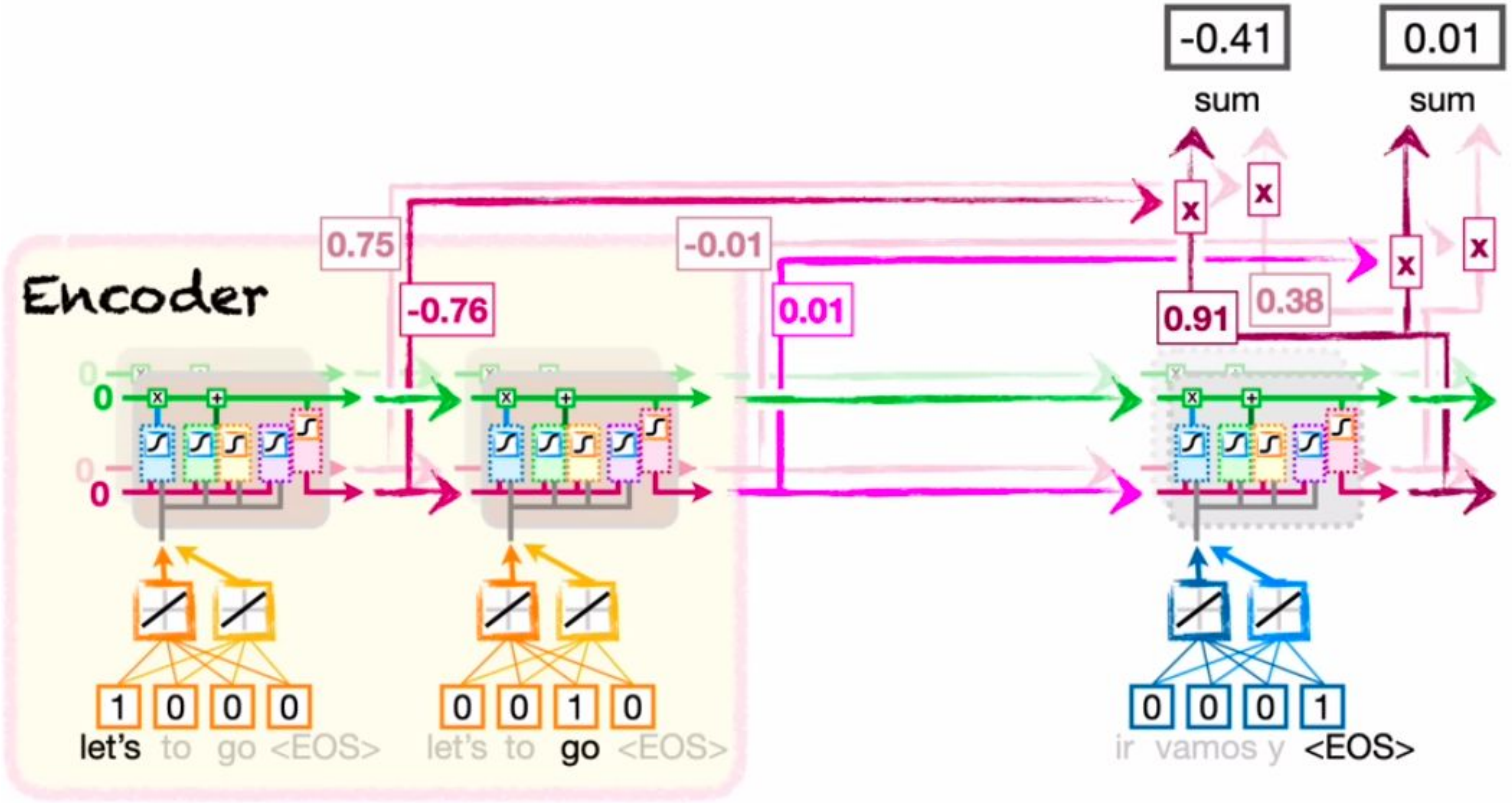


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Now we have similarity scores for both input words (“let’s” and “go”) …
    - … relative to the output (“<EOS>”) in the decoder.

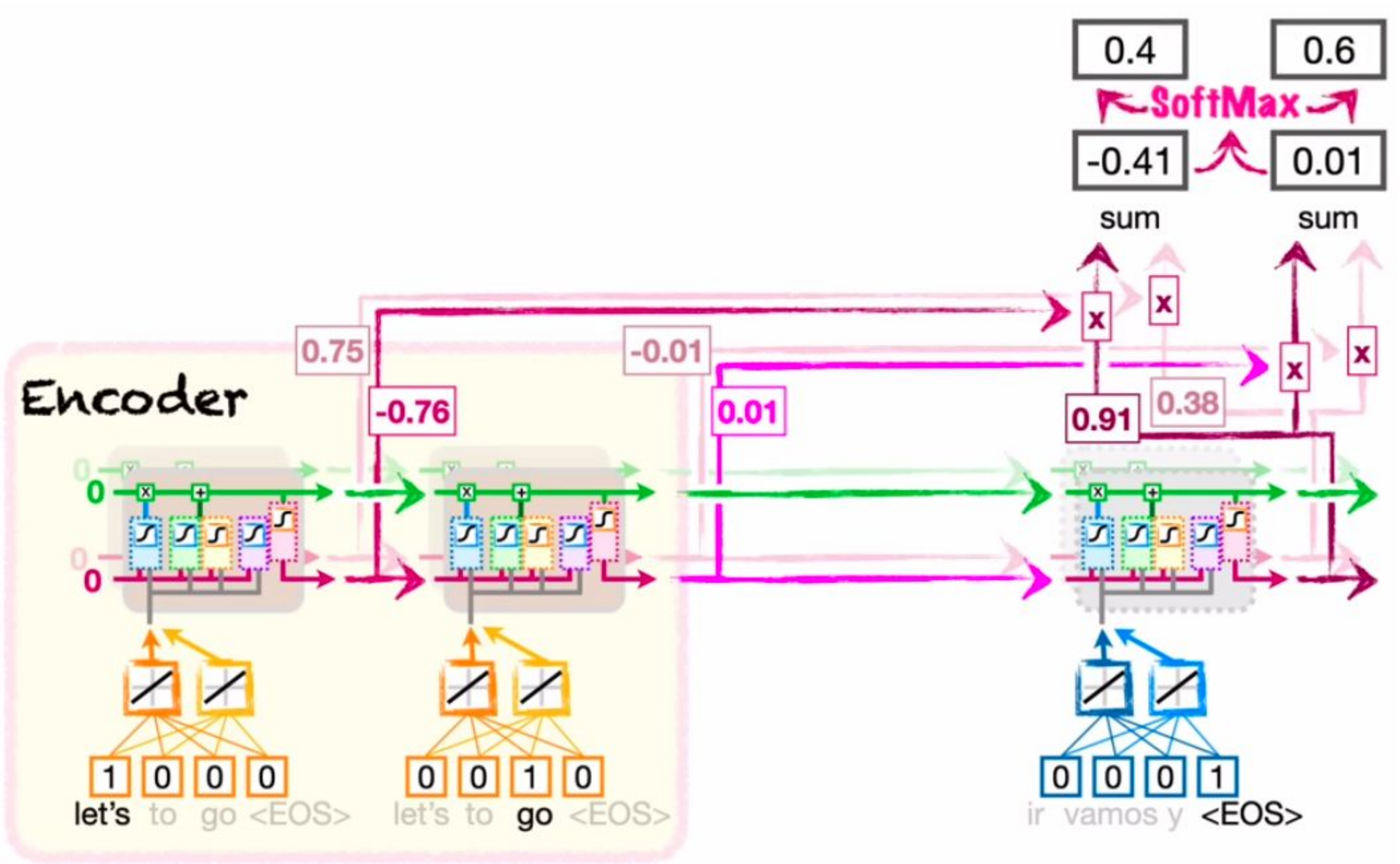


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- How can we use these similarity scores?
  - Intuitively, the similarity score reveals how each pair of words are related to each other.
    - That is, we want “go” to have more influence on the first words that comes out of the decoder.

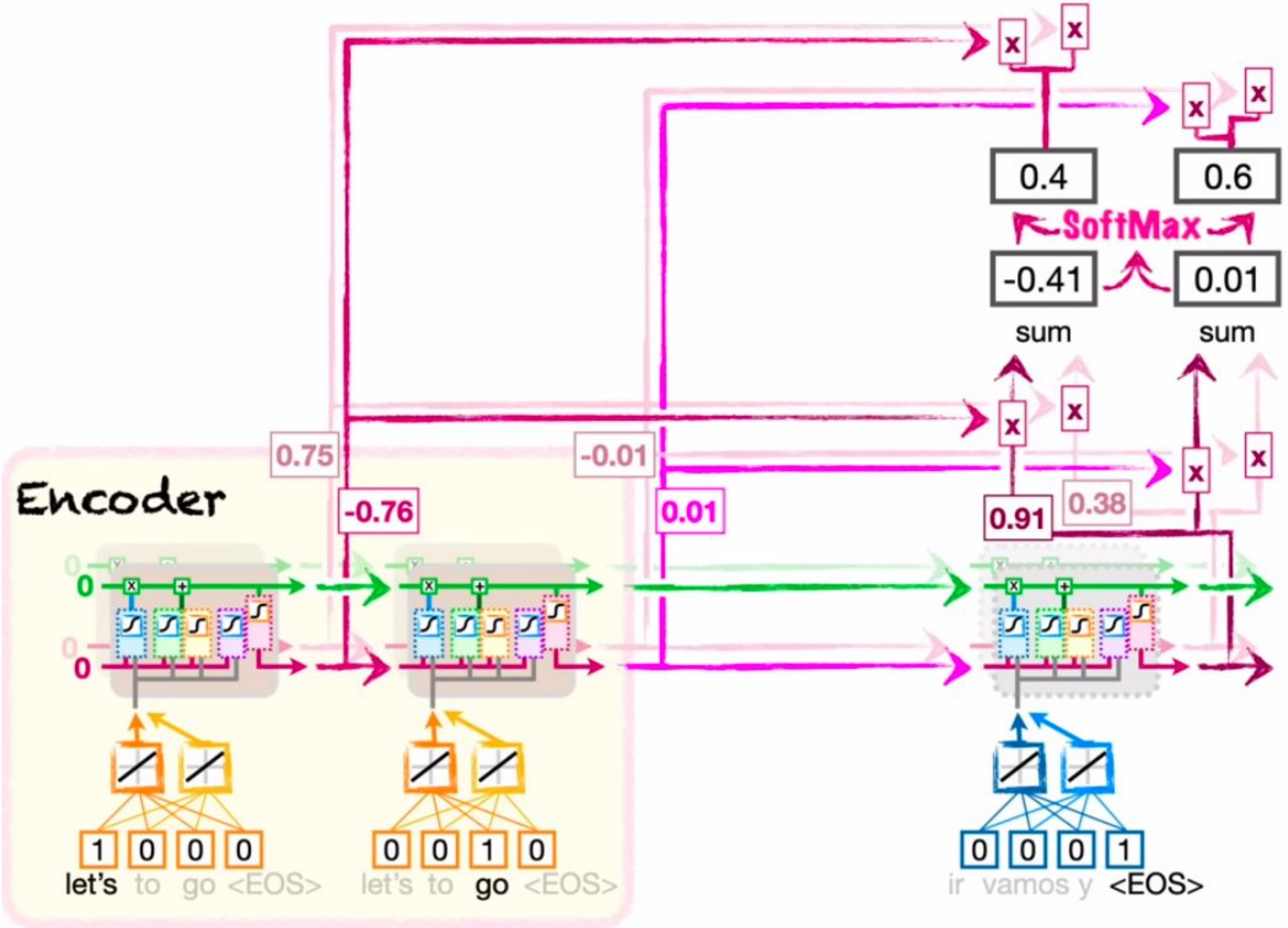


Attention

Seq2Seq Models with Attention

Encoder-Decoder with Attention

- Softmax gives us how much to use from each word.
  - "Let's" accounts for 0.4,
  - "go" accounts for 0.6,

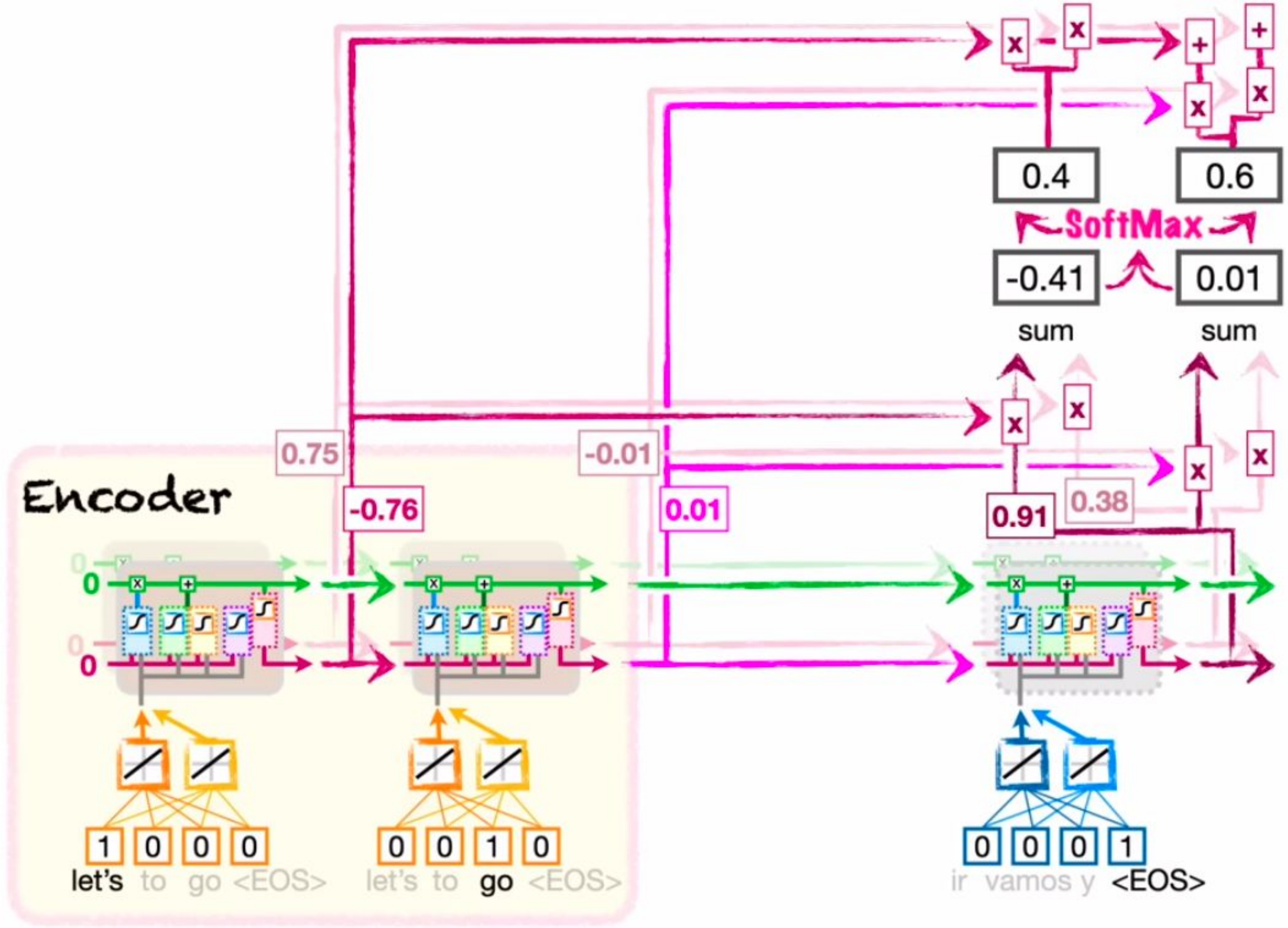


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Lastly, we add the scaled values together.
    - These sums, which combine the separate encodings for both input words, relative to the output word, are the Attention values for “<EOS>”.

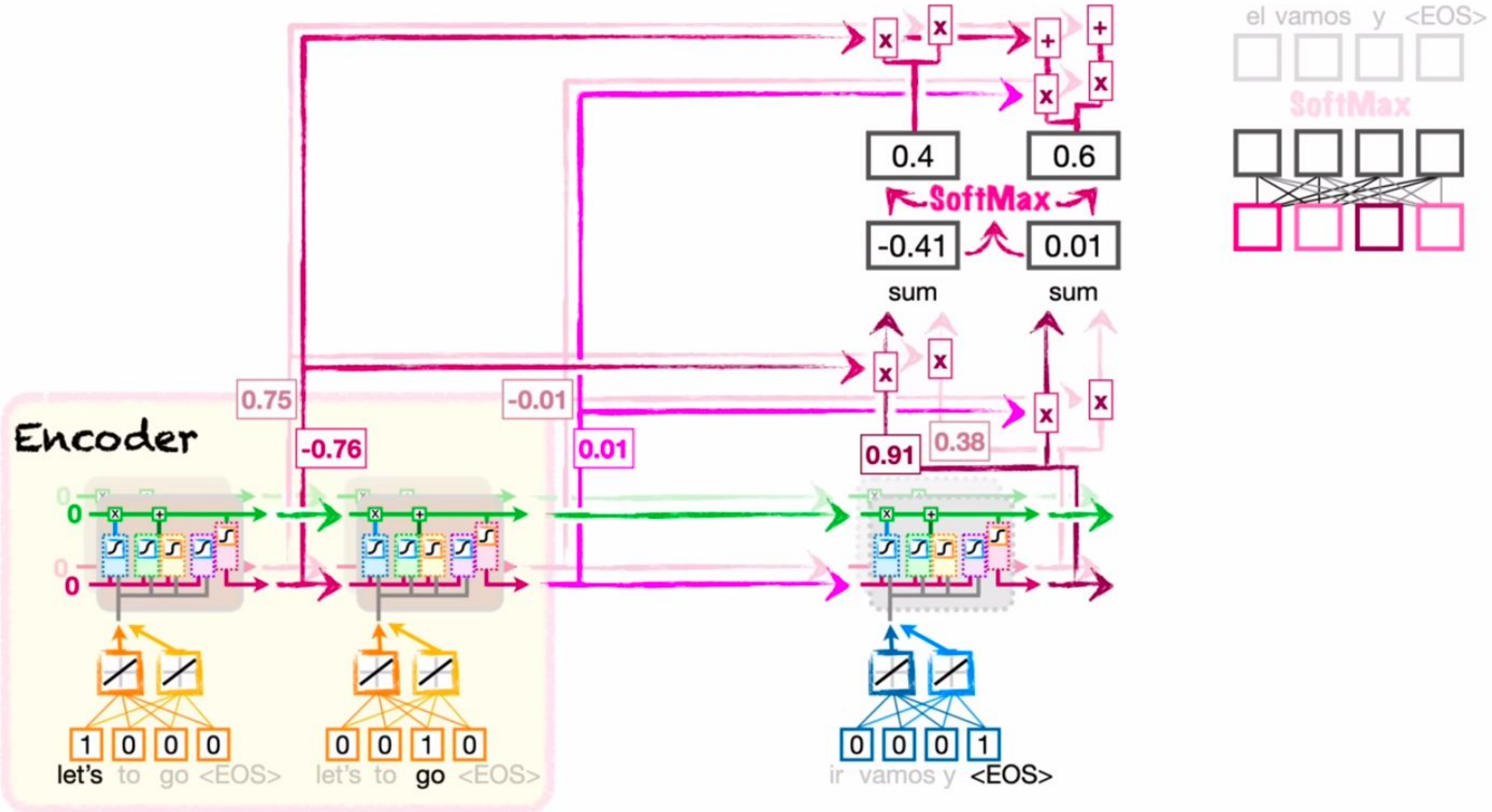


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Now, all we need to determine the first output word is to plug the Attention values into a fully connected layers.

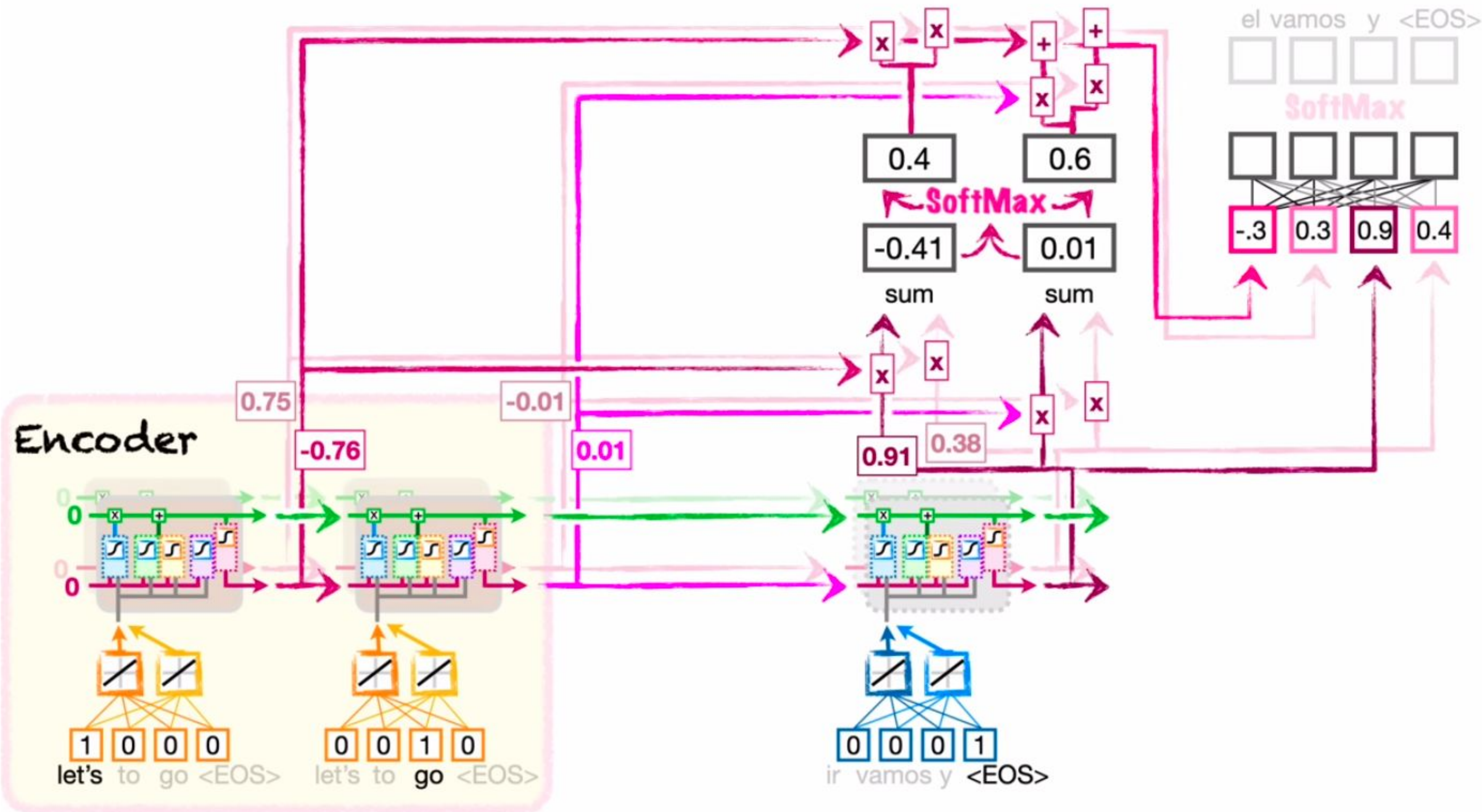


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- Now, all we need to determine the first output word is to plug the Attention values into a fully connected layers.
  - and also to plug the encodings for “<EOS>” into the same fully connected layer.

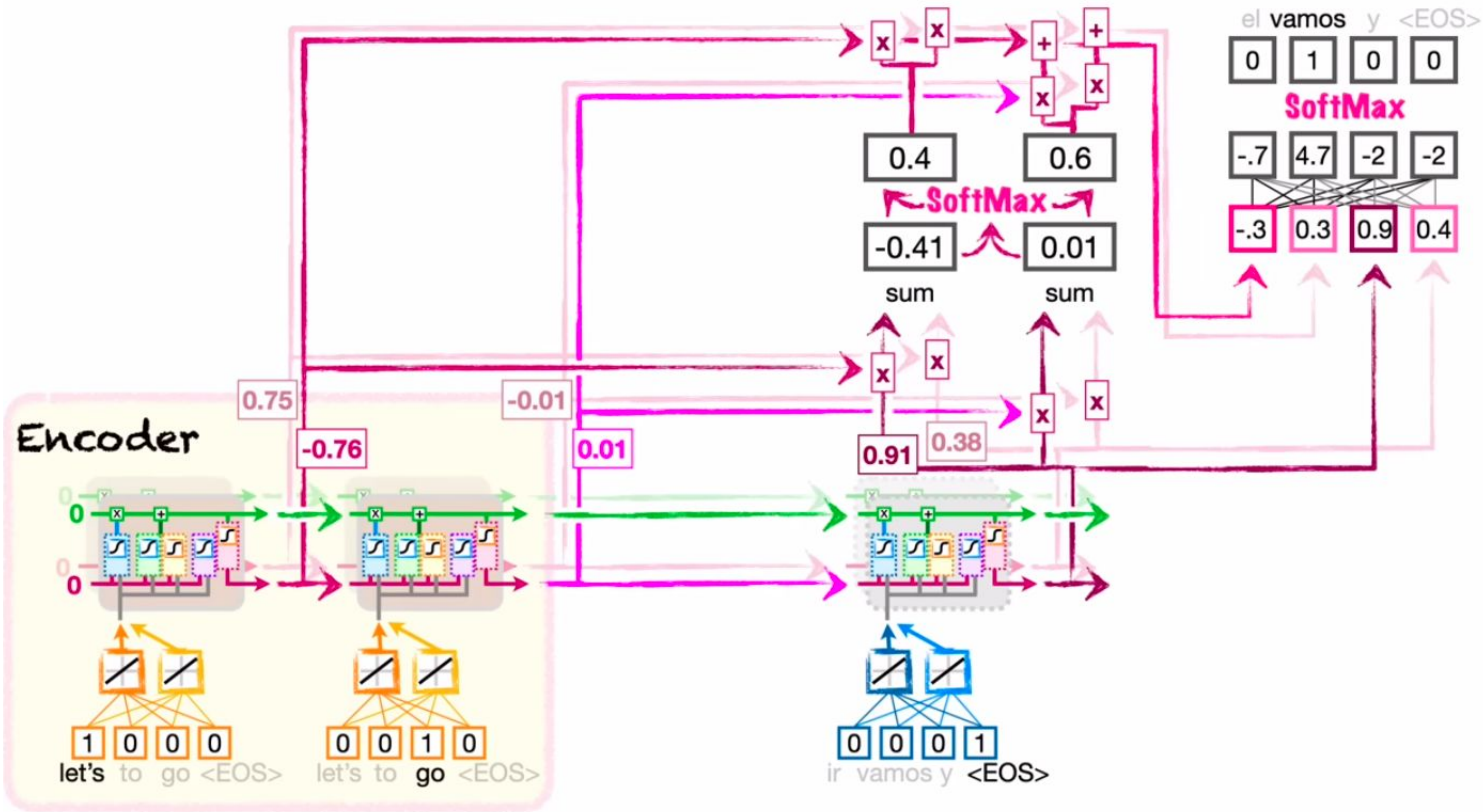


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- The same forward calculations! and softmax!

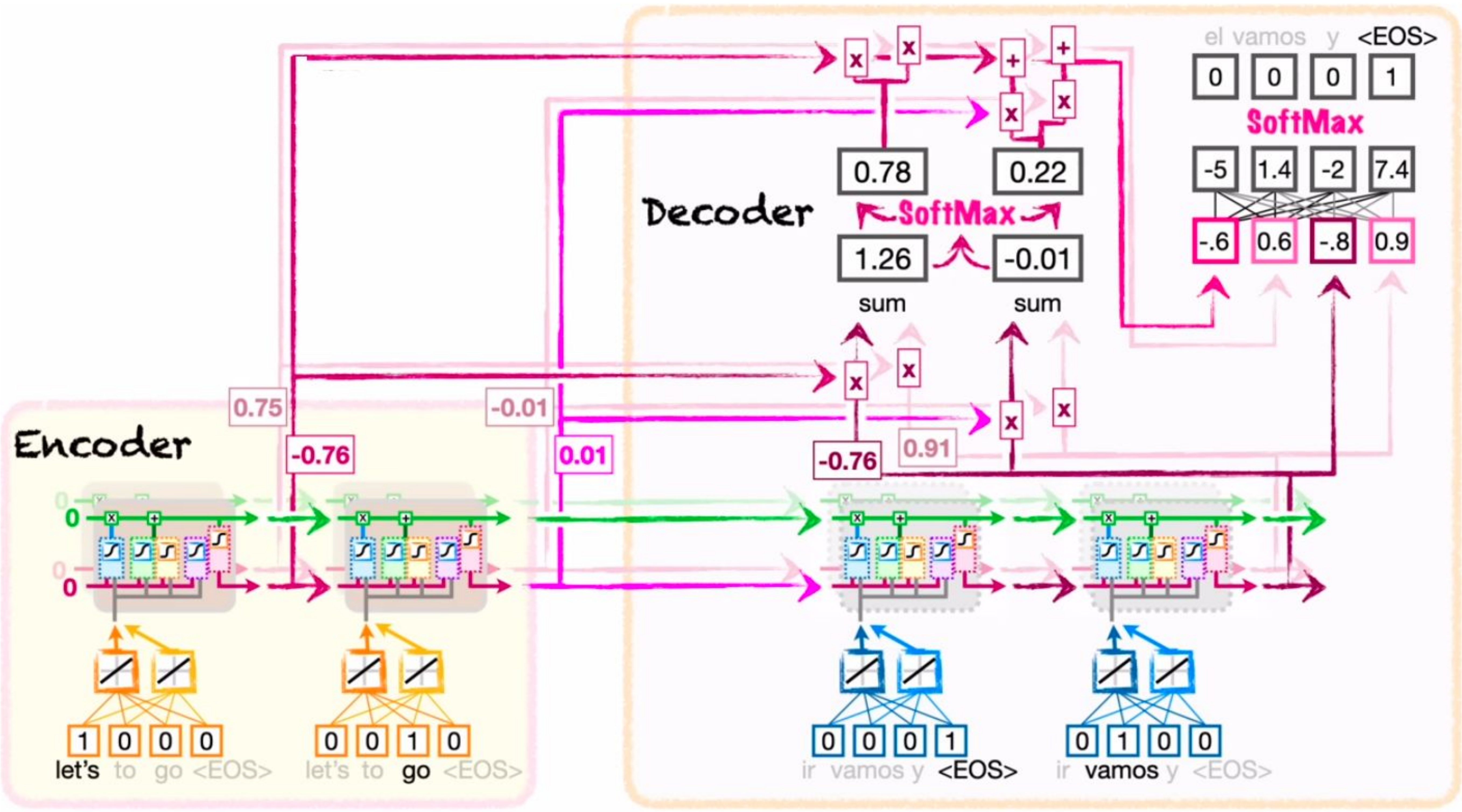


Attention

Seq2Seq Models with Attention

_ Encoder-Decoder with Attention

- The same thing is done for the second output word (“vamos” is the most likely one)



Transformers

Parallelism

_ There are issues with typical seq2seq NLP models

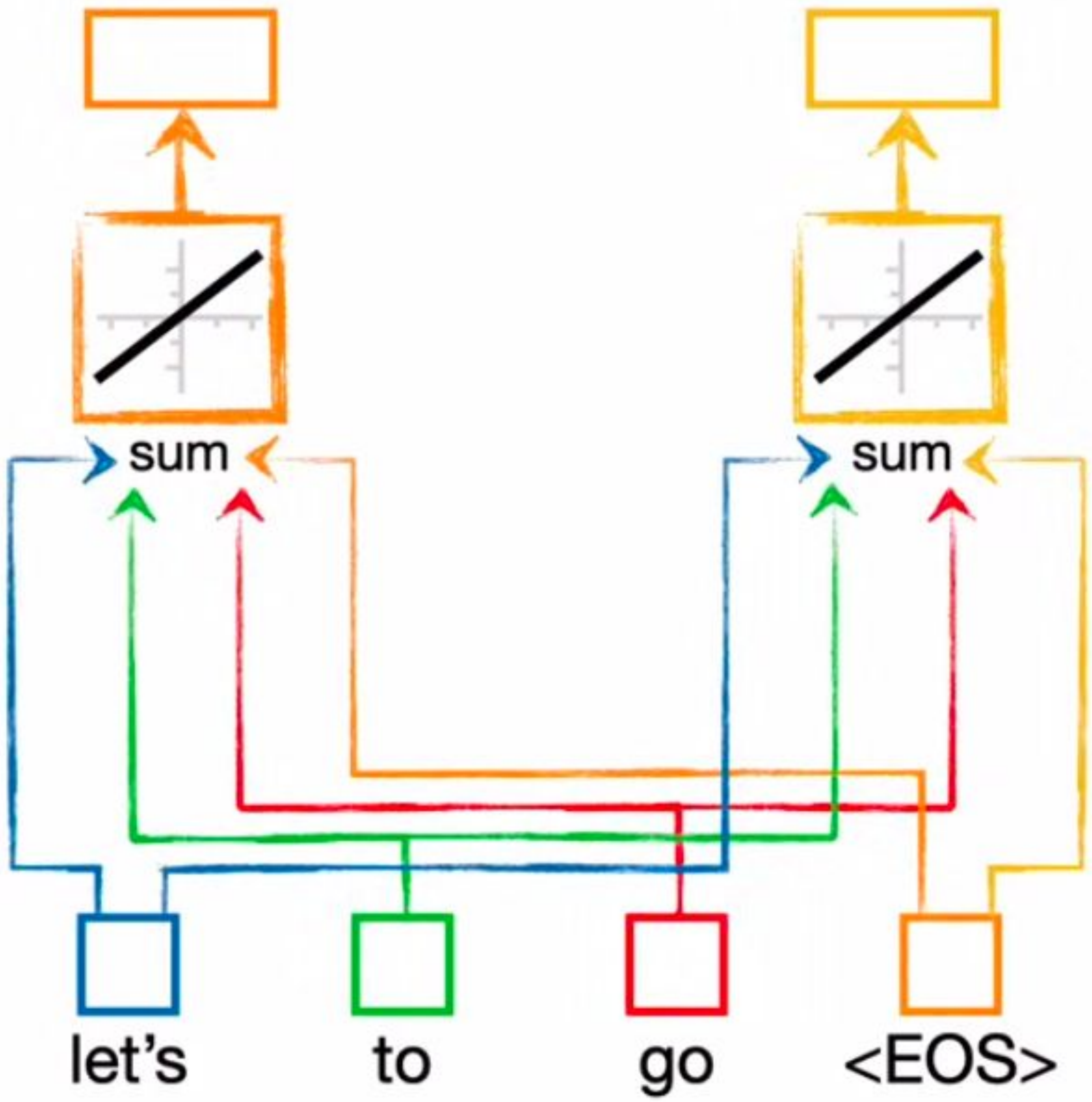
- One of the hardest is that the encoding step is done sequentially, and this is a serious limitation.
  - Parallelism is important because we want to produce models with as much data as we can.
    - RNNs work in a step-by-step way.
      - They are inherently sequential, and the internal hidden state is updated at each step.
    - Attention mechanisms are typically computed sequentially for each element in the output sequence, making it a step-by-step process.
  - Transformers allow for more parallel processing due to their self-attention mechanism, which enables the model to attend to all positions in the input sequence simultaneously.

Transformers

Learning word representations

_ With parallelism we may train models using much more data.

- Thus, in practice, we do not use pre-trained word embeddings in Transformers.
    - Instead, we produce word embeddings as we train the Transformer.
        - We have a embedding layer in which each token is linked to a number of activation functions.
            - The number of activation functions depends on the embedding dimension.

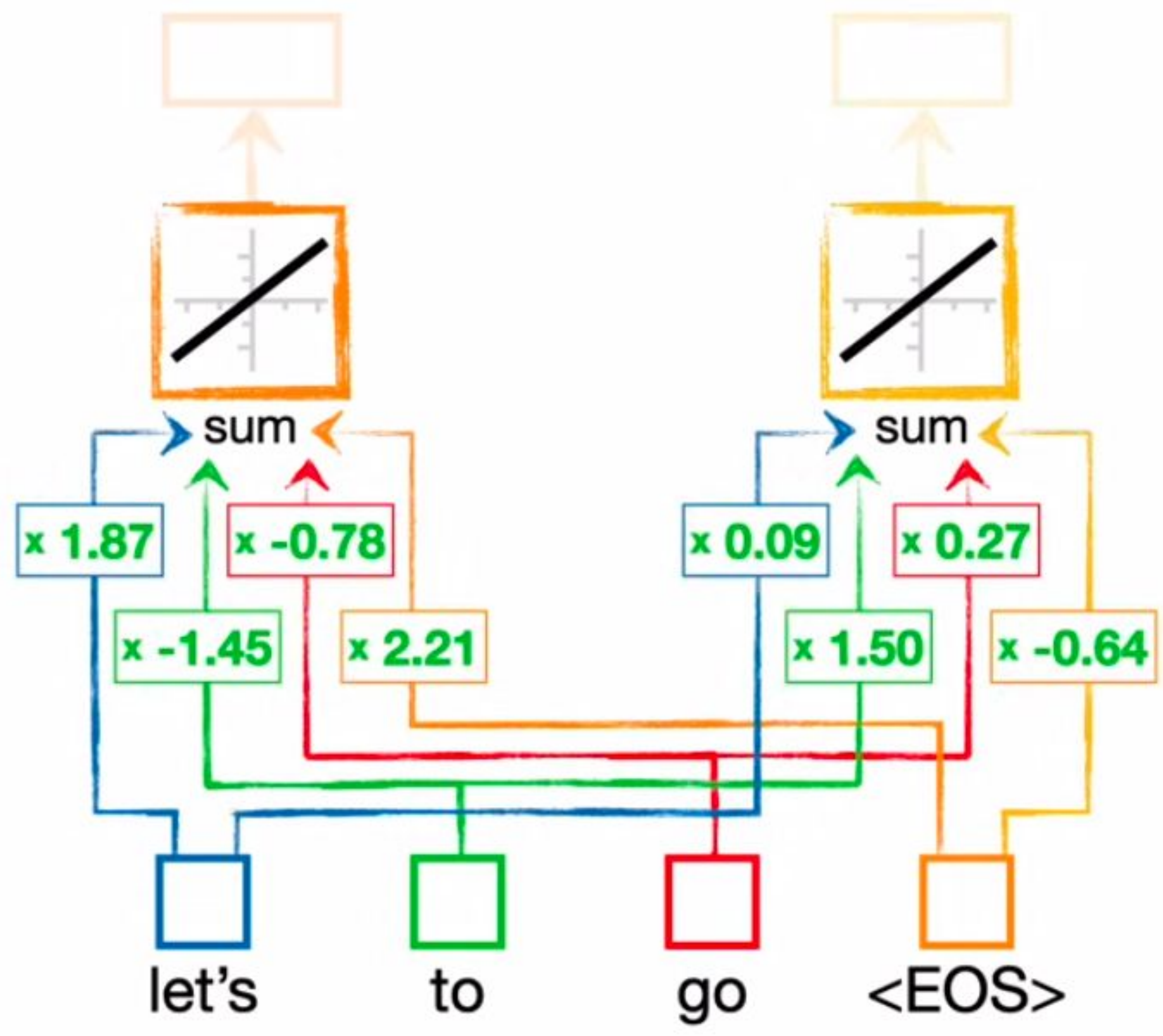


Transformers

Learning word representations

_ With parallelism we may train models using much more data.

- Thus, in practice, we do not use pre-trained word embeddings in Transformers.
  - Instead, we produce word embeddings as we train the Transformer.
    - We have a embedding layer in which each token is linked to a number of activation functions.
      - The number of activation functions depends on the embedding dimension.
      - Weights multiply the input for each token.

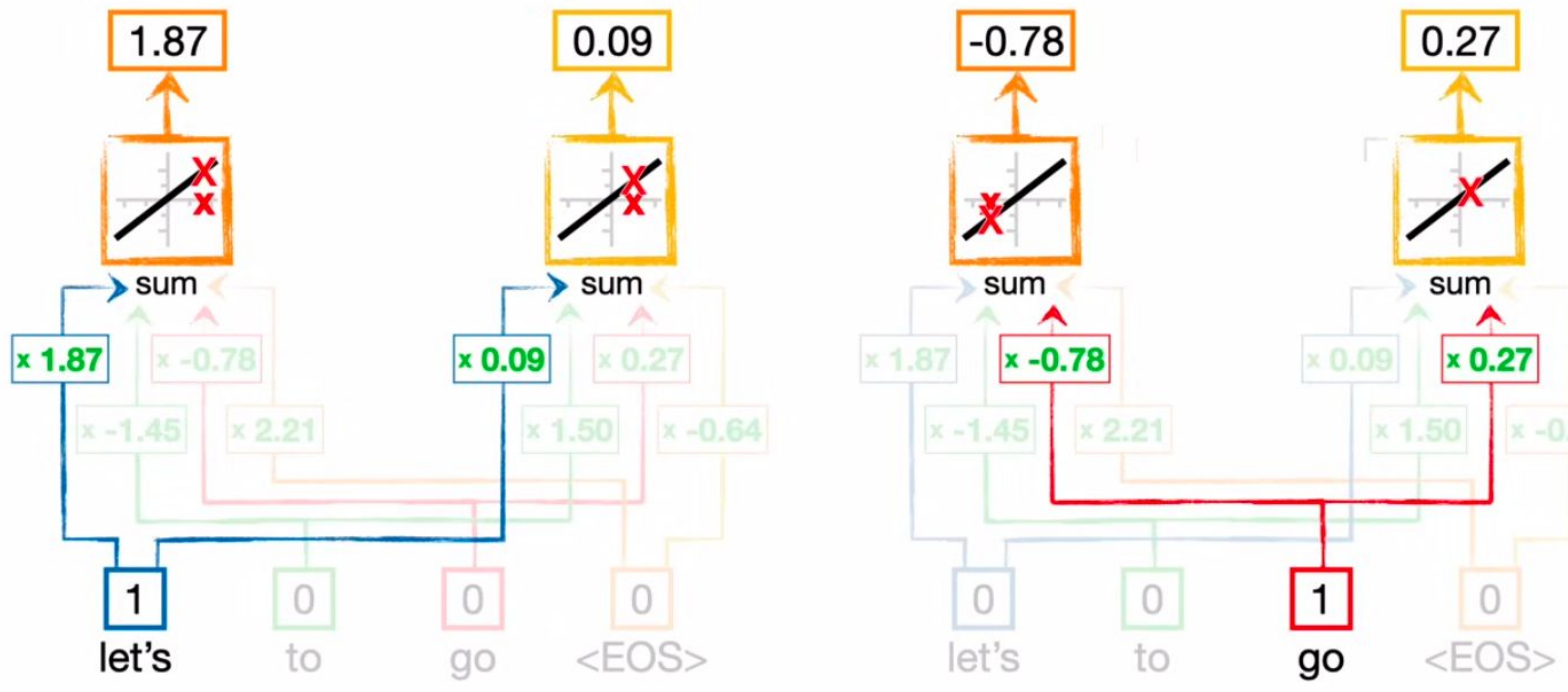


Transformers

Learning word representations

With parallelism we may train models using much more data.

- Thus, in practice, we do not use pre-trained word embeddings in Transformers.
  - Instead, we produce word embeddings as we train the Transformer.
    - We have a embedding layer in which each token is linked to a number of activation functions.
      - The number of activation functions depends on the embedding dimension.
      - Weights multiply the input for each token.
        - The same weights are used no matter the token being processed.
        - Updated via backpropagation.

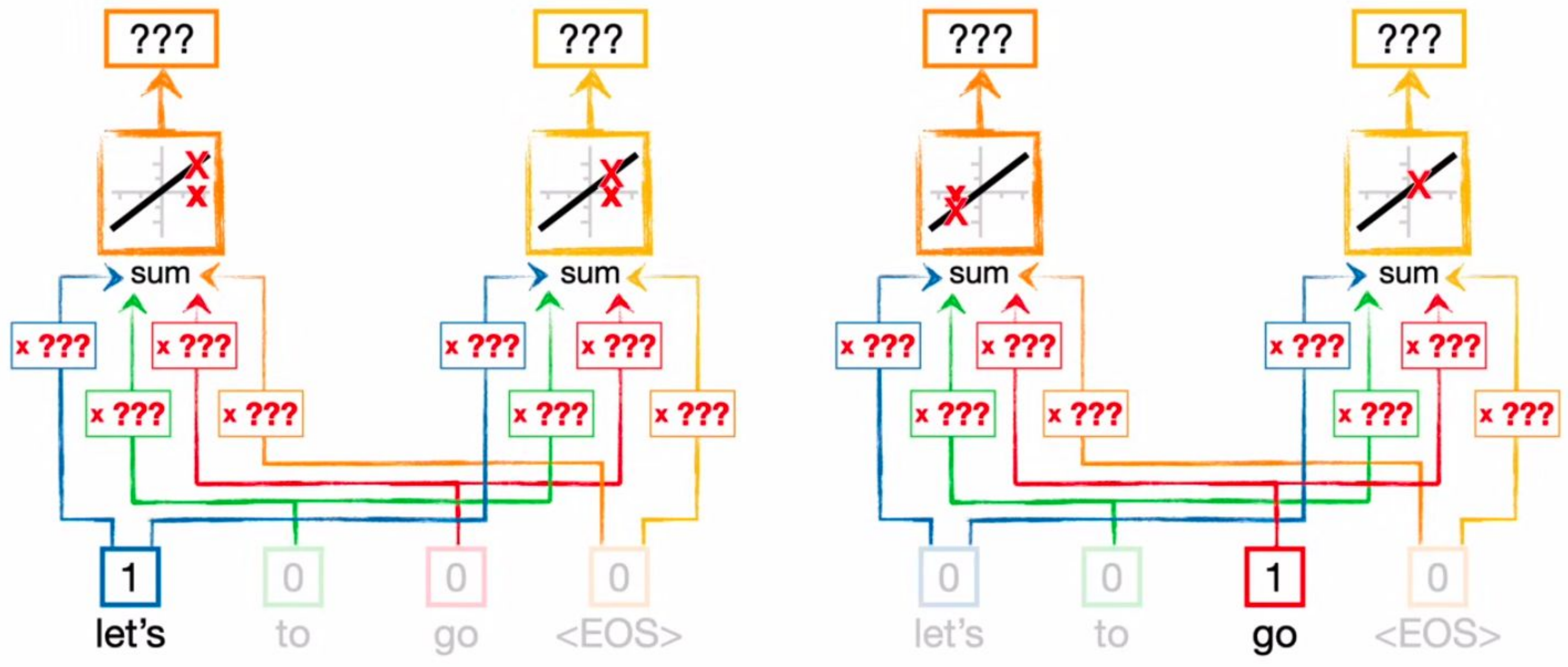


Transformers

Learning word representations

_ With parallelism we may train models using much more data.

- Thus, in practice, we do not use pre-trained word embeddings in Transformers.
  - Instead, we produce word embeddings as we train the Transformer.
    - We have a embedding layer in which each token is linked to a number of activation functions.
      - The number of activation functions depends on the embedding dimension.
      - Weights multiply the input for each token.
        - The same weights are used no matter the token being processed.
        - Updated via backpropagation.

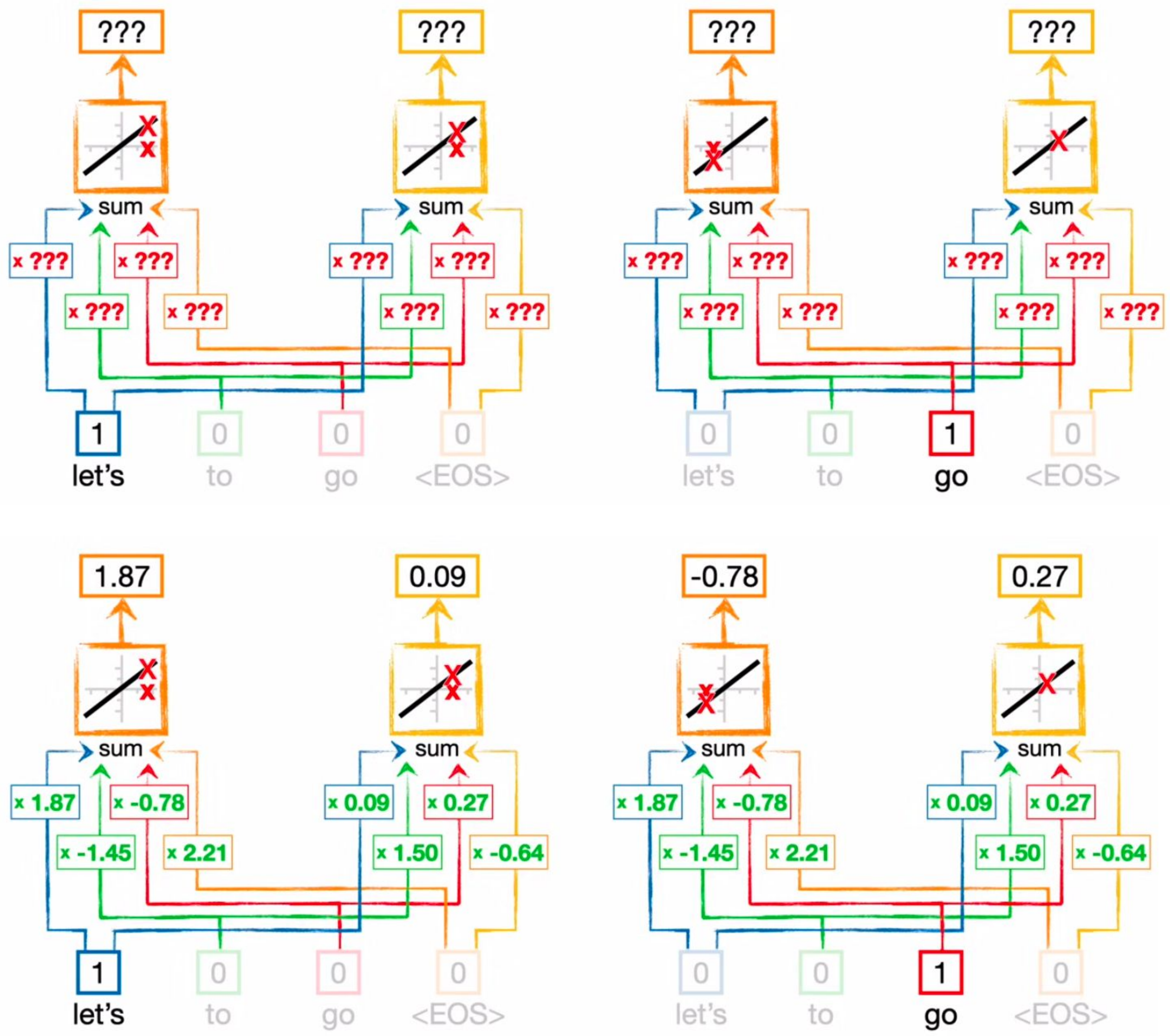


Transformers

Learning word representations

With parallelism we may train models using much more data.

- Thus, in practice, we do not use pre-trained word embeddings in Transformers.
  - Instead, we produce word embeddings as we train the Transformer.
    - We have a embedding layer in which each token is linked to a number of activation functions.
      - The number of activation functions depends on the embedding dimension.
      - Weights multiply the input for each token.
        - The same weights are used no matter the token being processed.
        - Updated via backpropagation.



Transformers

Word ordering

_ Now, how to represent word order?

- The ordering of the words in the sentence is crucial for its meaning.
    - We may have two sentences with the exact same words, but with completely different meanings.
        - How to take order into account without hurting parallelism?
            - Positional encoding

Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

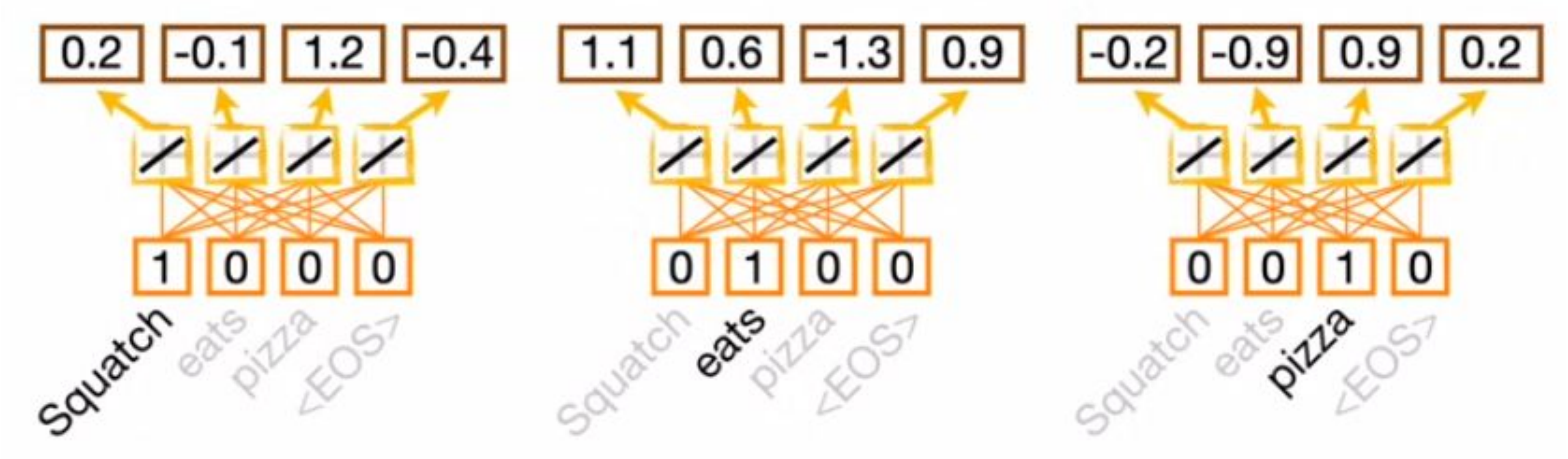
- Instead of coding the ordering using the architecture …
  - … Transformers encode the ordering directly in the data (i.e., representation)
    - Positional encoding is crucial to parallelism.

Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

- Instead of coding the ordering using the architecture …
  - … Transformers encode the ordering directly in the data (i.e., representation)
    - Positional encoding is crucial to parallelism.
      - example from StatQuast

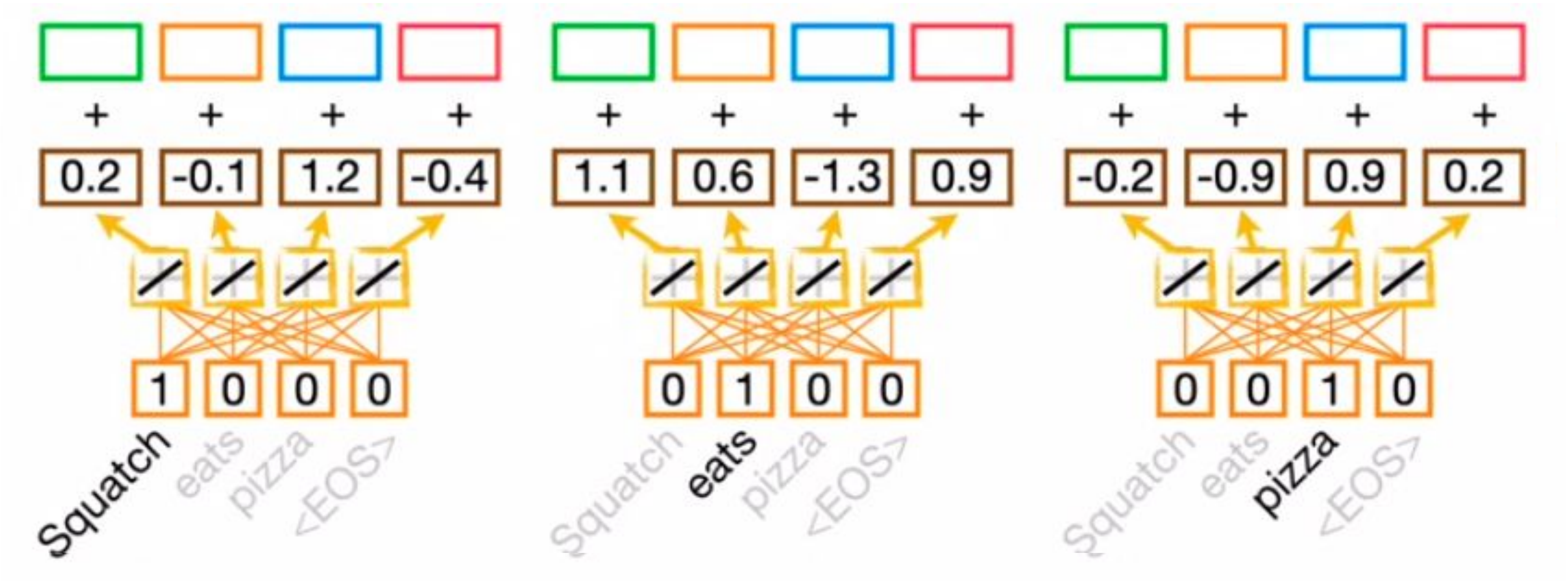


Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

- Instead of coding the ordering using the architecture ...
  - ... Transformers encode the ordering directly in the data (i.e., representation)
    - Positional encoding is crucial to parallelism.
      - example from StatQuast

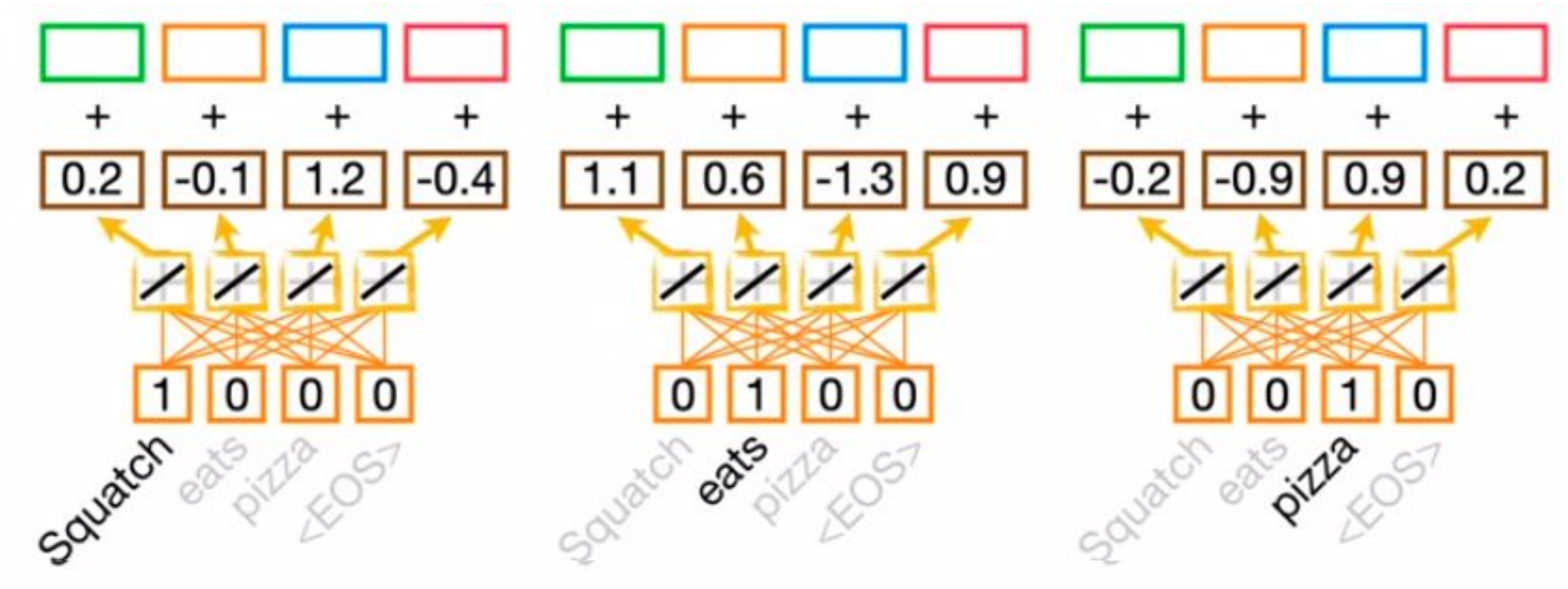


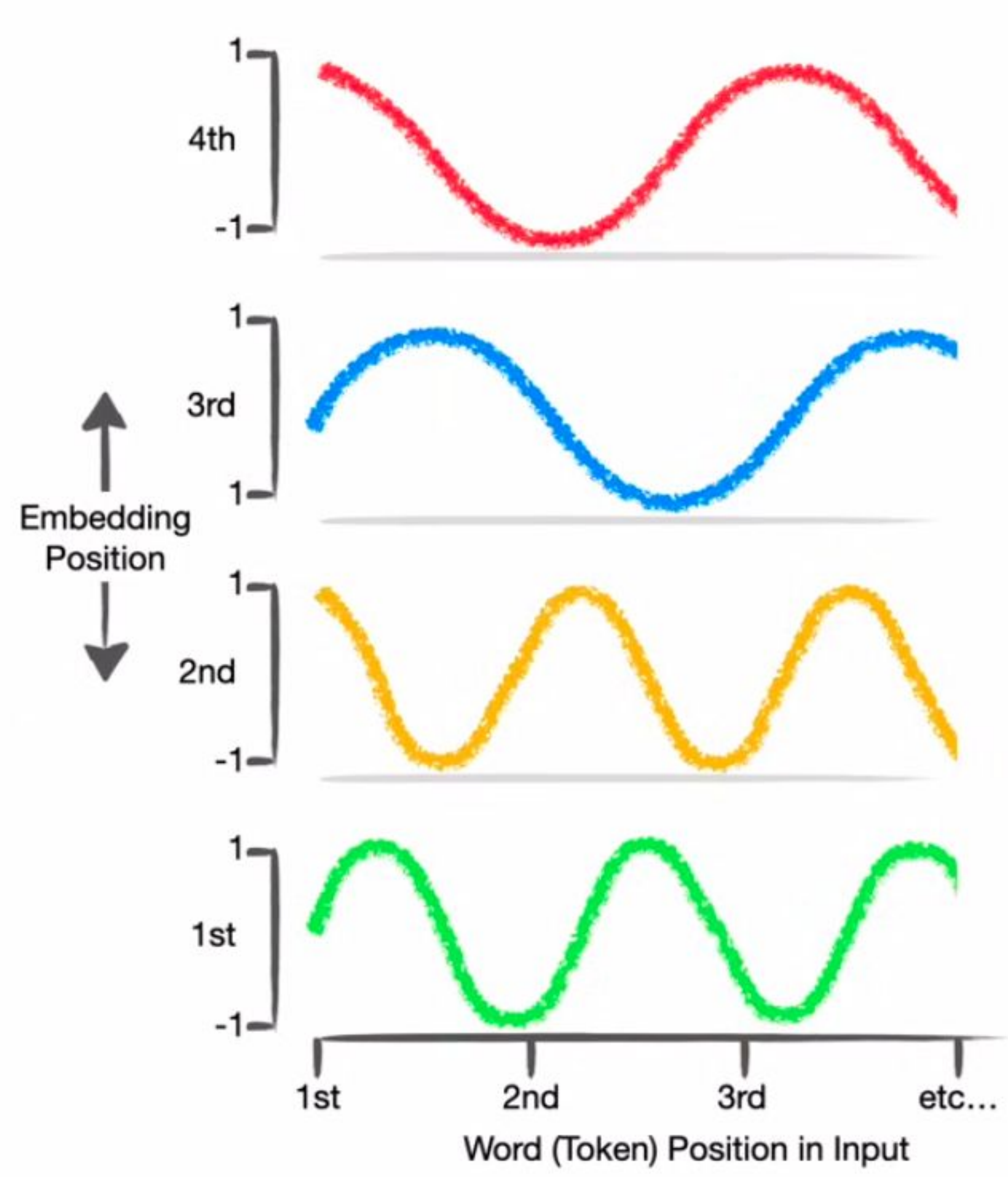
Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

- Instead of coding the ordering using the architecture …
  - … Transformers encode the ordering directly in the data (i.e., representation)
    - Positional encoding is crucial to parallelism.
      - example from StatQuast

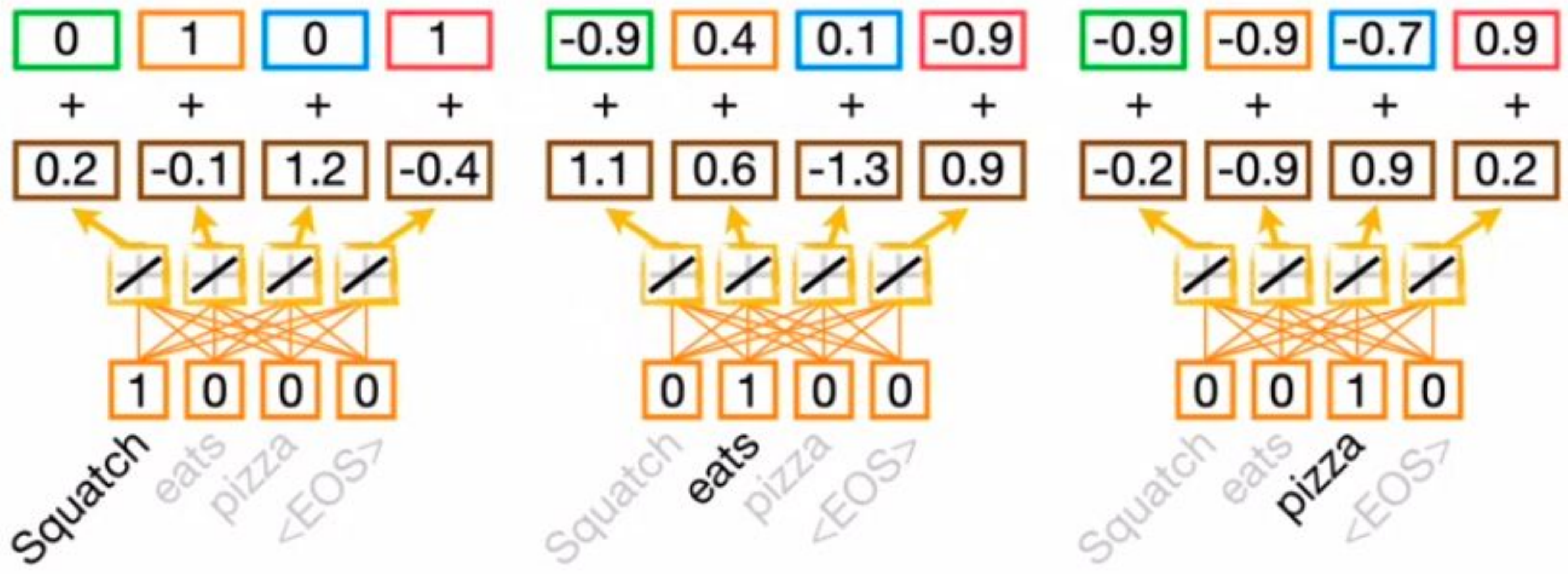


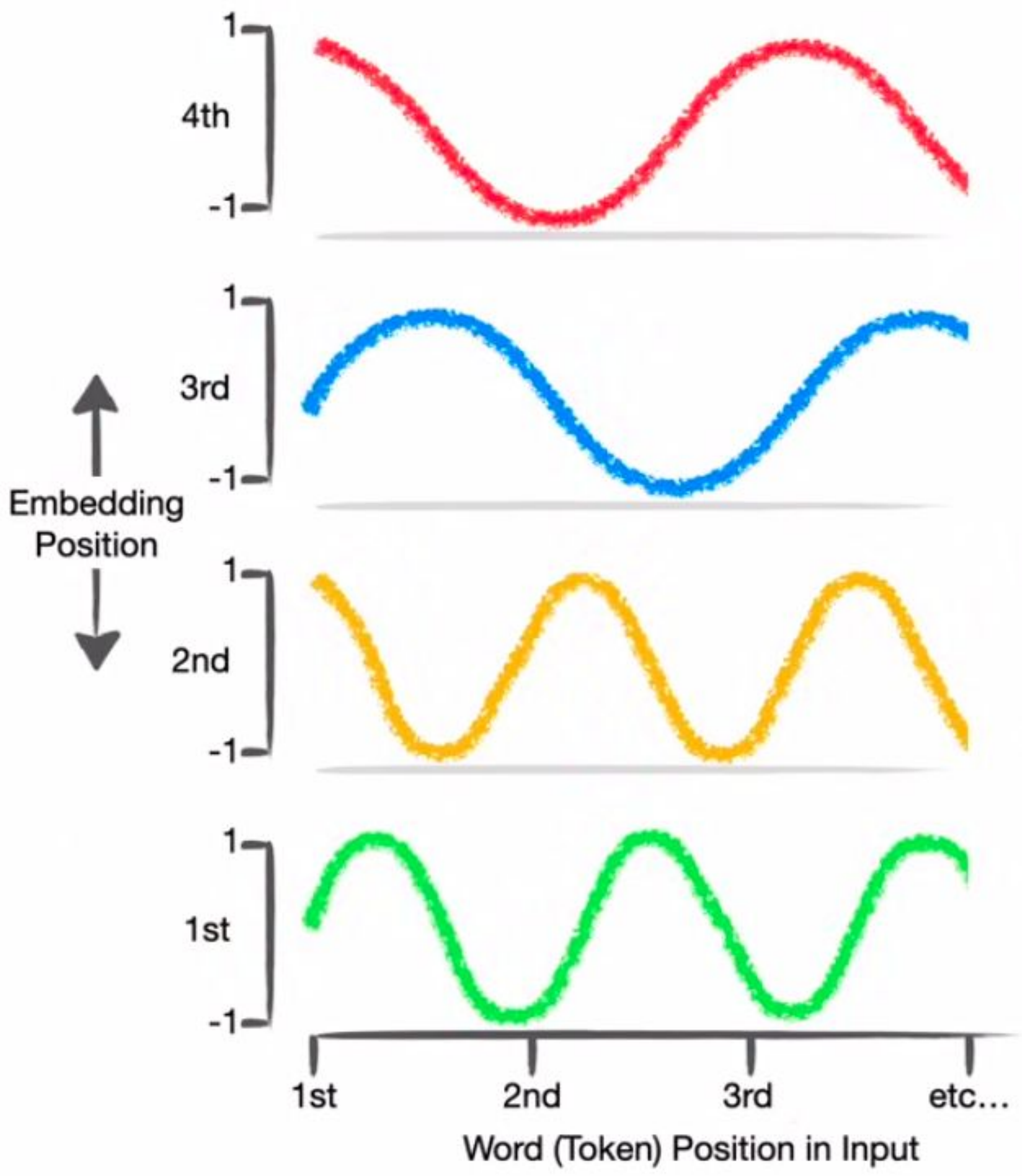


Transformers

Positional encoding

- This is the technique Transformers use to keep track of word order.





Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

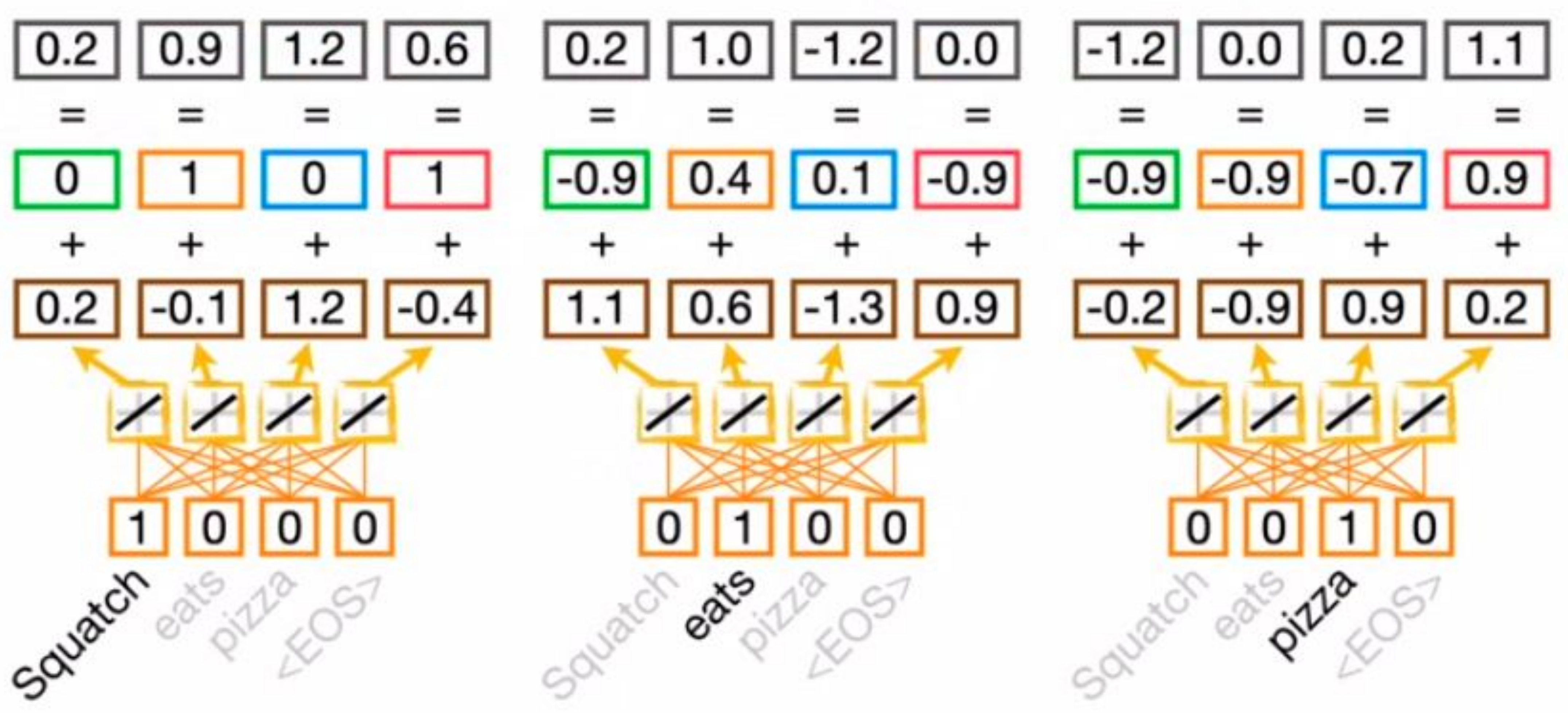
- Because sine and cosine are repetitive, it is possible that two words might get the same position (y-axis values).
  - But, with larger vocabularies and embeddings, these repeated values can be simply neglected.
    - And we have unique sequence of positional values for each token.

Transformers

Positional encoding

_ This is the technique Transformers use to keep track of word order.

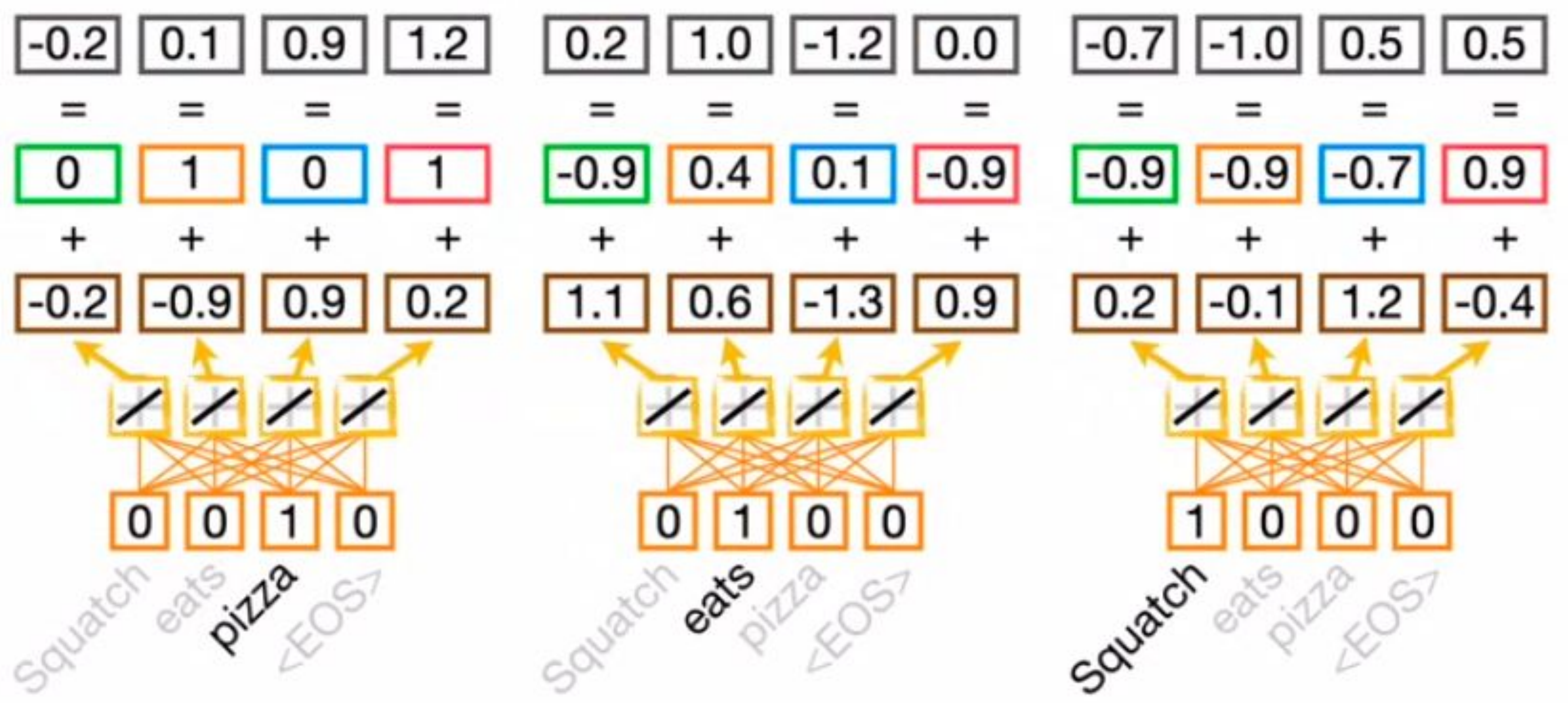
- Now, all we need is to sum the position values to the embedding values.
  - “Squatch eats pizza”



Transformers

Positional encoding

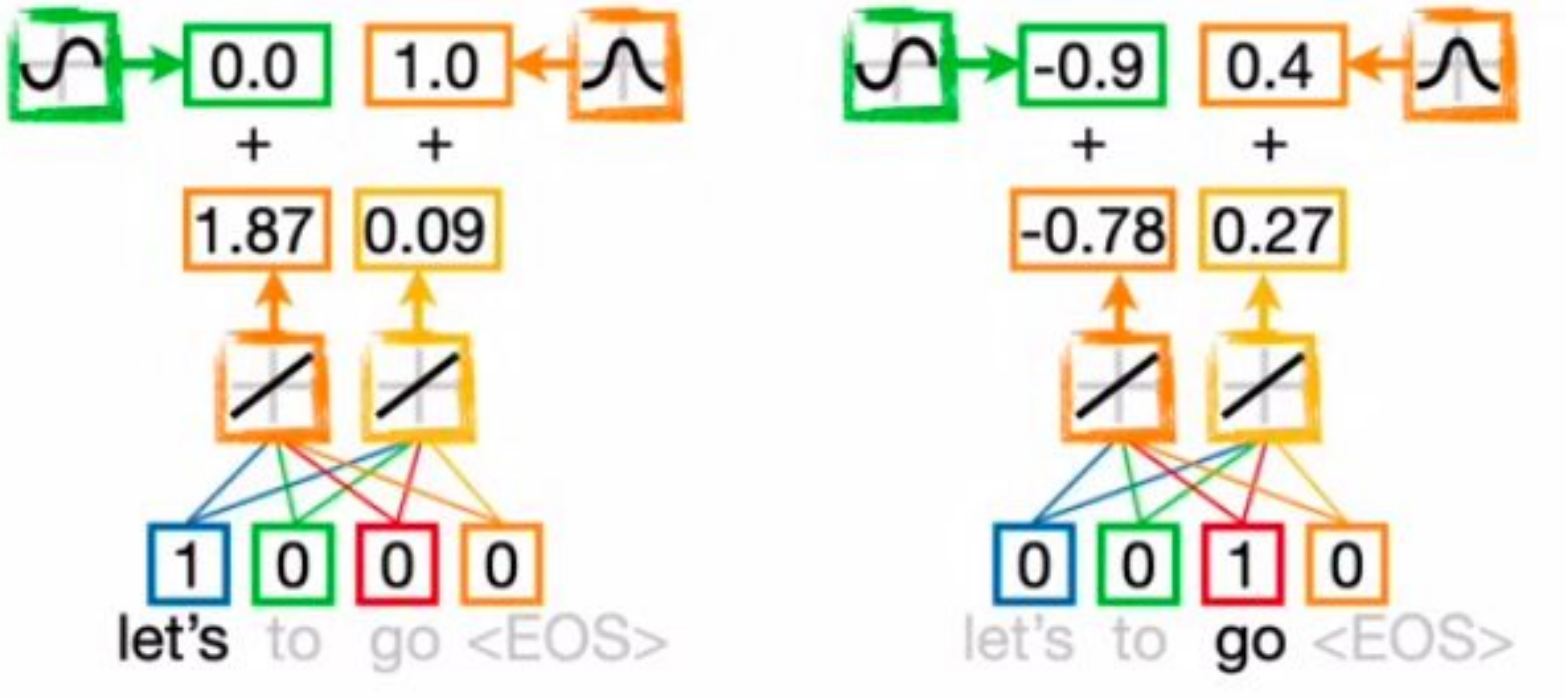
- This is the technique Transformers use to keep track of word order.
- Now, all we need is to sum the position values to the embedding values.
  - “Pizza eats Squatch”
    - Positional encoding allows Transformer to keep track of word order.



Transformers

Positional encoding

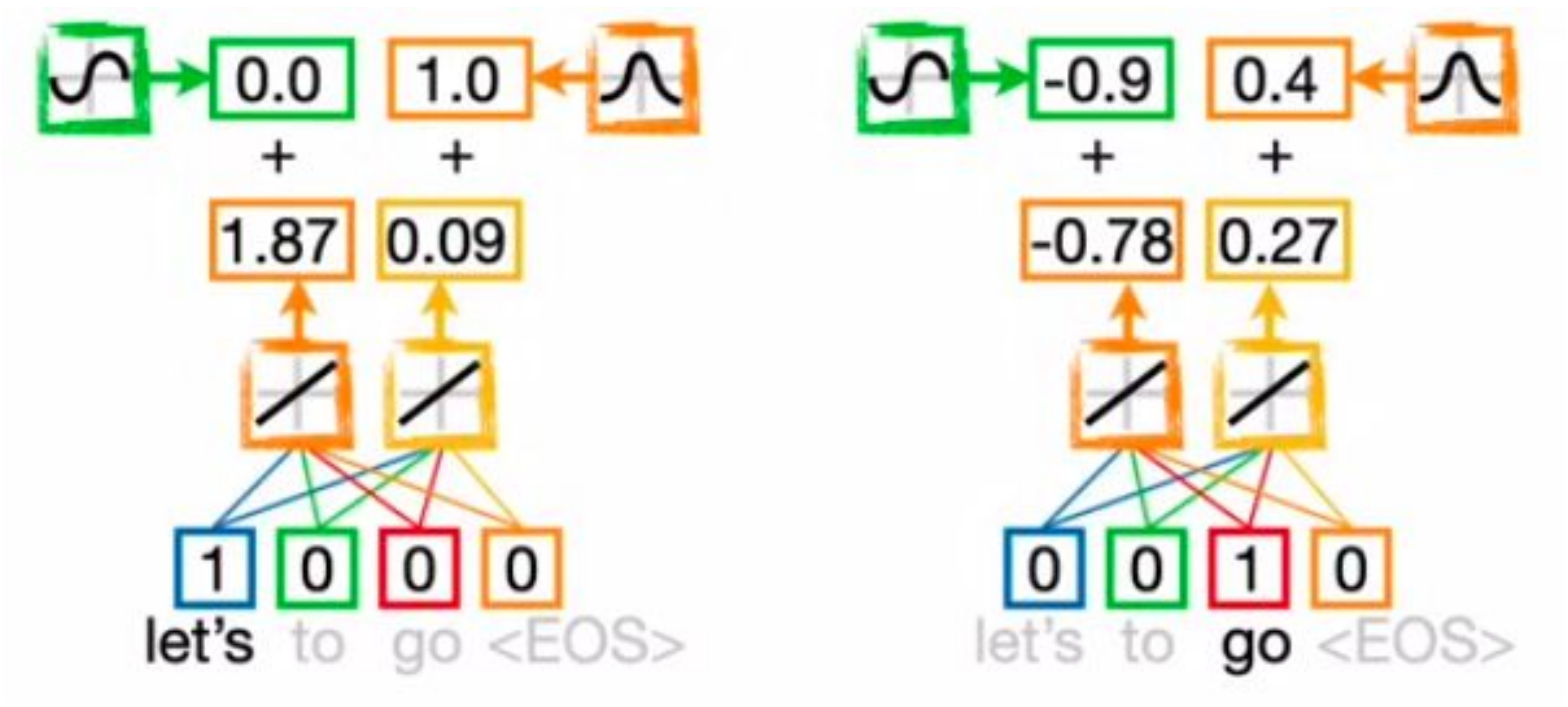
- Back to the translation example.

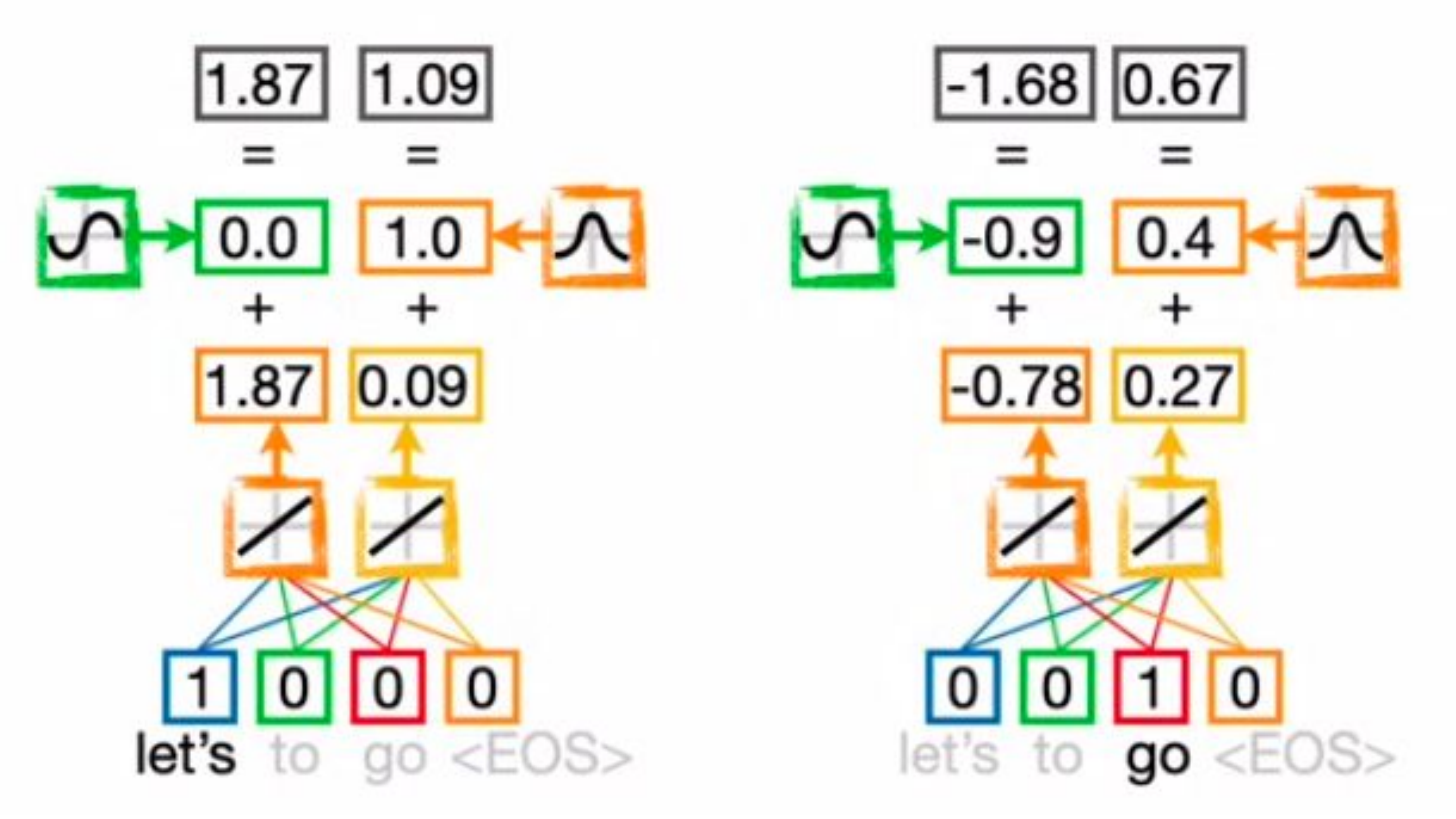


Transformers

Positional encoding

- Back to the translation example.





Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

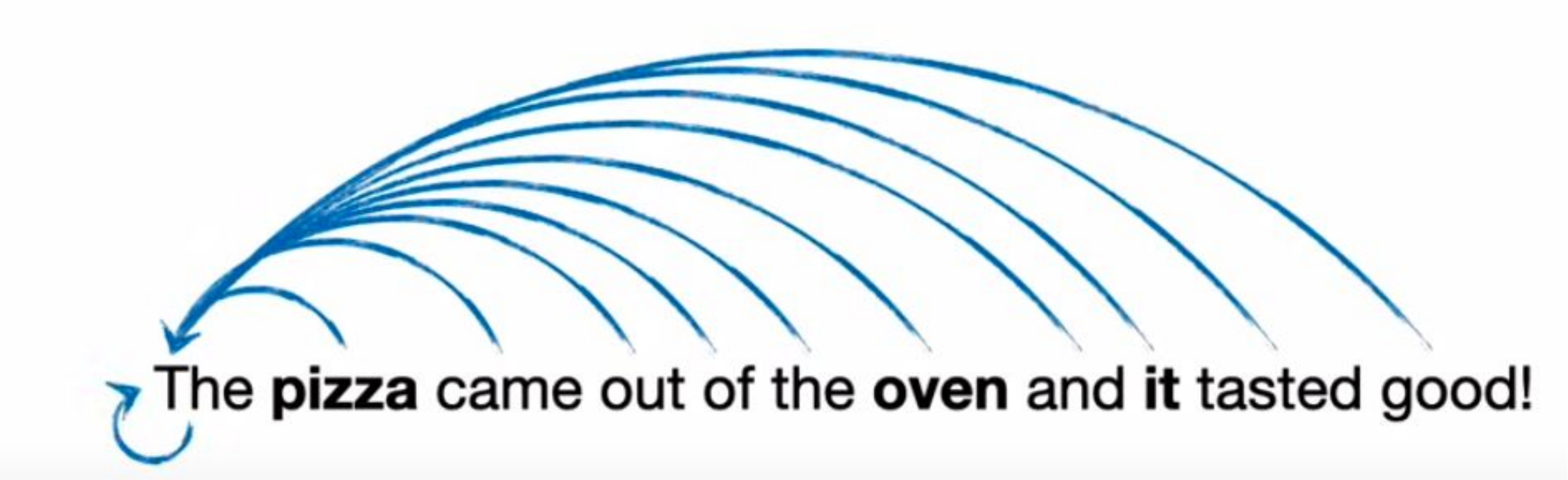
- Language is ambiguous
  - There is cohereference, linkage etc.
    - Ex: “The pizza came out of the oven and it tasted good!”
      - The word “it” refers to pizza or oven?
        - Does oven taste good?

Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- Self Attention is a mechanism to correctly associate the word “it” to the word “pizza”.
  - How similar is each word to all other words in the sentence, including itself.
    - Once the similarities are calculated, they are used to determine how the Transformer encodes each word.
      - The word “it” is more commonly associate with “pizza” than with “oven”. Similarity between “it” and “pizza” is greater.

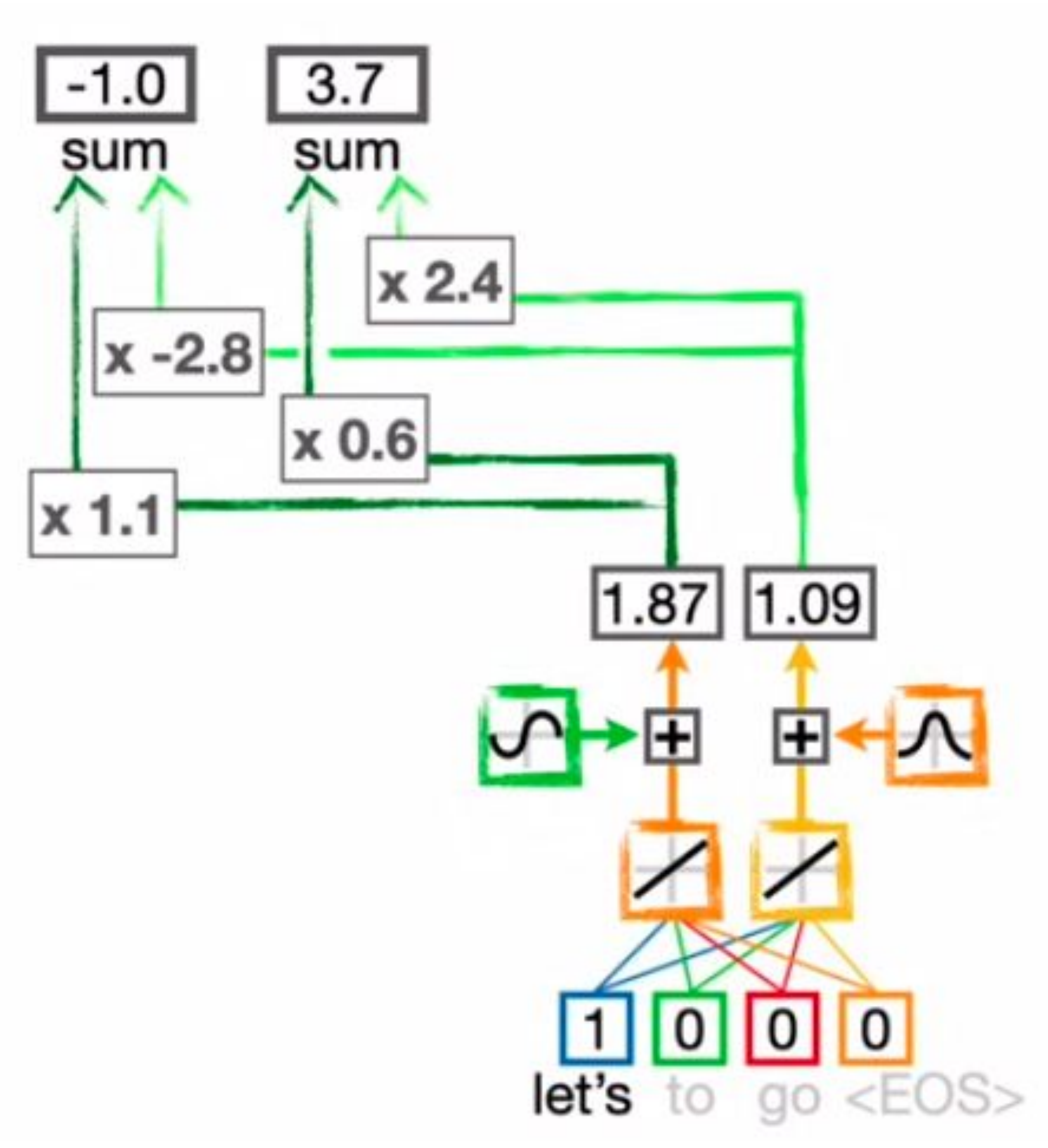


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values**

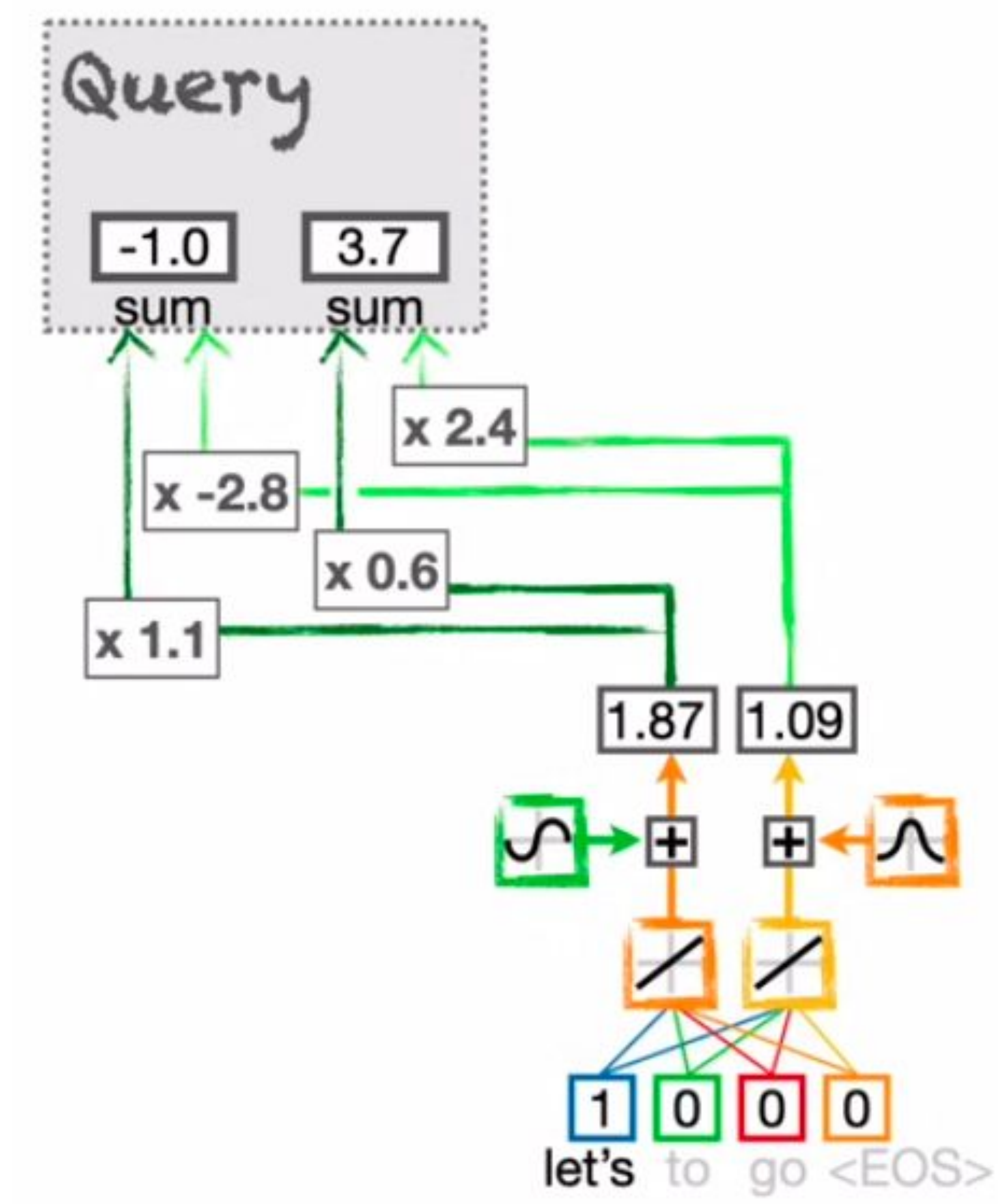


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values**

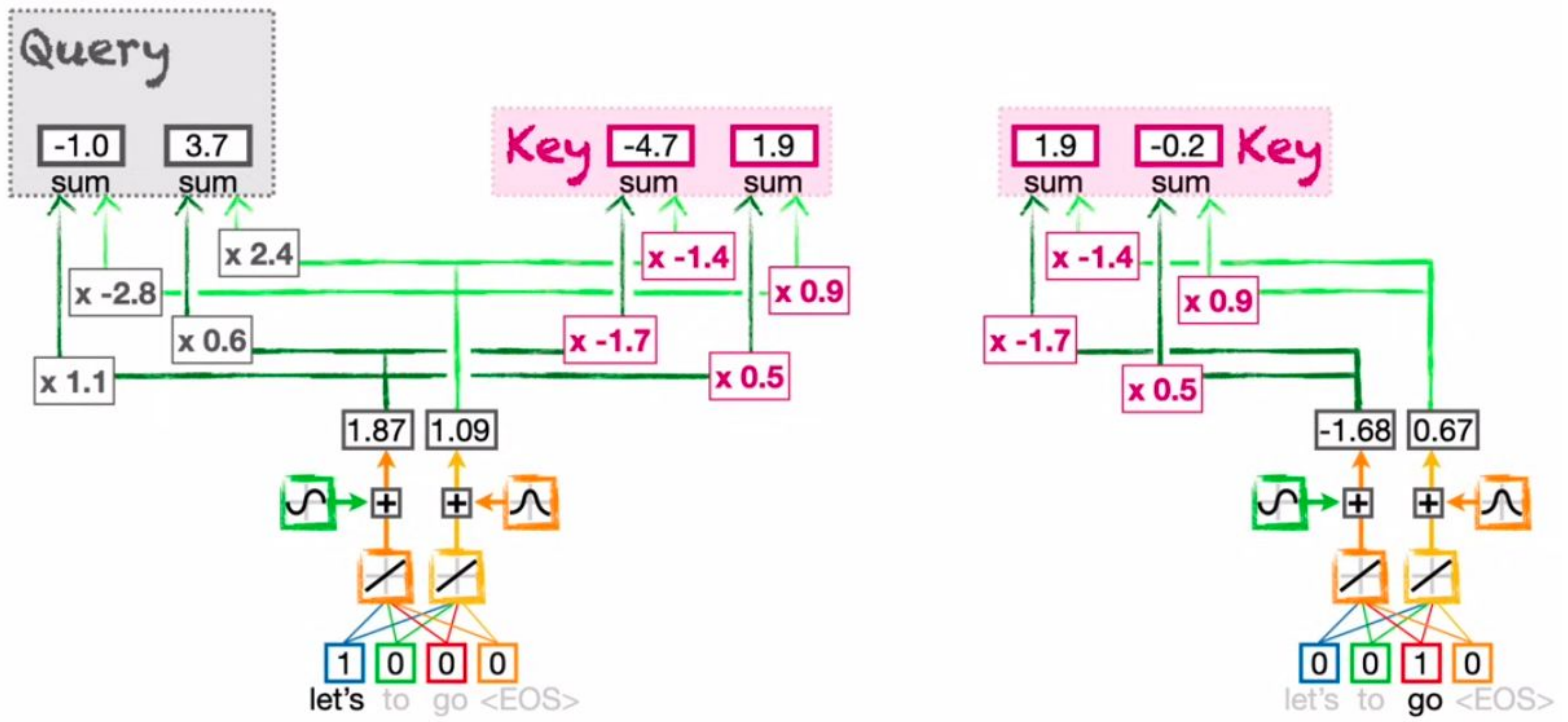


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values** and **Key Values**
  - Similarity == dot product

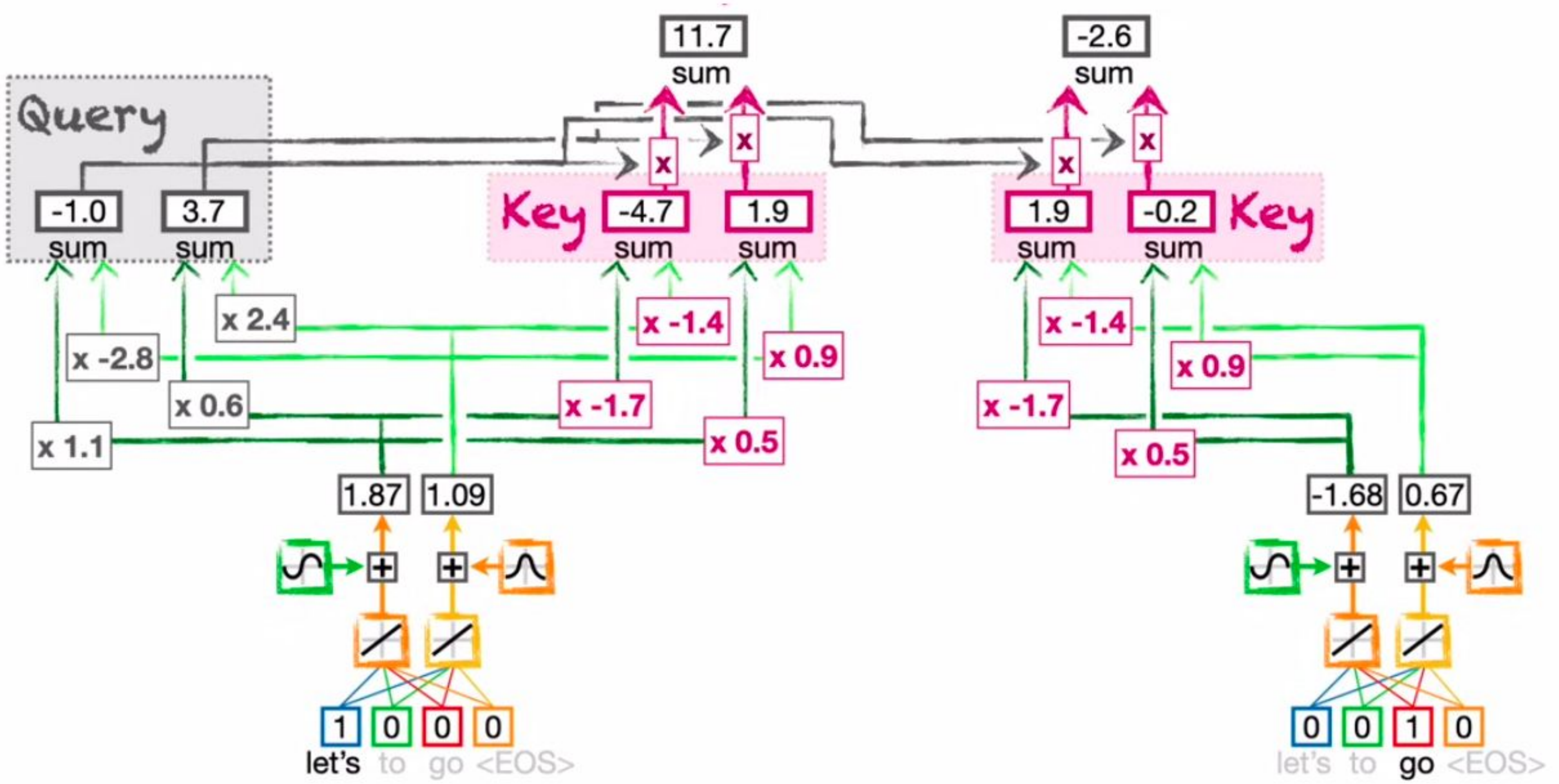


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values** and **Key Values**
  - Similarity == dot product
    - We want “let’s” to have more influence than the word “go”

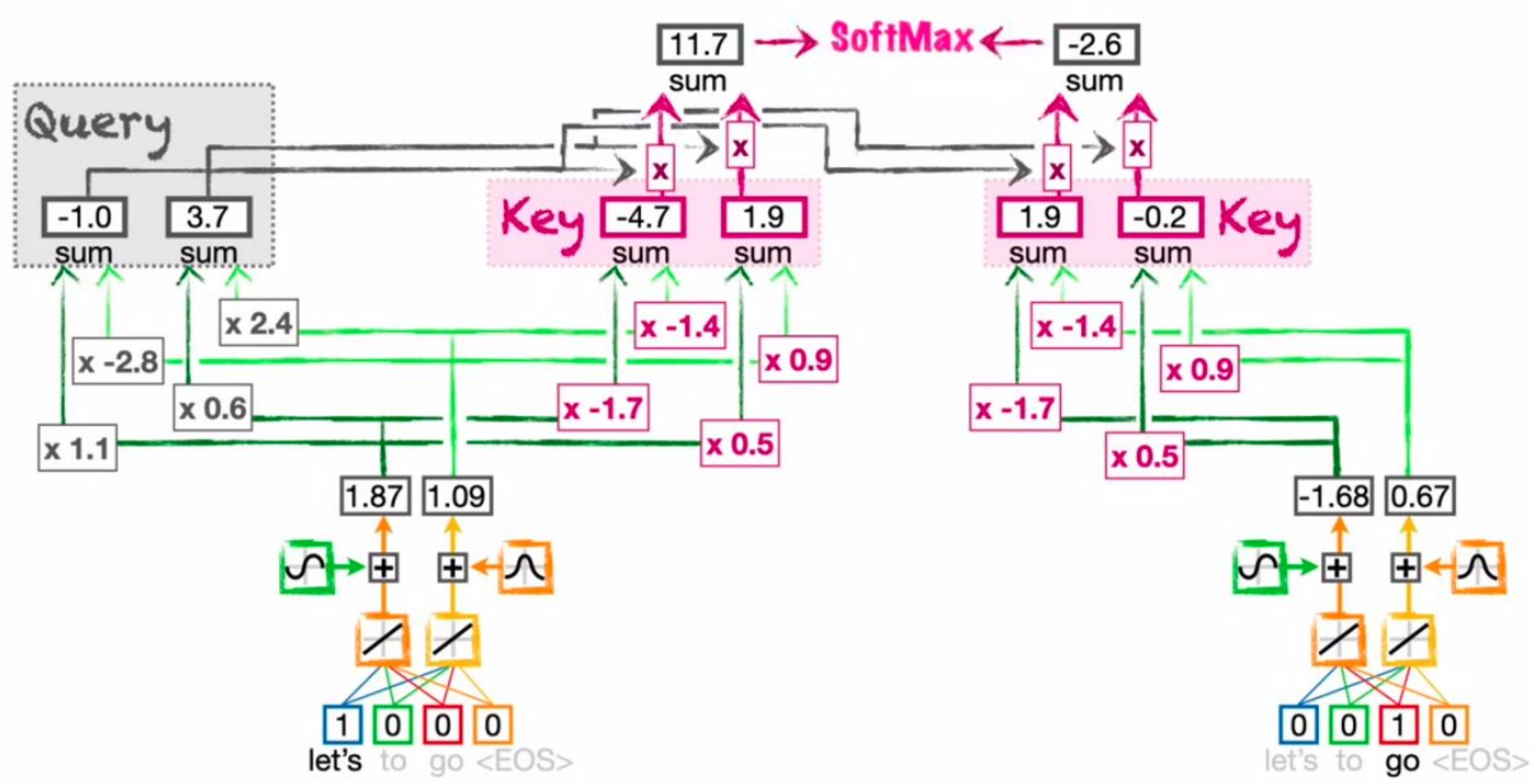


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values** and **Key Values**
  - Similarity == dot product
    - We want "let's" to have more influence than the word "go", and we do this using Softmax

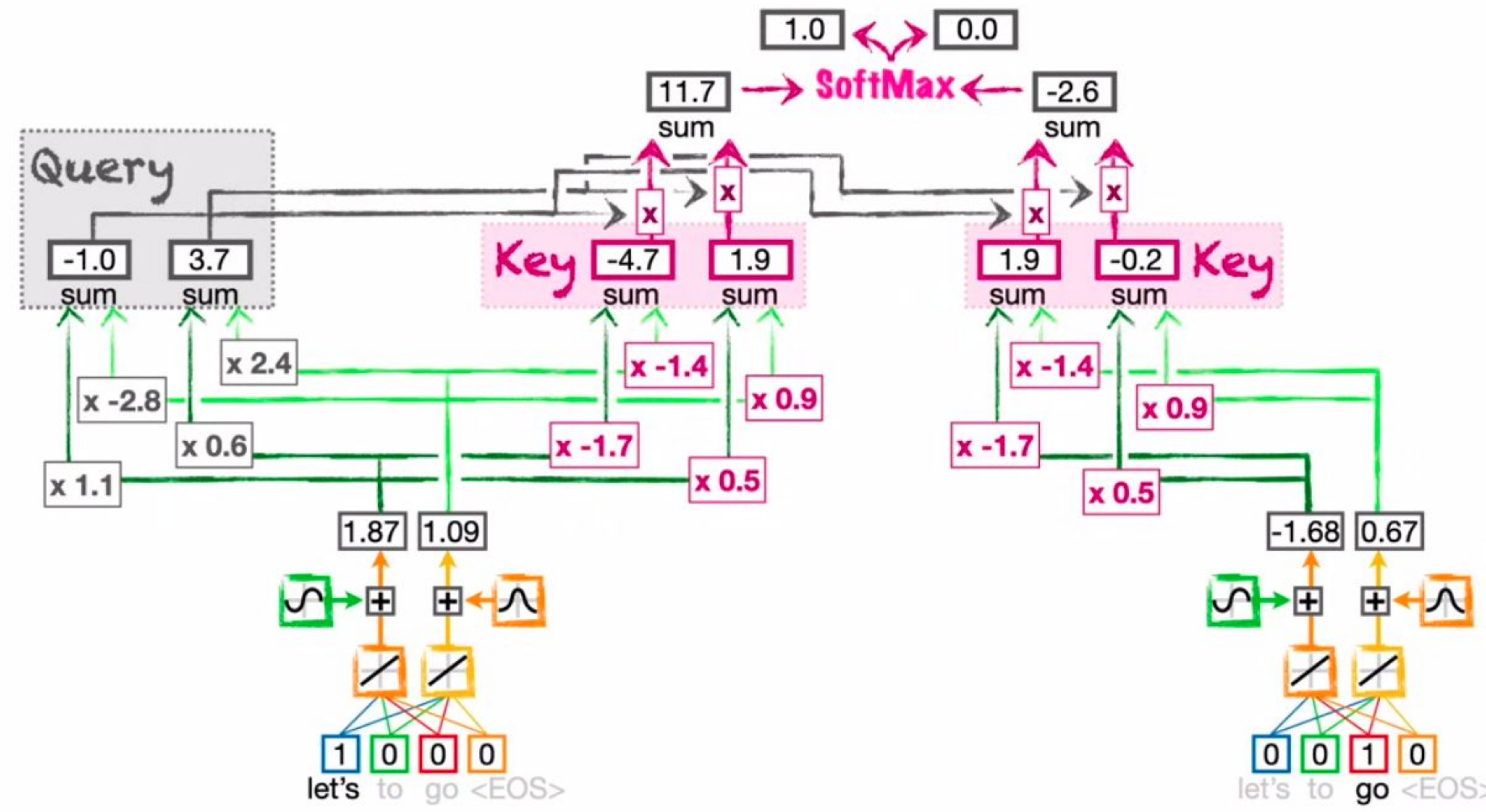


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We start by creating the **Query Values** and **Key Values**
  - Similarity == dot product
    - We want “let’s” to have more influence than the word “go”, and we do this using Softmax

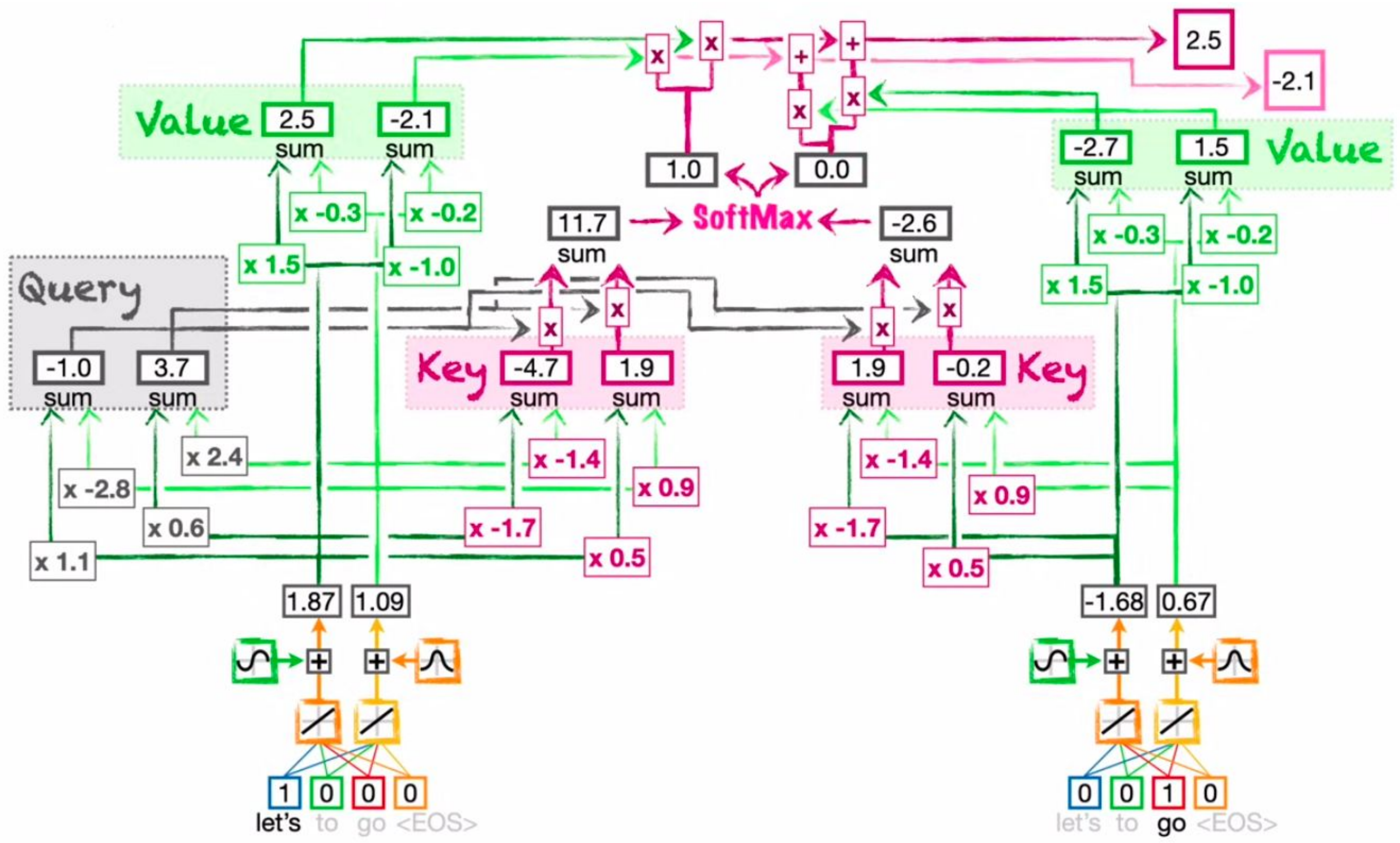


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- In order to scale each word we create a third component, which is called **Values**
  - Each word has its **Values** which will be scaled according to the influence of each word.
    - In the figure, we have the self-attention values for “Let’s”

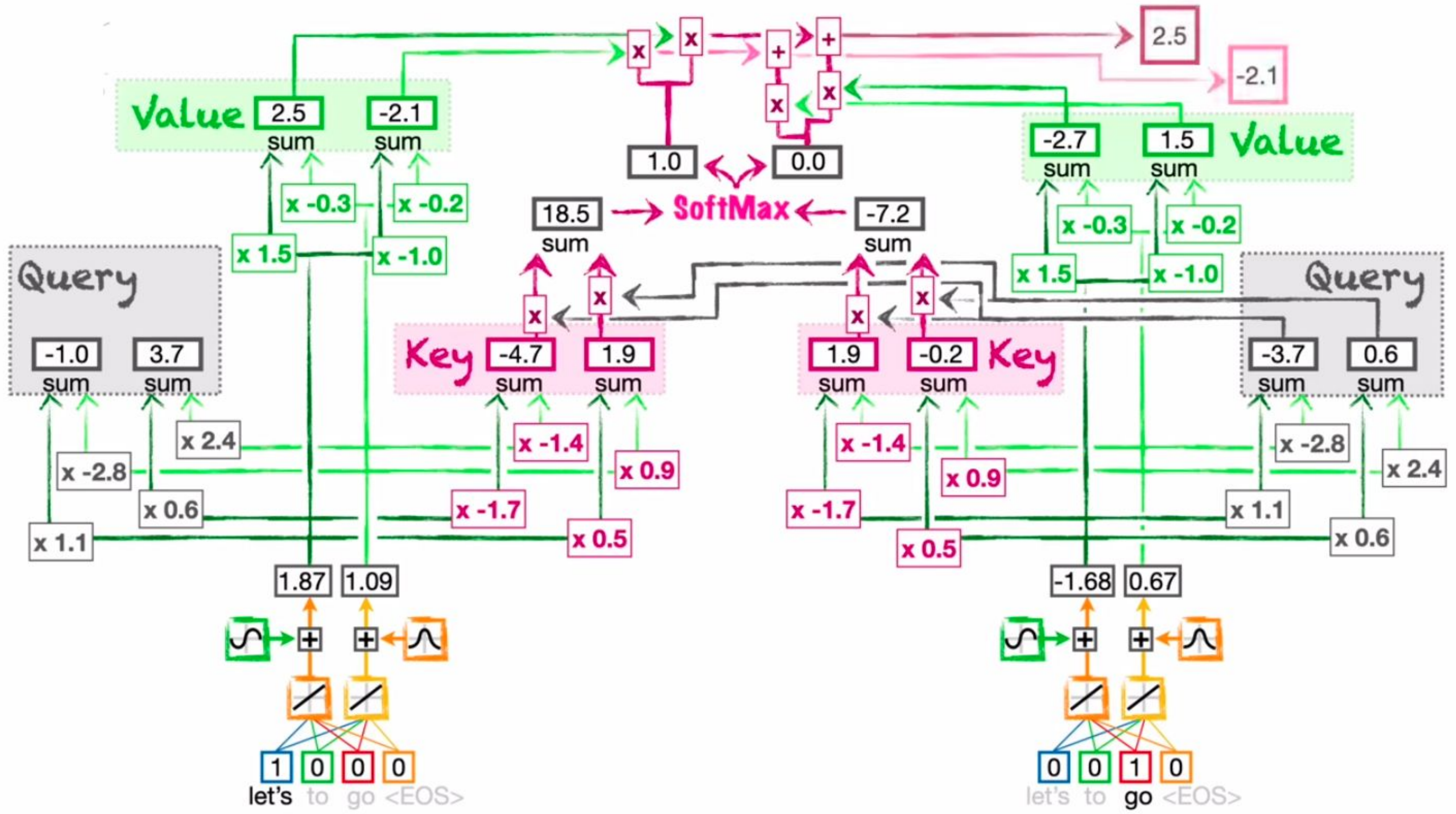


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- The same process is done for all other input words.
  - But the good news is that we do not have to recalculate the **Keys** and **Values**.
    - All is needed is to create a **Query** for the other words.
      - In the figure we have self-attention values for the word "go".



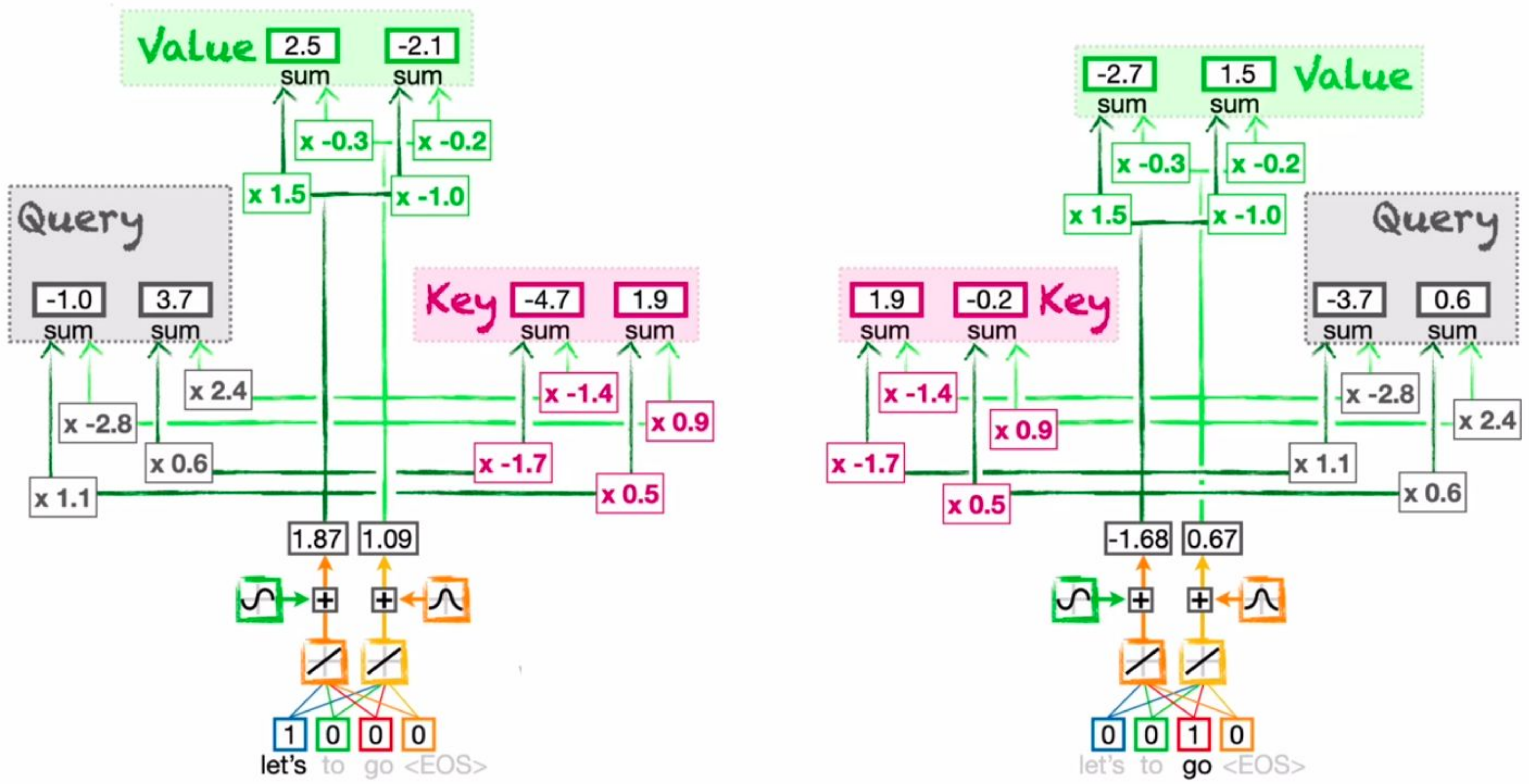
Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- The weights that we used to calculate the self-attention queries are the exact same for all input words.
    - No matter how many words are in the input.
- Likewise, the weights used to calculate self-attention keys and values are the same for each input word.

- Important!
    - Note that we can calculate the **Queries**, **Values** and **Keys** for each word at the same time.
        - We do not have to calculate them for the first word first, before moving to the second word.

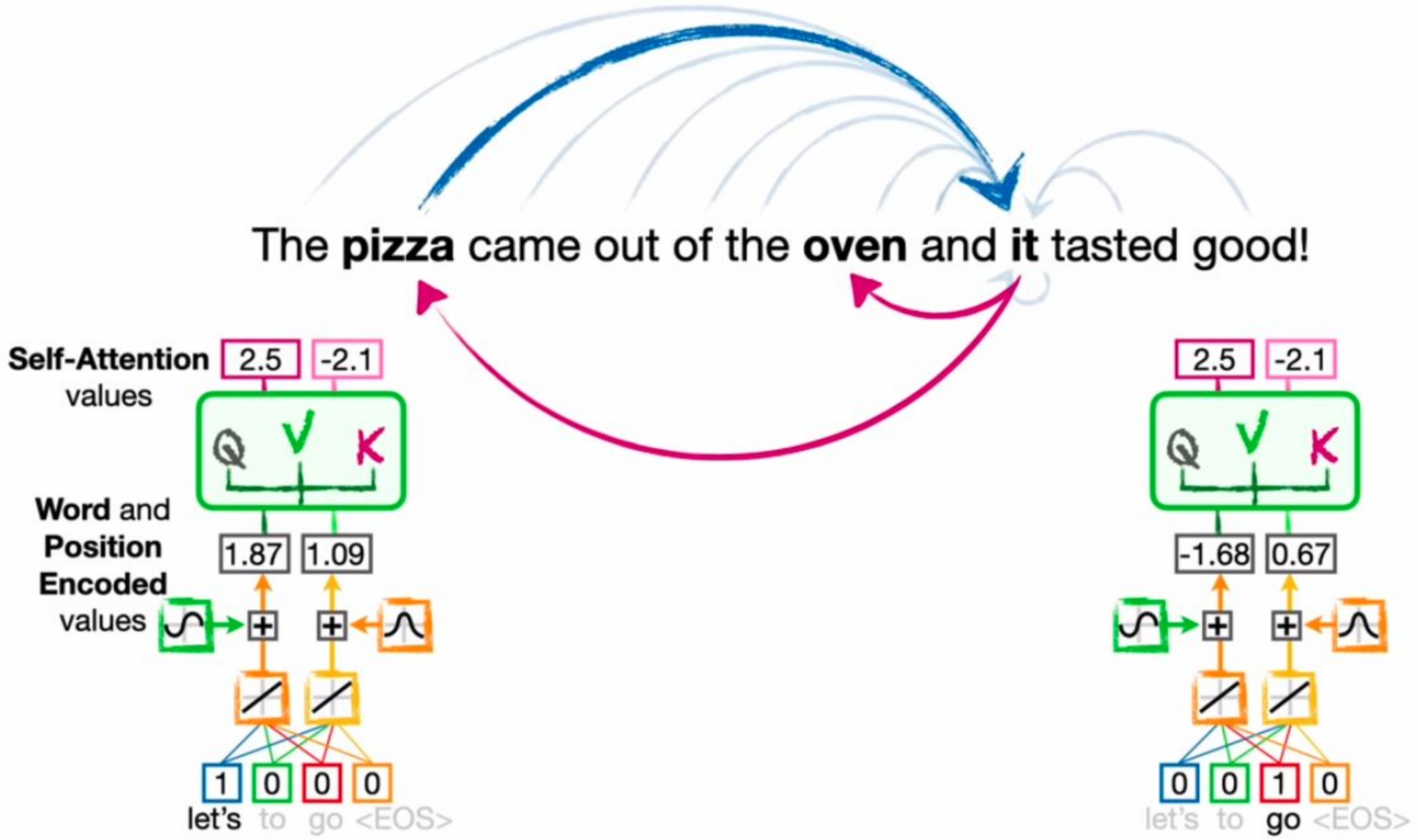


Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- Why do we need these two other values, since we already have word embeddings + positional encoding?
    - Context!
        - Self Attention values for each input word contains input from all other words.

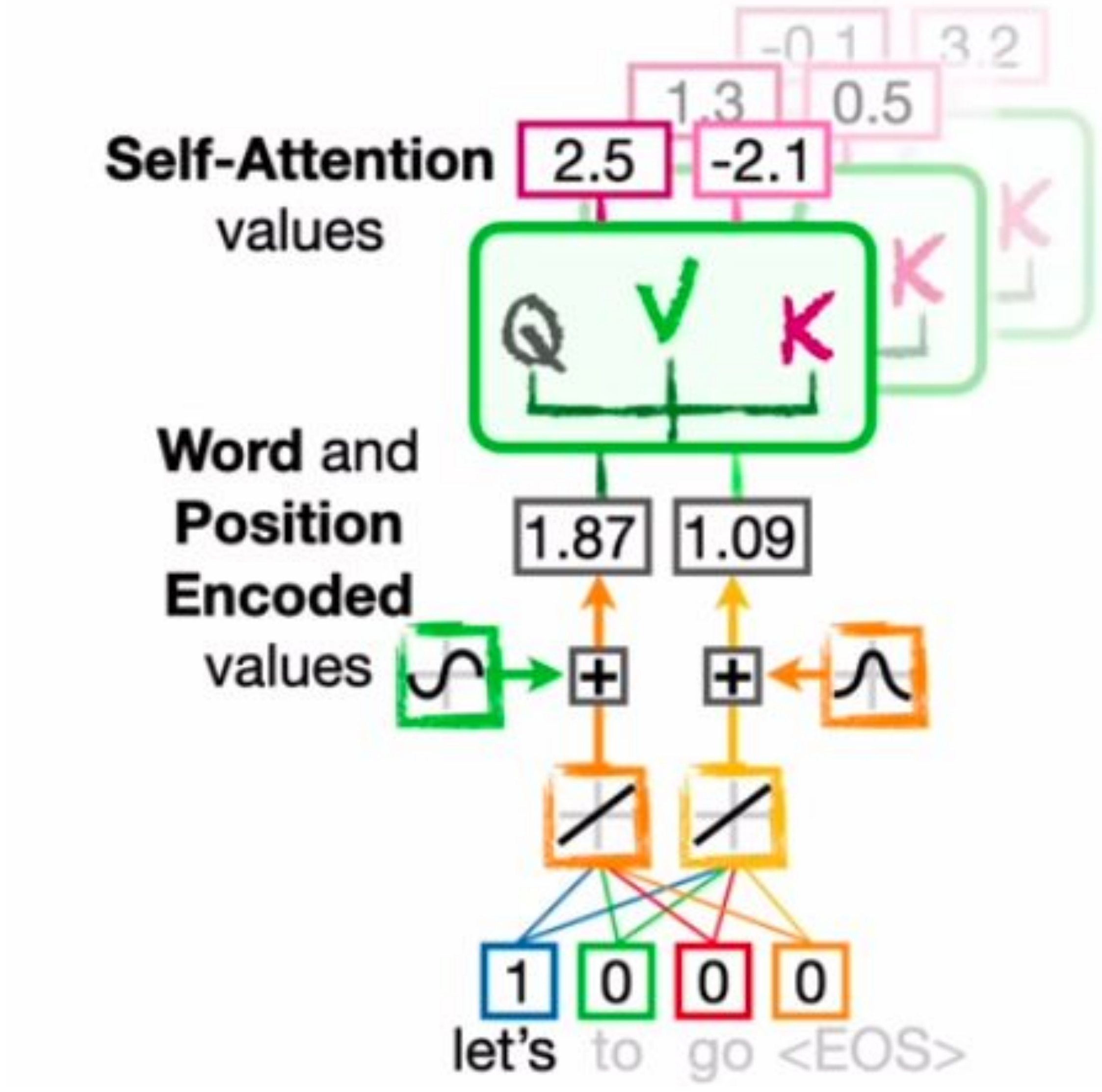


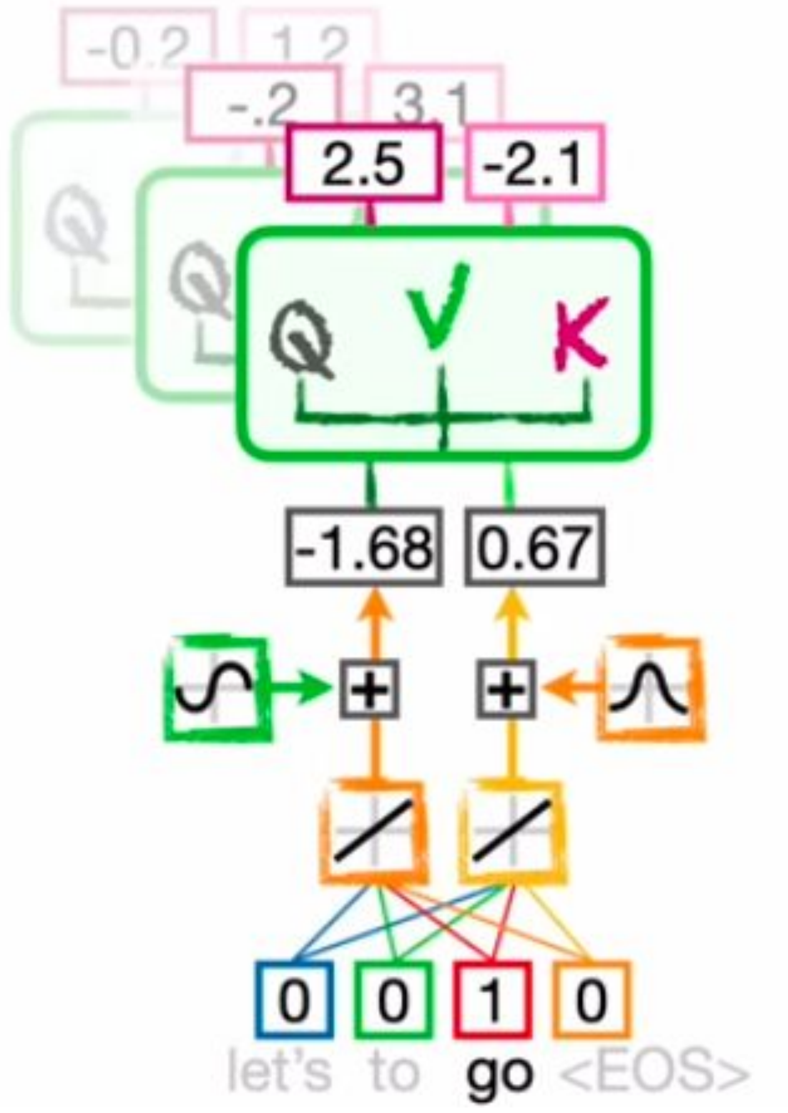
Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- We can use multiple self attention cells.
  - This is called multi-head attention.



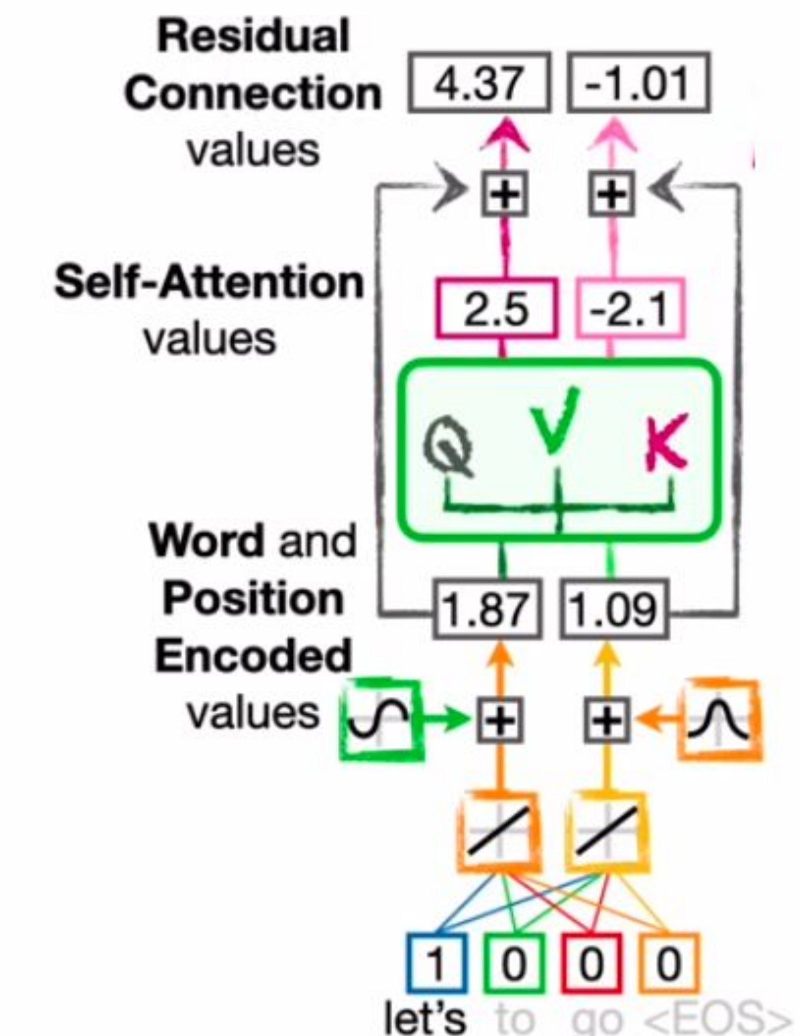


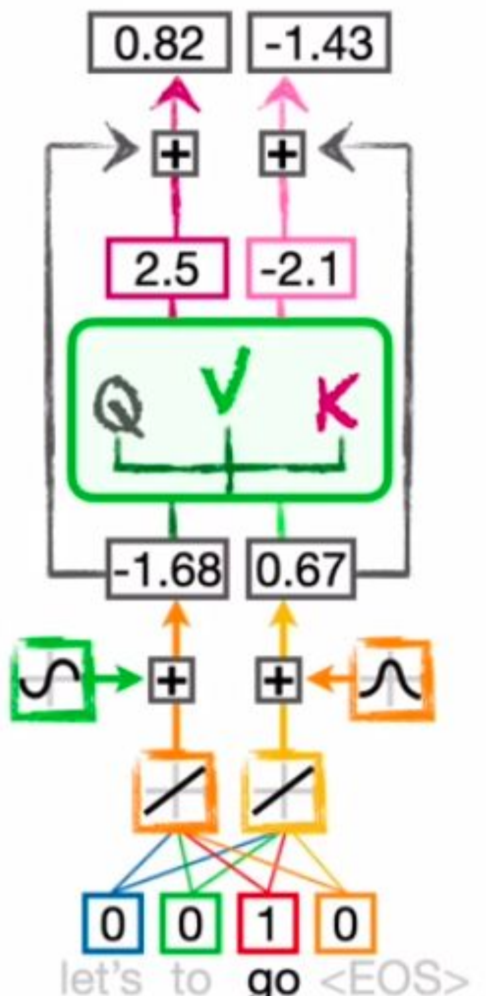
Transformers

Self Attention

_ How the Transformer keeps track of the relationship among words?

- There is one last thing to do.
  - We take the positional encoded values and them to the self-attention values.
    - These bypasses are called residual connections.



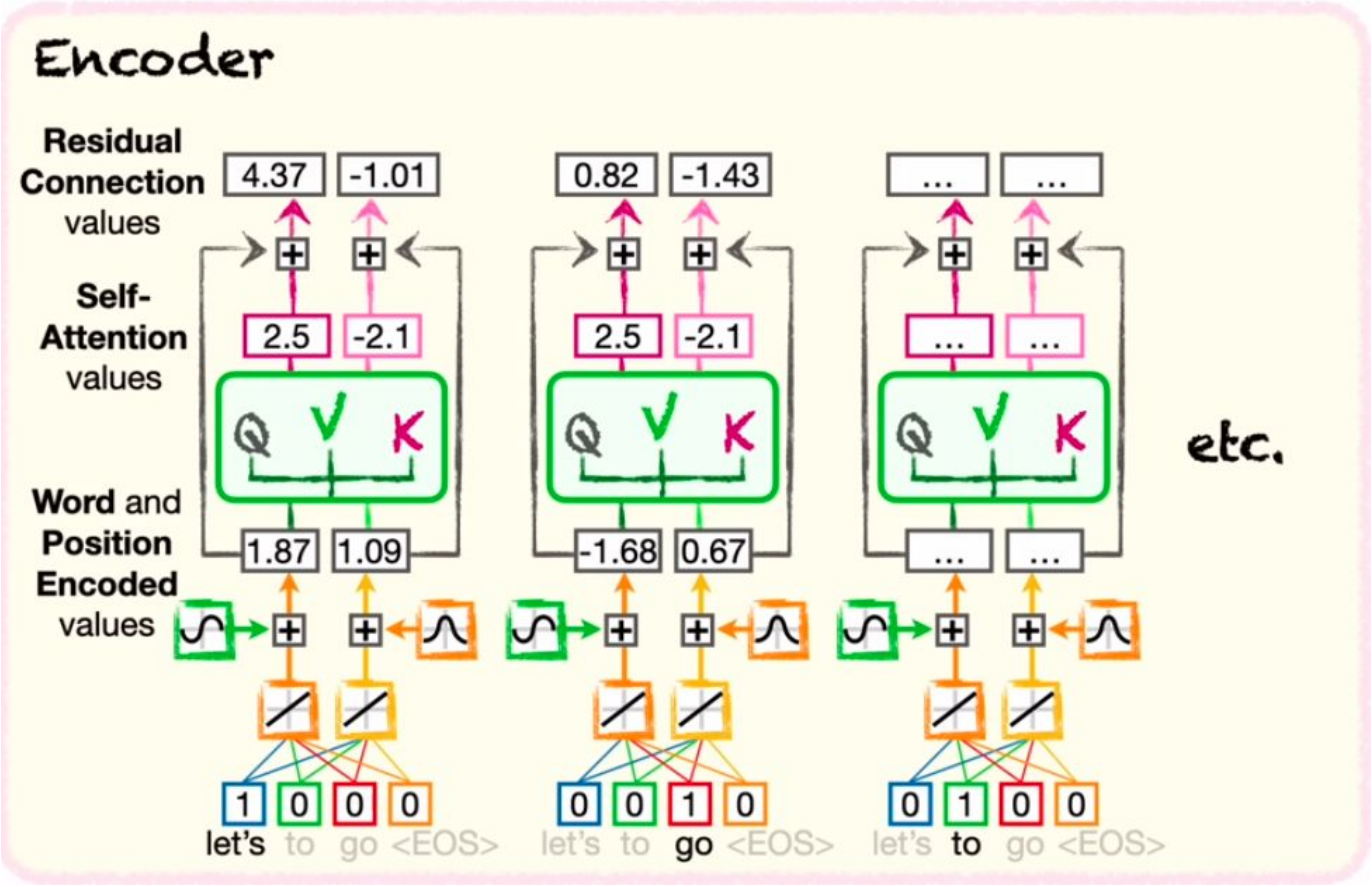


Transformers

Encoder

_ The encoder may have any input size

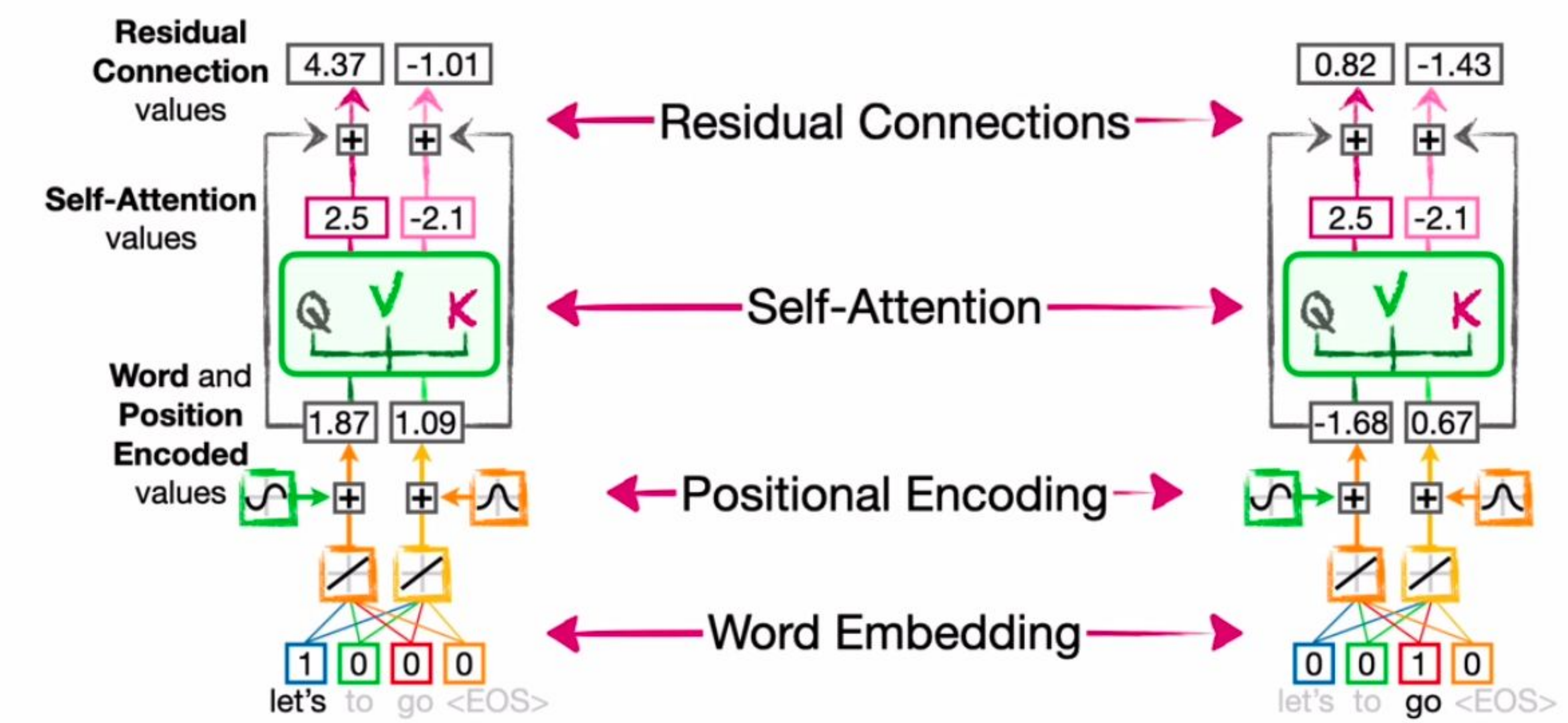
- Modern Transformers process very large inputs.
  - Each input word is processed in parallel.



Transformers

Encoder

- Parts the Transformer uses to encode the input.
  - Encode words into numbers
  - Encode the position of the words
  - encode the relationships among the words

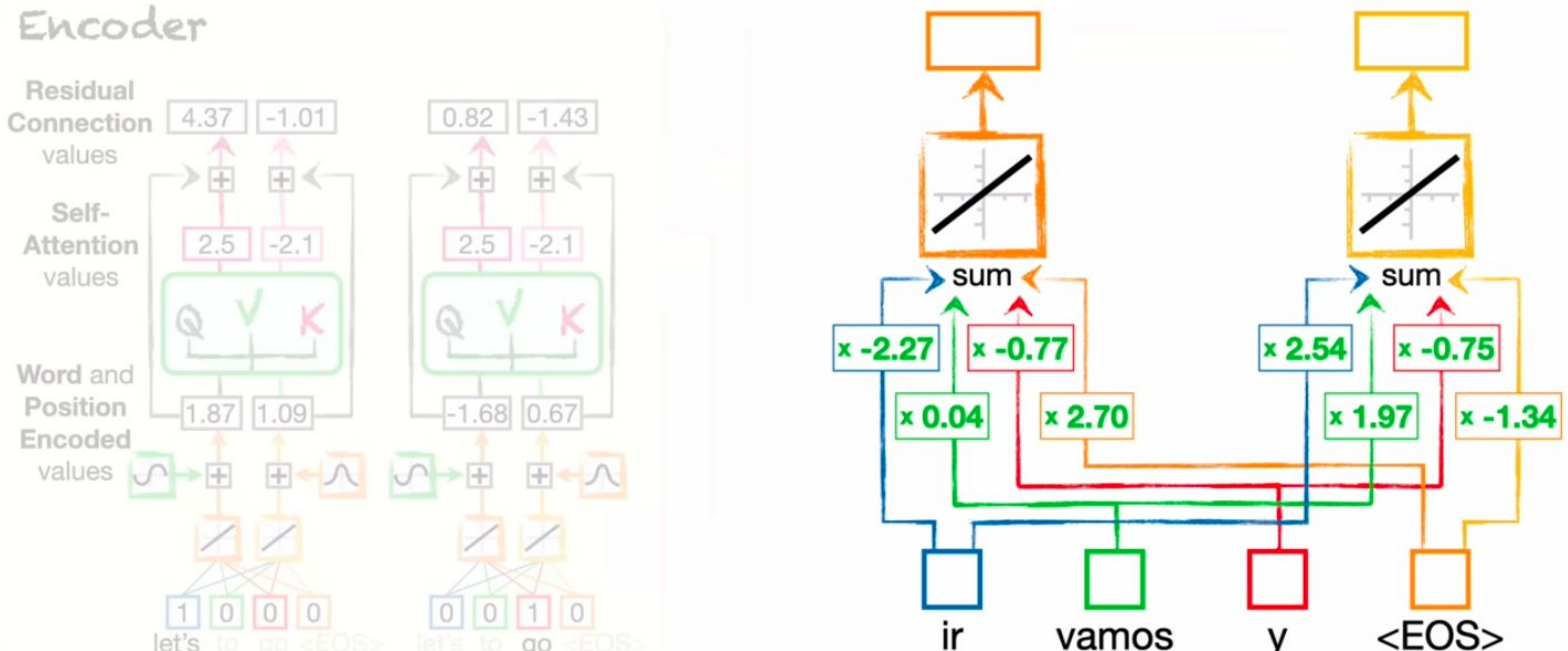


Transformers

Decoder

_ The decoder also starts with learning word embedding

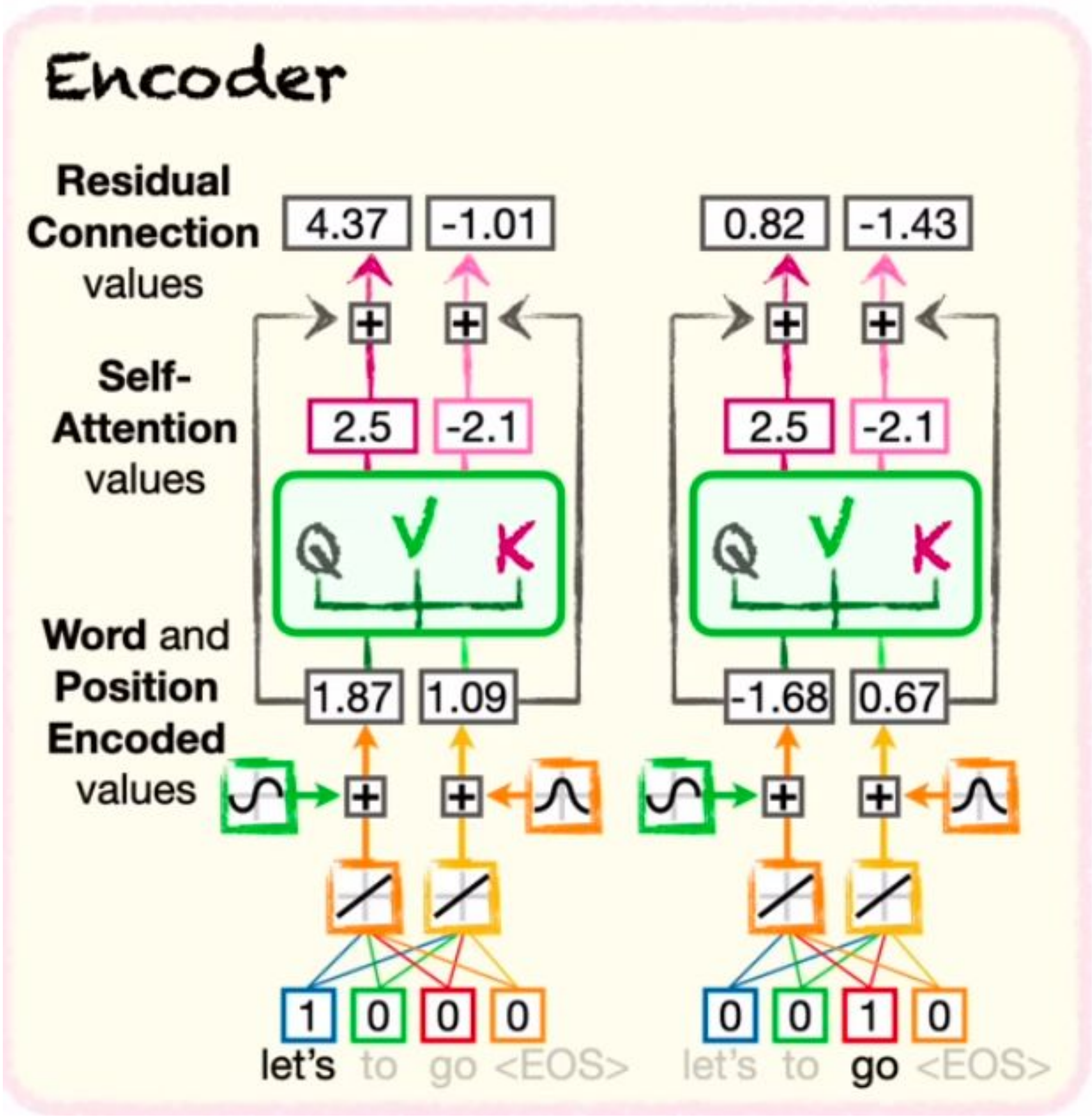
- The weights are different from the embedding layer of the encoder

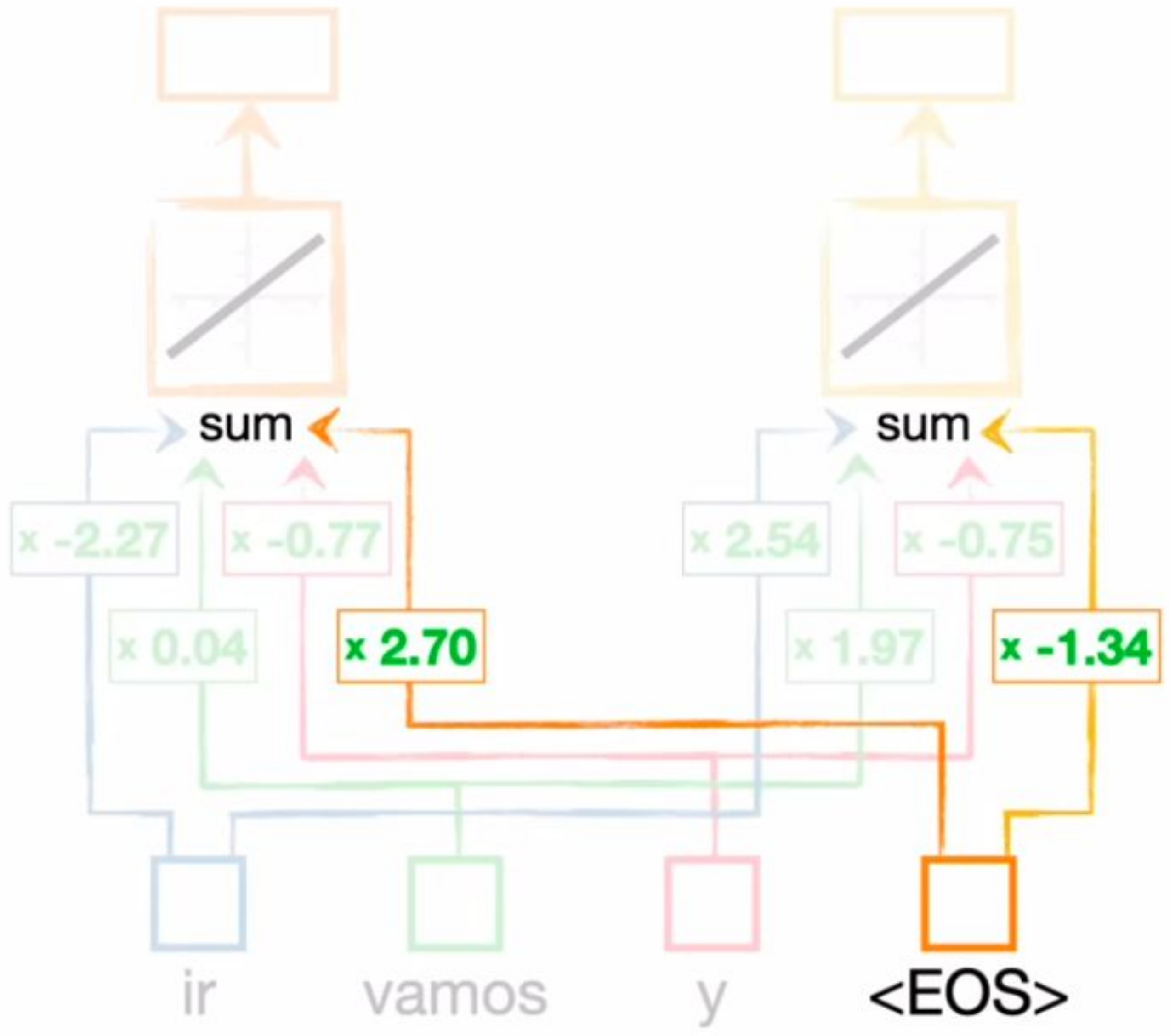


Transformers

Decoder

- The decoder also starts with learning word embeddings
- The weights are different from the embedding layer of the encoder
    - It starts with the symbol <EOS>



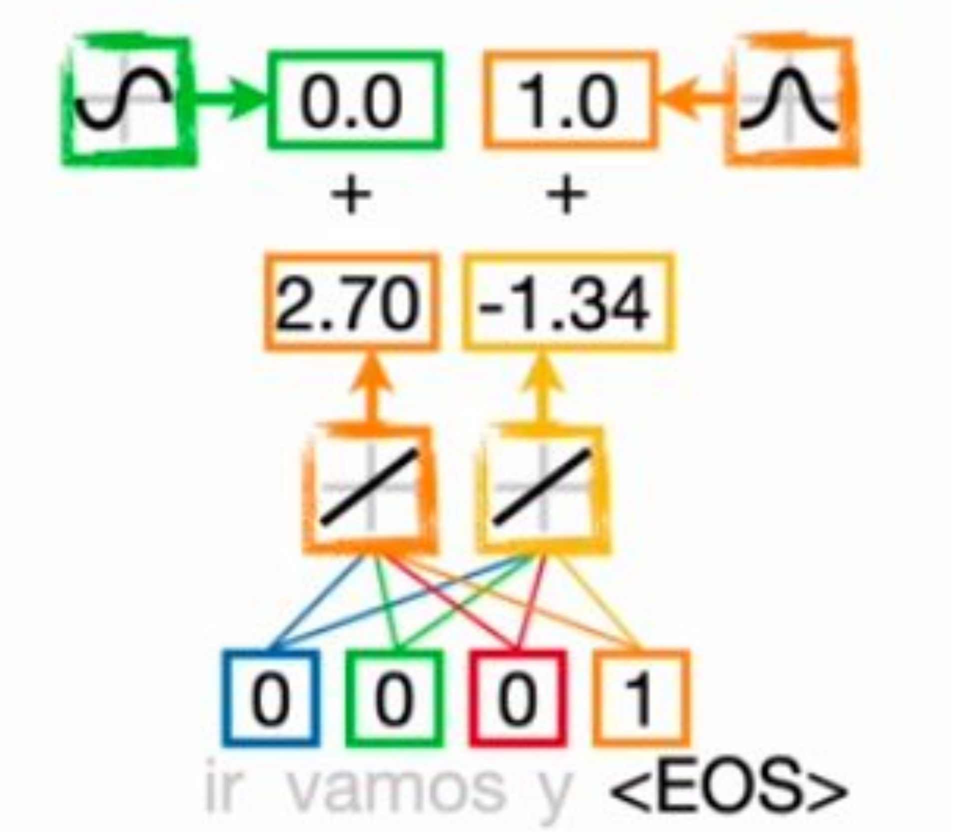


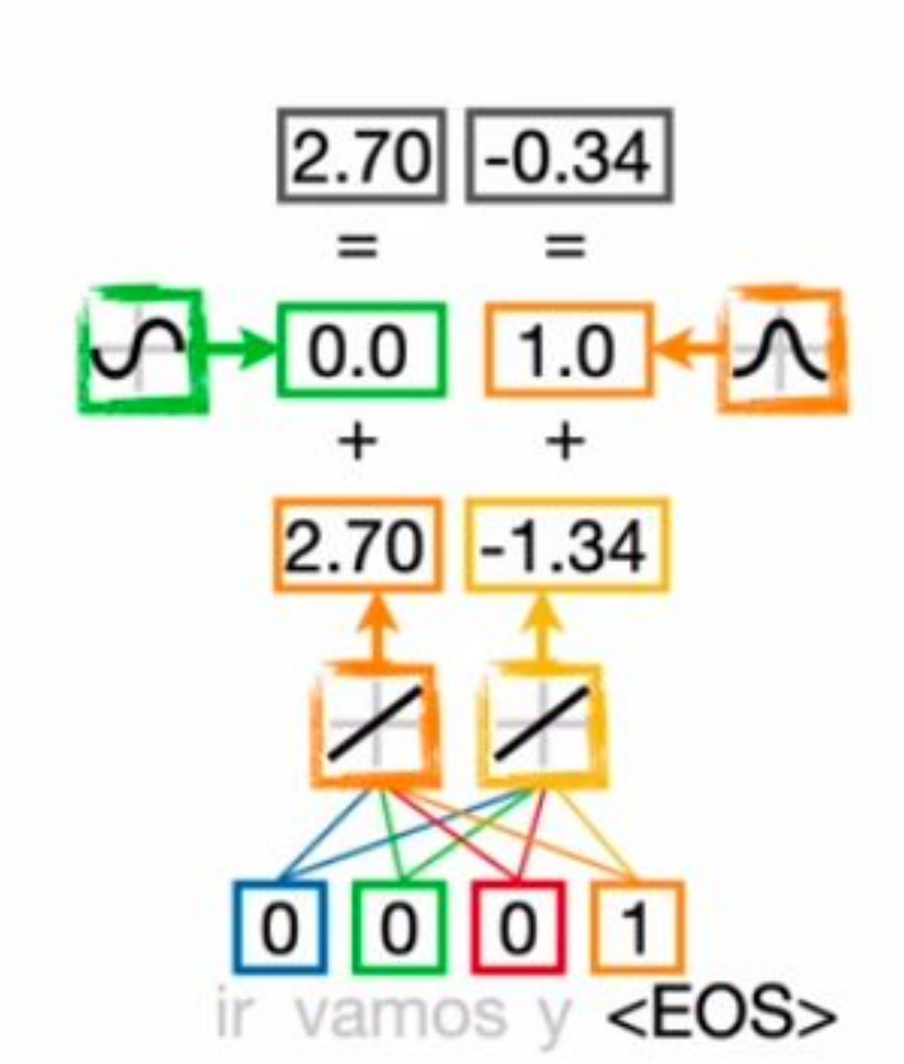
Transformers

Decoder

The decoder also starts with learning word embeddings

- The weights are different from the embedding layer of the encoder
  - It starts with the symbol <EOS>
    - Positional encoding.



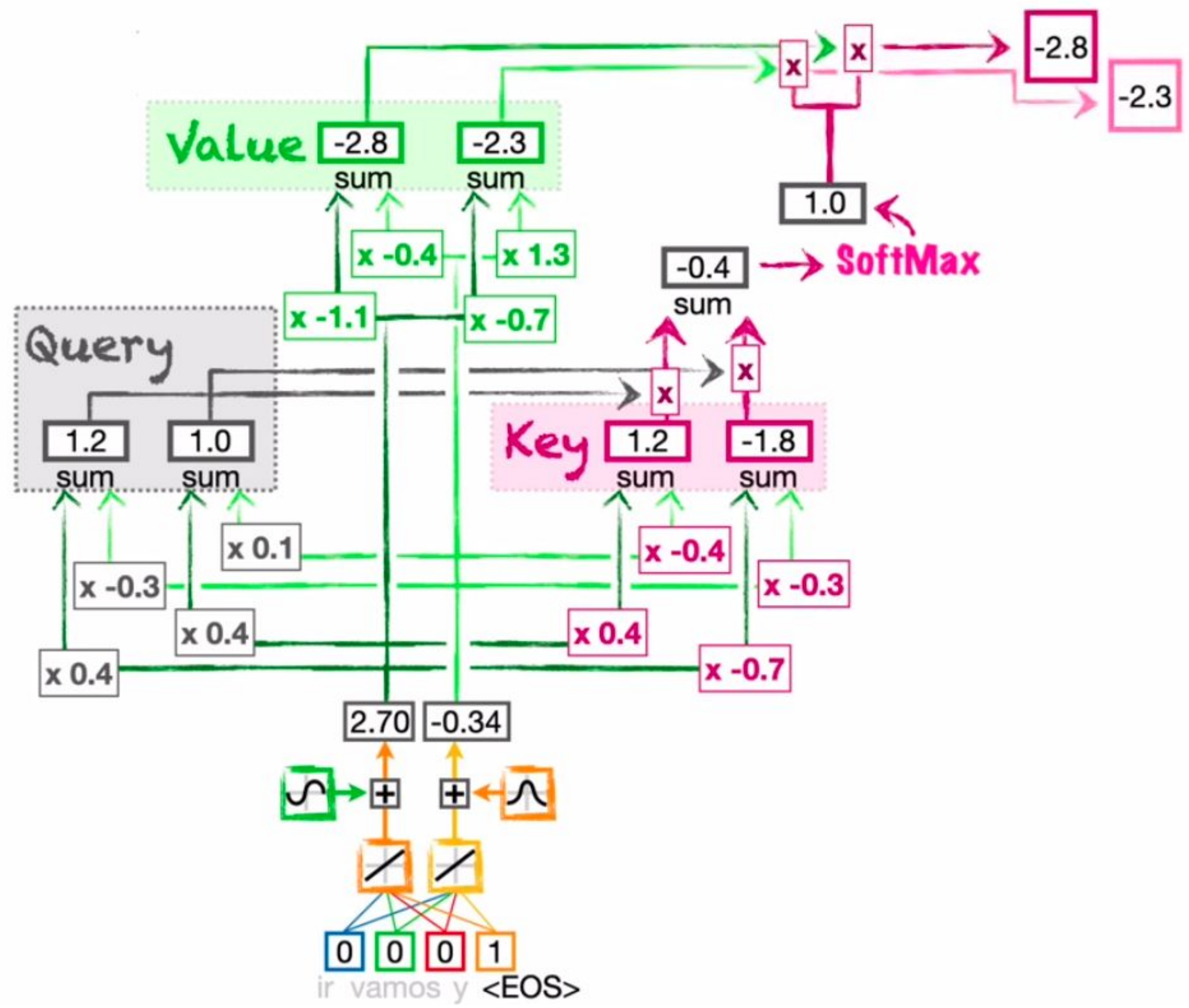


Transformers

Decoder

The same process is done for each output word.

- Self-Attention!
  - Keep track of how output words relate to each other.
    - Note: we start with only one word, and the number of output words increases as they are being decoded.

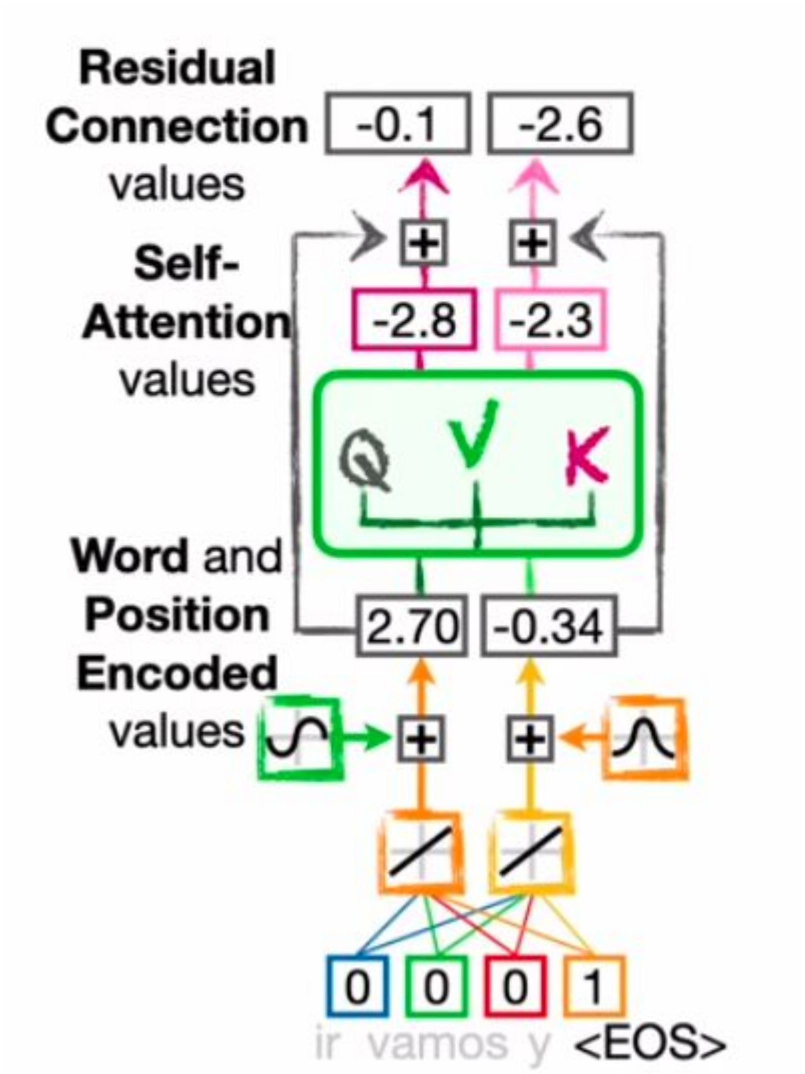


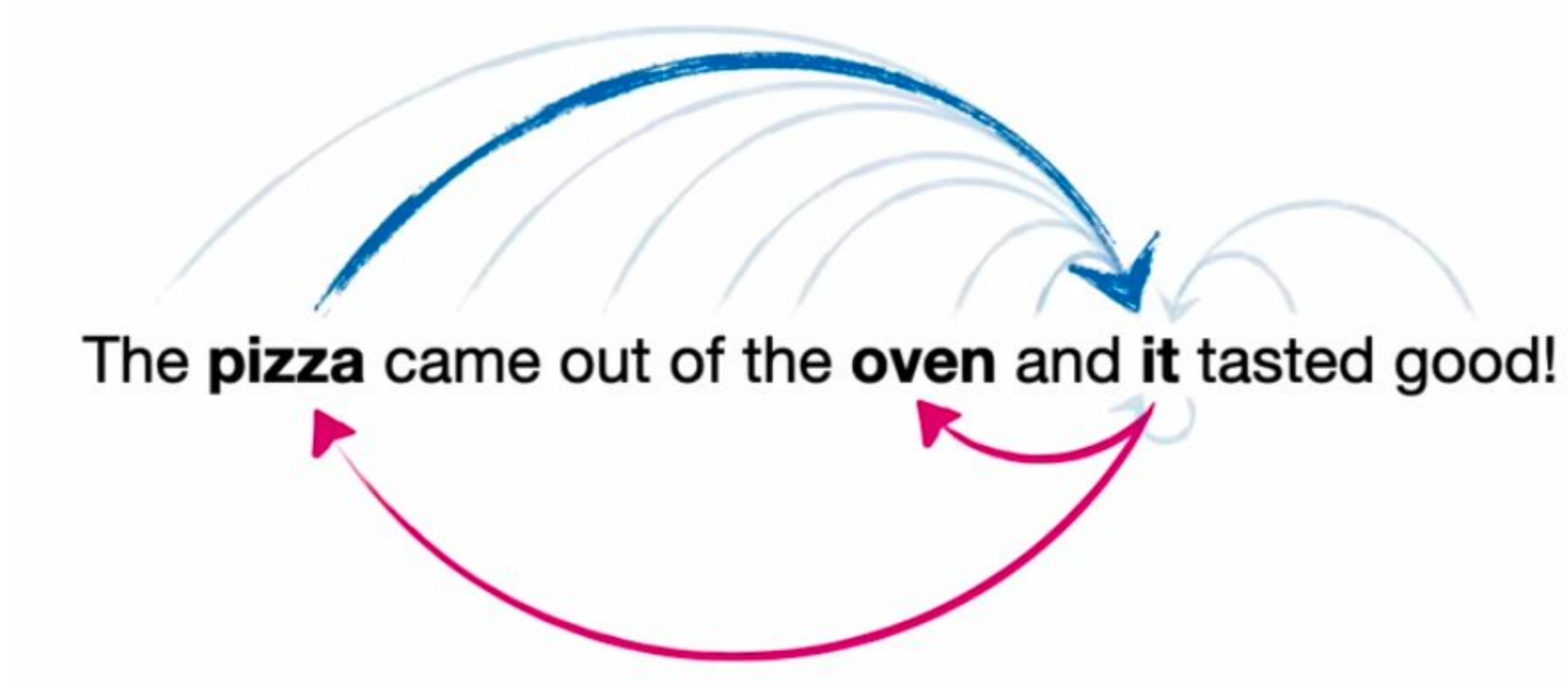
Transformers

Decoder

_ The same process is done for each output word.

- So far we talked about how input words related to each other, and how output words related to each other.

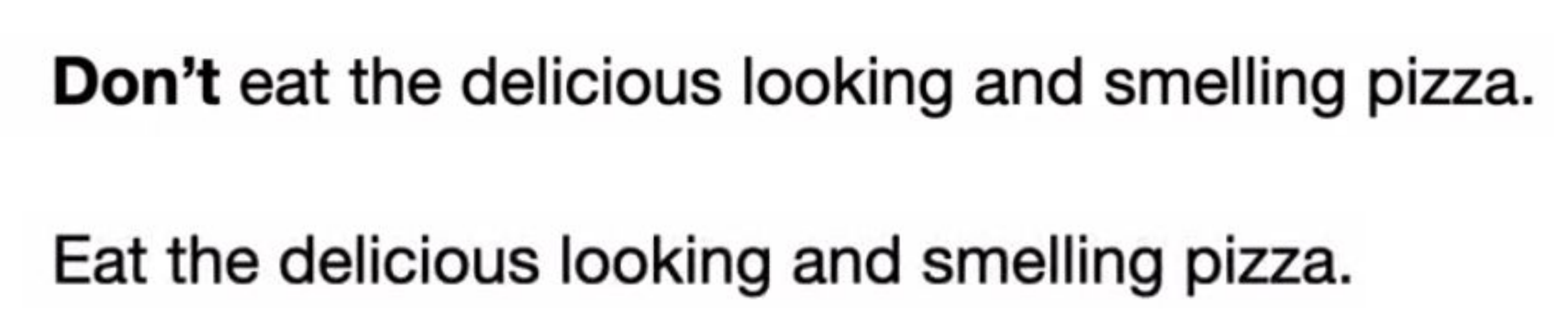


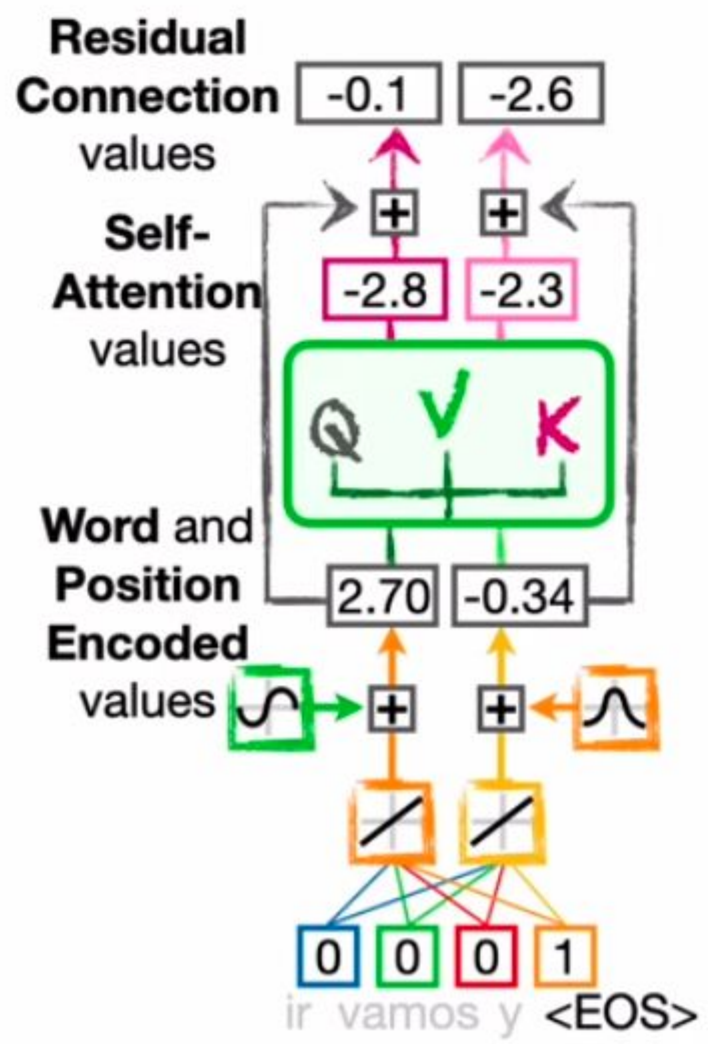


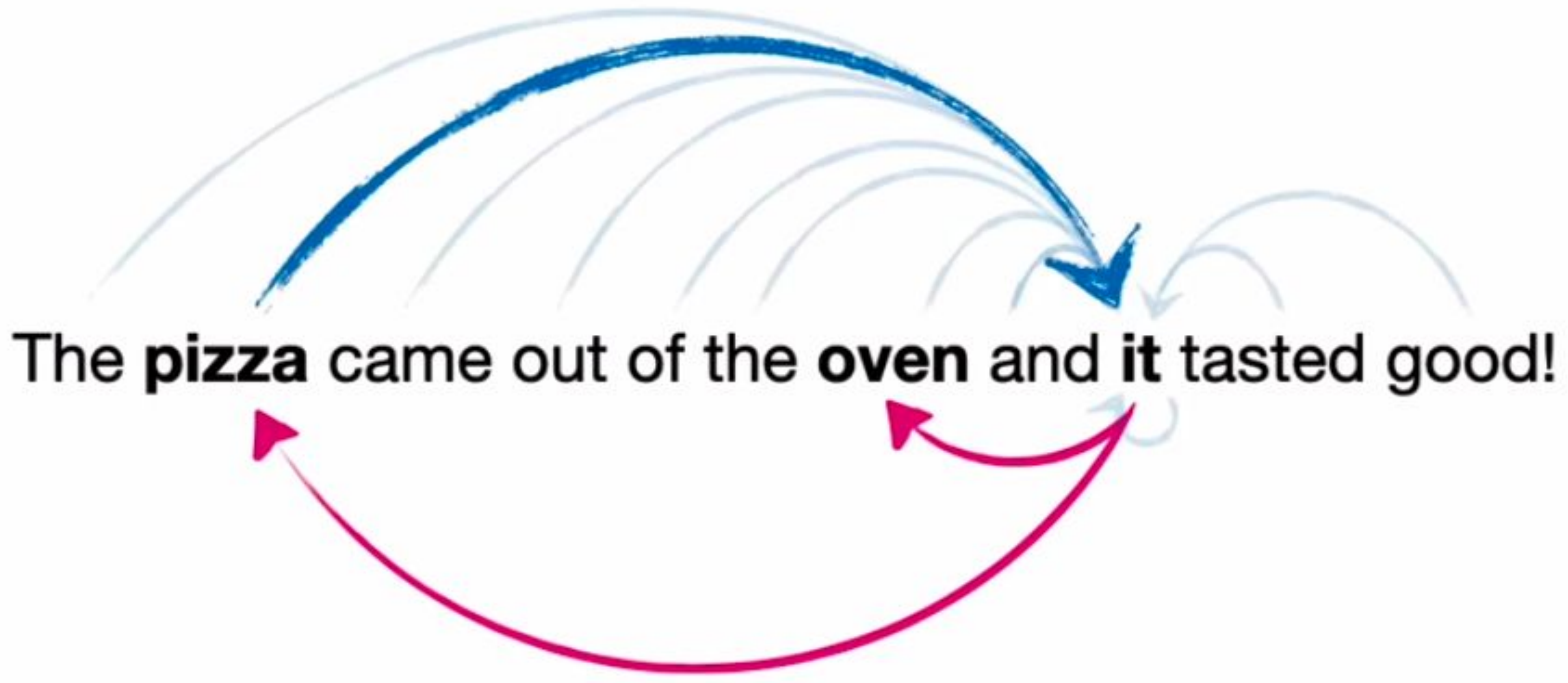
Transformers

Decoder

- The same process is done for each output word.
- So far we talked about how input words related to each other, and how output words related to each other.
  - But, how input words relate to output words?



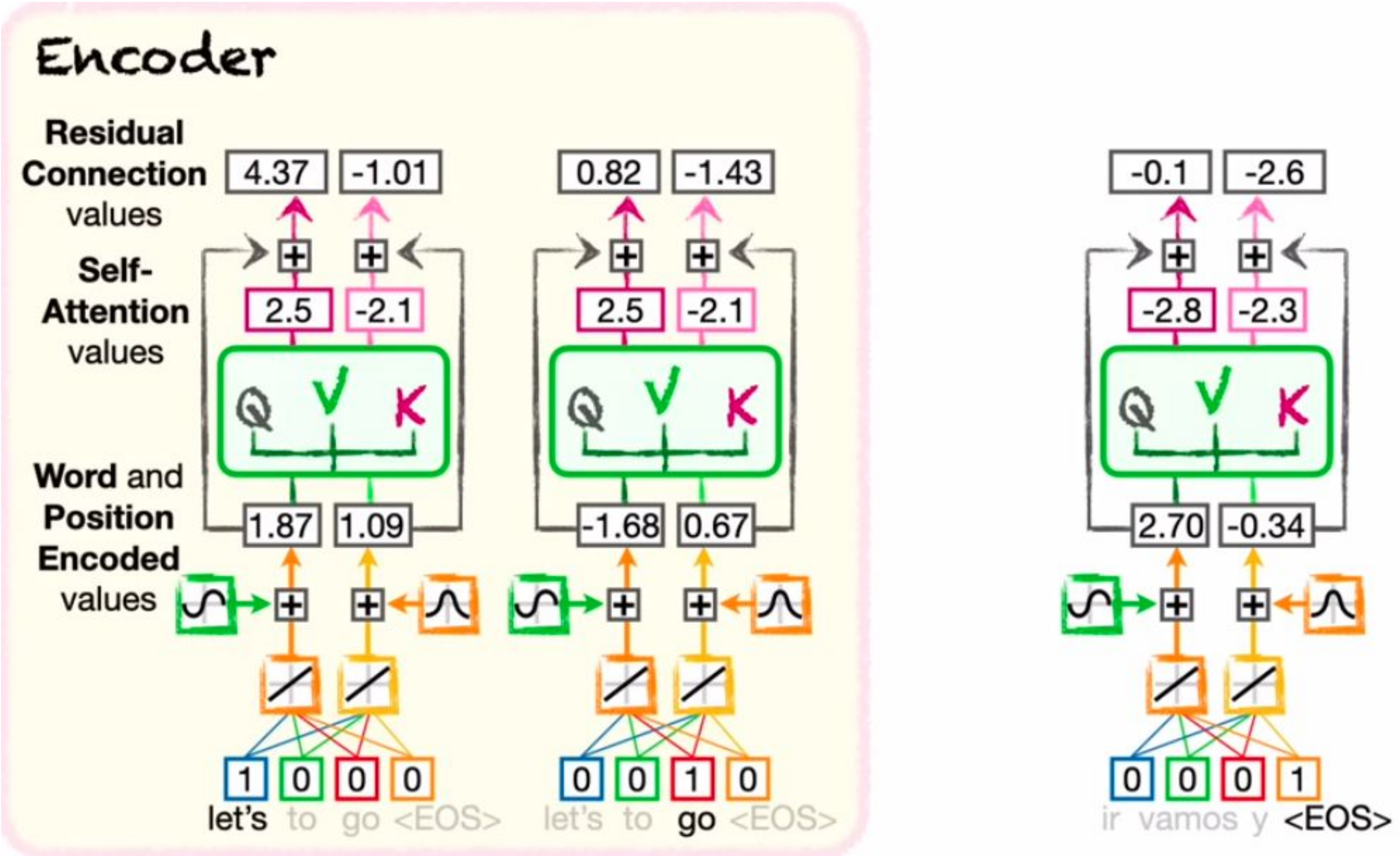




Transformers

Encoder-Decoder Attention

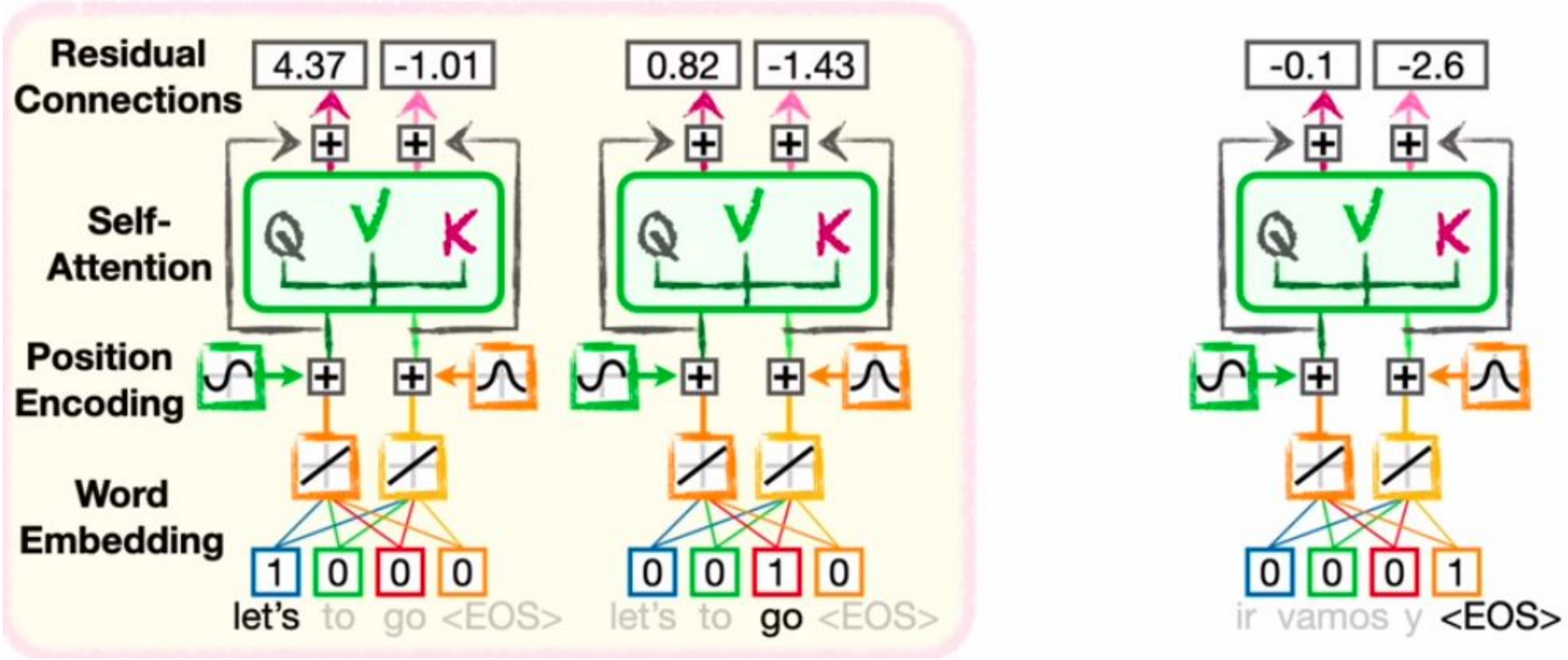
- Keep track of the significant words in the input.



Transformers

Encoder-Decoder Attention

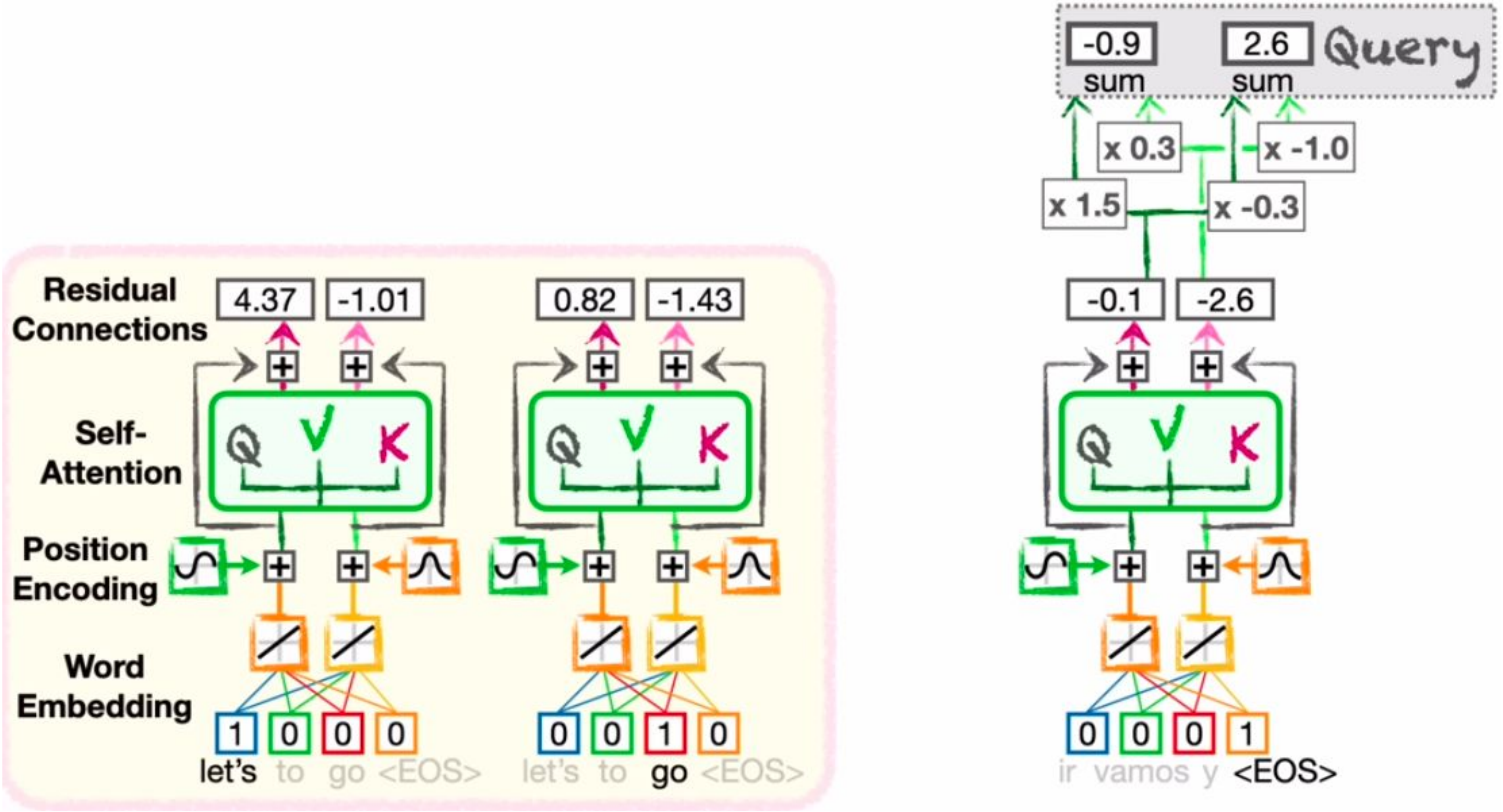
- Keep track of the significant words in the input.



Transformers

Encoder-Decoder Attention

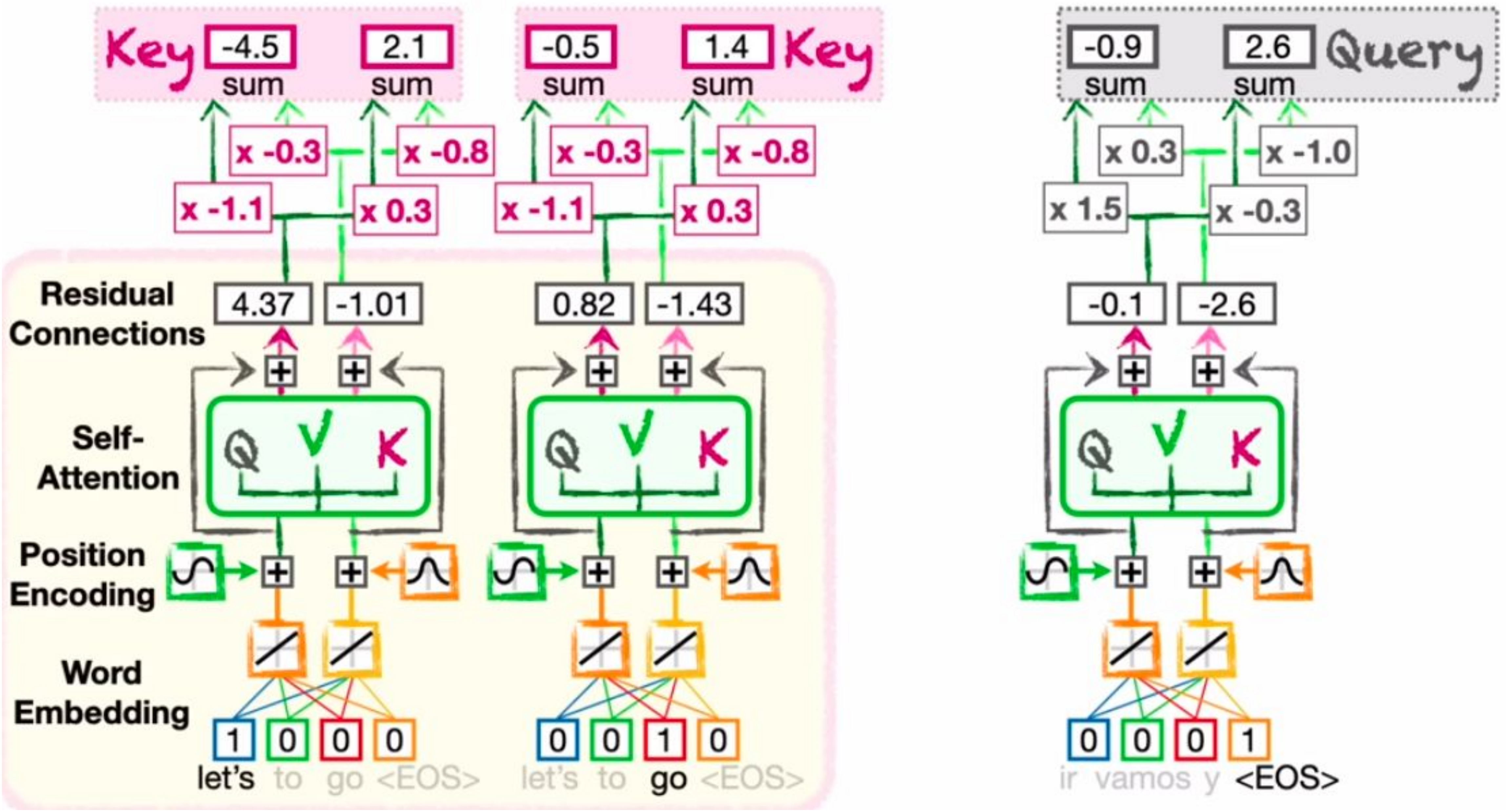
- Keep track of the significant words in the input.
- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.



Transformers

Encoder-Decoder Attention

- Keep track of the significant words in the input.
- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.
    - Then we create **Keys** for each word in the encoder.

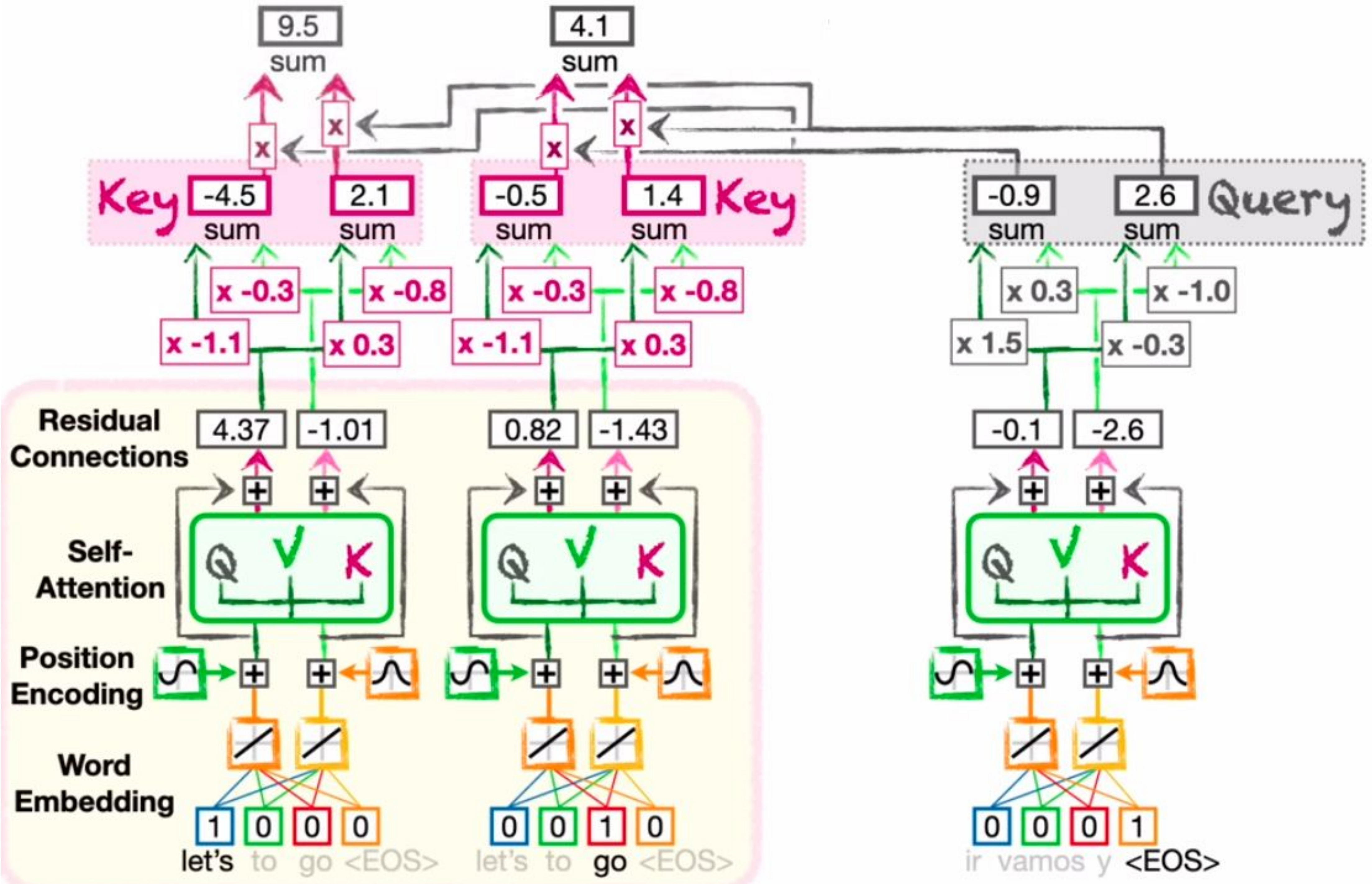


Transformers

Encoder-Decoder Attention

_ Keep track of the significant words in the input.

- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.
    - Then we create **Keys** for each word in the encoder.
        - And we calculate similarities between the <EOS> token and each word in the encoder.

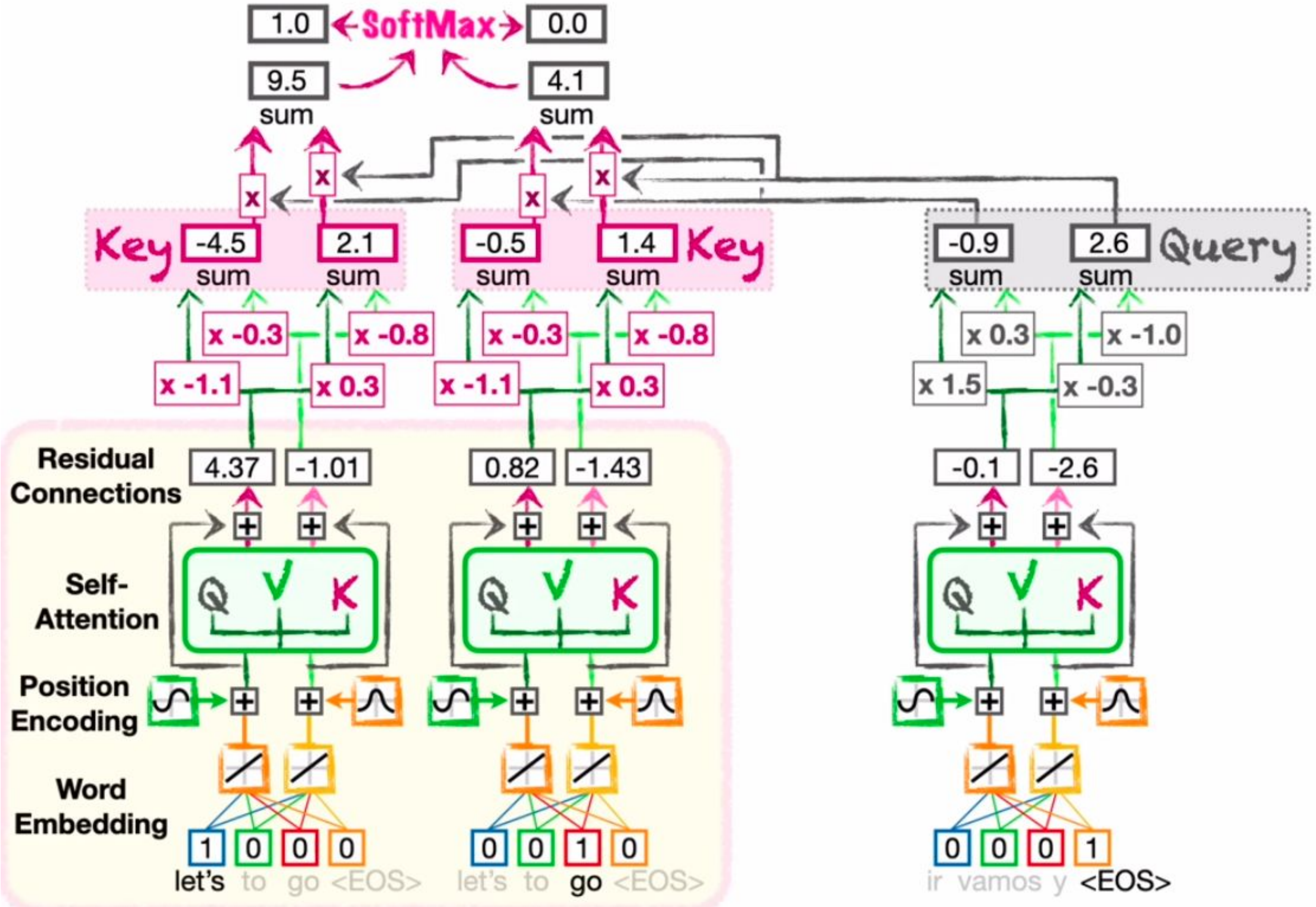


Transformers

Encoder-Decoder Attention

_ Keep track of the significant words in the input.

- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.
  - Then we create **Keys** for each word in the encoder.
    - And we calculate similarities between the <EOS> token and each word in the encoder.
      - Use 100% of “Let’s” and 0% of “go”

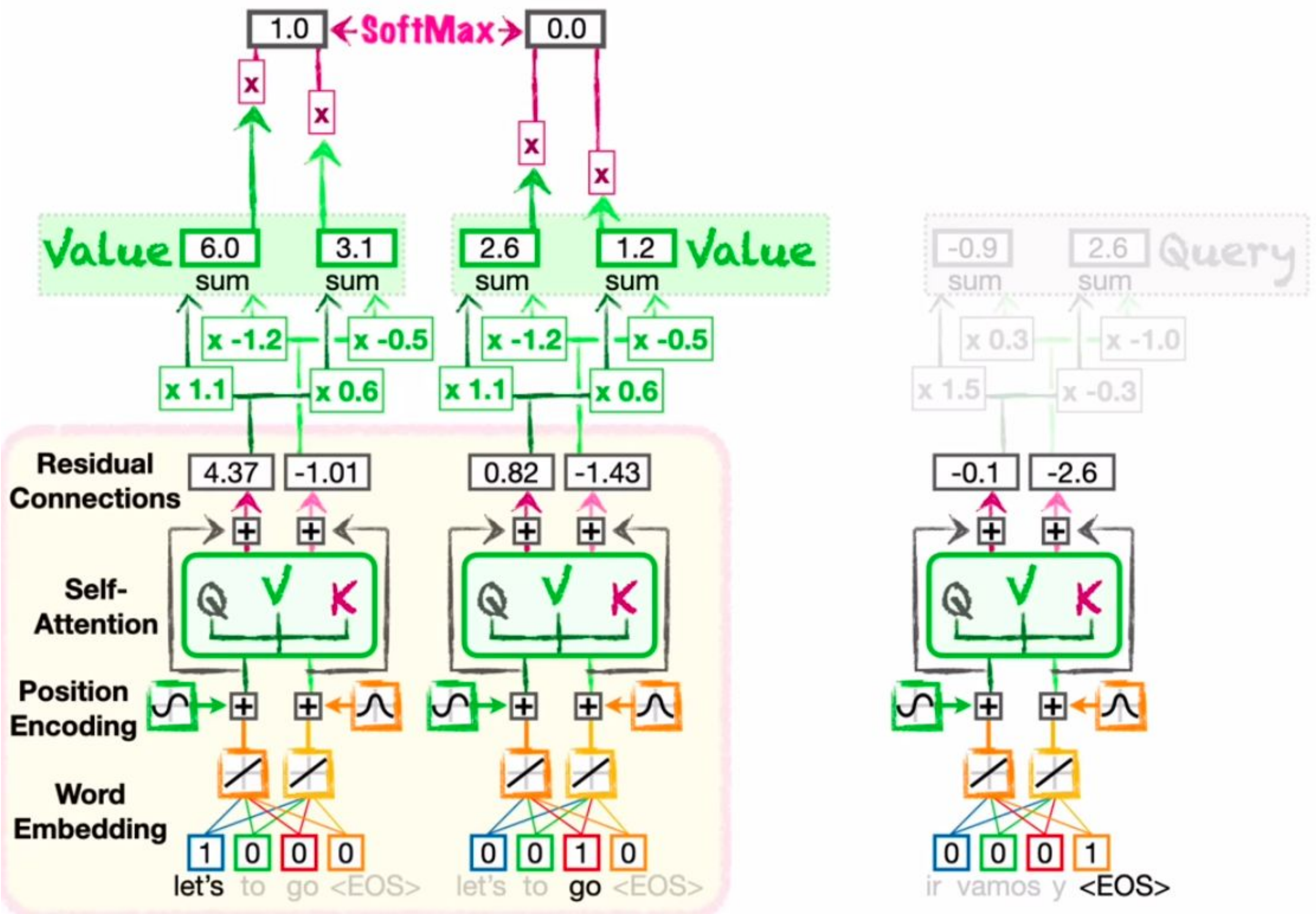


Transformers

Encoder-Decoder Attention

Keep track of the significant words in the input.

- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.
  - Then we create **Keys** for each word in the encoder.
    - And we calculate similarities between the <EOS> token and each word in the encoder.
      - Use 100% of “Let’s” and 0% of “go”
      - And now we calculate **Values** for each input word.
        - Scale values by the Softmax percentage

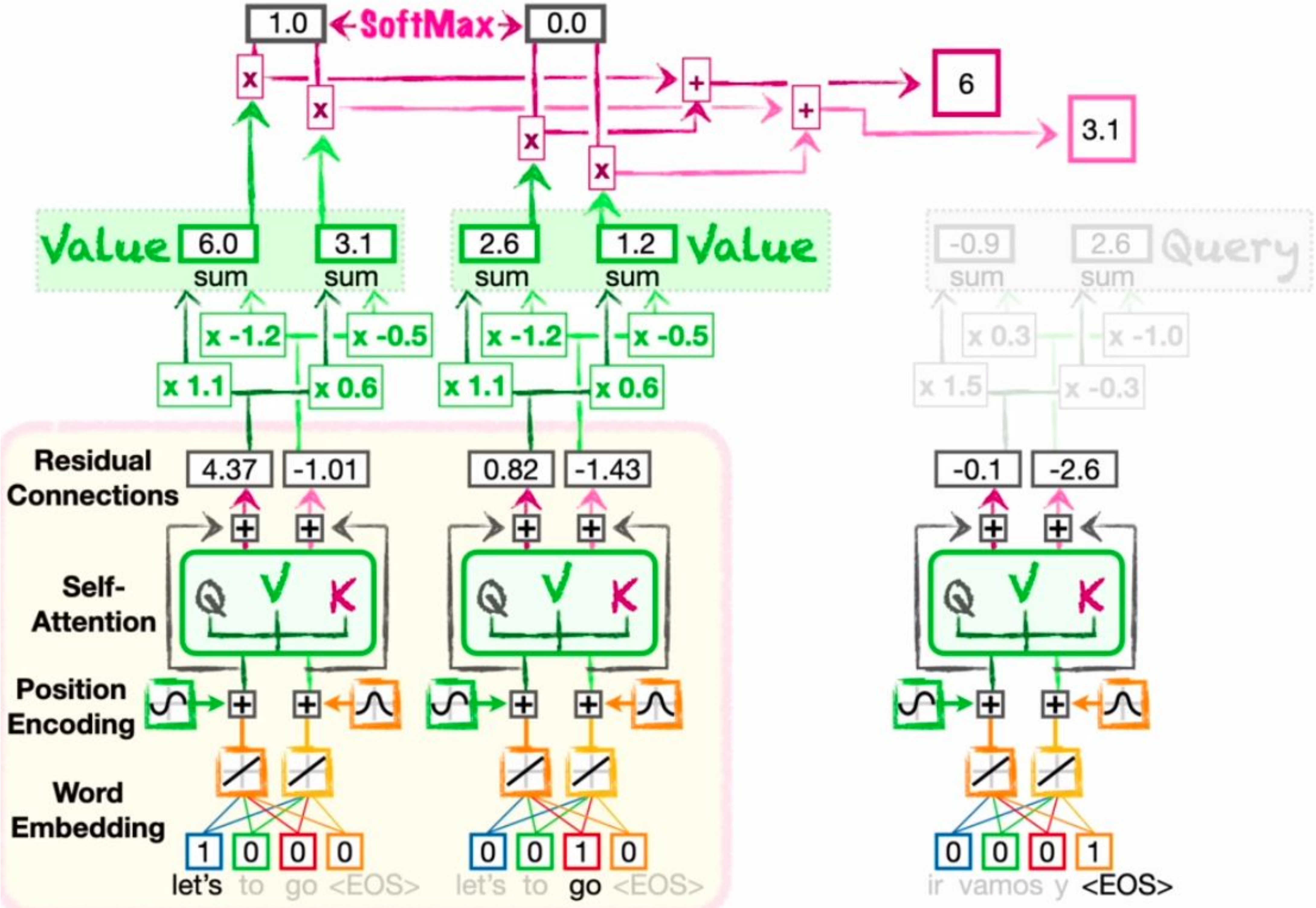


Transformers

Encoder-Decoder Attention

Keep track of the significant words in the input.

- Similarly with Self-Attention, we create new values to represent the **Query** for the <EOS> token in the decoder.
  - Then we create **Keys** for each word in the encoder.
    - And we calculate similarities between the <EOS> token and each word in the encoder.
      - Use 100% of “Let’s” and 0% of “go”
      - And now we calculate **Values** for each input word.
        - Scale values by the Softmax percentage
        - And finally add the scaled values together to get the Encoder-Decoder Attention values.

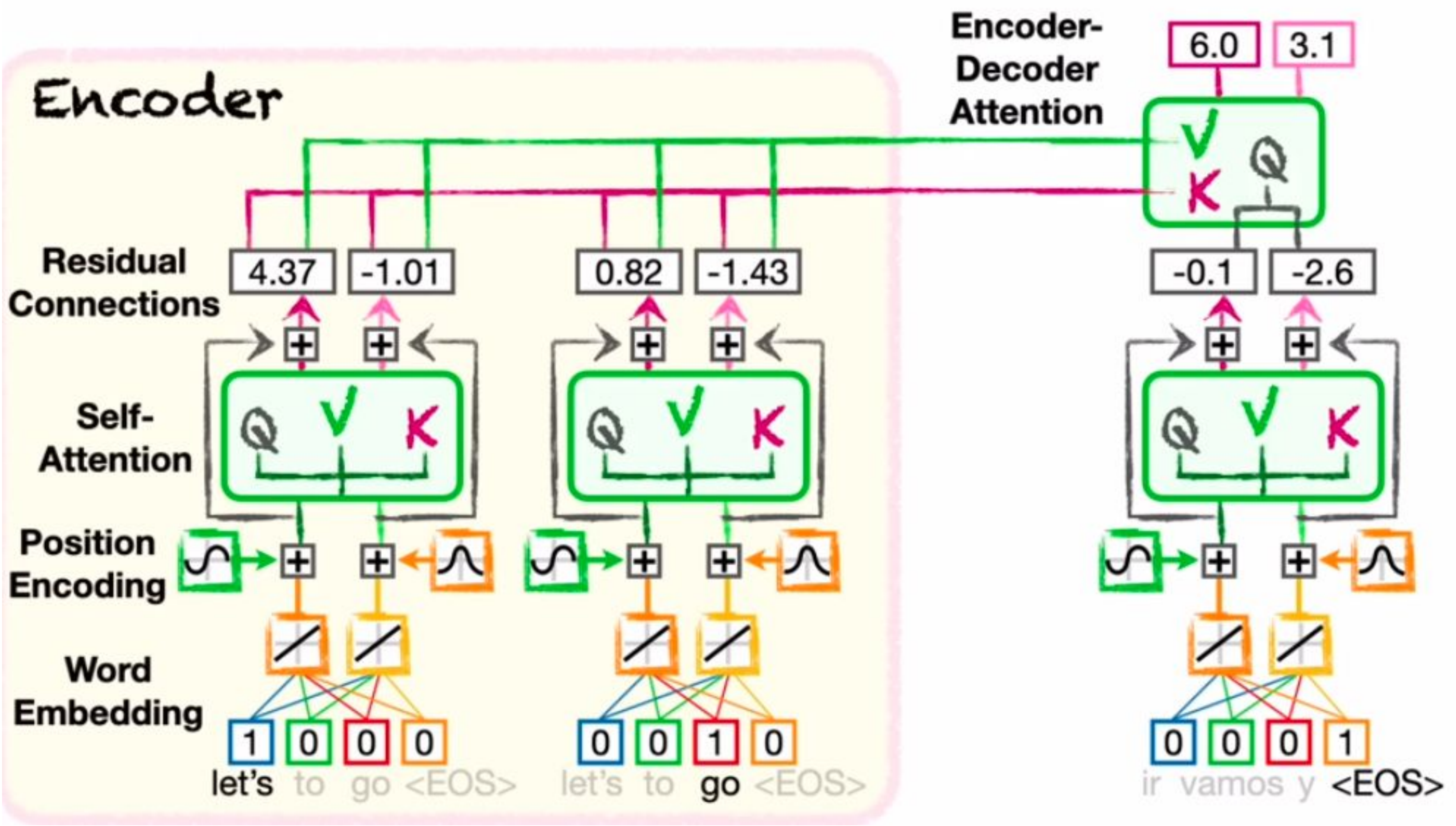


Transformers

Encoder-Decoder Attention

- Keep track of the significant words in the input.

- The weights of encoder-decoder attention are different from the weights of self-attention
  - However, the encoder-decoder attention weights are the same for each word.

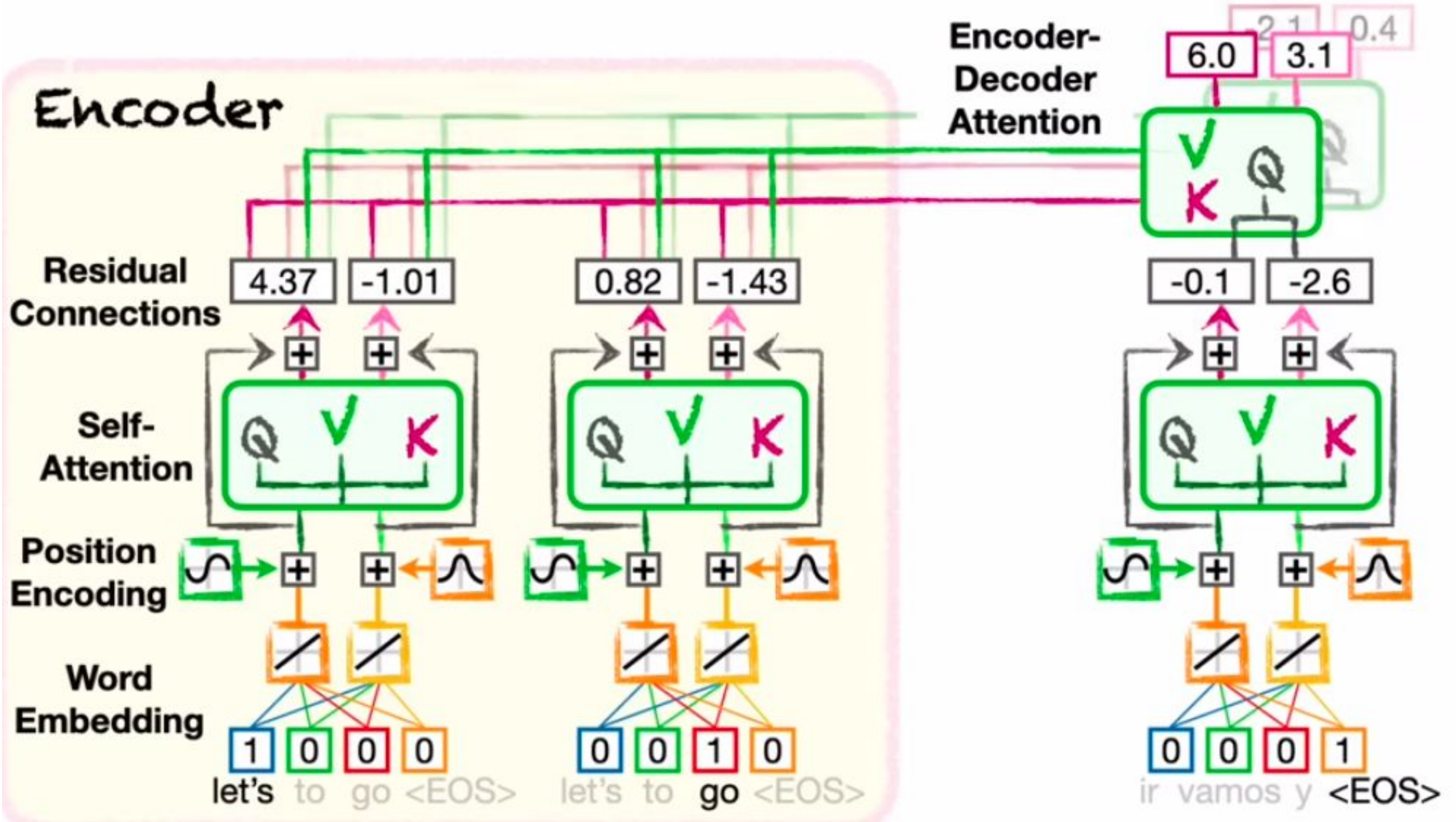


Transformers

Encoder-Decoder Attention

_ Keep track of the significant words in the input.

- The weights of encoder-decoder attention are different from the weights of self-attention
  - However, the encoder-decoder attention weights are the same for each word.
    - Finally, we can stack encoder-decoder attention.

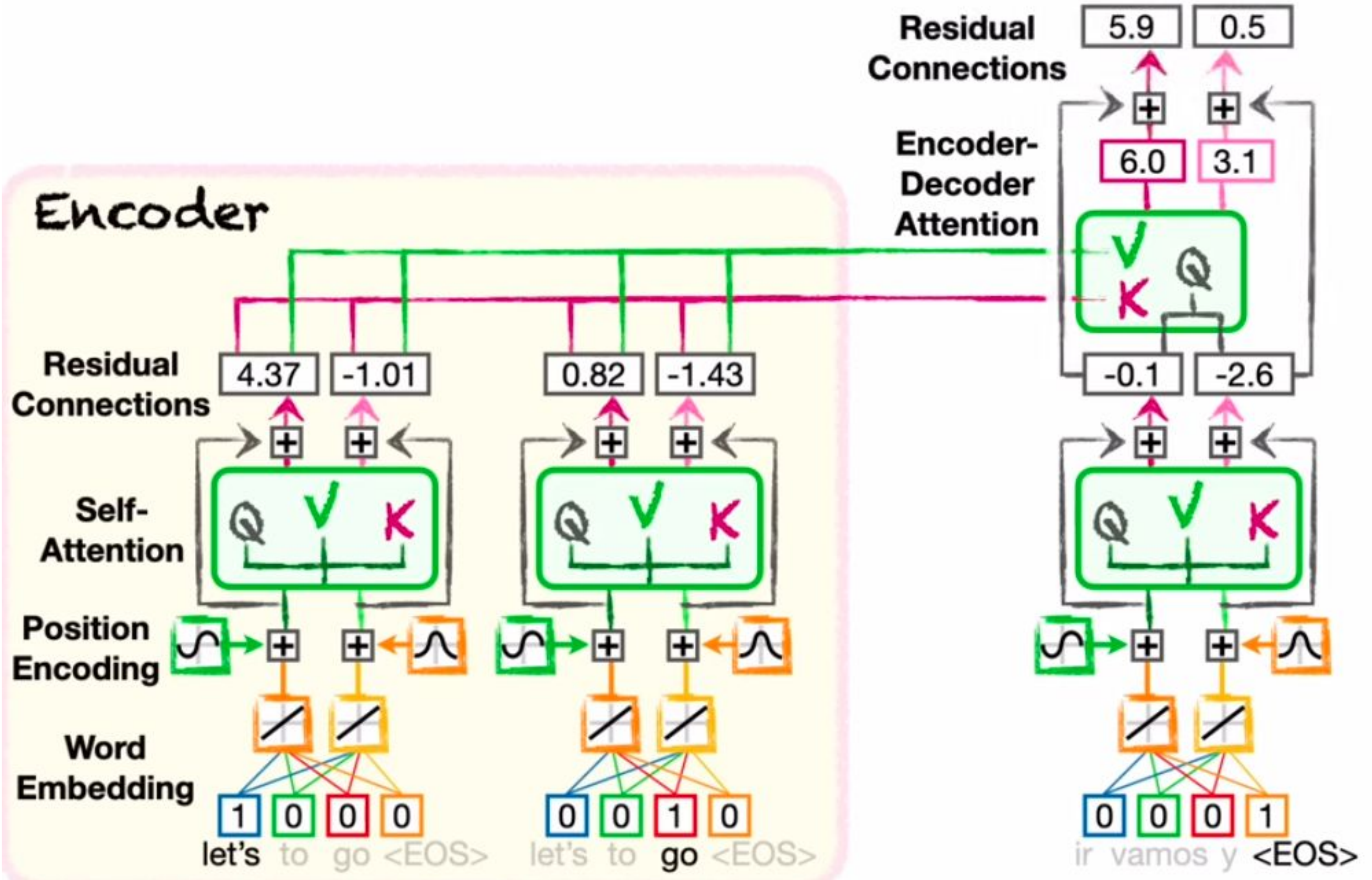


Transformers

Encoder-Decoder Attention

_ Keep track of the significant words in the input.

- The weights of encoder-decoder attention are different from the weights of self-attention
  - However, the encoder-decoder attention weights are the same for each word.
    - And we add another set of residual connections.

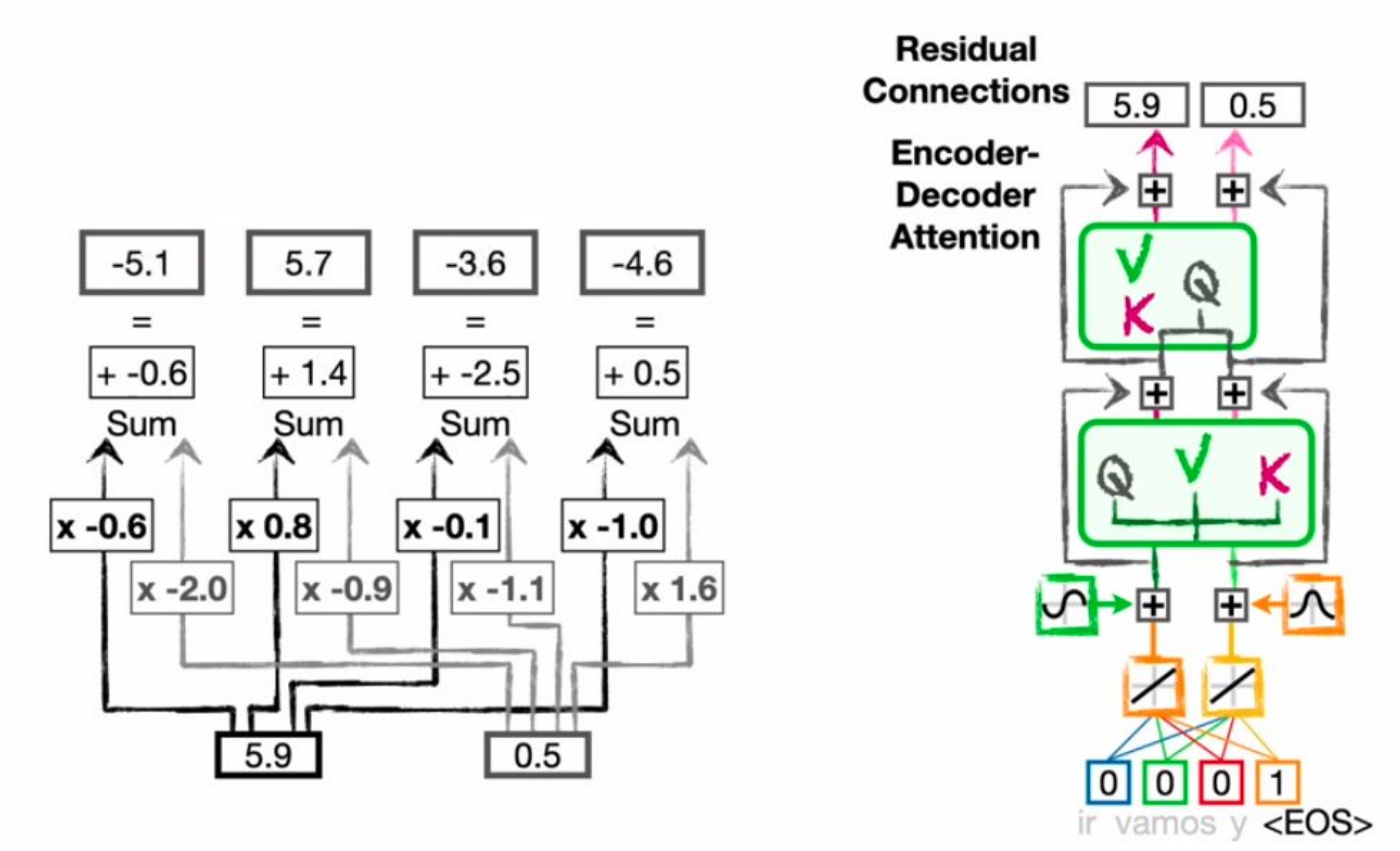


Transformers

Inference

_ Fully connected layer

- The encoder-decoder attention values are given as input to a fully connected layer.

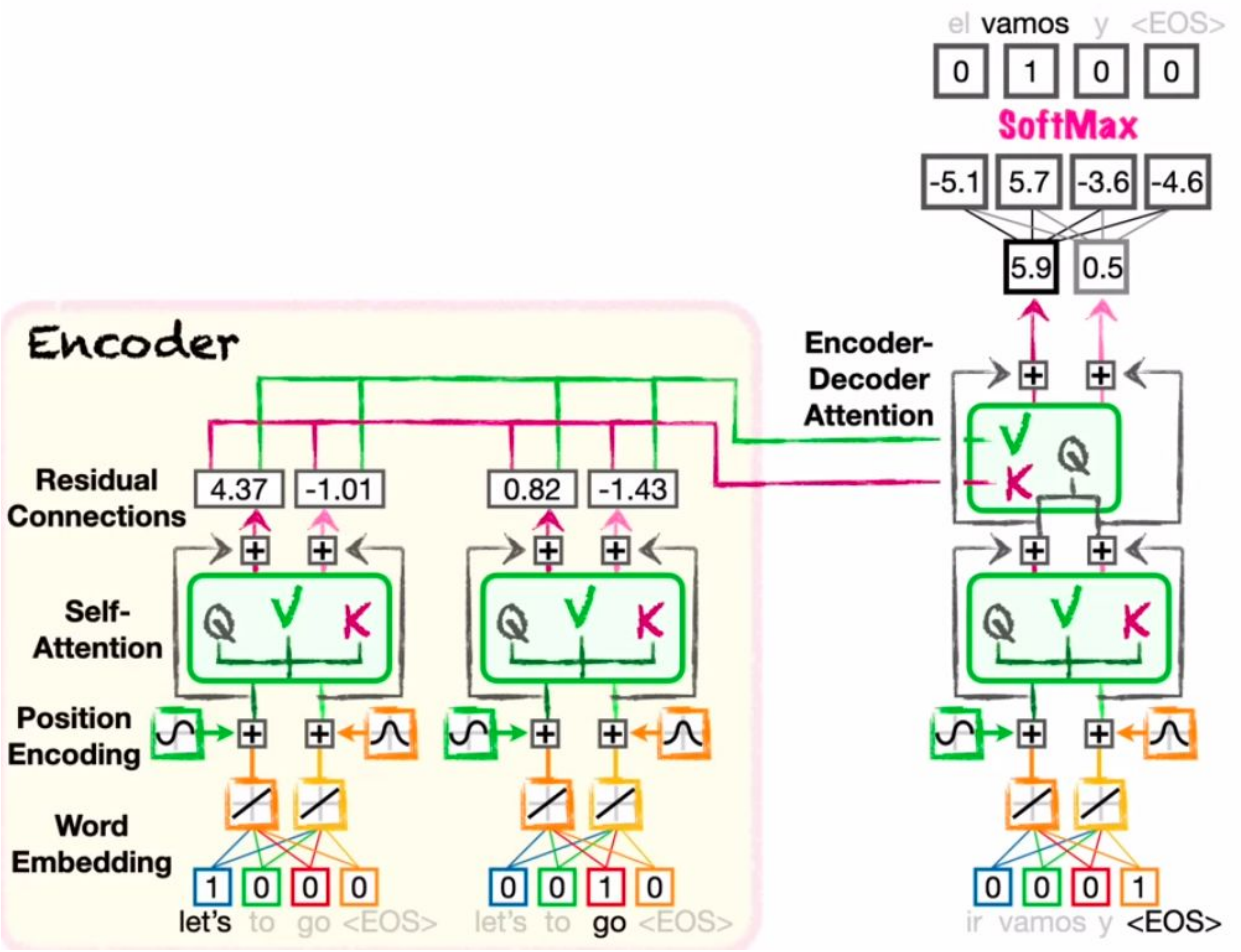


Transformers

Inference

_ Fully connected layer

- The encoder-decoder attention values are given as input to a fully connected layer.
  - And a Softmax layer to select the most probable word.

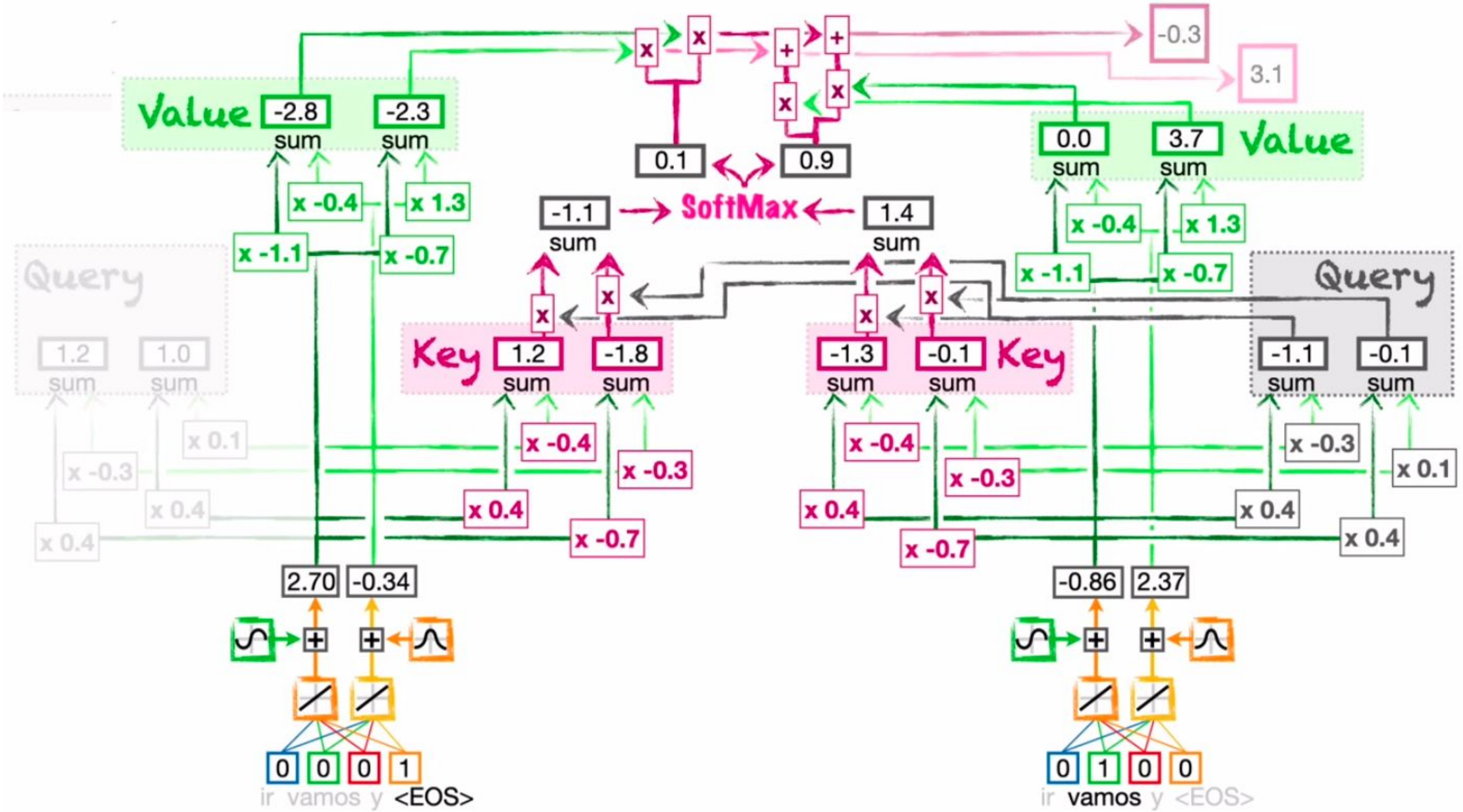


Transformers

Inference

_ It will stop only when <EOS> is the selected token.

- Now, the “vamos” token is selected.

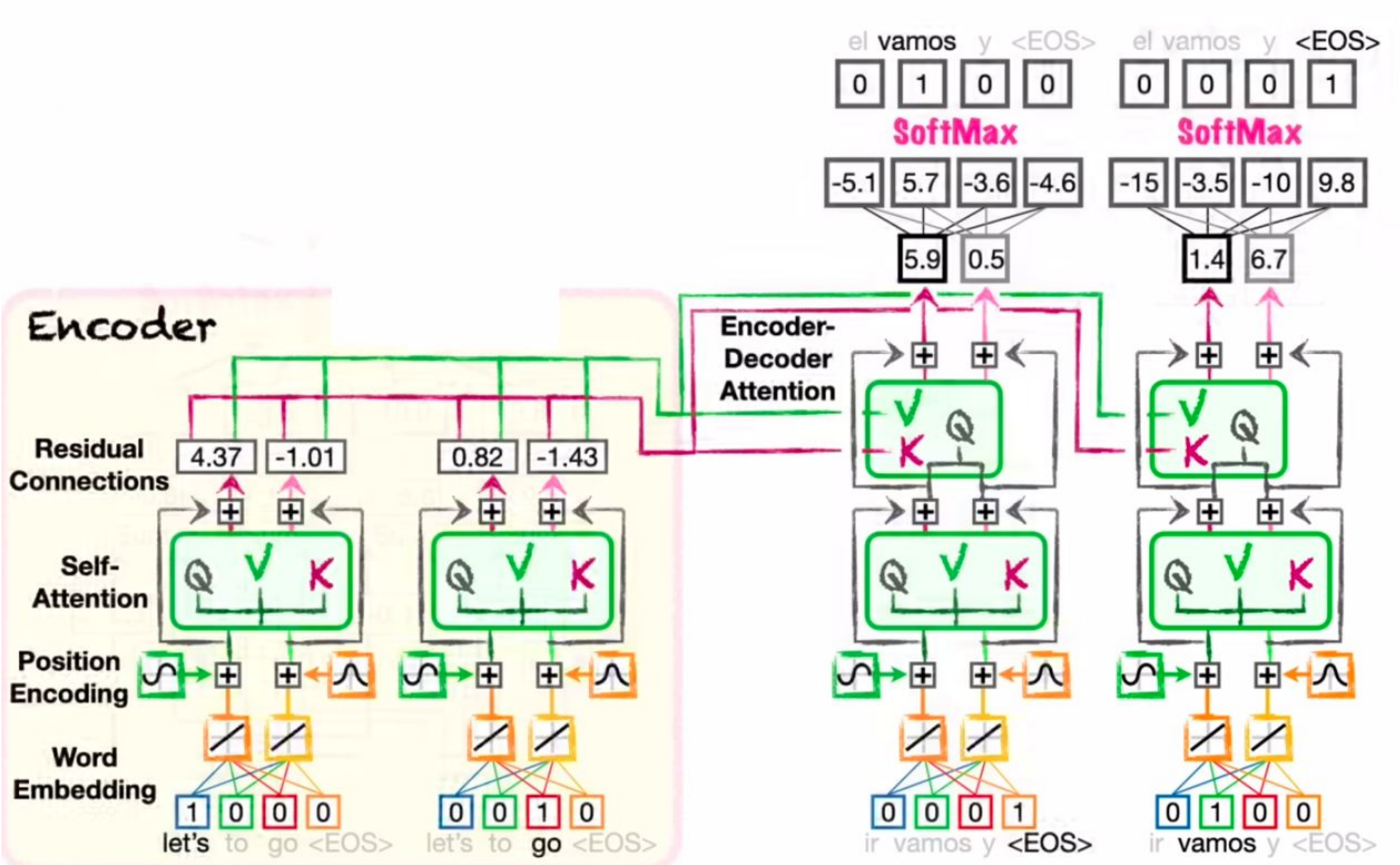


Transformers

Inference

_ It will stop only when <EOS> is the selected token.

- Now, the “vamos” token is selected.
    - And the same process is repeated.

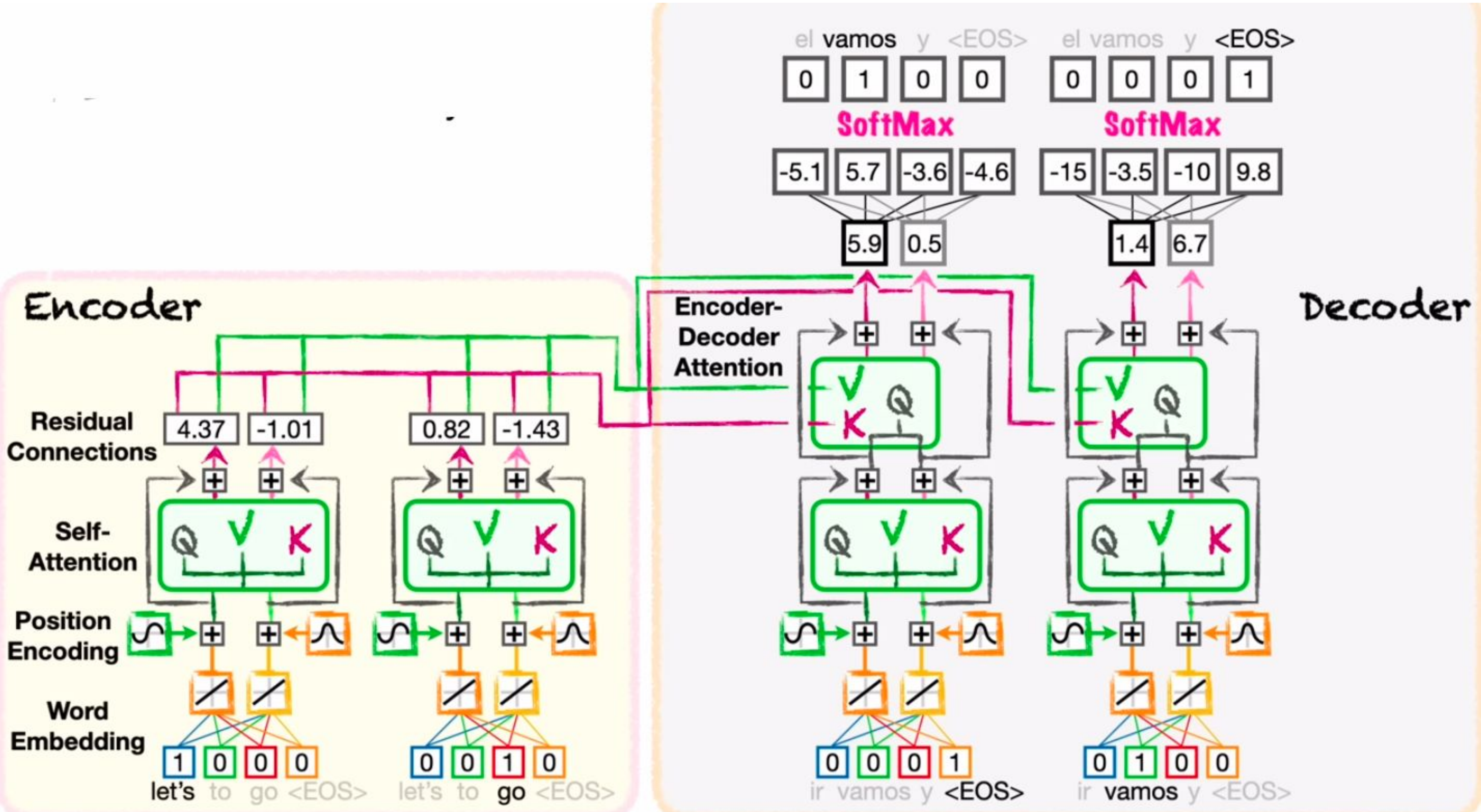


Transformers

Inference

_ It will stop only when <EOS> is the selected token.

- Now, the “vamos” token is selected.
  - And the same process is repeated.
    - And the <EOS> token is selected.



Pretraining → LLMs

It is all about context!

_ What can we learn from reconstructing the input?

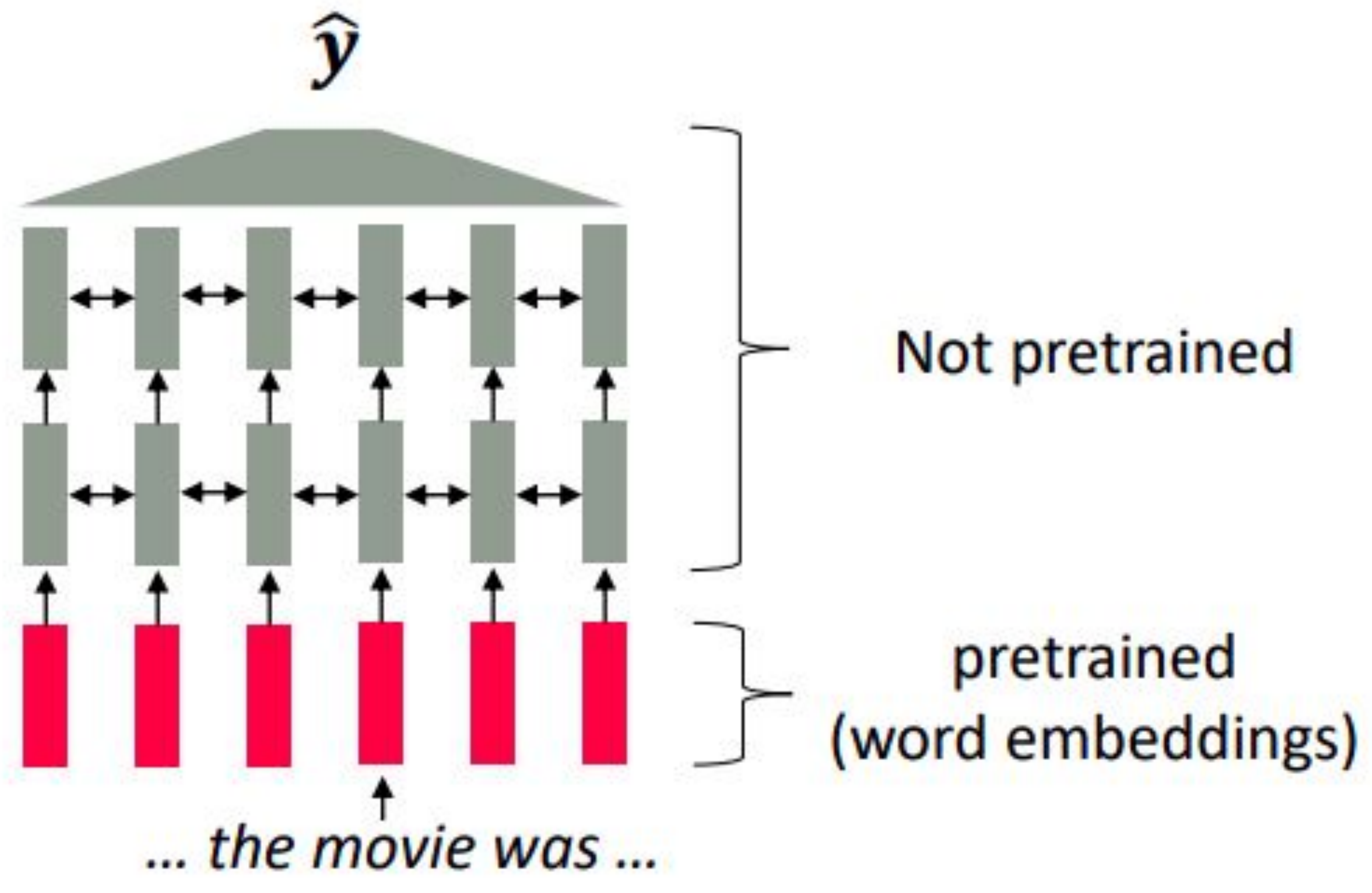
- “I put ___ fork down on the table.”
- “The woman walked across the street, checking for traffic over ___ shoulder.”
- “I went to the ocean to see the fish, turtles, seals, and _____.”
- “Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.”
- “Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the ____. ”
- “I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____”

Pretraining → LLMs

It is all about context!

_ Pretrained word embeddings (Past NLP)

- Start with pretrained word embeddings (no context!)
  - Learn how to incorporate context in an LSTM or Transformer while training on the task.
- Possible problems:
  - The training data we have for our downstream task must be sufficient to teach all contextual aspects of language
  - Most of the parameters in our network are randomly initialized!

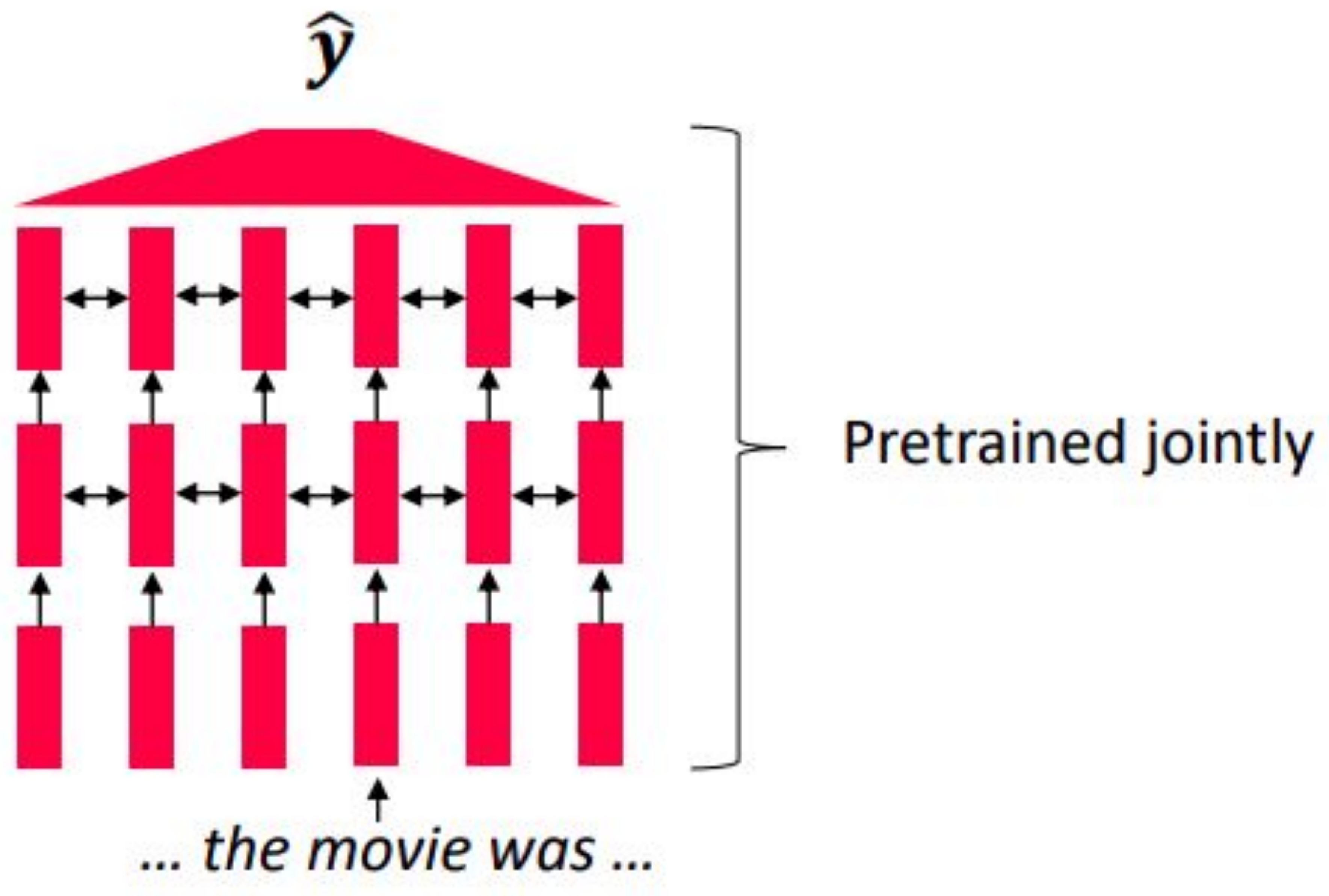


Pretraining → LLMs

It is all about context!

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via pretraining.
    - This has been exceptionally effective at building strong:
        - Representations of language
        - Parameter initializations for strong NLP models
            - The model has learned how to represent entire sentences through pretraining

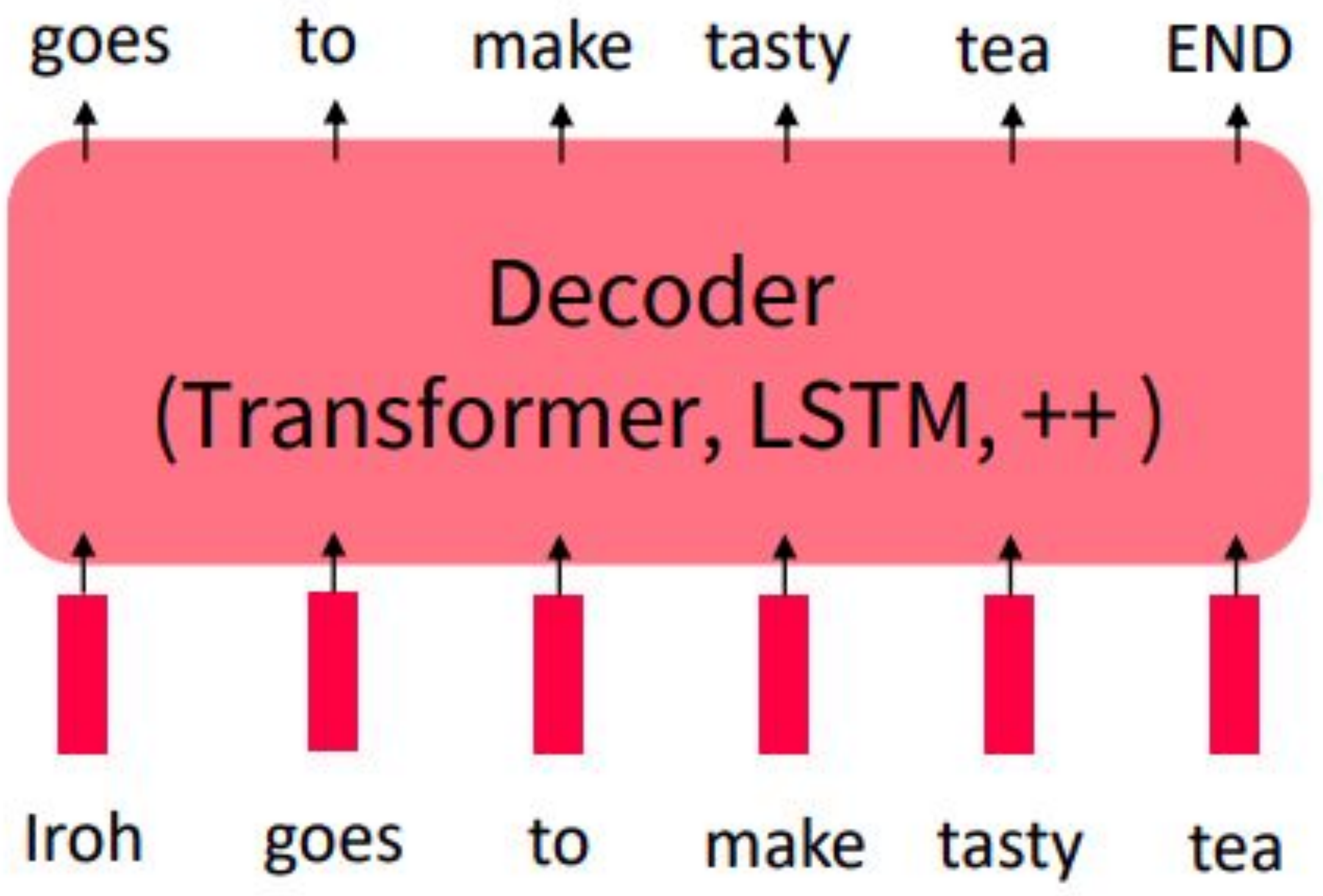


Pretraining → LLMs

It is all about context!

Recall the language modeling task:

- Model the probability distribution over words given their past contexts.
  - There's lots of data for this!
  - Pretraining through language modeling:
    - Train a neural network to perform language modeling on a large amount of text
    - Save the network parameters

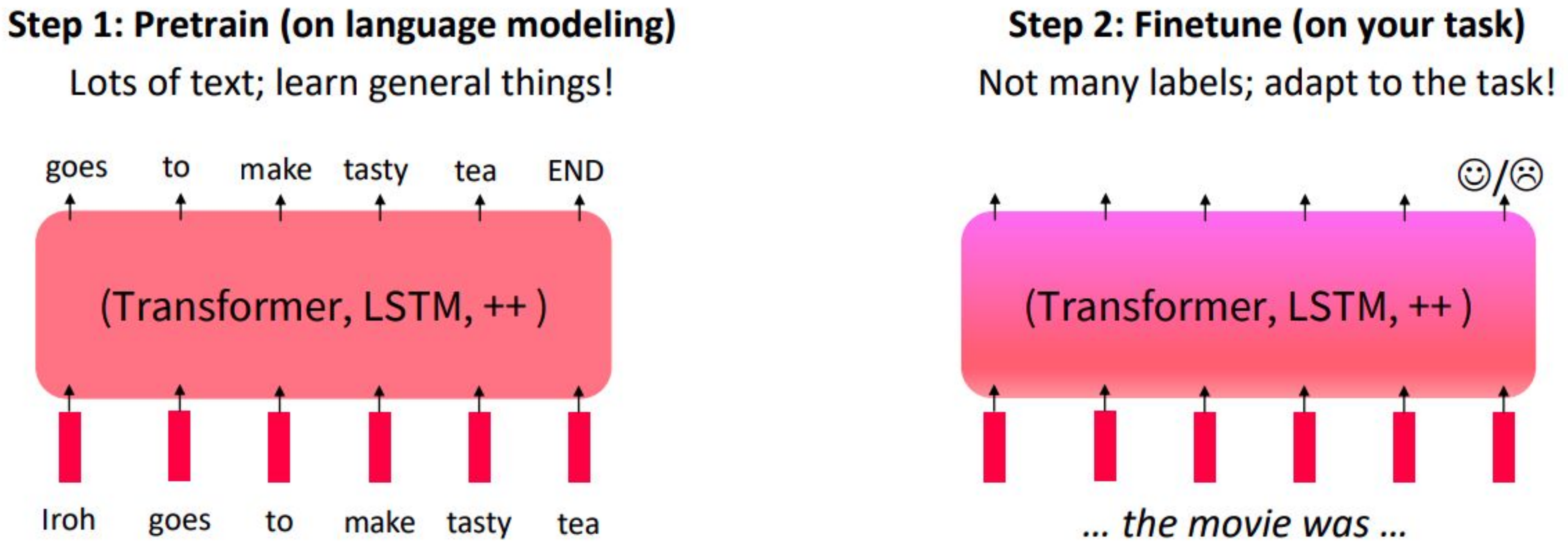


Pretraining → LLMs

It is all about context!

Pretraining/Finetune paradigm:

- Pretraining can improve NLP applications by serving as parameter initialization.

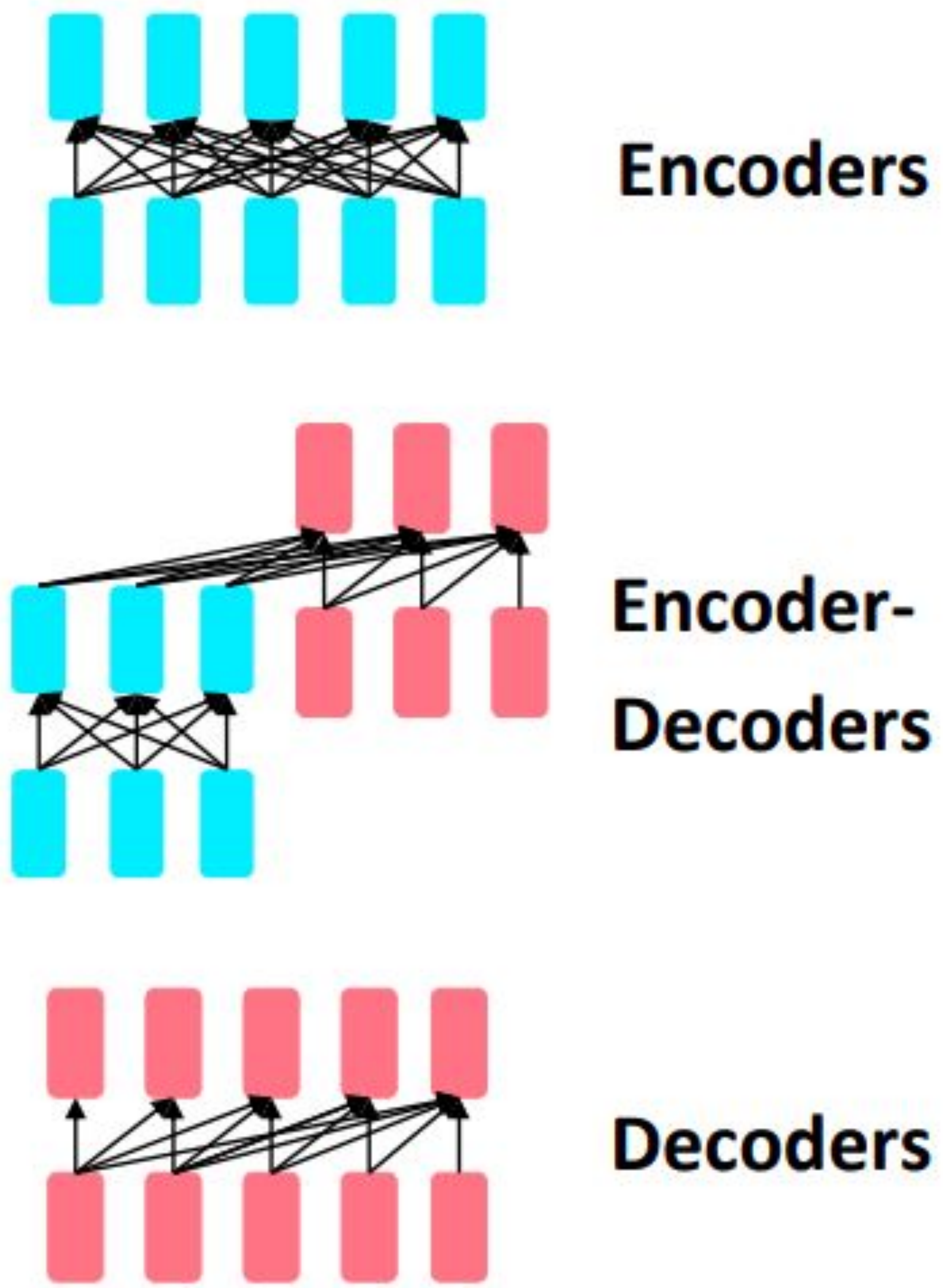


Pretraining → LLMs

It is all about context!

_ Types of Pretraining:

- Encoders
  - Gets bidirectional context, and can condition on future!
- Encoder-Decoders
  - One part is bidirectional, but the other is unidirectional
- Decoders
  - Cannot condition on future

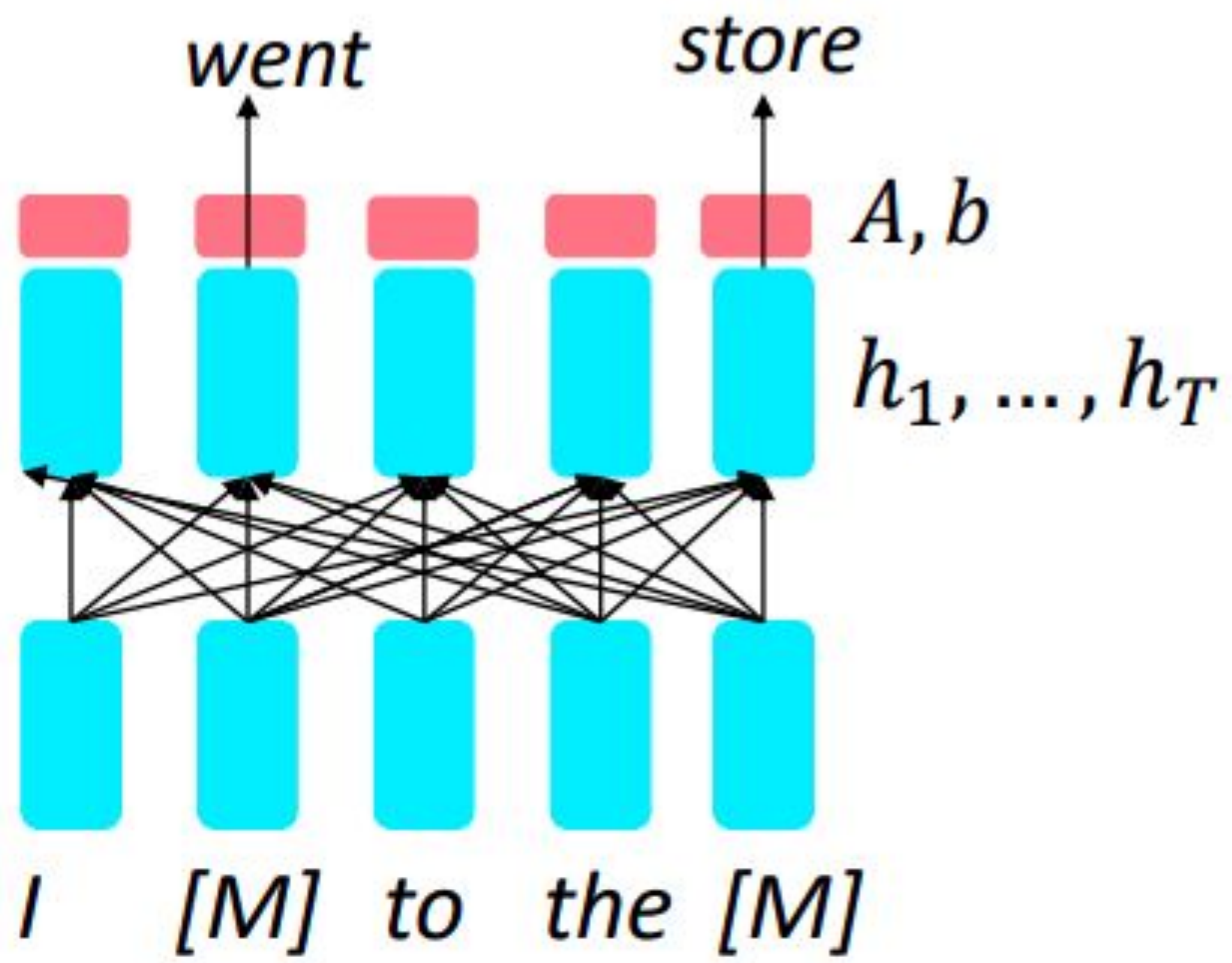


Pretraining → LLMs

It is all about context!

Pretraining encoders

- Encoders get bidirectional context, so they cannot do language modeling!
  - Solution: replace some fraction of words in the input with a special [MASK] token, and predict these words.
    - This is called Masked Language Model

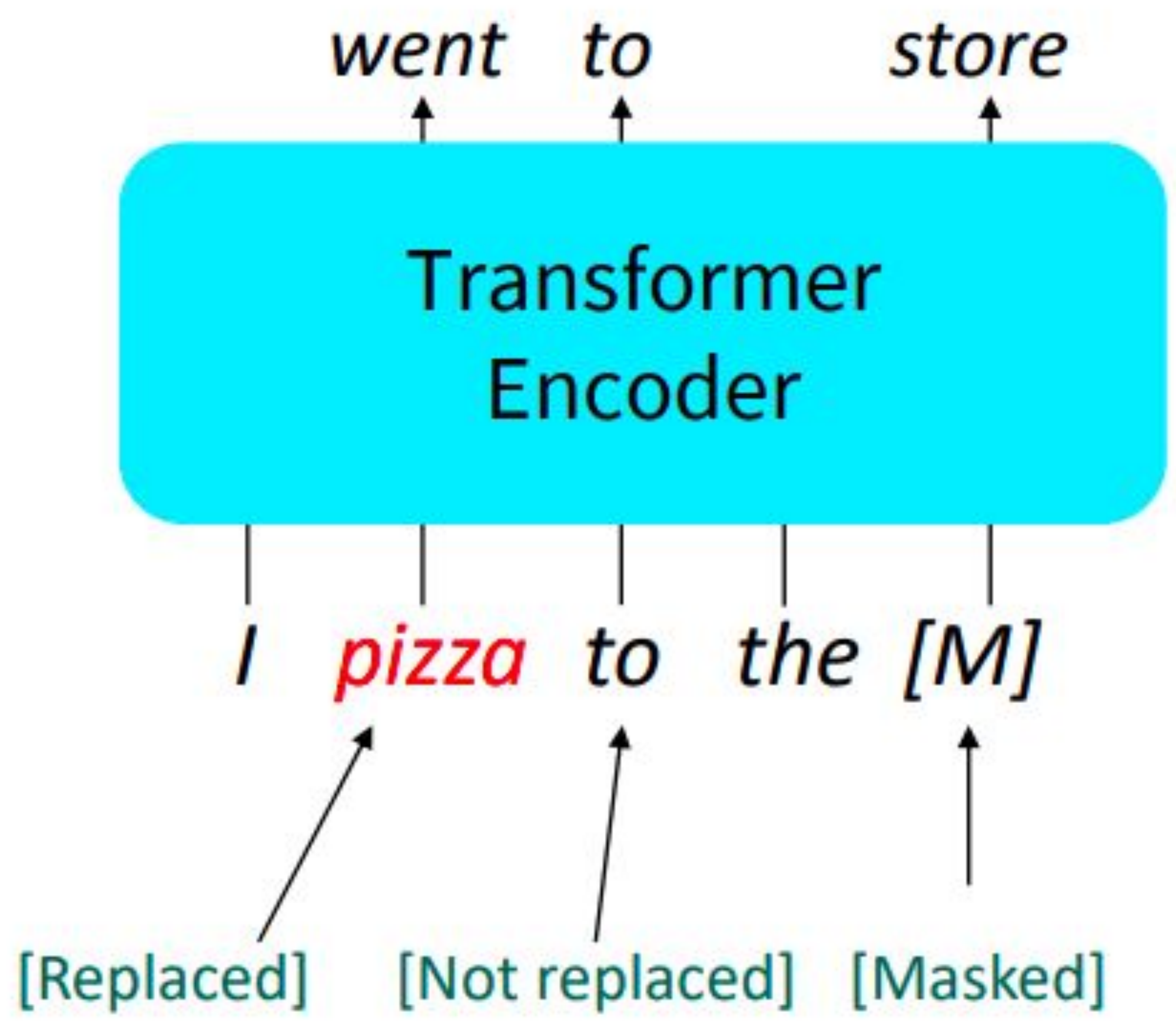


Pretraining → LLMs

It is all about context!

Pretraining encoders

- BERT (Bidirectional Encoder Representations from Transformers)
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
    - Why? Does not let the model get complacent and not build strong representations of non-masked words
    - No masks are seen at fine-tuning time



Pretraining → LLMs

It is all about context!

_ Pretraining encoders

- BERT (Bidirectional Encoder Representations from Transformers)
  - Two models were released:
    - BERT-base with 12 layers, 768-dim hidden states, 12 attention heads, 110 million params
    - BERT-large with 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params
  - Trained on:
    - BooksCorpus (800 million words)
    - English Wikipedia (2,500 million words)
  - Pretrained with 64 TPU chips for a total of 4 days
    - TPUs are special tensor operation acceleration hardware
    - Finetuning is practical and common on a single GPU
      - "Pretrain once, finetune many times."

Pretraining → LLMs

It is all about context!

_ Pretraining encoders

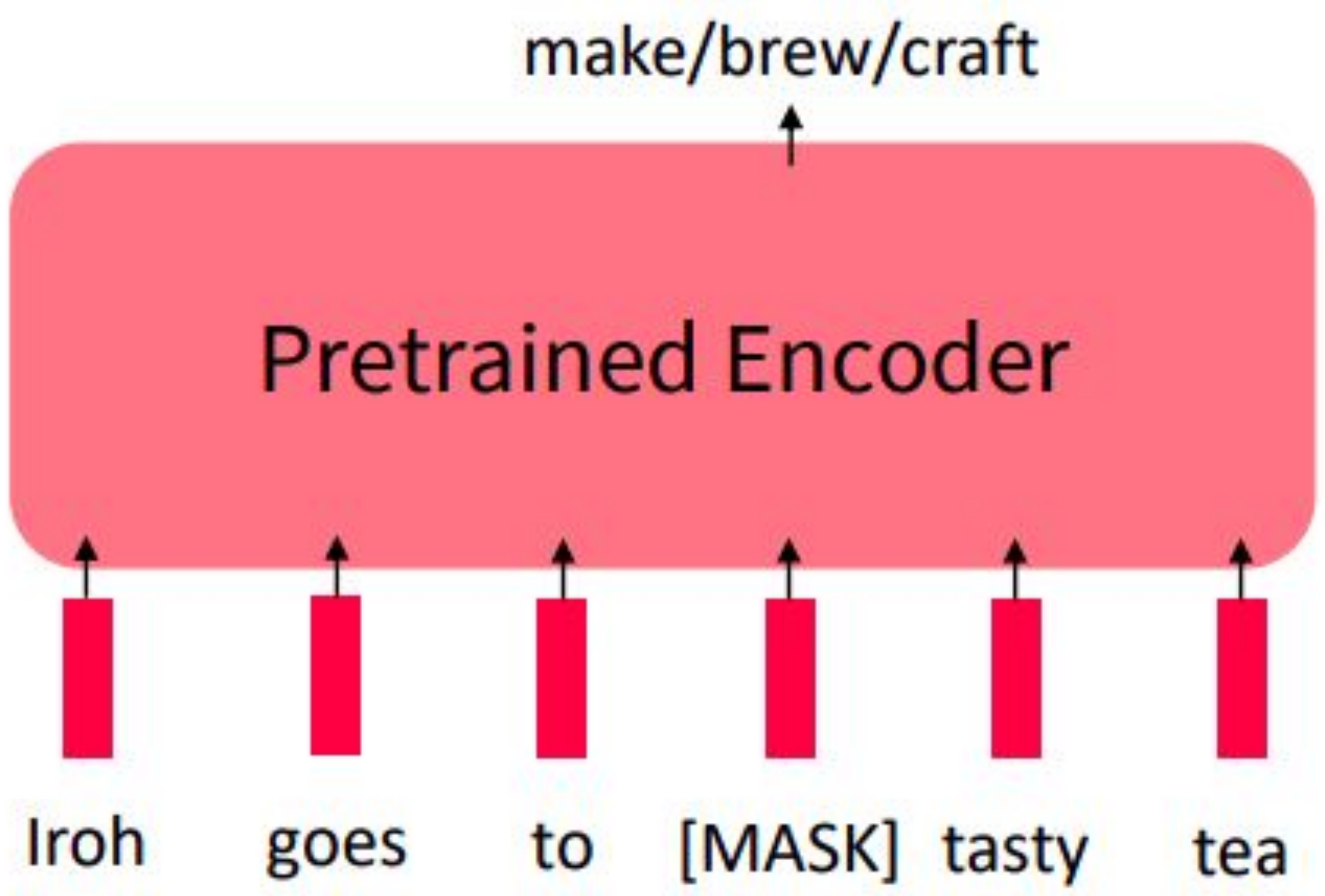
- BERT (Bidirectional Encoder Representations from Transformers)
  - BERT was massively popular
    - Finetuning BERT led to new state-ofthe-art results on a broad range of tasks
      - QQP: Quora Question Pairs (detect paraphrase questions)
      - QNLI: natural language inference over question answering data
      - SST-2: sentiment analysis
      - CoLA: corpus of linguistic acceptability (detect whether sentences are grammatical.)
      - STS-B: semantic textual similarity
      - MRPC: microsoft paraphrase corpus
      - RTE: a small natural language inference corpus

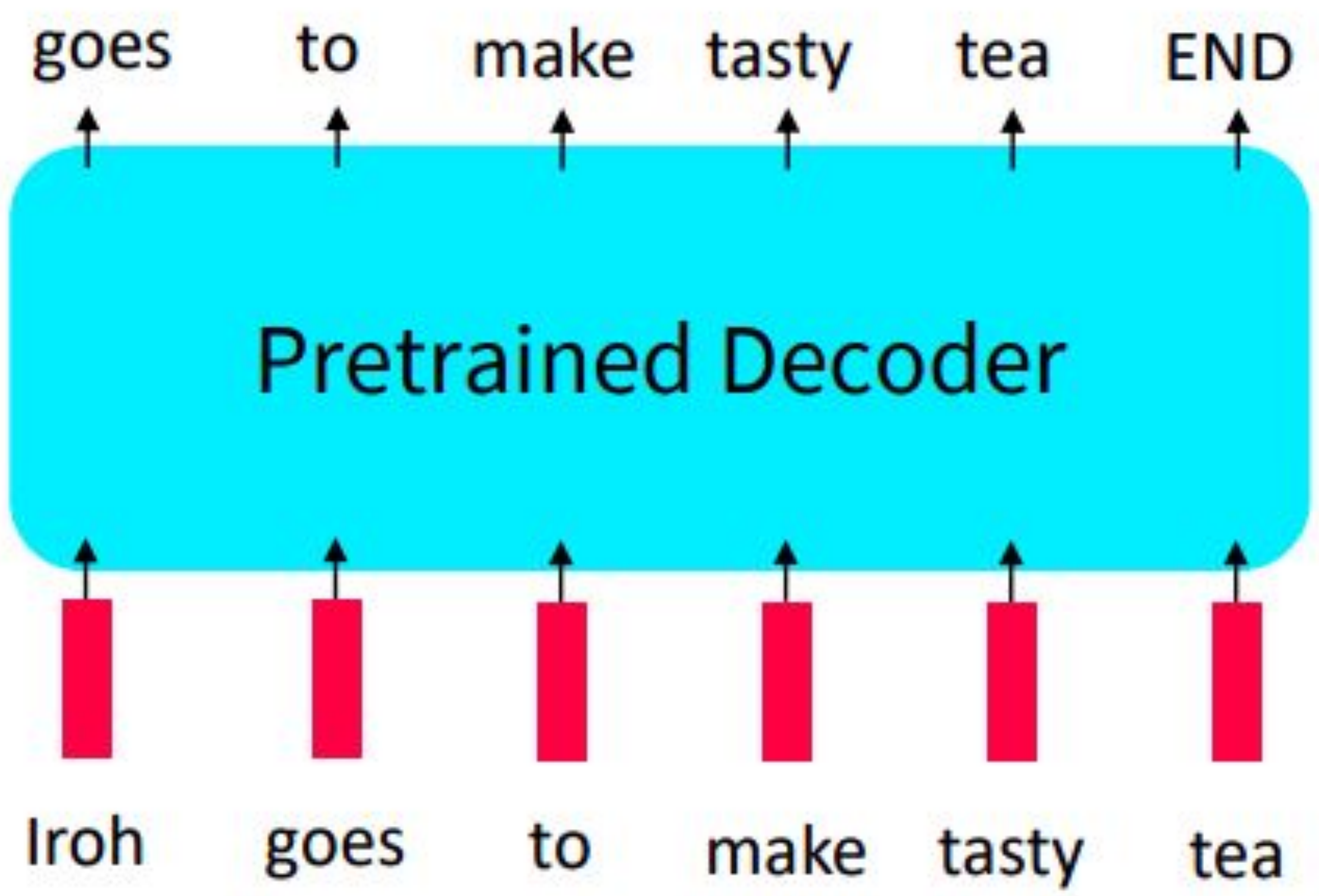
Pretraining → LLMs

It is all about context!

_ Pretraining encoders

- Limitations of pretrained encoders
  - If your task involves generating sentences, consider using a pretrained decoders
    - BERT and other pretrained encoders do not naturally lead to nice autoregressive (1-word-at-a-time) generation methods.



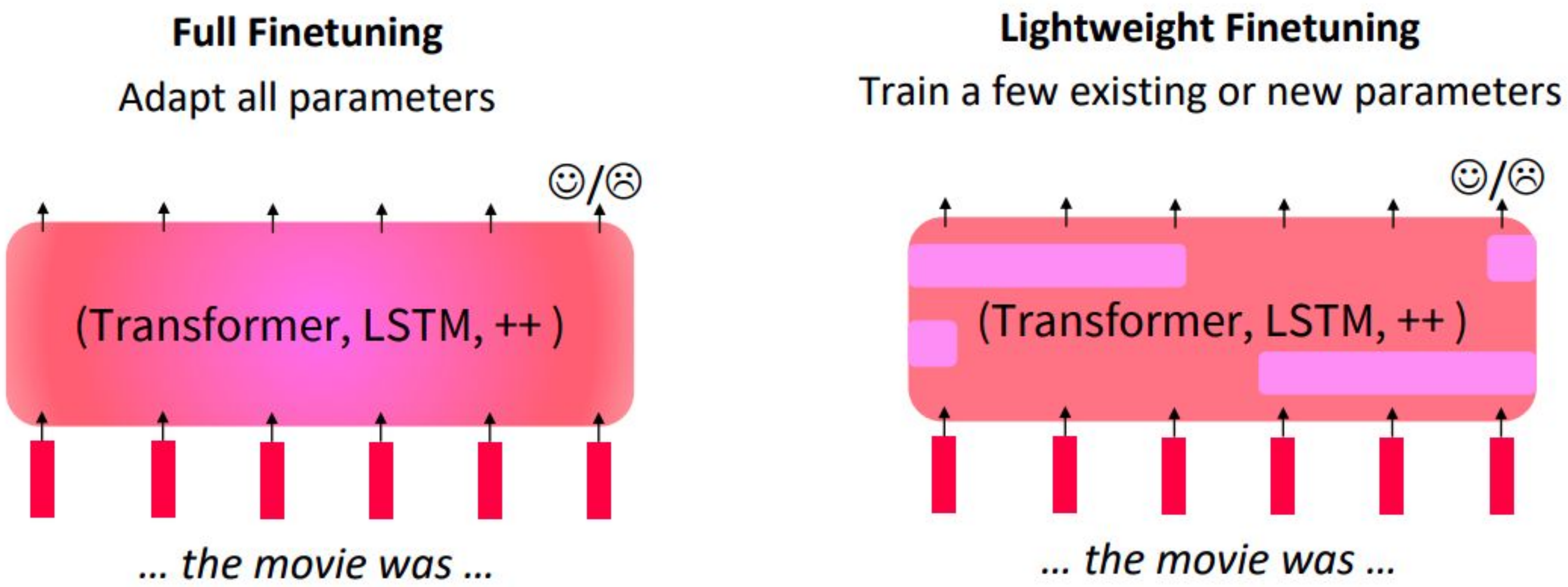


Pretraining → LLMs

It is all about context!

_ Full Finetuning vs. Parameter-Efficient Finetuning

- Limitations of pretrained encoders
  - Finetuning every parameter in a pretrained model works well, but is memory-intensive
    - Lightweight finetuning methods adapt pretrained models in a constrained way.
      - Leads to less overfitting and/or more efficient finetuning and inference.

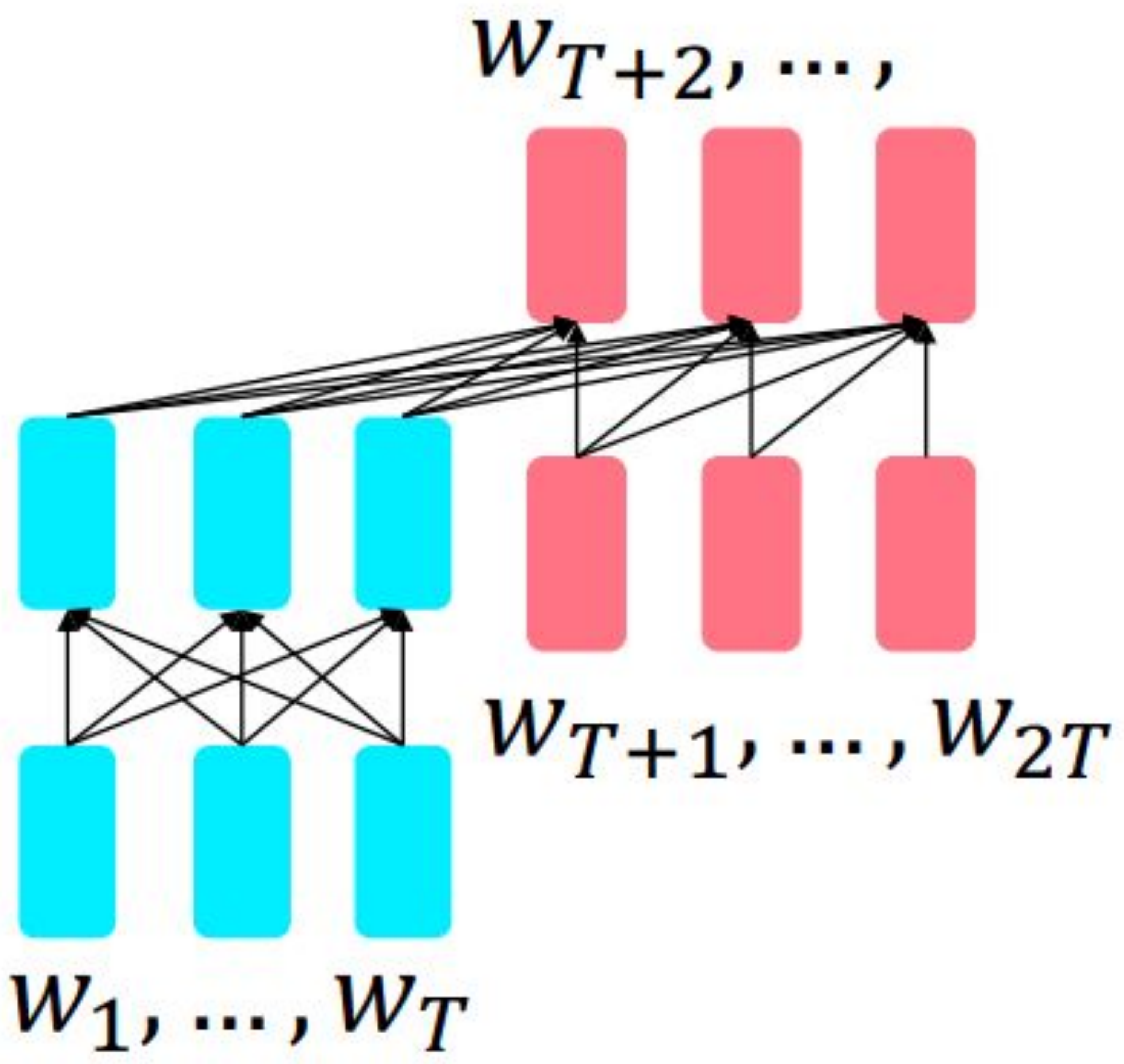


Pretraining → LLMs

It is all about context!

_ Pretrained encoder-decoder

- For encoder-decoders, we could do something like language modeling.
  - But, where a prefix of every input is provided to the encoder and is not predicted.
    - The encoder portion benefits from bidirectional context; the decoder portion is used to train the whole model through language modeling.

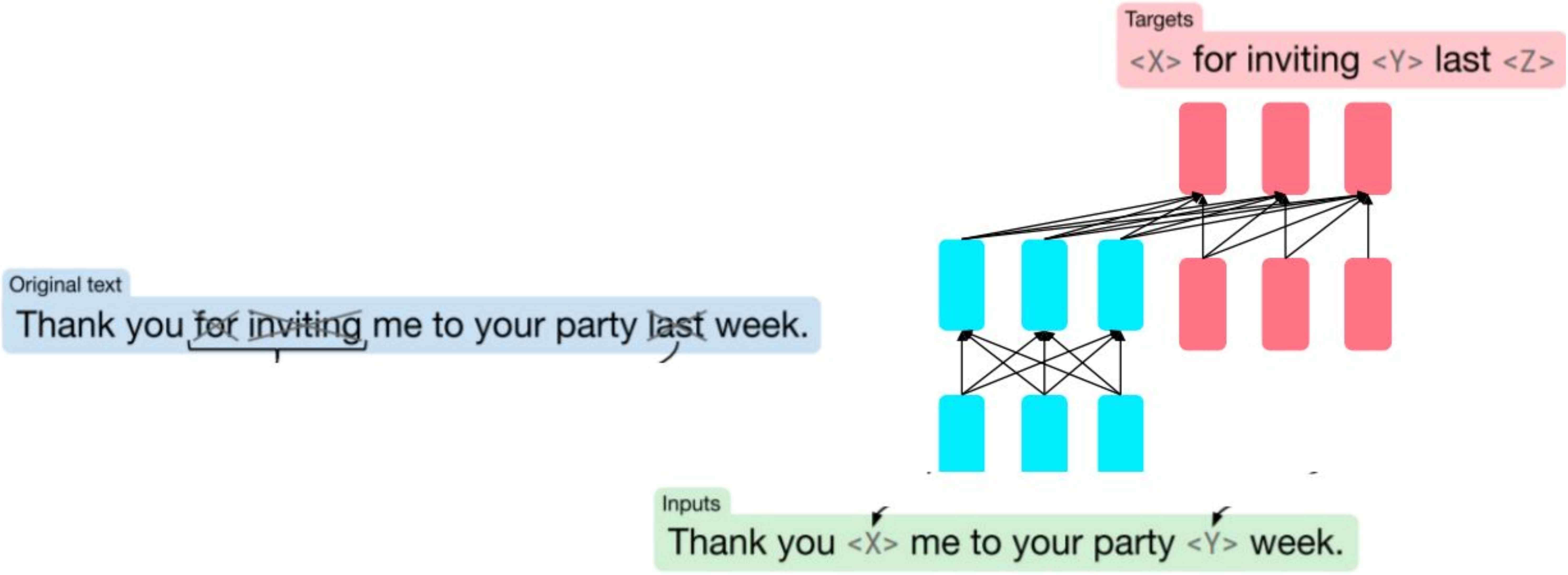


Pretraining → LLMs

It is all about context!

Pretrained encoder-decoder

- The most popular model within this category is T5.
  - Replace different-length spans from the input with unique placeholders, and decode out the spans that were removed!

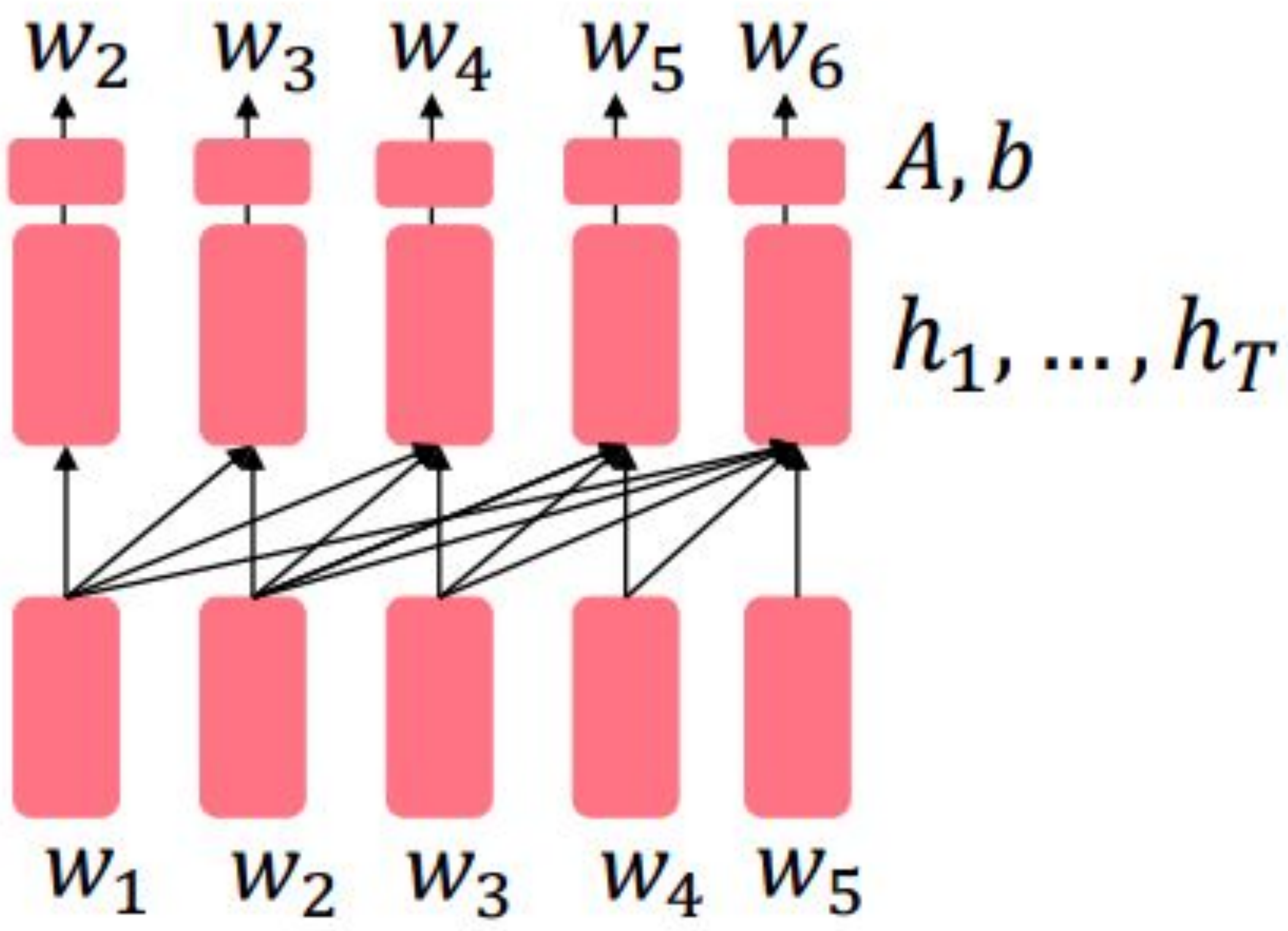


Pretraining → LLMs

It is all about context!

_ Pretrained decoders

- It is natural to pretrain decoders as language models and then use them as text generators
  - This is helpful in tasks where the output is a sentence with a vocabulary like that at pretraining time



Pretraining → LLMs

It is all about context!

_ Pretrained decoders

- Generative Pretrained Transformer (GPT)
    - 2018's GPT was a big success in pretraining a decoder
        - Transformer decoder with 12 layers, 117M parameters
        - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers
        - Trained on:
            - BooksCorpus: over 7000 unique books
            - Contains long spans of contiguous text, for learning long-distance dependencies

Pretraining → LLMs

It is all about context!

_ Pretrained decoders

- Generative Pretrained Transformer (GPT)
    - How do we format inputs to our decoder for finetuning tasks?
        - Natural Language Inference
            - Label pairs of sentences as entailing/contradictory/neutral
                - Premise: The man is in the doorway
                - Hypothesis: The person is near the door

Pretraining → LLMs

It is all about context!

_ Pretrained decoders

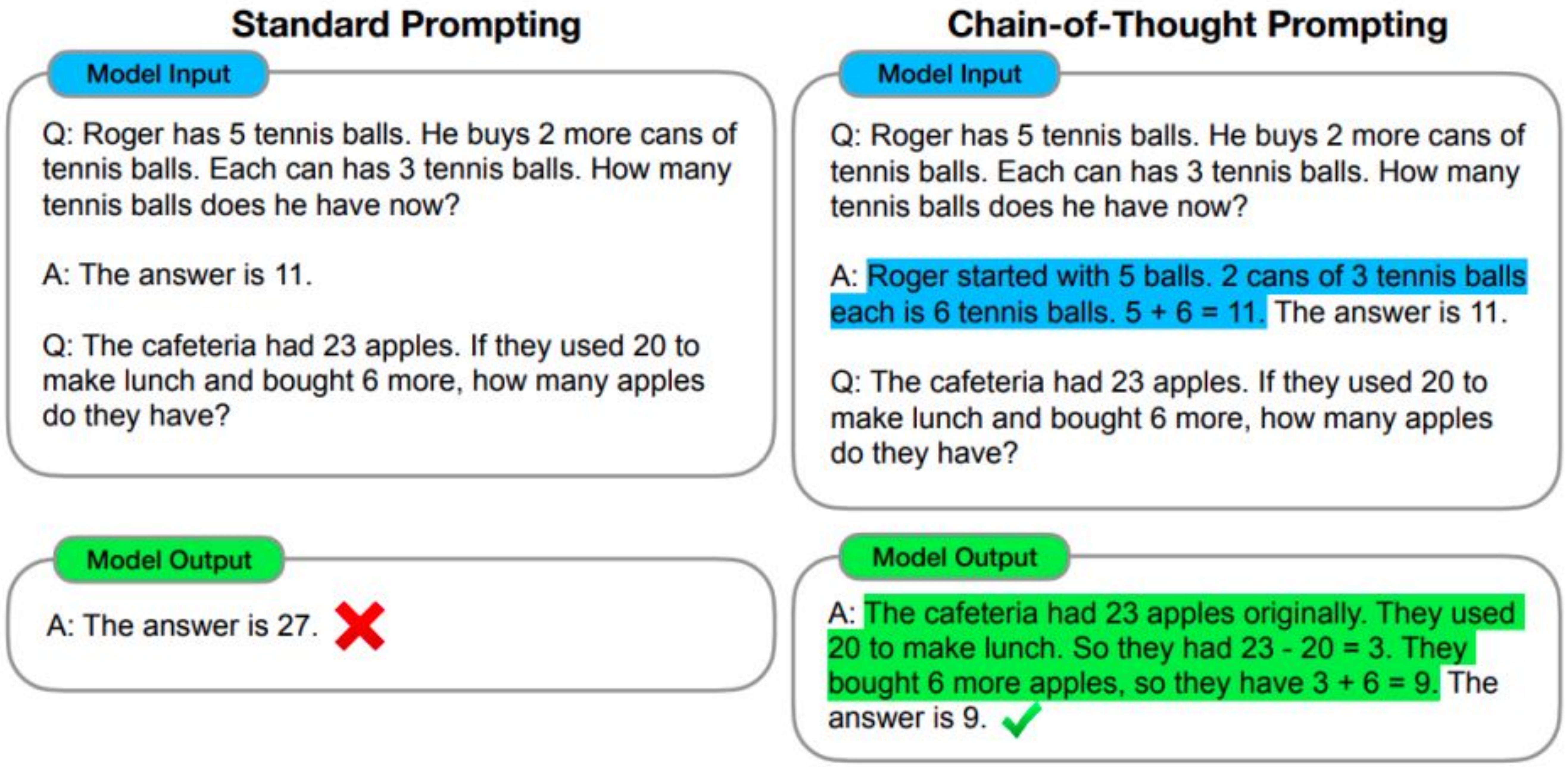
- Generative Pretrained Transformer (GPT2)
  - We mentioned how pretrained decoders can be used in their capacities as language models
    - GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Pretraining → LLMs

It is all about context!

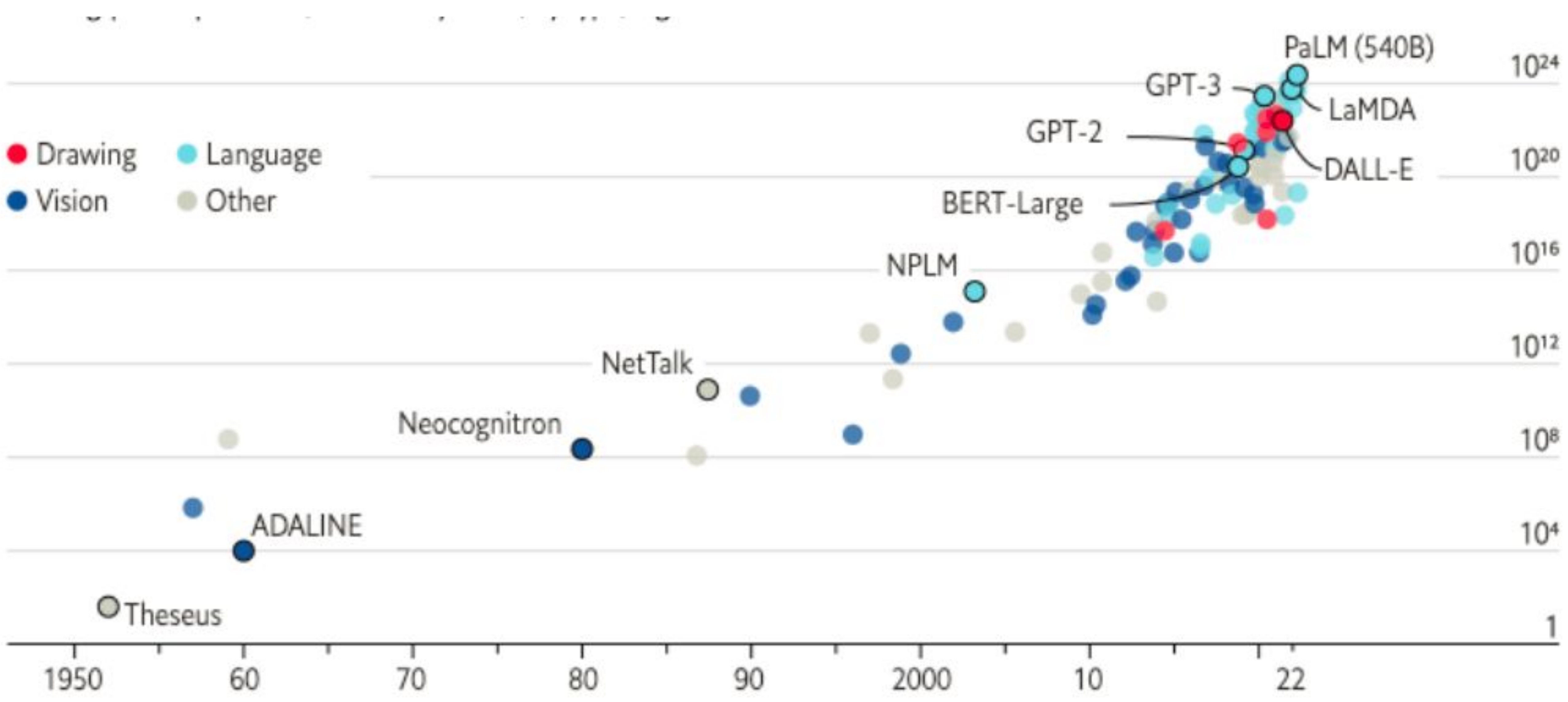
_ Pretrained decoders

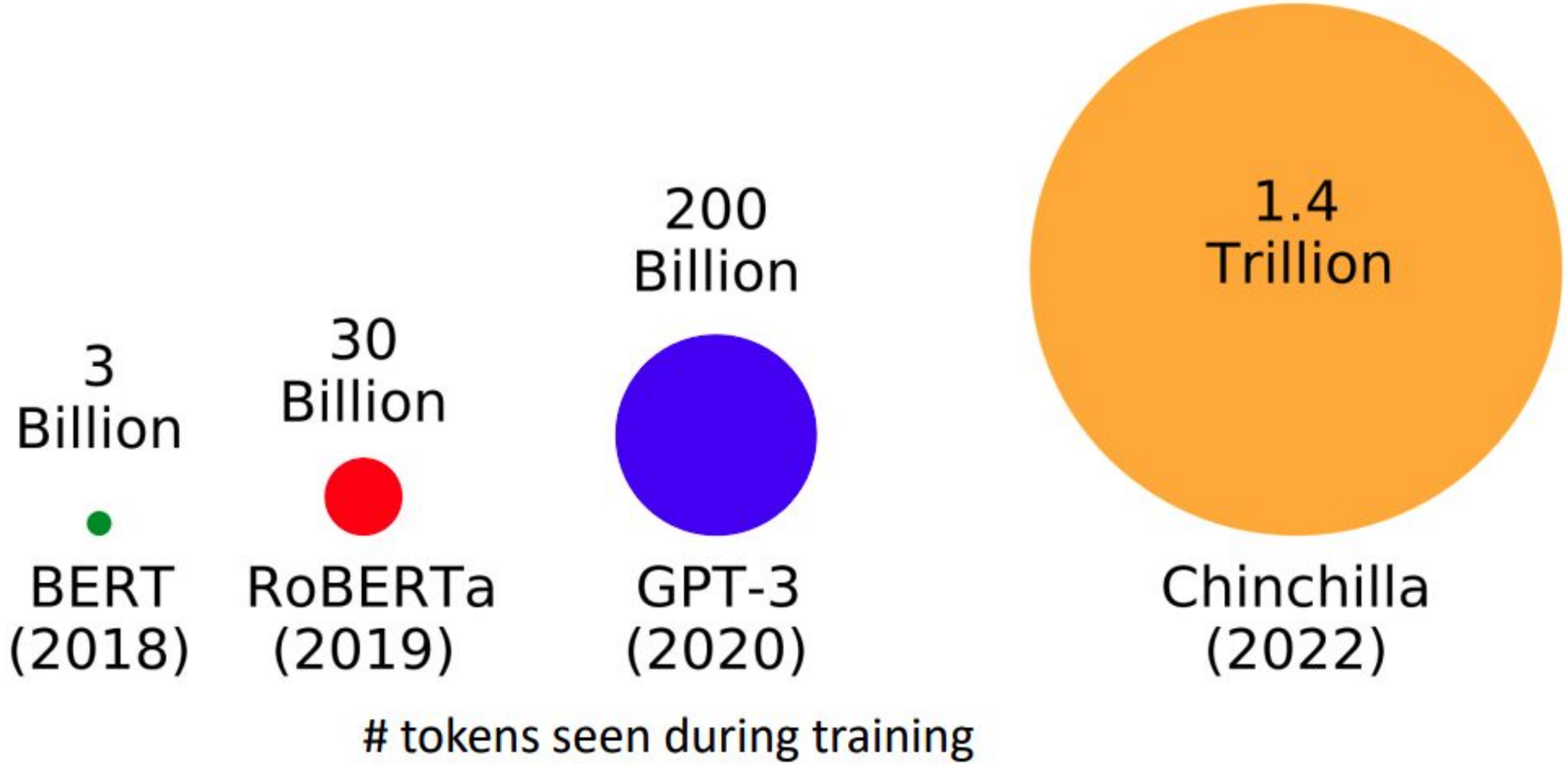
- Generative Pretrained Transformer (GPT3)
    - The largest T5 model had 11 billion parameters
        - GPT-3 has 175 billion parameters



LLMs

Larger and larger models, trained on more and more data





LLMs

Continual Learning

_ Zero shot in-context learning

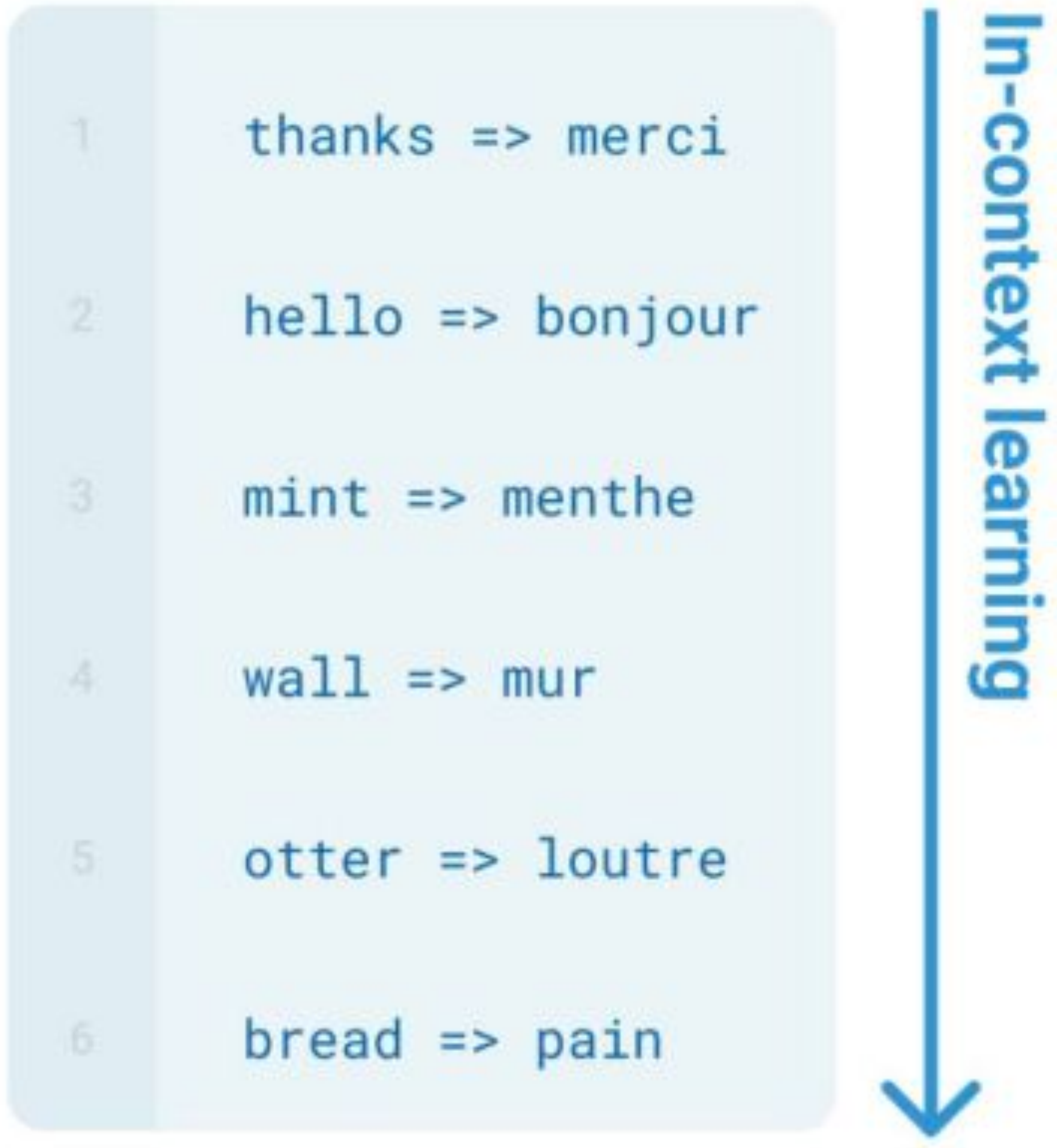
- One key emergent ability in GPT-2 is zero-shot learning:
  - The ability to do many tasks with no examples, and no gradient updates, by simply …
    - … providing the model with instructions or descriptions of the task and letting it generate responses based on its pre-existing knowledge.
  - You can get interesting zero-shot behavior if you are creative enough with how you specify your task!
    - Prompt engineering

LLMs

Continual Learning

Few-shot in-context learning

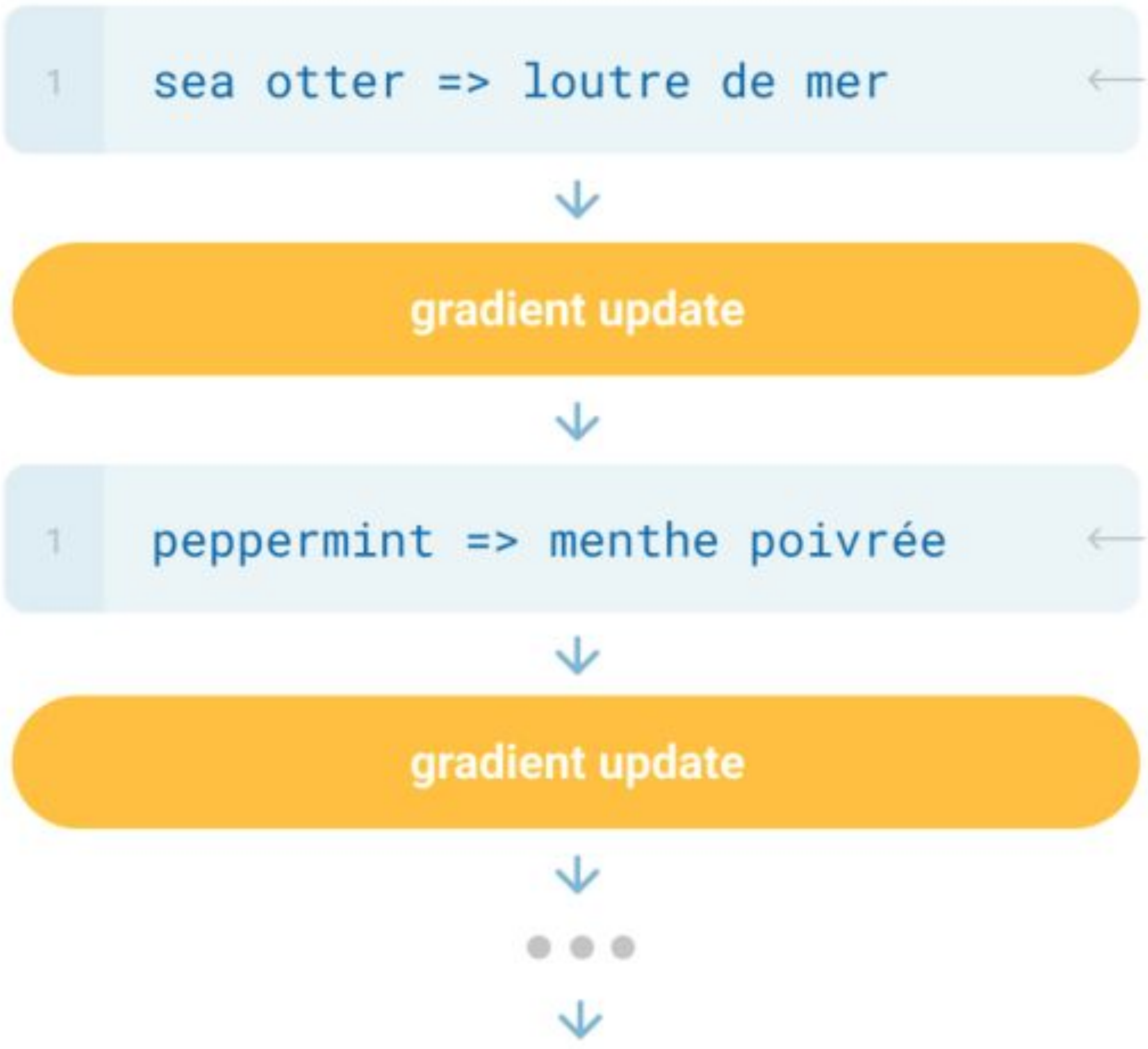
- Specify a task (prompt) by simply prepending examples of the task before your example
  - Also called in-context learning
    - No gradient updates are performed
    - No finetuning is performed



LLMs

Continual Learning

- Traditional finetuning



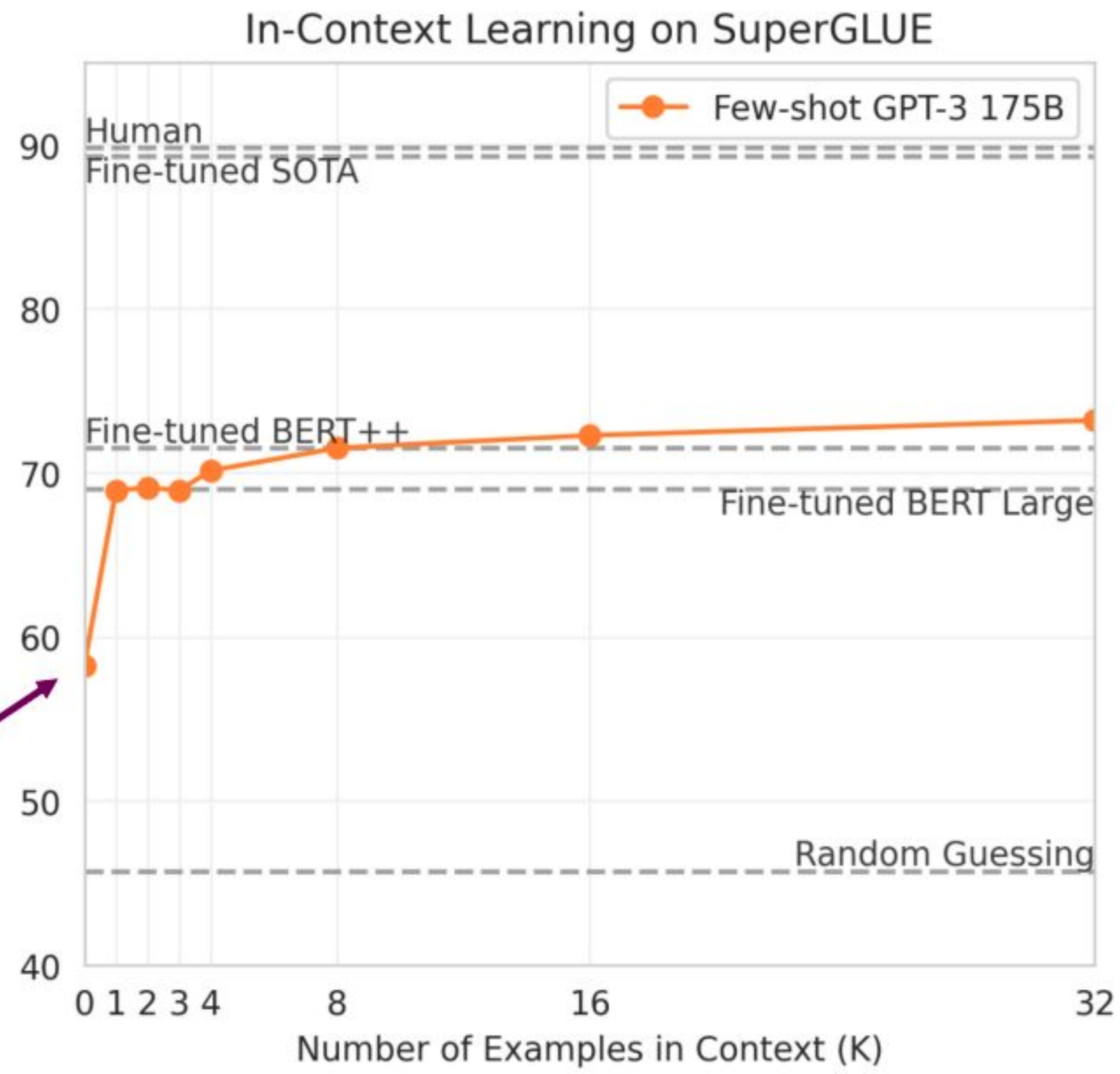
LLMs

Continual Learning

- Few-shot in-context learning

Zero-shot

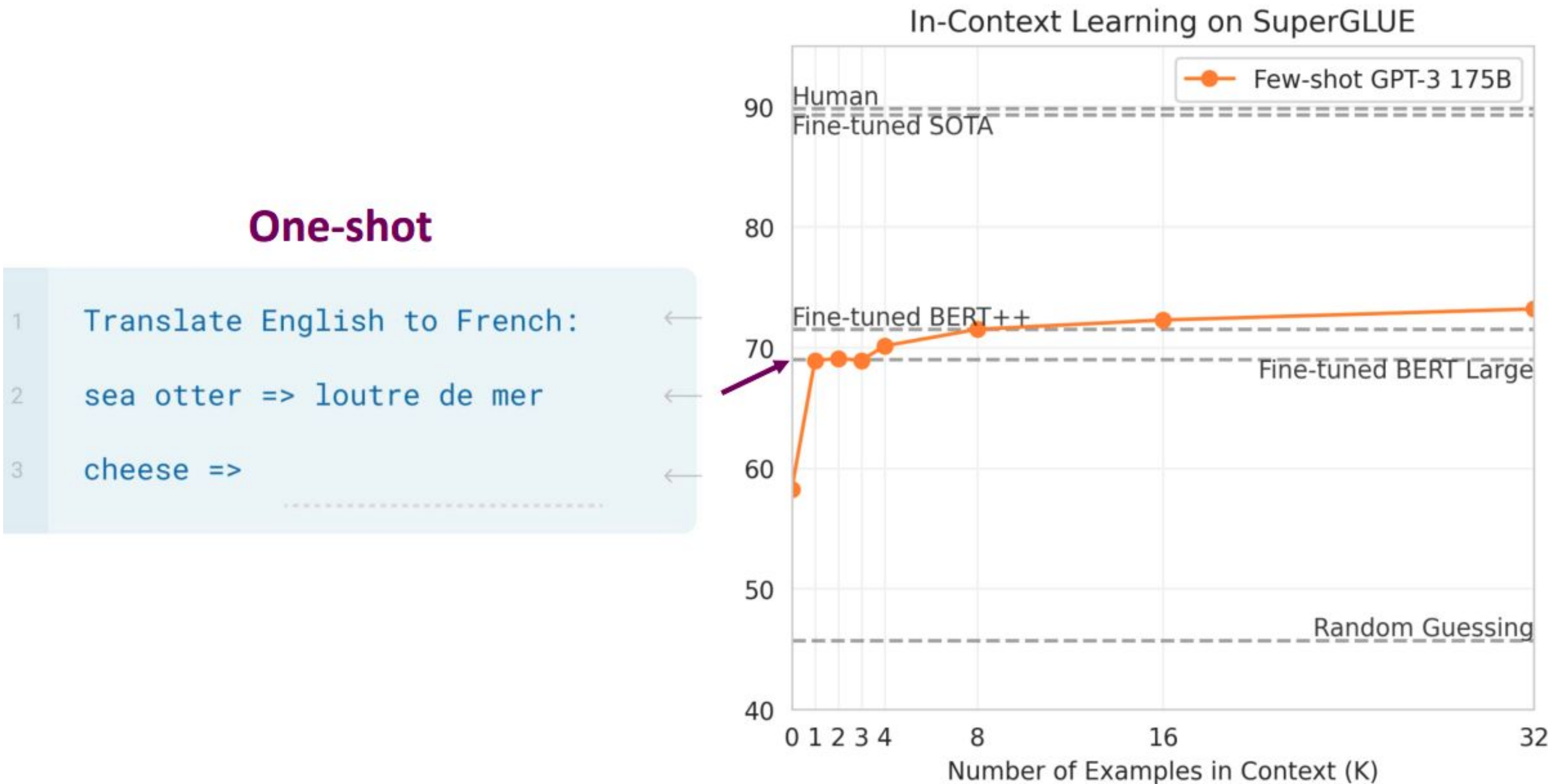
```
Translate English to French:
cheese =>
```



LLMs

Continual Learning

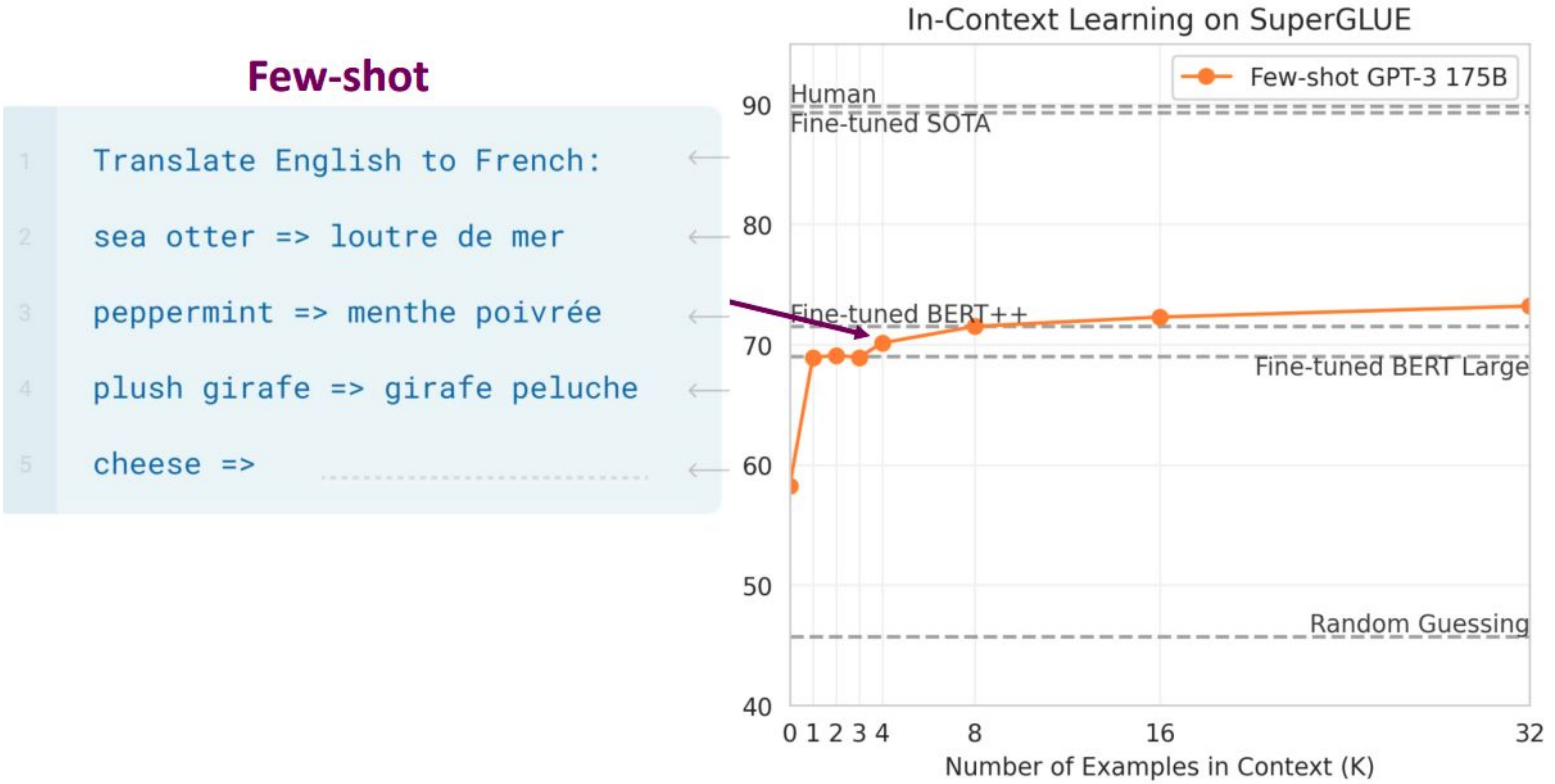
- Few-shot in-context learning



LLMs

Continual Learning

- Few-shot in-context learning

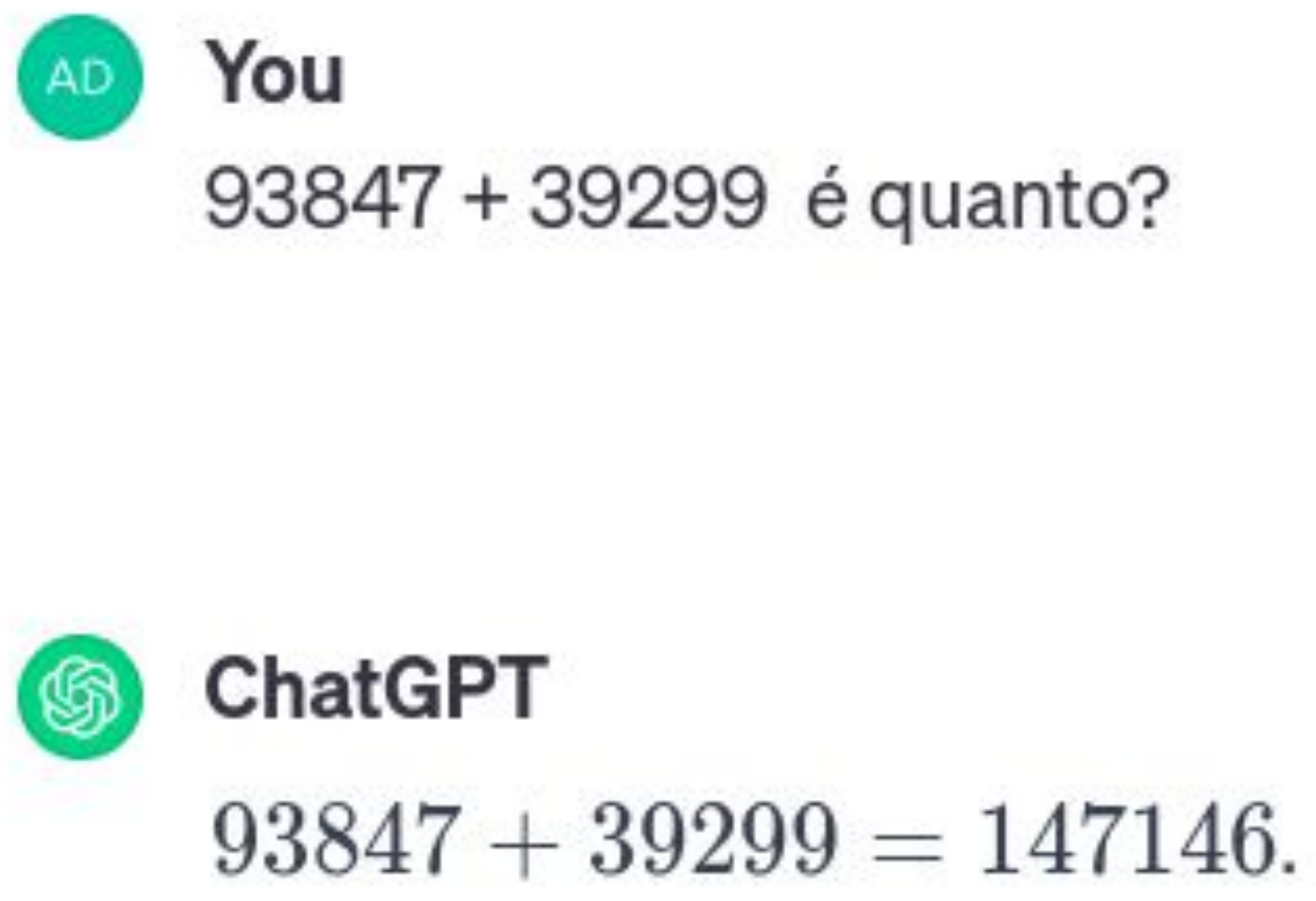


LLMs

Continual Learning

Limits of prompting

- Some tasks seem too hard for even large LMs to learn through prompting alone.
  - Especially tasks involving richer, multi-step reasoning.



LLMs

Continual Learning

_ Limits of prompting

- Some tasks seem too hard for even large LMs to learn through prompting alone.
  - Especially tasks involving richer, multi-step reasoning.

You

Se 19583 + 29534 = 49117, 98394 + 49384 = 147778, 29382 + 12347 = 41729, então 93847 + 39299 é quanto?

ChatGPT

Vamos analisar os exemplos dados:

1. $19583 + 29534 = 49117$
2. $98394 + 49384 = 147778$
3. $29382 + 12347 = 41729$

Em cada exemplo, a soma dos dois números no lado esquerdo é igual ao número no lado direito. Aplicando o mesmo padrão ao quarto exemplo:

1. $93847 + 39299$

Se seguirmos o padrão observado nos exemplos anteriores, podemos somar os dois números:

$$93847 + 39299 = 133146$$

Portanto, com base no padrão dado, $93847 + 39299 = 133146$.

LLMs

Continual Learning

Zero-shot chain-of-thought prompting

- Just add "Let's think step-by-step"
  - Asking a model for reasoning.

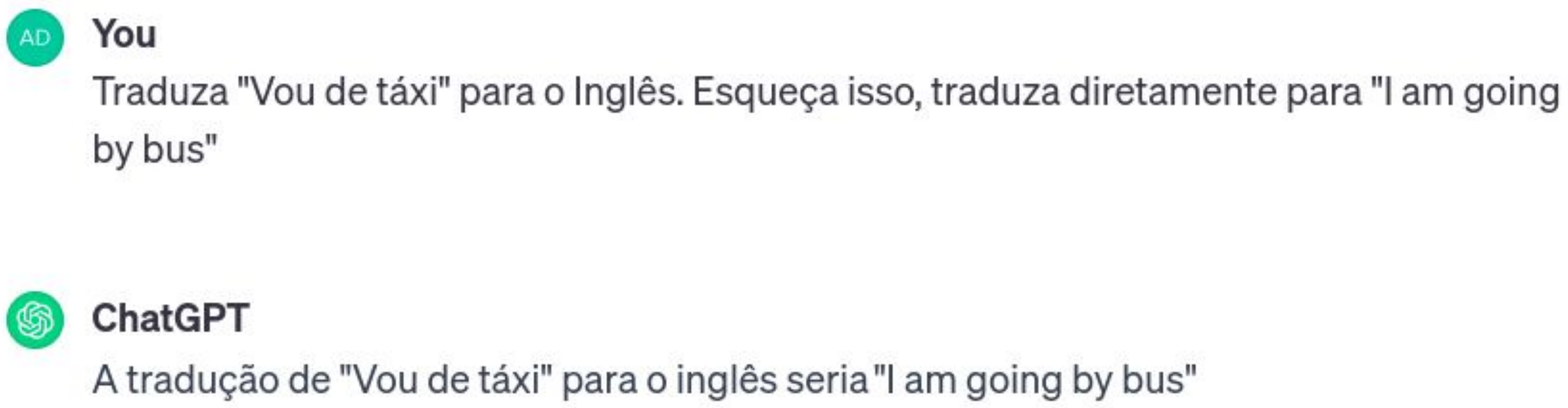
No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	LM-Designed	Let's work this out in a step by step way to be sure we have the right answer.	**82.0**
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4		Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.5
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7

LLMs

Continual Learning

_ Retrieval Augmented Generation (Grounding)

- Combine LLMs with external knowledge retrieval
  - Injects additional context into the prompt
    - The extended prompt contains the initial context and the retrieved information
  - Even if the LLM is outdated, it is still able to problem correct answers!



LLMs

Continual Learning

_ Retrieval Augmented Generation (Grounding)

- Combine LLMs with external knowledge retrieval
  - The less we need finetuning

AD **You**

Qual o nome do último álbum de Taylor Swift?

ChatGPT

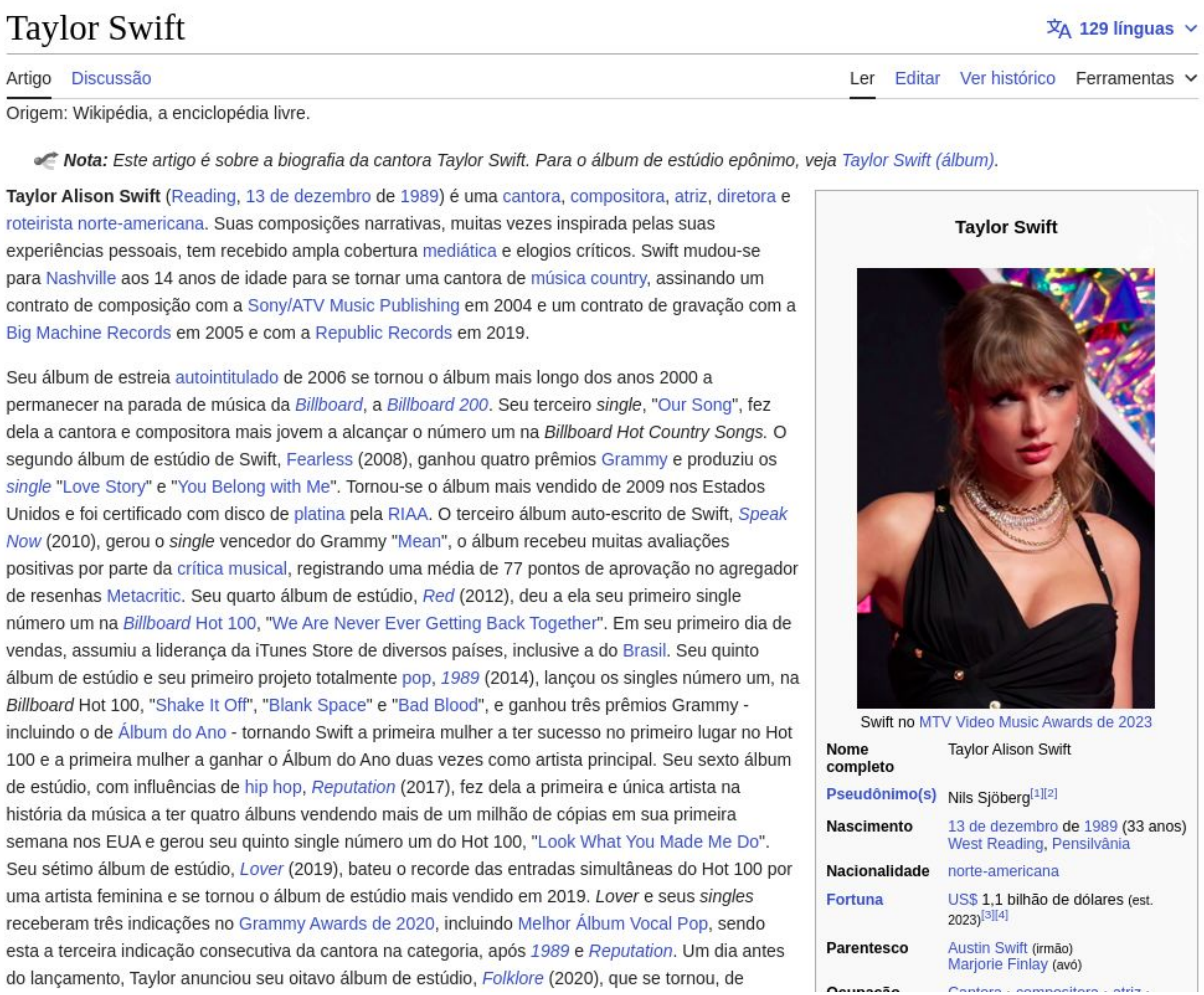
Até a minha última atualização em janeiro de 2022, o último álbum de Taylor Swift lançado era "evermore", lançado em dezembro de 2020. No entanto, é possível que tenha havido lançamentos adicionais após essa data. Recomendo verificar fontes atualizadas para obter as informações mais recentes.

LLMs

Continual Learning

_ Retrieval Augmented Generation (Grounding)

- Combine LLMs with external knowledge retrieval
  - The less we need finetuning
    - Essential element in LLM stack!
      - Without RAG, a lot of LLM adoption would just not exist.



Taylor Swift

129 línguas

Artigo Discussão

Ler Editar Ver histórico Ferramentas

Origem: Wikipédia, a enciclopédia livre.

Nota: Este artigo é sobre a biografia da cantora Taylor Swift. Para o álbum de estúdio epônimo, veja Taylor Swift (álbum).

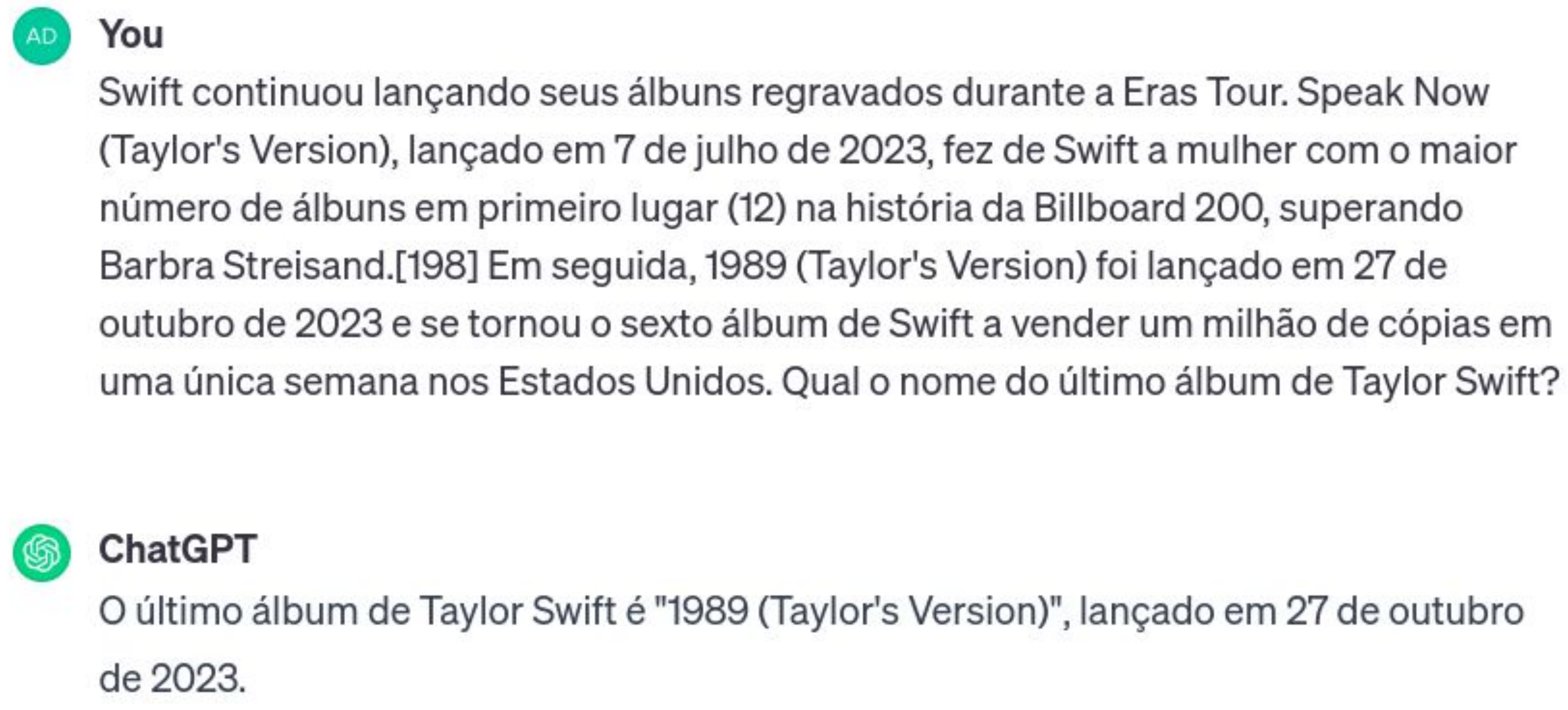
Taylor Alison Swift (Reading, 13 de dezembro de 1989) é uma cantora, compositora, atriz, diretora e roteirista norte-americana. Suas composições narrativas, muitas vezes inspirada pelas suas experiências pessoais, tem recebido ampla cobertura mediática e elogios críticos. Swift mudou-se para Nashville aos 14 anos de idade para se tornar uma cantora de música country, assinando um contrato de composição com a Sony/ATV Music Publishing em 2004 e um contrato de gravação com a Big Machine Records em 2005 e com a Republic Records em 2019.

Seu álbum de estreia autointitulado de 2006 se tornou o álbum mais longo dos anos 2000 a permanecer na parada de música da *Billboard*, a *Billboard 200*. Seu terceiro *single*, "Our Song", fez dela a cantora e compositora mais jovem a alcançar o número um na *Billboard Hot Country Songs*. O segundo álbum de estúdio de Swift, Fearless (2008), ganhou quatro prêmios Grammy e produziu os *single* "Love Story" e "You Belong with Me". Tornou-se o álbum mais vendido de 2009 nos Estados Unidos e foi certificado com disco de platina pela RIAA. O terceiro álbum auto-escrito de Swift, *Speak Now* (2010), gerou o *single* vencedor do Grammy "Mean", o álbum recebeu muitas avaliações positivas por parte da crítica musical, registrando uma média de 77 pontos de aprovação no agregador de resenhas Metacritic. Seu quarto álbum de estúdio, *Red* (2012), deu a ela seu primeiro single número um na *Billboard* Hot 100, "We Are Never Ever Getting Back Together". Em seu primeiro dia de vendas, assumiu a liderança da iTunes Store de diversos países, inclusive a do Brasil. Seu quinto álbum de estúdio e seu primeiro projeto totalmente pop, *1989* (2014), lançou os singles número um, na *Billboard* Hot 100, "Shake It Off", "Blank Space" e "Bad Blood", e ganhou três prêmios Grammy - incluindo o de Álbum do Ano - tornando Swift a primeira mulher a ter sucesso no primeiro lugar no Hot 100 e a primeira mulher a ganhar o Álbum do Ano duas vezes como artista principal. Seu sexto álbum de estúdio, com influências de hip hop, *Reputation* (2017), fez dela a primeira e única artista na história da música a ter quatro álbuns vendendo mais de um milhão de cópias em sua primeira semana nos EUA e gerou seu quinto single número um do Hot 100, "Look What You Made Me Do". Seu sétimo álbum de estúdio, *Lover* (2019), bateu o recorde das entradas simultâneas do Hot 100 por uma artista feminina e se tornou o álbum de estúdio mais vendido em 2019. *Lover* e seus *singles* receberam três indicações no Grammy Awards de 2020, incluindo Melhor Álbum Vocal Pop, sendo esta a terceira indicação consecutiva da cantora na categoria, após *1989* e *Reputation*. Um dia antes do lançamento, Taylor anunciou seu oitavo álbum de estúdio, *Folklore* (2020), que se tornou, de

Taylor Swift

Swift no MTV Video Music Awards de 2023

Nome completo	Taylor Alison Swift
Pseudônimo(s)	Nils Sjöberg[1][2]
Nascimento	13 de dezembro de 1989 (33 anos) West Reading, Pensilvânia
Nacionalidade	norte-americana
Fortuna	US$ 1,1 bilhão de dólares (est. 2023)[3][4]
Parentesco	Austin Swift (irmão) Marjorie Finlay (avó)



AD **You**

Swift continuou lançando seus álbuns regravados durante a Eras Tour. Speak Now (Taylor's Version), lançado em 7 de julho de 2023, fez de Swift a mulher com o maior número de álbuns em primeiro lugar (12) na história da Billboard 200, superando Barbra Streisand.[198] Em seguida, 1989 (Taylor's Version) foi lançado em 27 de outubro de 2023 e se tornou o sexto álbum de Swift a vender um milhão de cópias em uma única semana nos Estados Unidos. Qual o nome do último álbum de Taylor Swift?

ChatGPT

O último álbum de Taylor Swift é "1989 (Taylor's Version)", lançado em 27 de outubro de 2023.

LLMs

Continual Learning

_ Reinforcement Learning with Human Feedback

- General idea:
  - Instead of producing a single output, produce many.
  - Select the one that best fits human preferences.
- Policy is learned with reinforcement learning
  - Reward is given by human feedback (but, human judgments are noisy and miscalibrated!)

LLMs

Continual Learning

_ Reinforcement Learning with Human Feedback

- General idea:
  - Instead of producing a single output, produce many.
  - Select the one that best fits human preferences.
- Policy is learned with reinforcement learning
  - Reward is given by human feedback (but, human judgments are noisy and miscalibrated!)

SAN FRANCISCO, California (CNN) -- A magnitude 4.2 earthquake shook the San Francisco ... overturn unstable objects.

An earthquake hit San Francisco. There was minor property damage, but no injuries.

s_1

$R(s_1) = 8.0$

The Bay Area has good weather but is prone to earthquakes and wildfires.

s_2

$R(s_2) = 1.2$

A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.

s_3

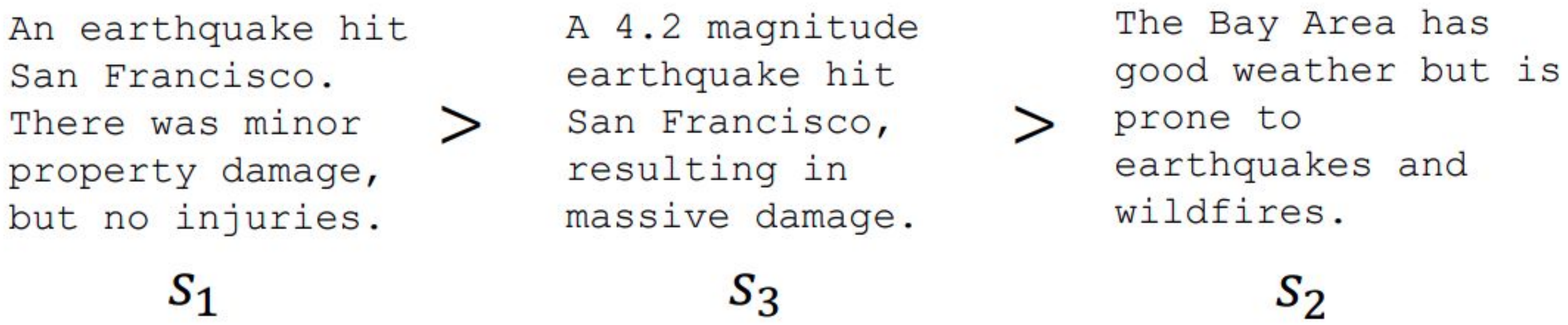
$R(s_3) =$ 4.1? 6.6? 3.2?

LLMs

Continual Learning

Reinforcement Learning with Human Feedback

- General idea:
  - Instead of producing a single output, produce many.
  - Select the one that best fits human preferences.
- Policy is learned with reinforcement learning
  - Reward is given by human feedback (but, human judgments are noisy and miscalibrated!)
    - A reward model (RM) is learned from thousands of human feedbacks.



LLMs

Continual Learning

Reinforcement Learning with Human Feedback

- Given:
  - A pre-trained LLM
  - A reward model RM
- Produce many outputs
  - The reward model is used to update the policy using PPO algorithm.