

# Backpropagation and Gradient Descent in Neural Networks

Antonio Rueda-Toicen

**Data Science Retreat**

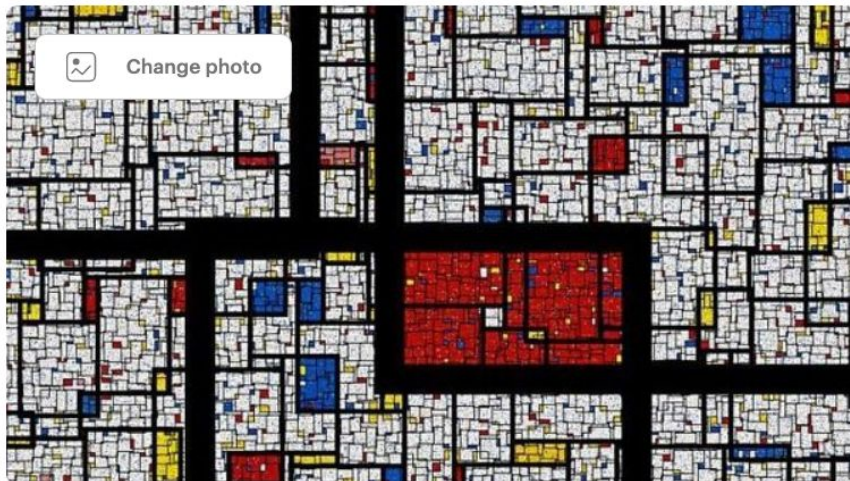
April 2023

# About me

- Senior Data Scientist at Vinted
- Background in academia (computer science and bioengineering)
  - Organizer of the [Berlin Computer Vision Group](#)
  - Arepa-lover (try them, they're *awesome*)



Fig 1: Arepas



## Berlin Computer Vision Group

📍 Berlin, Germany

👤 384 members · Public group ?

👤 Organized by Antonio Rueda Toicen

Share: [f](#) [t](#) [in](#)

[About](#)[Events](#)[Members](#)[Photos](#)[Discussions](#)[More](#)[Manage group](#) ▼[Create event](#) ▼

<https://www.meetup.com/Berlin-Computer-Vision-Group/>

# Agenda

- Intro to deep learning
  - DL in the machine learning project-cycle
  - What is DL and why should we use it?
    - Neural networks
    - Perceptrons
    - The feedforward pass
    - Backpropagation
    - Activation functions
    - Stochastic gradient descent
  - Our goal: build a image classifier for digits, first in Numpy, then in PyTorch

# Knowing the jargon

Model aka network aka architecture (although a fine-grained distinction exists here with respect to training)

Weight aka parameter aka connection strength

Hyperparameter

Capacity

Loss aka cost aka error function

Error rate aka  $1 - \text{accuracy}$

Sensitivity aka recall

Activation function

Transfer learning

Epoch vs Batch

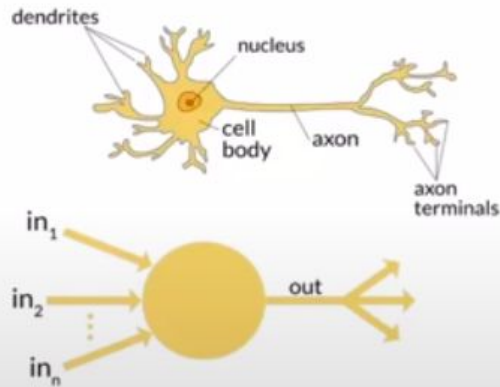
Convolution, receptive field

Linear aka “Dense” layer

# What is an artificial neural network?

In 1943 Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, teamed up to develop a mathematical model of an artificial neuron. They declared that:

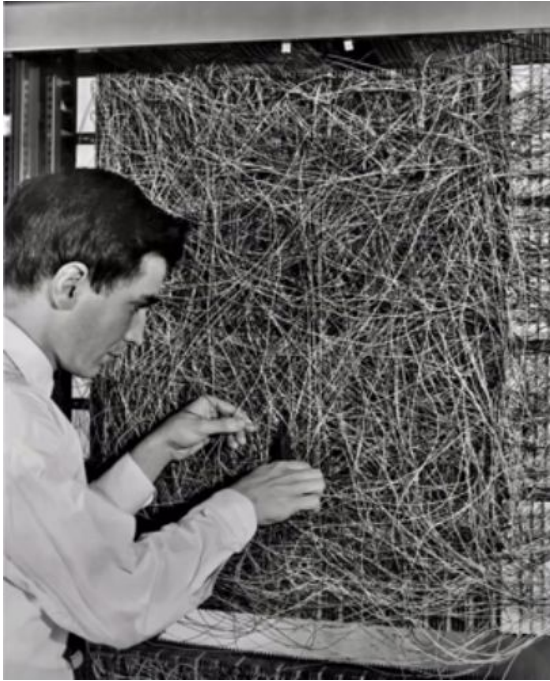
*Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms. (Pitts and McCulloch; A Logical Calculus of the Ideas Immanent in Nervous Activity)*



Artificial “neurons” are also called “processing units”

Recommended read: [“The Man Who Tried to Redeem the World with Logic”](#)

# What is an artificial neural network?

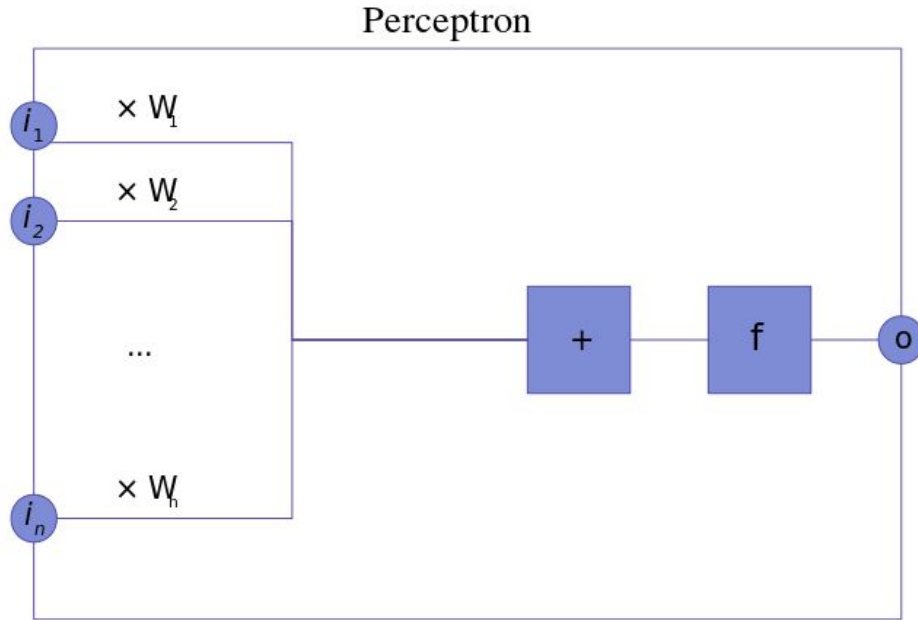


"we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control".  
Frank Rosenblatt

Image: Frank Rosenblatt with the Mark I Perceptron at Cornell University (1961)

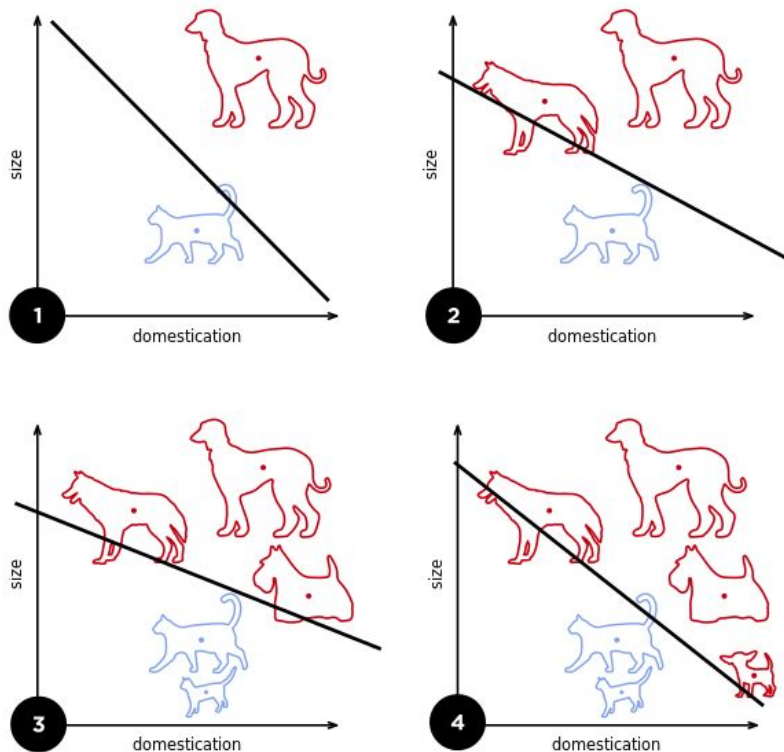


# What is a perceptron?



$$o = f\left(\sum_{k=1}^n i_k \cdot W_k\right)$$

# What is a perceptron?



[A fine-grained discussion about the differences in logistic regression and the perceptron algorithm](#)

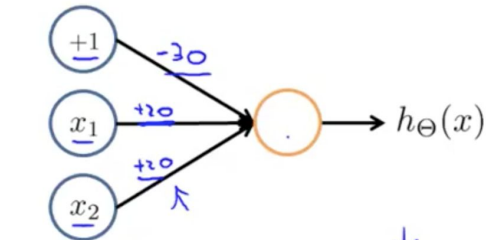
# What does a neural network 'learn'?

It learns “*weights*” aka “*parameters*” aka “*connection strengths*” that minimize a measure of error (aka “*loss*” aka “*cost*”) function given a set of training inputs  $x$  (aka “a dataset”)

## Simple example: AND

→  $x_1, x_2 \in \{0, 1\}$

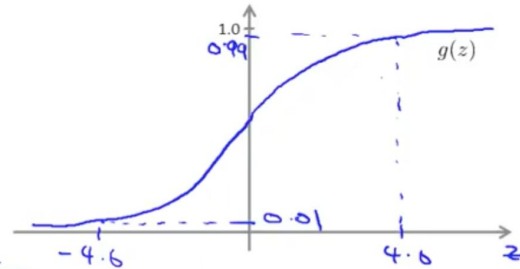
→  $y = x_1 \text{ AND } x_2$



$$\rightarrow h_{\theta}(x) = g(-30 + 20x_1 + 20x_2)$$

Annotations below the equation:

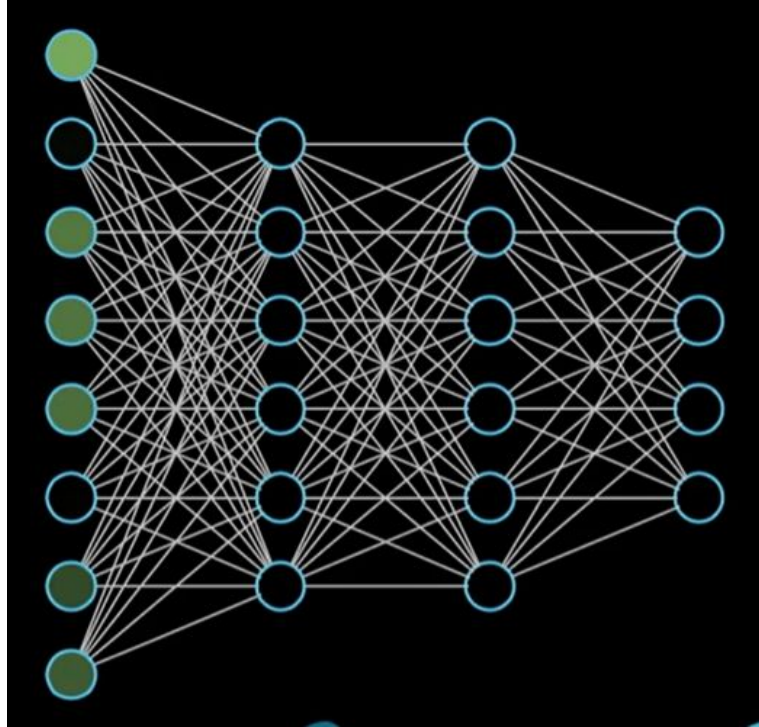
- $\omega_{10}^{(1)}$  points to -30
- $\omega_{11}^{(1)}$  points to 20
- $\omega_{12}^{(1)}$  points to 20



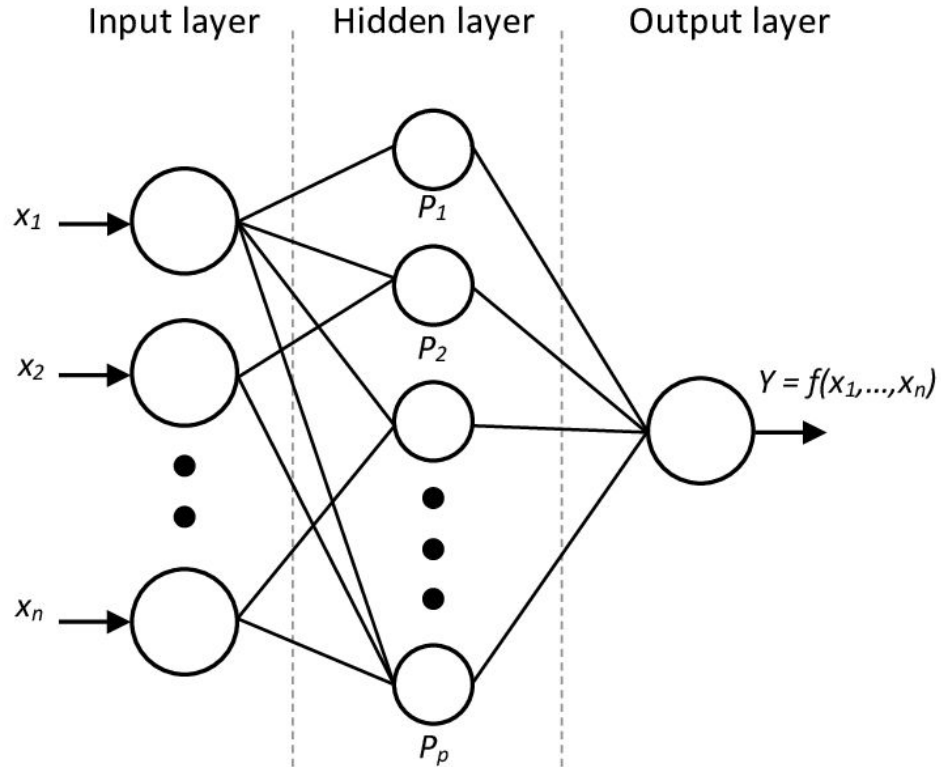
$x_1$	$x_2$	$h_{\theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$$h_{\theta}(x) \approx x_1 \text{ AND } x_2$$

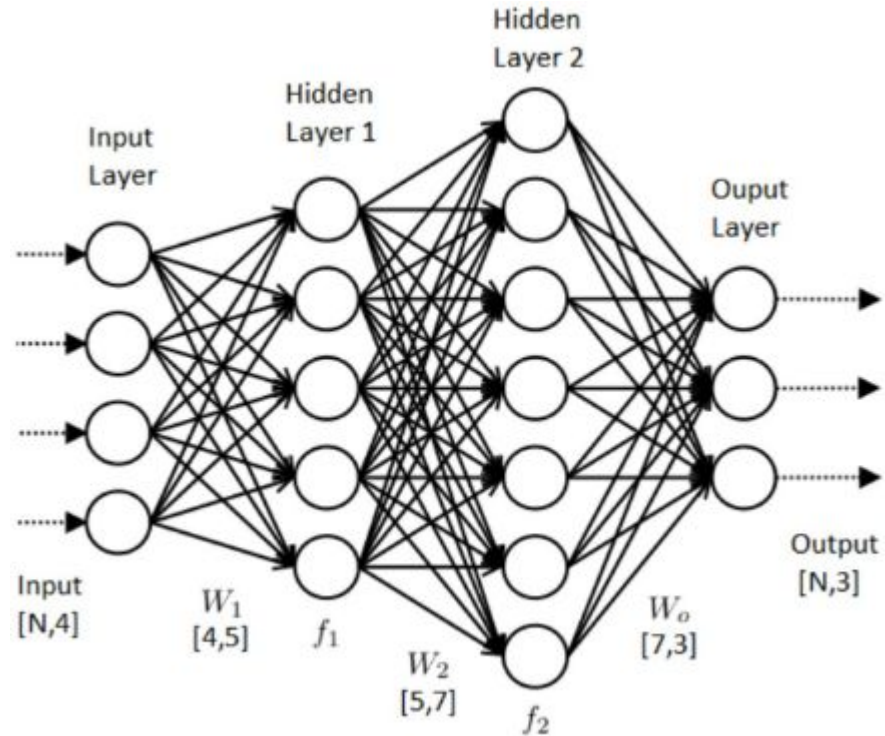
# Multilayer perceptrons



# Hidden layers



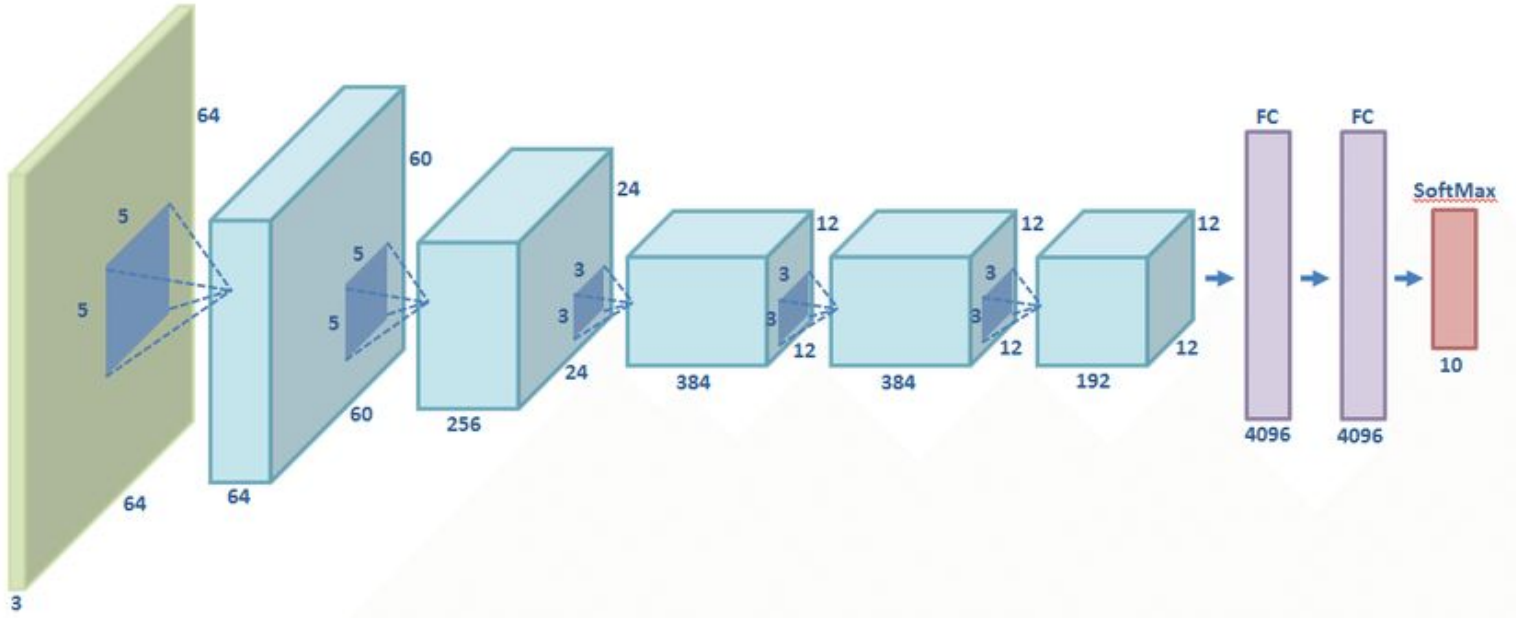
# Hidden layers



# Stir the pile



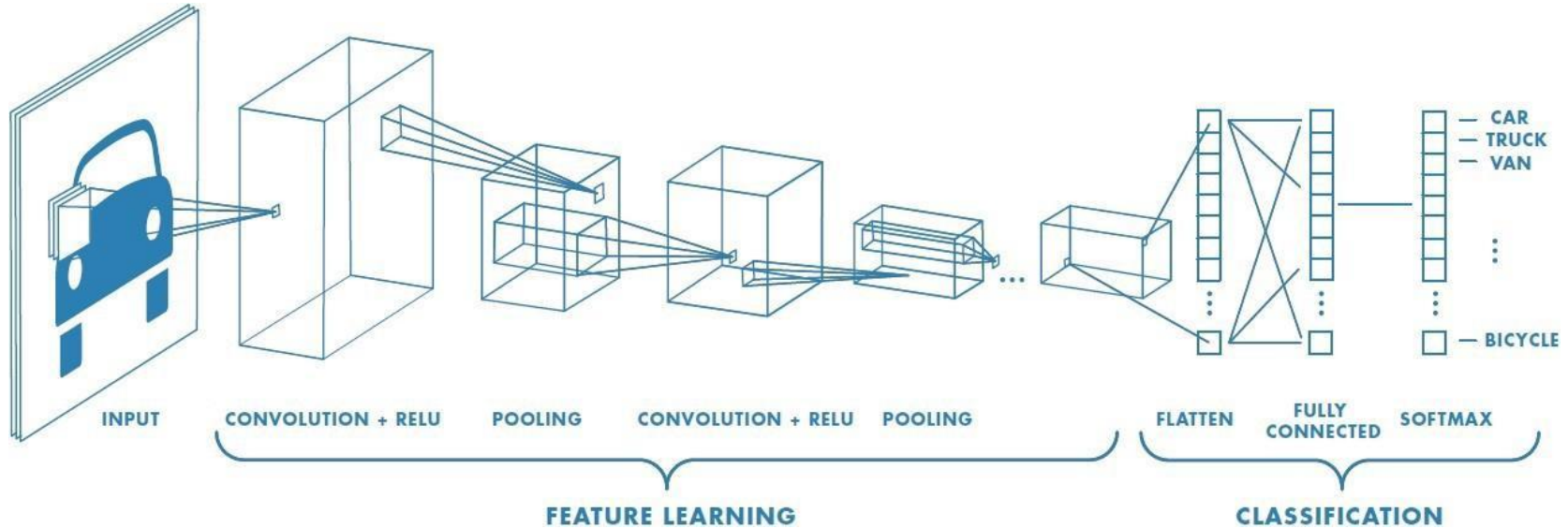
Different network architectures represent different ways to 'stir the pile'



[Architecture of the Alexnet network](#)



# What is deep learning?



# Visualizing a CNN

[Convolutional Neural Network Visualization by Otavio Good](#)

# What is a GPU?



Most deep learning libraries require the use of NVIDIA graphical processing units with CUDA capabilities

# Why use a GPU for deep learning?

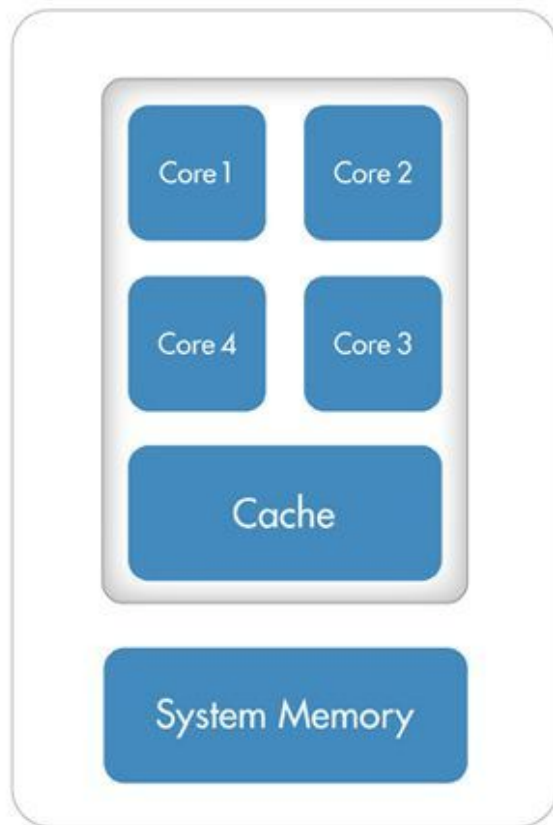
How is a CPU different from a GPU?

- More cores!

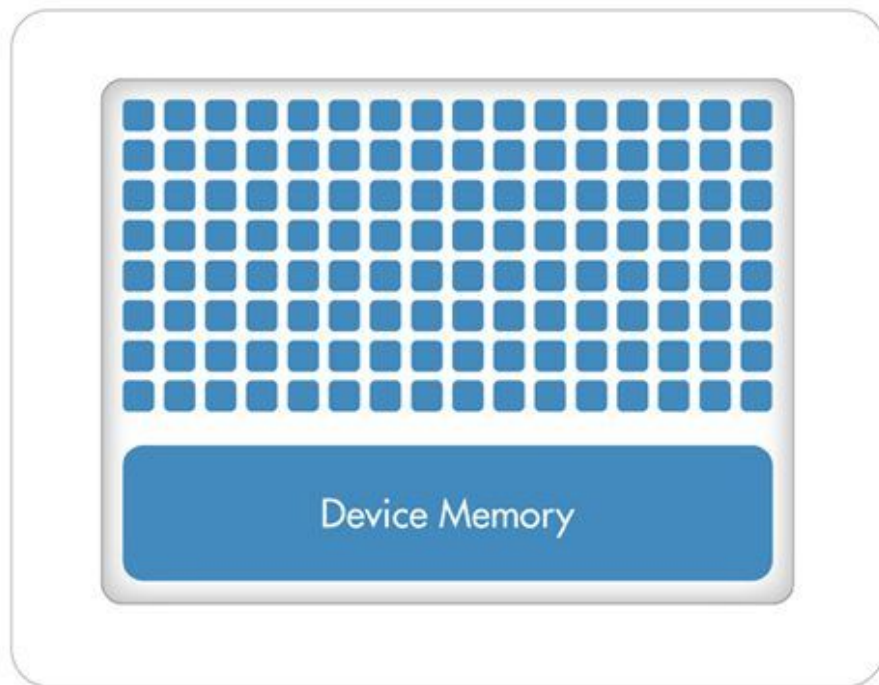
Why are CPUs less effective for deep learning?

- Less cores :(

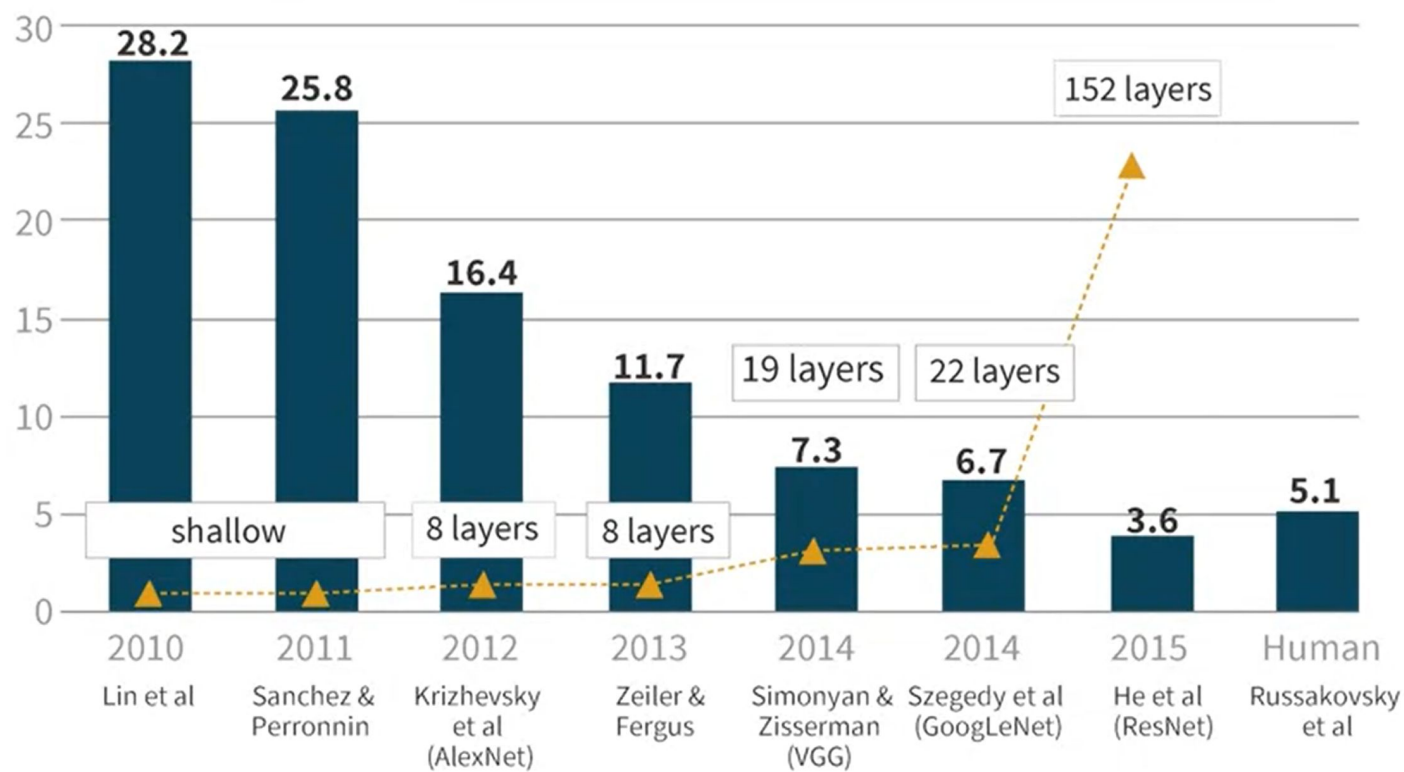
## CPU (Multiple Cores)



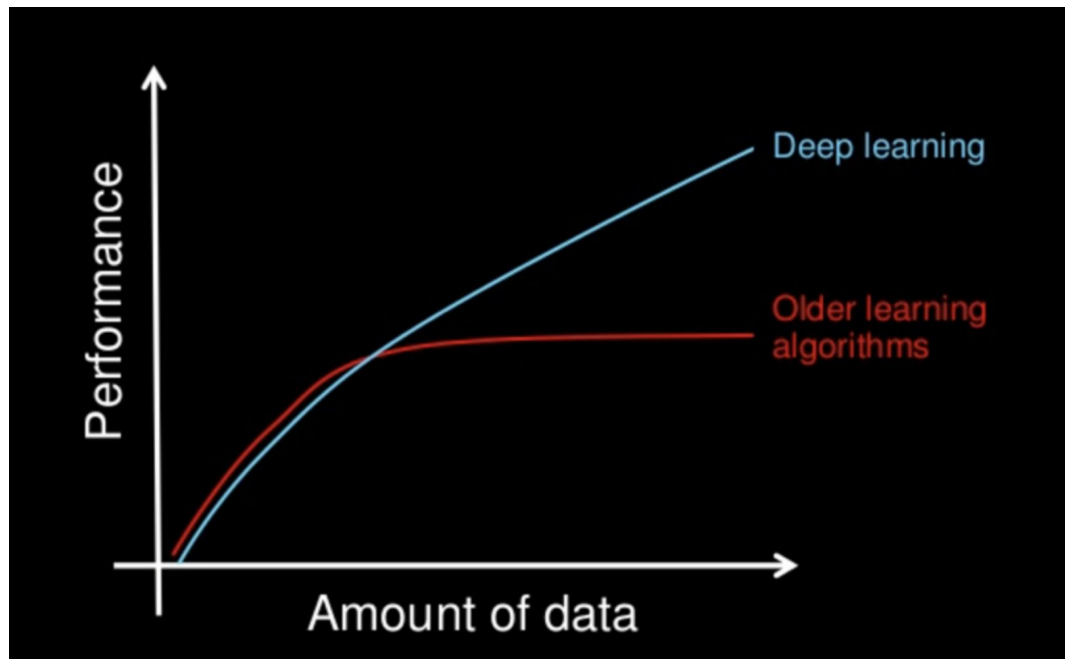
## GPU (Hundreds of Cores)



## IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE (ILSVRC) WINNERS



# Why do we use deep learning?



- Disclaimer: the [No Free Lunch Theorems](#) explain why this graph might be hype, although things like the [current GPT experiments](#) show promise

# Applications of deep learning

- Computer vision: image classification, object detection, segmentation, image description, image generation, satellite and drone image analysis
- Natural Language Processing: translation, text and DNA sequence prediction, text generation
- Audio understanding, audio generation, lip reading
- Playing games, finding optimal strategies (when paired with reinforcement learning)
- Any system that requires the finding of **significant correlations** or patterns/”intuitions” in the data, i.e. the ‘thinking fast’ systems referred by Daniel Kahneman in his famous book.



# Why do we use deep learning?

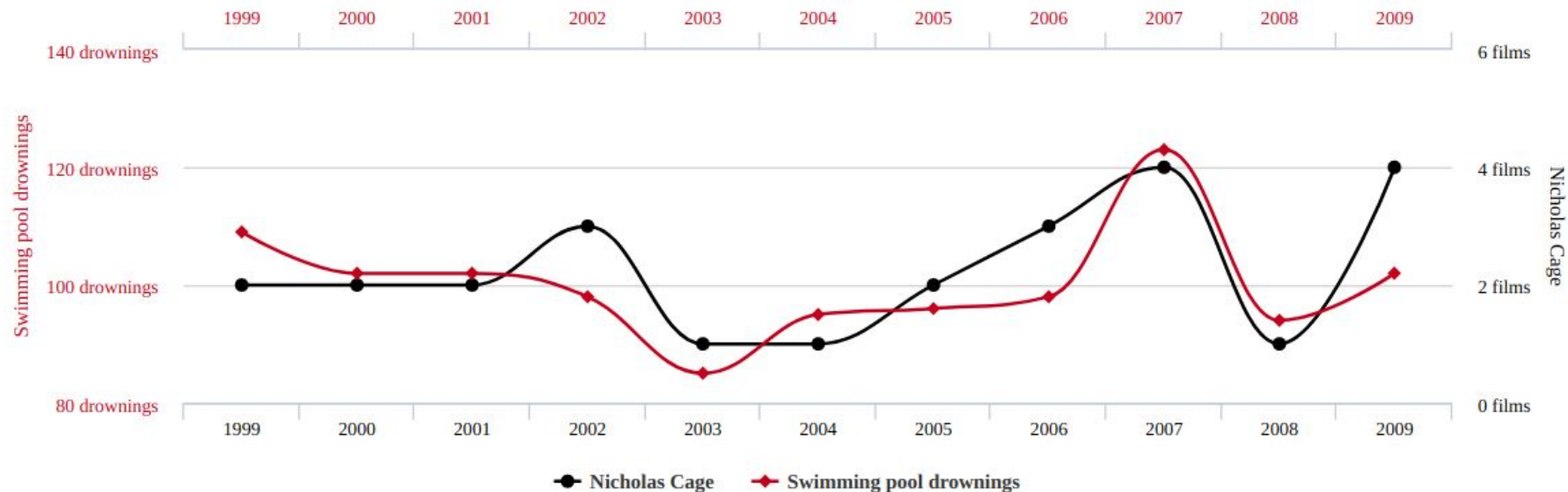
- Deep neural networks are **universal function approximators**
  - *This means that given the right architecture and training, they can **compute everything that's computable**, just like a [Universal Turing Machine](#)*
  - *In practice this means that deep networks are **very good** at finding very sophisticated decision boundaries (aka “curve fitting”)*
  - *Finding the function means the finding **architecture and or training** of the neural network*
  - *It's **fair** to call deep neural networks “**linear regression on steroids**”*
  - *Deep neural networks pick hidden **correlations** in the data, but most architectures [don't pick on causality](#) (current research topic)*

# Number of people who drowned by falling into a pool

correlates with

## Films Nicolas Cage appeared in

Correlation: 66.6% ( $r=0.666004$ )



Data sources: Centers for Disease Control & Prevention and Internet Movie Database

tylervigen.com

Choose the right loss function for the task

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

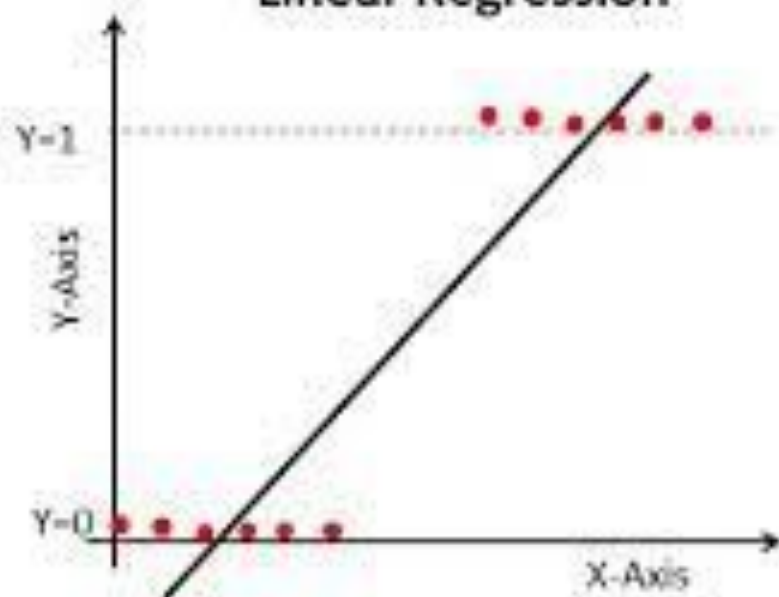
Mean Squared Error is a loss function for regression

# Choose the right loss function for the task

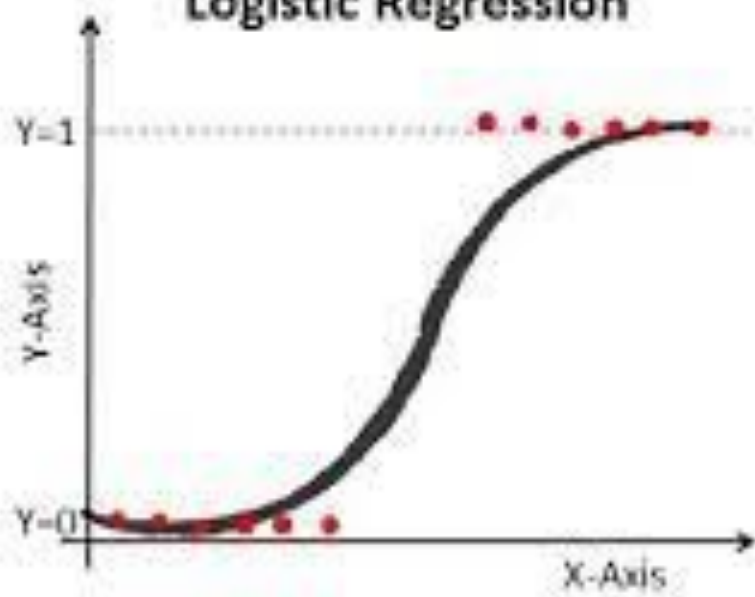
$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Cross entropy is a function for classification, used interchangeably with [negative log-likelihood](#), here we see it in [its binary case](#)

Linear Regression

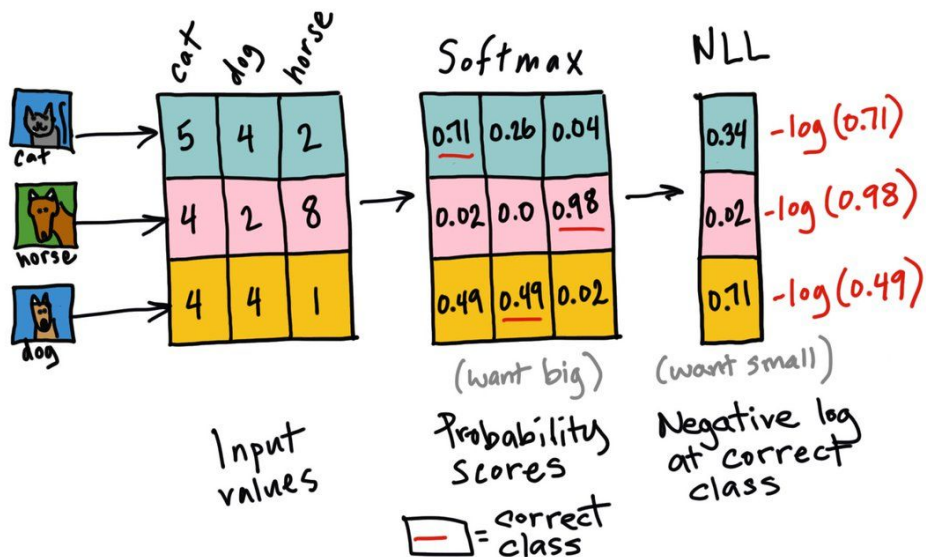


Logistic Regression



# Can we solve this task with logistic regression?

## Negative Log Likelihood (NLL) Loss



Q: Which loss function should we use for binary logistic regression?

- A. Mean Squared Error
- B. Cross Entropy
- C. Negative log-likelihood ([huh?](#) Maybe read [this](#) too later.)
- D. Precision & Recall
- E. None of the above


## Cross Entropy


---


$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x)$$



# Cross entropy

  $p_1 = 0.8$

  $p_2 = 0.7$

  $p_3 = 0.1$

$r_i = 1$  if present on box  $i$



 0.8

$p_1$

$y_1 = 1$



 0.7

$p_2$

$y_2 = 1$



 0.9

$1 - p_3$

$y_3 = 0$

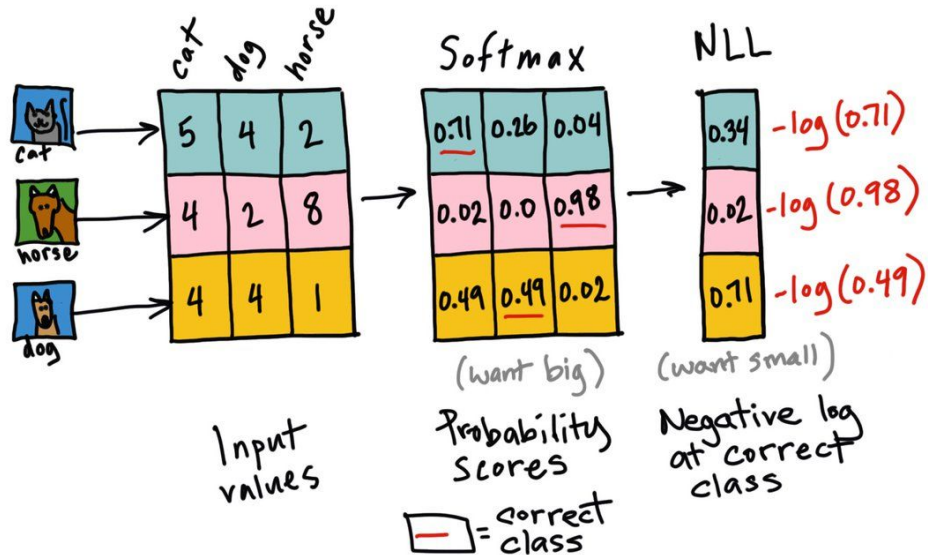
Cross-Entropy

$-\ln(0.8) - \ln(0.7) - \ln(0.9)$

$$\text{Cross-Entropy} = - \sum_{i=1}^m y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

[Explore the Colab notebook](#)

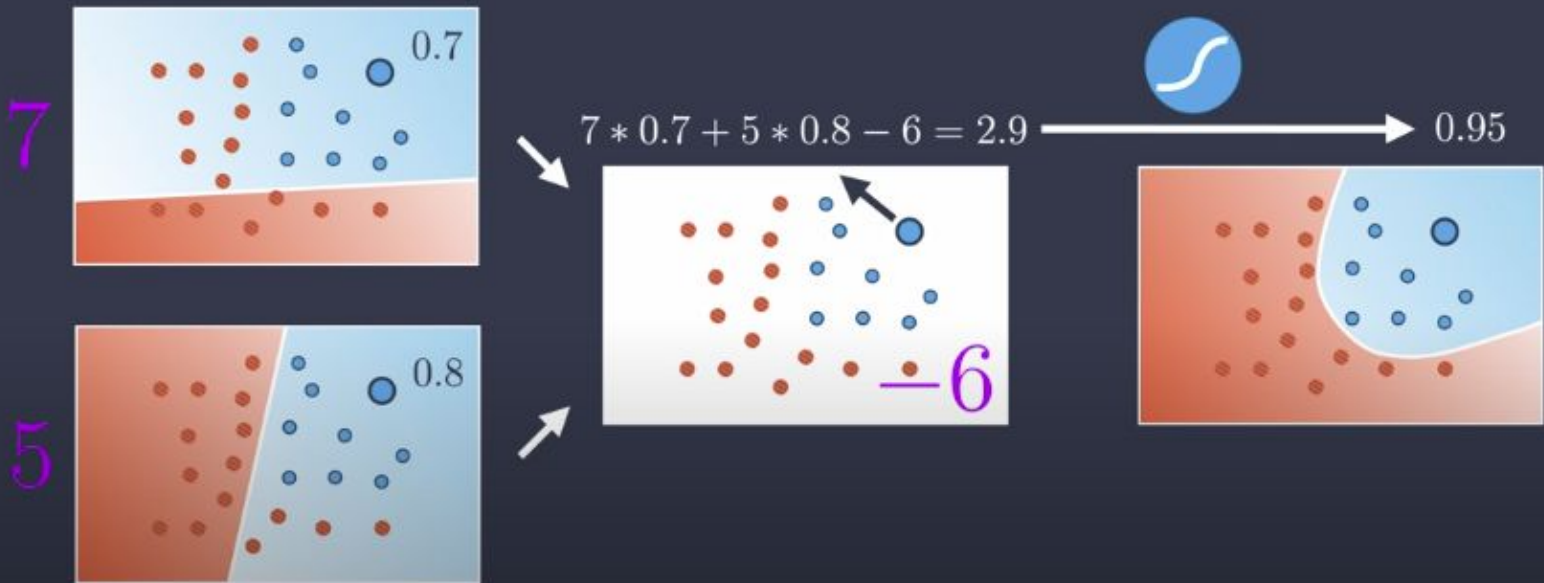
# Negative Log Likelihood (NLL) Loss



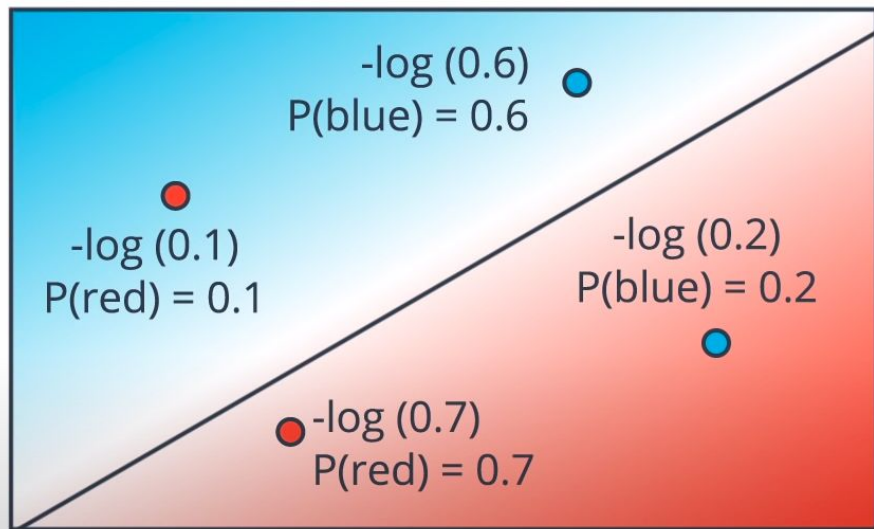
[Exercise](#)  
[Solution](#)

Refer to [this tutorial](#) and [this discussion](#)

# Neural Network



# Error Function



$$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7) = 4.8$$

0.51

1.61

2.3

0.36

If  $y = 1$

$$P(\text{blue}) = \hat{y}$$

$$\text{Error} = -\ln(\hat{y})$$

If  $y = 0$

$$P(\text{red}) = 1 - P(\text{blue}) = 1 - \hat{y}$$

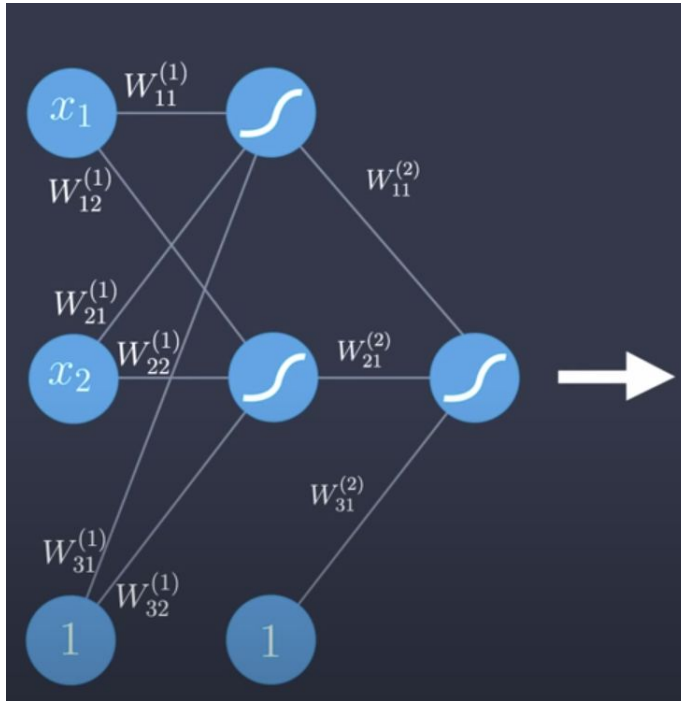
$$\text{Error} = -\ln(1 - \hat{y})$$

$$\text{Error} = -(1-y)(\ln(1-\hat{y})) - y\ln(\hat{y})$$

$$\text{Error Function} = -\frac{1}{m} \sum_{i=1}^m (1-y_i)(\ln(1-\hat{y}_i)) + y_i \ln(\hat{y}_i)$$

Q: Which of these points have higher cross entropy?

# The feedforward pass



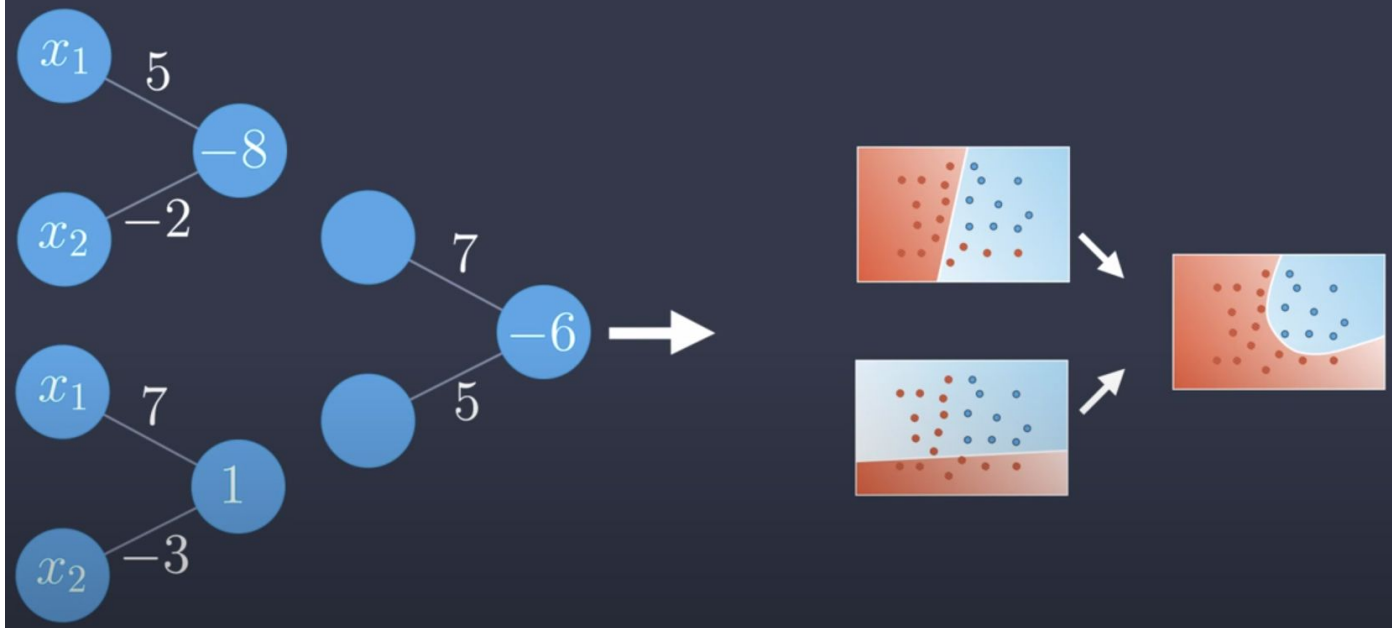
$$\hat{y} = \sigma \begin{pmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{pmatrix} \sigma \begin{pmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

$$\hat{y} = \sigma \circ W^{(2)} \circ \sigma \circ W^{(1)}(x)$$

[Understanding the feedforward pass](#)

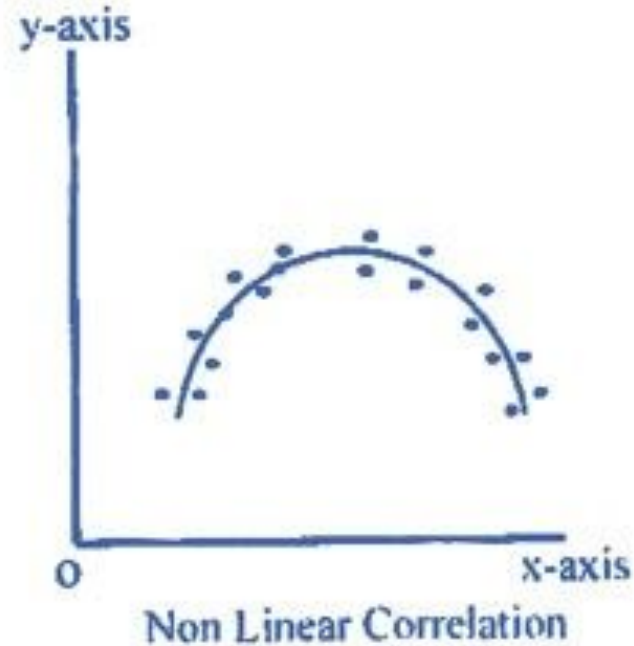
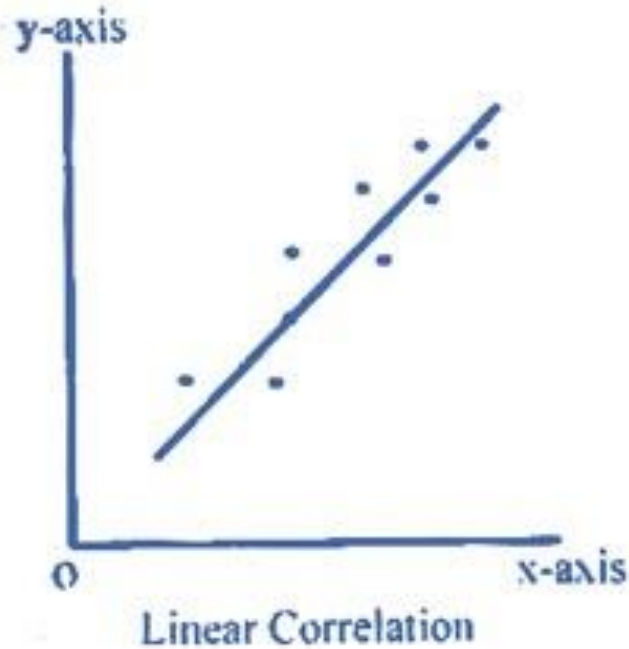
# Learning non-linearities

Neural Network



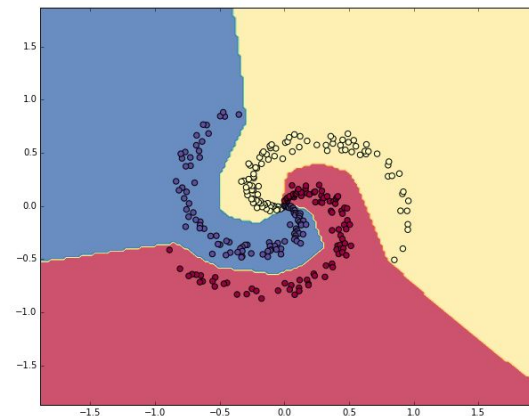
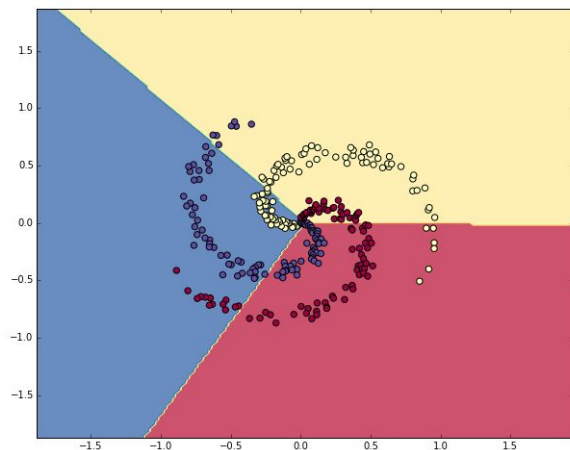
[How neural networks learn non-linearities](#)

# Learning non-linearities



[How neural networks learn non-linearities](#)

# Learning non-linearities



<https://cs231n.github.io/neural-networks-case-study>

[Experiment notebook](#)

**Q: What happens with large learning rates? What happens with large coefficients for l2 regularization?**



Scalar

Vector

Matrix

Tensor

1

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$
$$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$$

- [Read: scalars, vectors, matrices, and tensors](#)
  - [Understanding rank terminology](#)

# How do computers represent images?

```
206 205 247 245 244 253 247 245 136 151 258 255 255 255 255 234 207 231 255 254 254 255 255 254 255 252 252 255 255 254 255 247
244 181 137 244 254 255 254 255 118 103 209 238 155 153 238 103 74 52 66 173 239 254 254 255 255 254 255 254 255 254 254 184
232 254 75 200 249 255 255 255 103 96 84 61 35 44 89 53 44 45 54 140 213 252 255 255 255 255 245 187 186 176 223
983 209 985 142 223 255 255 252 117 75 41 35 31 24 25 36 45 44 44 46 81 118 148 234 252 254 255 245 231 148 250 254
67 69 107 196 238 255 255 255 104 25 34 35 29 20 25 34 32 30 32 34 53 85 100 142 231 242 247 249 255 255 255
55 51 45 134 218 251 255 232 51 12 26 33 24 24 40 79 82 78 71 60 58 53 67 90 136 228 208 158 252 246 249 255
79 56 56 75 224 255 255 118 11 27 74 99 91 106 140 162 173 173 172 172 158 137 92 46 70 187 217 106 254 222 233 255
383 43 47 52 147 255 229 56 41 81 129 145 180 169 169 172 178 178 178 177 177 172 110 11 82 209 238 255 244 249 255
403 40 33 36 98 245 171 32 65 110 139 143 151 162 171 174 178 178 182 184 187 183 173 162 71 45 167 255 254 255 254 255
37 44 44 31 69 250 158 36 70 129 143 142 153 162 171 177 177 178 182 181 184 188 180 170 120 51 137 255 254 250 254 255
34 45 51 64 110 237 182 53 116 138 140 143 154 164 176 178 174 177 182 186 188 185 183 178 140 66 141 254 252 225 249 255
34 36 52 74 71 188 156 63 132 134 144 155 180 163 173 179 178 179 180 182 180 185 187 182 156 93 148 250 254 234 247 255
32 35 52 54 159 250 126 57 129 138 138 140 151 156 166 168 171 178 180 187 188 185 185 183 180 102 136 242 255 255 254 254
36 32 72 129 212 228 115 85 121 104 102 104 94 103 134 158 170 182 125 108 121 143 155 180 191 104 134 230 252 252 255 253
61 82 116 107 179 247 124 60 101 90 111 119 103 81 94 147 181 178 126 98 123 153 147 161 200 62 100 222 207 167 227 215
144 178 167 231 230 232 170 67 115 88 78 62 83 85 88 139 182 180 135 80 53 99 141 165 201 97 70 182 245 235 248 249
127 145 149 195 204 213 187 95 133 123 117 133 126 108 110 139 191 187 167 129 127 148 147 171 186 110 121 128 233 180 215 212
87 112 100 70 85 102 85 75 142 148 151 153 135 125 120 149 181 180 183 175 174 192 186 186 208 127 163 239 216 149 186 195
63 83 106 134 129 106 39 78 132 142 155 159 139 111 124 164 195 200 186 182 181 195 200 202 200 143 217 253 249 142 230 234
69 70 78 113 97 74 43 106 127 140 152 155 125 97 112 150 185 184 174 183 196 196 202 208 208 166 247 254 255 254 254 254
72 44 63 50 40 52 49 74 127 137 146 149 132 103 70 90 134 141 168 165 189 187 204 203 216 183 236 244 251 242 236 243
55 20 69 73 59 80 46 74 117 127 144 161 148 124 105 120 156 187 183 182 189 206 201 206 214 204 174 185 187 188 183 193
65 49 77 89 50 68 43 61 108 127 141 147 113 100 121 145 148 189 183 178 181 201 201 205 202 174 186 189 178 183 188 184
82 70 92 70 54 58 37 47 98 121 132 116 89 70 111 146 163 149 122 124 160 187 187 186 178 149 148 252 155 157 150 168
104 107 122 123 105 79 27 33 66 111 122 120 114 114 147 175 180 196 163 101 170 200 187 185 156 146 145 139 137 141 140 145
117 124 127 133 135 105 21 28 37 88 119 121 128 128 141 142 168 102 212 253 164 186 180 188 154 146 144 149 151 153 147 144
119 116 118 125 128 111 21 29 28 58 100 118 131 140 151 159 188 101 205 182 160 168 149 186 119 144 147 143 140 141 144 148
117 119 125 130 139 106 18 29 44 58 70 102 132 147 168 187 212 215 120 195 177 152 133 195 57 59 126 151 145 143 142 141
119 123 126 134 145 102 27 54 52 38 49 69 105 135 175 189 190 216 106 166 139 111 164 203 74 5 121 151 142 142 143 146
101 108 123 121 132 125 44 40 31 35 57 44 58 101 147 144 138 183 145 94 80 145 196 187 84 40 185 180 142 144 142 145
983 97 97 96 104 70 34 33 30 40 41 49 51 58 74 53 55 66 63 89 150 188 208 156 62 108 140 149 125 133 131 131
102 102 97 88 73 35 30 23 42 50 65 41 90 80 59 51 57 82 123 157 187 205 168 62 95 151 105 101 154 135 130 129
```

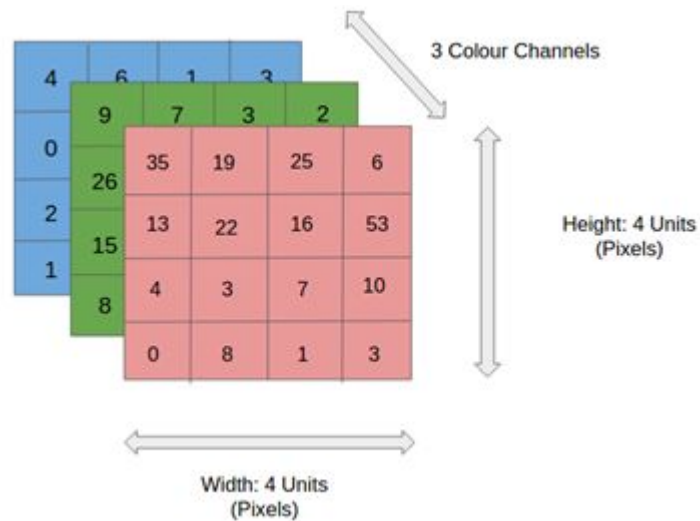


<https://setosa.io/ev/image-kernels/>

# Images as tensors

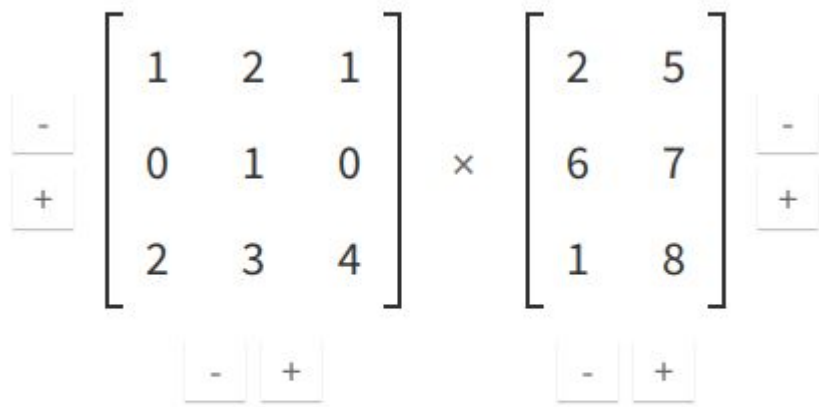


**What a human sees**



**What the computer 'sees'**

# Matrix multiplication



The image shows a matrix multiplication interface. It features two 3x3 matrices separated by a multiplication symbol (×). Each matrix is enclosed in large square brackets. To the left of the first matrix is a vertical stack of two buttons: a minus sign (-) on top and a plus sign (+) on the bottom. To the right of the second matrix is a similar vertical stack with minus (-) on top and plus (+) on the bottom. Below each matrix is a horizontal stack of two buttons: minus (-) on the left and plus (+) on the right. All buttons are white with black text and a thin border.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 6 & 7 \\ 1 & 8 \end{bmatrix}$$

▶ Multiply

<http://matrixmultiplication.xyz/>

# Matrix multiplication

$$\begin{bmatrix} 2 & -2 \\ 5 & 3 \end{bmatrix} \times \begin{bmatrix} -1 \\ 7 \end{bmatrix} \begin{bmatrix} 4 \\ -6 \end{bmatrix} = \begin{bmatrix} (2)(-1) + (-2)(7) & 2 \cdot 4 + (-2)(-6) \\ 5(-1) + 3(7) & 5 \cdot 4 + 3(-6) \end{bmatrix}$$

# Pay attention to the size of the input



**Anthony Wang** @aytwang · 15h

Replying to @the\_antlr\_guy

Wow, this is extraordinary. I can't count the number of hours I've spent in the debugger combing through every variable that could have gone wrong using `print(x.shape)` statements 😂



**Terence Parr** @the\_antlr\_guy · 15h

haha! Same here! Seems like `print(x.shape)` is the most common deep learning programming construct there is!



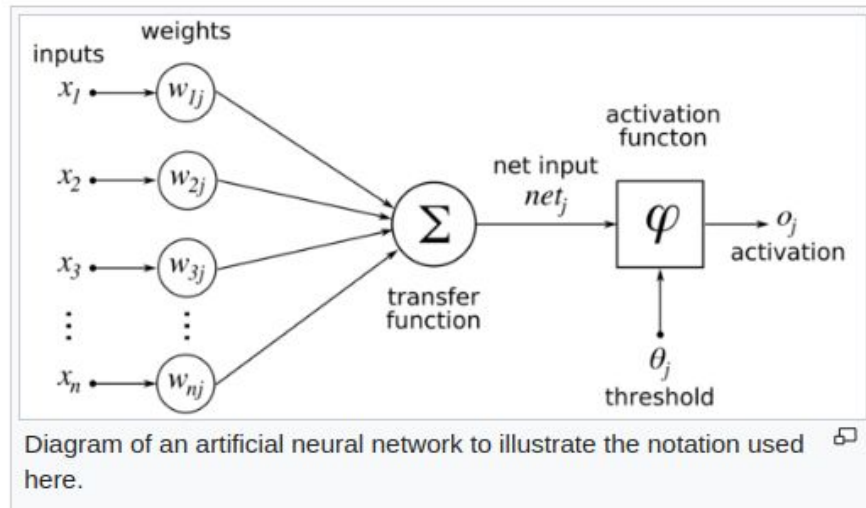
# Backpropagation

## Finding the derivative of the error [\[ edit \]](#)

Calculating the [partial derivative](#) of the error with respect to a weight  $w_{ij}$  is done using the [chain rule](#) twice:

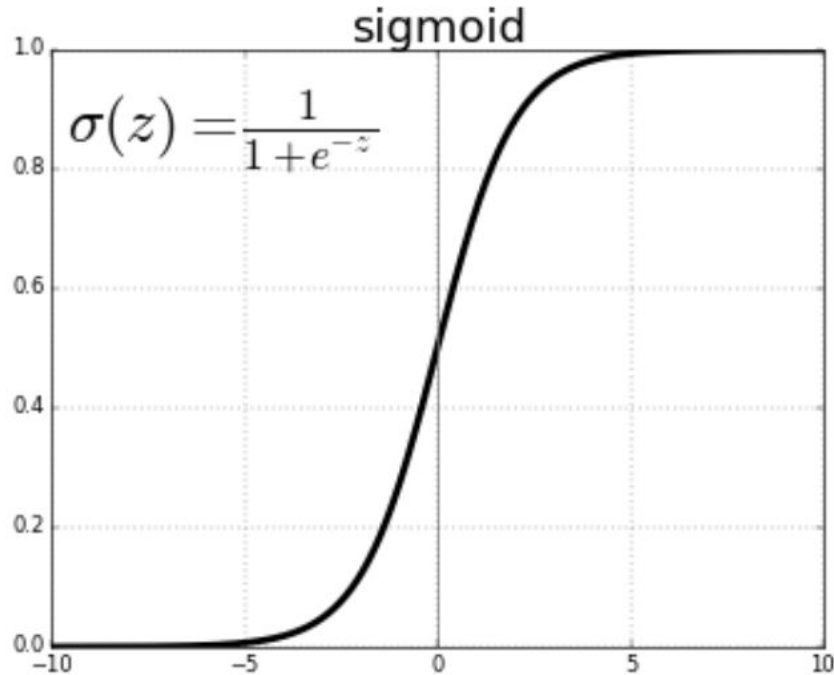
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}} \quad (\text{Eq. 1})$$

In the last factor of the right-hand side of the above, only one term in the sum  $\text{net}_j$  depends on  $w_{ij}$ , so that



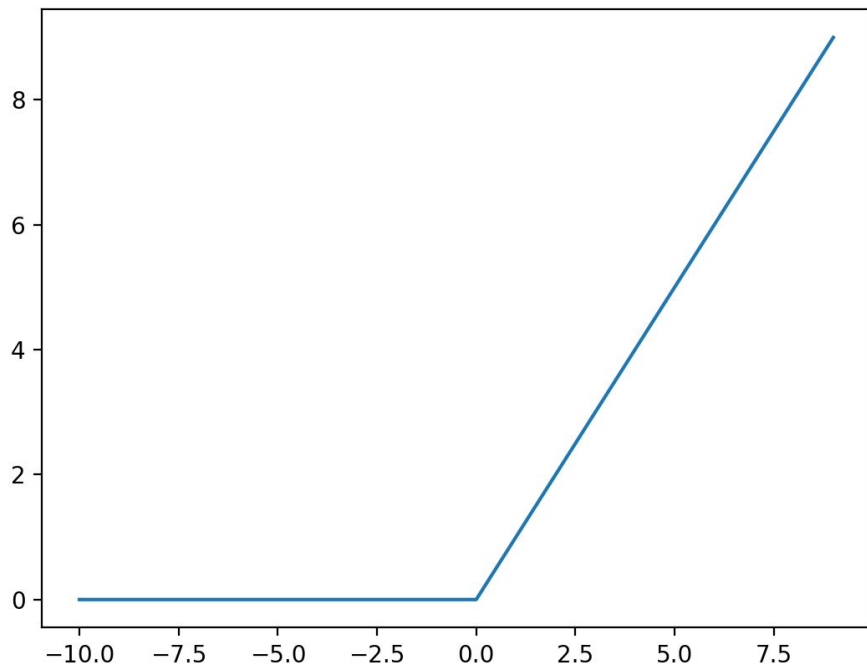
<https://en.wikipedia.org/wiki/Backpropagation>

# The sigmoid activation function





# The rectifier linear unit (ReLU) activation function

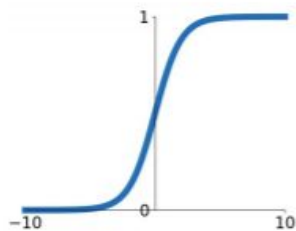


$$f(x) = x^+ = \max(0, x)$$

# Activation functions

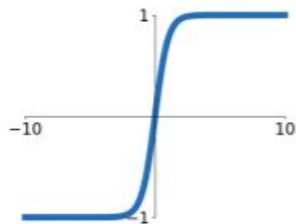
## Sigmoid

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



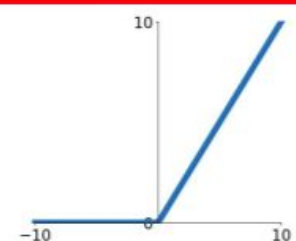
## tanh

$$\tanh(x)$$



## ReLU

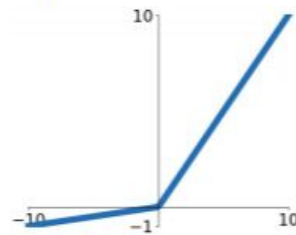
$$\max(0, x)$$



ReLU is a good default choice for most problems

## Leaky ReLU

$$\max(0.1x, x)$$

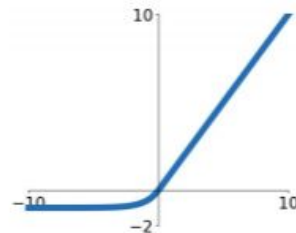


## Maxout

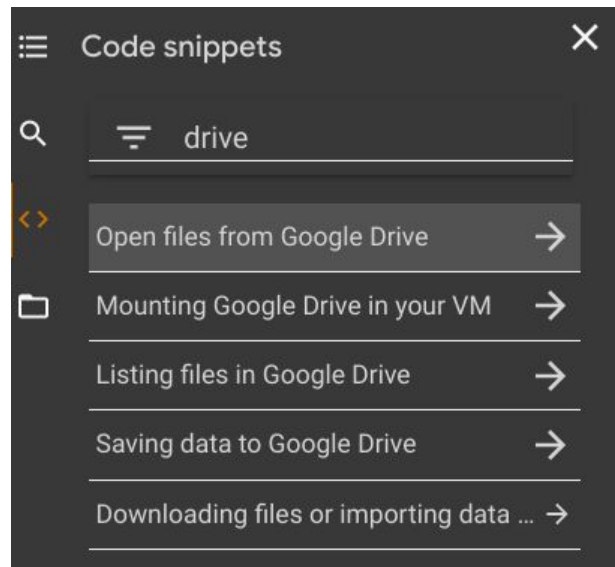
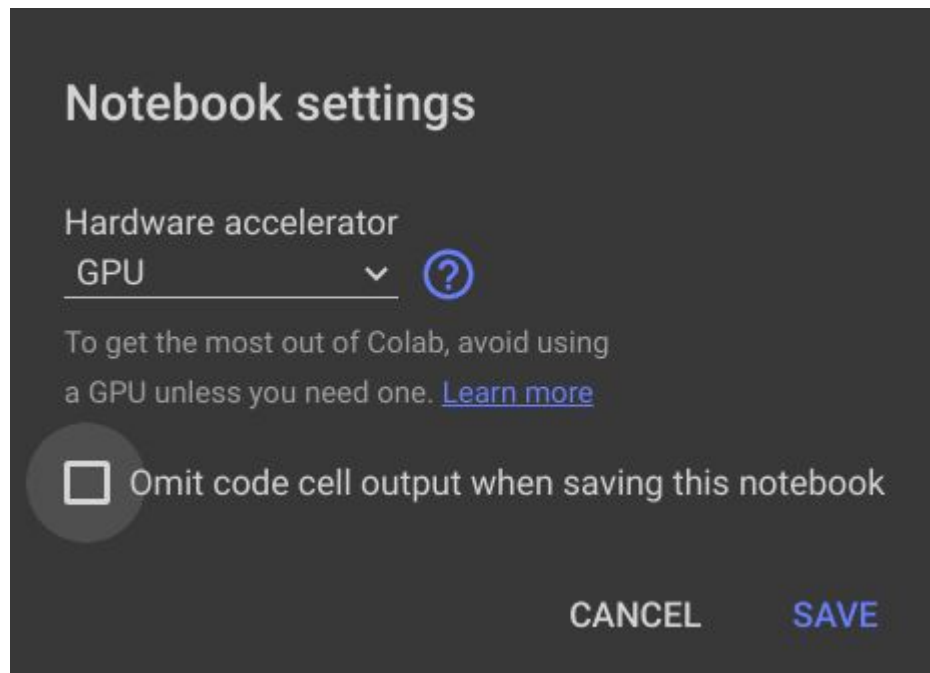
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Setting up Google Colab



# Installing PyTorch

PyTorch Build	Stable (1.7.1)				Preview (Nightly)		
Your OS	Linux		Mac		Windows		
Package	Conda		Pip		LibTorch		Source
Language	Python				C++ / Java		
CUDA	9.2	10.1	10.2	11.0	None		
Run this Command:	<pre>pip install torch==1.7.1+cpu torchvision==0.8.2+cpu torchaudio==0.7.2 -f https://download.pytorch.org/whl/torch_stable.html</pre>						

<https://pytorch.org/>

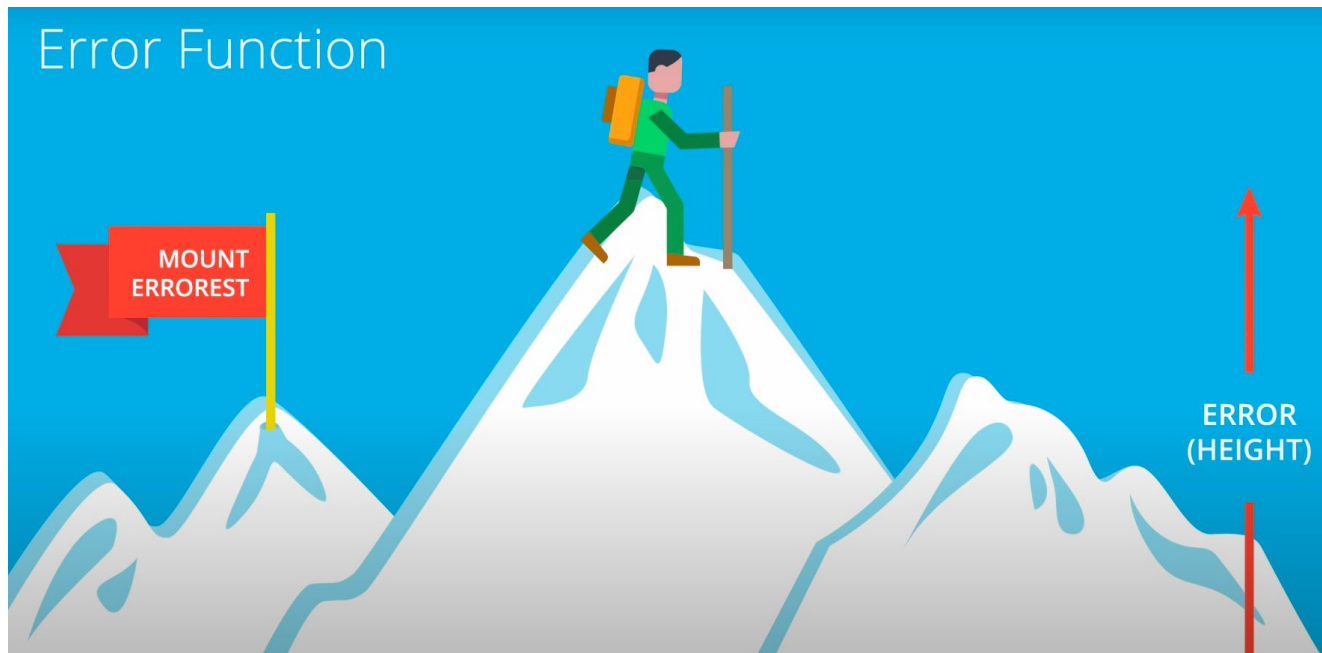
# Practice: tensors in Pytorch

- [Notebook](#)
- [Solutions](#)

## Questions:

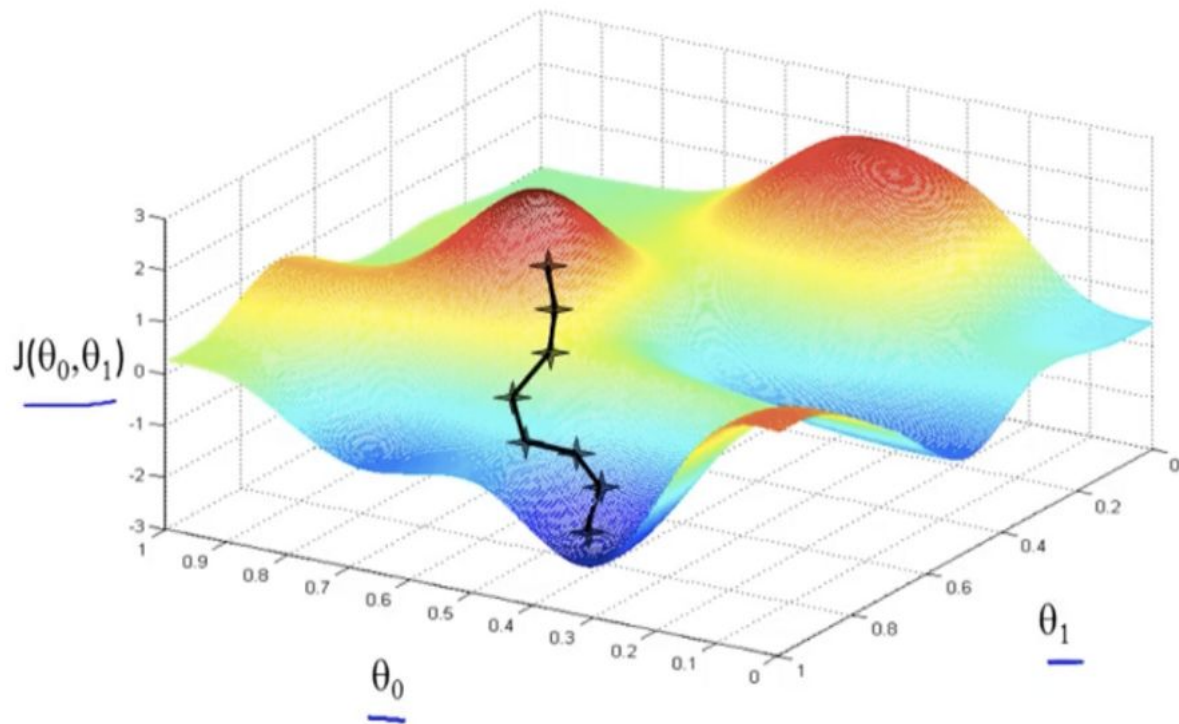
- Why do we use bias terms?
- How is a torch tensor different from a numpy array?
- Is a torch tensor always run on a GPU?
- What is an in-place operation? How are in-place methods named in Pytorch?
- Is the rectifier linear unit a linear or a non-linear function?

# Gradient descent

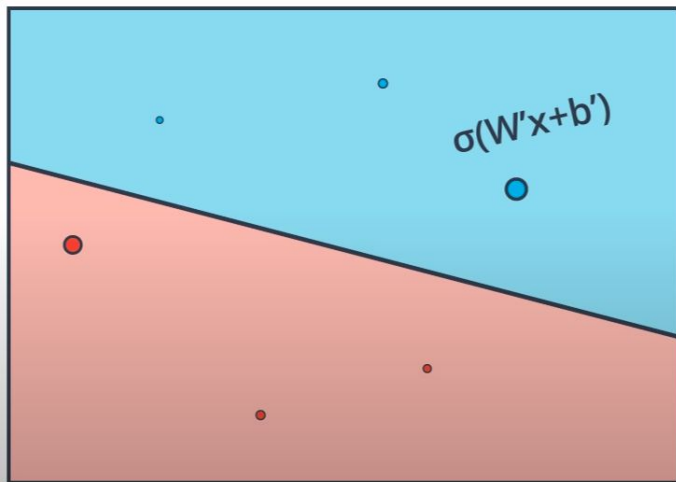


[https://github.com/fastai/fastbook/blob/master/04\\_mnist\\_basics.ipynb](https://github.com/fastai/fastbook/blob/master/04_mnist_basics.ipynb)

# Stochastic gradient descent



# Gradient Descent Algorithm



1. Start with random weights:

$$w_1, \dots, w_n, b$$

2. For every point  $(x_1, \dots, x_n)$ :

2.1. For  $i = 1 \dots n$

2.1.1. Update  $w'_i \leftarrow w_i - \alpha (\hat{y} - y)x_i$

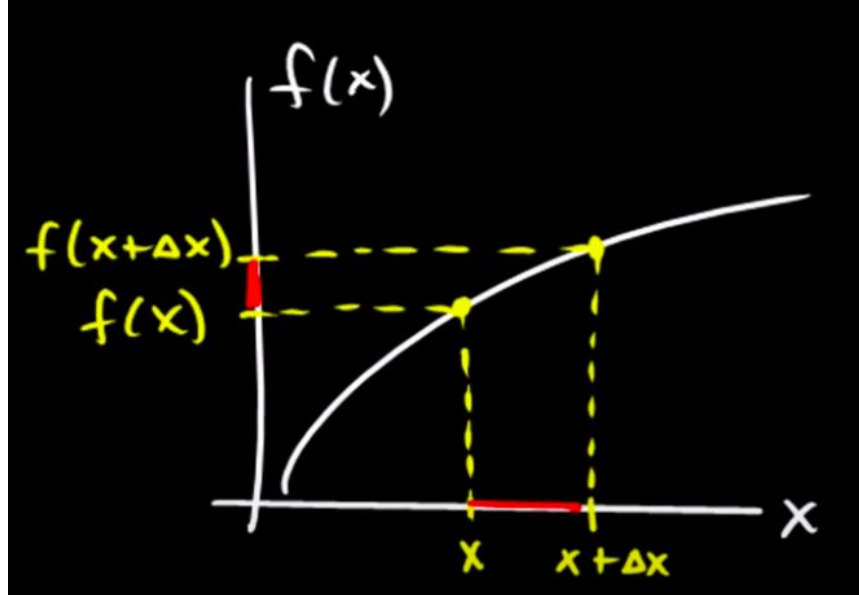
2.1.2. Update  $b' \leftarrow b - \alpha (\hat{y} - y)$

3. Repeat until error is small

[Implementing gradient descent](#)



# Numerical calculation of derivatives



[Numerical derivatives](#)

[Numerical and analytic derivatives in Python](#)

# Gradient computation graphs

Backpropagation: a simple example

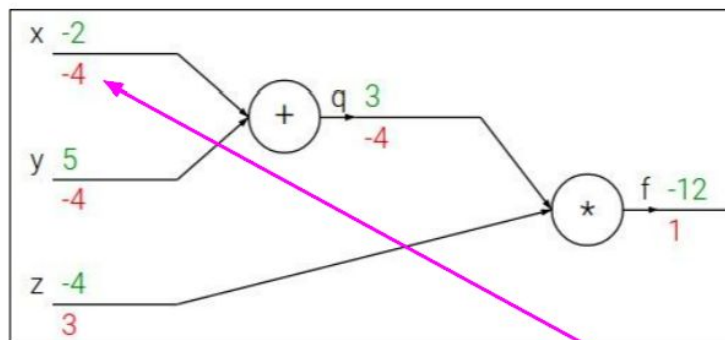
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



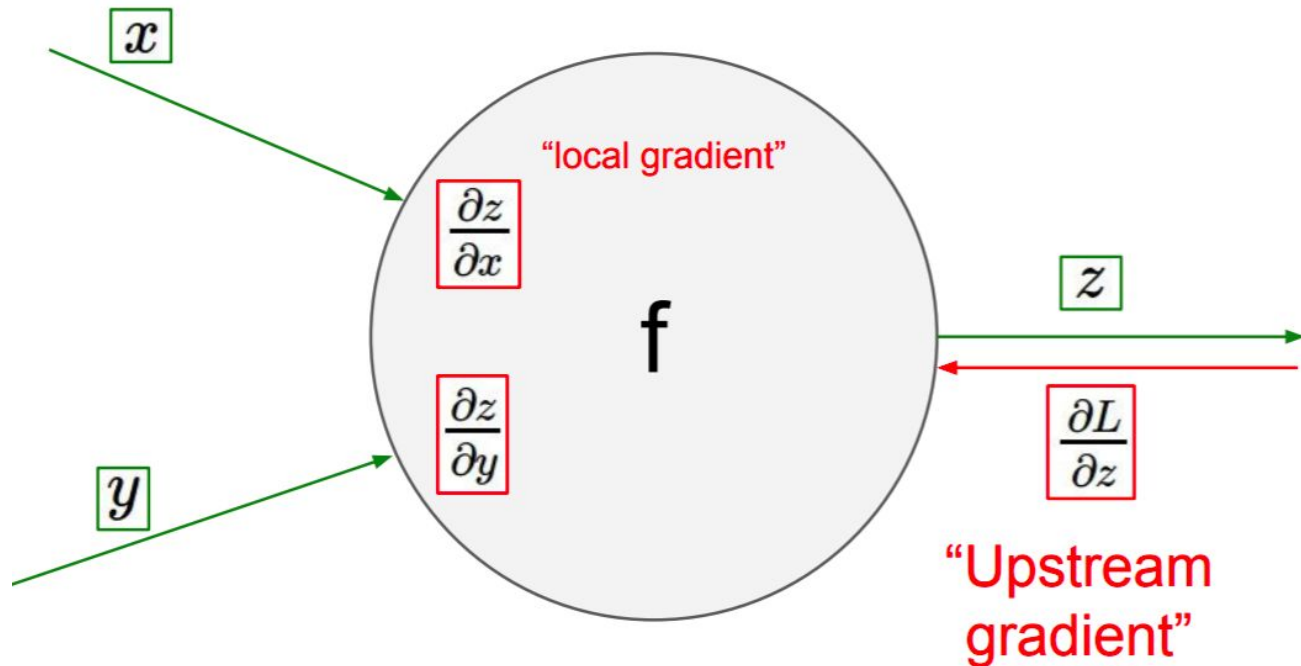
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream  
gradient

Local  
gradient

# Gradient computation graphs



## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If  $x$  changes by a small amount, how much will  $y$  change?

## Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will  $y$  change?

## Vector to Vector

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

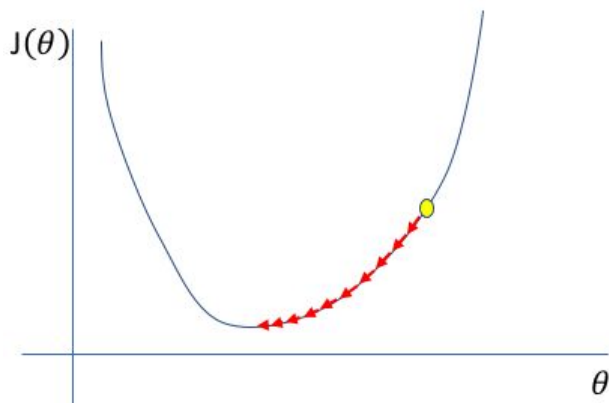
Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will each element of  $y$  change?

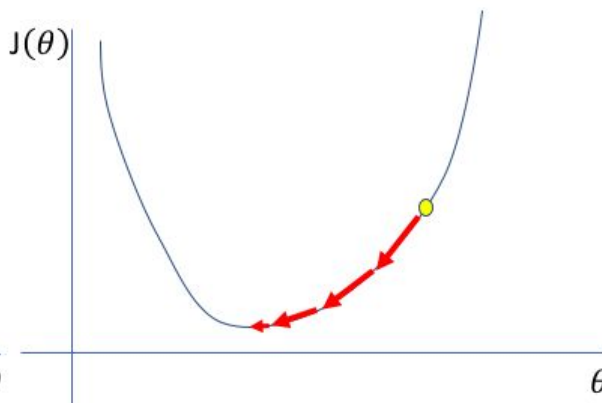
# The importance of the learning rate

Too low



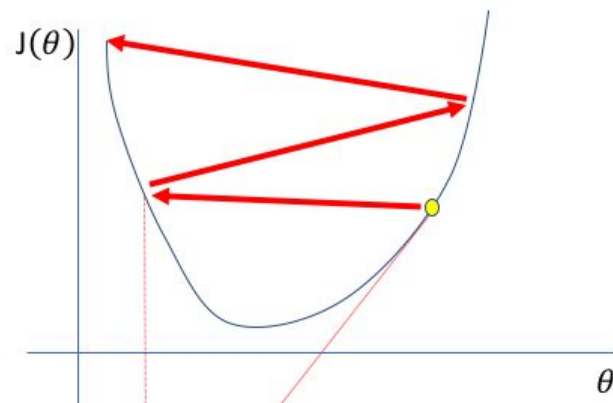
A small learning rate requires many updates before reaching the minimum point

Just right



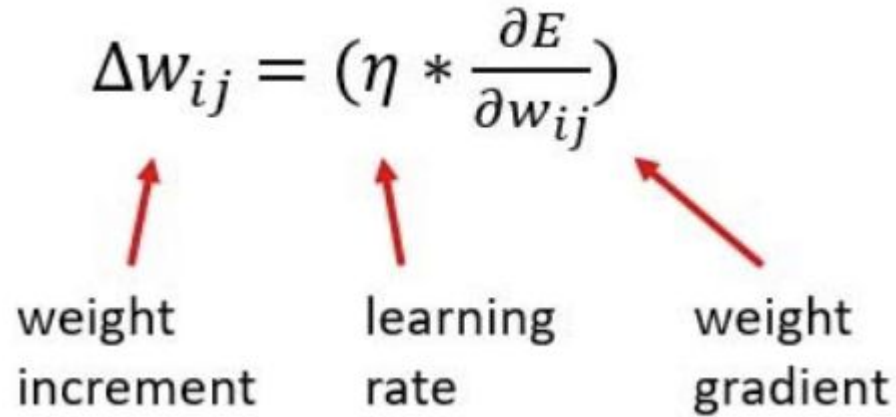
The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

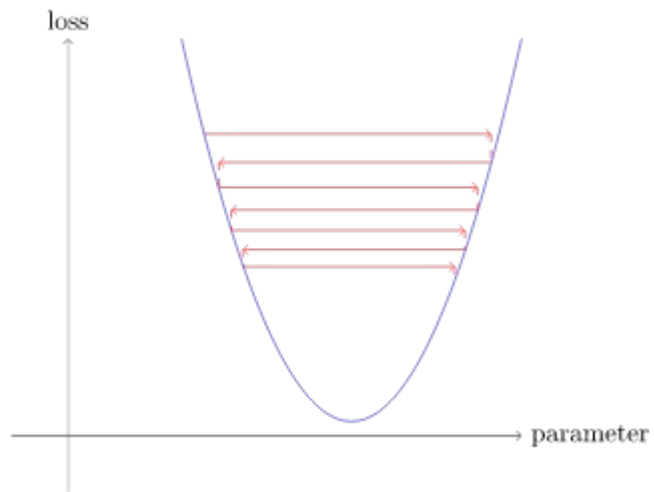
# The importance of the learning rate

$$\Delta w_{ij} = \left( \eta * \frac{\partial E}{\partial w_{ij}} \right)$$


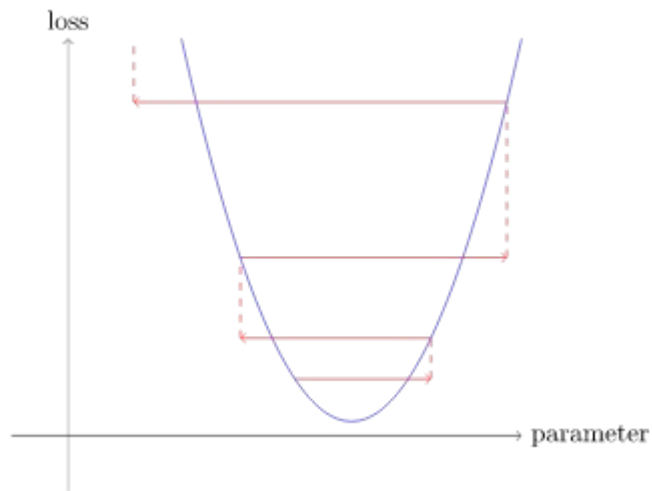
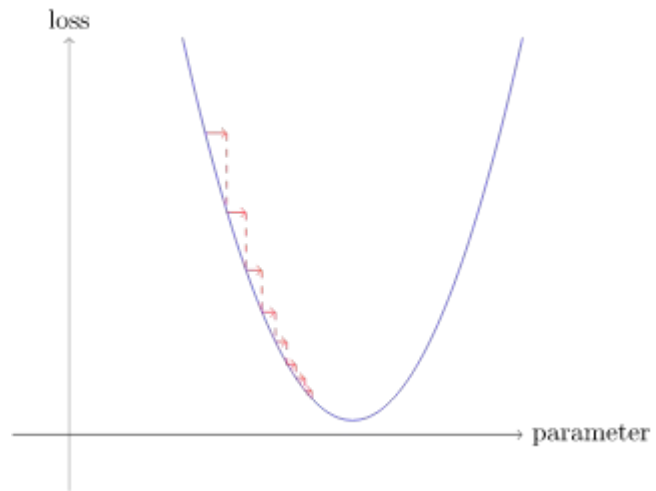
weight increment      learning rate      weight gradient

The diagram illustrates the components of the weight increment equation. Three red arrows point from the labels below to the terms in the equation: one from 'weight increment' to  $\Delta w_{ij}$ , one from 'learning rate' to  $\eta$ , and one from 'weight gradient' to  $\frac{\partial E}{\partial w_{ij}}$ .

# Choosing a learning rate



# Choosing a learning rate





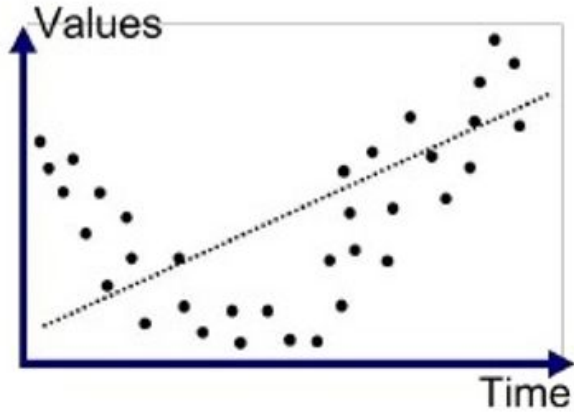
# Weight decay aka l<sub>x</sub>-regularization

$$\mathbf{L1 \text{ ERROR FUNCTION}} = -\frac{1}{m} \sum_{i=1}^m (1 - y_i) \ln(1 - \hat{y}_i) + y_i \ln(\hat{y}_i) + \lambda(|w_1| + \dots + |w_n|)$$

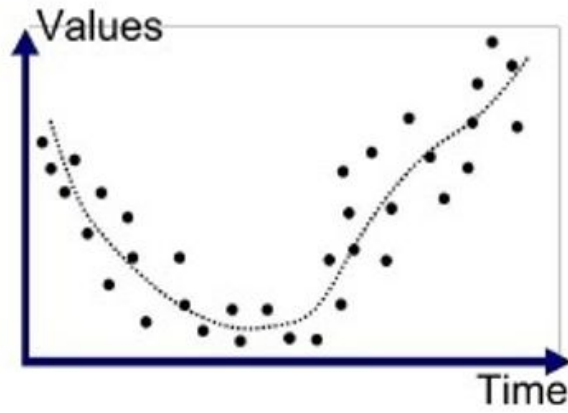
$$\mathbf{L2 \text{ ERROR FUNCTION}} = -\frac{1}{m} \sum_{i=1}^m (1 - y_i) \ln(1 - \hat{y}_i) + y_i \ln(\hat{y}_i) + \lambda(w_1^2 + \dots + w_n^2)$$

[L1 and L2 regularization](#)

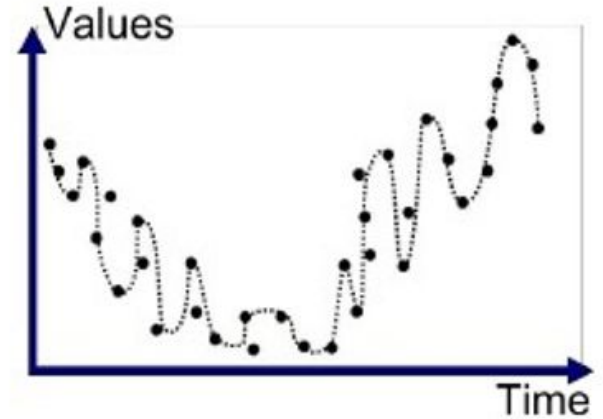
# The bias vs variance tradeoff



Underfitted

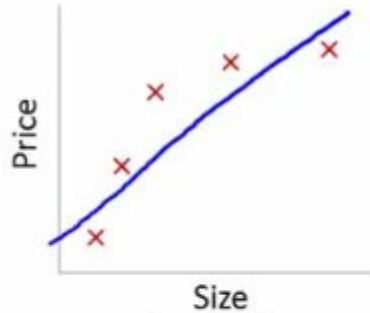


Good Fit/Robust



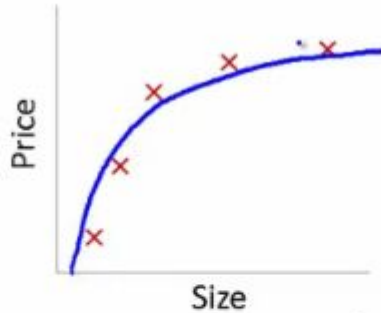
Overfitted

# The bias vs variance tradeoff



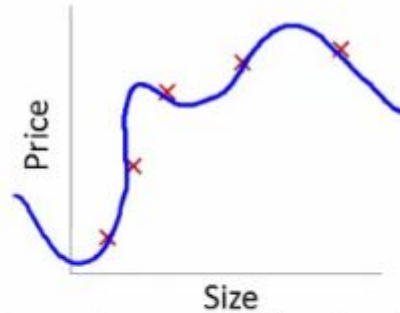
$$\theta_0 + \theta_1 x$$

High bias  
(underfit)



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

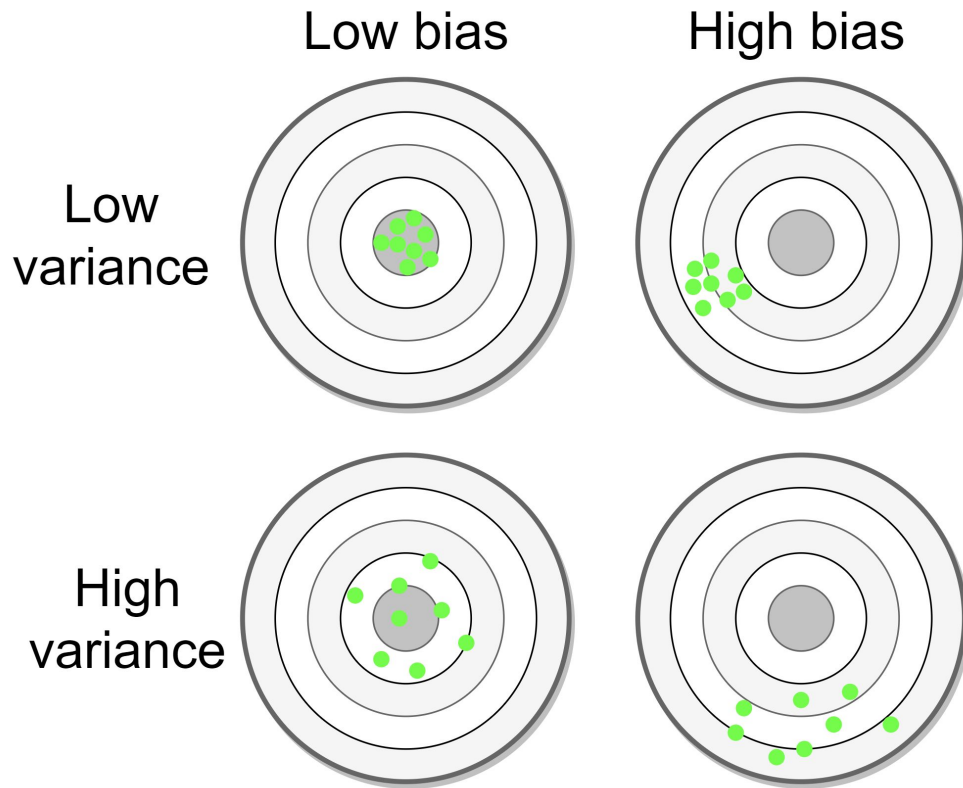
"Just right"



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

High variance  
(overfit)

# The bias vs variance tradeoff



# Practice: neural networks in PyTorch

- [Notebook](#)
- [Solutions](#)

## Questions

- How do we subclass `nn.Module`?
  - What is a subclass?
- How many layers do we need to learn non-linear mappings?
- What is the purpose of a dataloader?
- What is `nn.Linear`?
- What is the purpose of `nn.Sequential`?
- Why would we use `OrderedDict` to define an architecture?

# The Softmax function

The softmax function takes as input a vector  $\mathbf{z}$  of  $K$  real numbers, and normalizes it into a **probability distribution** consisting of  $K$  probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the **interval**  $(0, 1)$ , and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

The standard (unit) softmax function  $\sigma : \mathbb{R}^K \rightarrow \mathbb{R}^K$  is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

[https://en.wikipedia.org/wiki/Softmax\\_function](https://en.wikipedia.org/wiki/Softmax_function)

[The softmax activation function in Python](#)

# Practice: training in PyTorch

- [Notebook](#)
- [Solutions](#)

Questions:

- What is a logit (in the context of neural networks)?
- How do we turn logits into probabilities?
- What criterion should we use if we change `nn.LogSoftmax` to `Softmax`? (Read [this discussion](#))
- Where do we define the size of the batch?

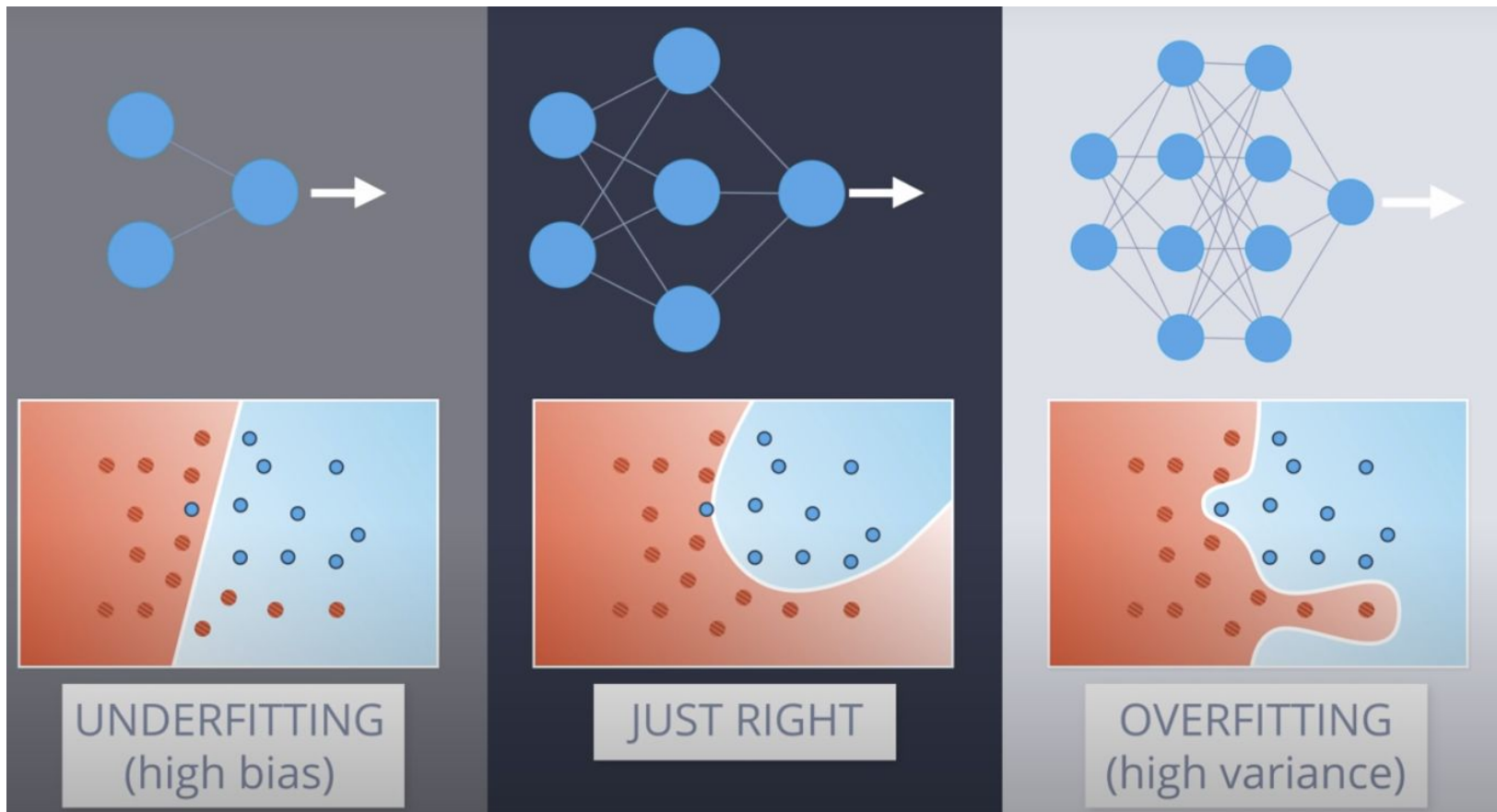
# Performance metrics vs loss functions

A model performance metric measures the quality of predictions in the validation or test sets. Accuracy, precision, recall, specificity, fbeta, f1, and ROC are performance metrics, not loss functions

The default classification error metric in fastai is `error_rate = 1 - accuracy`

The loss function is used to adjust the weights of the model (through its gradient). The error metric is a report of performance and it can be opposite to the loss function

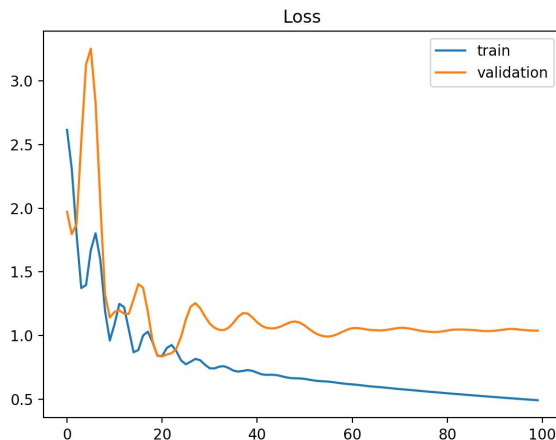




[Understanding the bias vs variance tradeoff in neural network architectures](#)

# How long should we train the network?

- Until it overfits! (Training loss goes down while validation loss goes up)
- This general principle applies to **all** hyperparameters related to model complexity



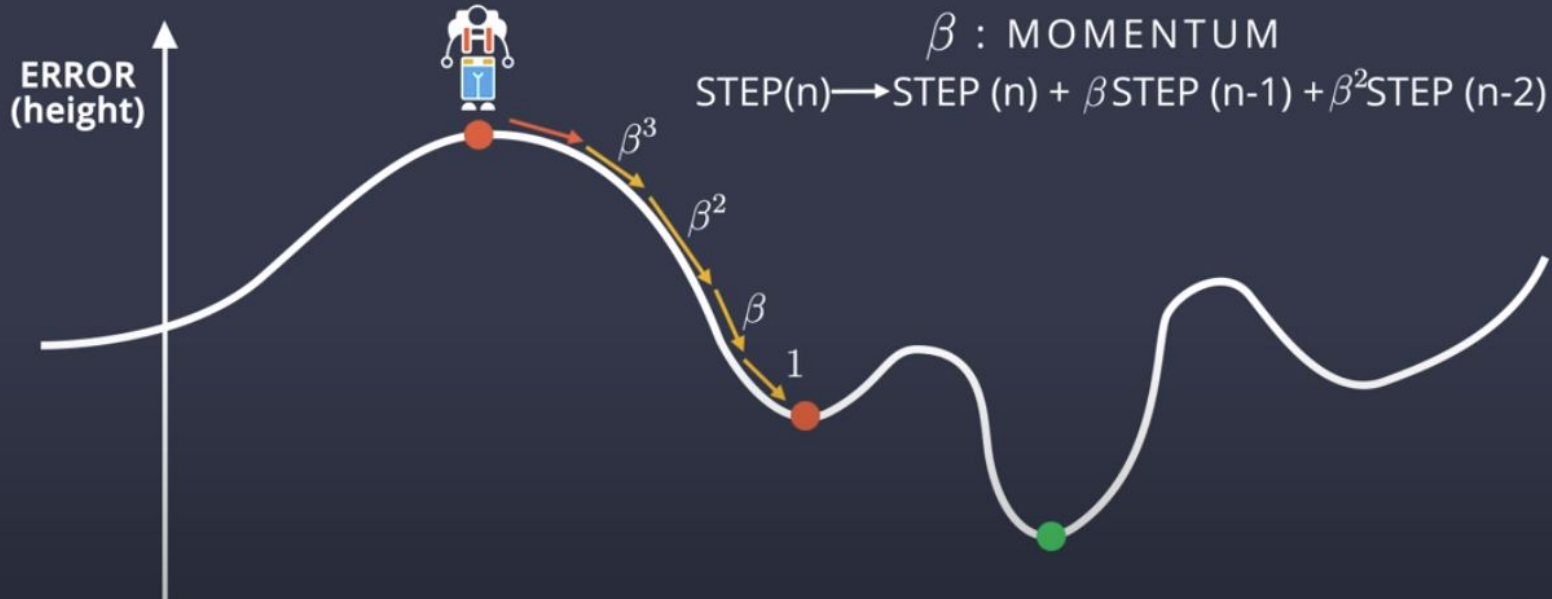
# GRADIENT DESCENT

IDEA: MOMENTUM

STEP  $\rightarrow$  AVERAGE OF PREVIOUS STEPS

$\beta$  : MOMENTUM

STEP(n)  $\rightarrow$  STEP (n) +  $\beta$  STEP (n-1) +  $\beta^2$  STEP (n-2) + ...




[The intuition of momentum](#)


[Exploring momentum's beta \(notebook\)](#)

# Understanding momentum

$$\Delta w_{ij} = \left( \eta * \frac{\partial E}{\partial w_{ij}} \right) + \left( \gamma * \Delta w_{ij}^{t-1} \right)$$

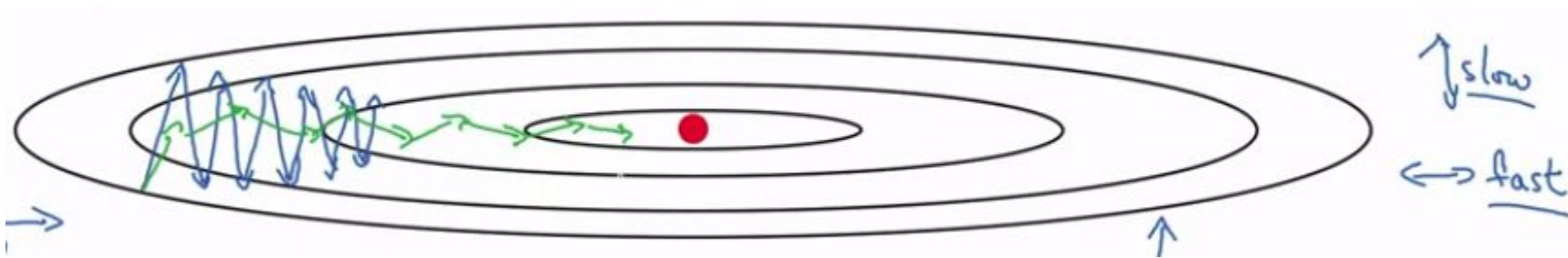


momentum  
factor



weight increment,  
previous iteration

# The ADAM optimizer



Exponentially [moving averages](#) and used to smoothen the trajectory of the gradient descent. Averages are computed across training batches. This optimization algorithm usually trains faster than SGD

# Practice: classifying images with fully connected networks

- [Notebook](#)
- [Solutions](#)

Questions:

- How is the image fed into the network?
- Why shouldn't we use the network's accuracy as a Pytorch **criterion**?
- What is the optimizer?
- How does ADAM differ from Stochastic Gradient Descent? How are they similar?
- What is the **lr** parameter that we pass to the optimizer?
- Why do we pass `model.parameters()` to the optimizer?

# Review questions

- What is a loss function? Is the f1 measure a loss function or a performance metric?
- Explain what the ReLU activation function does in one sentence.
- Can we use accuracy as a loss function? Can we use precision or recall as loss functions?
- Is the accuracy guaranteed to increase as the value of the loss function goes down?
- Which parts of the model need to be differentiable for an architecture to be trainable?
- What are we trying to minimize when using gradient descent? What is the learning rate?
- What is a GPU? Why are GPUs useful for deep learning?

# Review questions

- What is an epoch? What is a batch? What is the maximum size of a batch? How does the batch size influence training?
- What is the difference between stochastic gradient descent (SGD) and 'vanilla' AKA 'regular' gradient descent?
- What is momentum? What is the difference between ADAM and SGD?
- What is a Jacobian? What is a Hessian? Do we use Hessians in deep learning?
- Suppose that we have a ground truth for a 'cat' label that is encoded as  $[0, 1]$ . We can assume that the ground truth for a 'dog' label is  $[1, 0]$  (**cat** is the training point).
  - The network first outputs  $[0.75, 0.25]$  as prediction
  - In the next epoch it outputs  $[0.99, 0.01]$  as prediction
  - Has the cross entropy loss increased or decreased?



# Review questions

- What does it mean for a dataset to be ‘linearly separable’?
- How many layers does a neural network need in order to learn non-linear correlations?
- What’s the difference between a performance metric and a loss function?
- Why should we shuffle the data in every training epoch?
- Explain the bias vs variance tradeoff.
  - Effect of increasing the number of training epochs
  - Increasing the number of layers
  - Increasing the amount of dropped out connections
- How would you define ‘regularization’?
- What is the purpose of dropout? How does it work? Should dropout be active during inference time?

# Review questions

- What is the vanishing gradient problem?
- What does it mean to “transfer a tensor” to the GPU?
- What is CUDA?
- How does the amount of RAM available in the GPU affect the types of models and data that we can use to train?
- What should we try when torch reports ‘CUDA out of memory’ errors?

# Review questions

- Why do we need to call `optimizer.zero_grad()` in the training loop?
- What's the difference between a parameter and a hyperparameter? What's another common name for 'parameter' in a neural network?
- How can we know what's the “optimal” number of units in a hidden layer?
- How can we know the “optimal” number of layers in a network?
- Is data augmentation a regularization technique? Why should we apply data augmentations probabilistically?
- Why should we shuffle the data inside a training batch?
- If a model is increasing its loss through several epochs should we increase or decrease the learning rate?
- How do we evaluate training and validation loss to do early stopping?

# Review questions

- What is weight decay? What is learning rate decay?
- Suppose that after going through certain number of epochs the training loss is higher than the validation loss. Is this an example of underfitting or overfitting?
- What is the vanishing gradient problem? Why does it appear?
- Are vanishing gradients more likely to appear when representing Jacobians and weight vectors with 8 bits, 16 bits, or 32 bits of precision?
- Why are ReLU activation functions less likely to display vanishing gradients than sigmoid activation functions?

# Review questions

- Why do we resize images before training?
- If we retrain or do inference on a network pre-trained on Imagenet, why do we need to normalize its image channels using the mean and std of the intensities of the original dataset?
- What is a receptive field?
- What is a convolution?
- Why do we apply padding to images before doing convolutions?
- When calling `conv2d(..., ..., 64)` how many output channels are we producing? How are they different from each other?
- Why is the vanishing gradient a 'numerical' problem? What does that mean?
- Are we more likely to get vanishing gradients when using fp16 or fp32 data types for our weights?

# Resources

- [Intro to Deep Learning with PyTorch at Udacity](#)
- [Convolutional Neural Networks at Coursera](#)
- [Fast.ai - Deep Learning for Coders](#)
- [PyImageSearch Blog](#)
- [Stanford's CS231: Convolutional Neural Networks for Visual Recognition](#)
- [3blue1brown's series on neural networks](#)
- [Goodfellow's Deep Learning Book \(great for theory\)](#)
- [Visualizing and understanding neural networks \(Stanford lecture\)](#)
- Review content on cross entropy
  - [On ML Mastery](#)
  - [Brief video by Udacity](#)
  - [Aurelien Geron's explanation](#)

# Regression with a neural network

<https://utstat.toronto.edu/reid/sta2212s/2021/Galton85.pdf>

[https://www.researchgate.net/publication/280970132\\_Galton's\\_Family\\_Heights\\_Data\\_Revisited](https://www.researchgate.net/publication/280970132_Galton's_Family_Heights_Data_Revisited)