
Introductory Seminar on Artificial Intelligence and Machine Learning

Emanuele Ledda, Cagliari Digital Lab 2024 - Incontro 3



Deep Learning

Artificial Neural Networks

A Long History



A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS

University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

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, with the addition of more complicated logical means for net containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

1. Introduction. Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from $<1 \text{ ms}^{-1}$ in thin axons, which are usually short, to $>150 \text{ ms}^{-1}$ in thick axons which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequal in remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this depends upon reciprocity of individual synapses or merely upon prevalent anatomical configurations. To suppose the latter requires no hypothesis *ad hoc* and explains known exceptions, but any assumption as to cause is compatible with the calculus to come. No case is known in which excitation through a single synapse has elicited a nervous impulse in any neuron, whereas an neuron may be excited by impulses arriving at a sufficient number of neighboring synapses within the period of latent addition, which last $<0.25 \text{ ms}$. Observed temporal summation of impulses at greater intervals

* Reprinted from the *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133 (1943).

1940

PROCEEDINGS OF THE IRE

November

What the Frog's Eye Tells the Frog's Brain*

J. Y. LETTVIN†, H. R. MATORANA‡, W. S. MCCULLOCH§, SENIOR MEMBER, IRE,
AND W. H. PITTS||

Summary—In this paper, we analyze the activity of single fibers in the optic nerve of a frog. Our method is to find what sort of stimulus causes the largest activity in one nerve fiber and then what is the smallest stimulus that produces a response. We find that the response to a large stimulus is not due to the light intensity itself but rather the pattern of local variation of intensity that is exciting the fiber. There are four types of fibers, each type concerned with a different part of the visual field. We find that the frog's eye has four distinct parallel distributed channels whereby the frog's eye informs his brain about the visual image in terms of local patterns independent of overall illumination. We also find the size and shape of the functional and anatomical separation of the channels. This work has been done on the frog, and our interpretation applies only to the frog.

INTRODUCTION

Behavior of a Frog

A FROG hunts on land by vision. He escapes enemies mainly by seeing them. His eyes do not move, as do ours, to follow prey, attend suspicious events, or search for things of interest. If his body changes its position with respect to gravity or the whole visual world is rotated about him, then he shows compensatory eye movements. These are slow and clumsy, involving habits, more e.g., as he sits on a rocking lily pad. Thus his eyes are actively stabilized. He has no fovea, or region of greatest acuity in vision, upon which he must center a part of the image. He also has only a single visual system, retina to colliculus, not a double one such as ours where the retina sends fibers not only to colliculus but to the lateral geniculate body which projects to the optic tract and thence to the brain on the frog because of the uniformity of his retina, the normal lack of eye and head movements except for those which stabilize the retinal image, and the relative simplicity of the connection of his eye to his brain.

The frog is not seen to see, or, at any rate, is not concerned with the detail of stationary parts of the world around him. He will startle and deviate his body if it is suddenly moved. His choice of food is determined only by size and movement. He will leap to capture any object the size of an insect or worm, providing

* Original manuscript received by the IRE, September 3, 1939.
This work was supported in part by the U. S. Army (Signal Corps), the U. S. Office of Naval Research, and in part by Bell Telephone Labs., Inc.
† Research Laboratories of Electronics and Dept. of Biology, Mass. Inst. Tech., Cambridge, Mass.
‡ Dept. of Electronics, Mass. Inst. Tech., Cambridge, Mass., on leave from the University of Chile, Santiago, Chile.
|| Res. Lab. of Electronics, Mass. Inst. Tech., Cambridge, Mass.

Hartline [2] first used the term *receptive field* for the region of retina within which a local change of brightness would cause a ganglion cell to respond. His receptive field was a roughly circular area surrounded by annuli, within which changes of brightness affect the cell's response to what is occurring in the receptive field.

Authorized licensed use limited to: Universita di Cagliari. Downloaded on May 09 2024 at 14:00:05 UTC from IEEE Xplore. Restrictions apply.

it moves like one. He can be fooled easily not only by a bit of dangling meat but by any moving small object. His sex life is conducted by sound and touch. His choice of paths in escaping enemies does not seem to be governed by any fixed mechanism, but depends on where he is at the time. Since he is equally at home in water or on land, why should it matter where he lights after jumping or what particular direction he takes? Does he remember a moving thing providing it stays within his field of vision and he is not distracted.

Anatomy of Frog Visual Apparatus

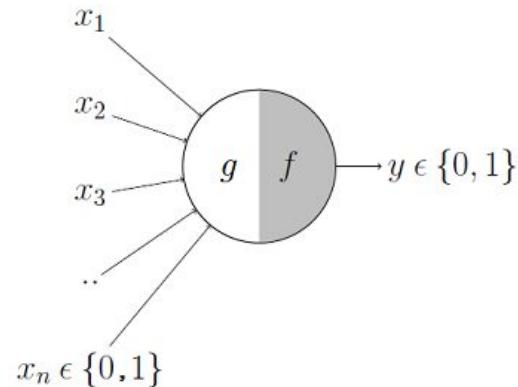
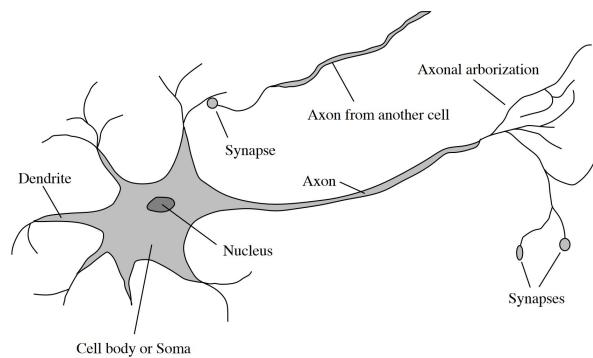
The eye of a frog is shown in Fig. 1(a). Between the lens and eye of the retina are the ganglion cells, whose axons form the optic nerve, lies a layer of connecting neurons (bipolars, horizontals, and amacrine). In the frog there are about 1 million receptors, 2) to 3) million connecting neurons, and half a million ganglion cells [1]. The connections are such that there is a synaptic connection of rod or cone to a given horizontal cell and a ganglion cell receives paths from at least a thousand receptors. Clearly, such an arrangement would not allow for good resolution were the retina meant to map an image in terms of light intensity point by point into a distribution of excitement on the optic nerve.

This is only one layer of ganglion cells in the frog. The cone cells are half as many in number (as against one million rods and cones). The neurons are packed together tightly in a sheet at the level of the cell bodies. Their dendrites, which may extend laterally from 50 to 500 μ , interface widely in what is called the inner plexiform layer, which is a close-packed neuropil containing the terminal arbor of those neurons that lie beneath receptors. The layer of ganglion cells, then, the layer of cell bodies of adjacent ganglion cells is enormous in respect to what they are. Morphologically, there are several types of these cells that are distinct in their dendritic patterns as different species of frogs, from which we infer that they work in different ways. The anatomy shown in the figures is that found in standard references. Further discussion of anatomical questions and additional original work on them will appear in a later publication.

Physiology as Known up to This Study

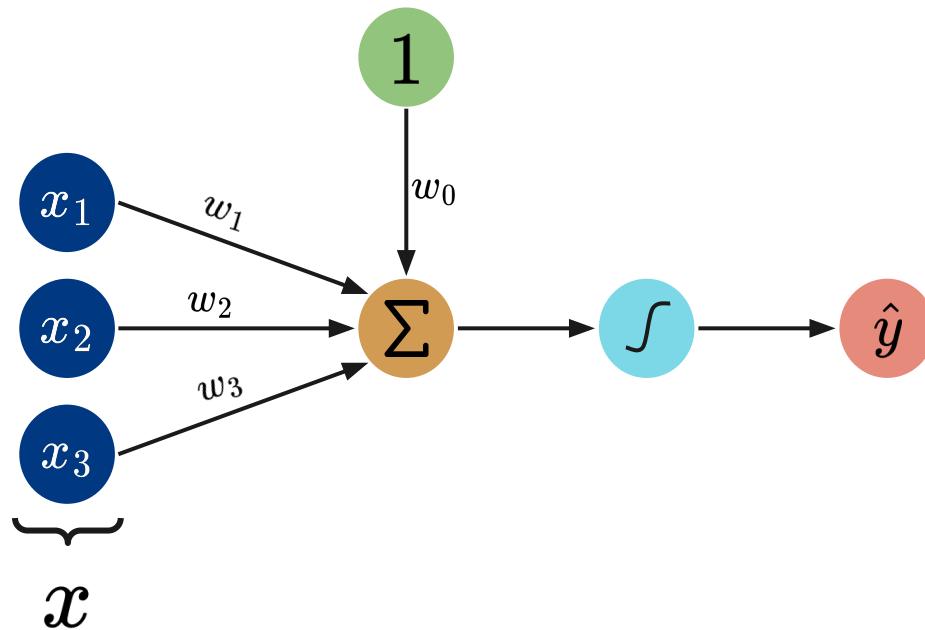
Hartline [2] first used the term *receptive field* for the region of retina within which a local change of brightness would cause a ganglion cell to respond. His receptive field was a roughly circular area surrounded by annuli, within which changes of brightness affect the cell's response to what is occurring in the receptive field.

A Long History



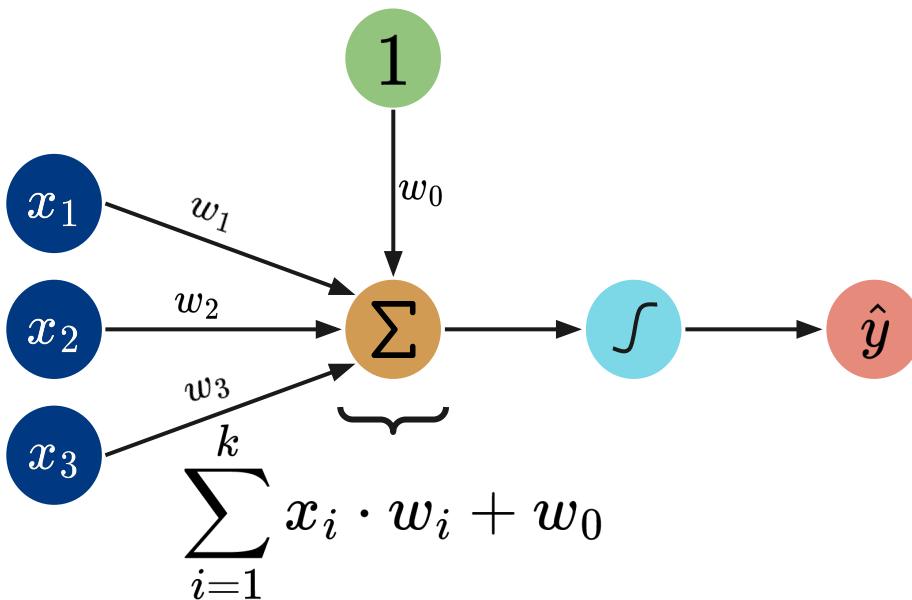
The Perceptron

Perceptron



At first, we have the **input x** , characterized by k values (k **features**)

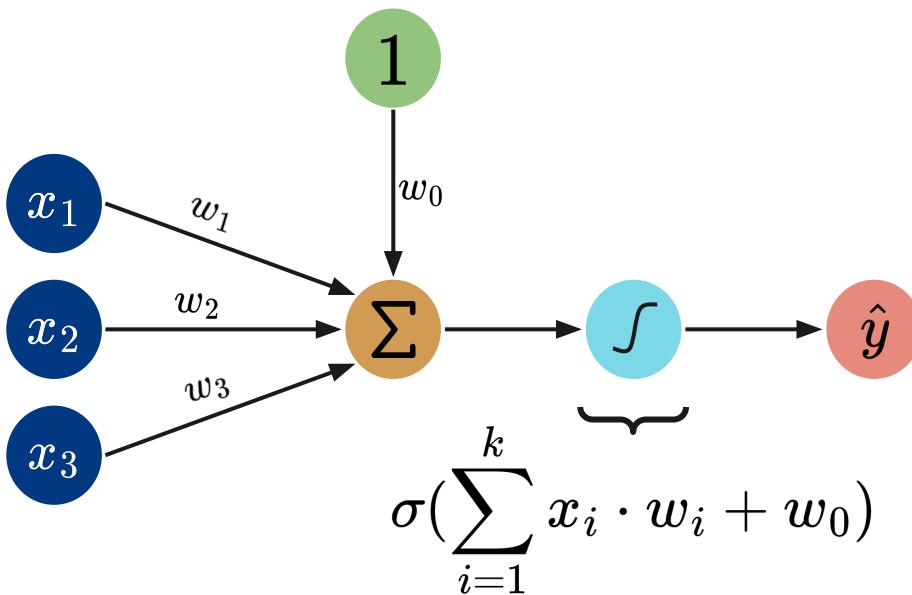
Perceptron



We **multiply** the input values for their relative **weights** "w," which are then **summed up**

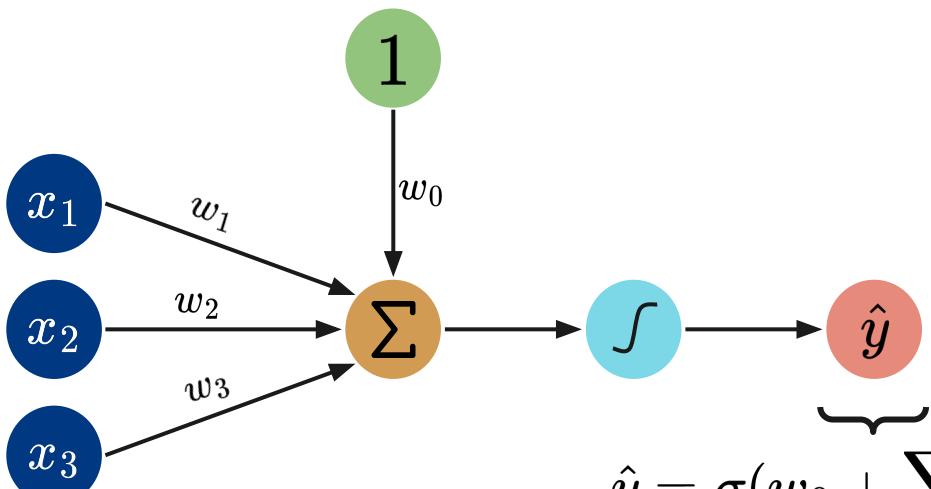
We also add w_0 , which is called the **bias**

Perceptron



Successively, we apply a nonlinear function called the "activation function"

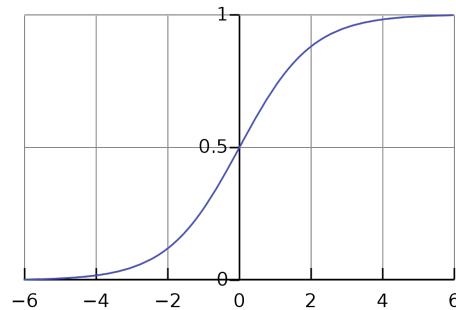
Perceptron



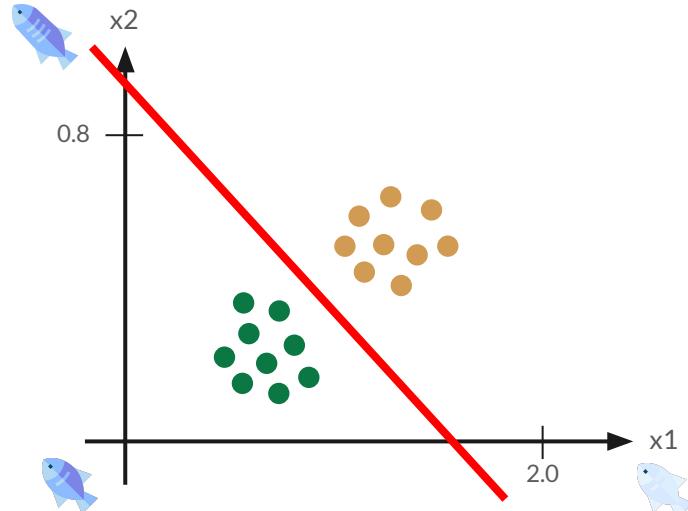
$$\hat{y} = \sigma\left(w_0 + \sum_{i=1}^k x_i \cdot w_i\right)$$

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

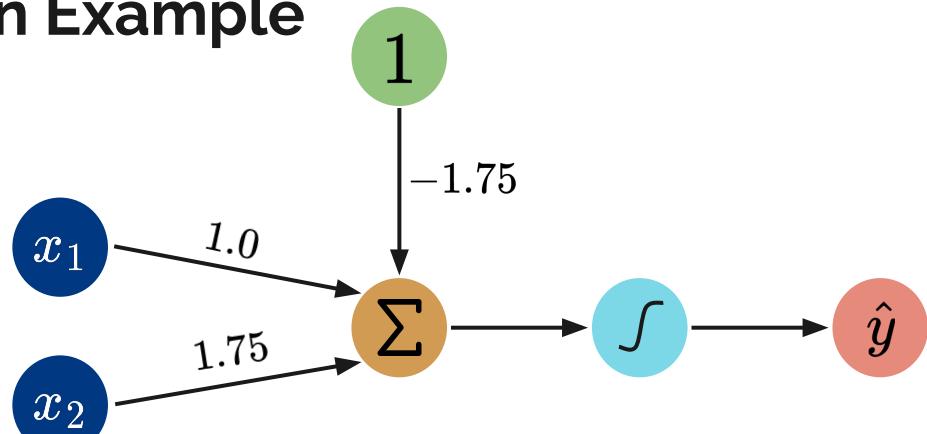
Finally, we obtain the **output y hat**



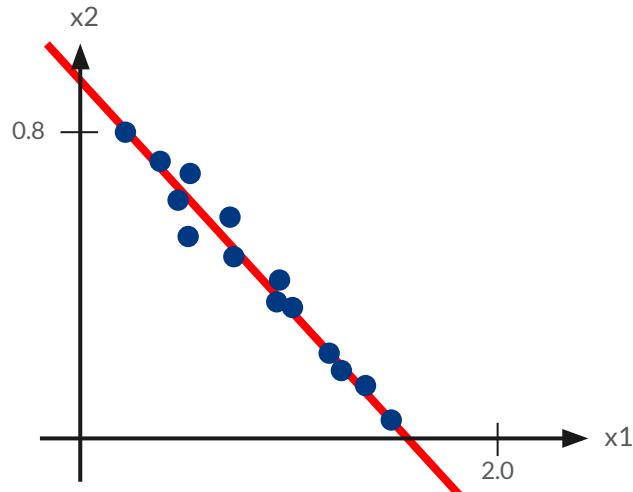
Perceptron: Classification Example



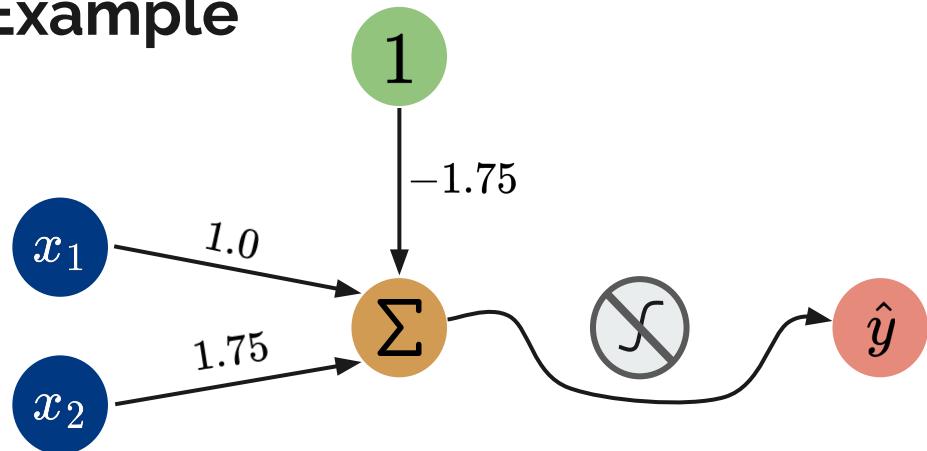
$$y = \sigma(x_1 + x_2 \cdot 1.75 - 1.75)$$



Perceptron: Regression Example

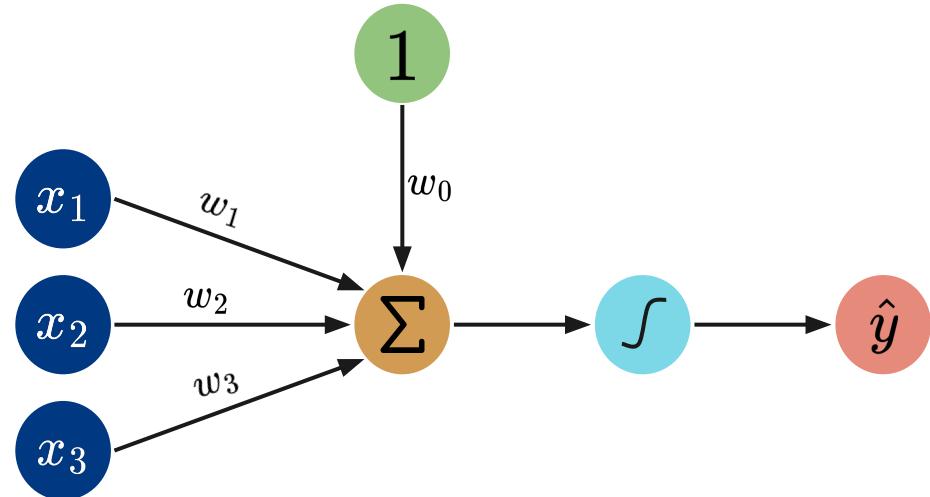
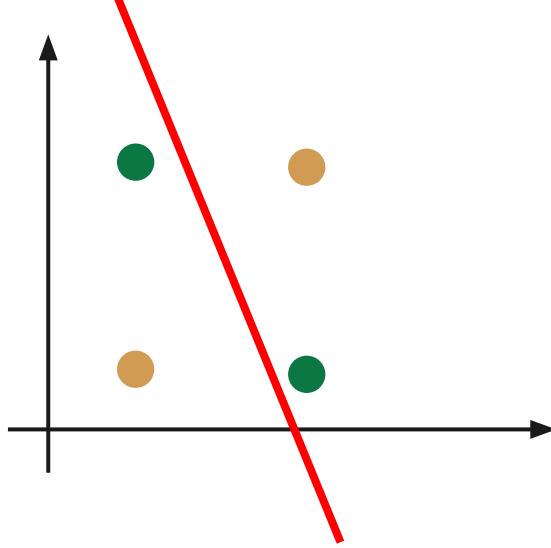


$$y = x_1 + x_2 \cdot 1.75 - 1.75$$



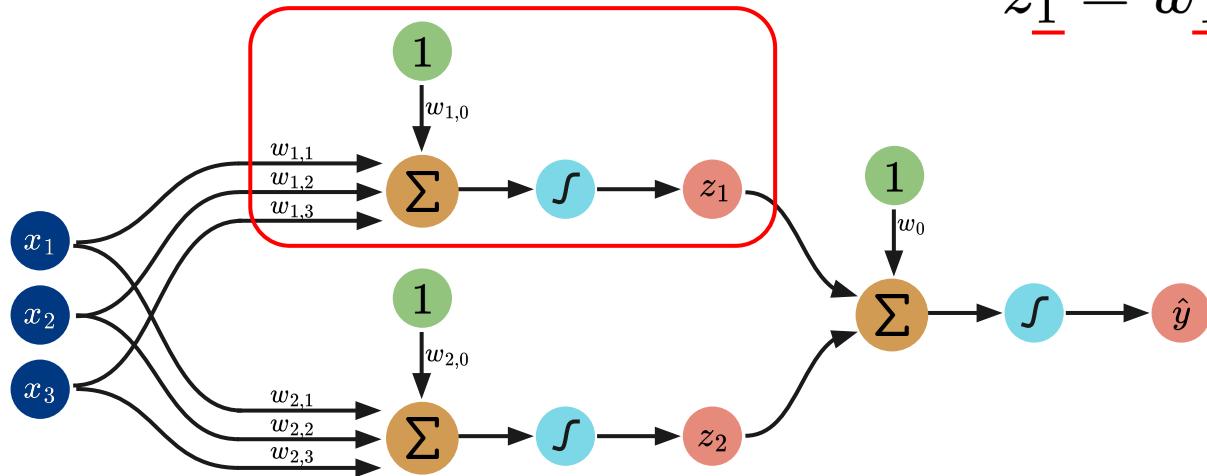
Multi Layer Perceptron

The XOR Problem



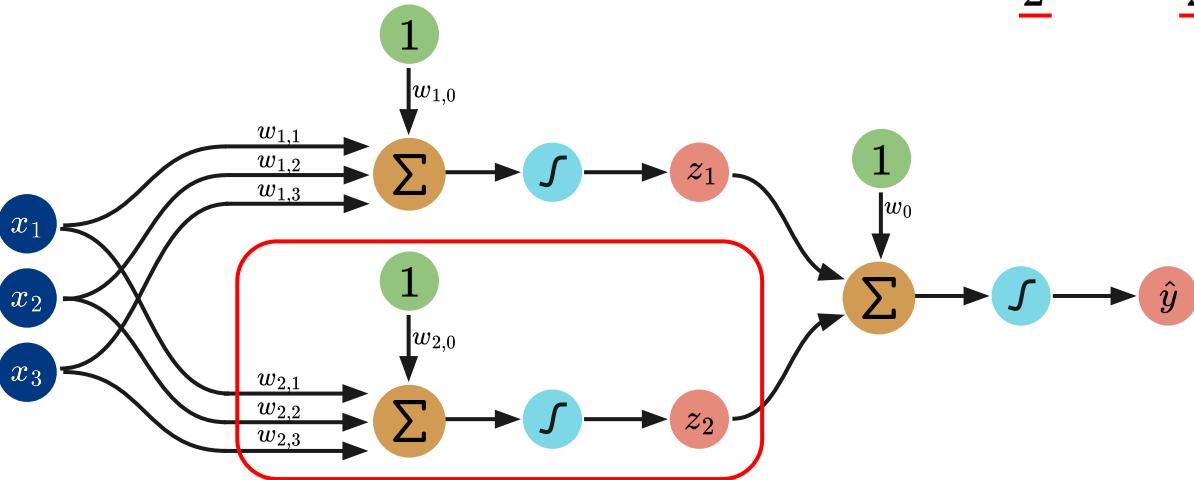
- How do I find a weight configuration for correctly discriminating between these points?
- I need more neurons!

Multi Layer Perceptron



$$z_1 = w_{1,0} + \sum_{i=1}^k x_i \cdot w_{1,i}$$

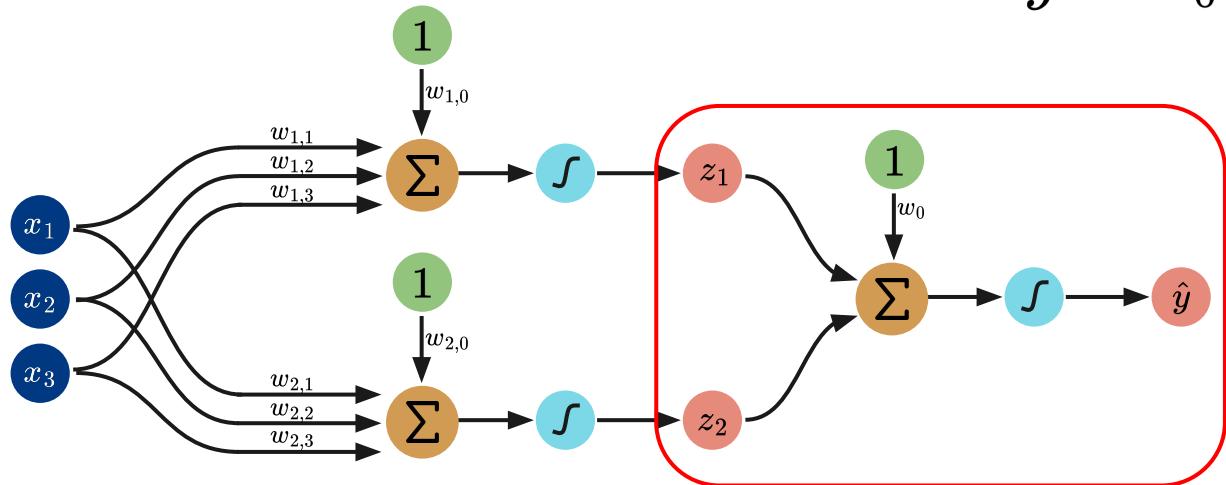
Multi Layer Perceptron



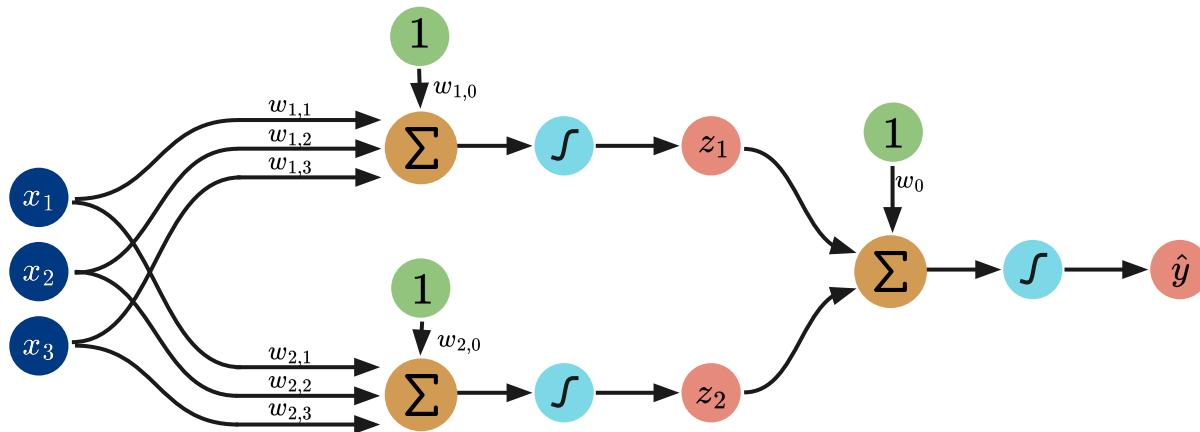
$$z_2 = w_{2,0} + \sum_{i=1}^k x_i \cdot w_{2,i}$$

Multi Layer Perceptron

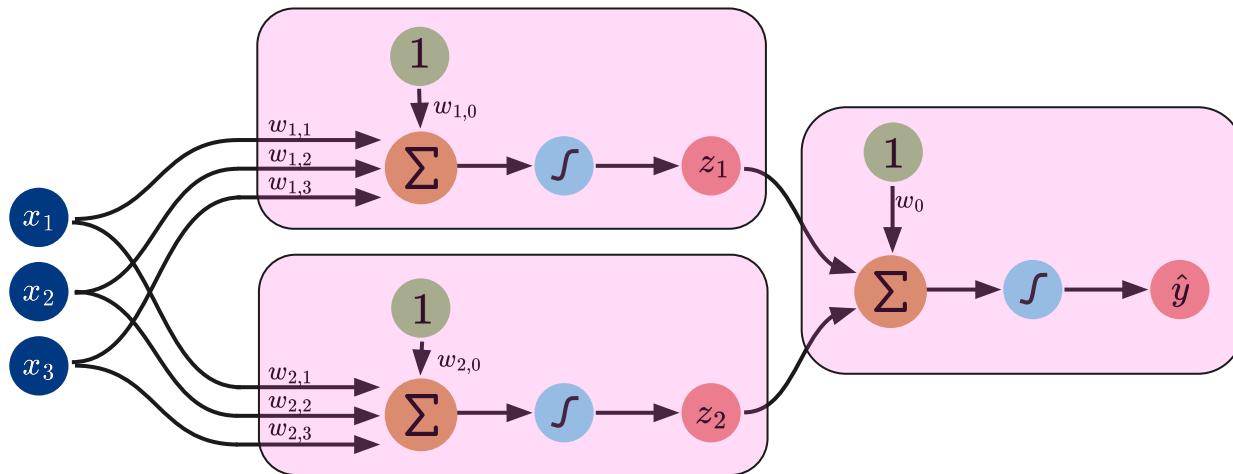
$$\hat{y} = w_0 + \sum_{i=1}^k x_i \cdot z_i$$



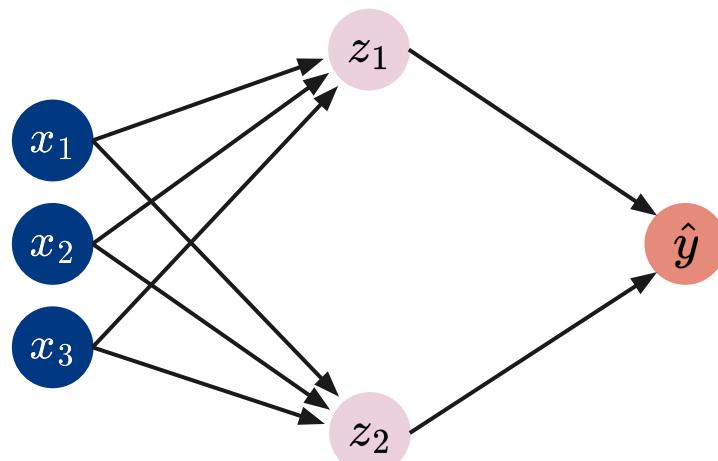
Multi Layer Perceptron



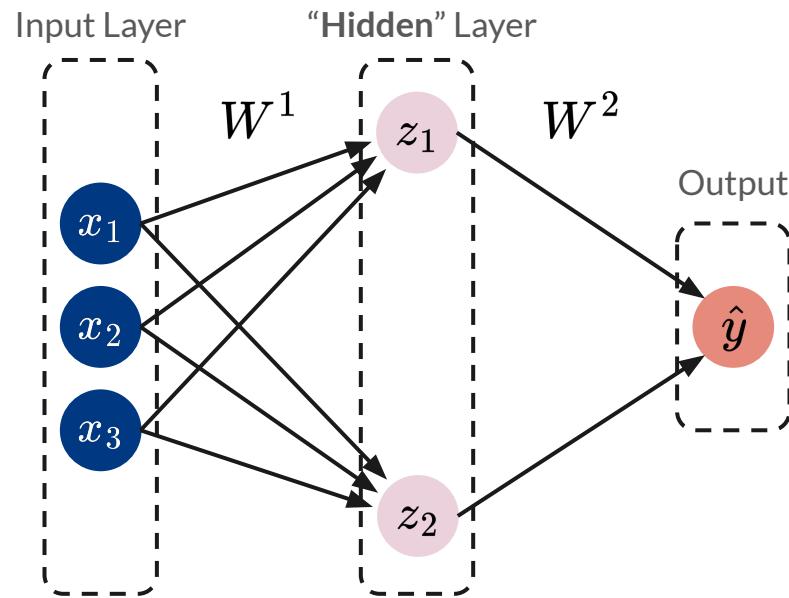
Multi Layer Perceptron



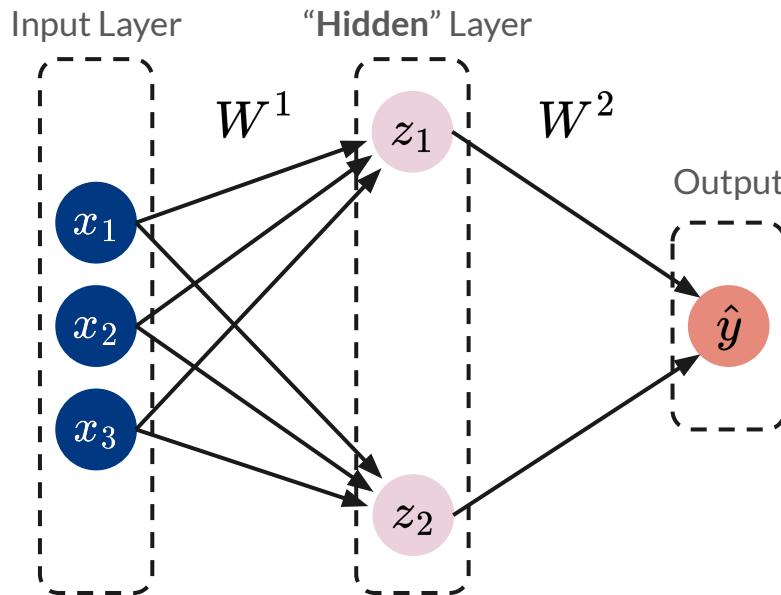
Multi Layer Perceptron



Multi Layer Perceptron



Multi Layer Perceptron



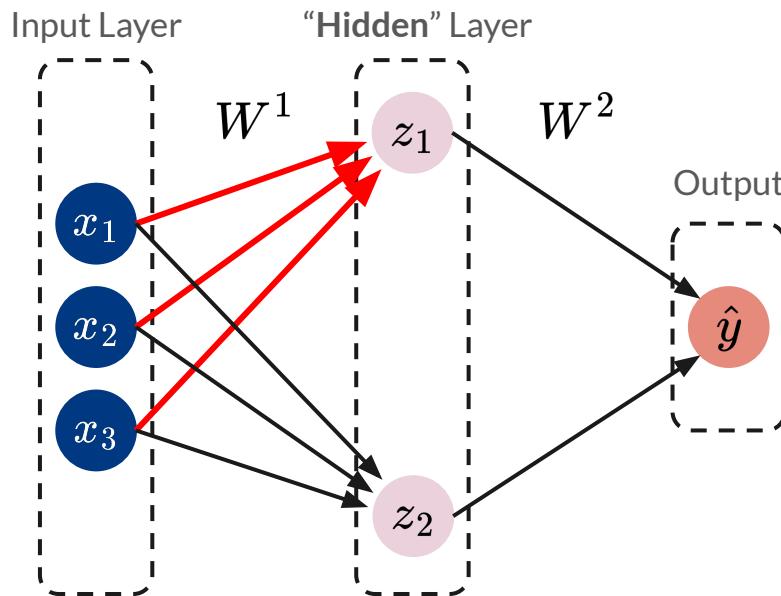
$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

$$x = [x_1, x_2, x_3]$$

$$z = x \cdot W^1$$

$$\hat{y} = z \cdot W^2$$

Multi Layer Perceptron

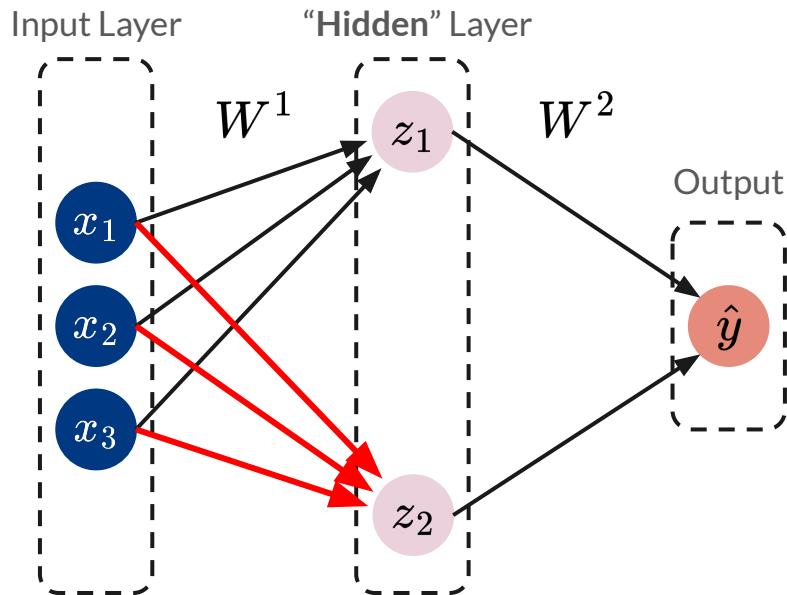


$$W^1 = \begin{bmatrix} b_0^1 \\ w_{0,1}^1 \\ w_{0,2}^1 \\ w_{0,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_1^2 \\ w_{1,1}^2 \\ w_{1,2}^2 \\ w_{1,3}^2 \end{bmatrix}$$

$x = [x_1, x_2, x_3]$

$$z = x \cdot W^1$$
$$\hat{y} = z \cdot W^2$$

Multi Layer Perceptron



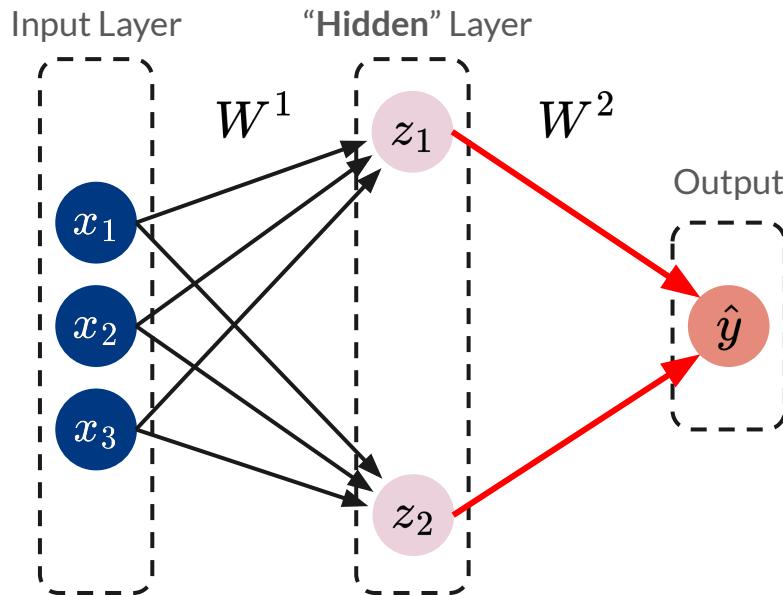
$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

$x = [x_1, x_2, x_3]$

$z = x \cdot W^1$

$\hat{y} = z \cdot W^2$

Multi Layer Perceptron



$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

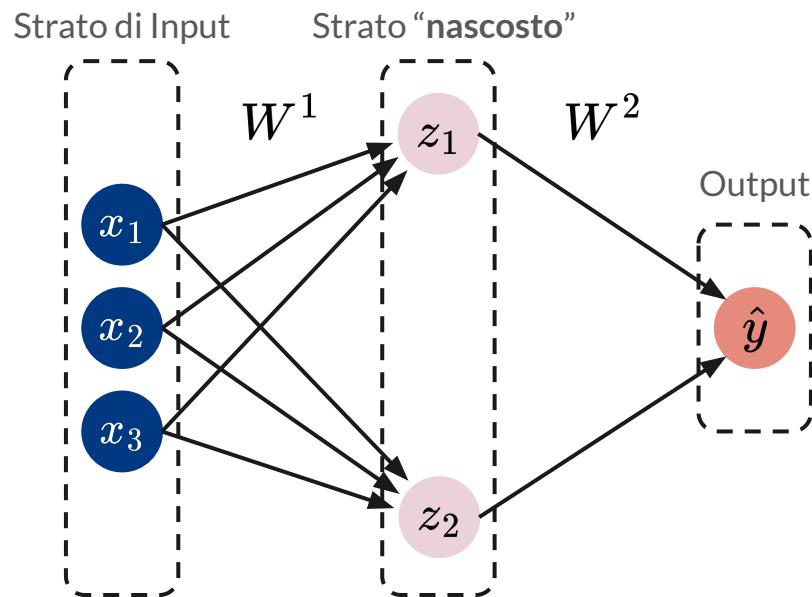
$x = [x_1, x_2, x_3]$

$z = x \cdot W^1$

$\hat{y} = z \cdot W^2$

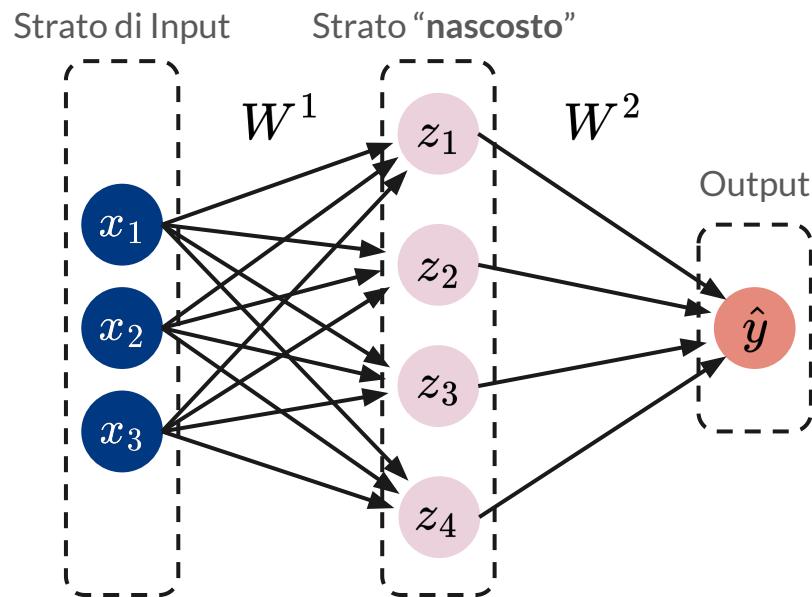
Multi Layer Perceptron

- The Multi Layer Perceptron (MLP) make it possible to build **deep** neural network



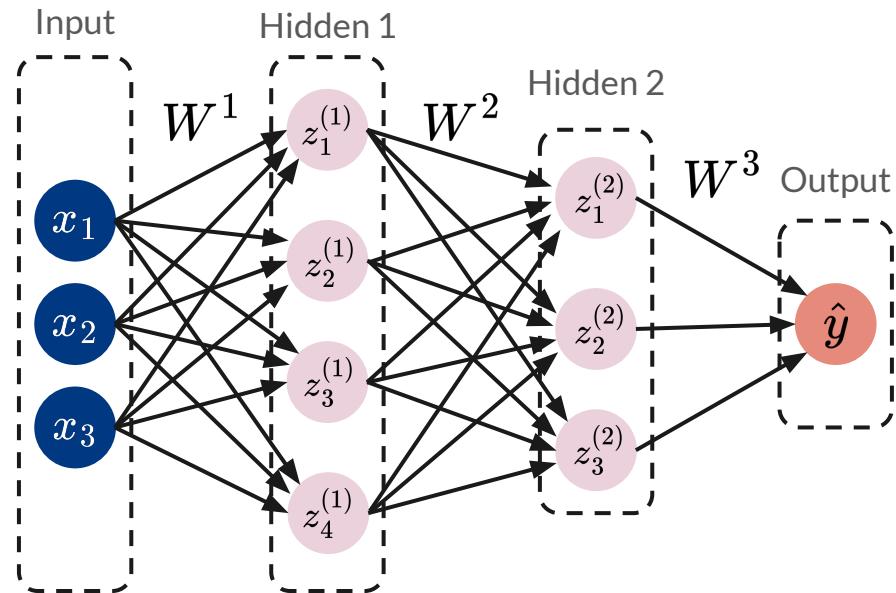
Multi Layer Perceptron

- The Multi Layer Perceptron (MLP) make it possible to build **deep** neural network
- Adding neurons...

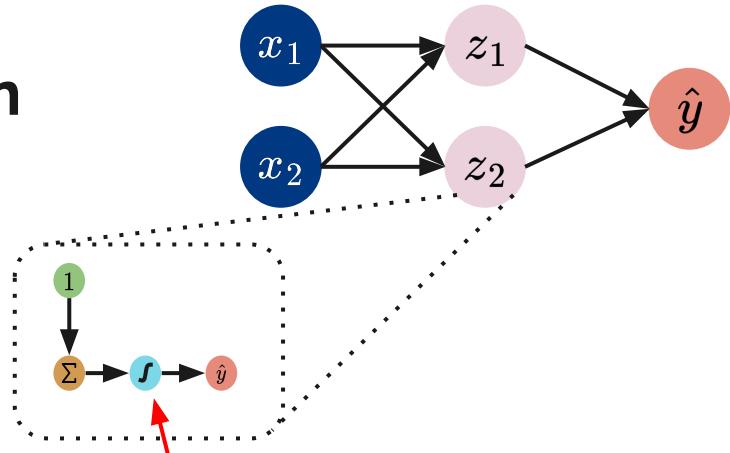
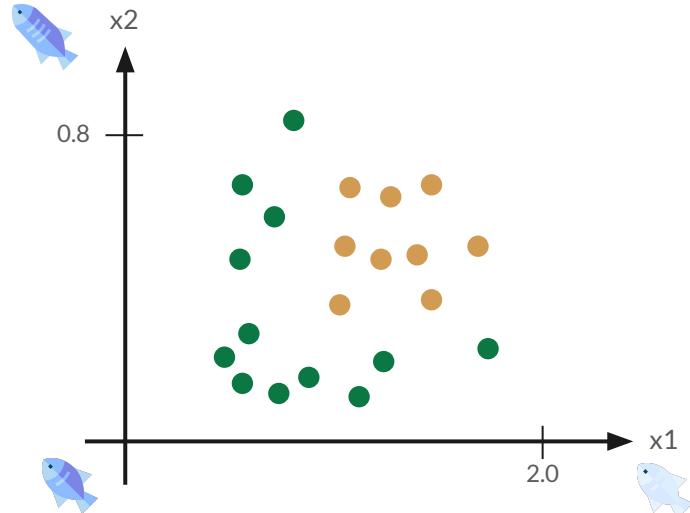


Multi Layer Perceptron

- The Multi Layer Perceptron (MLP) make it possible to build **deep** neural network
- Adding neurons...
- ... as well as layers

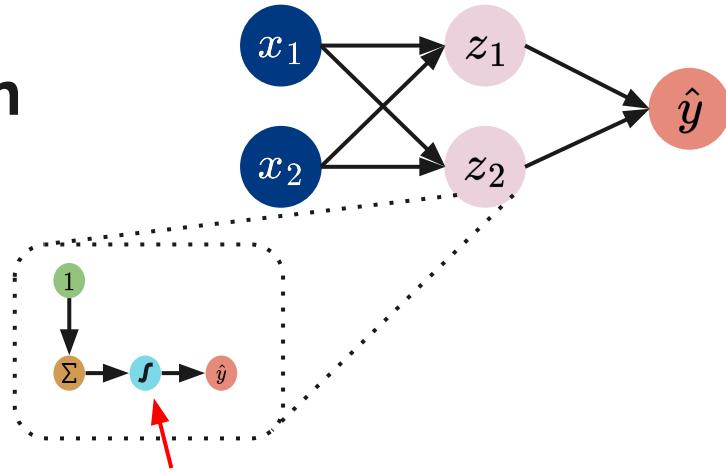
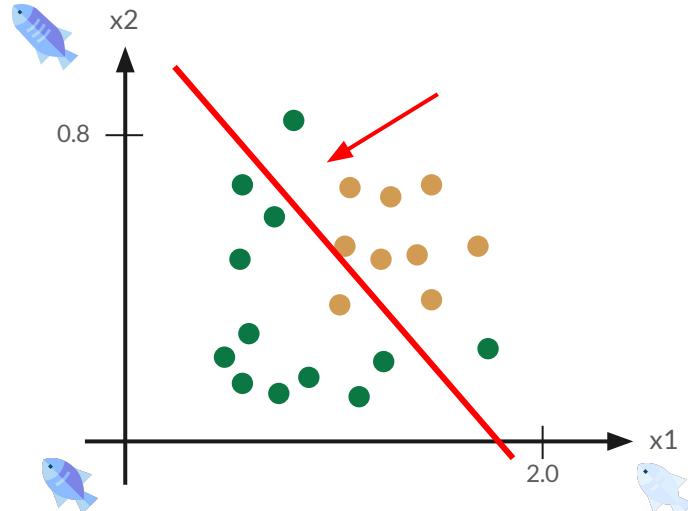


Non Linear Activation Function



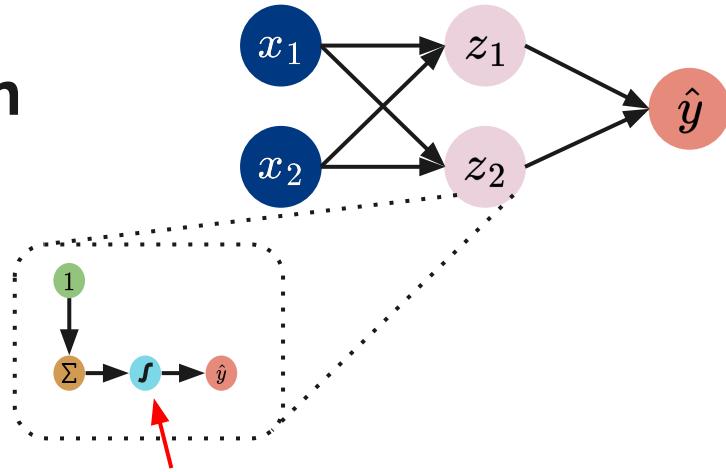
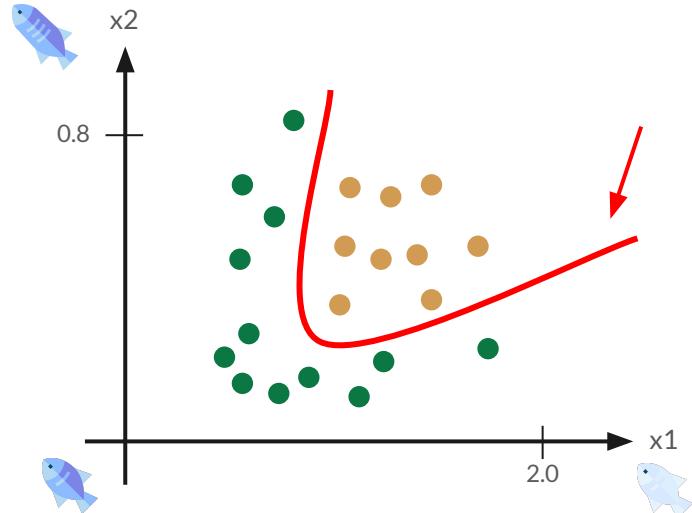
- But why do we need a non-linear activation function?

Non Linear Activation Function



- But why do we need a **non-linear activation function**?
- We can construct **only linear classifiers** using linear activations, no matter how deep the network is!

Non Linear Activation Function

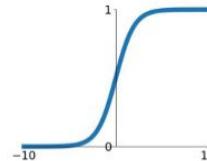


- But why do we need a **non-linear activation function**?
- We can construct **only linear classifiers** using linear activations, no matter how deep the network is!
- The non-linearity of the activation function also lets us **construct non-linear classifiers**

Activation Functions

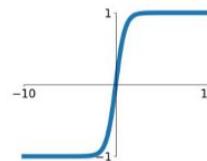
Sigmoid

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



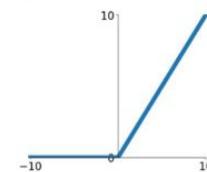
tanh

$$\tanh(x)$$



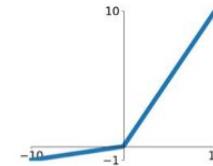
ReLU

$$\max(0, x)$$



Leaky ReLU

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

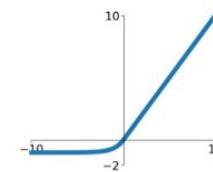


Maxout

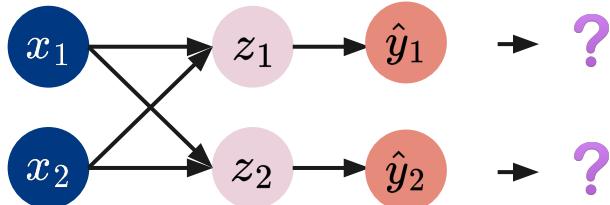
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

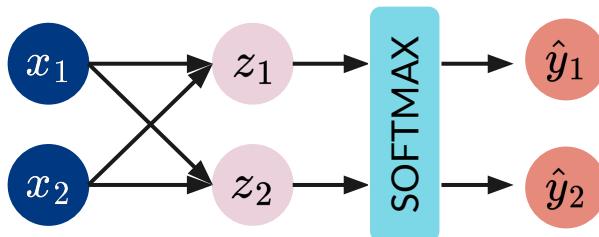


Multi-Class Problems



- Instead of using a single neuron we can use a neuron for each class
- But how can we obtain a probability distribution over the classes?
- We need to model a categorical distribution!

Multi-Class Problems: Softmax Function



- Instead of using a single neuron we can use a neuron for each class
- But how can we obtain a probability distribution over the classes?
- We need to model a categorical distribution!

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Softmax Function

Input pixels, x



Feedforward output, y_i

	cat	dog	horse
cat	5	4	2
dog	4	2	8
horse	4	4	1

Forward
propagation

Softmax output, $S(y_i)$

	cat	dog	horse
cat	0.71	0.26	0.04
dog	0.02	0.00	0.98
horse	0.49	0.49	0.02

Softmax
function

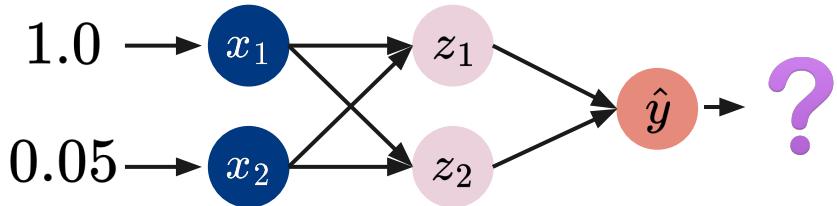
Shape: (3, 32, 32)

Shape: (3,)

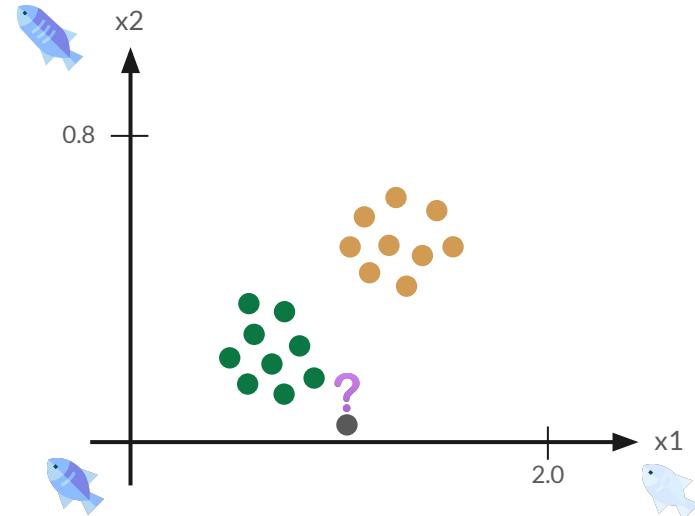
Shape: (3,)

Training a Neural Network

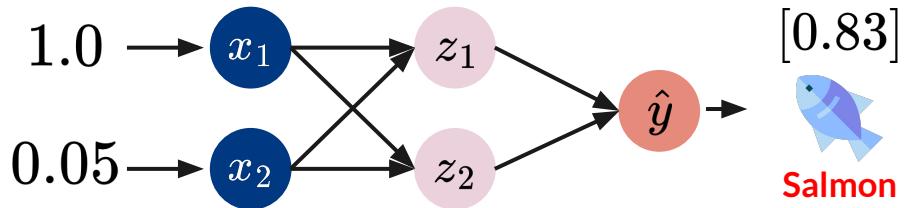
How to obtain useful weights



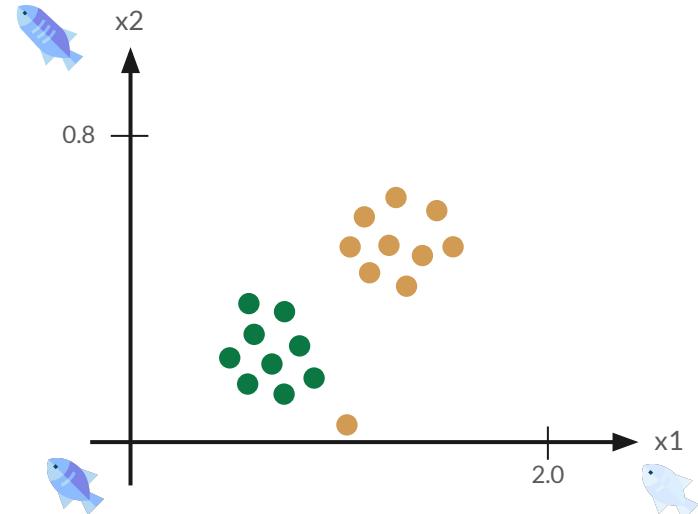
- At first the network weights are **random**



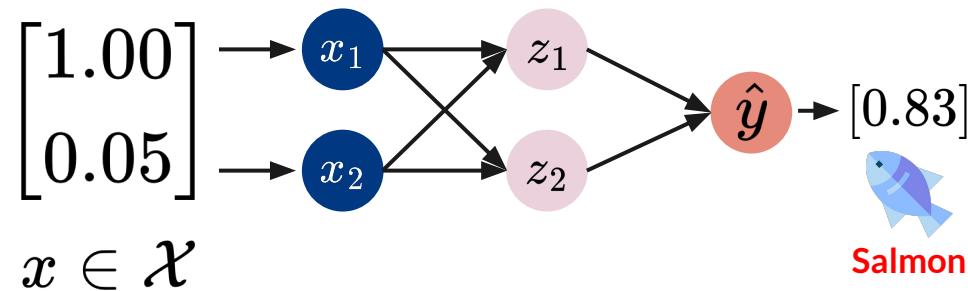
How to obtain useful weights



- At first the network weights are **random**
- In this case the **output is useless!**
- We should define a **learning algorithm**

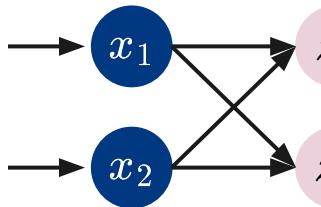


Loss Function



Loss Function

$$\begin{bmatrix} 1.00 \\ 0.05 \end{bmatrix} \rightarrow x \in \mathcal{X}$$



$$\hat{y} \rightarrow [0.83]$$



Sea Bass

$$y \rightarrow [0.00]$$

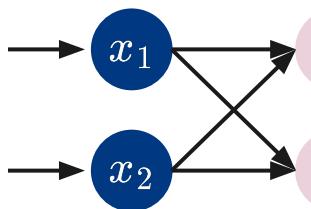


Salmon

$$\mathcal{L}(\hat{y}, y)$$

Loss Function

$$\begin{bmatrix} 1.00 \\ 0.05 \end{bmatrix} \rightarrow x \in \mathcal{X}$$



$$\hat{y} \rightarrow [0.83]$$



Salmon

$$y \rightarrow [0.00]$$



Sea Bass

$$\mathcal{L}(\hat{y}, y)$$

We need some way for encoding
the information of “error”

Cross Entropy

- The main idea is the notion of **diversity**
- How different is my output from the desired one?
- We can express this diversity by using the notion of **Cross-Entropy**
- Cross-Entropy measures the **divergence** between two categorical distributions

$$H(\hat{y}, y) = -y_0 \log \hat{y}_0 - y_1 \log \hat{y}_1$$

$$\begin{bmatrix} 0.83 \\ 0.17 \end{bmatrix}$$

Output (as a probability distribution)

$$\begin{bmatrix} 0.00 \\ 1.00 \end{bmatrix}$$

Label (as a probability distribution)

Cross Entropy - Example

$$H(\hat{y}, y) = -y_0 \log \hat{y}_0 - y_1 \log \hat{y}_1$$

$$H(\hat{y}, y) = -0 \log(0.83) - 1 \log(0.17)$$

$$= -\log(0.17) = 1.77$$

$$\begin{bmatrix} 0.83 \\ 0.17 \end{bmatrix}$$

Output (as a probability distribution)

$$\begin{bmatrix} 0.00 \\ 1.00 \end{bmatrix}$$

Label (as a probability distribution)

Cross Entropy - Example

$$H(\hat{y}, y) = -y_0 \log \hat{y}_0 - y_1 \log \hat{y}_1$$

$$H(\hat{y}, y) = -0 \log(0.83) - 1 \log(0.17)$$

$$= -\log(0.79) = \boxed{0.23}$$

- We can decrease the error by making the prediction closer to the desired output

$$\begin{bmatrix} 0.21 \\ 0.79 \end{bmatrix}$$

Output (as a probability distribution)

$$\begin{bmatrix} 0.00 \\ 1.00 \end{bmatrix}$$

Label (as a probability distribution)

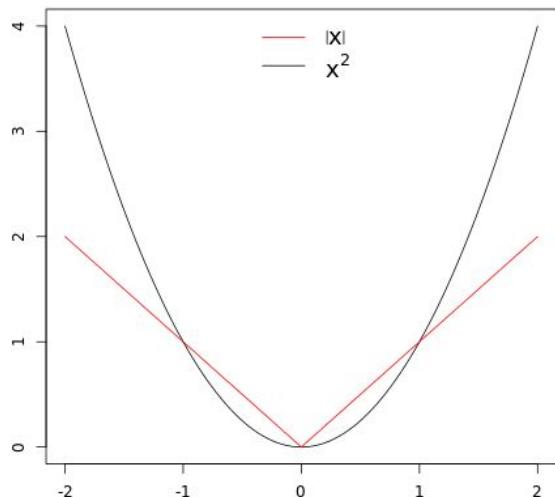
Mean Square Error

- If we are estimating continuous values we need a proper measure of diversity for real values
- In general, the divergence may be captured by computing the distance between the two numbers
 - Mean Absolute Error (MAE)
- The most common idea is however to weight differently large distances
 - Mean Square Error (MSE)

23.5
Output

25
Label

Mean Square Error



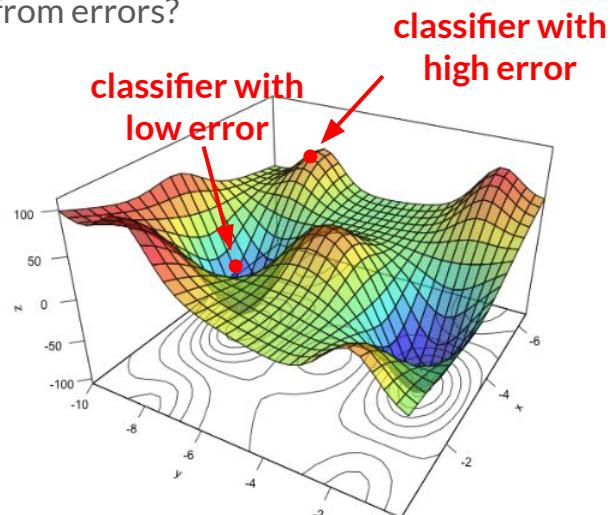
$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}|$$

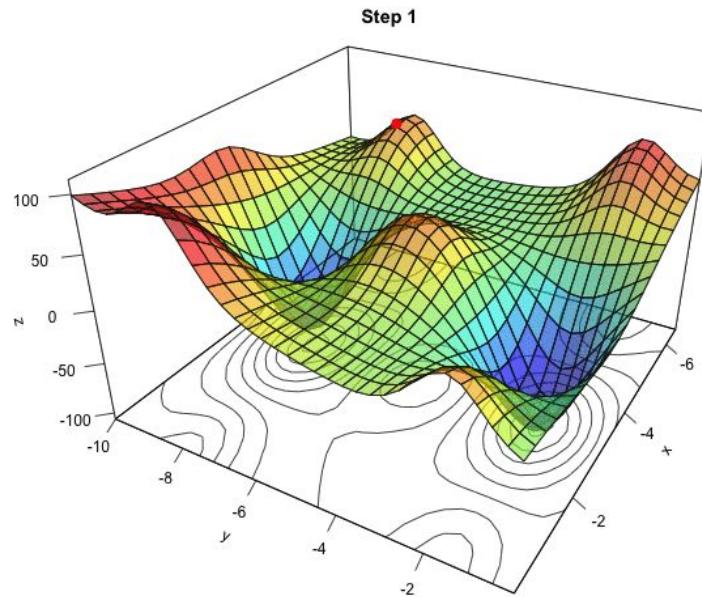
Learning from Errors



- Now we know how to quantify the errors... but how can we learn from errors?
- Changing the weights implies changing the output... and changing the output implies changing the error
- The loss function, indeed, depends upon the network parameters
- One can see each possible configuration as a point on a multi-dimensional surface called the Loss Surface

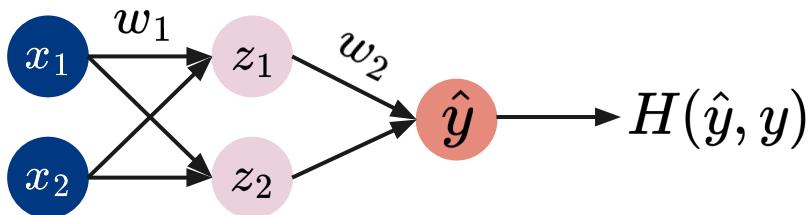


Gradient Descent



- L'idea è quella di “seguire” la direzione che mi porta verso il basso: Algoritmo di Discesa del Gradiente (Gradient Descent)
- The **gradient** of the loss function gives us information about the “**direction**” and the “**magnitude**” of the **slope of the surface**.

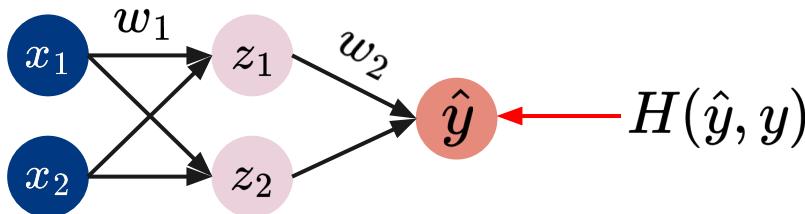
Backpropagation



- We can use the **chain rule** to compute the gradients with respect to the weights.

$$\frac{\partial H(\hat{y}, y)}{\partial w_2} = \frac{\partial H(\hat{y}, y)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

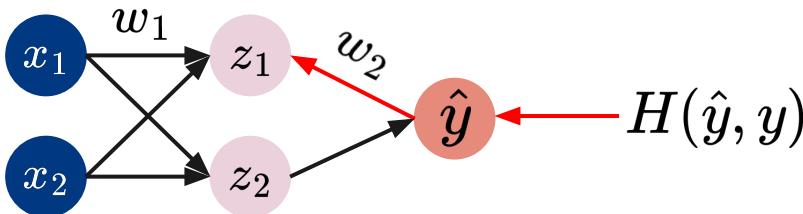
Backpropagation



- We can use the **chain rule** to compute the **gradients** with respect to the weights.
- First of all, we compute the **derivative of the loss with respect to the output**.

$$\frac{\partial H(\hat{y}, y)}{\partial w_2} = \boxed{\frac{\partial H(\hat{y}, y)}{\partial \hat{y}}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$

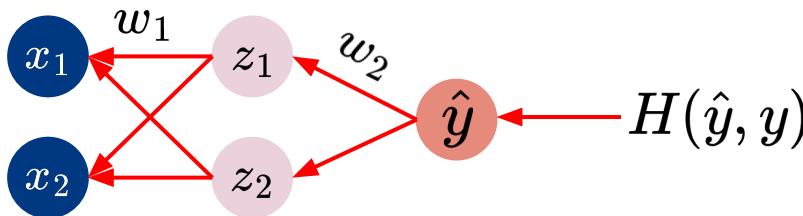
Backpropagation



$$\frac{\partial H(\hat{y}, y)}{\partial w_2} = \frac{\partial H(\hat{y}, y)}{\partial \hat{y}} \cdot \boxed{\frac{\partial \hat{y}}{\partial w_2}}$$

- We can use the **chain rule** to compute the **gradients** with respect to the weights.
- First of all, we compute the **derivative of the loss with respect to the output**.
- Then, we multiply the **derivative of the output with respect to the selected weight**.

Backpropagation



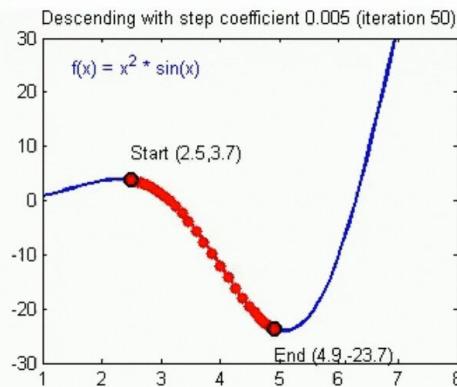
$$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial H(\hat{y}, y)}{\partial W^{(k)}}$$

Learning rate

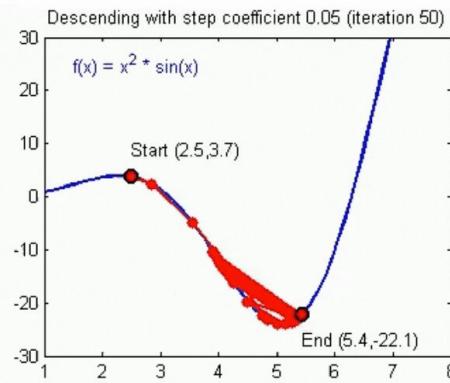
- We can use the **chain rule** to compute the **gradients** with respect to the weights.
- First of all, we compute the **derivative of the loss with respect to the output**.
- Then, we multiply the **derivative of the output with respect to the selected weight**.
- Once the derivative with respect to the weights is obtained, **we can use this value to update them**, leading toward directions for which the loss is minimized.

Learning Rate

Convergence

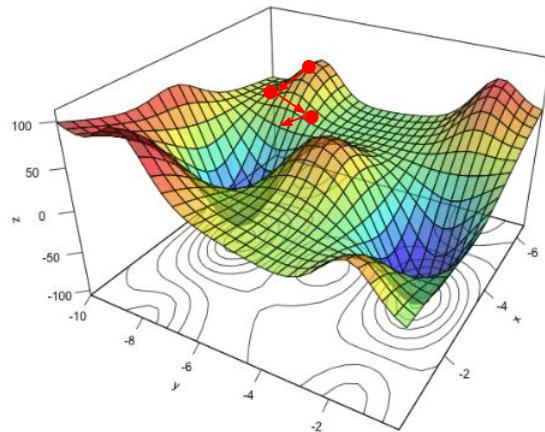


Divergence



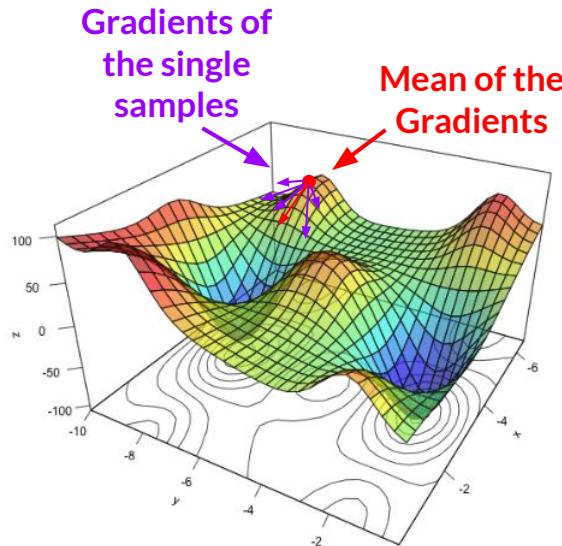
$$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial H(\hat{y}, y)}{\partial W^{(k)}}$$

Stochastic Gradient Descent



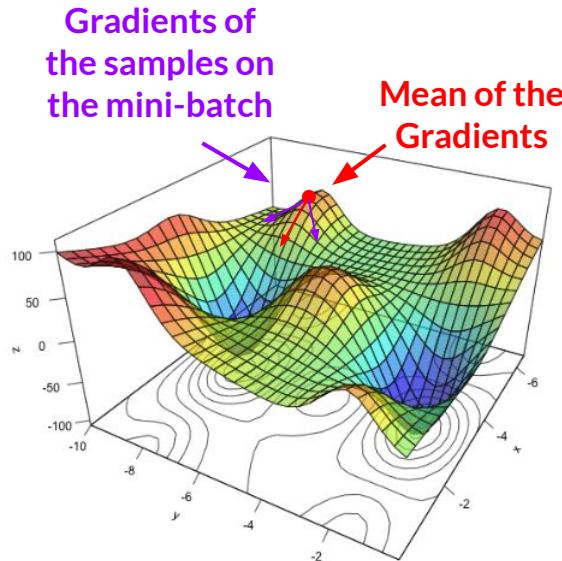
- Remember, our goal is to obtain **generalization capabilities** and not minimize the loss for a **single sample**
- A possible solution to this problem may be **randomly selecting samples from the training set** (Stochastic Gradient Descent)
- It is very **efficient** but leads to a **noisy descent**

Batch Gradient Descent



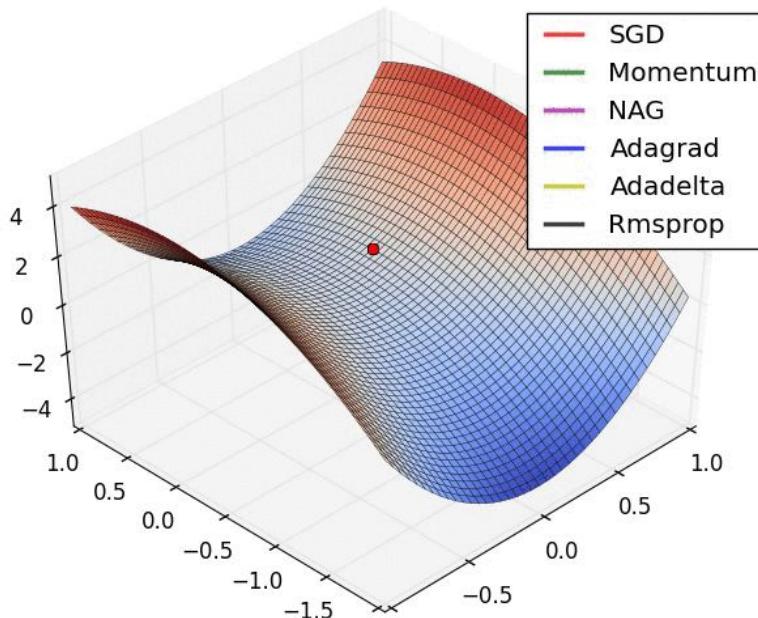
- A possible evolution of the SGD consists of using the **mean of the gradients** computed on **the whole training set** to make a single step (**Batch Gradient Descent**).
- It is a very **effective** strategy but also very **expensive**.

Mini-Batch Gradient Descent



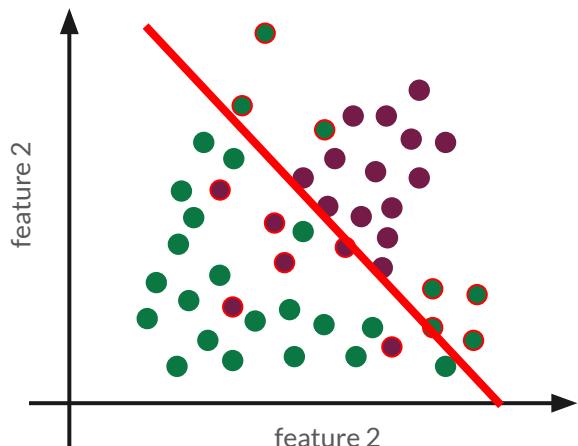
- A possible evolution of the SGD consists of using the **mean of the gradients** computed on **the whole training set** to make a single step (**Batch Gradient Descent**).
- It is a very **effective** strategy but also very **expensive**.
- The most common approach is, hence, something in between: using a **subset** (a "batch") of samples at each iteration (**Mini batch Gradient Descent**).

Can we do better?

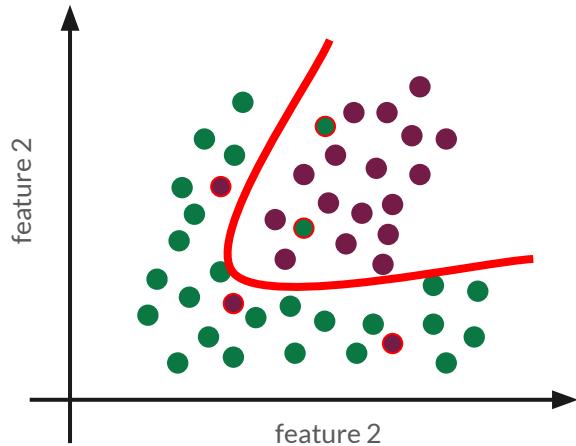


- Esistono diverse migliorie che si possono attuare matematicamente
- Oltre all'SGD, esistono degli **ottimizzatori** che mirano a evitare che i gradienti rimangano intrappolati in minimi locali in alcune dimensioni:
 - SGD + Momentum
 - RMSProp
 - Adam
- Le librerie per costruzioni di reti neurali già implementano questi ottimizzatori... Empiricamente, bisogna testarli per trovare quello più adatto al problema che stiamo risolvendo!

Overfitting



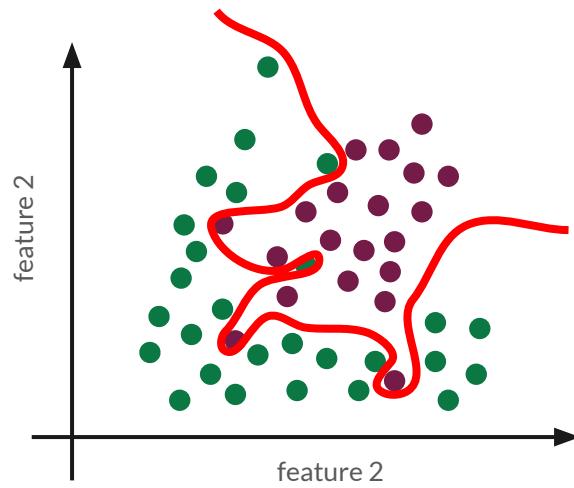
Underfitting



Good Fit



Neural Networks Suffers from high risk of Overfitting!



Overfitting

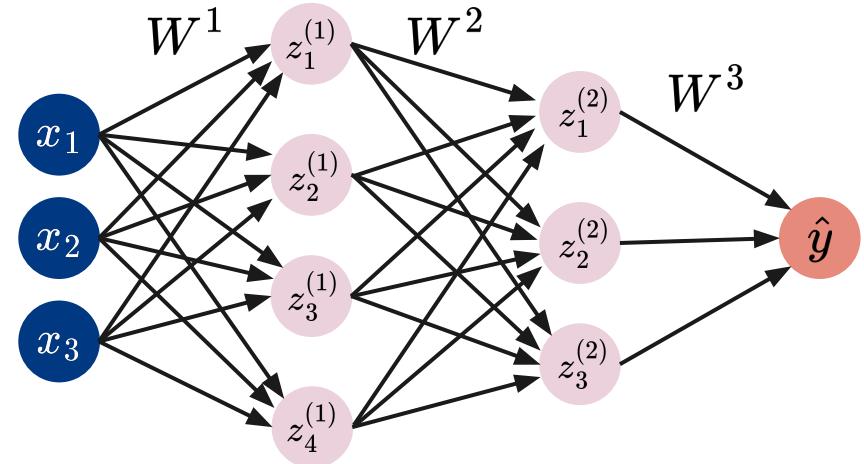
Regularization

Regularization term

$$W^{(t+1)} \leftarrow W^{(t)} - \alpha \frac{\partial H(\hat{y}, y)}{\partial W^{(t)}} + \overbrace{\|W^{(t)}\|}^{\text{Regularization term}}$$

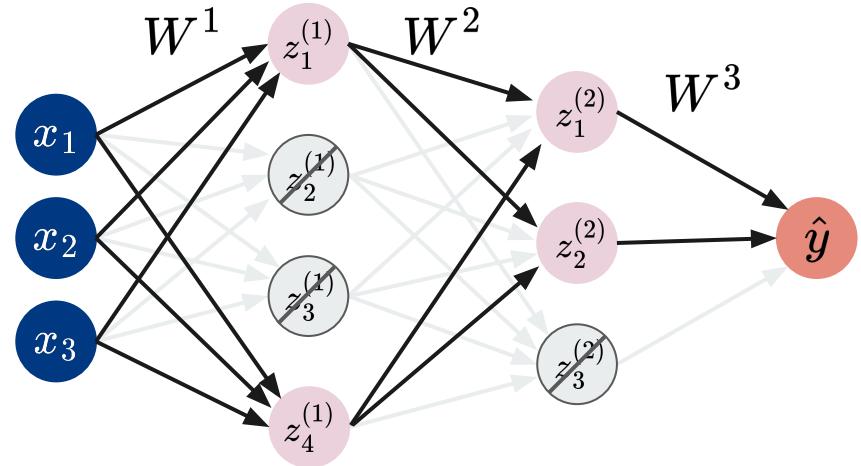
- An L1 regularization (norm 1) **enforce sparsity** but **penalizes the magnitude** of the weights
- An L2 regularization (norm 2) **penalizes sparsity** but **enforce the magnitude** of the weights
- In general, the desired effect is to diminish the **models' complexity**

Dropout Regularization



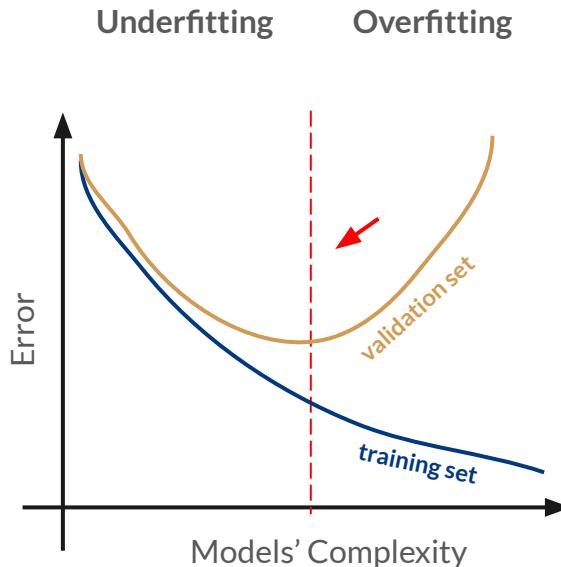
- In standard training we use all the weights during training...

Dropout Regularization



- When training with **dropout**, instead, we “turn off” some of the neurons at random at each iteration, in order to reinforce each connection of the network

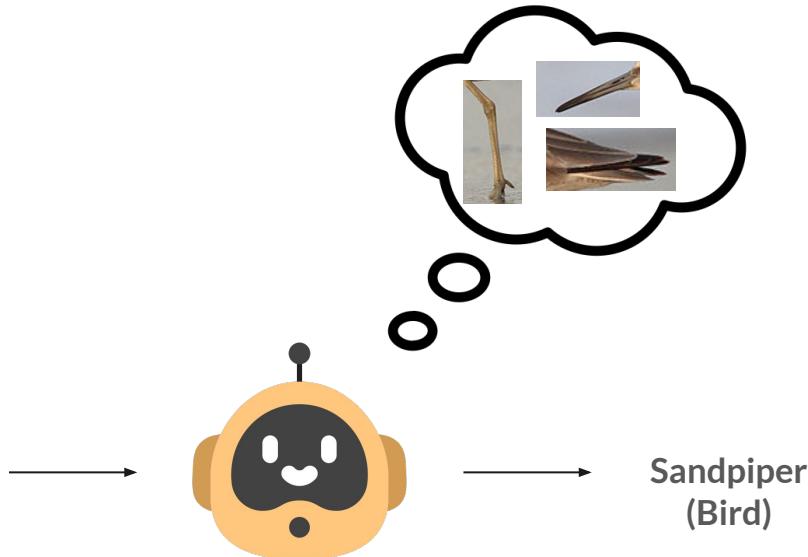
Early Stopping



- We want to keep training until the error on the validation set stop decreasing
- Being the SGD algorithm iterative, we can **force** an "early stopping" if, at some iteration, the error on the validation increases instead of decreasing.

Convolutional Neural Networks

Computer Vision



- Intuitively, we need an algorithm for extracting **important features** from images
- but this is not **straightforward**



Feature Extraction

DATA SCIENCE

Feature Extraction

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



Images from the Computer Perspective



How can an algorithm recognize images?



Images from the Computer Perspective

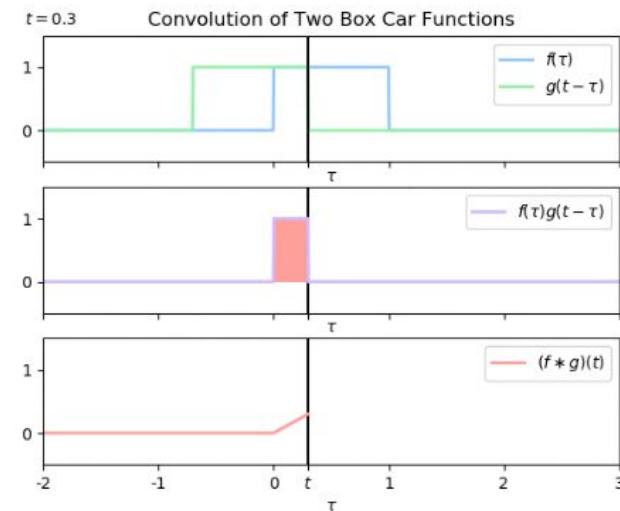
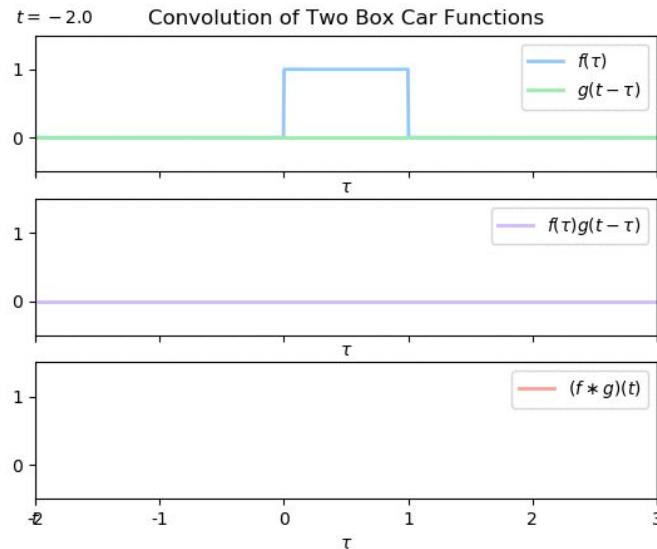


What Computer Sees

Come può un algoritmo riconoscere la classe d'appartenenza di un'immagine?

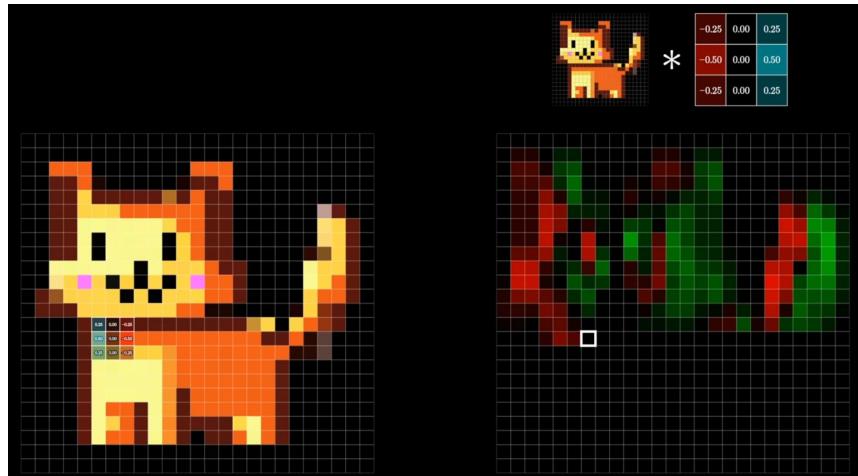
Convoluzioni!

Convolutions

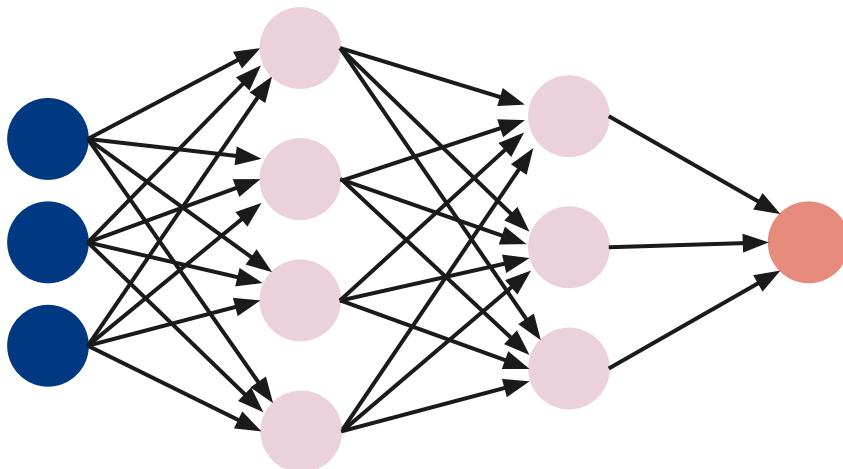


Convolutions

<https://youtu.be/KuXjwB4LzSA?feature=shared&t=388>



Fully Connected Layer



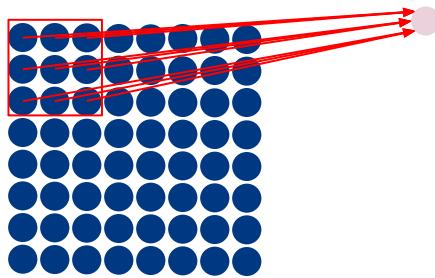
- In **fully connected** (FC) layers, each neuron of a given layer is connected to each neuron in the next layer
- This would lead to a **high number of connections**
- And lacks from **spatial awareness**

Convolutional Layer



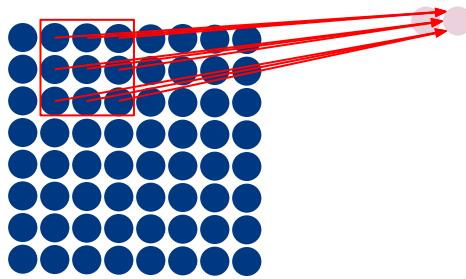
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



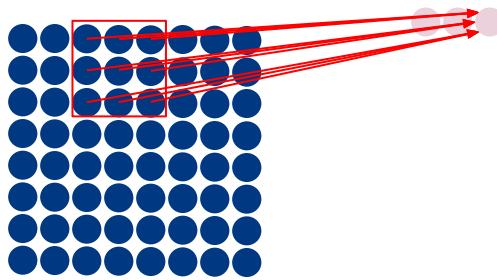
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



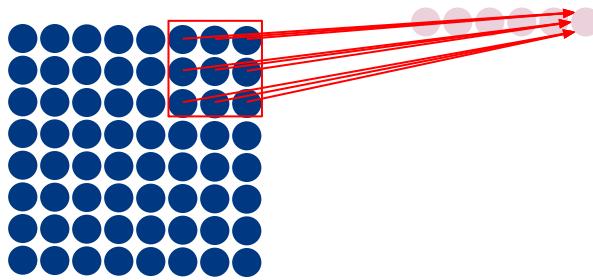
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



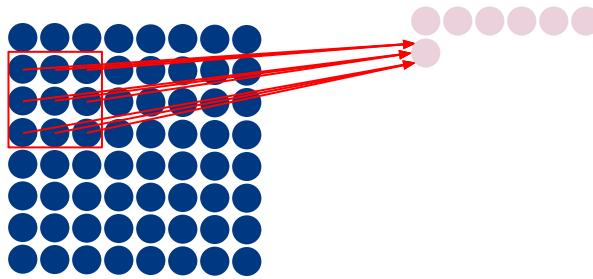
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



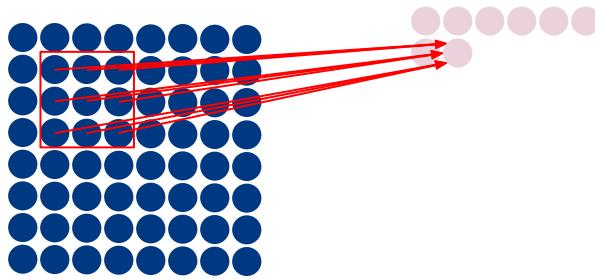
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



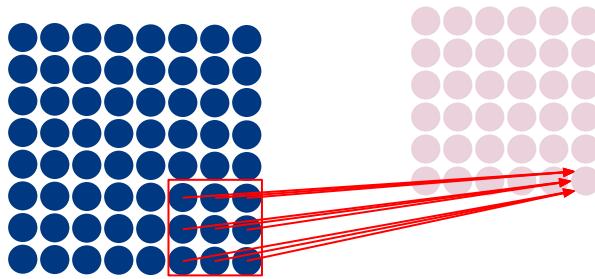
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



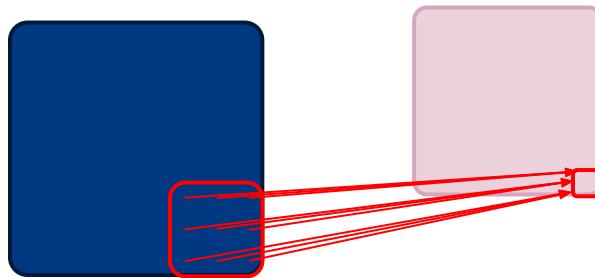
Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$



Convolutional Layer

$$W_1^{(1)} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix}$$

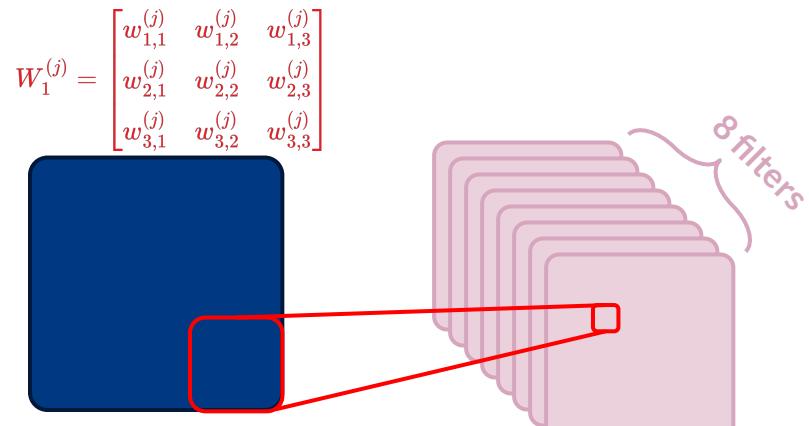


3x3 filter, replicated for
each pixel

CNN: **9+1** parameters!

FC: ~**2300** parameters!

Convolutional Layer

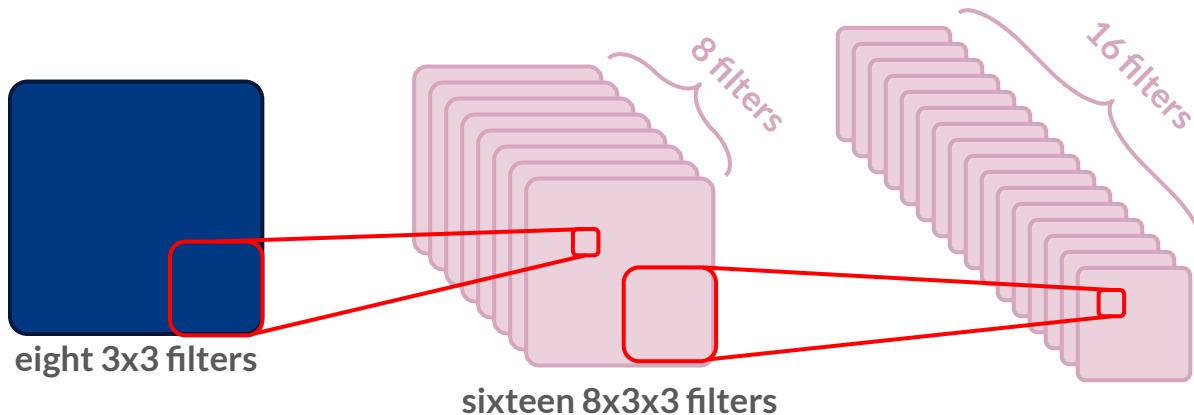


eight 3x3 filters

CNN: **9+1** x 8 = 80 parameters

FC: ~**2300** x 8 ~= 18500 parameters!

Convolutional Layer



Conv Layer 1

CNN: $9+1 \times 8 = 80$ parameters

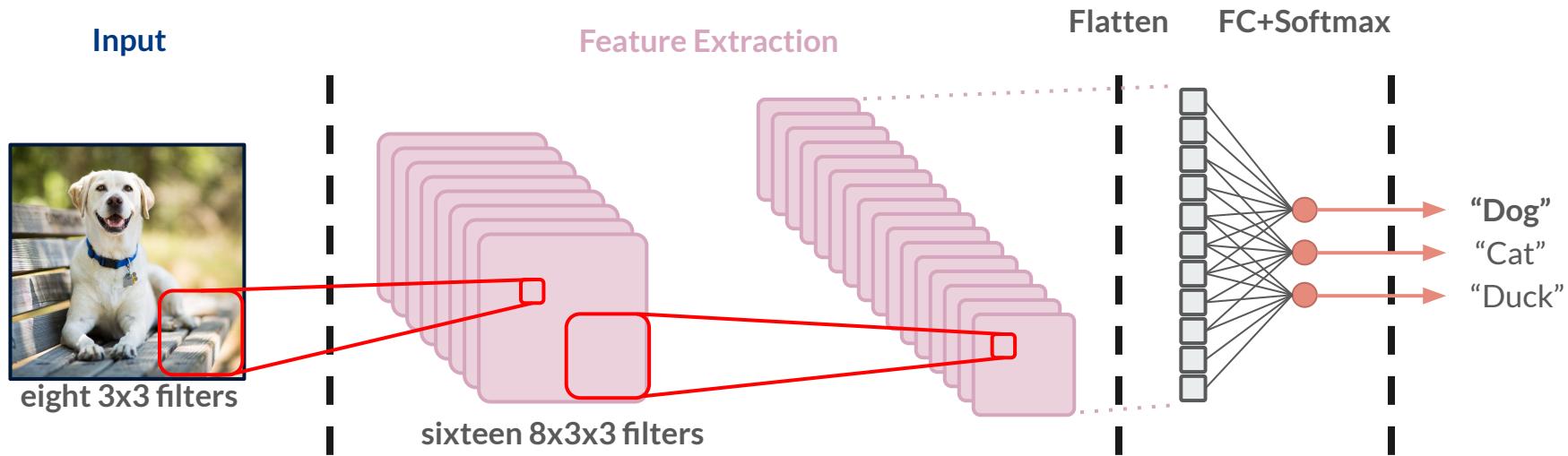
FC: $\sim 2300 \times 8 \sim= 18500$ parameters!

Conv Layer 1

CNN: $8 \times 3 \times 3 + 1 \times 16 = 1168$ parameters

FC: $\sim 4600 \times 16 \sim= 73700$ parameters!

Convolutional Neural Network



Conv Layer 1

CNN: $9+1 \times 8 = 80$ parameters

FC: $\sim 2300 \times 8 \sim= 18500$ parameters!

Conv Layer 1

CNN: $8 \times 3 \times 3 + 1 \times 16 = 80$ parameters

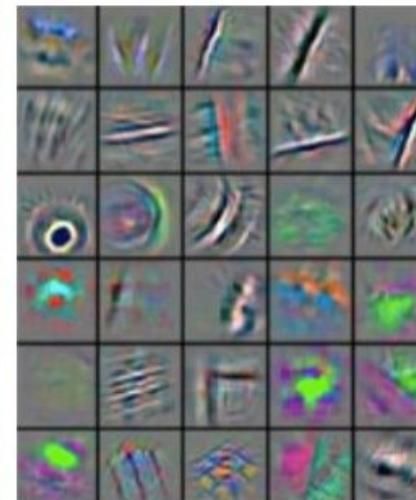
FC: $\sim 4600 \times 16 \sim= 73700$ parameters

Feature Extraction using CNNs

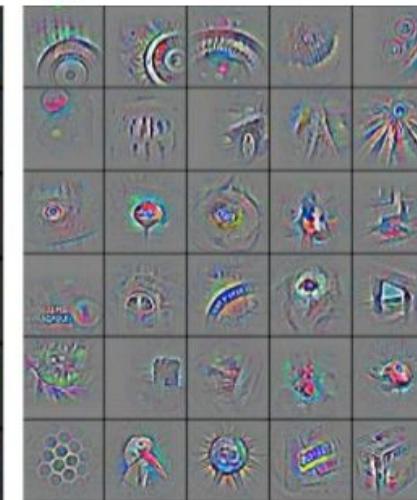
low-level features



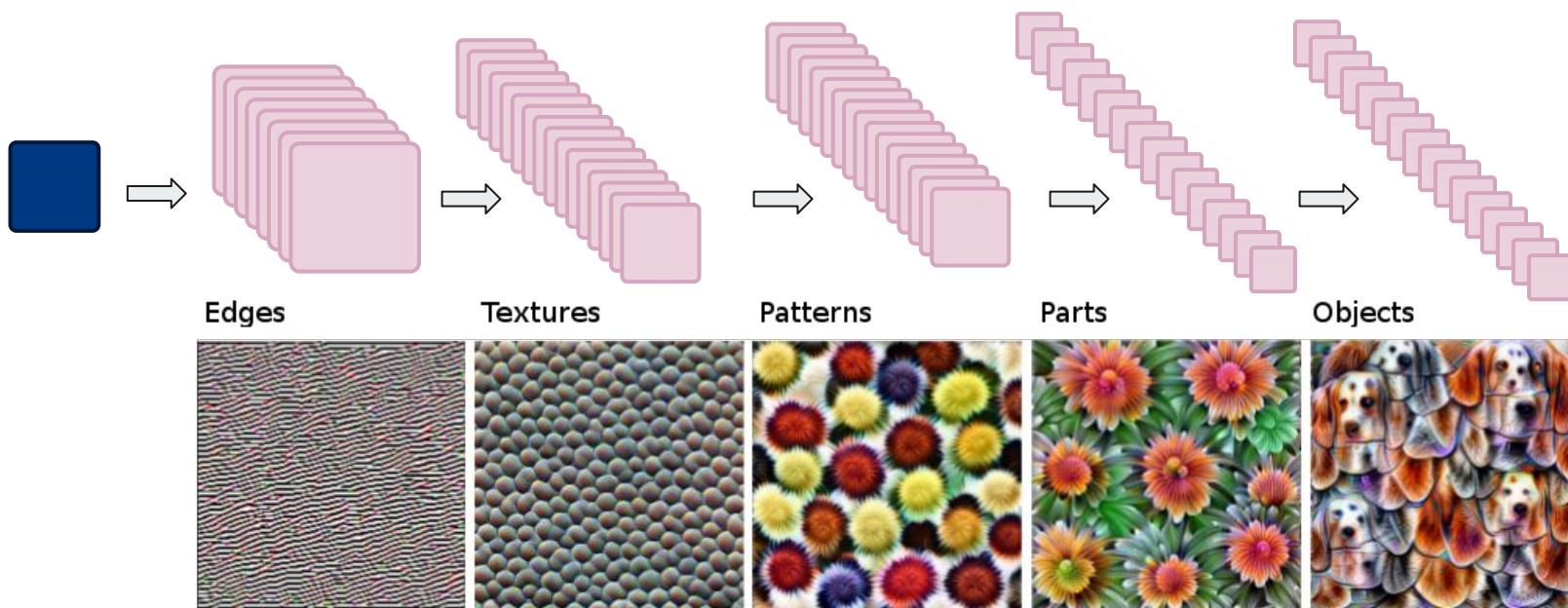
mid-level features



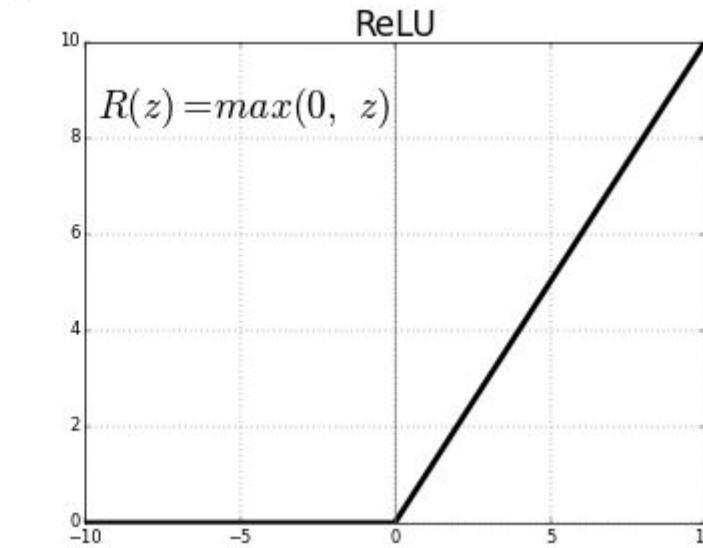
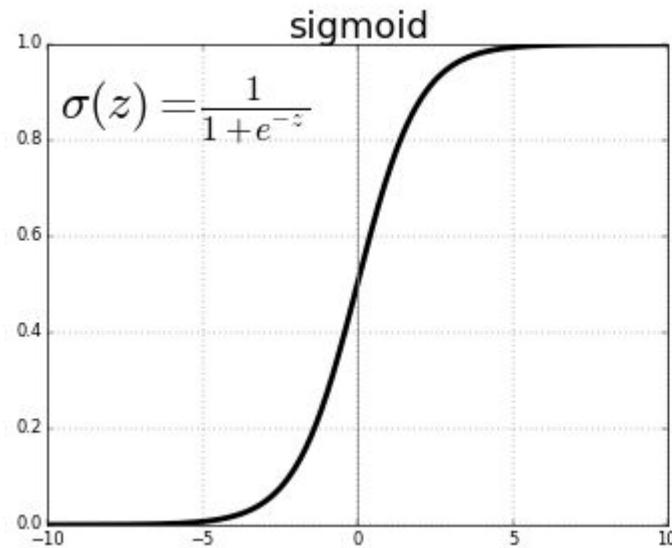
high-level features



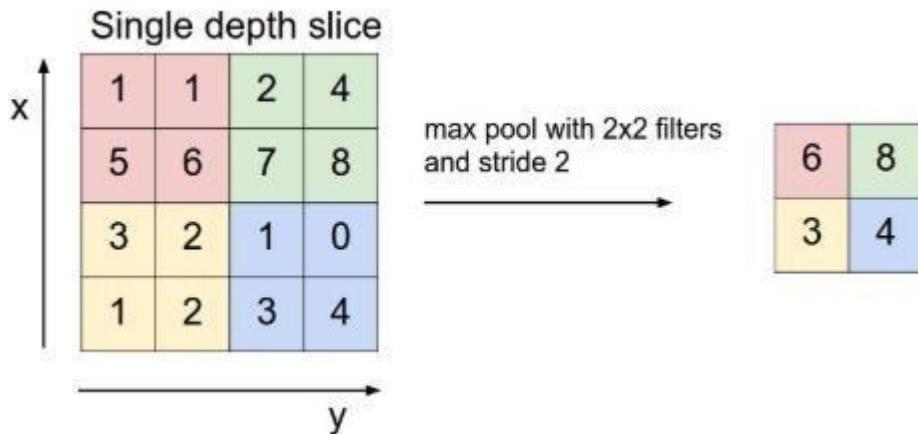
Feature Extraction using CNNs



ReLU vs Sigmoid (vanishing gradients)



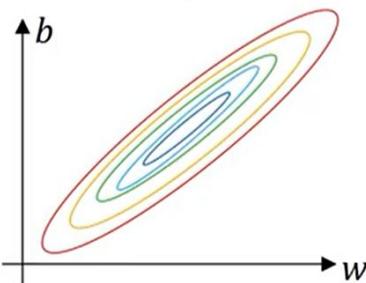
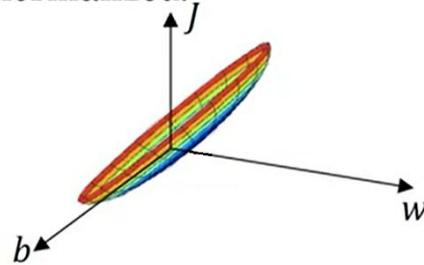
Pooling



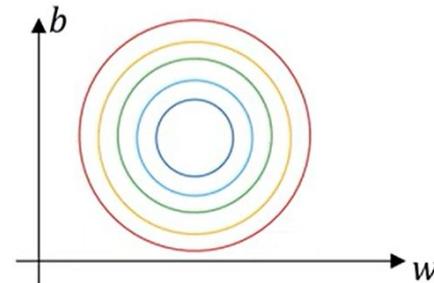
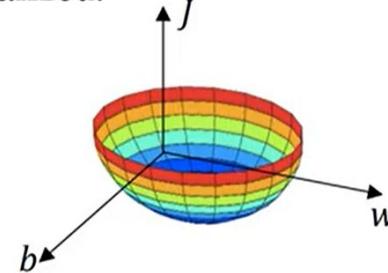
- Pooling Layers are useful for reducing the dimensionality across layers
- Save only the useful information
- Diverse type of pooling
 - Max Pooling
 - Average Pooling

Batch Normalization

Unnormalized:



Normalized:

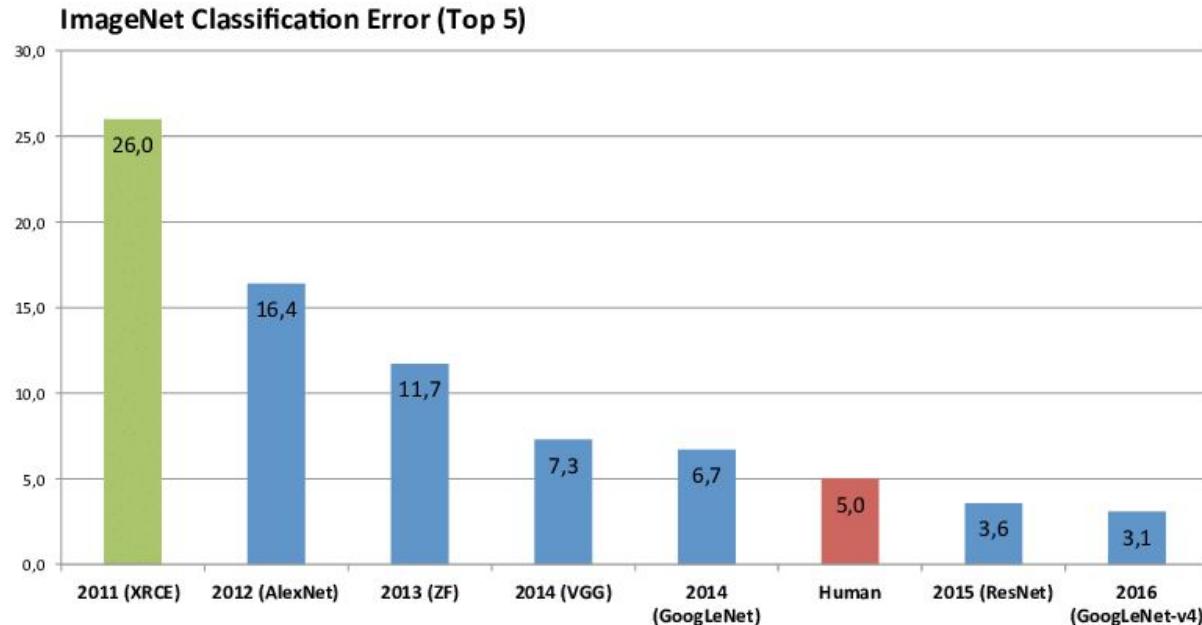


State-of-the-art CNNs

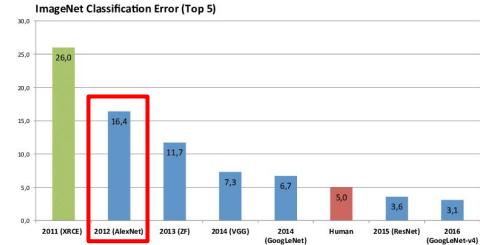
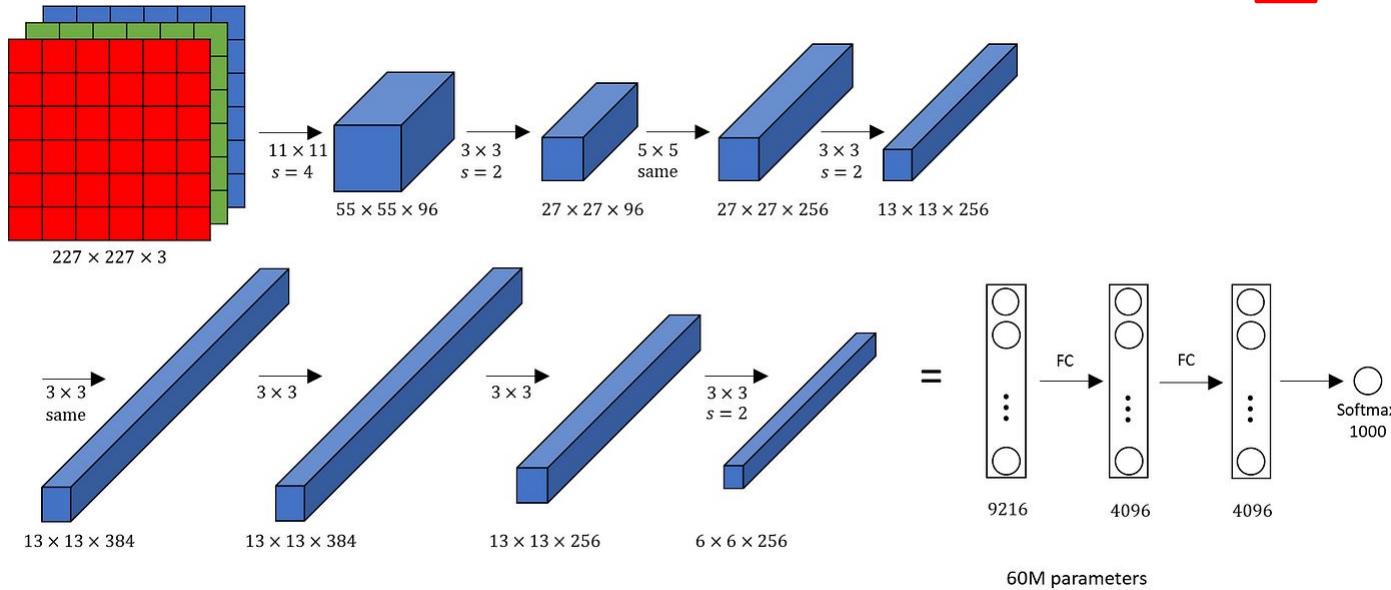
Classification



Classification



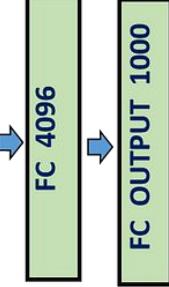
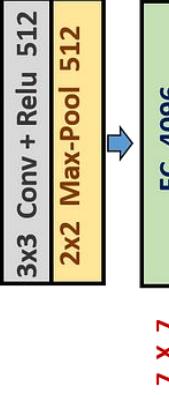
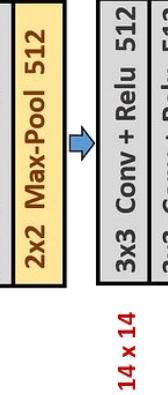
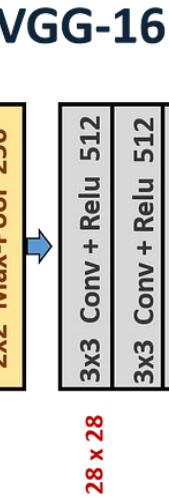
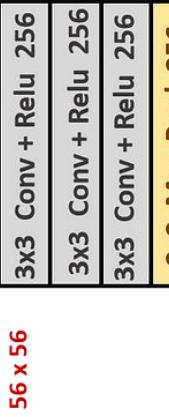
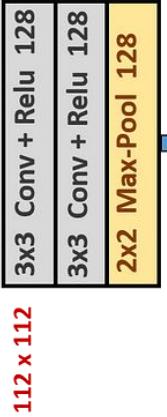
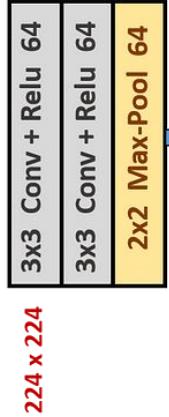
AlexNet



VGG

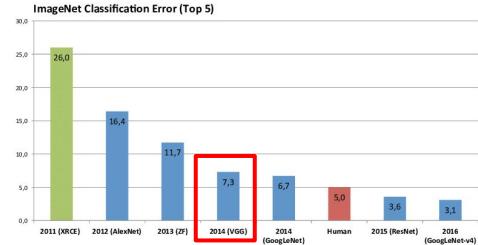
INPUT

224 x 224

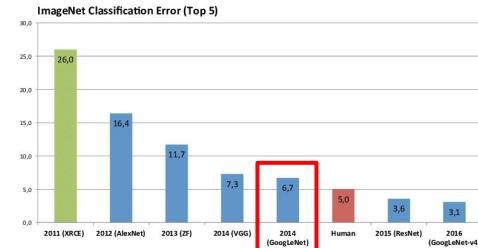
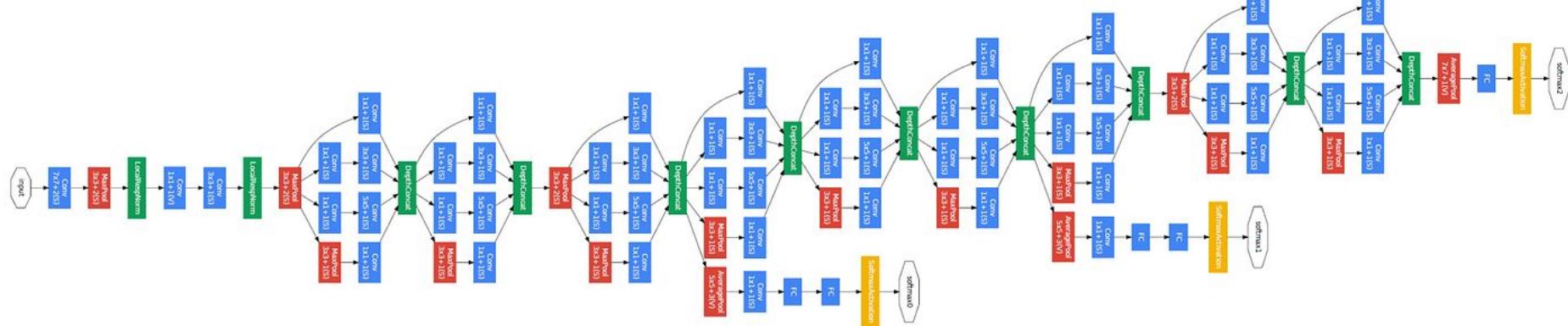


OUTPUT

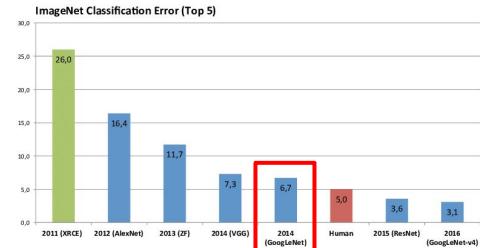
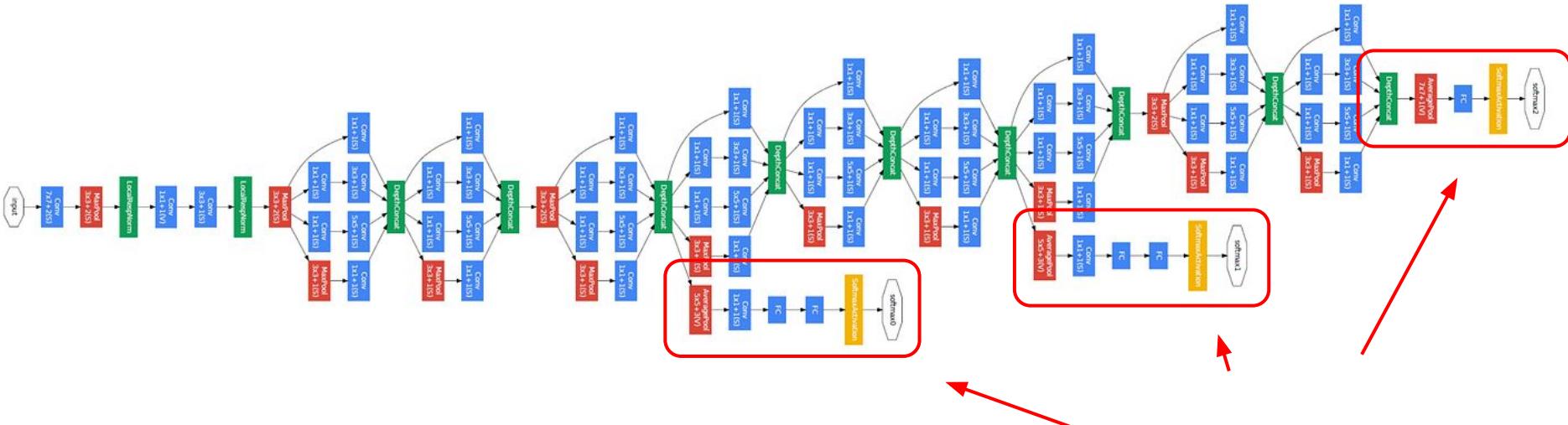
FC OUTPUT 1000



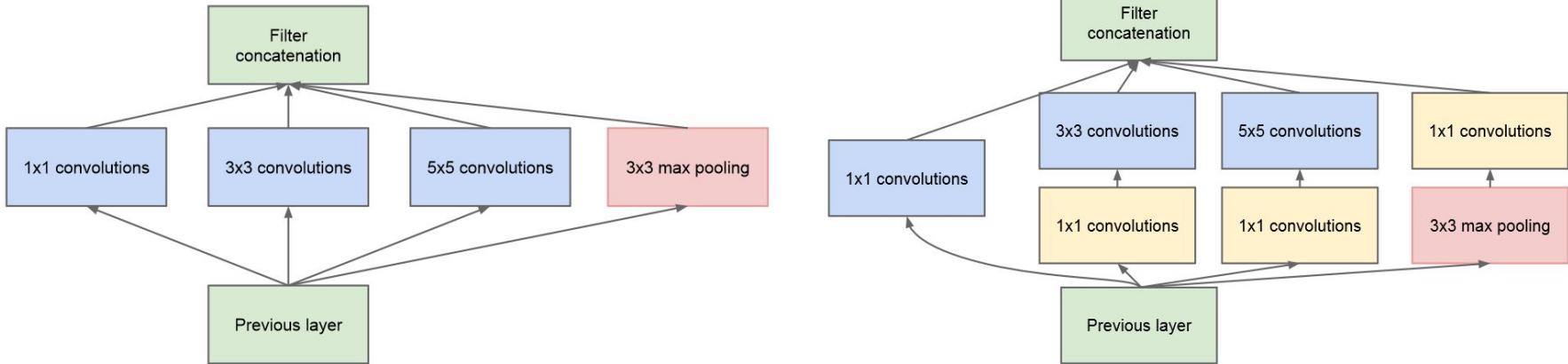
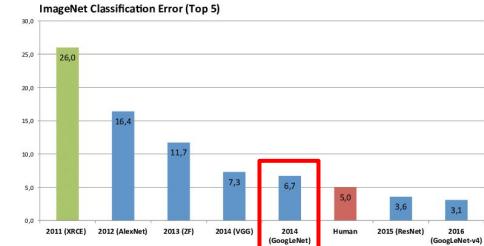
GoogLeNet



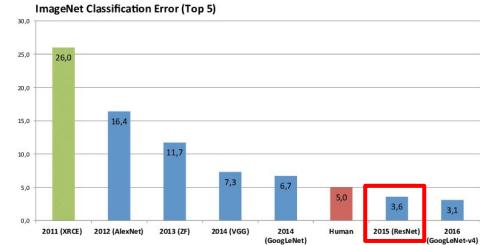
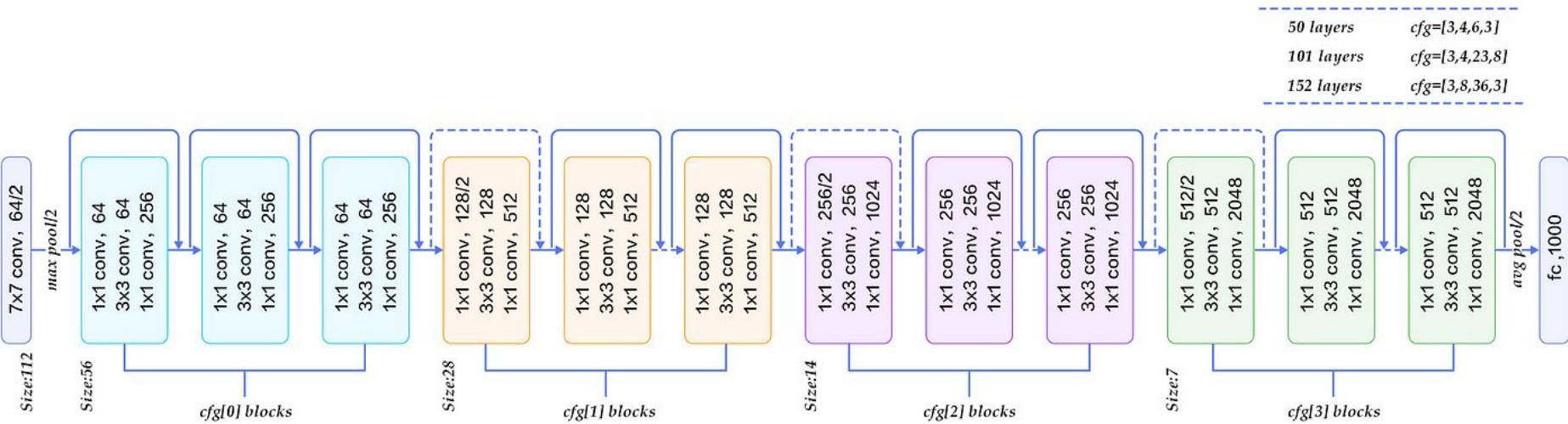
GoogLeNet



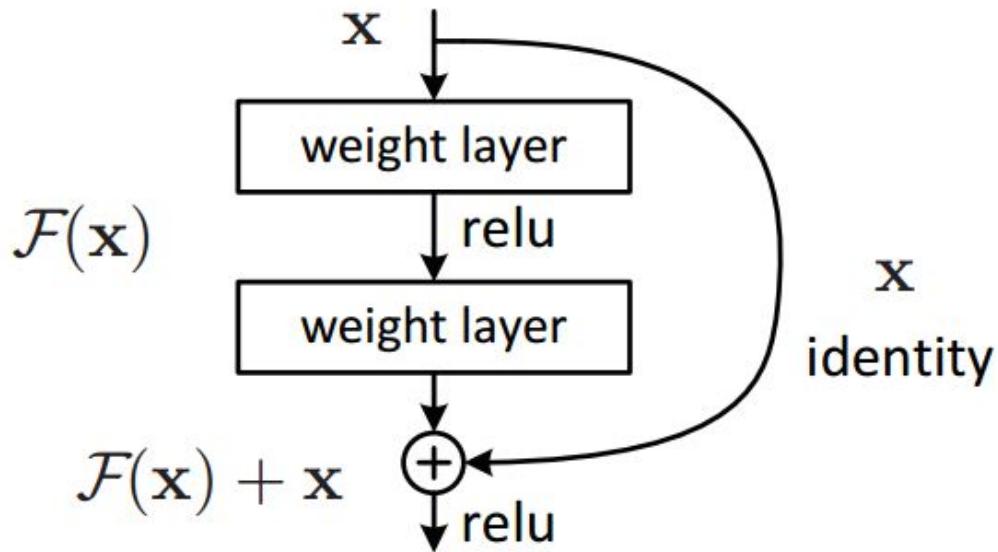
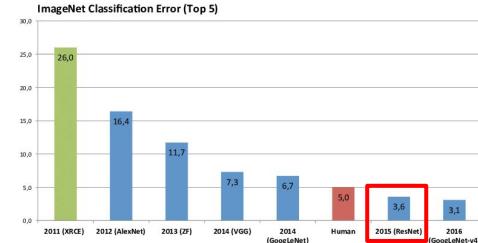
GoogLeNet - Blocks



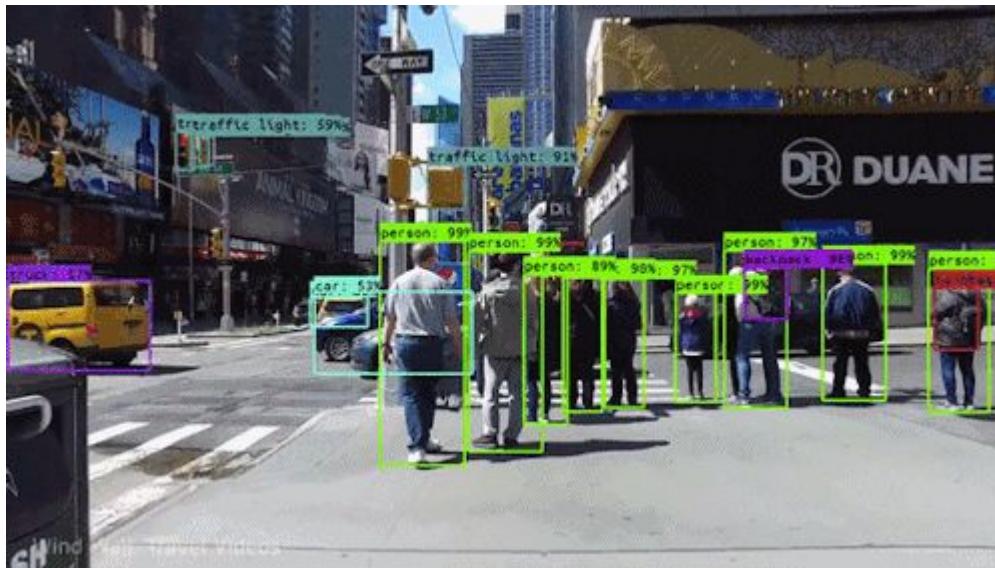
ResNet



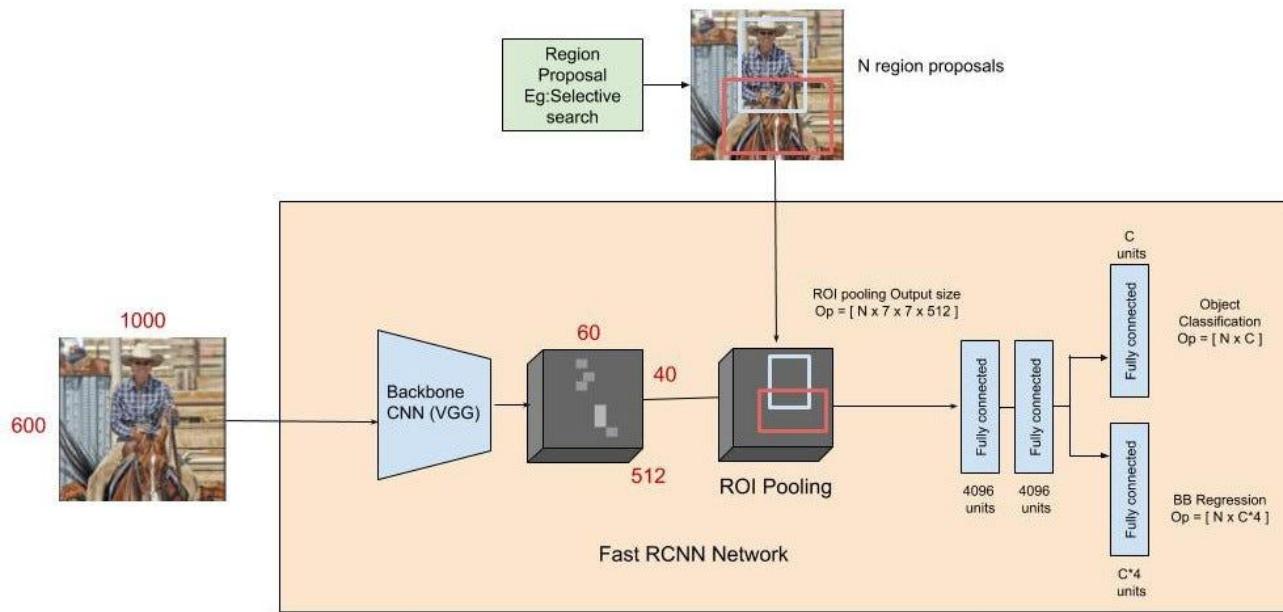
ResNet - Residual Blocks



Object Detection



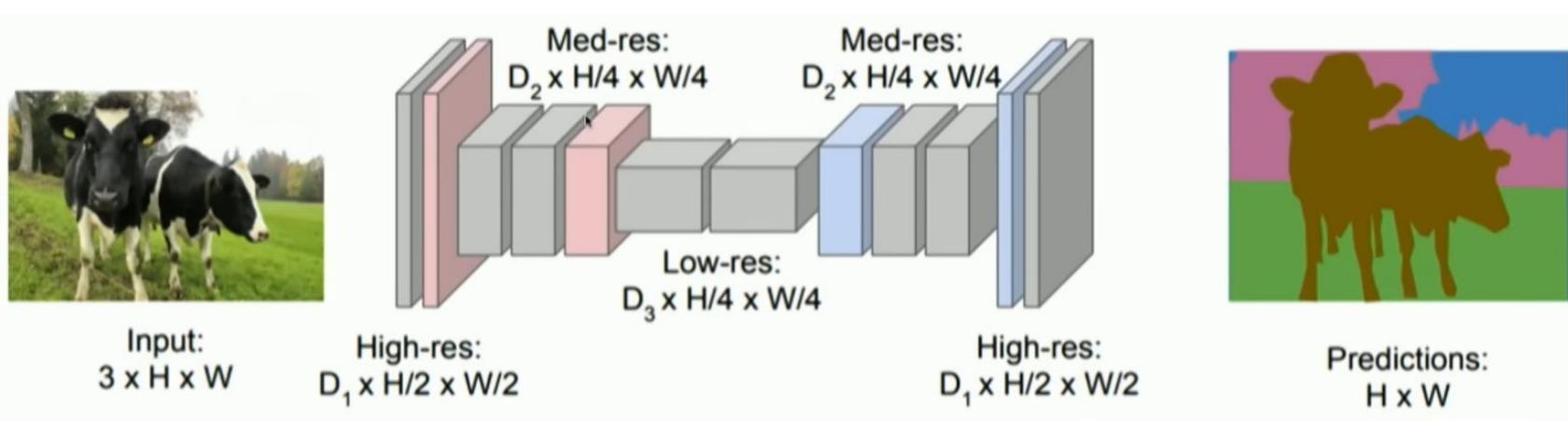
Object Detection (R-CNN)



Semantic Segmentation



Semantic Segmentation (U-Net)



Unpooling Layer

- We need to obtain a higher dimensionality
- We can expand the pixels using the same idea used for the pooling
- For each pixel we expand to many pixels

