# Language Modelling

**Cristina España-Bonet**

**UdS & DFKI, Saarbrücken, Germany**

Artificial Intelligence with Deep Learning

25th April 2019

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

`Messi scored a fantastic _____`

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

```
Messi scored a fantastic goal
```

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

`Messi scored a fantastic goal`

`It's raining cats and _____`

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

`Messi scored a fantastic goal`

`It's raining cats and dogs`

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

`Messi scored a fantastic goal`

`It's raining cats and dogs`

`I saw falling from that tree a red _____`

# Intuition

## *Language Modelling is...*

*...anticipating coherent words.*

**How works your language modelling?**

`Messi scored a fantastic goal`

`It's raining cats and dogs`

`I saw falling from that tree a red apple`

# Intuition

## *A Language Model...*

*...measures fluency in sequences.*

# Intuition

## *A Language Model...*

*...measures fluency in sequences.*

**Which sequence is preferred by your language modell?**

`I saw falling from that tree a red apple`

`I saw falling from that tree a clever apple`

`I saw falling from that tree a elephant apple`

# Outline

# Statistical Approach

## *Language Model Definition*

Estimation of how probable a sequence of tokens is.

# Statistical Approach

## *Language Model Definition*

Estimation of how **probable** a sequence of tokens is.

Naïve estimation on a corpus with $N$ sentences:

**Frequentist probability** of a sentence $e$:

$$P(e) = \frac{N_e}{N_{sentences}}$$

# Statistical Approach

## *Language Model Definition*

Estimation of how **probable** a sequence of tokens is.

Naïve estimation on a corpus with $N$ sentences:

**Frequentist probability** of a sentence $e$:

$$P(e) = \frac{N_e}{N_{sentences}}$$

Problem:

- Long chains are difficult to observe in corpora.
  $\Rightarrow$ Long sentences may have zero probability!

# Statistical Approach

## *The n-gram approach*

The language model assigns a probability $P(e)$ to a sequence of words $e \Rightarrow \{w_1, \ldots, w_m\}$.

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1})$$

- The probability of a sentence is the product of the conditional probabilities of each word $w_i$ given the previous ones
- Independence assumption: the probability of $w_i$ is only conditioned by the $n$ previous words

# Statistical Approach

## *Example, a 4-gram model*

*e*: `All work and no play makes Jack a dull boy`

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$P(e) = P(\texttt{All}|\phi, \phi, \phi)$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$P(e) = P(\texttt{All}|\phi, \phi, \phi)\ P(\texttt{work}|\phi, \phi, \texttt{All})$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$P(e) = P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work})$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$P(e) = P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work})$$
$$P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$P(e) = P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work})$$
$$P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no})$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$P(e) = P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work})$$
$$P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no})$$
$$P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$
\begin{aligned}
P(e) = {} & P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work}) \\
& P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no}) \\
& P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})P(\texttt{Jack}|\texttt{no},\texttt{play},\texttt{makes})
\end{aligned}
$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$
\begin{aligned}
P(e) = {} & P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work}) \\
& P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no}) \\
& P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})P(\texttt{Jack}|\texttt{no},\texttt{play},\texttt{makes}) \\
& P(\texttt{a}|\texttt{play},\texttt{makes},\texttt{Jack})
\end{aligned}
$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$
\begin{aligned}
P(e) = {} & P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work}) \\
& P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no}) \\
& P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})P(\texttt{Jack}|\texttt{no},\texttt{play},\texttt{makes}) \\
& P(\texttt{a}|\texttt{play},\texttt{makes},\texttt{Jack})P(\texttt{dull}|\texttt{makes},\texttt{Jack},\texttt{a})
\end{aligned}
$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$
\begin{aligned}
P(e) = {} & P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work}) \\
& P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no}) \\
& P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})P(\texttt{Jack}|\texttt{no},\texttt{play},\texttt{makes}) \\
& P(\texttt{a}|\texttt{play},\texttt{makes},\texttt{Jack})P(\texttt{dull}|\texttt{makes},\texttt{Jack},\texttt{a}) \\
& P(\texttt{boy}|\texttt{Jack},\texttt{a},\texttt{dull})
\end{aligned}
$$

# Statistical Approach

## *Example, a 4-gram model*

$e$: `All work and no play makes Jack a dull boy`

$$
\begin{aligned}
P(e) = {} & P(\texttt{All}|\phi,\phi,\phi)\ P(\texttt{work}|\phi,\phi,\texttt{All})\ P(\texttt{and}|\phi,\texttt{All},\texttt{work}) \\
& P(\texttt{no}|\texttt{All},\texttt{work},\texttt{and})\ P(\texttt{play}|\texttt{work},\texttt{and},\texttt{no}) \\
& P(\texttt{makes}|\texttt{and},\texttt{no},\texttt{play})P(\texttt{Jack}|\texttt{no},\texttt{play},\texttt{makes}) \\
& P(\texttt{a}|\texttt{play},\texttt{makes},\texttt{Jack})P(\texttt{dull}|\texttt{makes},\texttt{Jack},\texttt{a}) \\
& P(\texttt{boy}|\texttt{Jack},\texttt{a},\texttt{dull})
\end{aligned}
$$

where, for each factor,

$$P(\texttt{and}|\phi,\texttt{All},\texttt{work}) = \frac{N_{(\texttt{All work and})}}{N_{(\texttt{All work})}}$$

# Statistical Approach

## *n-gram Model, Comments*

4-gram conditions on a lot of context, so given sufficient training data these counts will be high and it will converge to the *true value*

4-gram many counts will be equal to zero, so we need many samples to get a good estimate

# Statistical Approach

## *n-gram Model, Comments*

4-gram conditions on a lot of context, so given sufficient training data these counts will be high and it will converge to the *true value*

4-gram many counts will be equal to zero, so we need many samples to get a good estimate

1-gram will converge relatively quickly to their expected value, and so don't need many samples

1-gram completely ignores context, and so it will converge to a less-good estimator as the number of training samples increase

# Statistical Approach

## *Known Problems*

Main problems and criticisims:

- Long-range dependencies are lost
- Still, some $n$-grams can be not observed in the corpus

# Statistical Approach

## *Known Problems*

Main problems and criticisims:

- Long-range dependencies are lost
- Still, some $n$-grams can be not observed in the corpus

**Solution**

Smoothing techniques:

- Linear interpolation
- Back-off models
- Discounting

# Statistical Approach

## *Smoothing techniques*

- Discounting
  - Keep part of the probabilty mass for unseen words

# Statistical Approach

## *Smoothing techniques*

- Discounting
  - Keep part of the probabilty mass for unseen words
- Back-off models
  - If $n$-gram is not present, go to $(n-1)$-gram

# Statistical Approach

## *Smoothing techniques*

- Discounting
  - Keep part of the probabilty mass for unseen words
- Back-off models
  - If $n$-gram is not present, go to $(n-1)$-gram
- Linear interpolation

$$P(\texttt{and}|\texttt{All},\texttt{work}) = \frac{N_{(\texttt{All},\texttt{work},\texttt{and})}}{N_{(\texttt{All},\texttt{work})}}$$

# Statistical Approach

## *Smoothing techniques*

- Discounting
  - Keep part of the probabilty mass for unseen words
- Back-off models
  - If $n$-gram is not present, go to $(n-1)$-gram
- Linear interpolation

$$P(\texttt{and}|\texttt{All},\texttt{work}) = \lambda_3 \frac{N_{(\texttt{All},\texttt{work},\texttt{and})}}{N_{(\texttt{All},\texttt{work})}} + \lambda_2 \frac{N_{(\texttt{work},\texttt{and})}}{N_{(\texttt{work})}} + \lambda_1 \frac{N_{(\texttt{and})}}{N_{words}} + \lambda_0$$

# Outline

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

**Goal**

Learn with a NN the probability of sequences of words:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{\mathbf{i-(n-1)}}, \ldots, w_{i-1}) \qquad n\text{-gram}$$

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

**Goal**

Learn with a NN the probability of sequences of words:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{\mathbf{i-(n-1)}}, \ldots, w_{i-1}) \qquad n\text{-gram}$$

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{\mathbf{0}}, \ldots, w_{i-1}) \qquad \text{neural}$$

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

### Goal

Learn with a NN the probability of sequences of words:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{\mathbf{i-(n-1)}}, \ldots, w_{i-1}) \qquad n\text{-gram}$$

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_{\mathbf{0}}, \ldots, w_{i-1}) \qquad \text{neural}$$

### Network tasks:

1. learn a representation for each word (embedding)
2. learn the probability function from these representations

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

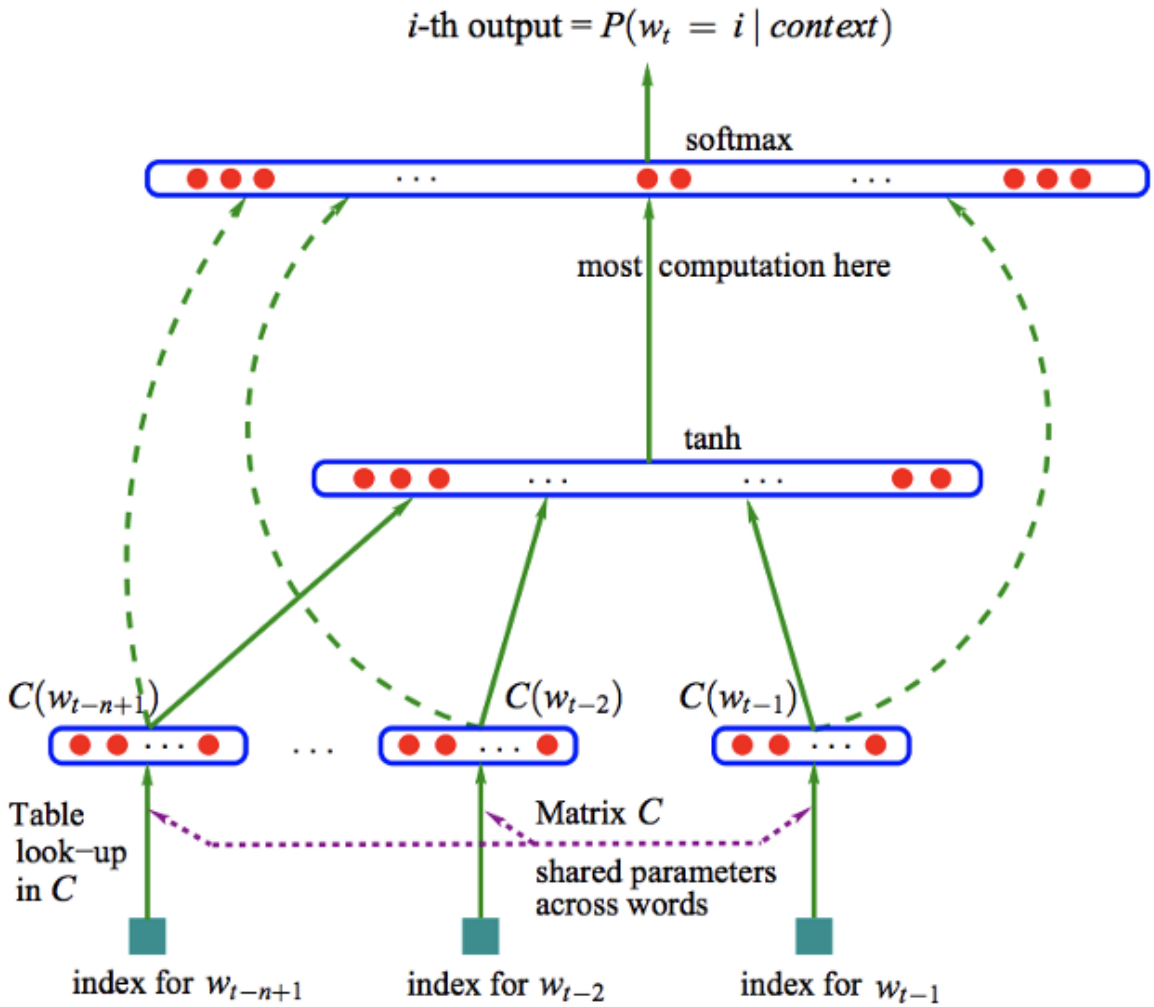



Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

### Why it works?

- First, it does work!

  Perplexity improvement wrt. $n$-gram
  (Brown and Hansard corpora in the paper)

- Word embeddings learn semantic similarities

  *The dog was sleeping* vs. *The cat was sleeping*

# Neural Approach

## *First Neural Language Model [Bengio et al. 2003]*

### More comments

- The model scales linearly with the vocabulary
- At that time (2003) the softmax was a bottleneck

$$softmax(a_k) = \frac{e^{\ a_k(\mathbf{x})}}{\sum_{j=1}^{T} e^{\ a_j(\mathbf{x})}}$$

- For next slides... if we share parameters (LSTMs!) it could be sublinear in the number of words

# Neural Approach

## *Feed Forward vs. Recurrent Language Models*

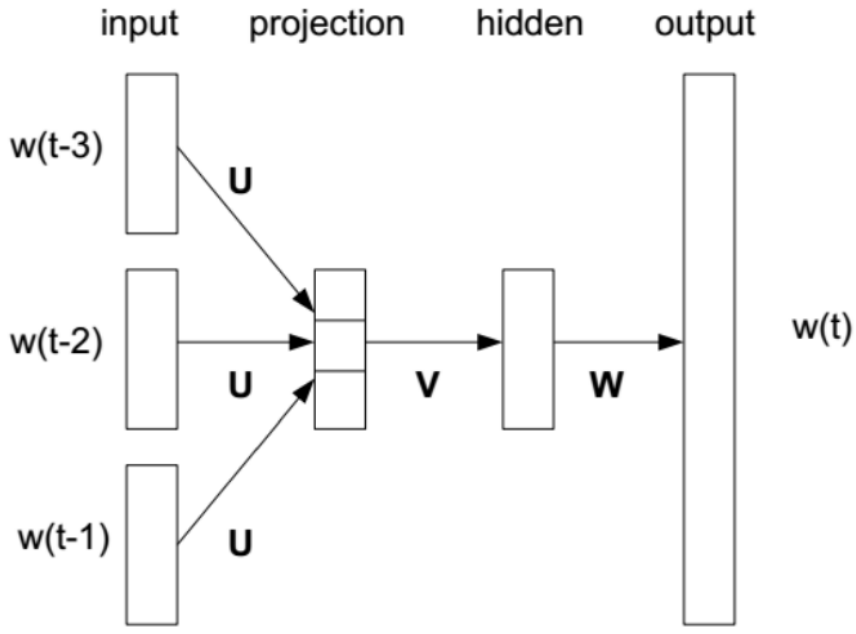
- Bengio's model is a FFN (that works great!)
- FFNs only indirectly deal with sequential data
- FF LMs are basically *soft* $n$-gram LMs
  (their history is still fixed)
- The model needs to remember a longer history,
  with loops
- A neural net with loops is a recurrent neural network
  (RNN)

# Neural Approach

## *Recurrent Neural Language Model [Mikolov et al. 2011]*

## Feedforward neural network LM

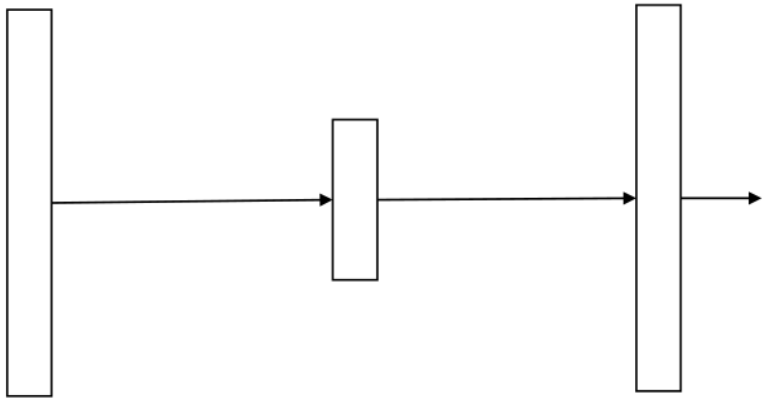
- Proposed by (Bengio et al, 2003)
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary
- The basic model is computationally very expensive



# Neural Approach

## *Recurrent Neural Language Model [Mikolov et al. 2011]*

### Towards recurrent neural network LM



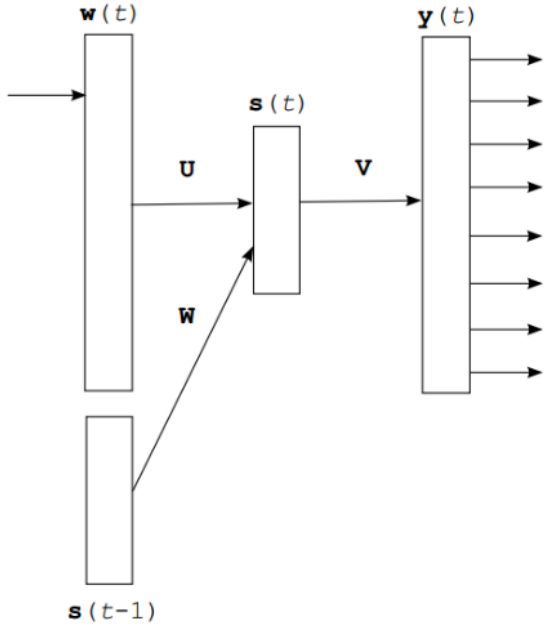

- Back to the bigram NNLM: is there a better way how to represent time than using N-1 previous words as separate inputs?

# Neural Approach

## *Recurrent Neural Language Model [Mikolov et al. 2011]*

### Recurrent neural network LM

- Recurrent NNLM is about the same as the bigram NNLM
- Additional weights from the hidden layer in the previous time step
- In theory, the hidden layer can learn to represent unlimited memory



# Neural Approach

## *Comparison of Architectures [Shi 2017]*

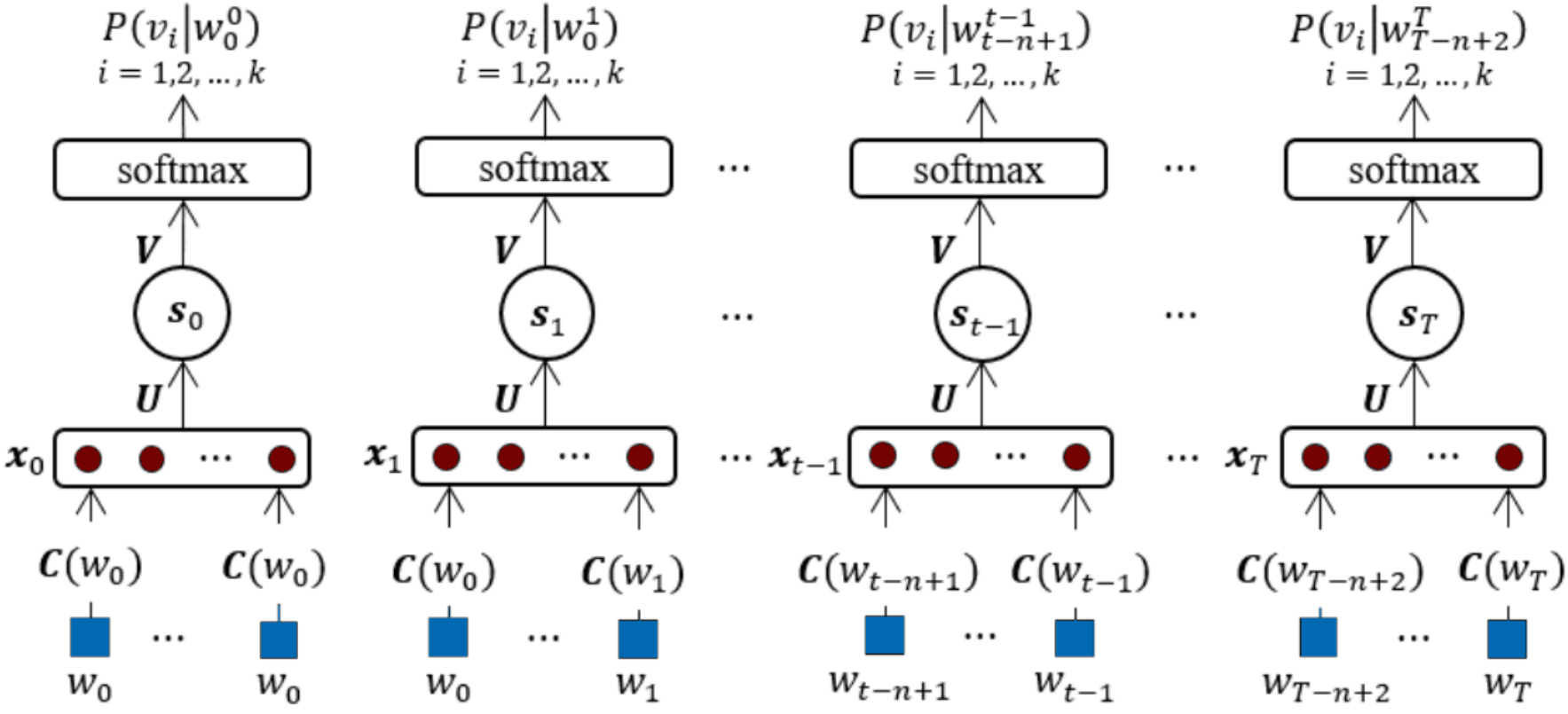



Figure 1: Feed-forward neural network language model

# Neural Approach

## *Comparison of Architectures [Shi 2017]*

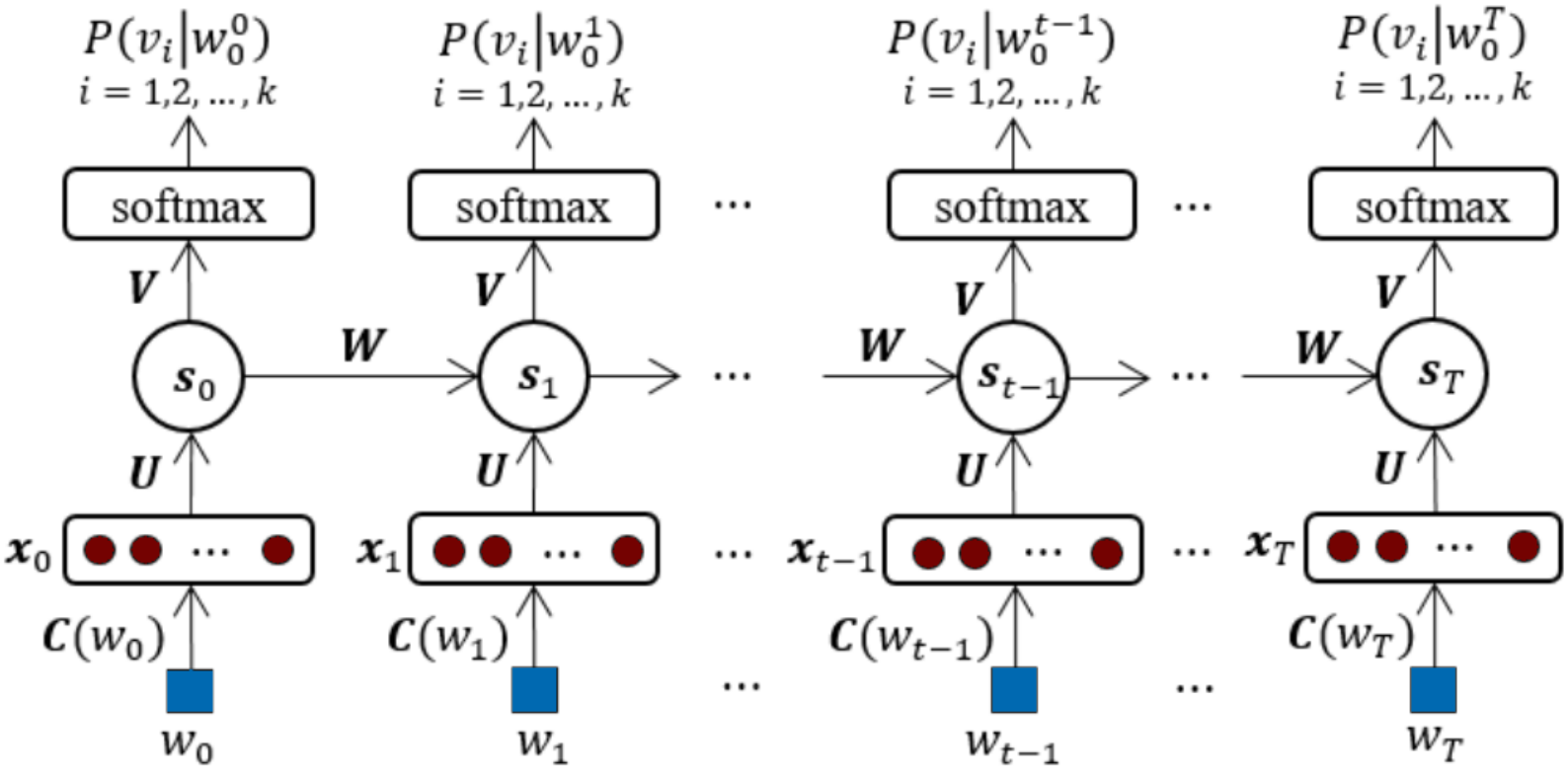



Figure 2: Recurrent neural network language model

# Training of NNLMs

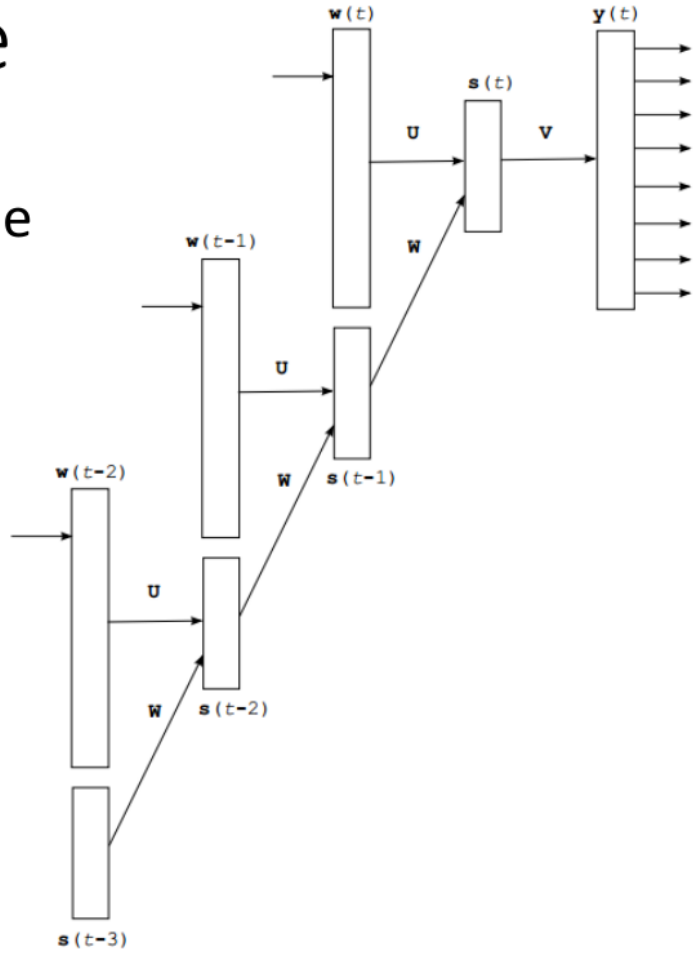
- Feedforward NNLM: the classis SGD + backpropagation
- Recurrent NNLM: the same, but the backpropagation part is more difficult to implement correctly
- The algorithm for computing gradients in RNN is called "Backpropagation through time"

# Neural Approach

## *Recurrent Neural Language Model [Mikolov et al. 2011]*

### Backpropagation through time

- The intuition is that we unfold the RNN in time
- We obtain deep neural network with shared weights **U** and **W**

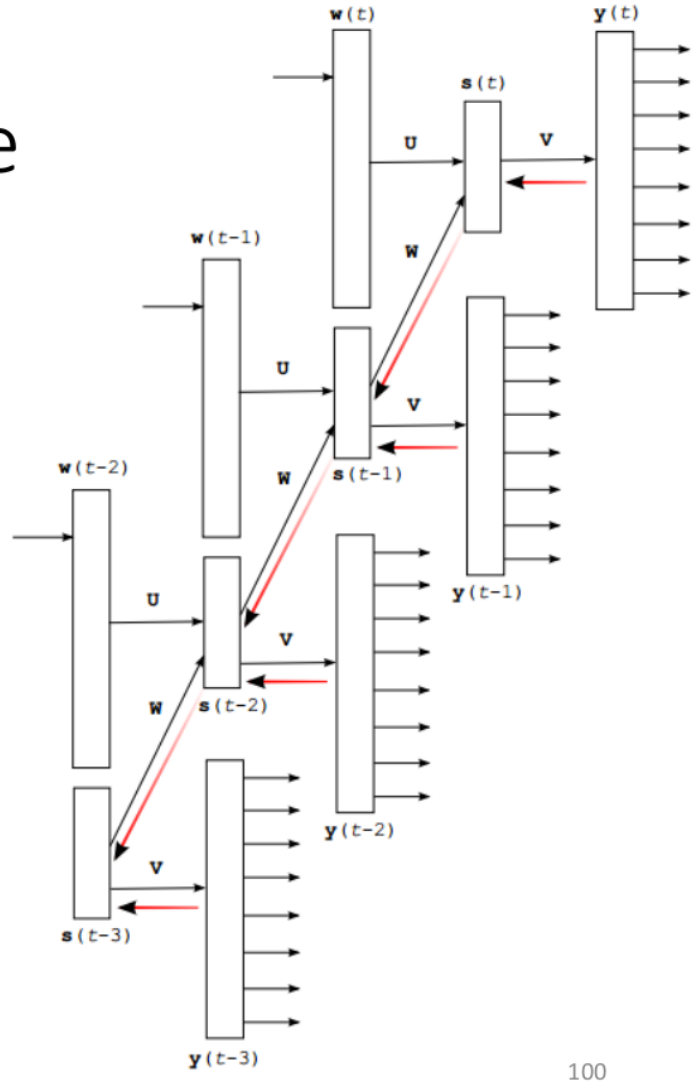


# Neural Approach

## *Recurrent Neural Language Model [Mikolov et al. 2011]*

### Backpropagation through time

- We train the unfolded RNN using normal backpropagation + SGD
- In practice, we limit the number of unfolding steps to 5 – 10
- It is computationally more efficient to propagate gradients after few training examples (batch mode)



# Neural Approach

## *Evaluation [Mikolov et al. 2011]*

Comparison of performance on small data: Penn Treebank

- Small, standard dataset, ~1M words
- RNN outperforms FNN by about 10%

| Model | Perplexity |
|---|---|
| Kneser-Ney 5-gram | 141 |
| Maxent 5-gram | 142 |
| Random forest | 132 |
| Feedforward NNLM | 140 |
| Recurrent NNLM | 125 |

More results in: *Empirical Evaluation and Combination of Advanced Language Modeling Techniques* (Mikolov et al, 2011)

# Neural Approach

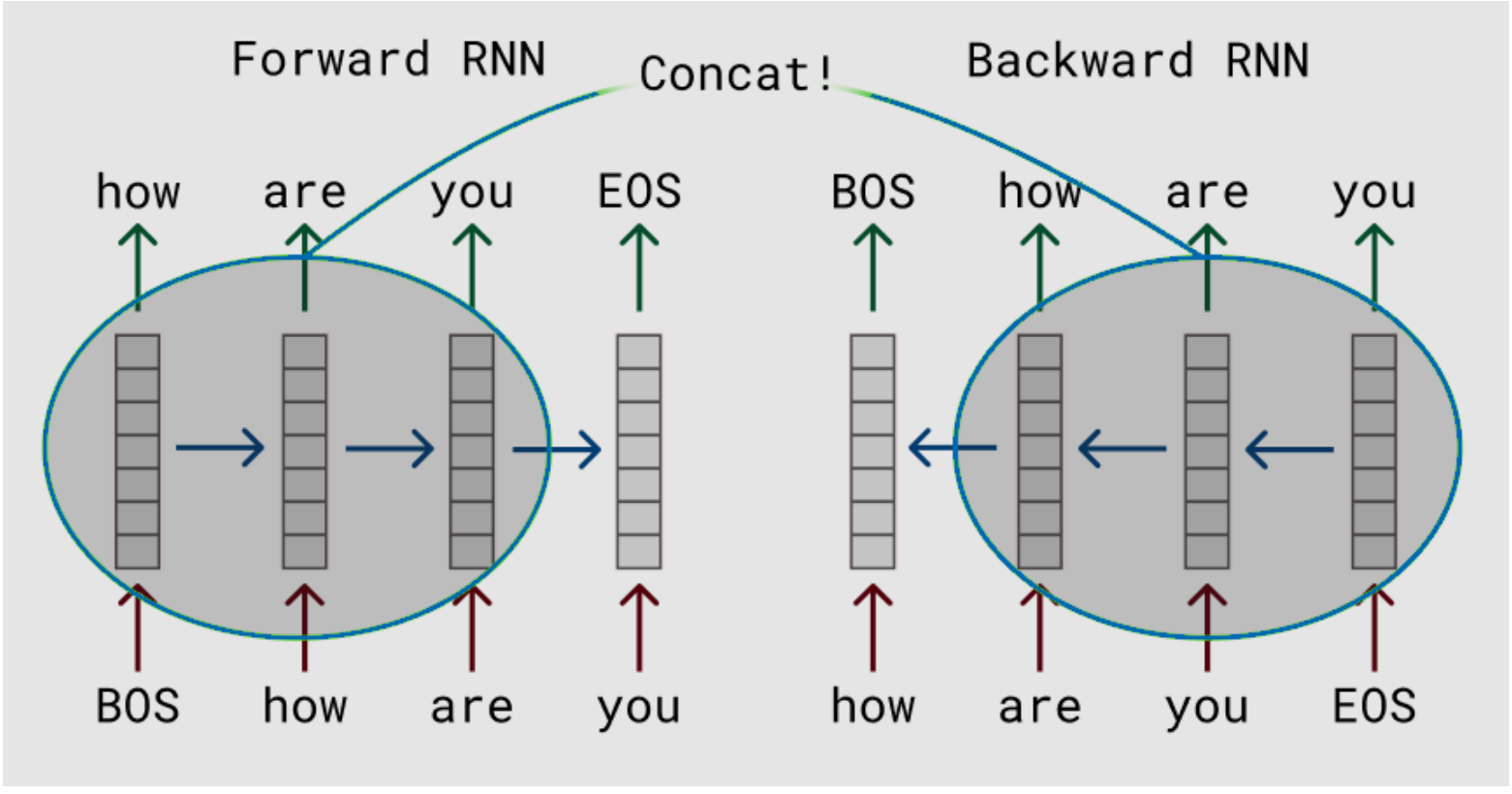
## *Large Scale Neural Language Model*

**Language models as auxiliary task for context aware word embeddings**

| Embeddings | LM Architecture | Reference |
|---|---|---|
| ELMo | biLSTM | [Peters et al., 2018] |
| OpenAI GTP | transformer decoder | [Radford et al., 2018] |
| Bert | biTransformer encoder | [Devlin et al., 2018] |

# Neural Approach

## *Remember from the Word Embeddings Lecture... biRNN*




https://medium.com/@plusepsilon/the-bidirectional-language-model-1f3961d1fb27

# Neural Approach

## *Stop! Acknowledgments*

In the following slides **figures** are borrowed from

**Lil'Log**

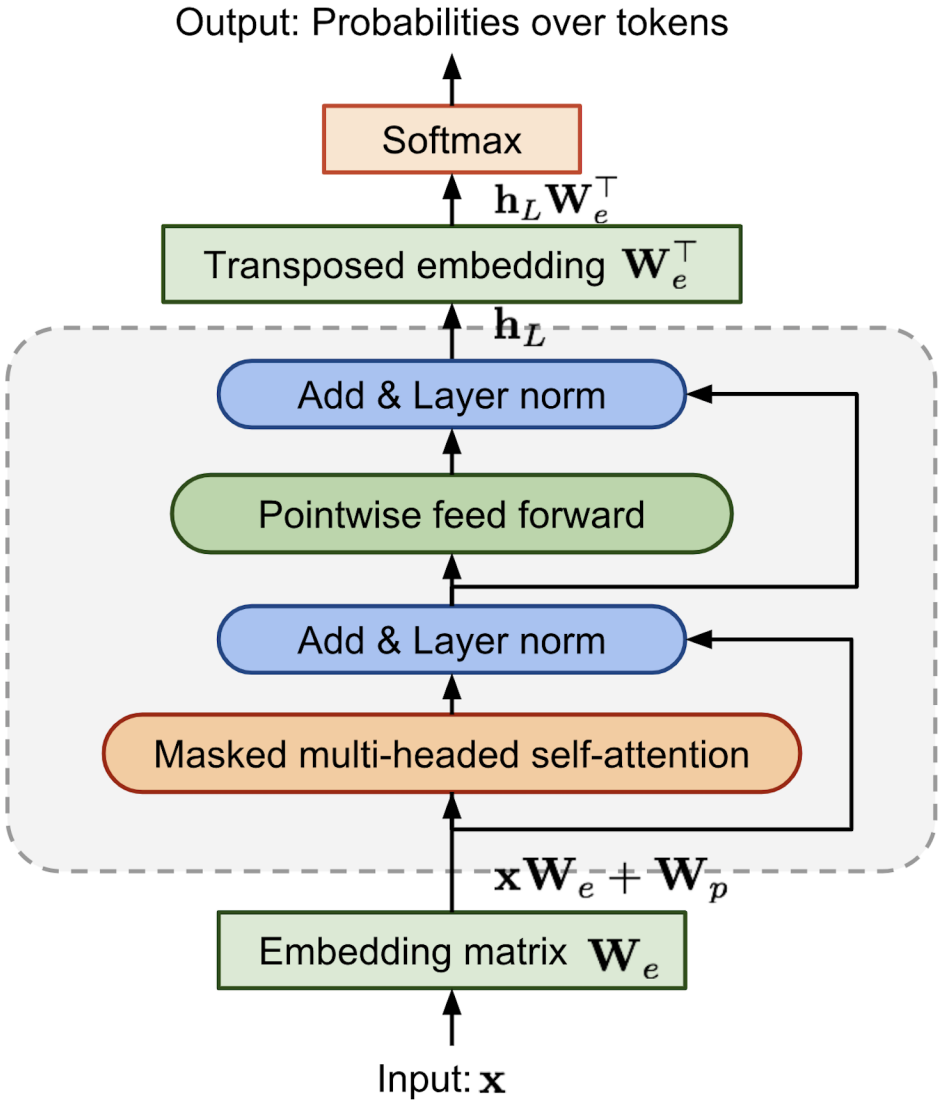
`https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html`

**Jay Alammar**

`http://jalammar.github.io/illustrated-transformer/`

# Neural Approach

## *Transformer Decoder Language Model [Radford et al., 2018]*

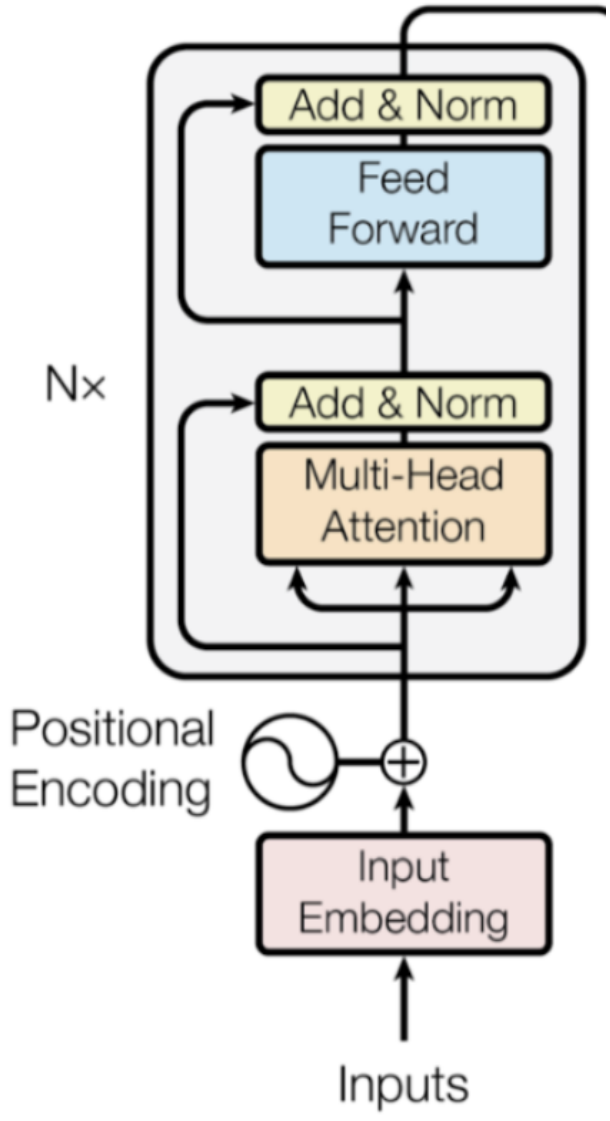



**Transformer Block**
Repeat x L=12

$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1})$$
$$\ell = 1, \ldots, L$$

# Neural Approach

## *Transformer Encoder Language Model [Devlin et al., 2018]*



# Neural Approach

## *Stop!*

Have you heard about them before?

**Transformer**

**Encoder**

**Decoder**

**Self-attention**

**Multi-head attention**

# Neural Approach

## *Stop!*

Have you heard about them before?

**Transformer**

**Encoder**

**Decoder**

**Self-attention**

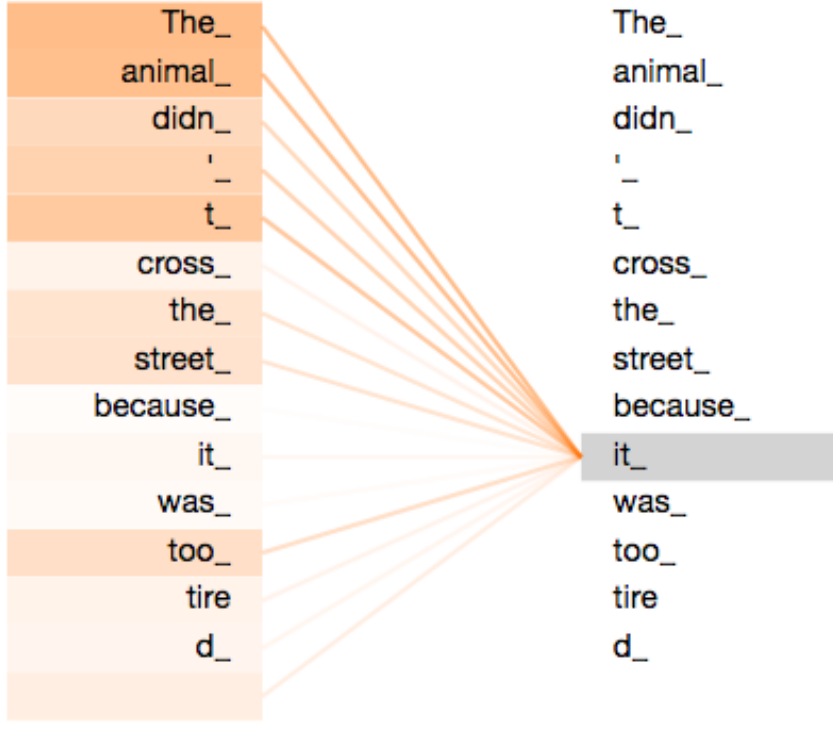
**Multi-head attention**

State-of-the art in **most** NLP tasks, **want details!!**

# Digression: Transformer Elements

## *Attention vs. Self-Attention*

- **Intra** endoder/decoder attention vs. **inter** encoder-decoder
- Always **alignment weights**



# Digression: Transformer Elements

## *Self-Attention*

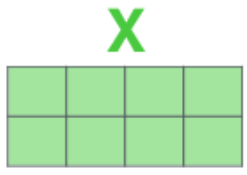
1) This is our input sentence

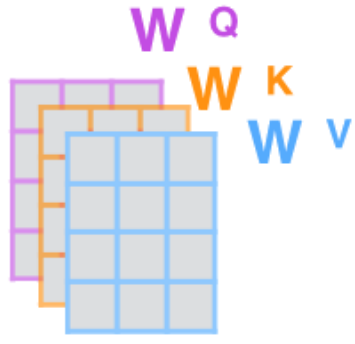
2) We embed each word

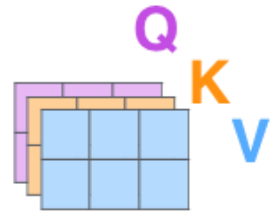
3) We multiply X with weight matrices

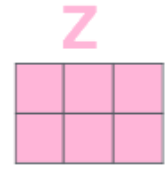
4) Calculate attention using the resulting Q/K/V matrices

Thinking
Machines



# Digression: Transformer Elements

## *Self-Attention*

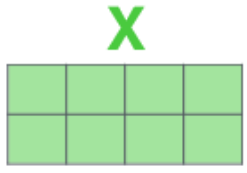
1) This is our input sentence

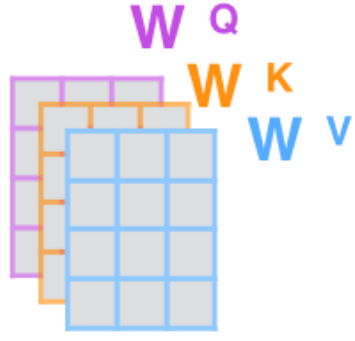
2) We embed each word

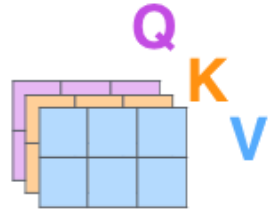
3) We multiply X with weight matrices

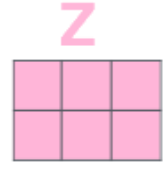
4) Calculate attention using the resulting Q/K/V matrices

Thinking
Machines



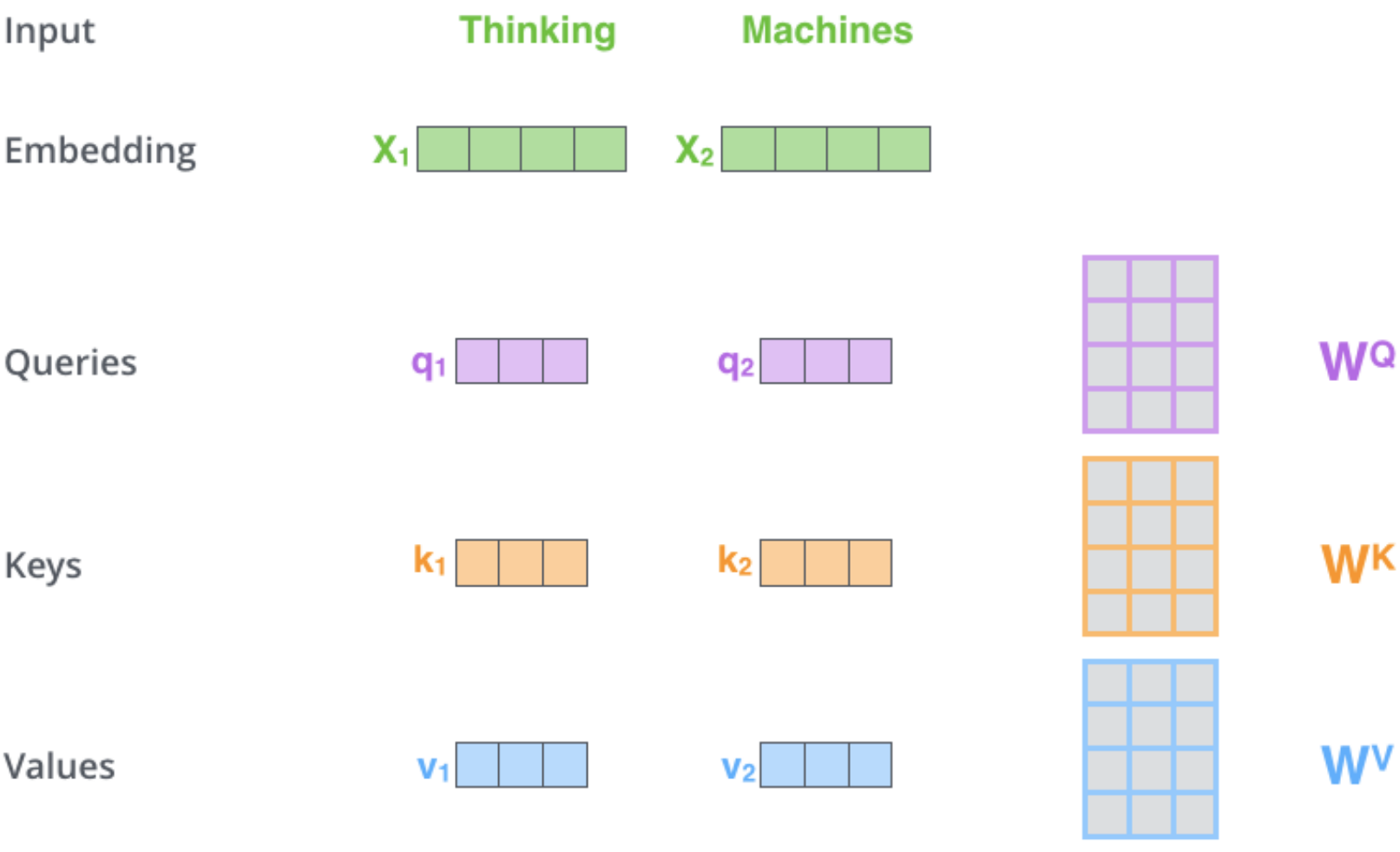




$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

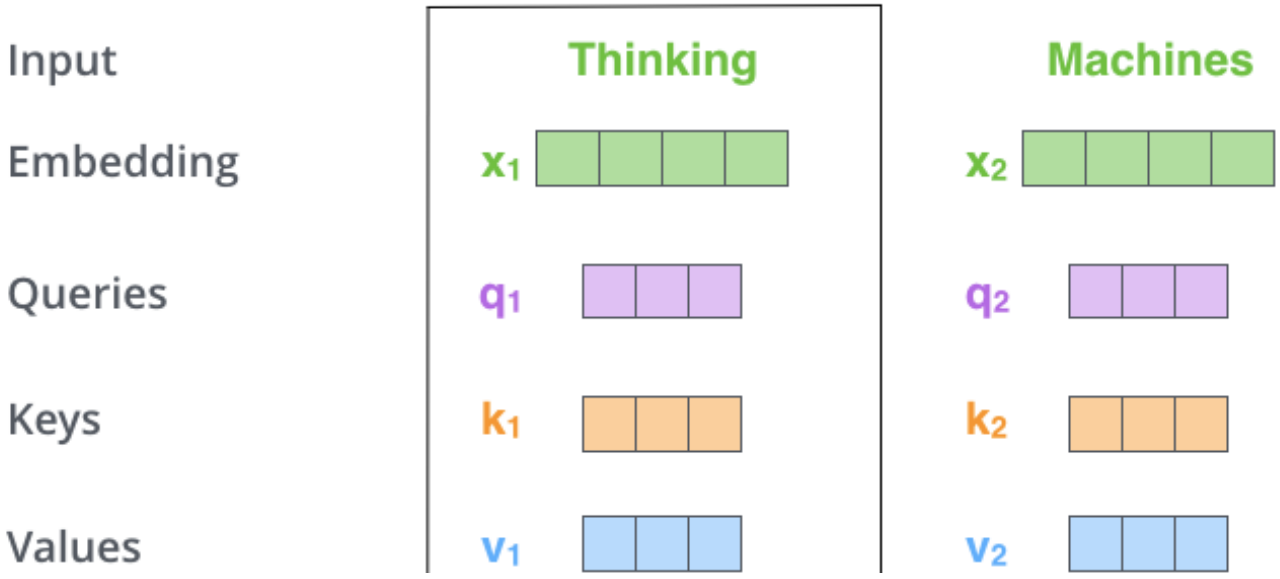
# Digression: Transformer Elements

## *Self-Attention*



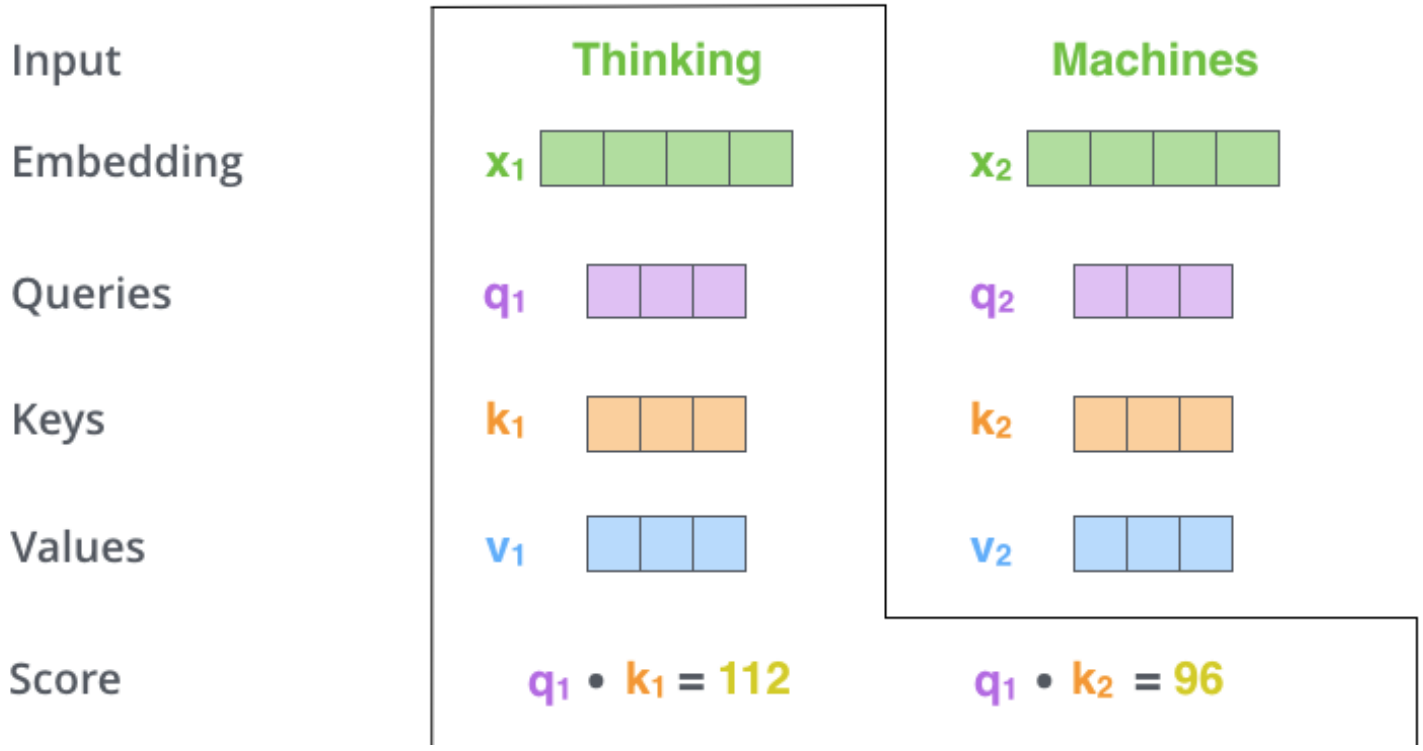
# Digression: Transformer Elements

## *Self-Attention*



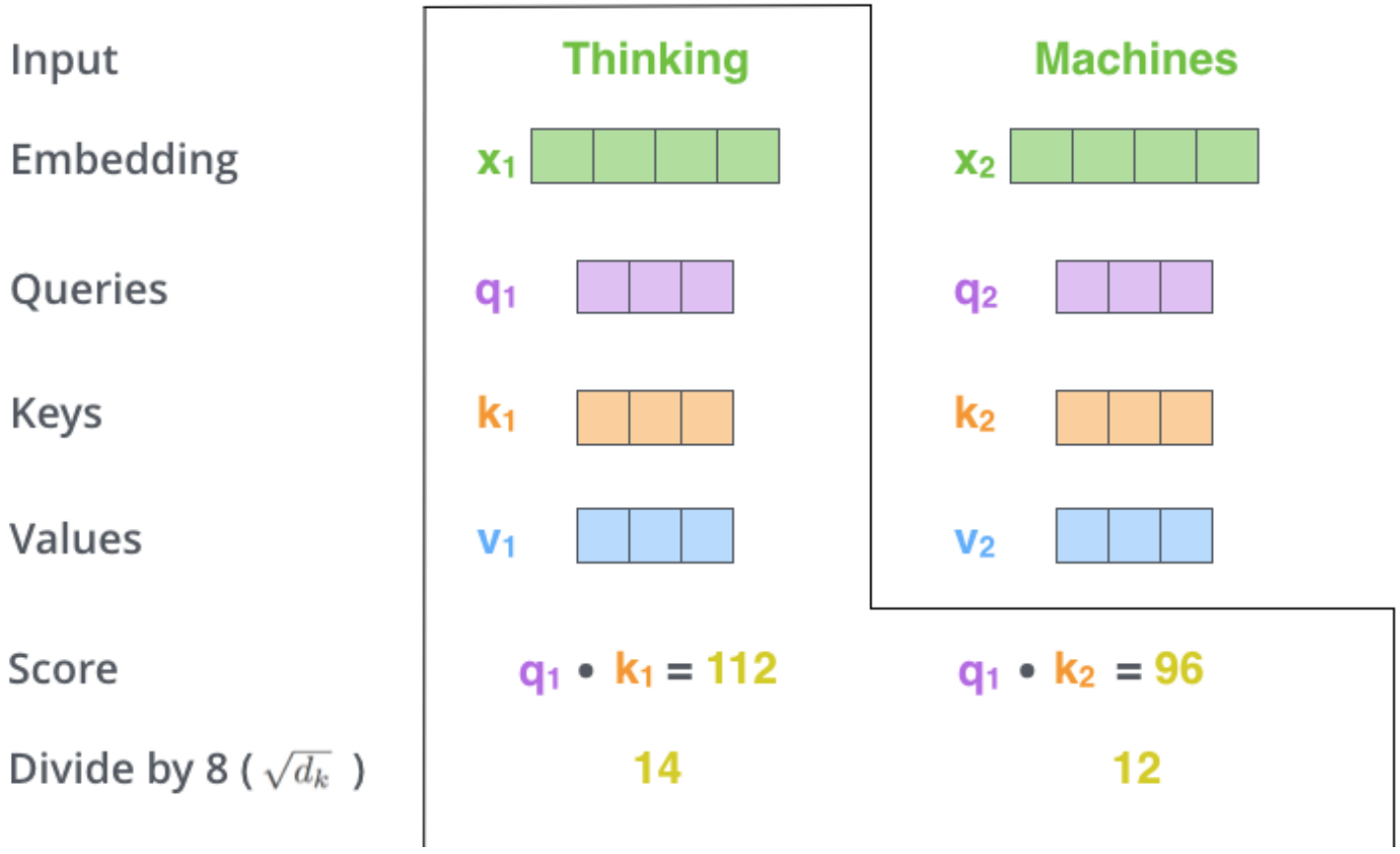
# Digression: Transformer Elements

## *Self-Attention*



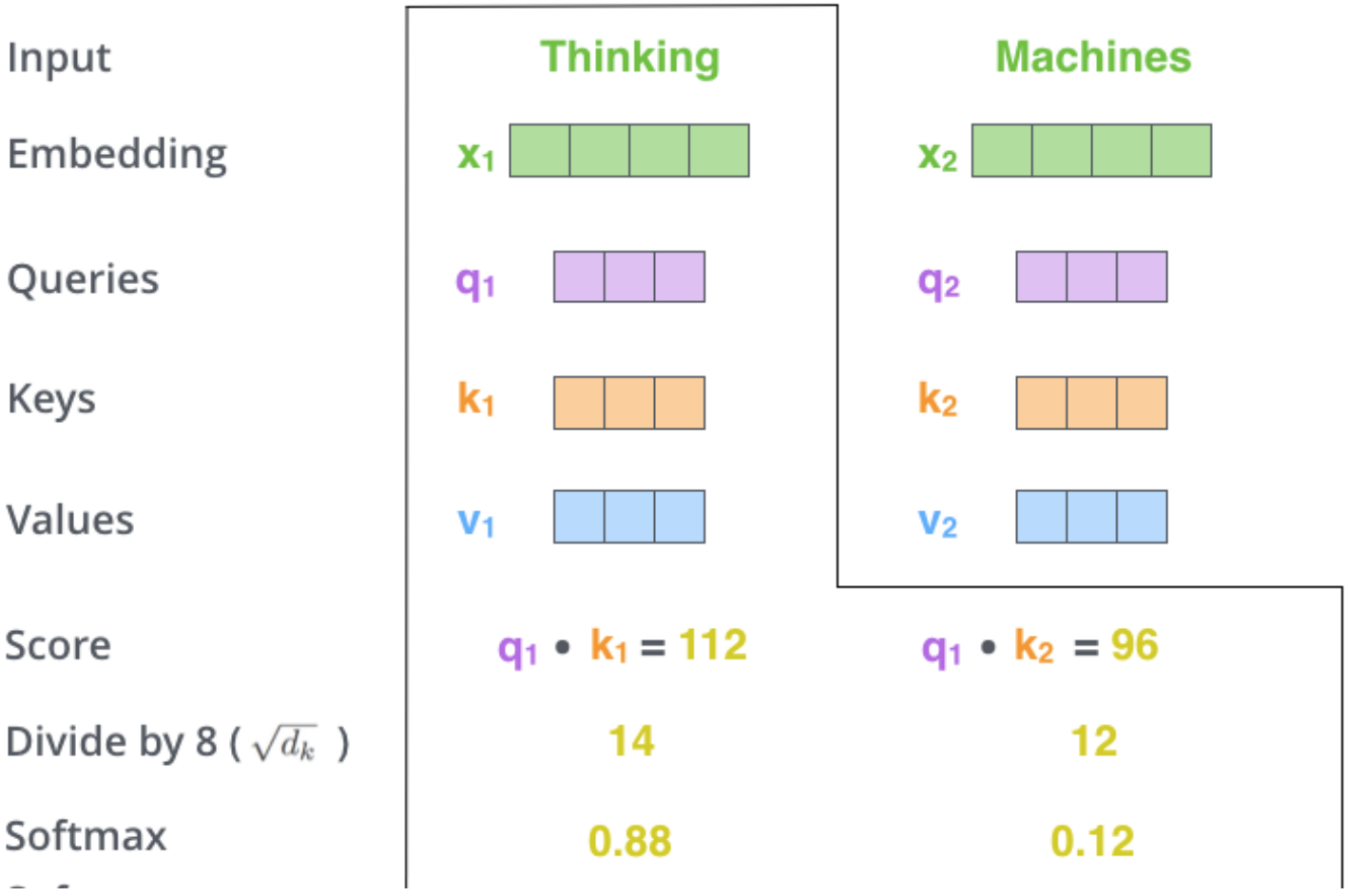
# Digression: Transformer Elements

## *Self-Attention*



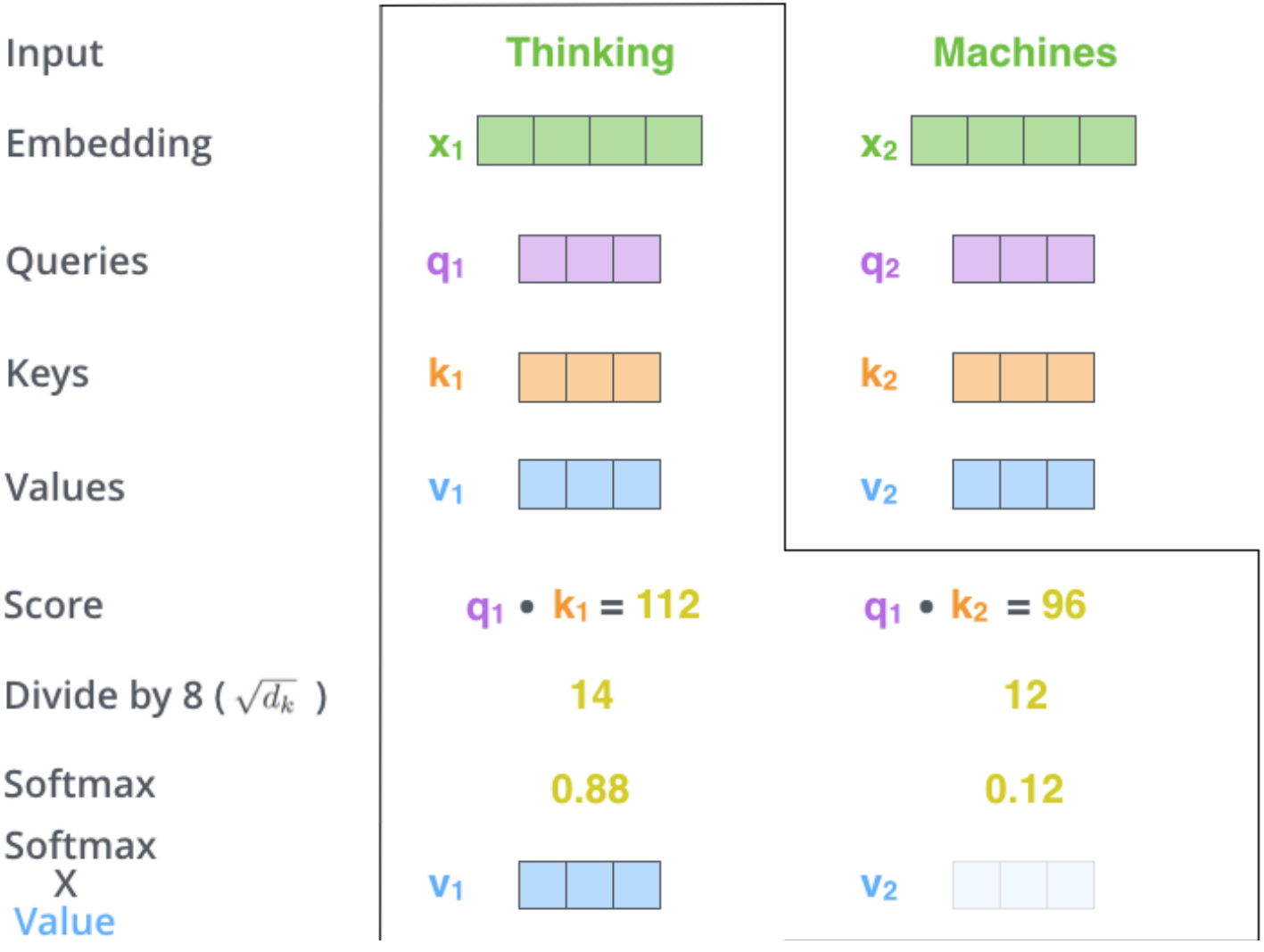
# Digression: Transformer Elements

## *Self-Attention*



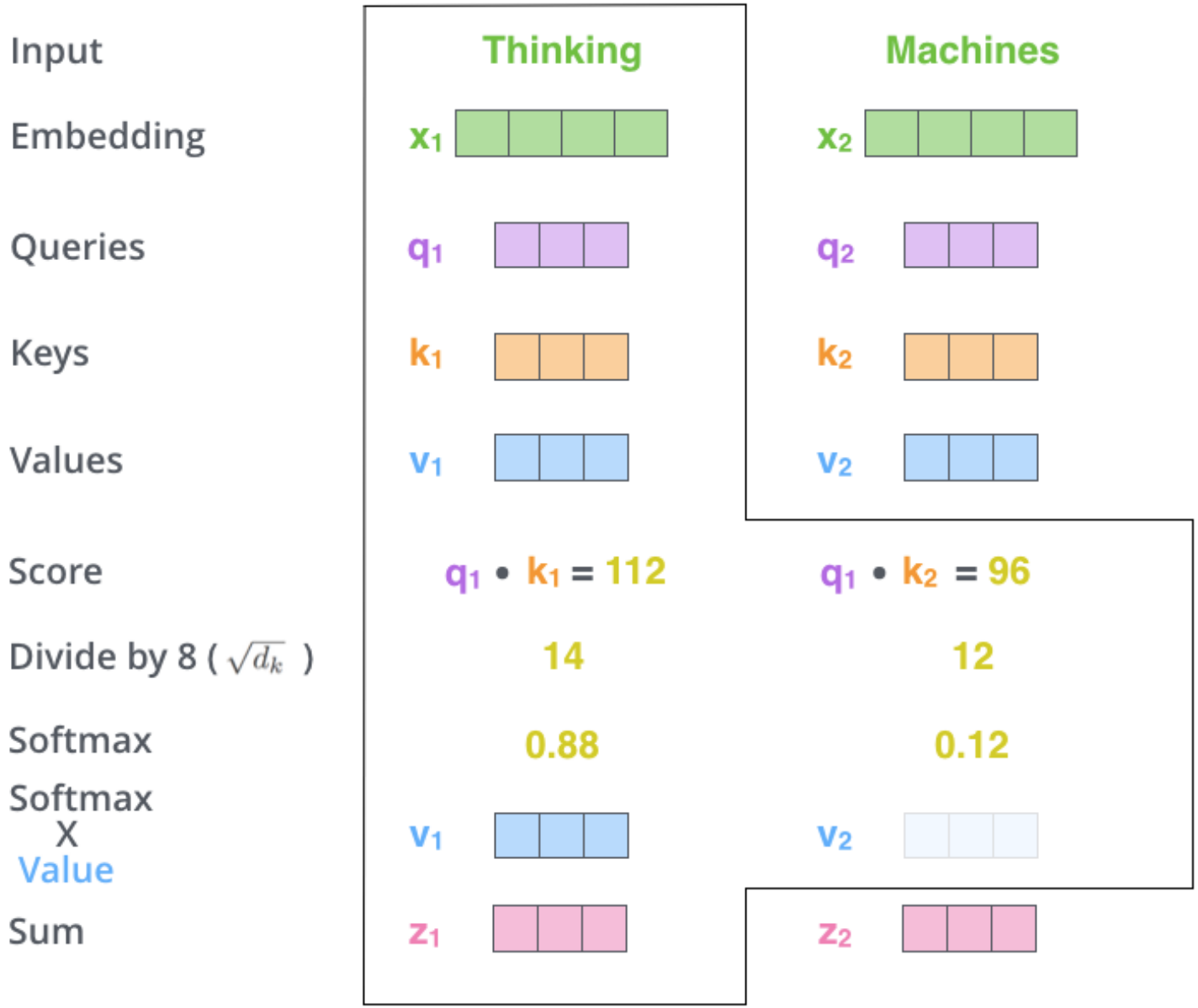
# Digression: Transformer Elements

## *Self-Attention*



# Digression: Transformer Elements

## *Self-Attention*



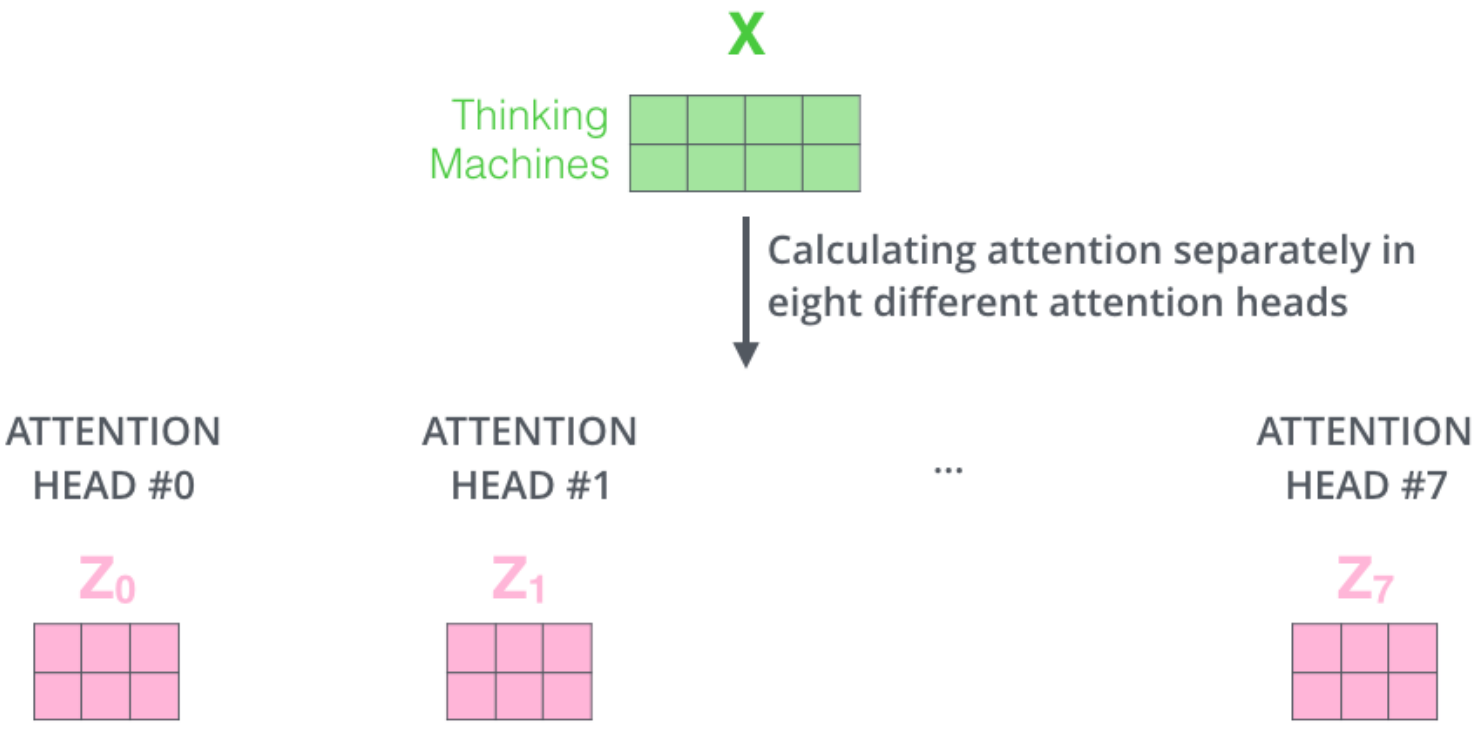
# Digression: Transformer Elements

## *Multi-head Attention*

- Self-attention with multiple heads
- Each head (weights $W^Q$, $W^K$ and $W^V$) is initialised independently
- Project inputs into different representation subspaces
- Transformer models [Vaswani et al., 2017] use 8 heads

# Digression: Transformer Elements

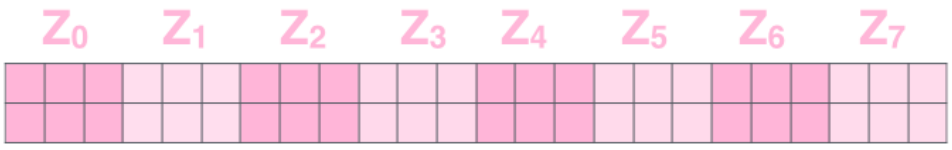
## *Multi-head Attention*



# Digression: Transformer Elements

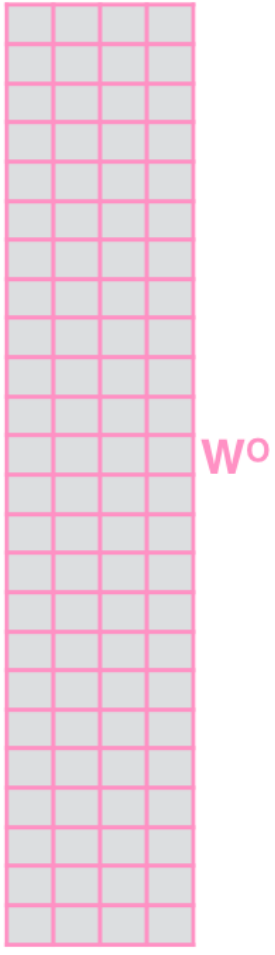
## *Multi-head Attention*

1) Concatenate all the attention heads



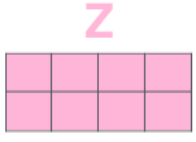

2) Multiply with a weight matrix $W^O$ that was trained jointly with the model

X




3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



# Digression: Transformer Elements

## *Multi-head Attention*

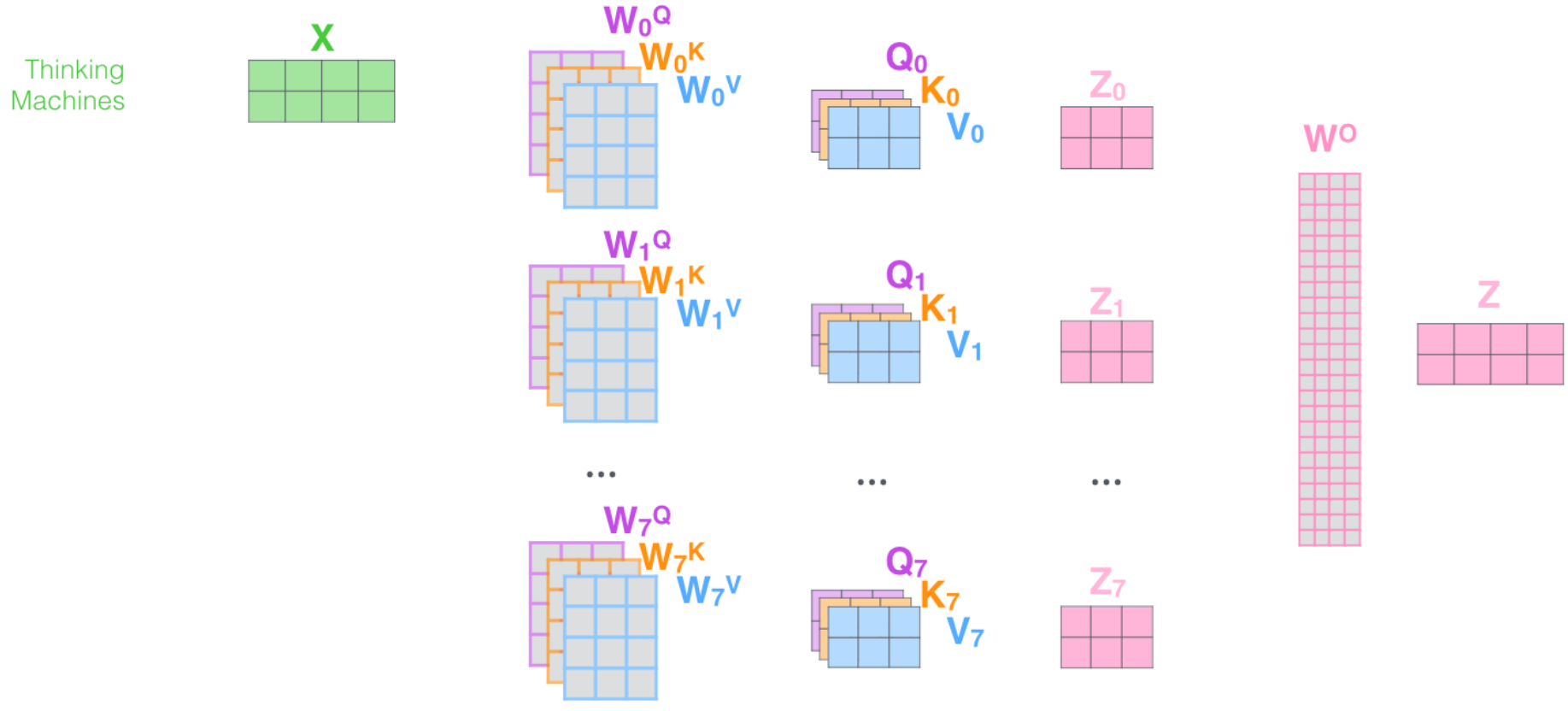
1) This is our input sentence

2) We embed each word

3) Split into 8 heads. We multiply X with weight matrices

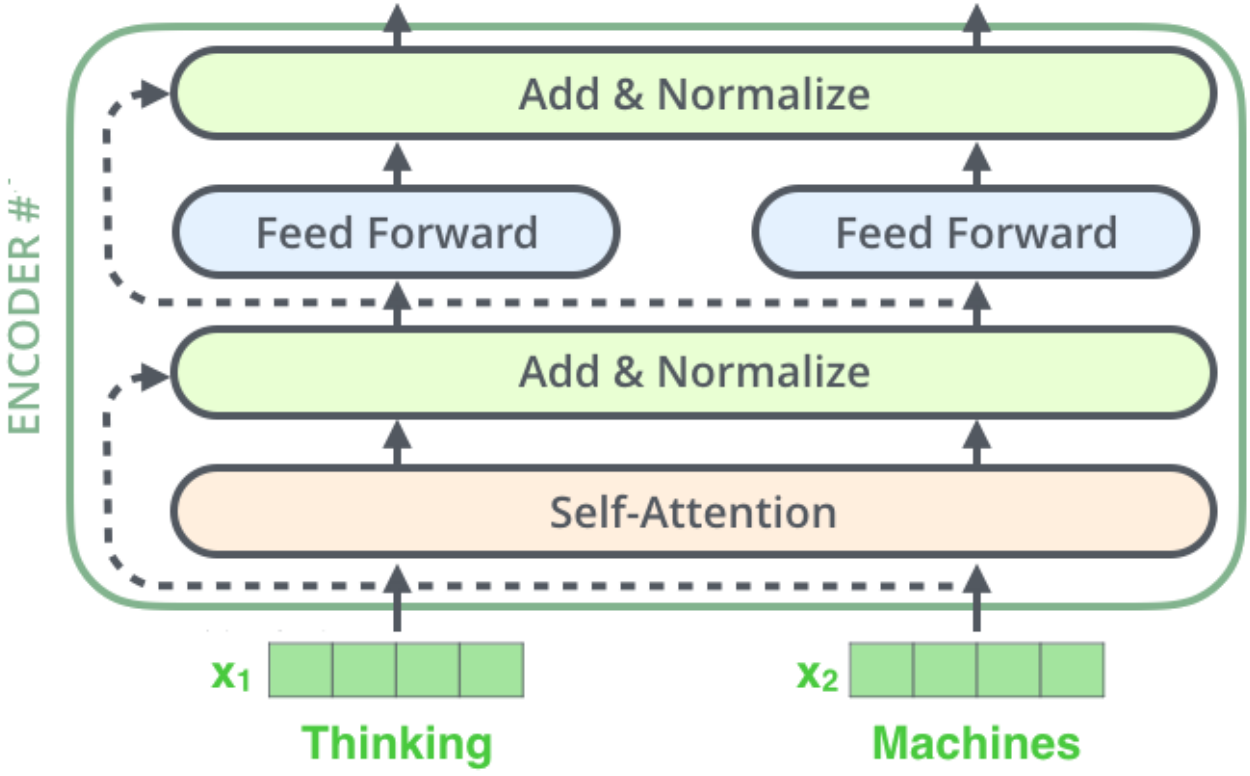
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix $W^O$ to produce the output of the layer



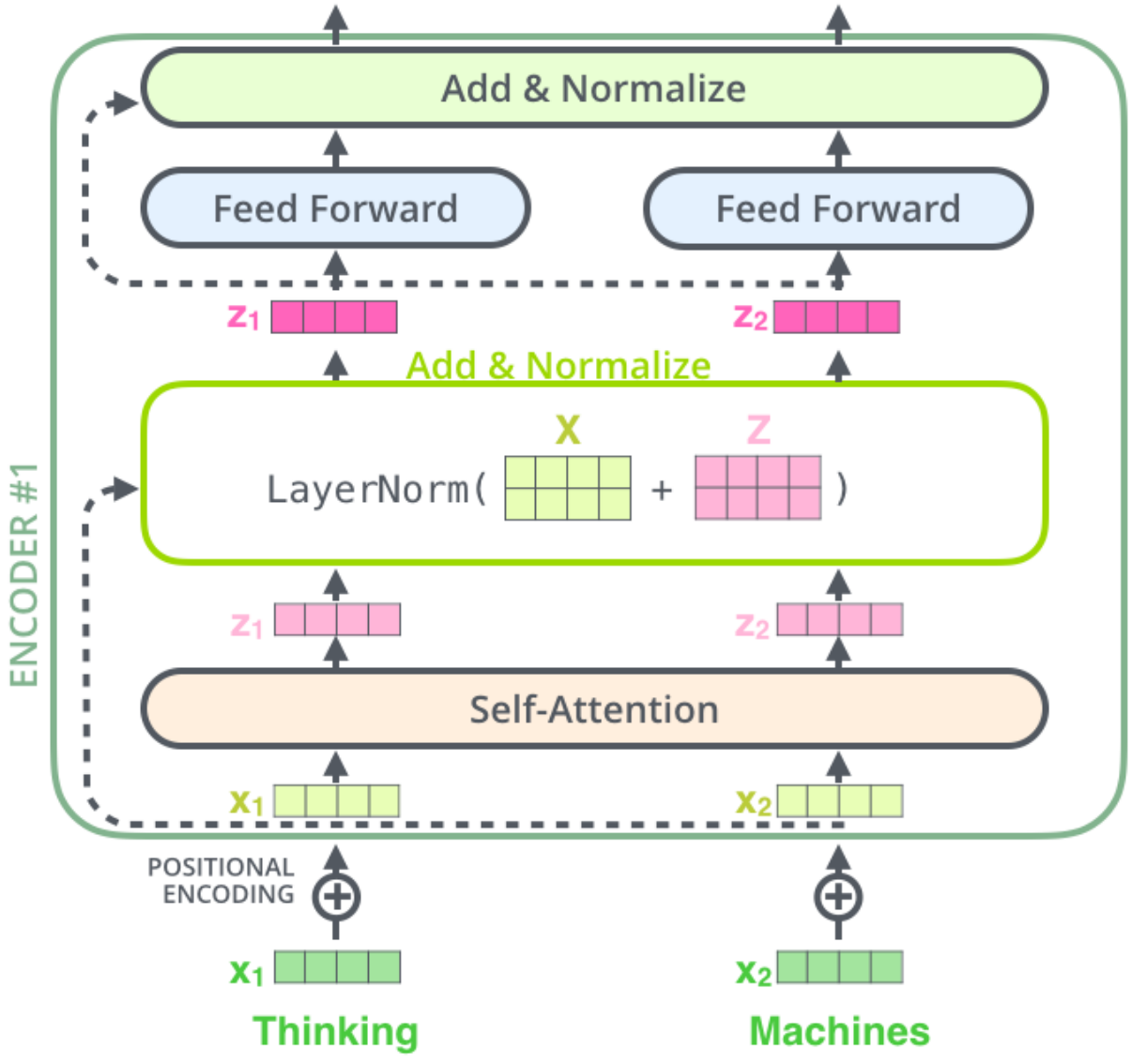
# Digression: Transformer Elements

## *An Encoder Layer*



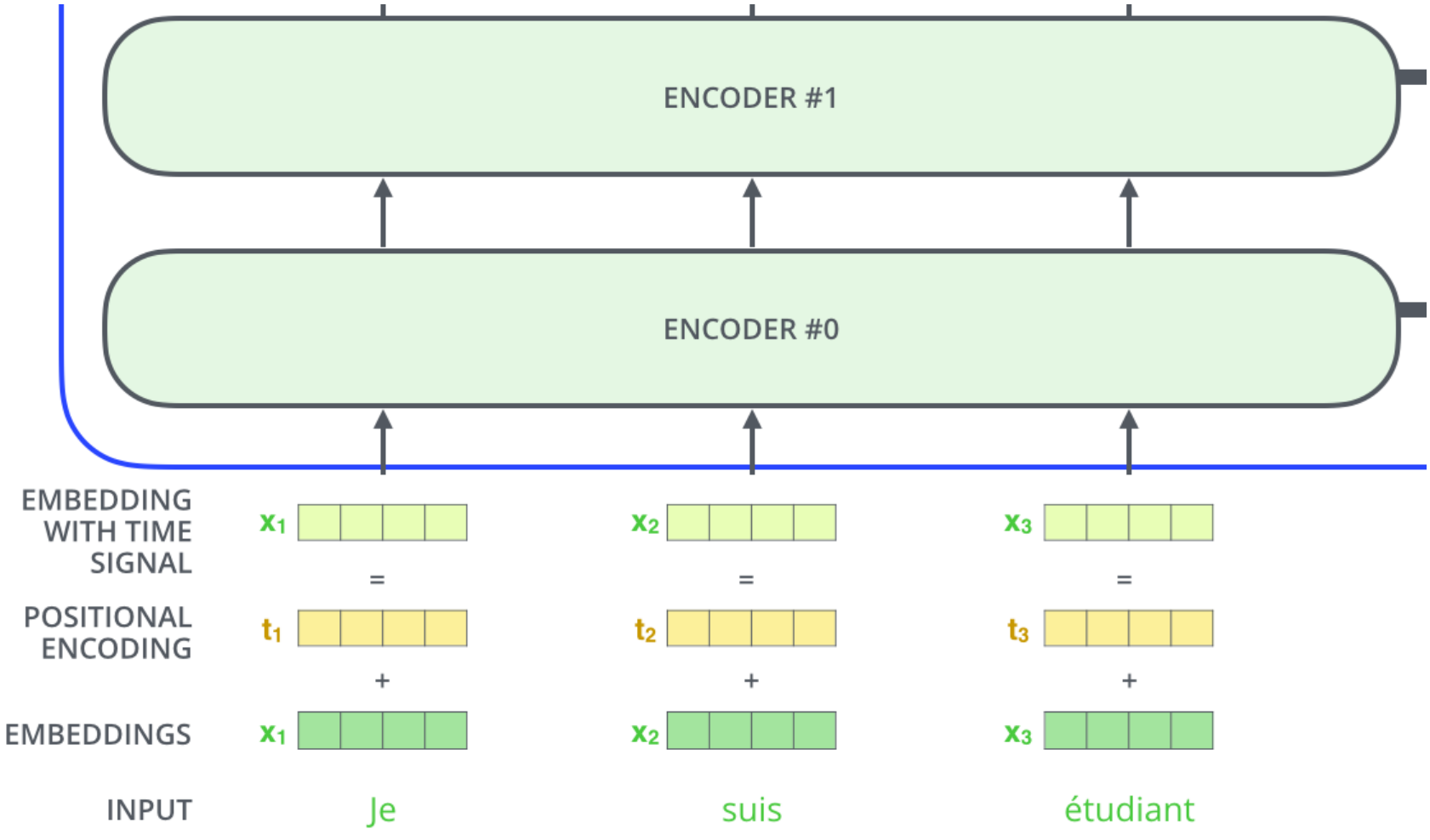
# Digression: Transformer Elements

## *An Encoder Layer: Residuals*



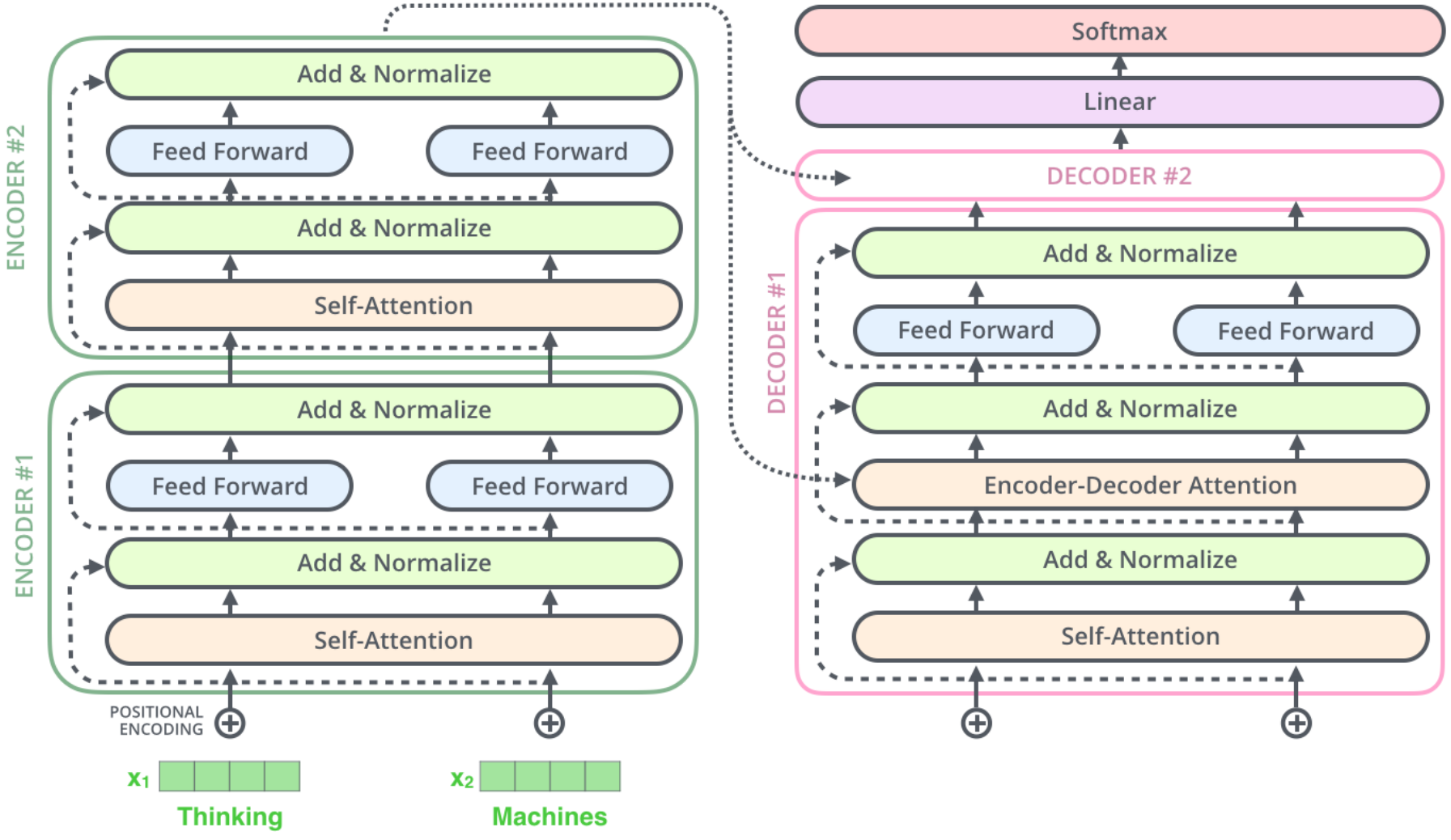
# Digression: Transformer Elements

## *Positional Embeddings*



# Digression: Transformer Elements

## *Toy Transformer (more on the NMT lecture!)*



# Neural Approaches

## *Stop!*

**Transformer**

**Encoder**

**Decoder**

**Self-attention**

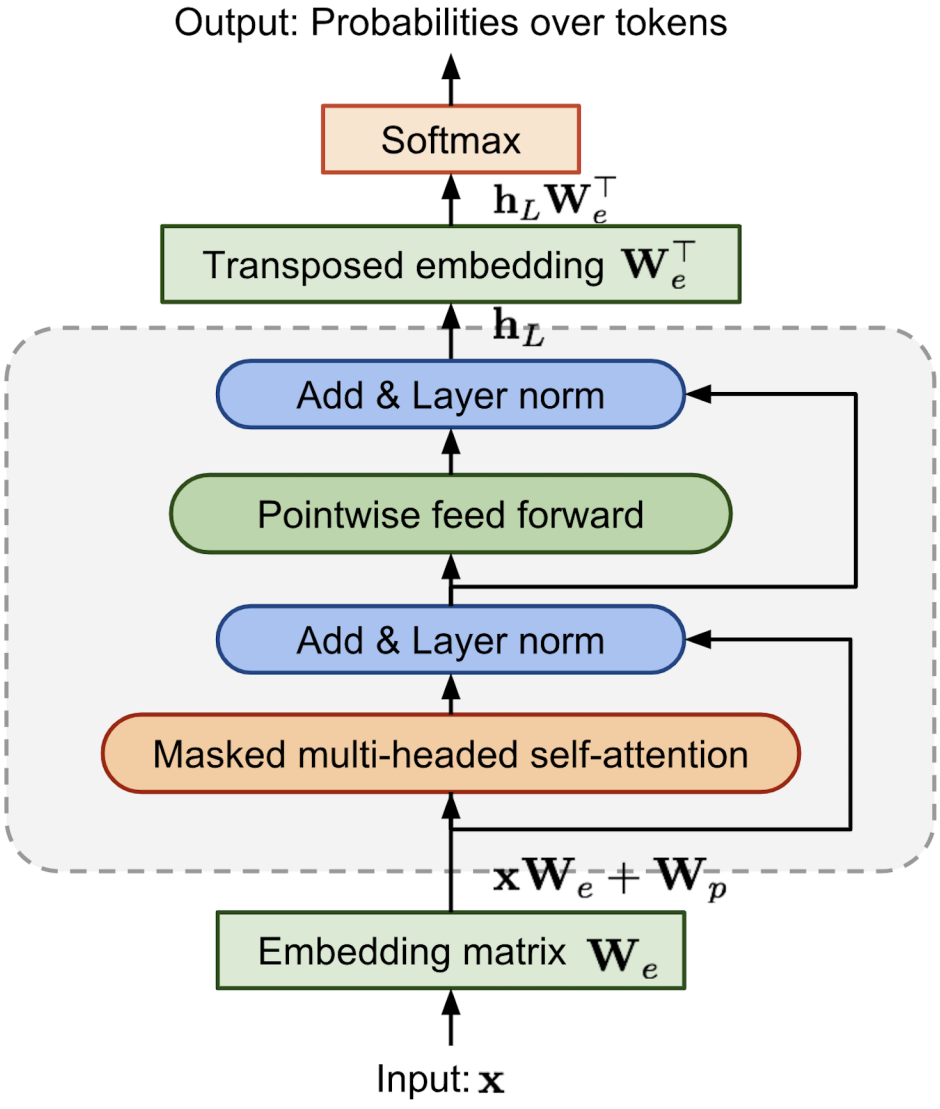
**Multi-head attention**

State-of-the art in **most** NLP tasks, **want details!!**

**Got them!!**

# Neural Approaches

## *Transformer Decoder Language Model [Radford et al., 2018]*




**Transformer Block**
Repeat x L=12

$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1})$$
$$\ell = 1, \ldots, L$$

# Neural Approaches

## *Transformer Decoder Language Model [Radford et al., 2018]*

### OpenAI GTP(-2)

- Large transformer-based language model with 1.5 billion parameters
- Trained on a dataset of 8 million web pages (40 GB)
- Objective: predict the next word, given all of the previous words within some text

`https://openai.com/blog/better-language-models/`

# Neural Approaches

## *Transformer Decoder Language Model [Radford et al., 2018]*

| | |
|---|---|
| SYSTEM PROMPT (HUMAN-WRITTEN) | *Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.* |
| MODEL COMPLETION (MACHINE-WRITTEN, SECOND TRY) | The 19-year-old singer was caught on camera being escorted out of the store by security guards.<br><br>The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back.<br><br>Scroll down for video<br><br>Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured)<br><br>The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back<br><br>The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals.<br><br>She was carrying a pair of black and white striped gloves and a small black bag. |

# Neural Approaches

## *After Basic FFNs, RNNs and Transformers LMs...*

**Things we should cover (with more time!)**

- LSTM cells
- Character-level LMs
- Bidirectional RNNs (ELMo)
- Combination of models
- ...

# Software & References

# Software & References

## *Libraries & Packages*

- **KenLM**
  `https://kheafield.com/code/kenlm/`
- **SRILM**
  `http://www.speech.sri.com/projects/srilm/download.html`
- **RNNLM**
  `https://github.com/mspandit/rnnlm`
- **CSLM**
  `https://github.com/hschwenk/cslm-toolkit/blob/master/README`

# Software & References

## *References Used*


- Reinhard Kneser and Hermann Ney. **Improved Backing-off for N-gram Language Modeling**. Proceedings ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing. Pages 181–184, 1995.

- Daniel Jurafsky and James H. Martin. 2018. **Speech and Language Processing.** Chapter 3: N-gram Language Models.

- Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. **A Neural Probabilistic Language Model**. Journal of Machine Learning Research, volume 3, pages 1137–1155, 2003.

# Software & References

## *References Used*


- Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, Jan "Honza" Cernocky. **Empirical Evaluation and Combination of Advanced Language Modeling Techniques**. 12th Annual Conference of the International Speech Communication Association, pages 605–608, 2011.

- Tomas Mikolov. **Statistical Language Models Based on Neural Networks**. PhD. Thesis. 2012.

- Dengliang Shi. **A Study on Neural Network Language Modeling**. *eprint arXiv:1708.07252*

# Everything fine?

**Questions?**

# Language Modelling

**Cristina España-Bonet**

**UdS & DFKI, Saarbrücken, Germany**

Artificial Intelligence with Deep Learning

25th April 2019

# Extra Slides

## *The language model $P(e)$ examples*

Examples

`http://files.asimihsan.com/courses/nlp-coursera-2013/notes/nlp.html`

# Extra Slides

## *The language model $P(e)$ examples*

**Example 1:** 3-gram language model with linear interpolation

- Consider the weights

$$\lambda_1 = \lambda_2 = \lambda_3 = 1/3;\ \lambda_0 = 0$$

- Consider the corpus
  - the green book STOP
  - my blue book STOP
  - his green house STOP
  - book STOP

# Extra Slides

## *The language model $P(e)$ examples*

**Example 1:** 3-gram language model with linear interpolation

$P(\text{book}|\text{the, green})$?

$P(\text{book}|\text{the, green}) = 0.571$

# Extra Slides

## *The language model $P(e)$ examples*

**Example 1:** 3-gram language model with linear interpolation

$P(\text{book}|\text{the, green})$?

$P(\text{book}|\text{the, green}) = 0.571$

$$\frac{1}{3}P(\text{book}|\text{the, green}) + \frac{1}{3}P(\text{book}|\text{green}) + \frac{1}{3}P(\text{book}) =$$

$$\frac{1}{3}\frac{\text{Count(the,green,book)}}{\text{Count(the,green)}} + \frac{1}{3}\frac{\text{Count(green,book)}}{\text{Count(green)}} + \frac{1}{3}\frac{\text{Count(book)}}{\text{Count()}} =$$

$$\frac{1}{3} \times \frac{1}{1} + \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \frac{3}{14}$$

# Extra Slides

## *The language model $P(e)$ examples*

**Example 2:** Discounting

| x | Count(x) | $P$(wi\|wi-1) | Count*(x) | $P$*(wi\|wi-1) |
|---|---|---|---|---|
| the | 48 | | | |
| the, dog | 15 | 15/48 | 14.5 | 14.5/48 |
| the, woman | 11 | 11/48 | 10.5 | 10.5/48 |
| the, man | 10 | 10/48 | 9.5 | 9.5/48 |
| the, park | 5 | 5/48 | 4.5 | 4.5/48 |
| the, job | 2 | 2/48 | 1.5 | 1.5/48 |
| the, telescope | 1 | 1/48 | 0.5 | 0.5/48 |
| the, manual | 1 | 1/48 | 0.5 | 0.5/48 |
| the, afternoon | 1 | 1/48 | 0.5 | 0.5/48 |
| the, country | 1 | 1/48 | 0.5 | 0.5/48 |
| the, street | 1 | 1/48 | 0.5 | 0.5/48 |

# Extra Slides

## *The language model P(e) examples*

**Example 2:** Discounting

Discounted counts:

$$\text{Count*(x)=Count(x)-constant}$$

Left-over probability mass:

$$\alpha(w_i - 1) = 1 - \sum_w \frac{\text{Count}^*(\text{w}_{i-1},\text{w})}{\text{Count}(\text{w}_{i-1})}$$

In the example: $\alpha(\text{the}) = 10 \times \frac{0.5}{48} = \frac{5}{48}$