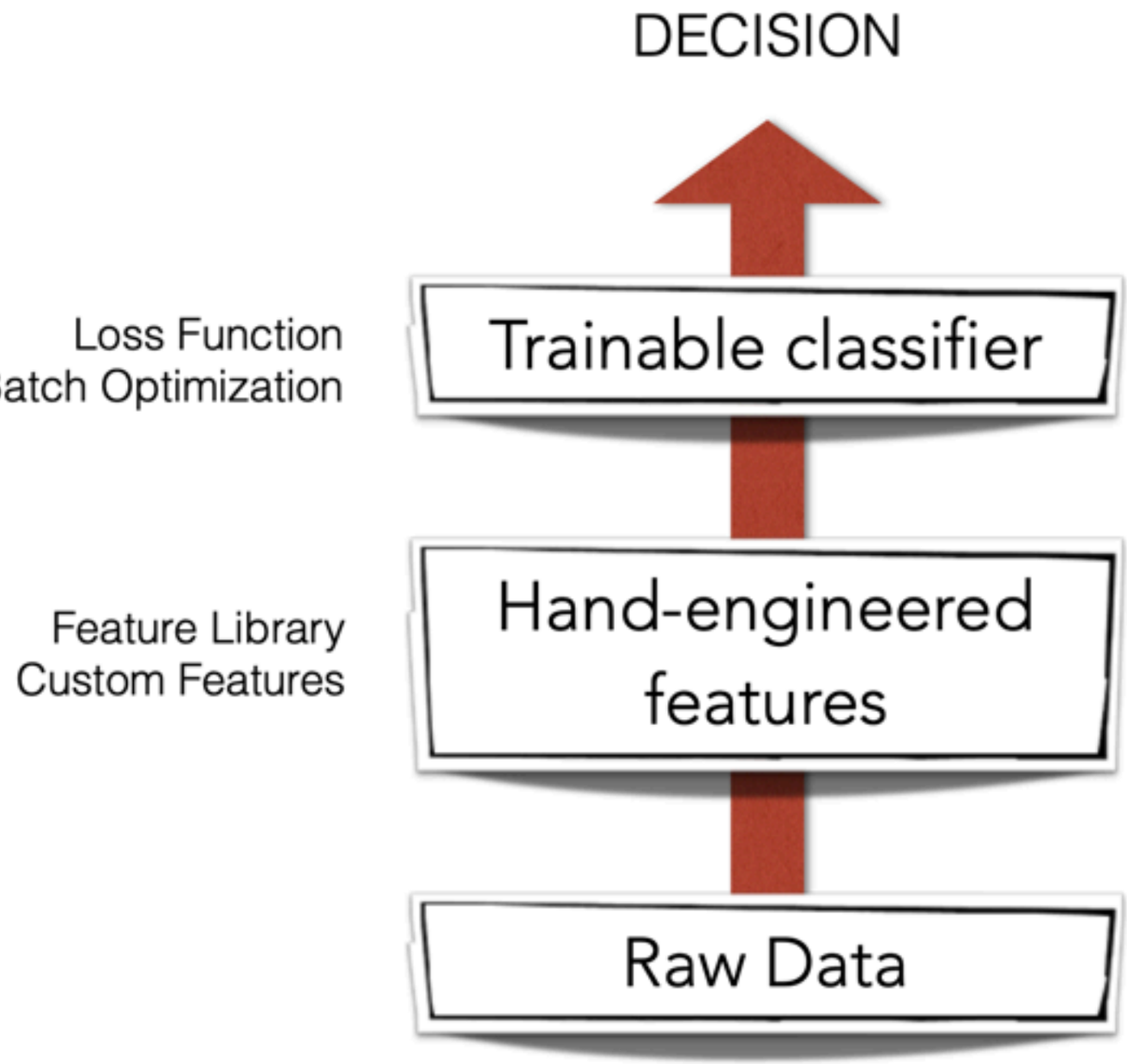
UNIVERSITAT DE BARCELONA

Neural Networks

Santi Seguí | IMIM-2020

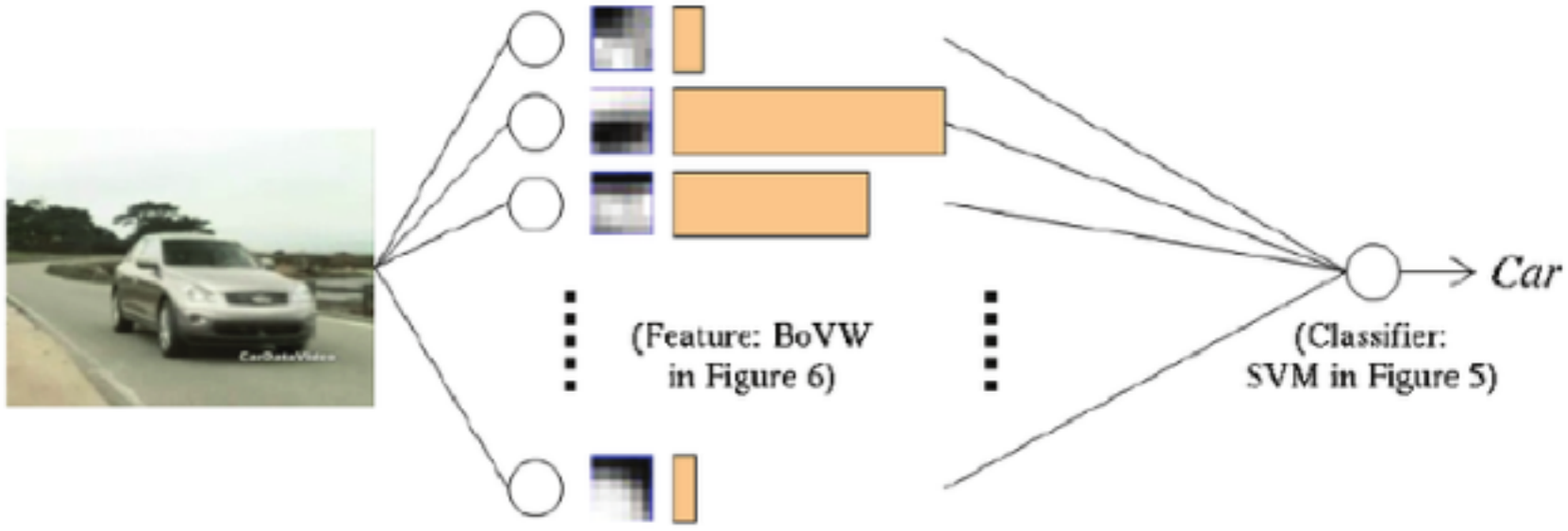
Standard Machine Learning



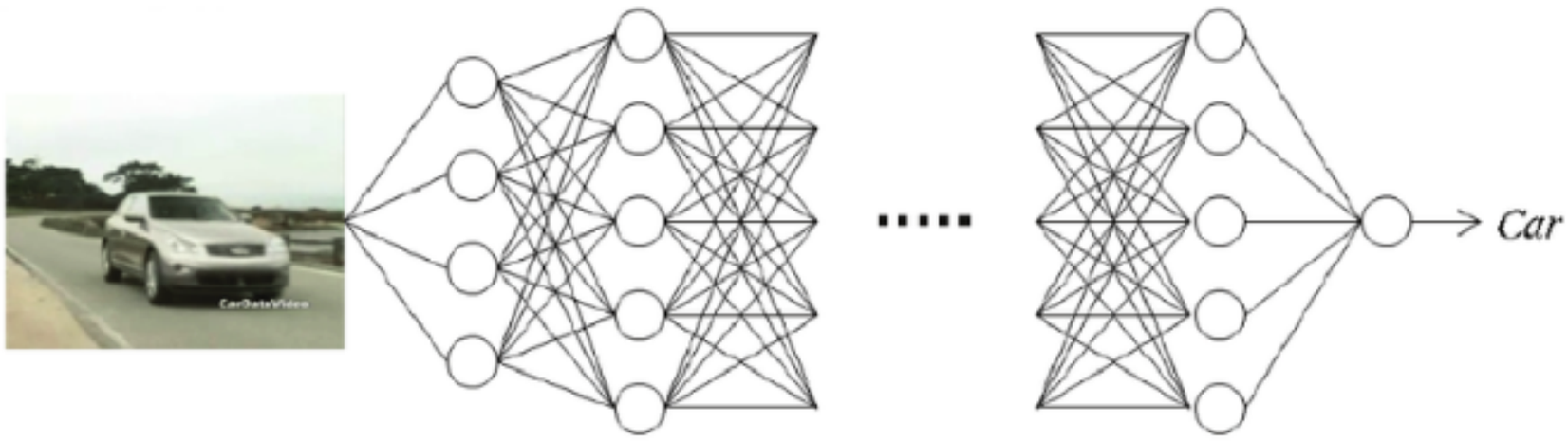
Deep Learning

Backpropagation + Tricks

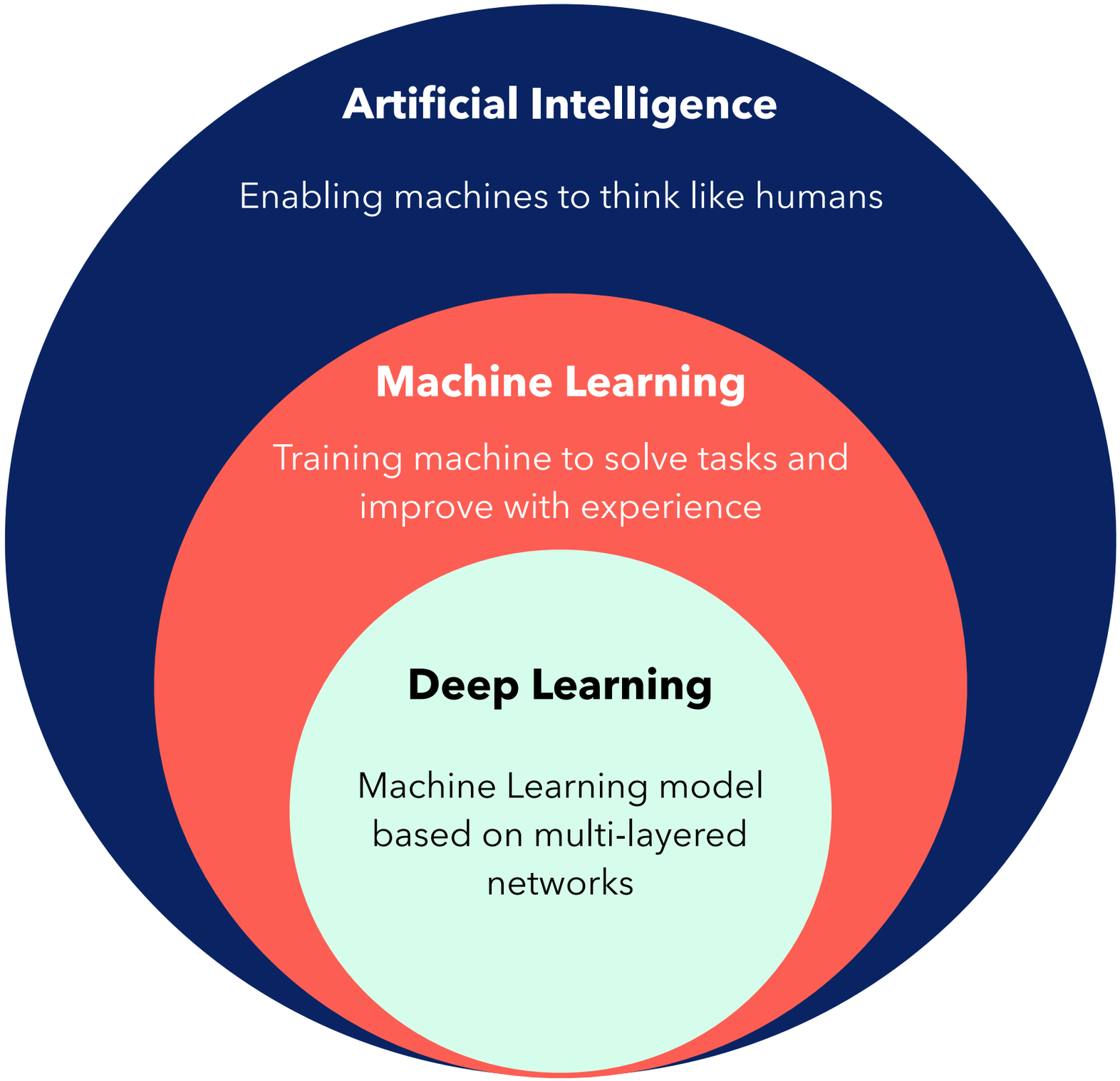
Loss Function
Network Architecture
Stochastic Gradient Descend

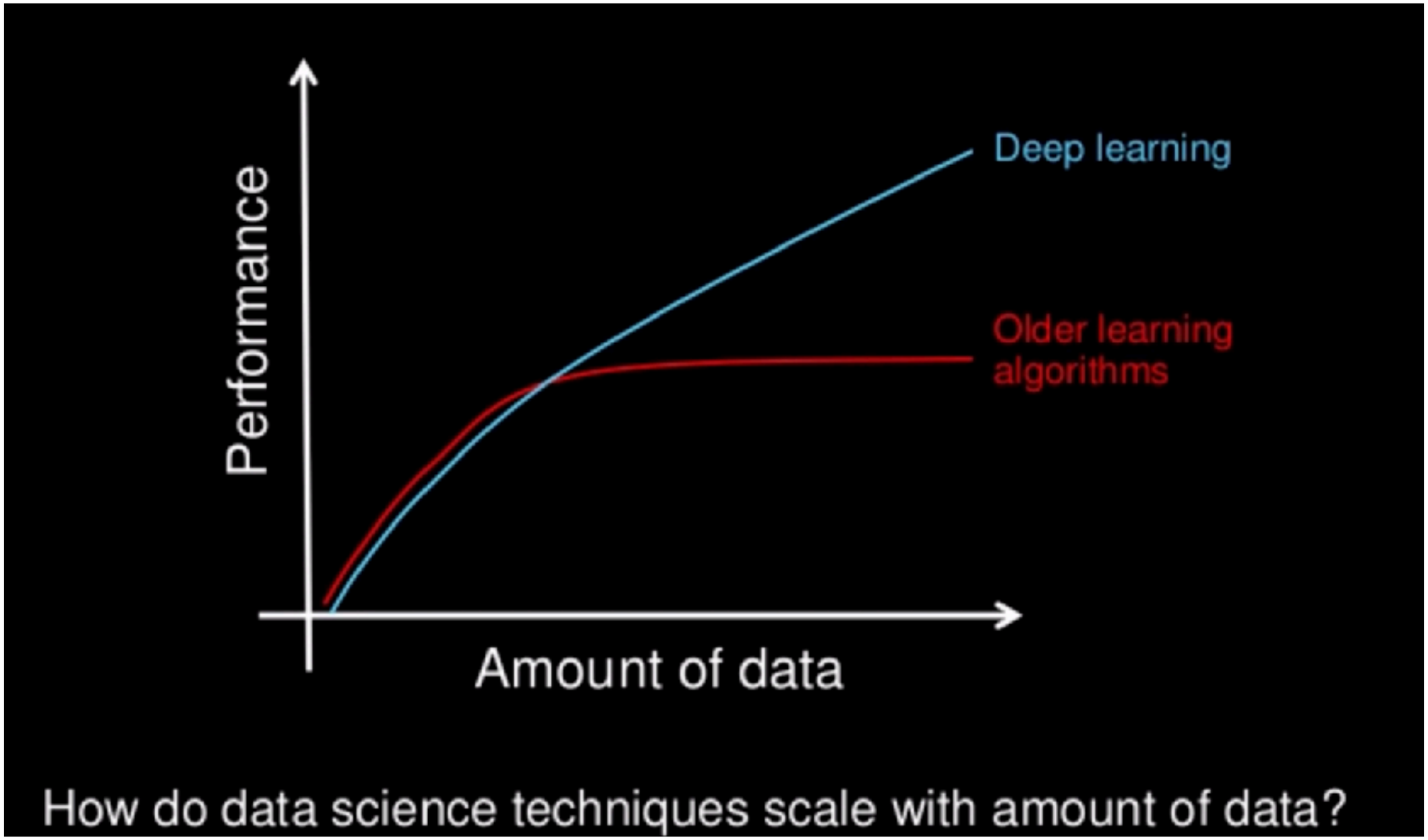


Traditional Machine Learning

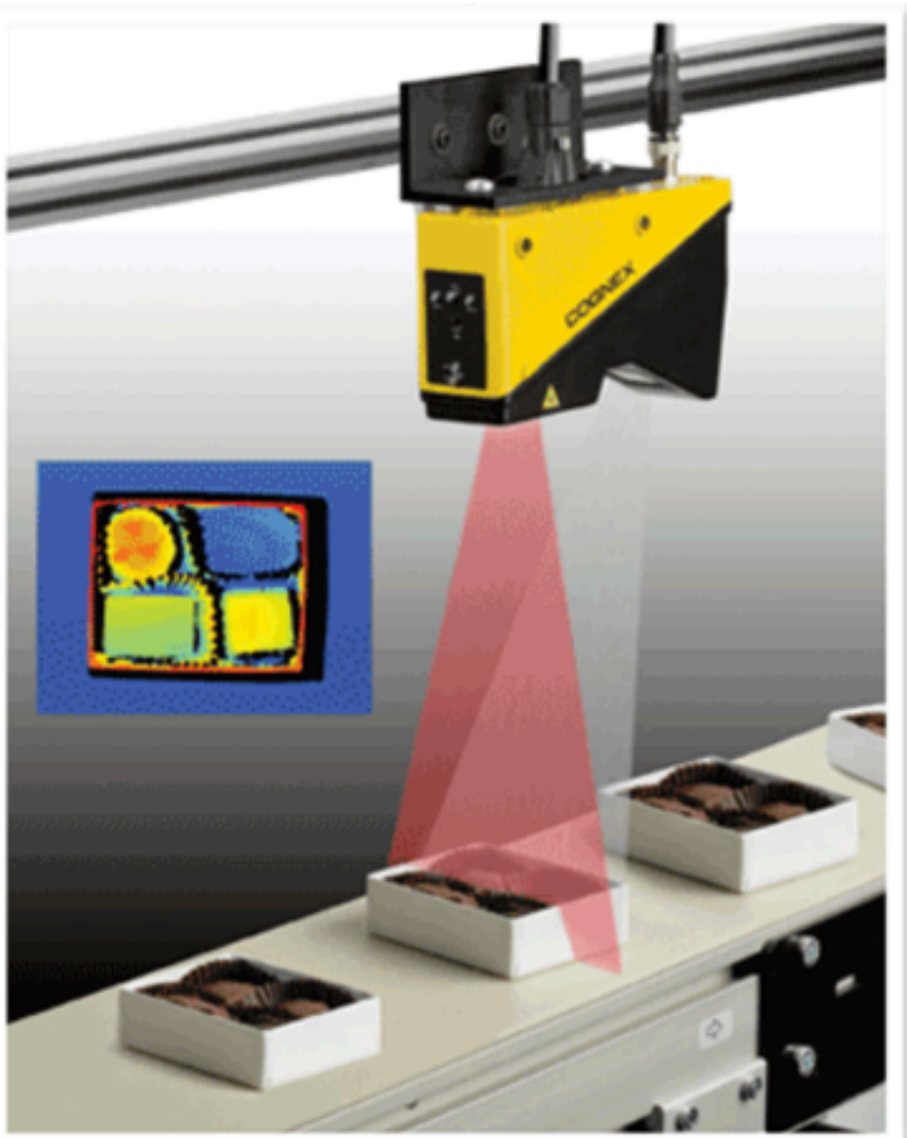


Deep Learning


Artificial Intelligence
Enabling machines to think like humans
Machine Learning
Training machine to solve tasks and improve with experience
Deep Learning
Machine Learning model based on multi-layered networks


Performance
Deep learning
Older learning algorithms
Amount of data
How do data science techniques scale with amount of data?

"Classical" applications of computer vision:
object classification, detection and segmentation… in controlled environments.



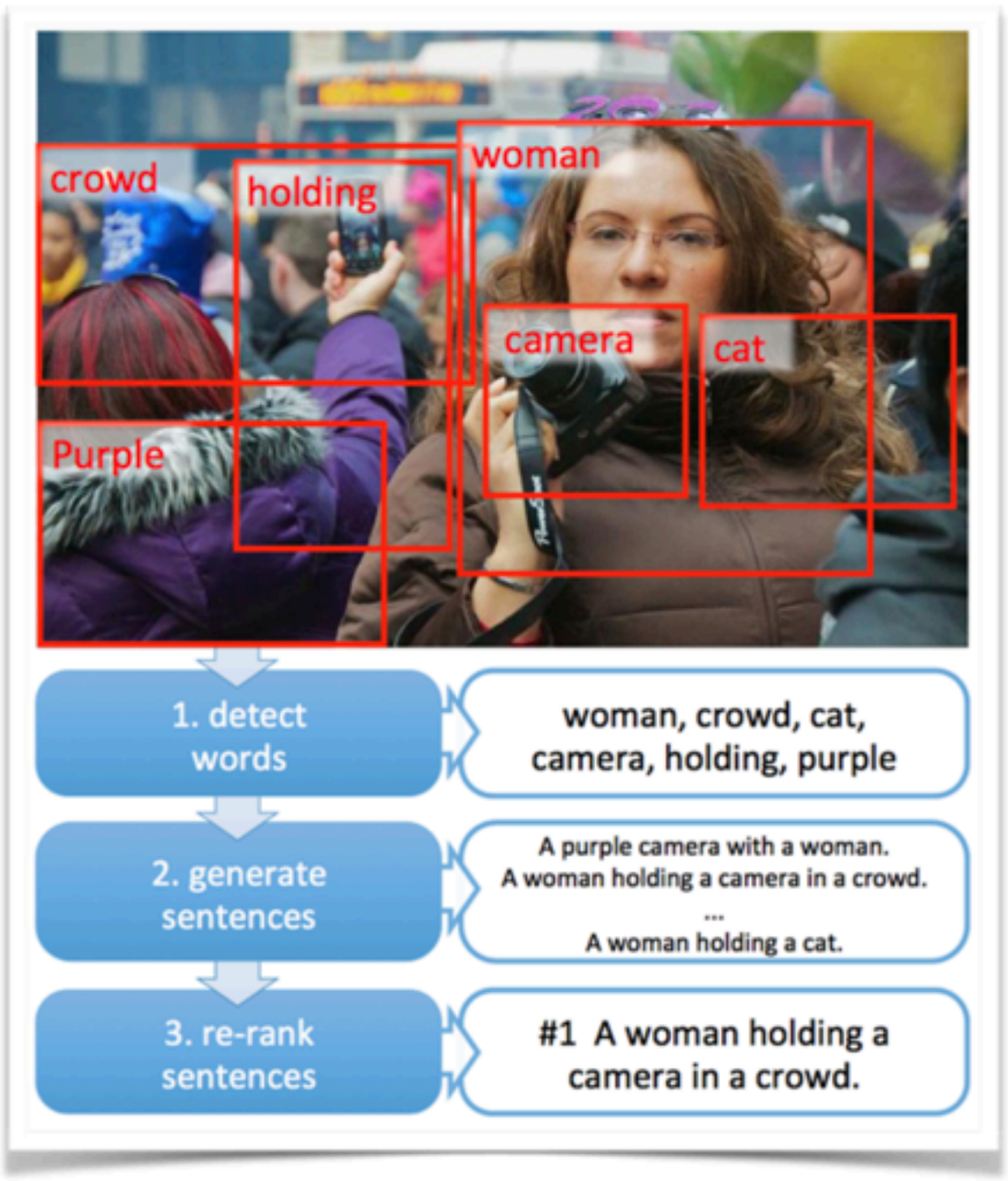
New applications: face recognition.

DeepFace (Facebook): Accuracy of 97.35%

From 2012 deep learning has paved the way for new applications: navigation and mapping.

New applications: Image Upscaling (Flipboard)

New applications: Automatic Image Captioning



http://blogs.technet.com/b/machinelearning/archive/2014/11/18/rapid-progress-in-automatic-image-captioning.aspx

When DL is useful?

- Anywhere where we have strong (spatial or temporal) **correlation between features** and a **lot of data** to automatically learn find patterns.

Images, text, audio, video…

What is a Neural Network?

- It is a system **biologically inspired** that tries to emulate **human brain**.

What is a Neural Network?

- It is a system **biologically inspired** that tries to emulate **human brain**.

- **Why** is it a good idea to try to **emulate** the **brain** when solving a recognition task?

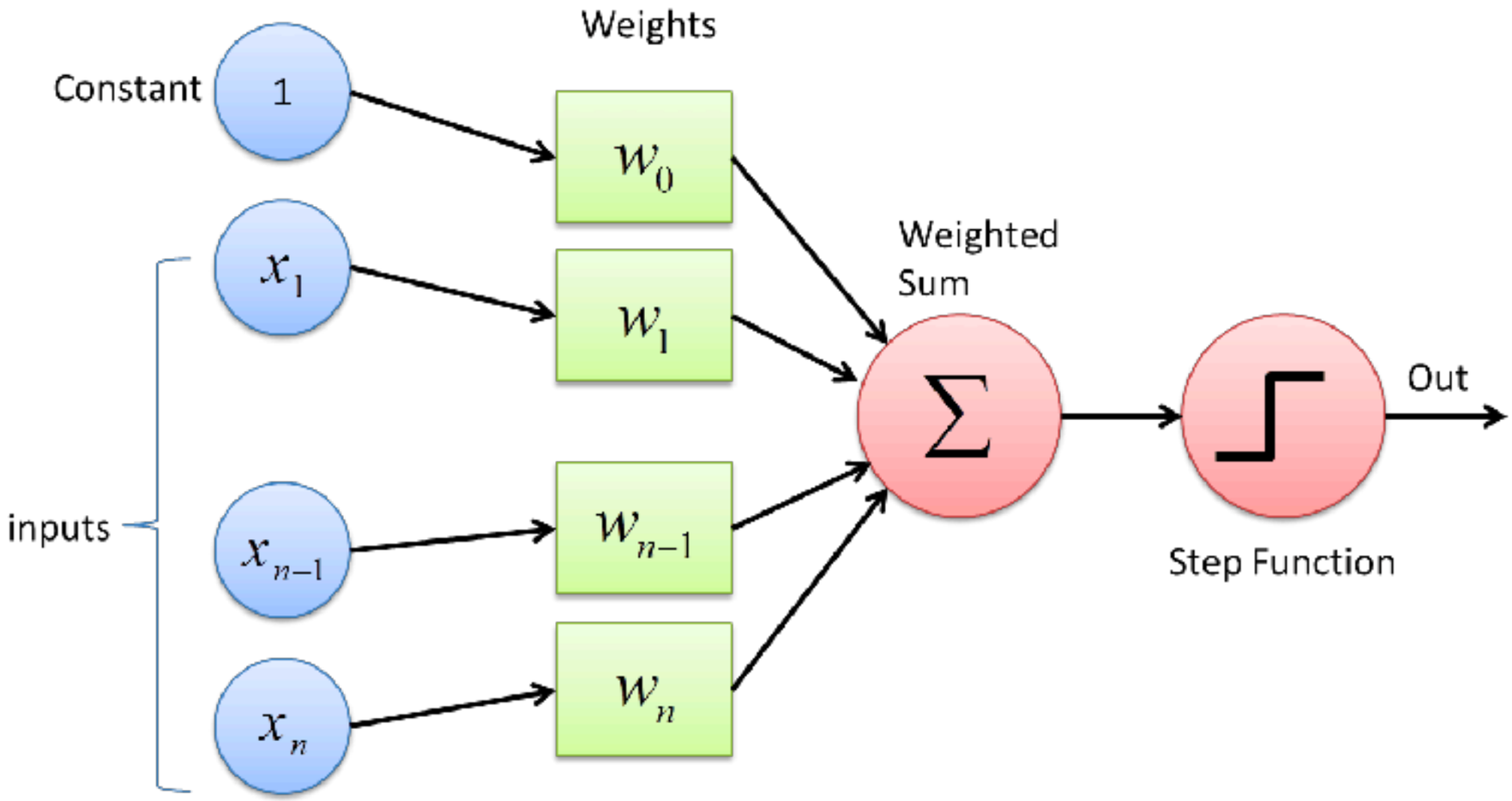
What is a neuron?

What is a neuron?

- In the human brain, a typical neuron collects signals from others through a host of fine structures called dendrites.

- The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches.

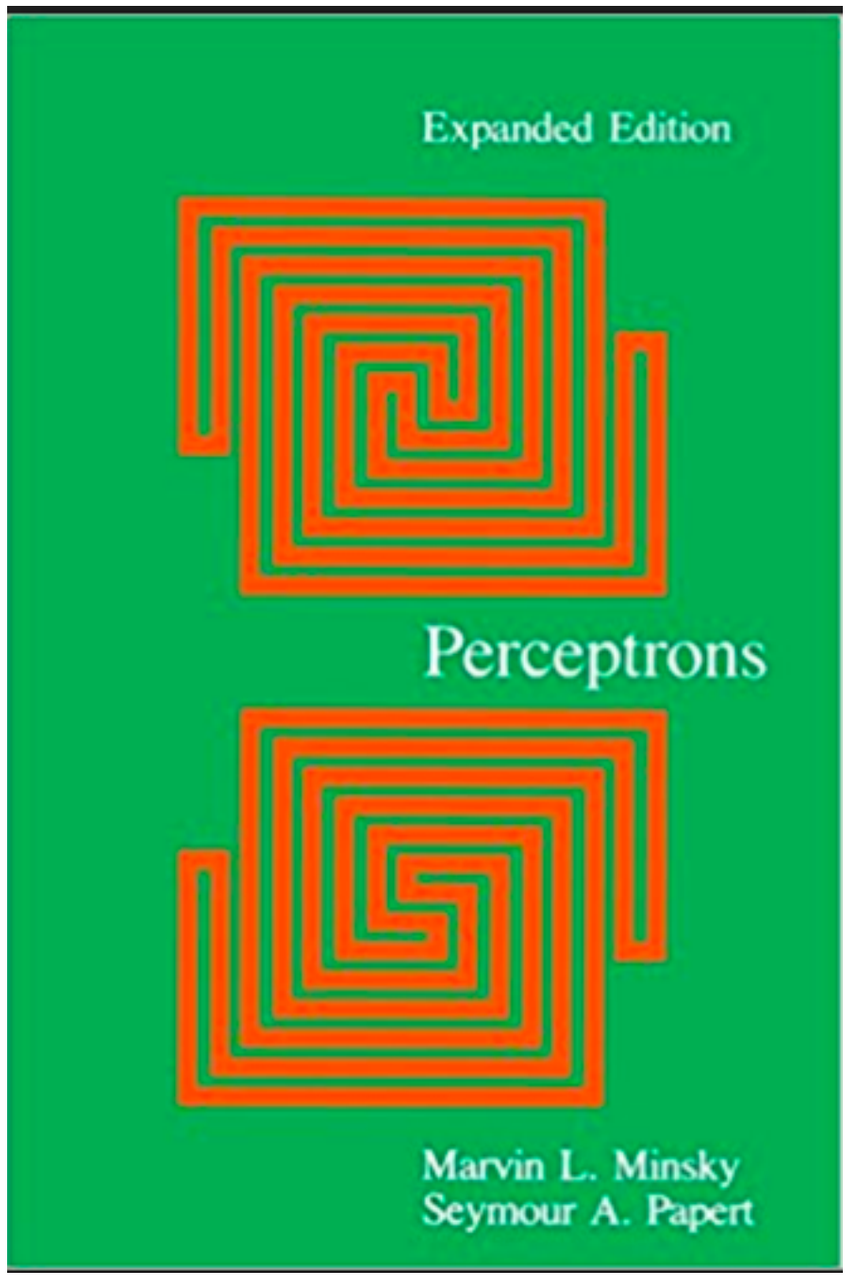
- At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

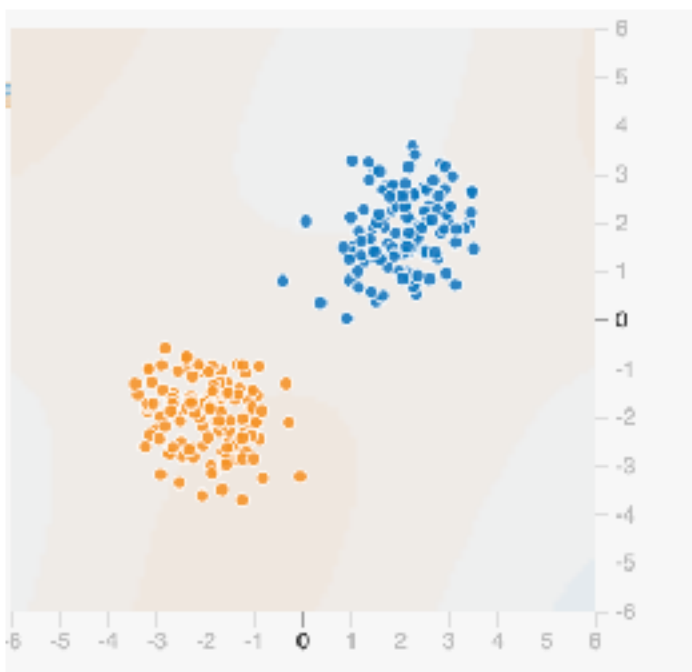


Perceptron

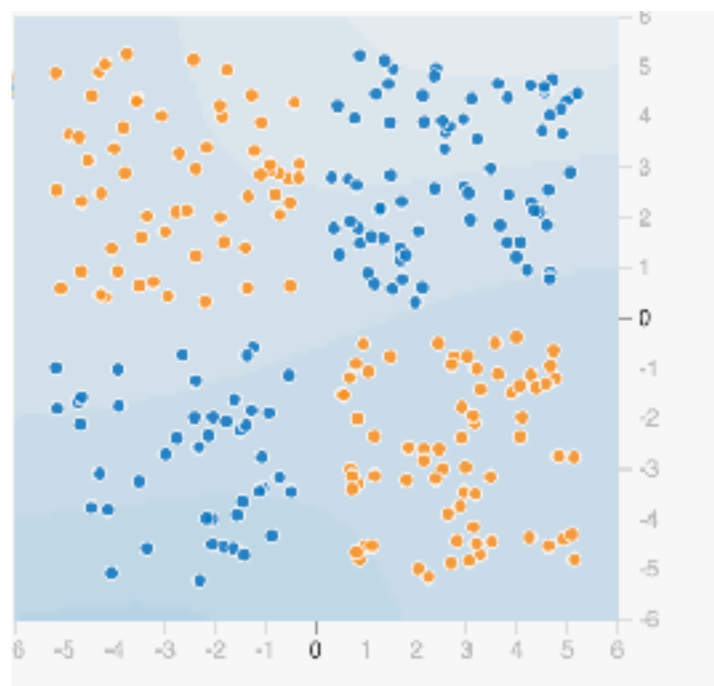
Perhaps the first AI model

The Book: Perceptrons

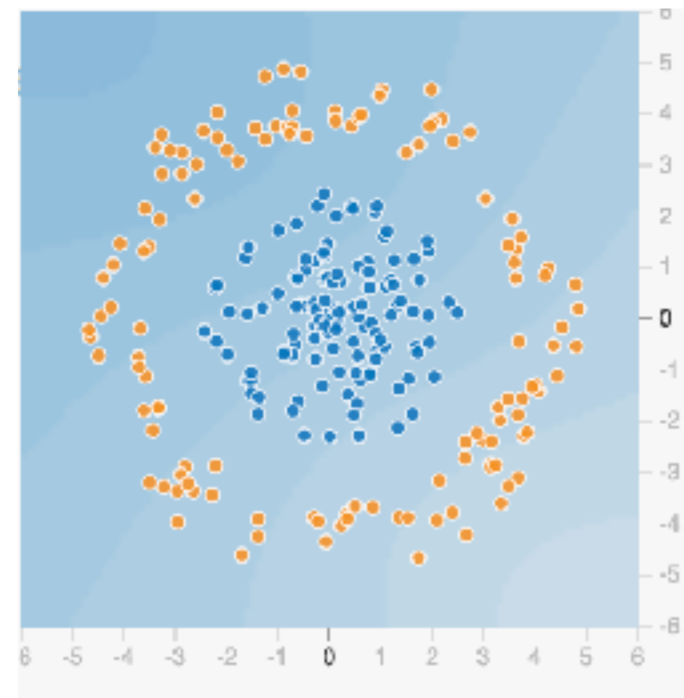




-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6



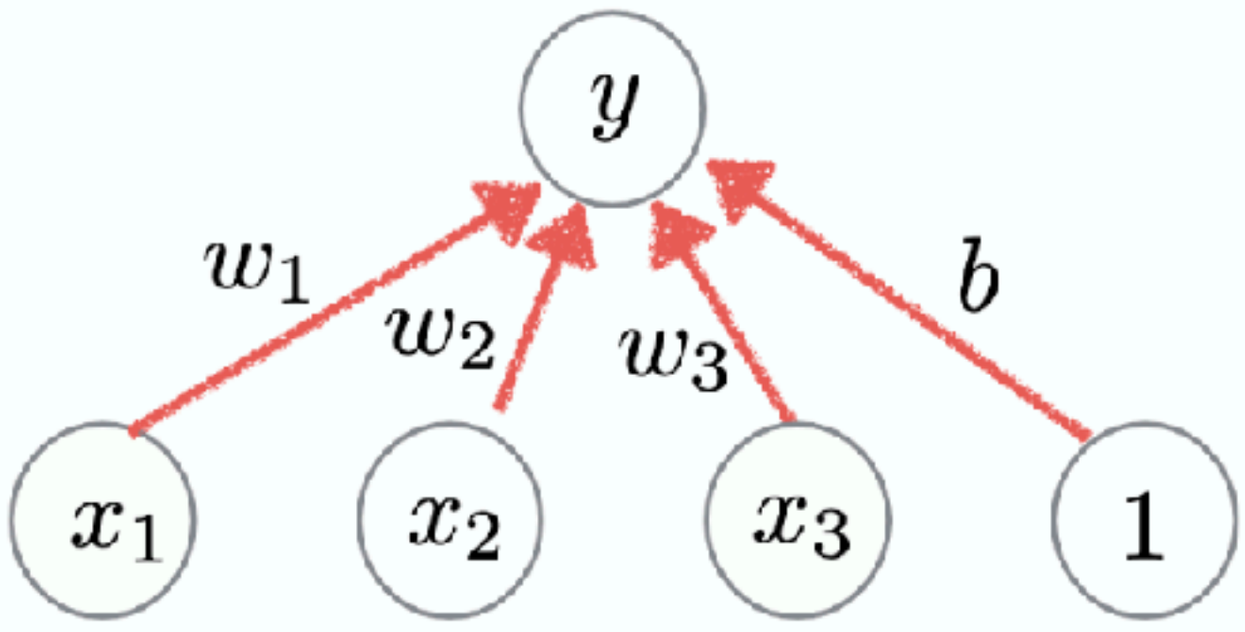
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6



-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6

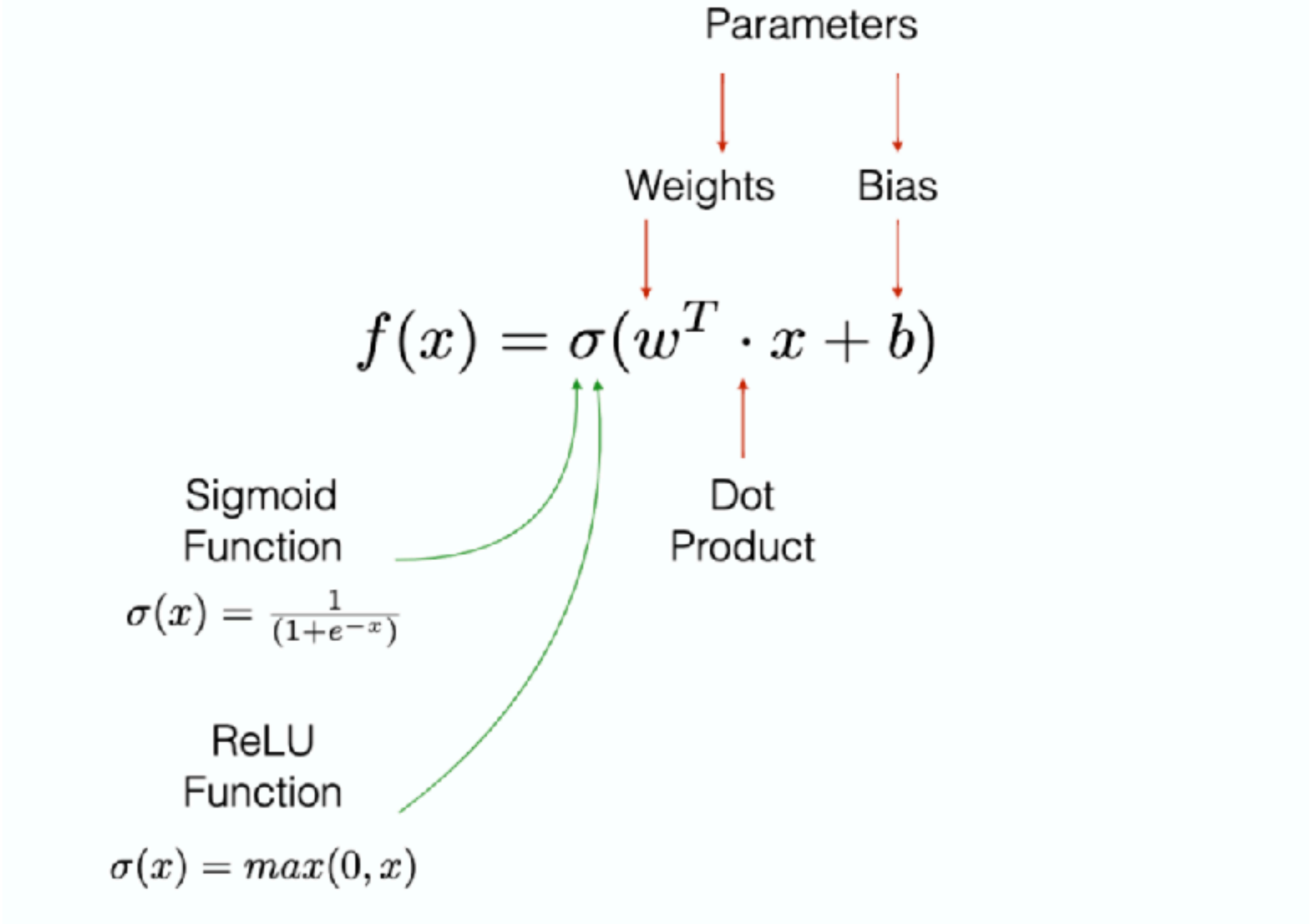
1 Layer Neural Net model

$$f(x) = \sigma(w^T \cdot x + b)$$

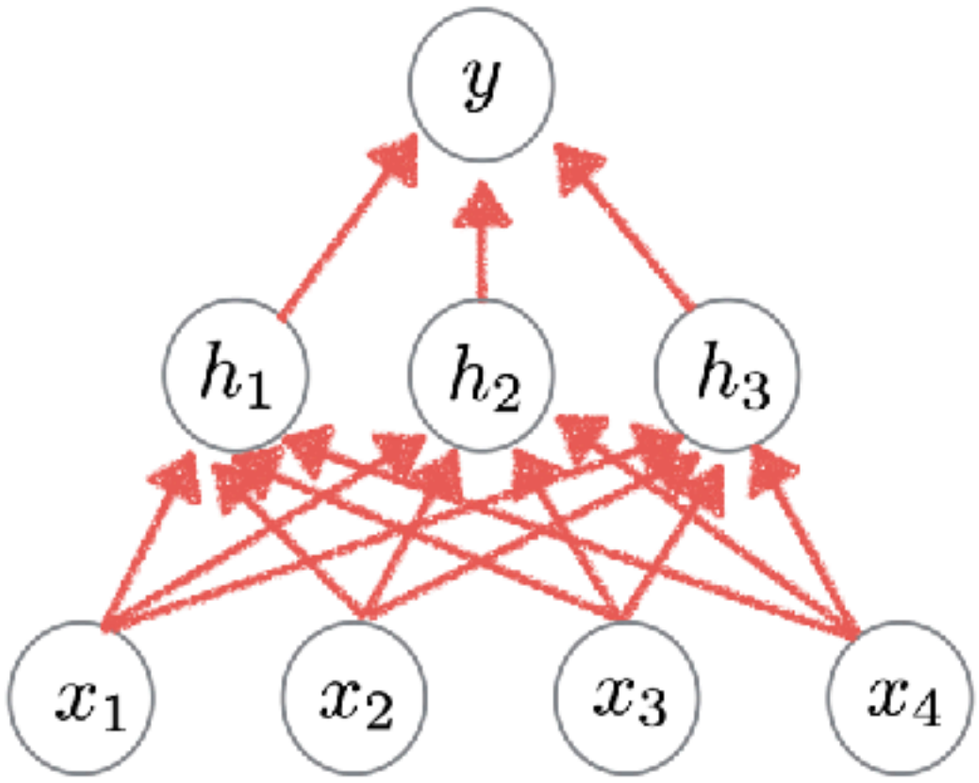


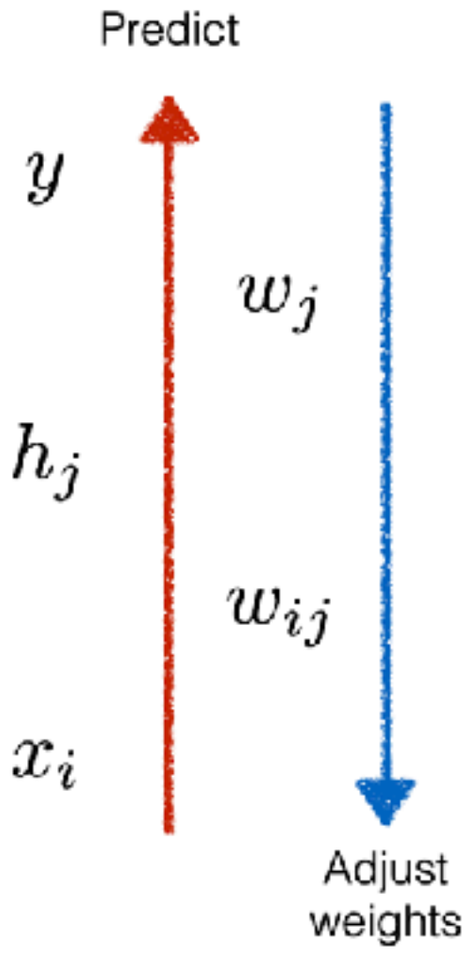
Graphical Representation

1 Layer Neural Net model



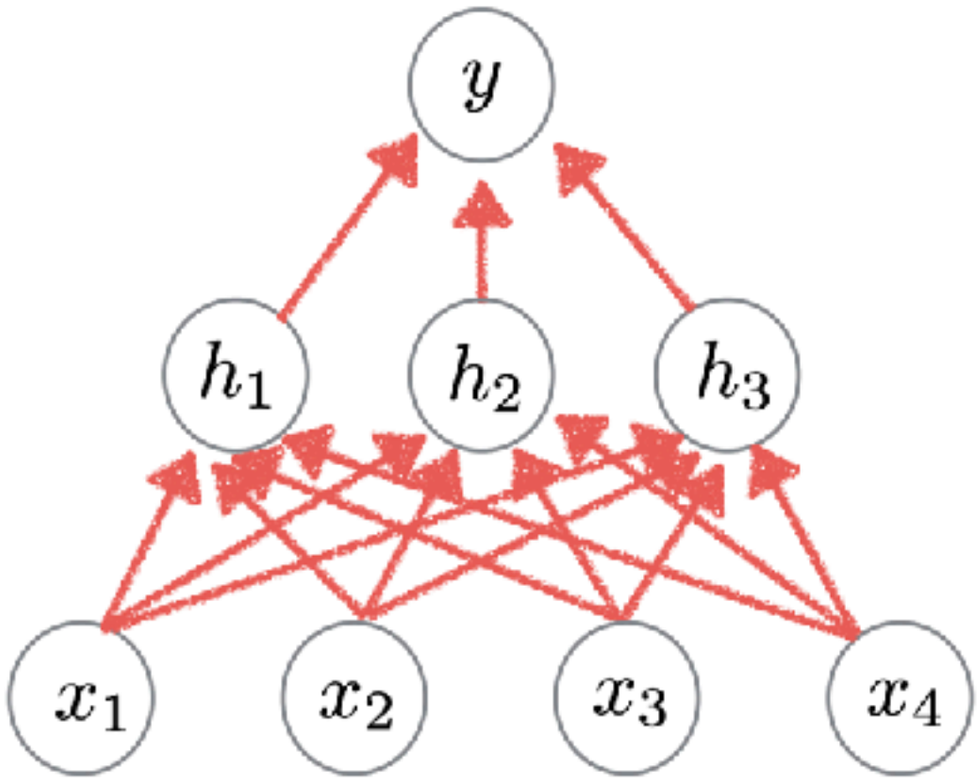
Training a two layer by **backpropagation**

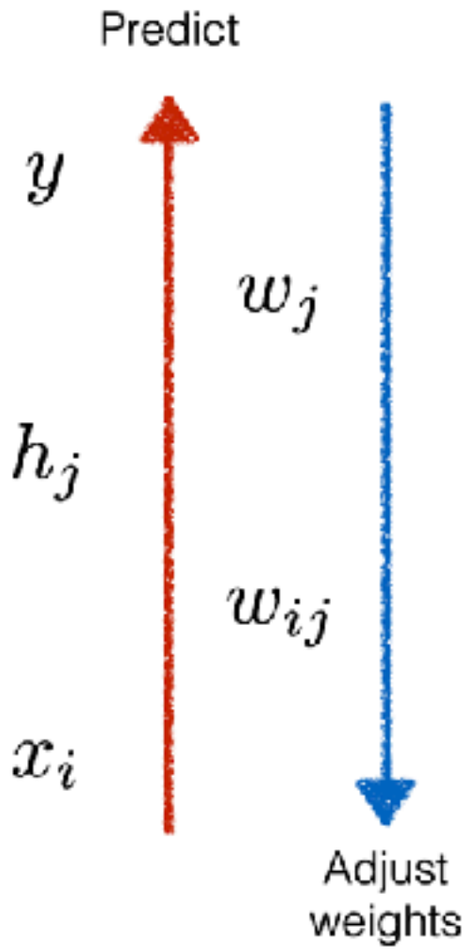




$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

Training a two layer by **backpropagation**





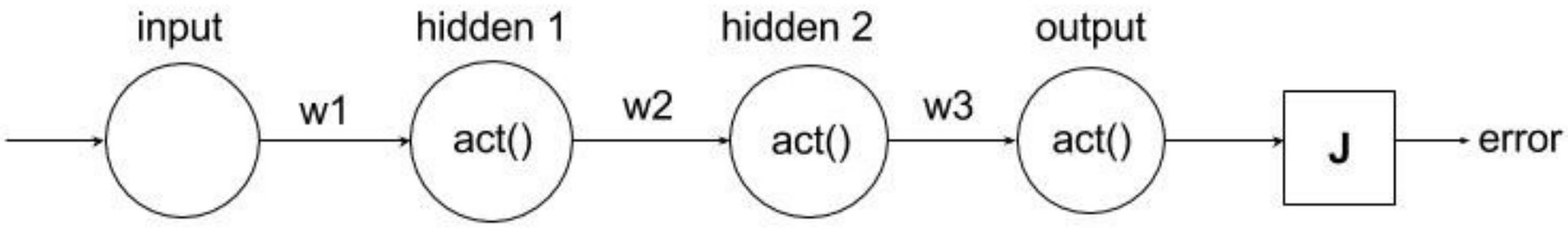
$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

1. For each sample, compute $f(x^{(m)}) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i^{(m)}))$.
2. If $f(x^{(m)}) \neq y^{(m)}$, back-propagate error and adjust weights $\{w_{ij}, w_j\}$

1-Layer Net function model: $f(x) = \sigma(w^T \cdot x + b)$

Training:

1. Assume a Squared-Error (or Cross Entropy) Loss $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$
2. Initialize w
3. Compute $\nabla_w Loss = \sum_m Error^{(m)} * \sigma'(w^T x^{(m)}) * x^m$
4. $w \rightarrow w - \gamma(\nabla_w Loss)$
5. Repeat steps 3-4 until some condition satisfied



$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

Different types of Neurons

- Linear Neurons
  - These are one of the most simple neurons models computationally limited

$$y = \beta + \sum_i X - i\beta_i$$

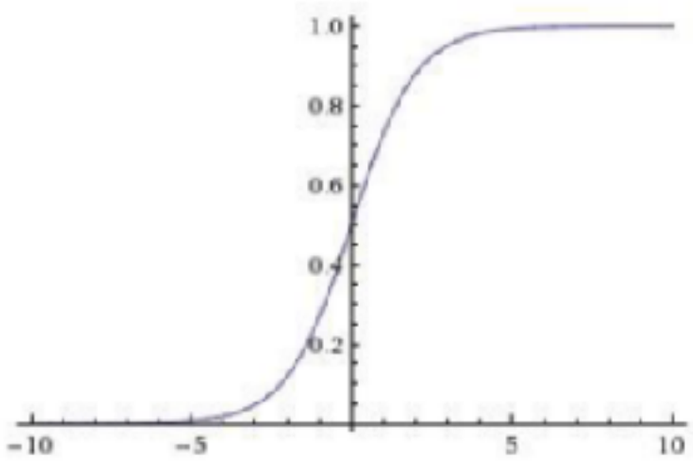
Different **types of Neurons**

- McCulloch-Pitts (Binary threshold neurons):
  - First compute a weighted sum of the inputs
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.

$$y = \beta_0 + \sum_i x_i \beta_i$$

$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Different **types of Neurons**



$$y = \frac{1}{1 + e^{-z}}$$

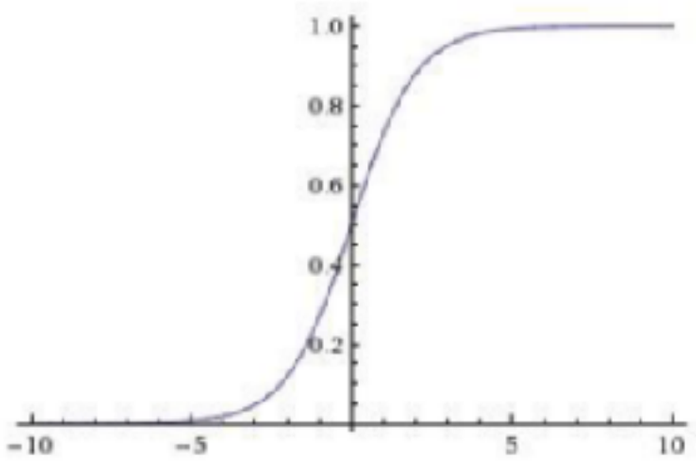
Sigmoid

- Squashes numbers to range [0,1].
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron.

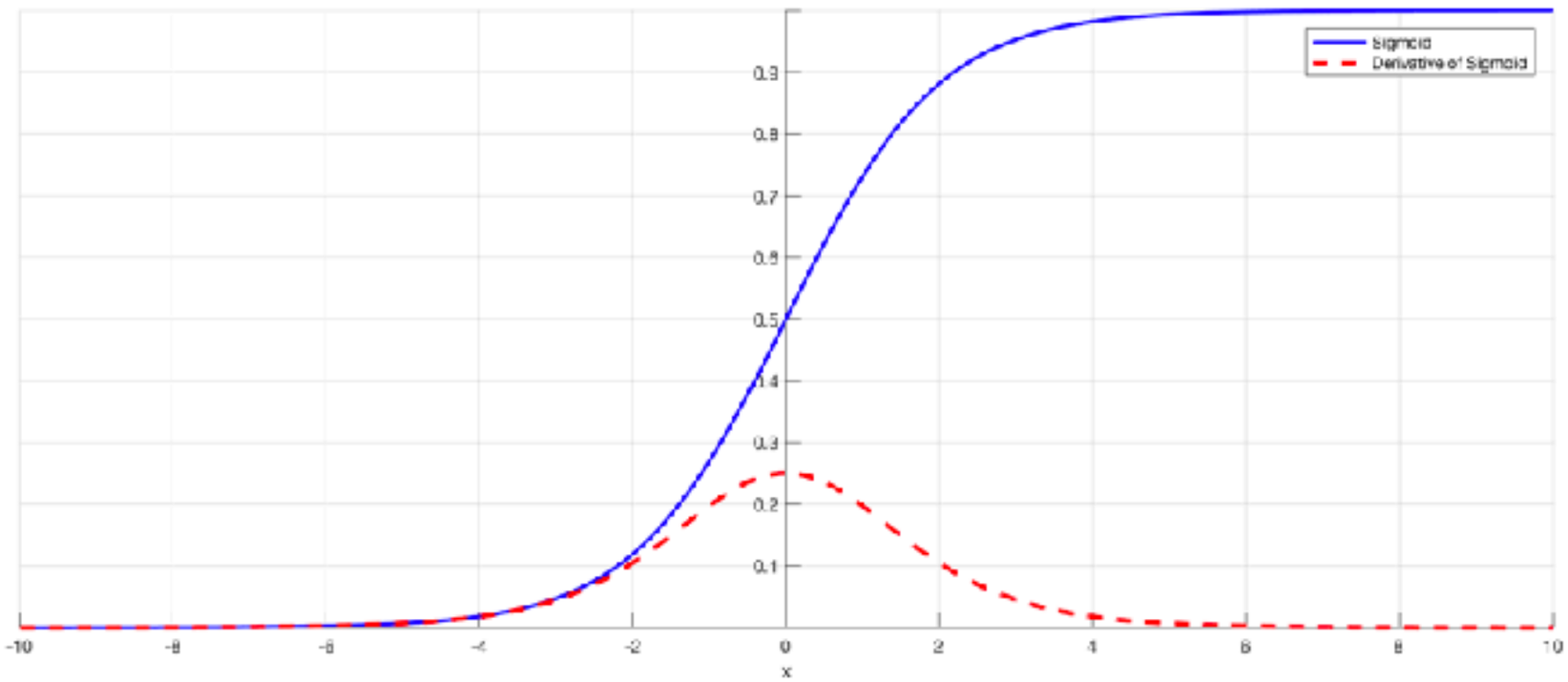
- 2 BIG problems:

- Saturated neurons “kill” the gradients.
- Sigmoid outputs are not zero centered.

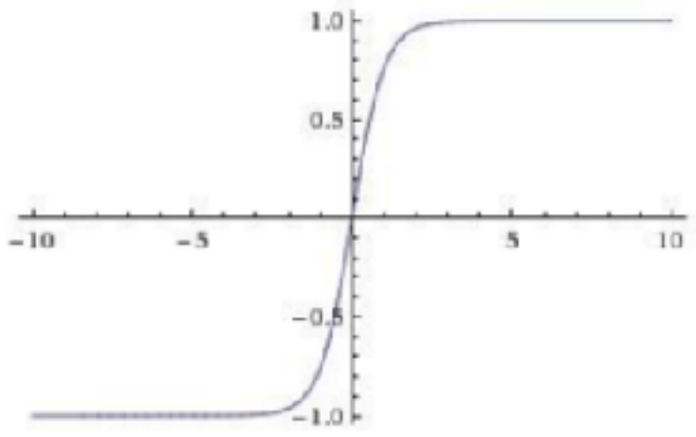
Different **types of Neurons**



Sigmoid



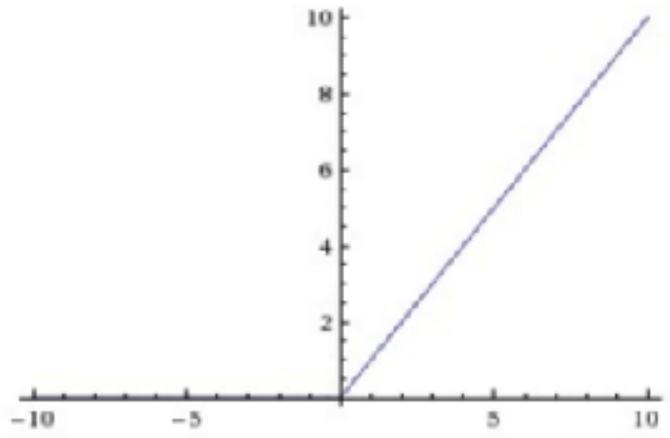
Different **types of Neurons**



tanh(x)

- Squashes numbers to range [-1,1].
- Zero centered.
- 1 BIG problems:
- Saturated neurons “kill” the gradients.

Different **types of Neurons**



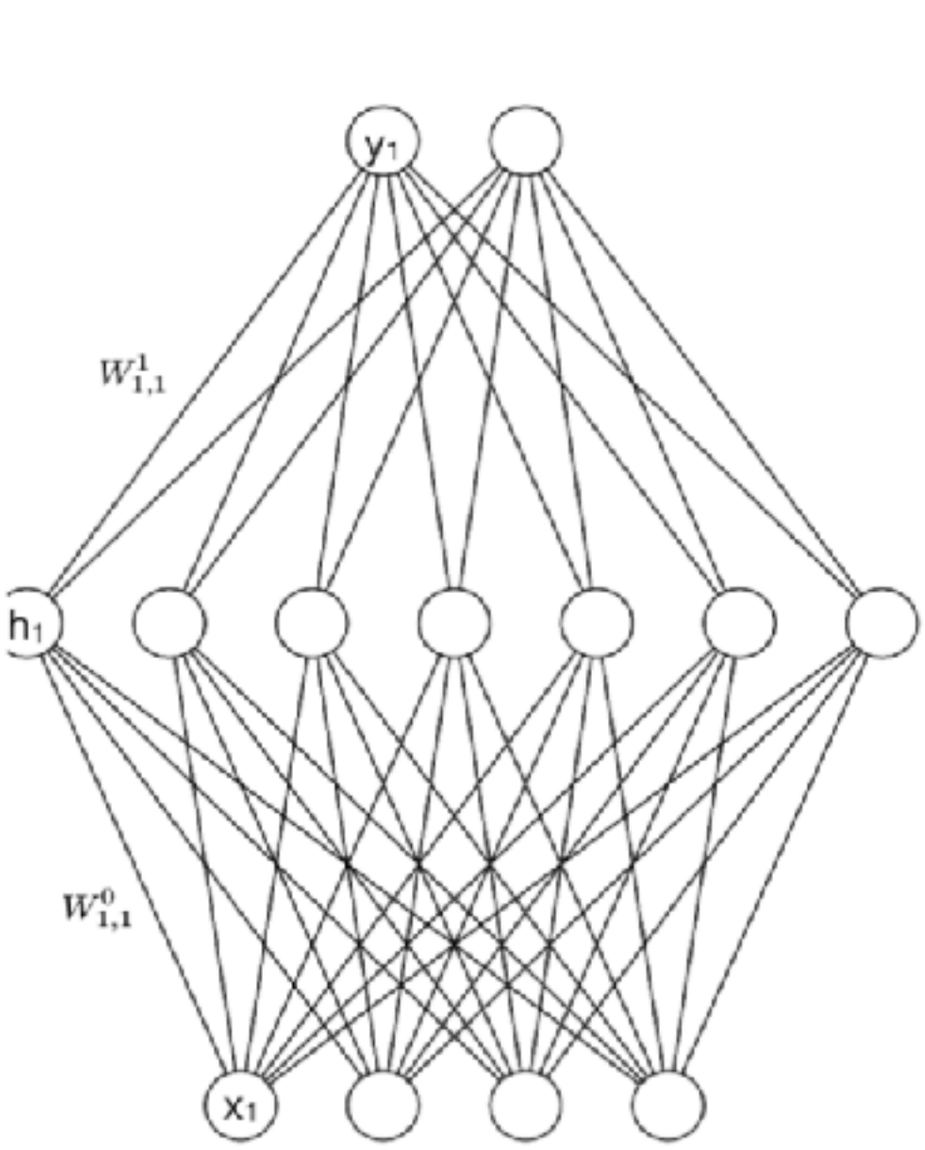
$$y = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

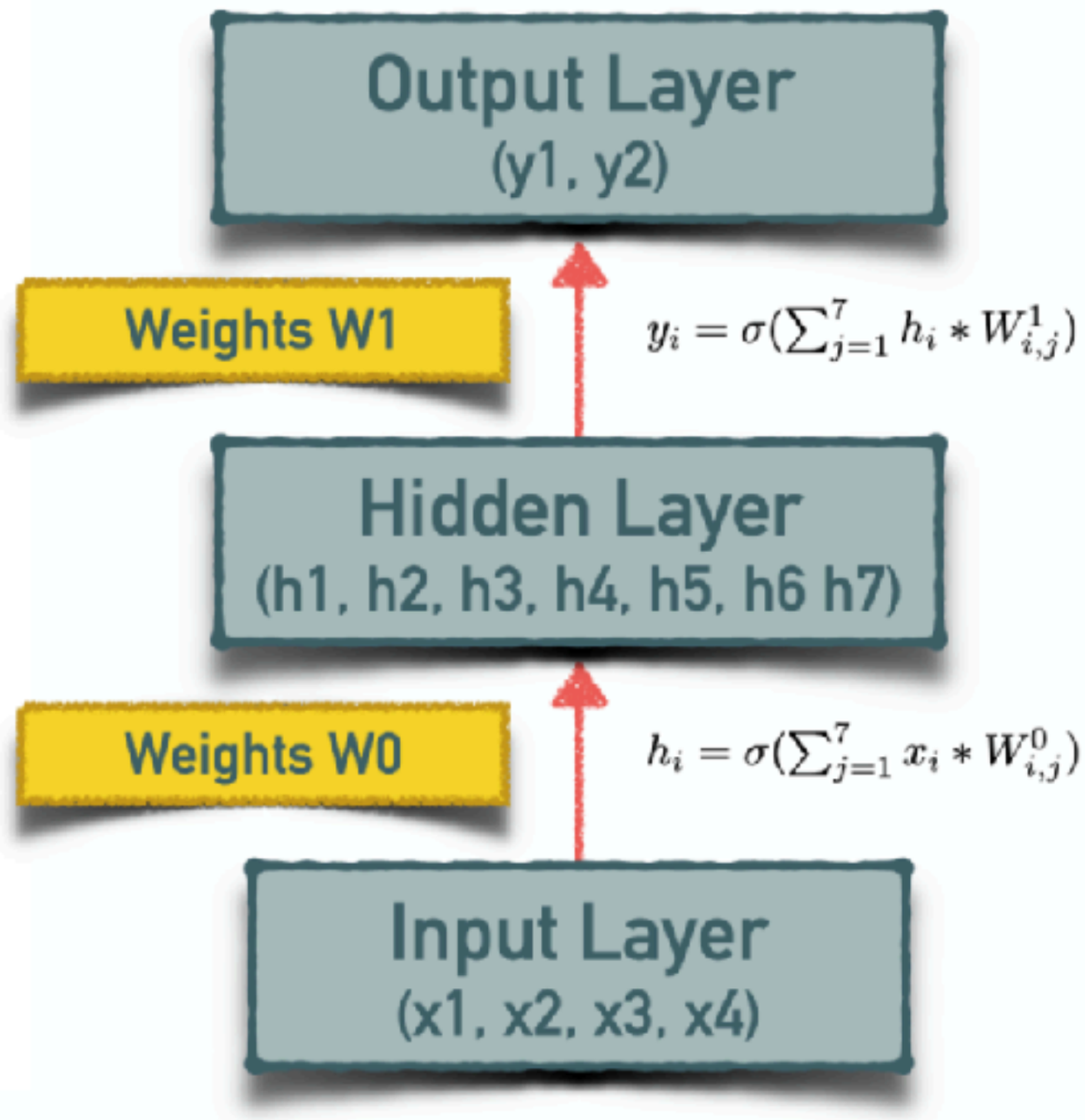
ReLU

- Does not saturate.
- Computationally efficient.
- Fast convergence.
- 1 small problems:
- Dead ReLu (x<0)

Feed-Fordward Neural Network

- These are the commonest type of neural networks in practical applications
    - The first layer is the input and the last layer is the output
    - If there is more than one hidden layer, we call them "deep" neural networks!
- They compute a series of transformations that change the similarities between cases.
    - The activities of the neuros in each layer are non-linear functions of the activites in the layer below.





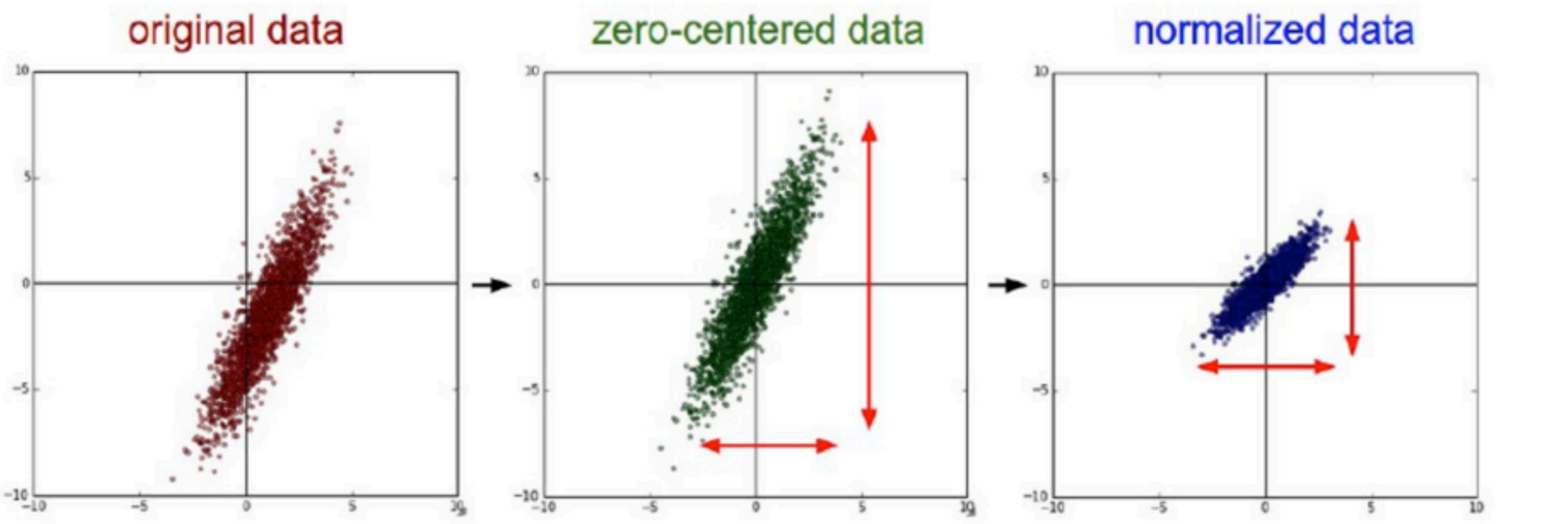
2 Layers Neural Net model

Learning algorithm

- Learning the parameters is a complex and time-comsuming task.
  - **Huge amount** of **data** is needed.
  - **Back progragation** method is used.
  - Great **packages** like Caffe, Theano or TensorFlow.
  - The make use of **GPU's** in order to optimize the weights.

Train a Network

Step 1: Get lot of data and preprocess it




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Train a Network

Step 2: Initializer the parameters

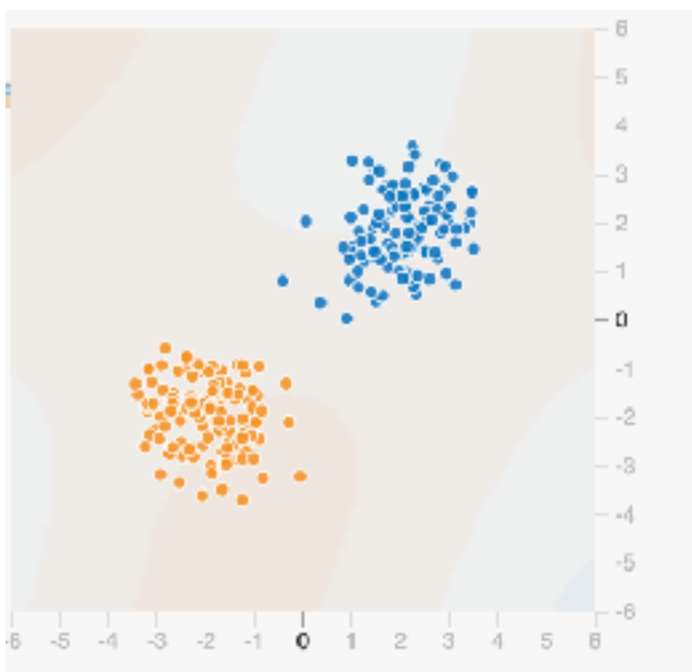
Set weights to small random numbers.
Set biases to zero.

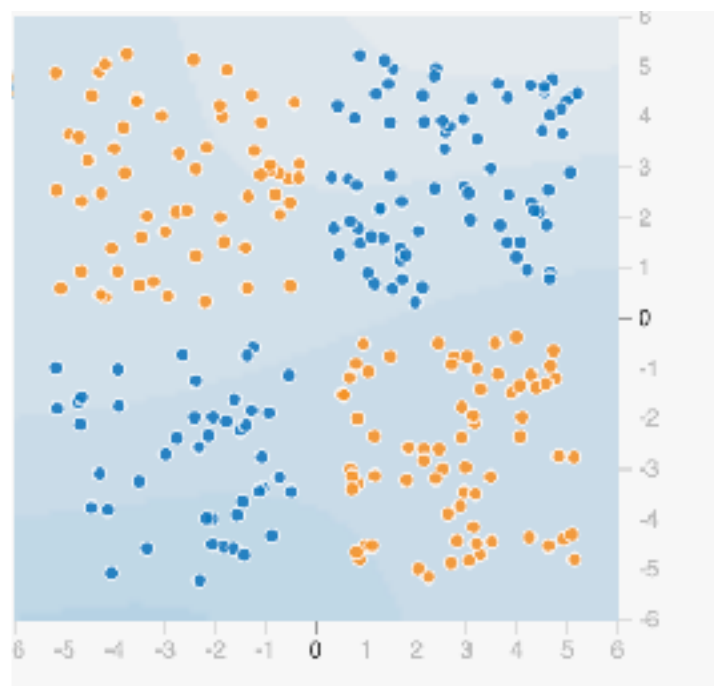
Train a Network

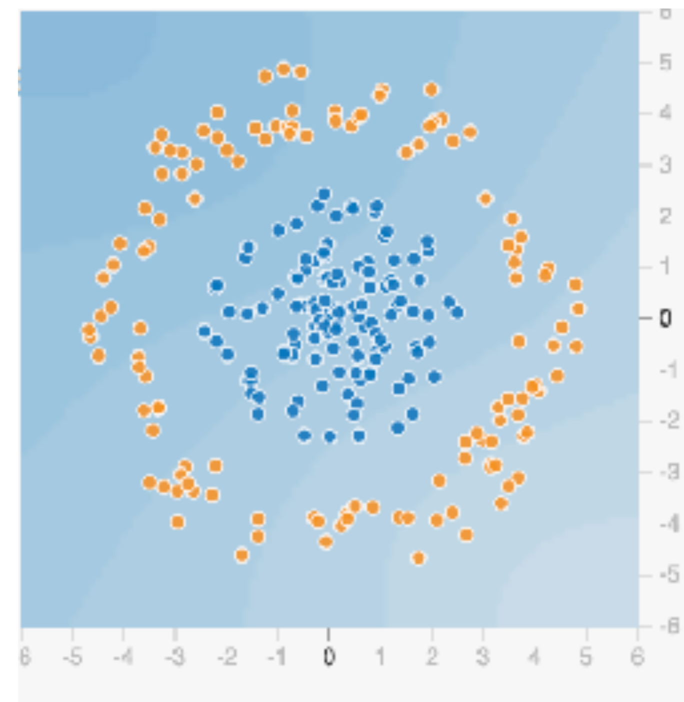
Step 3: Use Stochastic Gradient Descent

Train a Network

Step 4: Look for a GPU machine!

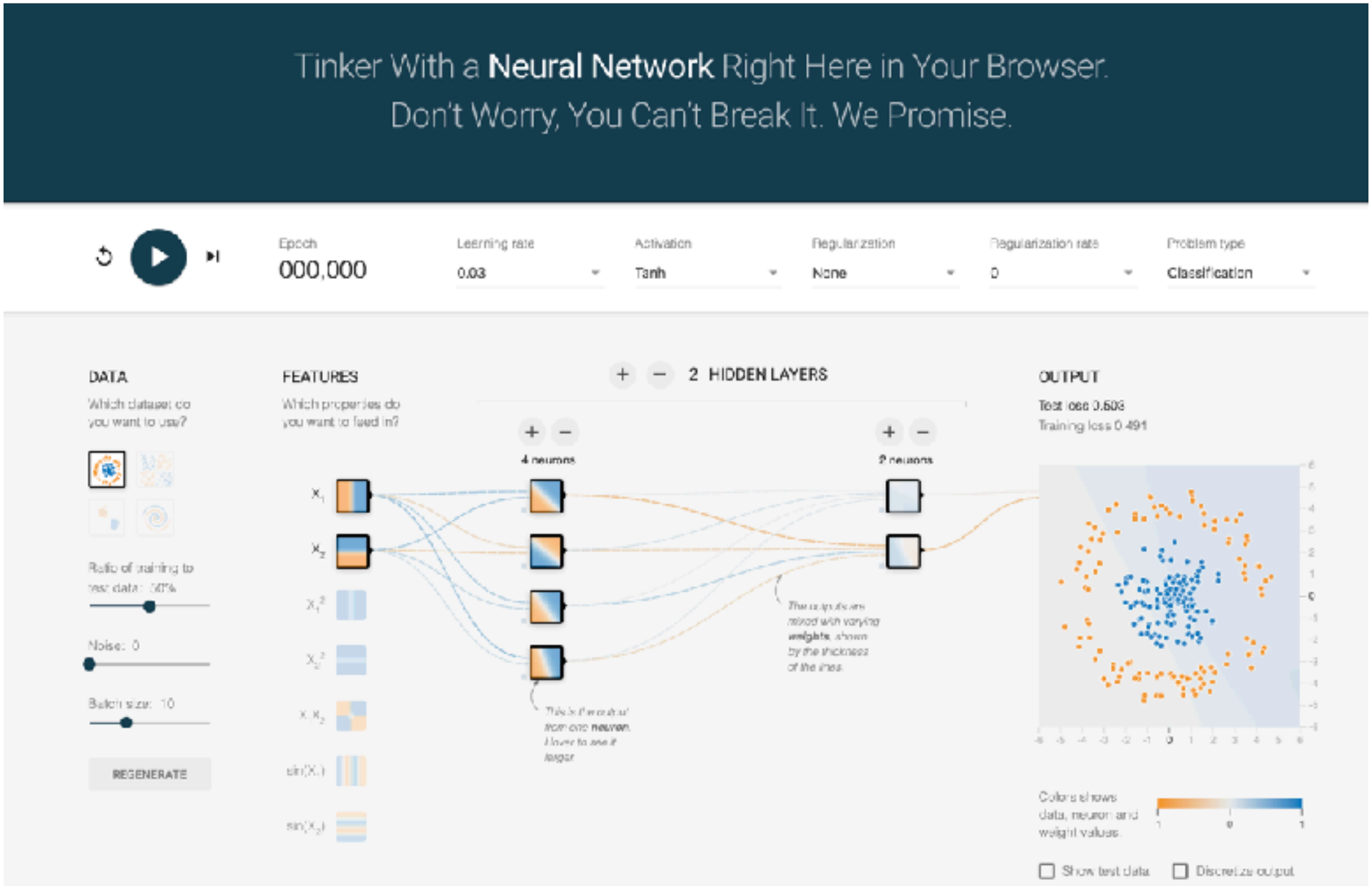






Let’s play a little bit! :)

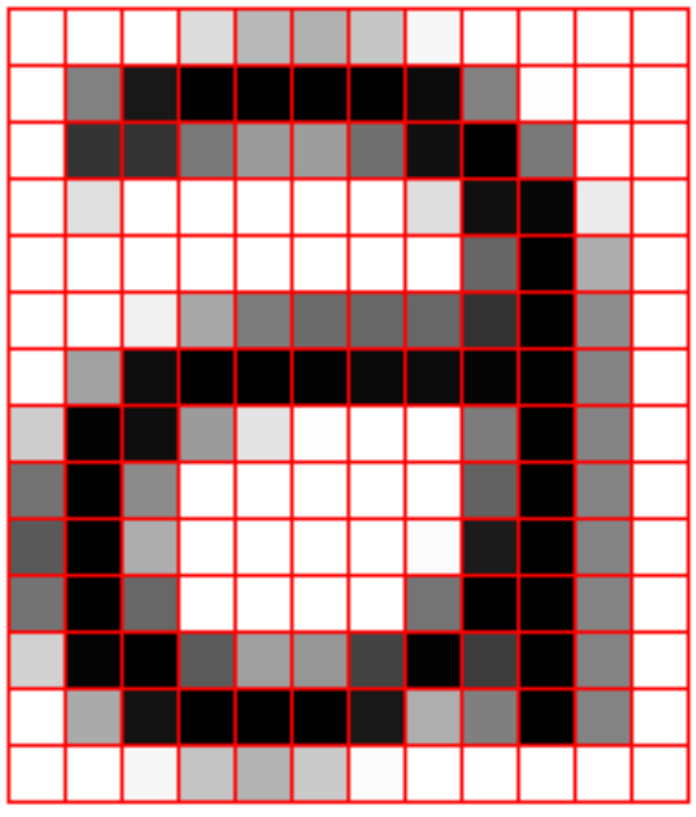
http://playground.tensorflow.org/



Deep Learning for Images

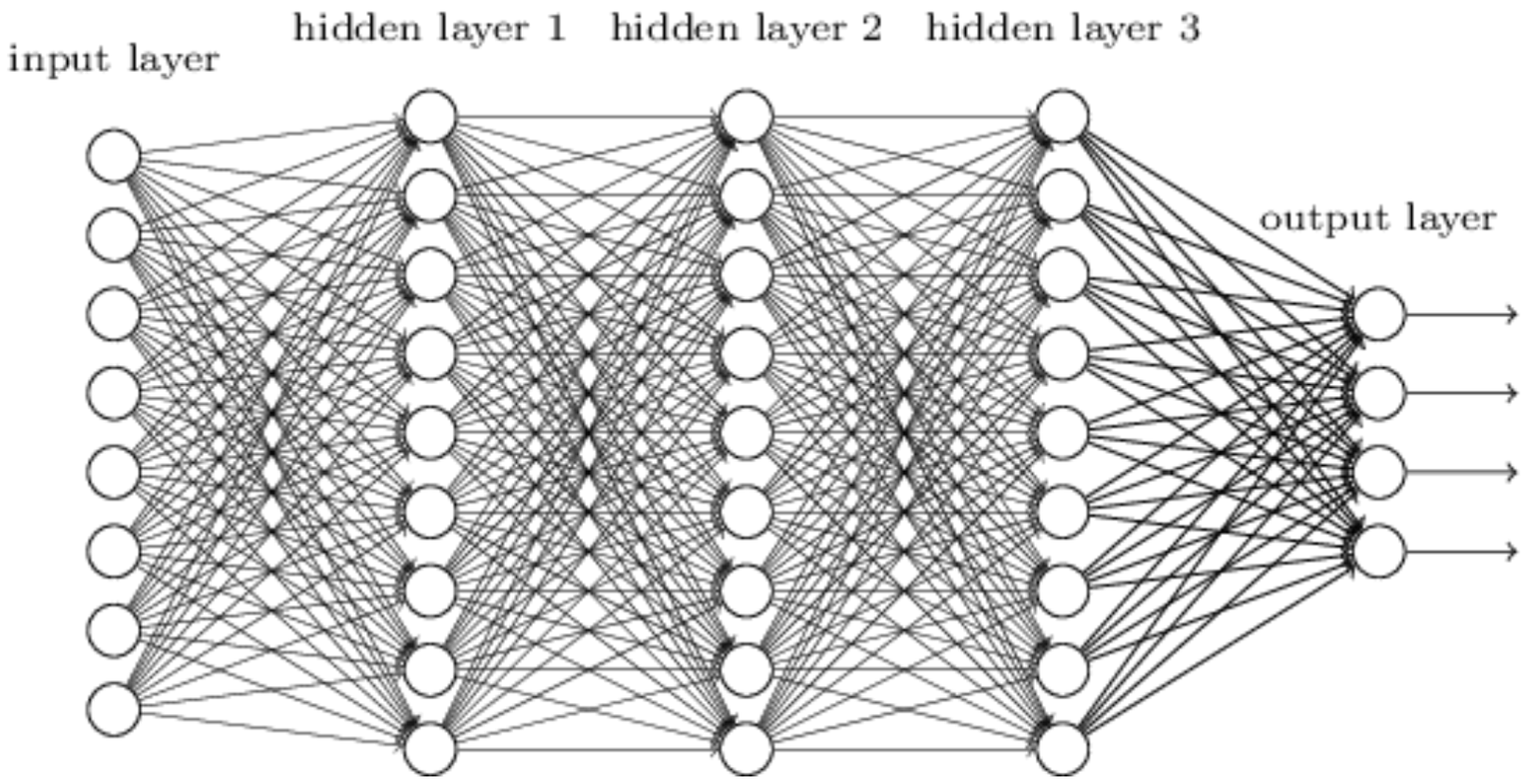
Neural Networks for Images?

An image is a matrix of size **`mxnxc`** pixels



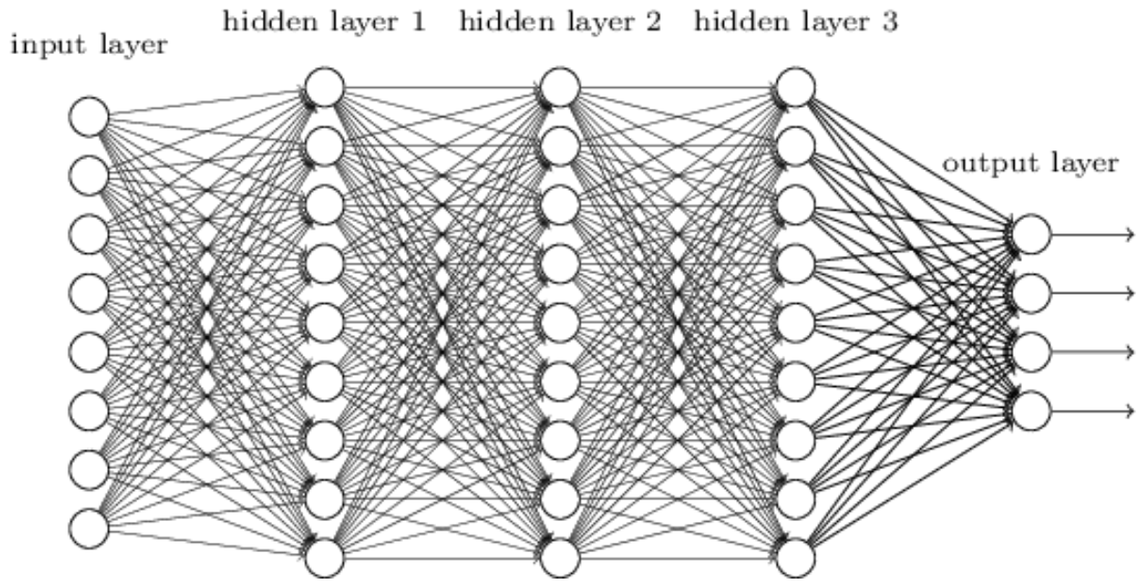
1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0

Neural Networks for Images?

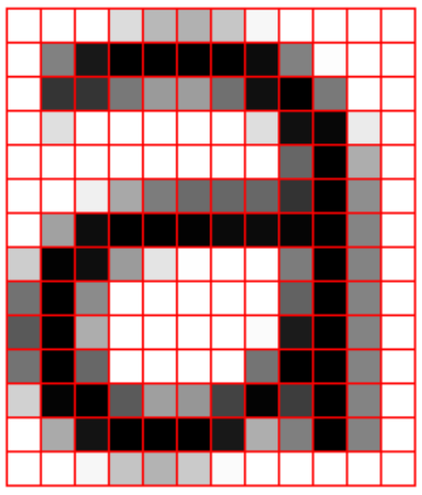


How many parameter does this MLP has?

Neural Networks for Images?



What happens if your data is an image like this one?



1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0

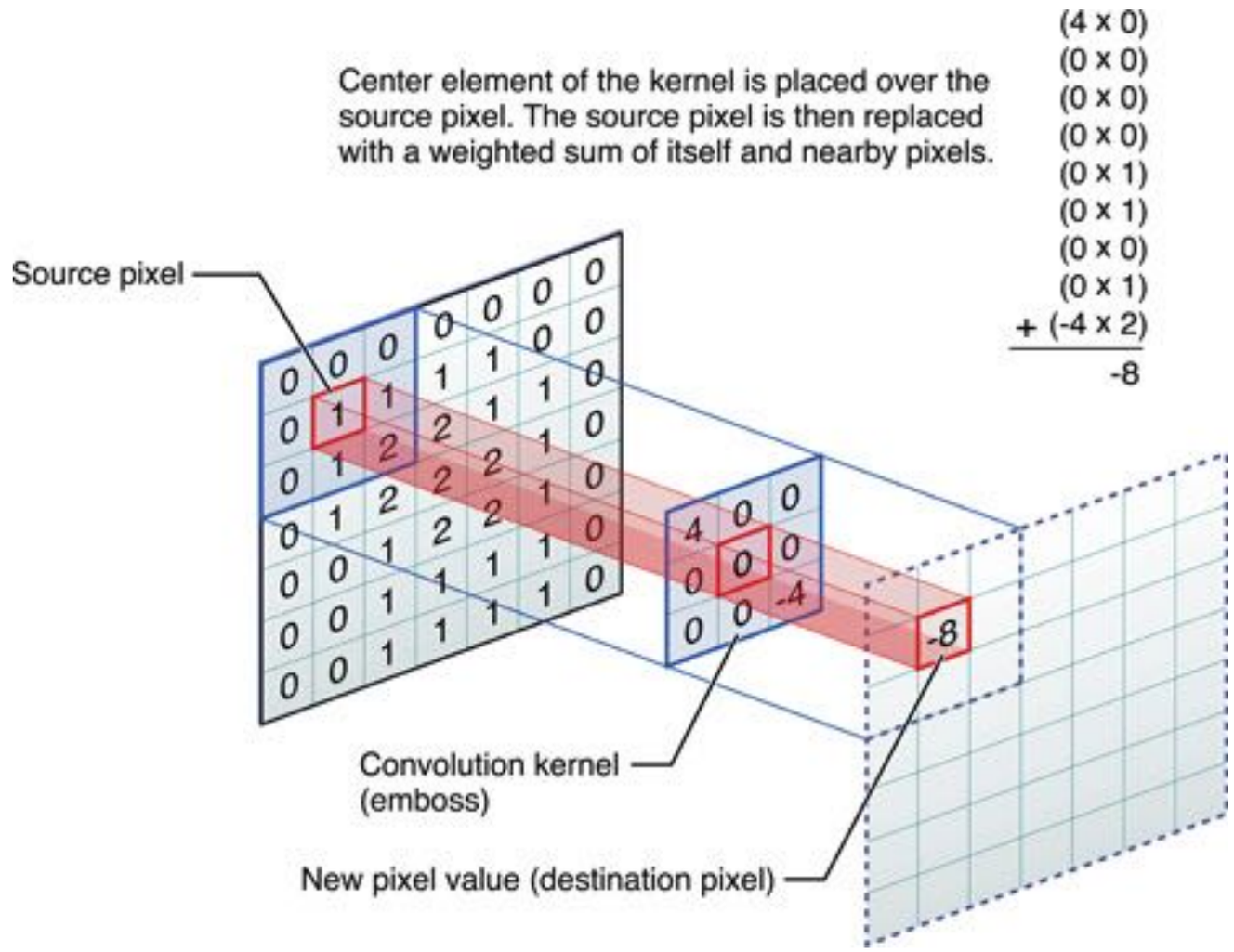
Neural Networks for Images?

But, in an image:

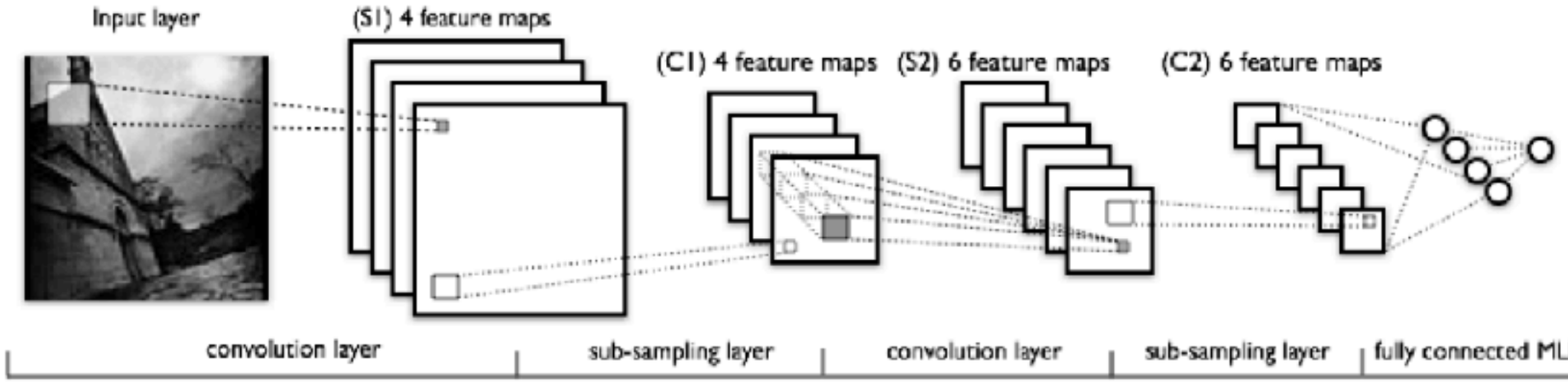
A dog can appear **anywhere** in the image!

Doesn’t matter where they appear,
they look similar anywhere!

Neural Networks for Images?



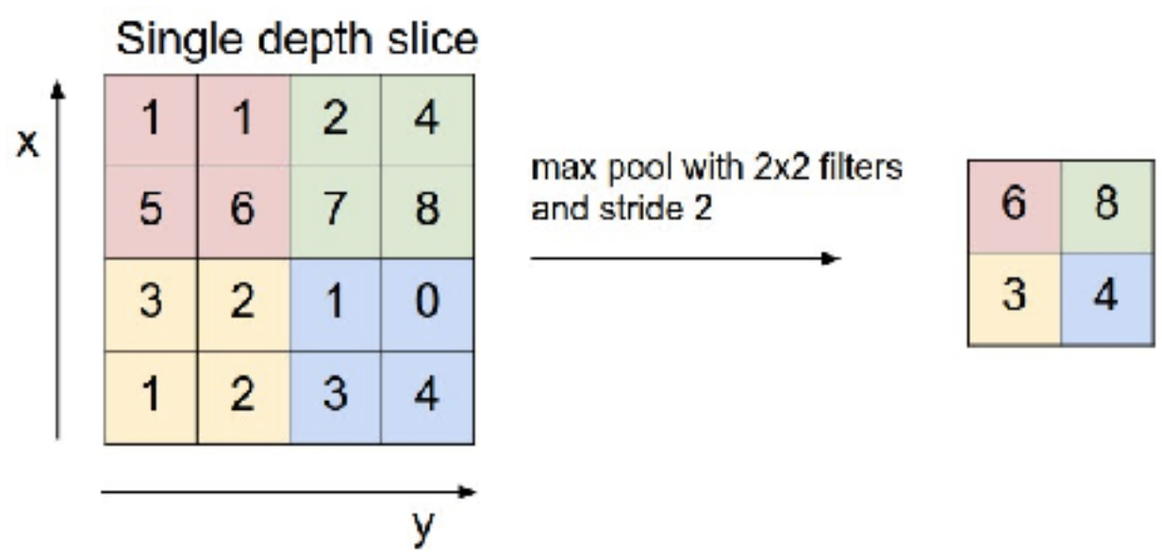
Input



Nice blog: http://cs231n.github.io/convolutional-networks/
Nice demo:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Max pooling

Pooling is a way of sub-sampling, i.e. reducing the dimension of the input (or at some hidden layer).
It is usually done after some of the convolutional layers



But it is also useful since it provides a form of translation **invariance**

Exercice

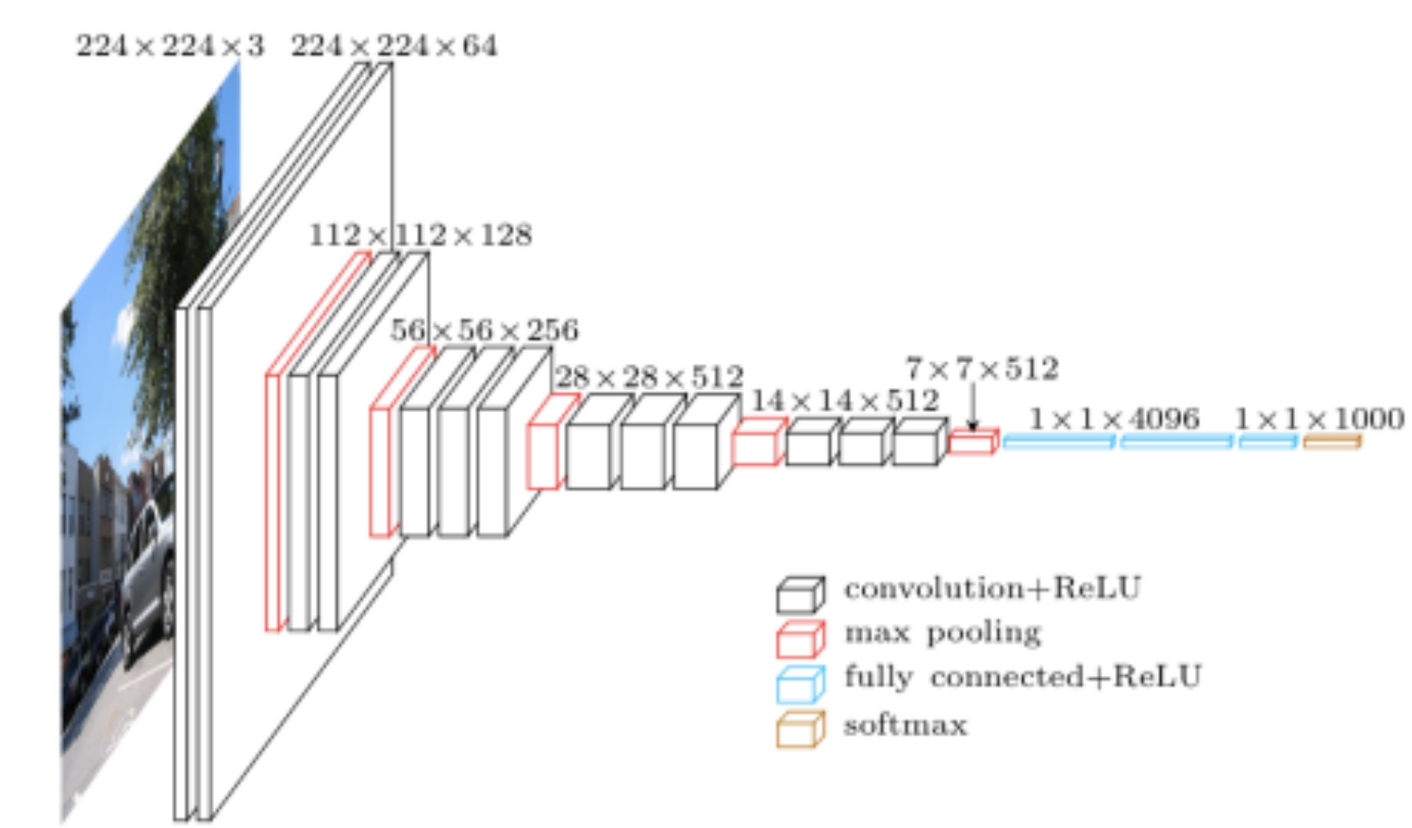
- How many parameter do the model has if:
  - it has 10 input values
  - 2 neurons in one hidden layer
  - 1 neuron in the output layer

Exercice

- How many parameter do the model has if:
  - it has 10 input values
  - 2 neurons in one hidden layer
  - 1 neuron in the output layer

- How many parameter do the model has if:
  - it has 10 input values
  - 25 neurons in the first hidden layer
  - 25 neurons in the second hidden layer
  - 1 neuron in the output layer

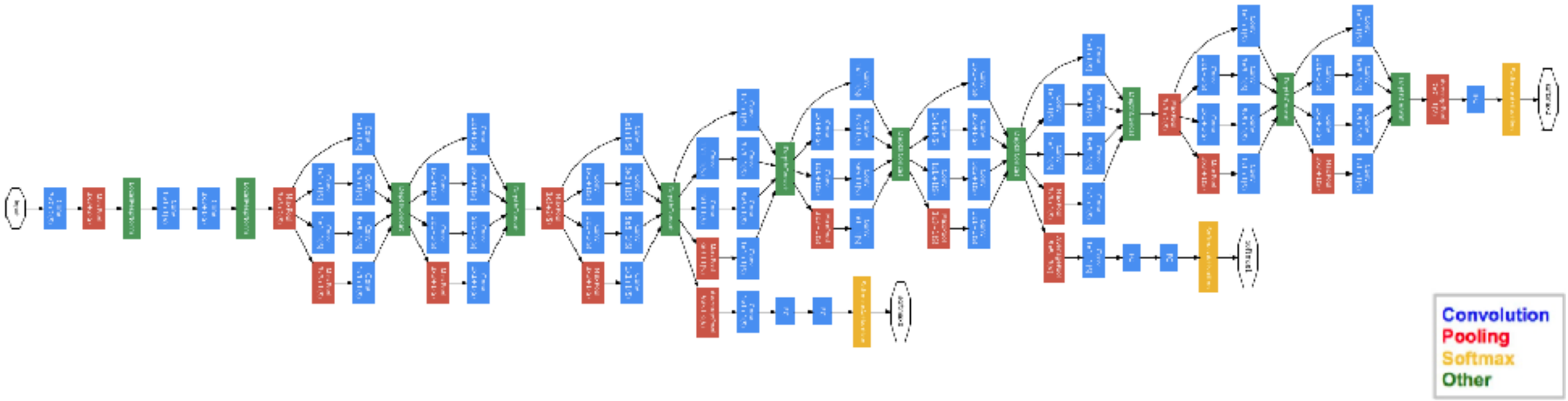
VGG Net


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax


WE NEED TO GO
DEEPER
troll-face.ru

GoogleNet

22 layers, but 12 times less parameters than AlexNet



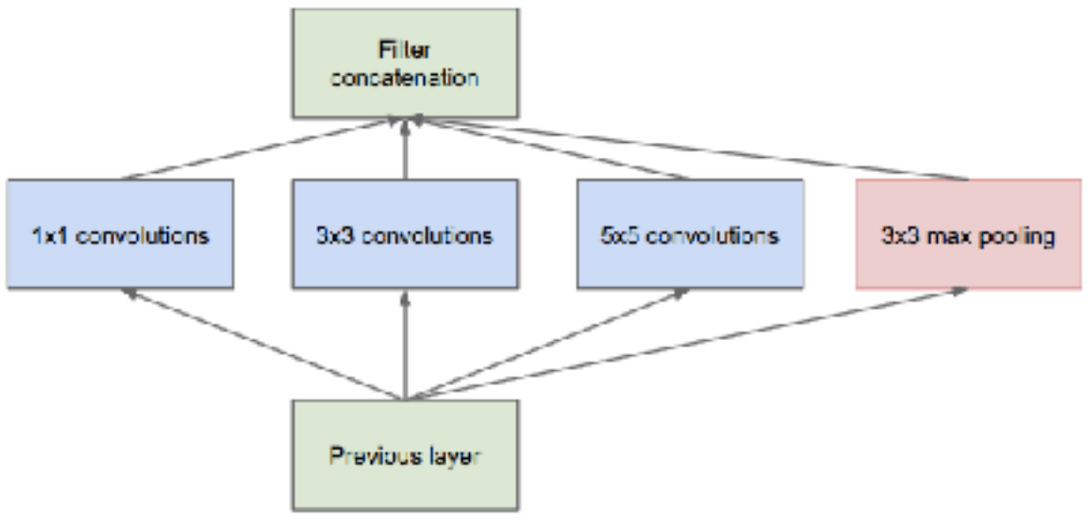
Inception Module:

Convolutional filters with different sizes can cover different clusters of information.

By finding the optimal local construction and repeating it spatially, they approximate the optimal sparse structure with dense components.

For convenience of computation, they use 1 x 1, 3 x 3 and 5 x 5 filters + pooling.

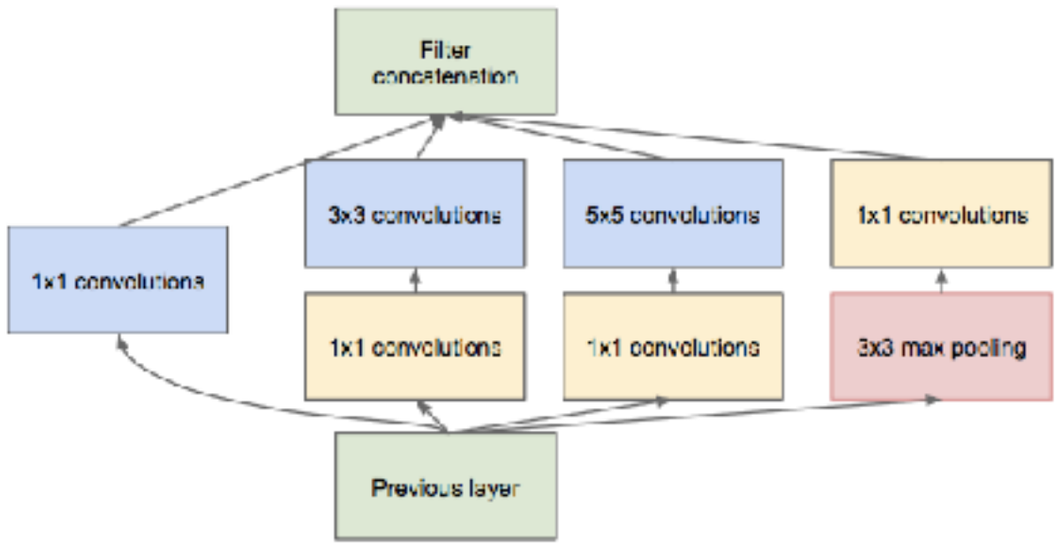
Together these made up the naive Inception module.



Inception Module:
Smart Version

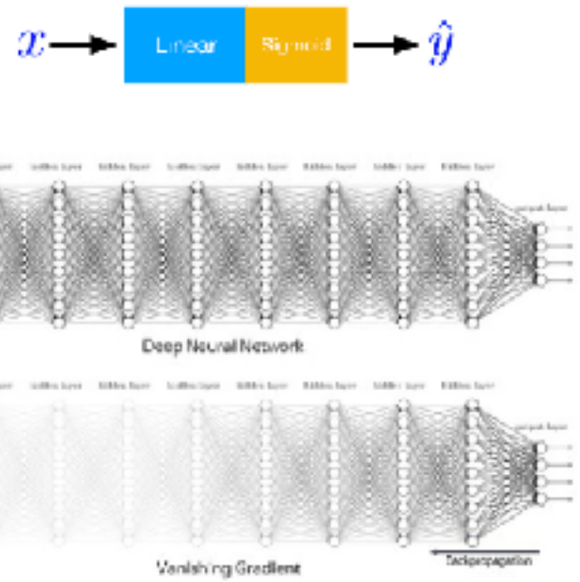
Stacking these inception modules on top of each would lead to an exploding number of outputs

Solution: inspired by "Network in Network" add 1x1 convolutions for dimensionality reduction

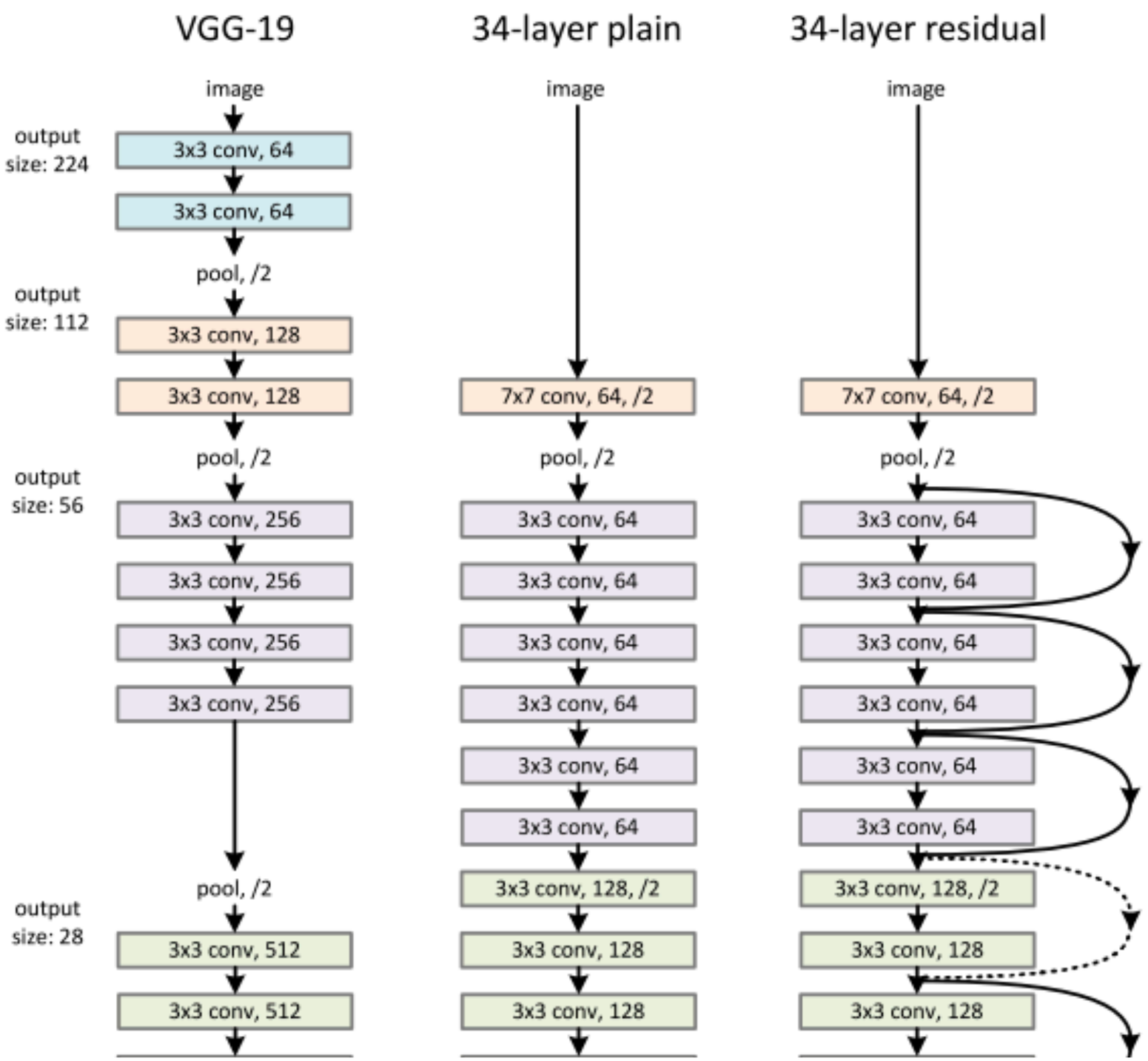



WE NEED TO GO
DEEPER
troll-face.ru

Not everything is that simple:
Vanishing Gradient Problem



As the gradient keeps flowing backward to the initial layers, this value keeps getting multiplied by each local gradient. Hence, the gradient becomes smaller and smaller, making the updates to the initial layers very small, increasing the training time considerably.


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128

Training a CNN

- Backpropagation + stochastic gradient descent with momentum
- Dropout
- Data Augmetation
- Batch Normalization
- Initialization
    - Transfer Learning

Reducing Overfitting

Data Augmentation

60 million parameters, 650,000 neurons
-> overfits a lot

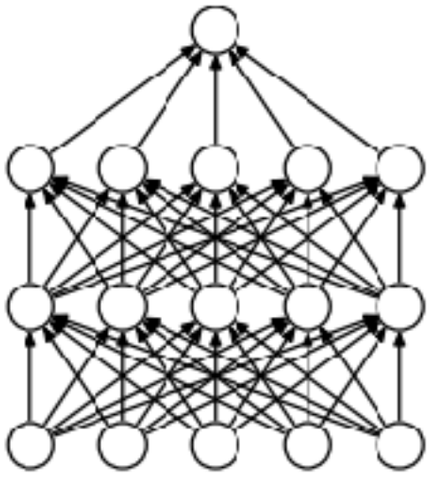
Crop 224x224 patches (and their horizontal reflections)

Data Augmentation at test
average the predictions on the 10 patches.

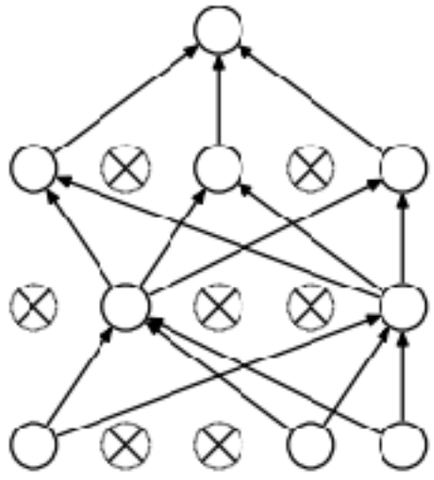
Reducing Overfitting

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Journal of Machine Learning Research 15 (2014) 1929-1958

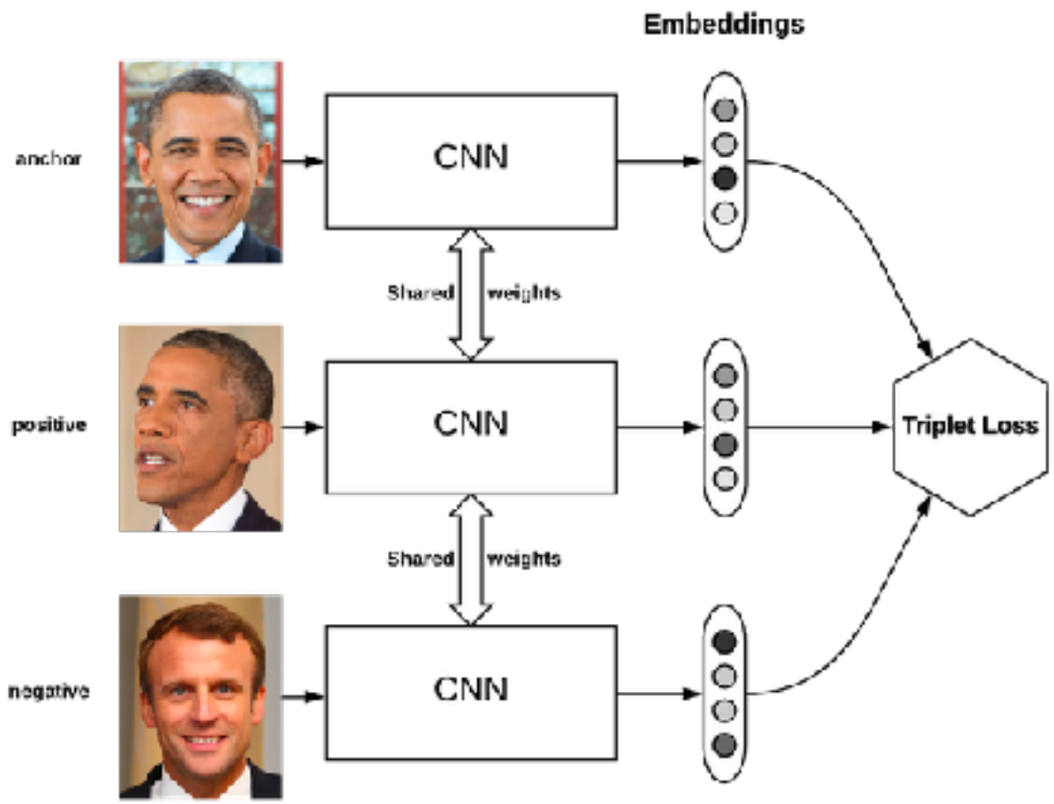


(a) Standard Neural Net



(b) After applying dropout.

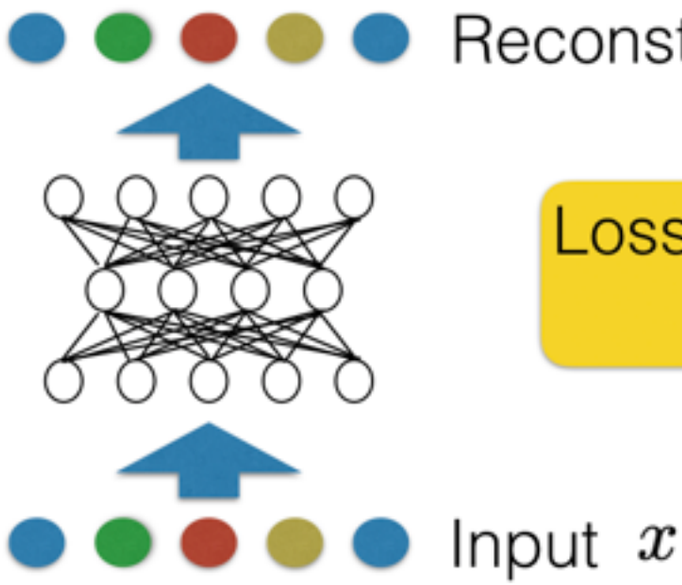
Other tasks:
Person Re-Identification



$$Loss = \sum_{i=1}^{N} \left[\| f_i^a - f_i^p \|_2^2 - \| f_i^a - f_i^n \|_2^2 + \alpha \right]_+$$

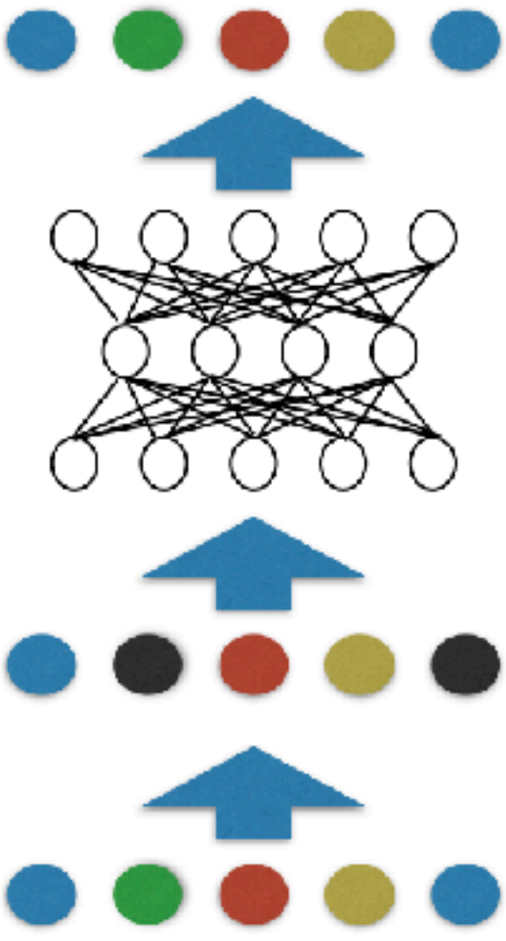
Autoencoders

- From high dimensinonal data to lower dimensional space



Loss $\mathcal{L}(\tilde{x}, x) = \sum(\tilde{x} - x)^2$

Denoising Autoencoders

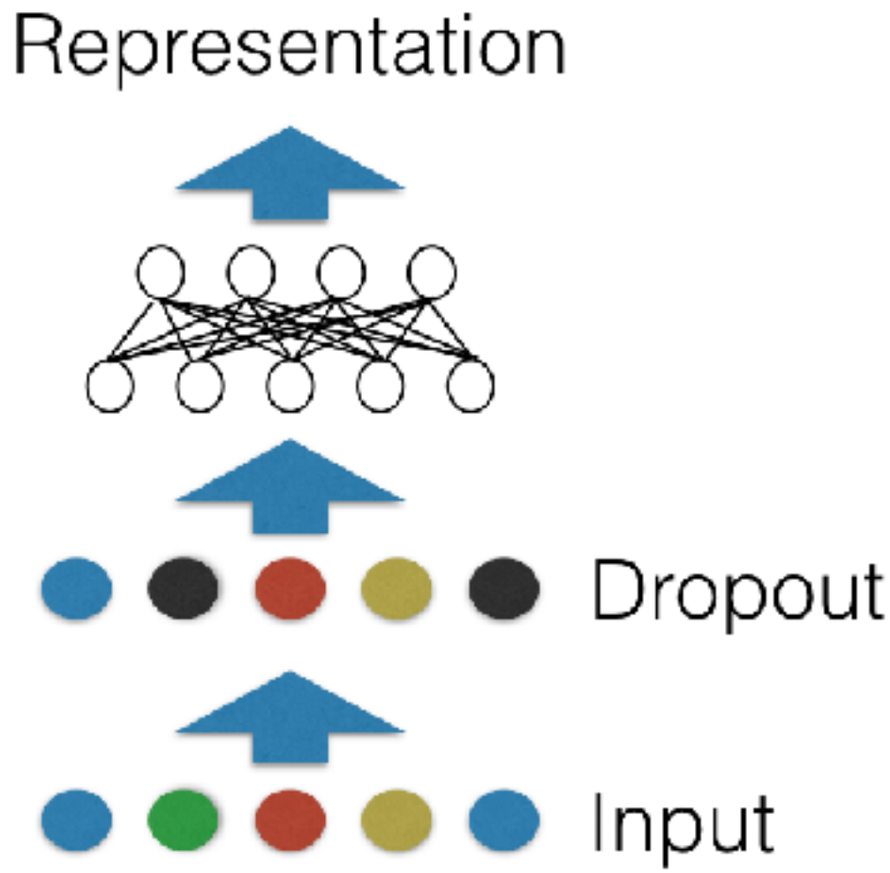


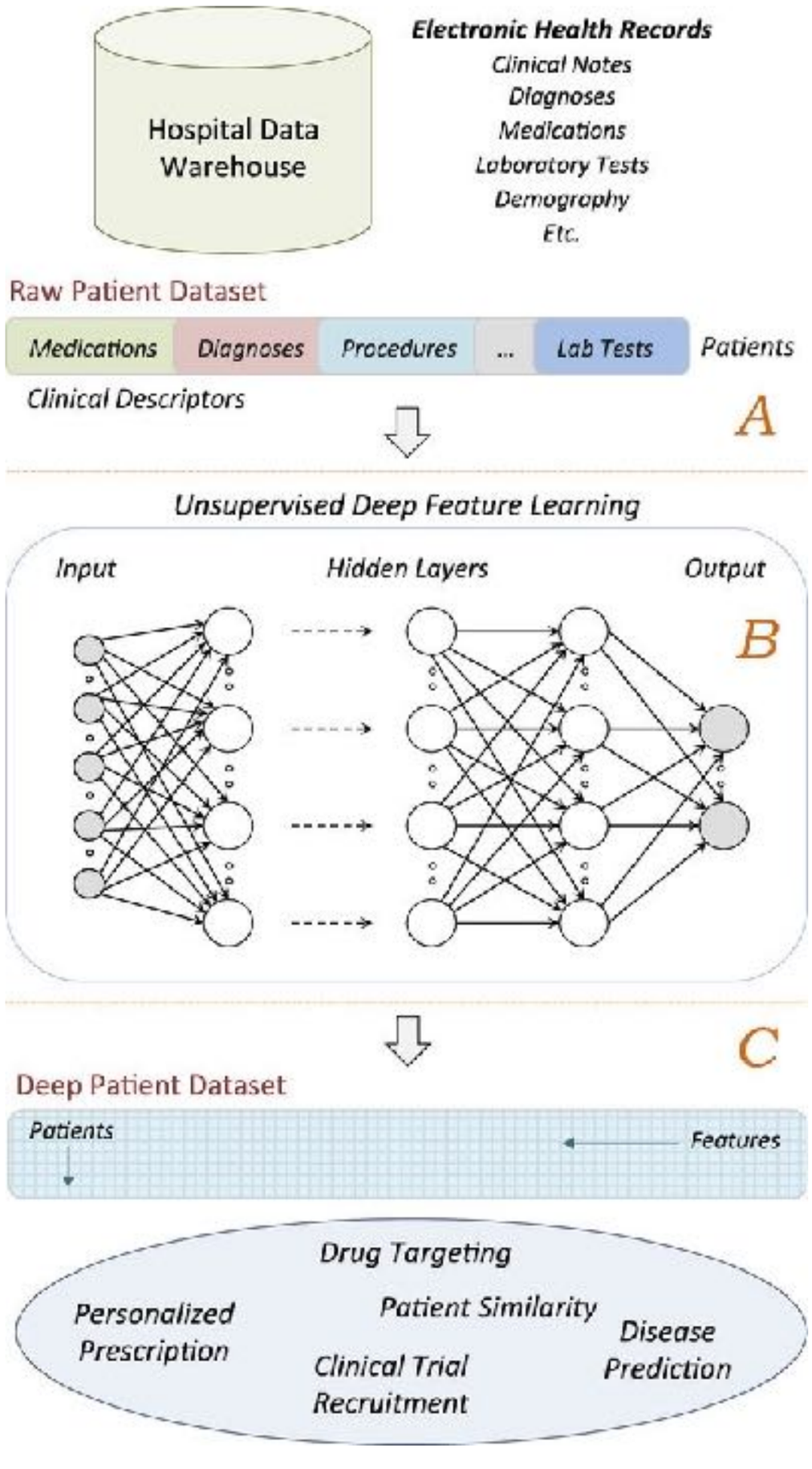
Reconstruction

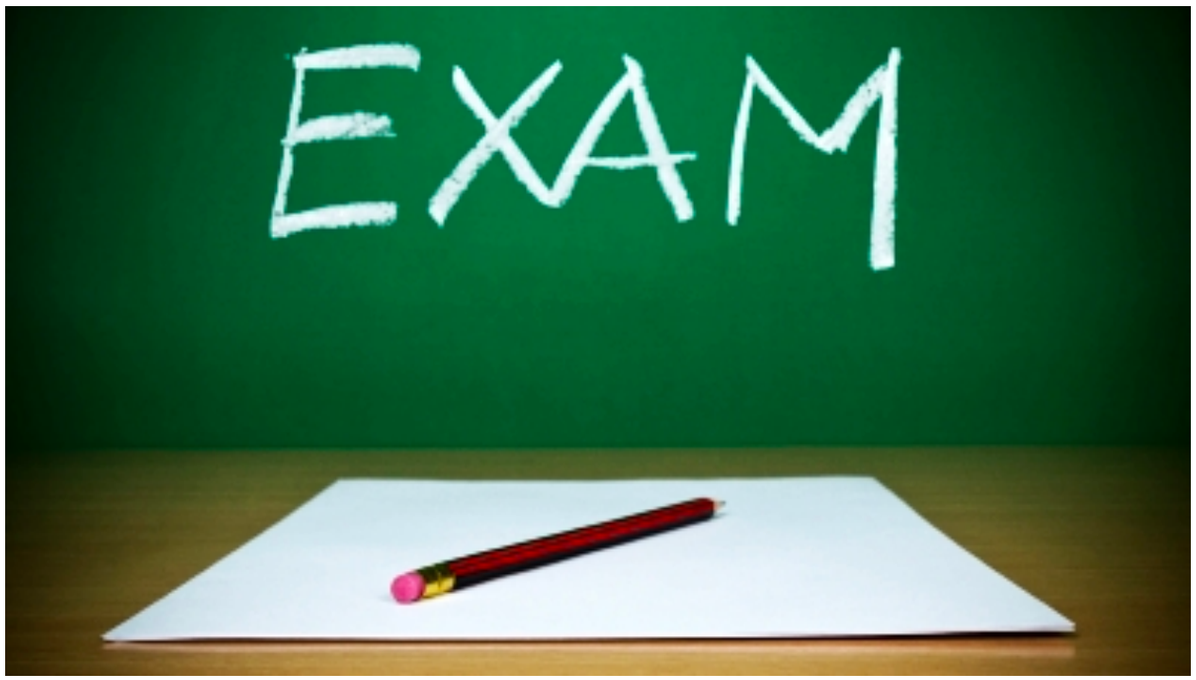
Noise: dropout input values

Input

Uses: Pretraining and transfer learning

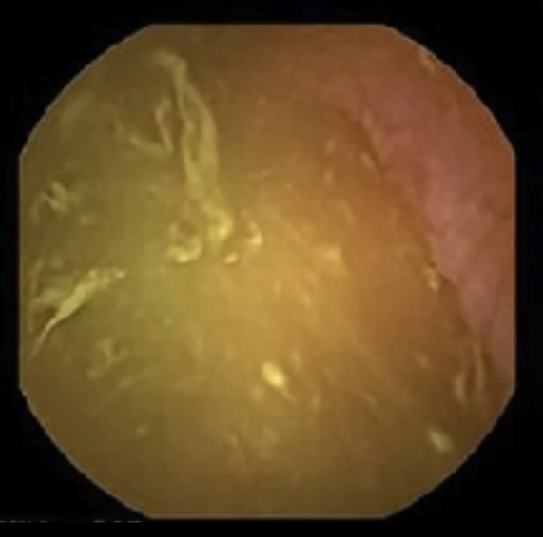



Hospital Data Warehouse
Electronic Health Records
Clinical Notes
Diagnoses
Medications
Laboratory Tests
Demography
Etc.
Raw Patient Dataset
Medications
Diagnoses
Procedures
...
Lab Tests
Patients
Clinical Descriptors
A
Unsupervised Deep Feature Learning
Input
Hidden Layers
Output
B
C
Deep Patient Dataset
Patients
Features
Drug Targeting
Personalized Prescription
Patient Similarity
Clinical Trial Recruitment
Disease Prediction


EXAM

AI system to detect polyps, bleeding, inflammatory lesions and intestinal content





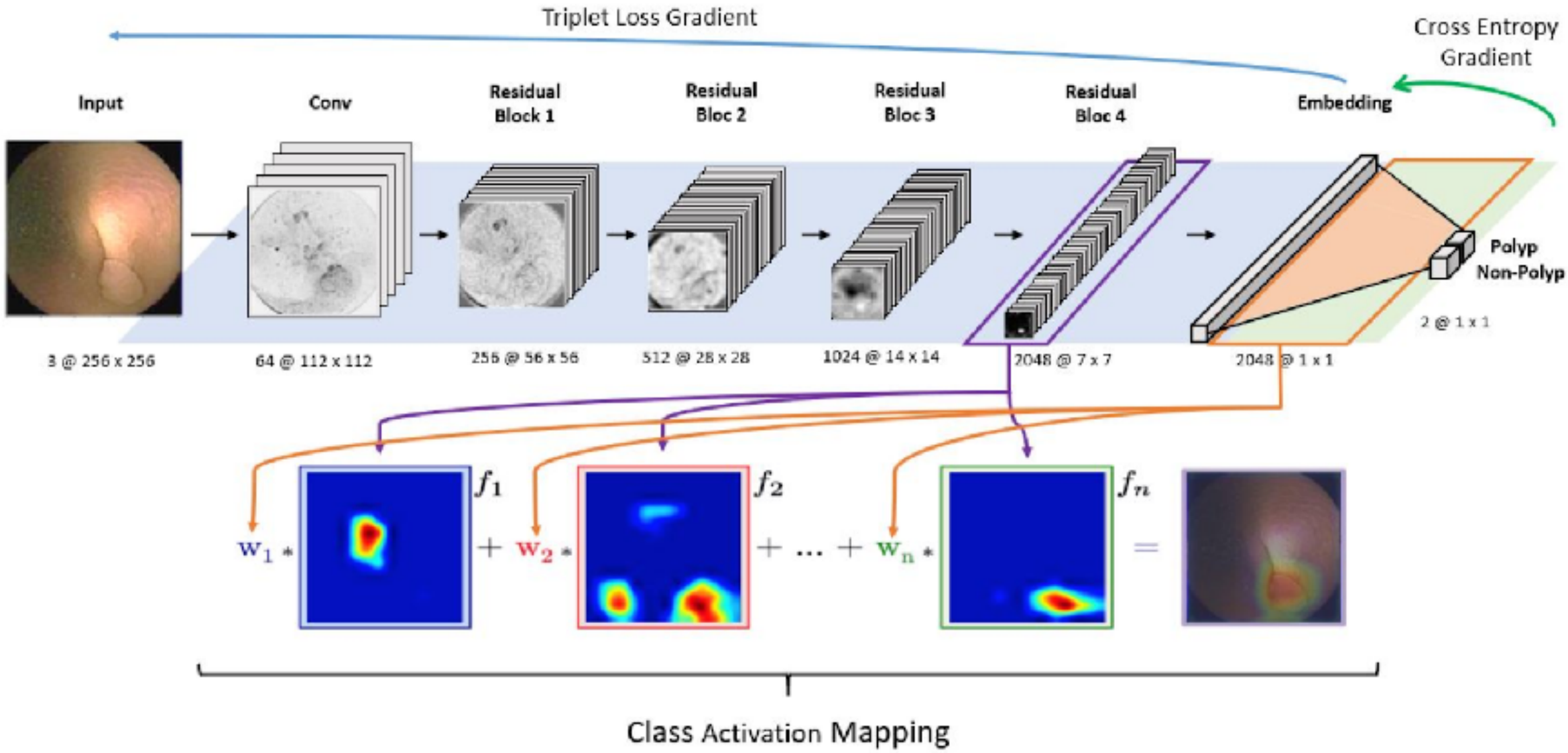


Fig. 3: Overview of the proposed CNN structure. The upper part of the scheme appear the ResNet architecture with our methodology applied in it. The background colour reflect the layers affected by each one of the gradient generated by the main losses. The lower part of the figure show how the class activation mapping is build.

Recurrent Neural Networks

Classical neural networks, including convolutional ones, suffer from **two severe limitations**:

- **They only accept a fixed-sized vector as input and produce a fixed-sized vector as output.**
- **They do not consider the sequential nature of some data (language, video frames, time series, etc.)**

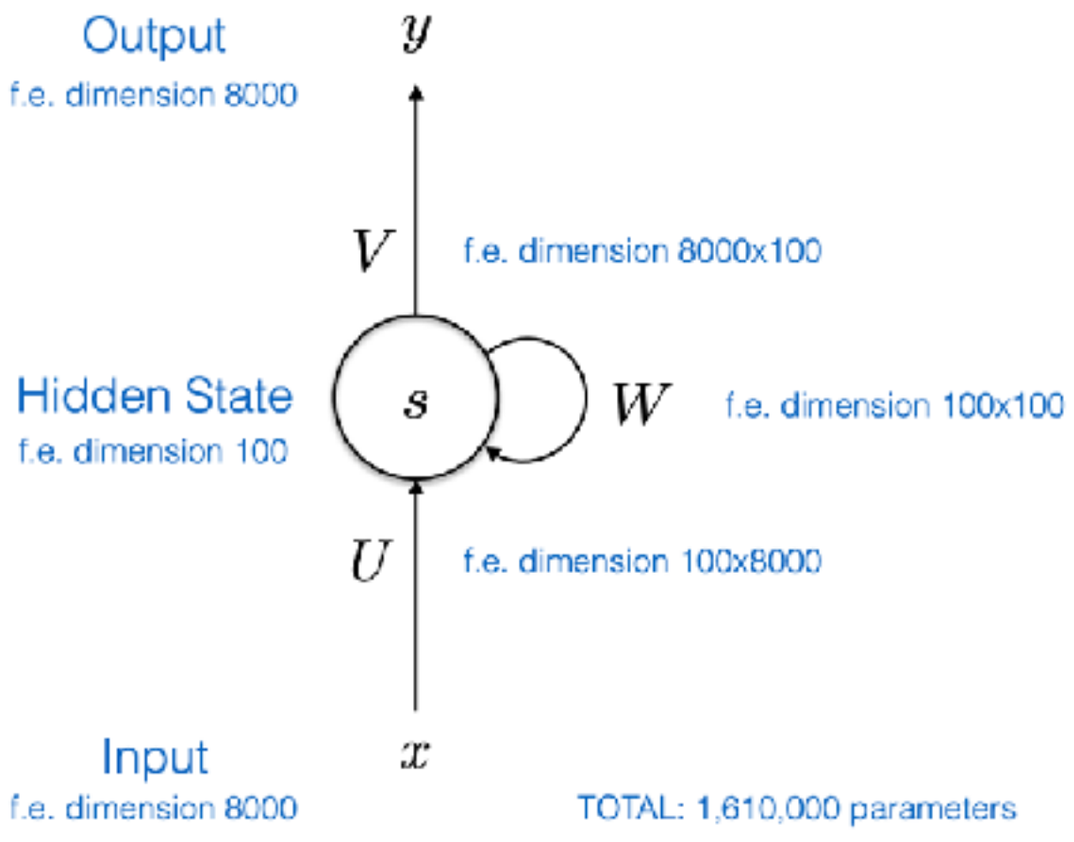
Recurrent neural networks (RNN) overcome these limitations by allowing to operate over sequences of vectors (in the input, in the output, or both).

RNN have shown **success** in:

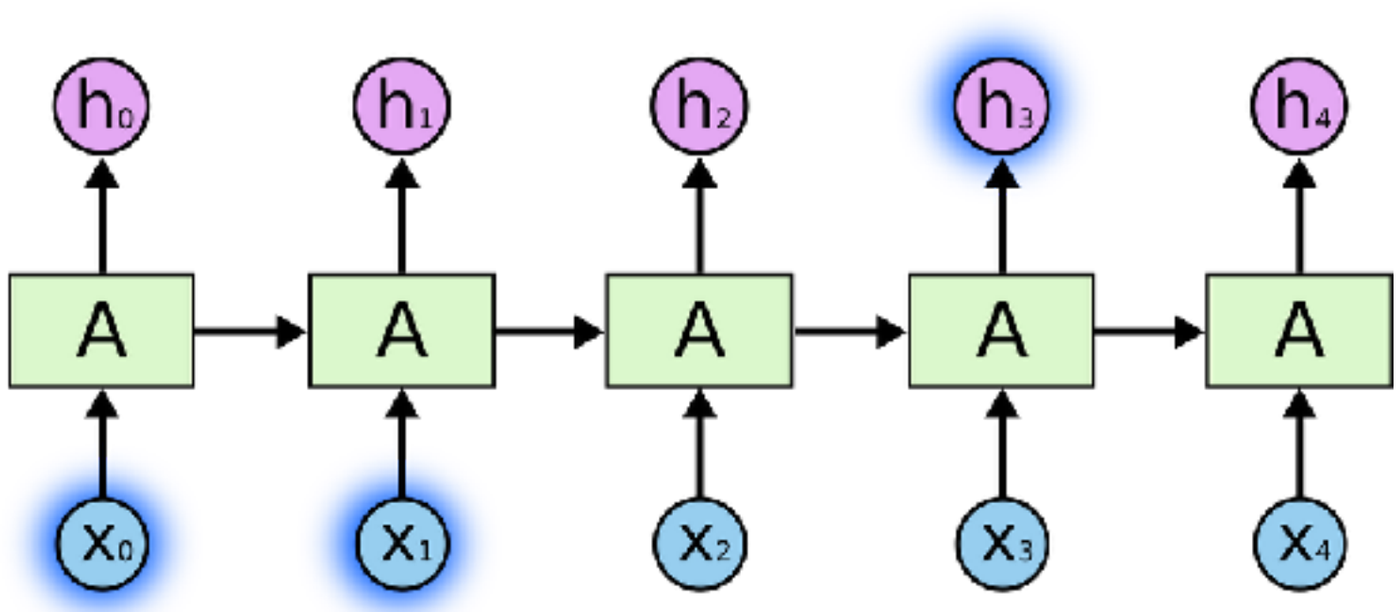
- Language modeling and generation.
- Machine Translation.
- Speech Recognition.
- Image Description.
- Question Answering.
- Etc.

the clouds are in the ***?***

Vanilla Recurrent Neural Network



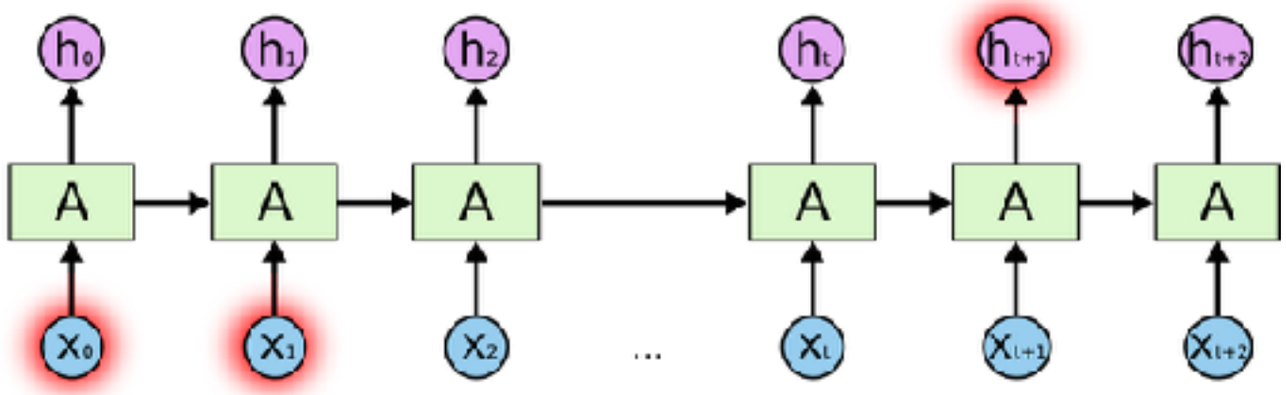
the clouds are in the ***sky***



I grew up in France….. … …

… … .. … …

…. … … … I speak fluent ***?***.

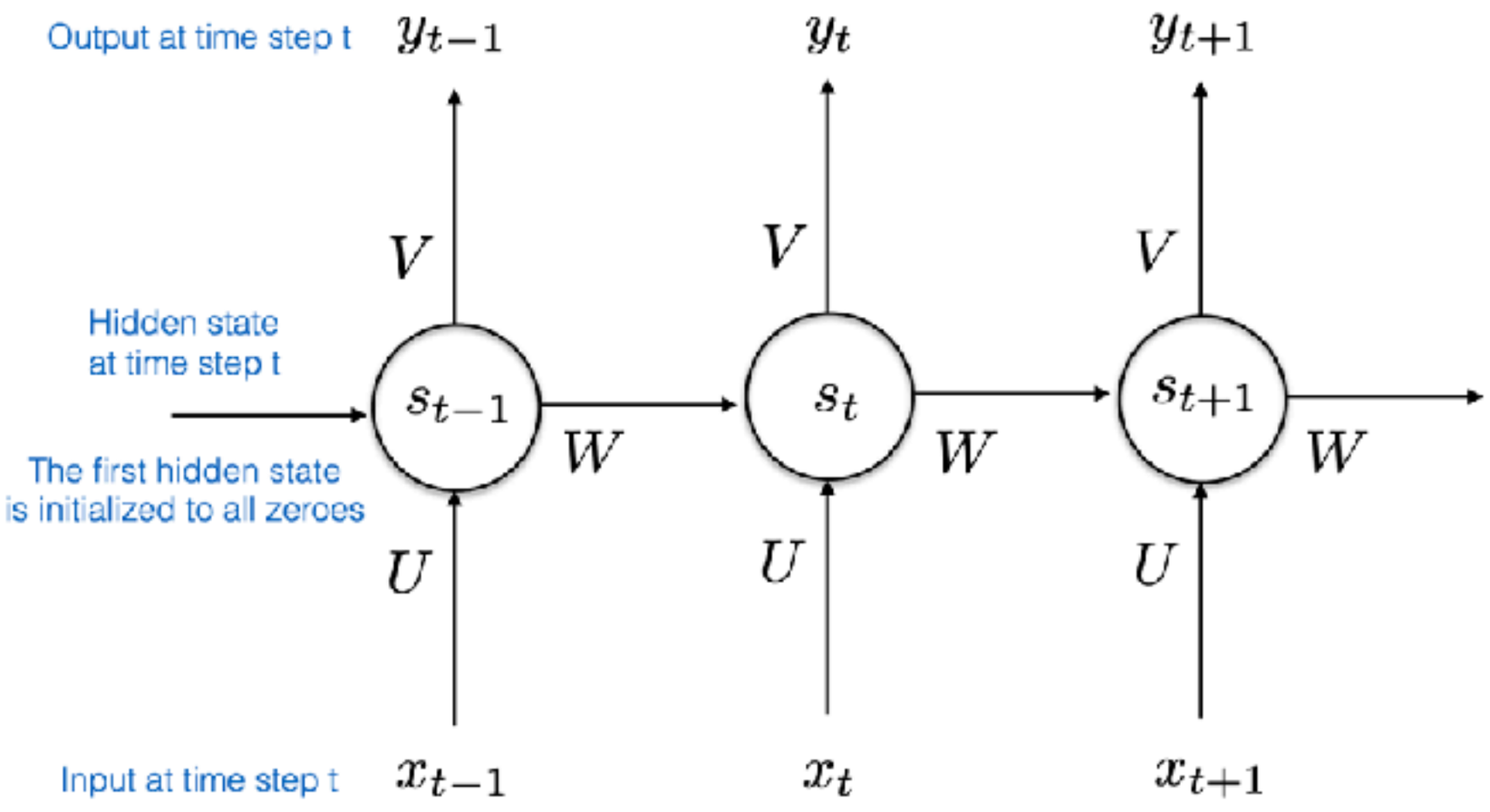


Unrolling in time of a RNN

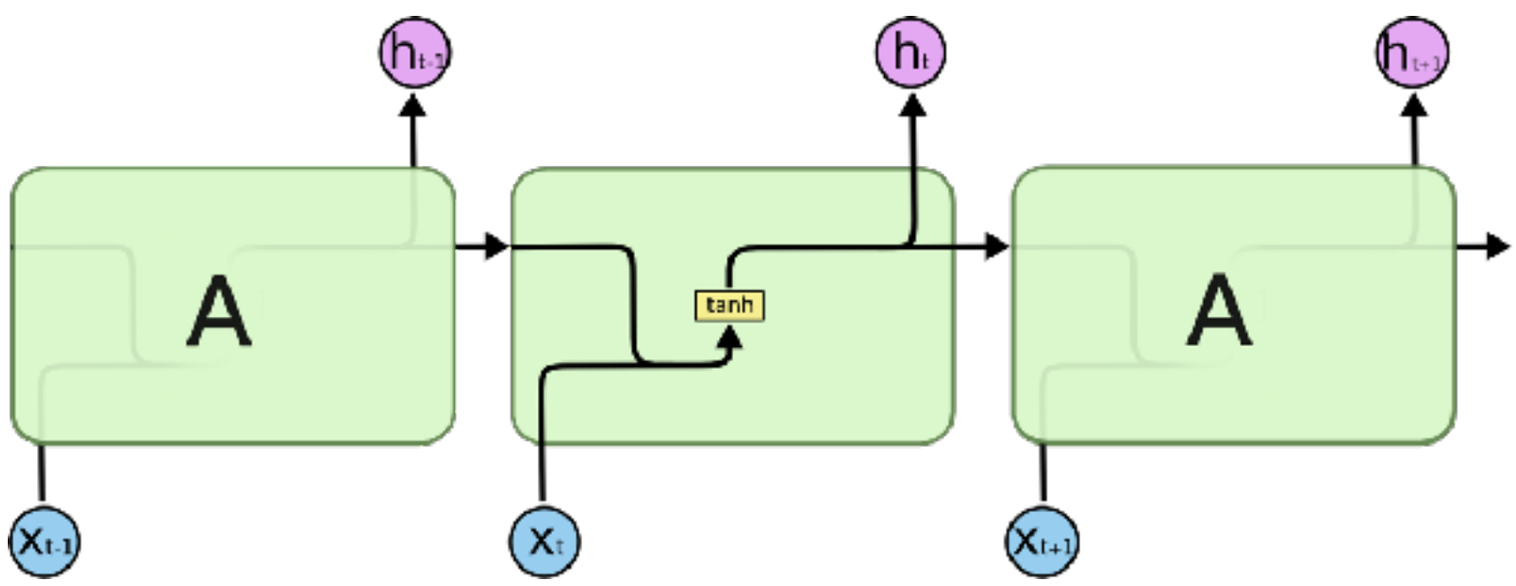
By unrolling we mean that we write out the network for the complete sequence.

Basic equations of the RNN

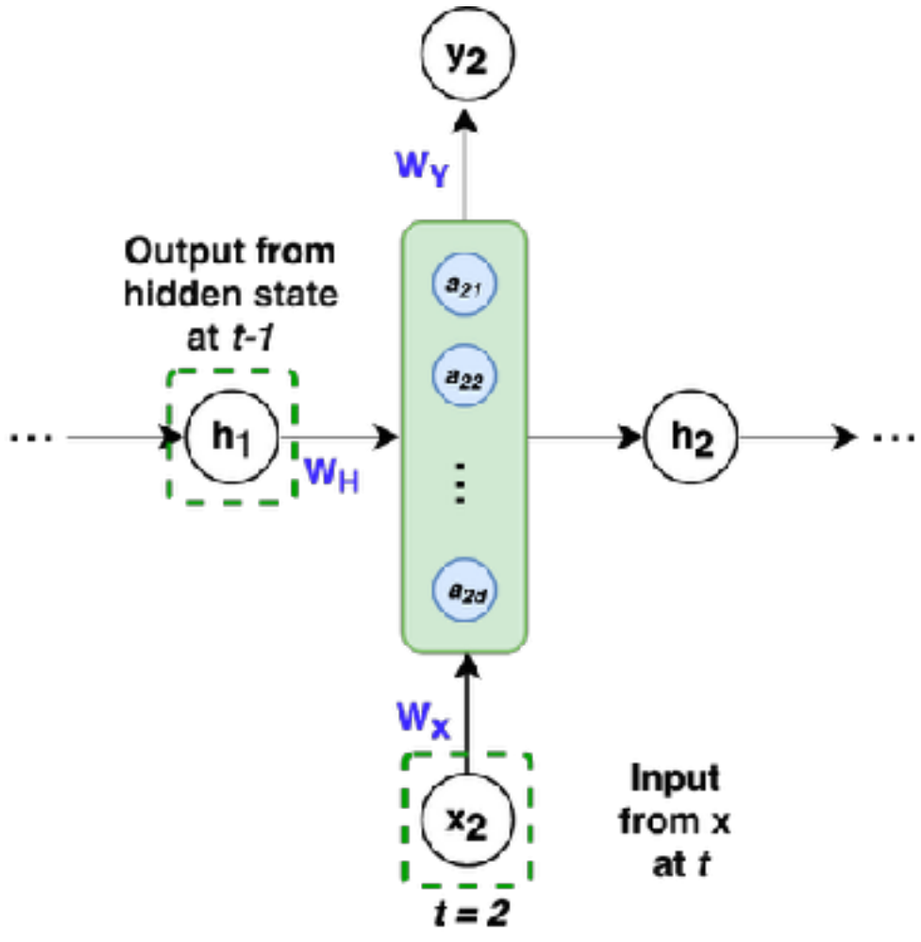
$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$y_t = \mathrm{softmax}(Vs_t)$$



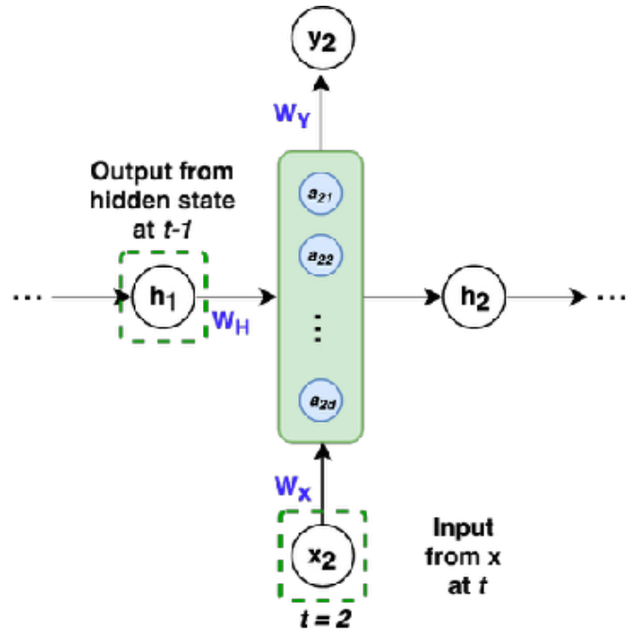
Vanilla Recurrent Neural Network



Vanilla Recurrent Neural Network



Vanilla Recurrent Neural Network



$$\mathbf{d} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{o} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{g} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{s} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

One-Hot Encoding of the word "dogs"

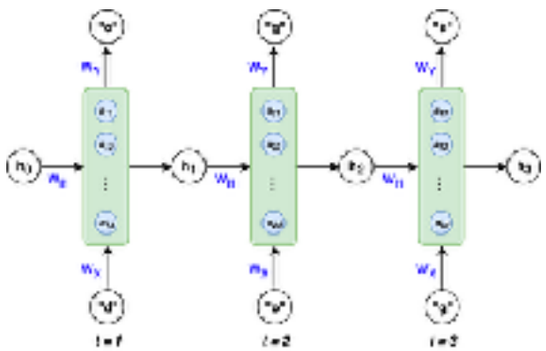
$$a_t = W_H h_{t-1} + W_X X_t \quad \text{Hidden Nodes}$$

Activation Function

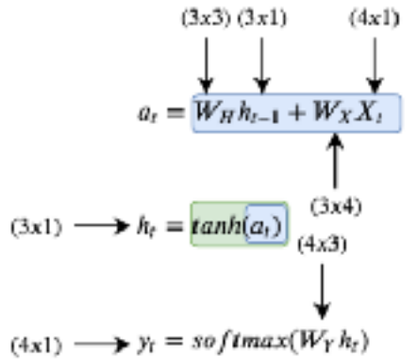
Output From Hidden State → $h_t = tanh(a_t)$

Hidden State

Prediction at time t → $y_t = softmax(W_Y h_t)$



If we use 3 hidden nodes çin our RNN *(d=3)*



At t=1

$$a_1 = \begin{pmatrix} W_{H,11} & W_{H12} & W_{H,13} \\ W_{H,21} & W_{H,22} & W_{H,23} \\ W_{H,31} & W_{H,32} & W_{H,33} \end{pmatrix} \begin{pmatrix} h_{0,1} \\ h_{0,2} \\ h_{0,3} \end{pmatrix} + \begin{pmatrix} W_{X,11} & W_{X,12} & W_{X,13} & W_{X,14} \\ W_{X,21} & W_{X,22} & W_{X,23} & W_{X,24} \\ W_{X,31} & W_{X,32} & W_{X,33} & W_{X,34} \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \end{pmatrix} = \begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}$$

$$a_1 = \begin{pmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.2 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \end{pmatrix}) = \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}$$

$$h_1 = tanh(\begin{pmatrix} 0.6 \\ 0.2 \\ 0.9 \end{pmatrix}) = \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} W_{Y,11} & W_{Y,12} & W_{Y,13} \\ W_{Y,21} & W_{Y,22} & W_{Y,23} \\ W_{Y,31} & W_{Y,32} & W_{Y,33} \\ W_{Y,41} & W_{Y,42} & W_{Y,43} \end{pmatrix} \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \end{pmatrix}) = \begin{pmatrix} y_{1,1} \\ y_{1,2} \\ y_{1,3} \\ y_{1,4} \end{pmatrix}$$

$$y_1 = softmax(\begin{pmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{pmatrix} \begin{pmatrix} 0.54 \\ 0.20 \\ 0.72 \end{pmatrix}) = \begin{pmatrix} 0.32 \\ 0.21 \\ 0.24 \\ 0.22 \end{pmatrix}$$

The power of RNN