CS 4973/ CS 6983

Trustworthy Generative AI
Fall 2024

Alina Oprea
Professor
Khoury College of Computer Science

September 12 2024

Outline: Review of ML

- Classification and Regression
- Gradient descent for training models
- Deep learning
  - Neural networks architectures
  - Feed-forward neural networks
  - Convolutional networks
- Large Language Models (LLMs)
  - Transformers and self-attention
  - GPT-2 architecture

Supervised Learning: Classification

Training

Data → Pre-processing → Feature extraction → ML model

Labeled

x_i

y_i

Normalization

Feature Selection

$h(x)$

Testing

New data → ML model → Predictions

$y' = h(x')$

Unlabeled

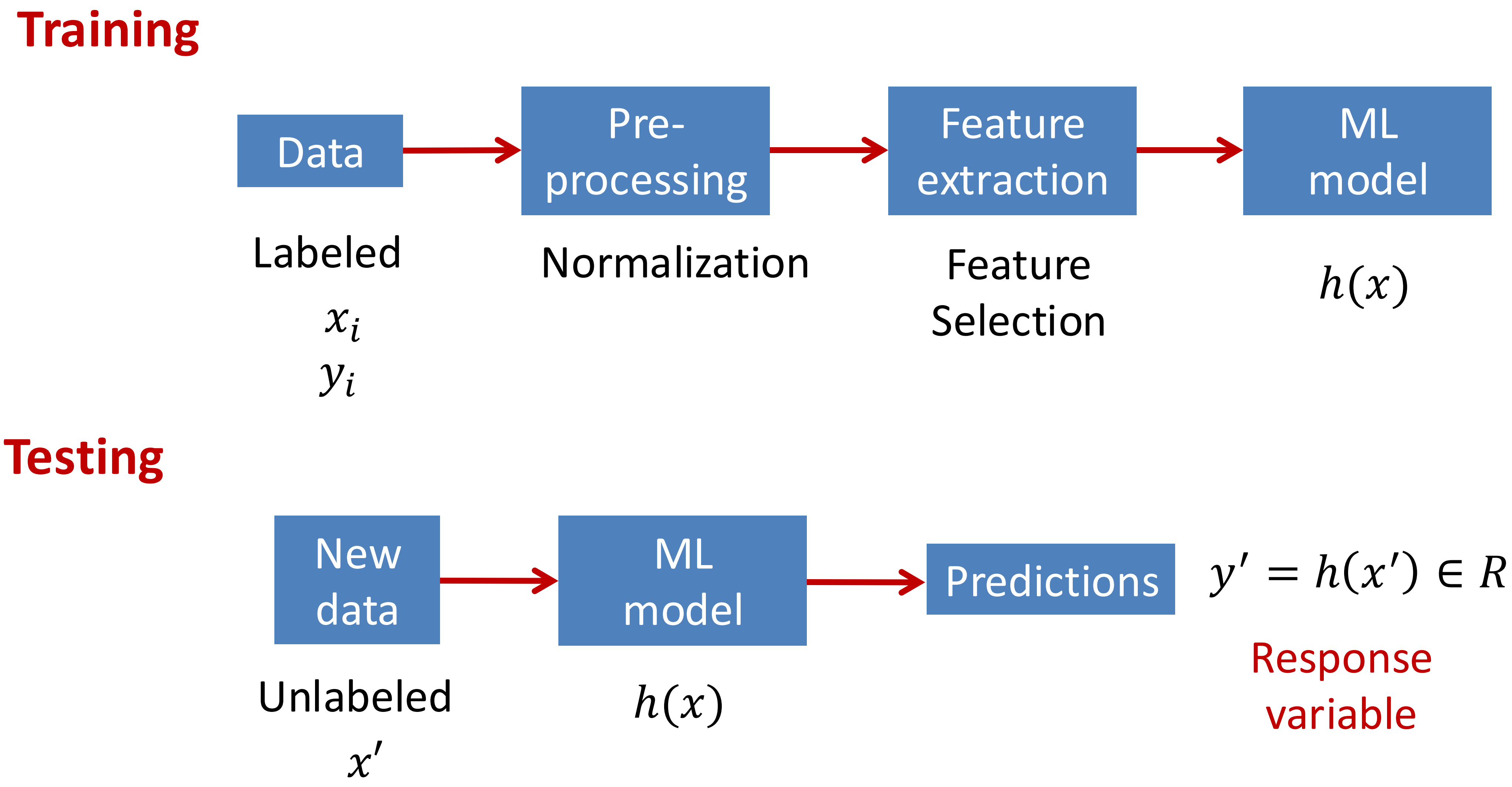
x'

$h(x)$

Positive

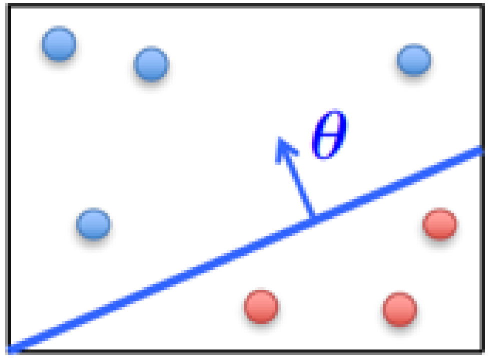
Negative

Supervised Learning: Regression



Supervised learning

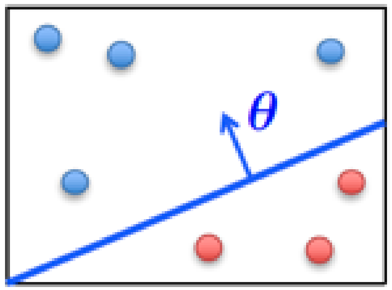
- Training data
  - $x_i = [x_{i,1}, \dots x_{i,d}]$: vector of features
  - y_i: labels
- Models (hypothesis)
  - Example: Linear model
    - $h_\theta(x) = \theta_0 + \theta_1 x$
- Loss function
  - Error function to minimize during training
- Training algorithm
  - Training: Learn model parameters θ to minimize objective
  - Output: "optimal" model according to loss function
- Testing
  - Apply learned model to new data x' and generate prediction $h(x')$



Linear Classifiers

- **Linear classifiers:** represent decision boundary by hyperplane

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \ldots & x_d \end{bmatrix}$$

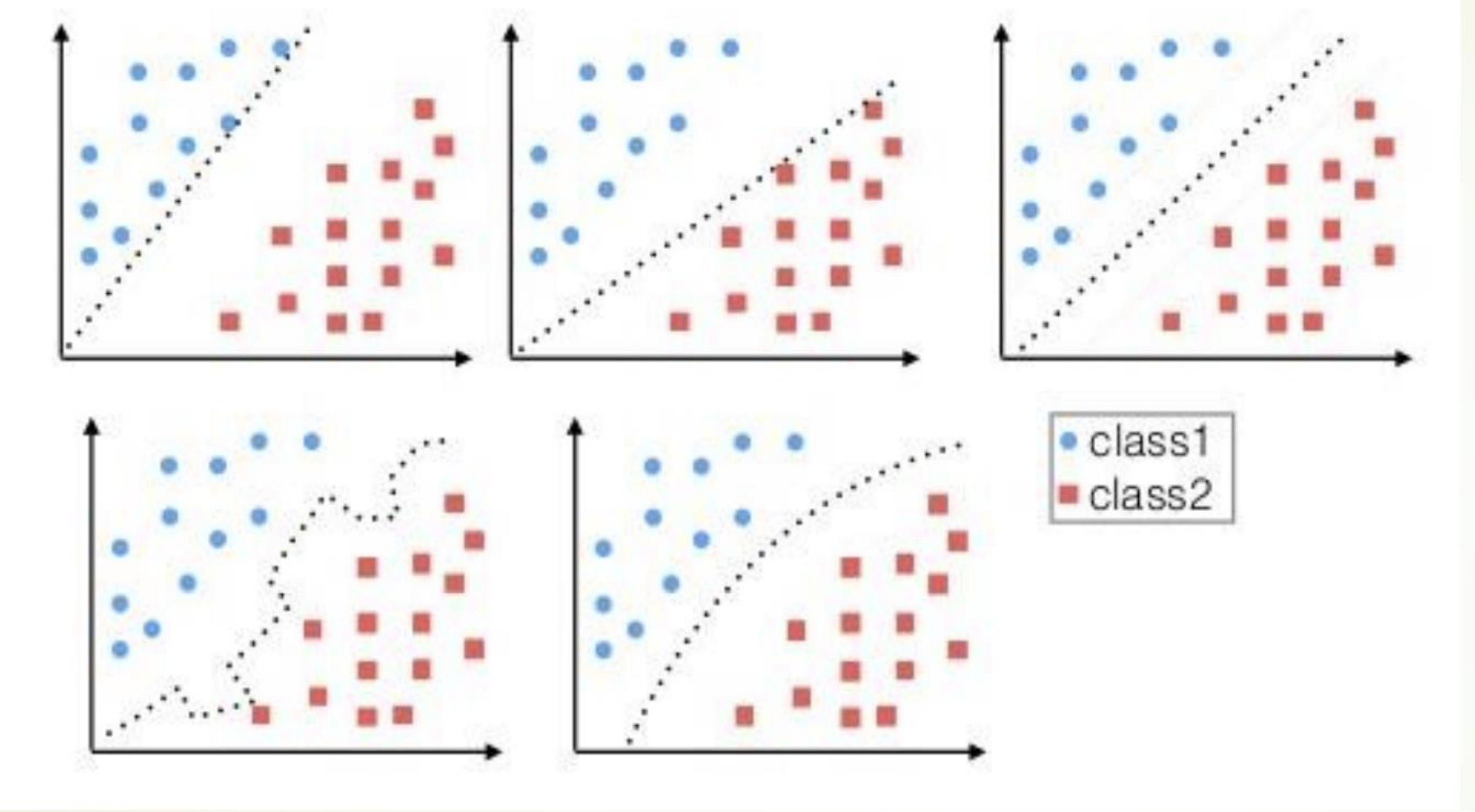


$$h_\theta(x) = f(\theta^T x)$$

For example f = sign:

- If $\theta^T x > 0$ classify "Class 1"
- If $\theta^T x < 0$ classify "Class 0"

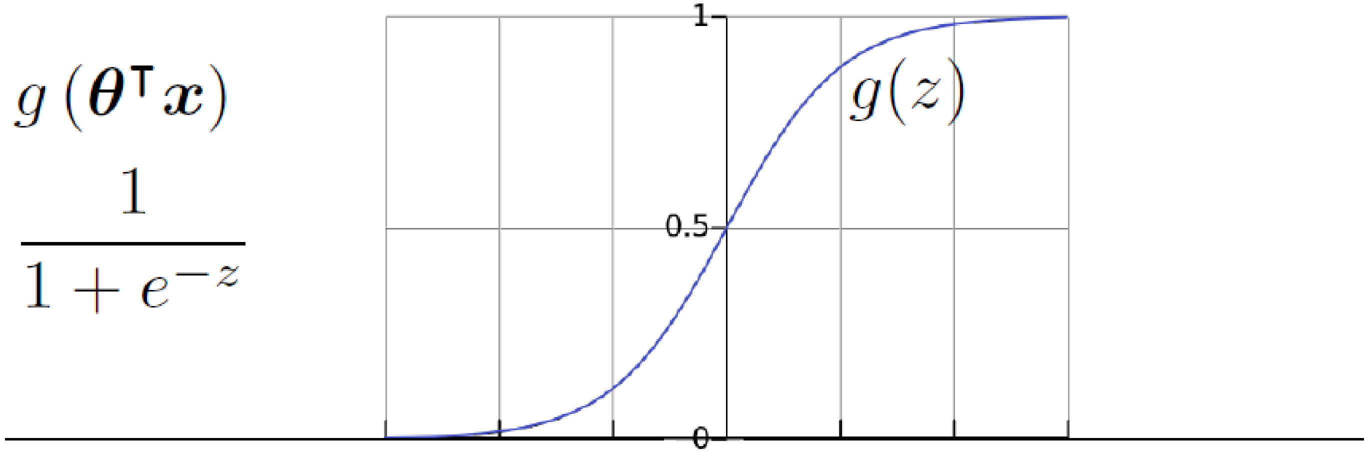
Linear vs Non-Linear Classifiers



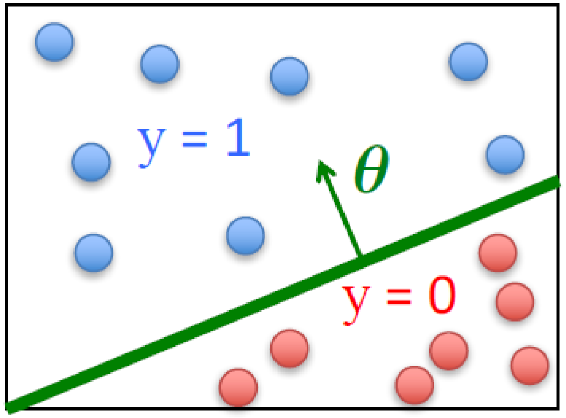
Logistic Regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\intercal}\boldsymbol{x}\right)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



- Assume a threshold and...
  - Predict $Y = 1$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict $Y = 0$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



Logistic Regression is a linear classifier!

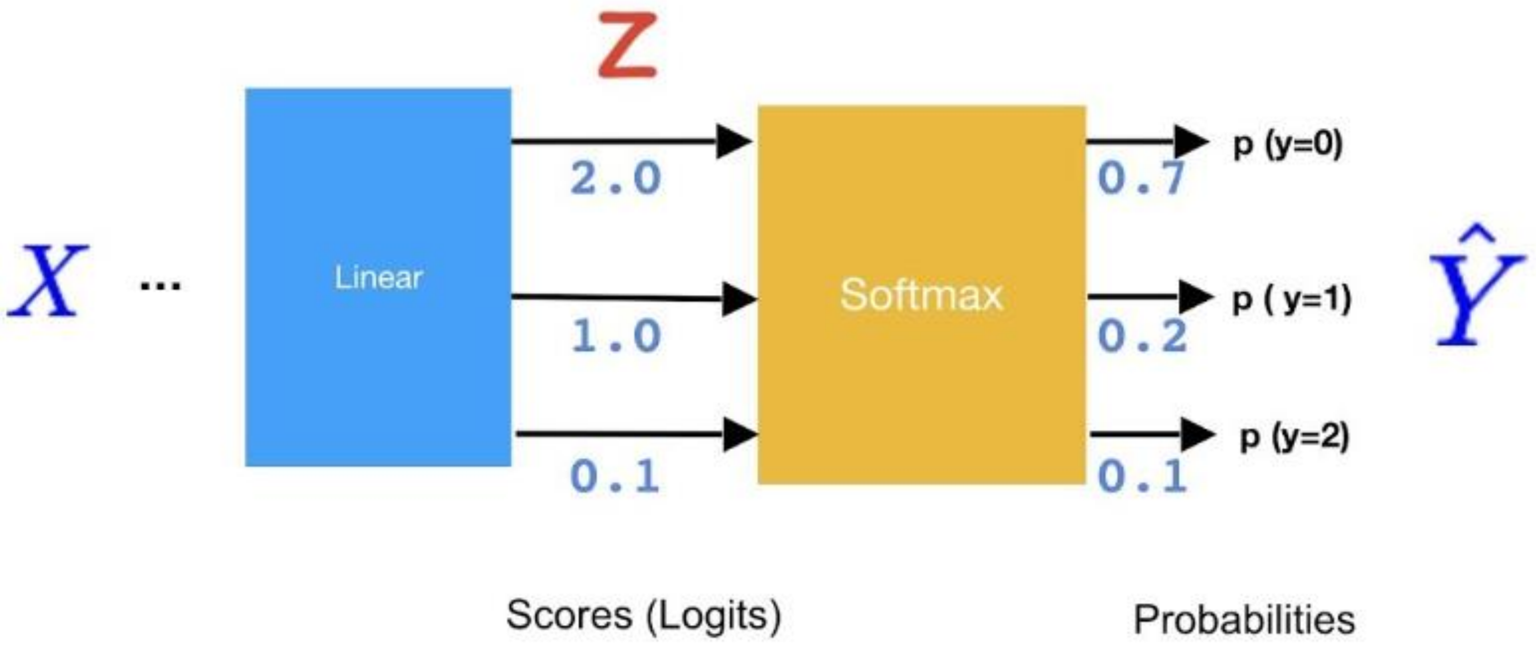
Cross-Entropy Loss

- Standard loss function for binary classification
- Derived from Maximum Likelihood Estimation (MLE)

$$\min_{\theta} J(\theta)$$

$$J(\theta) = -\sum_{i=1}^{N} [y_{\mathrm{i}} \log h_{\theta}(x_i) + (1 - y_i) \log \left(1 - h_{\theta}(x_i)\right)]$$

Softmax classifier



$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

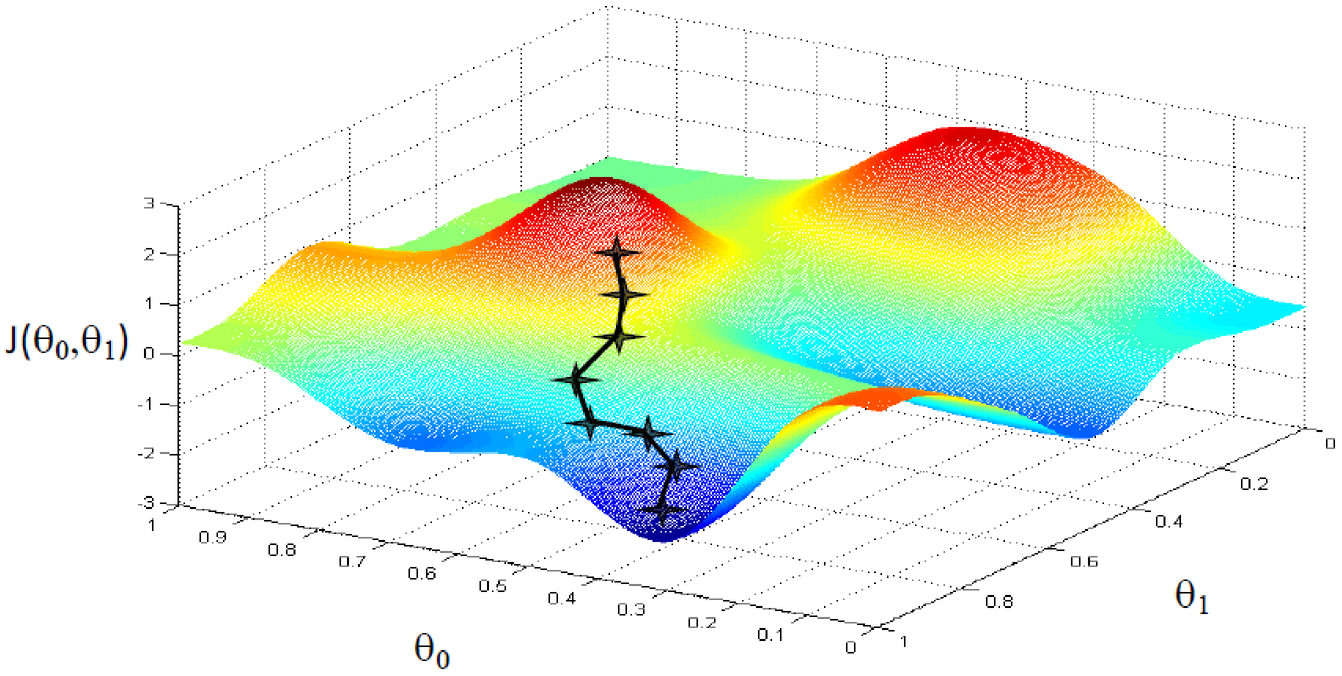
How to Train ML Models?

Goal: find $\boldsymbol{\theta}$ to $\min J(\boldsymbol{\theta})$

Gradient Descent

Goal: find $\boldsymbol{\theta}$ to $\min J(\boldsymbol{\theta})$

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for $\boldsymbol{\theta}$ to reduce $J(\boldsymbol{\theta})$



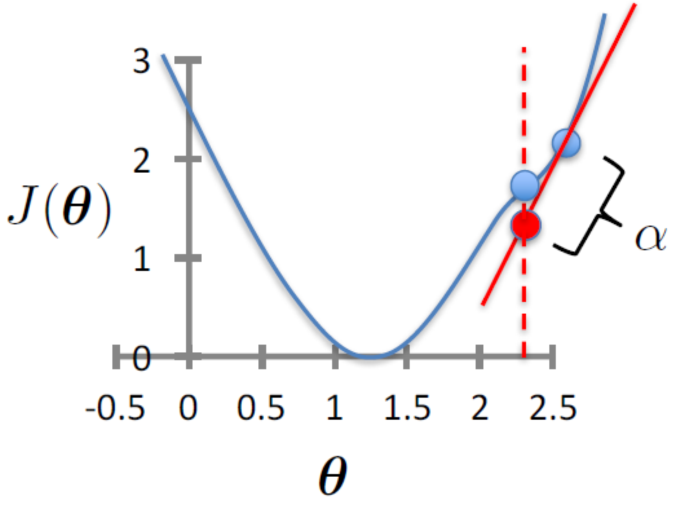
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- Gradient = slope of line tangent to curve
- Function decreases faster in negative direction of gradient

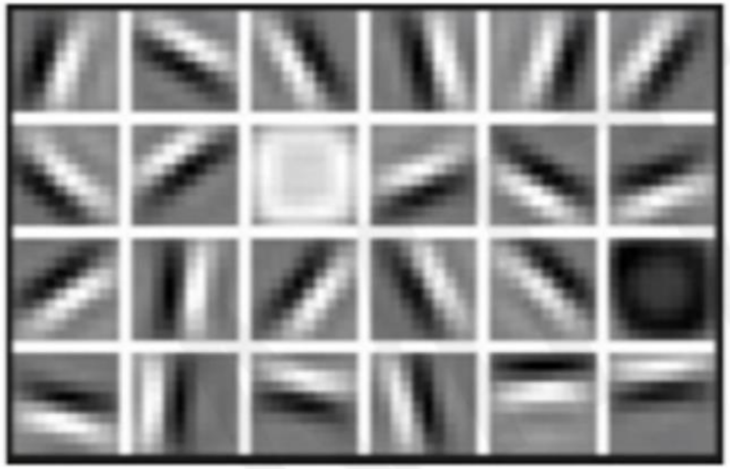
$$\boxed{\text{Vector update rule: } \theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}}$$

Deep Learning: End-to-End Representation Learning

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

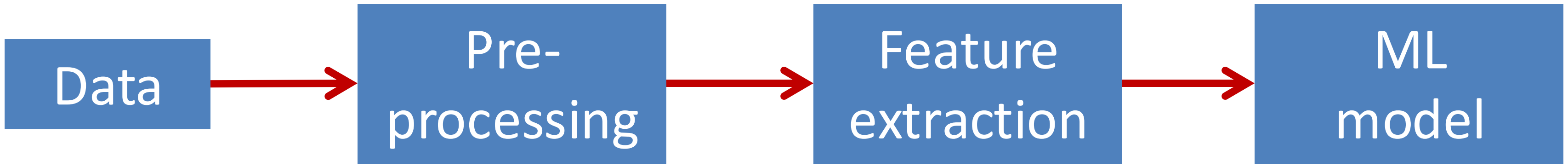
Eyes & Nose & Ears

High Level Features

Facial Structure

Deep Learning

Training



Labeled
x_i
y_i

Normalization

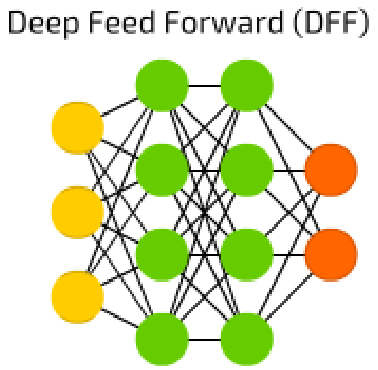
Feature Selection

$h(x)$

Neural Network Architectures

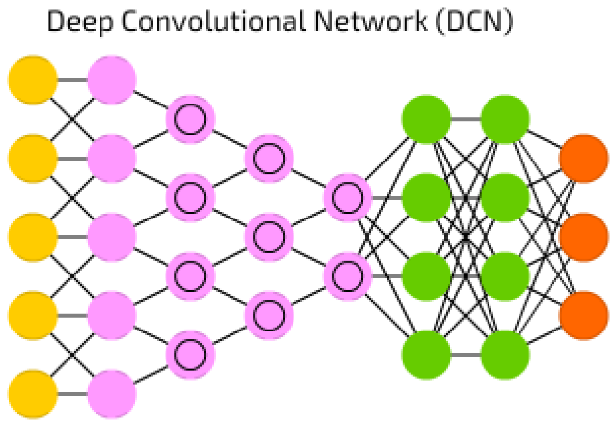
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



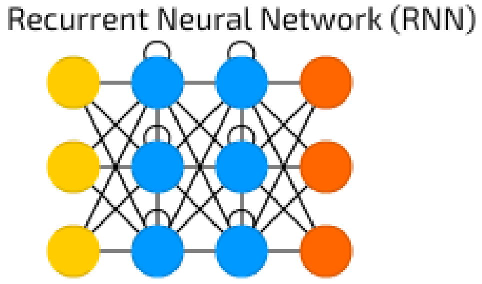
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

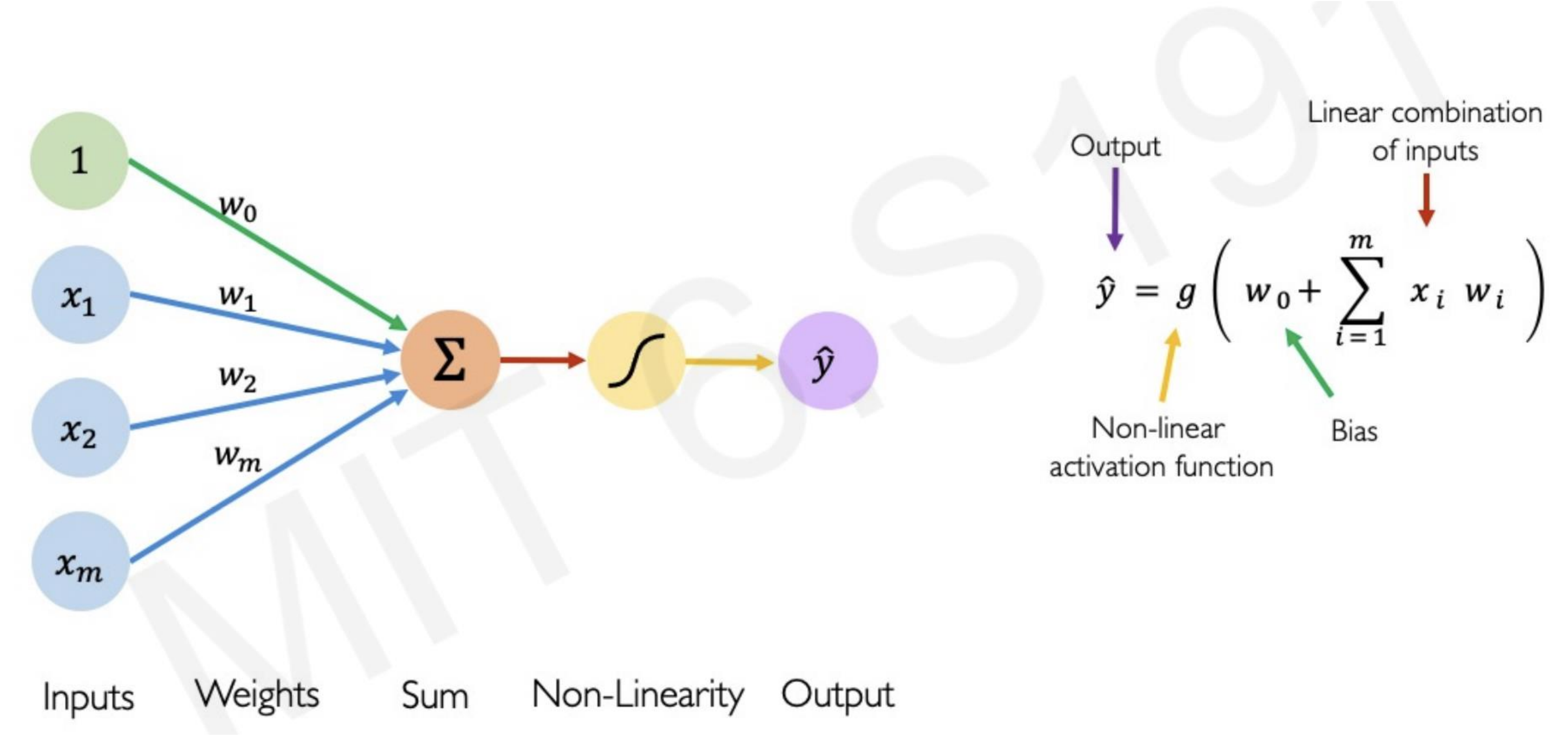


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph

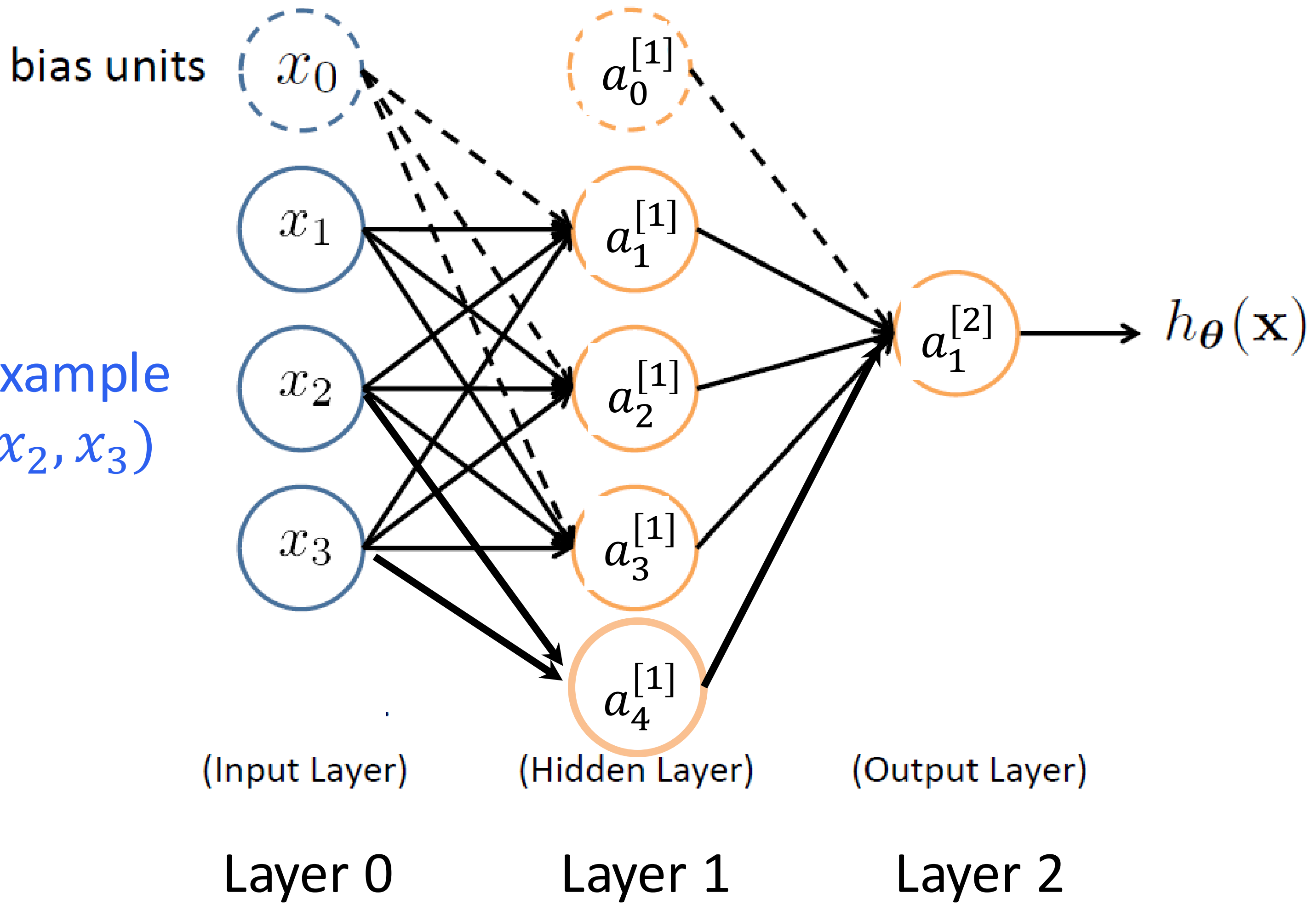


The Perceptron

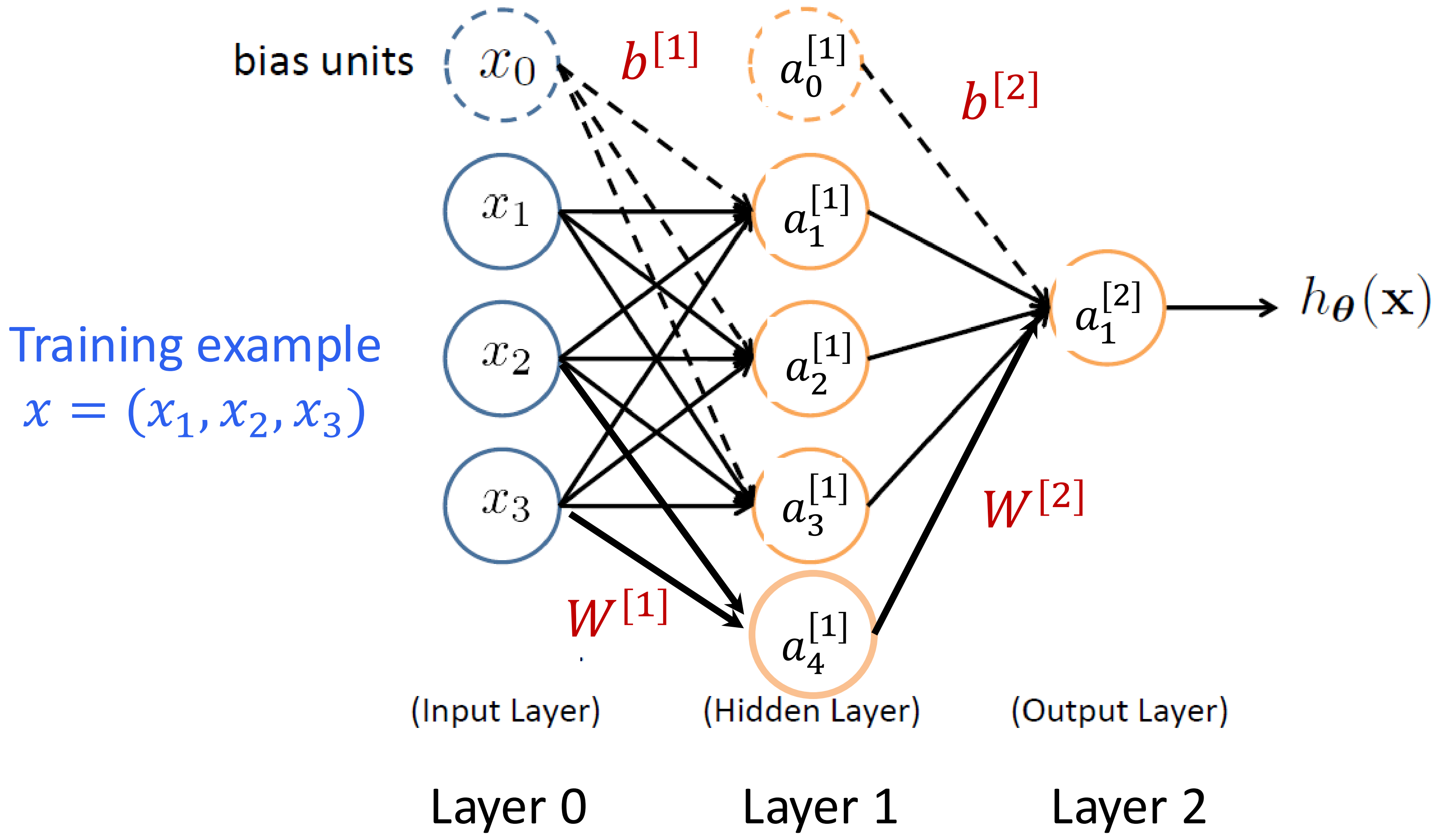


Feed-Forward Neural Network

Training example
$x = (x_1, x_2, x_3)$



Feed-Forward Neural Network



No cycles $\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Layer Operations

$$z_1^{[1]} = W_1^{[1]T} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$z_4^{[1]} = W_4^{[1]T} x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})$$

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} - & W_1^{[1]T} & - \\ - & W_2^{[1]T} & - \\ & \vdots & \\ - & W_4^{[1]T} & - \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Linear

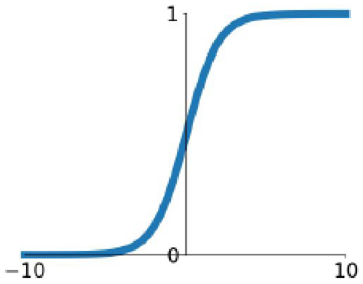
$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Activation Functions

Sigmoid

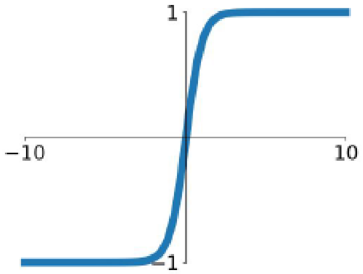
$\sigma(x) = \frac{1}{1+e^{-x}}$



Binary Classification

tanh

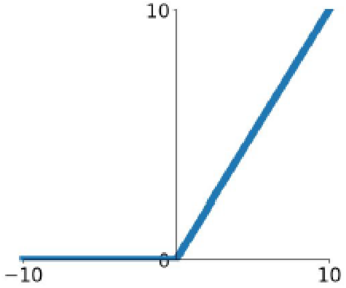
$\tanh(x)$



Regression

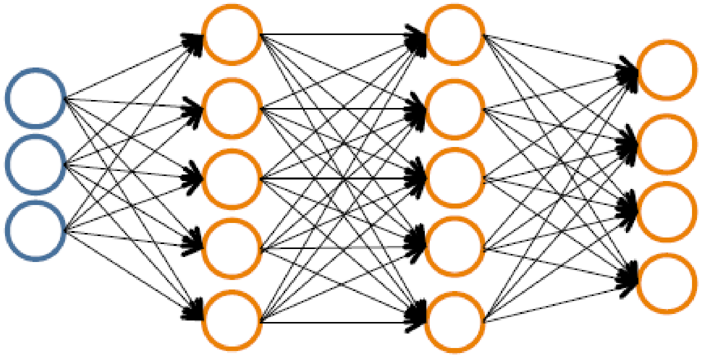
ReLU

$\max(0, x)$



Intermediary layers

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

– $s_0 = d$ (# features)

<u>Binary classification</u>

y = 0 or 1

1 output unit $(s_{L\text{-}1} = 1)$

Sigmoid

<u>Multi-class classification</u> (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L\text{-}1} = K)$

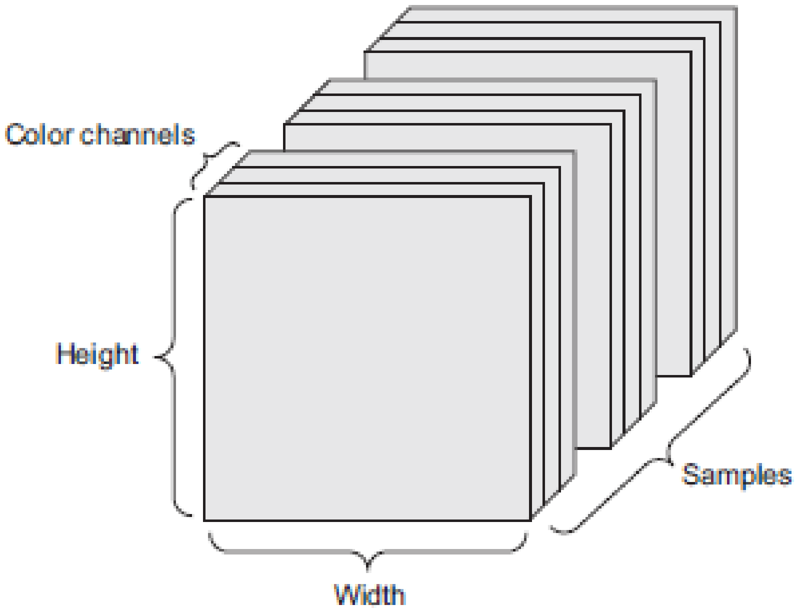
Softmax

Convolutional Nets

- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations for computer vision

Image Representation

- Image is 3D "tensor": height, width, color channel (RGB)
- Black-and-white images are 2D matrices: height, width
  - Each value is pixel intensity



The Convolution Operation

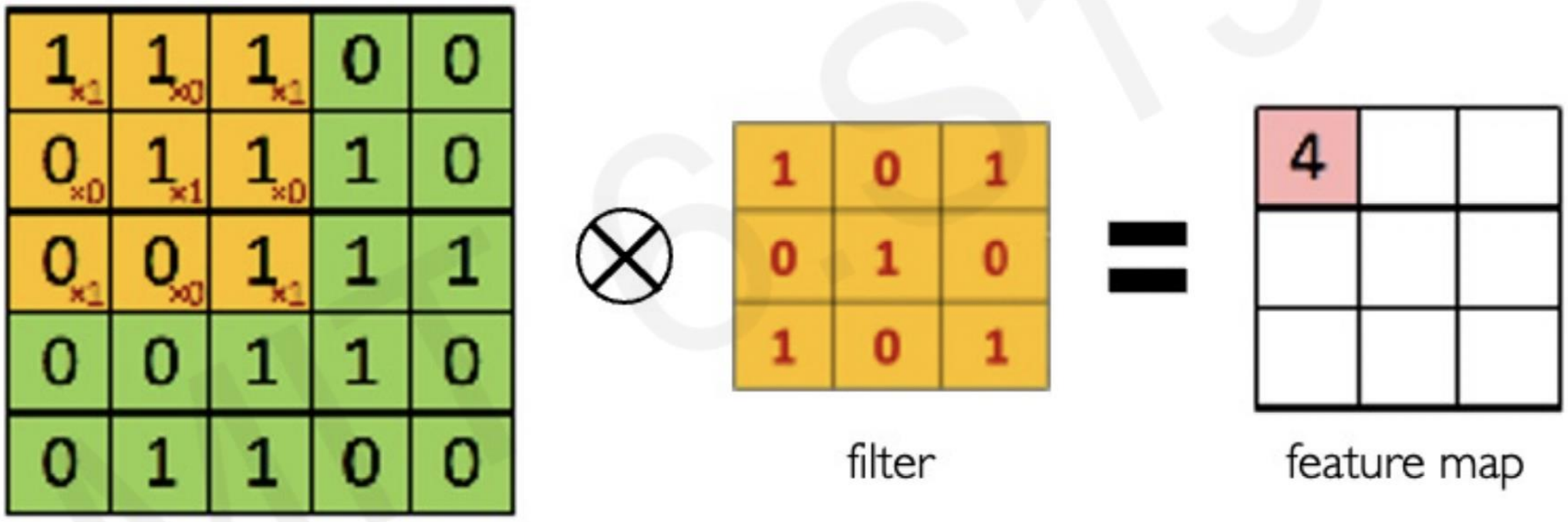
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

$$
\underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}}_{\text{image}} \otimes \underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}_{\text{filter}}
$$

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

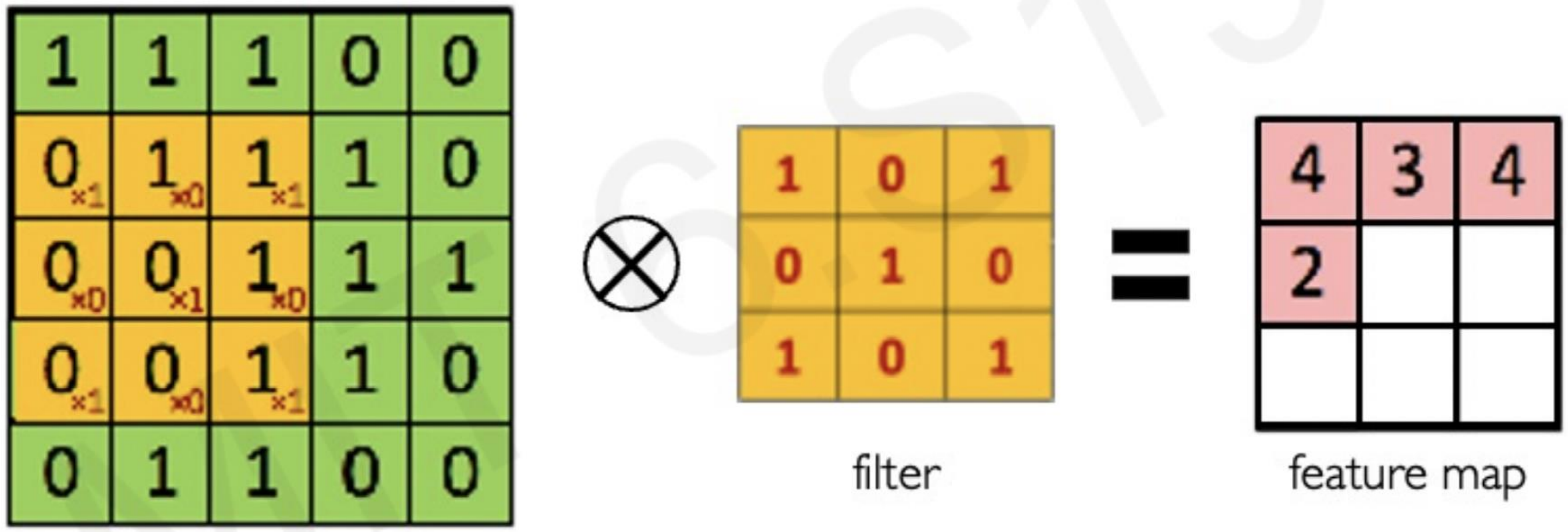
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

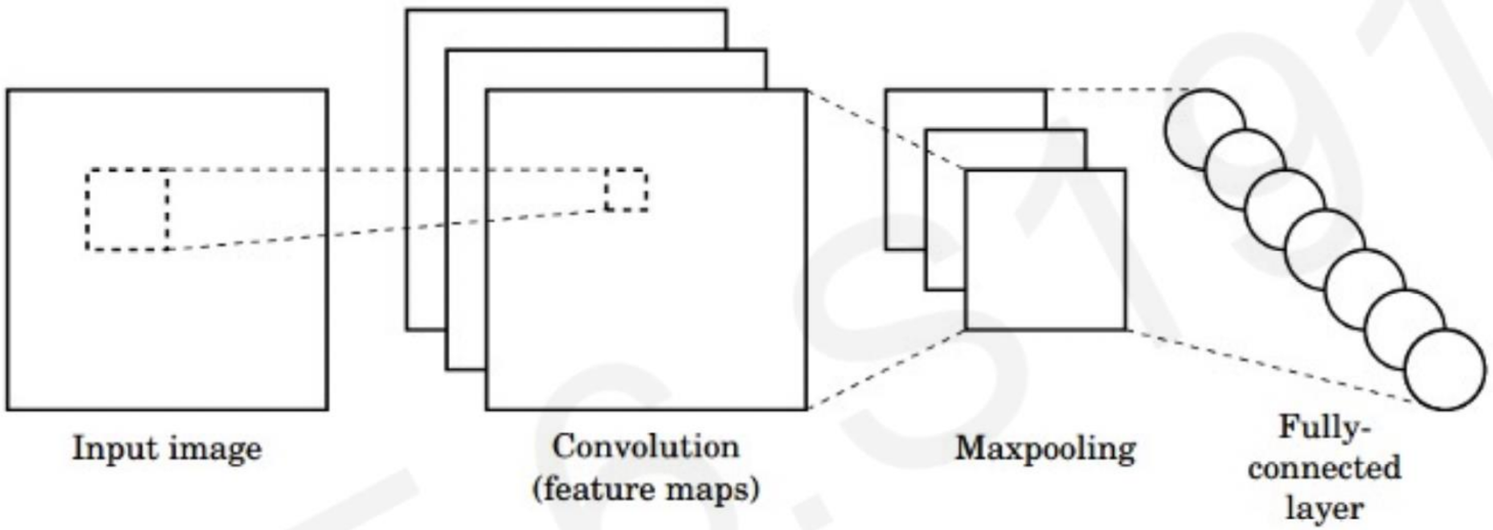


The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



CNNs for Classification



1. **Convolution**: Apply filters to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

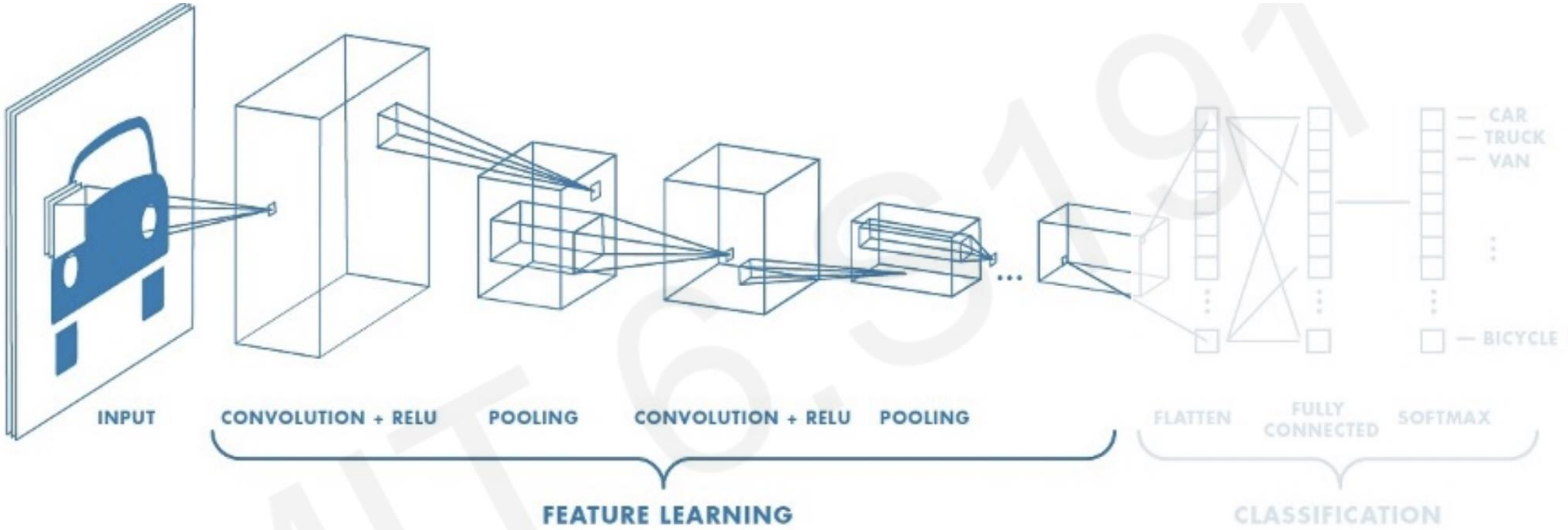
```
tf.keras.layers.Conv2D
```

```
tf.keras.activations.*
```

```
tf.keras.layers.MaxPool2D
```

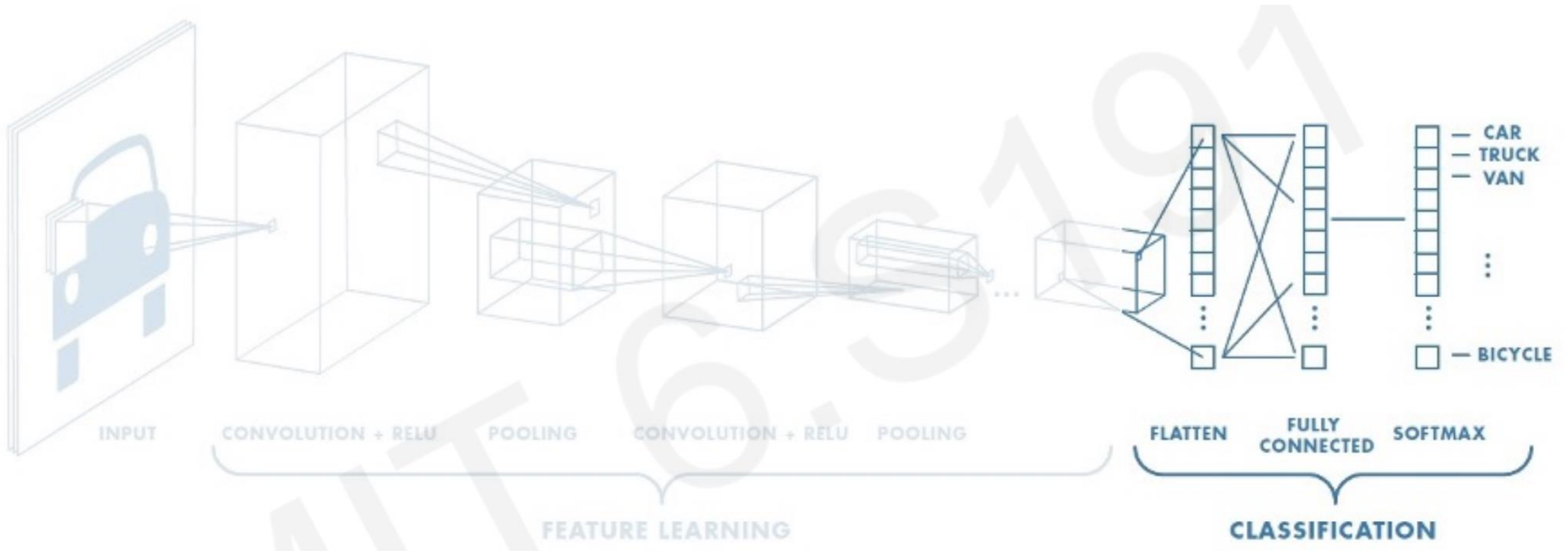
Train model with image data.
Learn weights of filters in convolutional layers.

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities

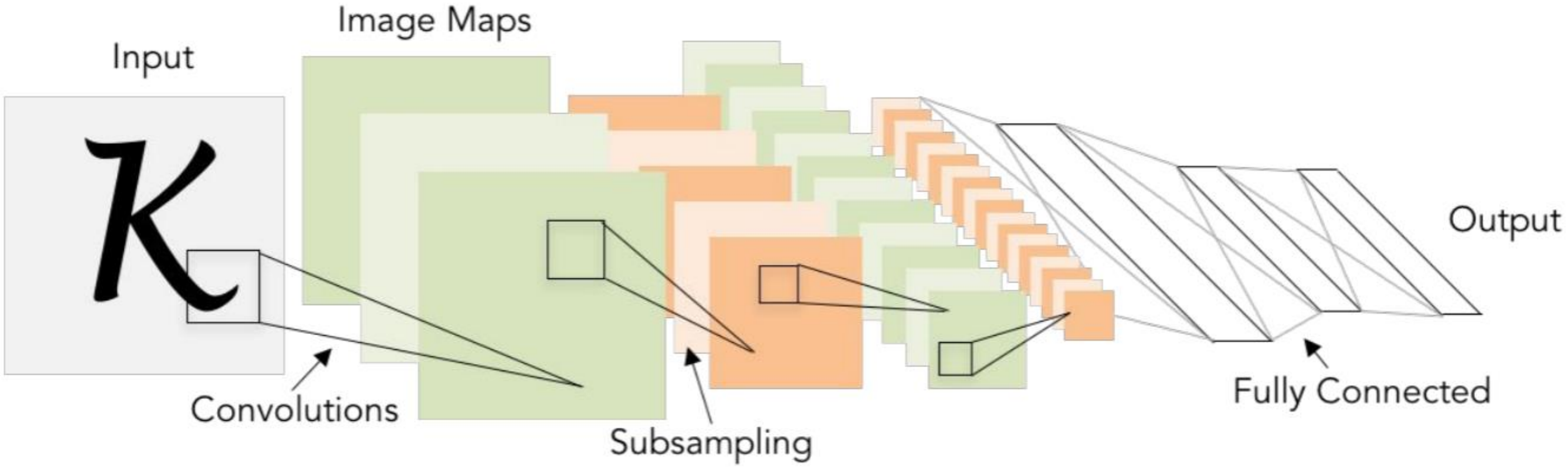


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\mathrm{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

LeNet 5

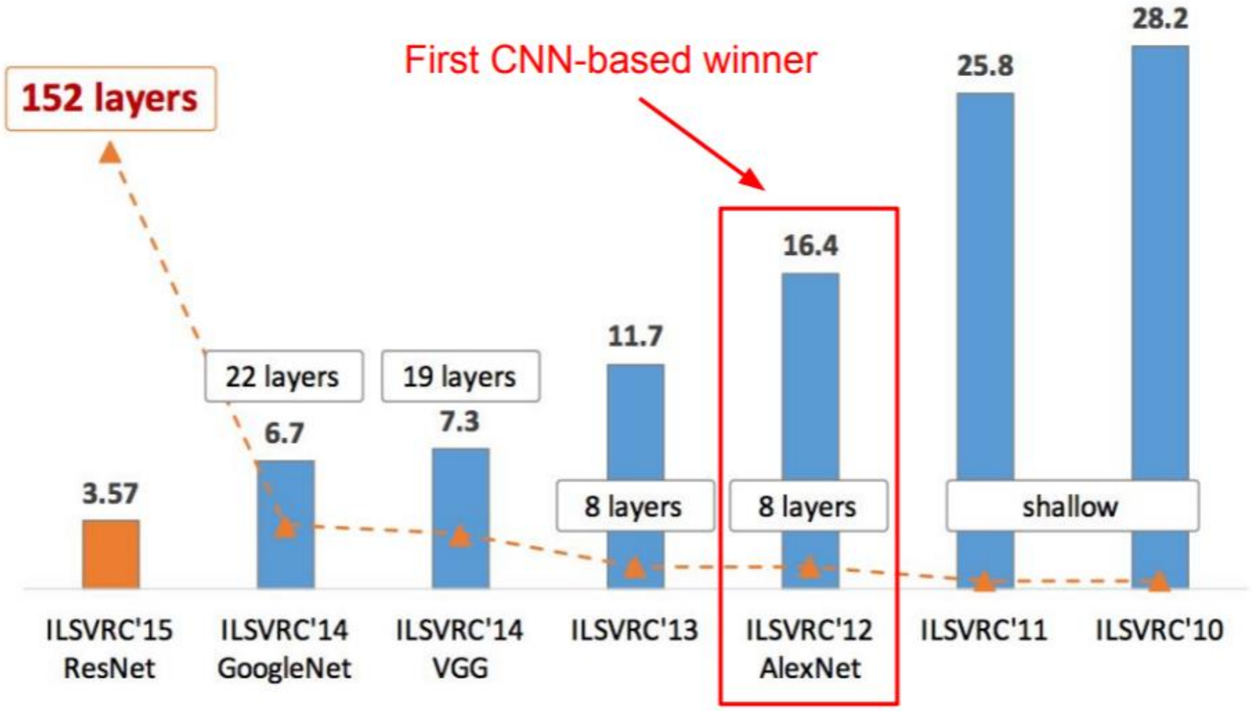
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



How to train Neural Networks?

- Backpropagation algorithm
- David Rumelhart, Geoffrey Hinton, Ronald Williams. "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. 1986
- Applicable to both FFNN and CNN
- Extension of Gradient Descent to multi-layer neural networks

Training Neural Networks

- Training data $x_1, y_1, \dots x_N, y_N$
- One training example $x_i = (x_{i1}, \dots x_{id})$, label y_i
- One forward pass through the network
  - Compute prediction $\widehat{y}_i = h(x_i)$
- Loss function for one example
  - $L(\hat{y}, y) = -[(1 - y)\log(1 - \hat{y}) + y \log \hat{y}]$

Cross-entropy loss

- Loss function for training data
  - $J(W, b) = \frac{1}{N} \sum_i L\,(\widehat{y}_i \;\;, y_i)$

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Initialize $W^{[\ell]}, b^{[\ell]}$
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
    - $W^{[\ell]} = \boxed{W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}}$
    - $b^{[\ell]} = \boxed{b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}}$

This is expensive!

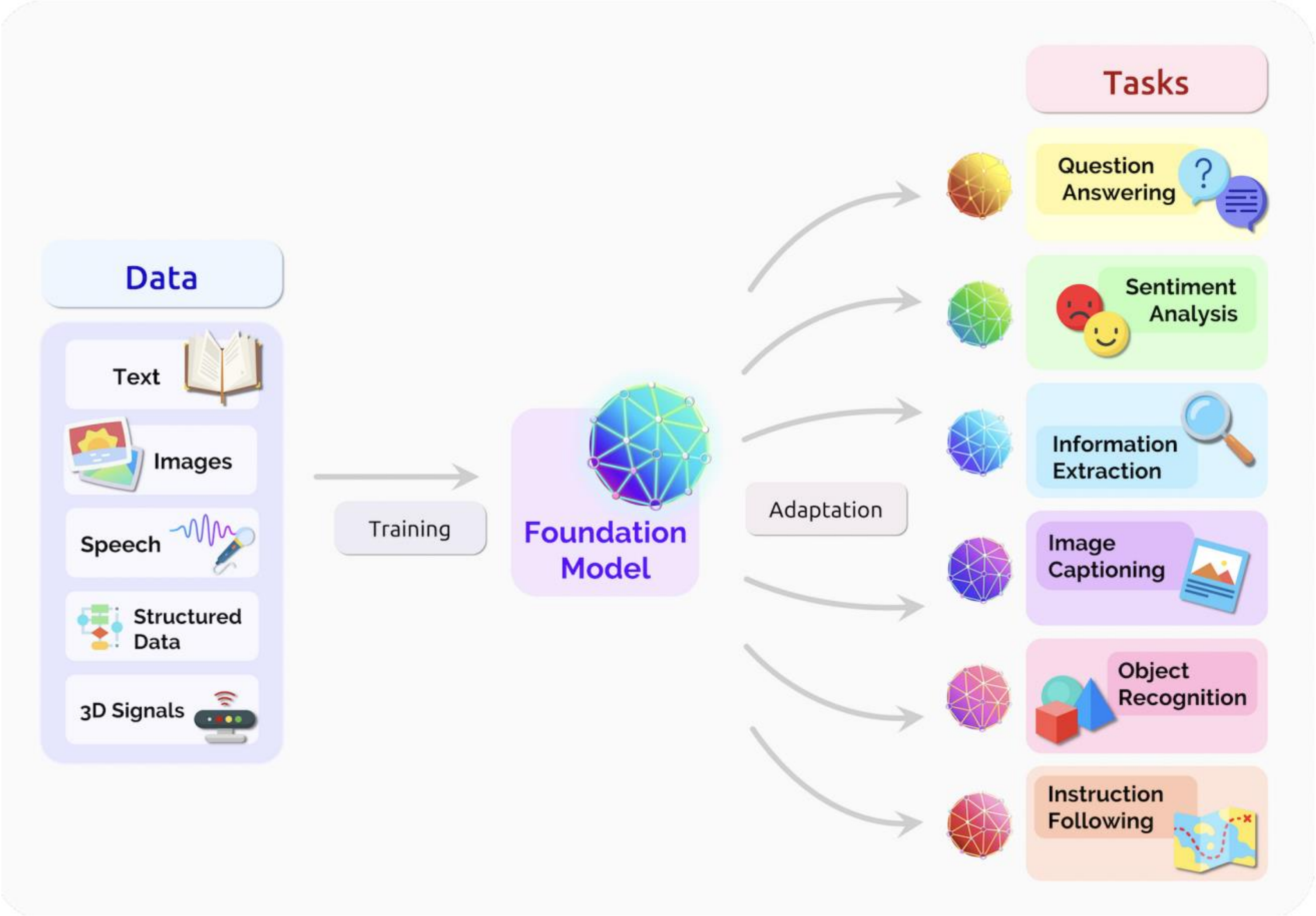
Mini-batch Stochastic Gradient Descent

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - Repeat
    - For all layers ℓ (starting backwards)
      - For all batches b of size B with training examples x_{ib}, y_{ib}

$$W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$$

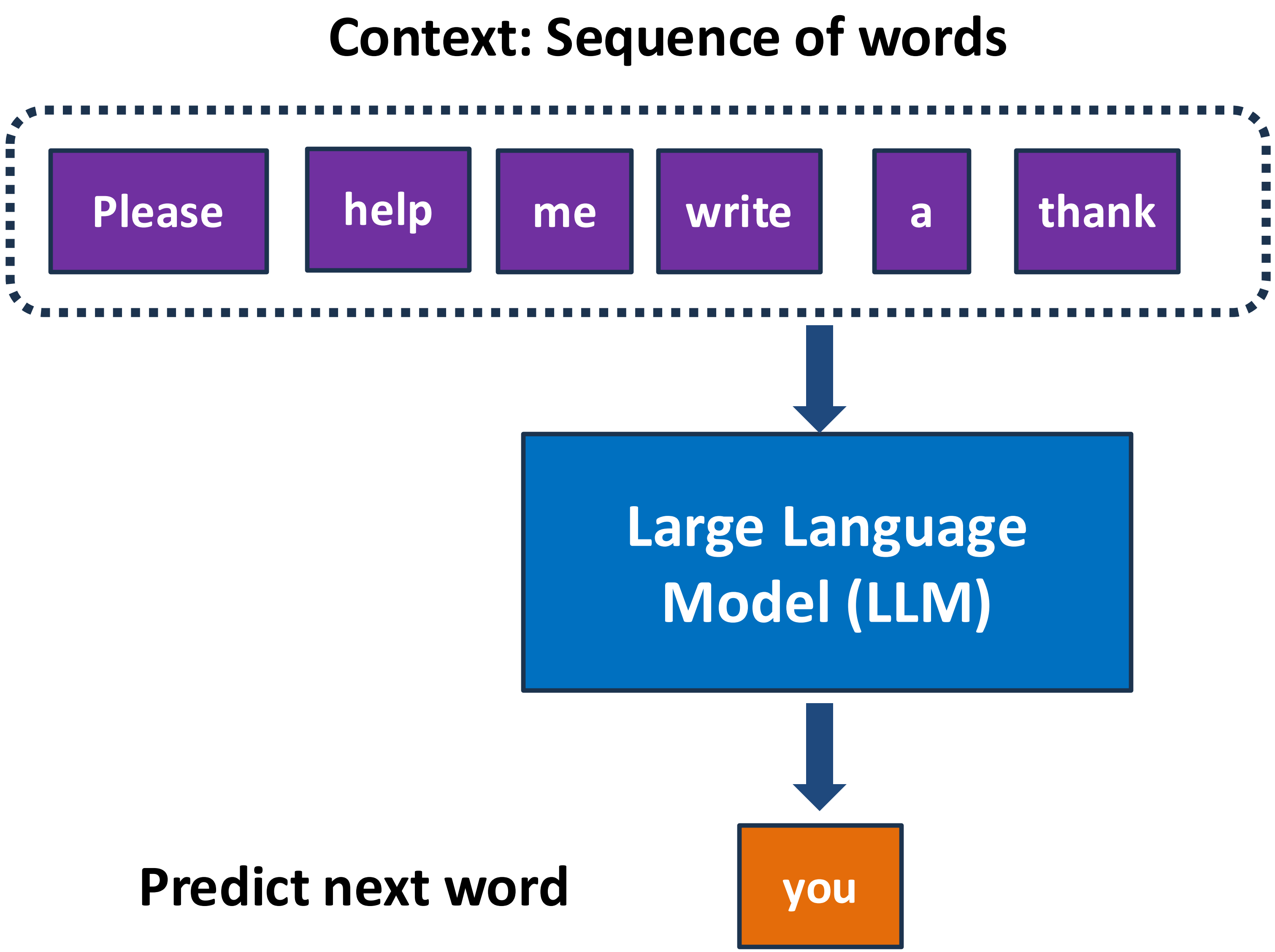
$$b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial b^{[\ell]}}$$

New Trend in AI: Foundation Models

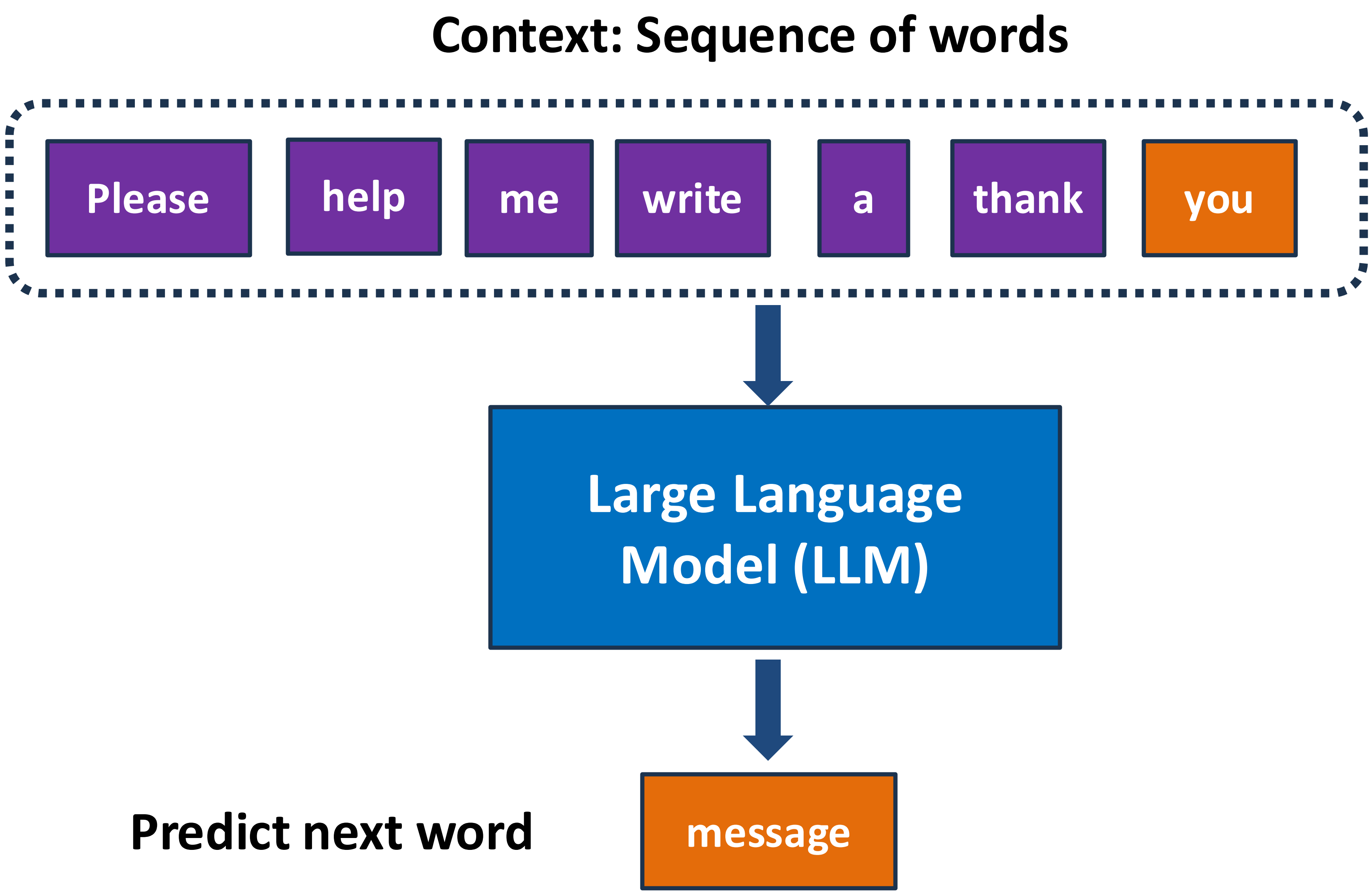


On the Opportunities and Risks of Foundation Models

Training LLMs

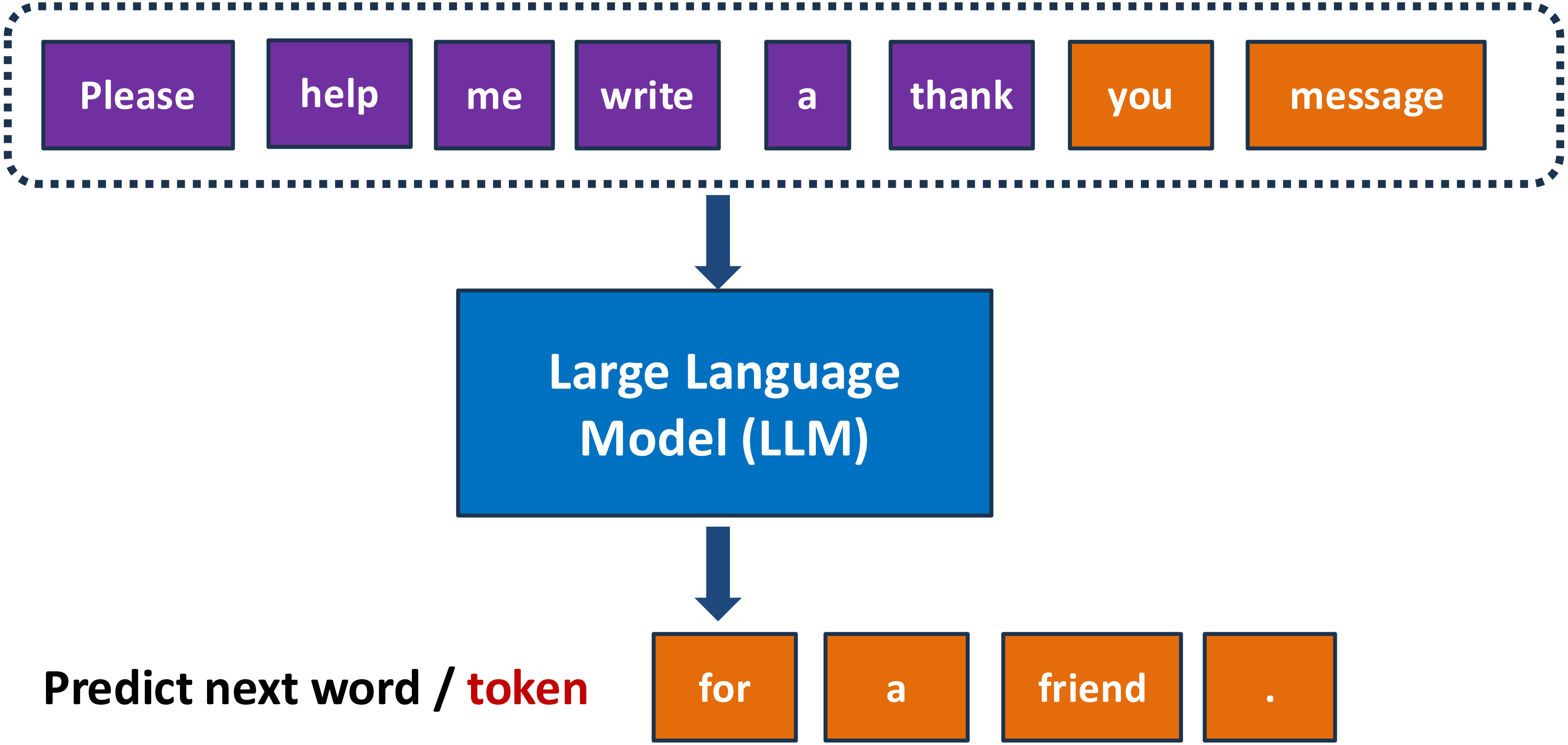


Training LLMs

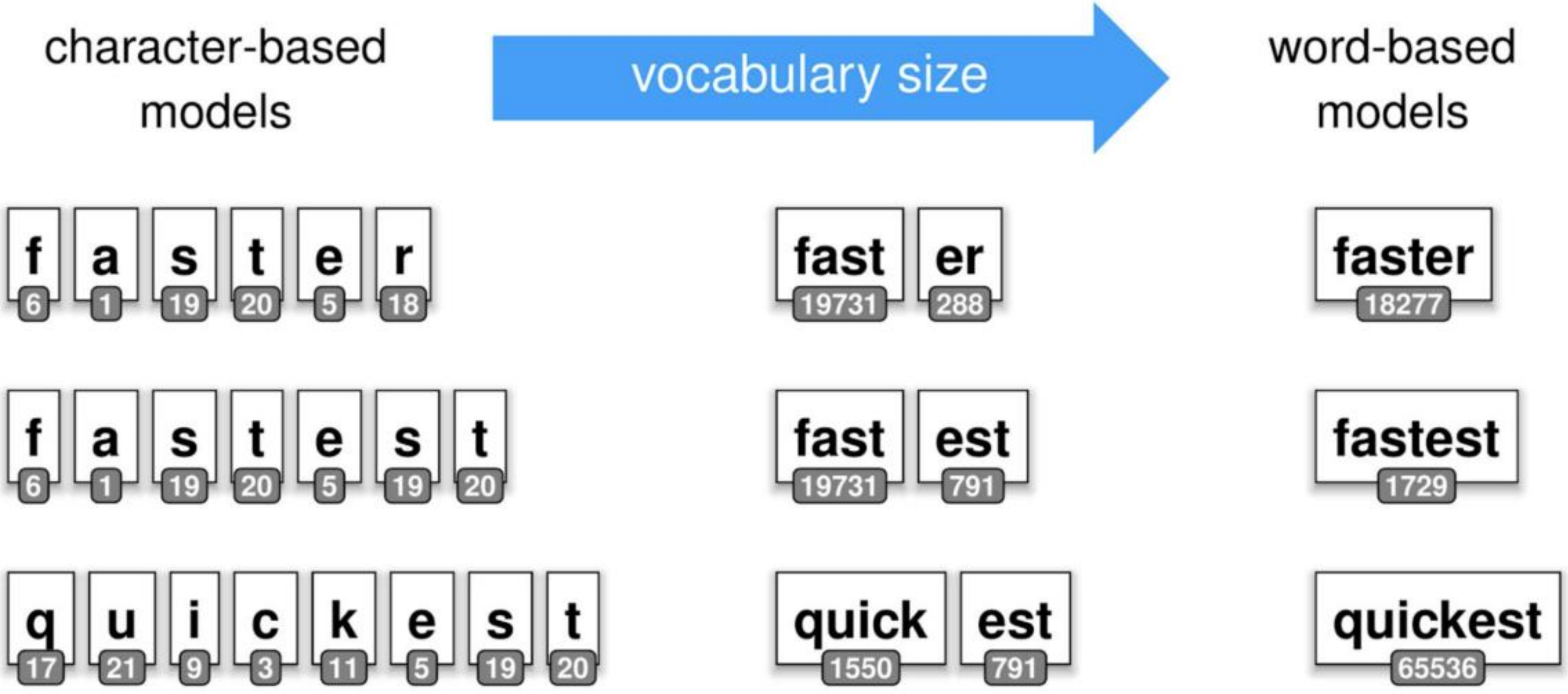


Training LLMs

Context: Fixed-size Sequence of words / tokens



Tokenization



Autoregressive Language Models

x_1 x_{i-1} x_i

Please help me write a thank you

Training Objective

Maximize likelihood of given context

Autoregressive Language Models

x_1					x_{i-1}	x_i
Please	help	me	write	a	thank	you

Training Objective

Maximize likelihood of given context

$$\textbf{max} \quad \prod_{i=1}^{n} p_\theta(\boxed{x_i} \mid \boxed{x_1, \ldots, x_{i-1}})$$

Likelihood

Previous Tokens

Autoregressive Language Models

x_1					x_{i-1}	x_i
Please	help	me	write	a	thank	you

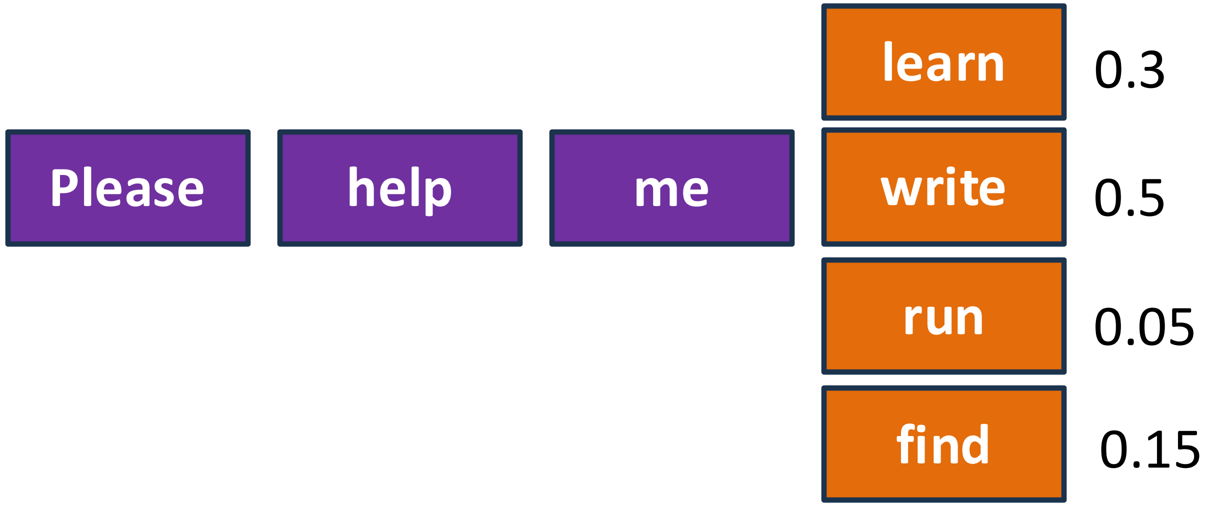
Training Objective Maximize likelihood of ▭ given context ▭ ▭ ▭

$$\textbf{min}\ \ L(\boldsymbol{\theta}) = \textbf{exp}\ [\ -\log \Pi_{i=1}^{n}\, p_\theta(\boxed{x_i} \mid \underbrace{\boxed{x_1,\ldots,x_{i-1}}}_{\text{Previous Tokens}})\ ]$$

Perplexity loss function

- Usually, when training a model, we minimize loss function
- Loss function is **perplexity**: exp of **negative log likelihood** of tokens given context
- Train with backpropagation

Generating Text

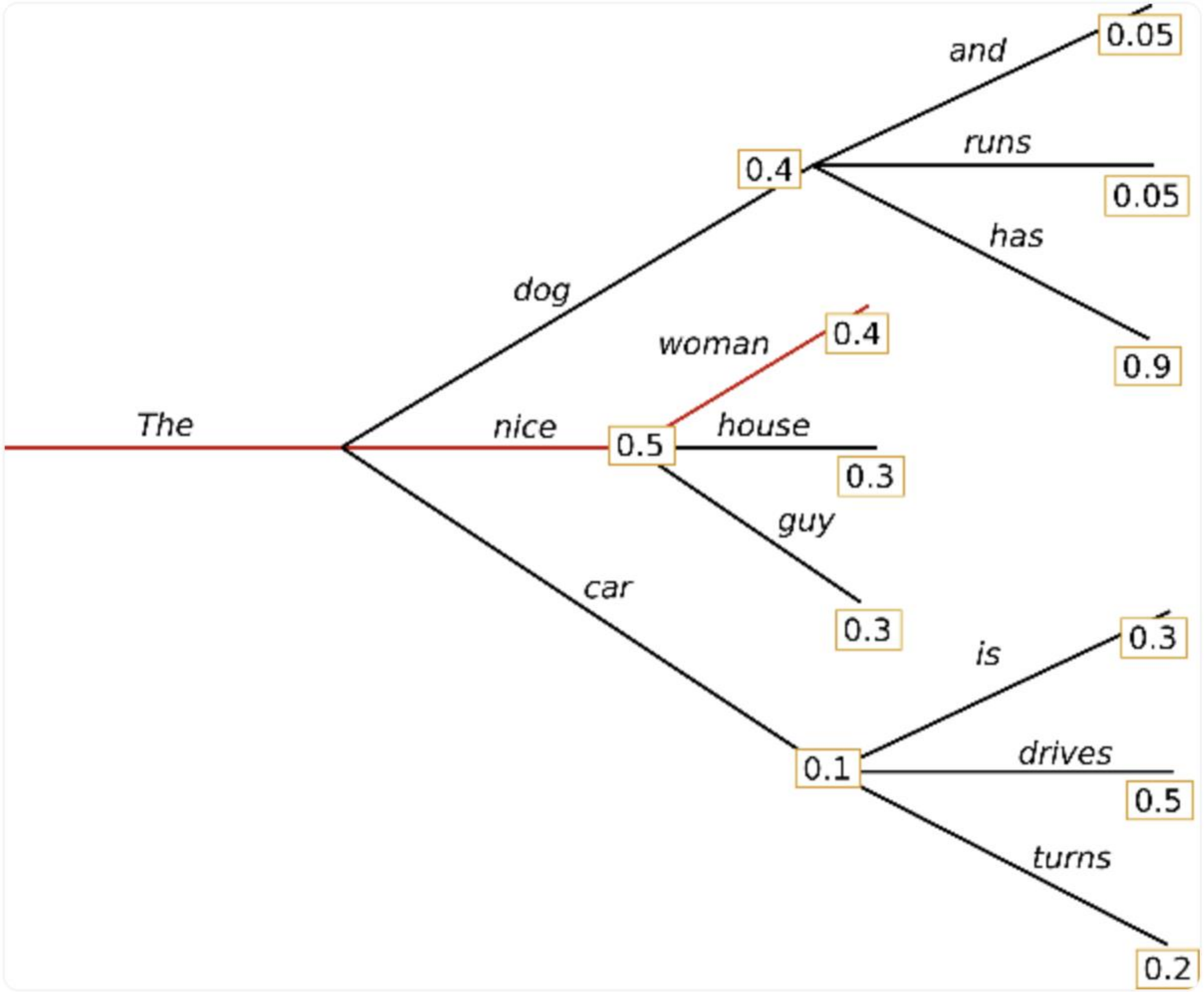


In training, model learns probability distribution of next token given context

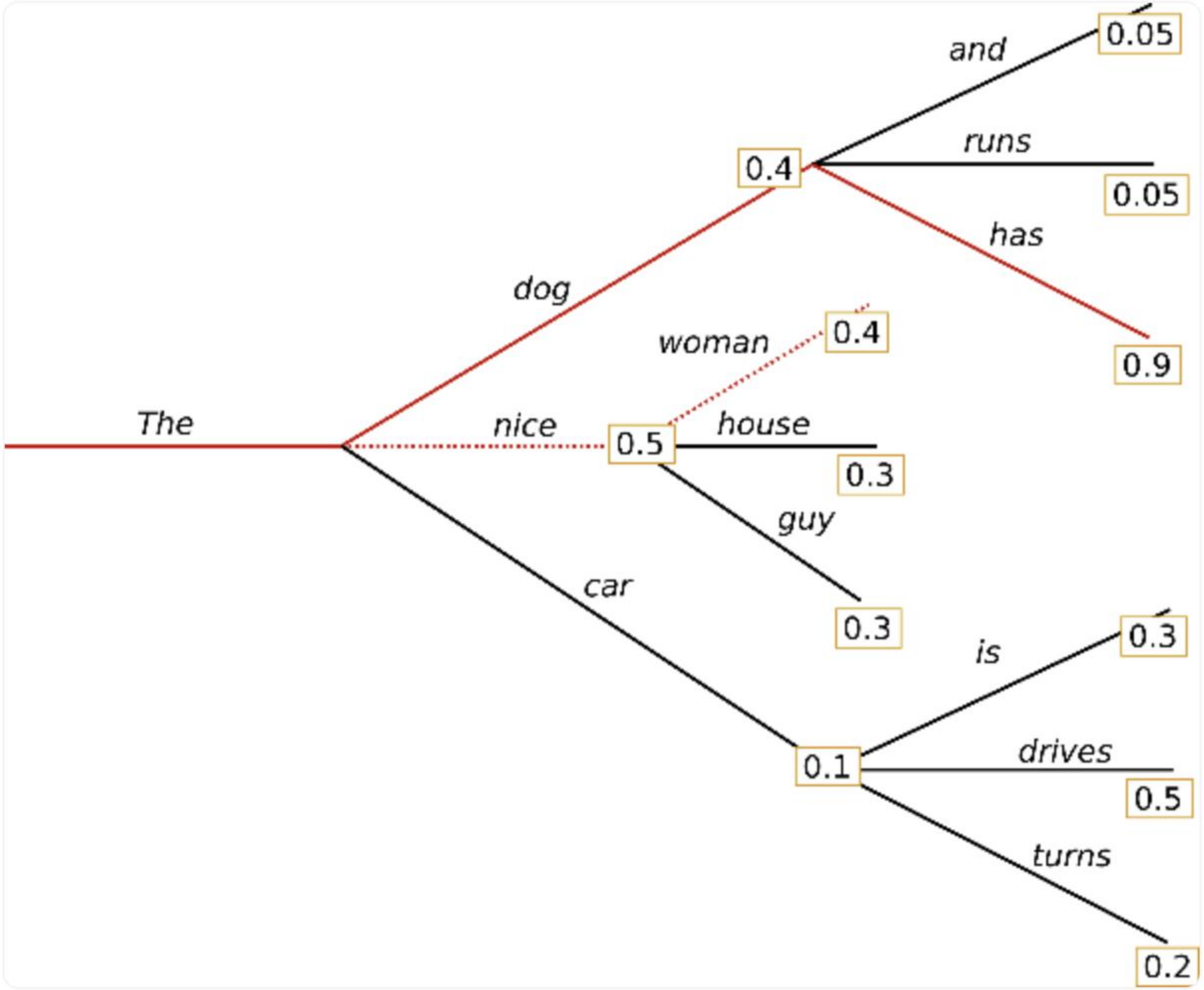
Sampling strategies for next token generation:

- **Greedy**: Get the most probable token (has low diversity)
- **Top-k**: Sample from top-k with highest likelihood by re-normalizing the probabilities
- **Beam search**: Keep a number of most likely sequences and then select highest probability

Greedy search

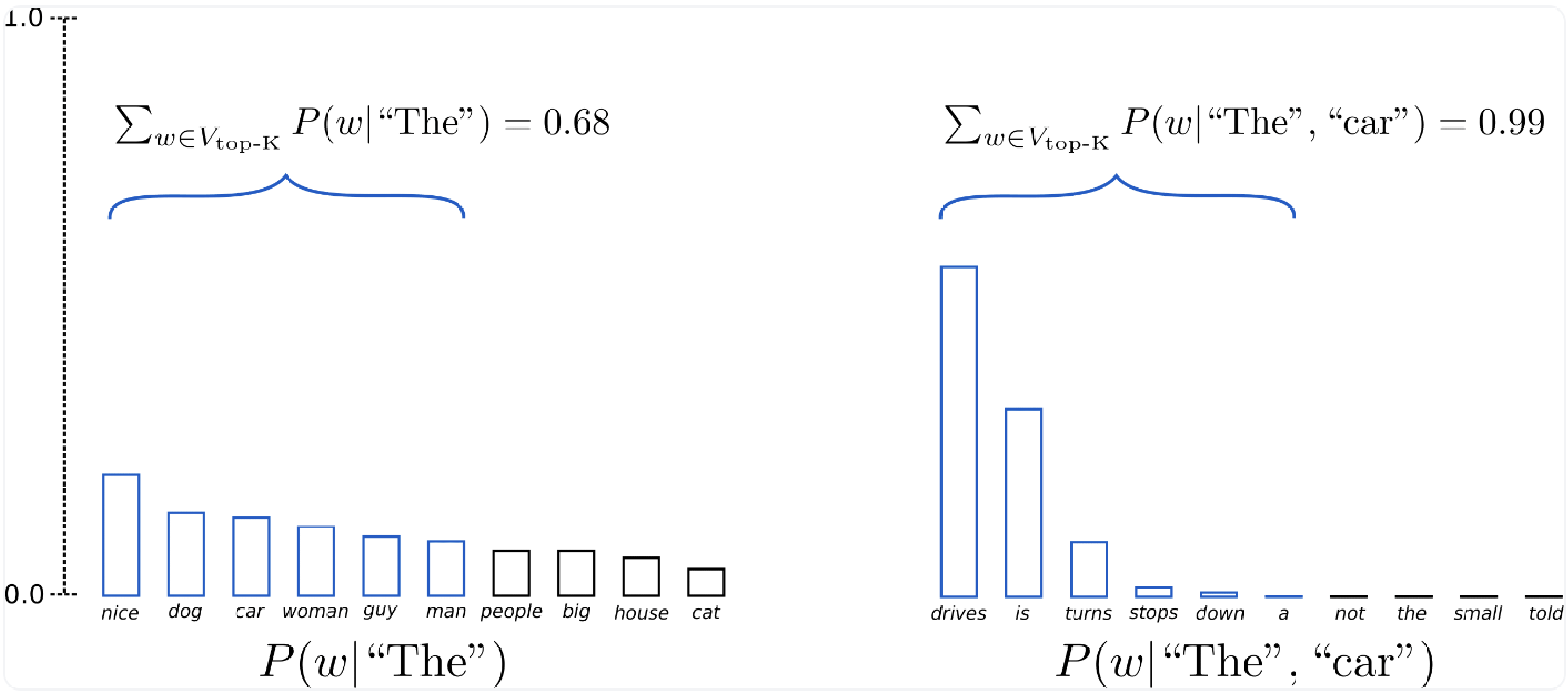


Beam search



The
dog
nice
car
0.4
0.5
0.1
and
0.05
runs
0.05
has
0.9
woman
0.4
house
0.3
guy
0.3
is
0.3
drives
0.5
turns
0.2

Top-k Sampling



LLM Architectures

LLMs are built out of transformers

Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

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

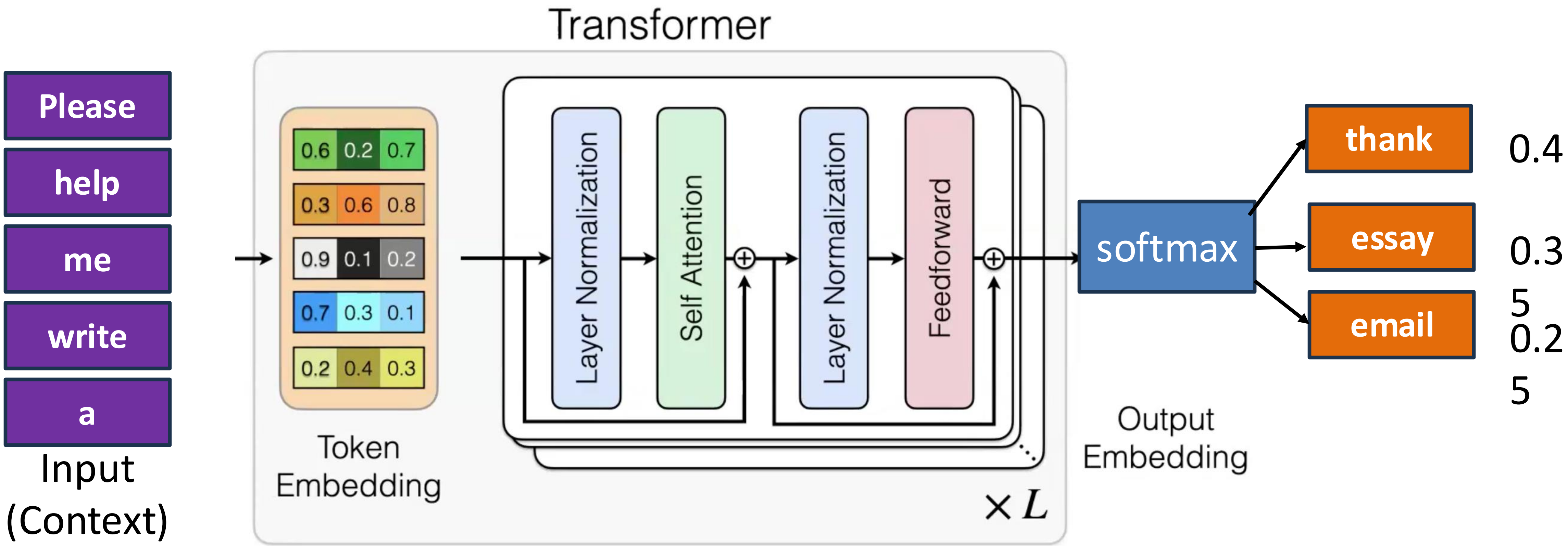
Llion Jones* Google Research llion@google.com

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

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

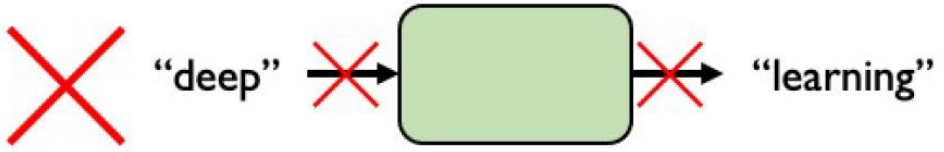
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer Models

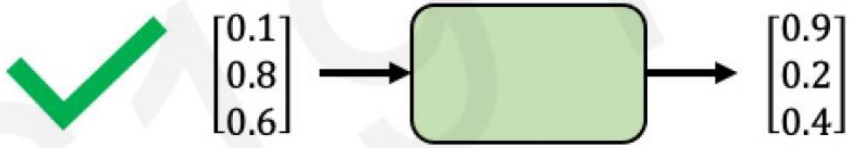


- **Token embedding**: Represent tokens as numerical vectors
- **Self-Attention**: Automatically learn which tokens in context are most relevant
- **Training**: Standard backpropagation to learn probability of next token

Encoding Language

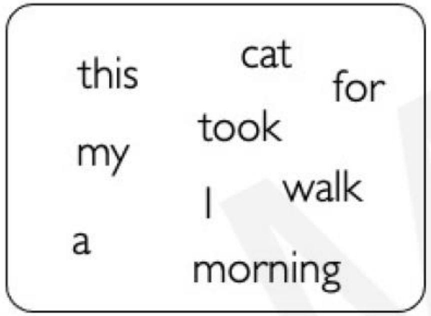


Neural networks cannot interpret words

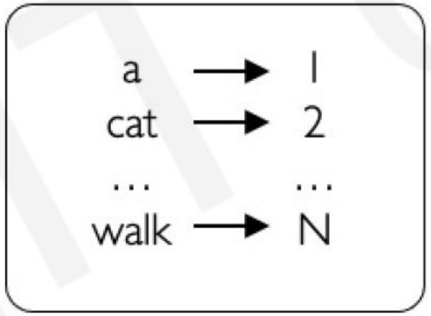


Neural networks require numerical inputs

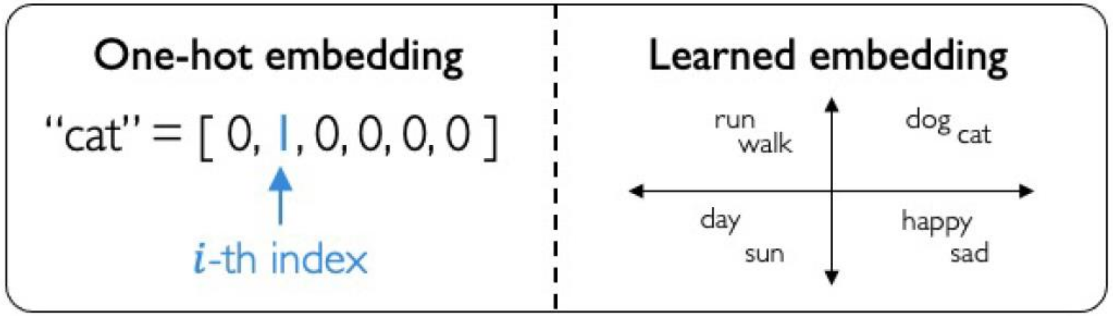
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



3. Embedding:
Index to fixed-sized vector

Problem with static embeddings (word2vec)

They are static! The embedding for a word doesn't reflect how its meaning changes in context.

`The chicken didn't cross the street because it was too tired`

What is the meaning represented in the static embedding for "it"?

Contextual Embeddings

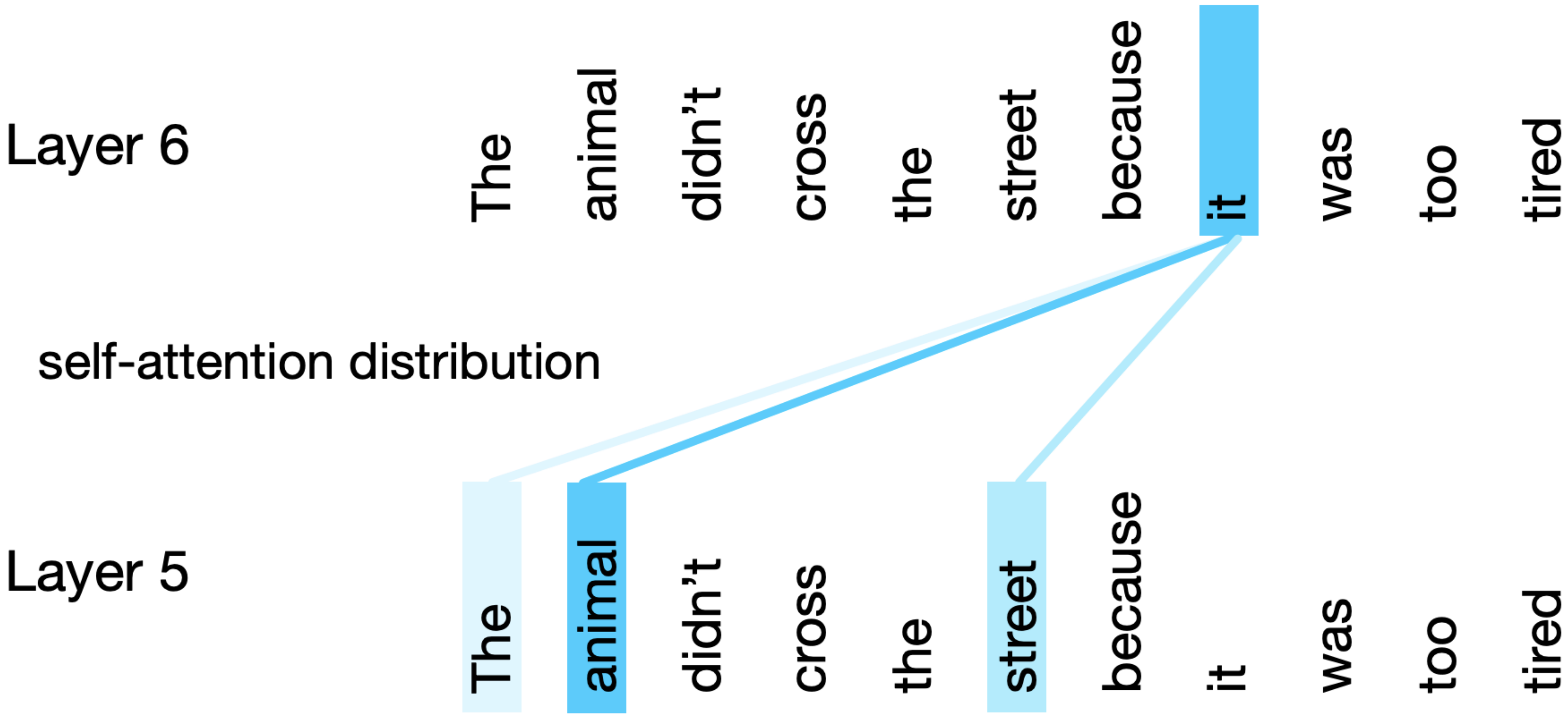
- Intuition: a representation of meaning of a word should be different in different contexts!
- **Contextual Embedding**: each word has a different vector that expresses different meanings depending on the surrounding words
- How to compute contextual embeddings?
  - **Attention**

Intuition of Attention

Build up the contextual embedding from a word by selectively integrating information from all the neighboring words

We say that a word "attends to" some neighboring words more than others

Attention


Layer 6
The
animal
didn't
cross
the
street
because
it
was
too
tired
self-attention distribution
Layer 5
The
animal
didn't
cross
the
street
because
it
was
too
tired

Attention Definition

Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_{\mathrm{i}}$$

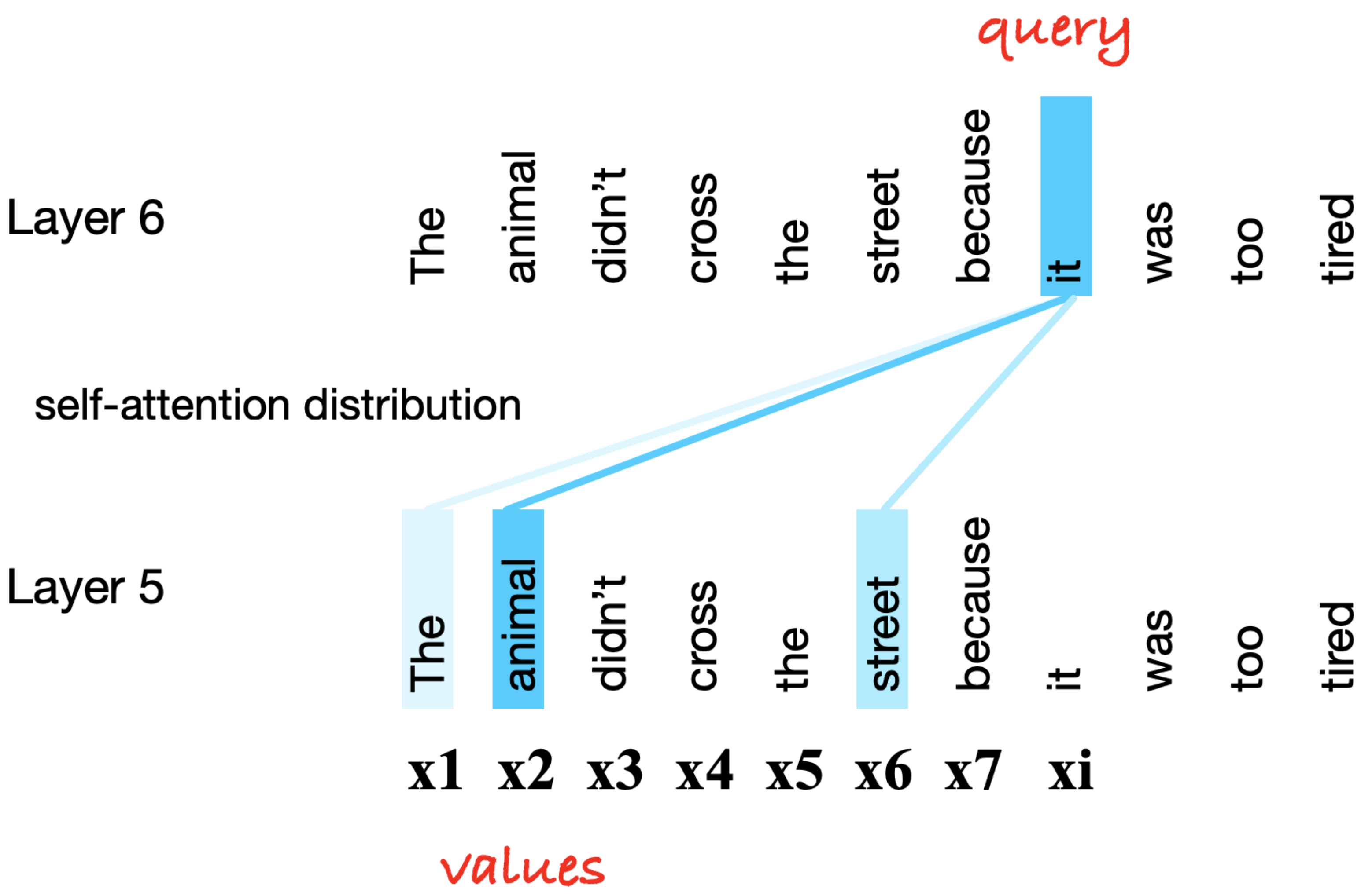
Produce: $\mathbf{a}_{\mathrm{i}}$ = a weighted sum of $\mathbf{x}_1$ through $\mathbf{x}_5$
Weighted by their similarity to $\mathbf{x}_{\mathrm{i}}$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

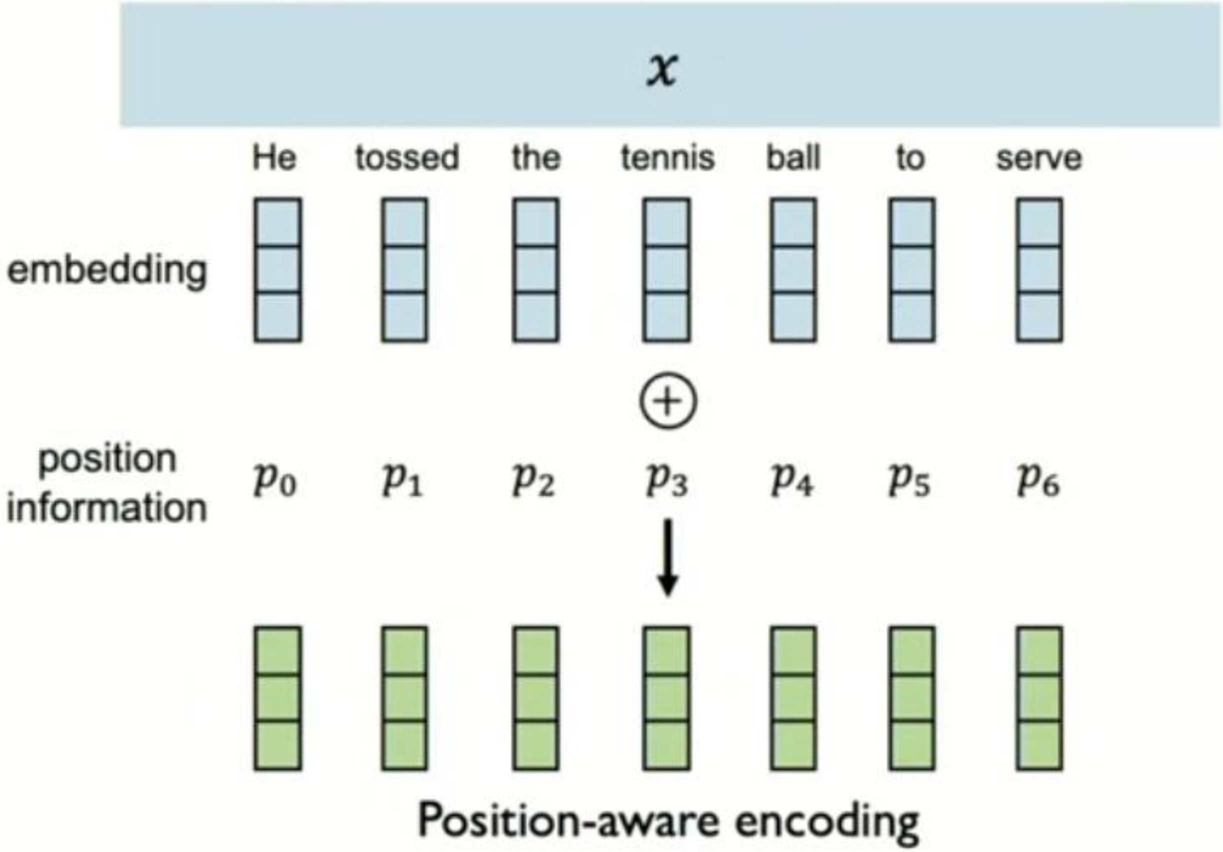
Attention


query
Layer 6
The animal didn't cross the street because it was too tired
self-attention distribution
Layer 5
The animal didn't cross the street because it was too tired
x1 x2 x3 x4 x5 x6 x7 xi
values

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

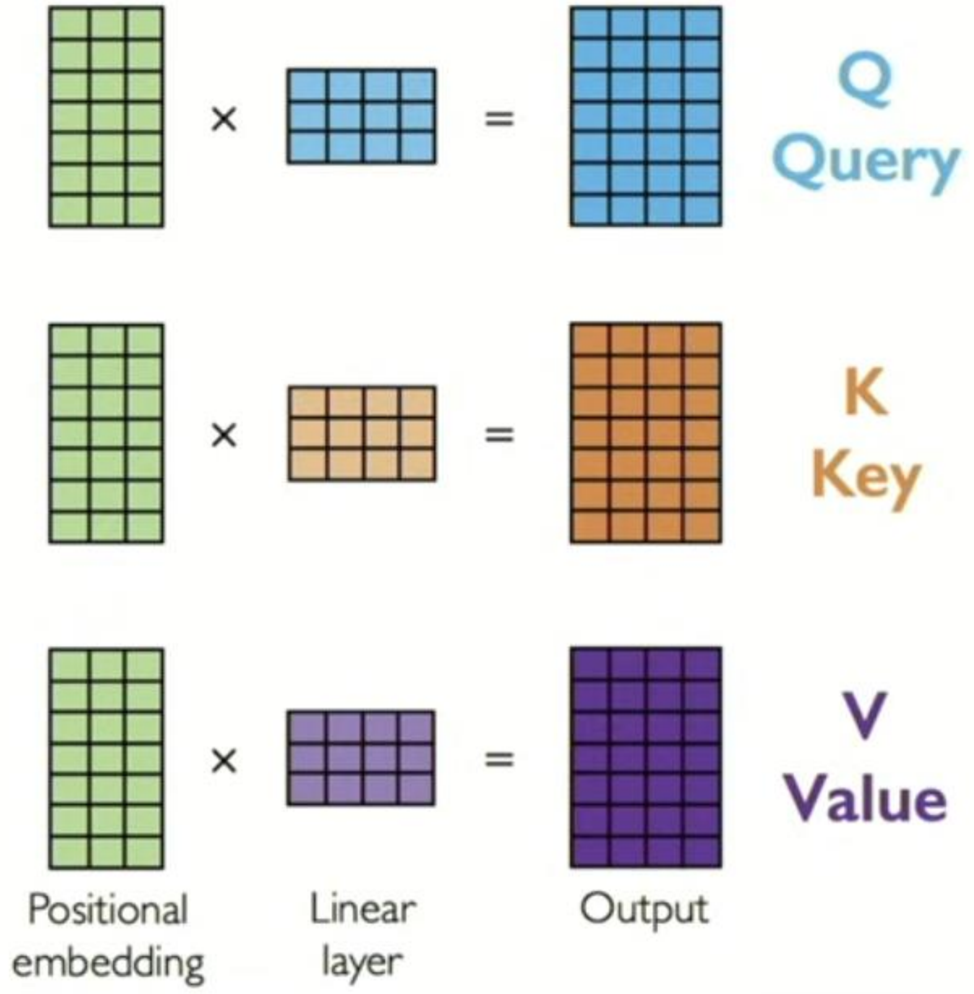


Data is fed in all at once! Need to encode position information to understand order.

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**



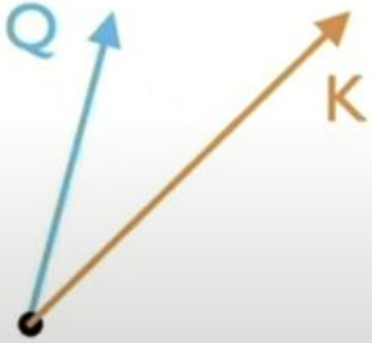
Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention score: compute pairwise similarity between each query and key

How to compute similarity between two sets of features?



Dot product

Scale

$$\boxed{\frac{Q \cdot K^T}{\text{scaling}}}$$

Similarity metric

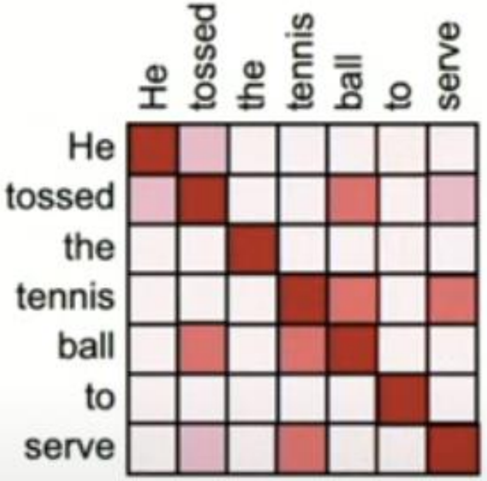
Also known as the "cosine similarity"

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract features with high attention

Attention weighting: where to attend to! How similar is the key to the query?



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right)$$

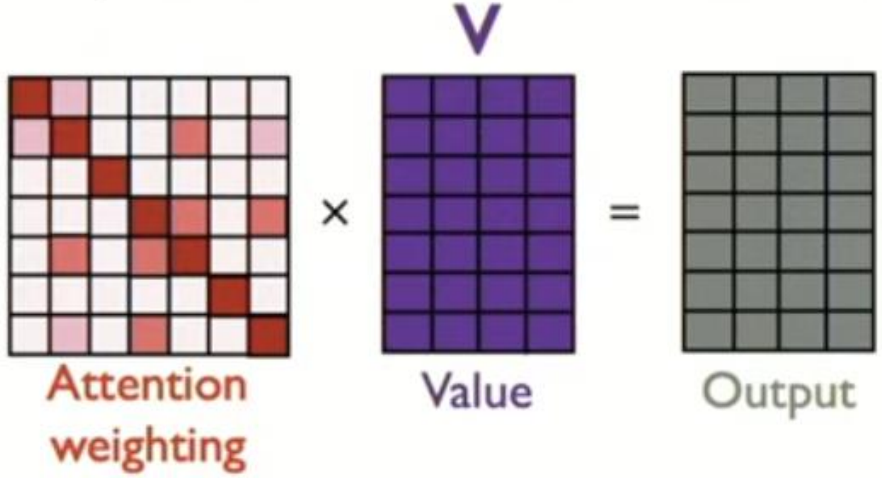
Attention weighting

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

Last step: self-attend to extract features



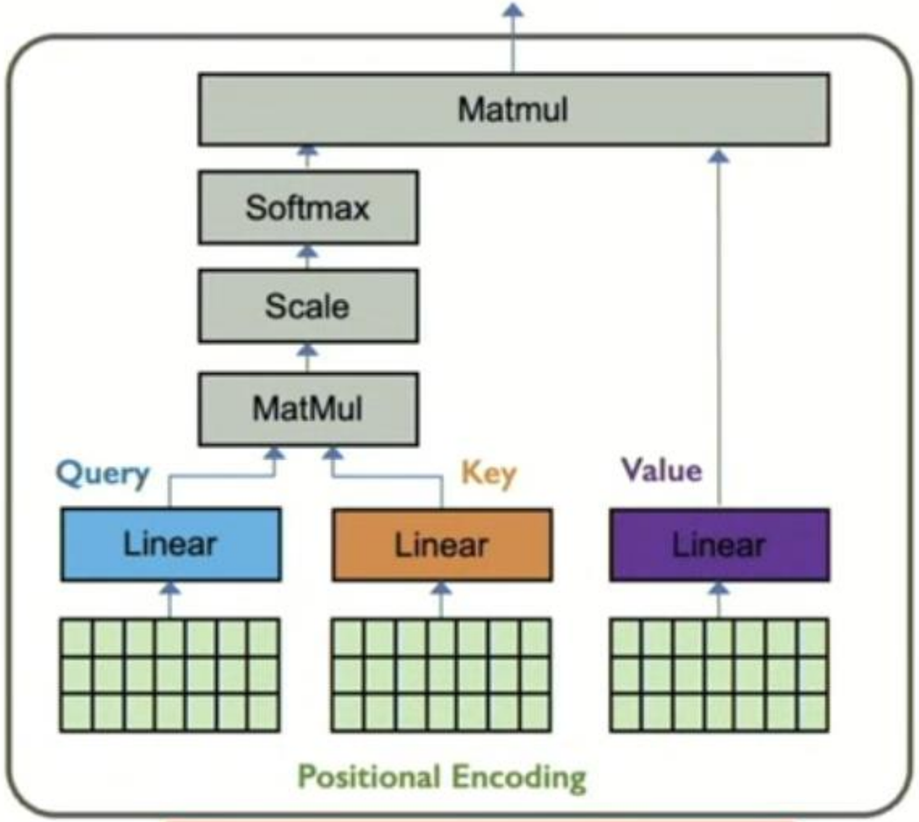
$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V = A(Q,K,V)$$

Learning Self-Attention with Neural Networks

Goal: identify and attend to most important features in input.

1. Encode **position** information
2. Extract **query, key, value** for search
3. Compute **attention weighting**
4. Extract **features with high attention**

These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of input.



$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V$$

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Dan Jurafsky
  - Alexander and Ava Amini
- Thanks!