CS 7775

Seminar in Computer Security: Machine Learning Security and Privacy
Fall 2023

Alina Oprea
Associate Professor
Khoury College of Computer Science

September 14 2023

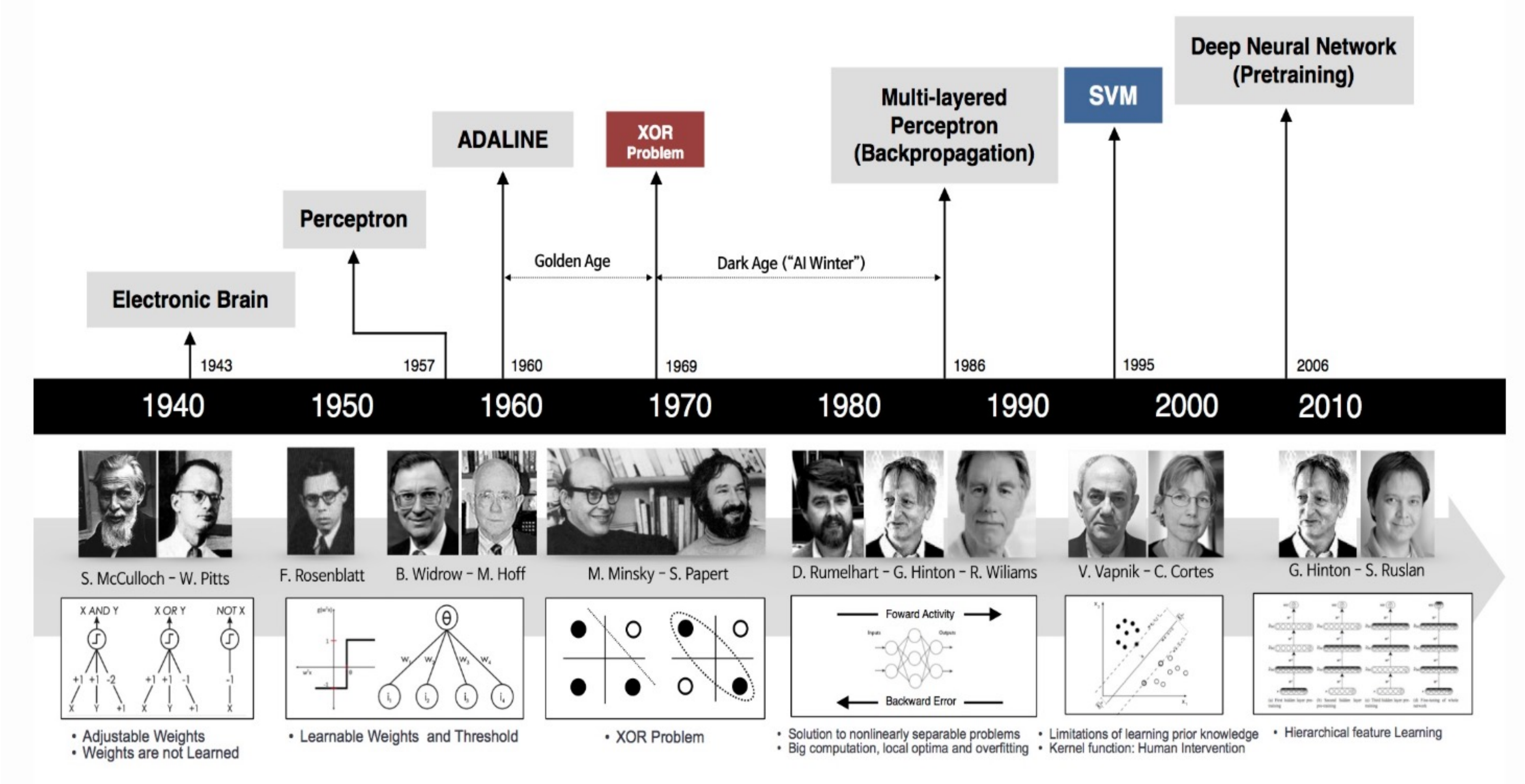
Outline: Review of Deep Learning

- History of deep learning
- Feed-forward neural networks
- Convolutional networks
- Regularization
- Backpropagation
- Comparing classifiers
- Transformers for language models

Deep Learning References

- Chapter 10 from Introduction to Statistical Learning
  - https://www.statlearning.com/
- Deep Learning books
  - https://d2l.ai/ (D2L)
  - https://www.deeplearningbook.org/ (advanced)
- Stanford notes on deep learning
  - http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf
- Stanford tutorial on training Multi-Layer Neural Networks
  - http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
- Notes on backpropagation by Andrew Ng
  - http://cs229.stanford.edu/notes-spring2019/backprop.pdf

Timeline



Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
ADALINE
XOR Problem
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

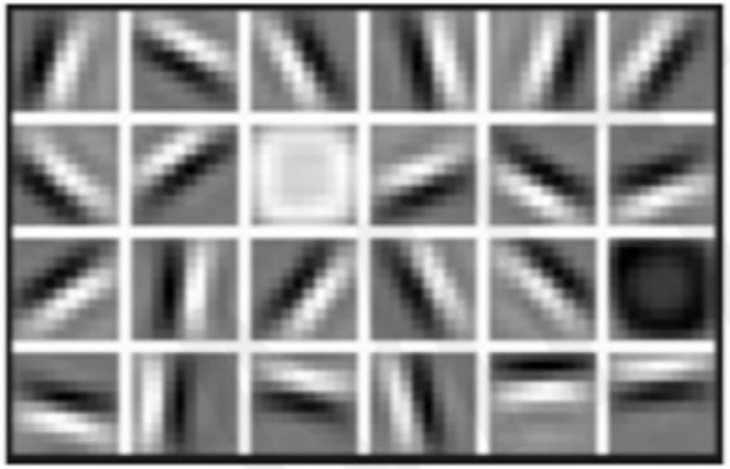
TensorFlow

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

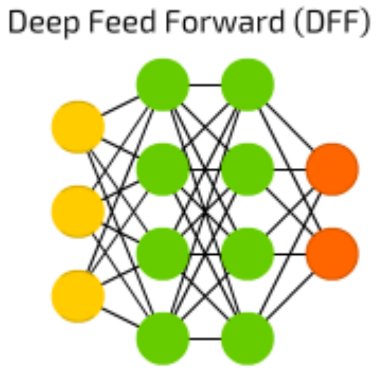
High Level Features

Facial Structure

Neural Network Architectures

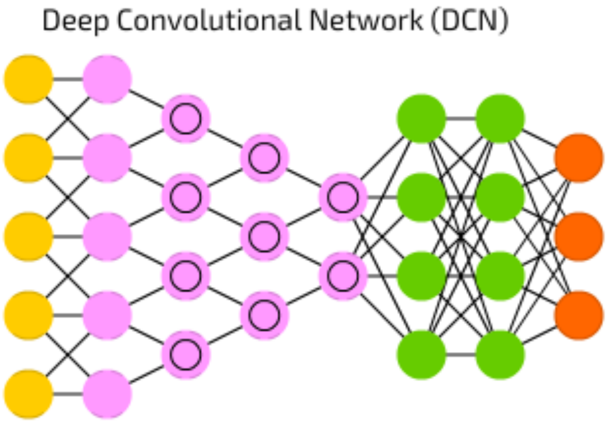
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



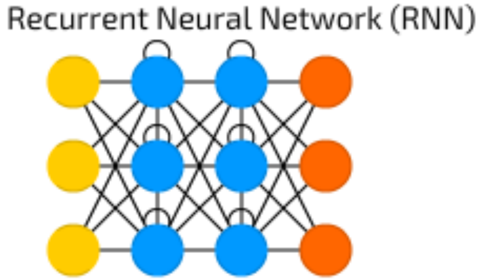
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

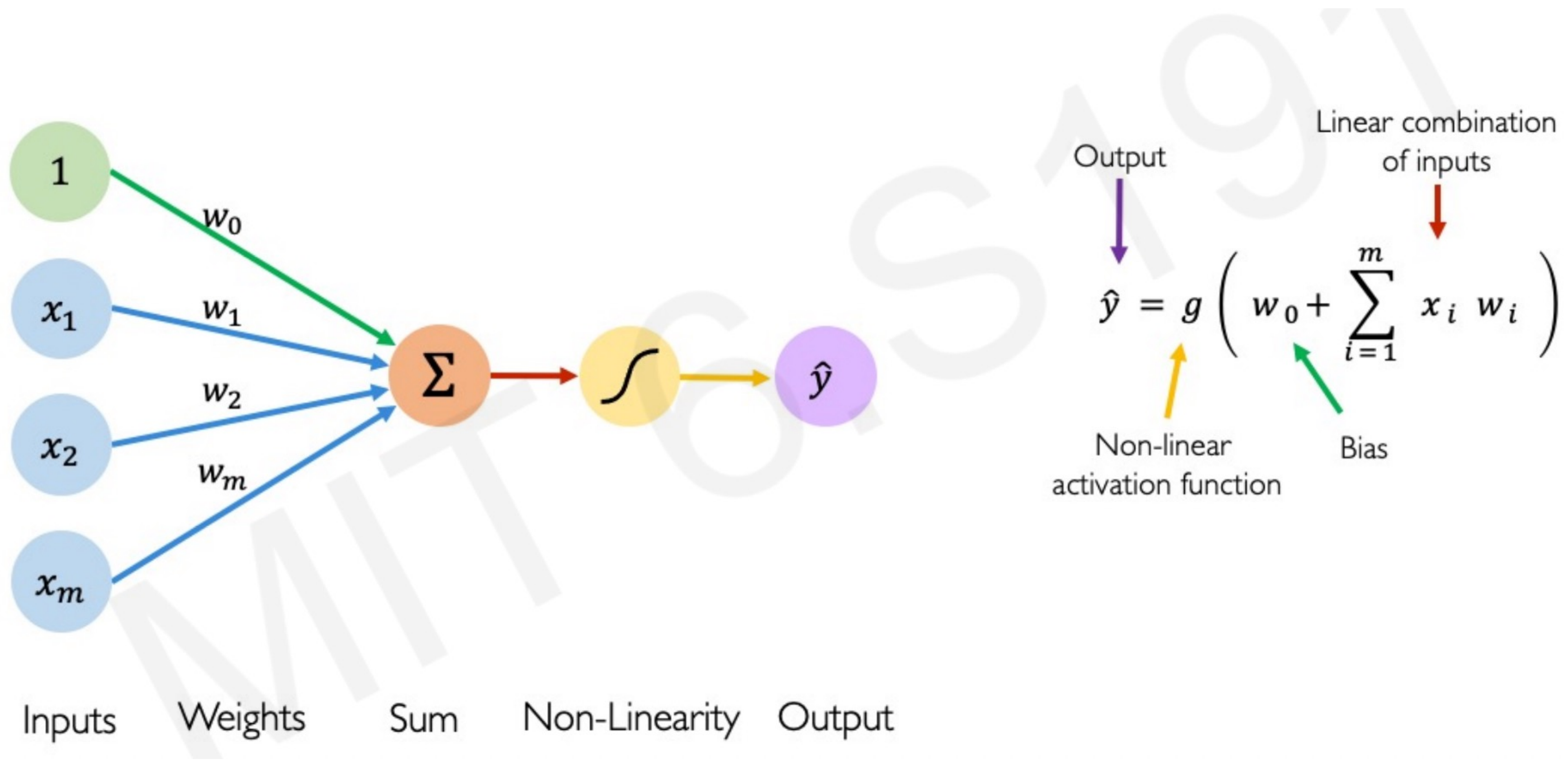


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



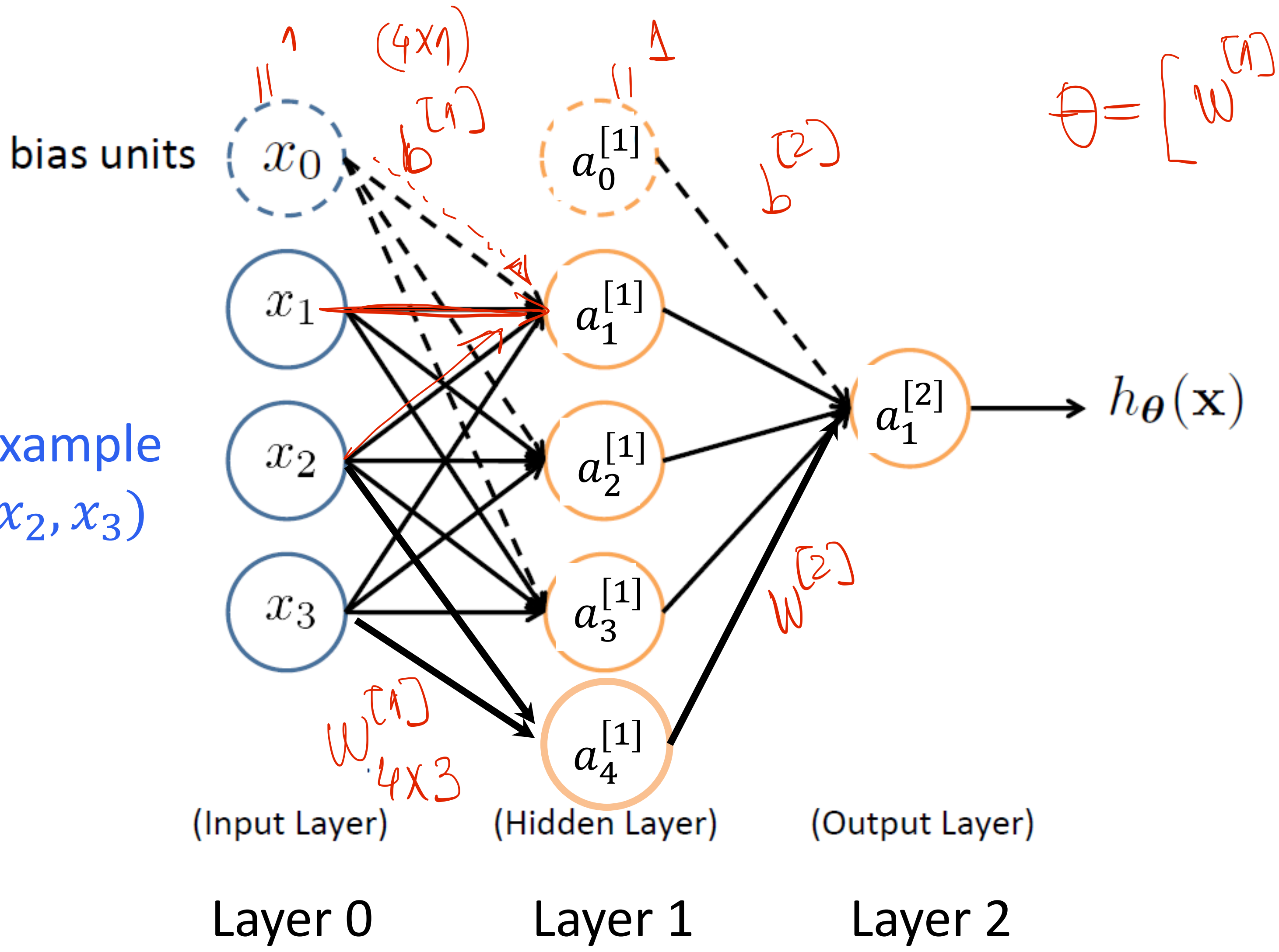
The Perceptron



Feed-Forward Neural Network

$\Theta = [W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}]$

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



Layer Operations

$g =$ ACT. FUNCTION

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

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

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

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

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

Linear

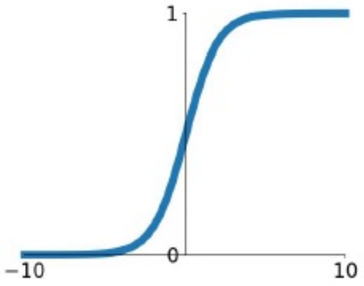
$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Activation Functions

Sigmoid

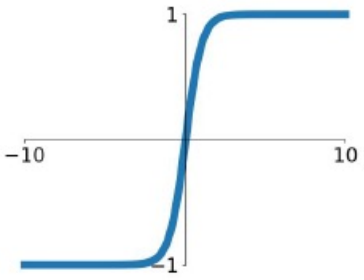
$\sigma(x) = \frac{1}{1+e^{-x}}$



Binary Classification

tanh

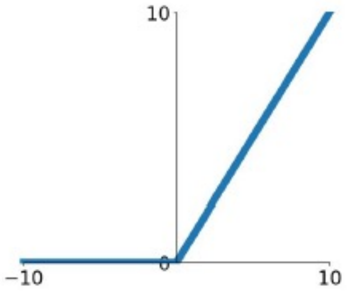
$\tanh(x)$



Regression

ReLU

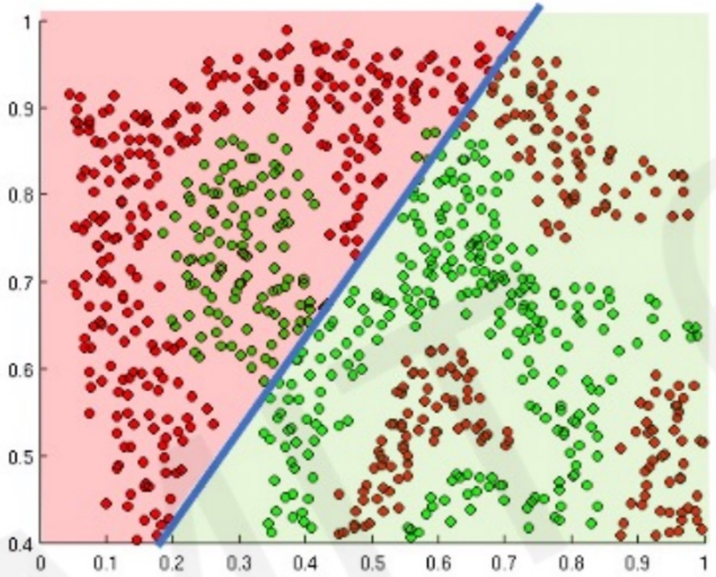
$\max(0, x)$



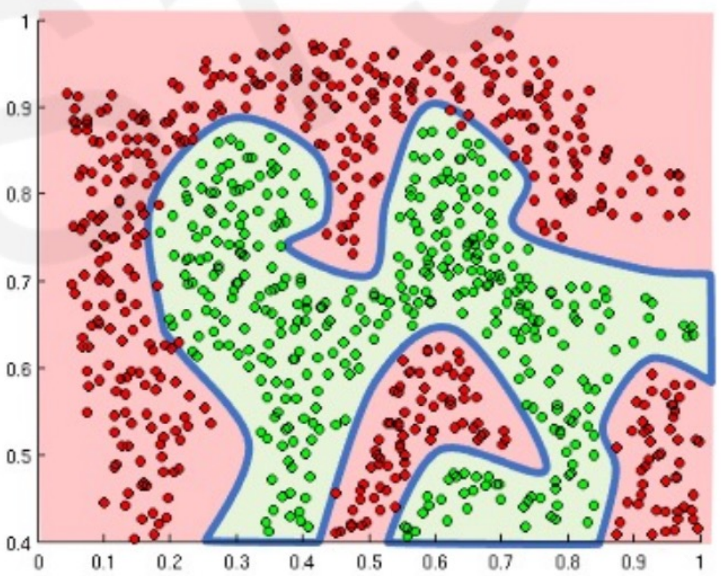
Intermediary layers

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*

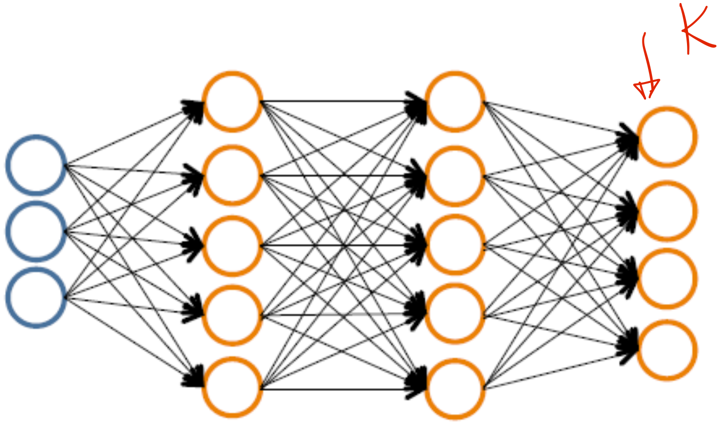


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Neural Network Classification



Given:

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

$\mathbf{s} \in \mathbb{N}^{+L}$ 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

$P[Y=1 \mid X=x]$

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

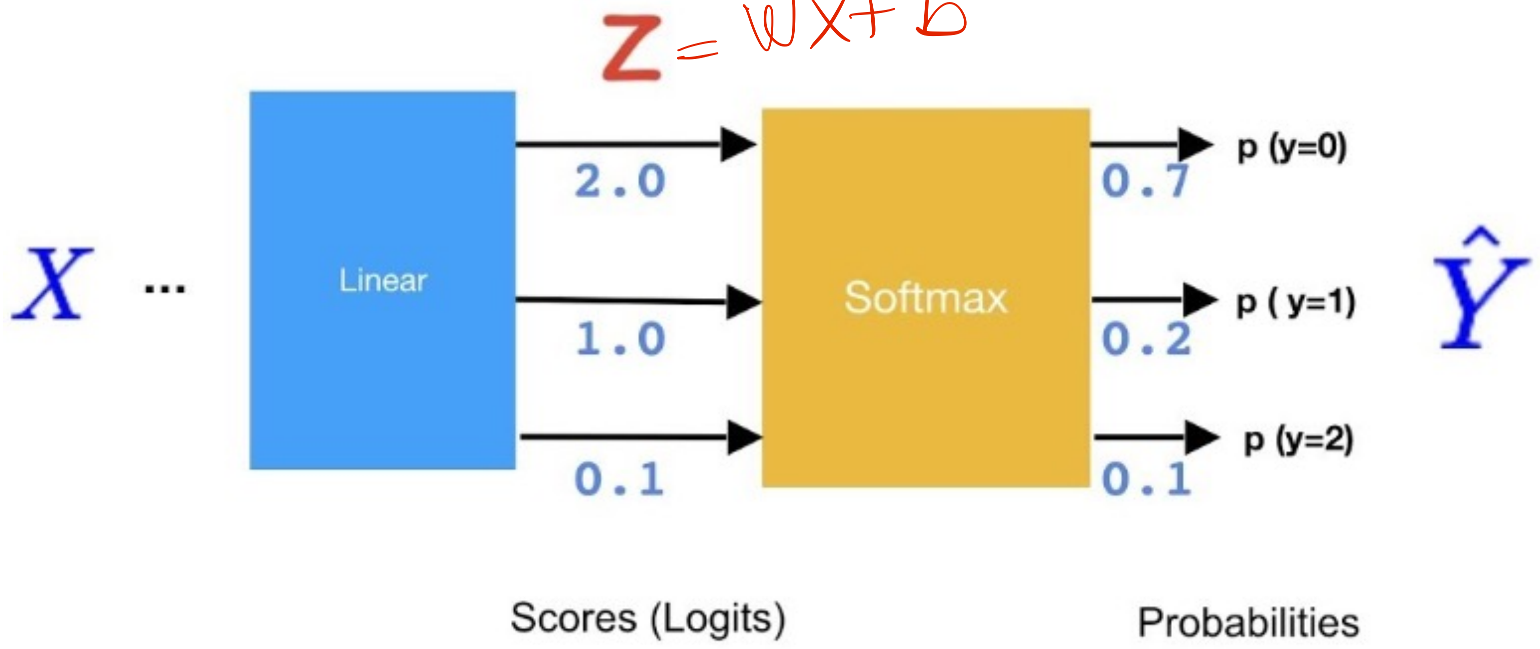
$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

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

Softmax

Softmax classifier



$= wx + b$

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

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

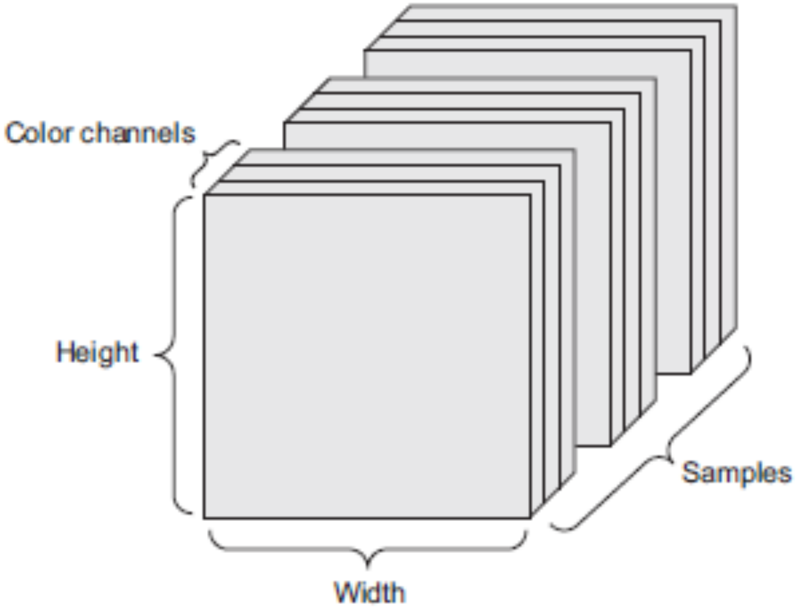
- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

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

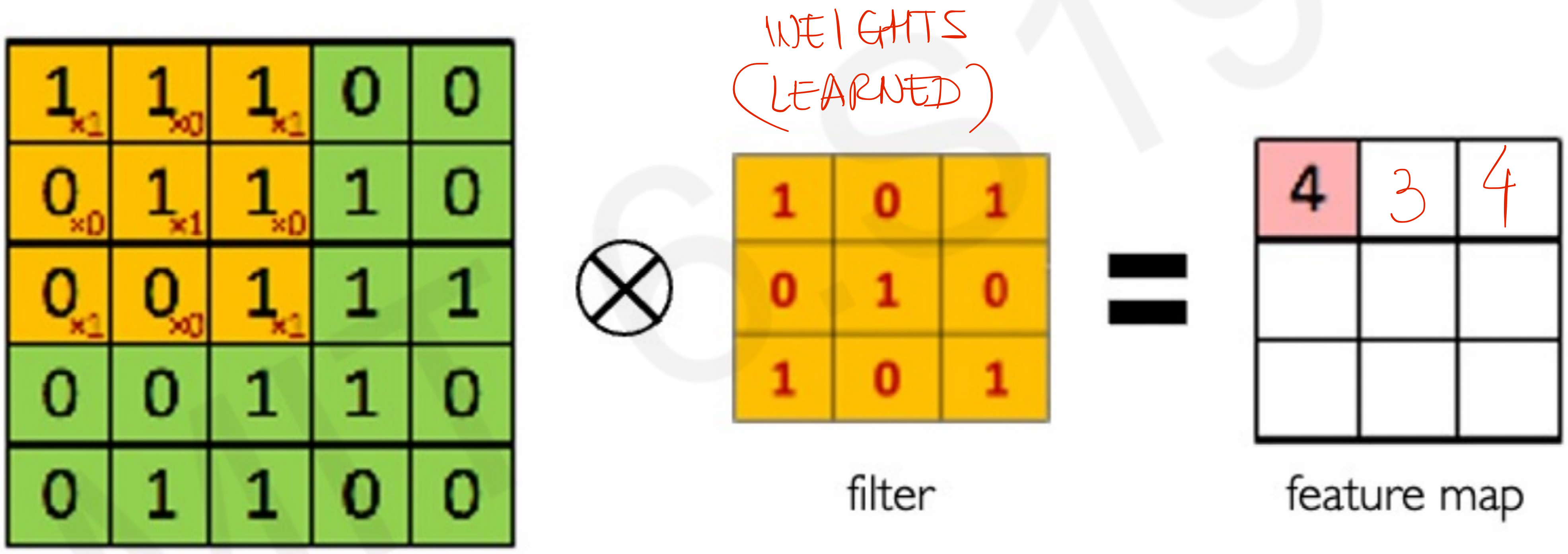
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

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

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

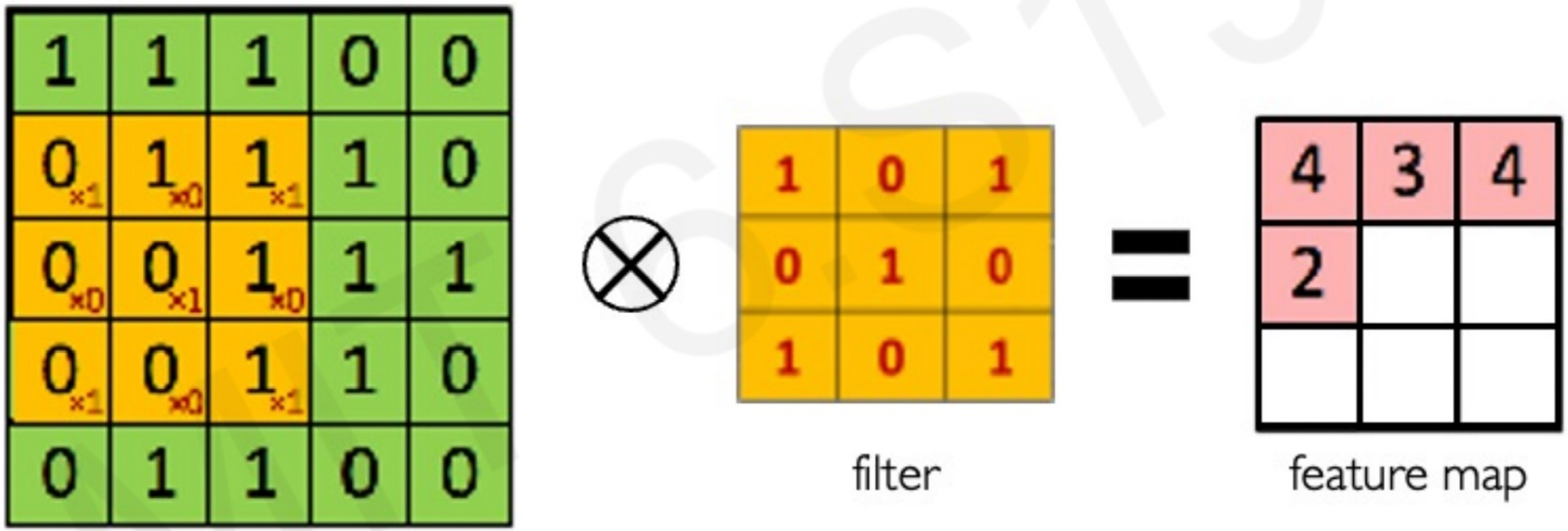
The Convolution Operation

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

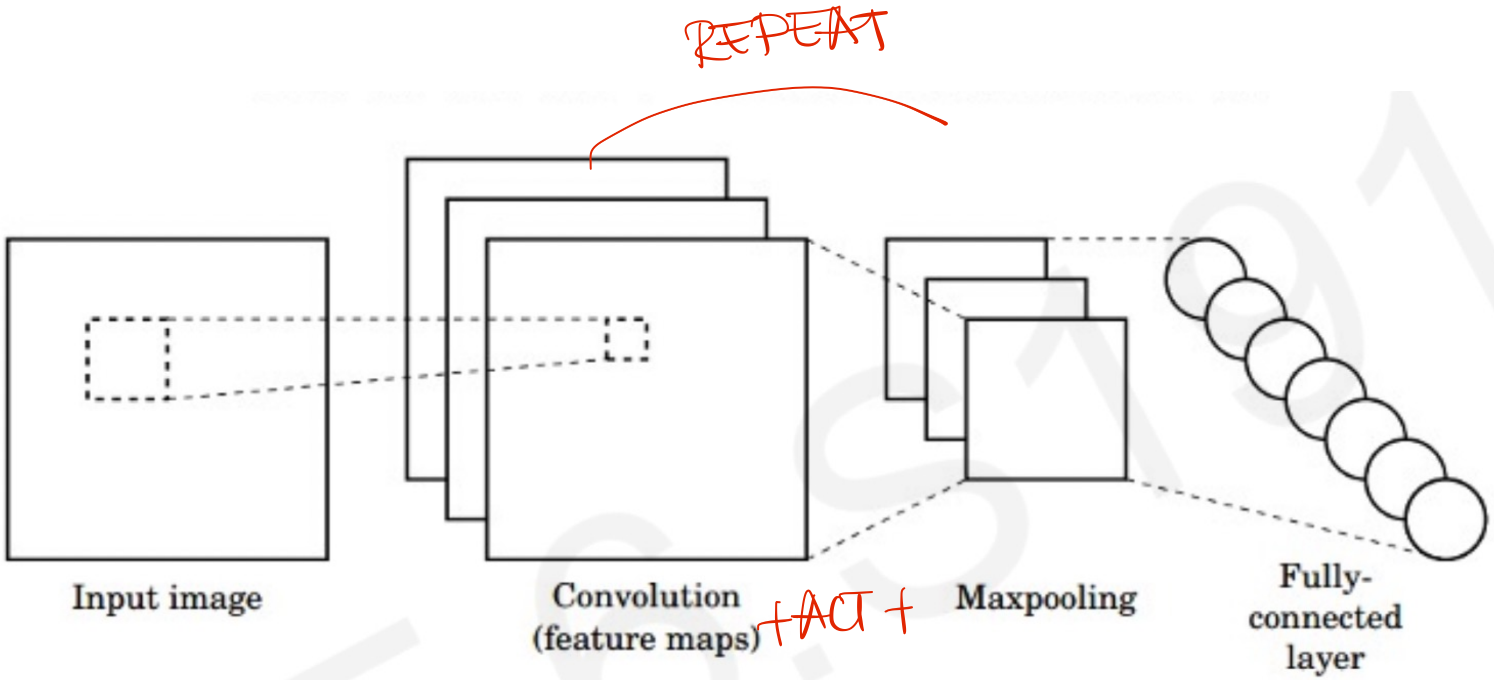


The Convolution Operation

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.

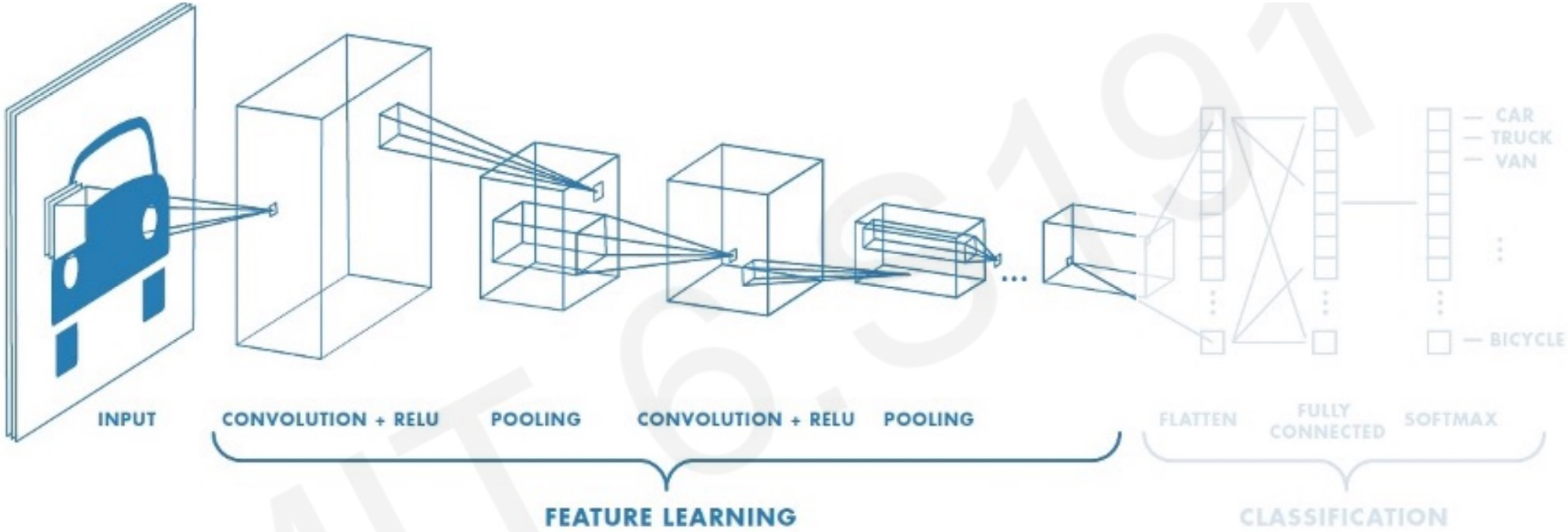
```
tf.keras.layers.Conv2D
```

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

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

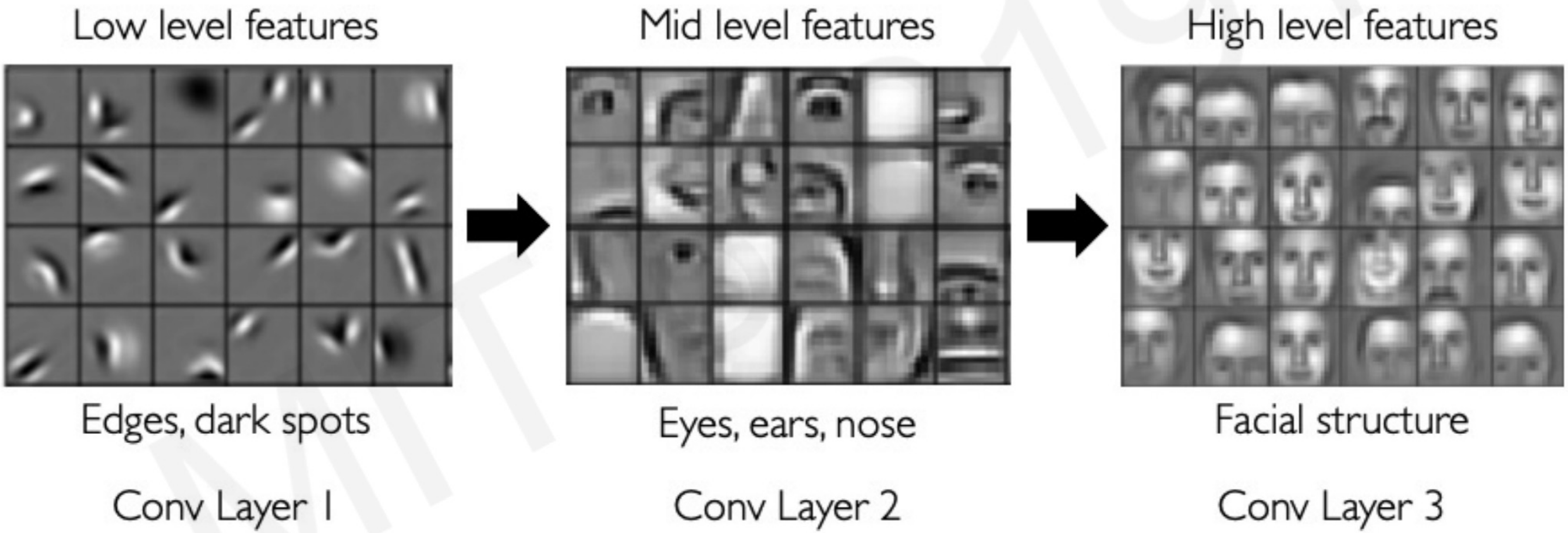
Train model with image data.
Learn weights of filters in convolutional layers.

CNNs for Classification: Feature Learning

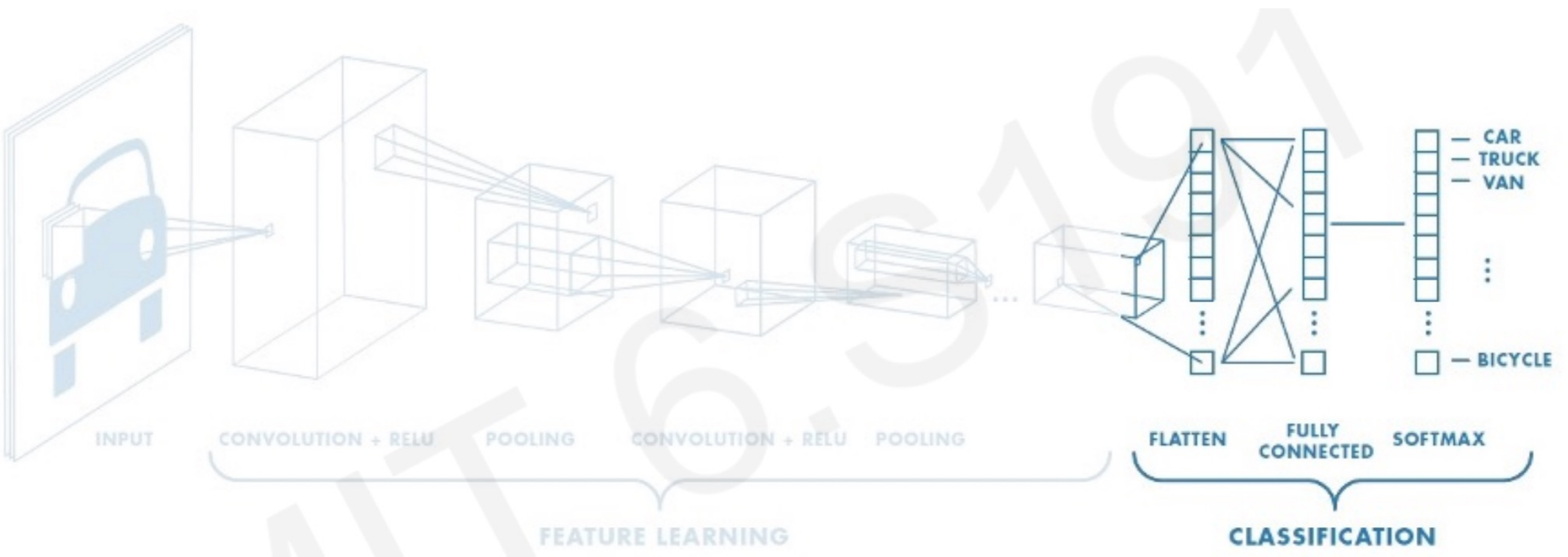


1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

Representation Learning with CNNs



CNNs for Classification: Class Probabilities

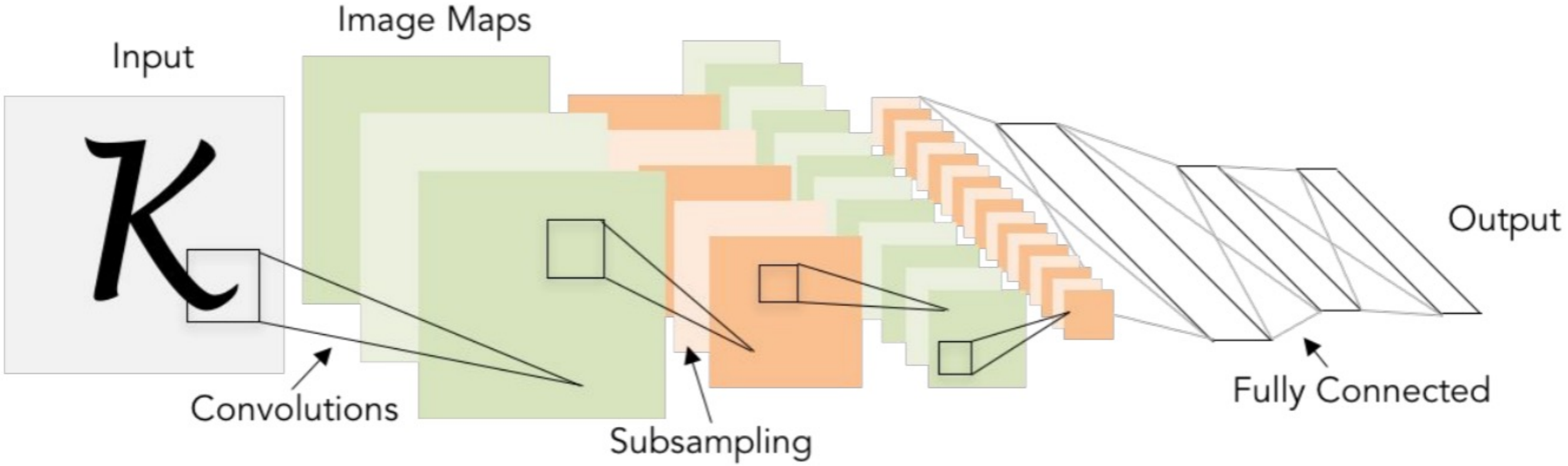


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

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

LeNet 5

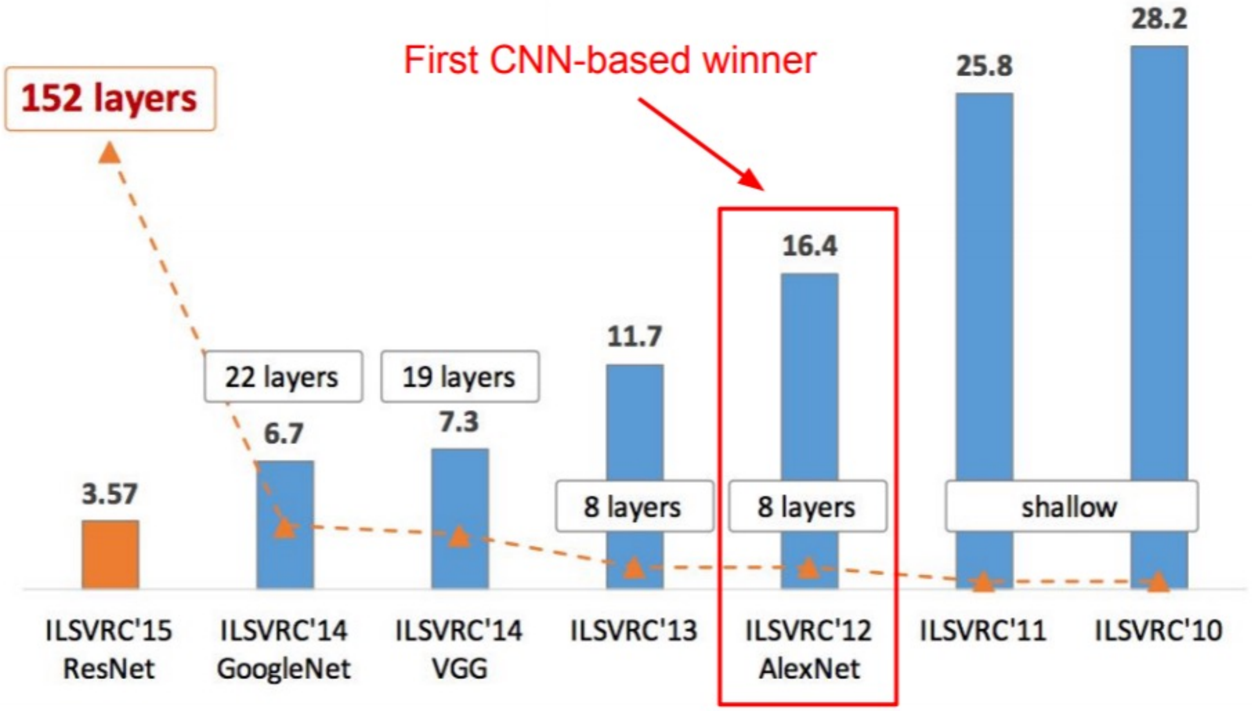
[LeCun et al., 1998]



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 $x_1, y_1, \ldots x_N, y_N$
- One training example $x_i = (\underbrace{x_{i1}, \ldots x_{id}}_{\in \mathbb{R}^d})$, label y_i
- One forward pass through the network
  - Compute prediction $\hat{y}_i = h(x_i)$
- 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
  - $\boxed{J(W, b) = \frac{1}{N}\sum_i L(\hat{y}_i, y_i)}$

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) last layer $L, L-1, \ldots, 1$
    - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \boxed{\frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \boxed{\frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}}$

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)
    - $W^{[\ell]} = W^{[\ell]} - \alpha \boxed{\sum_{i=1}^{N}} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

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}

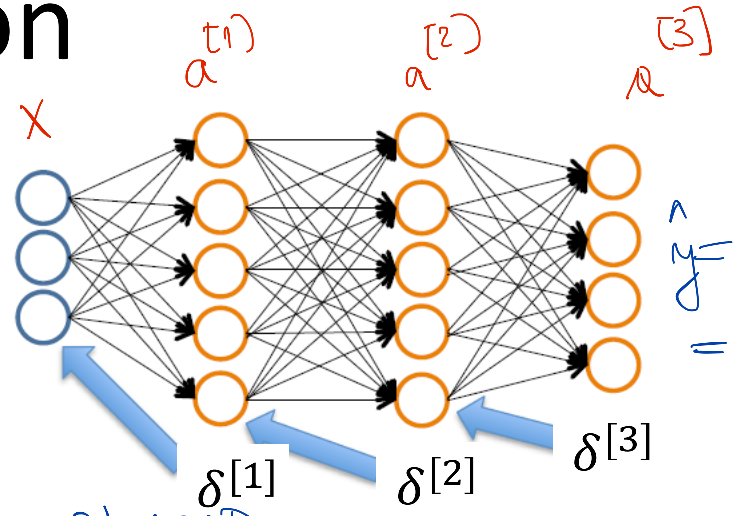
$$W^{[\ell]} = W^{[\ell]} - \alpha \boxed{\sum_{i=1}^{B}} \frac{\partial L(\hat{y}_{ib}, y_{ib})}{\partial W^{[\ell]}}$$

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

Backpropagation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

$$L(y,\hat{y}) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$$



$\hat{y} = a^{[L]} = g(z^{[L]})$

Definitions

- $\boxed{z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}}$, $\boxed{a^{[\ell]} = g(z^{[\ell]})}$
- $\boxed{\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}}$; Output $\hat{y} = a^{[L]} = g(z^{[L]})$

g = SIGMOID

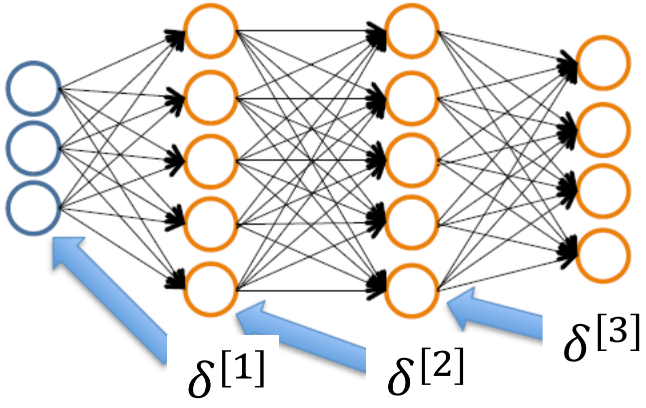
1) Layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \underbrace{g'(z^{[L]})}_{g(z^{[L]})(1-g(z^{[L]})) = \hat{y}(1-\hat{y})} = \hat{y} - y$

2) Layer $l < L$: $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} = \underbrace{\frac{\partial L(\hat{y},y)}{\partial z^{[\ell+1]}}}_{= \delta^{[\ell+1]}} \cdot \underbrace{\frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}}}_{W^{[\ell+1]}} \cdot \underbrace{\frac{\partial a^{[\ell]}}{\partial z^{[\ell]}}}_{g'(z^{[\ell]})}$

Backpropagation

Let $\delta_j^{(l)} =$ "error" of node j in layer l

$$L(y, \hat{y}) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$$



Definitions

- $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

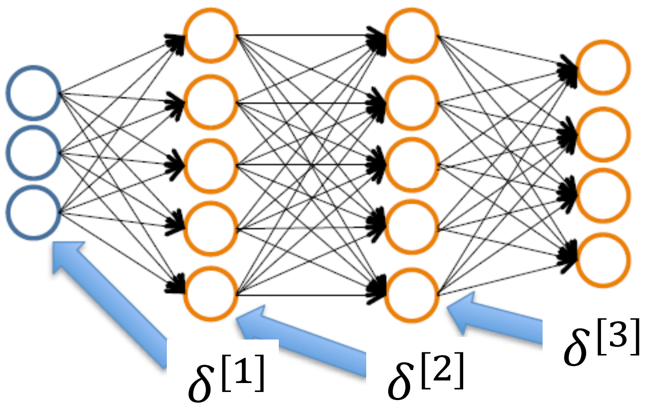
$$\frac{\partial L(\hat{y},y)}{\partial W^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \cdot \frac{\partial z^{[\ell]}}{\partial W^{[\ell]}} = \delta^{[\ell]} \cdot a^{[\ell-1]T}$$

$$\frac{\partial L(\hat{y},y)}{\partial b^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \cdot \frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = \delta^{[\ell]}$$

Backpropagation

Let $\delta_j^{(l)} =$ "error" of node j in layer l

$$L(y, \hat{y}) = -[(1-y)\log(1-\hat{y}) + \text{y}\log\hat{y}]$$



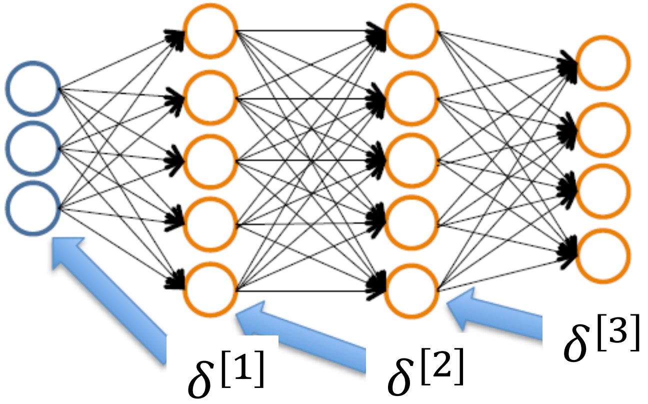
Definitions

- $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y}, y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

Backpropagation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

$$L(y,\hat{y}) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$$



Definitions

- $z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

1. For last layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \boldsymbol{z}^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}}g'\left(z^{[L]}\right)$
2. For layer ℓ: $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell+1]}}\frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}}\frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = \delta^{[\ell+1]}W^{[\ell+1]}g'(z^{[\ell]})$
3. Compute parameter gradients

- $\frac{\partial L(\hat{y},y)}{\partial W^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}\frac{\partial z^{[\ell]}}{\partial W^{[\ell]}} = \delta^{[\ell]}a^{[\ell-1]T}$
- $\frac{\partial L(\hat{y},y)}{\partial b^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}\frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = \delta^{[\ell]}$

Training NN with Backpropagation

Given training set $(x_1, y_1), \dots, (x_N, y_N)$

Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ

Loop

Set $\Delta_{ij}^{[l]}$ =0, for all layers l and indices i, j

EPOCH

For each training instance (x_k, y_k):

Compute $a^{[1]}, a^{[2]}, \dots, a^{[L]}$ via forward propagation

Compute errors $\delta^{[L]}, \delta^{[L-1]}, \dots \delta^{[1]}$

Aggregate gradients $\Delta_{ij}^{[l]} = \Delta_{ij}^{[l]} + a_j^{[l-1]}\delta_i^{[l]}$

MINI-BATCH

Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha\Delta_{ij}^{[\ell]}$
- Similarly for $b_{ij}^{[\ell]}$

Until weights converge or maximum number of epochs is reached

Overfitting

- The larger the network, the higher the capacity (more model parameters)
- But also more prone to overfitting!

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

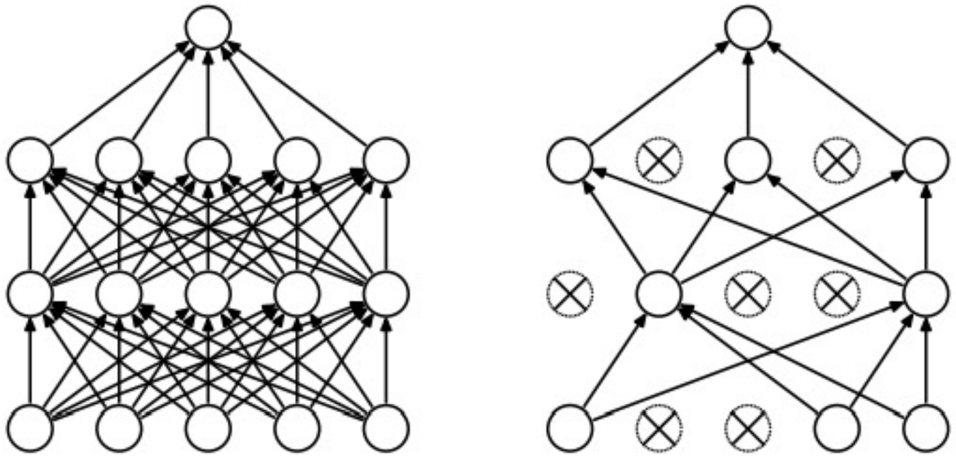
L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$ → Weight decay

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- When computing gradients of loss function, regularization term needs to be taken into account

Dropout



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

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- At training time, sample a sub-network per epoch (batch) and learn weights
  - Keep each neuron with probability p
- At testing time, all neurons are there, but multiply weight by a factor of p
- Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014

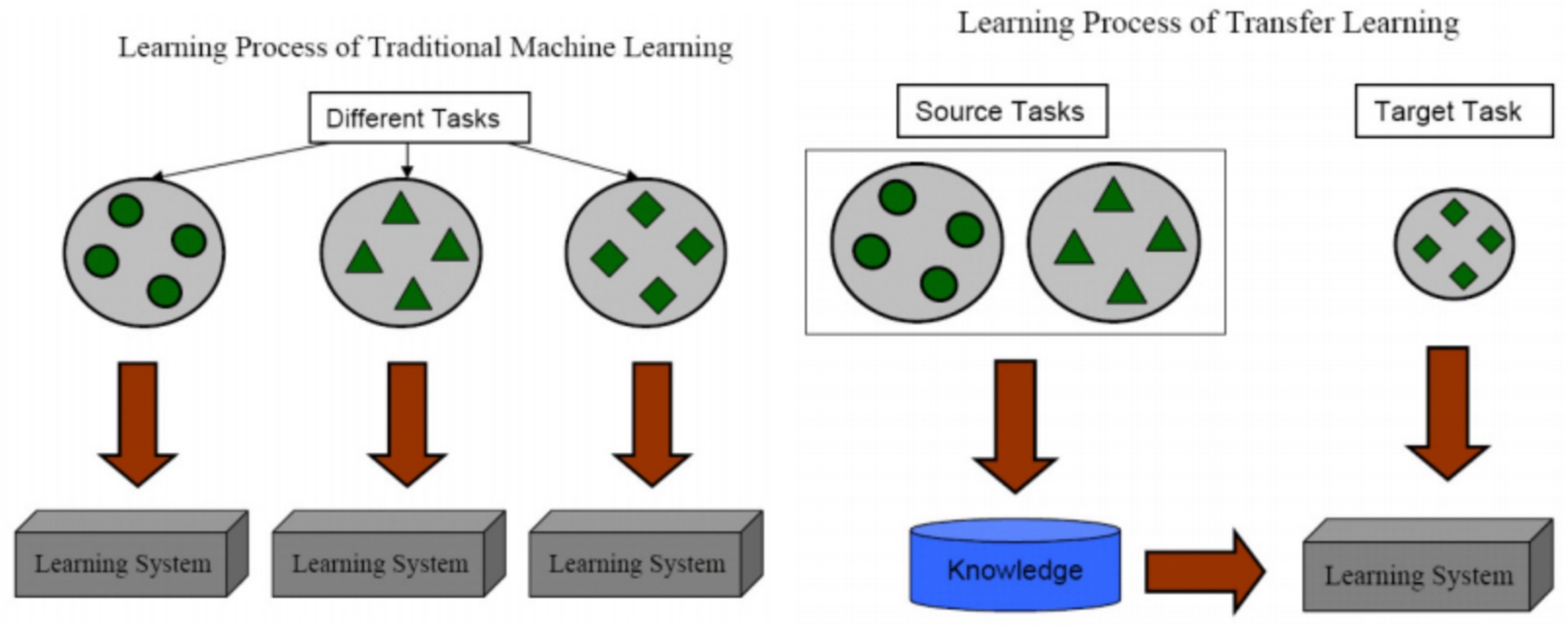
Comparing classifiers

Algorithm	Interpretable	Model size	Predictive accuracy	Training time	Testing time
Logistic regression	High	Small	Lower	Low	Low
Decision trees	High	Medium	Lower	Medium	Low
Ensembles	Low	Large	High	High	High
Naïve Bayes	Medium	Small	Lower	Medium	Low
SVM	Medium	Small	Lower	Medium	Low
Neural Networks	Low	Large	High	High	Low

Transfer Learning

- Improvement of learning in a **new** task through the *transfer of knowledge* from a **related** task that has already been learned.
- Motivation: Reuse representations learned by expensive training procedures that cannot be easily replicated
  - Image classification on ImageNet is very expensive (VGG-16: 138 million, ResNet 50: 23 million parameters)
  - Generative language models very large (BERT: 110 million, GPT-2: 1.5 billion, GPT-3: 175 billion parameters)
- Two major strategies
  - Pretrained Neural Network as fixed feature extractor
  - Fine-tuning the Neural Network

Transfer Learning



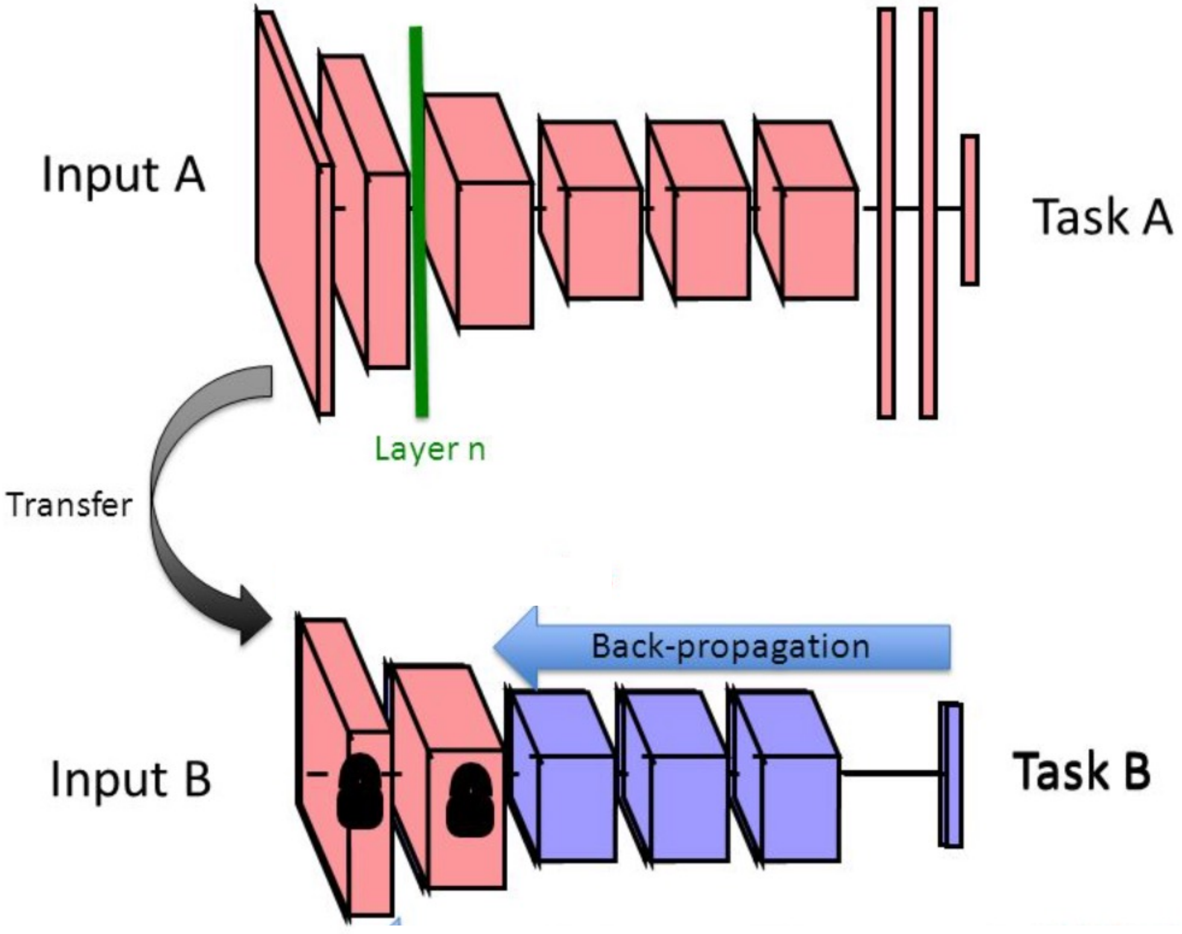
(a) Traditional Machine Learning (b) Transfer Learning

Methods for Transfer Learning

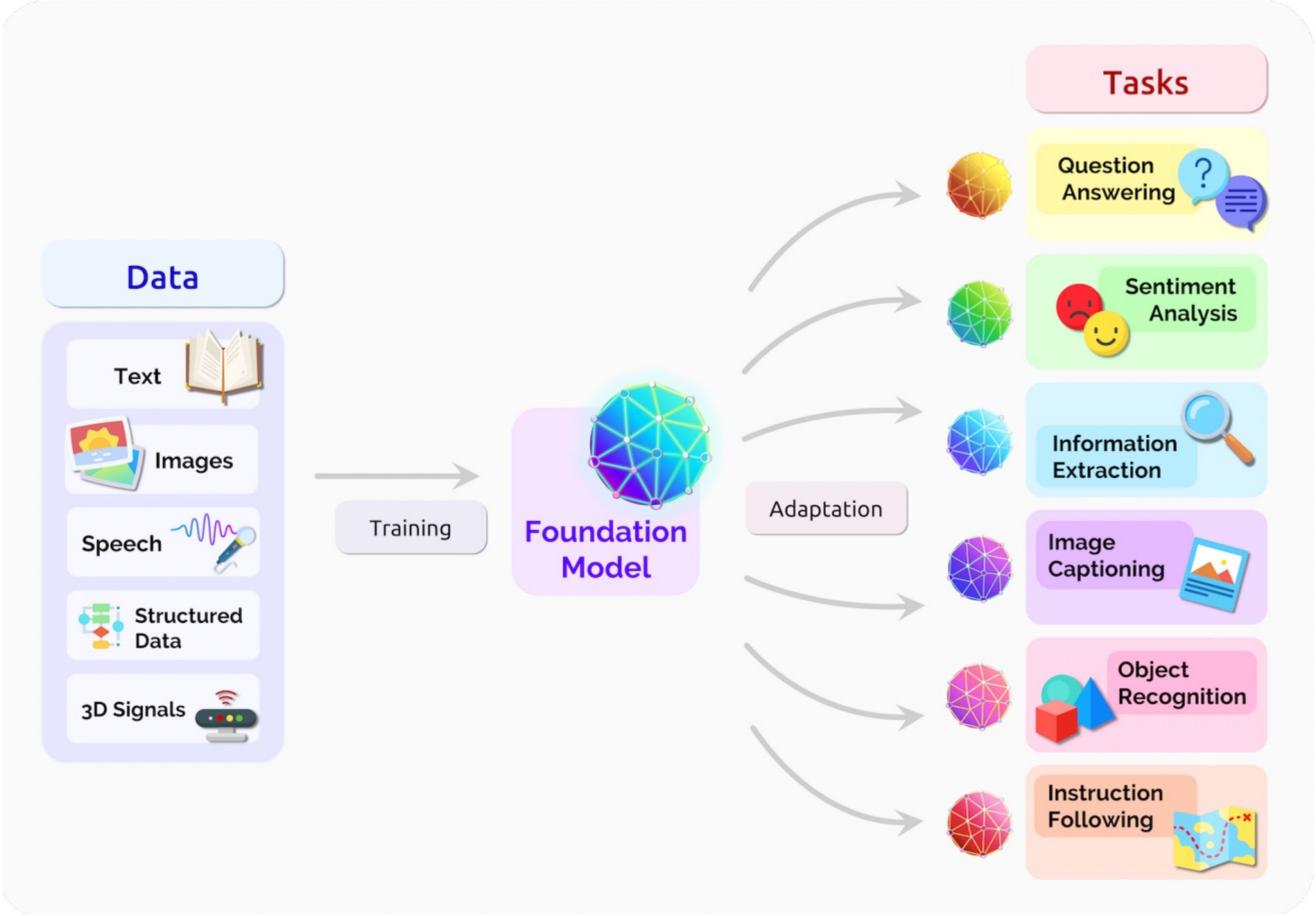
- Use a pre-trained model
  - https://modelzoo.co/

1. Use Convolutional Nets as Feature Extractor
   - Take a ConvNet pretrained on ImageNet
   - Remove the last fully-connected layer
   - Train the last layer on new dataset (usually a linear classifier such as logistic regression or softmax)
2. Fine-tuning
   - Decide to freeze first n layers
   - Train the remaining layers and stop backpropagation at layer n
   - Use a smaller learning rate
   - In the limit fine-tuning can be applied to all layers

Transfer Learning in NN: Freeze Layers

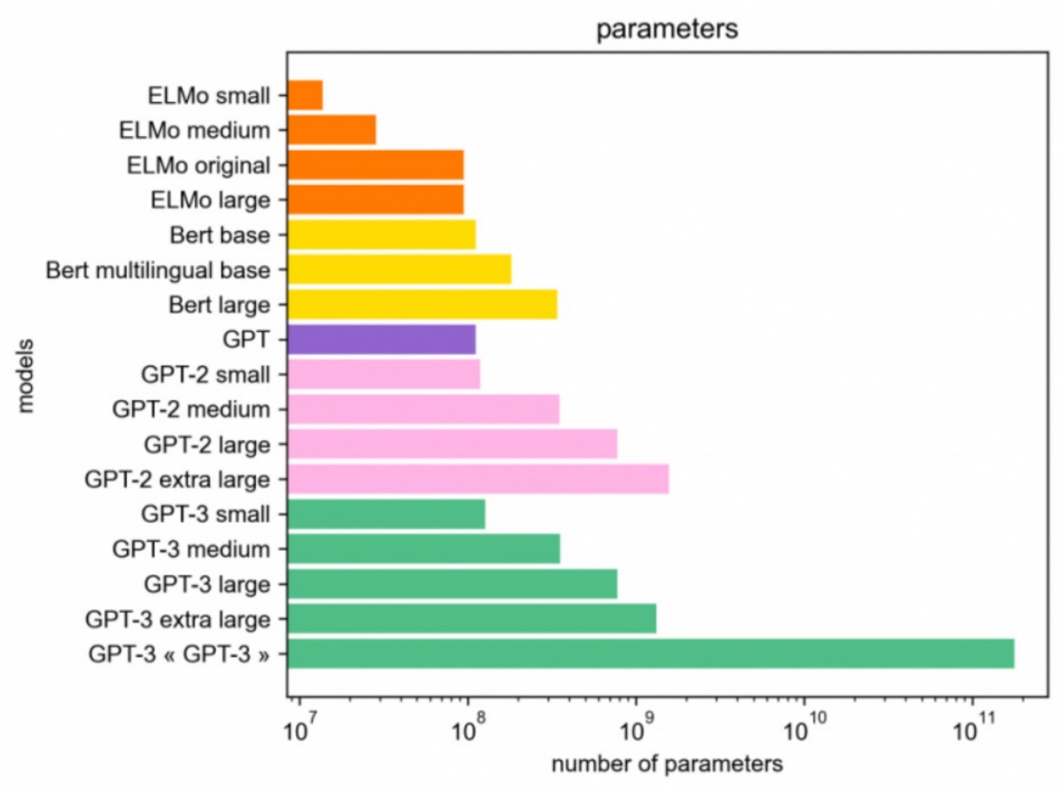


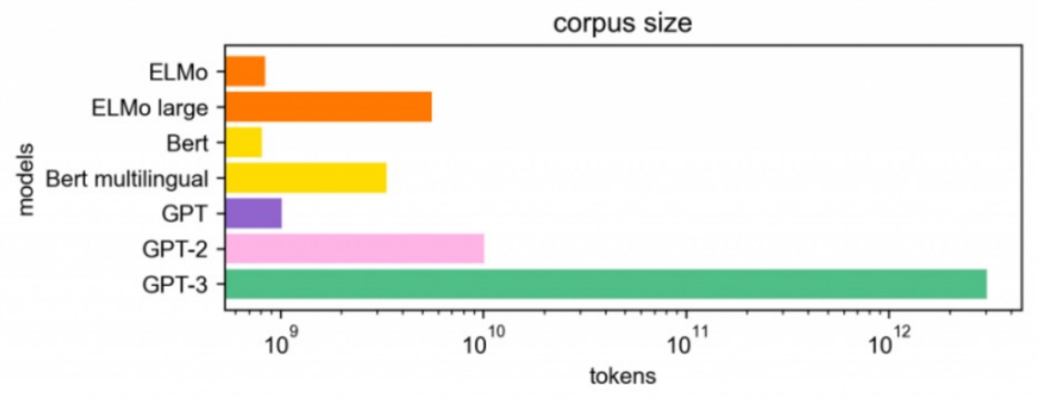
New Trend in AI: Foundation Models



On the Opportunities and Risks of Foundation Models

How Large are LLMs?

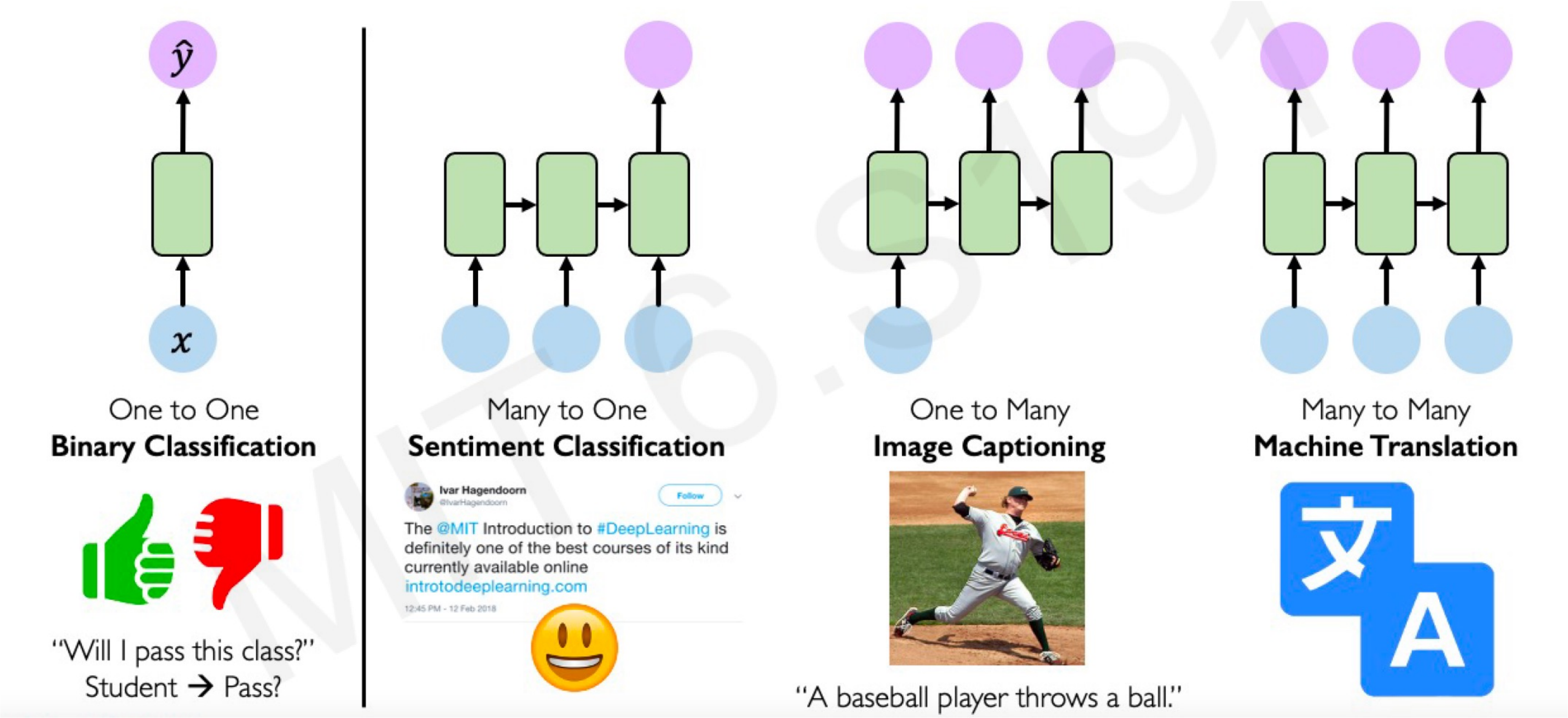




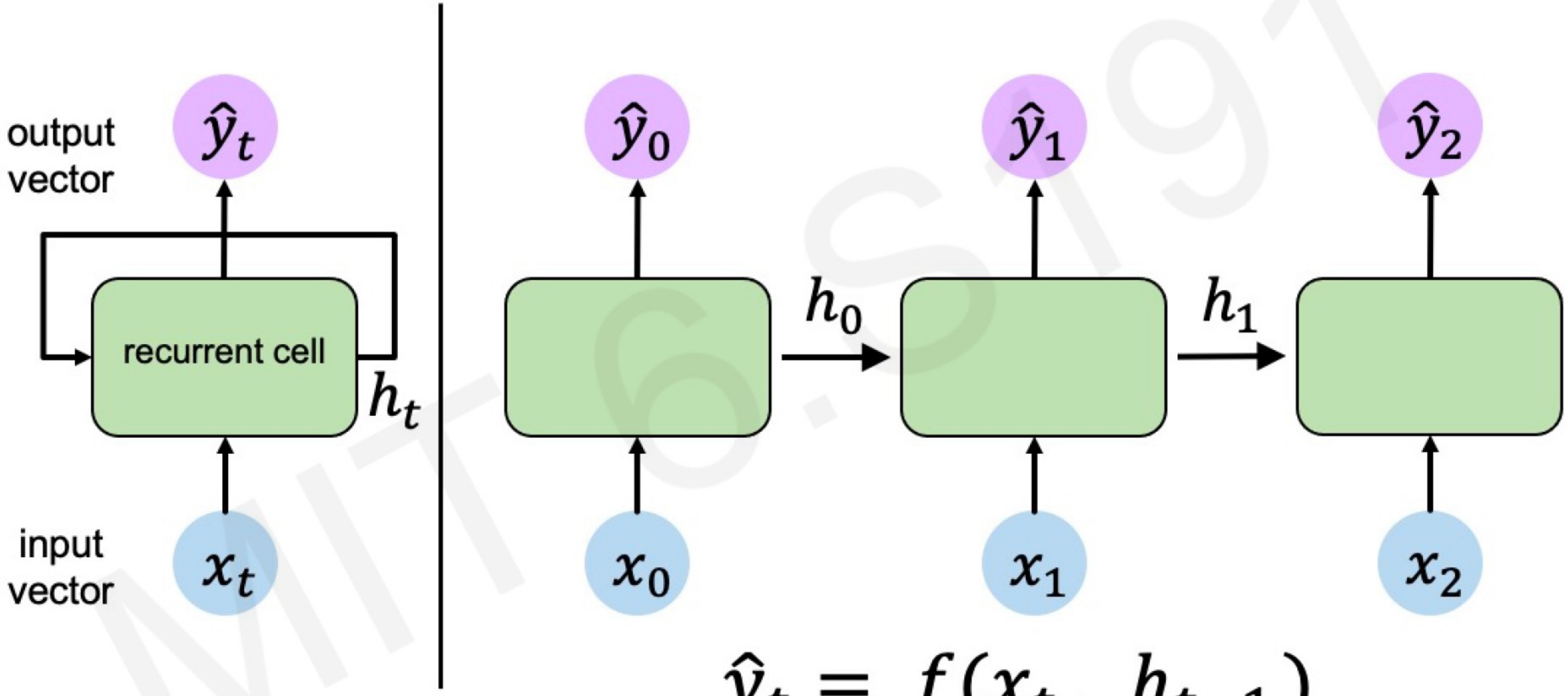
More recent models: PaLM (540B), OPT (175B), BLOOM (176B)…

Image source: https://hellofuture.orange.com/en/the-gpt-3-language-model-revolution-or-evolution/

Sequence Modeling Applications

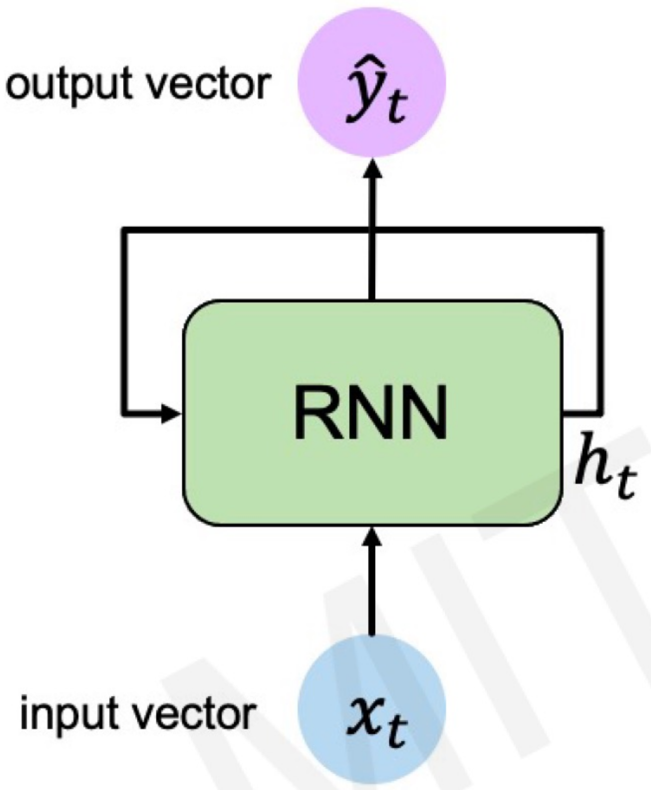


Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks (RNNs)



Apply a **recurrence relation** at every time step to process a sequence:

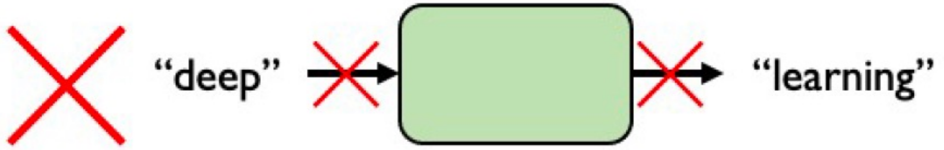
$$\boxed{h_t} = \boxed{f_W}(\boxed{x_t}, \boxed{h_{t-1}})$$

cell state | function with weights W | input | old state

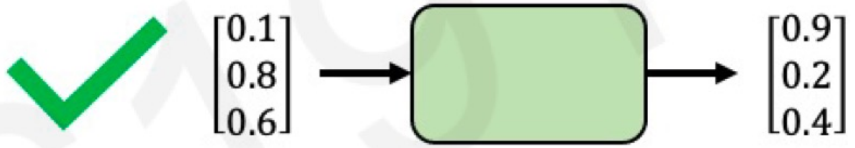
Note: the same function and set of parameters are used at every time step

RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

Encoding Language

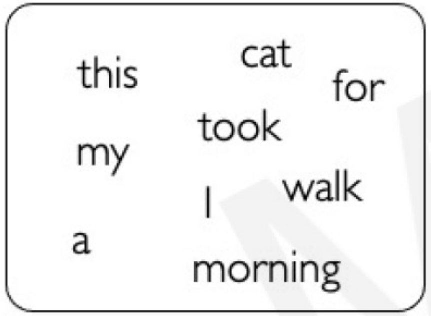


Neural networks cannot interpret words

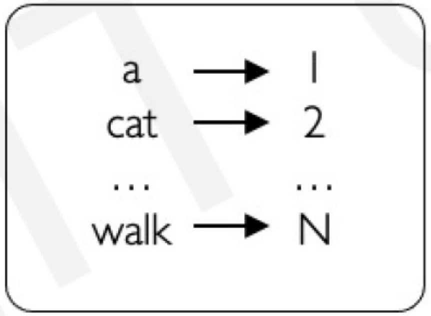


Neural networks require numerical inputs

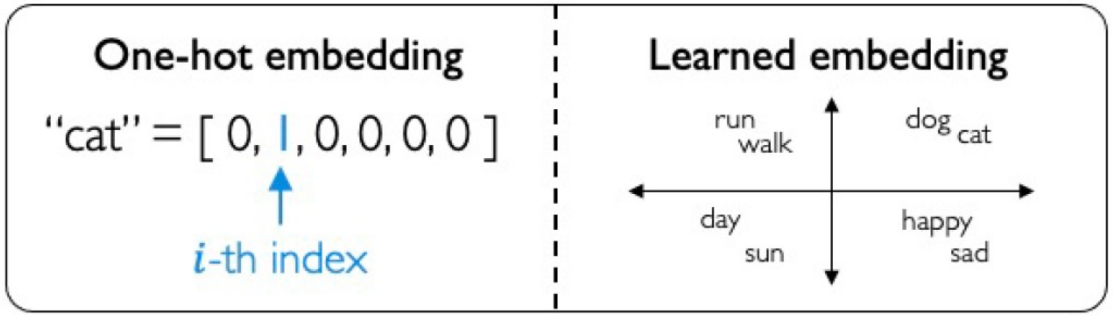
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



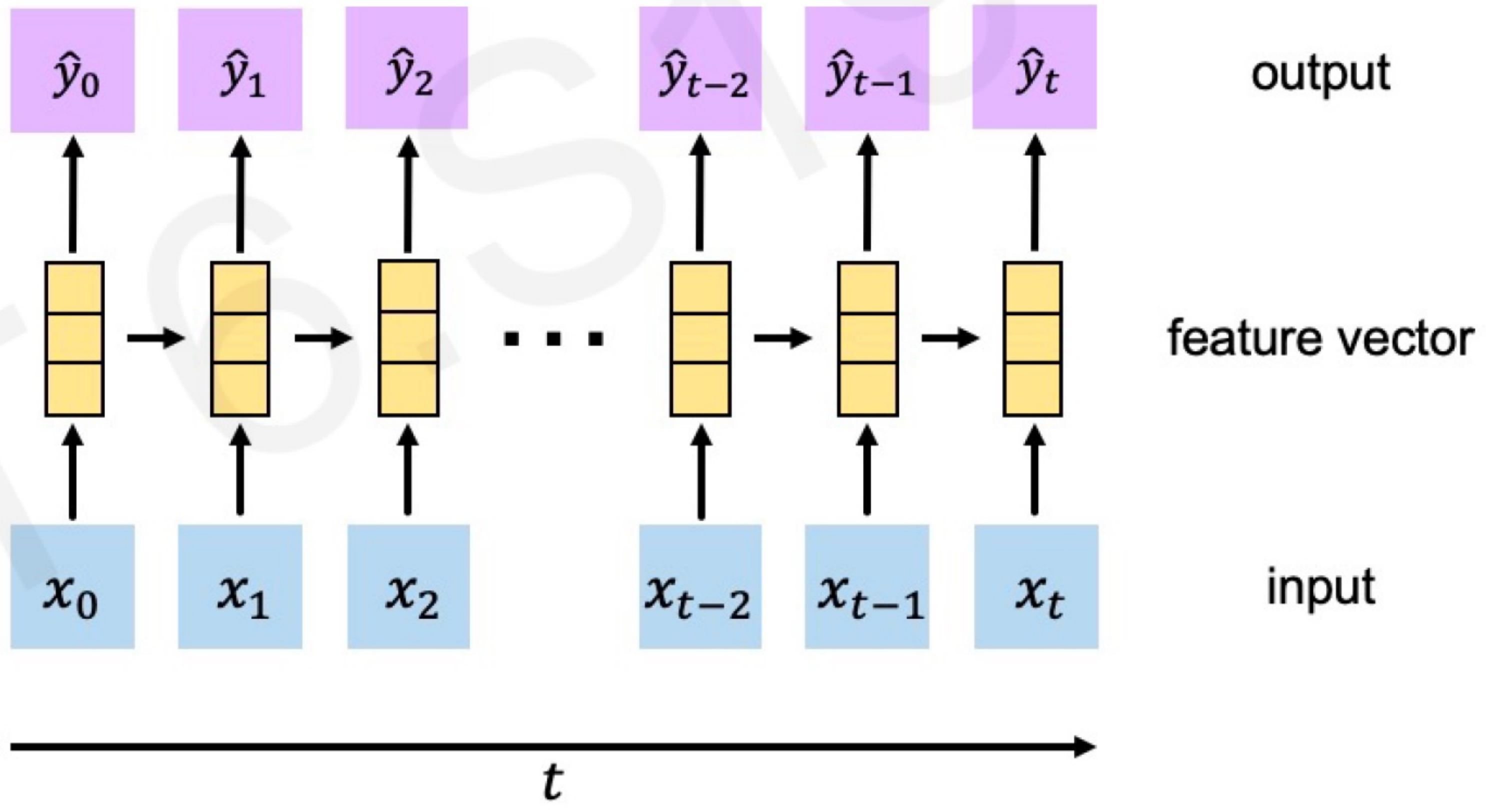
3. Embedding:
Index to fixed-sized vector

Goal of Sequence Modeling

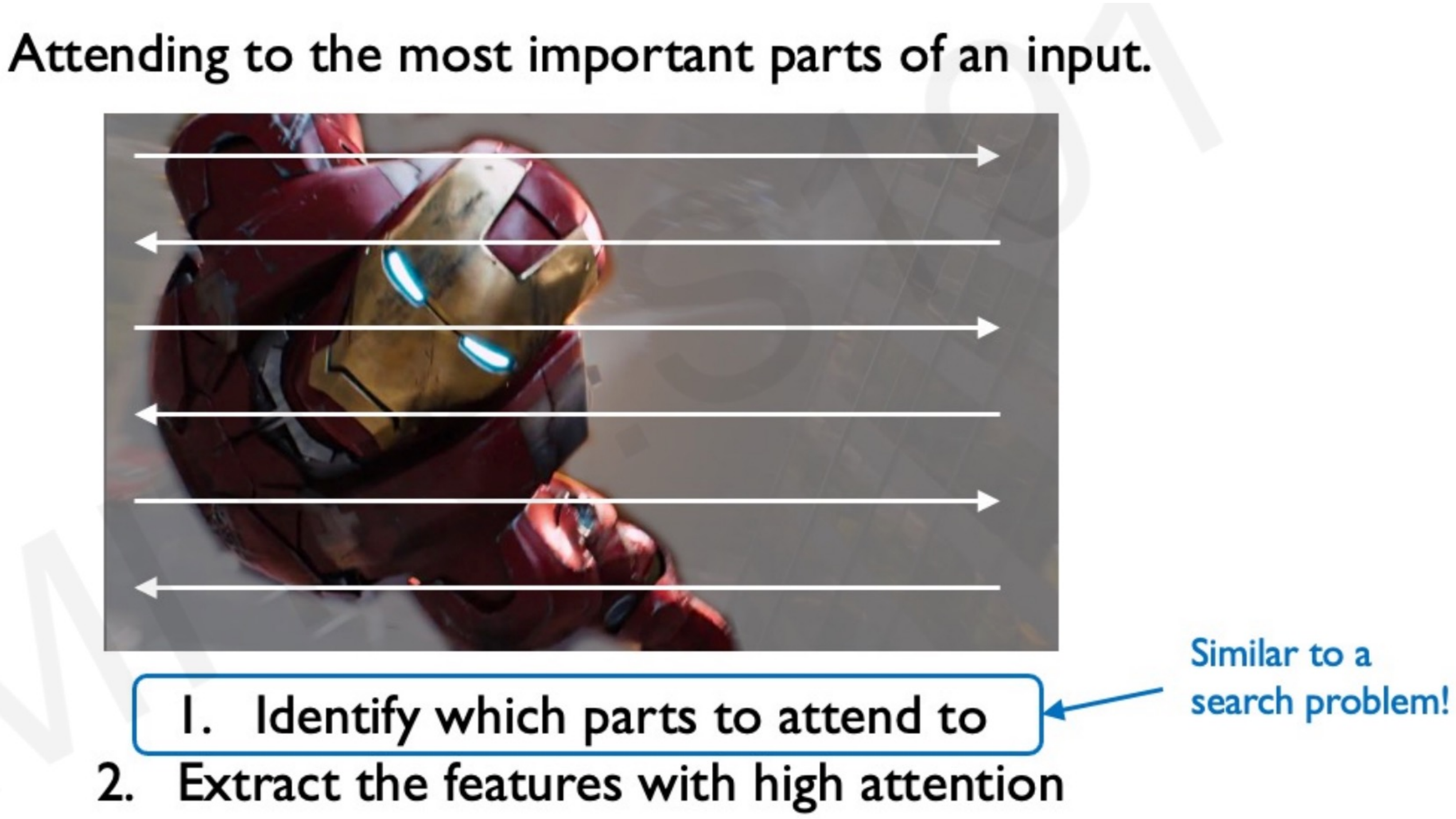
Limitations of RNNs

- Encoding bottleneck
- Slow, no parallelization
- Not long memory

RNNs: recurrence to model sequence dependencies

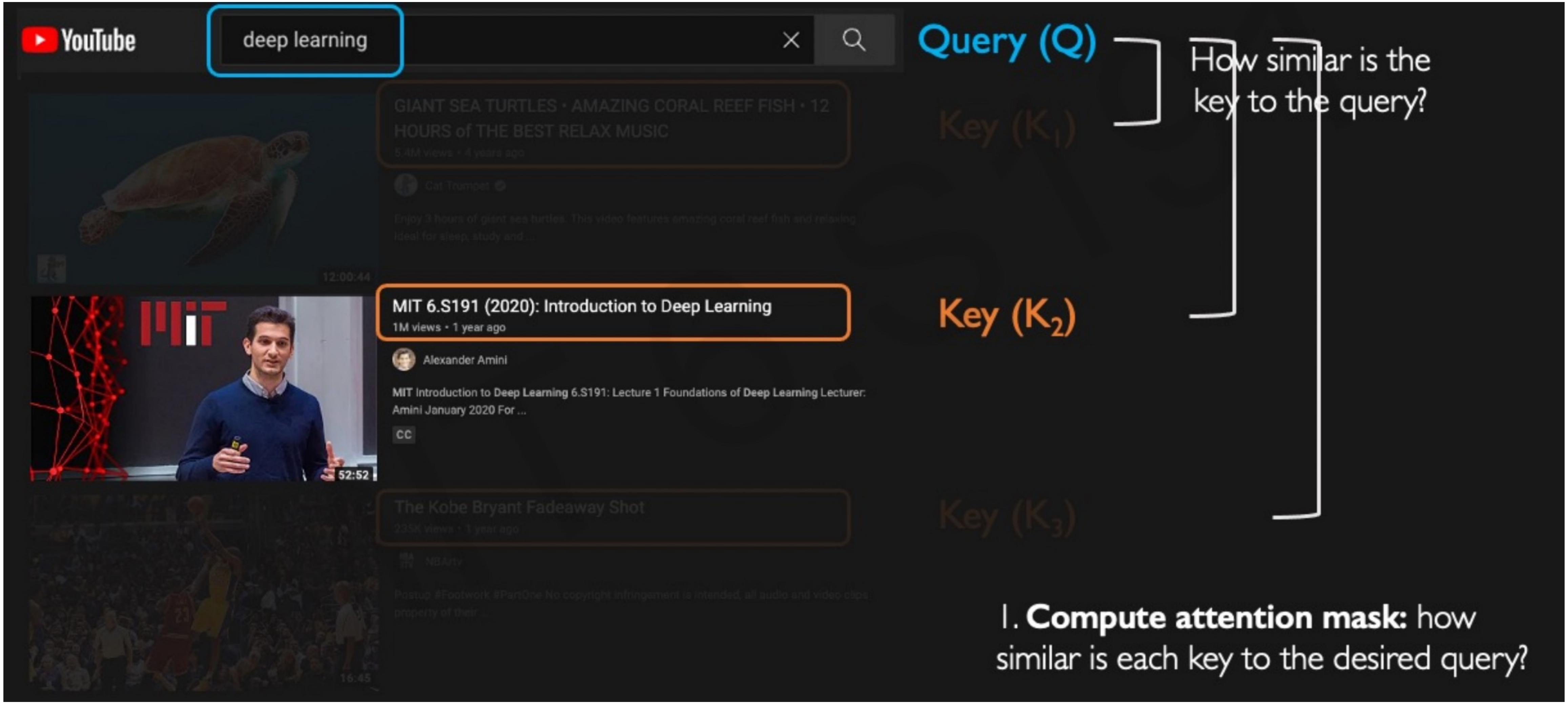


Intuition Behind Self-Attention

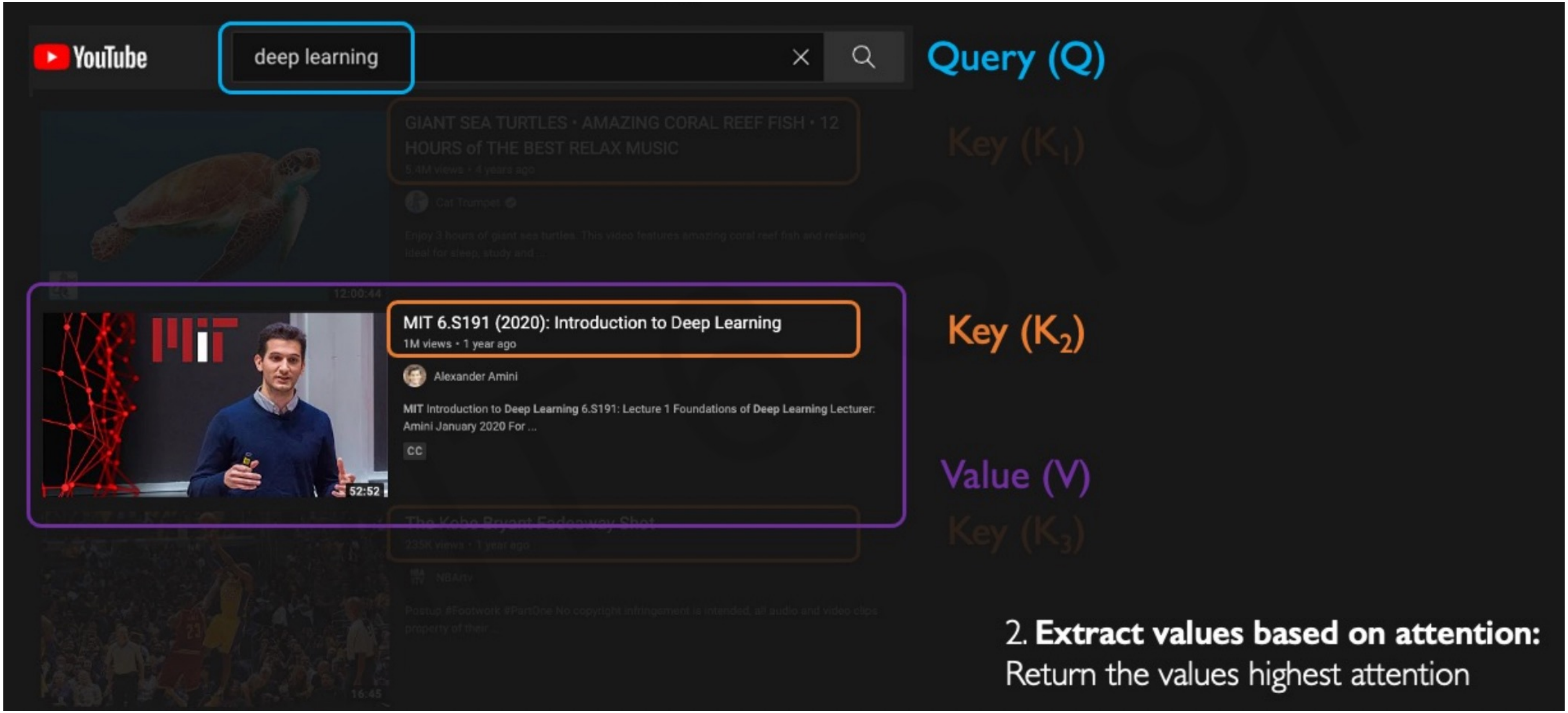


Vaswani et al. Attention is All You Need. NeurIPS 2017

Understanding Attention with Search



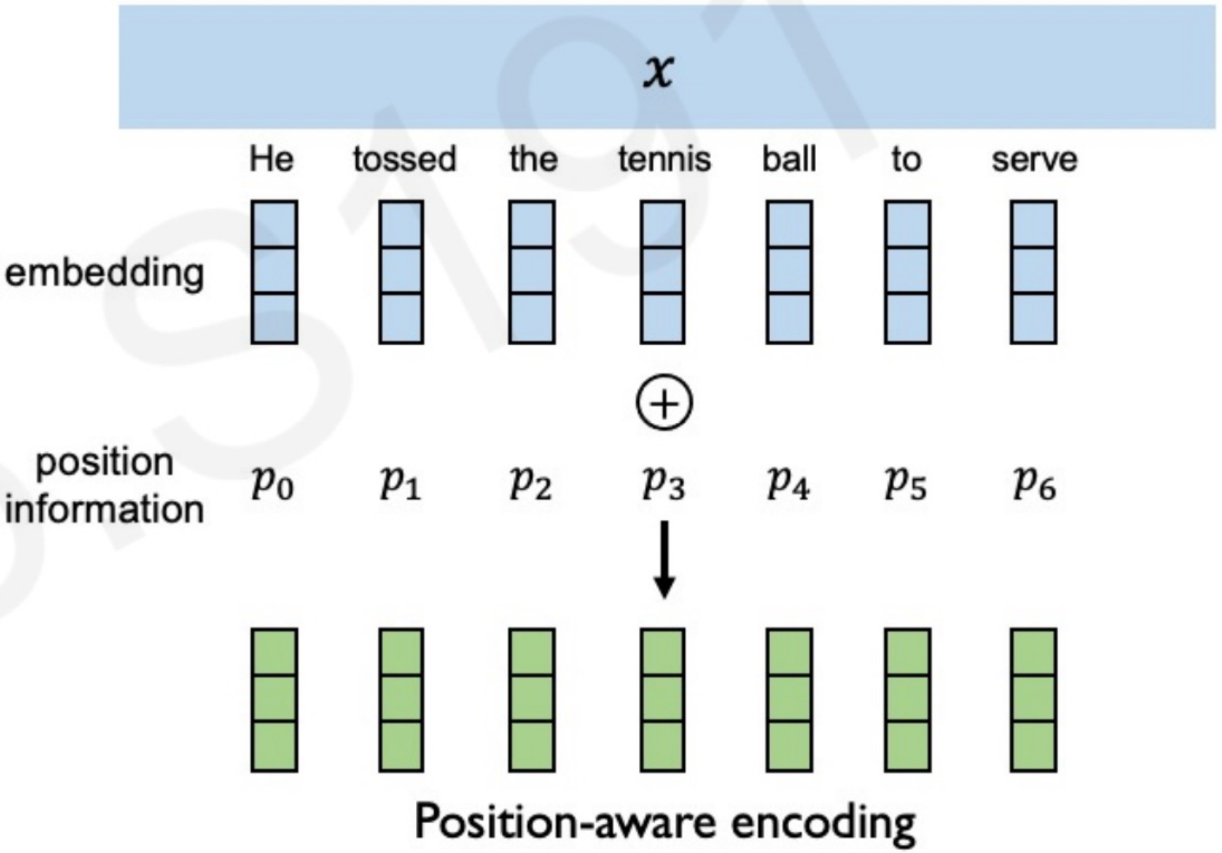
Understanding Attention with Search



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**

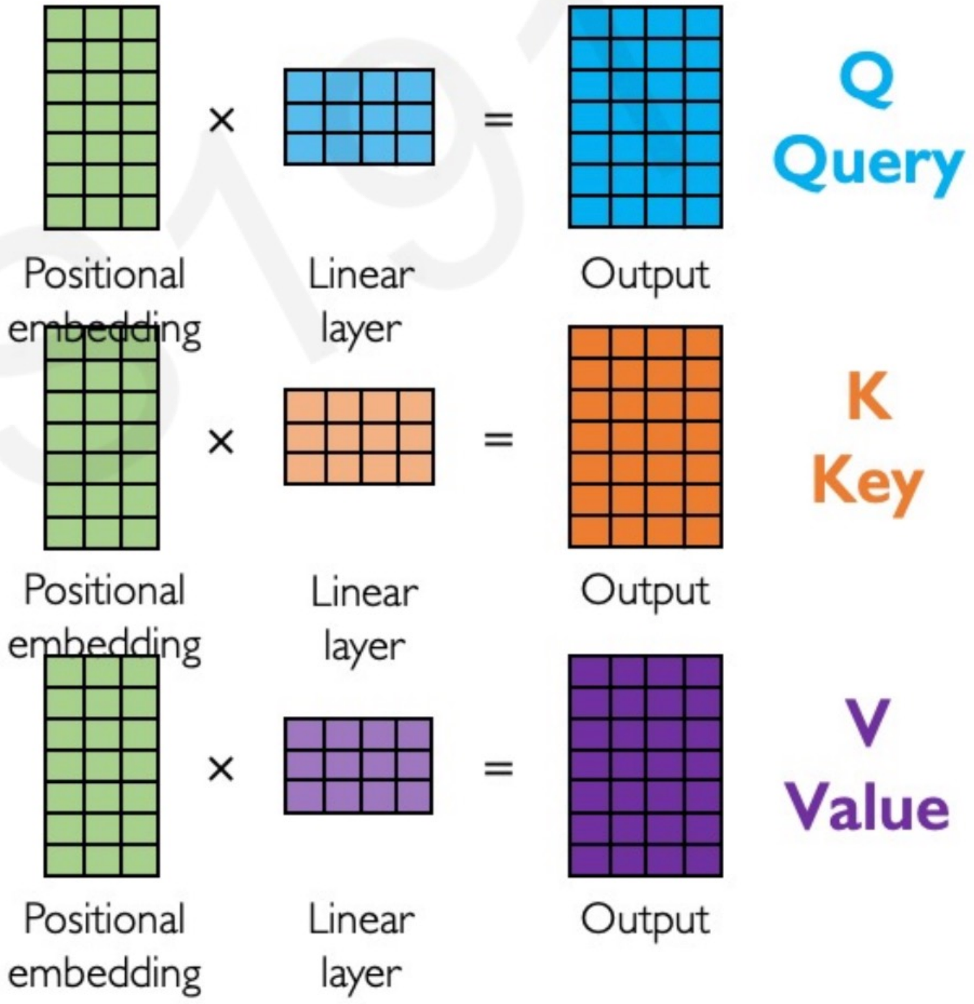


Data is fed in all at once! Need to encode position information to understand order.

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**



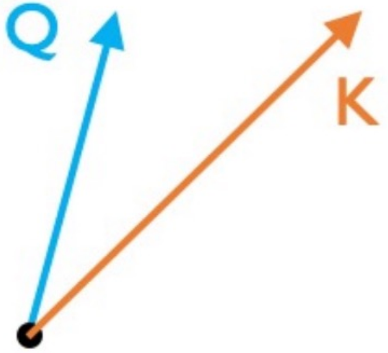
Learning Self-Attention with Neural Networks

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

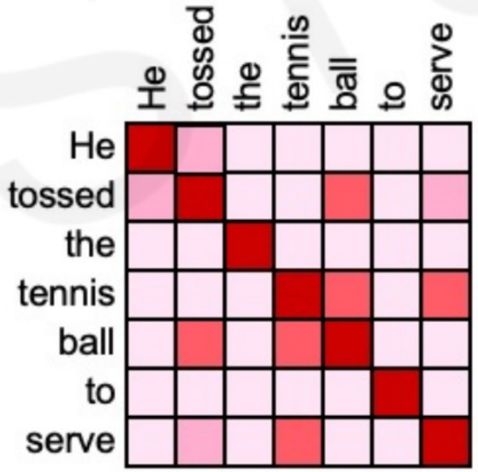
Also known as the "cosine similarity"

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?



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

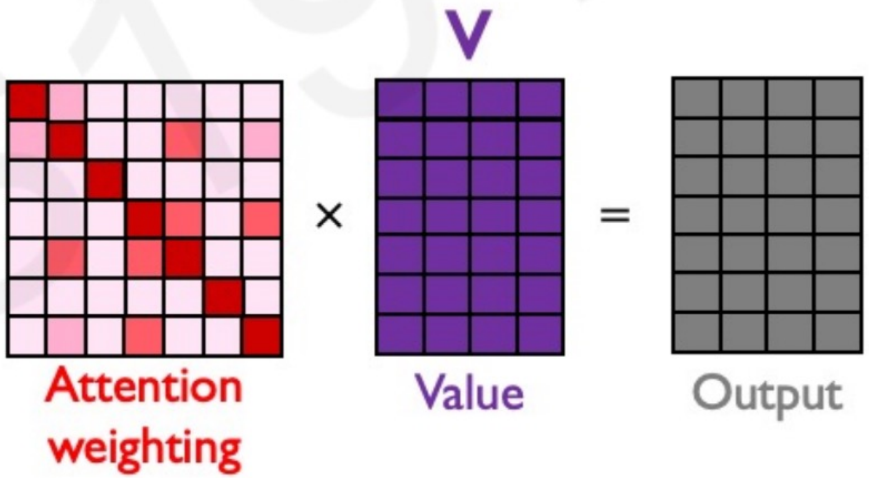
Attention weighting

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



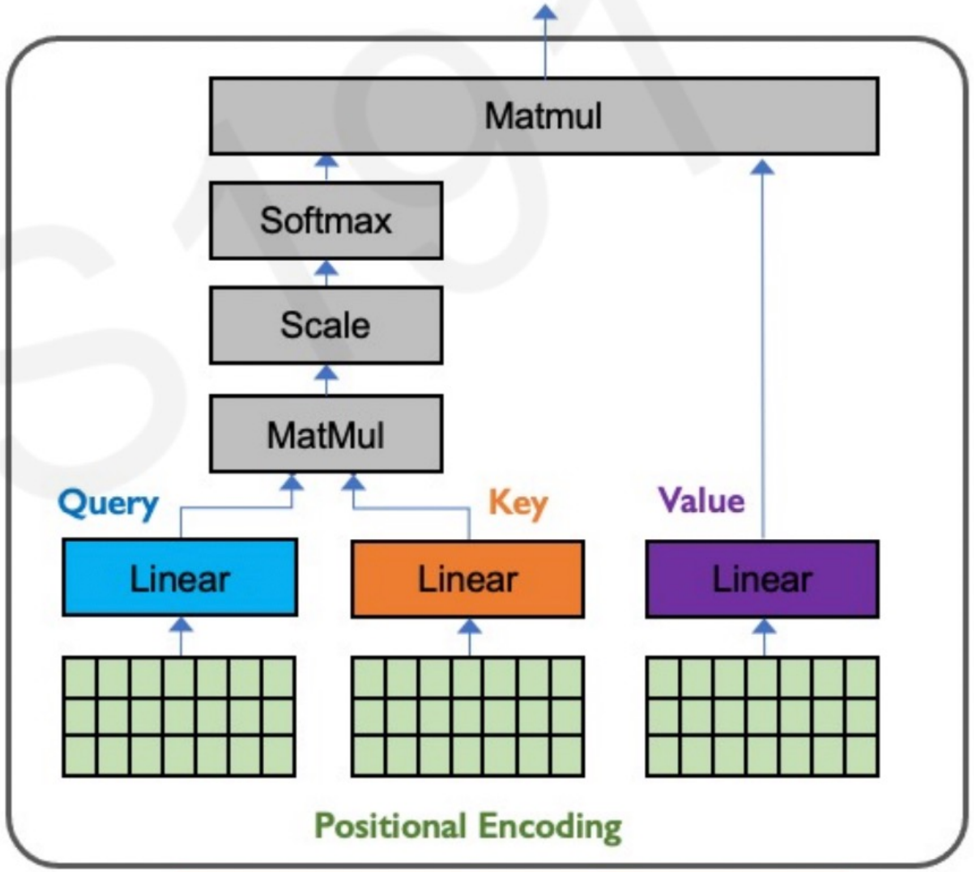
$$softmax\left(\frac{Q \cdot K^T}{scaling}\right) \cdot V = A(Q,K,V)$$

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.



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

Applying Multiple Self-Attention Heads

Attention weighting

×

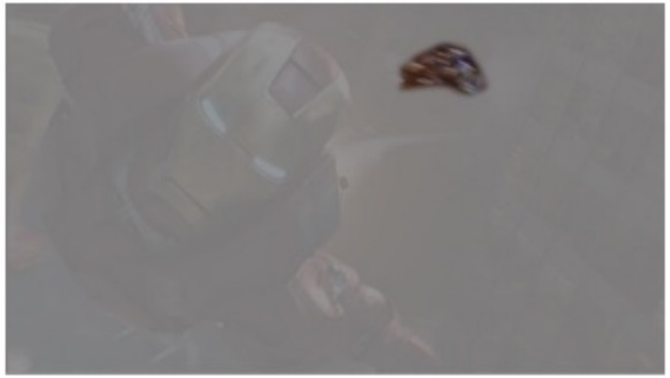
Value

=

Output

Output of attention head 1

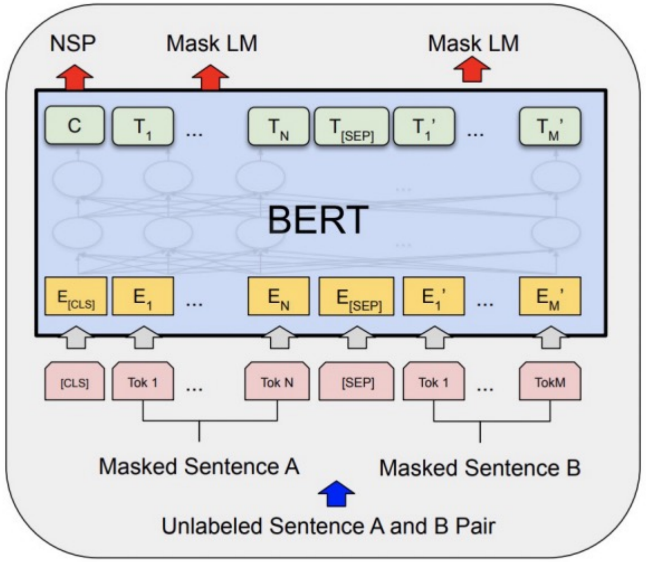
Output of attention head 2

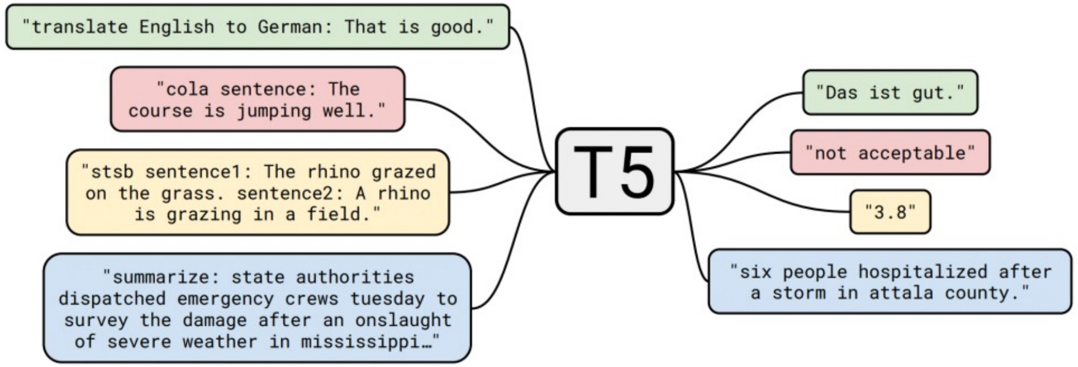


Output of attention head 3

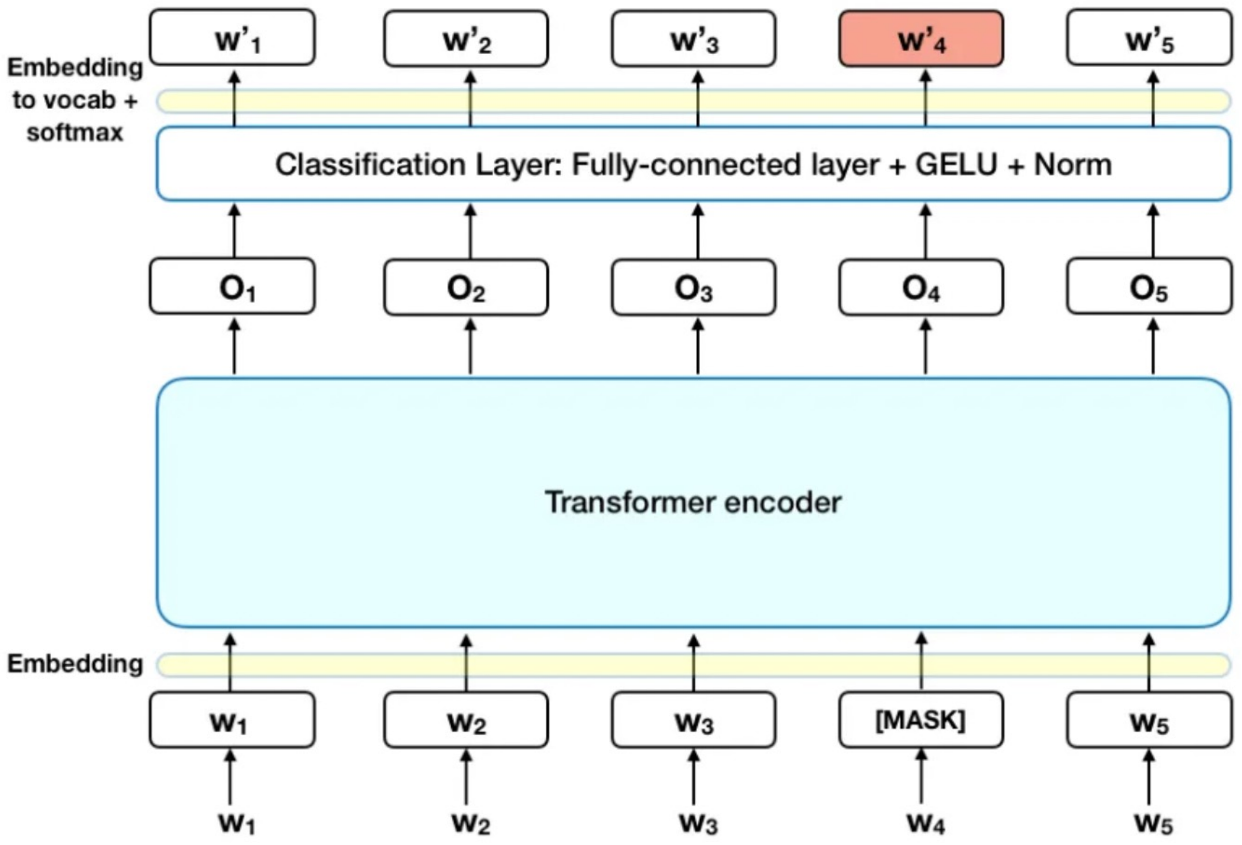
Types of Language Models

- Decoder-only models (GPT-x models)
- Encoder-only models (BERT, RoBERTa, ELECTRA)
- Encoder-decoder models (T5, BART)





BERT: Encoder Model

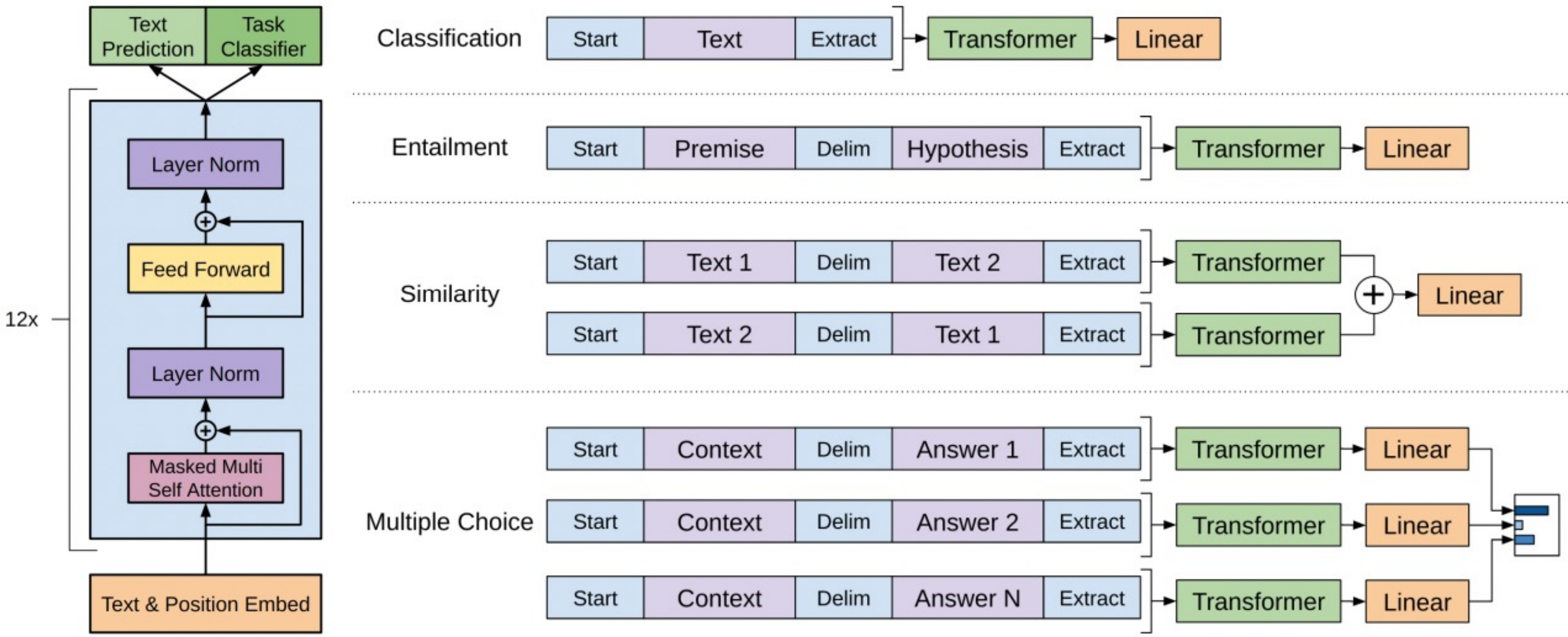


- Masked language model (MLM): mask 15% of words and try to predict
- Encoder useful for learning embeddings

GPT Training

Unsupervised training

Supervised fine tuning



Given unsupervised corpus of tokens

$$\mathcal{U} = \{u_1, \ldots, u_n\}$$

Max likelihood

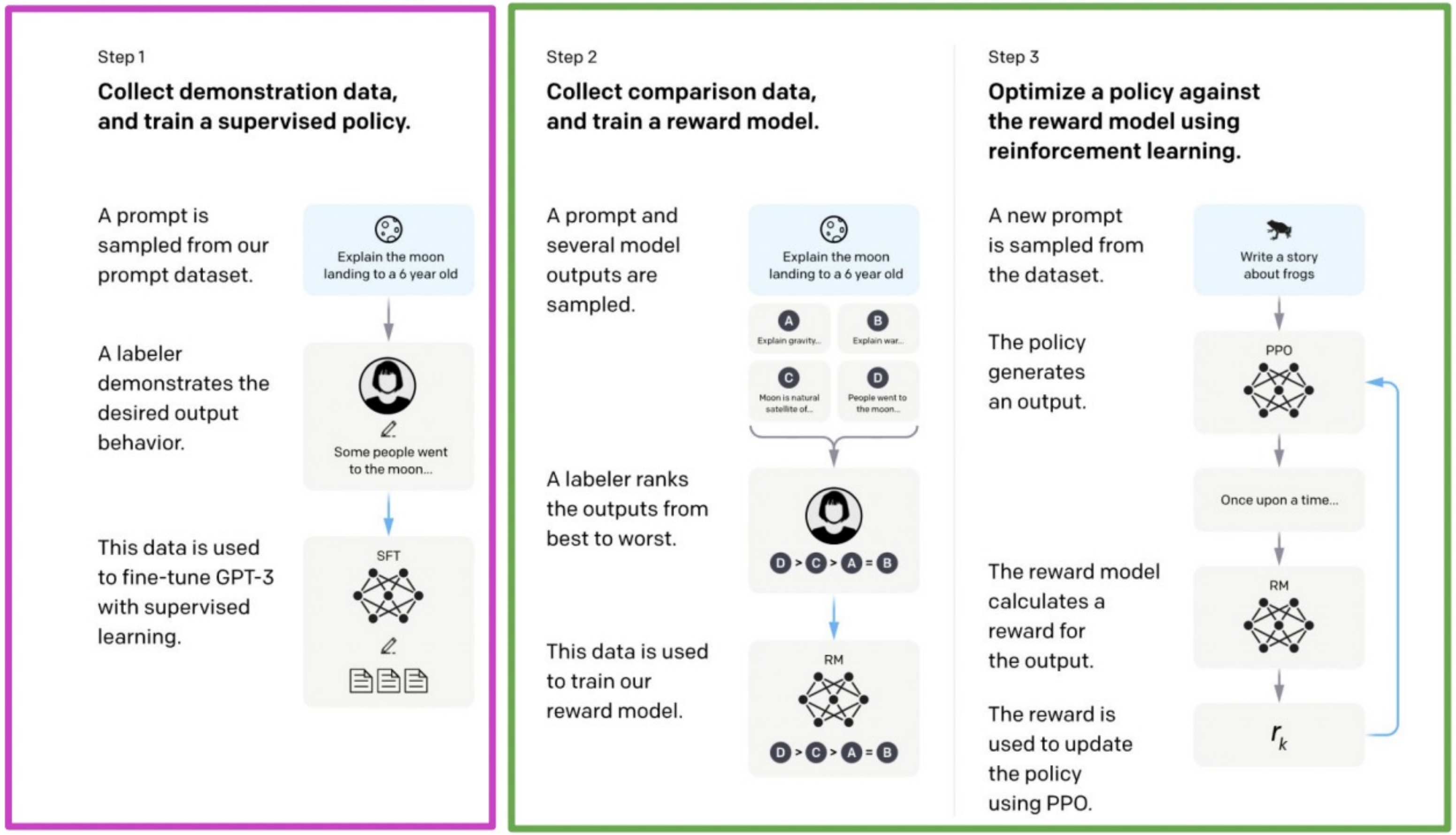
$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

- Decoder useful for generating text (sample from tokens of max likelihood)
- Radford et al. Improving Language Understanding by Generative Pre-Training

Training Chatbots

Supervised Fine-tuning
(instruction following and chatty-ness)

Reinforcement learning with human feedback (RLHF)
(aligning to target values and safety)



Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *arXiv preprint arXiv:2203.02155* (2022).

Acknowledgements

- Slides made using resources from
  - Alexander Amini and Ava Amini - MIT Introduction to Deep learning Course
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Thanks!