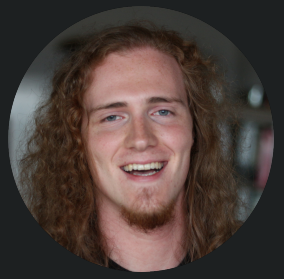


LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Introduction to Advanced Neural Network Architectures

for Natural Language Processing

Find the most up-to-date version of this talk at:
https://github.com/lschmelzeisen/talk-advanced-neural-network-architectures

License

- The contents of these slides are licensed as *CC BY-SA 4.0*
- You may *share* and *adapt* freely, under the following terms:
  - **Attribution**: you must credit me as the original author
  - **ShareAlike**: you must distribute your work under the same license
- See https://creativecommons.org/licenses/by-sa/4.0/

What is this talk about?

This Guy

This Guy

- **BERT**: Bidirectional Encoder Representations from Transformers (Devlin et al, 2019)
- Very influential paper
- State-of-the-Art for many Natural Language Processing tasks

Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

What will you get out of it?

- Deep Learning can appear like alchemy
- Learn individual buildings blocks (*spell/element names*)
- Learn how to combine blocks to solve language tasks (*which elements make useful potions?*)

Warning

- There will be math
  - Everything will be explained
  - Knowing undergrad linear algebra is sufficient
- Many simplifications and inaccuracies
  - When relevant, **correct me!**
- **Ask questions!**

What is missing?

- No evaluation
- No metrics
- No performance scores
- No listings of hyperparameters
- No task-specialized architectures
- No theoretical justification
  - Which usually exists whenever I say “empirically it works”

Advertisement: My Talk From Last Year

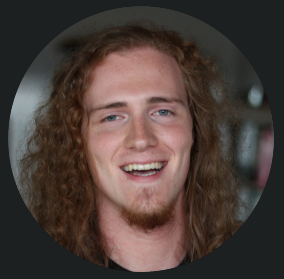
24 Sep 2018

Introduction to Supervised Learning with TensorFlow

Find it at: https://github.com/lschmelzeisen/talk-supervised-learning-tensorflow

Outline

1. Introduction
   - » Machine Learning Basics
   - » Natural Language Processing Tasks
2. Review: Deep Learning Basics
   - » Feed-forward Neural Networks
   - » Common Improvements
3. Classical Architectures
   - » Word Embeddings
   - » Recurrent Neural Networks
   - » Convolutional Neural Networks
4. Attention Mechanism
   - » Attention-based Recurrent Neural Networks
   - » Transformers
5. State of the Art
   - » OpenAI GPT
   - » BERT



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Introduction to Advanced Neural Network Architectures

for Natural Language Processing

» Introduction

Human Learning?

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f(1) = 1$$

$$f(2) = 4$$

$$f(3) = 9$$

$$f(x) = ?$$

Human Learning!

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f(1) = 1$$

$$f(2) = 4$$

$$f(3) = 9$$

$$f(x) = x^2$$

Machine Learning

» Machine learning is *just* function approximation

$$f : X \to Y$$

» We know inputs and outputs of an interesting **target function**, but not the function itself

» Find a **model function** which is approximately the same automatically

Supervised Learning

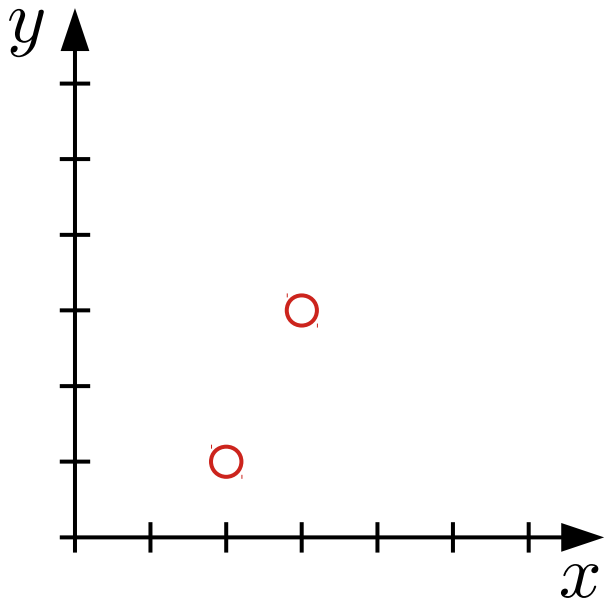
- **Dataset** of input-target-examples:
  $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}$
- **Model function** that approximates the target given an input $x^{(i)}$ and parameters Θ: $\hat{y}^{(i)} = f(x^{(i)}; \Theta)$
- **Loss function** that measures distance of expected target $y^{(i)}$ to predicted output $\hat{y}^{(i)}$: $L(y^{(i)}, \hat{y}^{(i)})$
- **Training algorithm** that finds parameters minimizing the loss: $\Theta_{\text{optimal}} = \arg\min_{\Theta} \sum_{(x^{(i)}, y^{(i)}) \in D} L(y^{(i)}, f(x^{(i)}; \Theta))$

Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$

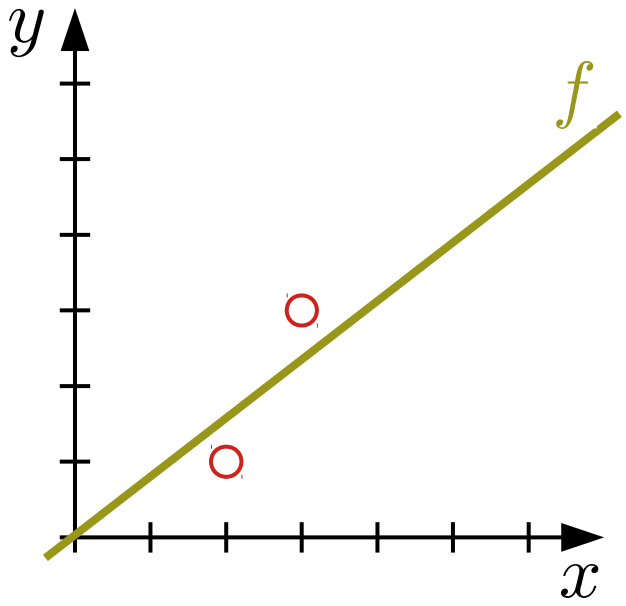
Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$
- Fit them as good as possible to these two input-target examples



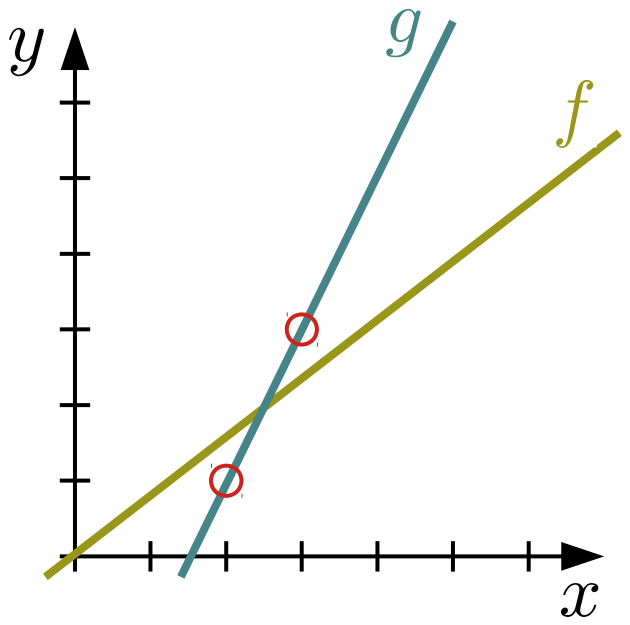
Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$
- Fit them as good as possible to these two input-target examples



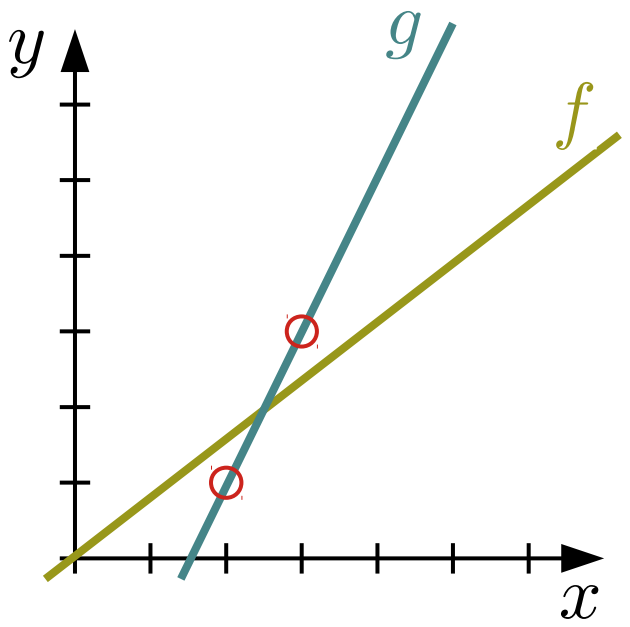
Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$
- Fit them as good as possible to these two input-target examples



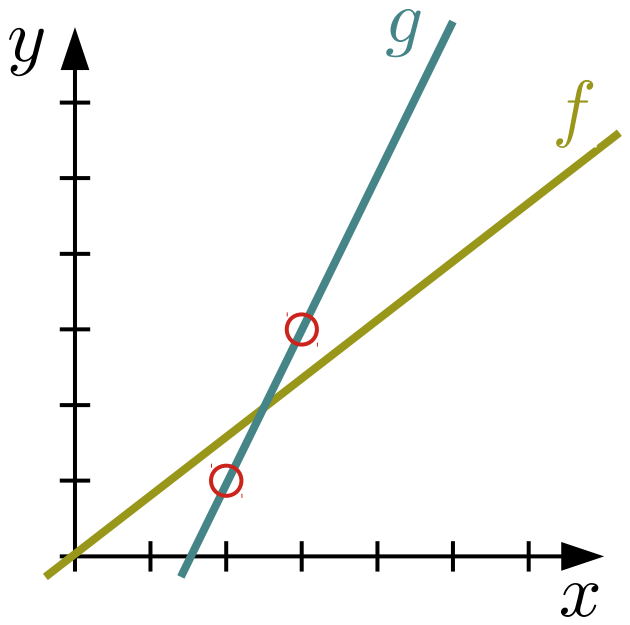
Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$
- Fit them as good as possible to these two input-target examples
- f can not perfectly fit both data points
- g can perfectly fit both data points



Model Function

- Compare these modeling functions:
  - $f(x; \Theta_f) = m \cdot x \qquad \Theta_f = \{m\}$
  - $g(x; \Theta_g) = m \cdot x + b \quad \Theta_g = \{m, b\}$
- Fit them as good as possible to these two input-target examples
- f can not perfectly fit both data points
- g can perfectly fit both data points
- g has higher **representation power** than f
- g uses **more parameters** than f



Stochastic Gradient Descent (SGD)

- Optimization algorithm to find parameters Θ that minimize loss function L
- Guaranteed to converge for convex functions, but can also be applied to non-convex loss functions like neural networks
- Requirement: need to be able to *differentiate* loss function with respect to parameters $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$

Stochastic Gradient Descent (SGD)

- Optimization algorithm to find parameters Θ that minimize loss function L
- Guaranteed to converge for convex functions, but can also be applied to non-convex loss functions like neural networks
- Requirement: need to be able to *differentiate* loss function with respect to parameters $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$

slope of loss function =
direction in which loss decreases
for current parameters

Stochastic Gradient Descent (SGD)

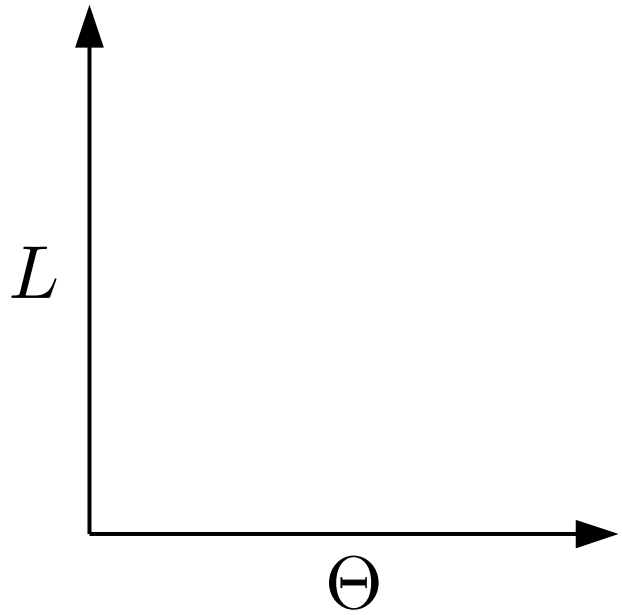
- Optimization algorithm to find parameters Θ that minimize loss function L
- Guaranteed to converge for convex functions, but can also be applied to non-convex loss functions like neural networks
- Requirement: need to be able to *differentiate* loss function with respect to parameters $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$

slope of loss function =
direction in which loss decreases
for current parameters

- Reminder: $\hat{y}^{(i)} = f(x^{(i)}; \Theta)$ so model function f also needs to be *differentiable*

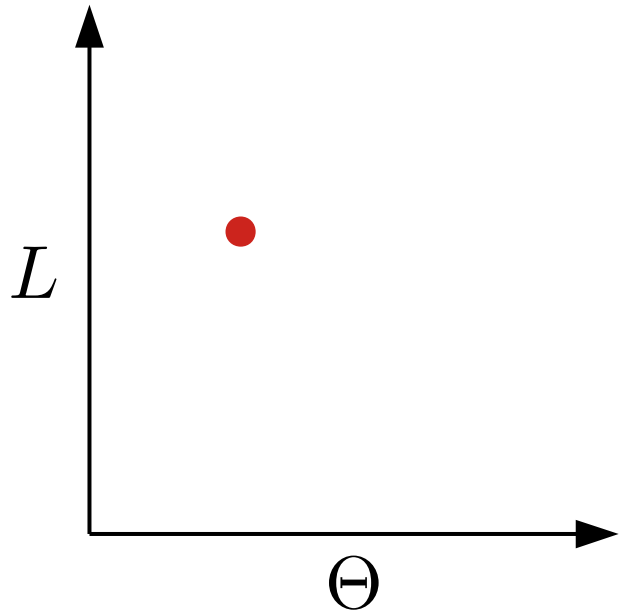
Stochastic Gradient Descent (SGD)

- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



Stochastic Gradient Descent (SGD)

- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$

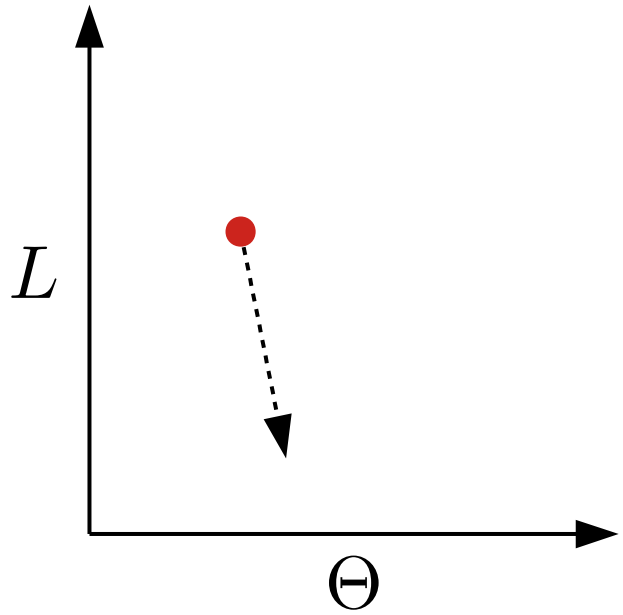


1. Initialize Θ randomly

Stochastic Gradient Descent (SGD)

» Setting: unknown relation between loss L and parameters Θ

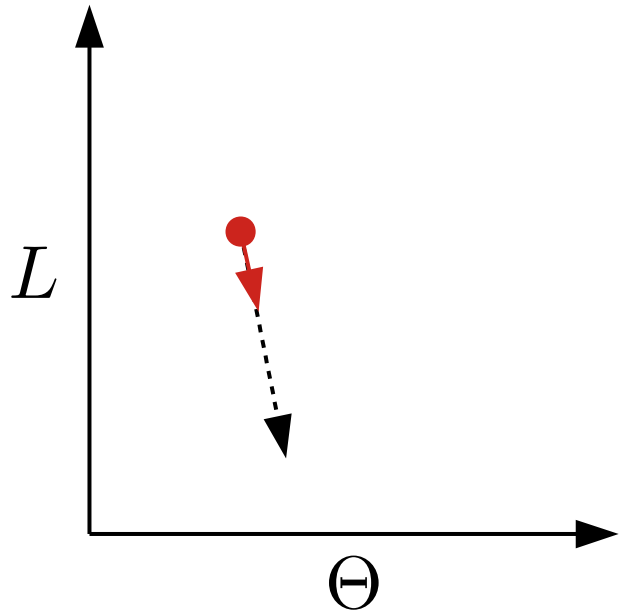
» Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient

Stochastic Gradient Descent (SGD)

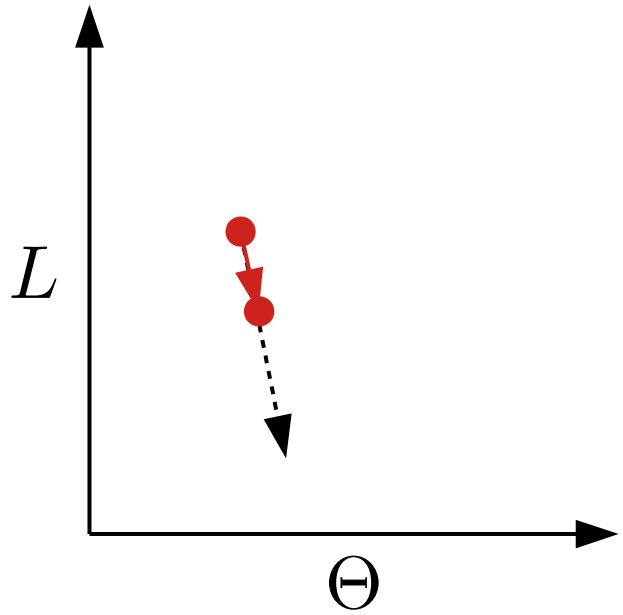
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient

Stochastic Gradient Descent (SGD)

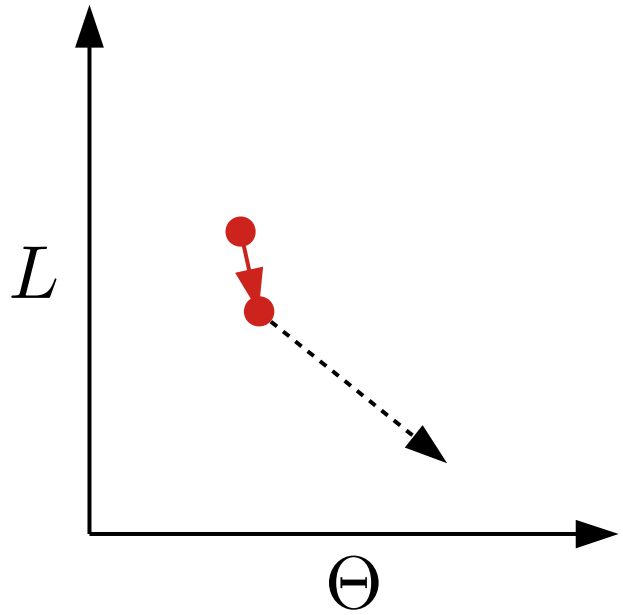
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

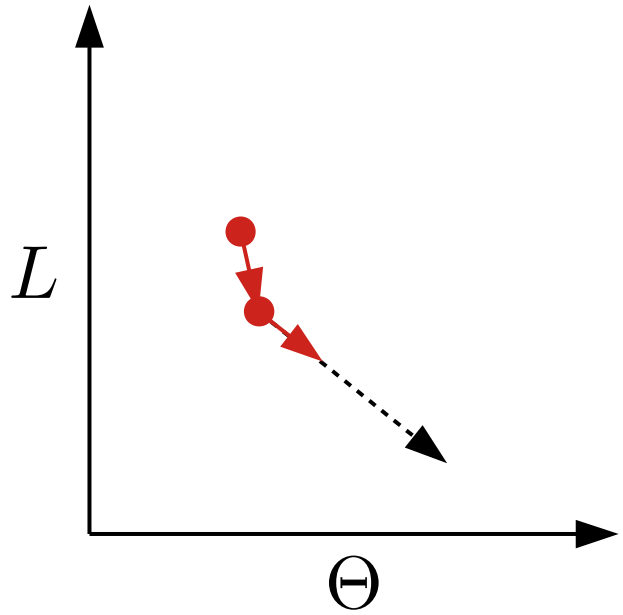
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

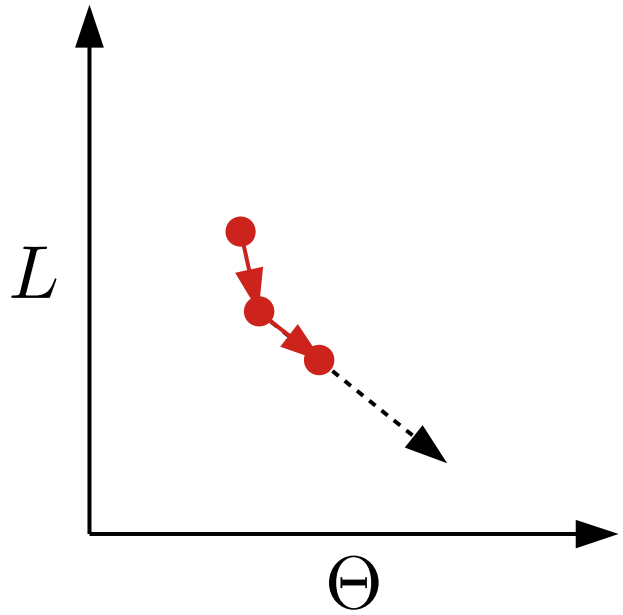
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

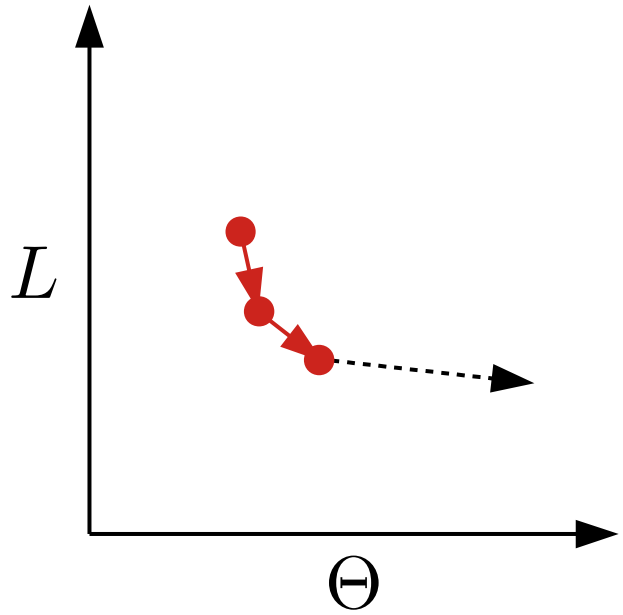
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

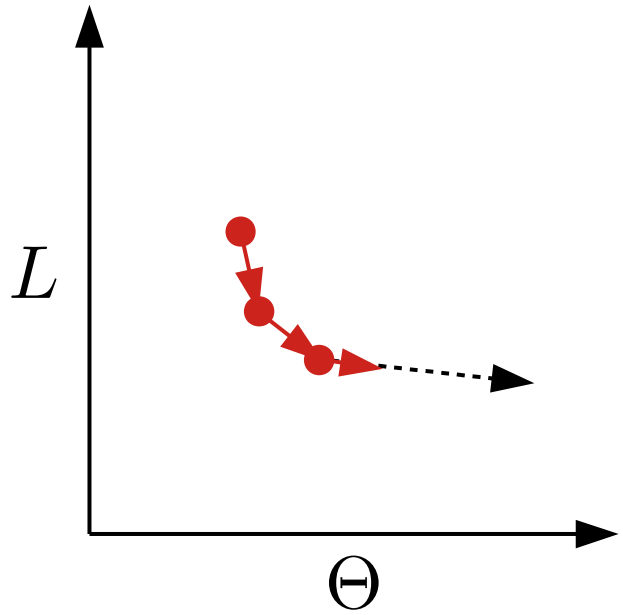
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

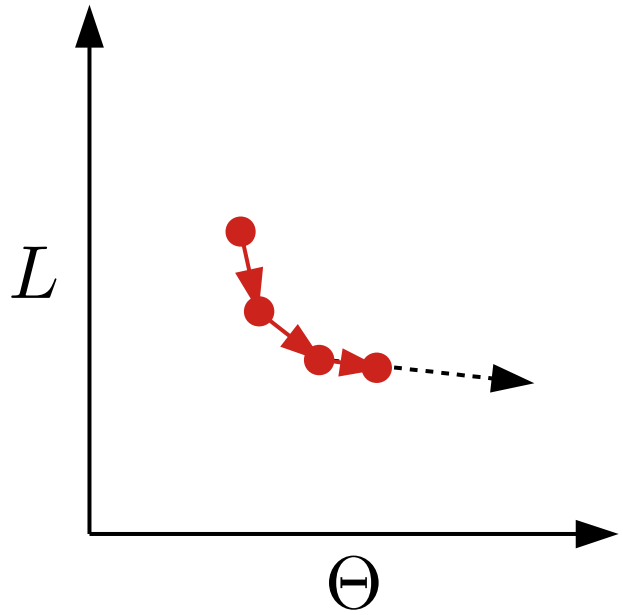
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2

Stochastic Gradient Descent (SGD)

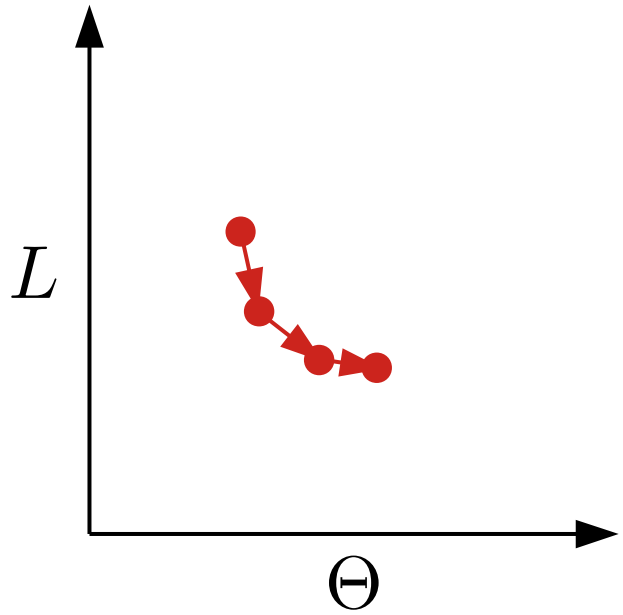
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2
5. Else: done

Stochastic Gradient Descent (SGD)

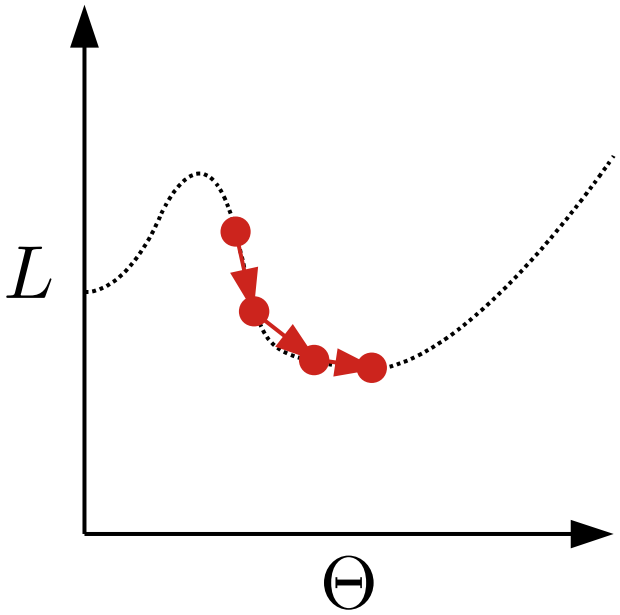
- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2
5. Else: done

Stochastic Gradient Descent (SGD)

- Setting: unknown relation between loss L and parameters Θ
- Task: find Θ that minimize L given gradient $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$



1. Initialize Θ randomly
2. Compute gradient
3. Move Θ a small step in direction of gradient
4. If not good enough: go to step 2
5. Else: done

(Actual loss function for this example)

Natural Language Processing Tasks

- Shape of input and output matters a lot

$$f : X \to Y$$

- Two important task types in natural language processing
  - **Sequence classification**
    input sequence of tokens, output single label
  - **Sequence to sequence**
    input sequence of tokens, output sequence of tokens/labels
- In general: tokens can be characters, words, sentences, …
- In this talk: always sequences of words

Sentiment Analysis

- Classify sentiment of given text
  - *"The voice quality of this phone is amazing"* → ***Positive Sentiment***
  - *"The earphones broke in two days"* → ***Negative Sentiment***

Sentiment Analysis

- Classify sentiment of given text
  - *"The voice quality of this phone is amazing"* → ***Positive Sentiment***
  - *"The earphones broke in two days"* → ***Negative Sentiment***
- Type: **sequence classification**

Machine Translation

- Translate given text from source language into target language
  - *"Ich liebe dich"* → *"I love you"*
  - *"I would rather have Pizza"* → *"Je préférerais avoir de la pizza"*

Machine Translation

- Translate given text from source language into target language
  - *"Ich liebe dich"* → *"I love you"*
  - *"I would rather have Pizza"* → *"Je préférerais avoir de la pizza"*
- Type: **sequence to sequence**

Textual Entailment

- Analyze whether a given hypothesis follows from a given premise
  - *Does "A man inspects the uniform of a figure in some East Asian county" entail "The man is sleeping"?* → ***Contradiction***
  - *Does "A soccer game with multiple males playing" entail "Some men are playing a sport"?* → ***Entailment***

Textual Entailment

- Analyze whether a given hypothesis follows from a given premise
  - *Does "A man inspects the uniform of a figure in some East Asian county" entail "The man is sleeping"?* → ***Contradiction***
  - *Does "A soccer game with multiple males playing" entail "Some men are playing a sport"?* → ***Entailment***
- Type: **sequence classification**
  - with some mechanism to combine both input sentences

Question Answering

- Formulate an answer to a given question using a given context
  - Context: *"In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity… Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals* ***within a cloud****. Short, intense …"*
  - Question: *"Where do water droplets collide with ice crystals to form precipitation?"*
  - Answer: ***"within a cloud"***

Question Answering

- Formulate an answer to a given question using a given context
  - Context: *"In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity… Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals* ***within a cloud****. Short, intense …"*
  - Question: *"Where do water droplets collide with ice crystals to form precipitation?"*
  - Answer: ***"within a cloud"***
- Type: **sequence to sequence**
  - with some mechanism to combine both input sentences

Language Modeling

- Estimate the probability of a sequence of words in a language
  - Usually formulated as: given all previous words in a sentence, guess the next word
  - *"I was born in France so I"* → ***"speak"***
  - *"I was born in France so I speak fluent"* → ***"French"***
  - *"I was born in France so I speak fluent French"* → ***"[EOS]"***
  - *"Lukas, who is pursuing his PhD at the Institute for Web Science and Technologies, still hopes to finish"* → ***"his"***

Language Modeling

- Estimate the probability of a sequence of words in a language
  - Usually formulated as: given all previous words in a sentence, guess the next word
  - *"I was born in France so I"* → ***"speak"***
  - *"I was born in France so I speak fluent"* → ***"French"***
  - *"I was born in France so I speak fluent French"* → ***"[EOS]"***
  - *"Lukas, who is pursuing his PhD at the Institute for Web Science and Technologies, still hopes to finish"* → ***"his"***
- Type: **sequence to sequence**
  - with constraint to not look at future words in the input

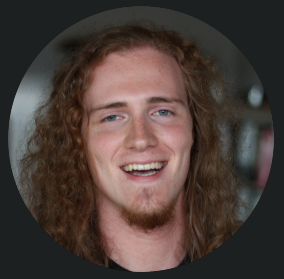
Summary: Introduction

- Machine learning is function approximation
- Gradient descent can find good model parameters
  - Only works if model function is differentiable
- Many natural language processing tasks are either **sequence classification** or **sequence-to-sequence**

How Can We Solve These Tasks?

How Can We Solve These Tasks?

» Neural Networks!



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

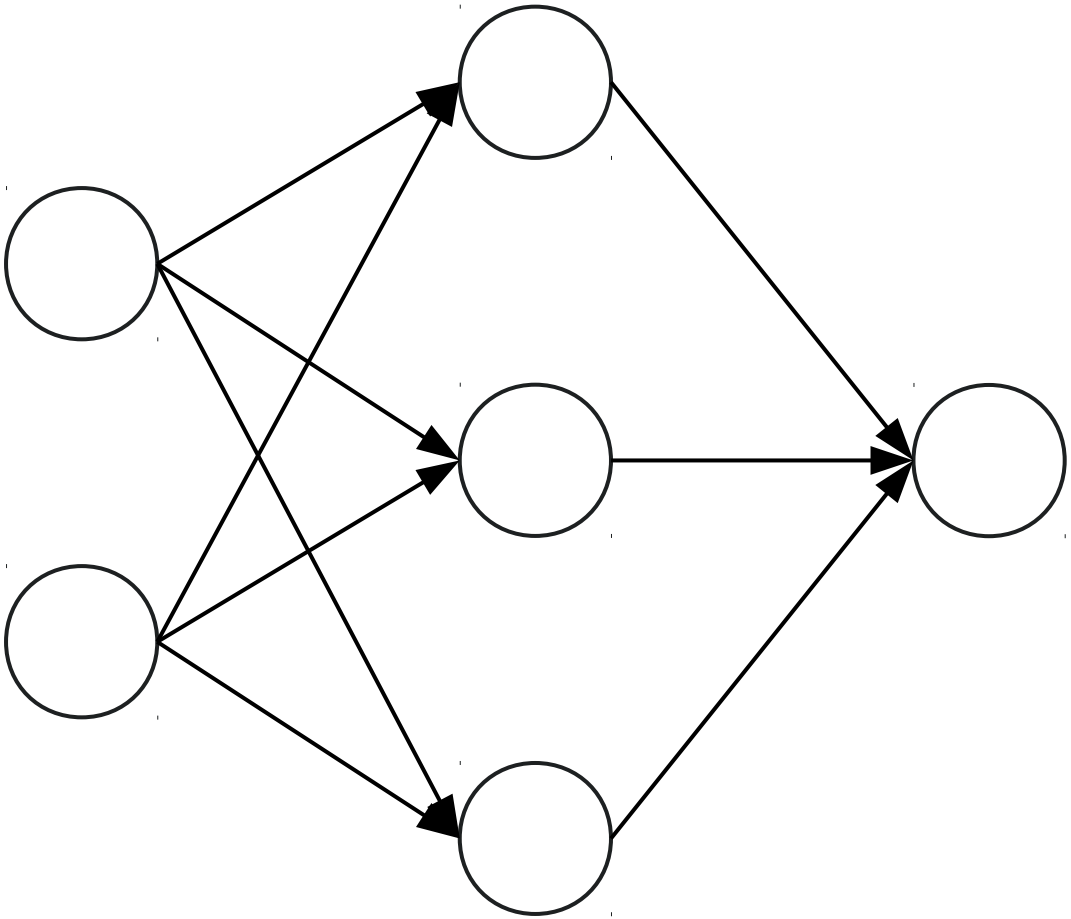
6 Sep 2019

Introduction to Advanced Neural Network Architectures

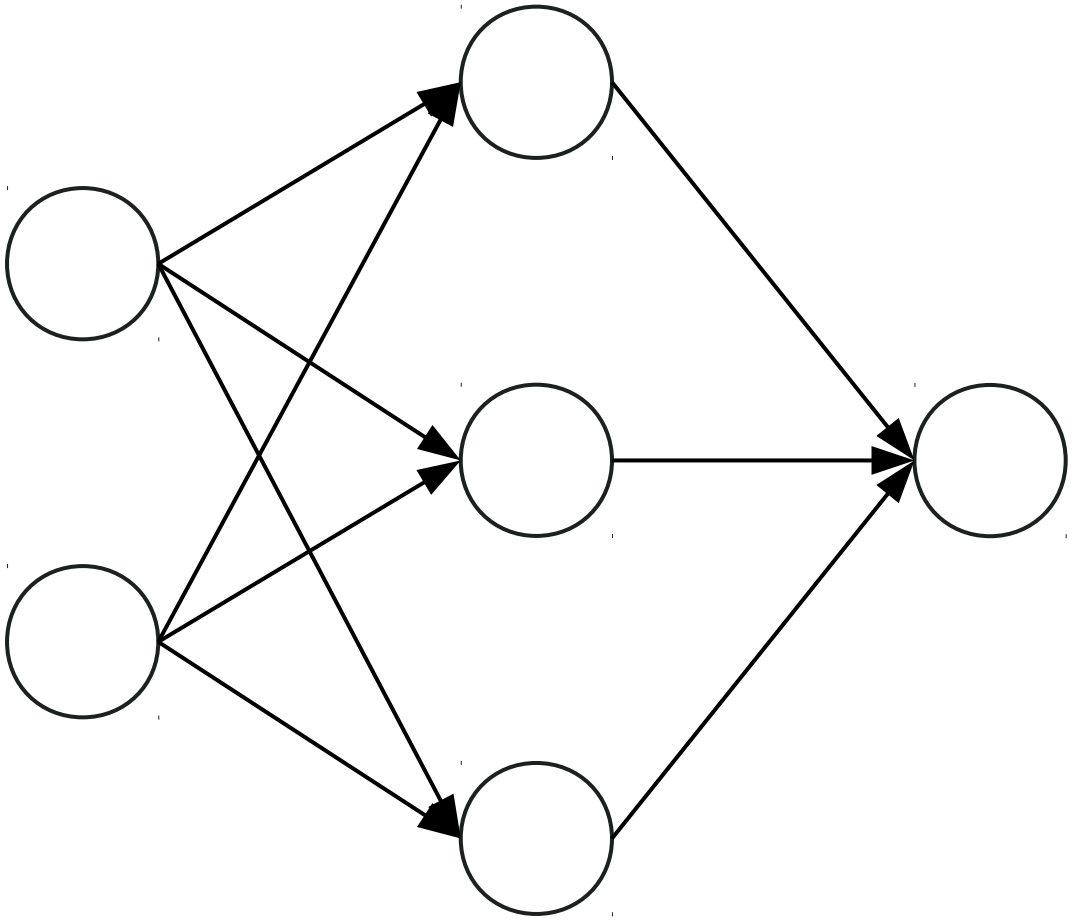
for Natural Language Processing

» Review: Deep Learning Basics

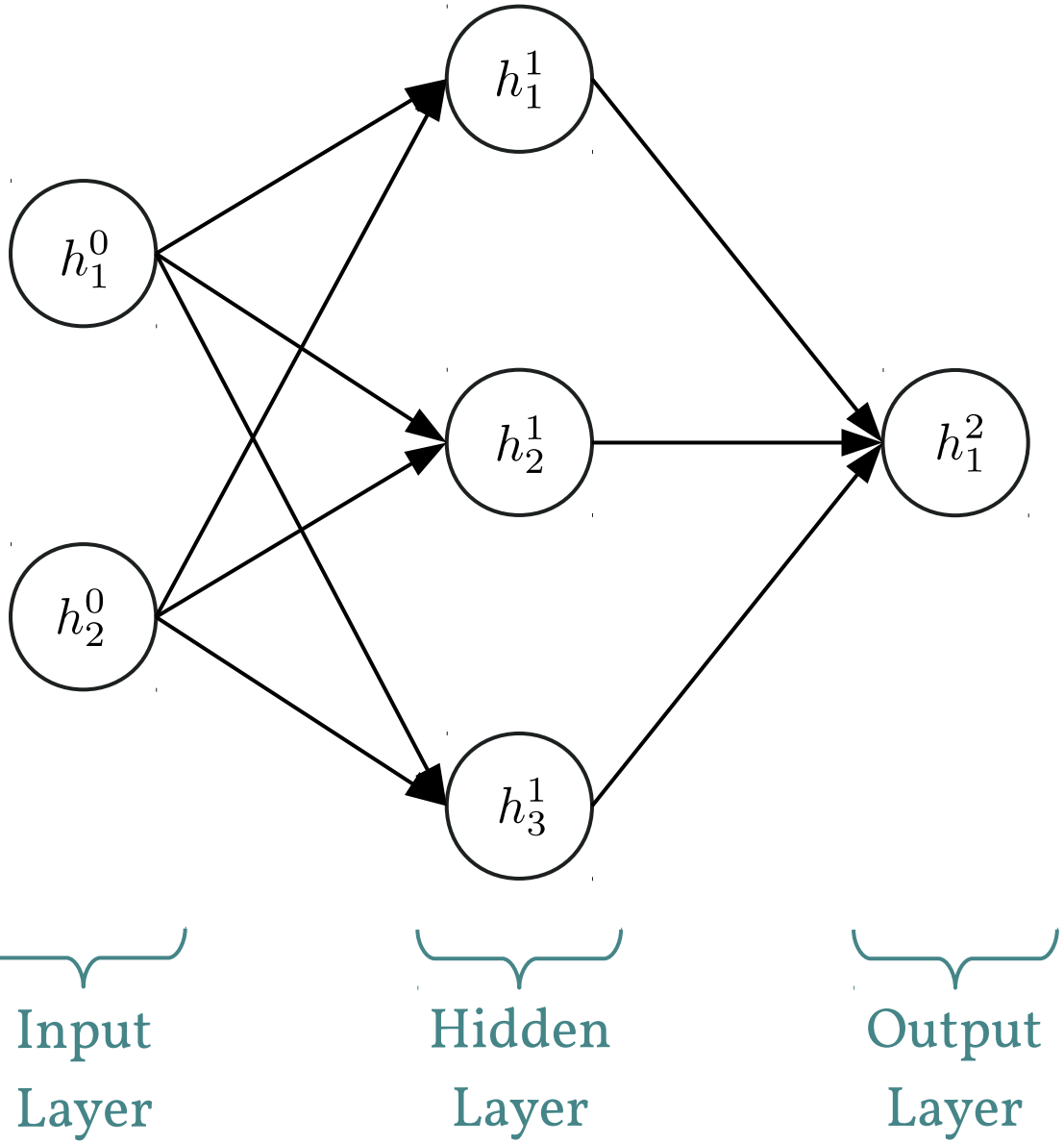
Feed-forward Neural Network



Feed-forward Neural Network

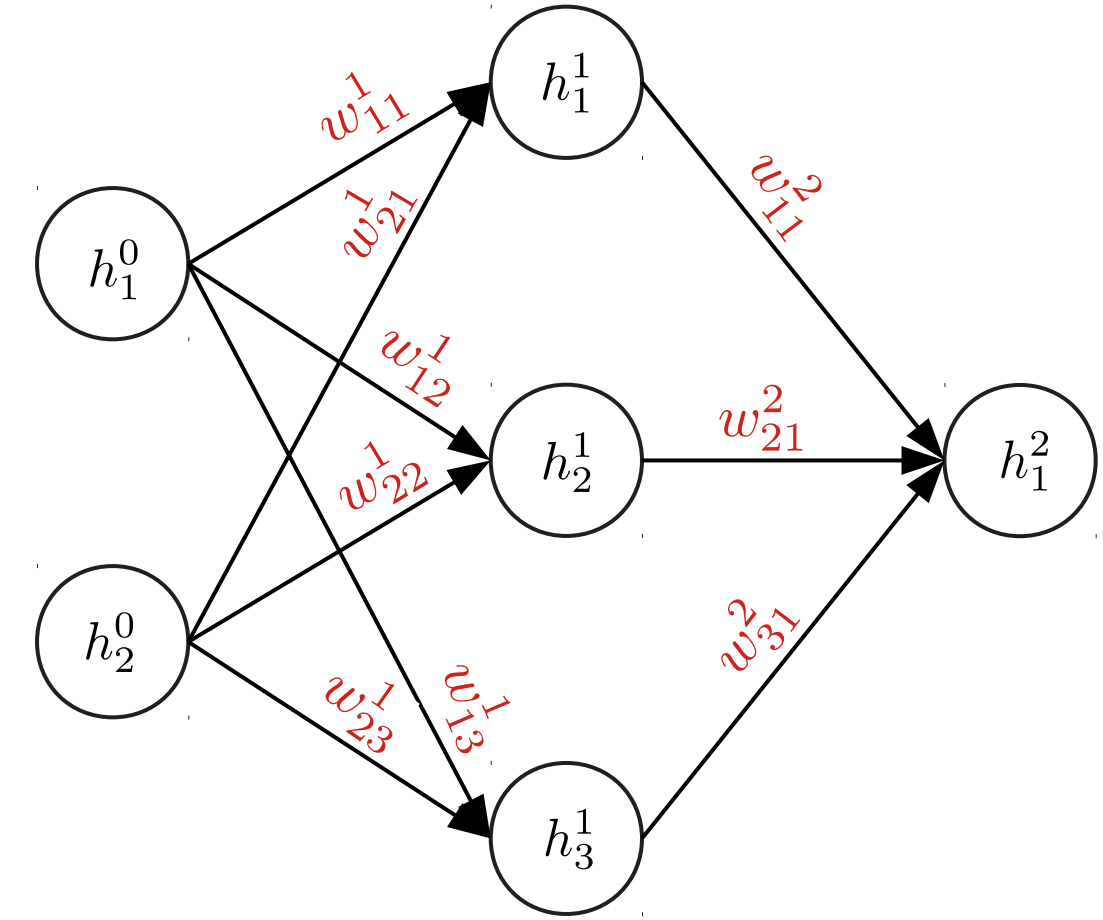


Feed-forward Neural Network



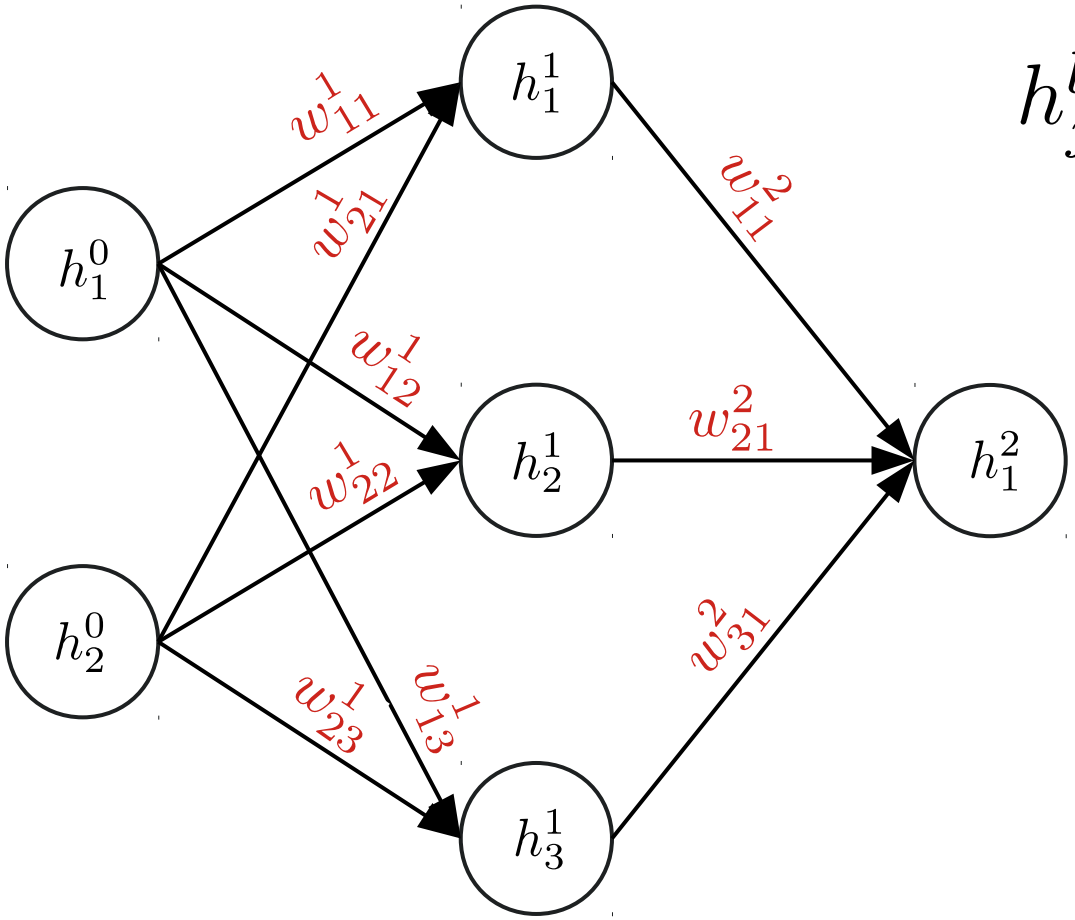
- Each neuron holds one number h_i^l
  - Input neurons are given per training example
  - Other neurons are calculated

Feed-forward Neural Network



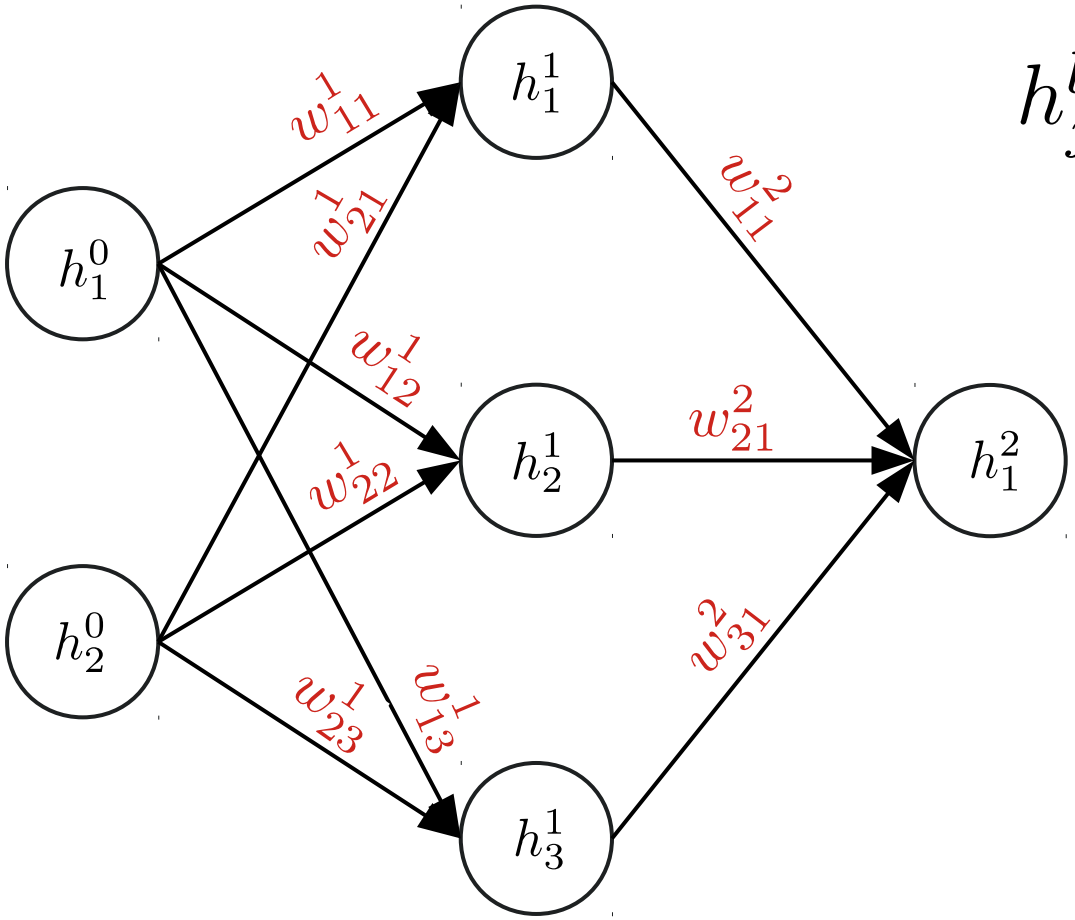
- Each neuron holds one number h_i^l
  - Input neurons are given per training example
  - Other neurons are calculated
- Each synapse holds one parameter weight w_{ij}^l connecting neurons h_i^{l-1} and h_j^l

Feed-forward Neural Network



$$h_j^l = \sigma\left(\sum_i h_i^{l-1} w_{ij}^l\right)$$

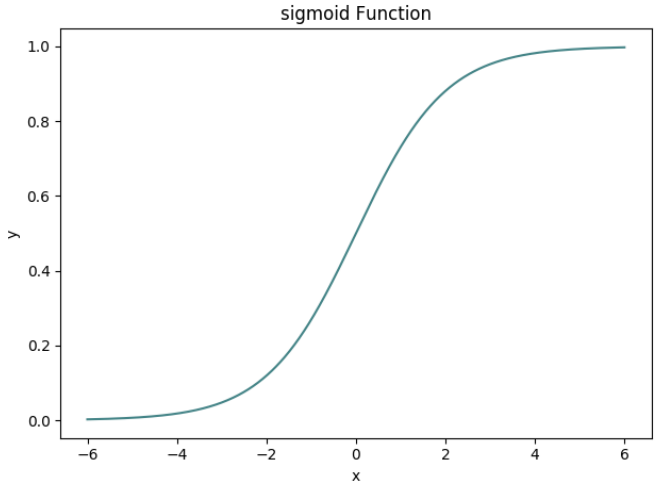
Feed-forward Neural Network



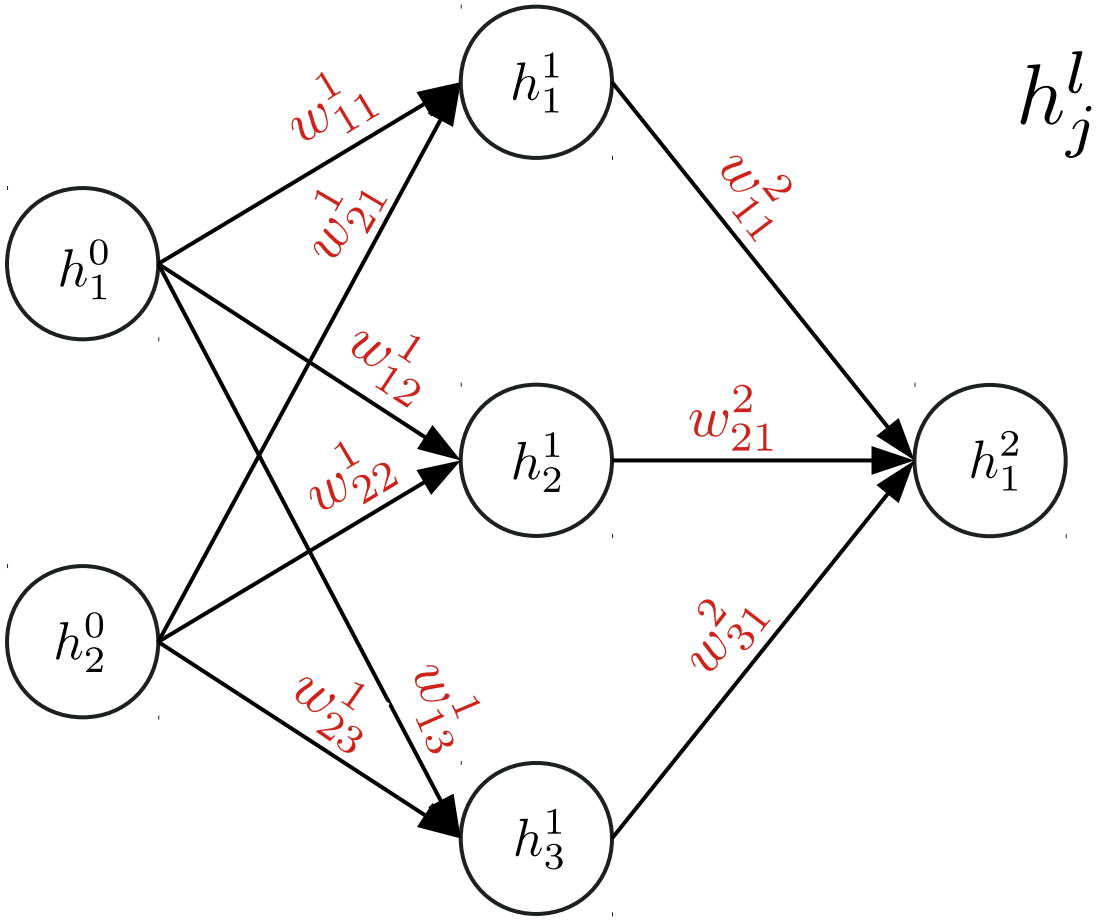
$$h_j^l = \sigma\left(\sum_i h_i^{l-1} w_{ij}^l\right)$$

Activation Function:

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



Feed-forward Neural Network



$$h_j^l = \sigma\left(\sum_i h_i^{l-1} w_{ij}^l\right)$$

Equivalent:

$$\mathbf{x} = \mathbf{h}^0 = \begin{bmatrix} h_1^0 & h_2^0 \end{bmatrix}$$

$$\mathbf{h}^1 = \begin{bmatrix} h_1^1 & h_2^1 & h_3^1 \end{bmatrix}$$

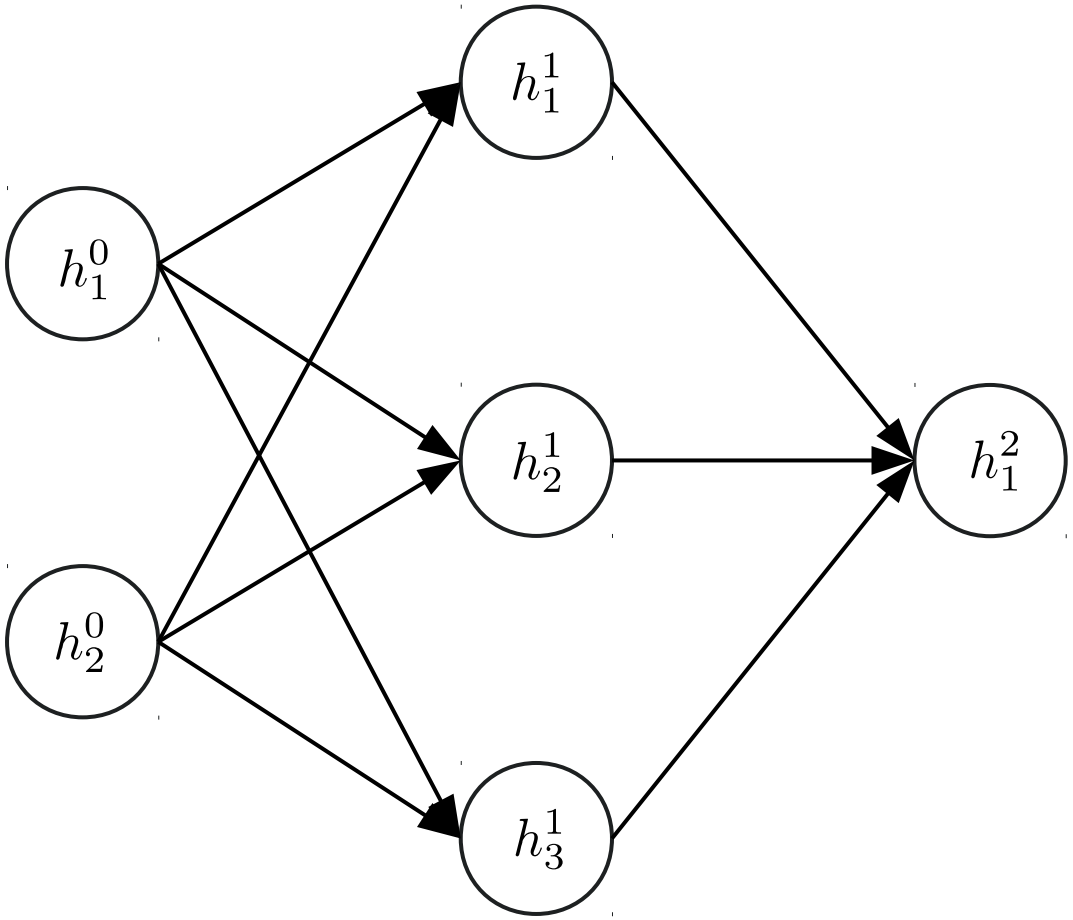
$$\hat{\mathbf{y}} = \mathbf{h}^2 = \begin{bmatrix} h_1^2 \end{bmatrix}$$

$$\mathbf{W}^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} w_{11}^2 \\ w_{21}^2 \\ w_{31}^2 \end{bmatrix}$$

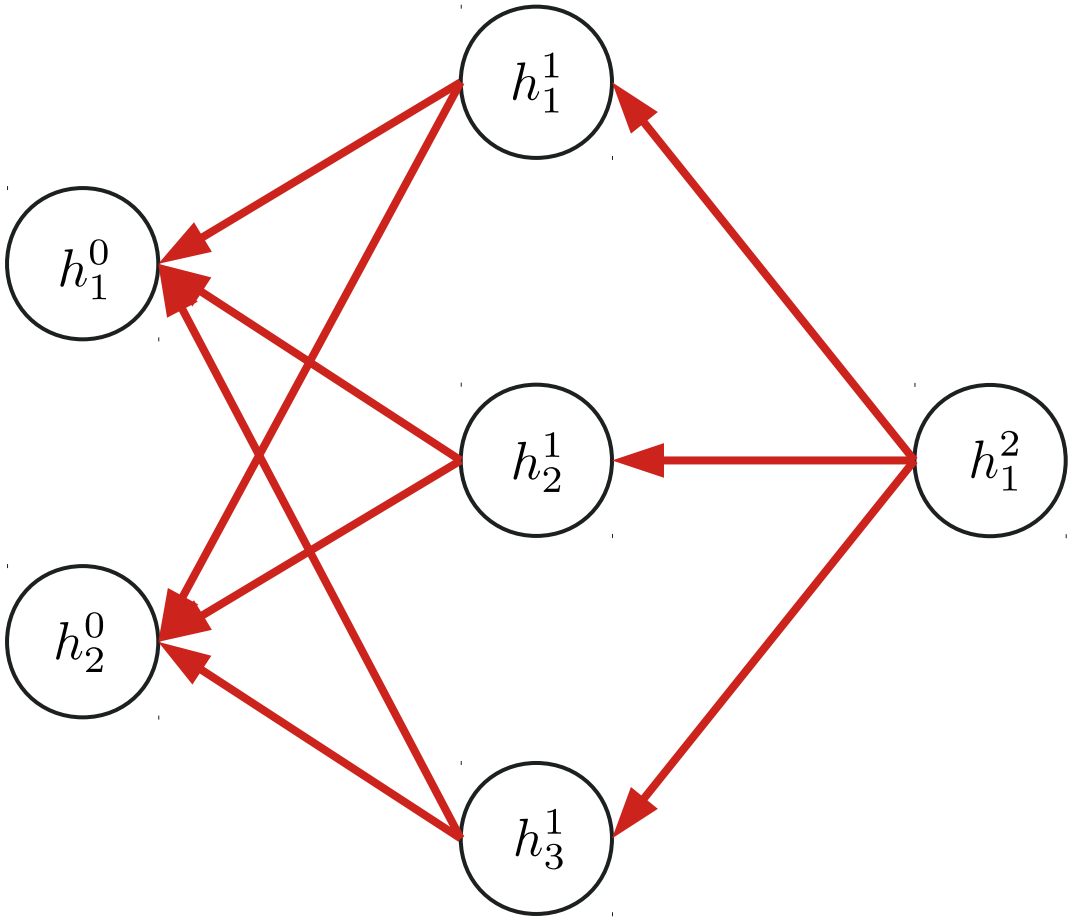
$$\mathbf{h}^l = \sigma\left(\mathbf{h}^{l-1}\mathbf{W}^l\right)$$

Feed-forward Neural Network



- Reminder: knowing the derivative $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$ is required for gradient descent
- Can automatically be calculated with **backpropagation**

Feed-forward Neural Network



- Reminder: knowing the derivative $\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial \Theta}$ is required for gradient descent
- Can automatically be calculated with **backpropagation**
- Requires calculating gradient flow for each arrow

Dense Layer

$$\mathbf{h}^l = \sigma\left(\mathbf{h}^{l-1}\mathbf{W}^l + \mathbf{b}^l\right)$$

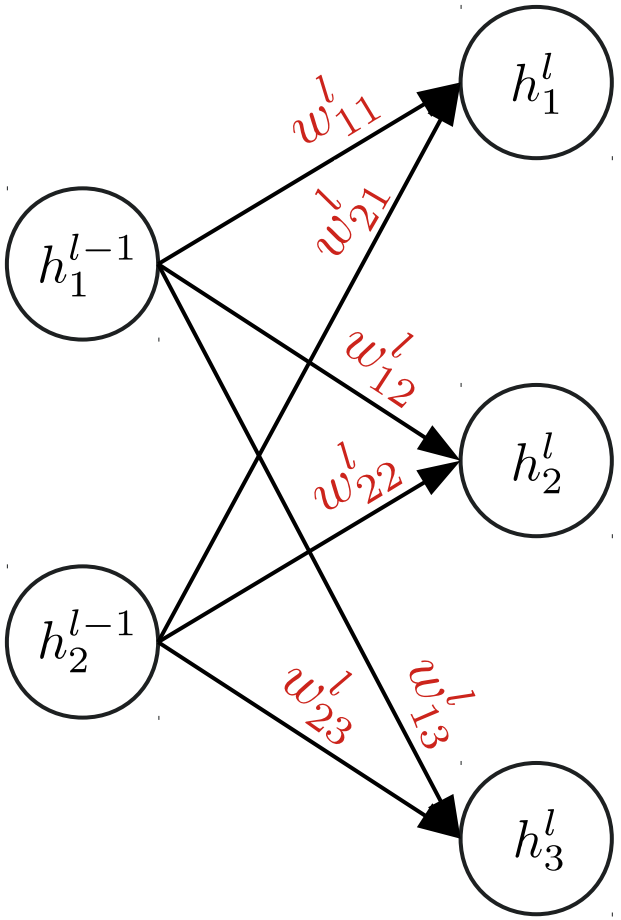
- Takes d_{in}-dimensional input

$$\mathbf{h}^{l-1} = \begin{bmatrix} h_1^{l-1} & \dots & h_{d_{\text{in}}}^{l_1} \end{bmatrix}$$

- Produces d_{out}-dimensional output

$$\mathbf{h}^l = \begin{bmatrix} h_1^l & \dots & h_{d_{\text{out}}}^l \end{bmatrix}$$

- Has $d_{\text{in}} \times d_{\text{out}}$-dimensional matrix of synapse weights $\mathbf{W}^l$ and d_{out}-dimensional vector of bias weights $\mathbf{b}^l$
  - Parameters: $\Theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$
- Has hyperparameter of activation function σ (more on that later)



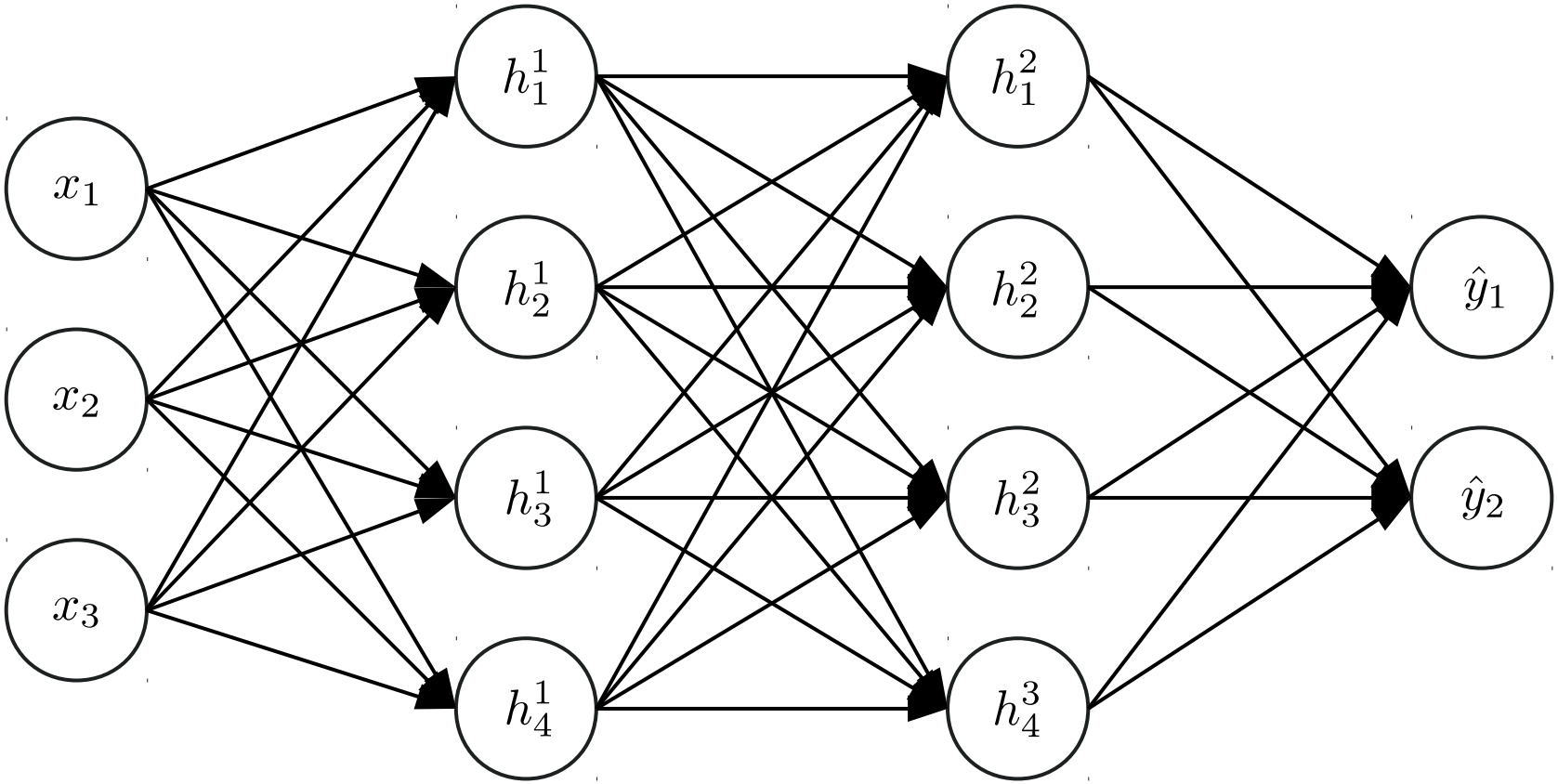
Remarks

- Usually, you can infer a variable's type from its name
  - Lowercase italic letters are **scalars**: $x = 3$
  - Lowercase bold letters are (row-) **vectors**: $\mathbf{v} = \begin{bmatrix} 4 & -2 & 5 \end{bmatrix}$
  - Uppercase bold letters are **matrices**: $\mathbf{M} = \begin{bmatrix} 1 & 7 \\ 3 & 8 \end{bmatrix}$

- People draw neural networks from left-to-right, bottom-to-top, top-to bottom
  - All mean the same thing
  - Look for arrow direction

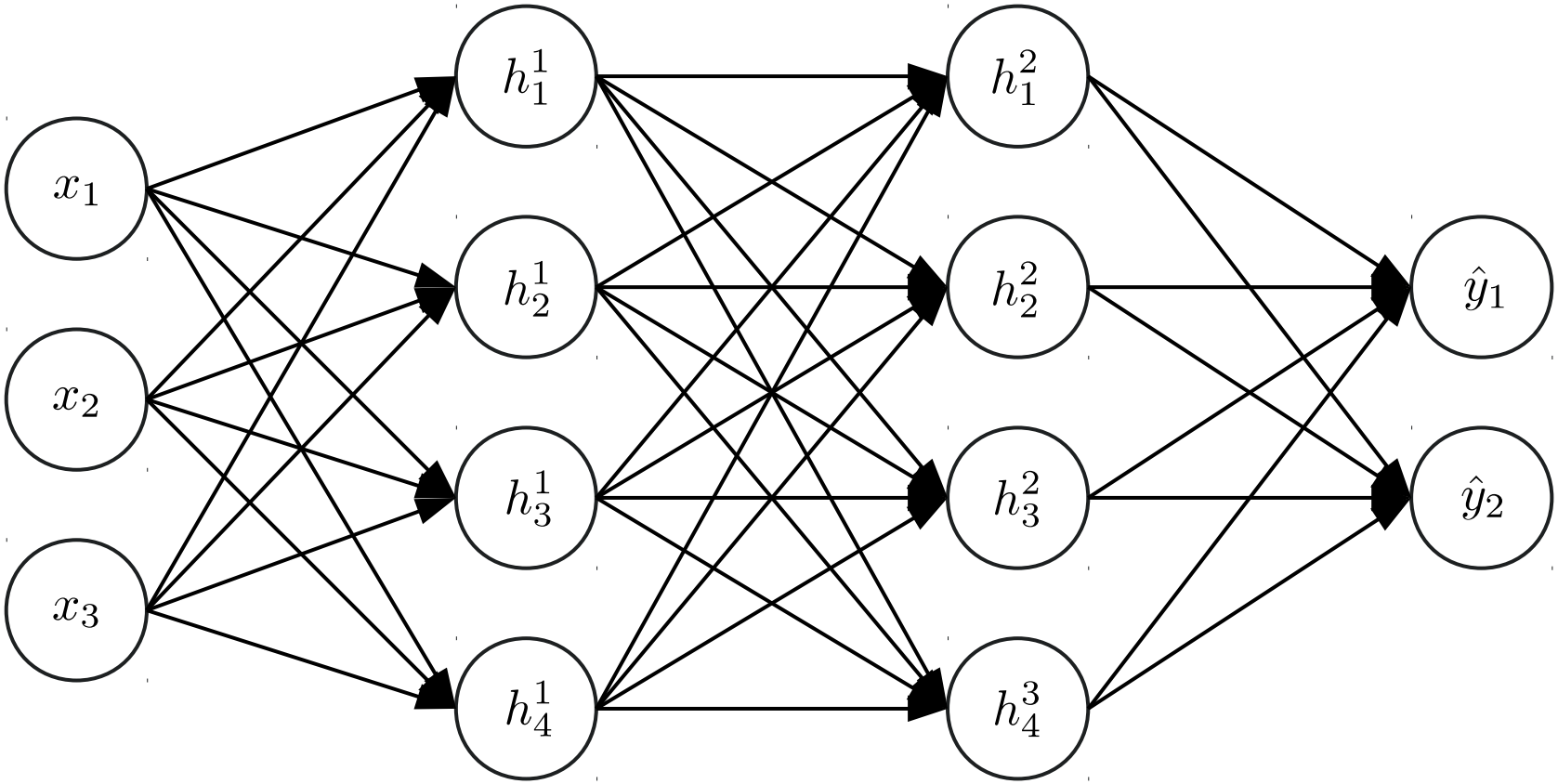
Multi-Layer Neural Network

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1\right)\mathbf{W}^2\right)\mathbf{W}^3\right)$$



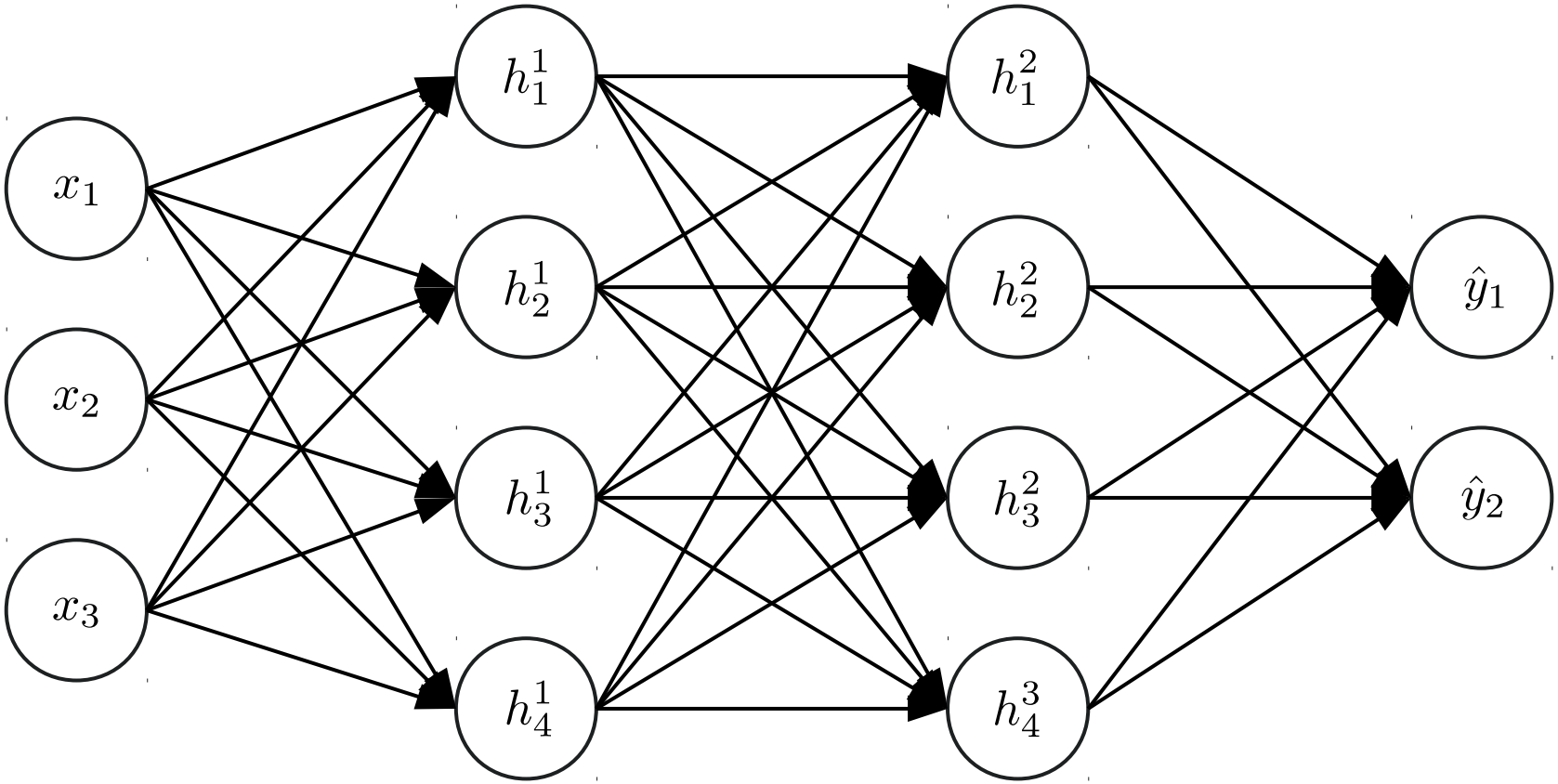
Multi-Layer Neural Network with Bias

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$



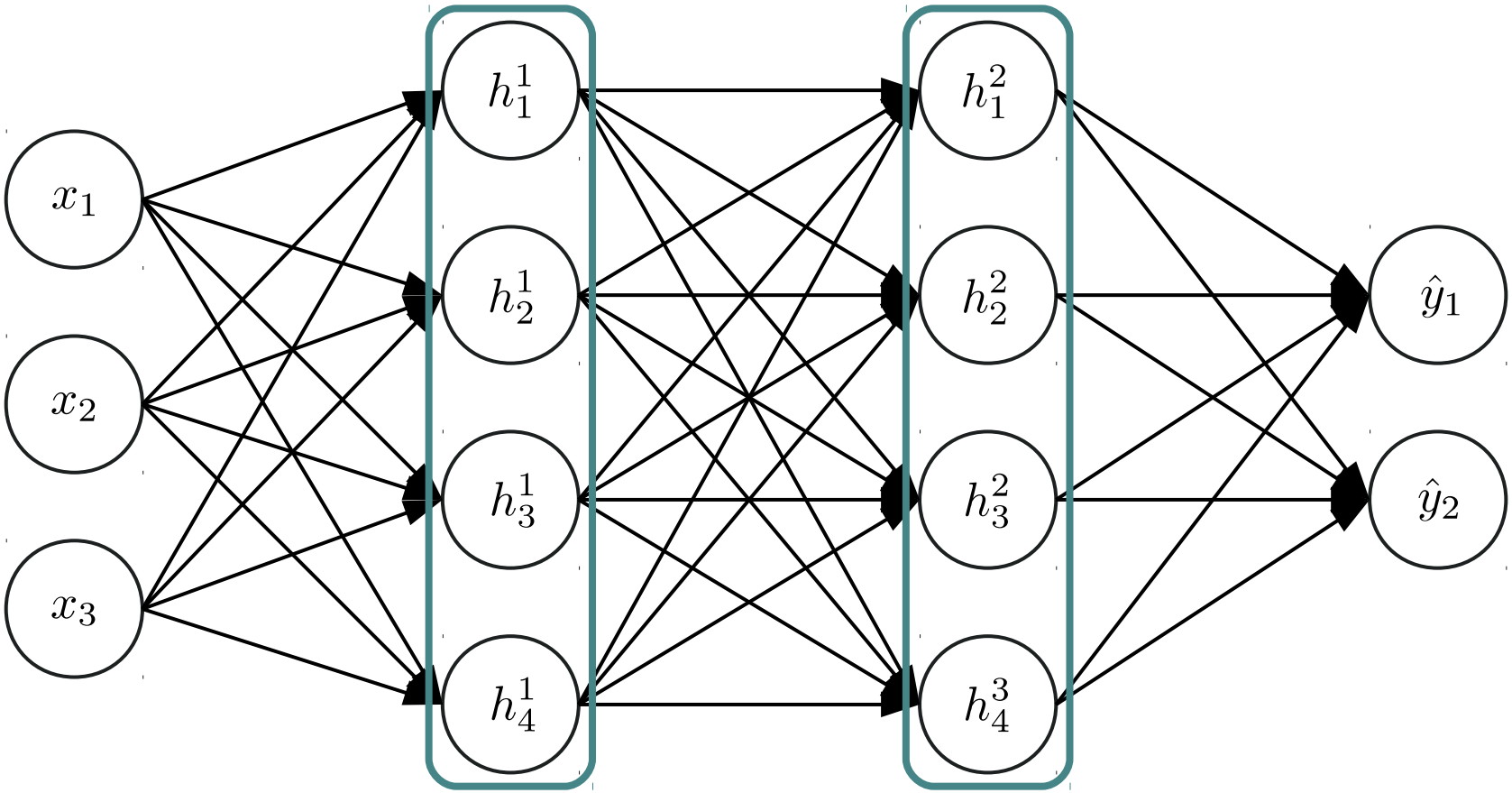
How Many Layers?

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$



How Many Layers?

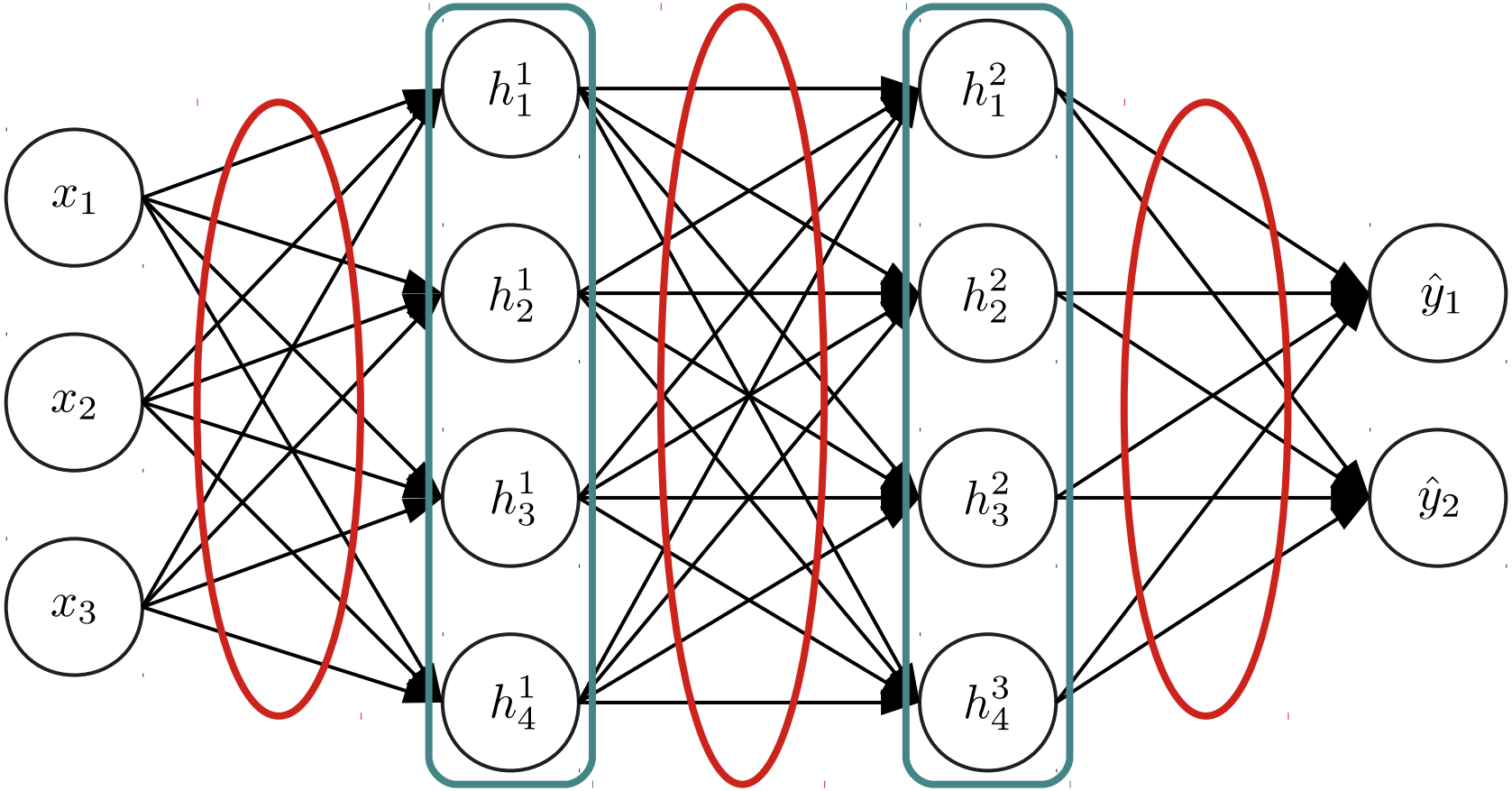
$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$



2 Hidden Layers

How Many Layers?

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$

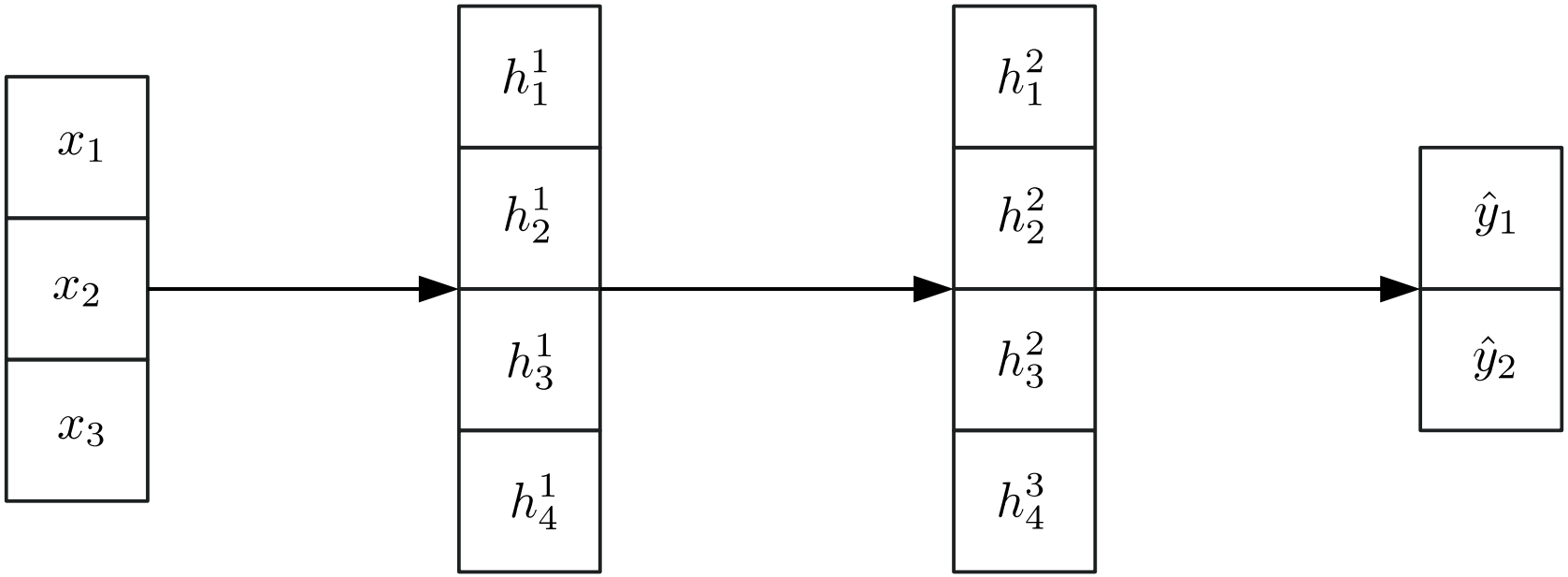


2 Hidden Layers

3 Dense Layers

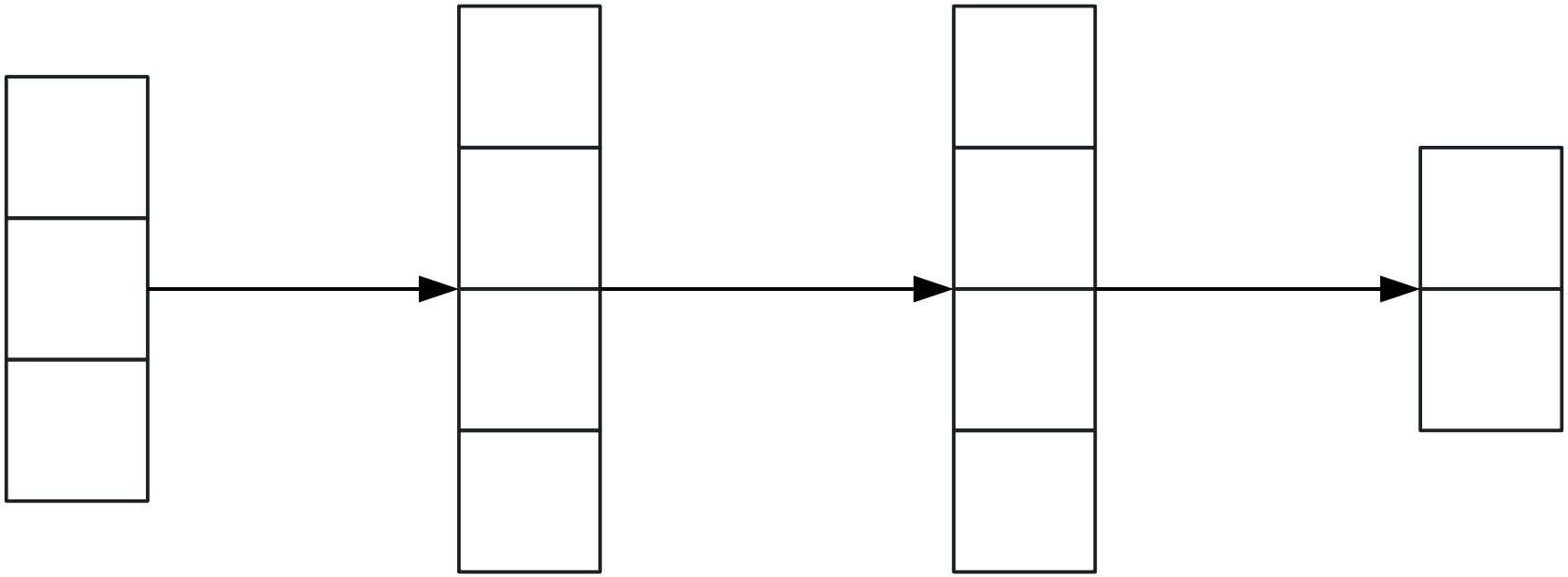
Multi-Layer Neural Network Simplified

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$



Multi-Layer Neural Network Simplified

$$\hat{\mathbf{y}} = \sigma\left(\sigma\left(\sigma\left(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2\right)\mathbf{W}^3 + \mathbf{b}^3\right)$$



Deep vs Shallow Neural Networks

» **Theoretical result**: A network with two dense layers can approximate any given function—that matches its input/output structure—to an arbitrary small degree

Deep vs Shallow Neural Networks

- **Theoretical result**: A network with two dense layers can approximate any given function—that matches its input/output structure—to an arbitrary small degree
  - However, this theorem does not tell us how many neurons per layer are necessary for good approximation (can be very many)
- **Practical result**: at the same number of parameters, deeper neural networks have **higher representation power** and are **easier to find good parameters** for

(Side note: which number of layers people call deep/shallow strongly depends on the task/discipline at hand and changes over time)

Activation Function

- Many choices for activation function
- Popular choices:

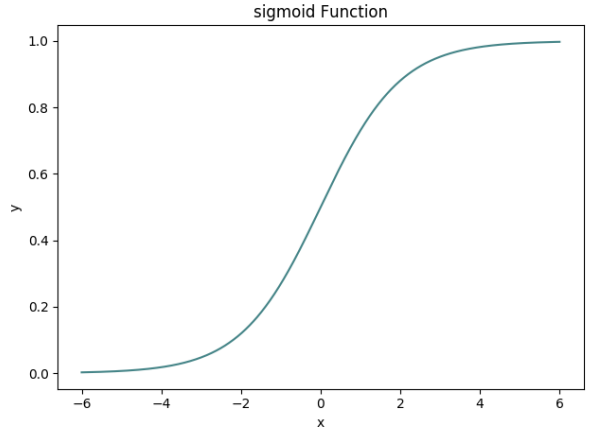
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

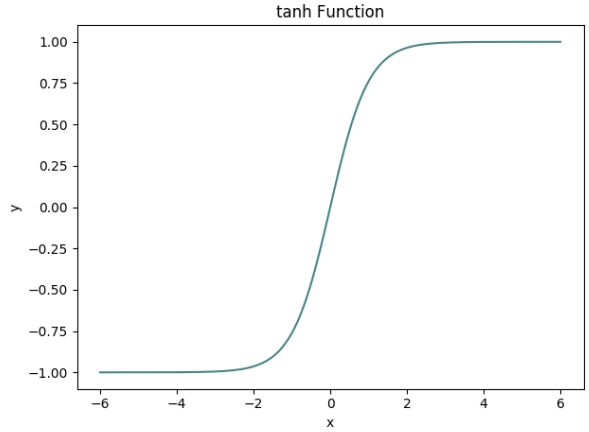
$$\tanh(x) = \frac{1+e^{-2x}}{1-e^{-2x}}$$

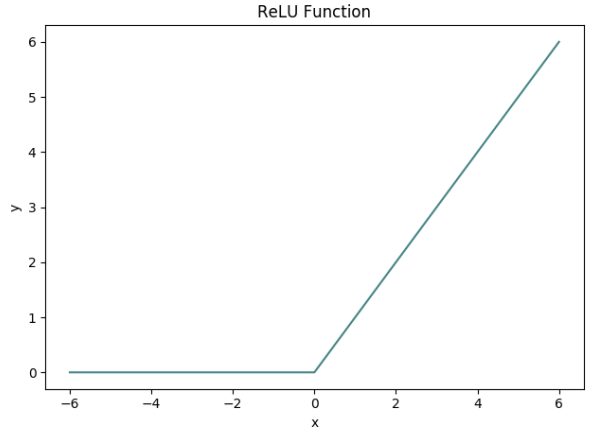
$$\mathrm{ReLU}(x) = \max(0, x)$$

$$\mathrm{id}(x) = x$$

- Main purpose is to control range of possible values of layer output
- For multiple layers to increase representation power, all intermediary activation functions must be non-linear







Softmax

- Often one wants to have the network output be a probability distribution over multiple items
  - Probabilities should lie in $[0, 1]$
  - Sum of probabilities should be one
- Solution: softmax as activation function

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{d_{\text{out}}} e^{z_k}}$$

- Input: vector $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_{d_{\text{out}}} \end{bmatrix}$ of arbitrary real numbers
- Output: probability distribution

Hyperparameters

- Hyperparameters of a machine learning model are parameters that are not learned during training
  - Usually because we can not calculate the model function's derivative with respect to them
  - Thus can not learn them with gradient descent

Hyperparameters

- Hyperparameters of a machine learning model are parameters that are not learned during training
  - Usually because we can not calculate the model function's derivative with respect to them
  - Thus can not learn them with gradient descent
- Examples:
  - choice of activation function
  - number of dense layers
  - dimensionality of hidden layers
- Need to be chosen through empirical experimentation

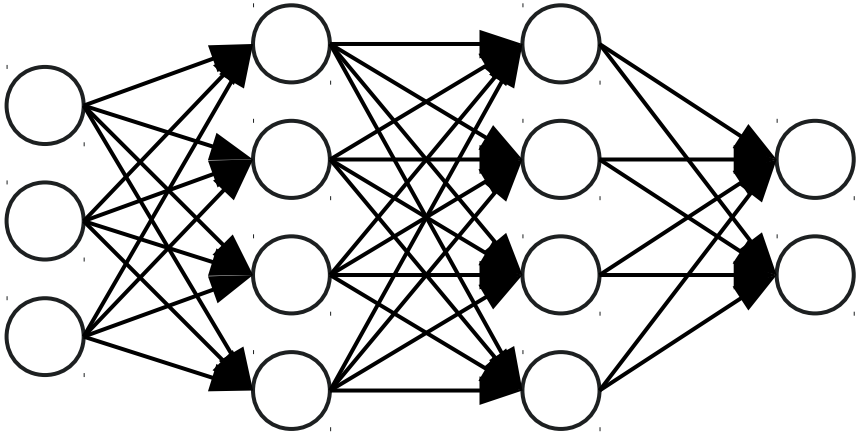
Common Neural Network Techniques

- Usually it is not enough to have a model function that could theoretically approximate the target function
- In practice, finding the correct parameters can be hard
- Some advanced techniques that are empirically beneficial:
  - Dropout
  - Layer Normalization
  - Residual Connections

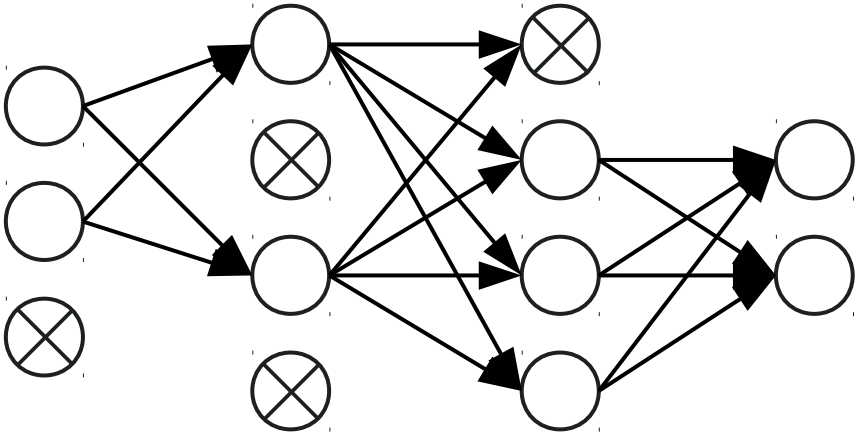
Dropout (Srivastava et al, 2014)

- Simple regularization technique
- During each training step: ignore a neuron with probability p
- *"Network can't rely on specific neurons and is forced to generalize"*
- Usually implemented as a **dropout layer** that sets selected neurons to zero

Standard Network:



After applying dropout:



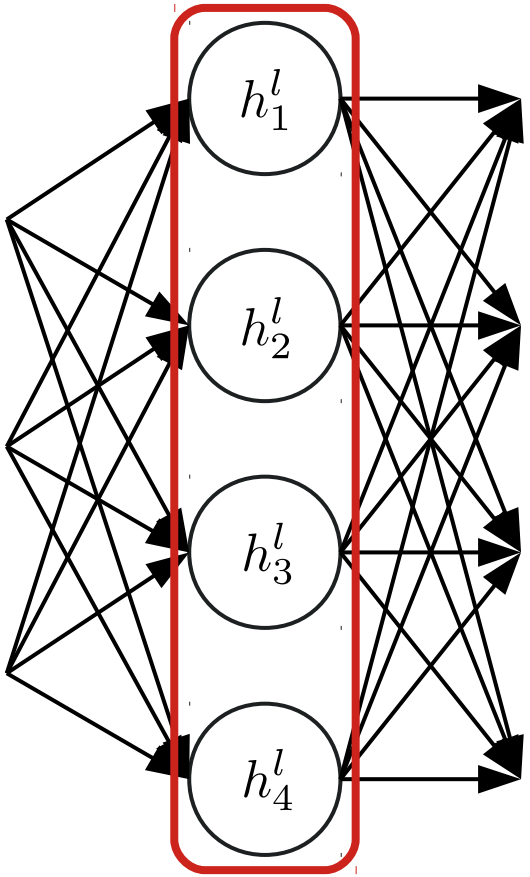
Srivastava et al (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *JMLR*.

Layer Normalization (Ba et al, 2016)

- Normalize output of a layer so that its mean is zero and its variance is one

$$\bar{\mathbf{h}}^l = \frac{\mathbf{h}^l - \mu^l}{\sigma^l}$$

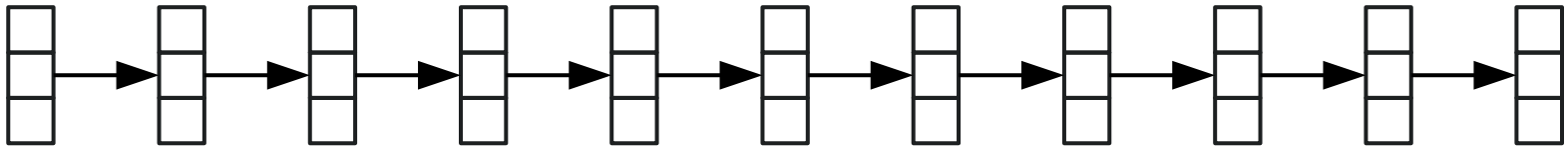
  - μ^l mean of layer
  - σ^l variance of layer
- *"Makes network robuster and thus train better"*
- Implemented as **normalization layer**



Ba et al (2016). "Layer Normalization". *CoRR abs/1607.06450*.

Residual Connections (He et al, 2016)

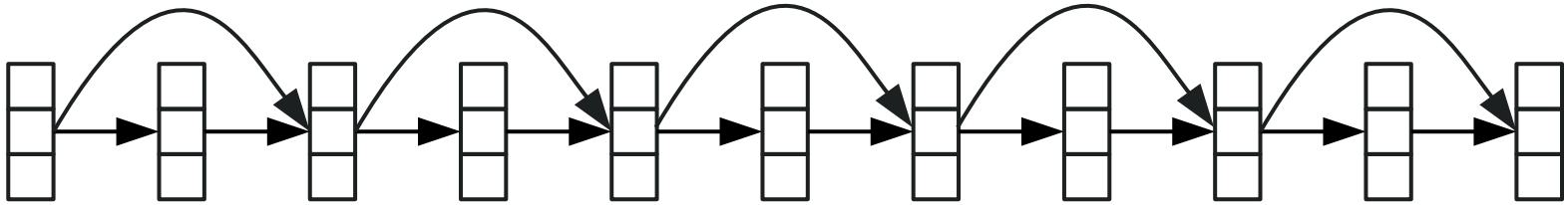
» Deep neural networks can have problems with vanishing and exploding gradients



He et al (2016). "Deep Residual Learning for Image Recognition". *CVPR*.

Residual Connections (He et al, 2016)

» Deep neural networks can have problems with vanishing and exploding gradients



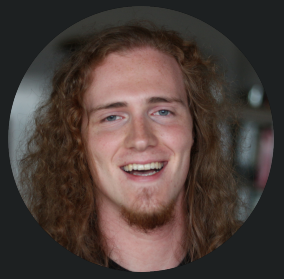
» Residual connections skip layers: $x = x + \text{Layer}(x)$

» *"Help gradients flow better"*

He et al (2016). "Deep Residual Learning for Image Recognition". *CVPR*.

Summary: Deep Learning Basics

- Neural networks have fixed input and output dimension
- Deep feed-forward neural networks are stacks of dense layers
- Finding good hyperparameter values requires lots of experimentation
- Dropout, layer normalization, and residual connections help networks learn better



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Introduction to Advanced Neural Network Architectures

for Natural Language Processing

» Established Architectures

How to do Sequence Classification?

- How can we use neural networks for sequence classification?
- For example **sentiment analysis**:
  "the voice quality of the phone is amazing" → ***Positive Sentiment***
  - Input: *sequence of words* $w_1, w_2, \ldots, w_n$
  - Output: ***Positive Sentiment*** or ***Negative Sentiment***
  - Assume lots of training data available

How to do Sequence Classification?

- How can we use neural networks for sequence classification?
- For example **sentiment analysis**:
  "the voice quality of the phone is amazing" → ***Positive Sentiment***
  - Input: *sequence of words* $w_1, w_2, \ldots, w_n$
  - Output: ***Positive Sentiment*** or ***Negative Sentiment***
  - Assume lots of training data available
- Output is easy, in this case two-dimensional, so $d_{\text{out}} = 2$
  - Predict the label which receives the higher score
- But we don't know number of input words n and we need some constant value for d_{in}

One Solution: Bag-of-Words

- Count how often each word occurs in the input (discard order)
- Limit yourself to a vocabulary V of $|V|$ most common words

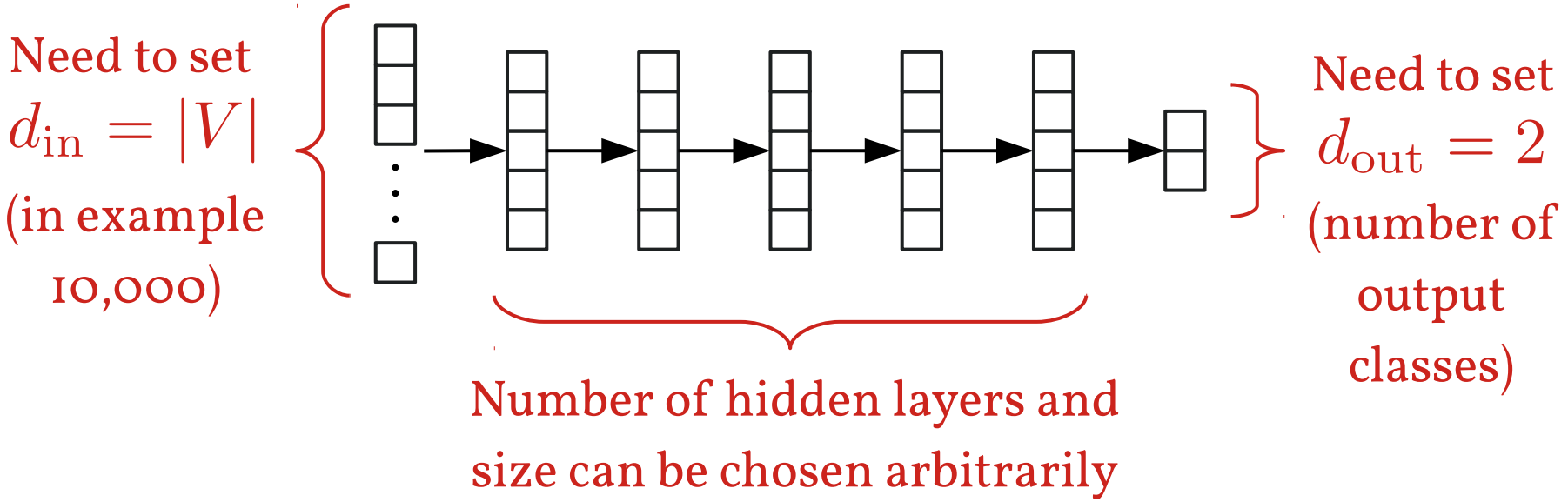
$$V = \underset{1}{[\text{the}}, \underset{2}{\text{of}}, \underset{3}{\text{and}}, \underset{4}{\text{to}}, \ldots, \underset{10,000}{\text{poison}}]$$

- Set $d_{\text{in}} = |V|$ and have input x_i be the number of times the word at index i of the vocabulary V occurred in the input text

$$x_{\text{text}} = \text{“}\textit{the voice quality of the phone is amazing}\text{”}$$

$$\mathbf{x} = \begin{bmatrix} 2 & 1 & 0 & 0 & \ldots & 0 \end{bmatrix}$$

Bag-of-Words Architecture



Problem: Not Enough Data

- Classify the sentiment of the following sentences
  - $x^{(1)} =$ *"the voice quality of the phone is amazing"* $y^{(1)} =$ ***Positive***
  - $x^{(2)} =$ *"the voice quality of the phone is lacking"* $y^{(2)} =$ ***Negative***
  - $x^{(3)} =$ *"the voice quality of the phone is superb"* $y^{(3)} =$ ***?***
- Assume we only have $x^{(1)}$ and $x^{(2)}$ as training data and want to classify the unseen $x^{(3)}$

Problem: Not Enough Data

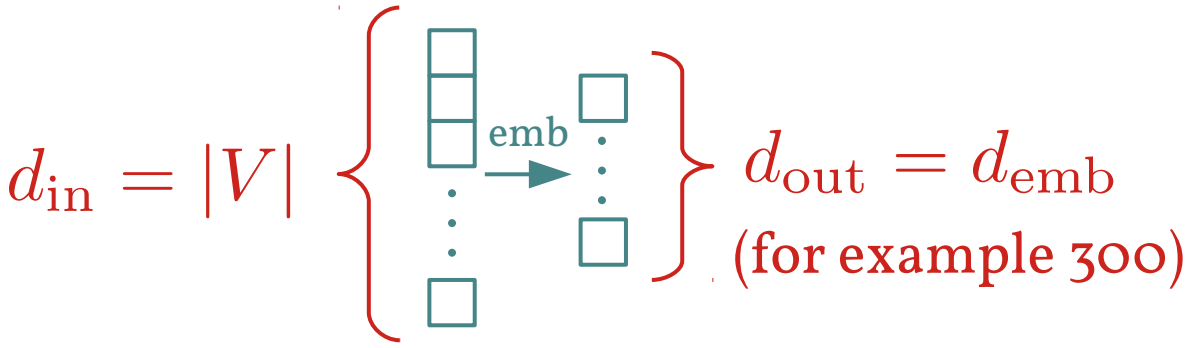
- Classify the sentiment of the following sentences
  - $x^{(1)}$ = *"the voice quality of the phone is amazing"* $y^{(1)}$ = ***Positive***
  - $x^{(2)}$ = *"the voice quality of the phone is lacking"* $y^{(2)}$ = ***Negative***
  - $x^{(3)}$ = *"the voice quality of the phone is superb"* $y^{(3)}$ = ***?***
- Assume we only have $x^{(1)}$ and $x^{(2)}$ as training data and want to classify the unseen $x^{(3)}$
  - Representation would be equally similar to both training examples
  - If we would know that the meaning of *"superb"* is more similar to *"amazing"* than to *"lacking"* we could classify successfully…

Solution: Word Embeddings

- Take (or learn) a function that maps each word into a lower-dimensional space

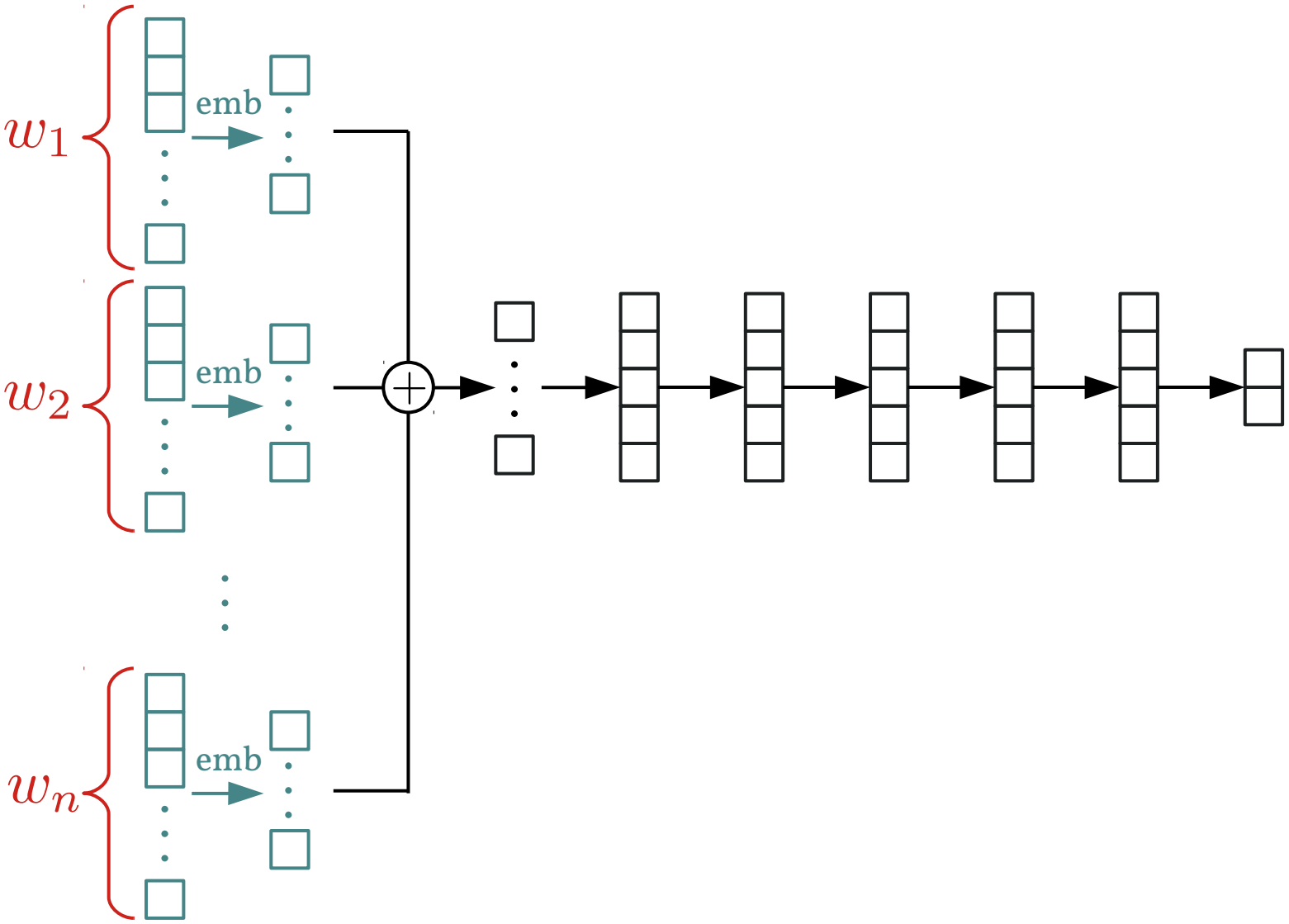
$$\text{emb} : V \to \mathbb{R}^{d_{\text{emb}}}$$

- Can be implemented with a single dense layer with one-hot input vector (only zeros, expect a one at index of word)
  - is equivalent to looking up a specific row in the weight matrix



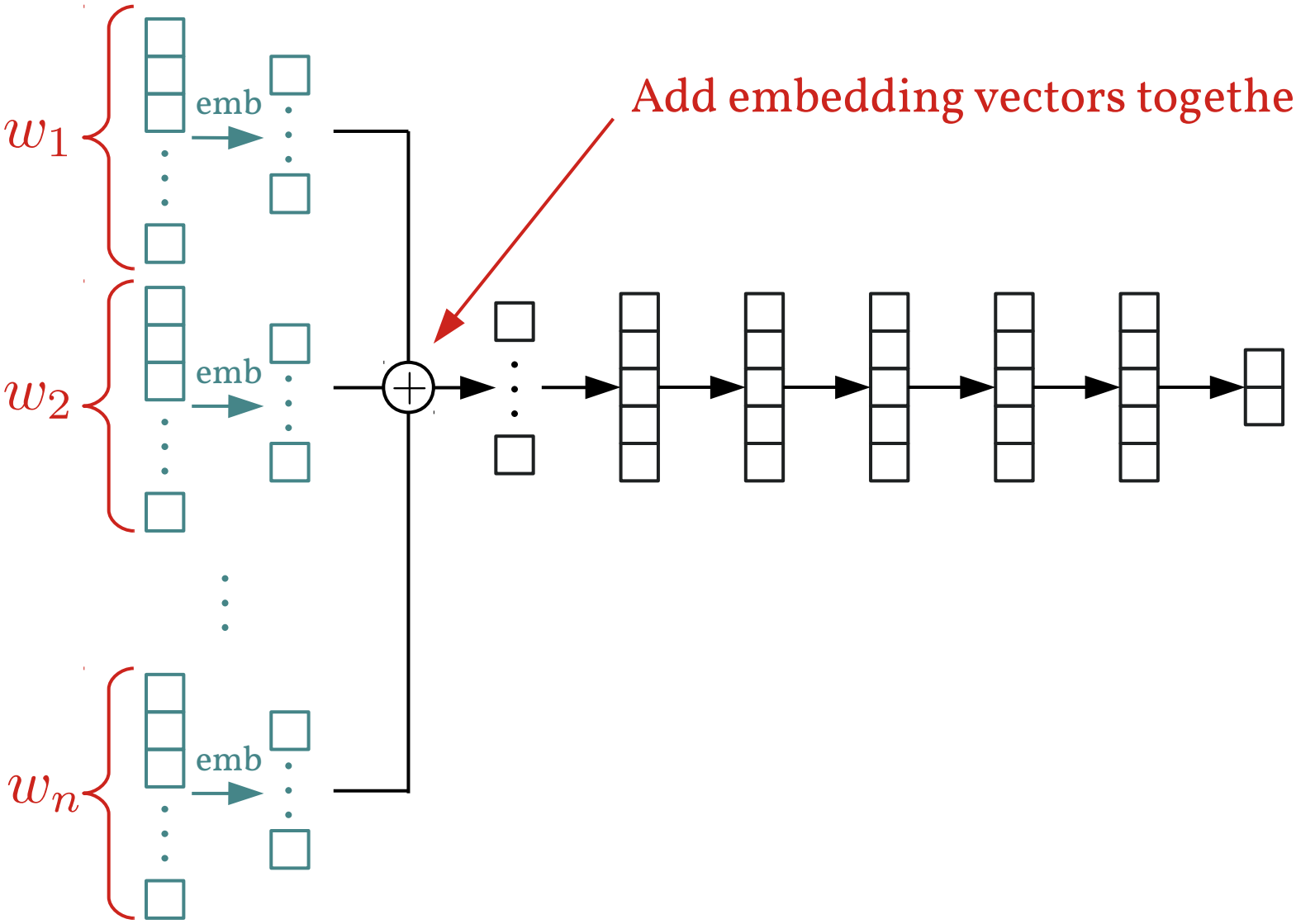
Bag-of-Words with Embeddings

» For text input $w_1, w_2, \ldots, w_n$:



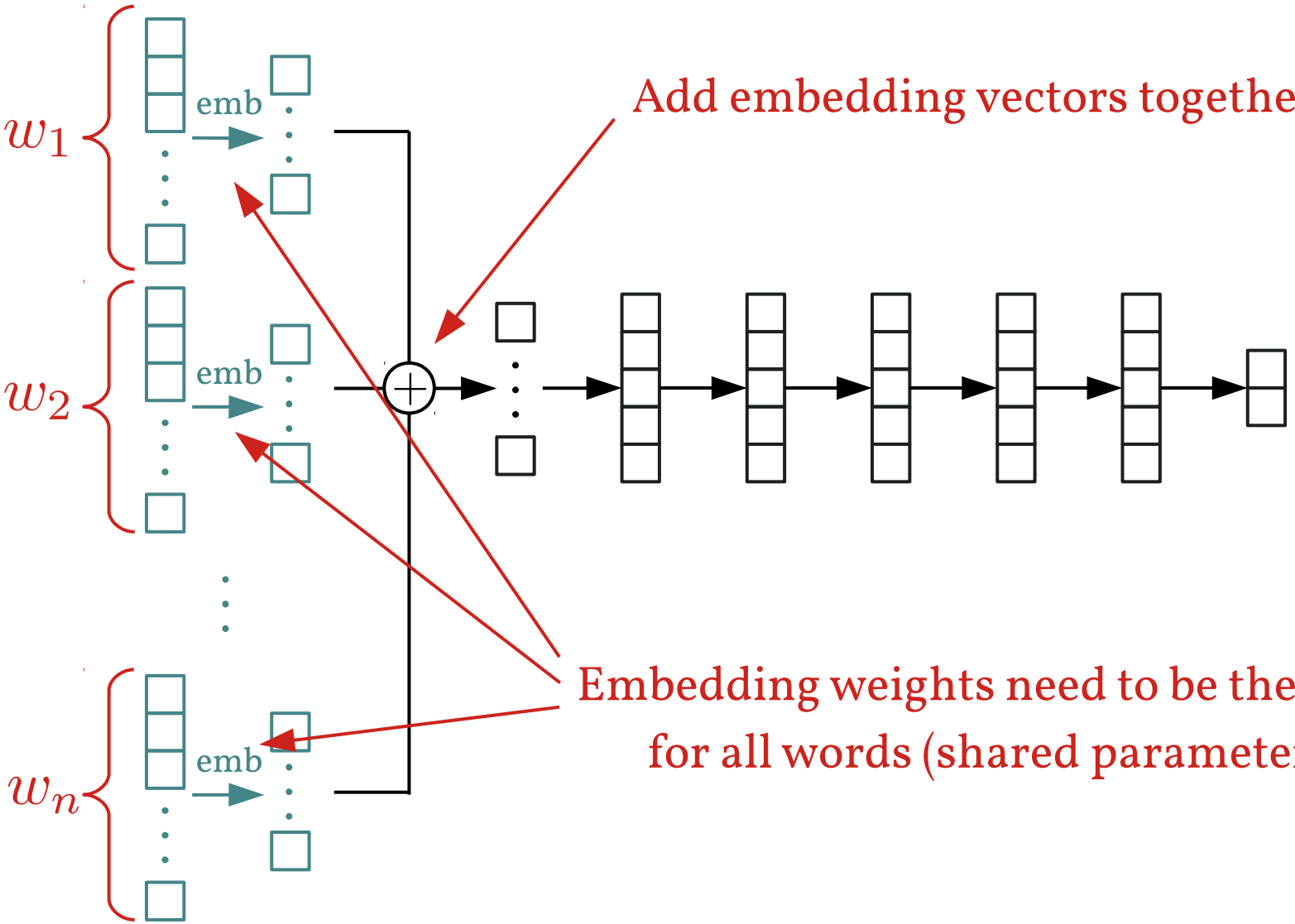
Bag-of-Words with Embeddings

» For text input $w_1, w_2, \ldots, w_n$:



Bag-of-Words with Embeddings

» For text input $w_1, w_2, \ldots, w_n$:



Why do Word Embeddings Help?

- Classify the sentiment of the following sentences
  - $x^{(1)}$ = *"the voice quality of the phone is amazing"* $y^{(1)}$ = ***Positive***
  - $x^{(2)}$ = *"the voice quality of the phone is lacking"* $y^{(2)}$ = ***Negative***
  - $x^{(3)}$ = *"the voice quality of the phone is superb"* $y^{(3)}$ = ***?***
- By definition, word embeddings project words into a lower dimensional vector space
- Projection forces semantically similar words to be near to each other in the vector space
- Therefore, $x^{(3)}$ will be more similar to $x^{(1)}$ than to $x^{(2)}$
  - So, we will predict the correct $y^{(3)}$ = ***Positive***

Why do Word Embeddings Help?

- Classify the sentiment of the following sentences
  - $x^{(1)}$ = *"the voice quality of the phone is amazing"* $y^{(1)}$ = ***Positive***
  - $x^{(2)}$ = *"the voice quality of the phone is lacking"* $y^{(2)}$ = ***Negative***
  - $x^{(3)}$ = *"the voice quality of the phone is superb"* $y^{(3)}$ = ***?***
- By definition, word embeddings project words into a lower dimensional vector space
- Projection forces semantically similar words to be near to each other in the vector space
- Therefore, $x^{(3)}$ will be more similar to $x^{(1)}$ than to $x^{(2)}$
  - So, we will predict the correct $y^{(3)}$ = ***Positive***

(Input of bag-of-word word counts followed by dense layer is actually equal to the presented embedding mechanism)

Does Word Oder Matter?

» *"the phone did not work"*

vs *"work the phone did not"*

Does Word Oder Matter?

- *"the phone did not work"*
  vs *"work the phone did not"*
  - In bag-of-words both are represented the same way
  - But intended semantics are probably the same

Does Word Oder Matter?

- *"the phone did not work"*
  vs *"work the phone did not"*
  - In bag-of-words both are represented the same way
  - But intended semantics are probably the same
- *"I don't hate the phone, I love it!"*
  vs *"I don't love the phone, I hate it!"*
  - In bag-of-words both are represented the same way
  - But expressed semantics are opposite

Review: Dense Layer

$$\mathbf{h}^l = \sigma\left(\mathbf{h}^{l-1}\mathbf{W}^l + \mathbf{b}^l\right)$$

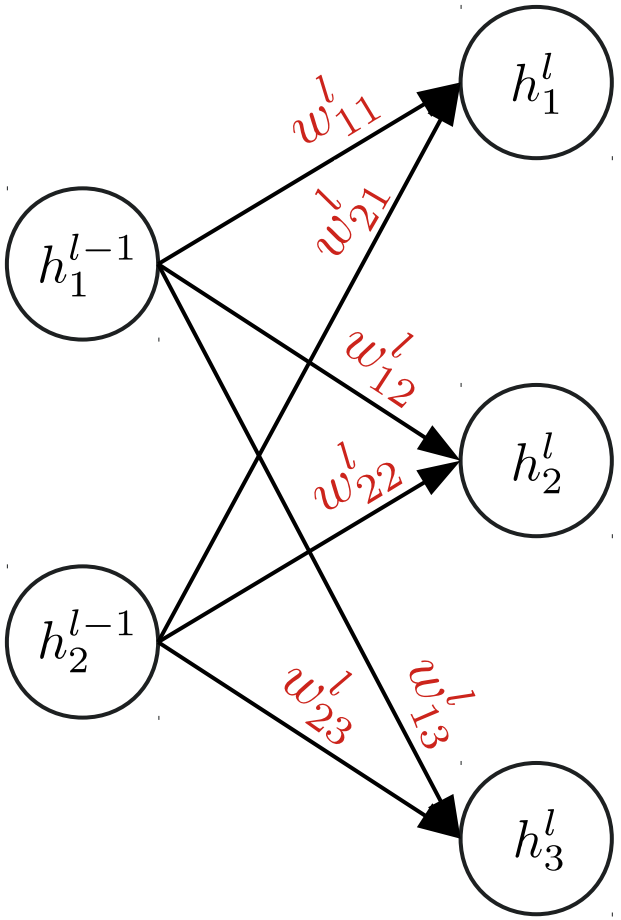
- Takes d_{in}-dimensional input

$$\mathbf{h}^{l-1} = \begin{bmatrix} h_1^{l-1} & \dots & h_{d_{\text{in}}}^{l_1} \end{bmatrix}$$

- Produces d_{out}-dimensional output

$$\mathbf{h}^l = \begin{bmatrix} h_1^l & \dots & h_{d_{\text{out}}}^l \end{bmatrix}$$

- Has $d_{\text{in}} \times d_{\text{out}}$-dimensional matrix of synapse weights $\mathbf{W}^l$ and d_{out}-dimensional vector of bias weights $\mathbf{b}^l$
  - Parameters: $\Theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$
- Has hyperparameter of activation function σ

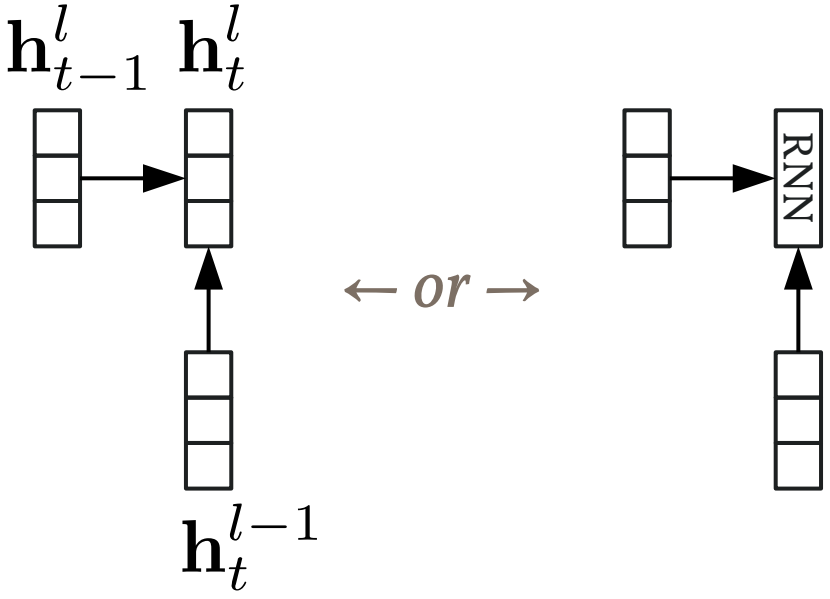


Recurrent Layer

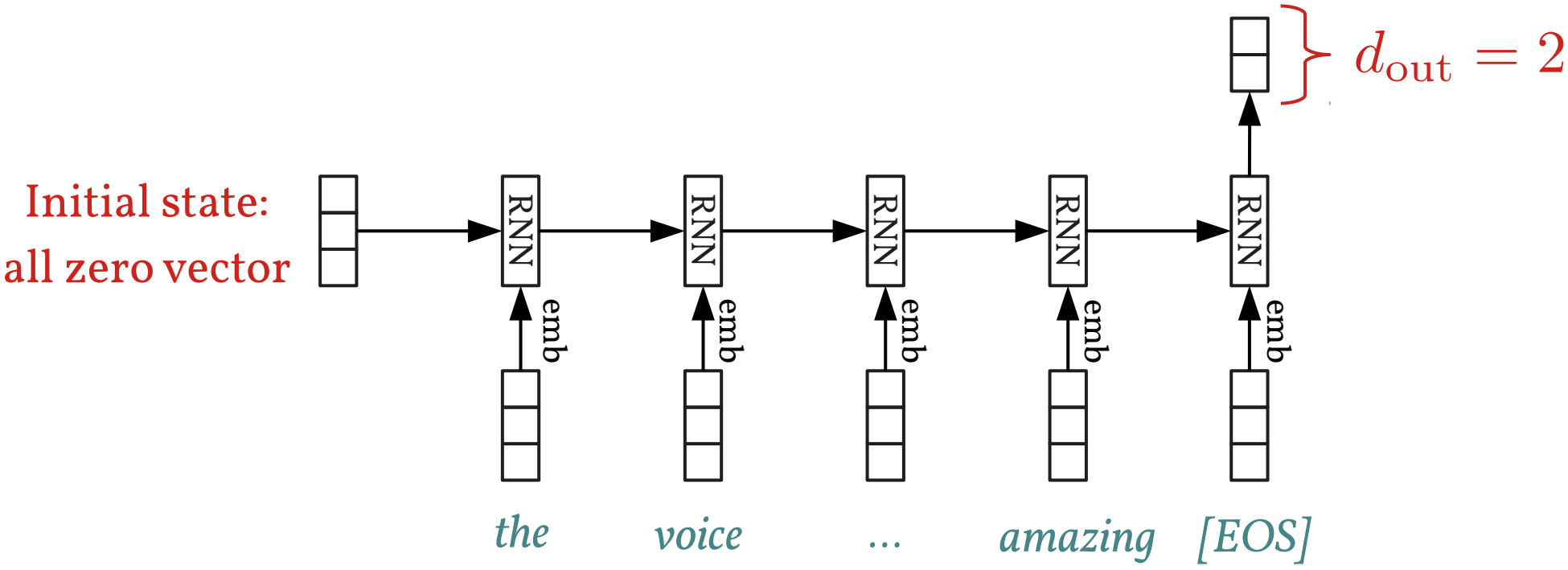
- Basically, just like dense layer but takes two input vectors
  - previous layer output $\mathbf{h}_t^{l-1}$
  - previous own state $\mathbf{h}_{t-1}^{l}$
- Trainable parameters

  $\Theta^l = \{\mathbf{W}^l, \mathbf{V}^l, \mathbf{b}^l\}$
- Used to handle sequential inputs

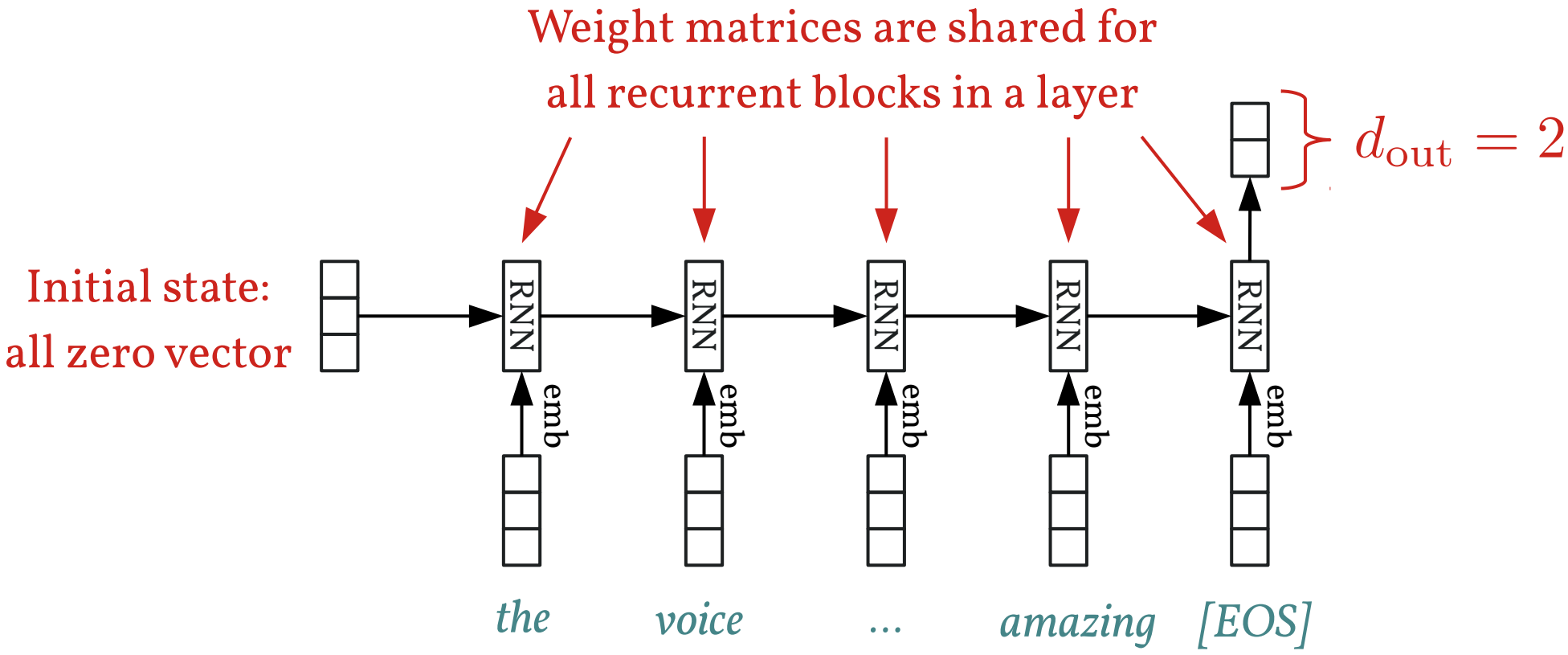
$$\mathbf{h}_t^l = \sigma(\mathbf{h}_t^{l-1}\mathbf{W}^l + \mathbf{h}_{t-1}^l\mathbf{V}^l + \mathbf{b}^l)$$



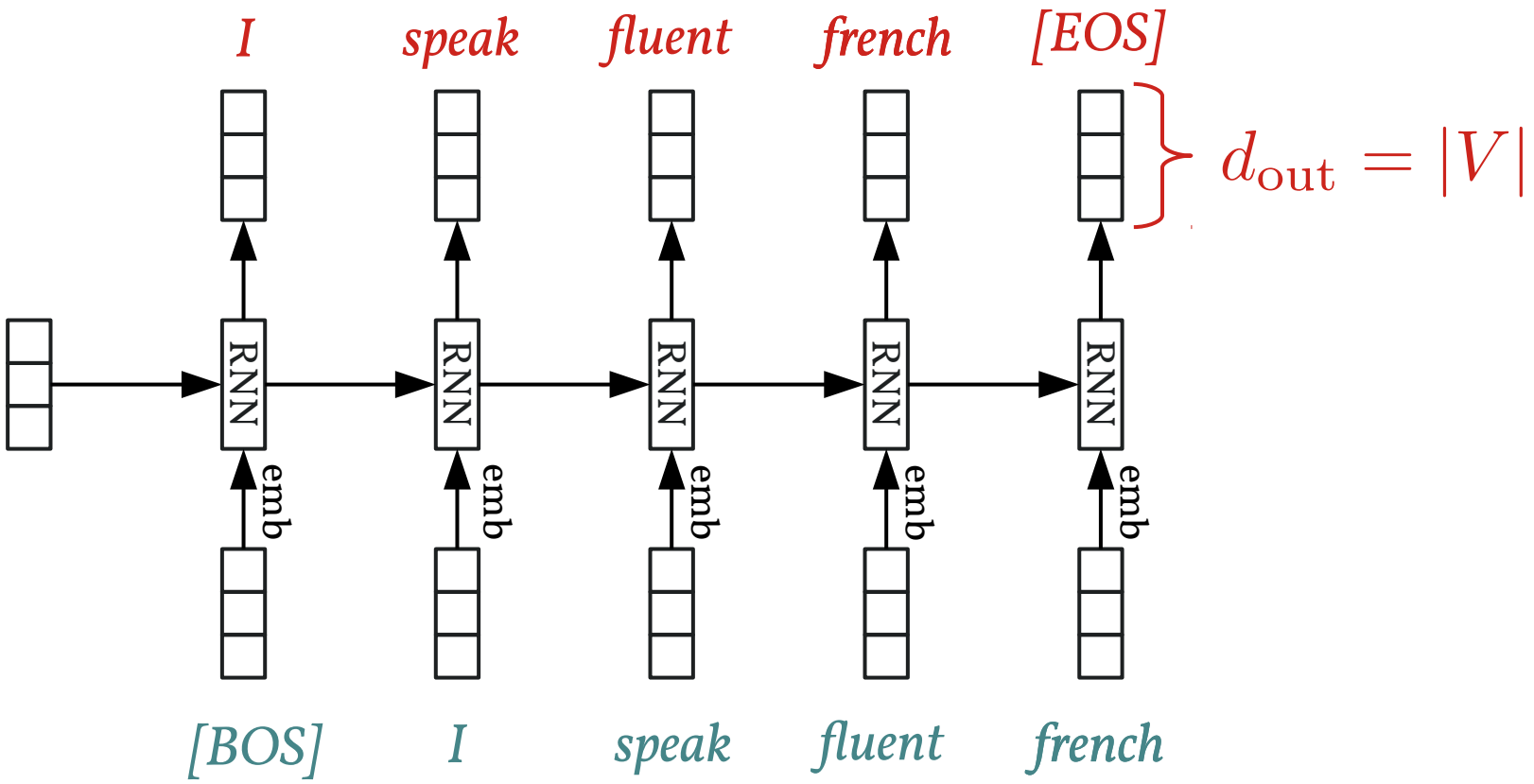
Recurrent Neural Network for Sequence Classification



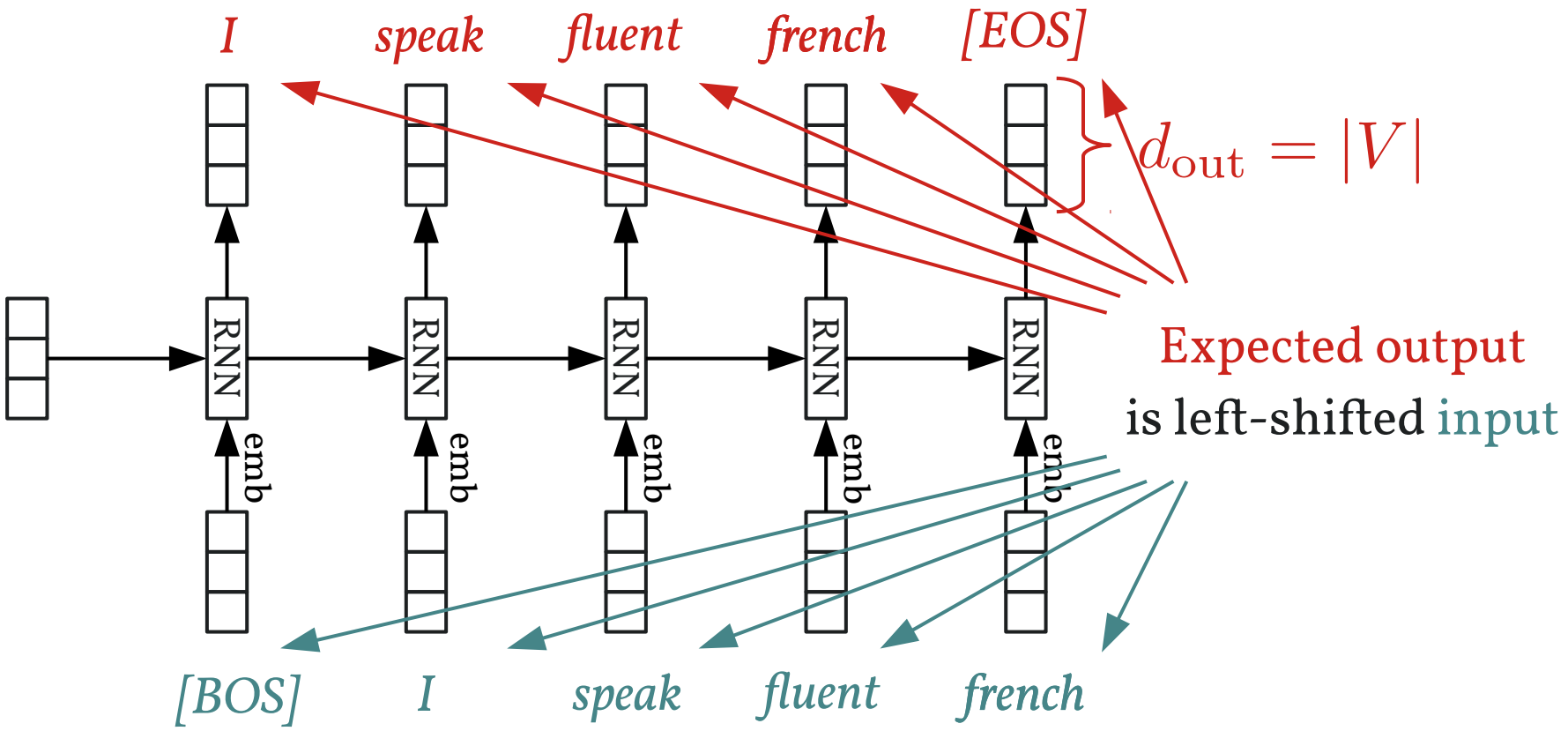
Recurrent Neural Network for Sequence Classification



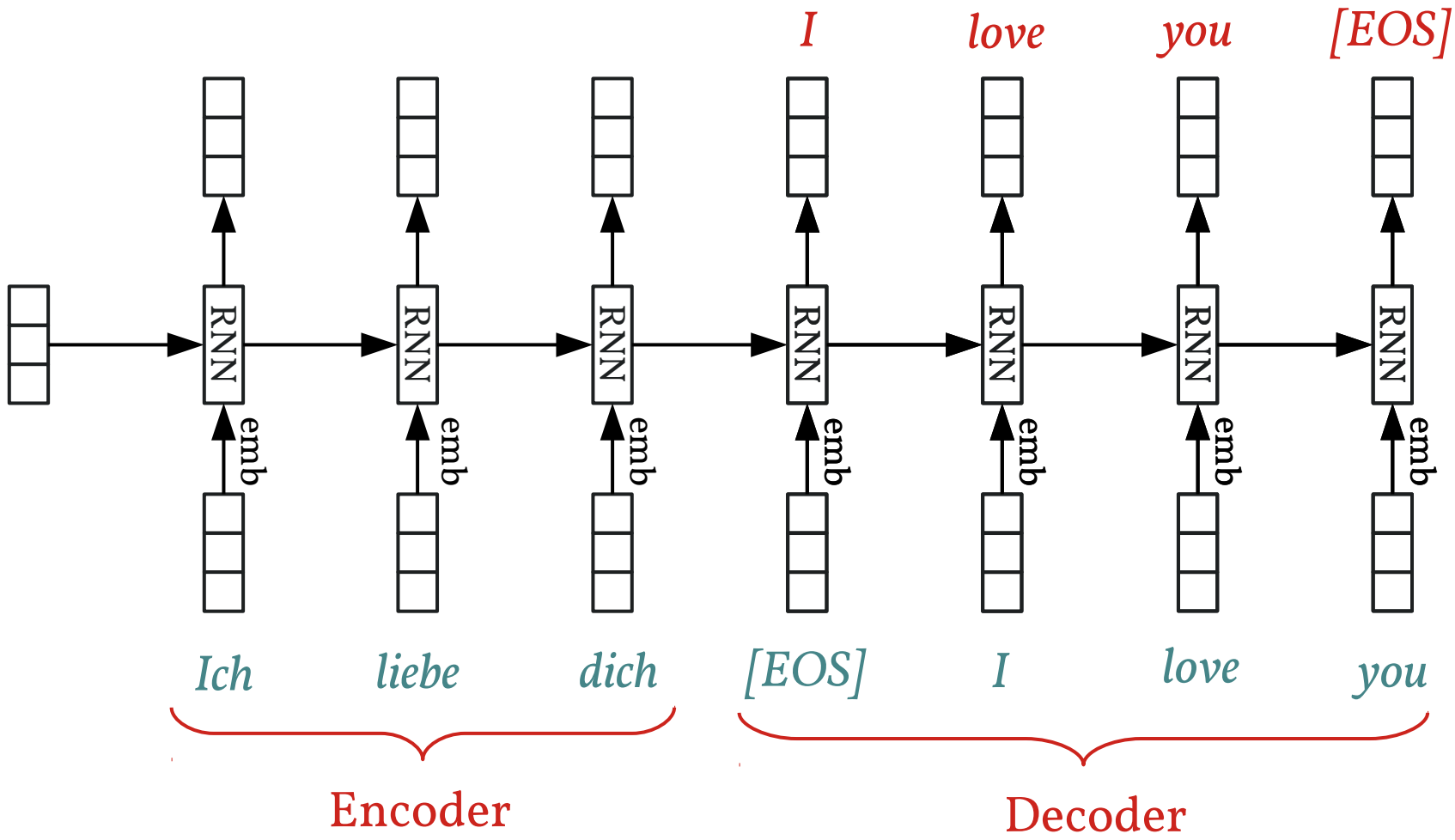
Recurrent Neural Network for Language Modeling



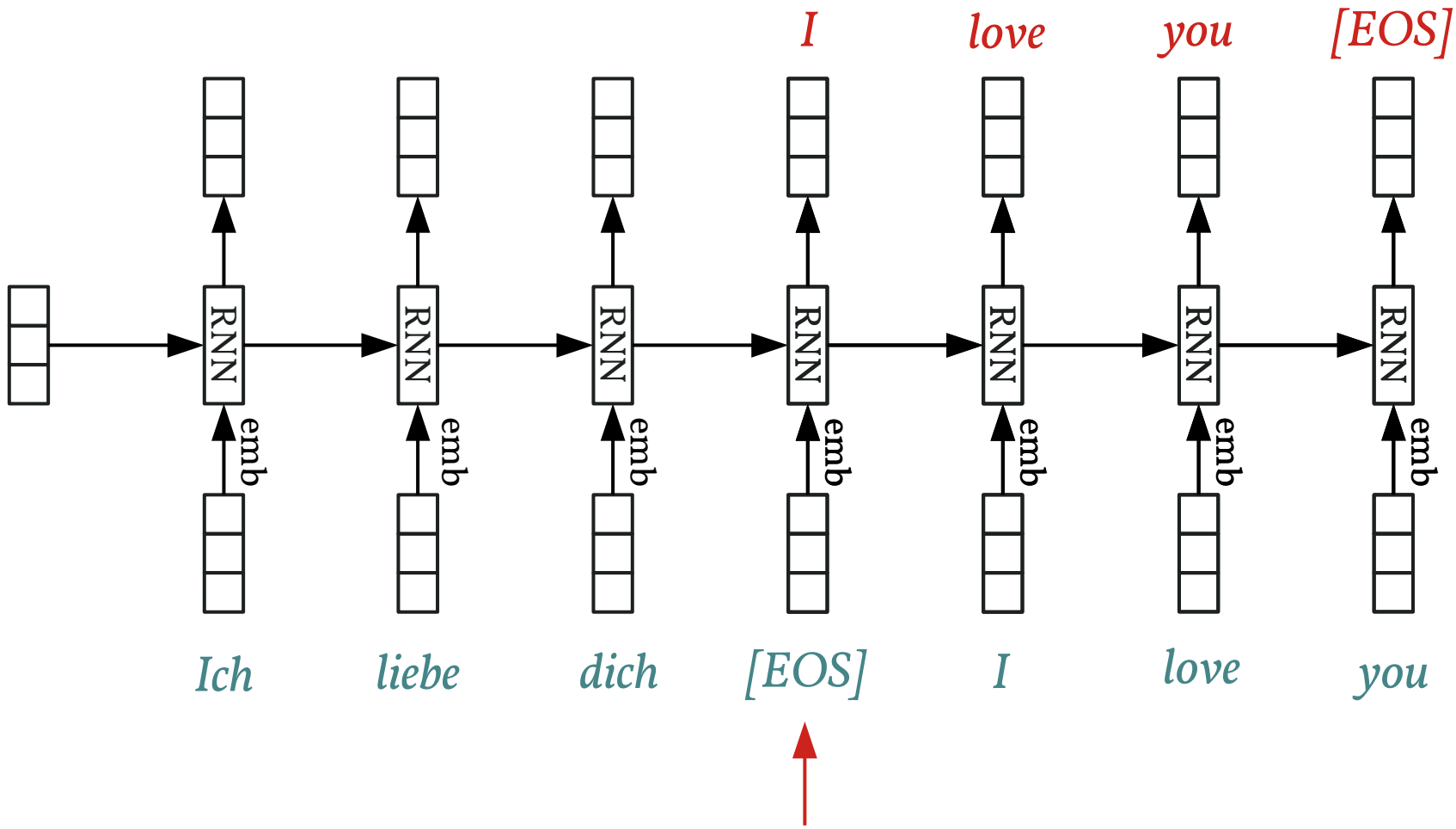
Recurrent Neural Network for Language Modeling



Recurrent Neural Network for Machine Translation

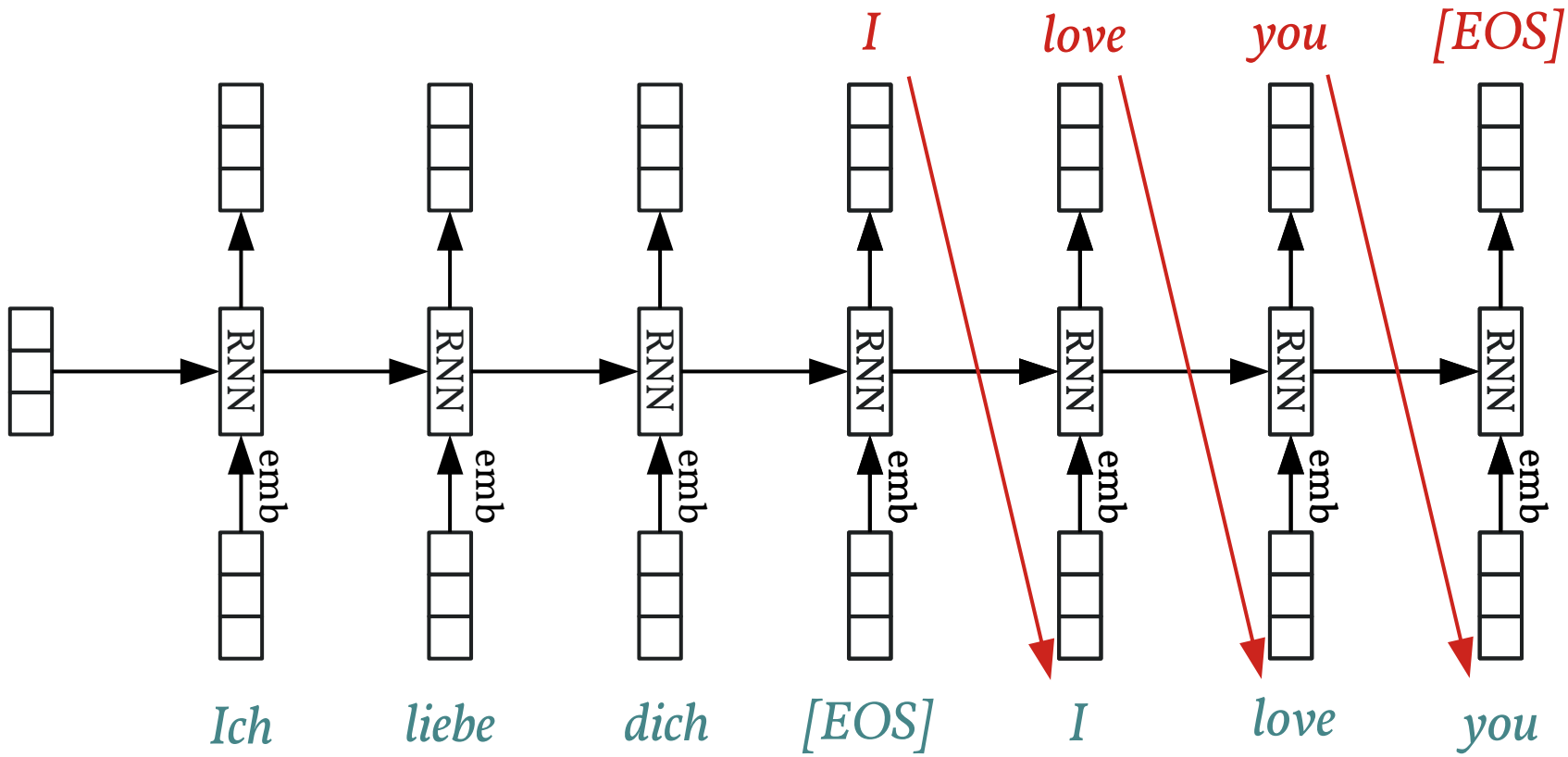


Recurrent Neural Network for Machine Translation



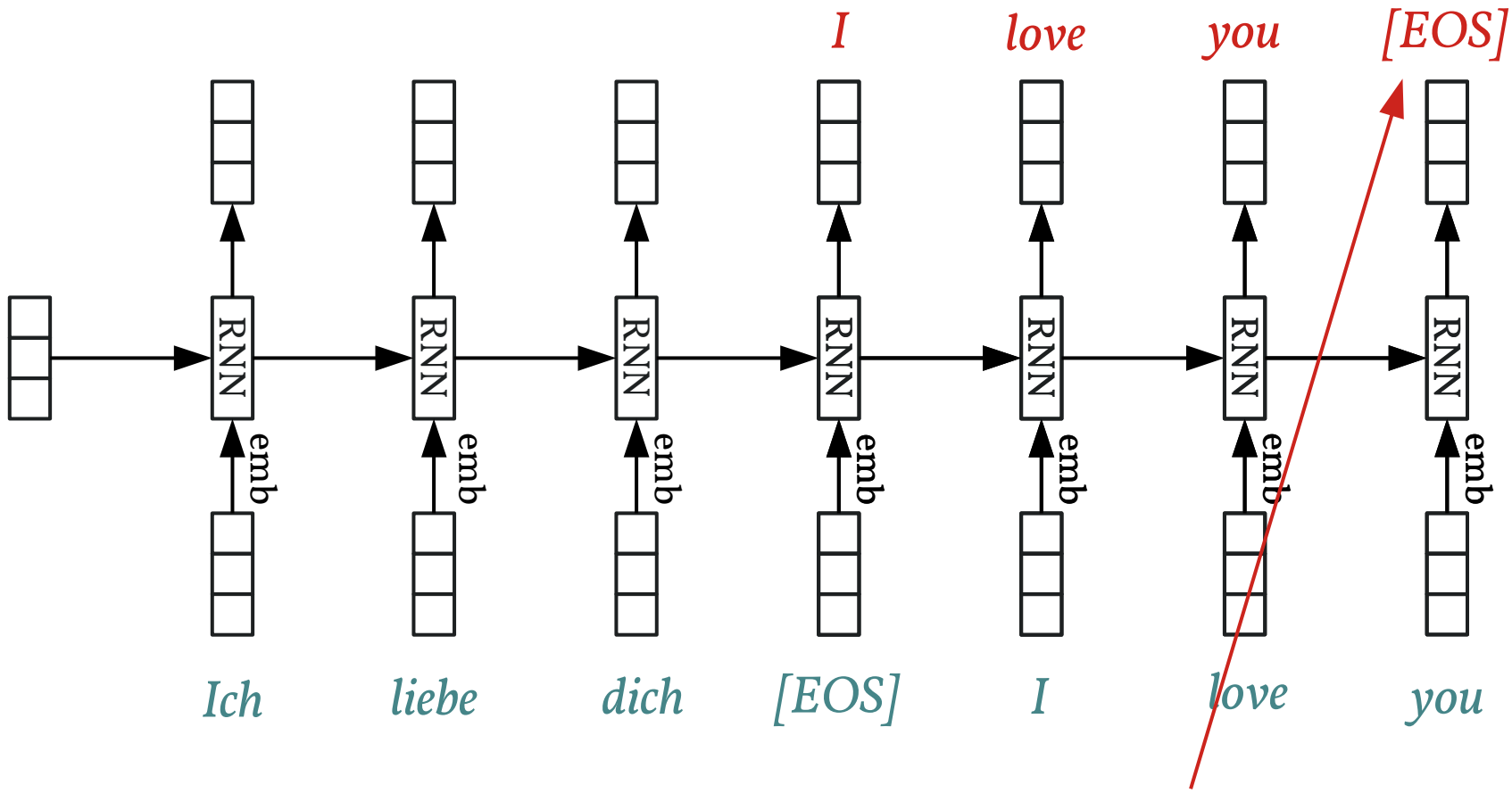
[EOS] token signals end of input

Recurrent Neural Network for Machine Translation



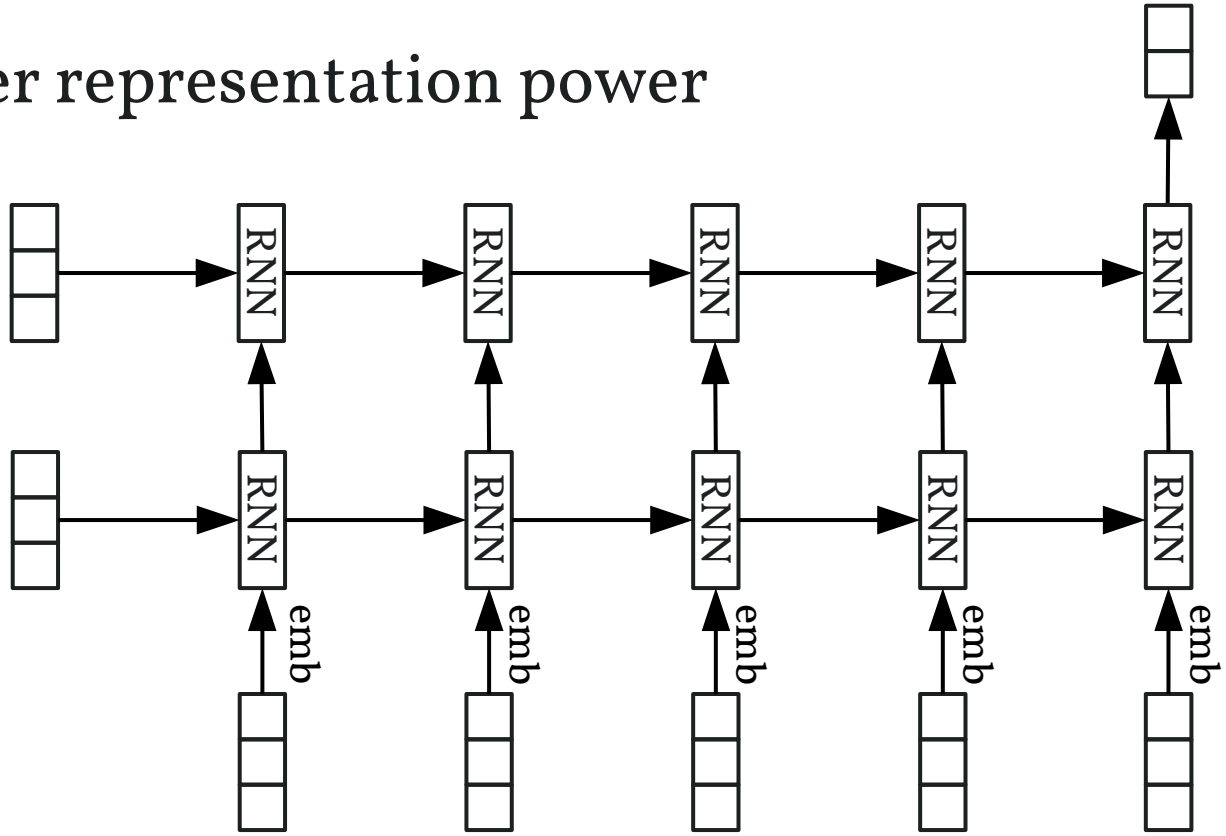
Output tokens are appended to input

Recurrent Neural Network for Machine Translation



Stacked Recurrent Neural Network (Pascanu et al, 2014)

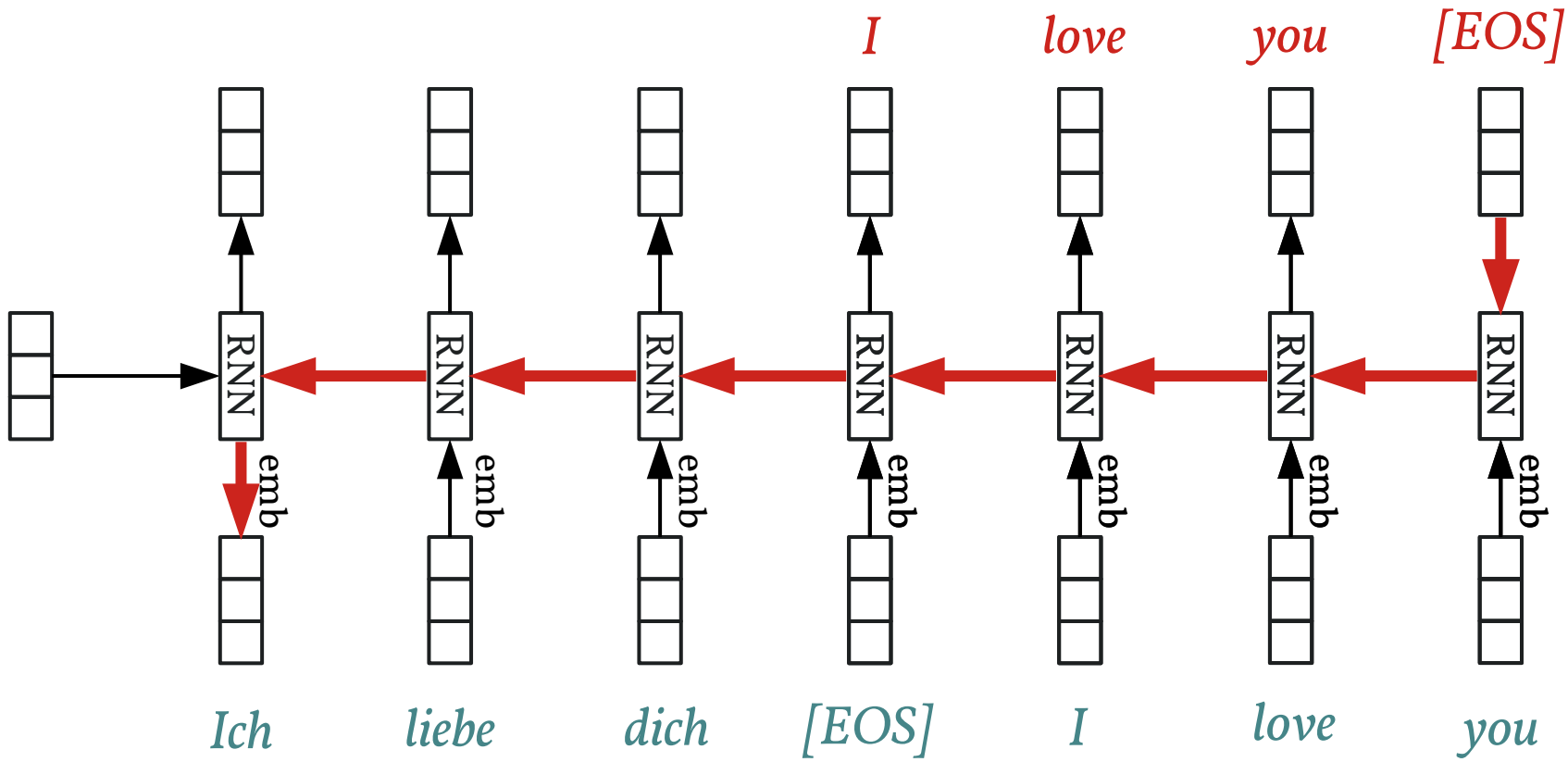
» For higher representation power



Pascanu et al (2014). "How to Construct Deep Recurrent Neural Networks". *ICLR*.

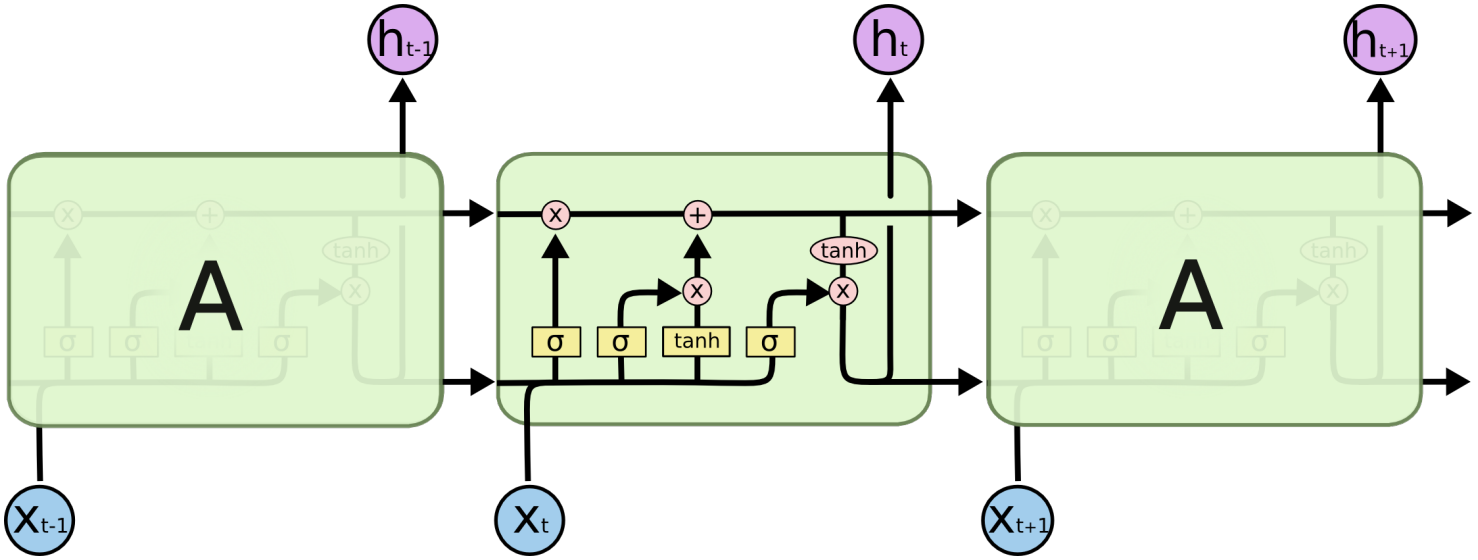
Problem: Long-Range Dependencies

- Very simple example translation
- Already gradient flows through 8 layers
- Easily leads to vanishing/exploding gradients



LSTM: Long Short-Term Memory (Hochreiter & Schmidhuber, 1997; Gers, 2000)

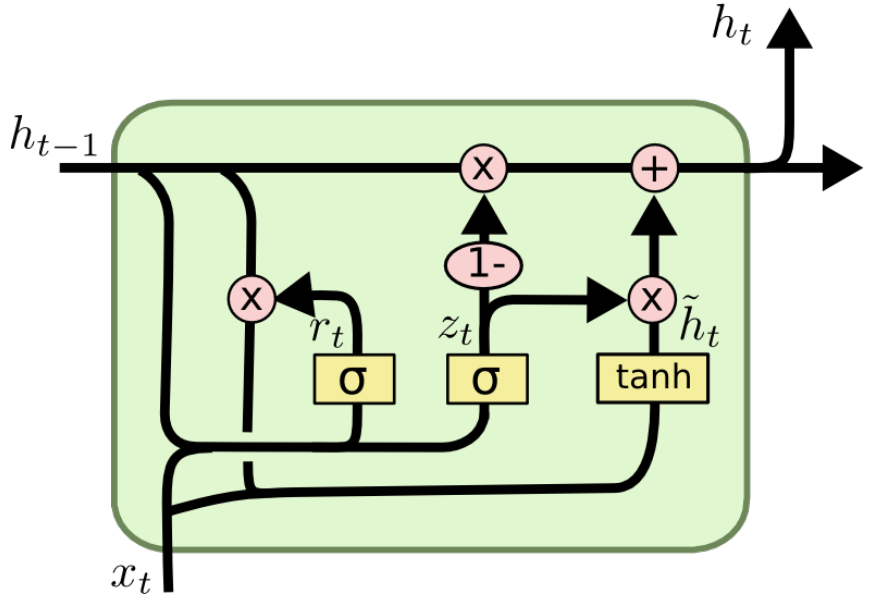
- Usable as drop-in replacement for regular recurrent block
- Avoid vanishing/exploding gradients and thus learn better



Hochreiter & Schmidhuber (1997). "Long Short-Term Memory". *Neural Computation*.
Gers et al (2000). "Learning to Forget: Continual Prediction with LSTM". *Neural Computation*.
Image credit: Olah (2015). "Understanding LSTM Networks".
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU: Gated Recurrent Unit (Cho et al, 2014)

» Same goal as LSTM but simplified implementation

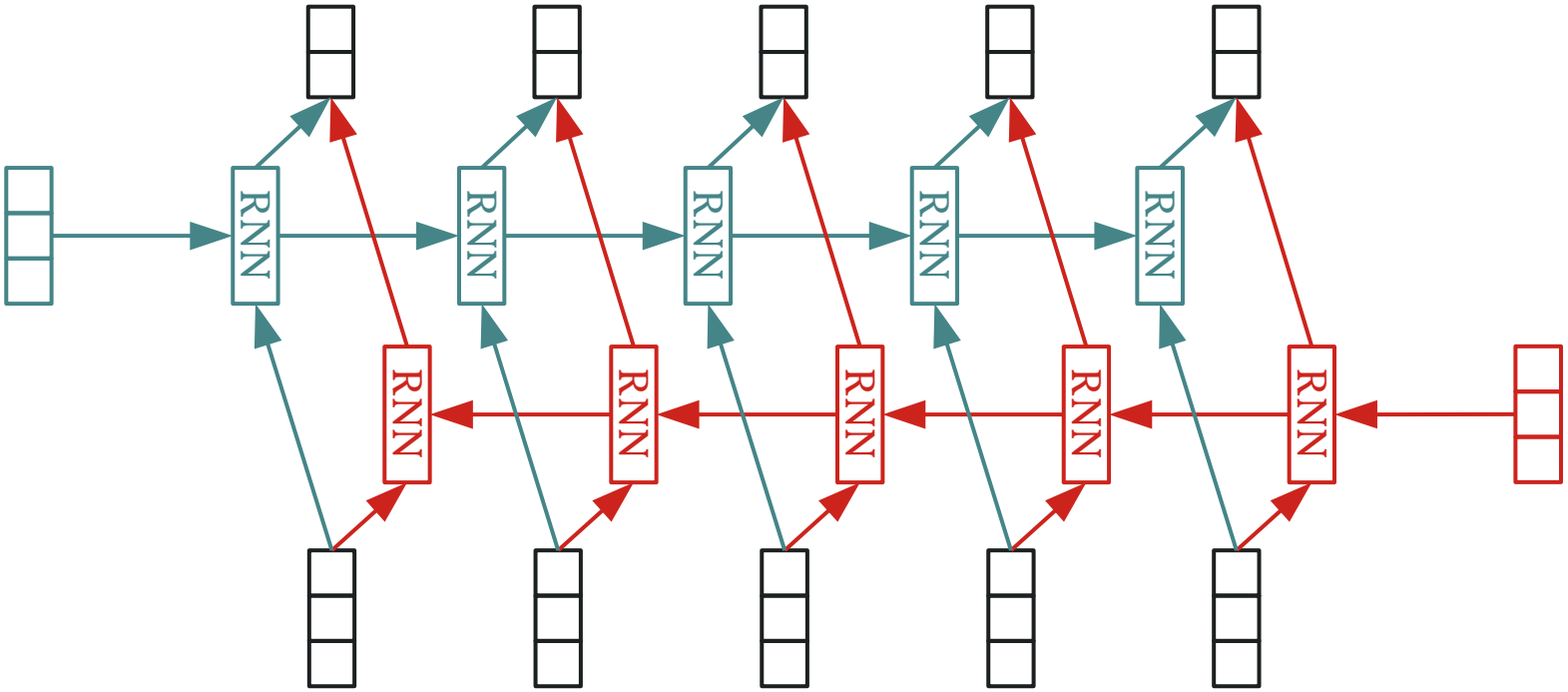


Cho et al (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". *EMNLP*.

Image credit: Olah (2015). "Understanding LSTM Networks". https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional Recurrent Neural Network (Schuster & Paliwal, 1997)

- Output at each token has seen all previous and past tokens
- Still, only two unidirectional networks



Schuster & Paliwal (1997). "Bidirectional Recurrent Neural Networks". *`IEEE Trans. Signal Processing*.

Recurrent Neural Network Analysis

- Theoretically able to consider **global backwards context**
  - Global back- and forward context for bidirectional RNN
- Recurrent layers are hard to train because of vanishing/exploding gradients due to **long gradient paths**
- LSTMs and GRUs have comparable performance but both are much better than the vanilla recurrent layer
- All recurrent layer variants implement **sequential** operations and are thus not parallelizable

2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)

Input

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$*$

Kernel

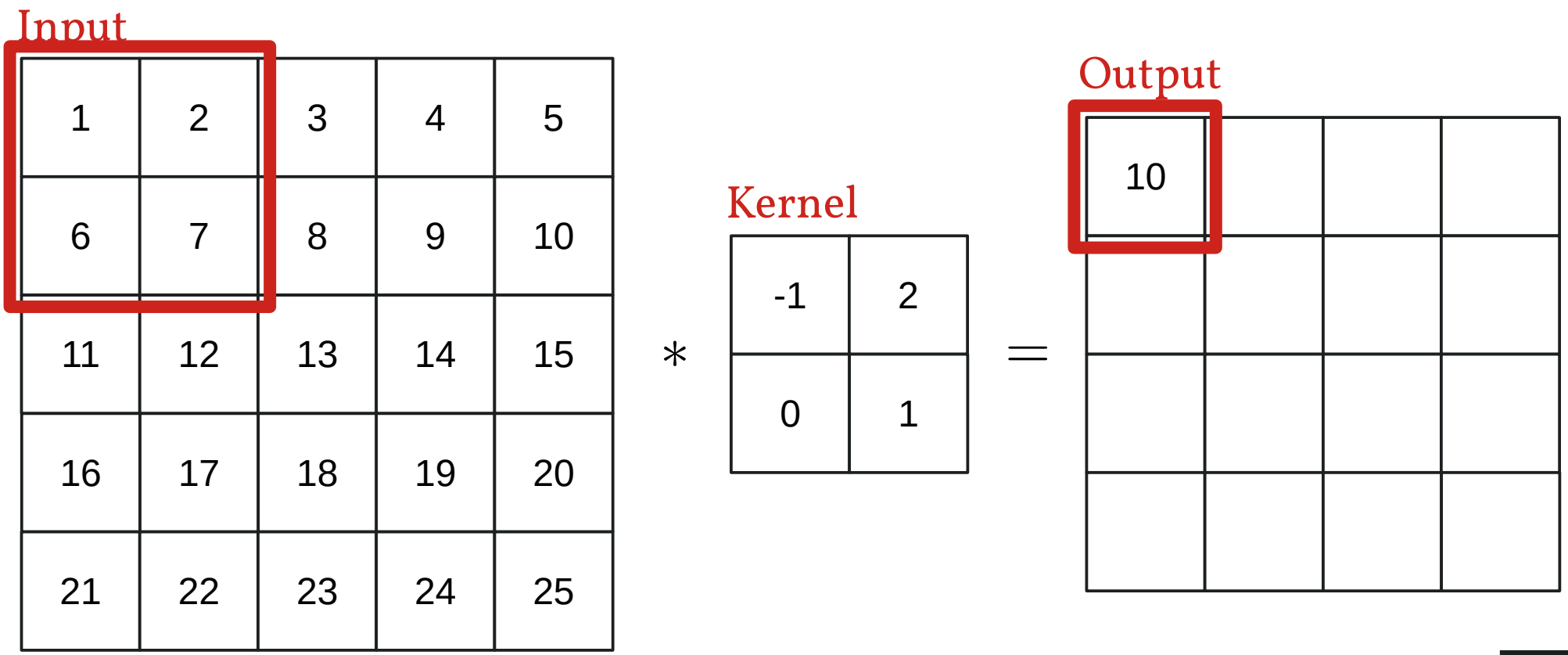
-1	2
0	1

$=$

Output

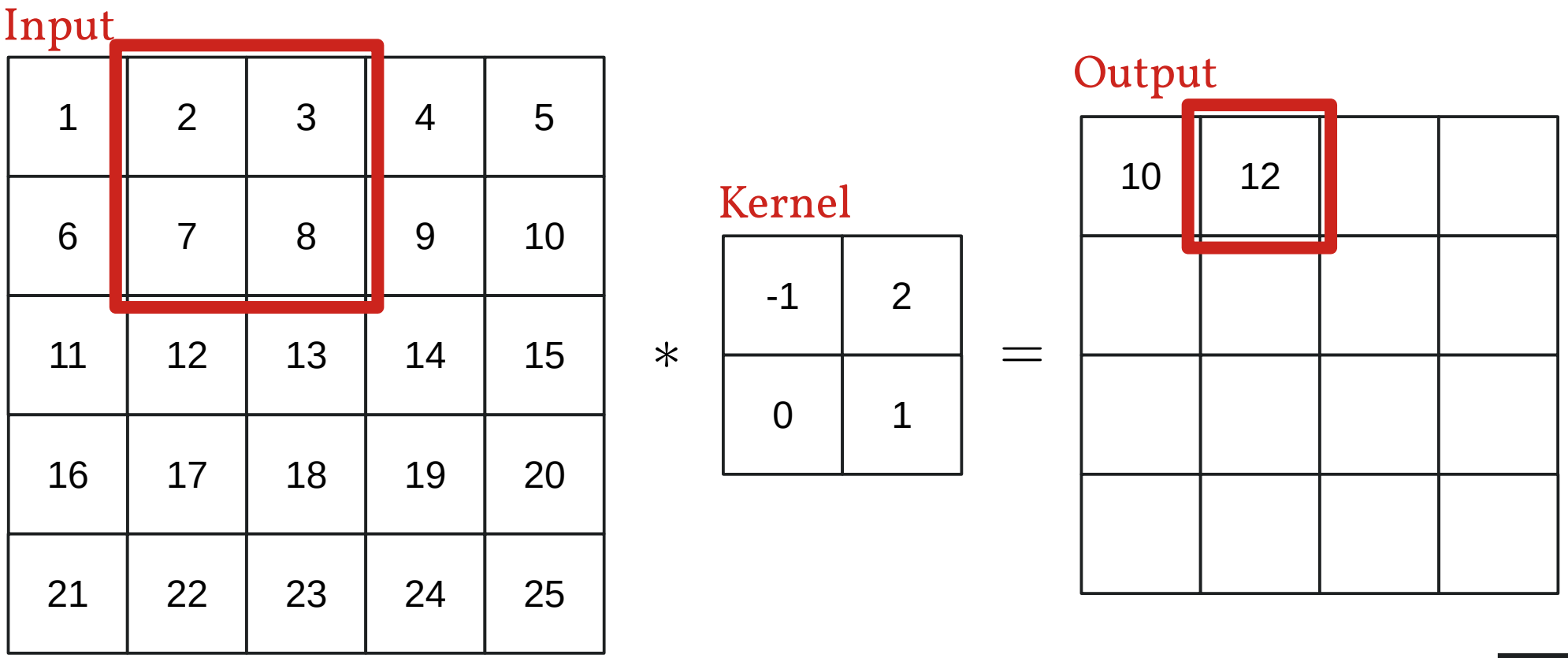
2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)



2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)



2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)

Input

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$*$

Kernel

-1	2
0	1

$=$

Output

10	12	14	

2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)

Input

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$*$

Kernel

-1	2
0	1

$=$

Output

10	12	14	16
20	22	24	26
30	32	34	36
40	42	44	46

2D-Convolution

- Input: 2D input matrix and 2D kernel matrix
- Output 2D matrix
- Can detect patterns in 2D images (e.g. edges)

Input

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

$*$

Kernel

-1	2
0	1

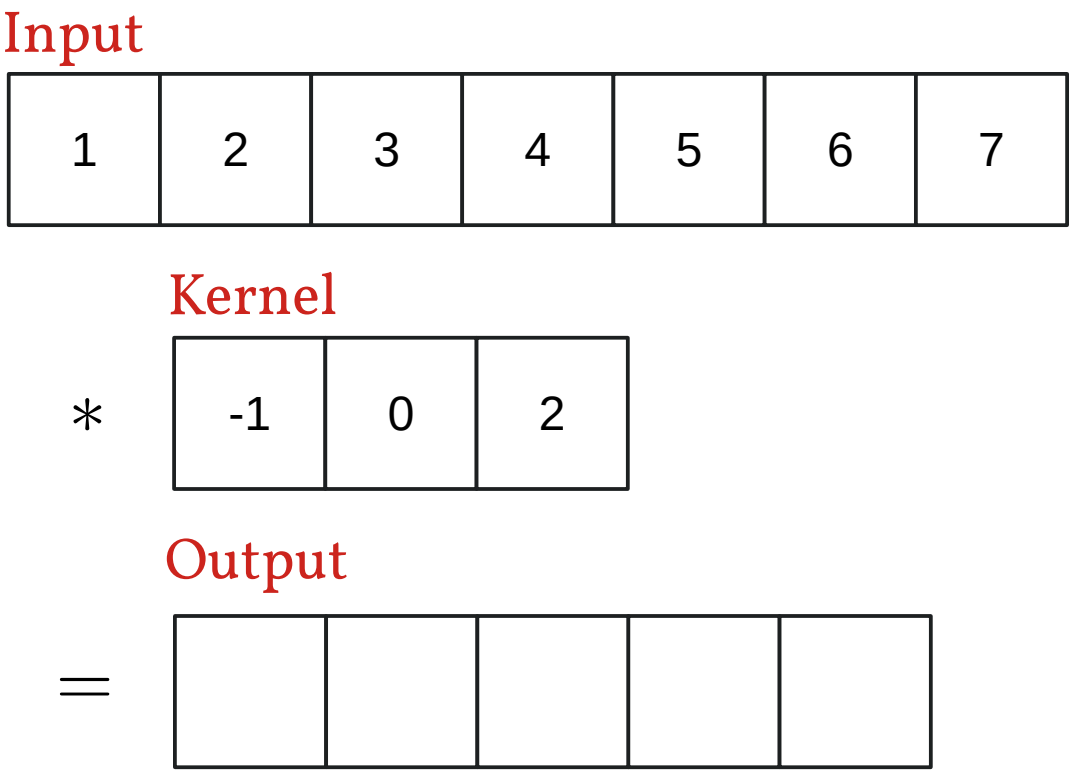
$=$

Output

10	12	14	16
20	22	24	26
30	32	34	36
40	42	44	46

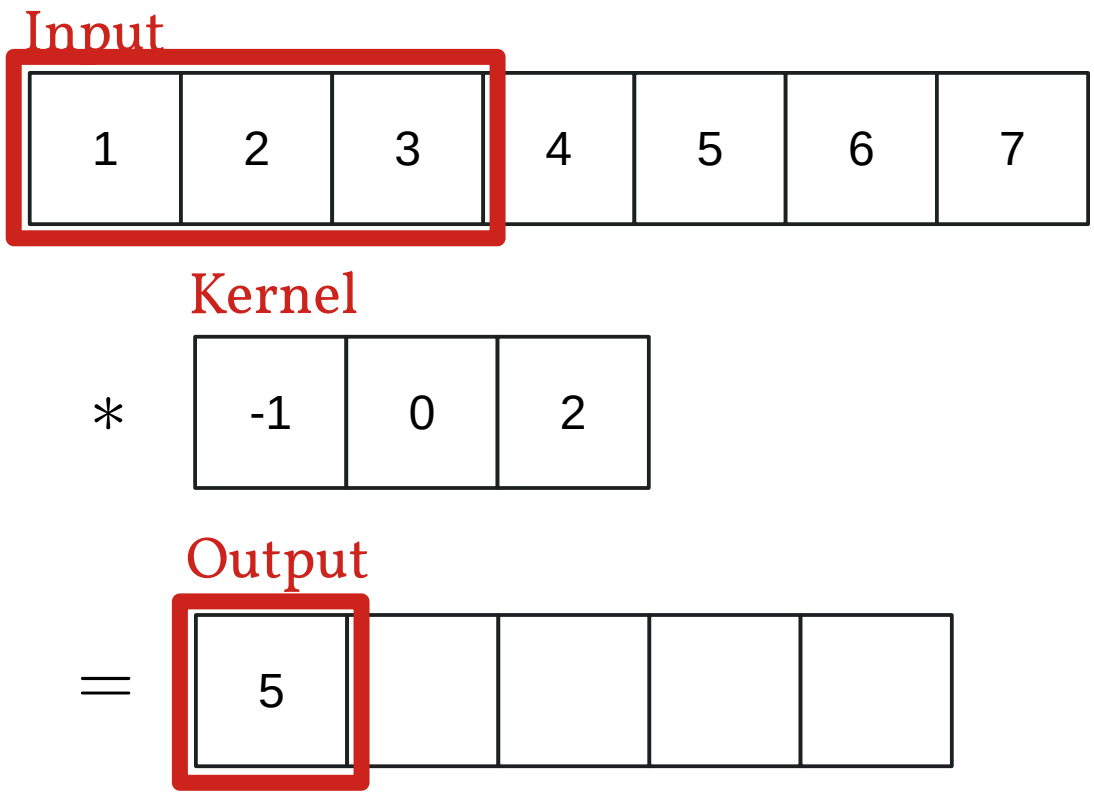
1D-Convolution

» Same idea as 2D-convolutions



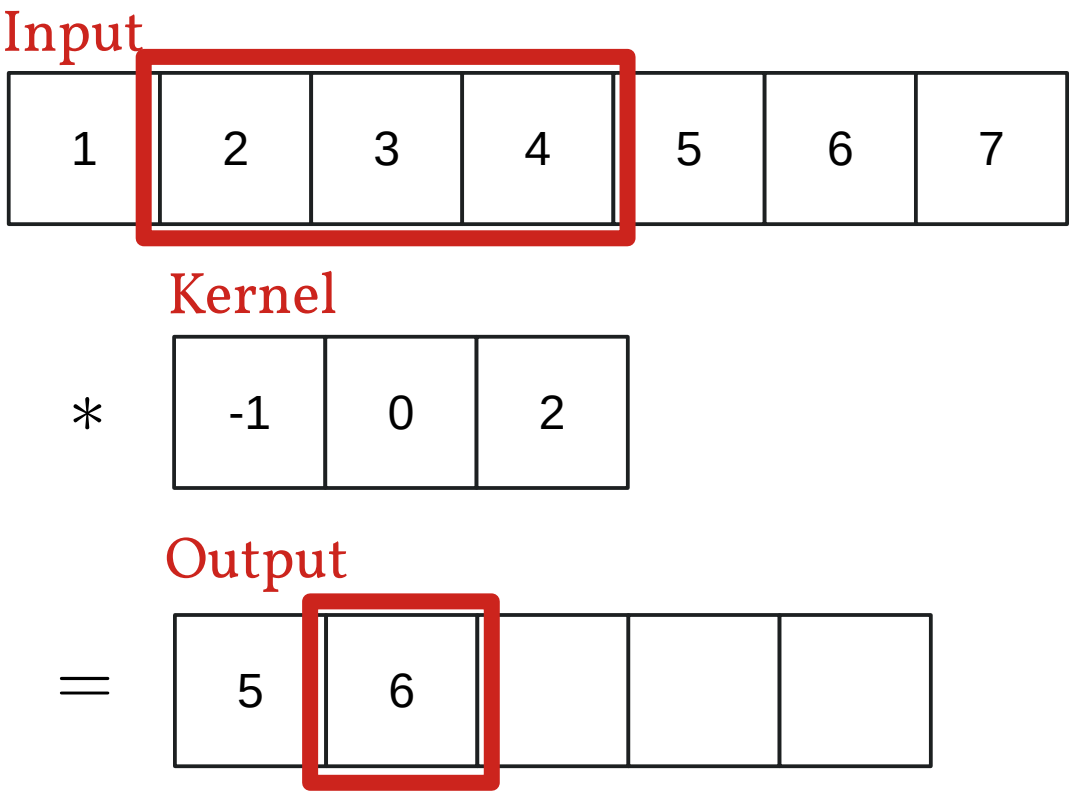
1D-Convolution

» Same idea as 2D-convolutions



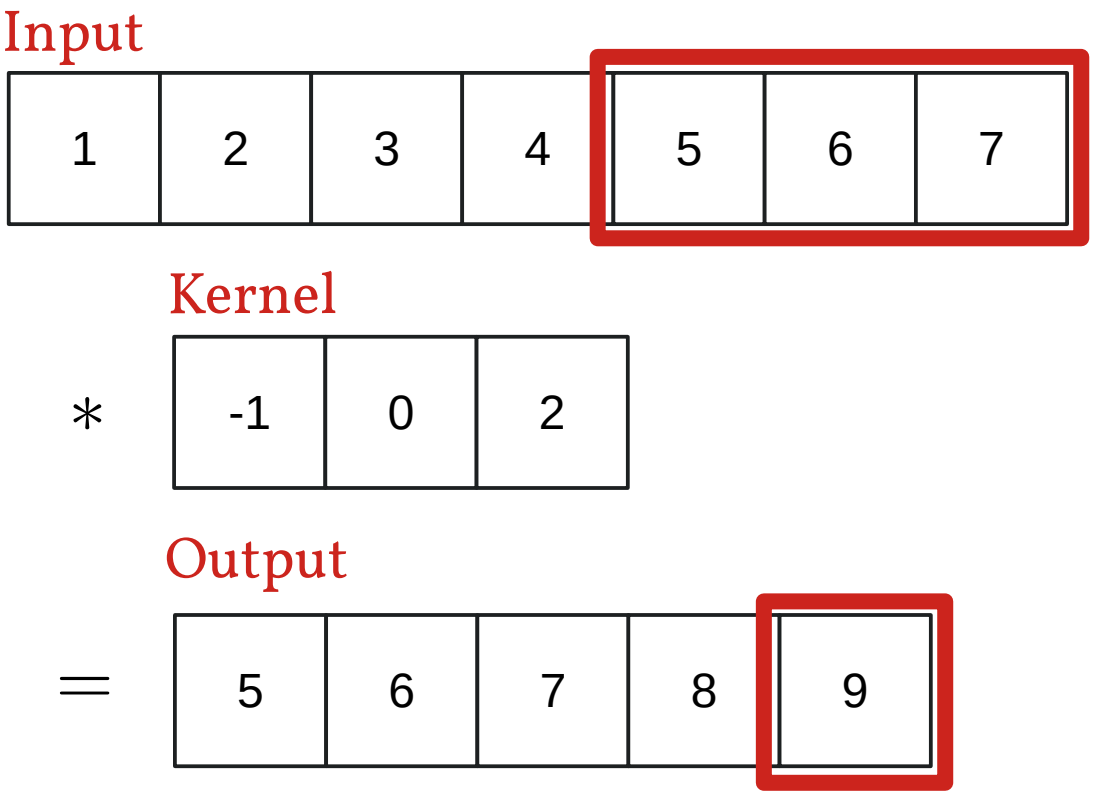
1D-Convolution

» Same idea as 2D-convolutions



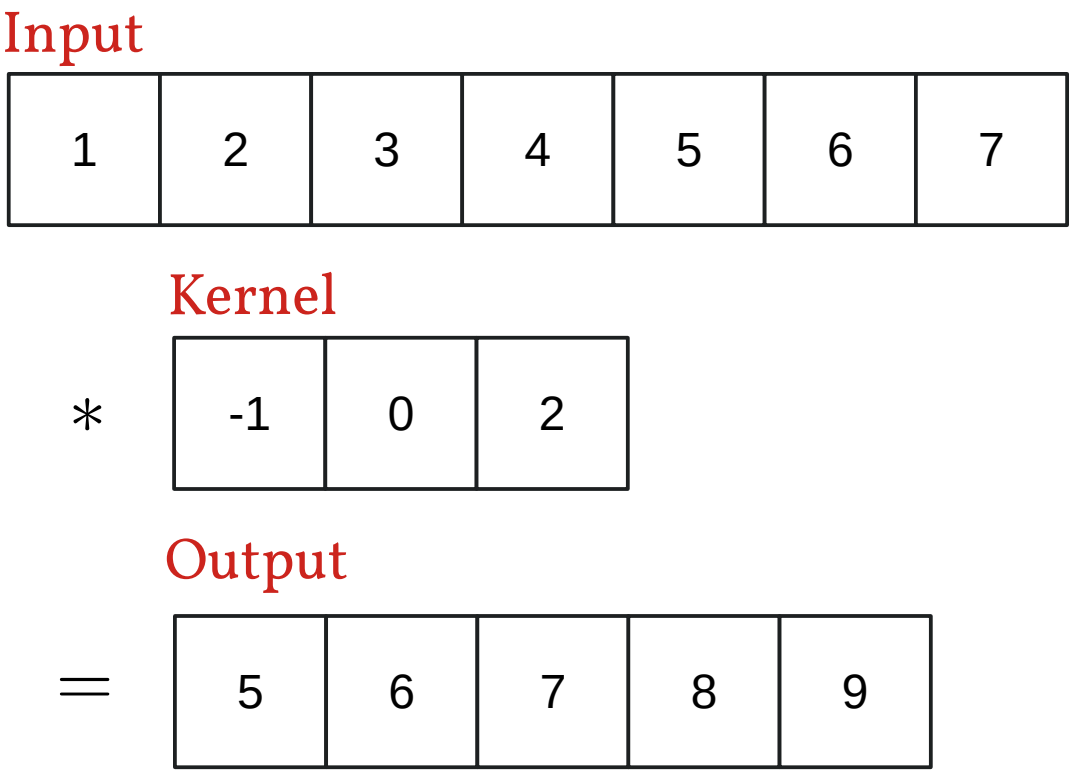
1D-Convolution

» Same idea as 2D-convolutions



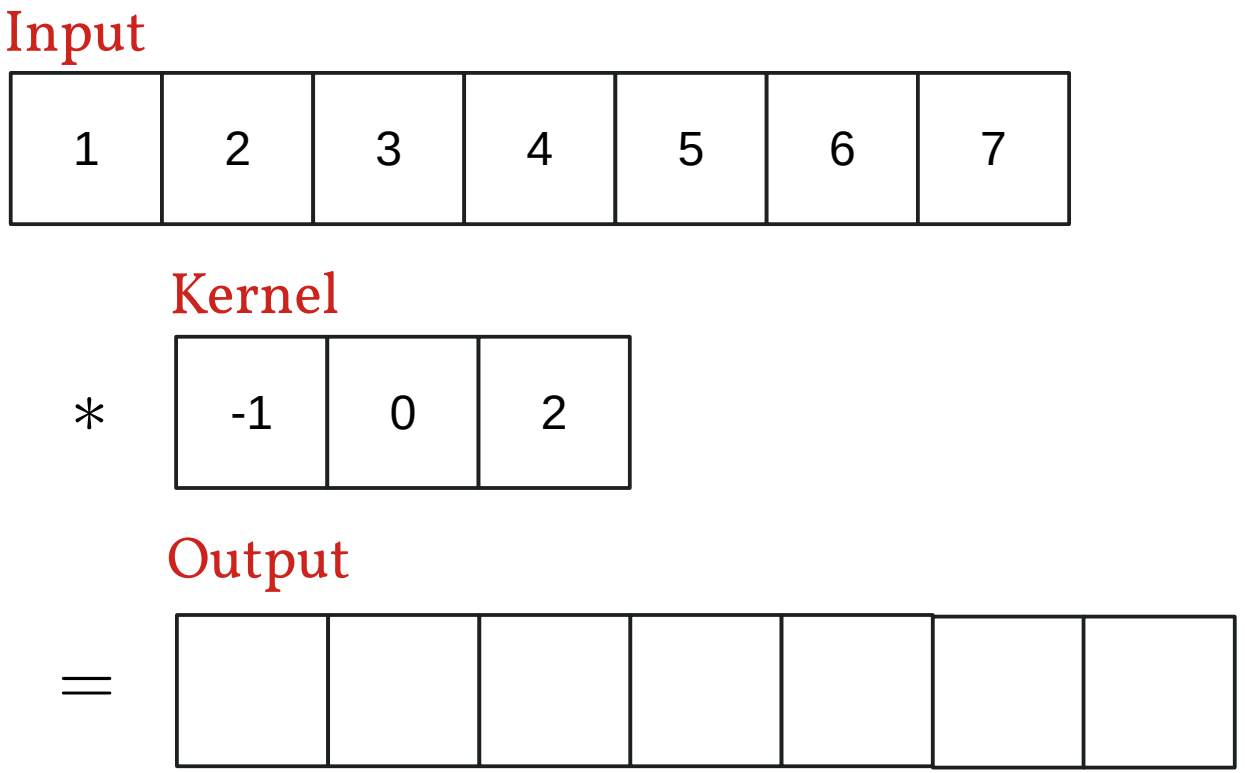
1D-Convolution

» Same idea as 2D-convolutions



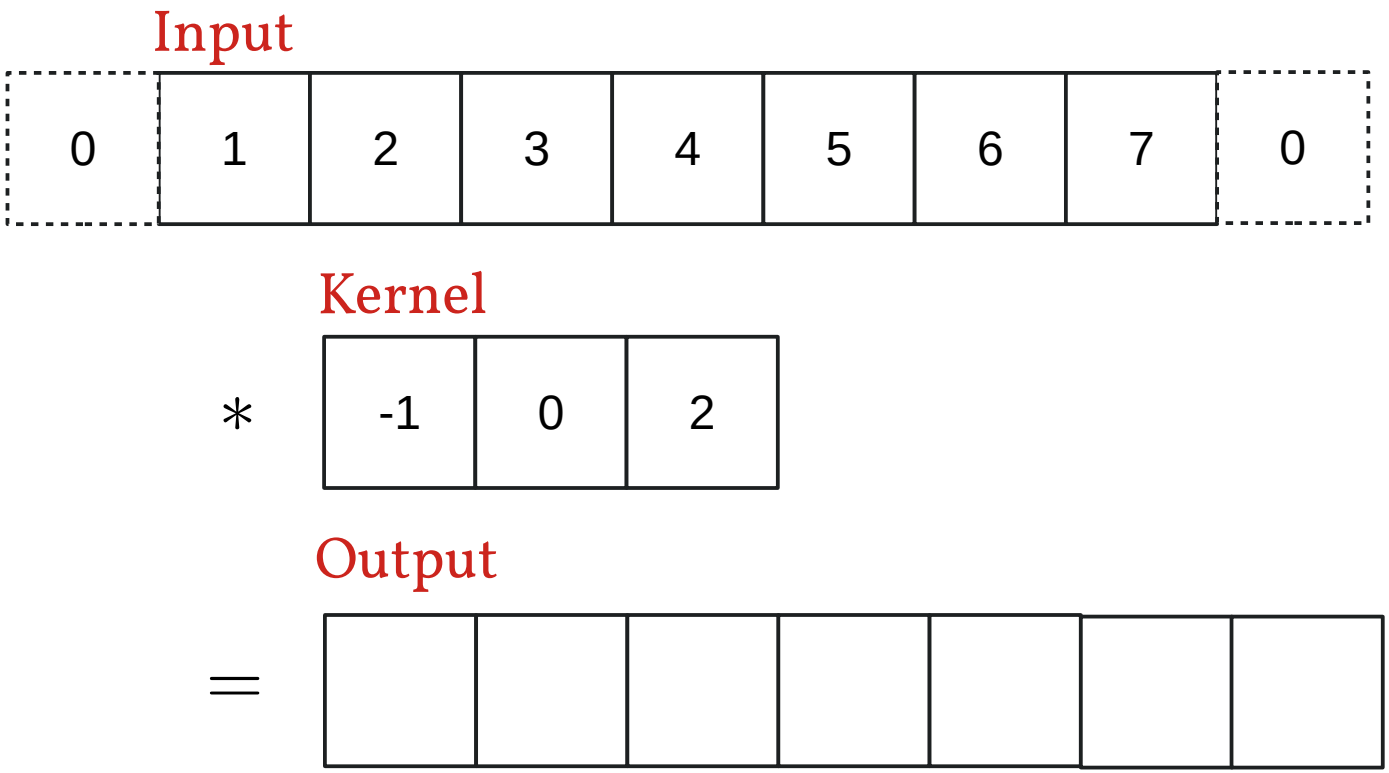
1D-Convolution: Padding

» Problem: we want to have same output dimension as input



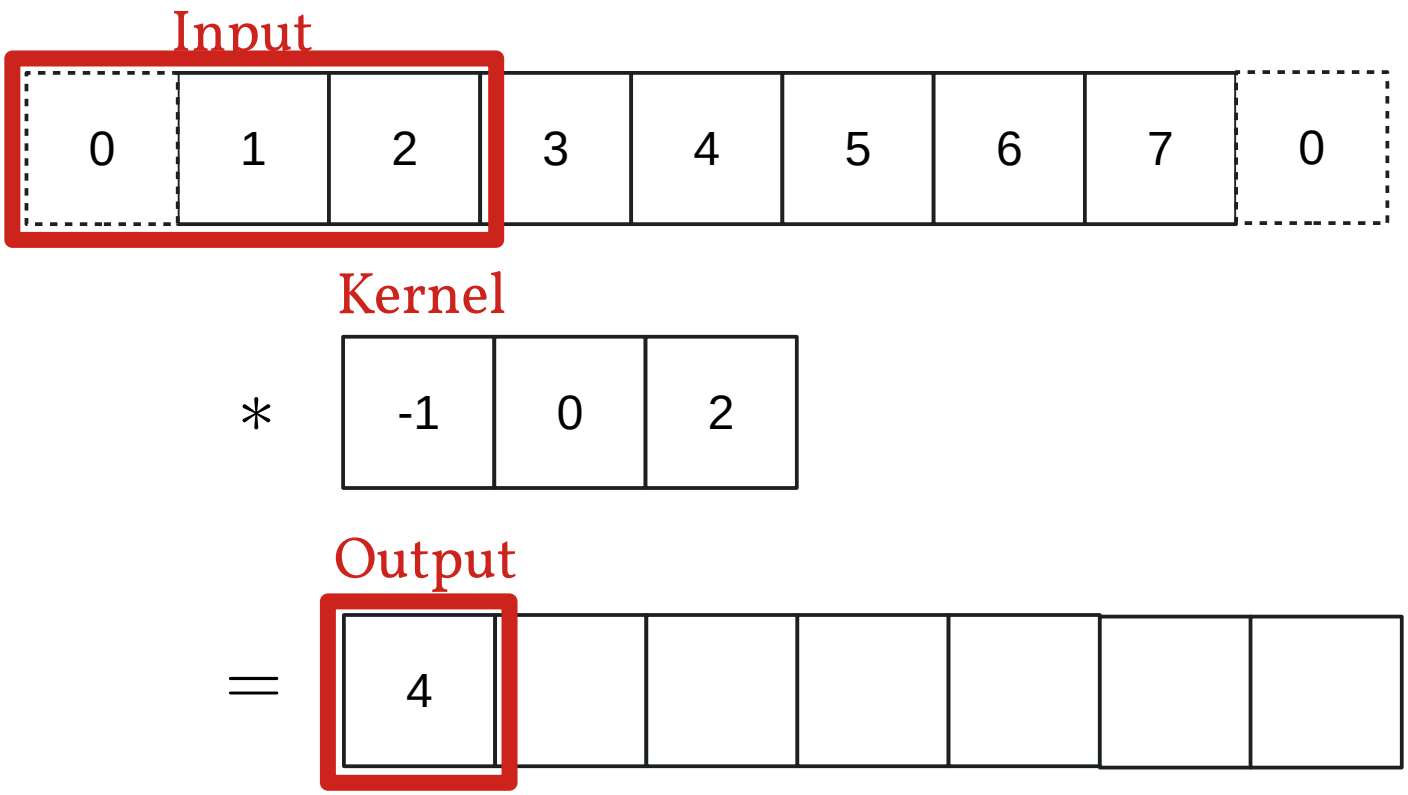
1D-Convolution: Padding

- Problem: we want to have same output dimension as input
- Solution: add zero padding around border



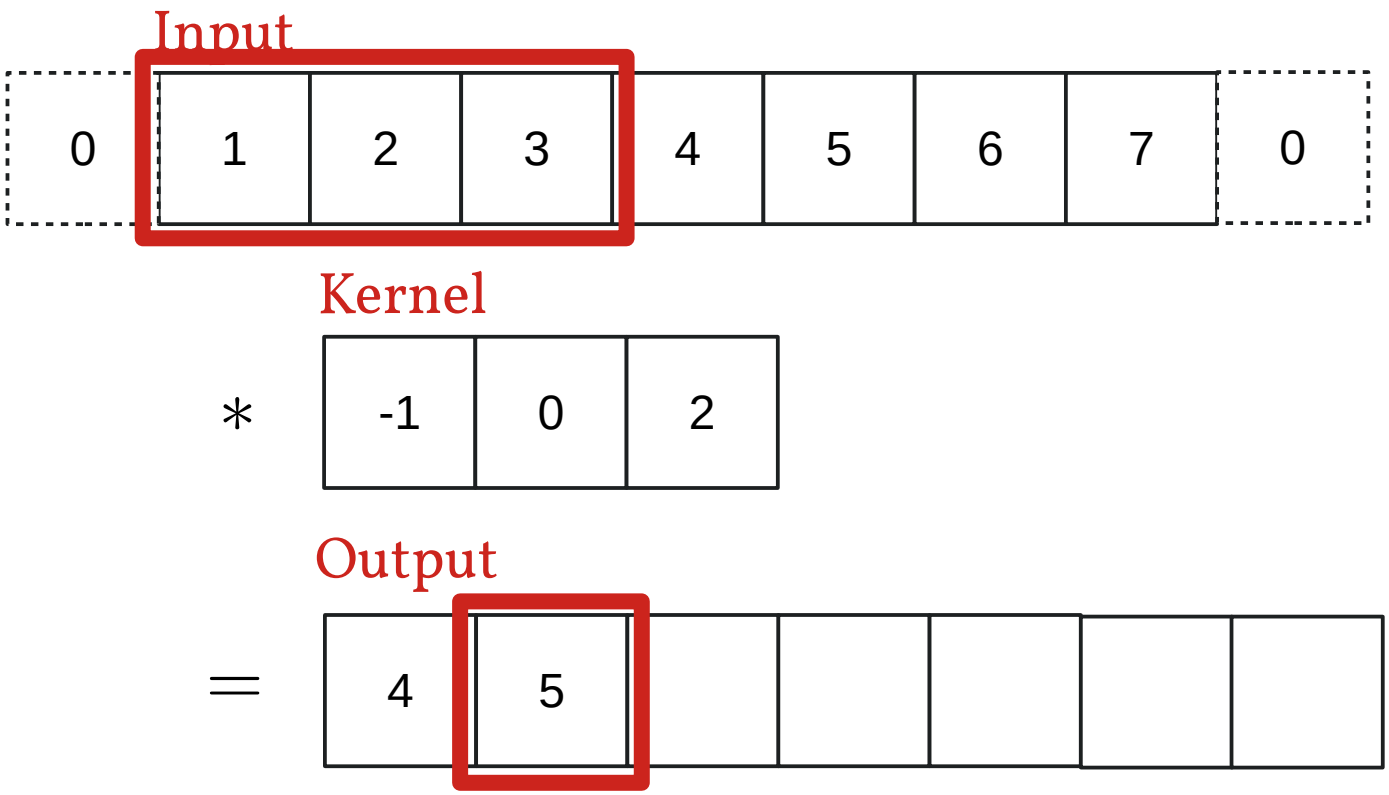
1D-Convolution: Padding

- Problem: we want to have same output dimension as input
- Solution: add zero padding around border



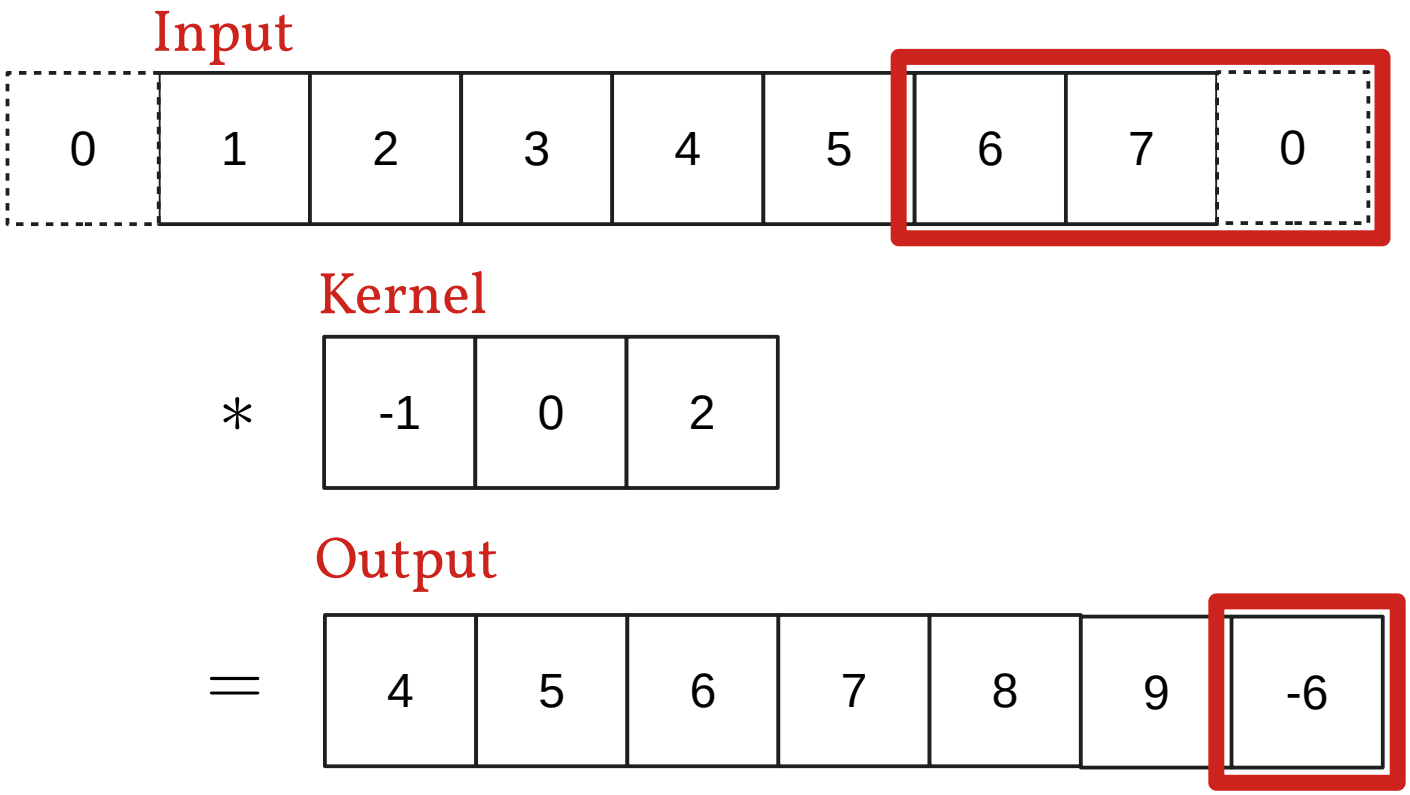
1D-Convolution: Padding

- Problem: we want to have same output dimension as input
- Solution: add zero padding around border



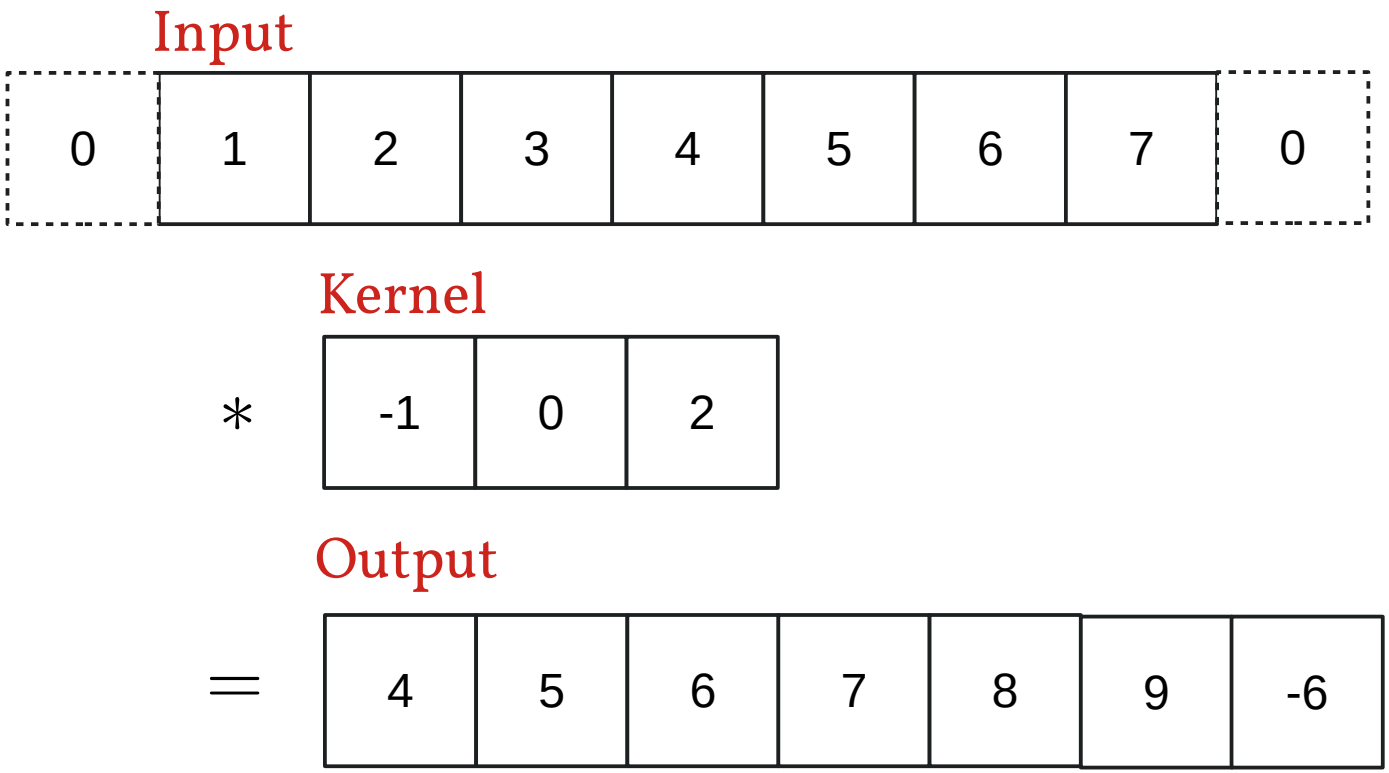
1D-Convolution: Padding

- Problem: we want to have same output dimension as input
- Solution: add zero padding around border



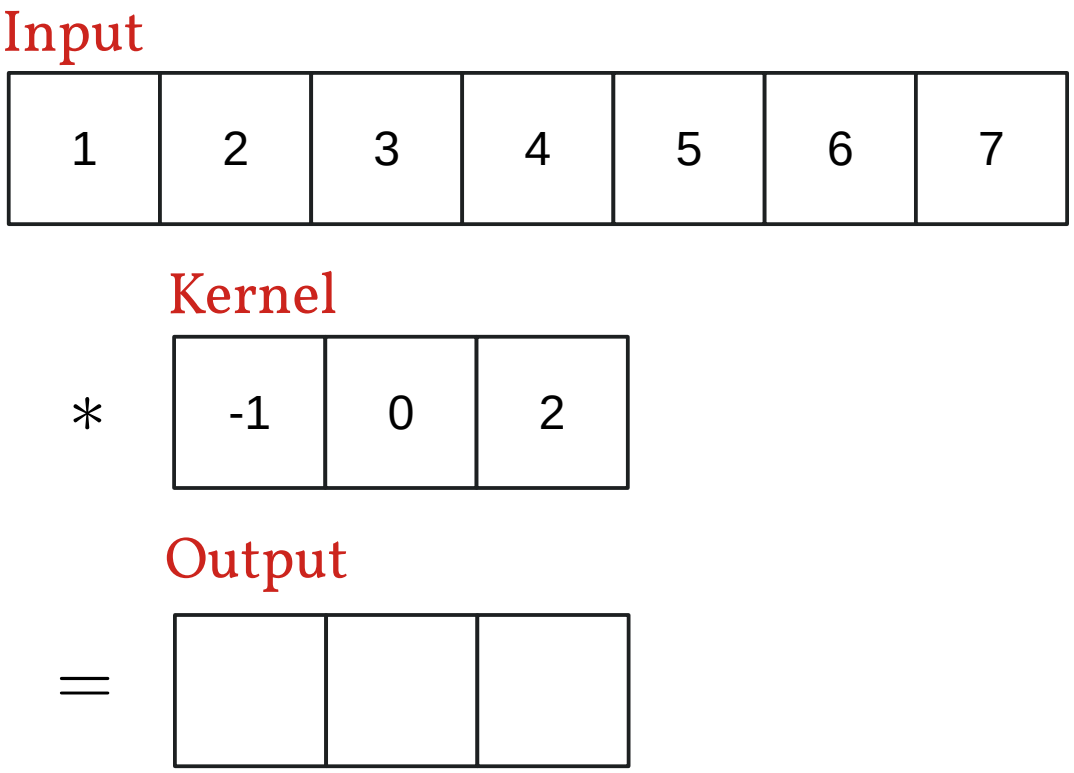
1D-Convolution: Padding

- Problem: we want to have same output dimension as input
- Solution: add zero padding around border



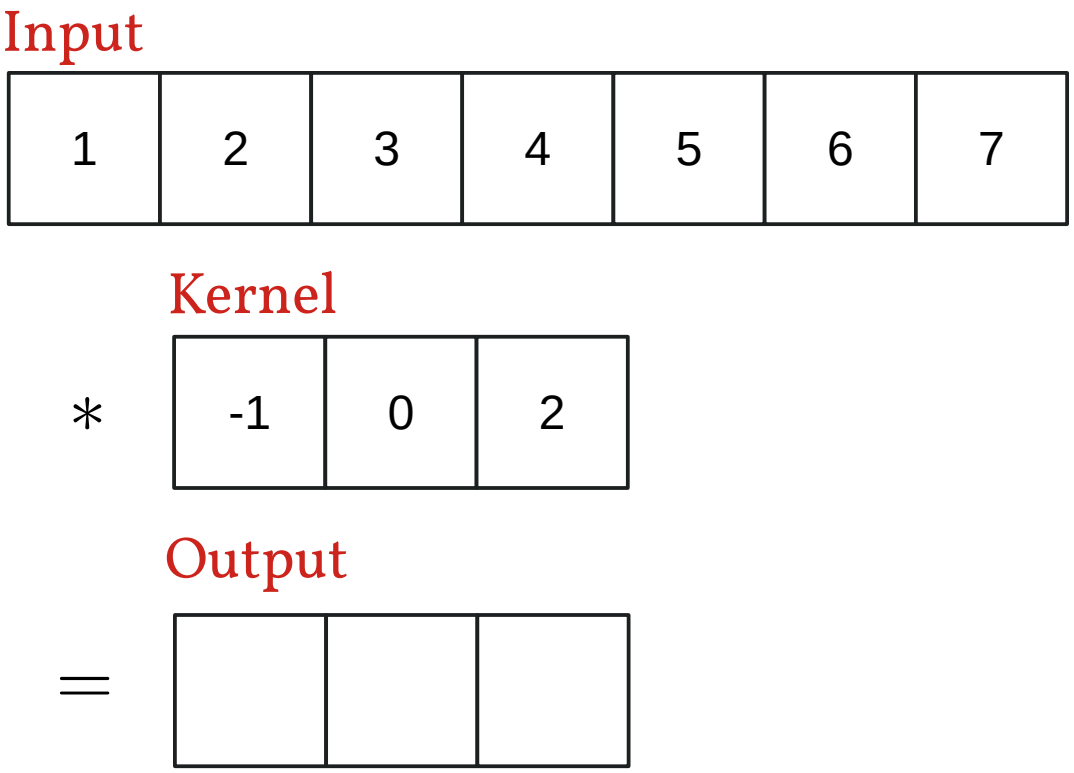
1D-Convolution: Stride

» Problem: we don't want to have overlapping receptive fields



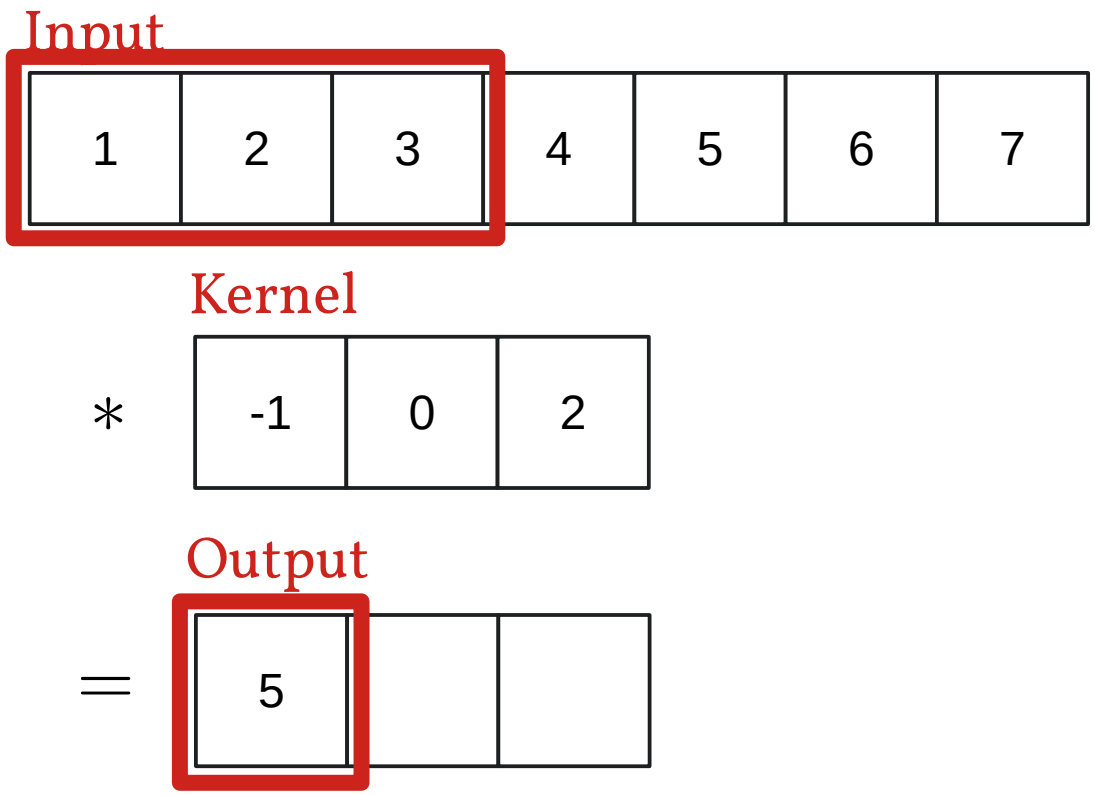
1D-Convolution: Stride

- Problem: we don't want to have overlapping receptive fields
- Solution: advance receptive field more than one cell



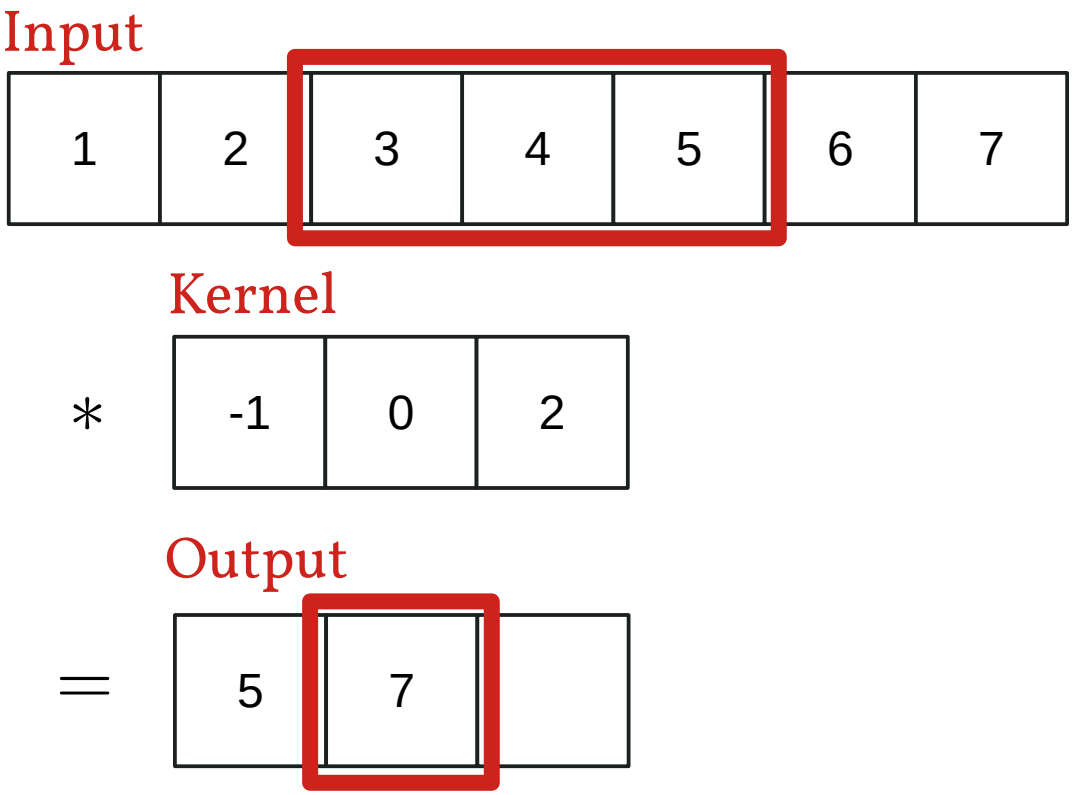
1D-Convolution: Stride

- Problem: we don't want to have overlapping receptive fields
- Solution: advance receptive field more than one cell



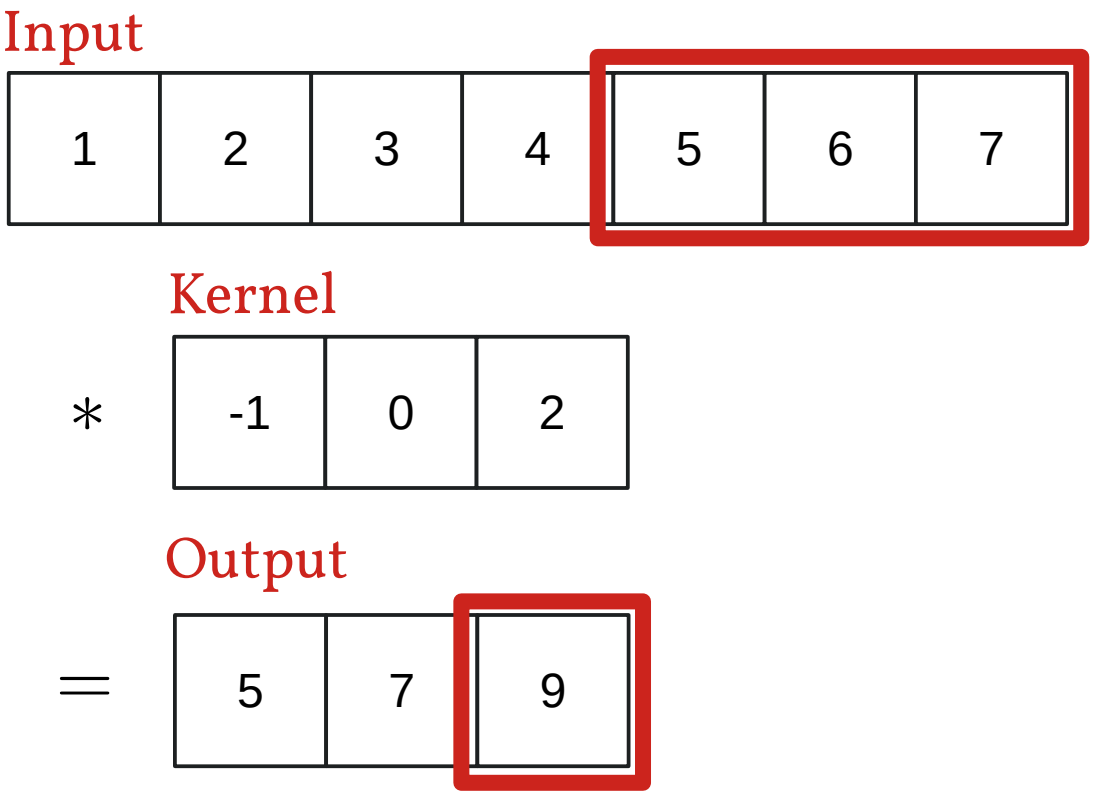
1D-Convolution: Stride

- Problem: we don't want to have overlapping receptive fields
- Solution: advance receptive field more than one cell



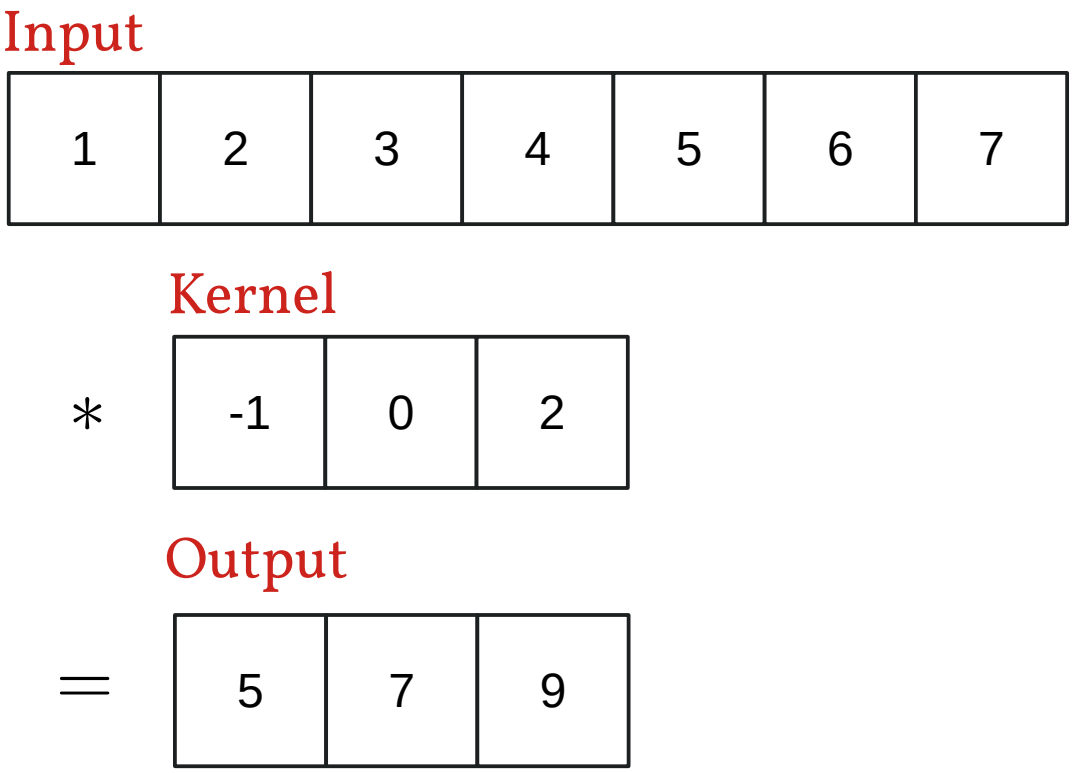
1D-Convolution: Stride

- Problem: we don't want to have overlapping receptive fields
- Solution: advance receptive field more than one cell



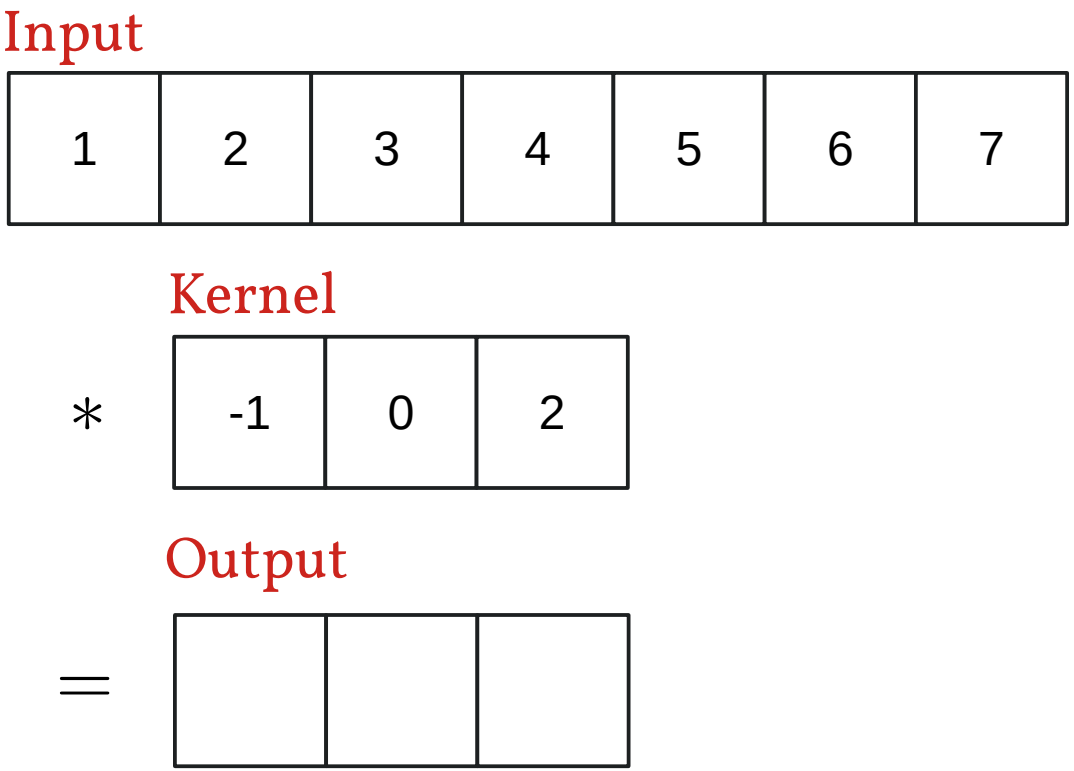
1D-Convolution: Stride

- Problem: we don't want to have overlapping receptive fields
- Solution: advance receptive field more than one cell



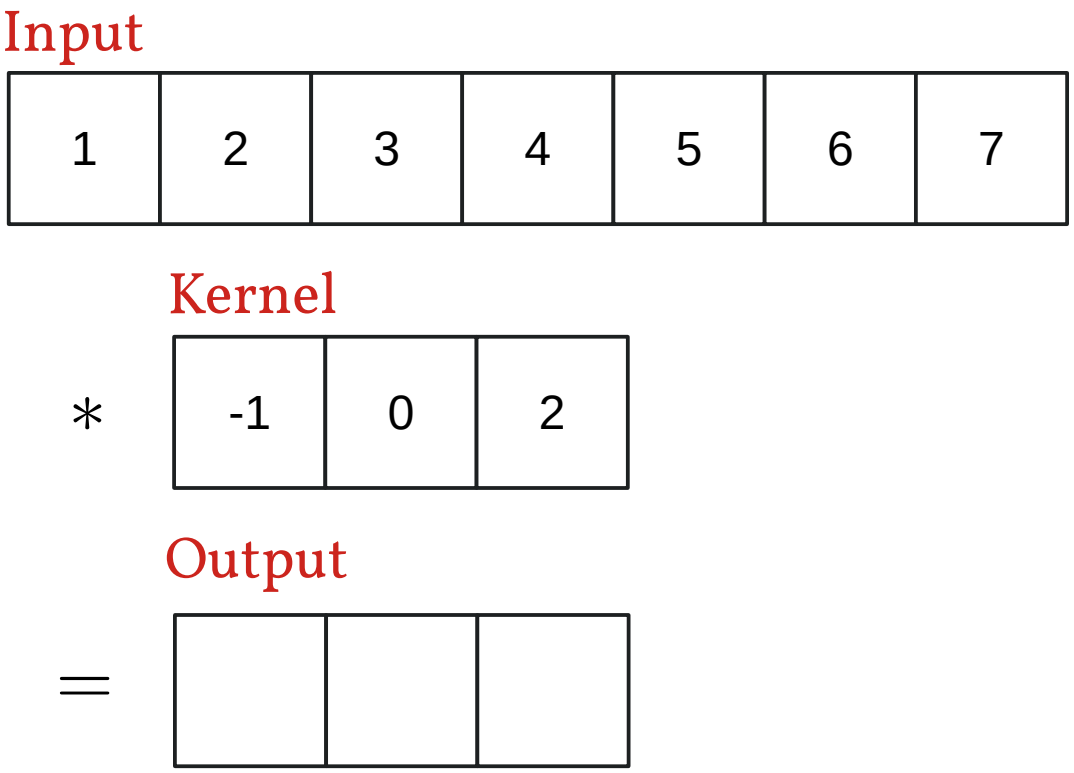
1D-Convolution: Dilation

» Problem: we want to recognize larger patterns



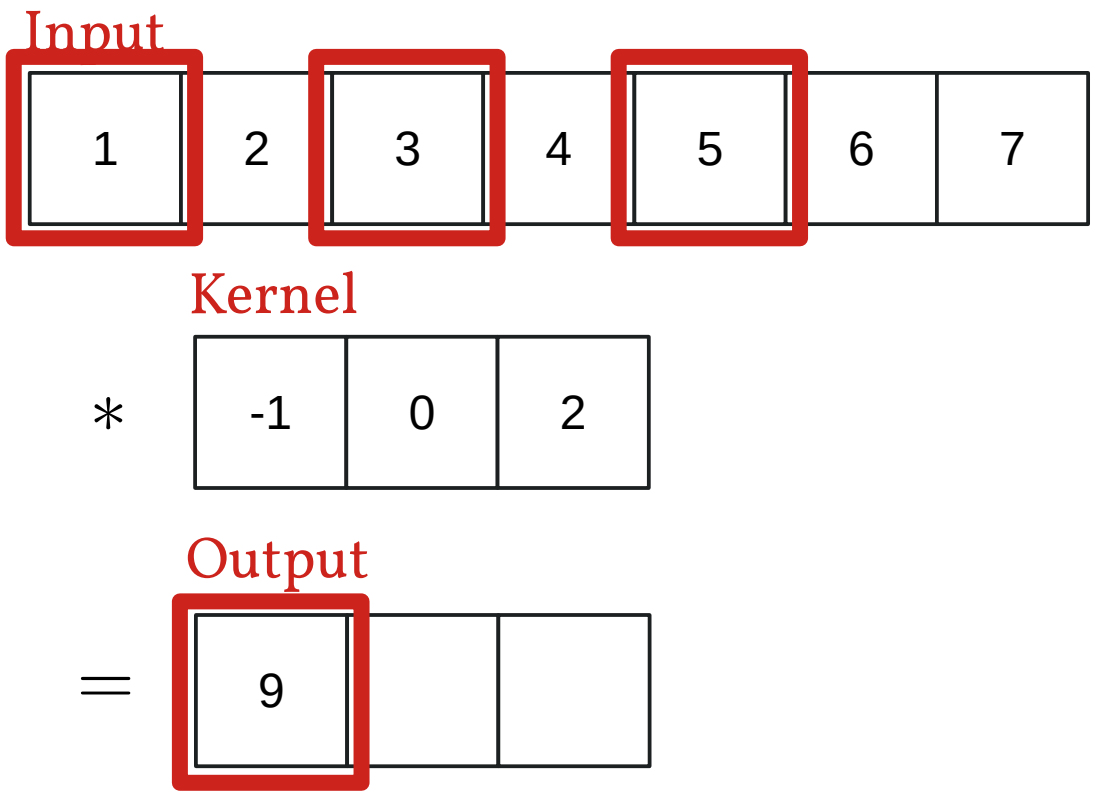
1D-Convolution: Dilation

- Problem: we want to recognize larger patterns
- Solution: dilate receptive field



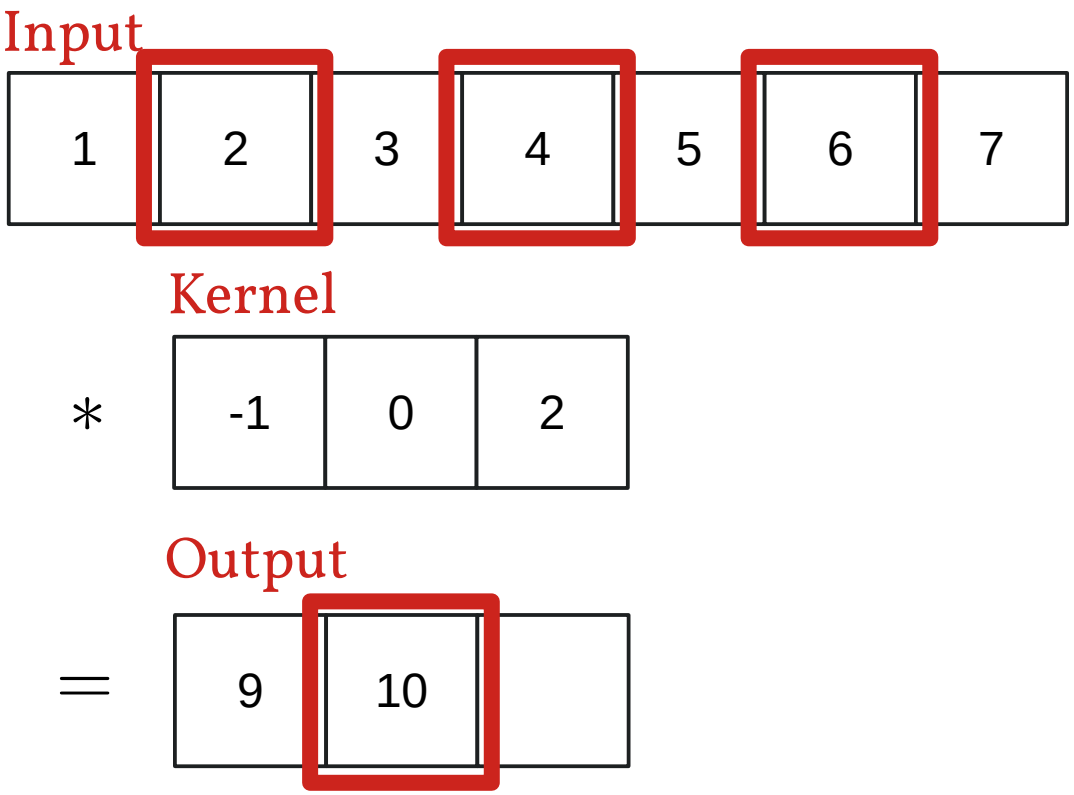
1D-Convolution: Dilation

- Problem: we want to recognize larger patterns
- Solution: dilate receptive field



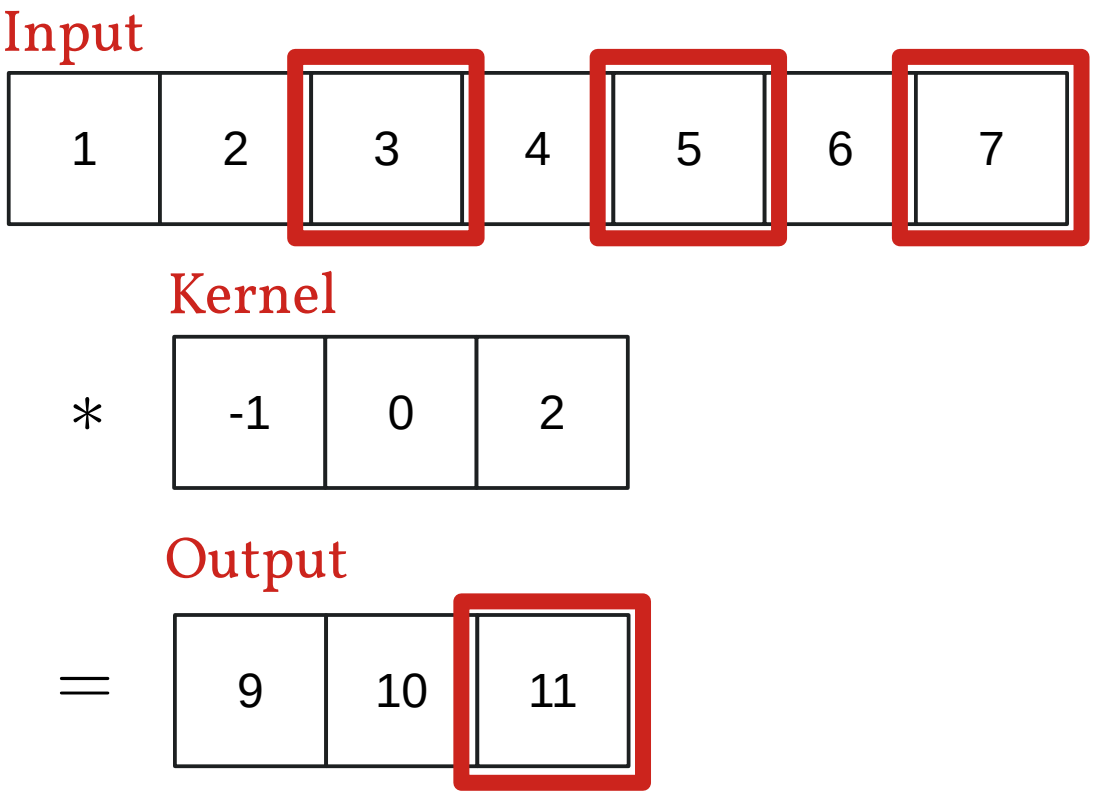
1D-Convolution: Dilation

- Problem: we want to recognize larger patterns
- Solution: dilate receptive field



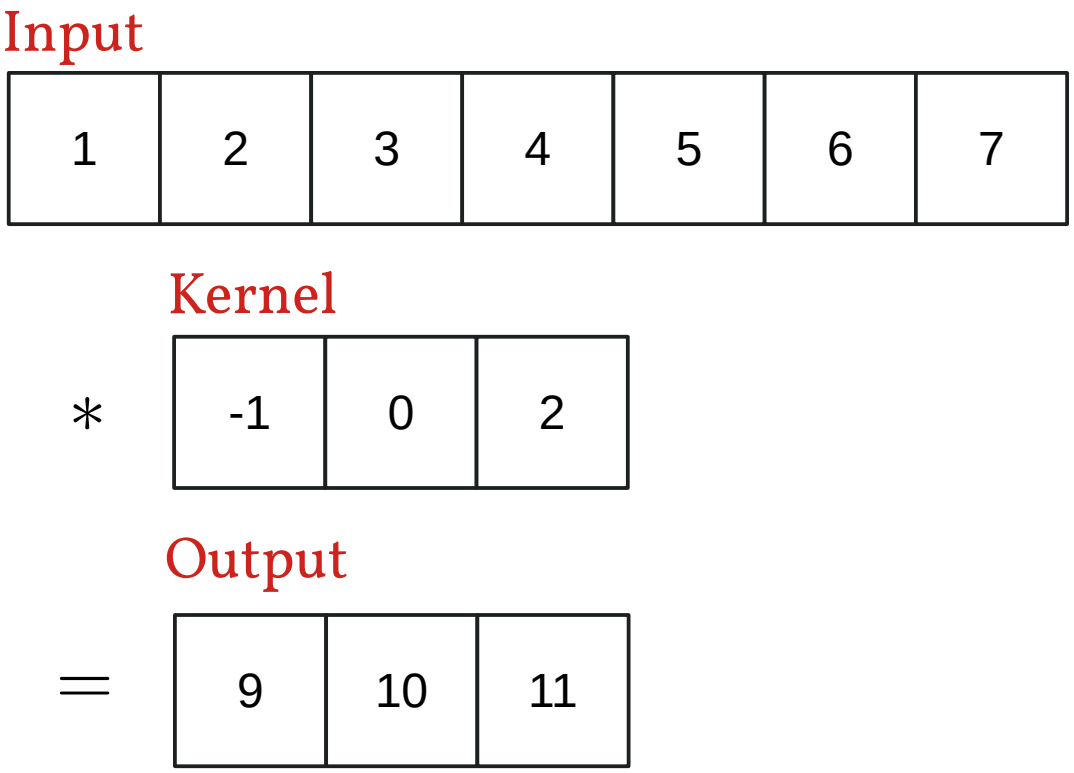
1D-Convolution: Dilation

- Problem: we want to recognize larger patterns
- Solution: dilate receptive field



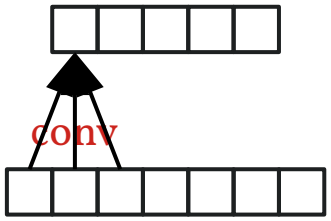
1D-Convolution: Dilation

- Problem: we want to recognize larger patterns
- Solution: dilate receptive field



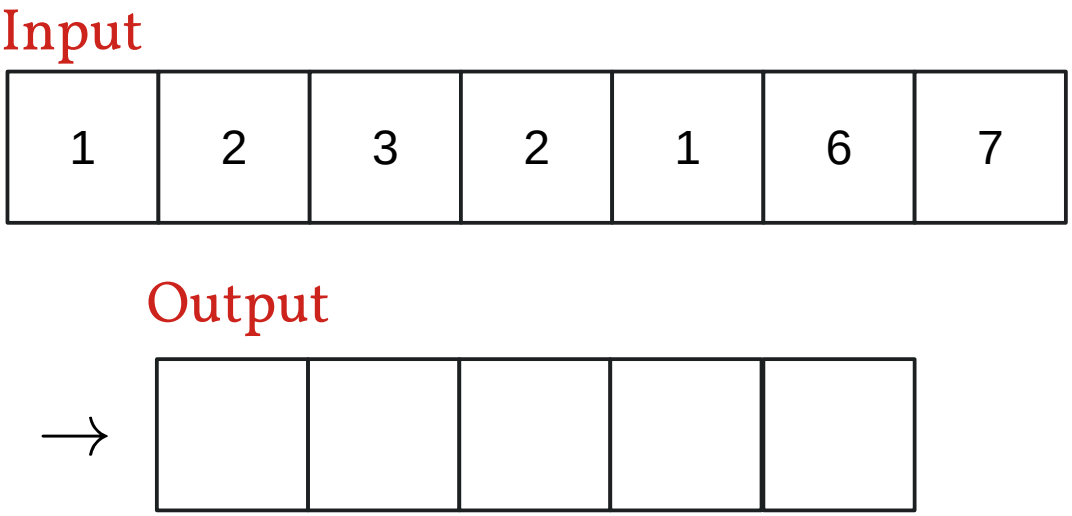
1D-Convolutional Layer

- Idea: convolution with trainable parameter kernels
- Accepts $n_{\text{in-channel}}$-many 1D input sequences
- Holds $n_{\text{out-channel}}$-many trainable parameter kernels
- Performs convolution with each kernel on each input sequence and returns $n_{\text{in-channel}} \cdot n_{\text{out-channel}}$-many 1D output sequences
- Padding, stride, and dilation configurable



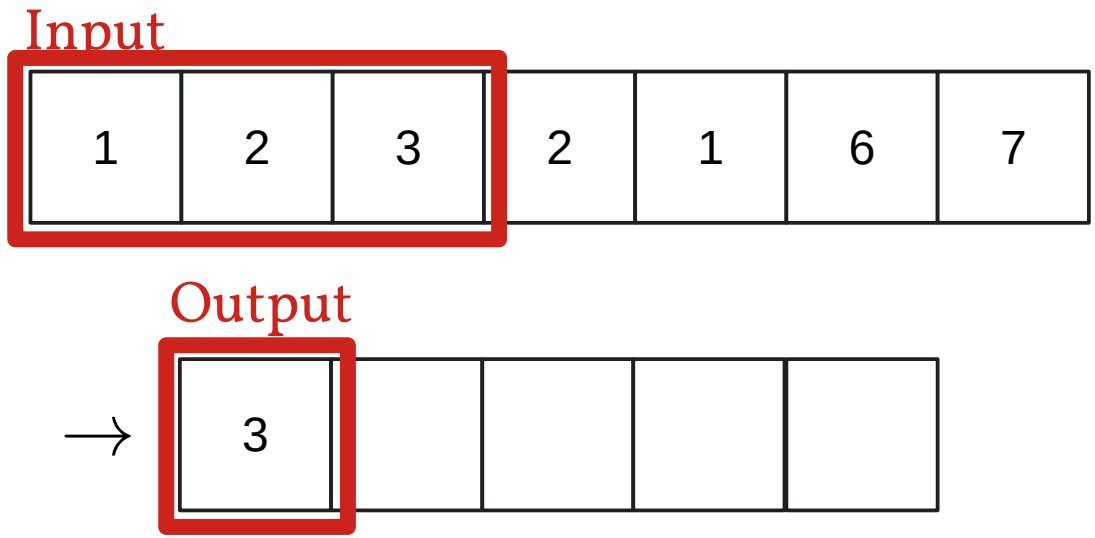
Max-Pooling Layer

- Reduces dimensionality of big hidden layers
- Can be applied along sequence or along channel dimension
- Example of max pooling with window of size 3



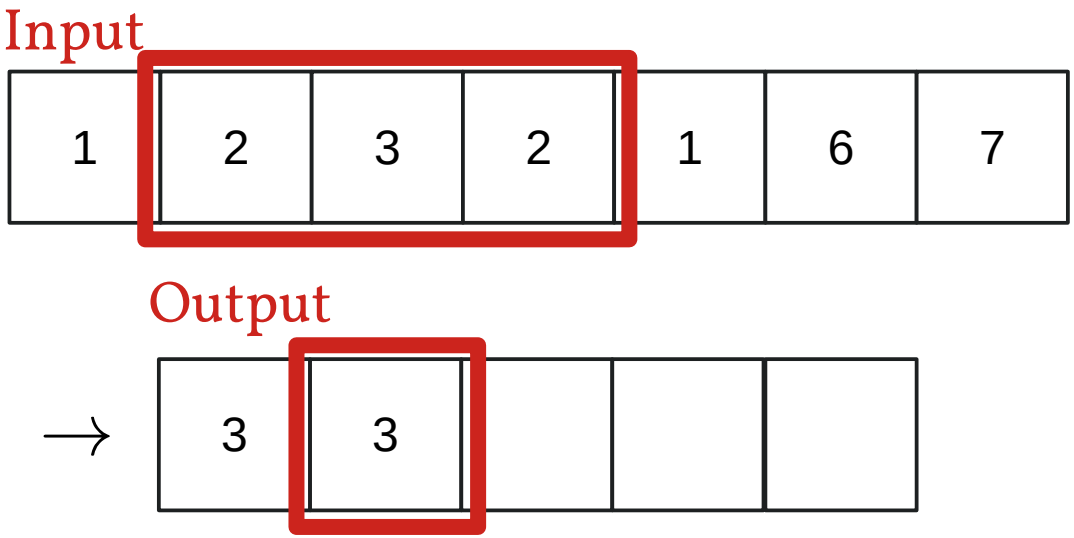
Max-Pooling Layer

- Reduces dimensionality of big hidden layers
- Can be applied along sequence or along channel dimension
- Example of max pooling with window of size 3:



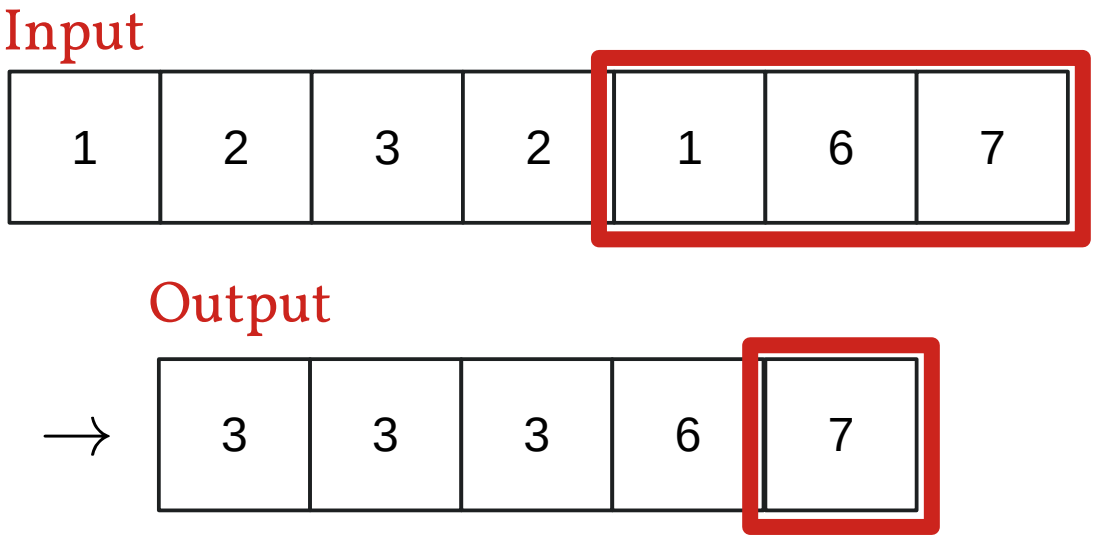
Max-Pooling Layer

- Reduces dimensionality of big hidden layers
- Can be applied along sequence or along channel dimension
- Example of max pooling with window of size 3:



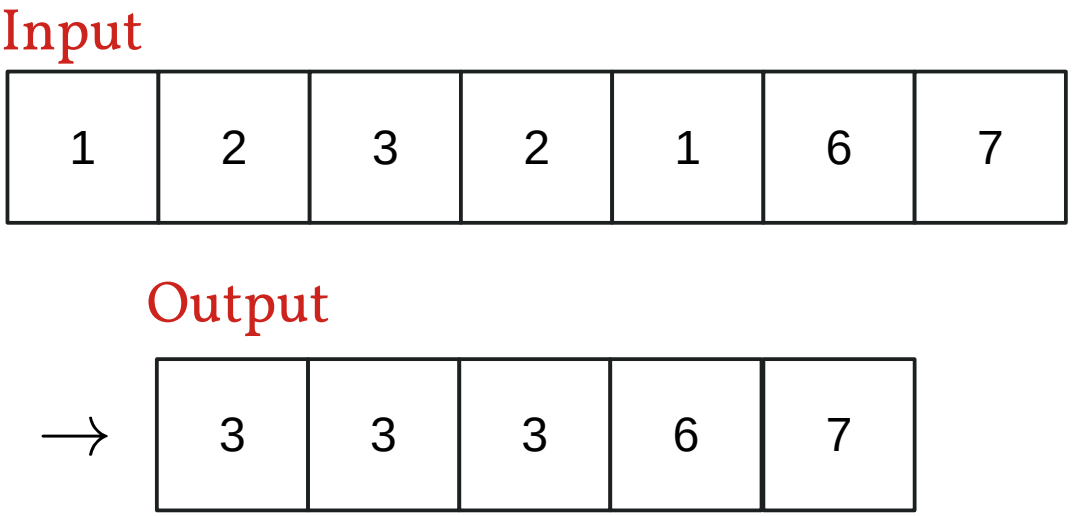
Max-Pooling Layer

- Reduces dimensionality of big hidden layers
- Can be applied along sequence or along channel dimension
- Example of max pooling with window of size 3:



Max-Pooling Layer

- Reduces dimensionality of big hidden layers
- Can be applied along sequence or along channel dimension
- Example of max pooling with window of size 3:



Convolutional Neural Network

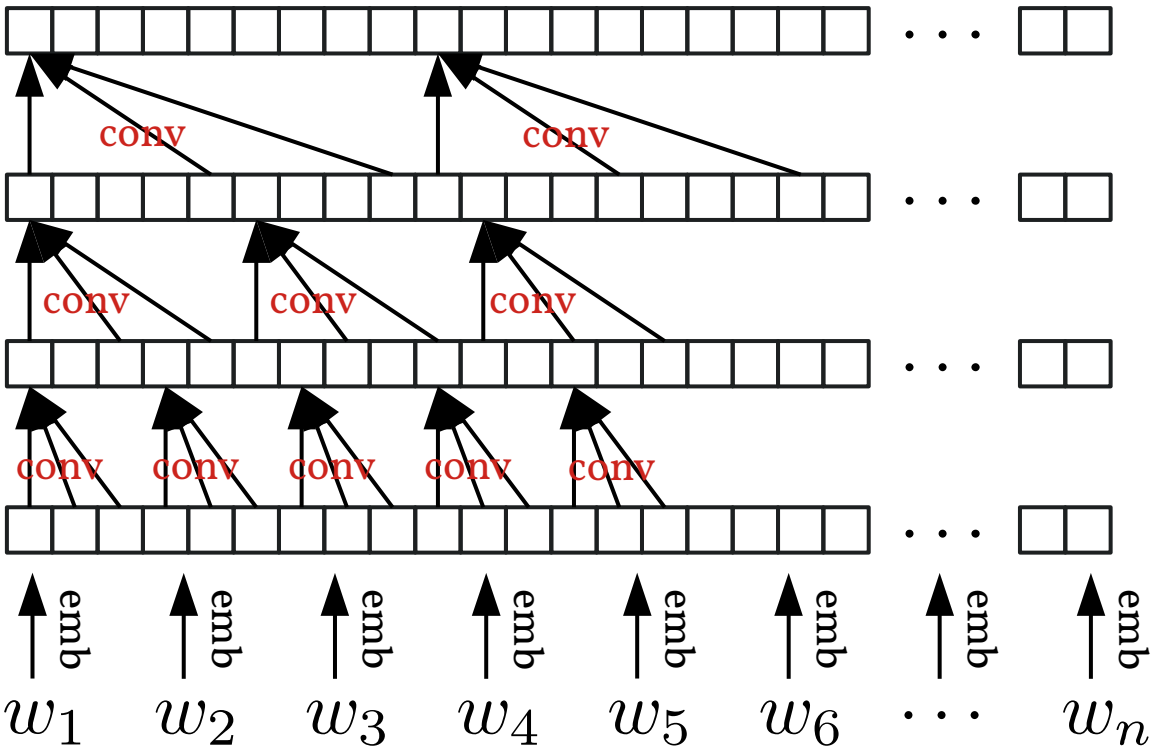
- Input sequence is transformed with embedding layer
- Many stacked convolutional layers
  - For natural language processing almost always 1D-convolutions
- Interspersed pooling layers to reduce dimensionality
- For classification: stacked dense layers at the end

Convolutional Neural Network Analysis

- Convolutional layers only consider **local context**
- Thus easier to train because gradient paths are shorter
  - Allows for deeper networks
- Also **easy to parallelize**
- Nowadays, equal performance to recurrent neural networks
- Interpretation of convolutional layer is to detect specific word sequences of interest

TCN: Temporal Convolutional Network (Bai et al, 2018)

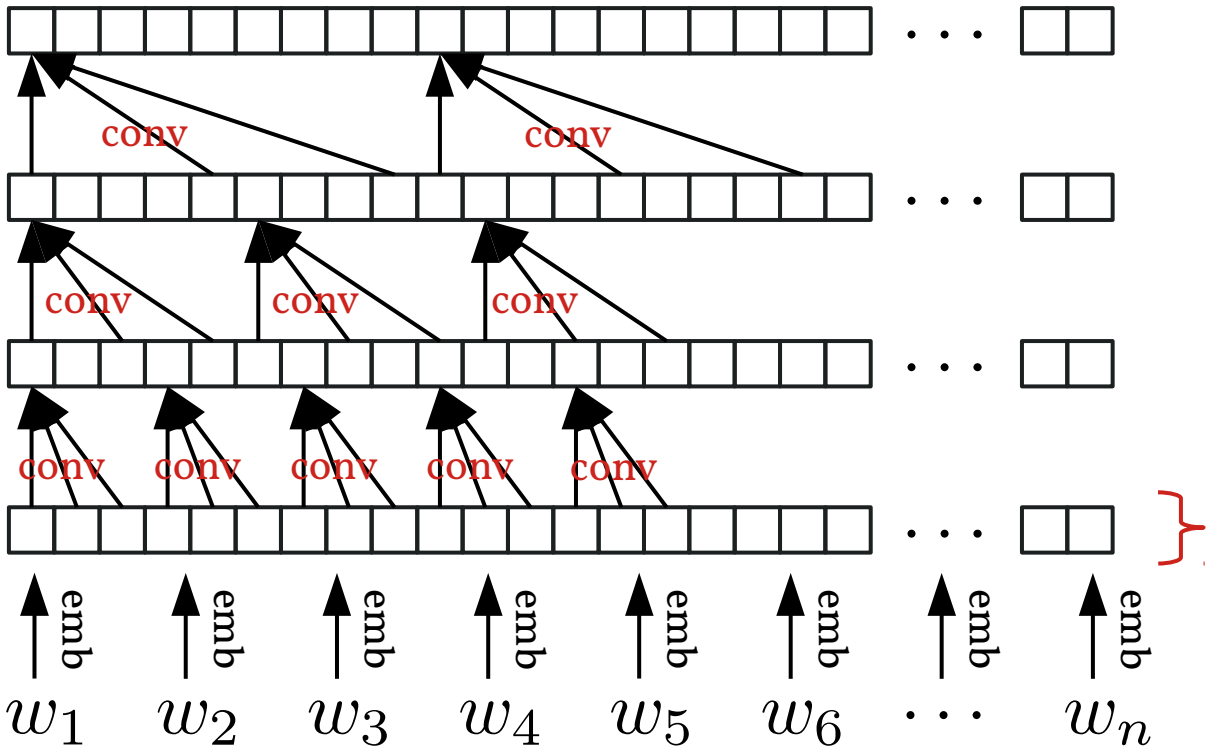
» Sensible generic architecture for sequence modeling



Bai el al (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". *CoRR abs/1803.01271*.

TCN: Temporal Convolutional Network (Bai et al, 2018)

» Sensible generic architecture for sequence modeling

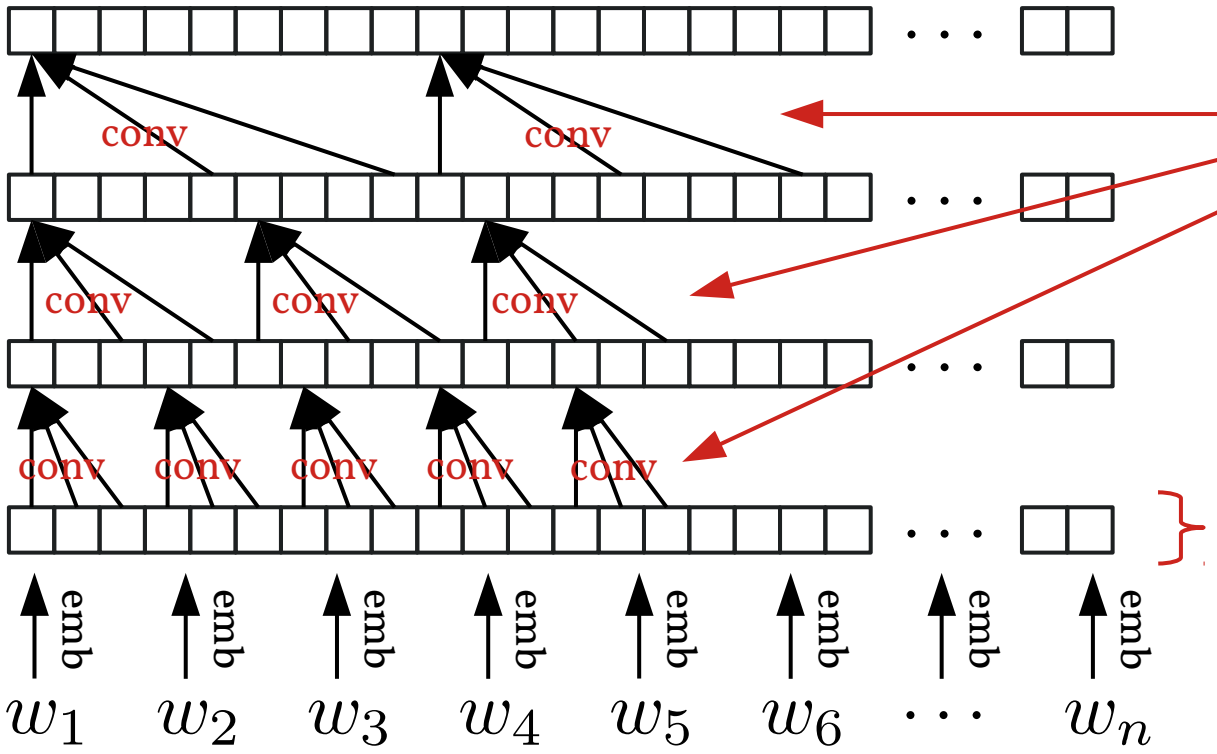


Embeddings become one input sequence with $n_{\text{in-channels}} = d_{\text{emb}}$

Bai el al (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". *CoRR abs/1803.01271*.

TCN: Temporal Convolutional Network (Bai et al, 2018)

» Sensible generic architecture for sequence modeling



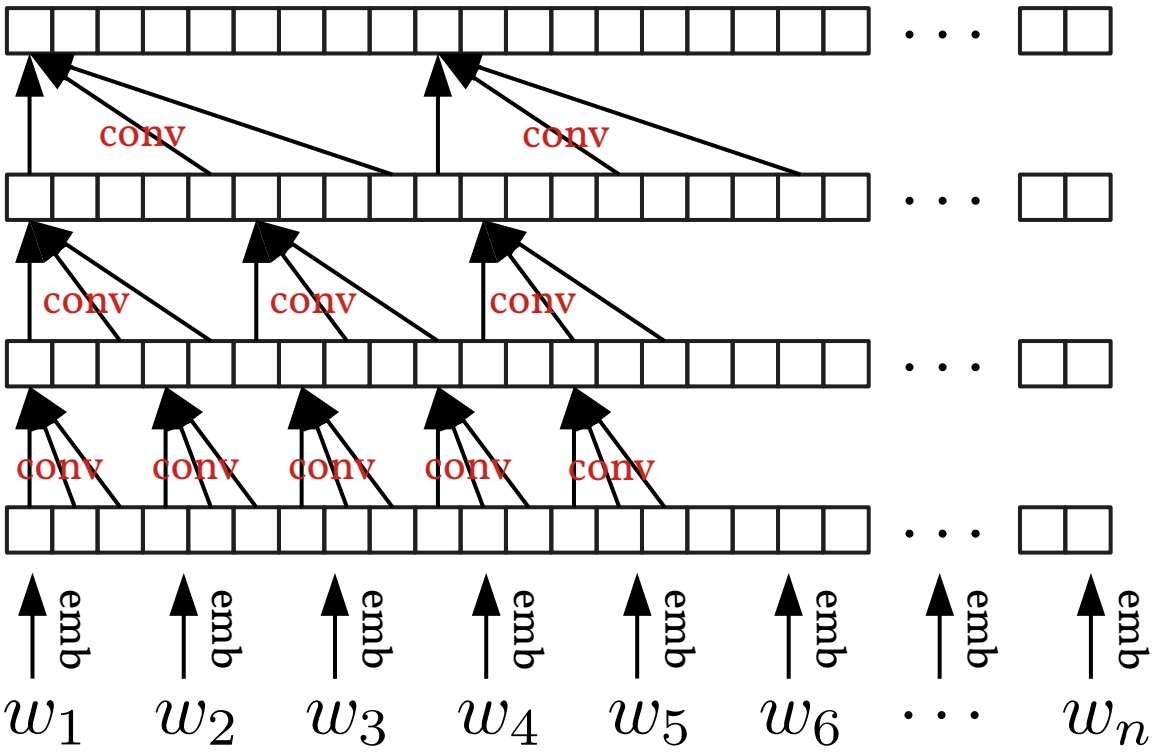
Dilation increases exponentially per layer

Embeddings become one input sequence with $n_{\text{in-channels}} = d_{\text{emb}}$

Bai el al (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". *CoRR abs/1803.01271*.

TCN: Temporal Convolutional Network (Bai et al, 2018)

» Sensible generic architecture for sequence modeling



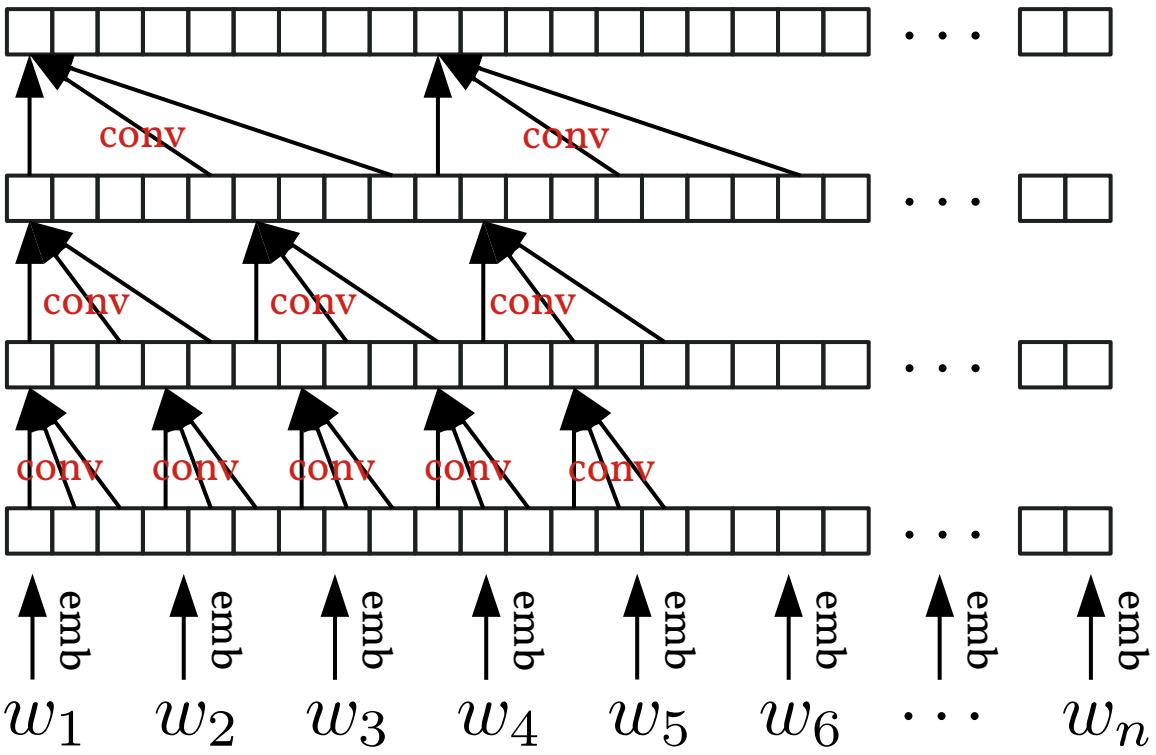
» Stacking of convolutional layers allows for global-ish context

» Exponential dilation results in logarithmic path length

Bai el al (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". *CoRR abs/1803.01271*.

TCN: Temporal Convolutional Network (Bai et al, 2018)

» Sensible generic architecture for sequence modeling



» Not depicted here:

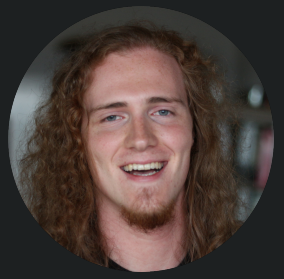
» Dropout and weight normalization after every layer

» Residual connections skipping every second layer

Bai el al (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". *CoRR abs/1803.01271*.

Summary: Established Architectures

- Bag-of-words discards word order
- Word embeddings encode word similarity
- Recurrent and convolutional layers allow to model sequential input and achieve comparable performance
- Recurrent networks are hard to train
  - not parallelizable
  - long gradient paths because of global context
- Convolutional networks are easy to train
  - easily parallelizable
  - short gradient paths because of local context



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Introduction to Advanced Neural Network Architectures

for Natural Language Processing

» Attention

Motivation of Attention

- Review question answering:

  Formulate an answer to a given question using a given context

  - Context: *"In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity… Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals* ***within a cloud****. Short, intense …"*
  - Question: *"Where do water droplets collide with ice crystals to form precipitation?"*
  - Answer: ***"within a cloud"***

- For some tasks, input sequences can be extremely long and not every word has equal importance
- Idea: give more weight to important words

Review: Lookup Table

Keys Values

```
t = {“a”: 10,
     “b”: 15,
     “c”: 200,
     “d”: -1}
```

Queries Result

```
t[“b”] = 15
t[“d”] = -1
```

Attention Mechanism

» An attention mechanism implements a lookup table only that values, keys, and the query are continuous vectors

Keys: $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subset \mathbb{R}^{d_k}$

Values: $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^{d_v}$

Query: $\mathbf{q} \in \mathbb{R}^{d_k}$

Attention per key for query: $a_i = \frac{\exp(\mathbf{q} \cdot \mathbf{k}_i^\top)}{\sum_{j=1}^{n} \exp(\mathbf{q} \cdot \mathbf{k}_j^\top)}$

Result: $\mathbf{c} = \sum_{i=1}^{n} a_i \cdot \mathbf{v}_i$

Attention Mechanism

» An attention mechanism implements a lookup table only that values, keys, and the query are continuous vectors

Keys: $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subset \mathbb{R}^{d_k}$

Values: $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \mathbb{R}^{d_v}$

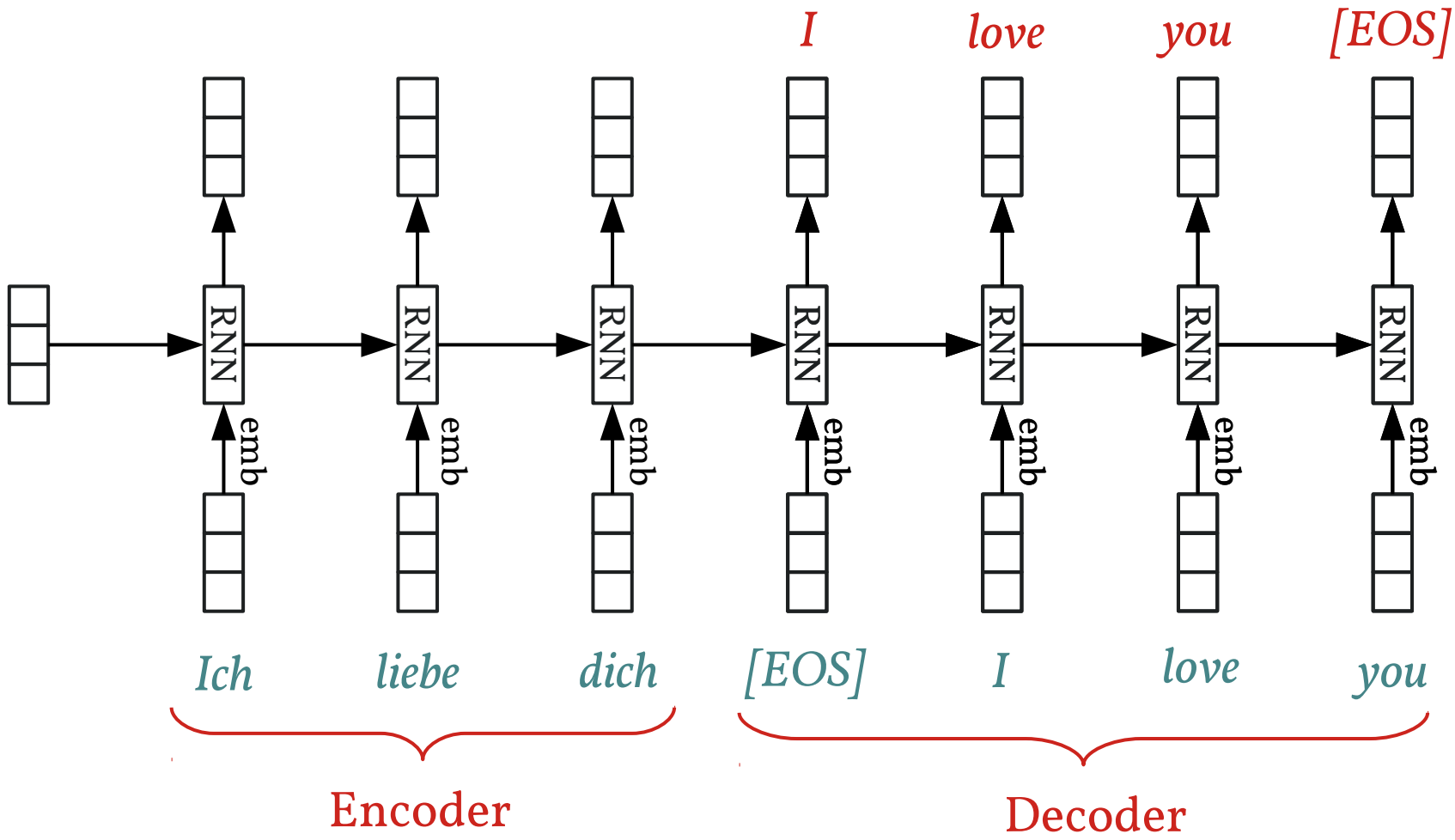
Query: $\mathbf{q} \in \mathbb{R}^{d_k}$

Attention per key for query: $a_i = \frac{\exp(\mathbf{q} \cdot \mathbf{k}_i^\top)}{\sum_{j=1}^{n} \exp(\mathbf{q} \cdot \mathbf{k}_j^\top)}$

Result: $\mathbf{c} = \sum_{i=1}^{n} a_i \cdot \mathbf{v}_i$

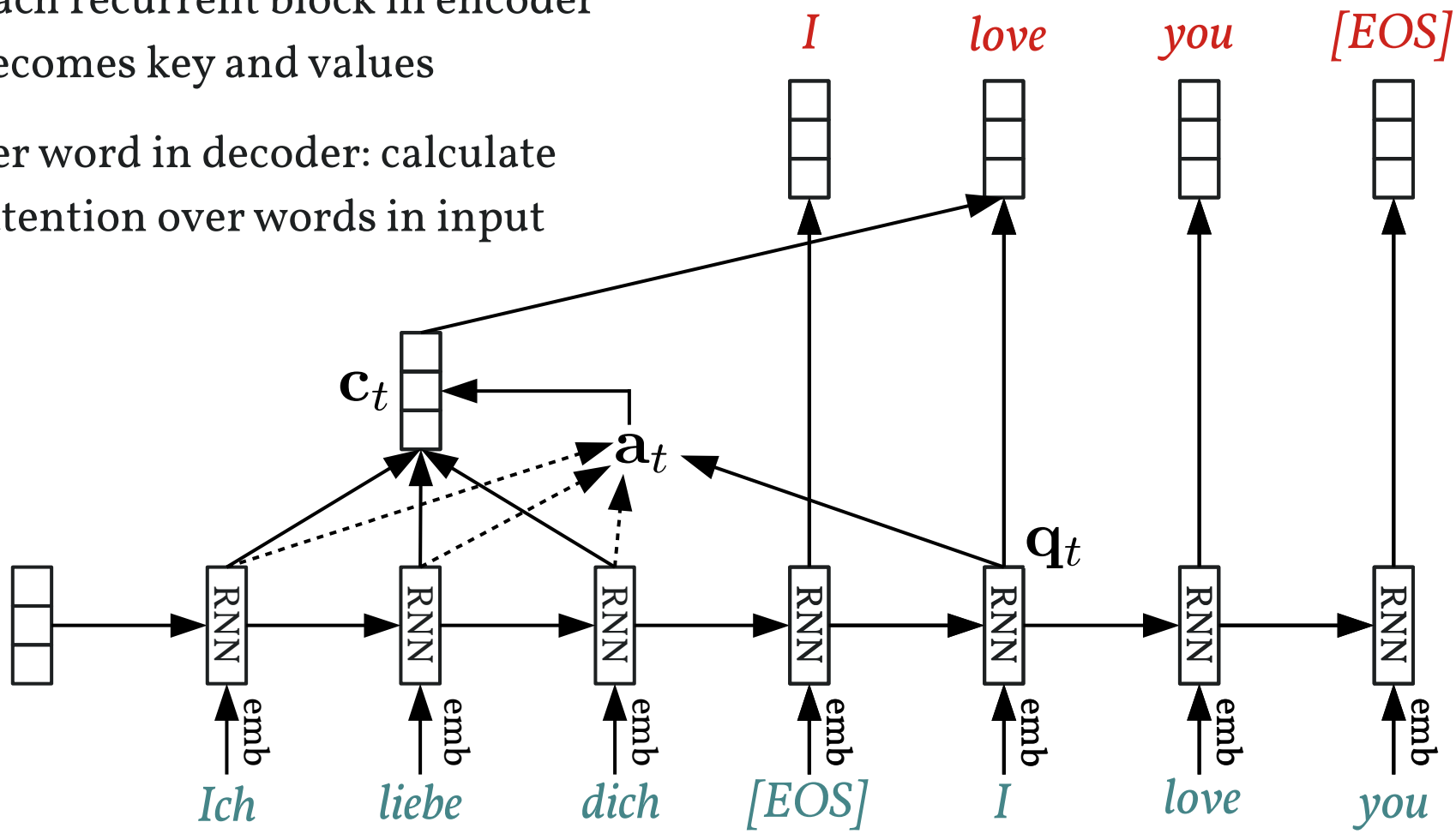
(other ways to calculate attention weights exists)

Review: Recurrent Neural Network for Machine Translation



Recurrent Neural Network with Attention for Machine Translation (Luong, 2015)

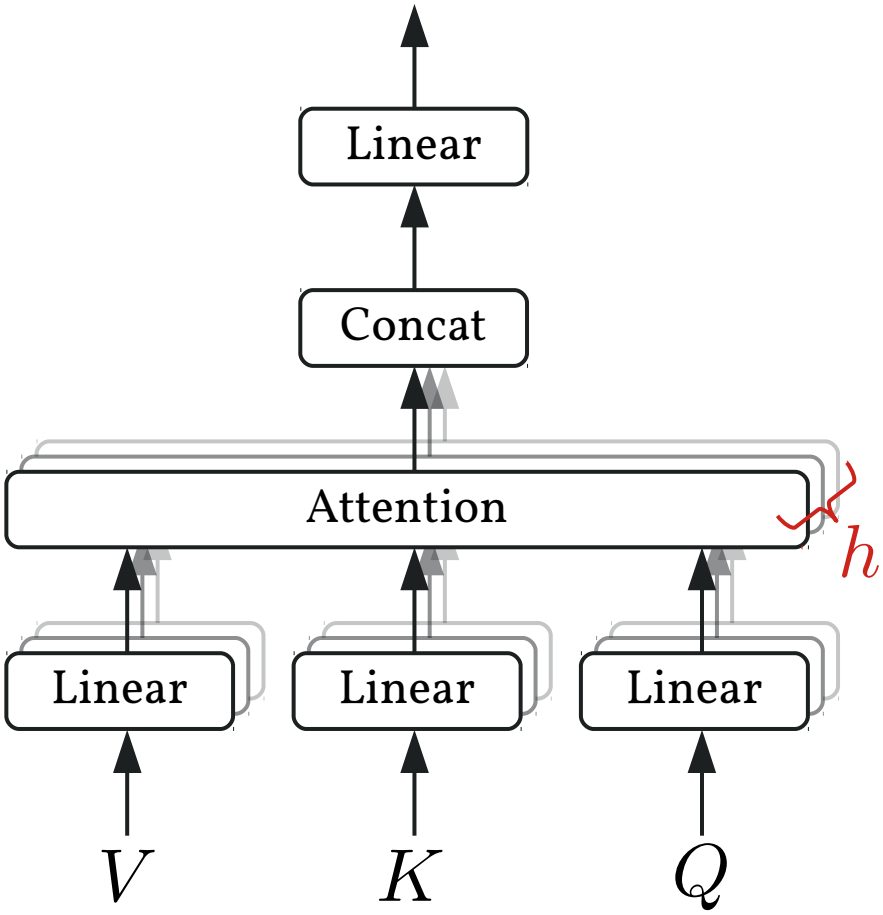
- Each recurrent block in encoder becomes key and values
- Per word in decoder: calculate attention over words in input



Luong (2015). "Effective Approaches to Attention-based Neural Machine Translation". *EMNLP*.

Multi-Head Attention (Vaswani et al, 2017)

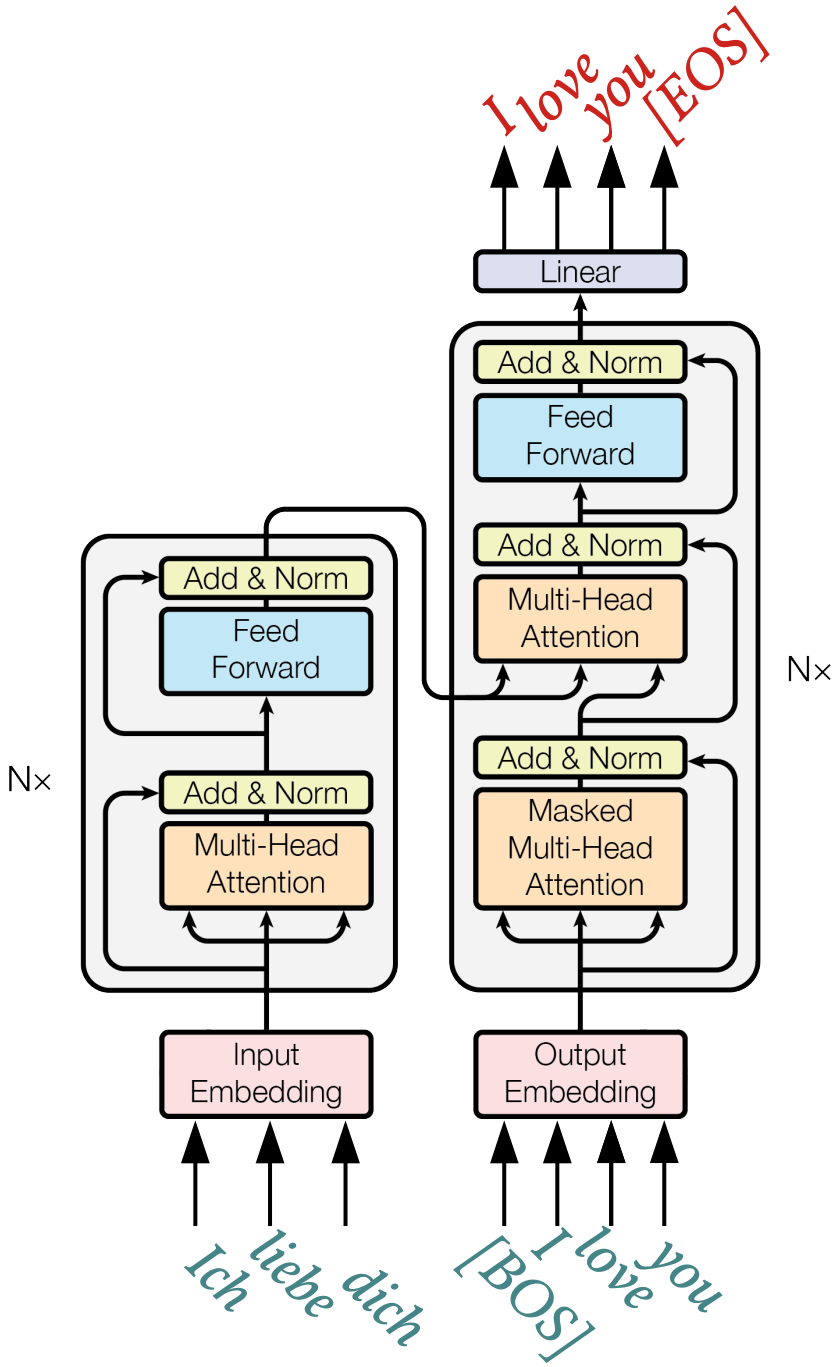
- Have a set of queries Q
  - Each query has own result
- Learn h-many linear projections for values, keys, and queries
  - Per query all h results concatenated together



Vaswani et al (2017). "Attention Is All You Need". *NIPS*.

Transformer (Vaswani et al, 2017)

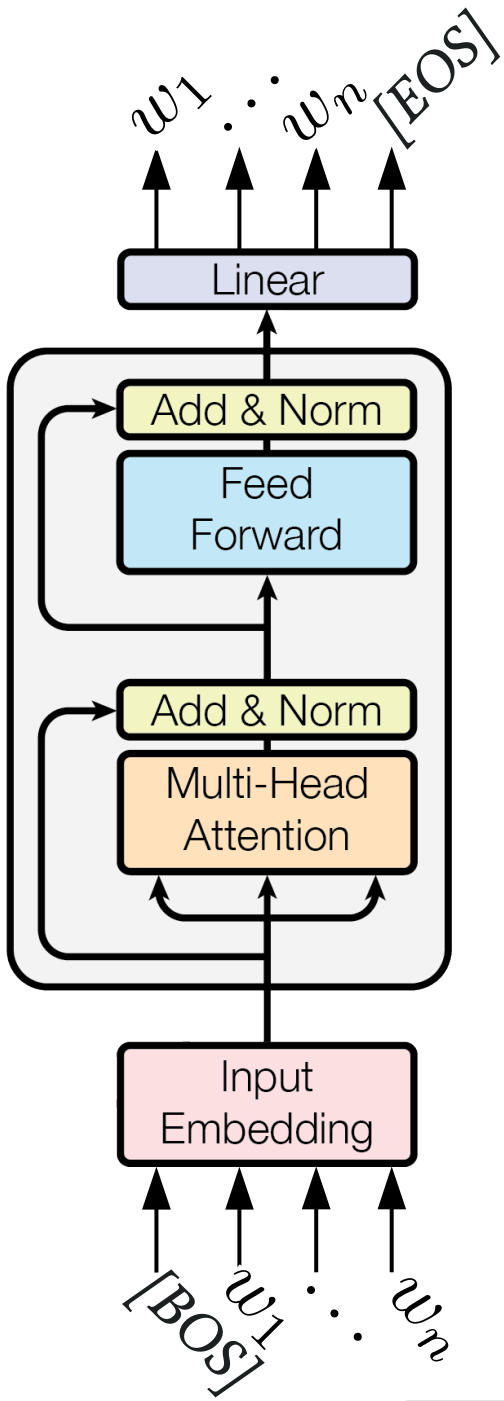
- Machine translation with familiar encoder/decoder architecture
- Avoids recurrent/convolutional blocks completely, only uses attention
- **Allows global dependencies with constant path length**



Vaswani et al (2017). "Attention Is All You Need". *NIPS*.

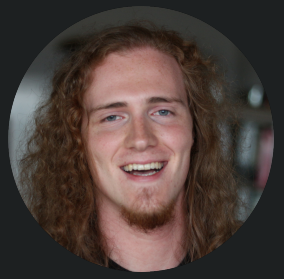
Transformer Analysis

- Alternative to recurrent or convolutional layers
- Similar to the convolution layer there is no sequential operation and short gradient paths
  - Easy to parallelize
  - Easy to train
- Similar to the recurrent layer it can consider global context
- For some tasks: need to manually mask some attentions paths to not cheat



Summary: Attention

- Attentions allows to give extra weight to important words in the input sequence
- Continuous extension of lookup table
- Can be used in recurrent and convolutional architectures
- Transformer block is alternative to recurrent and convolutional layers
  - Completely build on attention
  - Usually at least equal performance



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

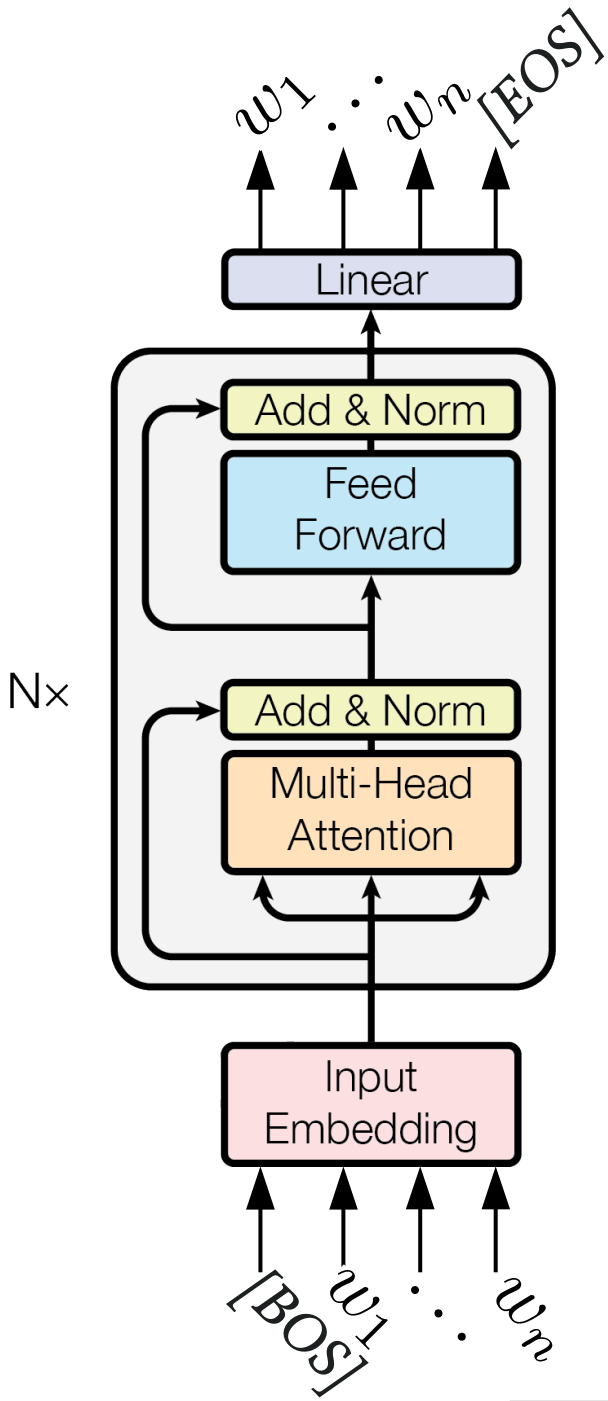
Introduction to Advanced Neural Network Architectures

for Natural Language Processing

» State of the Art

OpenAI GPT (Radford, 2018)

- General model for solving many different supervised natural language tasks
  - Model is just stack of transformer blocks
- Problem: in supervised setting, we often don't have enough training data
- Parameters are first *pre-trained* using language modeling (much data available)
- Learned parameters are then *fine-tuned* for each specific task of interest



Radford (2018). "Improving Language Understanding by Generative Pre-Training". https://openai.com/blog/language-unsupervised/

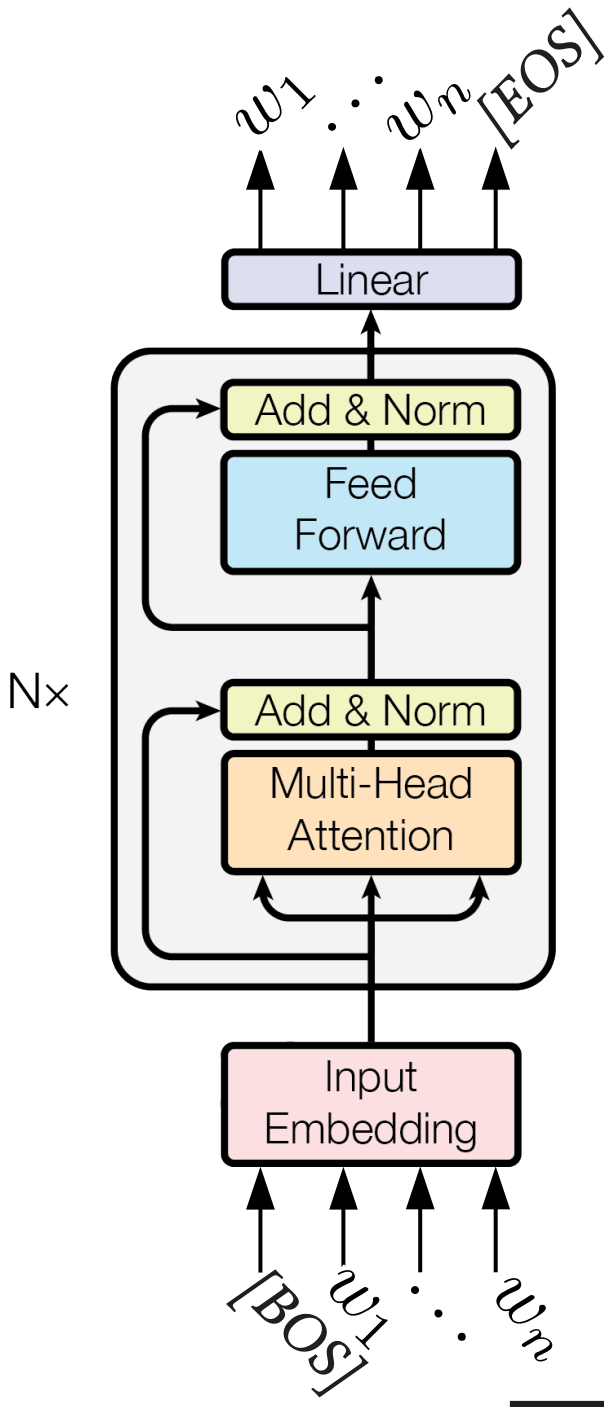
Unsupervised Pre-Training

- Use language modeling for parameter pre-training

  Input: *"[BOS] I speak fluent french"*

  Output: *"I speak fluent french [EOS]"*

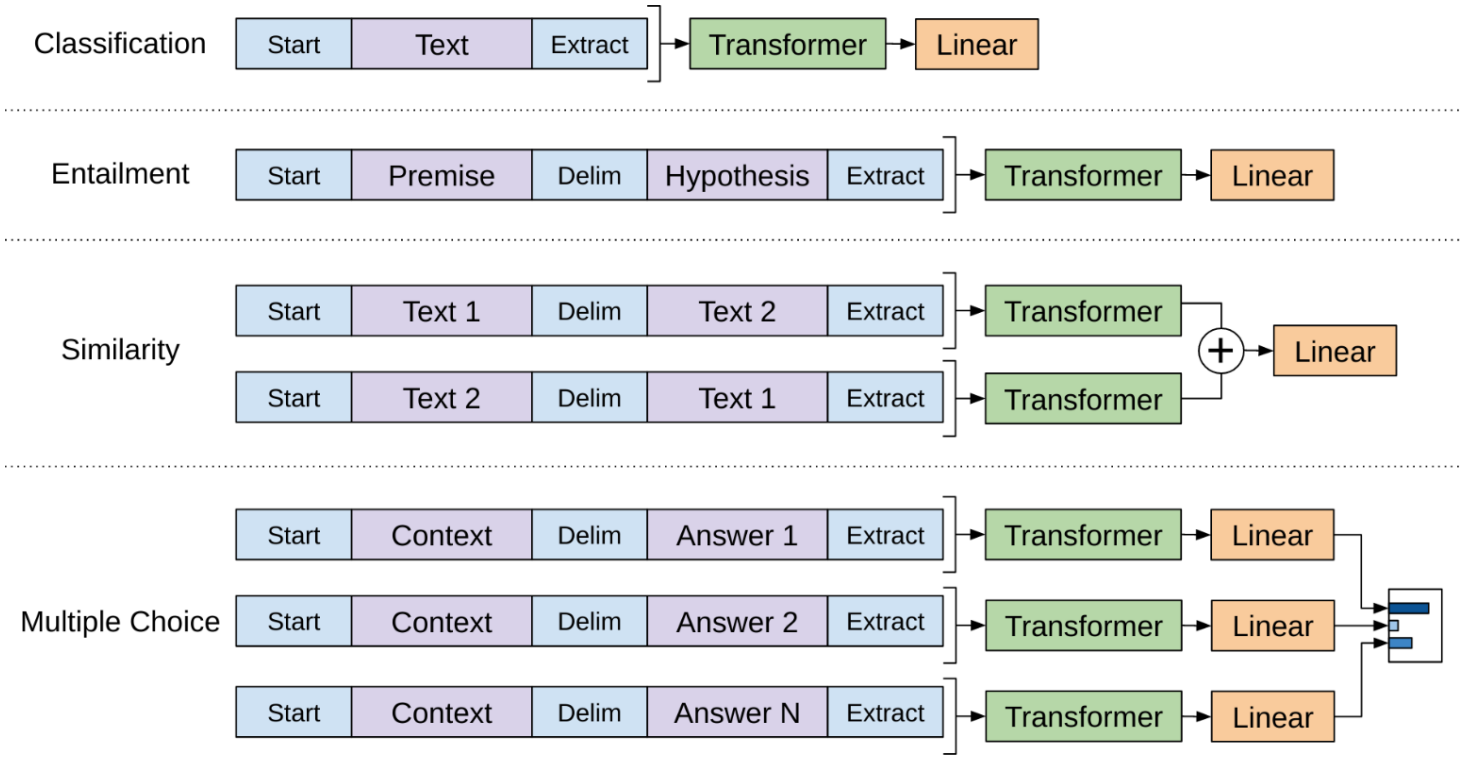
- Input: sentence from unlabeled corpus
- Output: same sentences shifted left
- Task: predict next word
- **Enforces general understanding of the target language**
- **Model can't allow flow from future input tokens to current output**



Radford (2018). "Improving Language Understanding by Generative Pre-Training". https://openai.com/blog/language-unsupervised/

Fine-Tuning

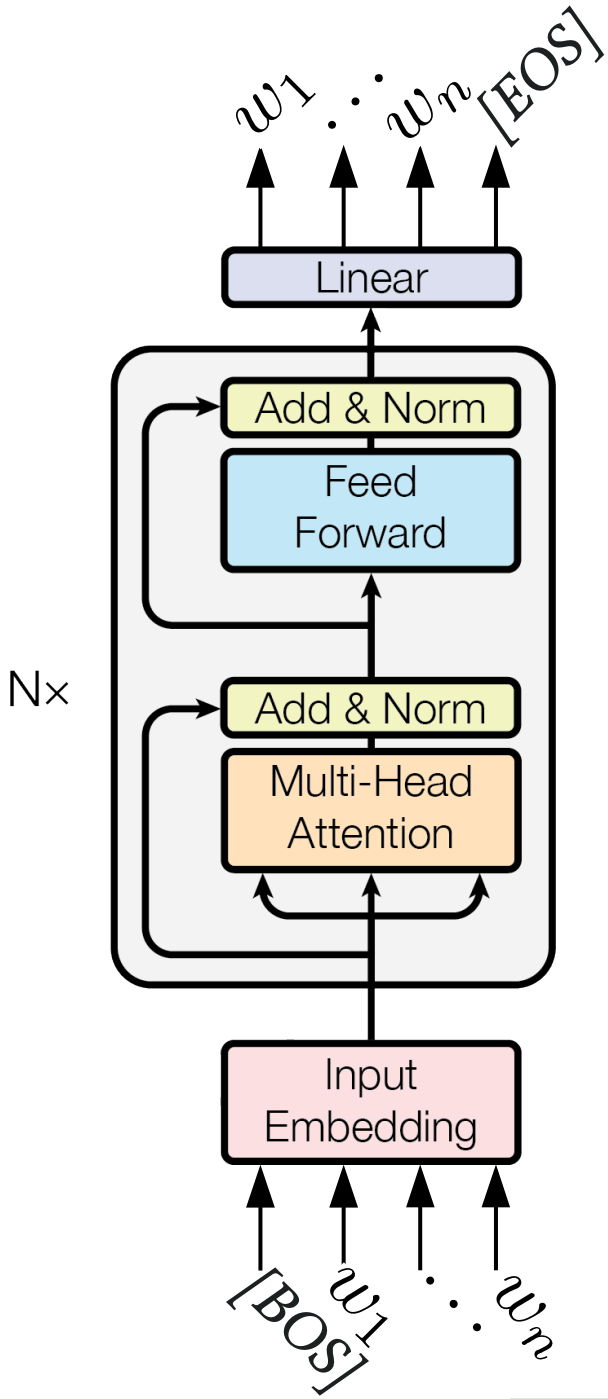
- Start with trained parameters from language modeling
- Fine-tune for each task with input mapping scheme



Radford (2018). "Improving Language Understanding by Generative Pre-Training". https://openai.com/blog/language-unsupervised/

BERT (Devlin et al, 2019)

- Same model architecture as OpenAI GPT, also pre-training and fine-tuning
- Main downside of previous work: while architectures might support global attention the language modeling task forces models to train unidirectional
- **BERT introduces different pre-training task that allows bidirectional training**



Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

Pre-Training: Masked Language Model

Input: *"I [MASK] fluent french"*

Output: *"I speak fluent french"*

- Replace 15% of input tokens with *[MASK]*
- Task: reconstruct missing tokens
  - Allows to consider context before and after those tokens
- But *[MASK]*-token never occurs during fine-tuning: mismatch!
  - To mitigate: 10% of time replace *[MASK]*-token with random word

Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

Pre-Training: Next Sentence Prediction

- BERT's actual input are always two sentences:

  "[CLS] I was raised in France [SEP] So I [MASK] fluent French"

- Task: perform masked language modeling and predict whether the second sentence follows the first sentence
- Can also be generated from unlabeled corpora (much data)

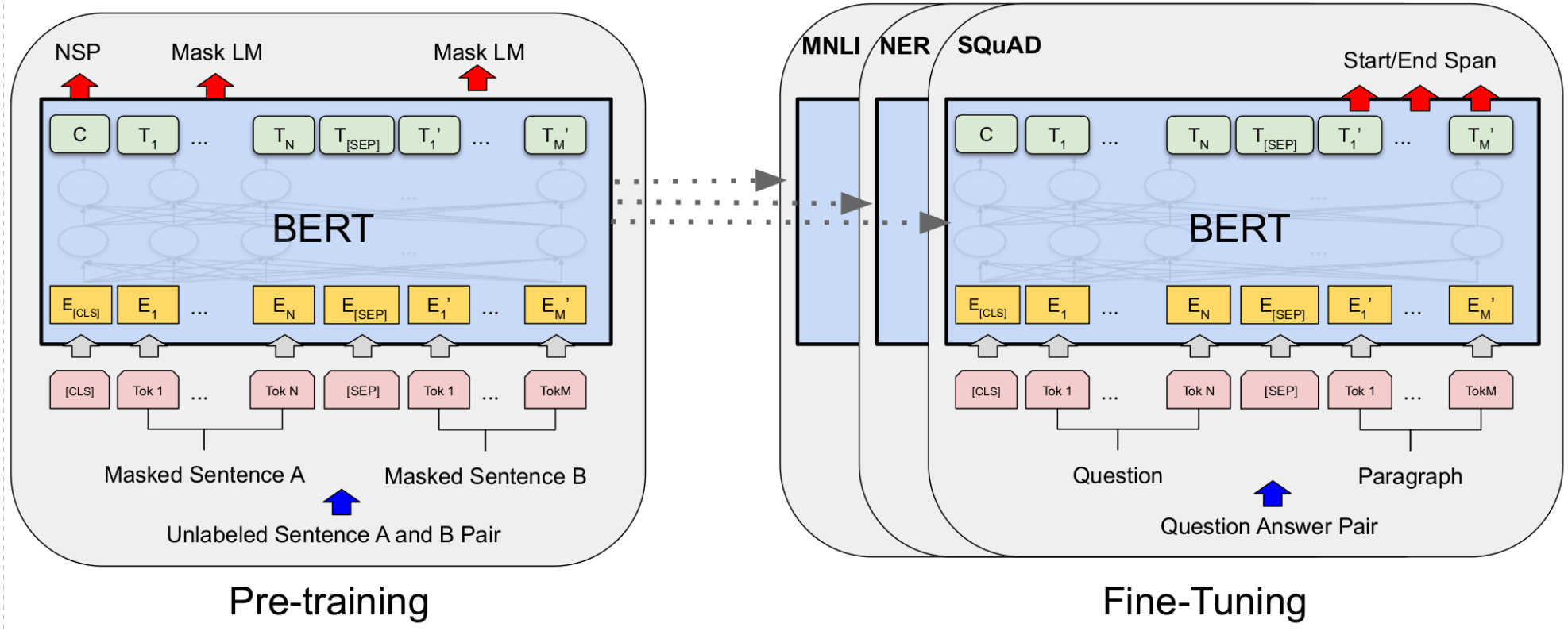
Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

BERT's Pre-training Analysis

- Input are always to sentences:
  - $[\text{CLS}]\ s_1^1\ s_2^1\ s_3^1\ \ldots\ s_n^1\ [\text{SEP}]\ s_1^2\ s_2^2\ s_3^2\ \ldots\ s_m^2$
- During pre-training
  - First task is to detect mask tokens in both sentences
  - Second task is to detect whether sentences occur after each other
- Trains model to
  - Understand general word semantics in a language
  - Understand relation of sentences

Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

Fine-Tuning

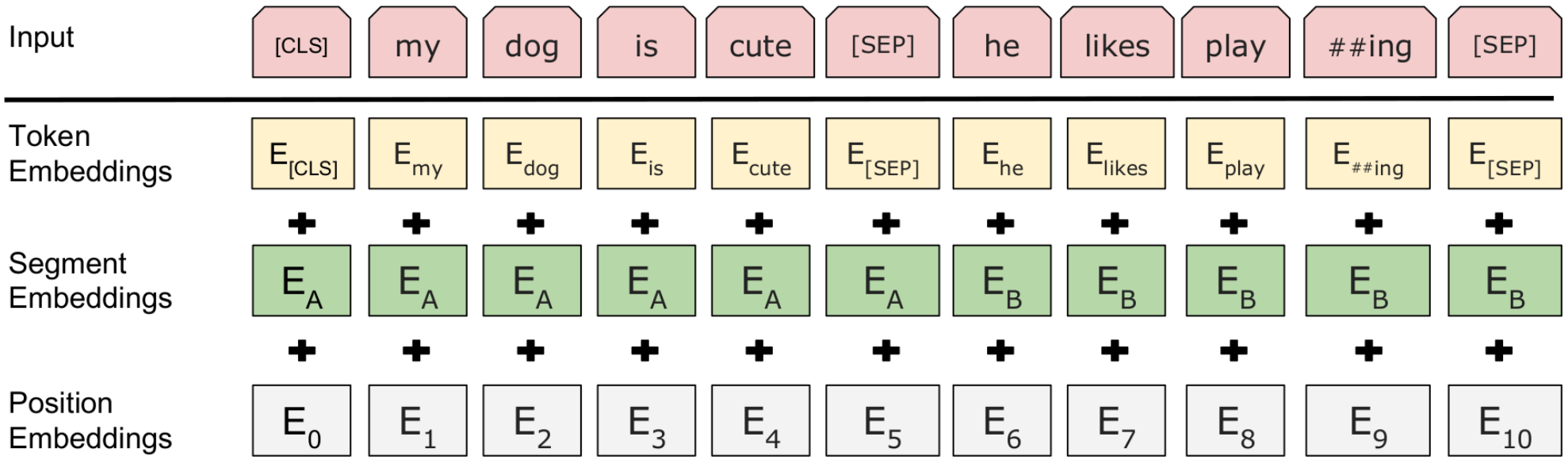


» BERT achieved state-of-the-art in eleven very competitive natural language tasks (big improvements over second places)

Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

BERT: Details Not Mentioned

» WordPiece, position and segment embeddings



Devlin et al (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT*.

Can I use BERT on my own task?

- Check out **PyTorch-Transformers**!
  https://huggingface.co/pytorch-transformers/
- PyTorch implementations of many transformer models
- Includes pre-trained model weights shared by paper authors as well as additional ones
- Includes helpers to apply pre-trained transformers to most down-streak task types
- Exemplary code style and quality

PyTorch-Transformers: Implemented Models

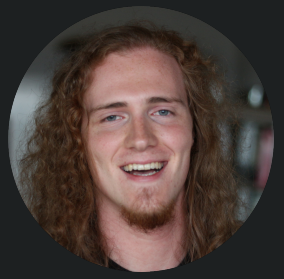
11 Jun 2018	Improving Language Understanding by Generative Pre-Training
11 Oct 2018	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
9 Jan 2019	Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
22 Jan 2019	Cross-lingual Language Model Pretraining
14 Feb 2019	Language Models are Unsupervised Multitask Learners
19 Jun 2019	XLNet: Generalized Autoregressive Pretraining for Language Understanding
26 Jul 2019	RoBERTa: A Robustly Optimized BERT Pretraining Approach
28 Aug 2019	Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT

Pre-training vs Word Embeddings

- Both
  - trained on large unlabeled corpora
  - used to capture general language understanding
  - used to transfer that to specific down-stream tasks
- Word embeddings
  - are much faster and cheaper to train and use (less disk-space, no reliance on GPUs)
  - require task-specific architectures
- Pre-training
  - uses task-agnostic architectures
  - achieves much better performance
  - models use word embeddings at the input layer

Summary: State of the Art

- Pre-training is an initial expensive training phase on some auxiliary tasks
  - Usually one where lots of data is available
  - Pre-trained weights available for download
- Fine-tuning then takes trained parameters and adjust them further based on the target task
  - Doesn't require to much computation
  - Pre-trained weights can be used for multiple tasks
- Architectures easy to adapt to new tasks



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Introduction to Advanced Neural Network Architectures

for Natural Language Processing

» Summary

Summary: Introduction

- Machine learning is function approximation
- Gradient descent can find good model parameters
  - Only works if model function is differentiable
- Many natural language processing tasks are either **sequence classification** or **sequence-to-sequence**

Summary: Deep Learning Basics

- Neural networks have fixed input and output dimension
- Deep feed-forward neural networks are stacks of dense layers
- Finding good hyperparameter values requires lots of experimentation
- Dropout, layer normalization, and residual connections help networks learn better

Summary: Established Architectures

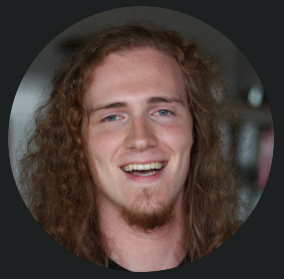
- Bag-of-words discards word order
- Word embeddings encode word similarity
- Recurrent and convolutional layers allow to model sequential input and achieve comparable performance
- Recurrent networks are hard to train
  - not parallelizable
  - long gradient paths because of global context
- Convolutional networks are easy to train
  - easily parallelizable
  - short gradient paths because of local context

Summary: Attention

- Attentions allows to give extra weight to important words in the input sequence
- Continuous extension of lookup table
- Can be used in recurrent and convolutional architectures
- Transformer block is alternative to recurrent and convolutional layers
  - Completely build on attention
  - Usually at least equal performance

Summary: State of the Art

- Pre-training is an initial expensive training phase on some auxiliary tasks
  - Usually one where lots of data is available
  - Pre-trained weights available for download
- Fine-tuning then takes trained parameters and adjust them further based on the target task
  - Doesn't require to much computation
  - Pre-trained weights can be used for multiple tasks
- Architectures easy to adapt to new tasks



LUKASSCHMELZEISEN.

lukas@uni-koblenz.de
lschmelzeisen.com

6 Sep 2019

Questions?

1. Introduction
   - » Machine Learning Basics
   - » Natural Language Processing Tasks
2. Review: Deep Learning Basics
   - » Feed-forward Neural Networks
   - » Common Improvements
3. Classical Architectures
   - » Word Embeddings
   - » Recurrent Neural Networks
   - » Convolutional Neural Networks
4. Attention Mechanism
   - » Attention-based Recurrent Neural Networks
   - » Transformers
5. State of the Art
   - » OpenAI GPT
   - » BERT

Thank you for reading so far!

- Contact me if you have any questions
- Also, please forward all typos, errors, inaccuracies or inconsistencies you find in these slides to me