CS 489/698: Introduction to Natural Language Processing

Lecture 5: Neural Networks – Common Architectures

Instructor: Freda Shi

fhs@uwaterloo.ca

January 19th, 2026

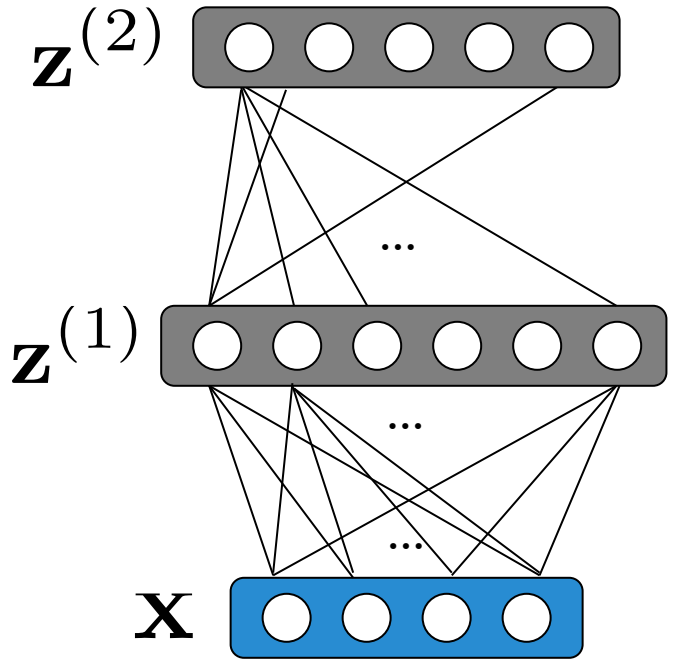
UNIVERSITY OF WATERLOO

Recap: Multi-Layer Perceptron

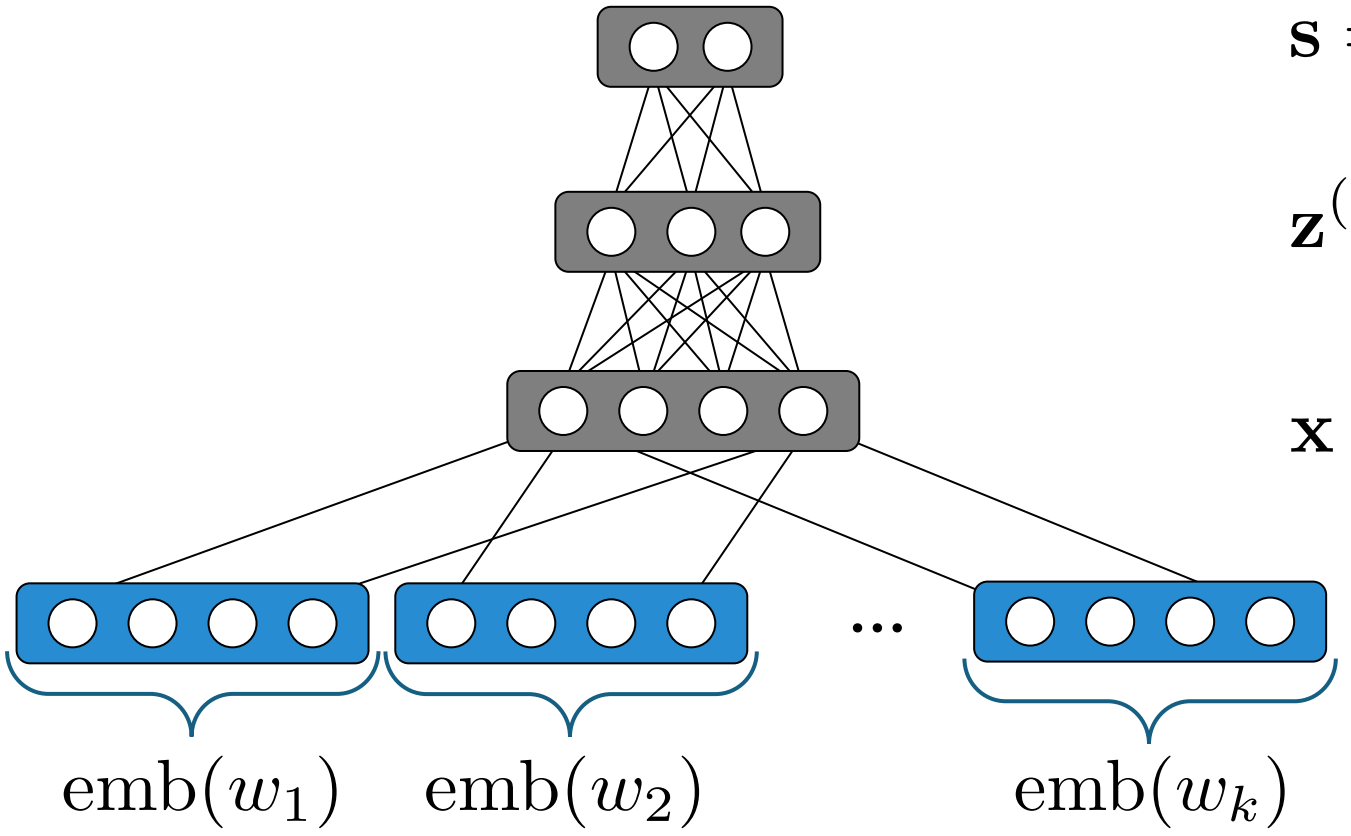
$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{z}^{(2)} = g\left(\mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\dots$$



Recap: Simple Neural Classifier with BoW Feature



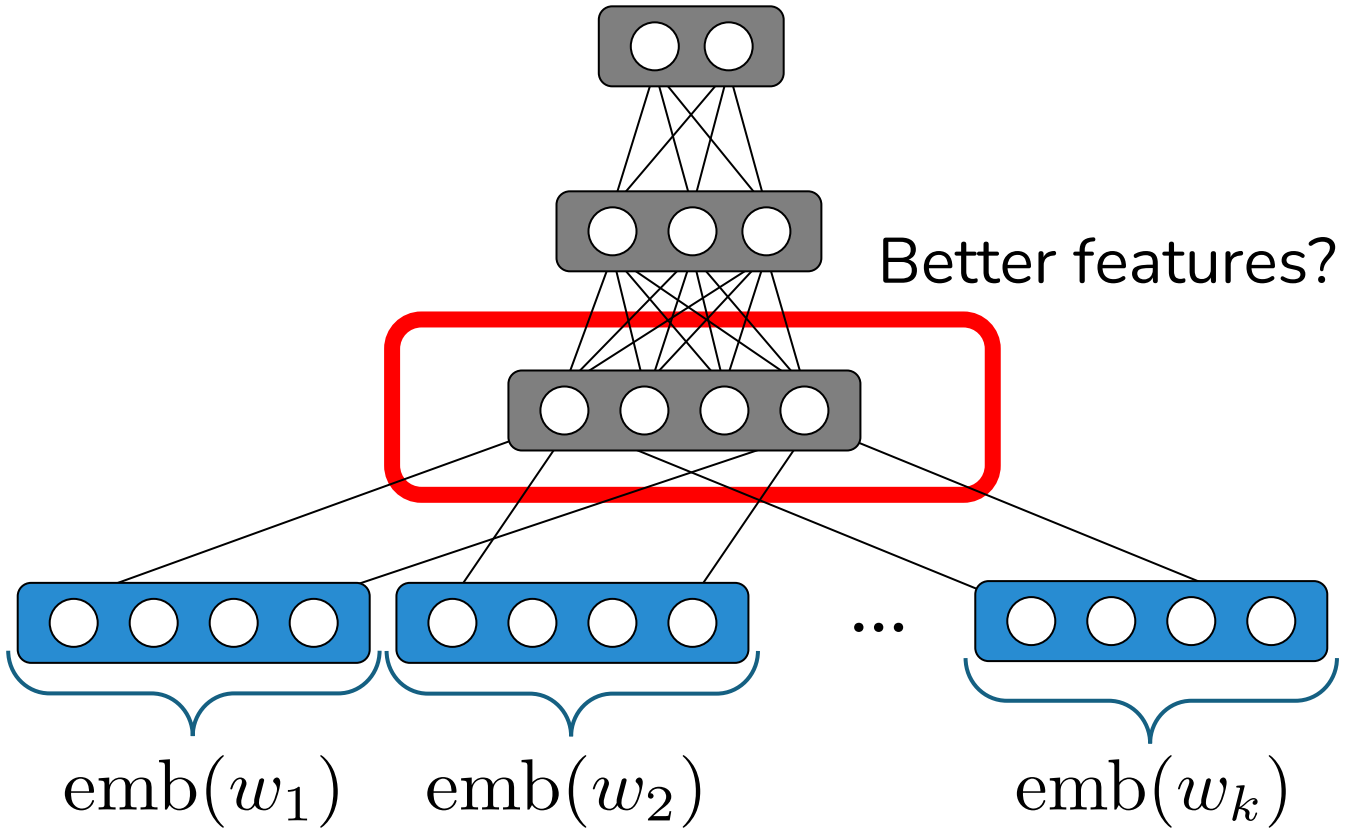
$$\mathbf{s} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{x} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{emb}(w_k)$$

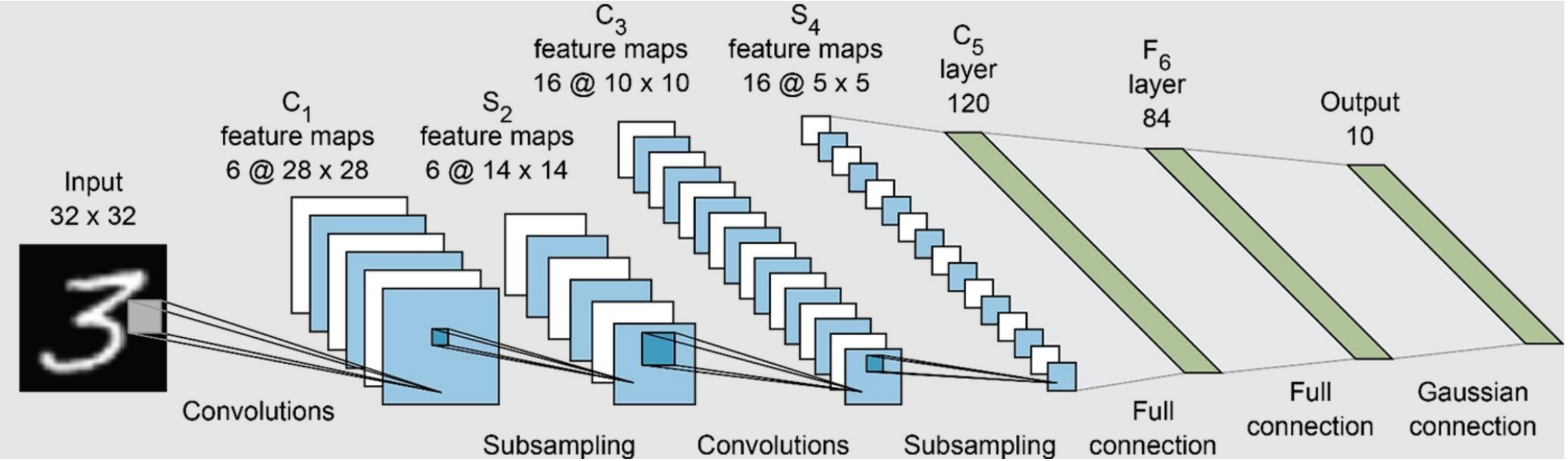
This Lecture: Common Neural Architectures

- Convolutional neural networks
- Recurrent neural networks
- Recursive neural networks
- Transformers

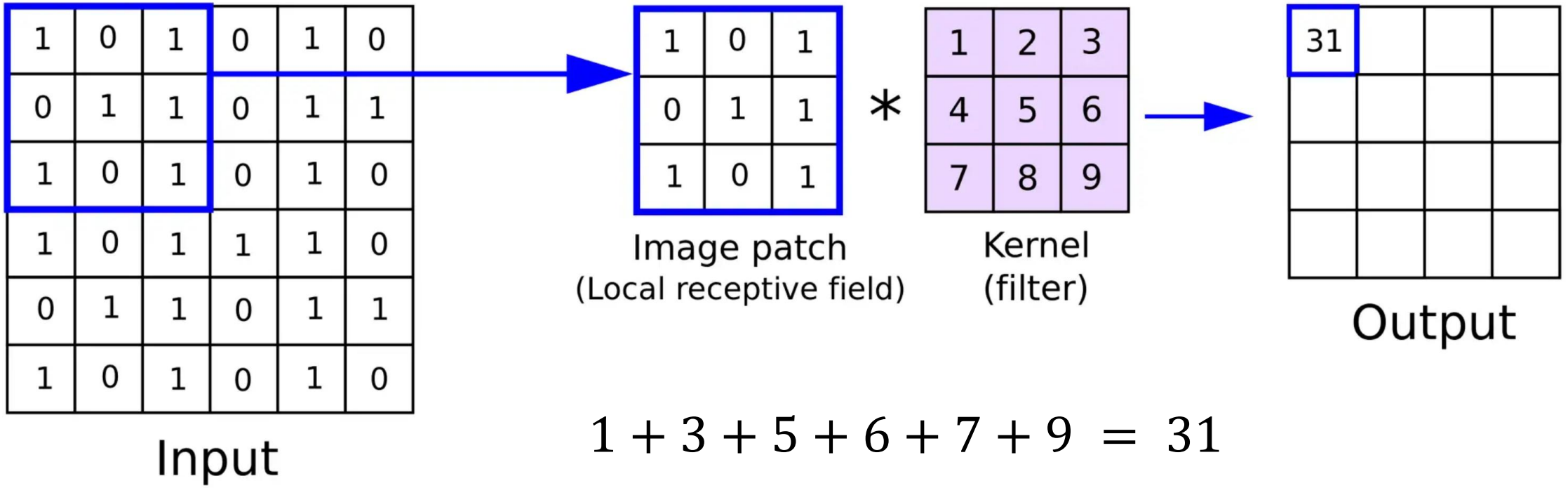


Convolutional Neural Networks

Introduced for vision tasks; also used in NLP to extract feature vectors.

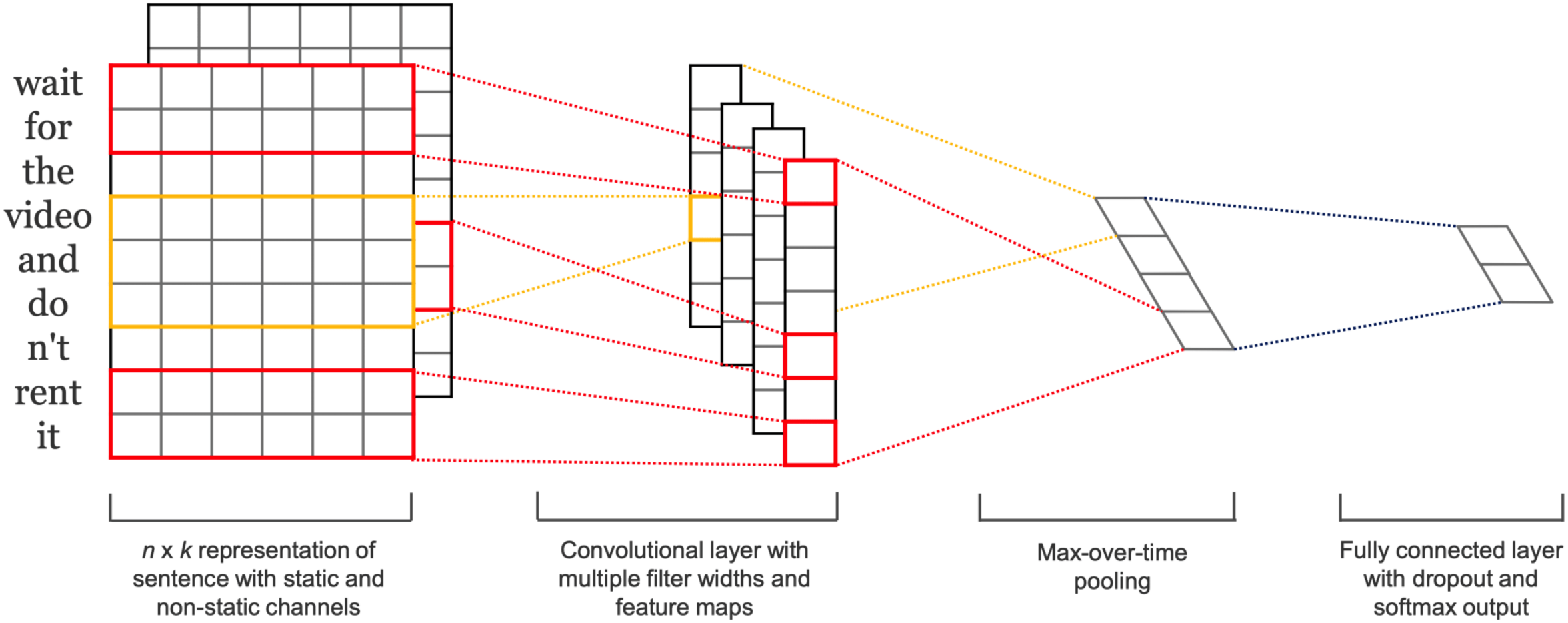


Convolutional Neural Networks



$$1 + 3 + 5 + 6 + 7 + 9 = 31$$

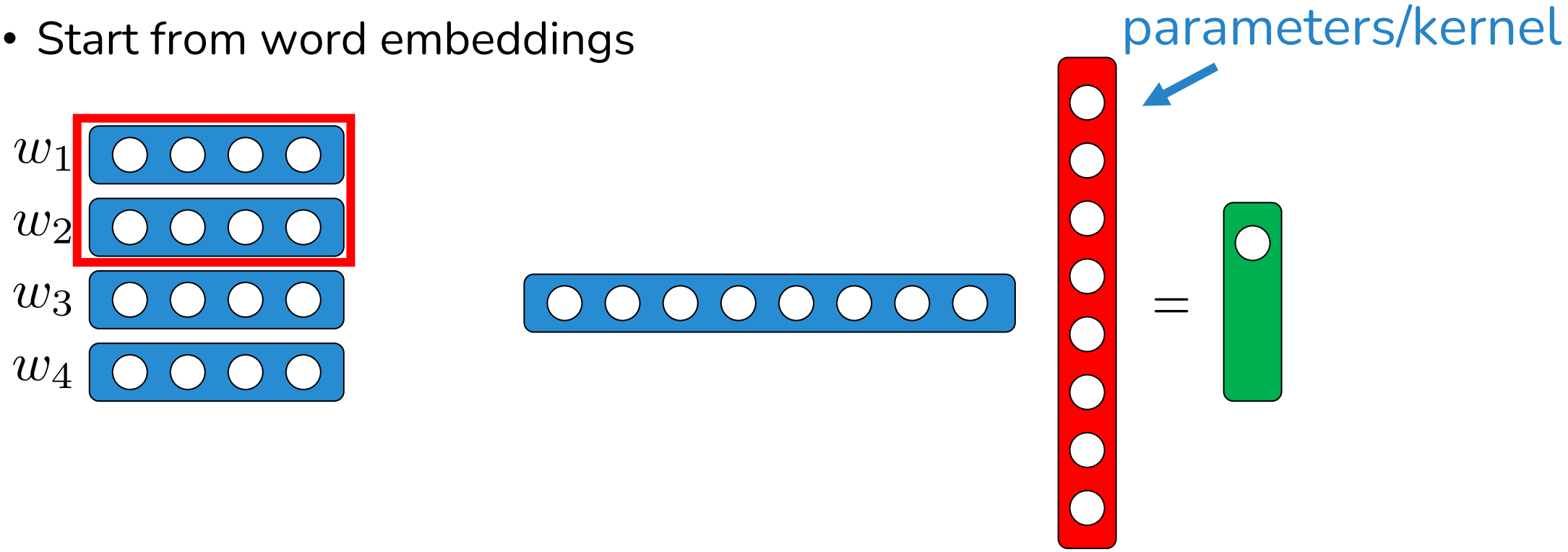
From 2D to 1D: Overview



Source: Y. Kim.(2014). Convolutional Neural Networks for Sentence Classification

Kernel/Filter

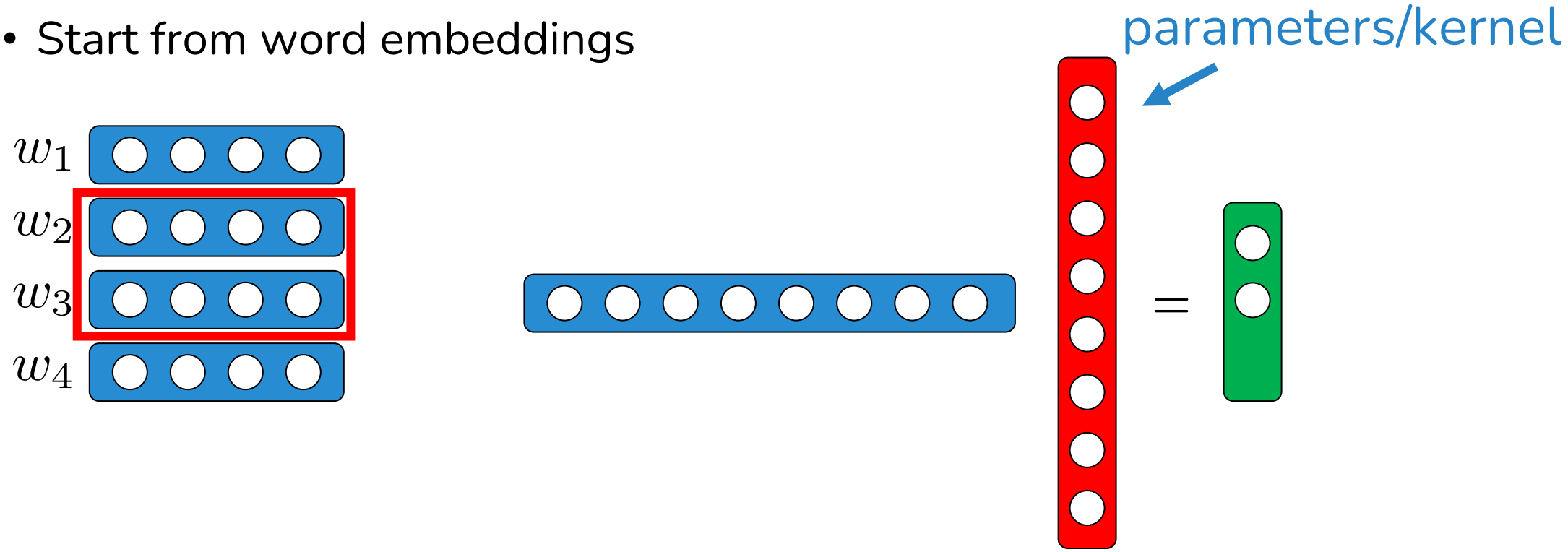
- Start from word embeddings



- Take dot product between filter and (stretched) word embeddings

Kernel/Filter

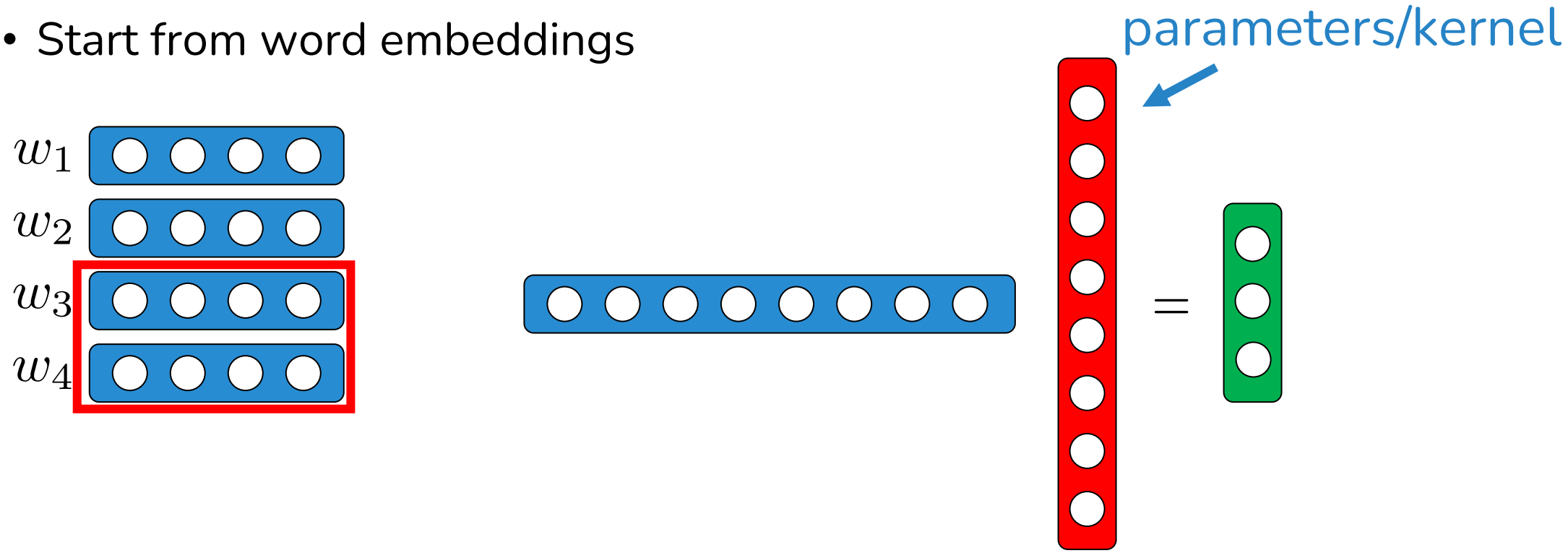
- Start from word embeddings



- Take dot product between filter and (stretched) word embeddings

Kernel/Filter

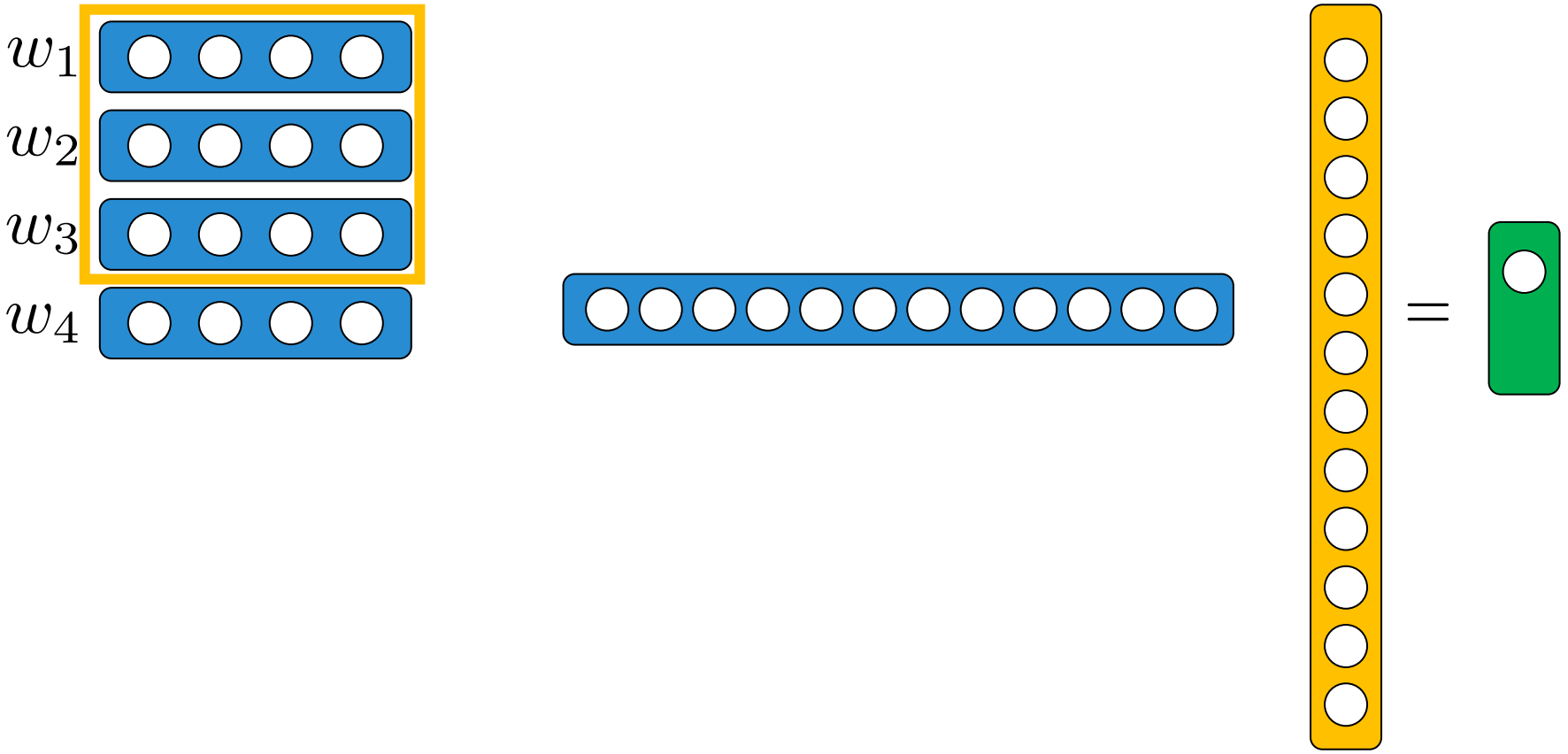
- Start from word embeddings



- Take dot product between filter and (stretched) word embeddings

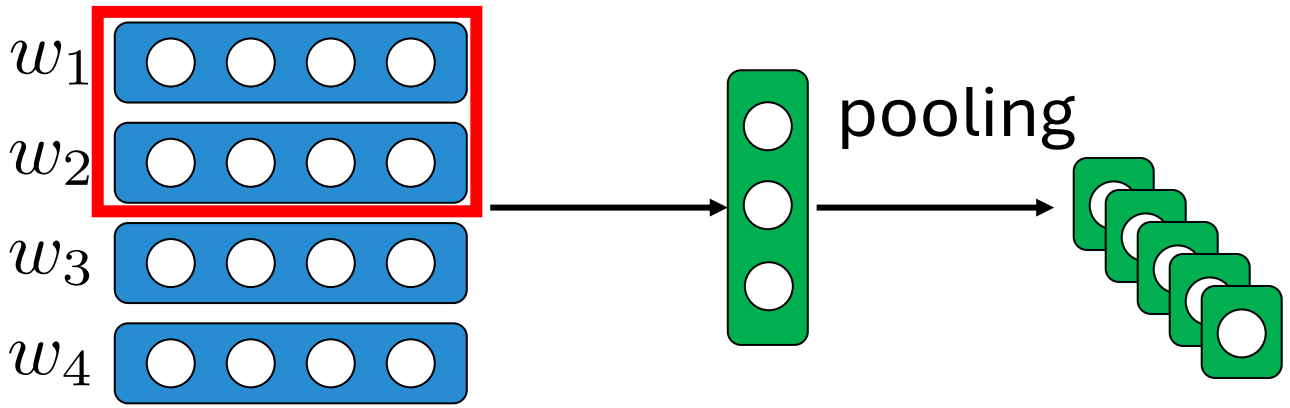
Kernel/Filter

- What about a kernel/filter with a different size?



Pooling

- Each kernel/filter extracts one type of features
- However, a kernel's output size depends on sentence length
  A **fixed dimensional** vector is desirable for MLP inputs
- Solution: mean pooling/max pooling converts a vector to a scalar
- Final feature: concatenating pooling results of all filters



Summary: Convolutional Neural Networks for NLP

- Word order matters
  Example (kernel size = 2):
  a cat drinks milk → *(a cat)*, *(cat drinks)*, *(drinks milk)*
  a milk drinks cat → *(a milk)*, *(milk drinks)*, *(drinks cat)*
- An n-gram “matches” with a kernel when they have high dot product.

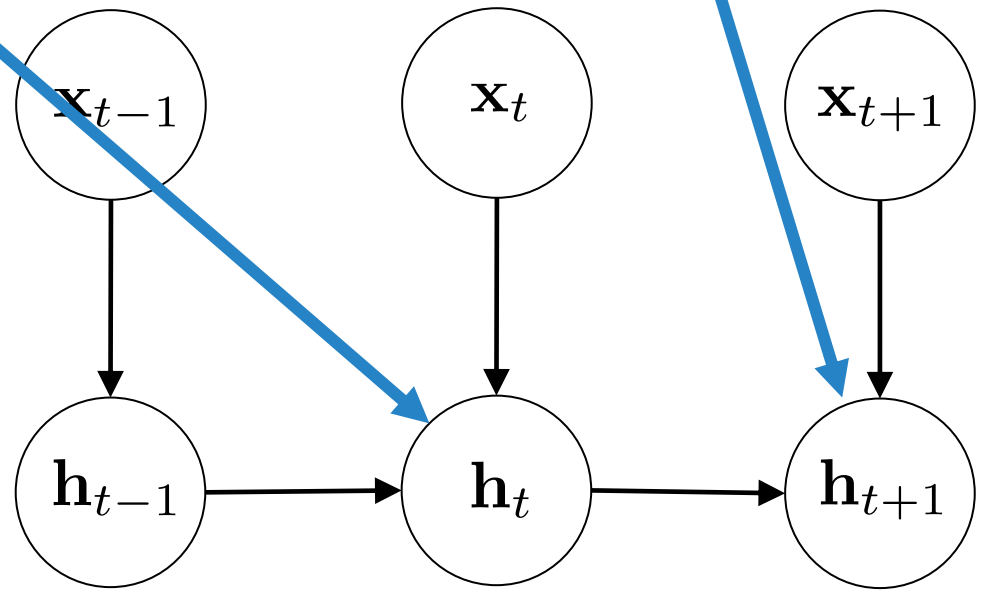
Drawbacks?

- Cannot capture long-term dependency.
- Often used for character-level processing: filters look at character n-grams.

Recurrent Neural Networks (RNNs)

- Idea: apply the same transformation to tokens in time order

$$\mathbf{h}_t = \mathbf{W}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b} \qquad \mathbf{h}_{t+1} = \mathbf{W}[\mathbf{x}_{t+1}; \mathbf{h}_t] + \mathbf{b}$$



Recurrent Neural Networks

- Gradient update for $\mathbf{h}_t = \mathbf{W}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}$
- Suppose $\mathbf{h}_k$ is the representation passed to the classifier

  We can easily calculate $\dfrac{\partial \text{loss}}{\partial \mathbf{h}_k}$

- What about $\dfrac{\partial \text{loss}}{\partial \mathbf{W}}$?

$$\frac{\partial \text{loss}}{\partial \mathbf{W}} = \sum_{t=1}^{k} \frac{\partial \text{loss}}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}}$$

$$\frac{\partial \text{loss}}{\partial \mathbf{h}_t} = \frac{\partial \text{loss}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

An important issue of simple RNNs

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b} \\ &= \mathbf{W}[\mathbf{x}_t; (\mathbf{W}[\mathbf{x}_{t-1}; \mathbf{h}_{t-2}] + \mathbf{b})] + \mathbf{b}\end{aligned}$$

- Absolute value of entries grow/vanish exponentially w.r.t. sequence length.

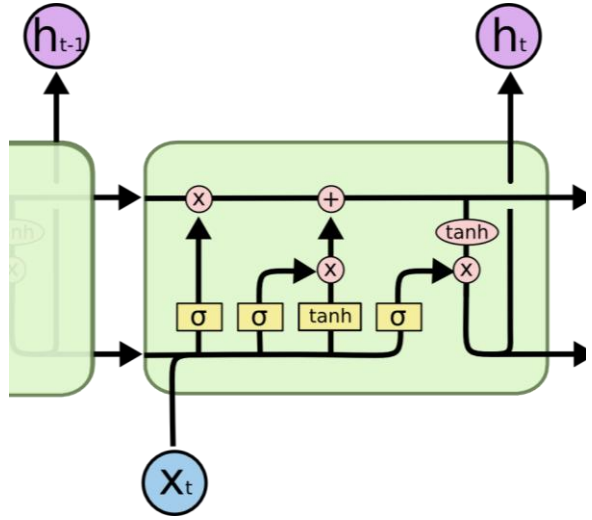
This motivates the development of more advanced RNN architectures.

The Long Short-Term Memory Networks (LSTMs)

Designed to tackle the gradient vanishing problem [Hochreiter and Schmidhuber, 1997]

- Forget gate: $\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_f\right)$
- Input gate: $\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_i\right)$
- Cell: $\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_c\right)$
- Update: $\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{c}}_t$
- Output gate: $\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_o\right)$
- Hidden state: $\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{c}_t)$

Idea: keep entries in $\tilde{\mathbf{c}}_t$ and $\mathbf{h}_t$ in the range of $(-1, 1)$.



[Figure credit: Chris Olah]

Gated Recurrent Units

Fewer parameters and generally works quite well.

- Update gate: $\mathbf{z}_t = \sigma\left(\mathbf{W}_z[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_z\right)$
- Reset gate: $\mathbf{r}_t = \sigma\left(\mathbf{W}_r[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_r\right)$

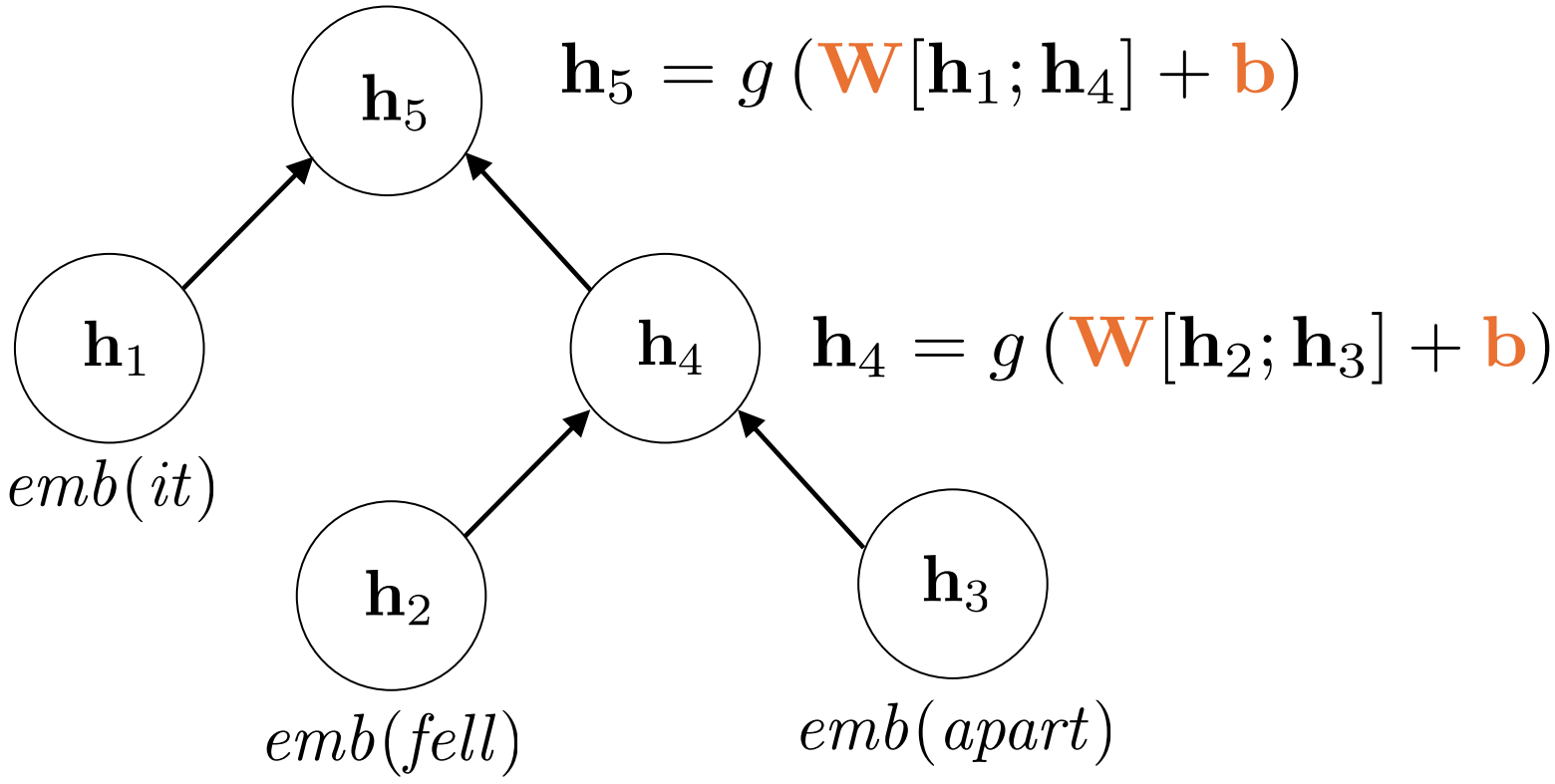
$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \tanh\left(\mathbf{W}[\mathbf{x}_t; \mathbf{r}_t * \mathbf{h}_{t-1}] + \mathbf{b}\right)$$

RNN: Practical Approaches

- Gradient clip: gradient sometimes goes very large even with LSTMs.
  Empirical solution: After calculating gradients, require the L_2 norm to be at most C (set by hyperparameters).
- At time step t, what matters to $\mathbf{h}_t$ is mostly $\mathbf{x}_{t'}$ where $t'$is close to t.
  [Khandelwal et al., ACL 2018]
- Bidirectional modeling typically results in more powerful features.

Recursive Neural Networks

- Run constituency parser on sentence, and construct vector recursively
- All nodes share the same set of parameters [Socher et al., 2011&2013]



Recursive Neural Networks

- Tree LSTMs typically work well.
  (slight modification of LSTM cells needed)

$$
\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_f\right) \\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_i\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(\mathbf{W}_c[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_c\right) \\
\mathbf{c}_t &= \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{c}}_t \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_o\right) \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(\mathbf{c}_t)
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\mathbf{l}_n &= \sigma\left(\mathbf{W}_\ell[\mathbf{h}_\ell; \mathbf{h}_r] + \mathbf{b}_\ell\right) \\
\mathbf{r}_n &= \sigma\left(\mathbf{W}_r[\mathbf{h}_\ell; \mathbf{h}_r] + \mathbf{b}_r\right) \\
\tilde{\mathbf{c}}_n &= \tanh\left(\mathbf{W}_c[\mathbf{h}_\ell; \mathbf{h}_r] + \mathbf{b}_c\right) \\
\mathbf{c}_n &= \mathbf{l}_n * \mathbf{c}_\ell + \mathbf{r}_n * \mathbf{c}_r + \tilde{\mathbf{c}}_n \\
\mathbf{o}_n &= \sigma\left(\mathbf{W}_o[\mathbf{h}_\ell; \mathbf{h}_r] + \mathbf{b}_o\right) \\
\mathbf{h}_n &= \mathbf{o}_n * \tanh(\mathbf{c}_n)
\end{aligned}
$$

Recursive Neural Networks

- Tree LSTMs typically work well.
  (slight modification of LSTM cells needed)
- Recursive neural networks with left-branching trees are basically equivalent to recurrent neural networks.
- Syntactically meaningful parse trees are not necessary for good representations: balanced trees work well (or even better) for most tasks.
  [Shi et al., EMNLP 2018]

Attention

- Can be thought of as weighted sum; each token receives a weight
- From (unweighted) bag of words to (weighted) bag of words
  - Each word receives a fixed weight
  - Normalize the weights with softmax

$$\alpha_{w_i} = \text{softmax}_{i'=1}^{k}\left(weight_{w_i}\right) = \frac{e^{weight_{w_i}}}{\sum_{i'=1}^{k} e^{weight_{w_{i'}}}}$$

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_{w_i} \cdot emb(w_i)$$

Parameterized Attention

- Word tokens with the same word type should probably receive different weights in different sentences
- Implement attention with an MLP (example below)

$$\bar{\mathbf{x}} = \frac{1}{k} \sum_{i=1}^{k} emb(w_i)$$

$$\alpha(w_i \mid \bar{\mathbf{x}}) = \text{softmax}_{i'=1}^{k} \left(\text{MLP}([emb(w_i); \bar{\mathbf{x}}])\right) \in \mathbb{R}$$

$$\mathbf{x} = \sum_{i=1}^{k} \alpha(w_i \mid \bar{\mathbf{x}}) \cdot emb(w_i)$$

Self-Attentive RNNs

- The last hidden state of RNN could be bad feature. Why?
- At time step t, what matters to $\mathbf{h}_t$ is mostly $\mathbf{x}_{t'}$ where $t'$is close to t [Khandelwal et al., ACL 2018]

$$\alpha_i = \mathrm{softmax}_{i'=1}^{k}\left(\mathrm{MLP}(\mathbf{h}_i)\right) \in \mathbb{R}$$

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_i \mathbf{h}_i$$

Trainable parameters,
Jointly trained w/ RNN parameters

Caveat: attention weights over RNN hidden states could be bad indicators on which token is more important

Transformers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer Encoder

- Transformer: attention-based sentence encoding, and optionally, decoding.
- Idea: every token has "attention" to every other token.

- For sentence with tokens $(w_1, \ldots, w_k)$

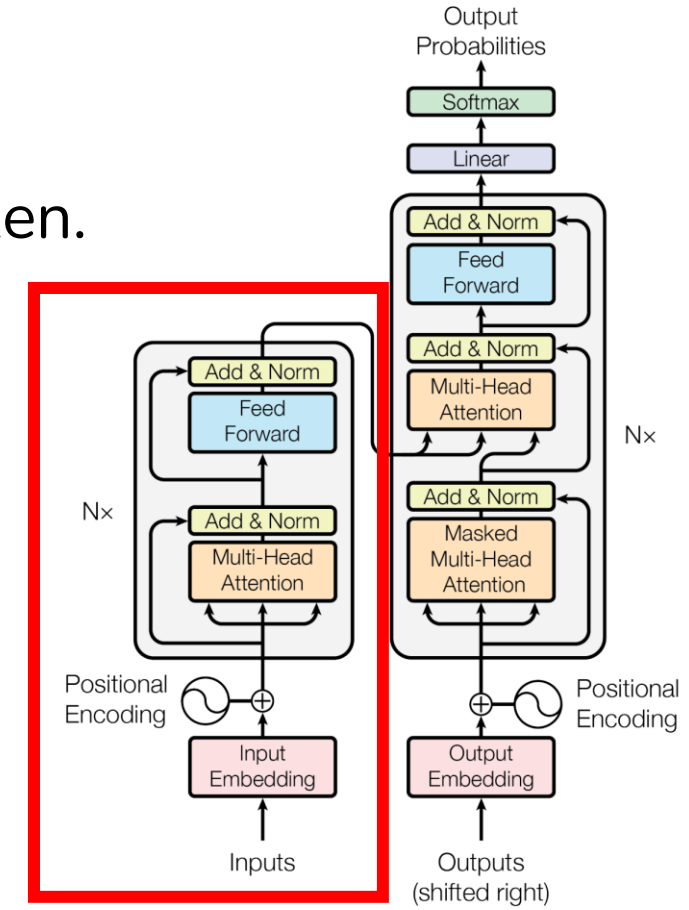
$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) \in \mathbb{R}^{d_1 \times k}$$

$$\mathbf{K} = \mathbf{W}_k\mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{Q} = \mathbf{W}_q\mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{V} = \mathbf{W}_v\mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$

Trainable parameters



Transformer Encoder

$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) \in \mathbb{R}^{d_1 \times k}$$

$$\mathbf{K} = \mathbf{W}_k \mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}, \mathbf{K} \in \mathbb{R}^{d_2 \times k}$$

$$\mathbf{Q} = \mathbf{W}_q \mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}, \mathbf{Q} \in \mathbb{R}^{d_2 \times k}$$

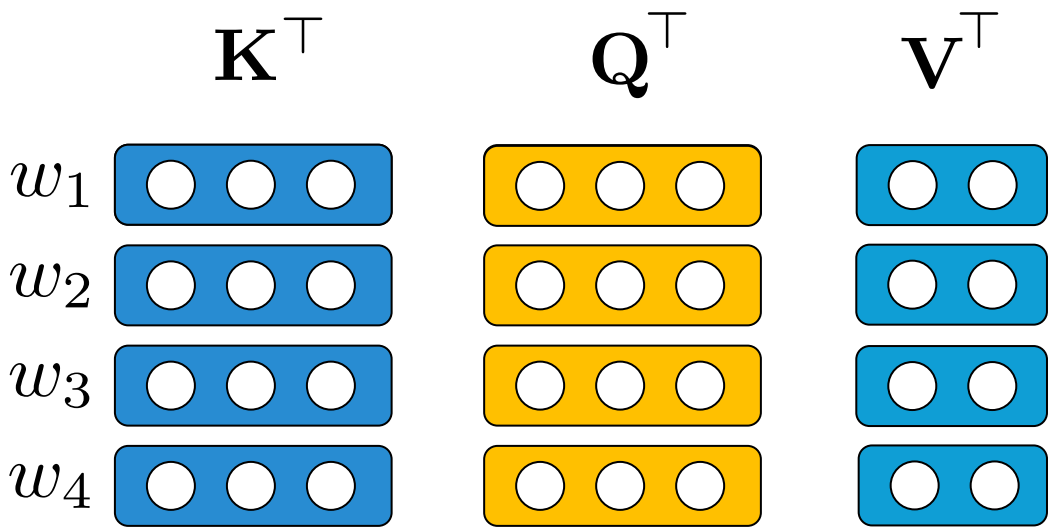
$$\mathbf{V} = \mathbf{W}_v \mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}, \mathbf{V} \in \mathbb{R}^{d_3 \times k}$$

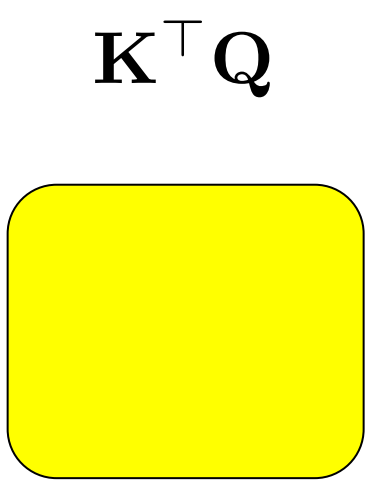
$$\tilde{\mathbf{E}} = \mathbf{V} \text{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right) \in \mathbb{R}^{d_3 \times k}$$

$k \times k$ matrix, softmax over the first dimension

Transformer Encoder

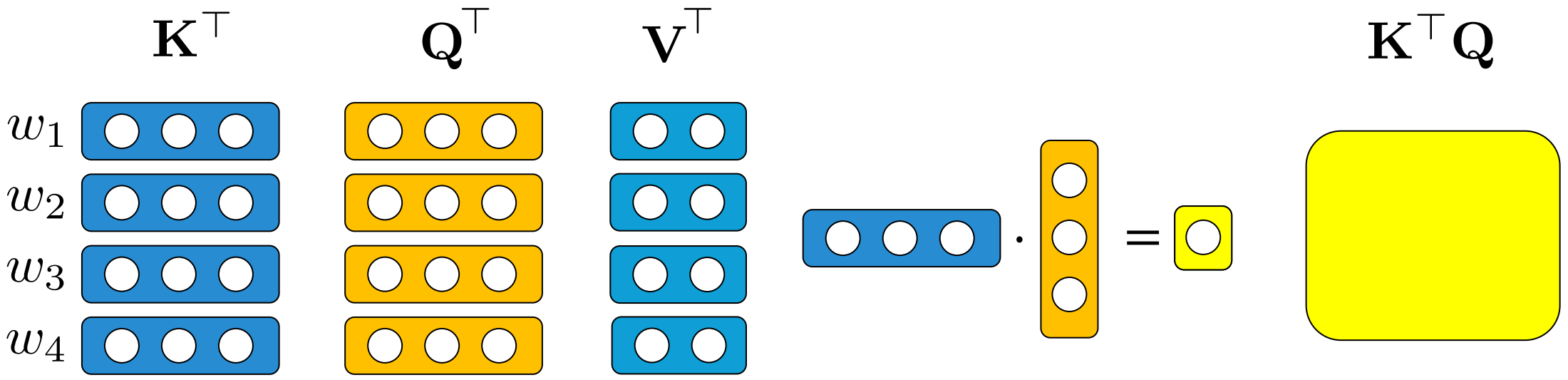
$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$





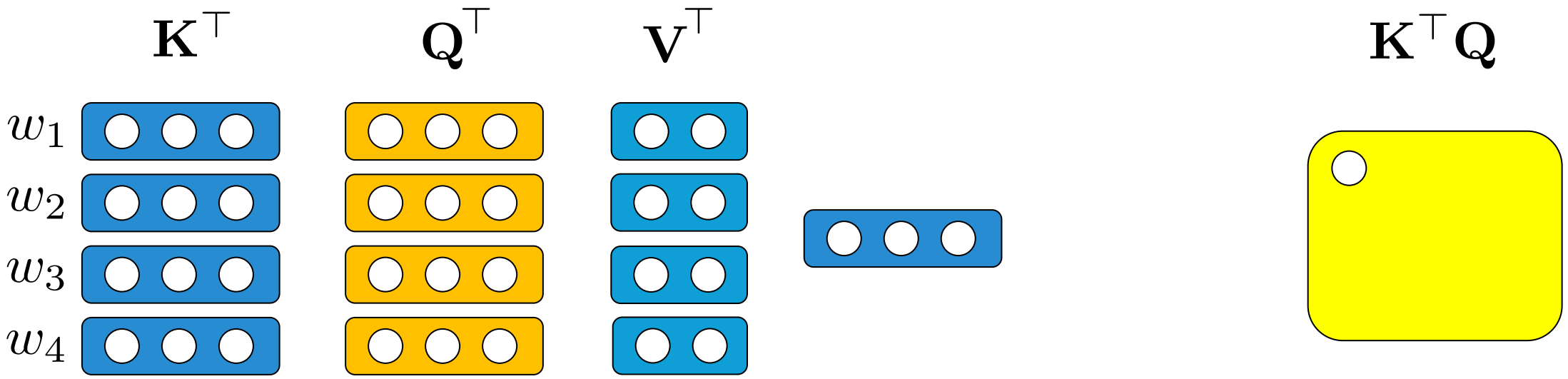
Transformer Encoder

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



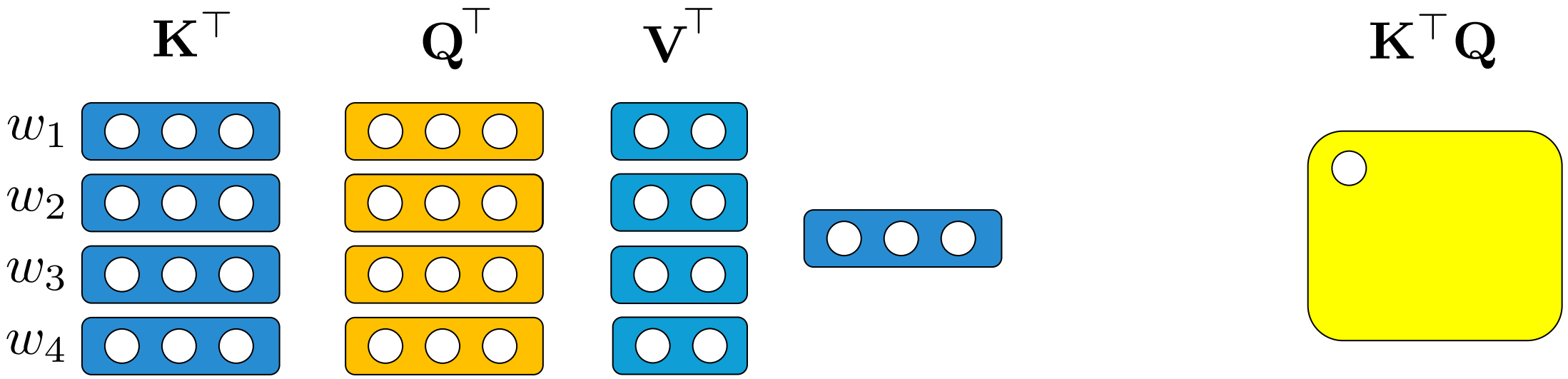
Transformer Encoder

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



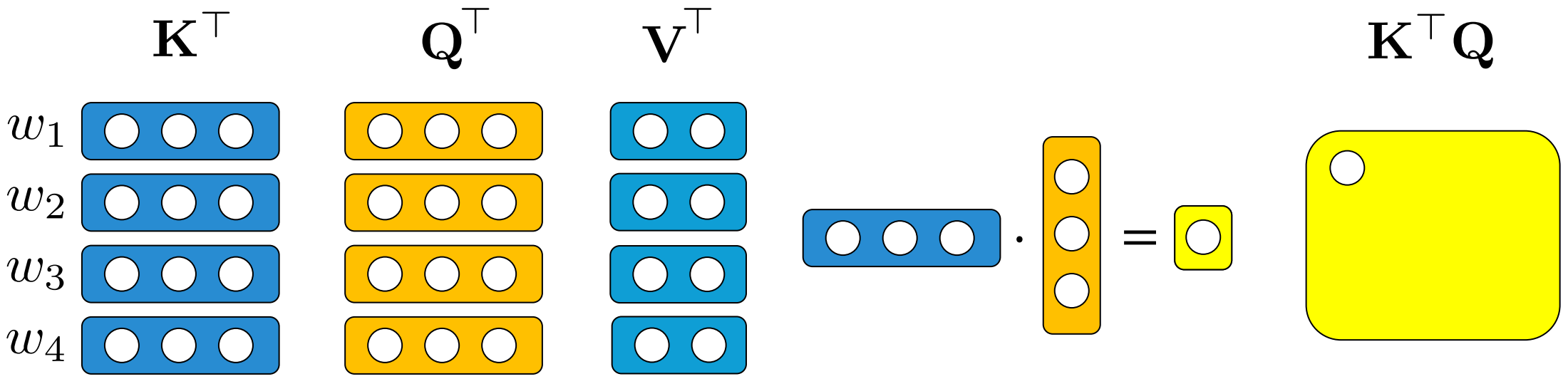
Transformer Encoder

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



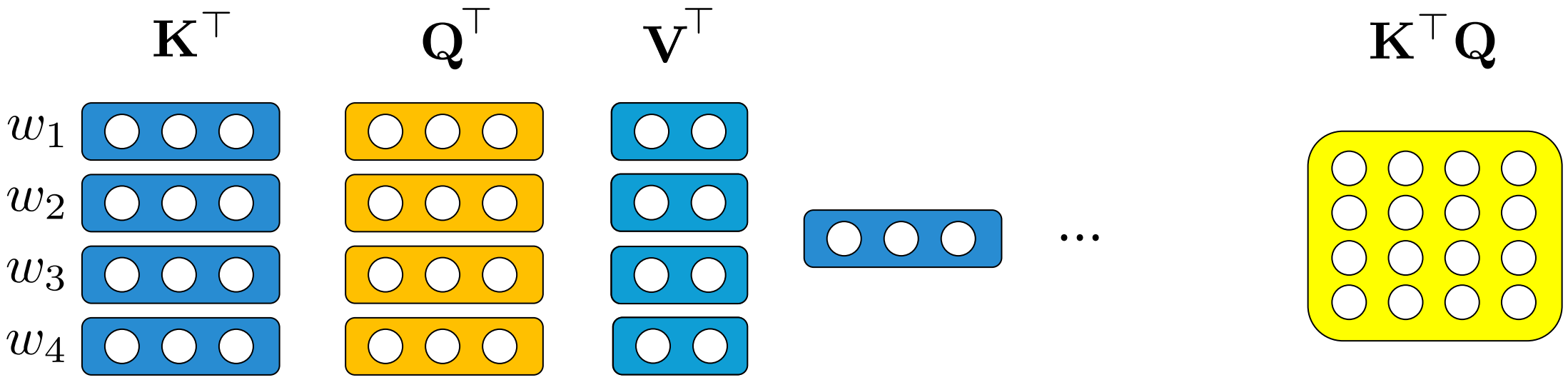
Transformer Encoder

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



Transformer Encoder

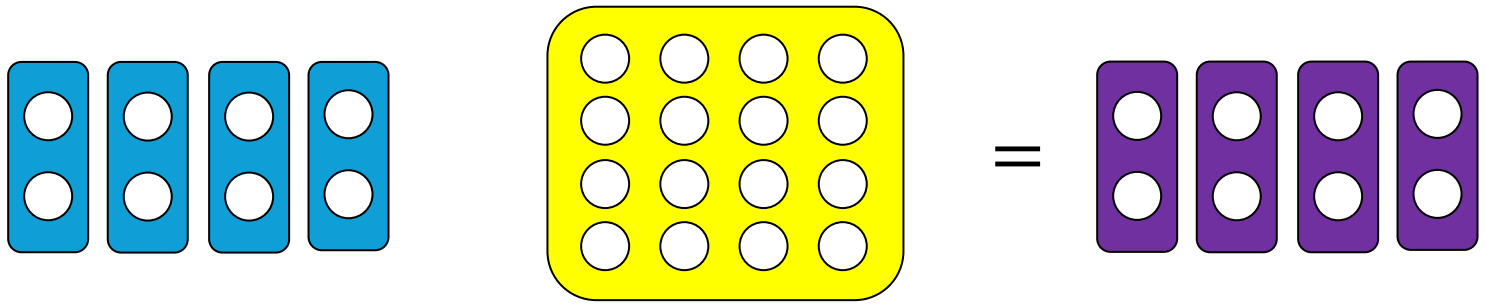
$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



Transformer Encoder

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$

$\mathbf{V}$ $\quad$ $\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$



Transformer Encoder: Variance Preservation

$$\tilde{\mathbf{E}} = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$

- What is $\sqrt{d_2}$ for?
- Consider $\langle \mathbf{a}, \mathbf{b} \rangle$: if each entry in both vector is drawn from a distribution with zero mean and unit variance, what would happen if the dimensionality grows?
- The variance of dot product grows.

$$\mathrm{softmax}([1, -1]) = [.8808, .1192]$$

$$\mathrm{softmax}([10, -10]) = [1, 2.0612 \times 10^{-9}]$$

Recap: Variance and Covariance

For independent zero-mean, unit-variance random variables X and Y

$$
\begin{aligned}
Var[XY] &= \mathbb{E}[X^2Y^2] - \mathbb{E}^2[XY] \\
&= (Cov[X^2, Y^2] + \mathbb{E}[X^2][Y^2]) - (Cov[X, Y] + \mathbb{E}[X]\mathbb{E}[Y])^2 \\
&= \mathbb{E}[X^2]\mathbb{E}[Y^2] - \mathbb{E}^2[X]\mathbb{E}^2[Y] \\
&= Var[X]\,Var[Y] + Var[X]\mathbb{E}^2[Y] + Var[Y]\mathbb{E}^2[X] \\
&= 1
\end{aligned}
$$

Recap: Variance and Covariance

For independent zero-mean, unit-variance random variables X and Y

$$Var[XY] = 1$$

If we have $2n$ independent zero-mean, unit variance variables $X_1, Y_1, X_2, Y_2, \ldots, X_n, Y_n$

$$Var[\sum_{i=1}^{n} X_i Y_i] = \sum_{i=1}^{n} Var[X_i Y_i] = n$$

$$Var[\sum_{i=1}^{n} \frac{X_i Y_i}{\sqrt{n}}] = \sum_{i=1}^{n} Var[\frac{X_i Y_i}{\sqrt{n}}] = \sum_{i=1}^{n} \frac{1}{n} Var[X_i Y_i] = 1$$

Transformer Encoder

$$Var[\sum_{i=1}^{n} \frac{X_i Y_i}{\sqrt{n}}] = \sum_{i=1}^{n} Var[\frac{X_i Y_i}{\sqrt{n}}] = \sum_{i=1}^{n} \frac{1}{n} Var[X_i Y_i] = 1$$

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$

The application of $\sqrt{d_2}$ is theoretically motivated.

See also Xavier initialization: initialize a dot product parameter vector

with values drawn from $U\left(-\sqrt{\frac{3}{d}}, \sqrt{\frac{3}{d}}\right)$

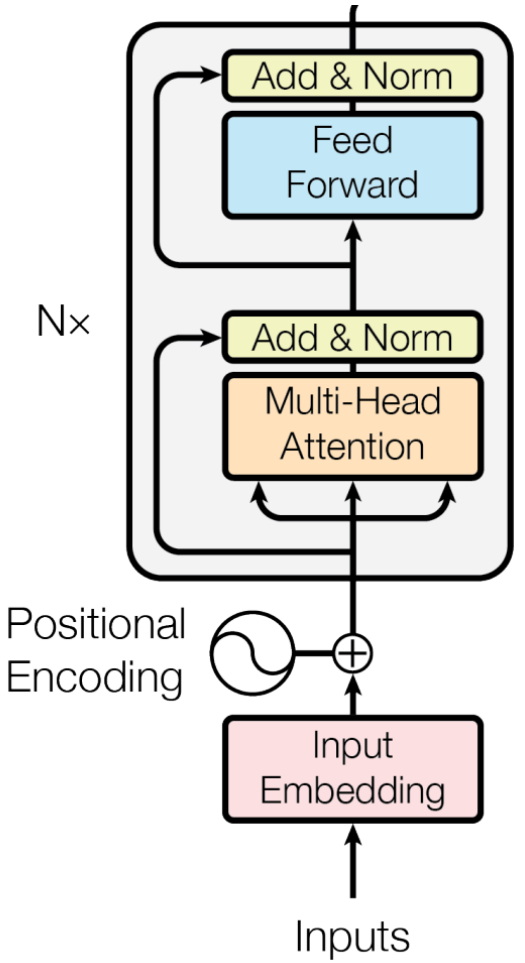
Positional Encoding

$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) \in \mathbb{R}^{d_1 \times k}$$
$$\mathbf{K} = \mathbf{W}_k \mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$
$$\mathbf{Q} = \mathbf{W}_q \mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$
$$\mathbf{V} = \mathbf{W}_v \mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$
$$\tilde{\mathbf{E}} = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$

This is just complicated bag of words…

Columns of $\tilde{\mathbf{E}}$ for “a cat”

= permutation of columns of $\tilde{\mathbf{E}}$ for “cat a”



Positional Encoding

$$\mathbf{p}_{p,2i} = \sin\left(\frac{p}{10000^{\frac{2i}{d}}}\right), \mathbf{p}_{p,2i+1} = \cos\left(\frac{p}{10000^{\frac{2i}{d}}}\right)$$

- The choice of $n = 10{,}000$ is somewhat arbitrary, but it's overall theoretically motivated: The positional add-δ relation can be represented by a linear transformation.

$$\forall \delta, \exists \mathbf{M}_\delta, \text{s.t. } \mathbf{p}_{p+\delta} = \mathbf{M}_\delta \mathbf{p}_p \quad (\forall p)$$

- Proof idea: use the addition theorems on trigonometric functions

$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$

$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$

Applying Positional Encoding

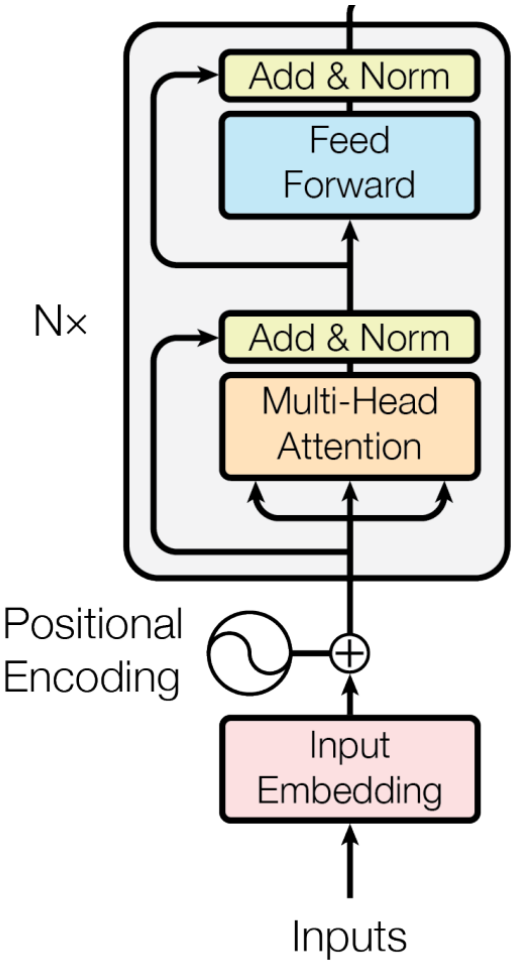
$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) + \mathbf{P} \in \mathbb{R}^{d_1 \times k}$$

$$\mathbf{K} = \mathbf{W}_k \mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{Q} = \mathbf{W}_q \mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{V} = \mathbf{W}_v \mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$

$$\tilde{\mathbf{E}} = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$

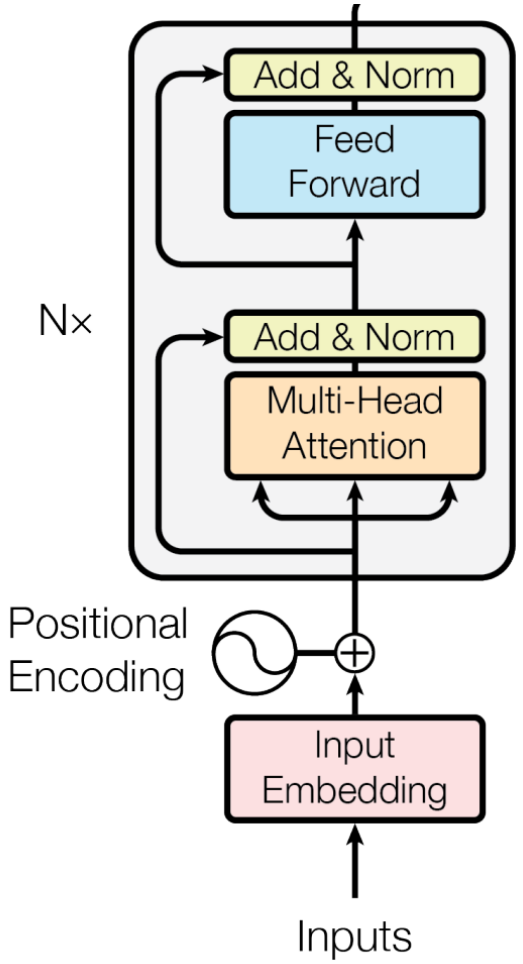


- Limitation: only fixed number of positions available
- Another option: learnable positional encoding

Multi-Head Attention

$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) + \mathbf{P} \in \mathbb{R}^{d_1 \times k}$$
$$\mathbf{K} = \mathbf{W}_k \mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$
$$\mathbf{Q} = \mathbf{W}_q \mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$
$$\mathbf{V} = \mathbf{W}_v \mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$
$$\tilde{\mathbf{E}} = \mathbf{V}\mathrm{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$

- We can parallelize multiple $\mathbf{W}_k, \mathbf{W}_q, \mathbf{W}_v$ with different random initialization (and hope they learn different ways to attend tokens.



Stacking Transformer Layers

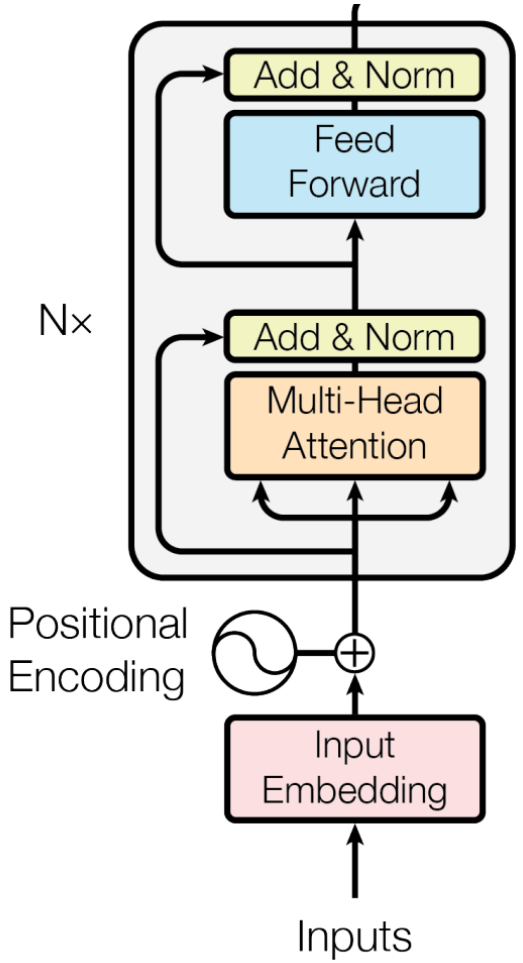
$$\mathbf{E} = (emb(w_1), \ldots, emb(w_k)) + \mathbf{P} \in \mathbb{R}^{d_1 \times k}$$

$$\mathbf{K} = \mathbf{W}_k \mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{Q} = \mathbf{W}_q \mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{V} = \mathbf{W}_v \mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d_2}}\right)$$



Residual Connection and Normalization

- Earlier layers' output are added to the "vector stream."
- Normalization: preserve variance.

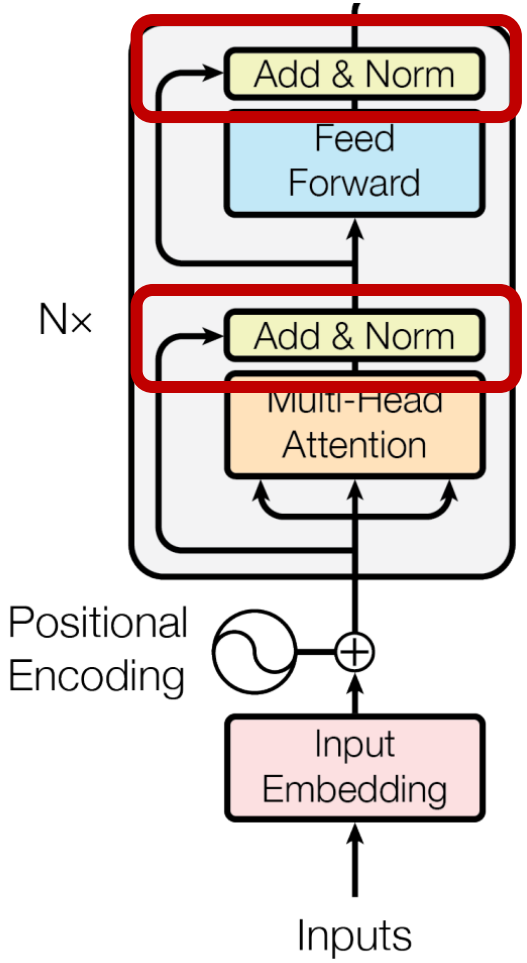
$$\mathbf{E}' = \text{Normalized}(E^{(0)} + P + \tilde{\mathbf{E}})$$

$$\mathbf{K} = \mathbf{W}_k\mathbf{E} \quad \mathbf{W}_k \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{Q} = \mathbf{W}_q\mathbf{E} \quad \mathbf{W}_q \in \mathbb{R}^{d_2 \times d_1}$$

$$\mathbf{V} = \mathbf{W}_v\mathbf{E} \quad \mathbf{W}_v \in \mathbb{R}^{d_3 \times d_1}$$

$$\tilde{\mathbf{E}} = \mathbf{V}\text{softmax}\left(\frac{\mathbf{K}^\top\mathbf{Q}}{\sqrt{d_2}}\right)$$



Next

Language Modeling