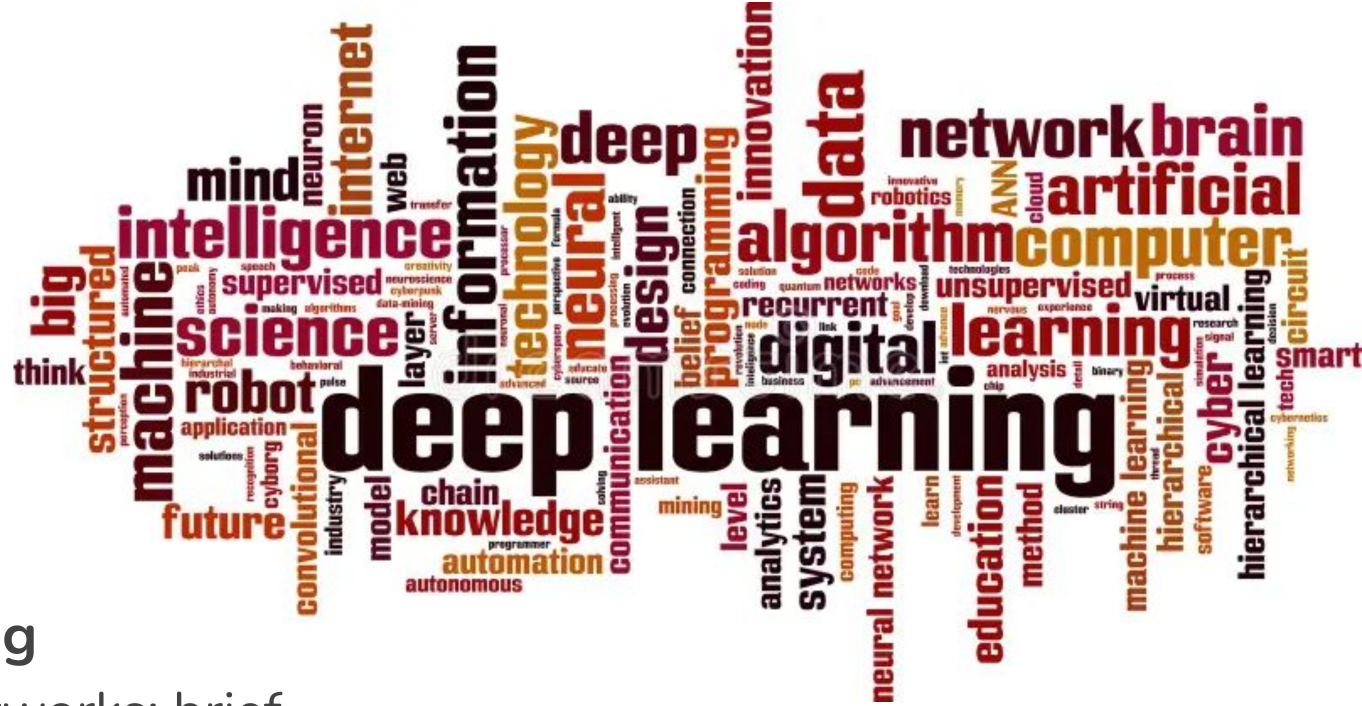
Big data science Day 4

F. Legger - INFN Torino

https://github.com/Course-bigDataAndML/MLCourse-2324

We learned

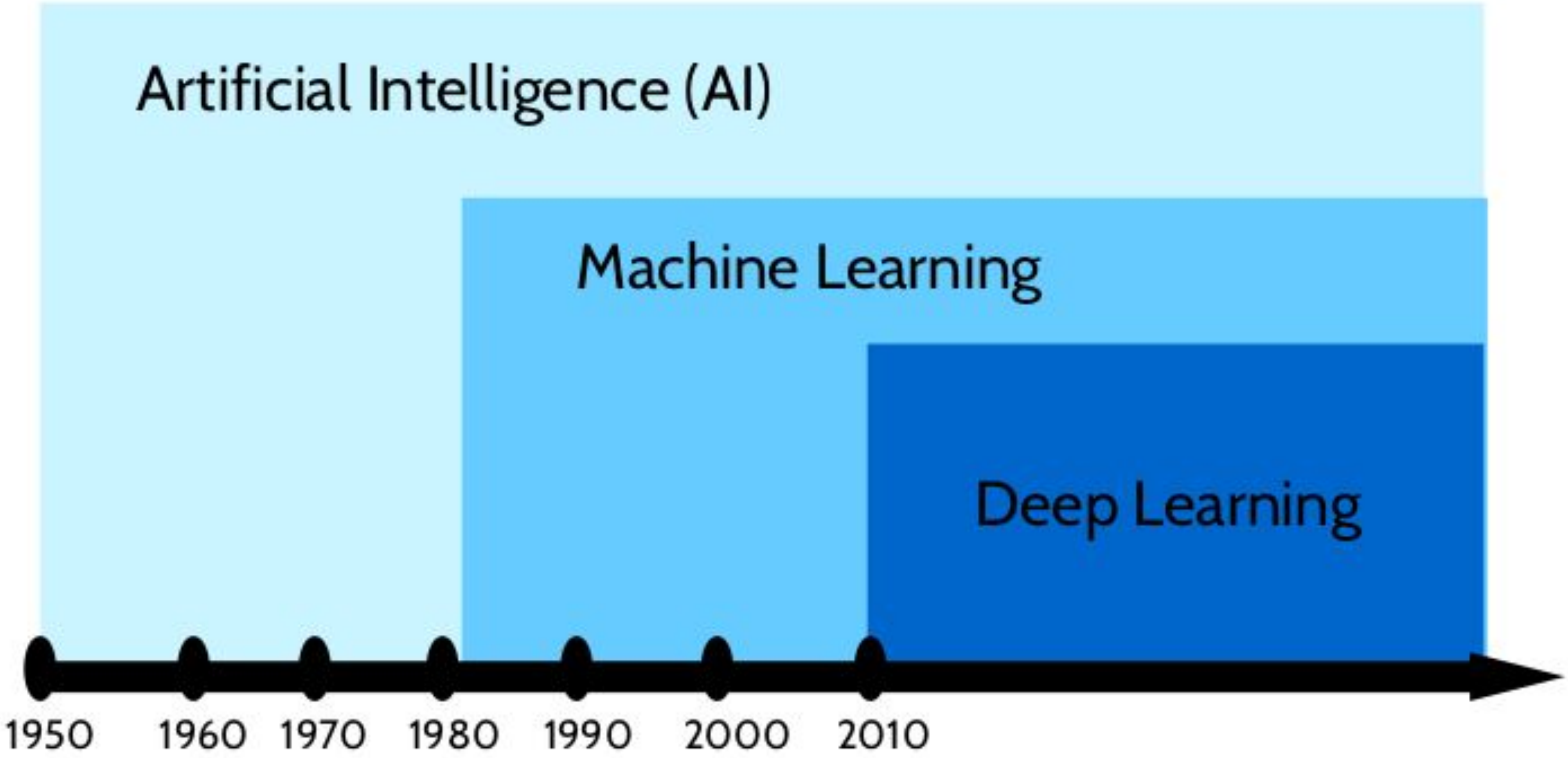
- Big data
- Analytics
- ML



Today

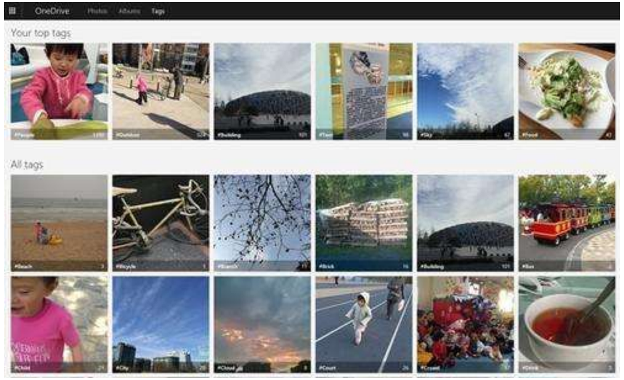
- **Deep learning**
  - Neural networks: brief history, main architectures and how to train them

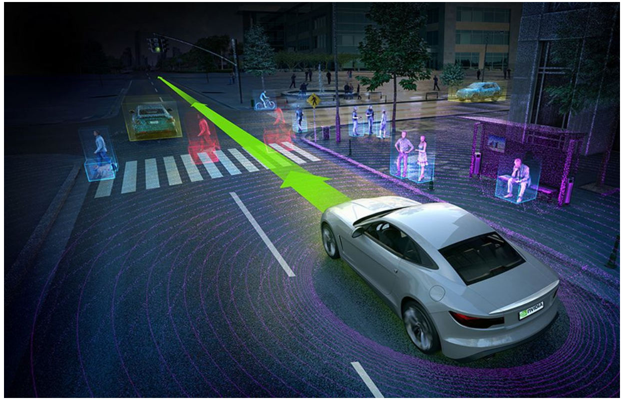
Deep Learning is a subfield of ML concerned with algorithms inspired by the structure and function of the brain called artificial neural networks [Jason Brownlee]



Machine translation
Real-time translation into Mandarin Chinese (2012)

Visual recognition, CNN better than humans (2012)





Self driving cars, Tesla autopilot (2014)

Chatbots, CHAT GPT (2022)

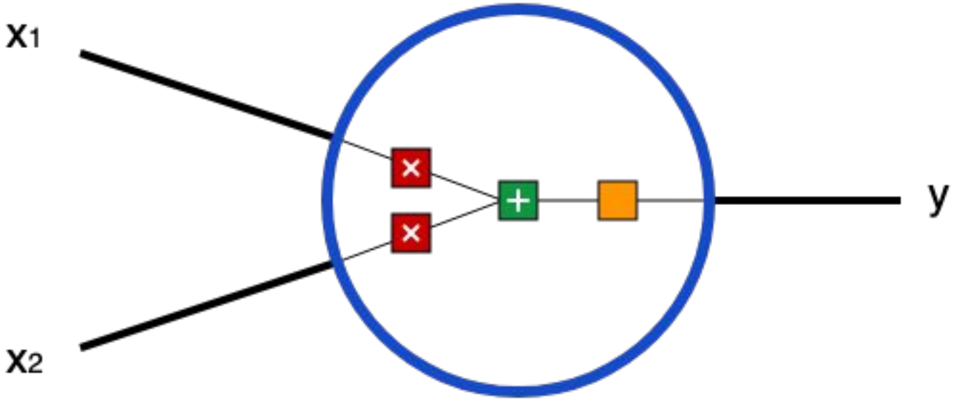
Creativity
NST (2015), GAN (2014)

Strategy games DeepMind beats Go world champion (2017)

And the winner is…

- The Nobel Prize in **Physics 2024** was awarded jointly to **John J. Hopfield** (Princeton)and **Geoffrey E. Hinton** (U. Toronto) *"for foundational discoveries and inventions that enable machine learning with artificial neural networks"*
- The Nobel Prize in **Chemistry 2024** was divided, one half awarded to **David Baker** (U. Washington) *"for computational protein design"*, the other half jointly to **Demis Hassabis** (Google DeepMind) and **John Jumper** (Google DeepMind)*"for protein structure prediction"*

Short recap: Neuron



- each **input x** is multiplied by a **weight w**

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

X

- all the weighted inputs are added together with a **bias *b***

$$(x_1 * w_1) + (x_2 * w_2) + b$$

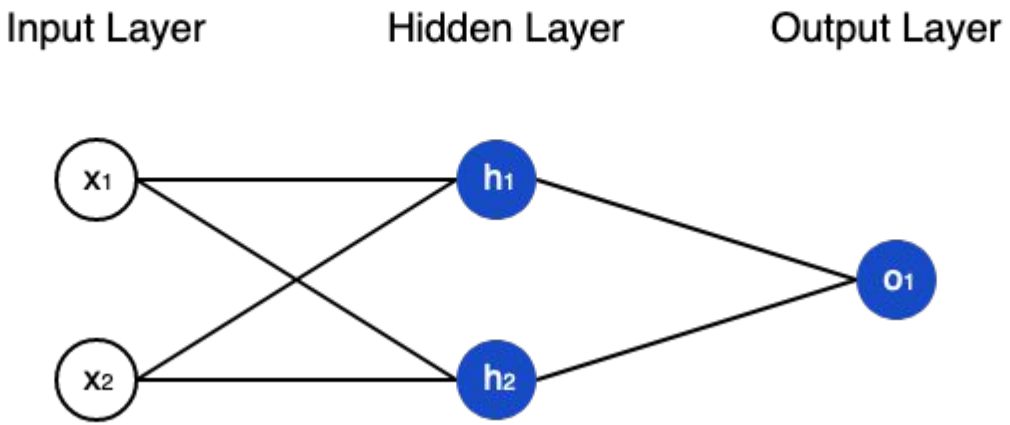
+

- the sum is passed through an **activation function f**

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

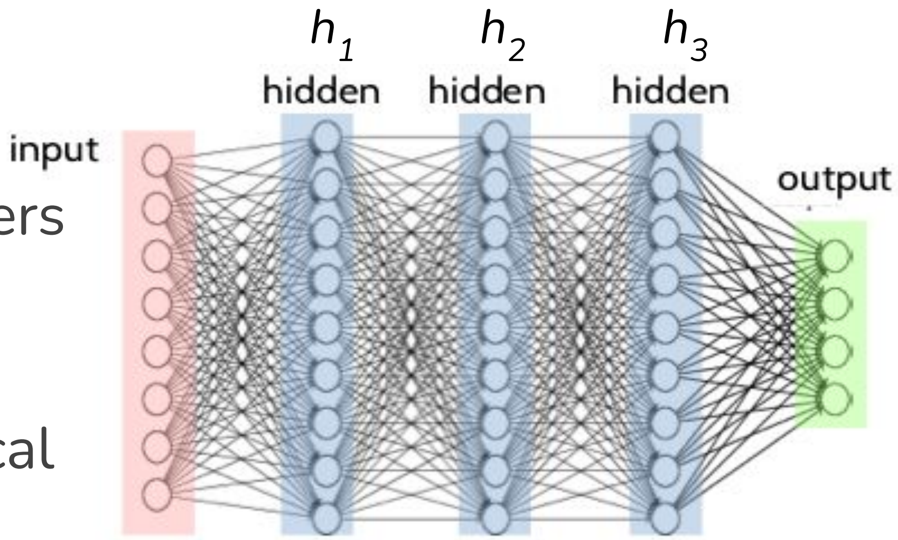
Neural network

- Combining more neurons
- A **hidden layer** is any layer between the input (first) layer and output (last) layer
    - There can be multiple hidden layers
- **Feedforward**: process of passing inputs forward to get an output



This network has:
- one **input** layer with 2 inputs
- one **hidden** layer with 2 neurons
- one **output** layer with 1 neuron

Deep Learning



- Neural network with several layers
  - Deep vs shallow
- A family of parametric models which learn non-linear hierarchical representations:

$$a_L(\mathbf{x};\mathbf{\Theta}) = h_L(h_{L-1}(...(h_1(\mathbf{x},\boldsymbol{\theta}_1),\boldsymbol{\theta}_{L-1}),\boldsymbol{\theta}_L)$$

input

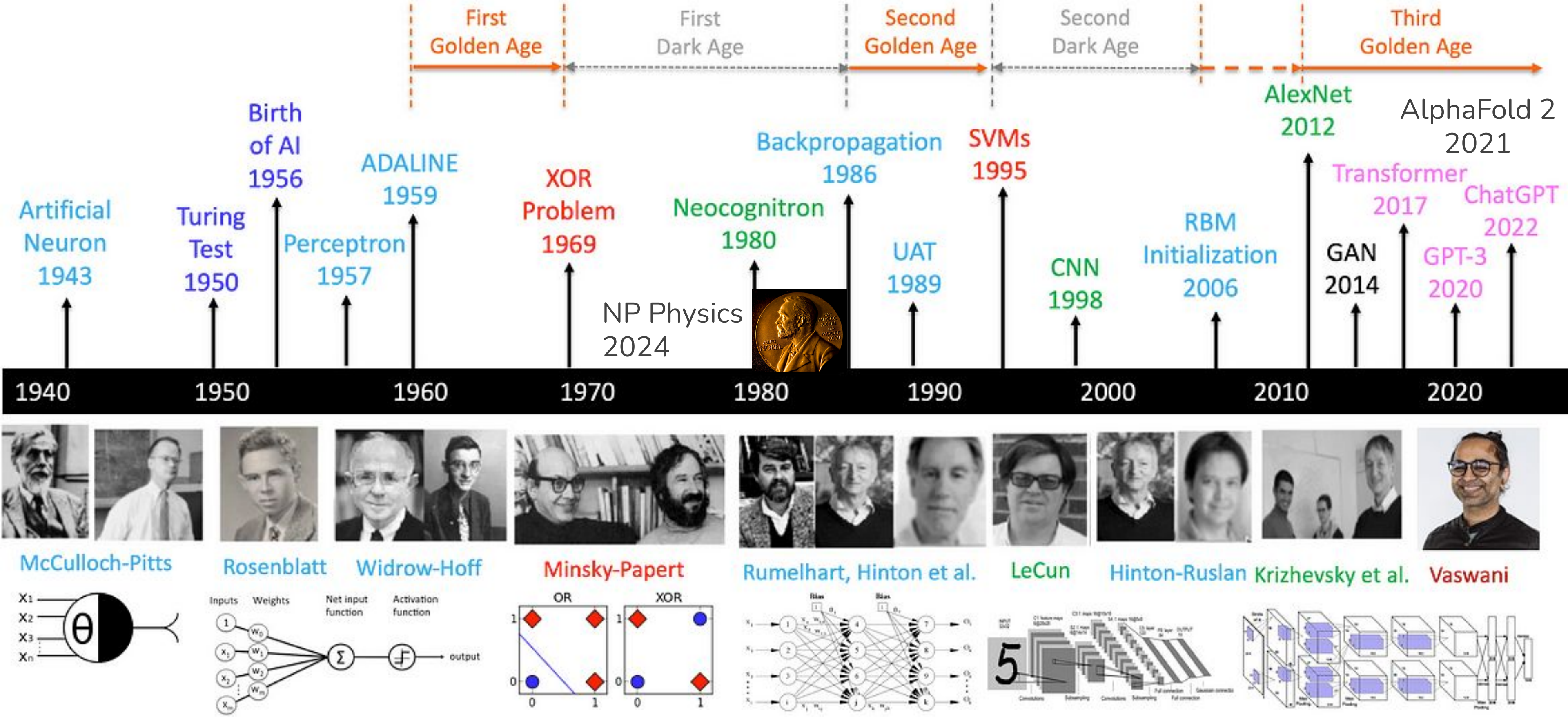
parameters of the network

non-linear activation function

parameters of layer *L*

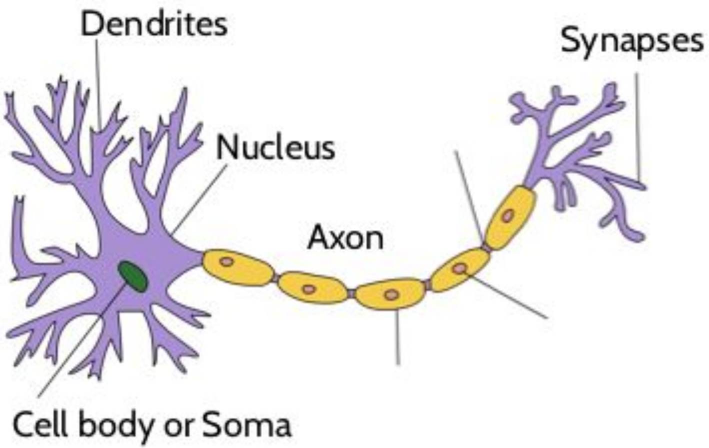
A Brief History of AI with Deep Learning

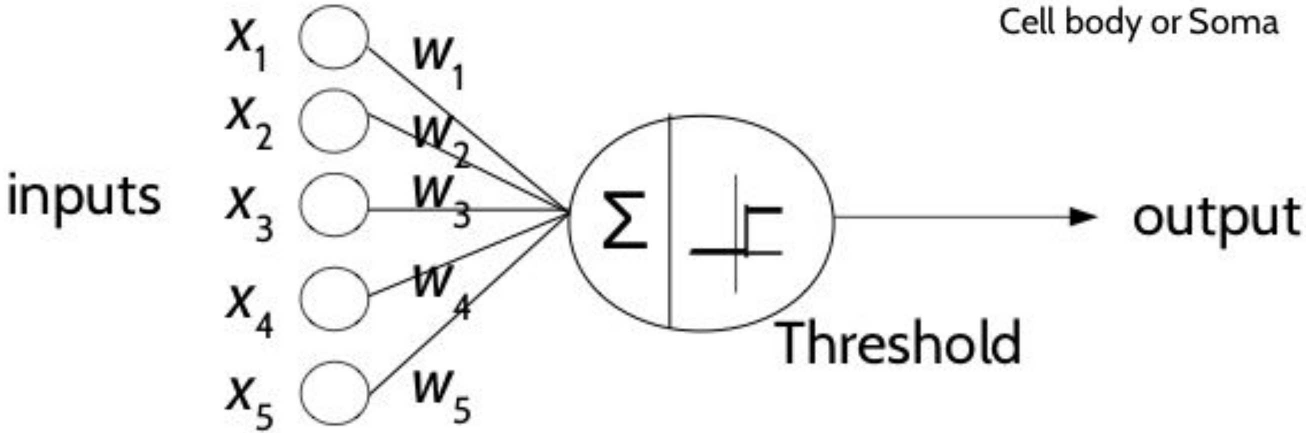
NP Chemistry 2024



1943 – McCulloch & Pitts Model

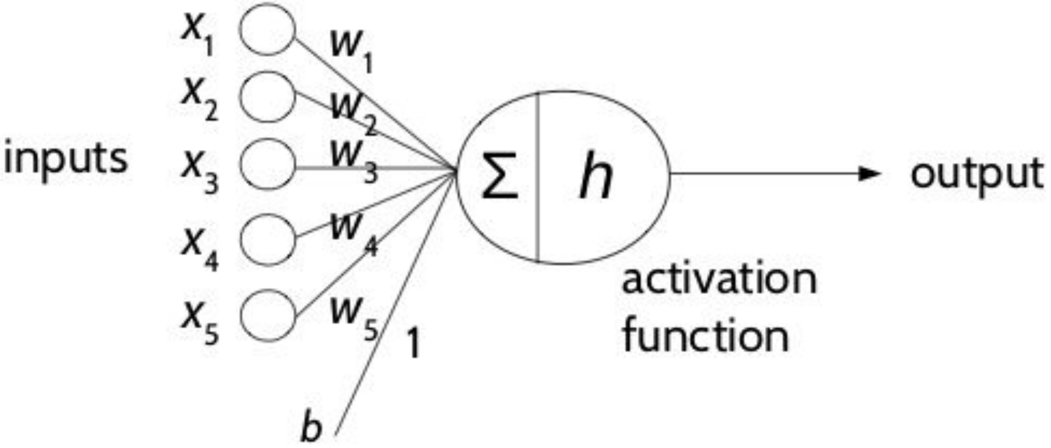
- Early model of artificial neuron
- Generates a binary output
- The weights values are fixed

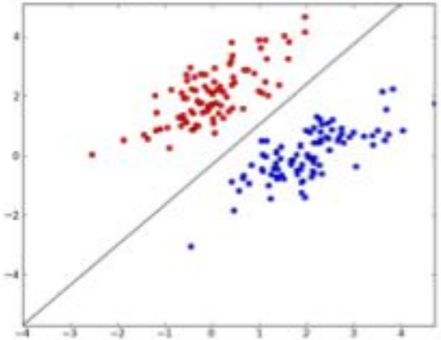




1958 – Perceptron by Rosemblatt

- Perceptron as a machine for linear classification
- Main idea: Learn the weights and consider bias.
  - One weight per input
  - Multiply weights with respective inputs and add bias
  - If result larger than **threshold** return 1, otherwise 0



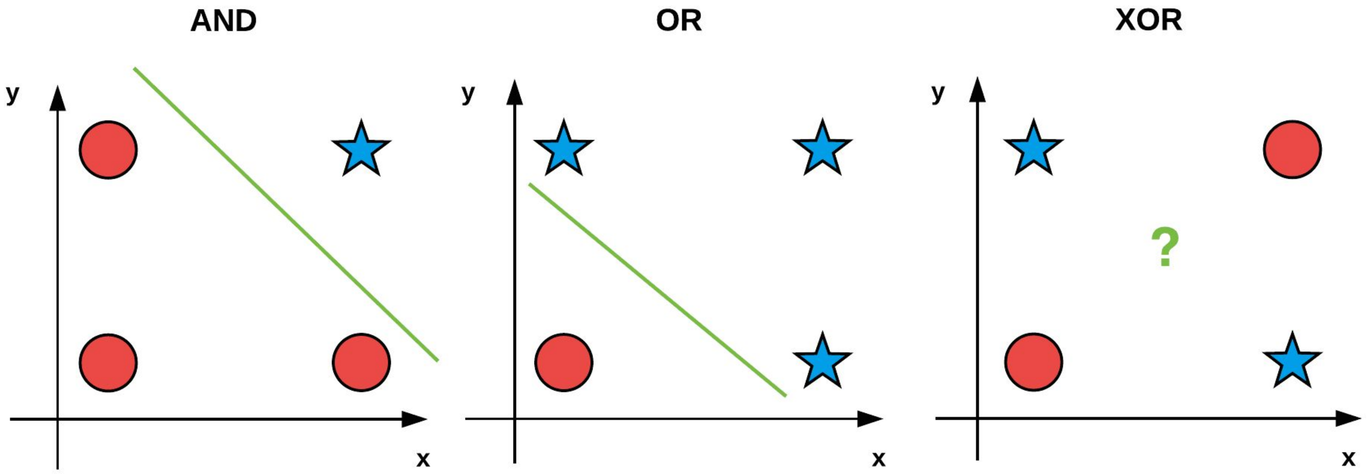


First NN winter

- Single-layer perceptrons are only capable of learning linearly separable patterns.

- 1970- Minsky. The XOR cannot be solved by perceptrons.

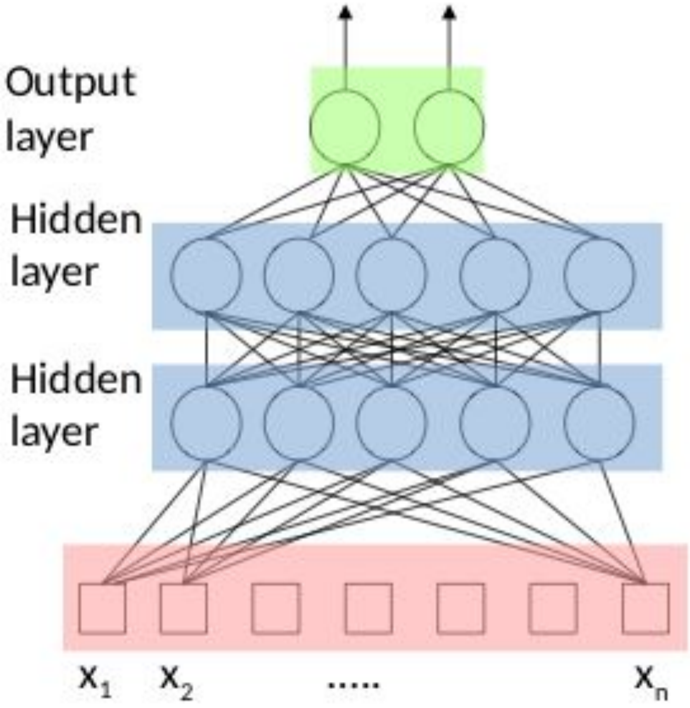
- Neural models cannot be applied to complex tasks.



Multi-layer Feed Forward Neural Network

- 1980s. Multi-layer Perceptrons (MLP) can solve XOR.

- ML Feed Forward Neural Networks:
  - Densely connect artificial neurons to realize compositions of non-linear functions
  - The information is propagated from the inputs to the outputs
  - The input data are usually *n*-dimensional feature vectors
  - Tasks: Classification, Regression



How to train it?

- Rosenblatt algorithm* not applicable, as it expects to know the desired target
  - For hidden layers we cannot know the desired target
- Learning MLP for complicated functions can be solved with **Back propagation** (1986)**
  - efficient algorithm for complex NN which processes large training sets

* Remember? Rosenblatt developed a method to train a single neuron

** later today

The Universal Approximation Theorem (UAT) - 1989

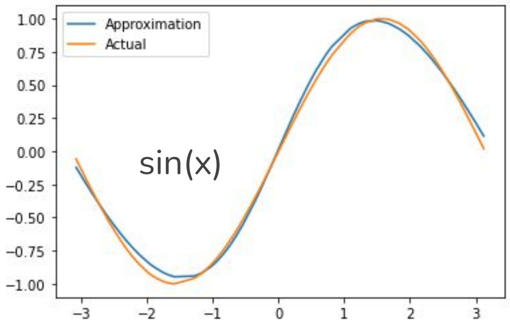
- A multilayered neural network with a single hidden layer can approximate any continuous function to any desired precision

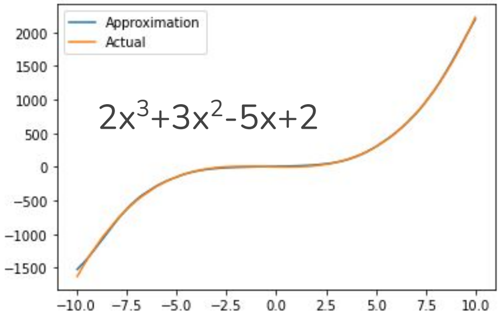
Theorem 1. *Let σ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \qquad (2)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \textit{for all} \quad x \in I_n.$$



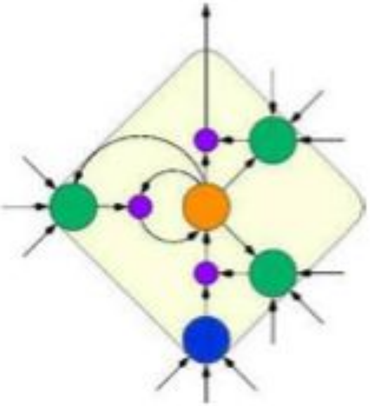


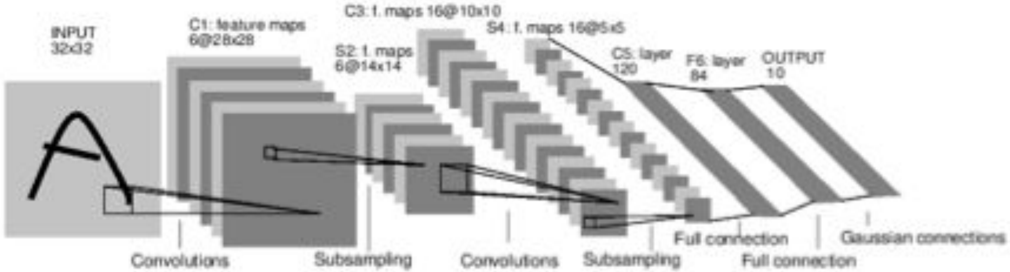
Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals, and Systems*. **2** (4): 303–314

Hornik, Kurt; Stinchcombe, Maxwell; White, Halbert (January 1989). "Multilayer feedforward networks are universal approximators". *Neural Networks*. **2** (5): 359–366

1990s - CNN and LSTM

- Important advances in the field:
  - Backpropagation
  - Recurrent Long-Short Term Memory Networks (Schmidhuber, 1997)
  - Convolutional Neural Networks - LeNet: OCR solved before 2000s (LeCun, 1998).

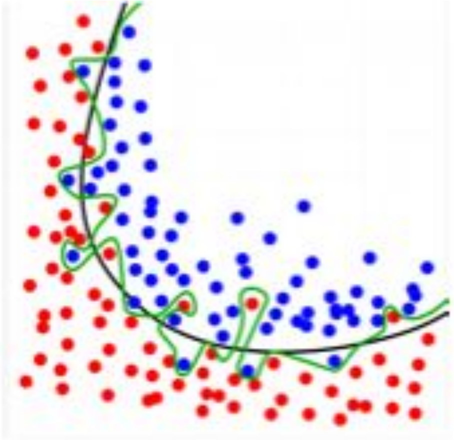


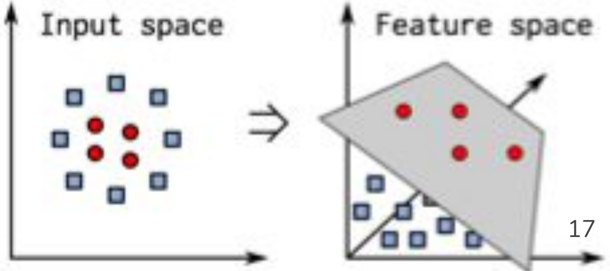


OCR: Optical character recognition

Second NN Winter

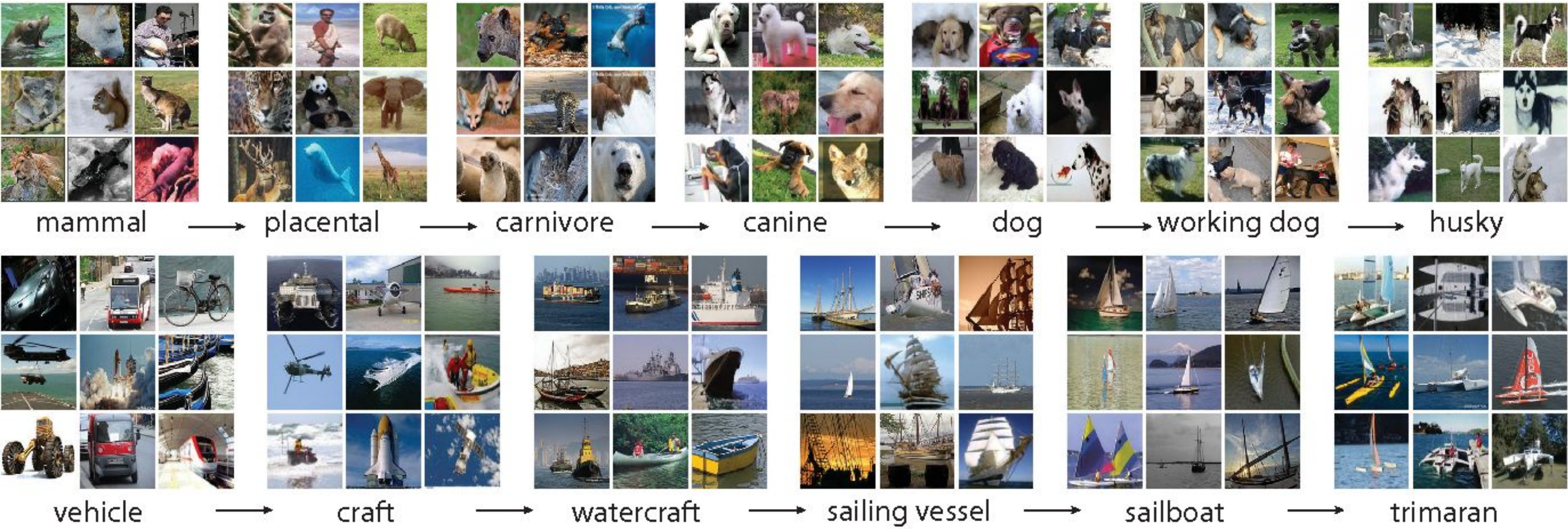
- NN cannot exploit many layers
  - Overfitting
  - Vanishing gradient (with NN training you need to multiply several small numbers → they become smaller and smaller)
- Lack of processing power (no GPUs)
- Lack of data (no large annotated datasets)
- Kernel Machines (e.g. SVMs) suddenly become very popular





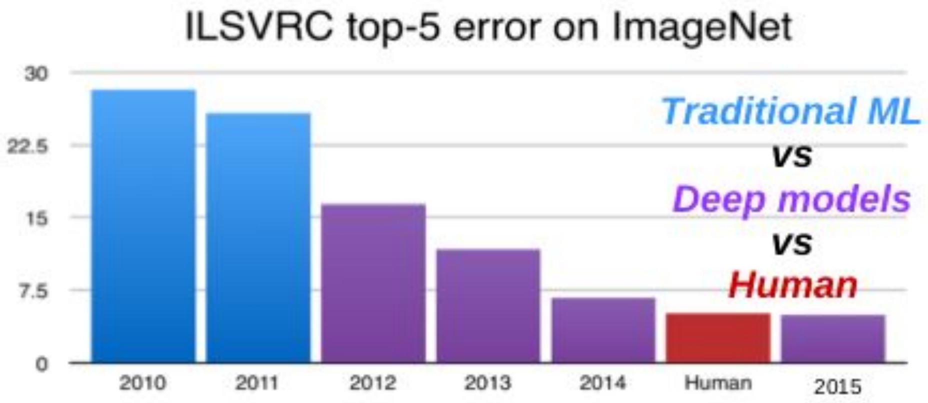
ImageNet - 2009

A Large-Scale Hierarchical Image Database



AlexNet - 2012

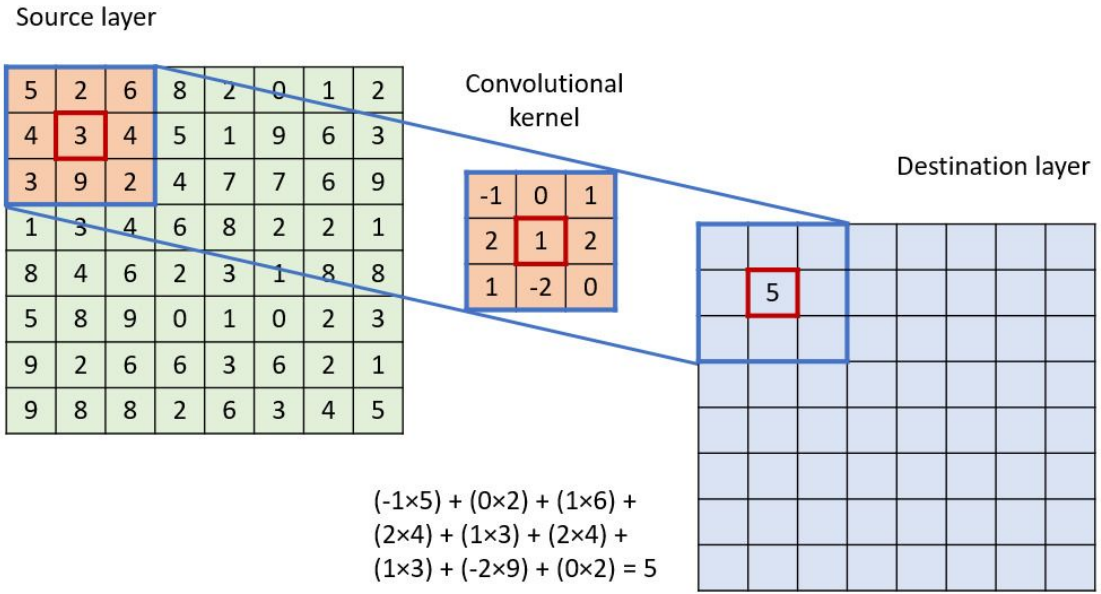
- Hinton's group implemented a CNN similar to LeNet [LeCun1998] but…
  - Trained on ImageNet (1.4M images, 1K categories)
  - With 2 GPUs
  - Other technical improvements (ReLU, dropout, data augmentation)

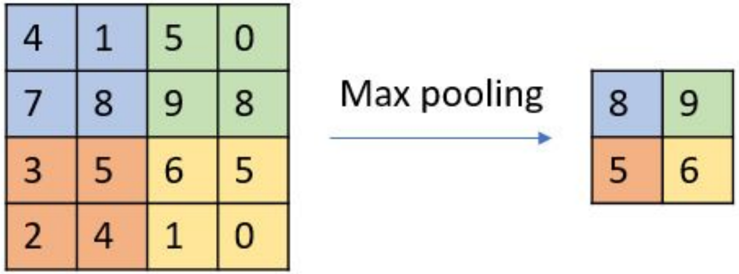


https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

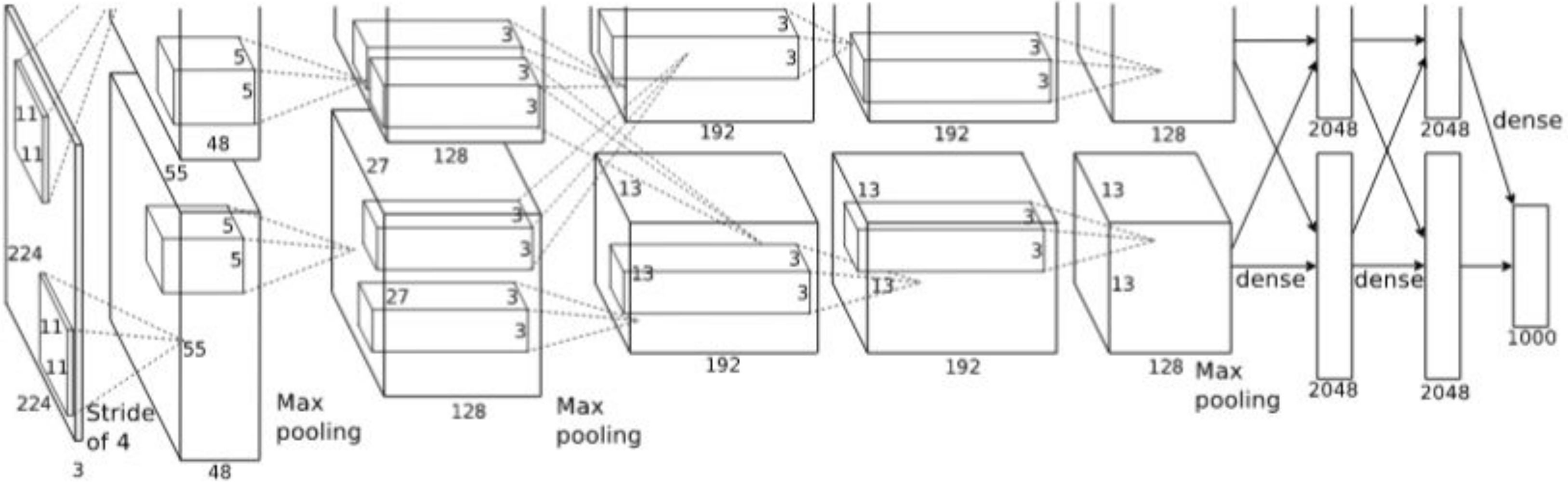
Convolutional Neural Networks (CNN)

- **Convolutional** layer: two functions produce a third that describes how the shape of one is changed by the other
- **pooling** layer: reduce dimensionality





AlexNet



GPU1

GPU2

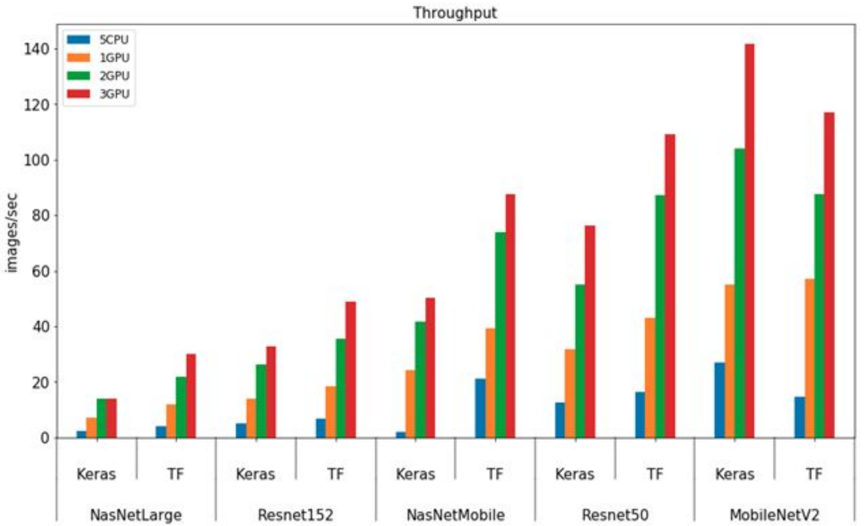
- 60M parameters
- Limited information exchange between GPUs

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Why Deep Learning now?

- Three main factors:
  - Better hardware
  - Big data
  - Technical advances:
    - Layer-wise pretraining
    - Optimization (e.g. Adam, batch normalization)
    - Regularization (e.g. dropout)

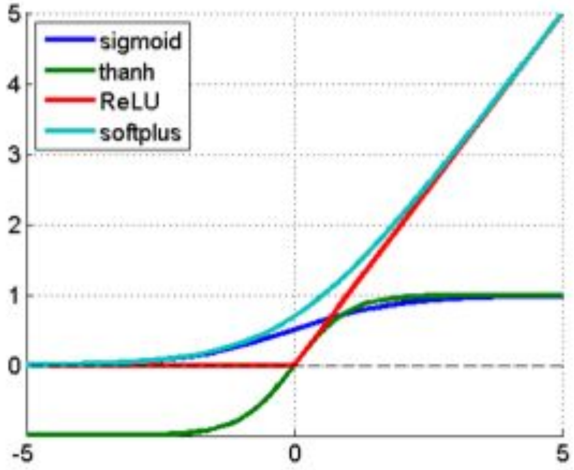
    ….



Rectified Linear Units (ReLU) - 2010

Activation function:

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



- More efficient gradient propagation:
  (derivative is 0 or constant)

- More efficient computation:
  (only comparison, addition and multiplication).

- Sparse activation: e.g. in a randomly initialized networks, only about 50% of hidden units are activated (having a non-zero output)

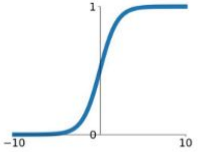
https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

Activation functions

- Classification: sigmoid functions
  - sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- **ReLU** function is a general activation function
  - dead neurons in our networks -> the leaky ReLU
  - ReLU function should only be used in the hidden layers
  - As a rule of thumb, start with ReLU

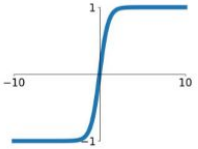
Sigmoid

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



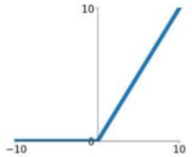
tanh

$\tanh(x)$



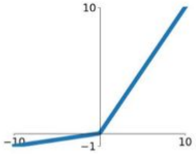
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

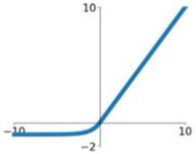


Maxout

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

ELU

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



Regularization

- One of the major aspects of training the model is overfitting -> the ML model captures the noise in your training dataset
- The **regularization** term is an addition to the loss function which helps generalize the model
  - **L1** or Lasso regularization adds a penalty which is the sum of the absolute values of the weights
    - L1+MSE

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}|w_i|)$$

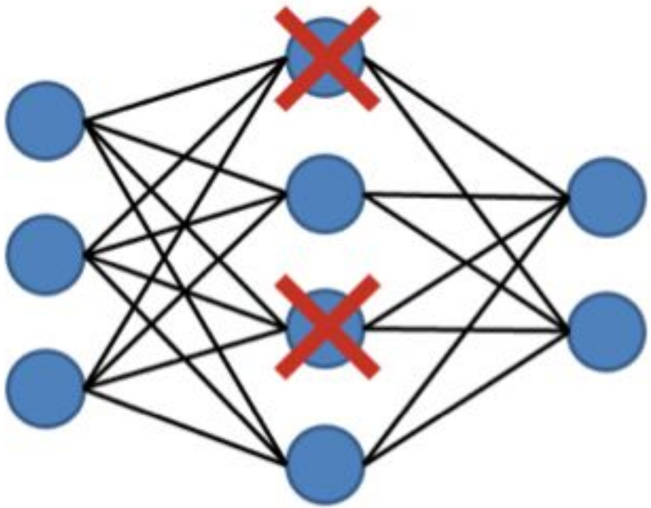
  - **L2** or Ridge regularization adds a penalty which is the sum of the squared values of weights
    - L2+MSE

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}(w_i)^2)$$

- **Early Stopping** is a time regularization technique which stops training based on given criteria

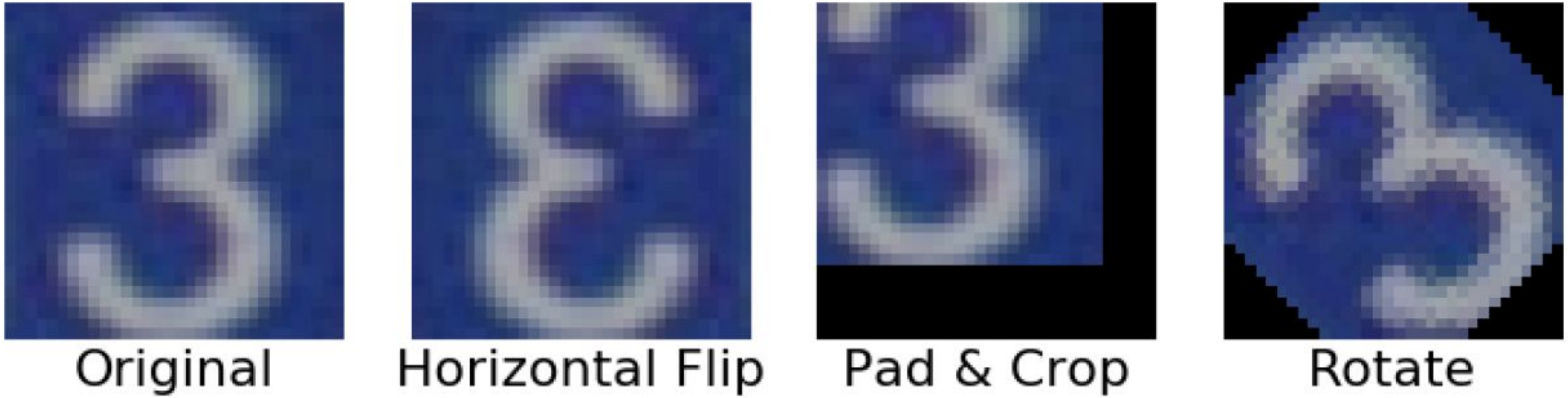
Dropout

- For each instance drop a node (hidden or input) and its connections with probability *p* and train
- Final net just has all averaged weights (actually scaled by 1-*p*)
- As if ensembling 2^n different network substructures

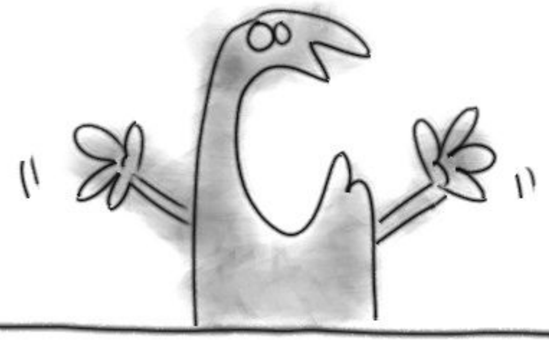


Data augmentation

- Techniques to significantly increase the diversity of data available for training models, without actually collecting new data
- Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks



Now What?!!



Training a neural network

- **Problem:** Predict gender from weight and height
- **Input Dataset:**

Features | Labels

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

1. Feature engineering

- Symmetrize numeric values
- Category -> numbers

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

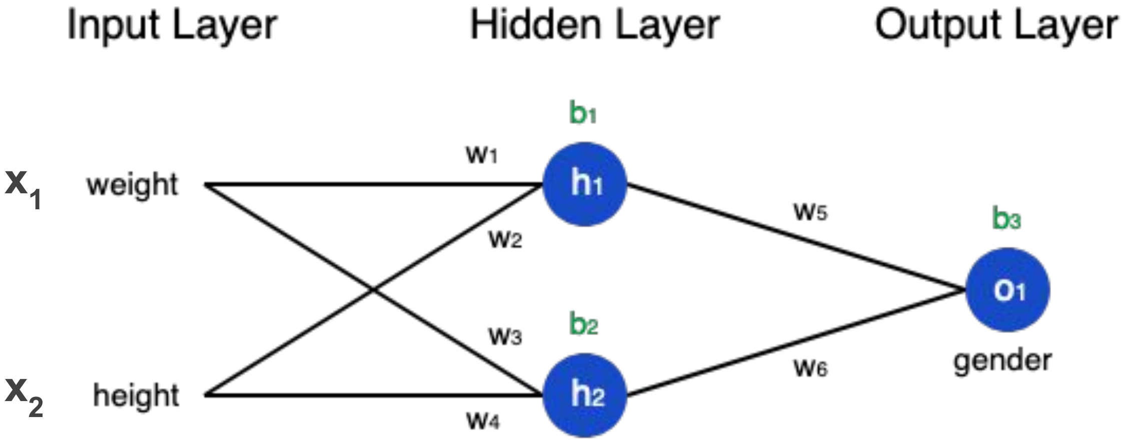
Ingredients

- **n** : 4, number of samples (Alice, Bob, Charlie, Diana)
- **Inputs:** **X**, dimension = 2
  - weights **(X_1)** and heights **(X_2)**
- **Features: x = (x_1, x_2)**, transformed inputs
- **y** : variable being predicted (Gender)
- y_{true} : true value of y

Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

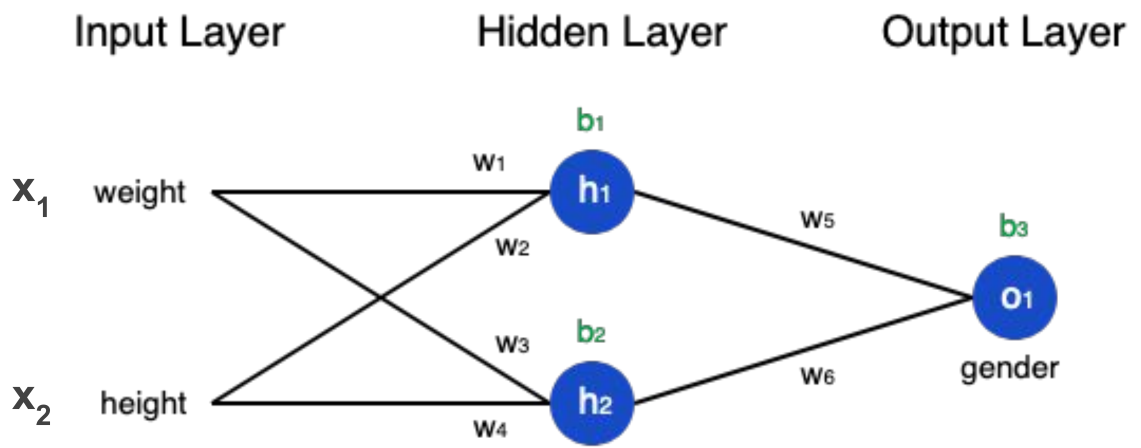
2. Model

- $\mathbf{y_{pred}}$: predicted value of y = **o**
- Neural network: with 1 hidden layer with 2 neurons
- Outputs of the hidden layer: **h**
- Unknown parameters: weights **w** and biases **b**



Some math

- For each neuron: $y_j = b_j + \mathbf{f} \sum_i x_i w_{ij}$, **f** is the activation function
- For the net:
  - $\mathbf{h_1 = f(w_1x_1+w_2x_2)+b_1}$
  - $\mathbf{h_2 = f(w_3x_1+w_4x_2)+b_2}$
  - $\mathbf{o_1 = f(w_5h_1+w_6h_2)+b_3}$



3. Model training

- Training the network == Find weights **w** and biases **b** that minimize the loss
- Loss function **L: MSE**

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

- Find weights **w** and biases **b** to minimise

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

- Typically, this is already implemented in ML software packages

Back propagation - 1986

- Gradient descent
- Chain rule
- Layered Computation

- Minimization taking **partial derivatives**
- For very simple case: with only Alice in the dataset, n=1

$$\boxed{\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

$$L = (1 - y_{pred})^2$$

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}}$$

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$

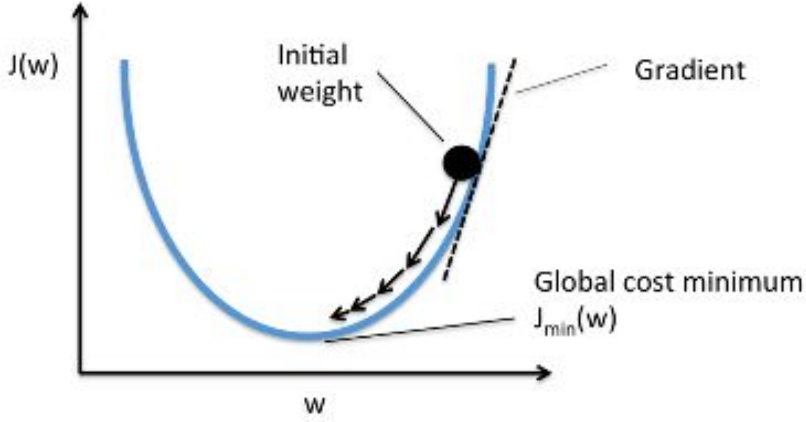
$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)}$$

Stochastic Gradient Descent (SGD)

- Start with randomly initialised weights
- **update equation**:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

- **η** is a constant called the **learning rate** that controls how fast we train

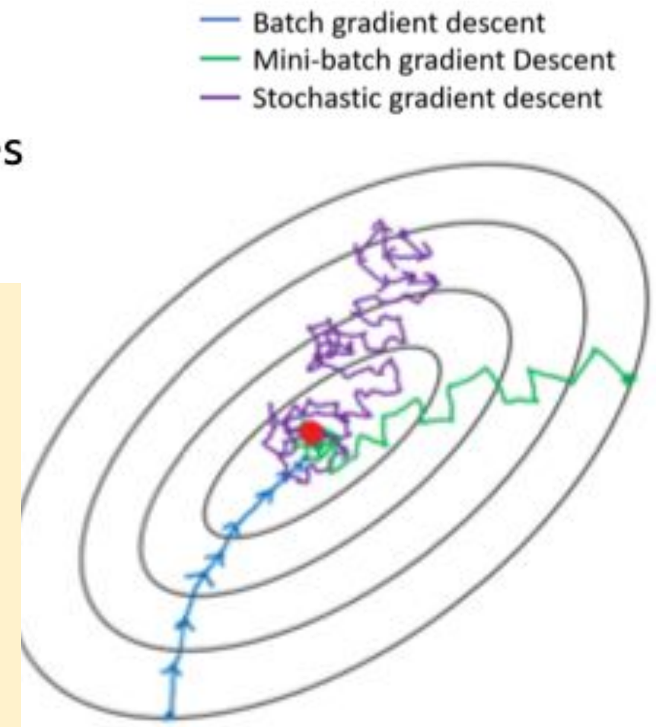


- If $\frac{\partial L}{\partial w_1}$ is positive, w_1 will decrease, which makes L decrease.
- If $\frac{\partial L}{\partial w_1}$ is negative, w_1 will increase, which makes L decrease.

- **Stochastic** vs **Batch (BGD)**-> the parameters are updated using only one single training instance (usually randomly selected) in each iteration vs the whole training set (== batch)

Mini-batch Gradient Descent

- Use mini-batch **sampled** in the dataset for gradient estimate.
- Sometimes helps to escape from local minima
- Noisy gradients act as regularization
- Variance of gradients increases when batch size decreases
- Not clear how many sample per batch



What happens in **one epoch** for SGD:

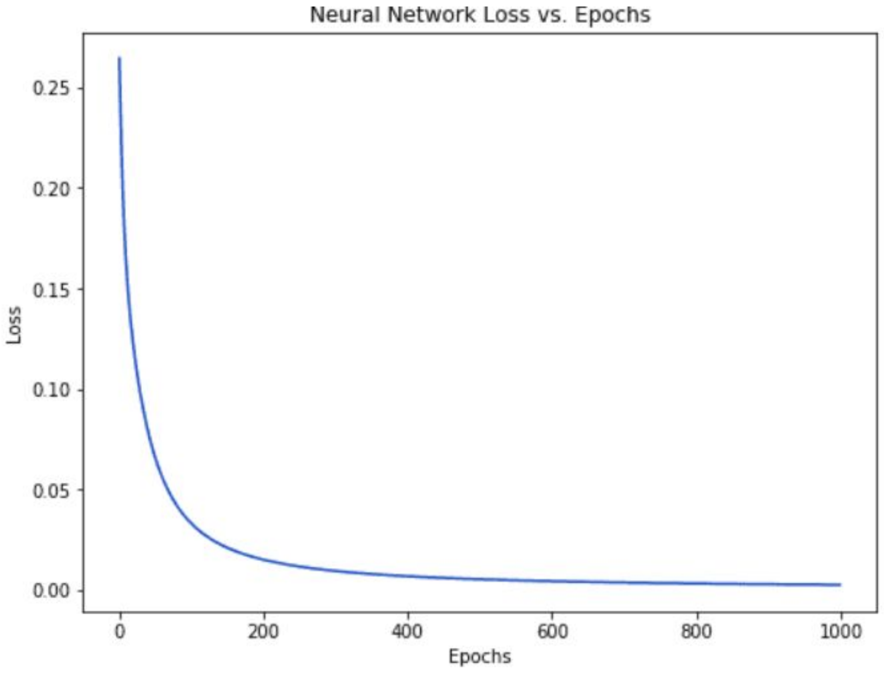
1. Take a random sample
2. Feed it to Neural Network
3. Calculate its gradient
4. Use the gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training dataset

What happens in **one epoch** for Mini-BGD:

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created

Take home messages

- **Gradient descent** is an iterative learning algorithm that uses a training dataset to update a model.
    - **BGD**: Batch Size = Size of Training Set
    - **SGD**: Batch Size = 1
    - **Mini-BGD**. 1 < Batch Size < Size of Training Set



- The **batch size** is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.

- The number of **epochs** is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset

Loss functions

- https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0
- https://www.wikiwand.com/en/Loss_functions_for_classification