Jheronimus Academy of Data Science

EINDHOVEN UNIVERSITY OF TECHNOLOGY

On Deep Learning

An Overview

João Pereira

PDEng Data Science Talks

October 2019

Outline – Part 1

15/10/2019

- Introduction to deep learning
- Types of learning
- Historical Perspective
- Neural networks
- Training Neural Networks
- Optimization Challenges
- Regularization strategies
- Generalization

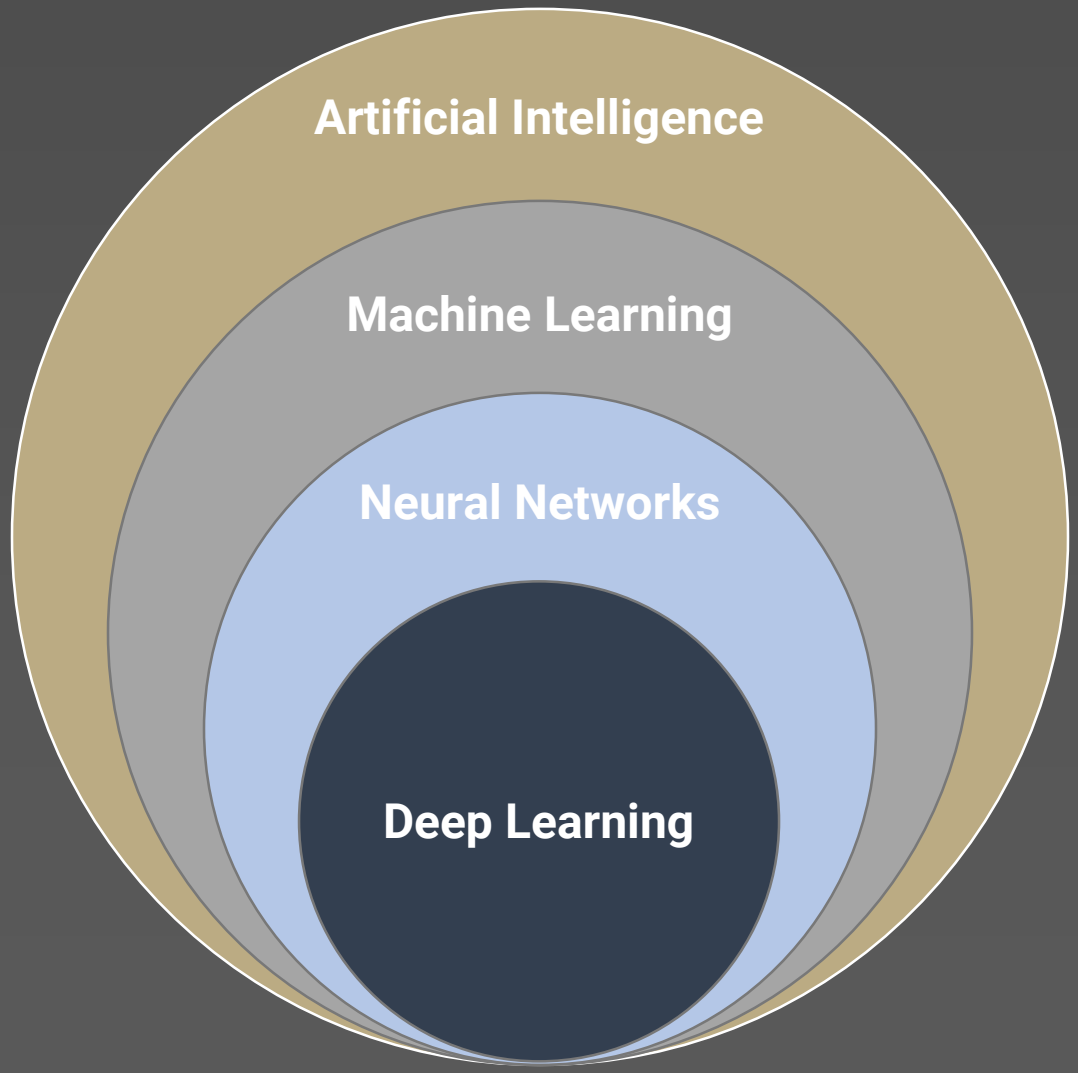
Outline – Part 2

22/10/2019

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
  - Vanilla RNNs
  - LSTMs and GRUs
- Sequence to Sequence Models (Seq2Seq)
- Attention Mechanisms
- Applications

Part 1

Big Picture


Artificial Intelligence
Machine Learning
Neural Networks
Deep Learning

Learning from data

Inputs and Outputs

- **Inputs**
  - Words, sentences
  - Images, videos
  - Observations, time series
  - Voice

x

- **Outputs**
  - Translation
  - Class label

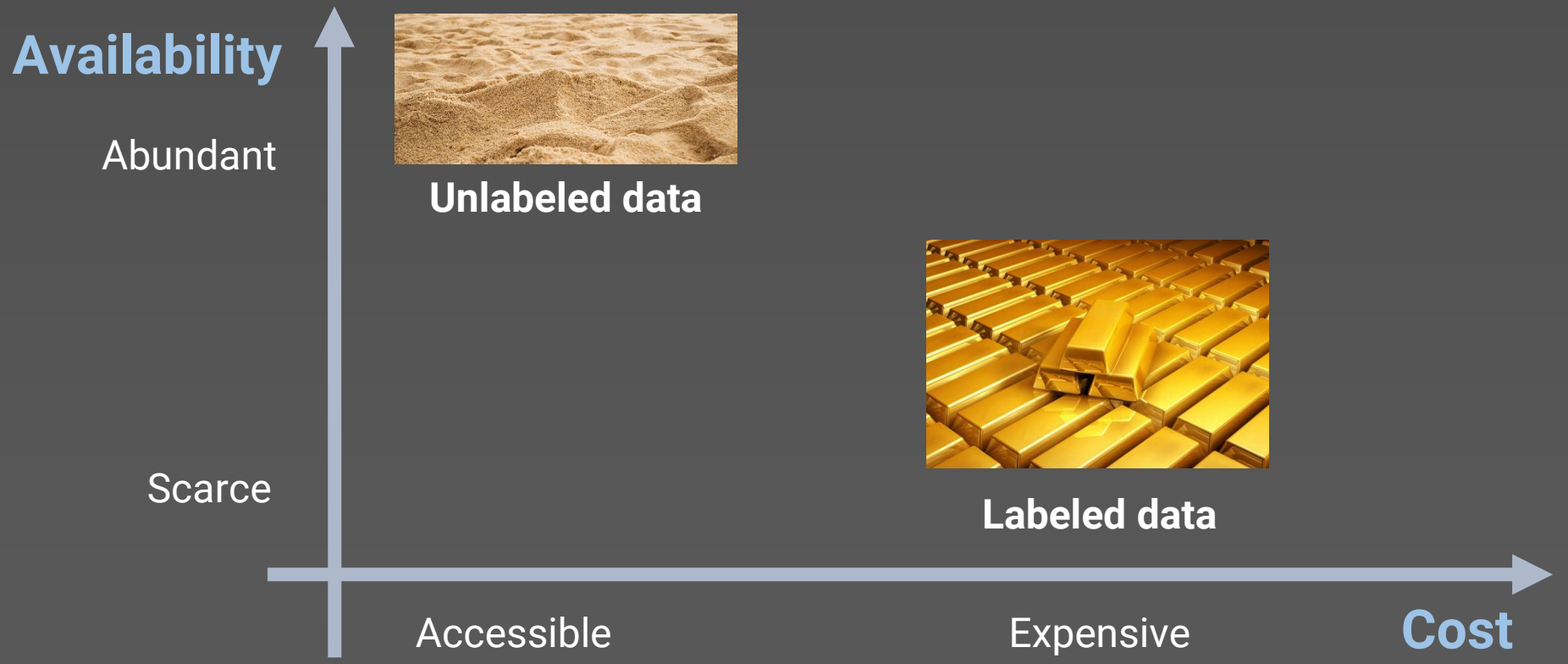
y

What is learning?

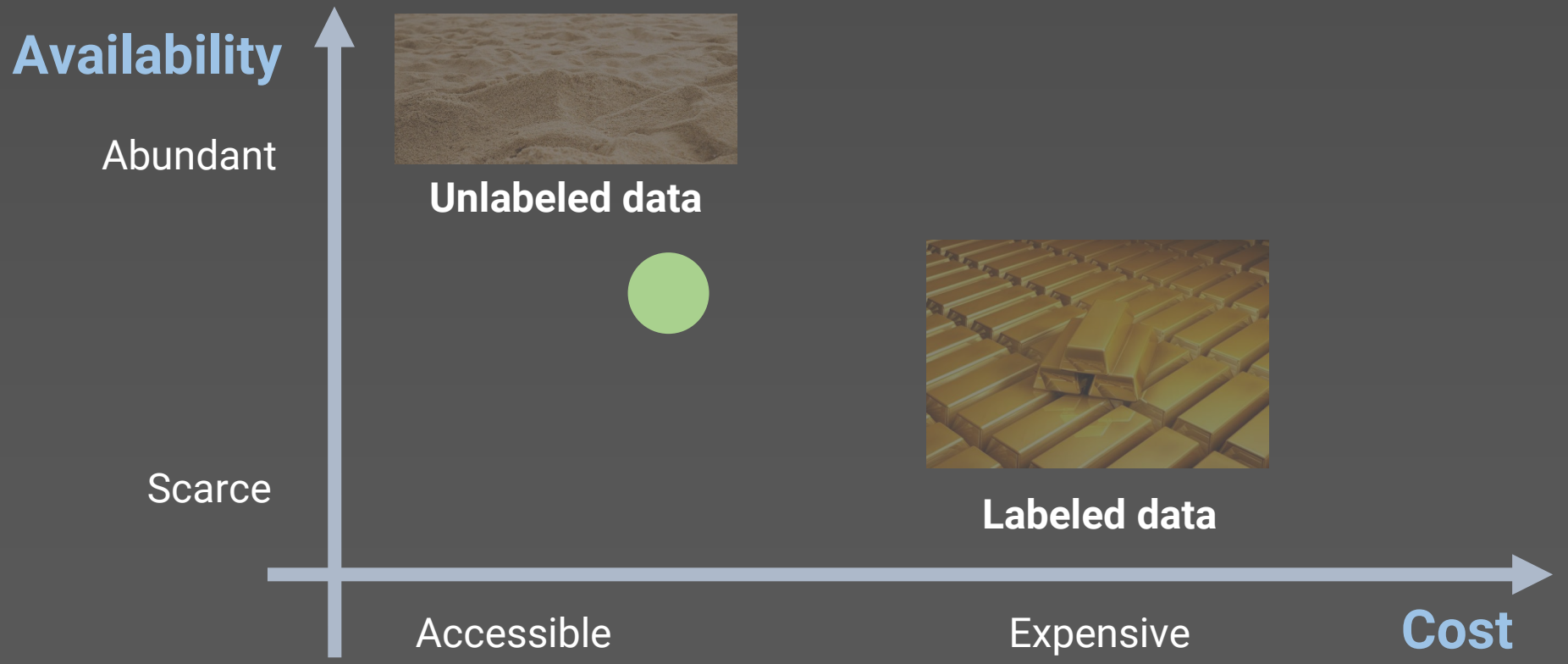
- Learning or training refers to estimate the parameters of a model.

Machine learning is about generalizing to unseen data.

What data is out there?



Data in the PDEng Data Science



Types of learning

- Supervised learning
- Unsupervised learning
- Reinforcement learning

LeCun's Cake Analogy

Y. LeCun

How Much Information is the Machine Given during Learning?

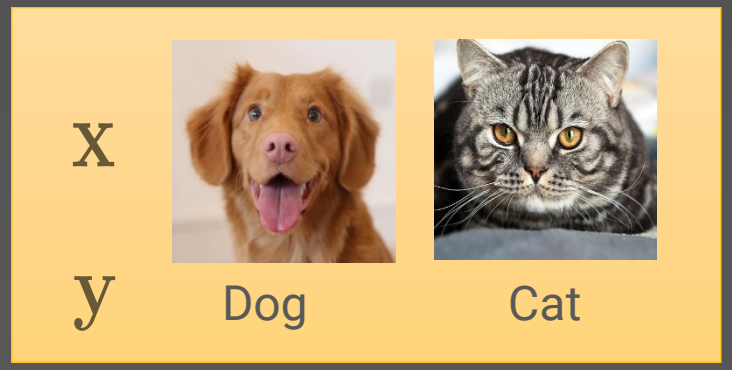
- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

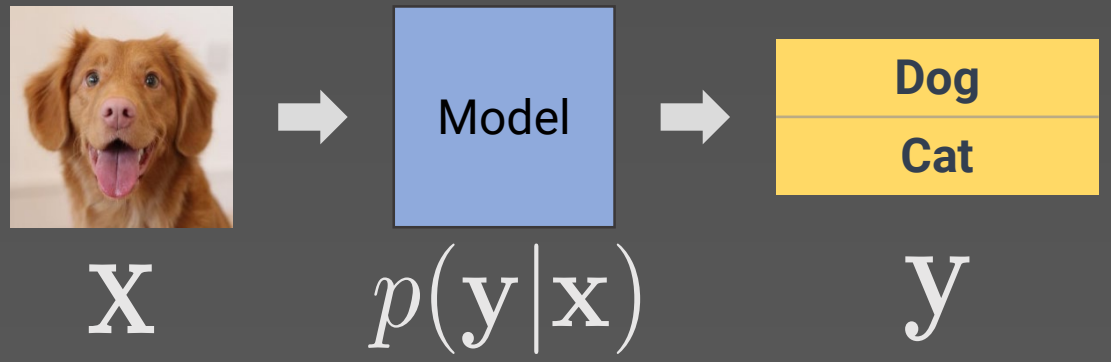
Supervised Learning

- Given a dataset D of inputs x and labeled targets y,

  ***learn* to predict** y from x.

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$$



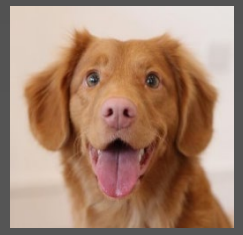


- Most successful paradigm in deep learning.

Unsupervised Learning

- Given only the inputs $\mathbf{x}$, models $p(\mathbf{x})$ and find

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$$



$\mathbf{X}$

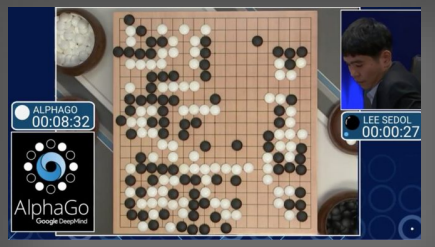
Model

- Patterns
- Structure
- Clusters
- Generation

Why Did Deep Learning Become So Popular?

Important breakthroughs in AI-class of problems

- Speech recognition
- Image classification
- Machine translation
- Chatbots
- Self-driving cars
- Playing games



Why Now?

Among other reasons:

- Fifty years of AI research in machine learning
- Explosion of convenient compute power
  (*e.g.*, GPUs, TPUs)

- Lots of (labeled) data
- New software and tools

- Strong investments and interest of large organizations

IBM **Research** AI

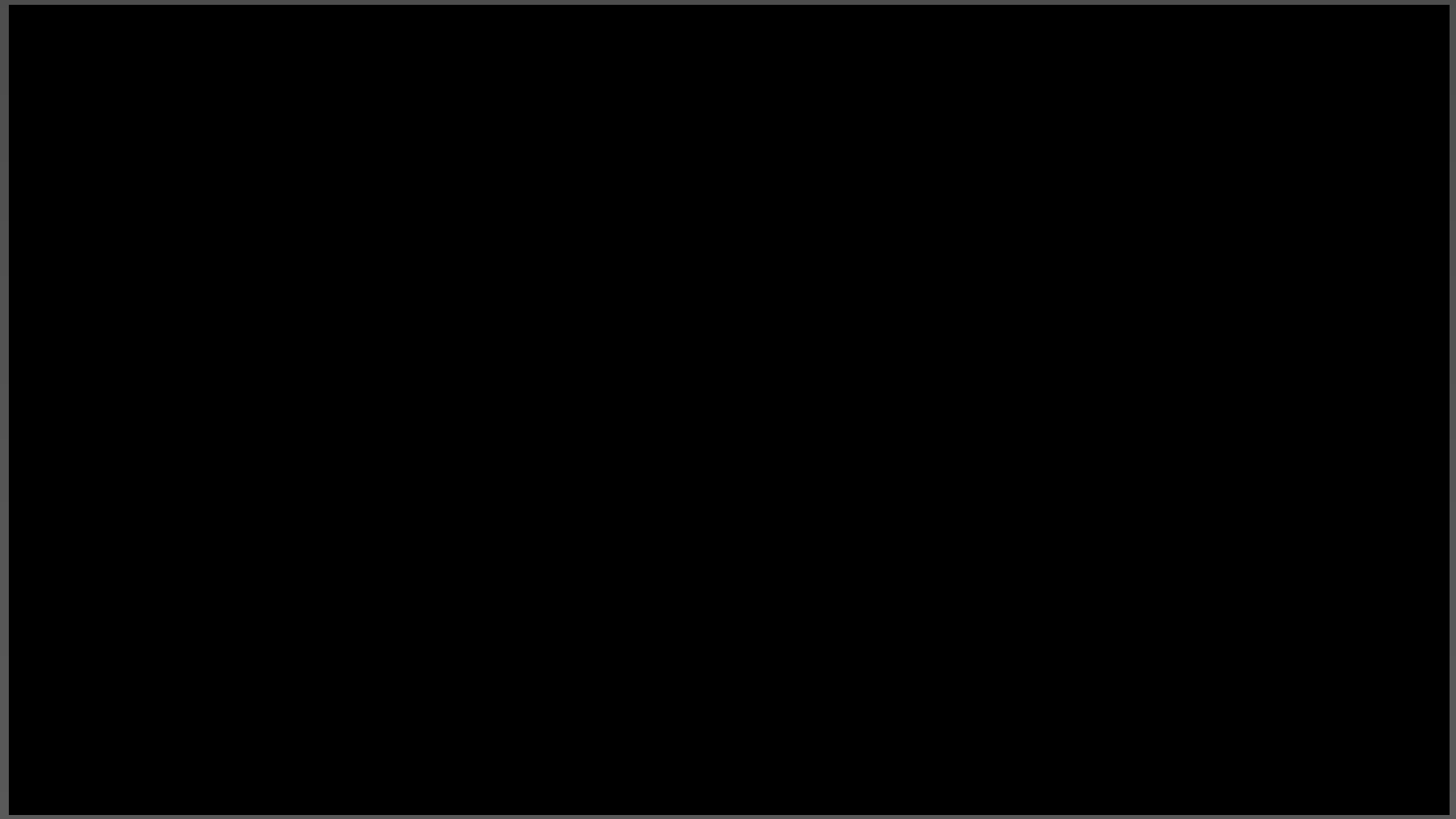
Facebook AI Research

Google AI

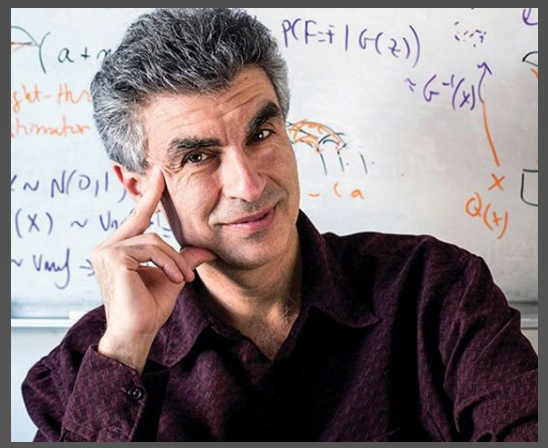
Baidu 百度

Cepticism and AI winters

Link



Deep Learning Heroes



Yoshua Bengio
Université de Montreal

Geoffrey Hinton
University of Toronto
Google Brain

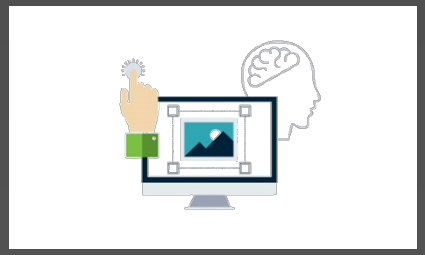
Yann LeCun
New York University
Facebook AI Research

Turing Award

Why Deep Learning?

Traditional Machine Learning

Input



Feature Extraction

Classifier

Classification

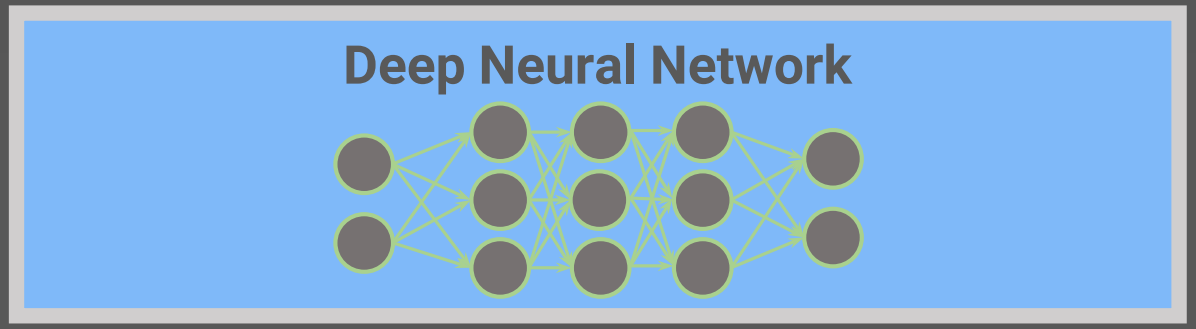
Duck

Not a Duck

Output

Deep Learning

Input



Feature Extraction + Classification

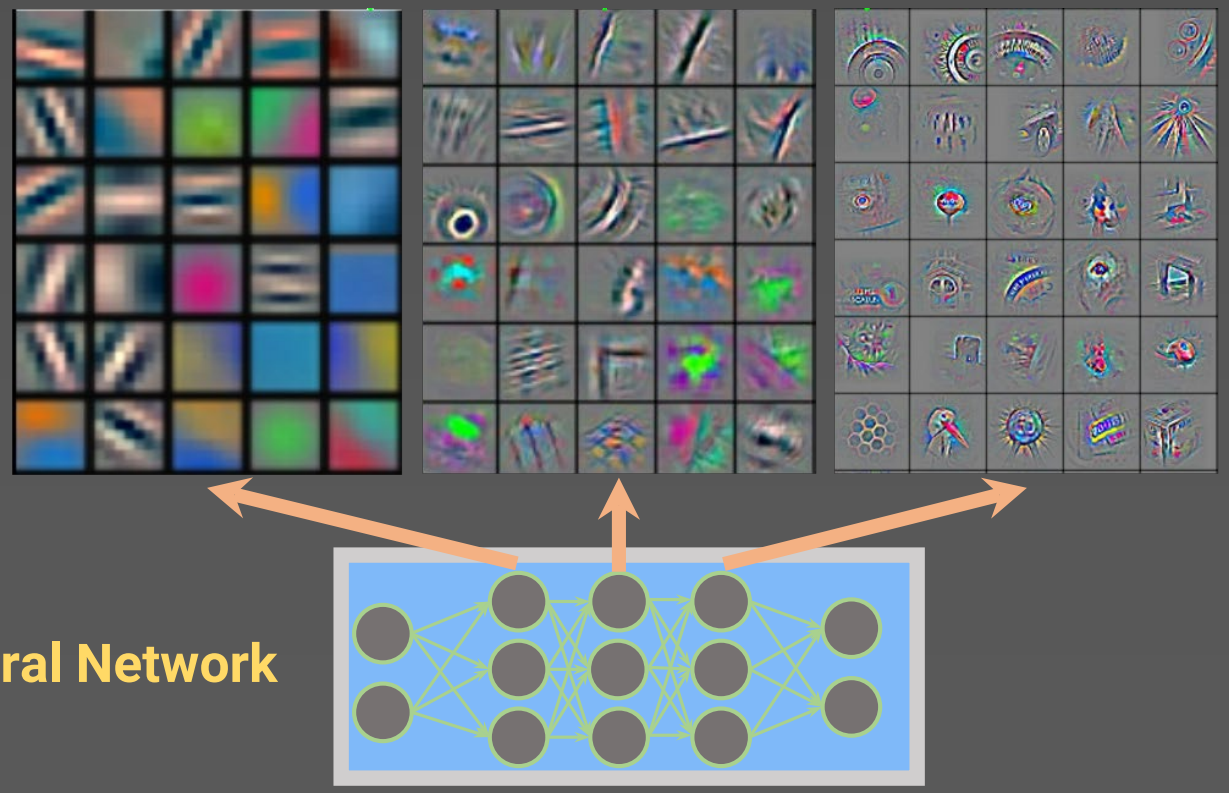
Duck

Not a Duck

Output

Why Deep Learning?

- Hand-crafting features is time consuming and not scalable.
- We can learn the underlying features from data, **automatically.**



Deep Neural Network

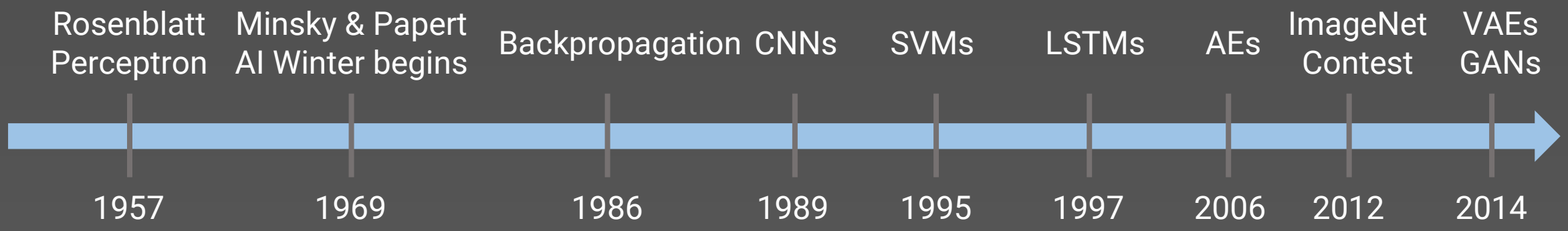
Zeiler & Fergus, 2013
Example by Yann LeCun

Deep learning is about representation learning!

Learning good features for downstream tasks (regression, classification, …)

Hand-crafting *vs* learning features

Historical Perspective

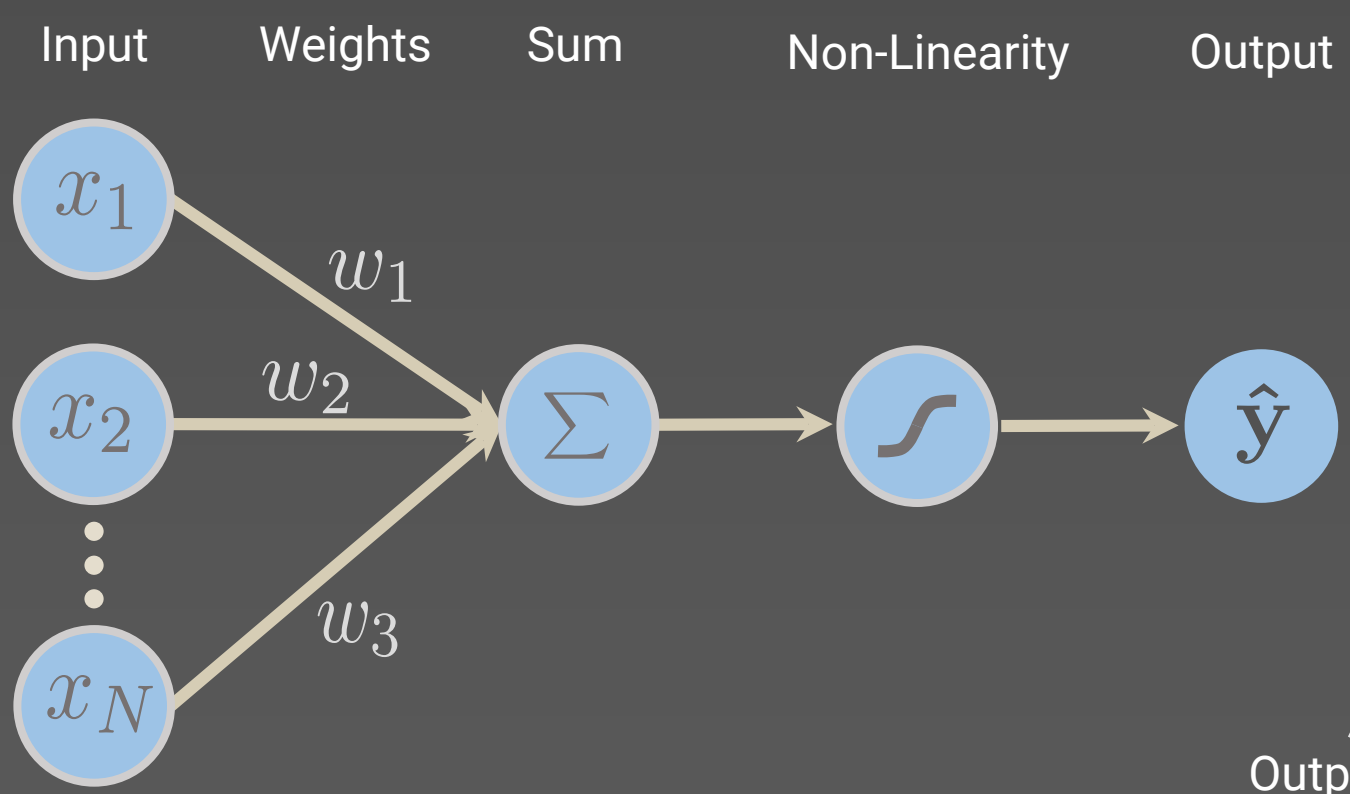


Perceptron

Structural building block of deep learning

The Perceptron

Rosenblatt, 1958



$$\hat{y} = g\left(\sum_{i=1}^{N} x_i w_i\right)$$

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{N} x_i w_i\right)$$

Output Prediction

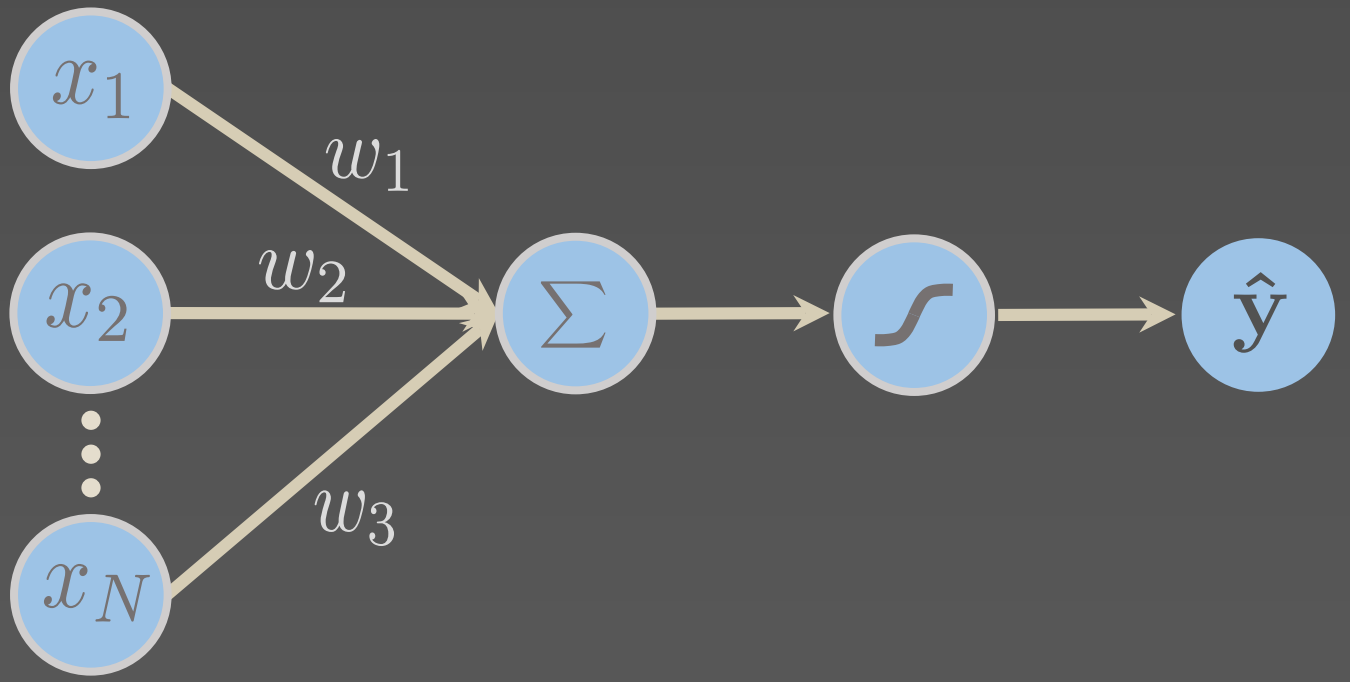
Non-linearity

Bias

Inputs

Weights

The Perceptron



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{N} x_i w_i\right)$$

In matrix form:

$$\hat{y} = g(w_0 + \mathbf{x}^\top \mathbf{w})$$

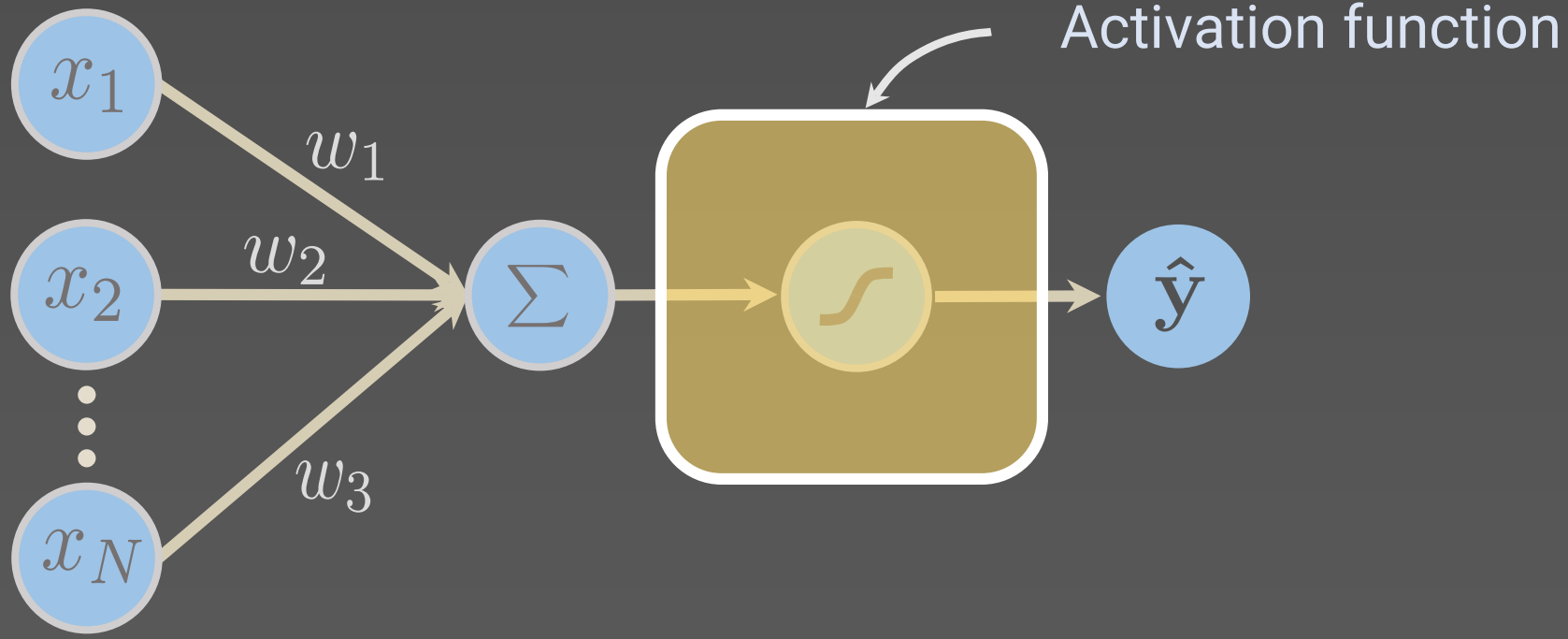
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{bmatrix}$$

Activation Functions

Sigmoid
Hyperbolic Tangent
Rectified Linear Unit
Softmax

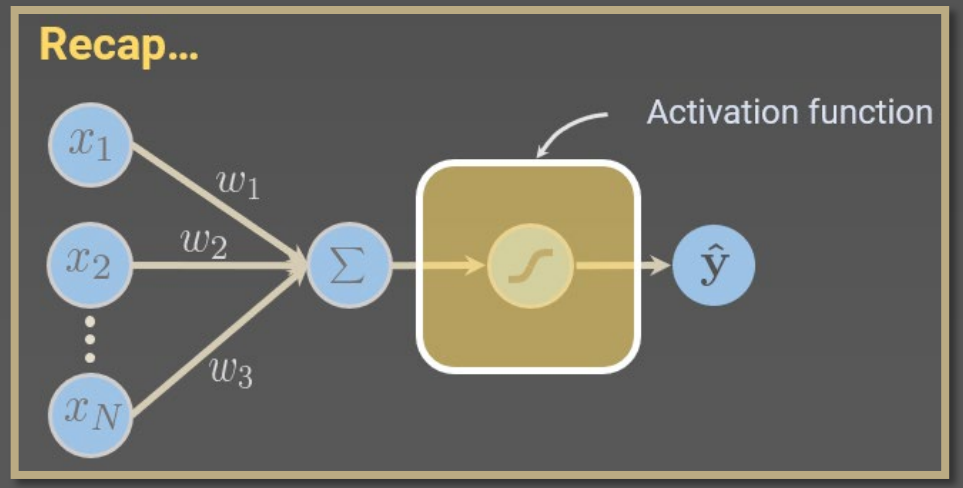
Activation Functions

Recap...



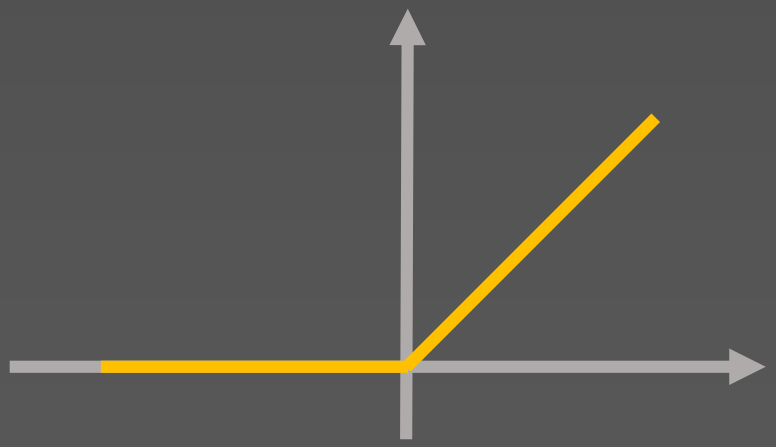
Activation Functions

- The goal of the activation functions is to introduce **non-linearities** into the network
- The are many of these functions
- They are all **non-linear**



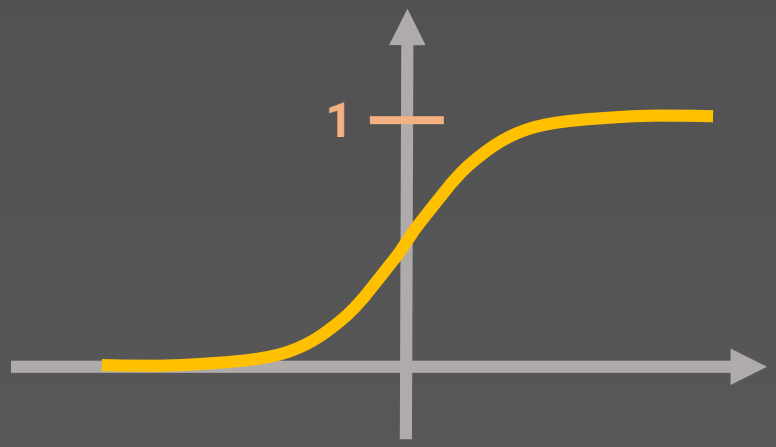
Activation Functions

Rectified Linear Unit
ReLU



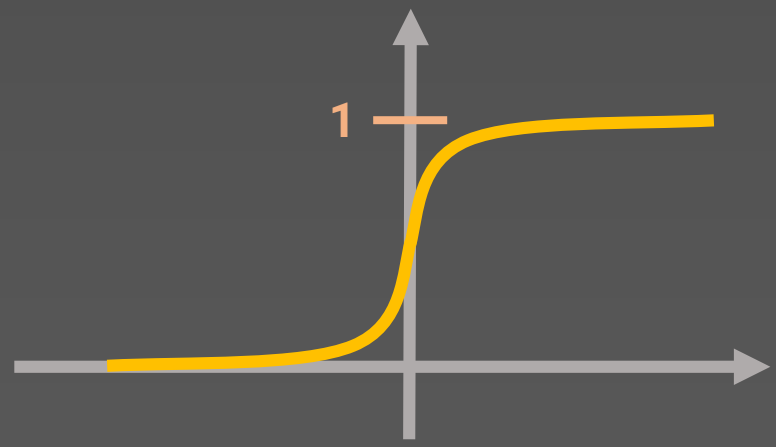
`tf.keras.activations.relu(x)`

Sigmoid



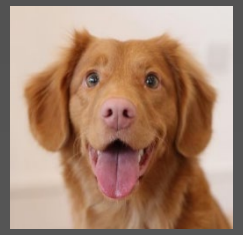
`tf.keras.activations.sigmoid(x)`

Hyperbolic Tangent



`tf.keras.activations.tanh(x)`

Activation Functions: Softmax



Neural Network

Output

$$\begin{bmatrix} 2.0 \\ 19.0 \end{bmatrix}$$

Softmax

Probabilities

$$\begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}$$

Cat

Dog

$\hat{\mathbf{y}}$

Targets

$$\begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

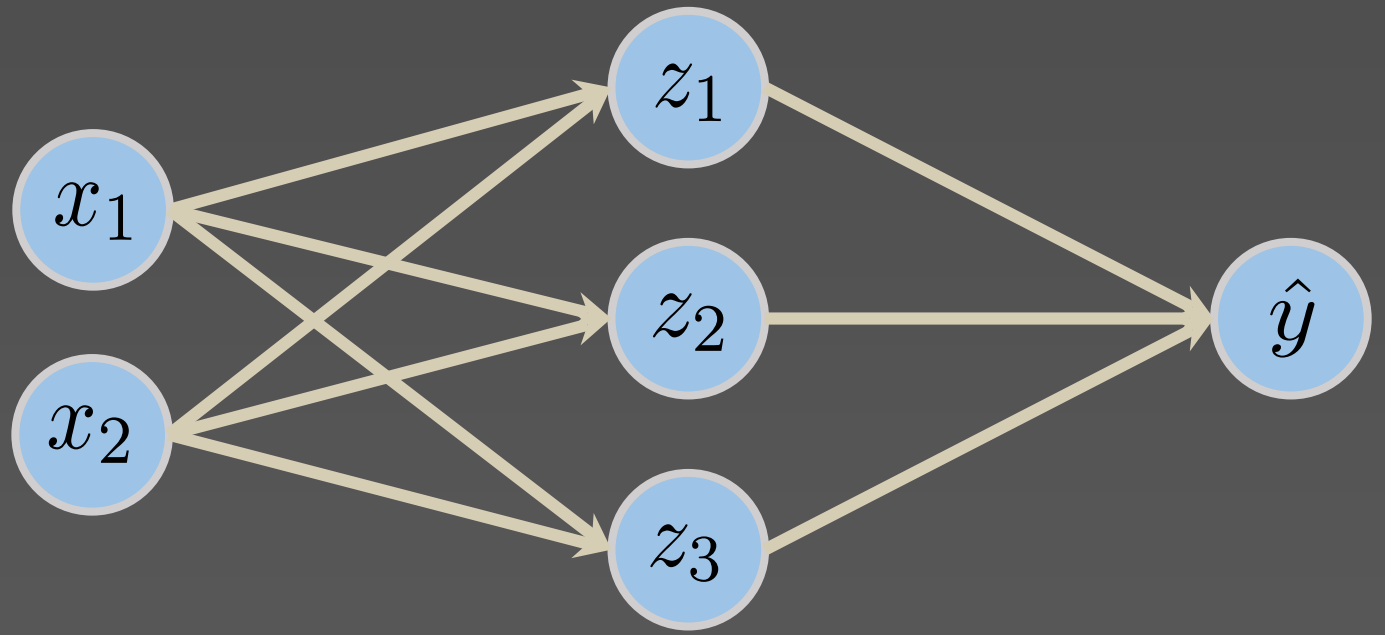
$\mathbf{y}$

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

```
tf.keras.activations.softmax(x)
```

Building Neural Networks with Perceptrons

Single-Layer Neural Network



Properties

Neural networks are universal function approximators

Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of Rn, under mild assumptions on the activation function.

Cybenko, 1989

Properties

Neural networks are universal function approximators

Universal Approximation Theorem

In a nutshell:
They can approximate many functions up to any precision.

Cybenko, 1989

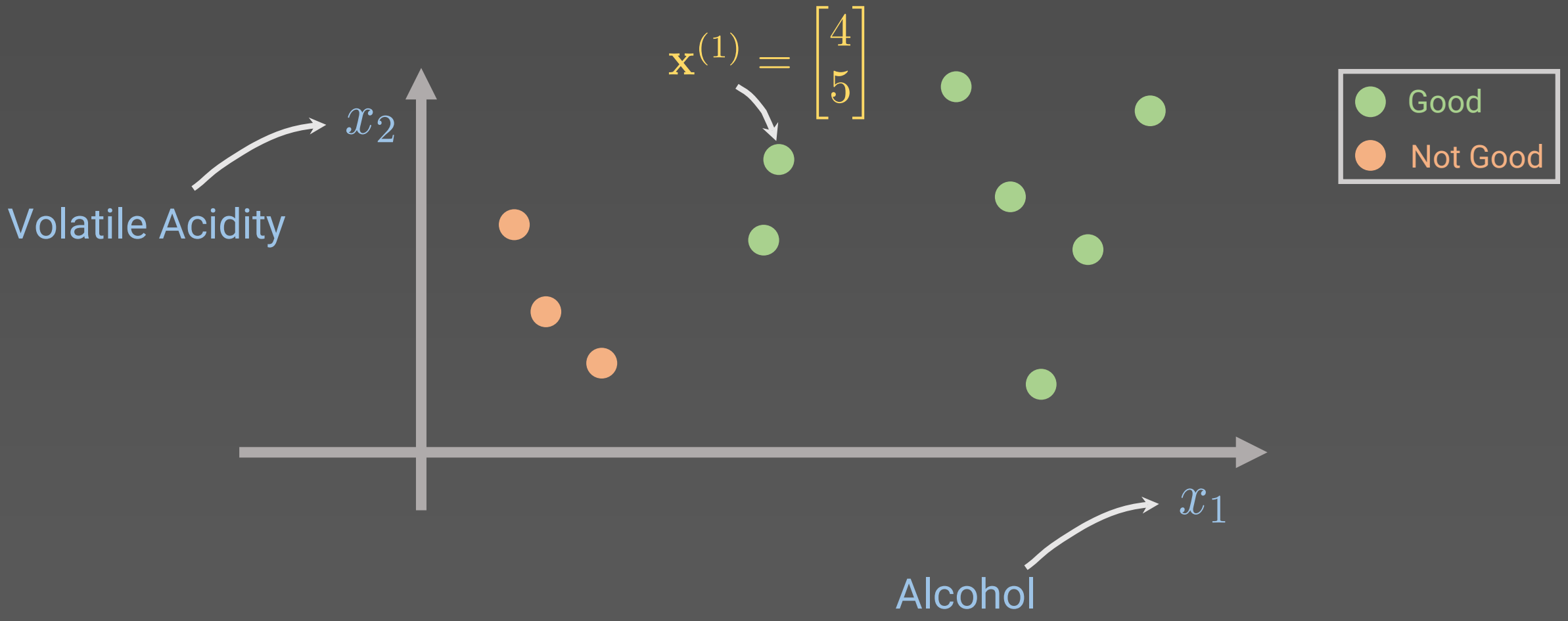
Training Neural Networks

Loss Functions

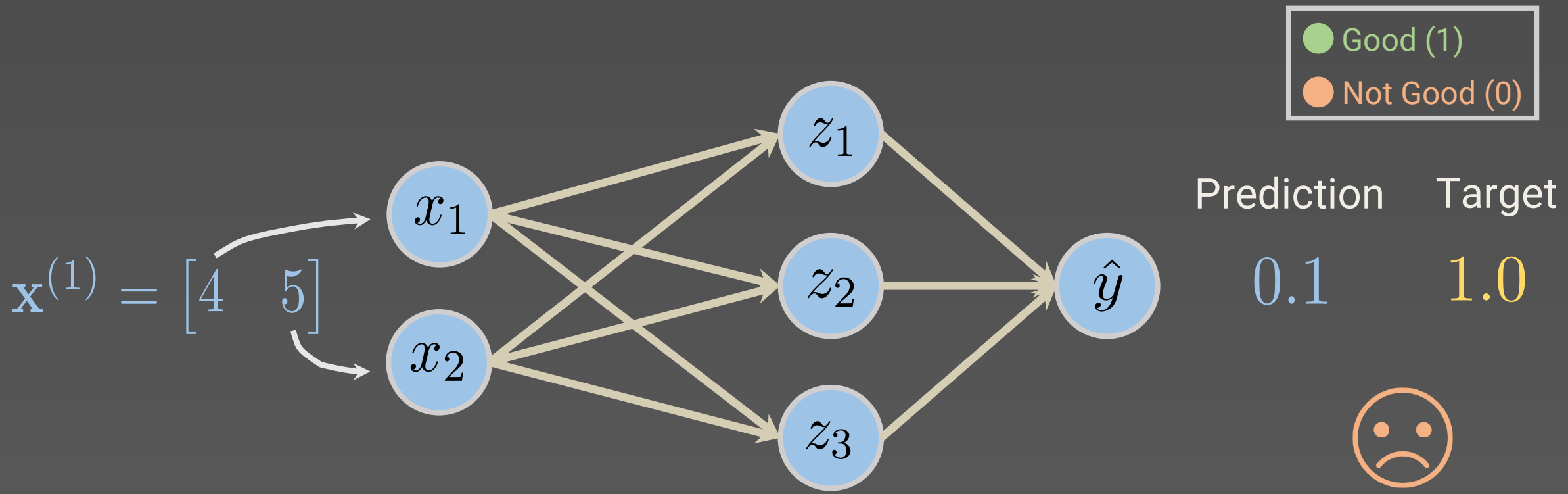
Gradient Descent

Stochastic Gradient Descent

Example: Wine Quality

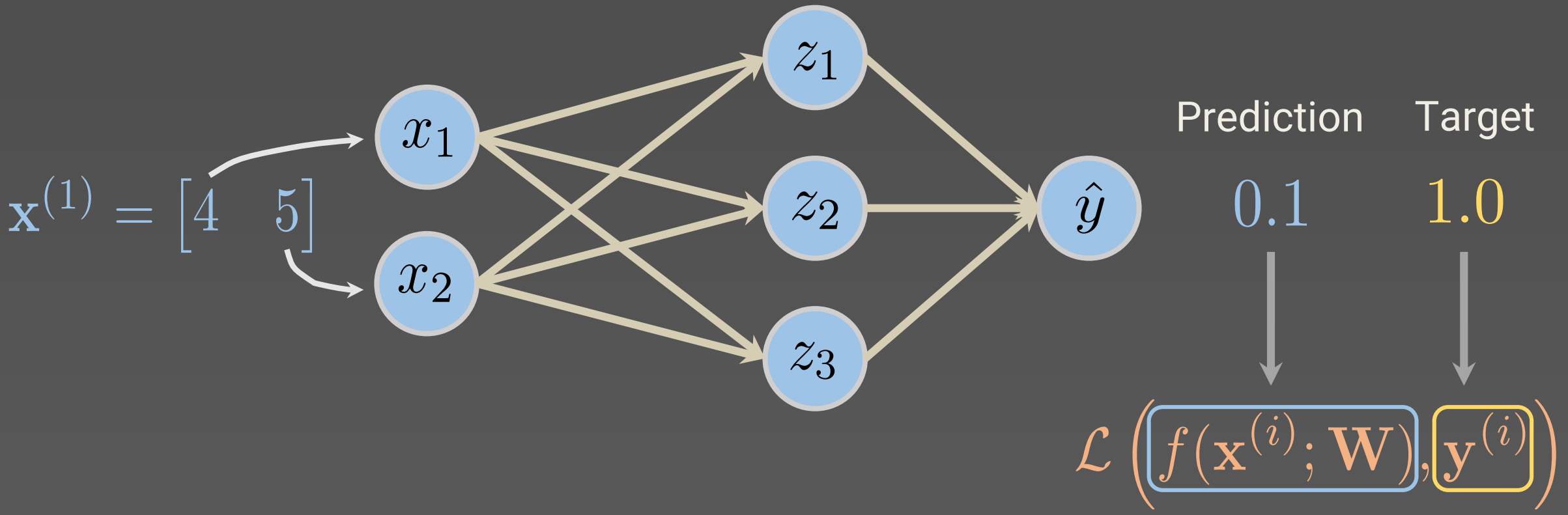


Example: Wine Quality



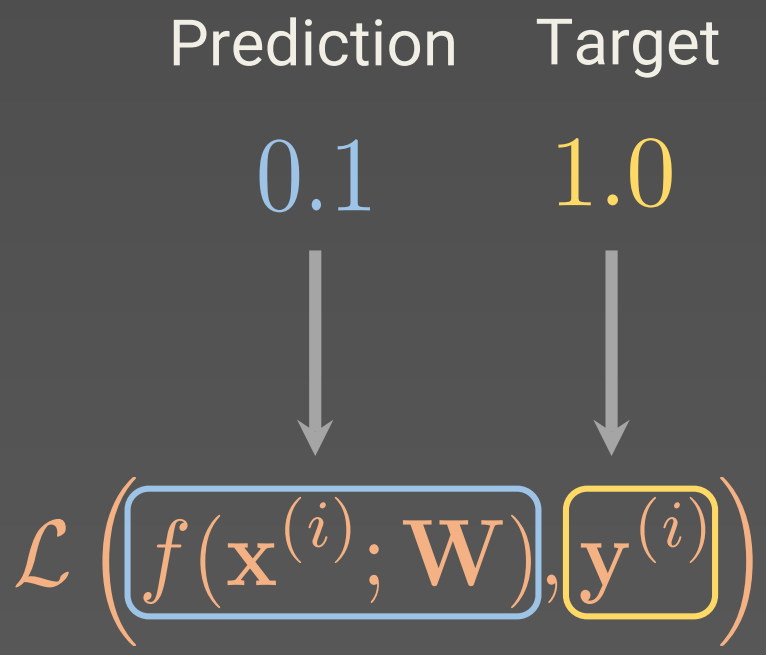
- How to quantify prediction error?

Loss



The loss measures the cost due to wrong predictions.

Loss



Dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$

$$\mathcal{J}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(f(\mathbf{x}^{(i)};\mathbf{W}),\mathbf{y}^{(i)}\right)$$

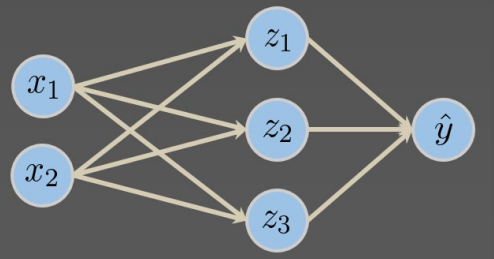
This is the **empirical loss**.
Aka

Objective function
Cost function
Empirical risk

Binary Cross Entropy Loss

- Used in **classification** problems with models that output a **probability** between 0 and 1.

$$\mathbf{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ \dots & \\ 5, & 8 \end{bmatrix}$$



Prediction

$$\begin{bmatrix} 0.1 \\ 0.8 \\ \dots \\ 0.6 \end{bmatrix}$$

Target

$$\begin{bmatrix} 1 \\ 0 \\ \dots \\ 1 \end{bmatrix}$$

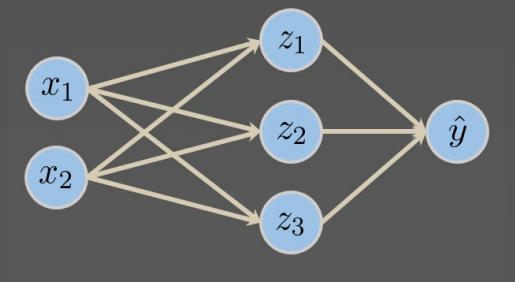
$$\mathcal{J}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} y^{(i)} \log\left(f(\mathbf{x}^{(i)}, \mathbf{W})\right) + (1 - y^{(i)}) \log\left(1 - f(\mathbf{x}^{(i)}, \mathbf{W})\right)$$

`tf.keras.losses.BinaryCrossentropy()`

Mean Squared Error Loss

- Used in **regression** problems with models that output **continuous real numbers**.

$$\mathbf{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ \dots & \\ 5, & 8 \end{bmatrix}$$



Prediction

$$\begin{bmatrix} 30 \\ 80 \\ \dots \\ 85 \end{bmatrix}$$

Target

$$\begin{bmatrix} 90 \\ 20 \\ \dots \\ 95 \end{bmatrix}$$

Wine Quality [%]

$$\mathcal{J}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - f(\mathbf{x}^{(i)}, \mathbf{W})\right)^2$$

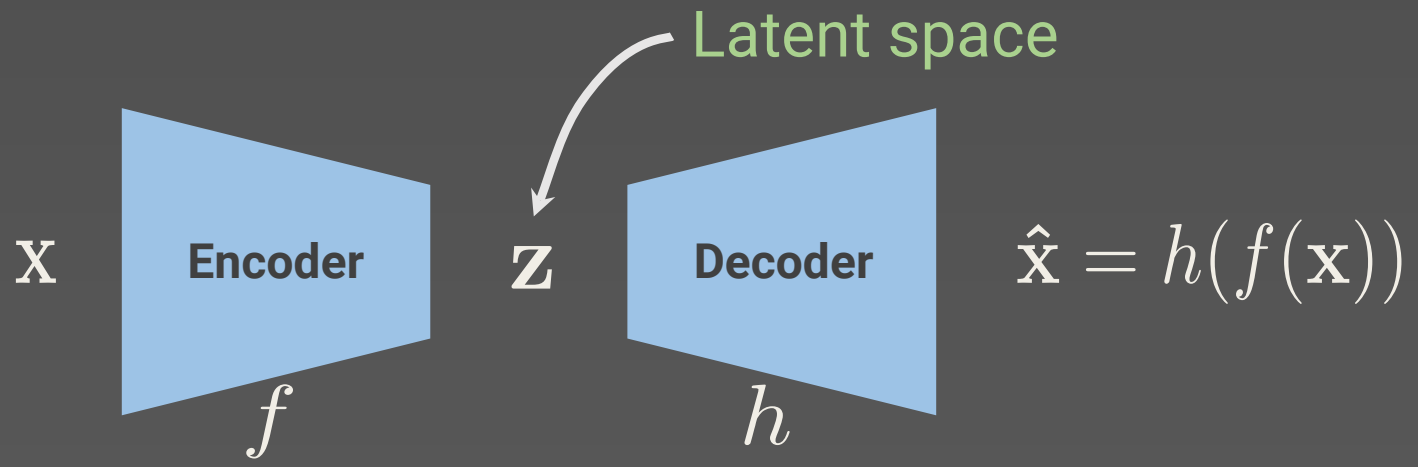
`tf.keras.losses.MSE()`

Unsupervised Deep Learning

Autoencoders

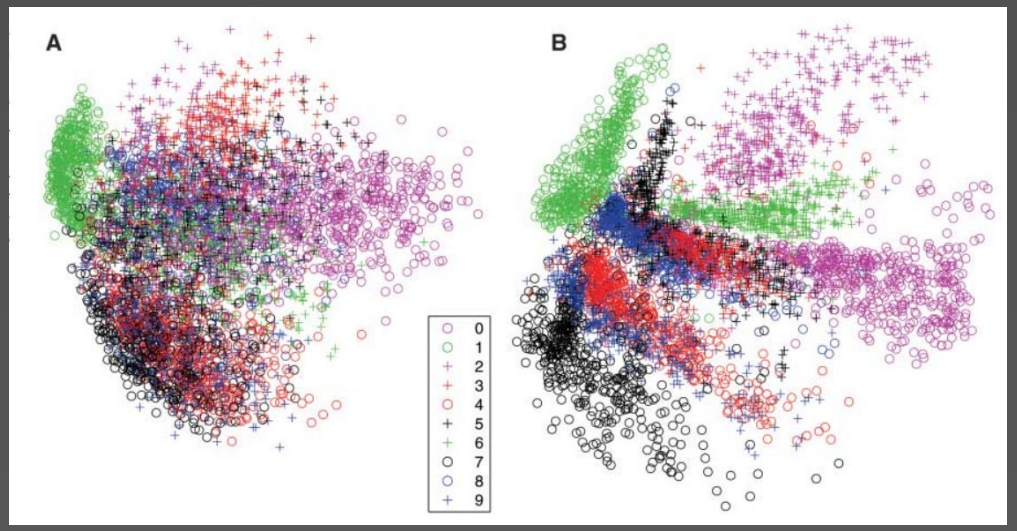
Unsupervised Learning

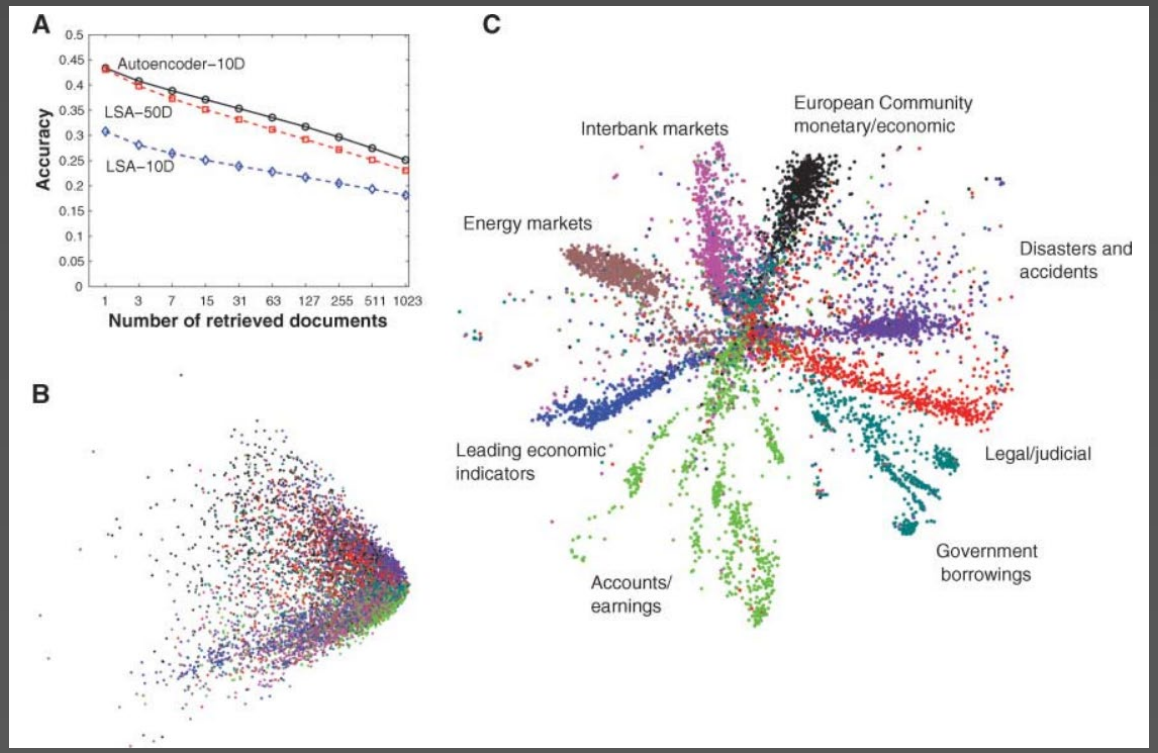
- Most used model for representation learning.



$$\mathcal{L}(\mathbf{x}; \mathbf{W}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$$

Autoencoders



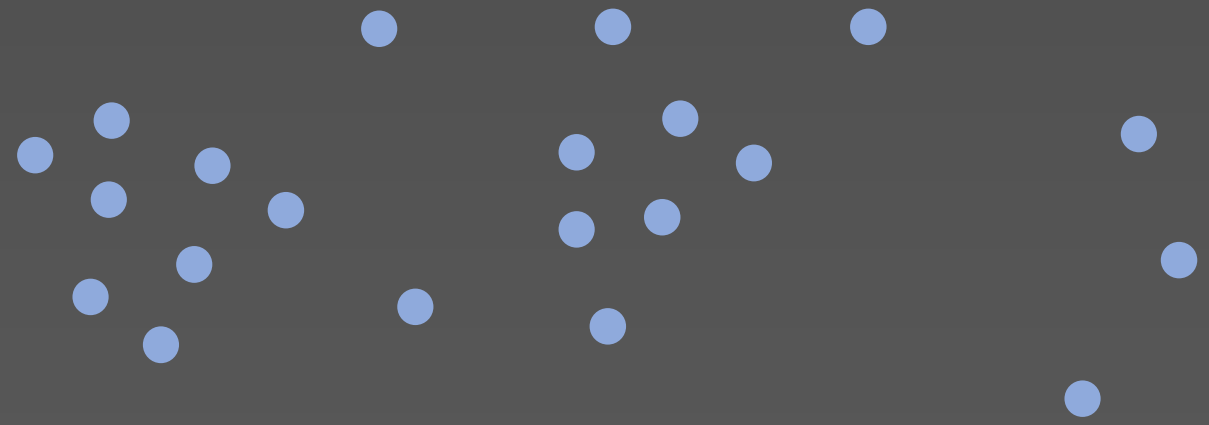


Reducing the Dimensionality of Data with Neural Networks, Hinton & Salakhutdinov, 2006

An autoencoder with linear activations does principal component analysis!

Unsupervised Representation Learning

Tip #1: Always “look” at your data before designing your model!

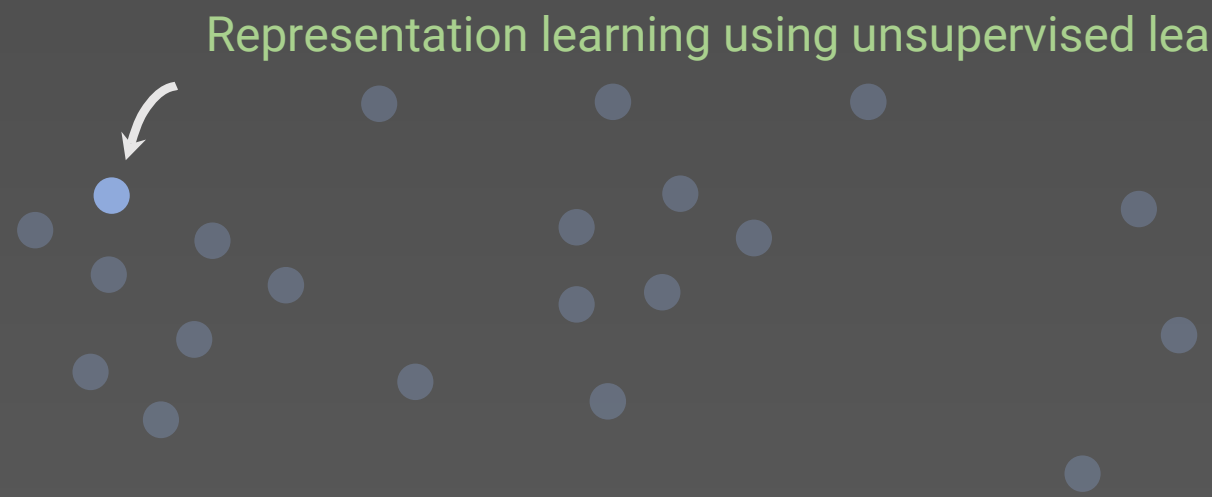


Mean & Covariance analysis
PCA (check eigenvalue decay)
t-sne visualization

Based on Marc’Aurelio Ranzato´s (Facebook AI Research) tutorial on Unsupervised Deep Learning, NeurIPS’18

Unsupervised Representation Learning

Tip #1: Always “look” at your data before designing your model!



Mean & Covariance analysis
PCA (check eigenvalue decay)
t-sne visualization

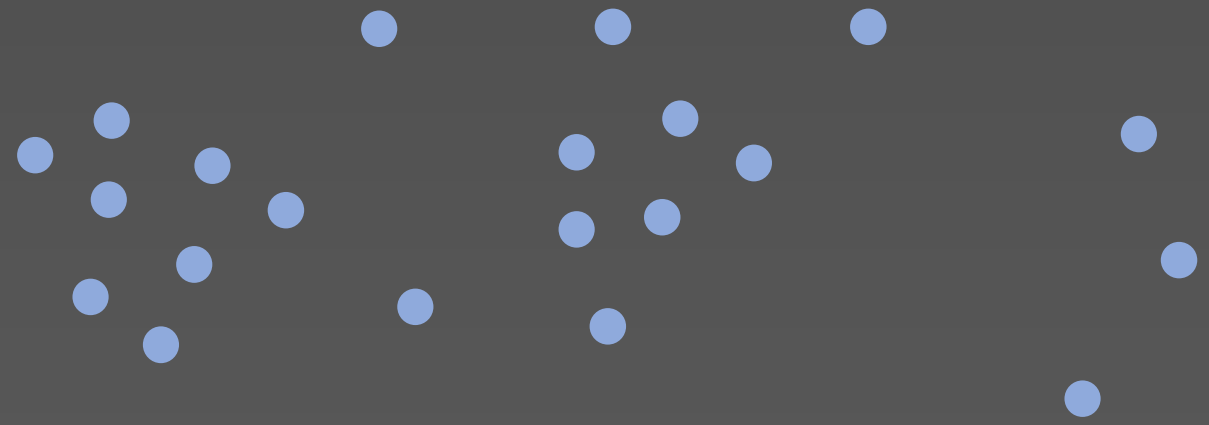
Learned features may be useful for down-stream tasks.

(*e.g.*, classification, regression, …)

Based on Marc’Aurelio Ranzato´s (Facebook AI Research) tutorial on Unsupervised Deep Learning, NeurIPS’18

Unsupervised Representation Learning

Tip #2: PCA and K-Means are very often a strong baseline.



Based on Marc'Aurelio Ranzato´s (Facebook AI Research) tutorial on Unsupervised Deep Learning, NeurIPS'18

Coffee Break

16:14 + **15min**

Optimizing the Loss

We want to find the neural network weights that achieve the lowest loss

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f(\mathbf{x}^{(i)}; \mathbf{W}), \mathbf{y}\right)$$

Weights

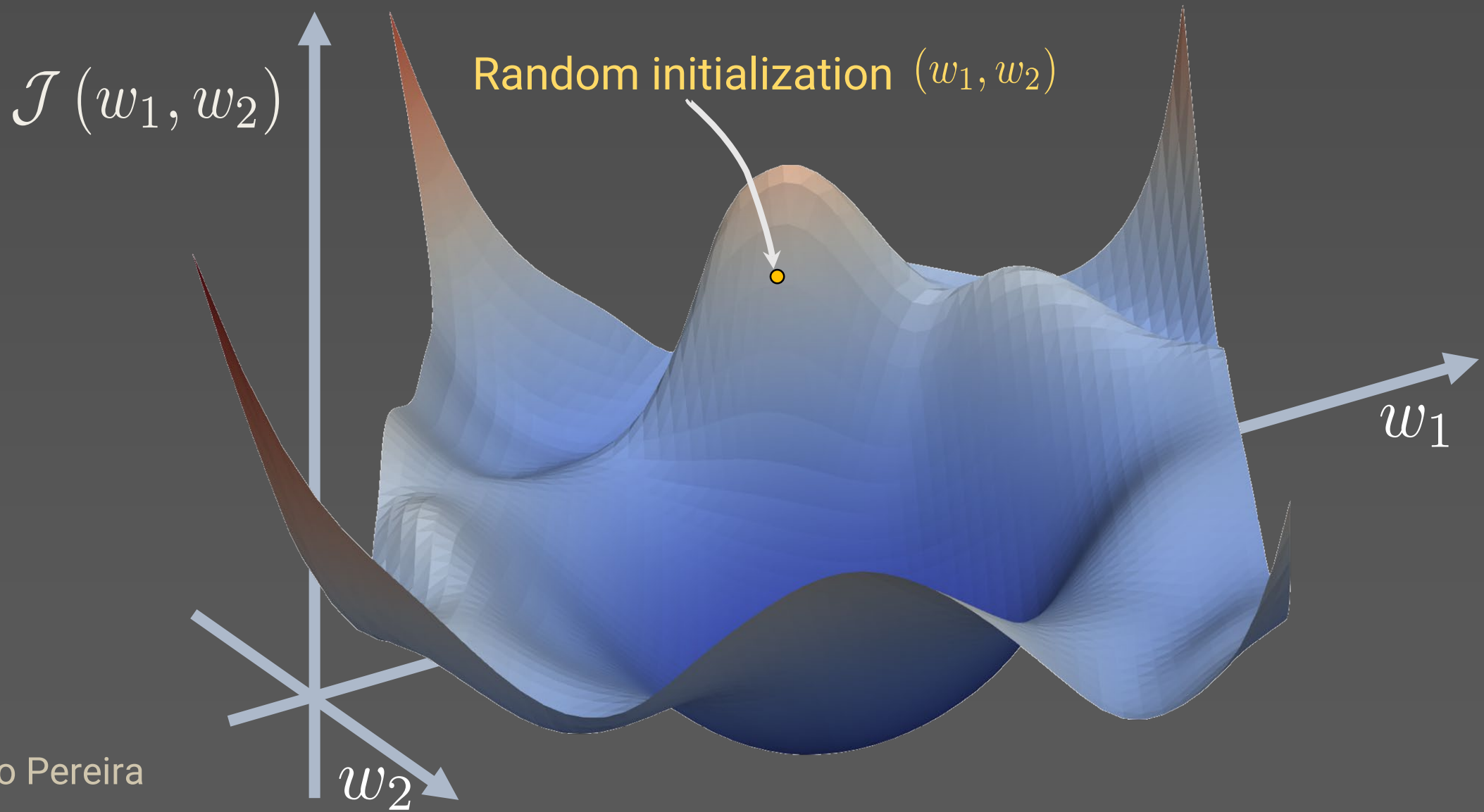
Training Examples

Or simply

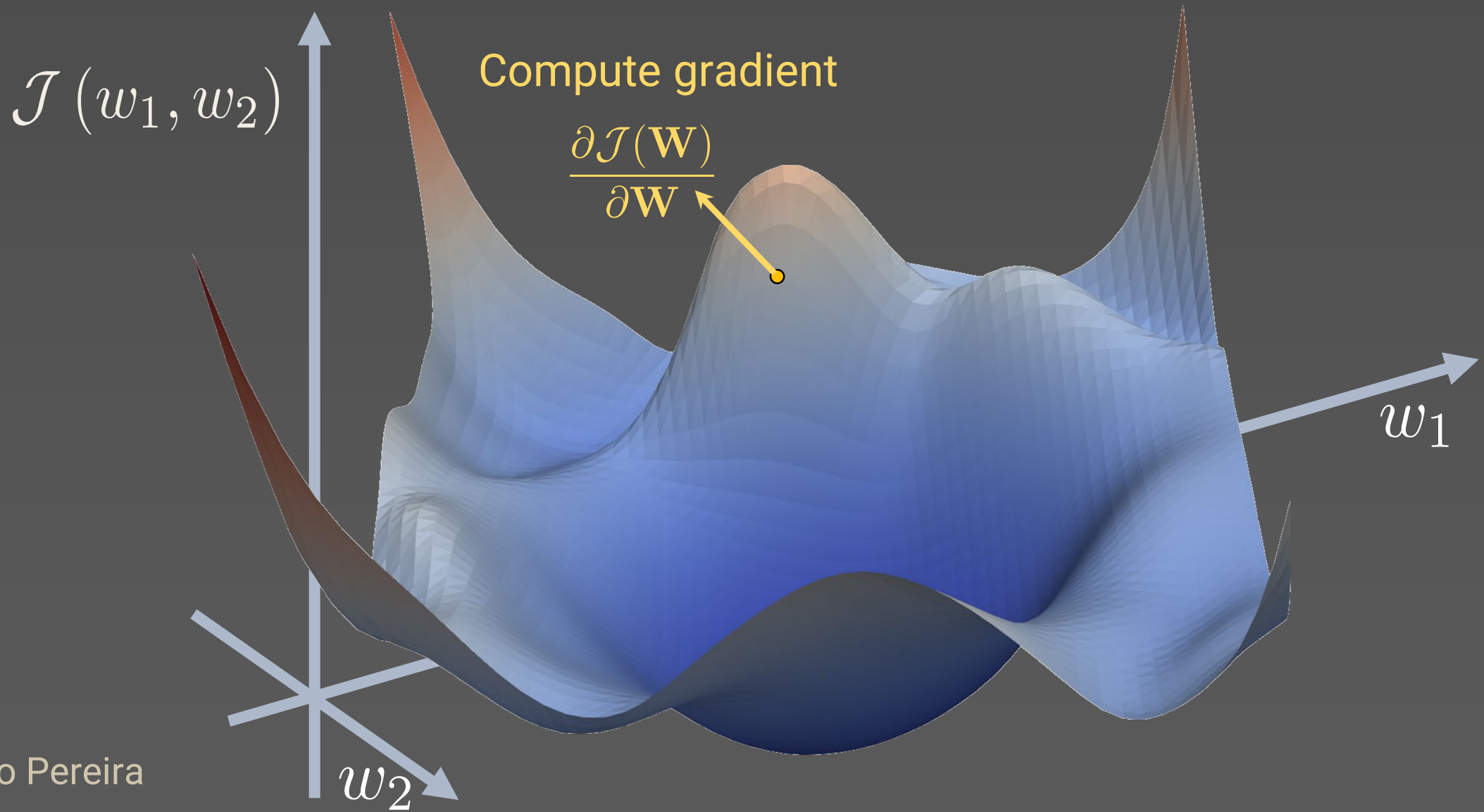
$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \mathcal{J}\left(\mathbf{W}\right)$$

Based on first class of MIT 6.S191 by Alexander Amini

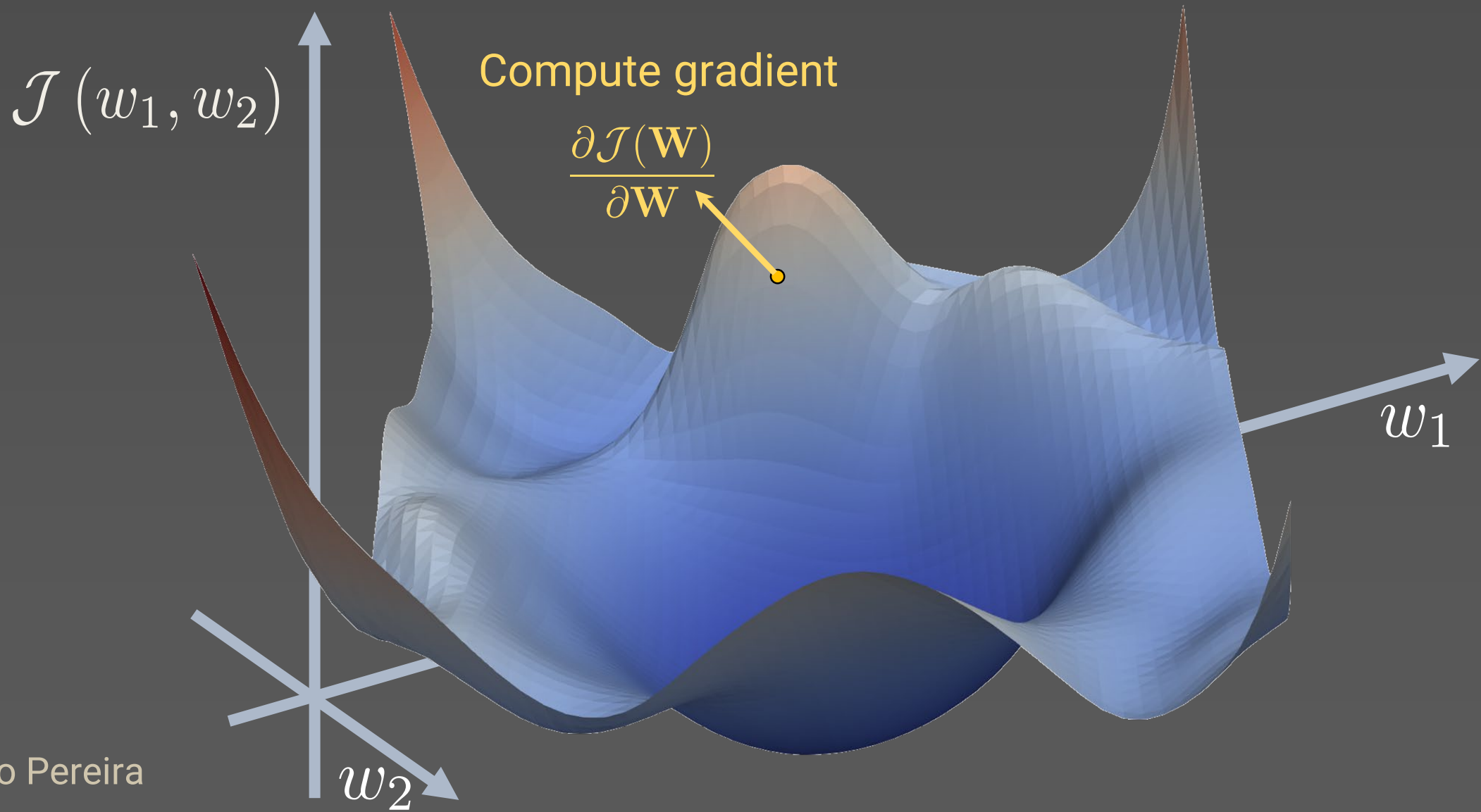
Optimizing the Loss



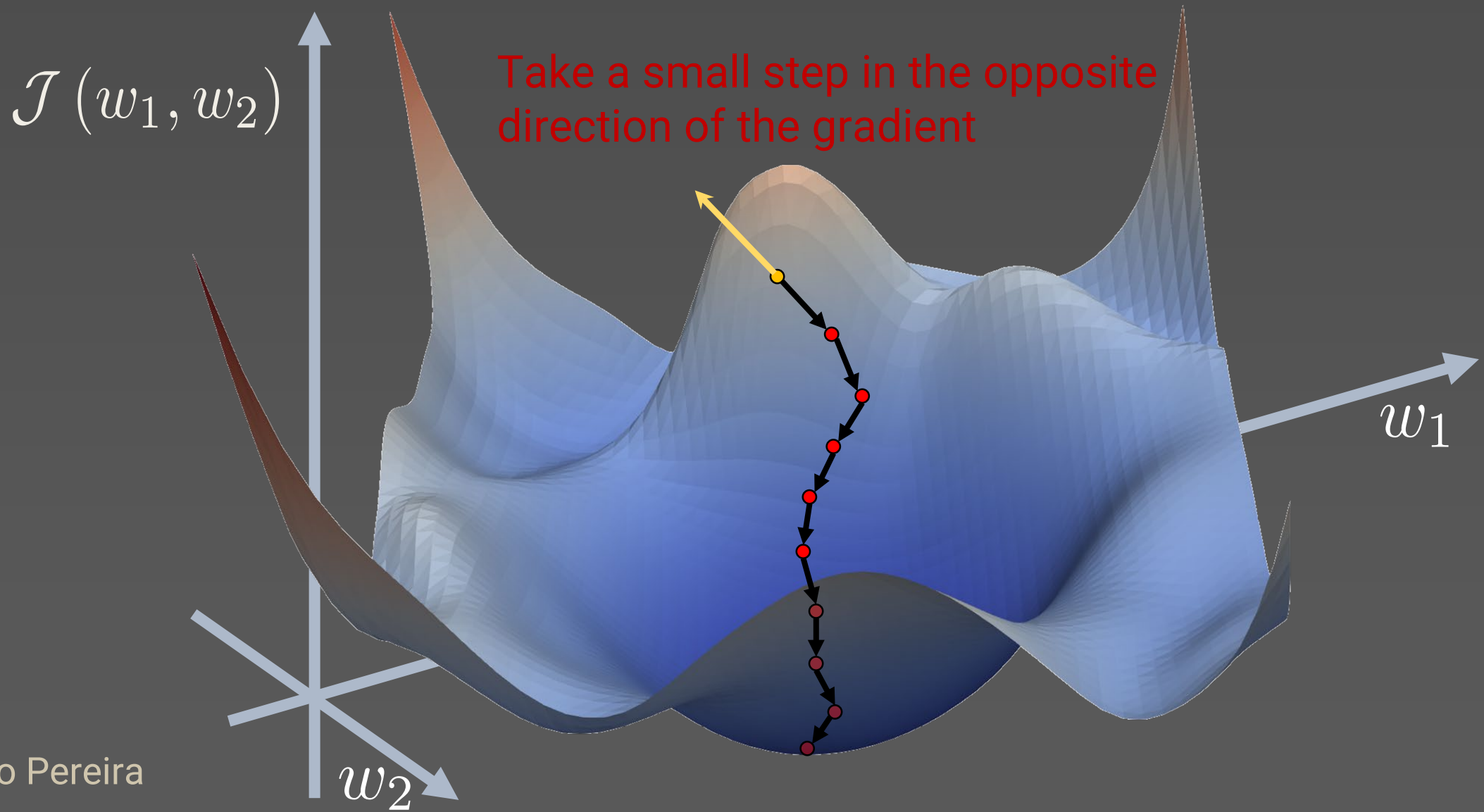
Optimizing the Loss



Optimizing the Loss



Optimizing the Loss



“It's like walking in the mountains in a fog and following the direction of steepest descent to reach the village in the valley”

Yann LeCun

Gradient Descent

This procedure is called **gradient descent**.

1. Initialize weights randomly.
2. Repeat until convergence.
    2.1. Computer gradient $\frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$
    2.2. Update weights $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$
3. Return weights.

Stochastic Gradient Descent

1. Initialize weights randomly.
2. Repeat until convergence.
    - **2.1. Choose 1 datapoint randomly**
    - 2.2. Computer gradient $\frac{\partial \mathcal{J}_m(\mathbf{W})}{\partial \mathbf{W}}$
    - 2.3. Update weights $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$
3. Return weights.

```
tf.keras.optimizer.SGD(x)
```

Stochastic Gradient Descent

1. Initialize weights randomly.
2. Repeat until convergence.
    2.1. Choose **mini-batch of M samples** randomly
    2.2. Computer gradient $\frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial \mathcal{J}_m(\mathbf{W})}{\partial \mathbf{W}}$
    2.3. Update weights $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$
3. Return weights.

```
tf.keras.optimizer.SGD(x)
```

Stochastic Gradient Descent

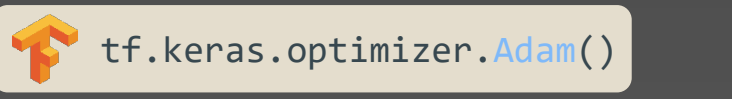
Many variants of SGD

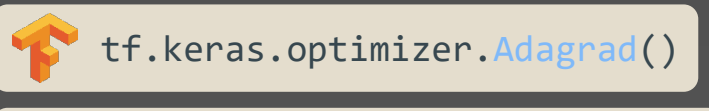
Adam

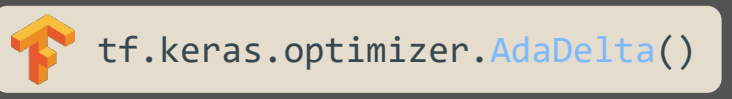
AdaGrad

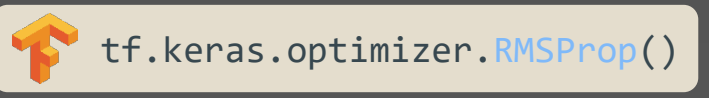
AdaDelta

RMSProp









Mini-batch training is faster (parallel computation on GPUs).

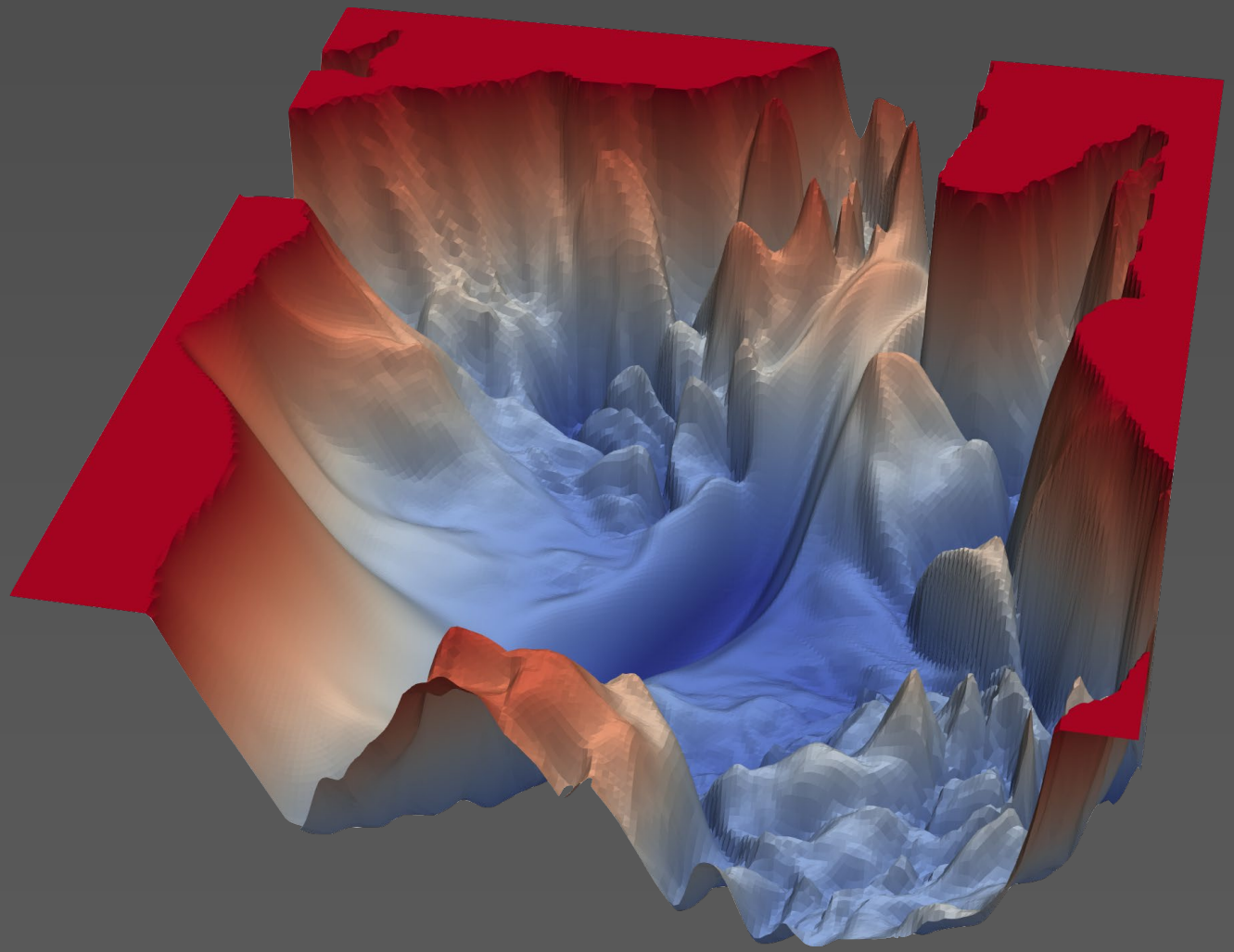
Backpropagation

How to compute $\frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$ **?**

- Employs the **chain rule** for deriving the gradient.
- There are tools that compute these gradients automatically.
- Check Hinton's course for details.
- Very good **interactive explanation of backpropagation.**

https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

In real-life deep learning...

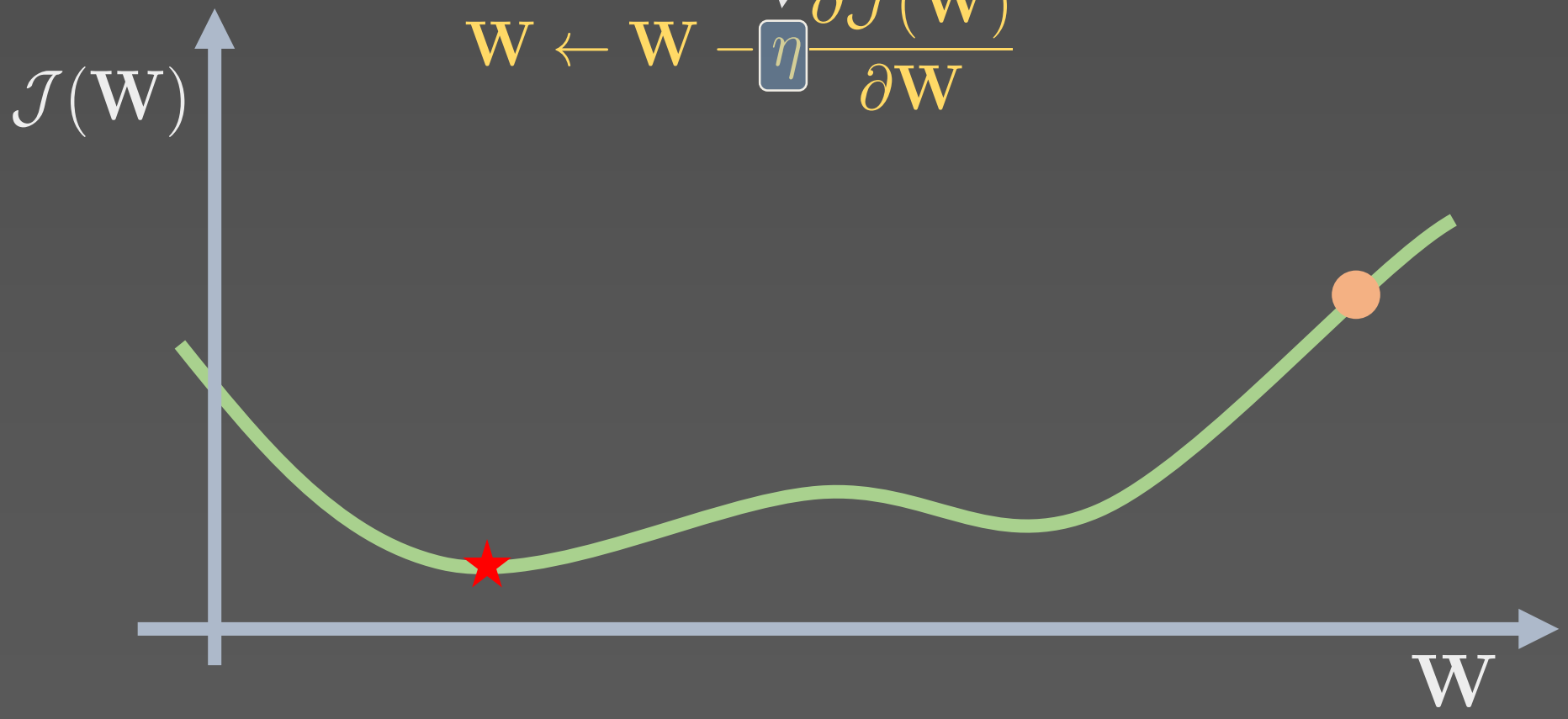


Training neural networks is hard!

Learning Rate

How to choose η ?

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{J}(\mathbf{W})}{\partial \mathbf{W}}$$



Learning Rate

- Small learning makes learning slow and gets stuck in **local minima**.
- Large learning rates overshoot, are unstable and diverge.

Tips:

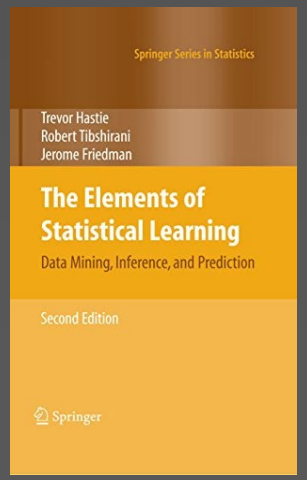
- Try many learning rates and choose the best
- Adaptive learning rate
  - How large gradient is…
  - How fast learning occurs…

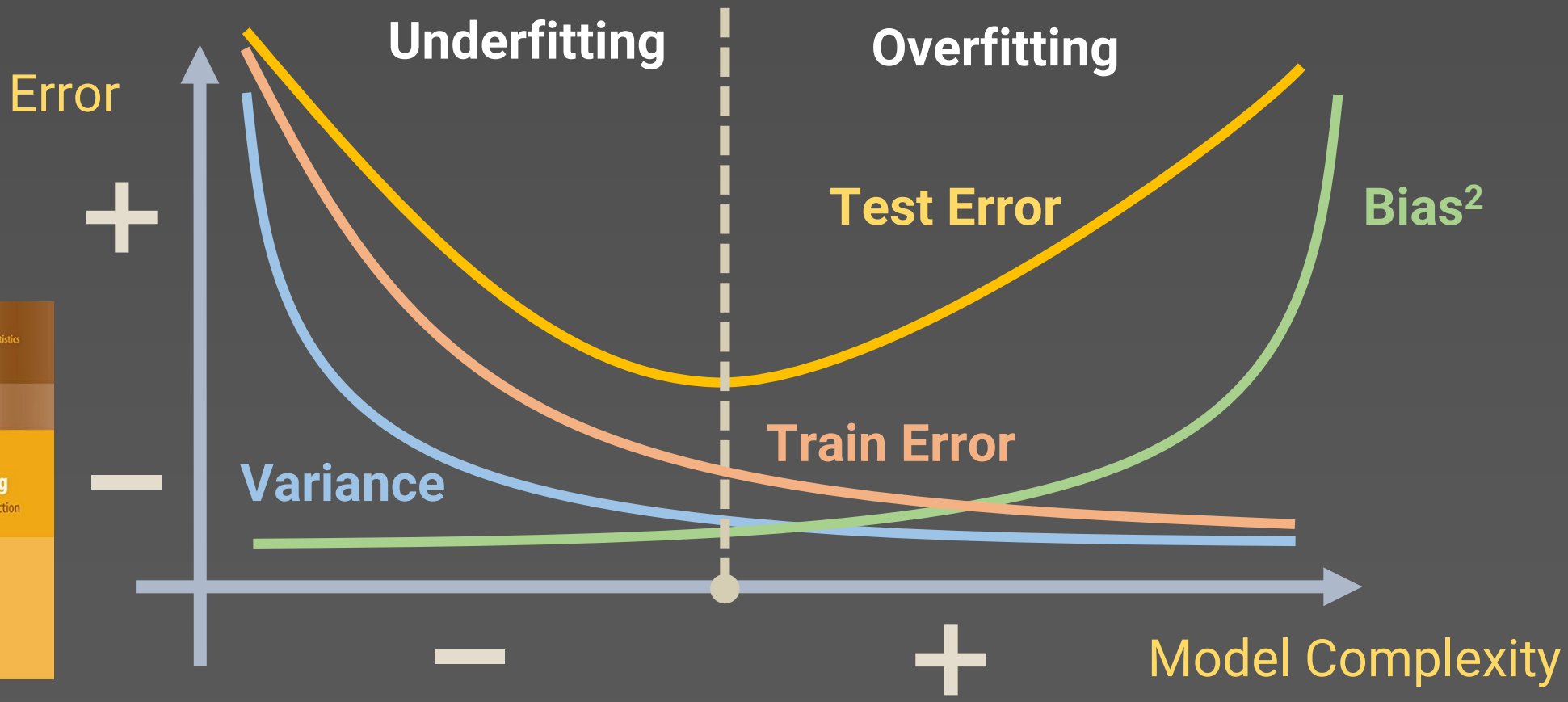
Optimization Challenges

Overfitting and Underfitting

Bias-Variance Trade-Off

Bias-Variance Trade-off





Regularization

Dropout

Early Stopping

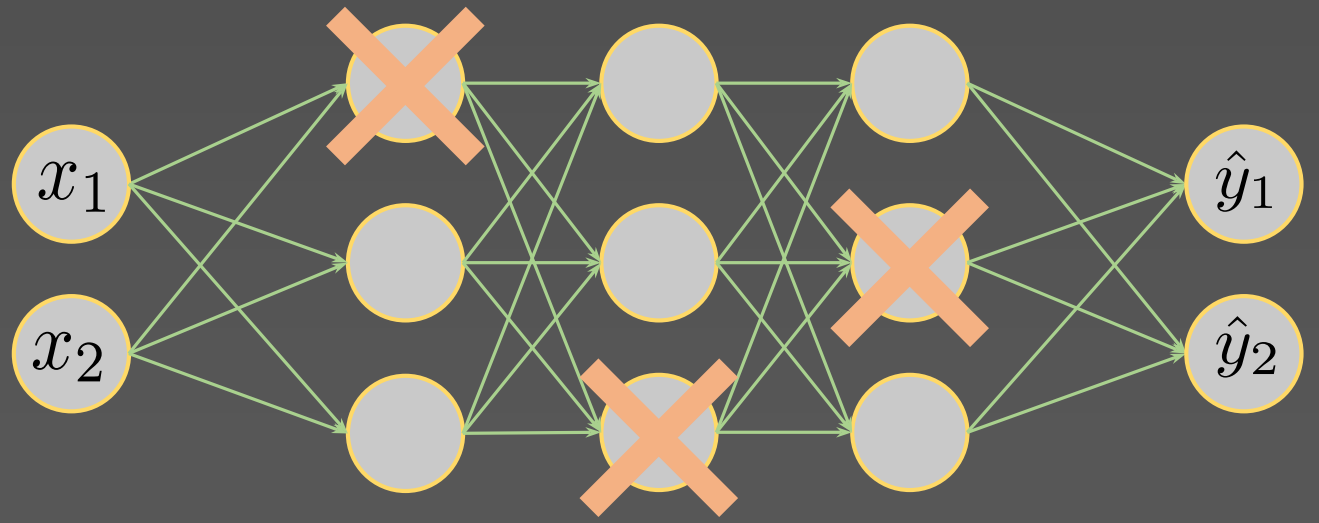
Why do we need regularization?

- To make our models **generalize better to unseen data**.
- We are going to look at two strategies:
  - Dropout
  - Early stopping

Dropout

Randomly set activations to zero.

Typically around 50%.

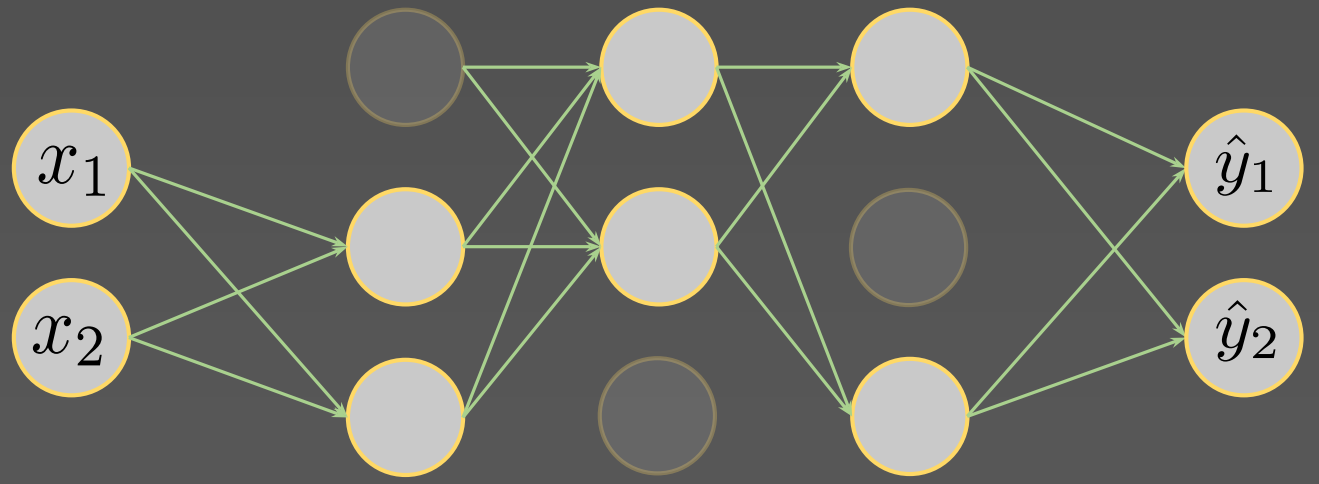


Dropout

Randomly set activations to zero.

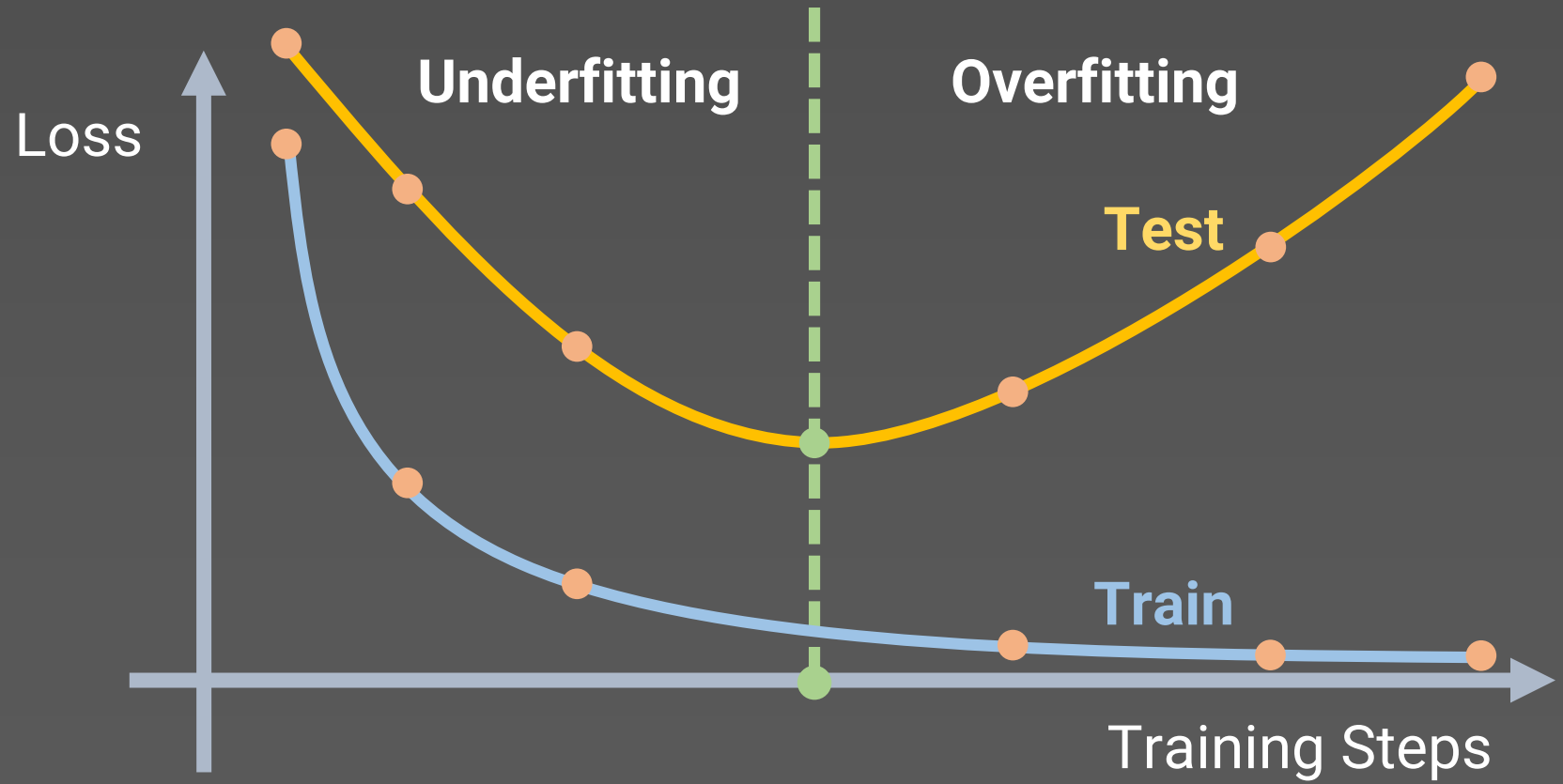
Typically around 50%.

```
tf.keras.layers.Dropout(p)
```



Early Stopping

- Early stop training before overfitting.



Generalization in Deep Learning

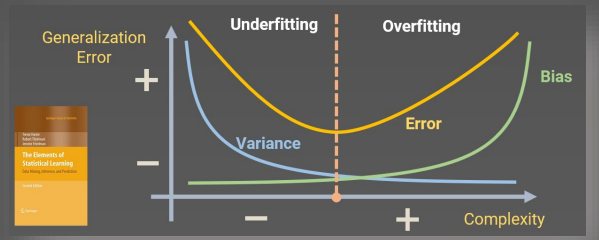
Generalization in Deep Learning

model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75
(fitting random labels)		no	no	100.0	9.78

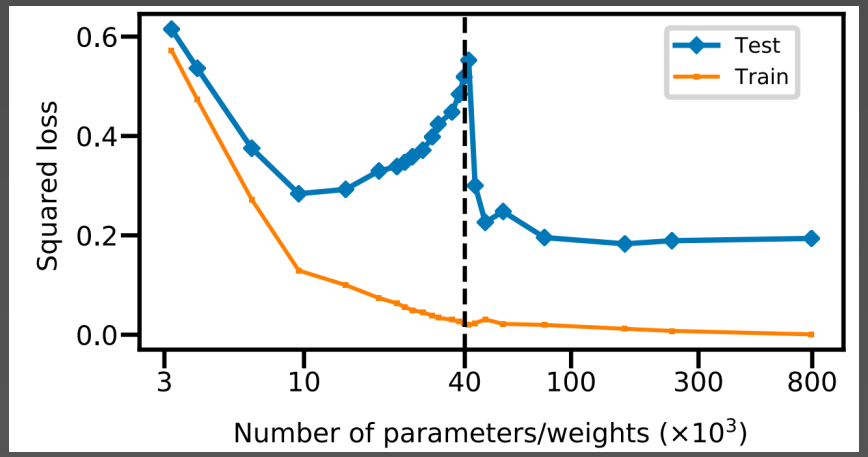
Understanding Deep Learning Requires Rethinking Generalization, Zhang *et al.*, 2017

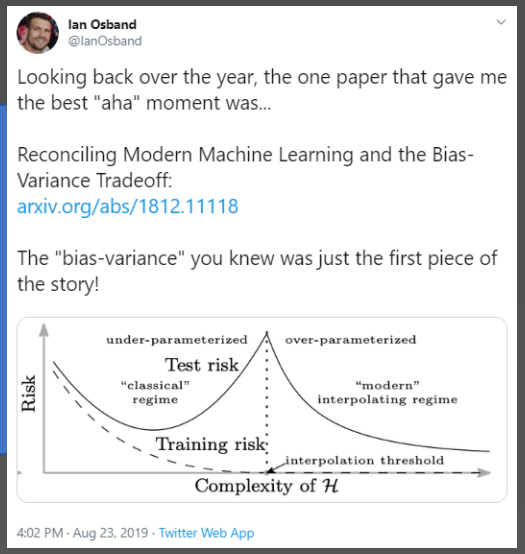
Large models overfit very little

???



Generalization in Deep Learning

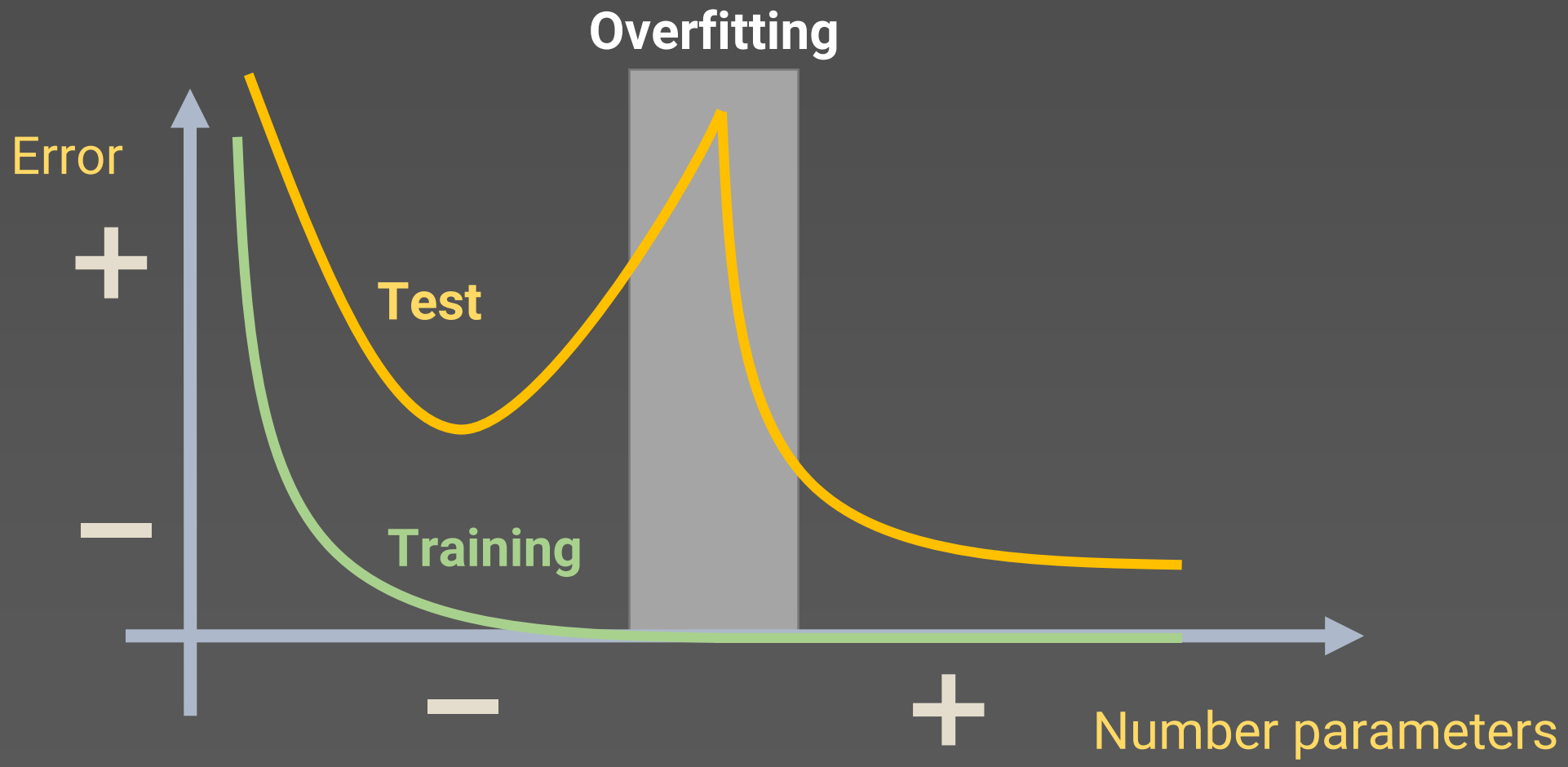




Reconciling modern machine-learning practice and the classical bias–variance trade-off, Belkin *et al.*, 2019

The bias-variance trade-off was just the first episode of the series!

Bias-Variance Trade-off



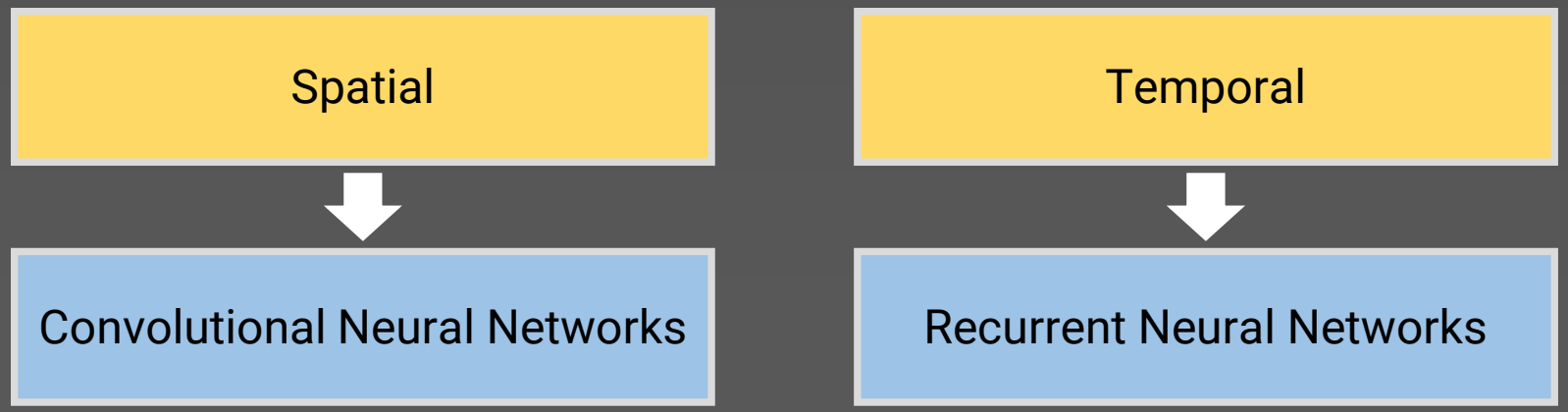
Part 2

Convolutional Neural Networks

Recurrent Neural Networks

Motivation

- In feed-forward neural networks we assume data is independent and identically distributed in space or in time.
- In many applications, data has **spatial** or **temporal** dependencies.



Convolutional Neural Networks

Images

- **An image is a matrix of numbers.**

200x200=40k pixels

200

2	4	3	2	4	3	...	3
3	8	5	3	8	5		7
1	2	1	1	2	1		5
2	4	3	2	4	3		2
3	8	5	3	8	5		1
1	2	1	1	2	1		3
...						...	4
5	6	8	2	4	6	8	9

200

Values between 0 and 255

Images

200 x 200 **x 3** = 120k

- **An image is a matrix of numbers.**

R	B	G							
	5	2	4	3	2	4	3	...	3
		3	8	5	3	8	5		7
2	6	1	2	1	1	2	1		5
3	9	2	4	3	2	4	3		2
1	2	3	8	5	3	8	5		1
2	4	1	2	1	1	2	1		3
3	7	...						...	4
1	4	5	6	8	2	4	6	8	9
...	3	8	5	3	8	5		1	
8	4	2	1	7	9	9	7		

Values between 0 and 255

Tasks in Computer Vision

Regression

Output is a continuous value.

Classification

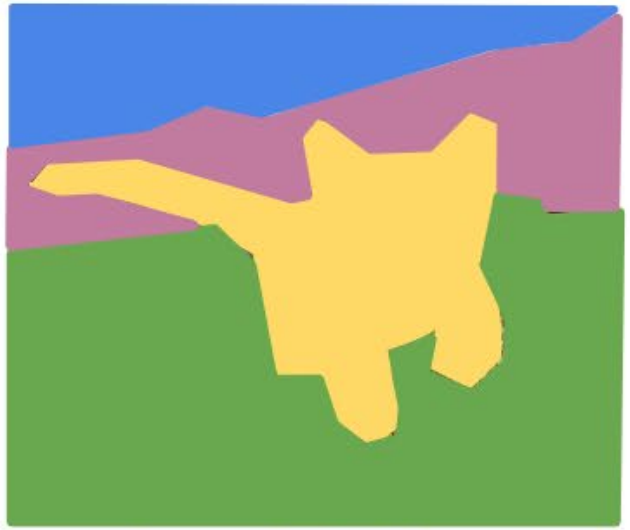
Output is a class.

Applications:

Classification, segmentation, localization, detection.

Examples

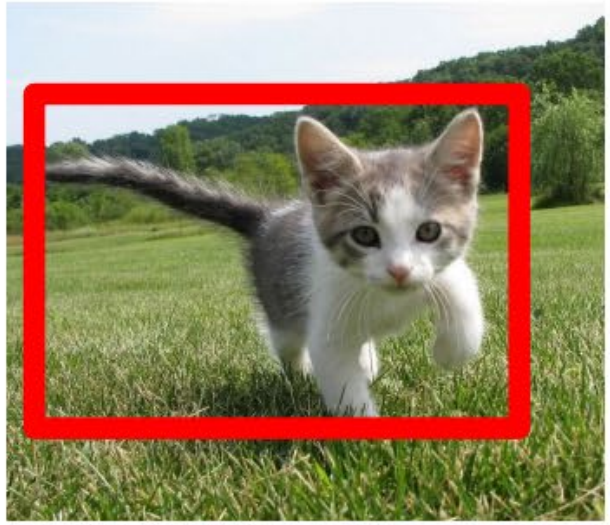
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

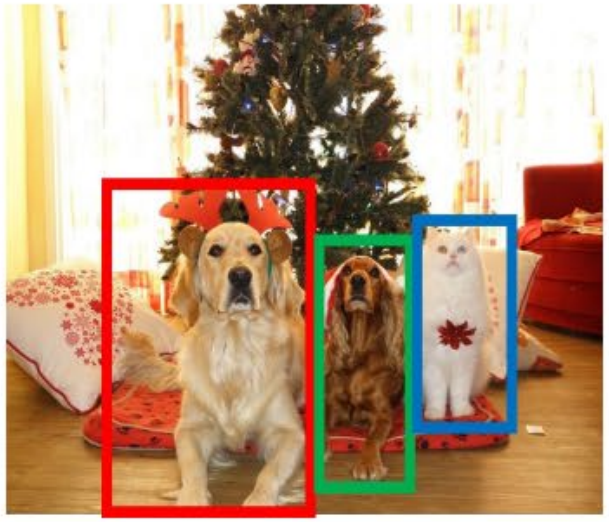
Classification + Localization



CAT

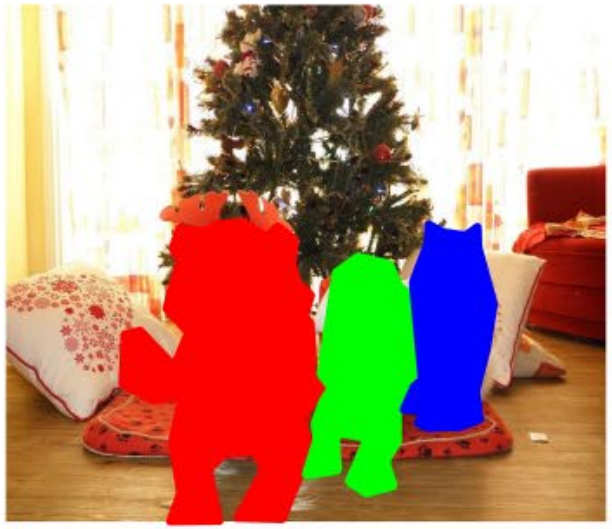
Single Object

Object Detection



DOG, DOG, CAT

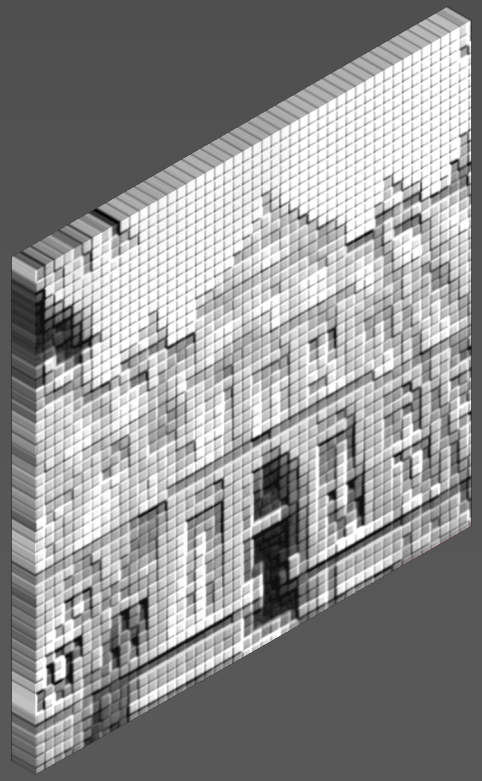
Instance Segmentation

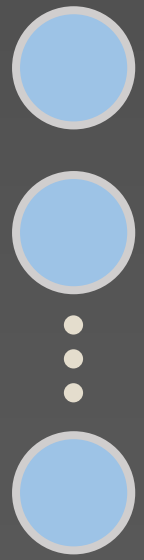


DOG, DOG, CAT

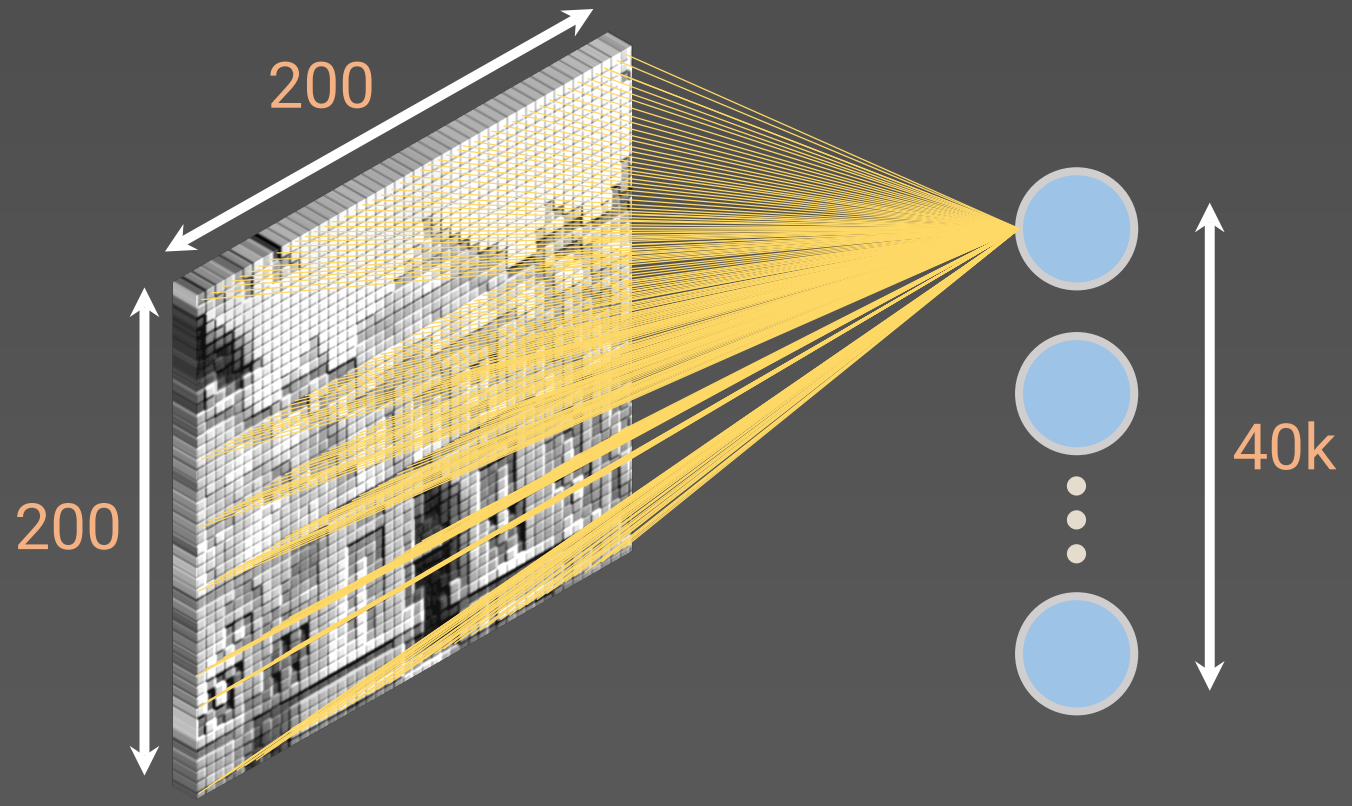
Multiple Object

Why do we need convolutional neural networks?





Why do we need convolutional neural networks?

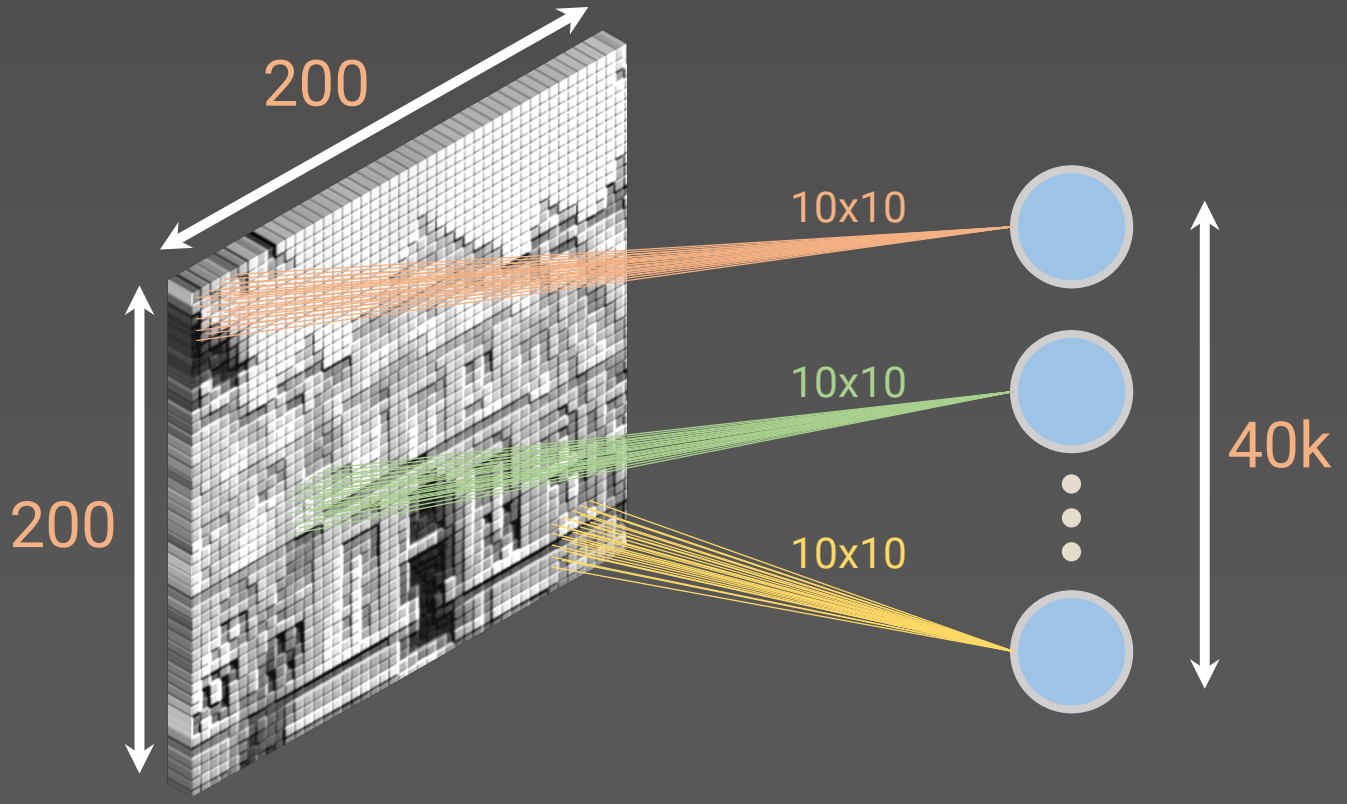


Fully connected layer

Requires
1.6 billion parameters

☹

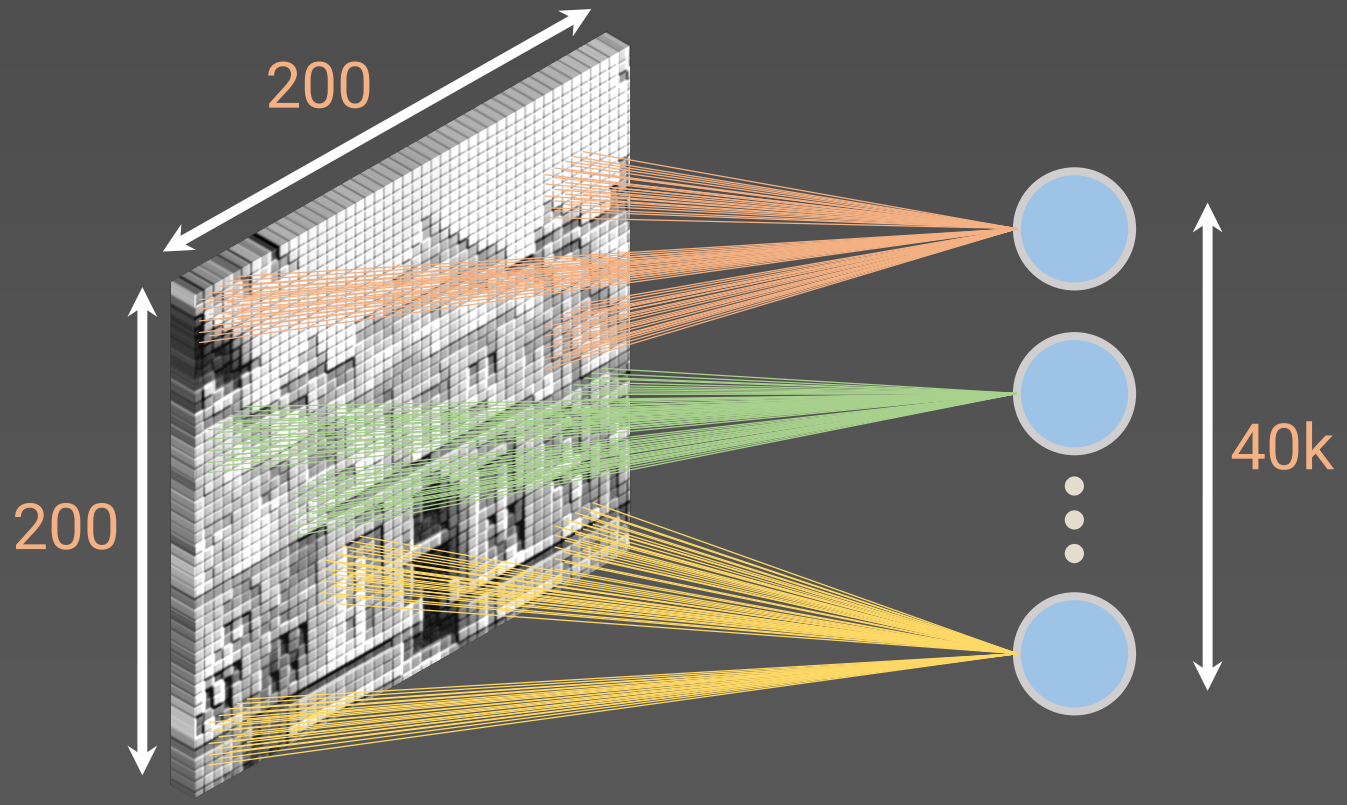
Why do we need convolutional neural networks?



Locally connected layer

Requires
4 million parameters

Convolutional Layer Idea

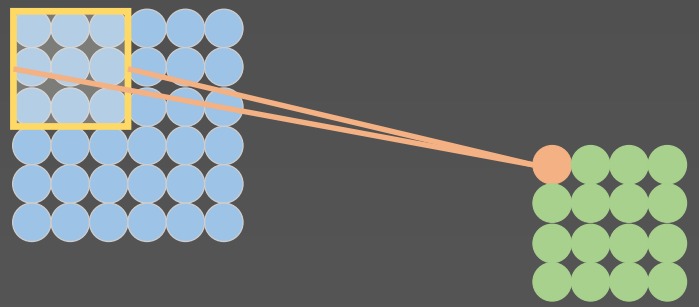


Use multiple **filters** to extract different kinds of features.

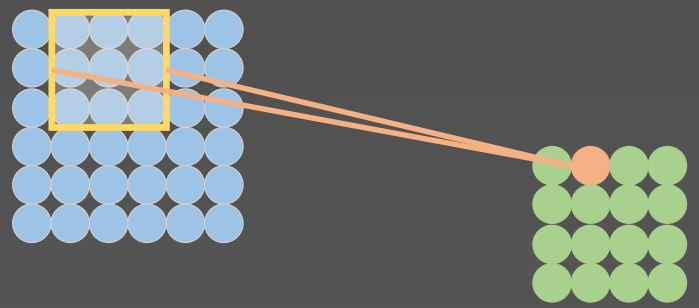
Spatially share the **same parameters** across different locations.

Intuition: important features can happen anywhere in the image.

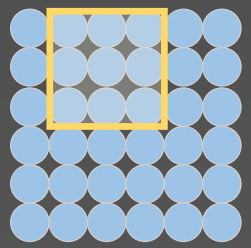
Idea



Idea



Idea



What is a convolution?

Input Image

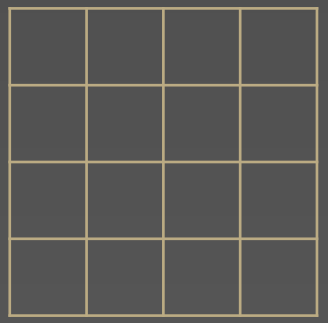
2	4	3	5	6	8
3	8	5	7	5	1
1	2	1	5	6	7
2	4	6	2	3	4
3	5	7	2	5	6
6	8	9	7	9	8

⊙

1	0	1
0	0	1
1	0	1

Filter
Kernel
Feature Detector

=



Feature Map
Activation Map
Convolved Feature

What is a convolution?

Input Image

2	4	3	5	6	8
3	8	5	7	5	1
1	2	1	5	6	7
2	4	6	2	3	4
3	5	7	2	5	6
6	8	9	7	9	8

$\odot$

1	0	1
0	0	1
1	0	1

$=$

12	0	1	0
0	0	0	0
1	1	1	0
1	0	1	1

$2x1 + 4x0 + 3x1 + 5x1 + 8x0 + 3x0 + 1x1 + 2x0 + 1x1 = \mathbf{12}$

Filter
Kernel
Feature Detector

Feature Map
Activation Map
Convolved Feature

Convolutional Layer

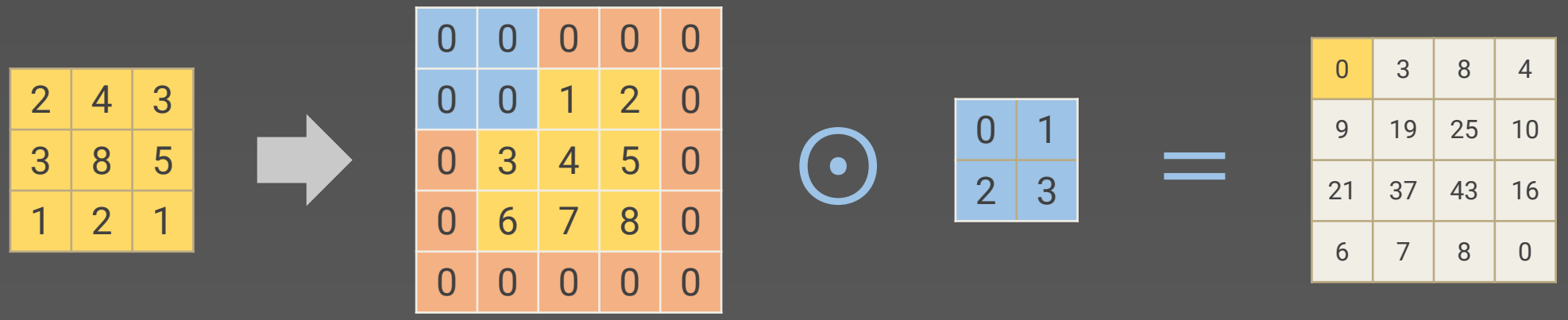
- **Weights of the filters are learned** during training
- We need to define:
  - Number of filters
  - Filter dimensions
  - Padding
  - Stride

1	0	1
0	0	1
1	0	1

Padding

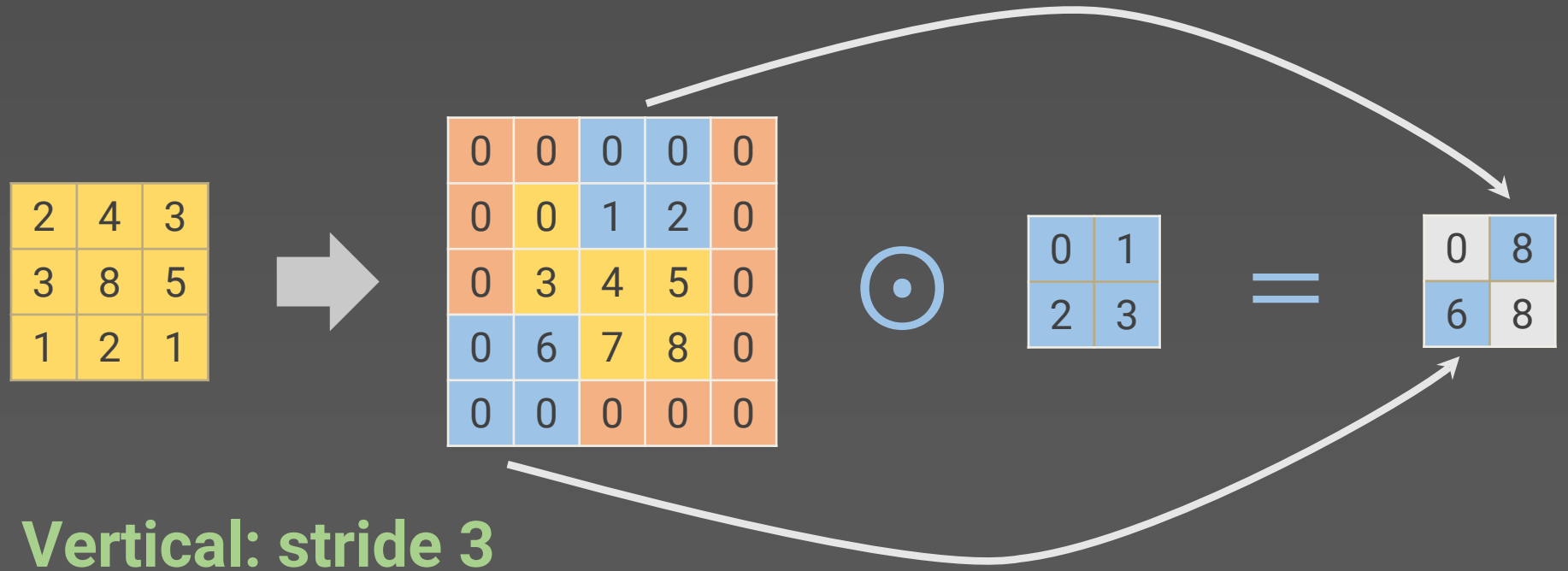
- Used to avoid loosing pixels around the image.

Padding with zeros



Stride

- Is the number of rows and columns traversed per slide.



Vertical: stride 3
Horizontal: stride 2

In a nutshell...

Feed-forward neural networks applied to images:

- Scale quadratically with the input size

Solution: Use a **convolutional layer**

- Connect each hidden unit to a small patch of the input.
- **Share weights** across space.

A network built with these structure is called a

Convolutional Neural Network (CNN)

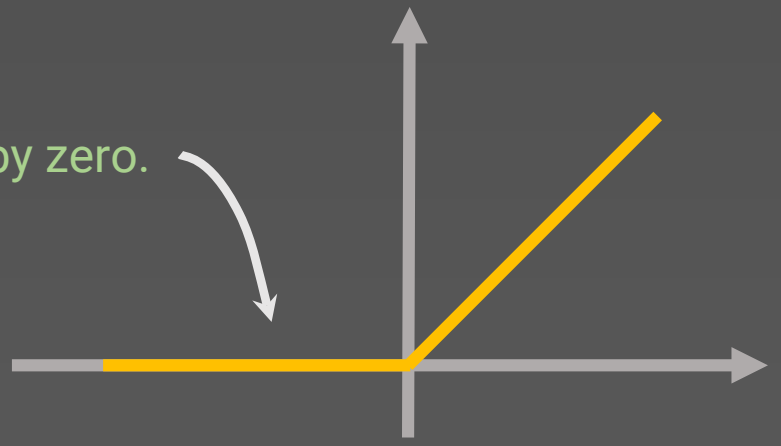
LeCun *et al.*, 1998, **Gradient-based learning applied to document recognition**

Based on Marc'Aurelio Ranzato's slides, Facebook AI Research, **Image Classification with Deep Learning**

Non-Linearity

- Introduce non-linearities after convolutional layers.
- The **rectified linear unit** (ReLU) works very well.

Replaces negative values by zero.



```
tf.keras.activations.relu(x)
```

Pooling Layer

- Pooling promotes **robustness** to the specific location of the features.

Example:

2	4	3	5	6	3
3	8	5	7	5	1
1	2	1	5	6	7
2	4	6	2	3	4
3	5	2	2	5	6
6	4	5	7	9	8

Dimension reduced

8	7
5	9

Max pooling
Filters 3x3
Stride 3

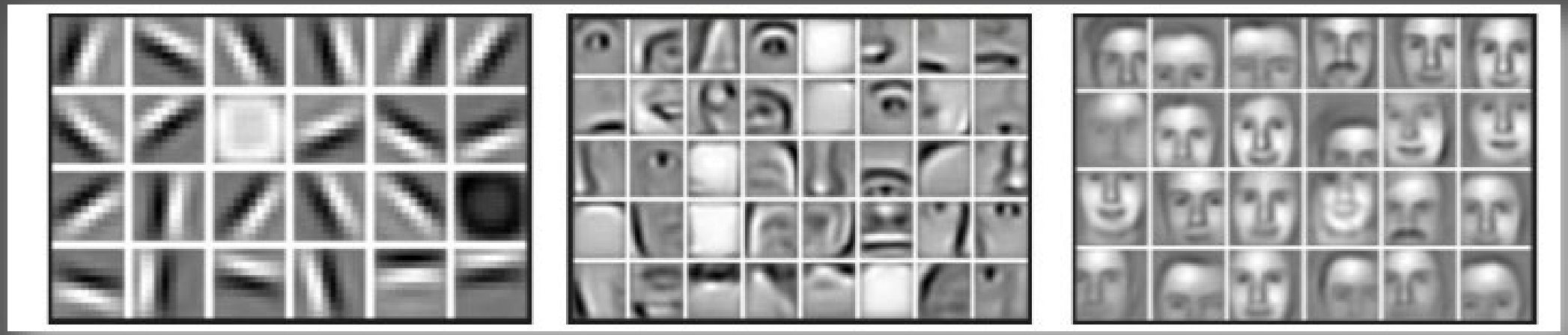
Max, Avg, L2-pooling, ...

Representation Learning with CNNs

Low Level Features

Medium Level Features

High Level Features



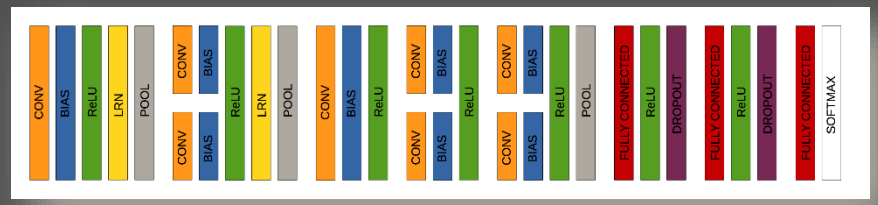
Conv. Layer 1

Conv. Layer 2

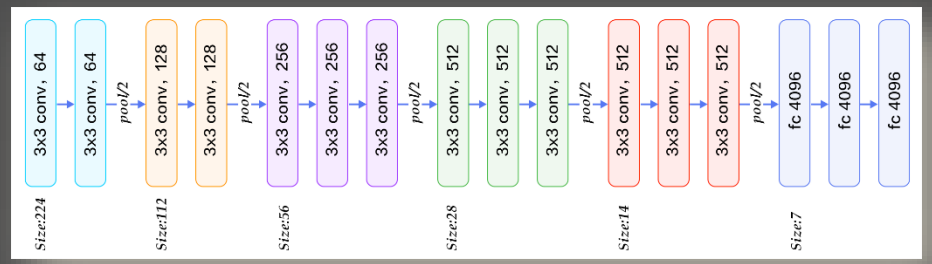
Conv. Layer 3

Progress in CNN Architectures

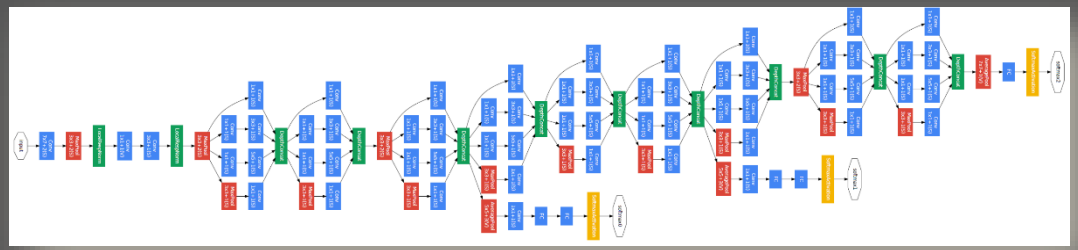
AlexNet, 2012
Krizhevsky, Sutskever, and Hinton



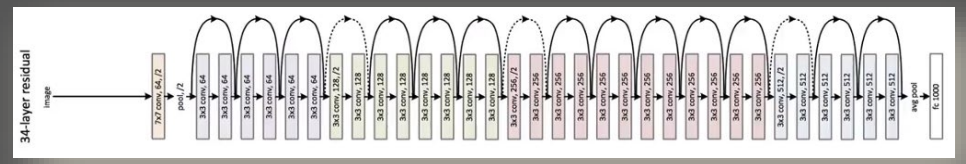
VGG, 2014
Zisserman, Simonyan



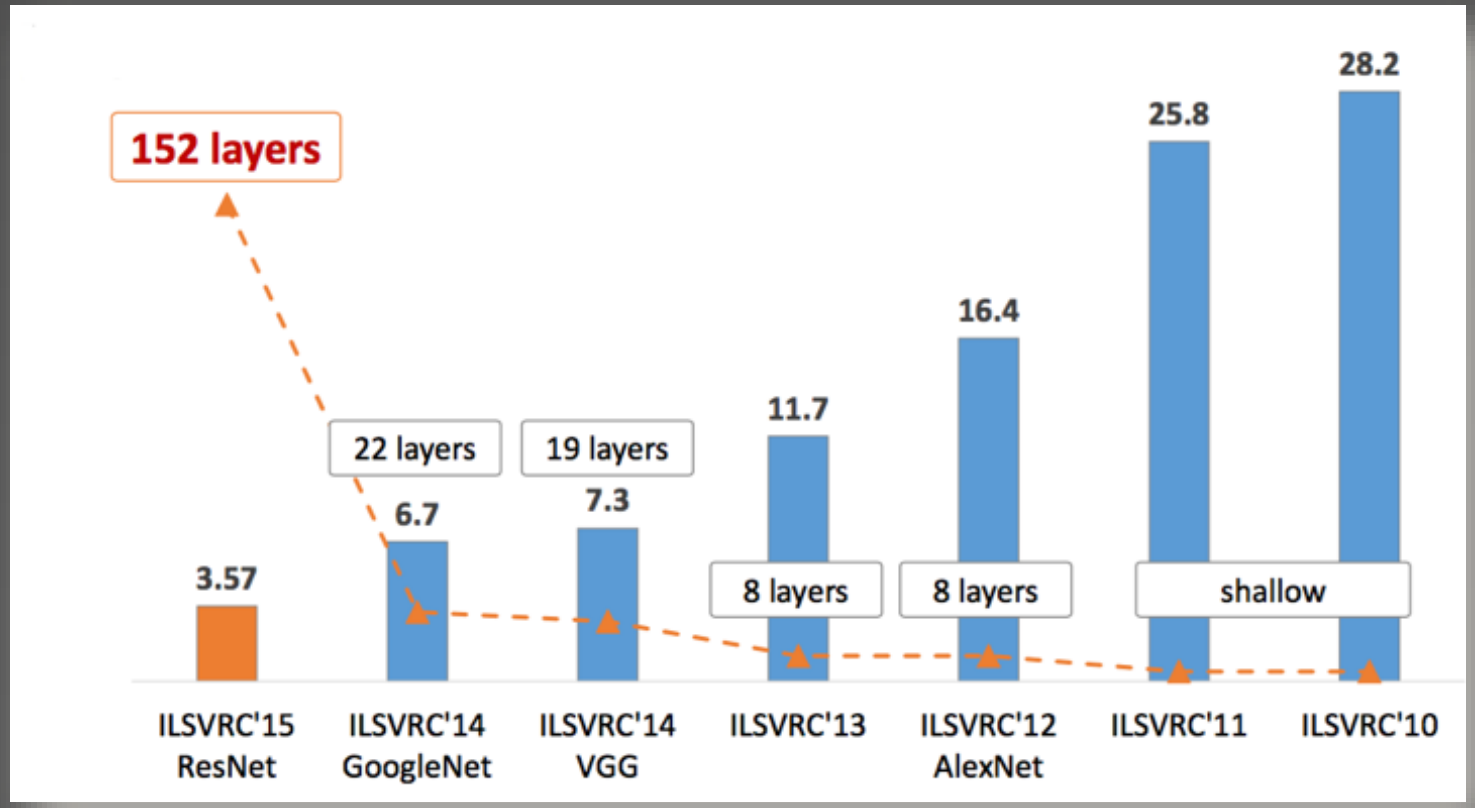
GoogLeNet, 2014
Szegedy *et al.*



ResNet, 2015
He



Importance of Depth



Link

Deep learning is in production

Coffee Break

16:16 + **15min**

Recurrent Neural Networks

Why do we need recurrent neural networks?

In many applications, data has a **sequential** nature:

- Sensor time series
- Natural speech
- Words in sentences
- DNA
- Videos

What if data is not i.i.d. in time?

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$$

Example:

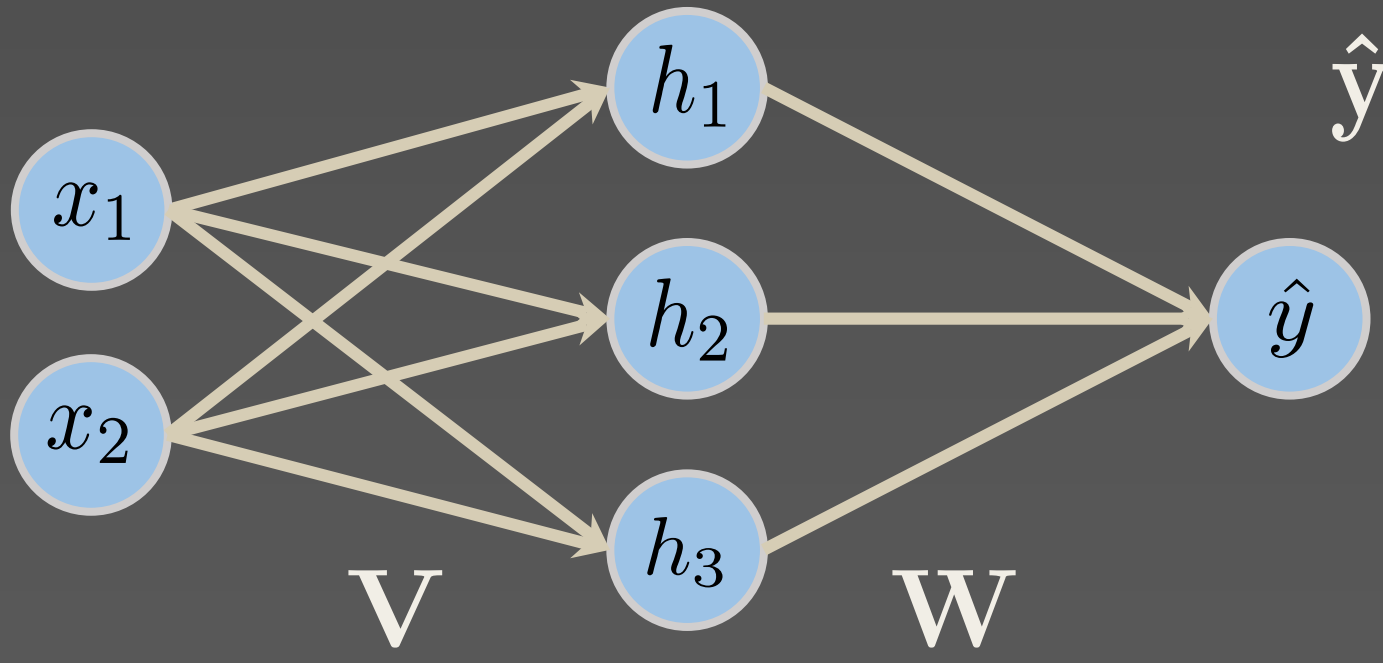
My name is João

$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$

Recap

- In feed-forward neural networks:

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{b})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{c}$$



Recurrent Neural Network

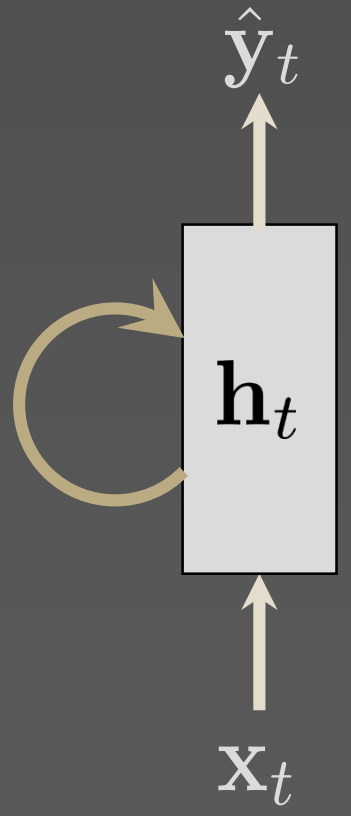
Feed-forward Neural Network

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{b})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{c}$$

Recurrent Neural Network

Bengio *et al.*, 1994



$$\mathbf{h}_t = g\left(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}\right) + \mathbf{b}$$

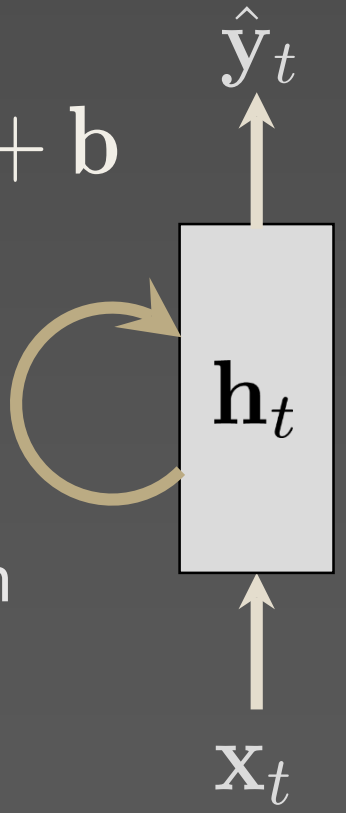
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{c}$$

Recurrent Neural Network

$$\mathbf{h}_t = g\left(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}\right) + \mathbf{b}$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{c}$$

g is typically a sigmoid or tanh



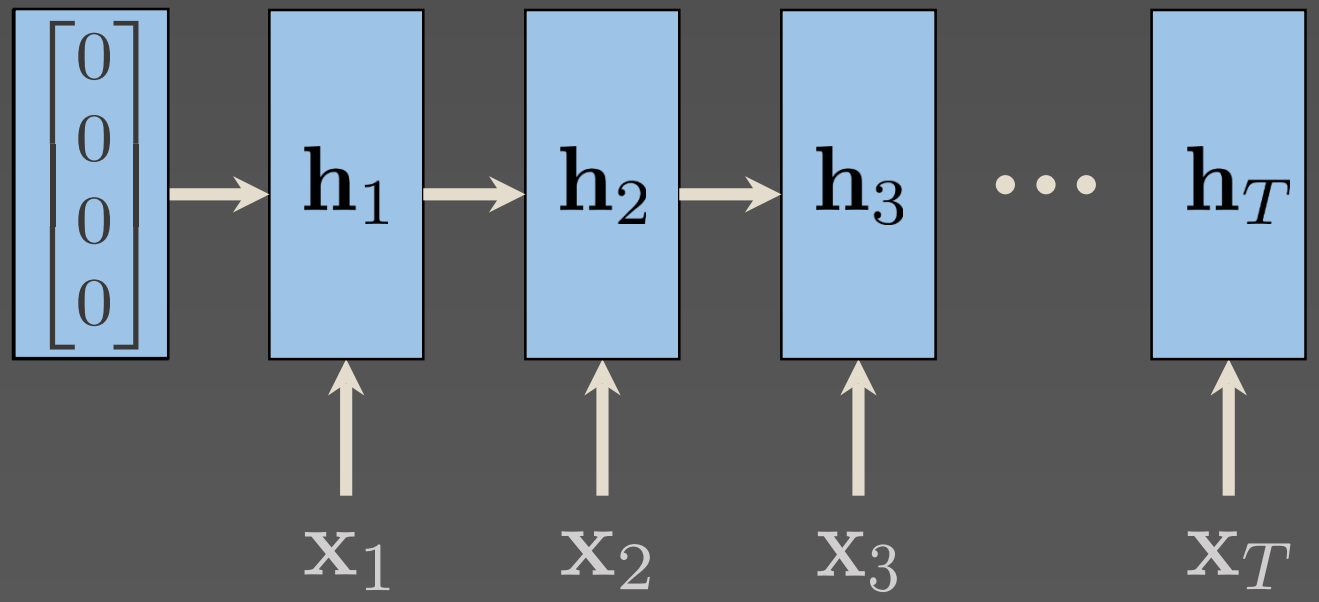
RNNs capture the temporal dependencies of the data.

- Real-valued hidden state
- Feedback connection
- Parameters **shared** across timesteps

RNN Unrolled

$$\mathbf{h}_1 = \mathbf{U}\mathbf{x}_1 + \mathbf{W}\mathbf{h}_0$$

Bengio *et al.*, 1994

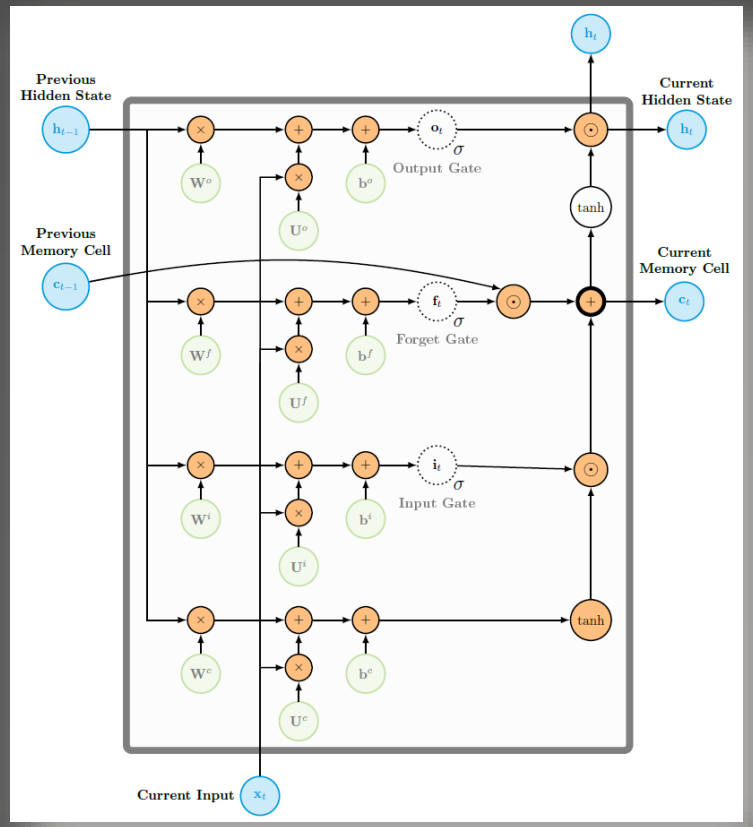


Notes

- $\mathbf{h}_T$ is a **vector representation** of the entire sequence $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$
- Summarizes sequences into fixed-length vectors
- Input sequences can have arbitrary length.
- Final hidden state can be used as a **feature vector** for classification or regression tasks.

Long-Short Term Memory Networks

Hochreiter & Schmidhuber, 1997



- Proposed to solve the **vanishing gradient problem**.
- New cell and three gates.

Sequence to Sequence Models (Seq2Seq)

Sequence to Sequence Models

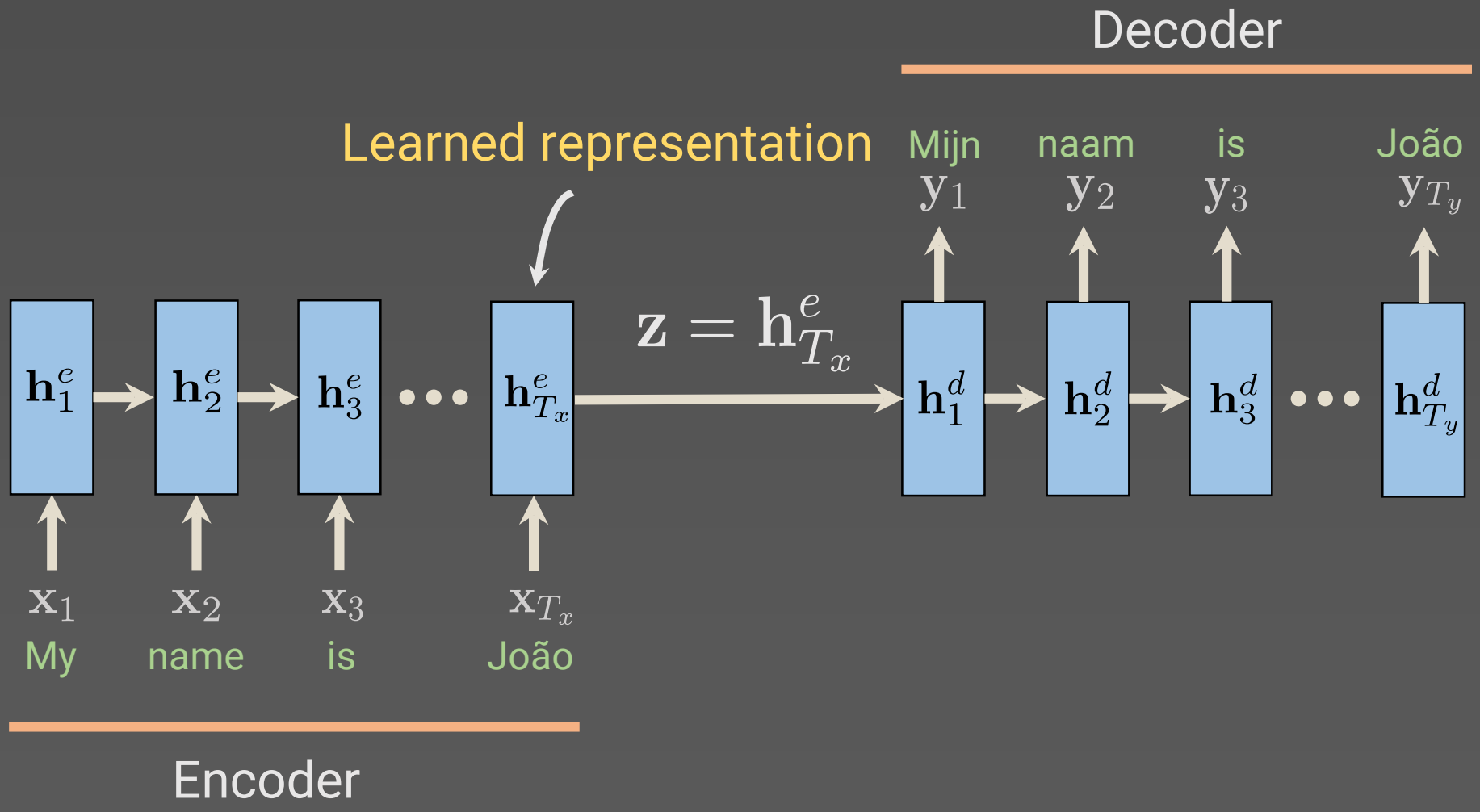
- In some applications, mapping **sequences to sequences** is useful.

Example: machine translation

My name is João ➡ Mijn naam is João

- A sequence to sequence model is an **encoder-decoder model**.

Sequence to Sequence Models



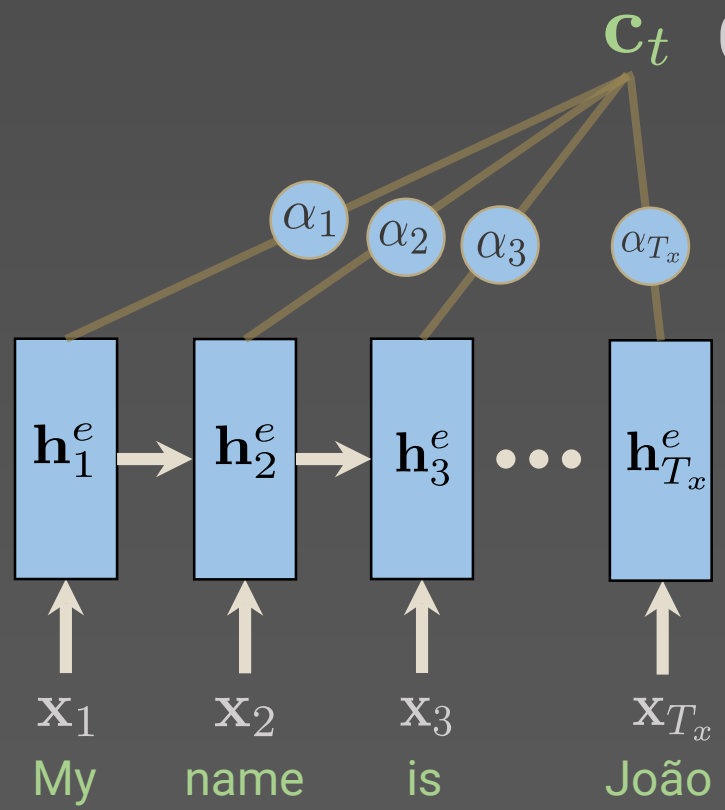
Attention

Why do we need attention?

Problem: Fixed-size representation z is the bottleneck of seq2seq.
Question: Is there a better way to pass the information from the encoder to the decoder?
Solution: Attention Mechanism

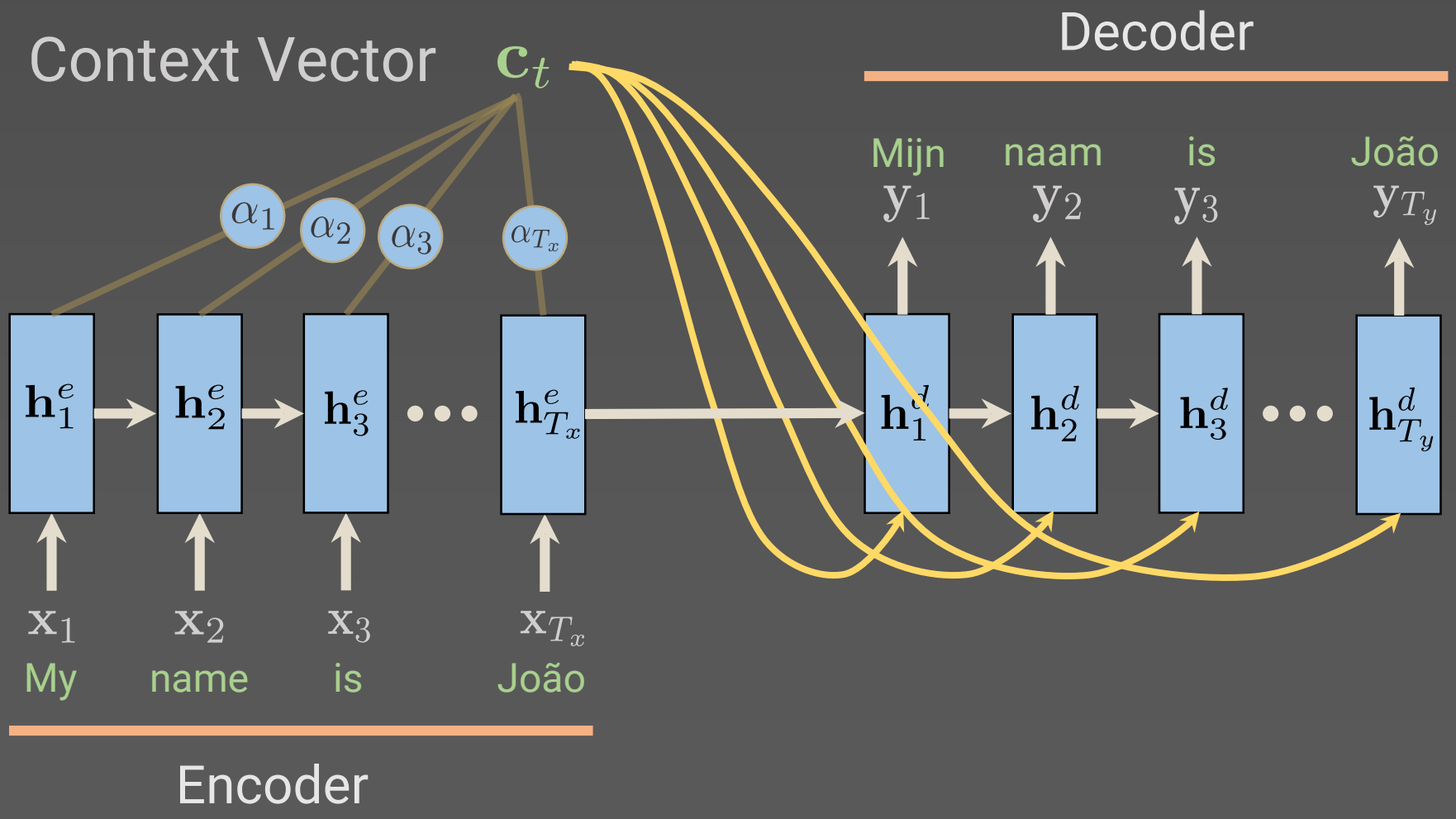
Bahdanau, Cho, and Bengio, **Neural Machine Translation by Jointly Learning to Align and Translate**
Luong, Pham, and Manning, **Effective Approaches to Attention-based Neural Machine Translation**

Attention Mechanism



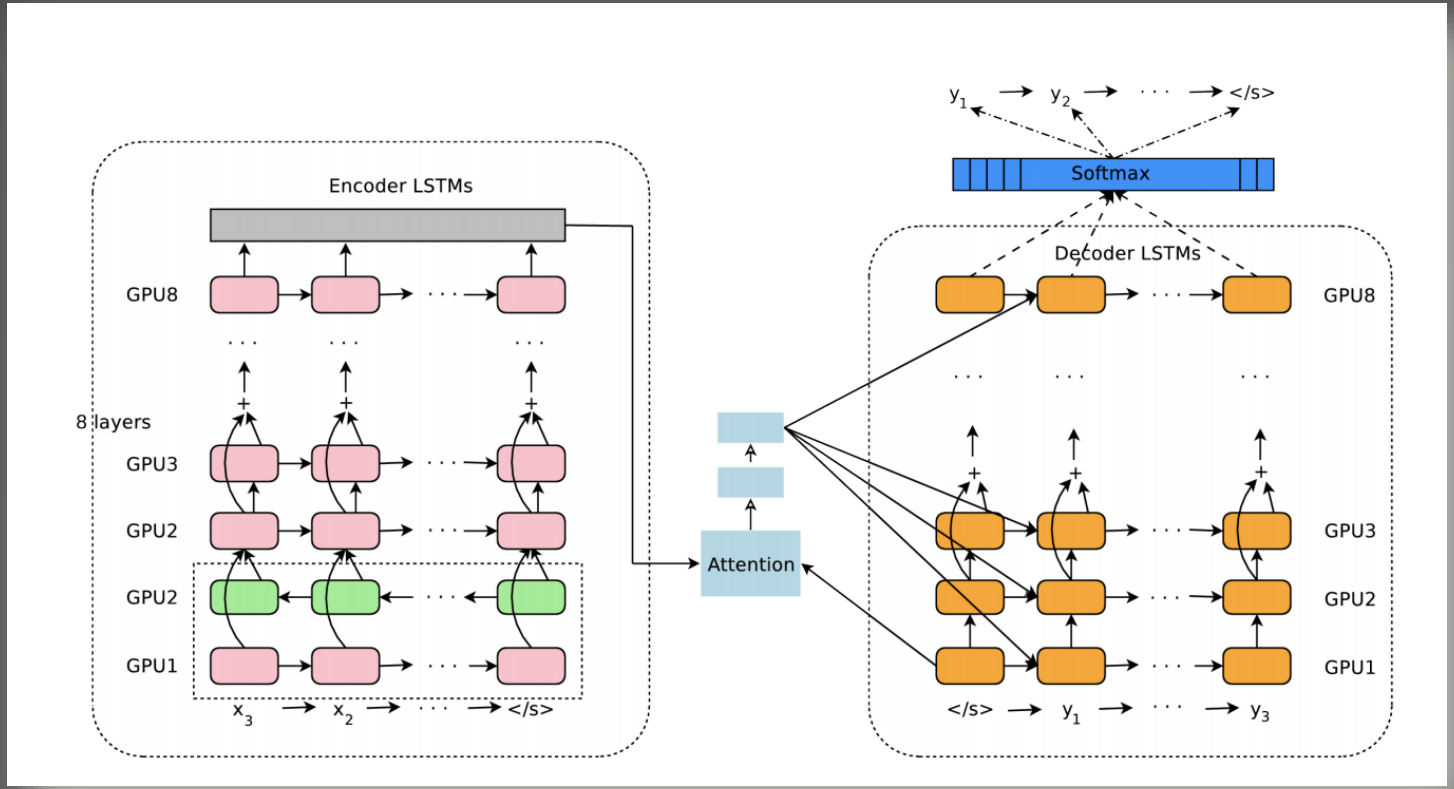
$$\mathbf{c}_t = \sum_{j=1}^{T_x} a_{tj} \mathbf{h}_j$$

Sequence to Sequence + Attention



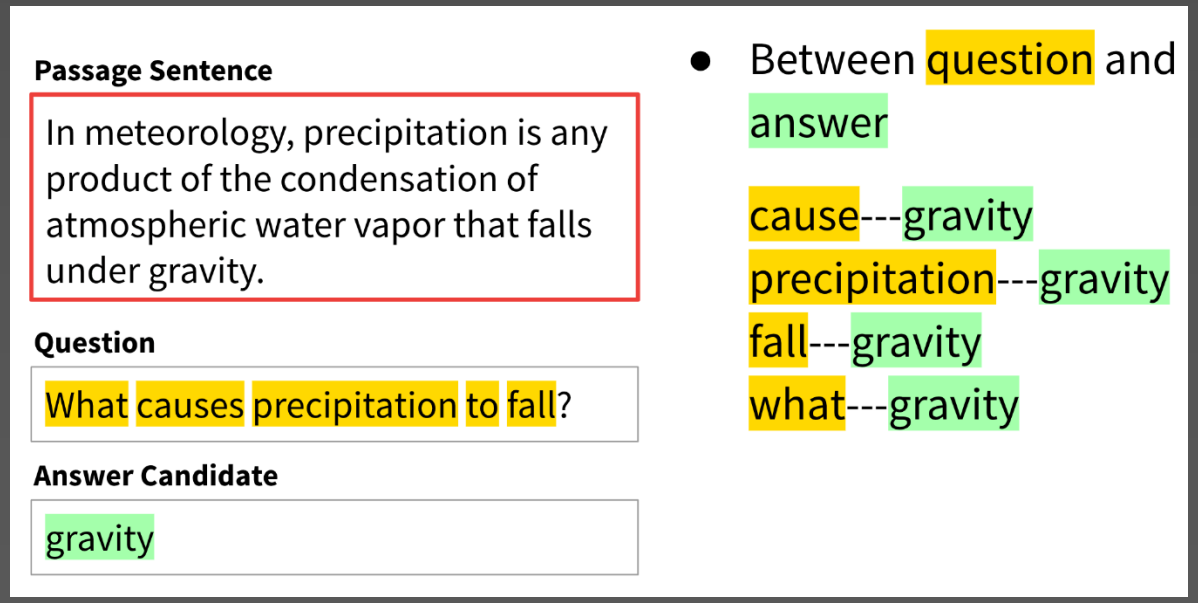
Application Examples

Google´s Neural Machine Translation System



Application Examples

Question Answering



What was missing?

Probabilistic deep learning models

- Deep generative models: variational autoencoders and generative adversarial networks

References – Part 1

Papers/Books

- A Few Useful Things to Know About Machine Learning, Domingos, 2012 (Link)
- Deep Learning, Ian Goodfellow *et al.* (Link)

Tutorials

- Tensorflow and Deep Learning Without a PhD (Link)
- Unsupervised Deep Learning, NeurIPS18 (Link)

References – Part 1

Courses

- Neural Networks for Machine Learning, Geoffrey Hinton, (Link)
- EE-559 Deep Learning, EPFL (Link)
- 6.S191 Deep Learning, MIT (Link)
- CS230 Deep Learning, Stanford, Andrew Ng (Link)
- Deep Learning, NYU, Yann LeCun (Link)

CNN References – Part 2

Courses/Tutorials

- Deep Learning for Computer Vision MIT 6.S191 (Link)
- Convolutional Neural Networks Lecture, Stanford (Link)
- Image Classification with Deep Learning (Link)
- Deep Learning And The Future of AI, Yann LeCun (Link)
- Deep dive into deep learning (Link)

Video

Deep Learning and the Future of Artificial Intelligence (Link)

How does the brain learn so much so quickly? (Link)

RNN References – Part 2

Courses/Tutorials

- LxMLS slides, Chris Dyer, DeepMind (Link)
- Deep Sequence Modeling MIT 6.S191 (Link)

Video

DLRLSS, Recurrent Neural Networks, Yoshua Bengio (Link)

Questions?

✉ mail@joao-pereira.pt

Thank you for your attention!