Introductory Seminar on Artificial Intelligence and Machine Learning

Emanuele Ledda, Cagliari Digital Lab 2024 - Day 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 nets 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 fac and the time of its occurence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts o the neuron. The velocity along the axon varies directly with its diameter, from <1 ms^{-1} in thin axons, which are usually short, to >150 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 unequally 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 irreciprocity 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 o neighboring synapses within the period of latent addition, which lasts <0.25 ms. Observed temporal summation of impulses at greater interval

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

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

J. Y. LETTVIN†, H. R. MATURANA‡, 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 exciting aspect of that stimulus such that variations in everything else cause little change in the response. It has been known for the past 20 years that each fiber is connected not to a few rods and cones in the retina but to very many over a fair area. Our results show that for the most part within that area, it is not the light intensity itself but rather the pattern of local variation of intensity that is the exciting factor. There are four types of fibers, each type concerned with a different sort of pattern. Each type is uniformly distributed over the whole retina of the frog. Thus, there are four distinct parallel distributed channels whereby the frog's eye informs his brain about the visual image in terms of local pattern independent of average illumination. We describe the patterns and show 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 movements enter his hunting and evading habits only, *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 relays to cerebral cortex. Thus, we chose to work 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 does not seem to see or, at any rate, is not concerned with the detail of stationary parts of the world around him. He will starve to death surrounded by food if it is not moving. 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 it moves like one. He can be fooled easily not only by a bit of dangled 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 anything more devious than leaping to where it is darker. Since he is equally at home in water and on land, why should it matter where he lights after jumping or what particular direction he takes? He does remember a moving thing providing it stays within his field of vision and he is not distracted.

Anatomy of Frog Visual Apparatus

The retina of a frog is shown in Fig. 1(a). Between the rods and cones of the retina and the ganglion cells, whose axons form the optic nerve, lies a layer of connecting neurons (bipolars, horizontals, and amacrines). In the frog there are about 1 million receptors, $2\frac{1}{2}$ to $3\frac{1}{2}$ million connecting neurons, and half a million ganglion cells [1]. The connections are such that there is a synaptic path from a rod or cone to a great many ganglion cells, and a ganglion cell receives paths from a great many 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 in the optic nerve.

There is only one layer of ganglion cells in the frog. These cells are half a million 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 μ to 500 μ, interlace widely into what is called the inner plexiform layer, which is a close-packed neuropil containing the terminal arbors of those neurons that lie between receptors and ganglion cells. Thus, the amount of overlap of adjacent ganglion cells is enormous in respect to what they see. Morphologically, there are several types of these cells that are as distinct in their dendritic patterns as different species of trees, 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.

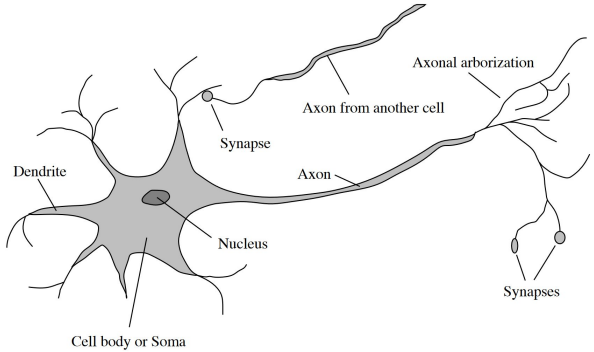
Physiology as Known up to This Study

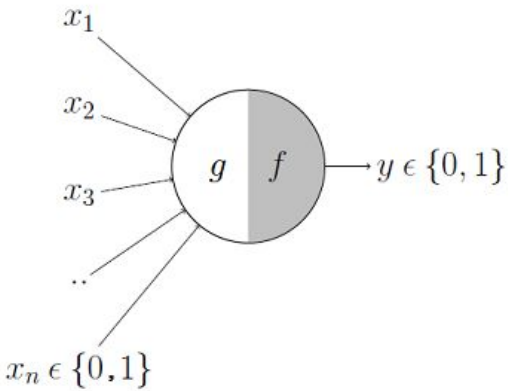
Hartline [2] first used the term *receptive field* for the region of retina within which a local change of brightness would cause the ganglion cell he was observing to discharge. Such a region is sometimes surrounded by an annulus, within which changes of brightness affect the cell's response to what is occurring in the receptive field,

* Original manuscript received by the IRE, September 3, 1959. This work was supported in part by the U. S. Army (Signal Corps), the U. S. Air Force (Office of Sci. Res., Air Res. and Dev. Command), and the U. S. Navy (Office of Naval Res.); and in part by Bell Telephone Labs., Inc.

† Res. Lab. of Electronics and Dept. of Biology, Mass. Inst. Tech., Cambridge, Mass.
‡ Res. Lab. 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.

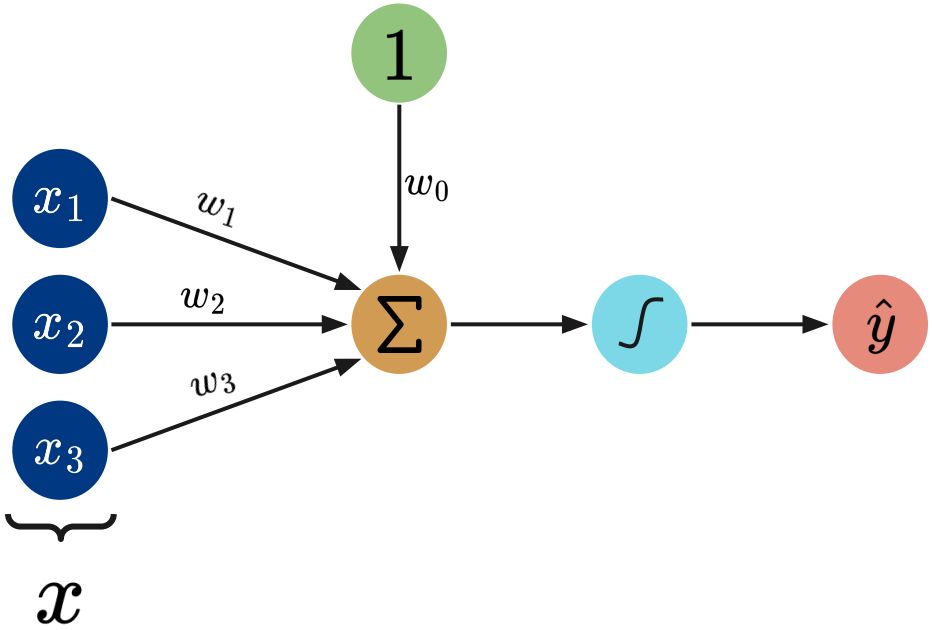
A Long History





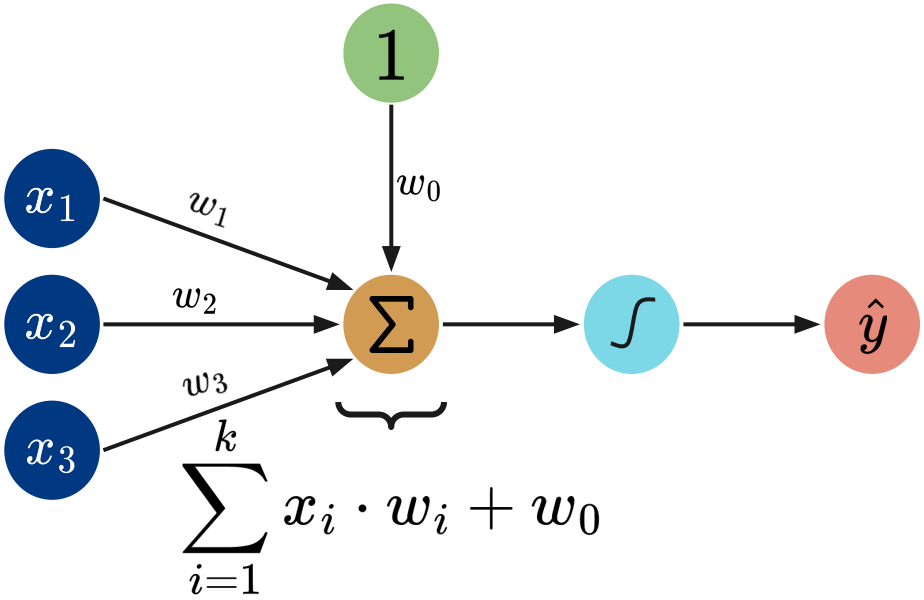
The Perceptron

Perceptron



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

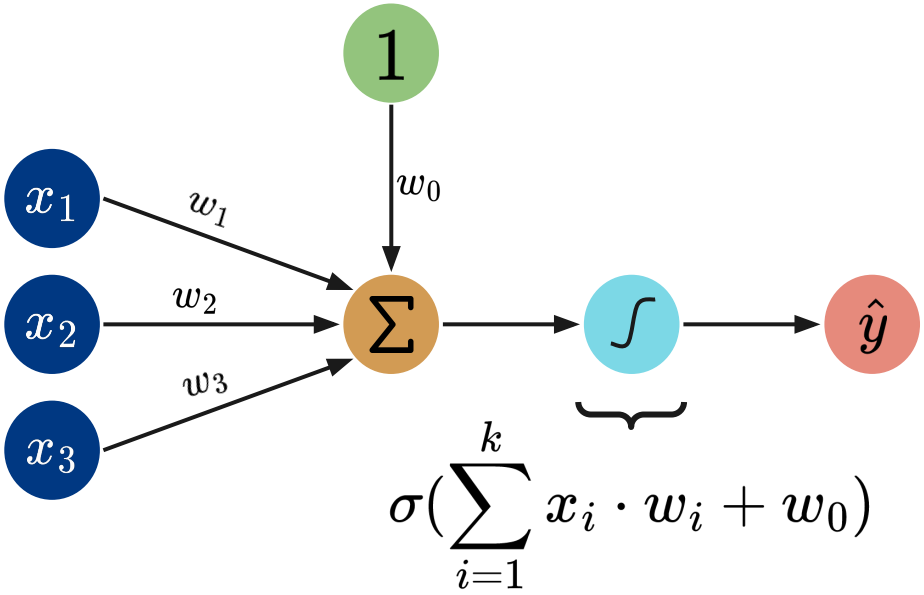
Perceptron



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

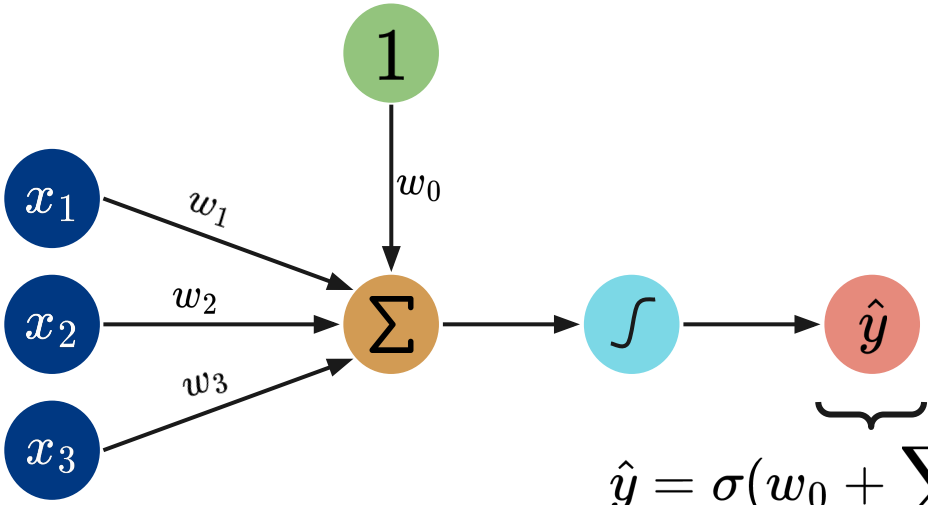
We also add w0, which is called the **bias**

Perceptron

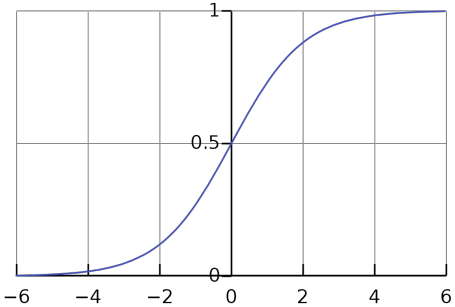


Successively, we apply a nonlinear function called the **"activation function"**

Perceptron

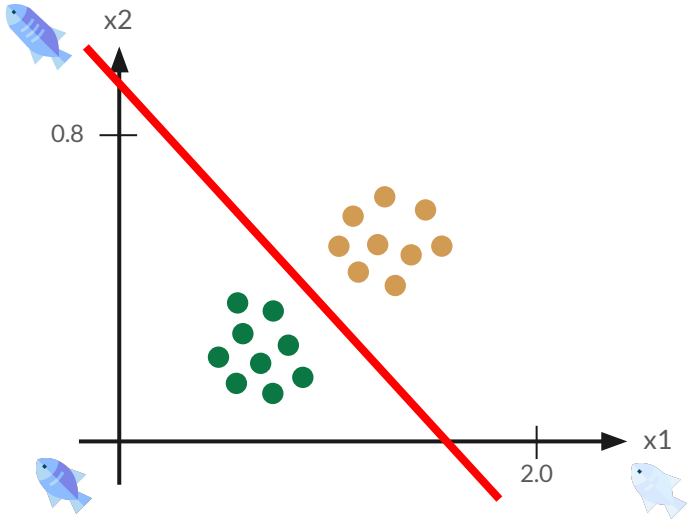


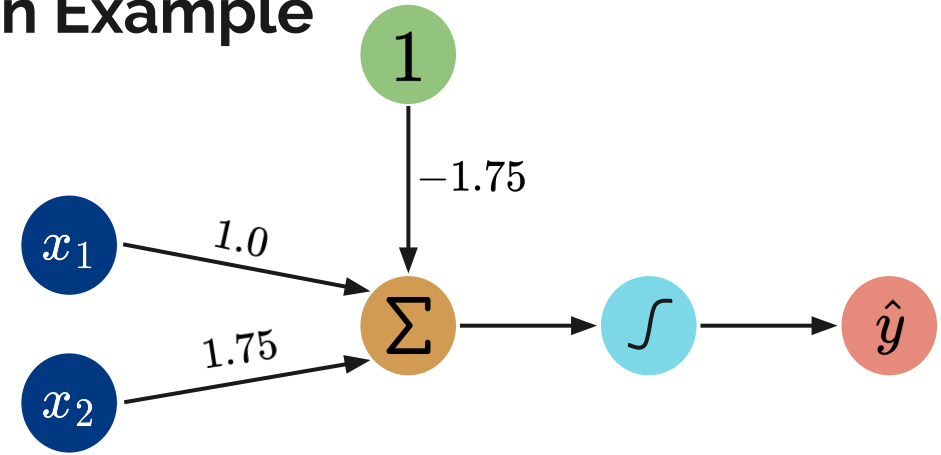
Finally, we obtain the **output** y hat



$$\hat{y} = \sigma(w_0 + \sum_{i=1}^{k} x_i \cdot w_i) \qquad \sigma(x) = \frac{1}{1 + e^{-x}}$$

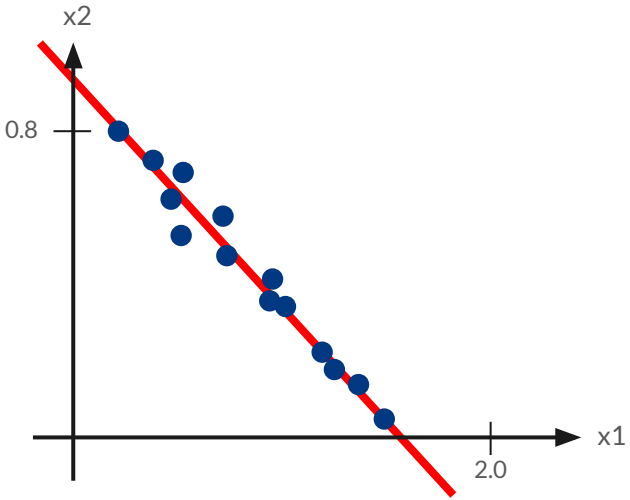
Perceptron: Classification Example

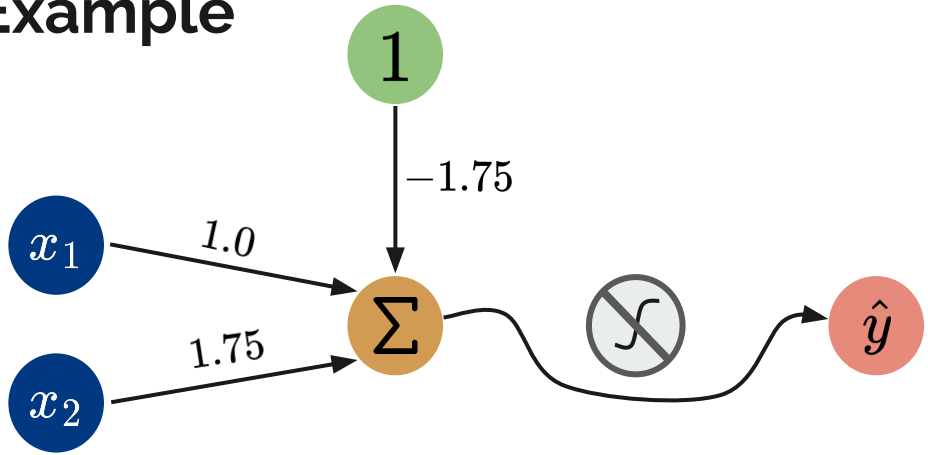




$$y = \sigma(x_1 + x_2 \cdot 1.75 - 1.75)$$

Perceptron: Regression Example

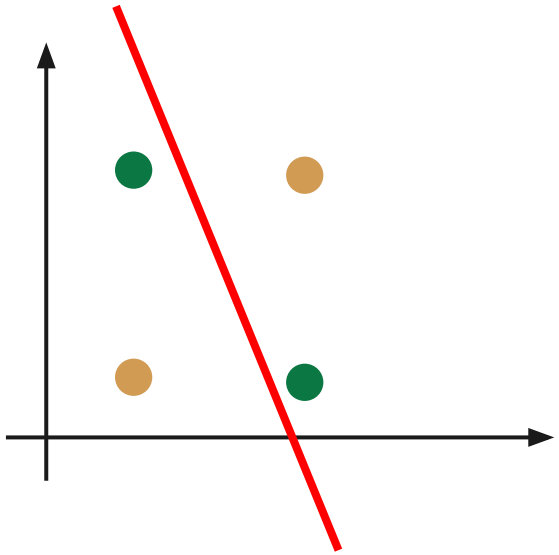


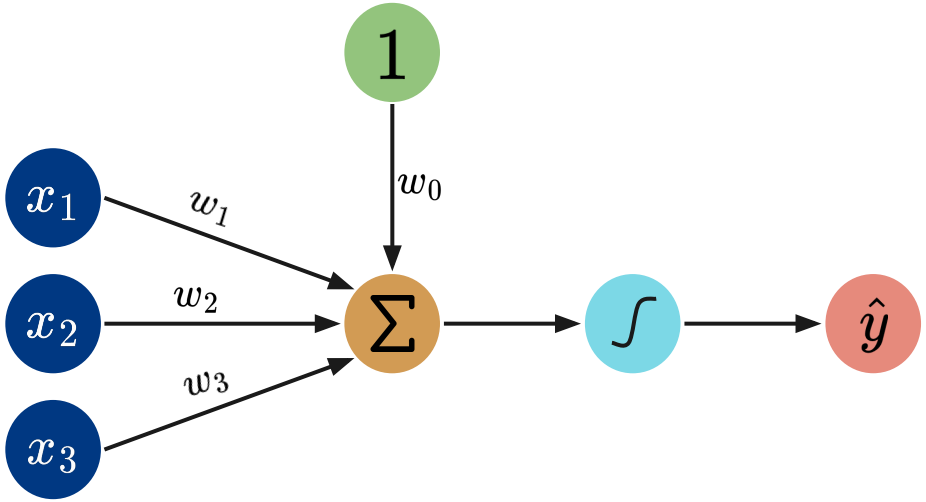


$$y = x_1 + x_2 \cdot 1.75 - 1.75$$

Multi Layer Perceptron

The XOR Problem

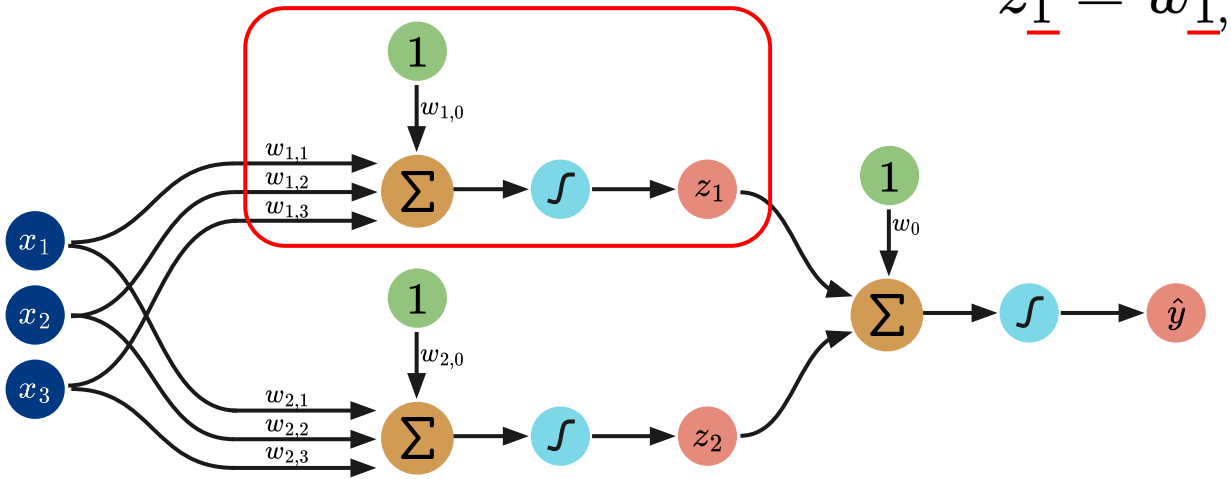




- How do I find a weight configuration for correctly discriminating between these points?
- I need more neurons!

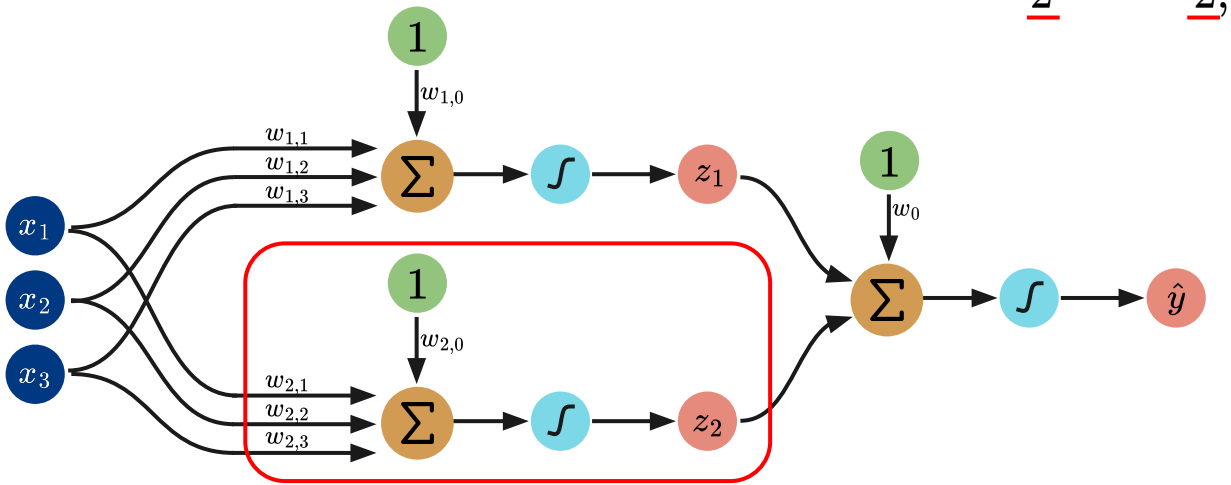
Multi Layer Perceptron

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



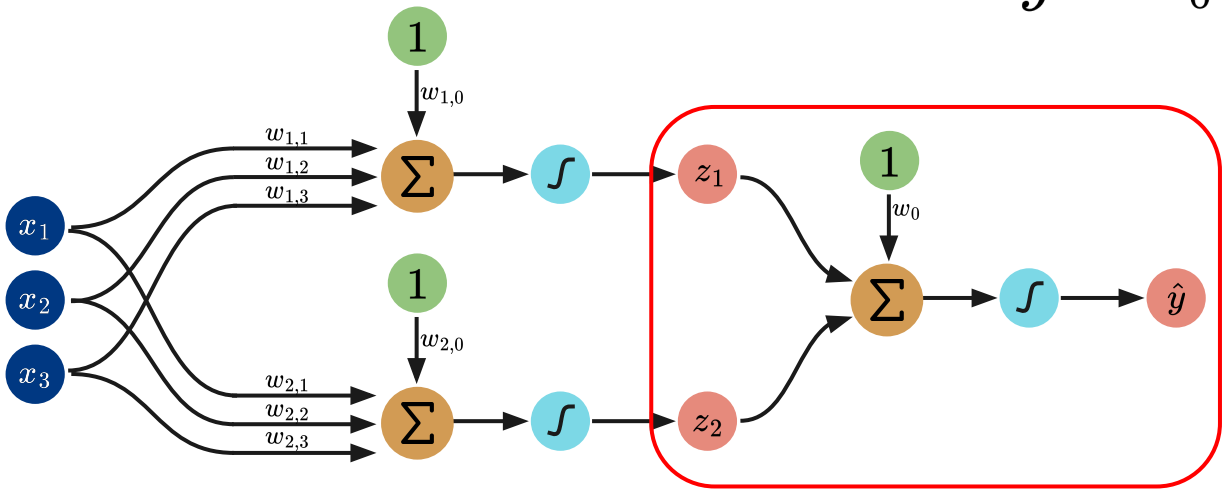
Multi Layer Perceptron

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

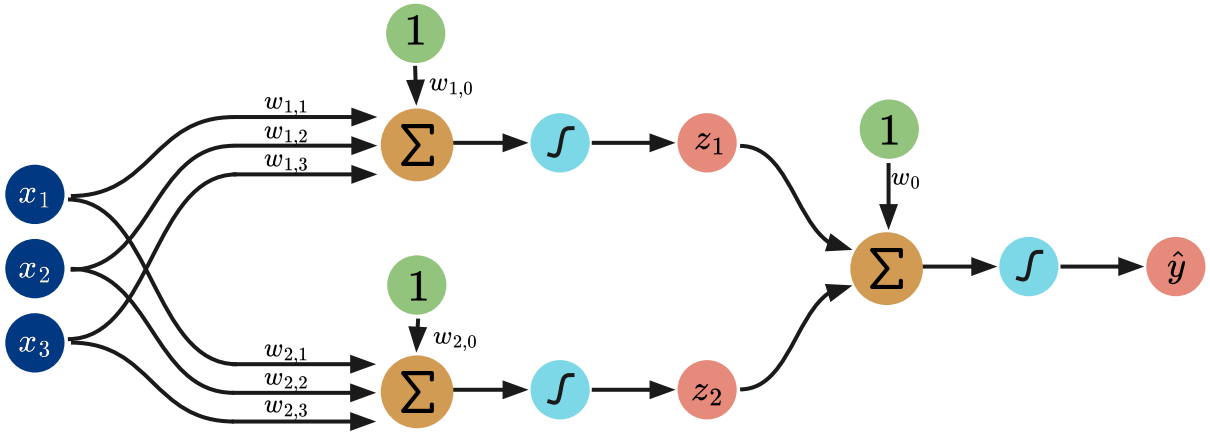


Multi Layer Perceptron

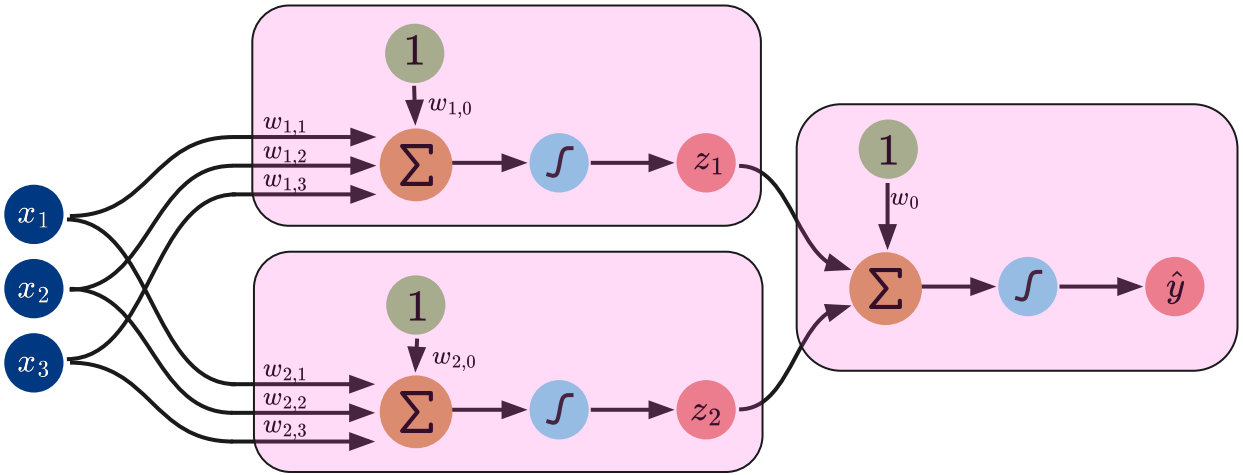
$$\hat{y} = w_0 + \sum_{i=1}^{k} x_i \cdot z_i$$



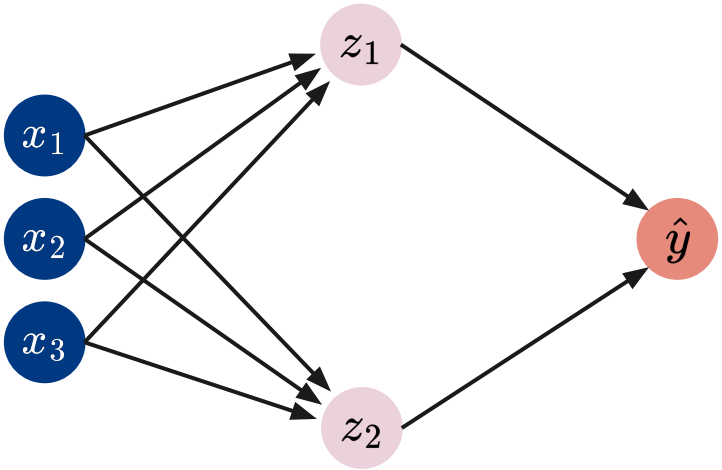
Multi Layer Perceptron



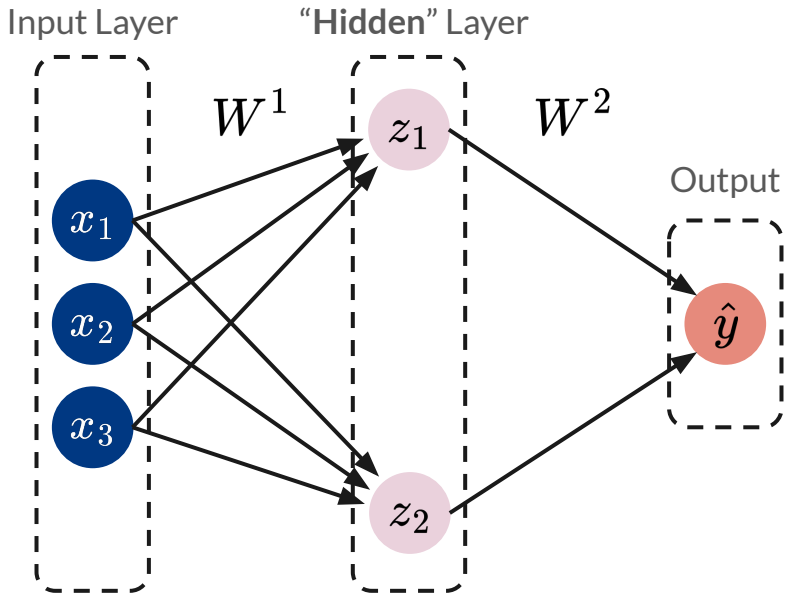
Multi Layer Perceptron



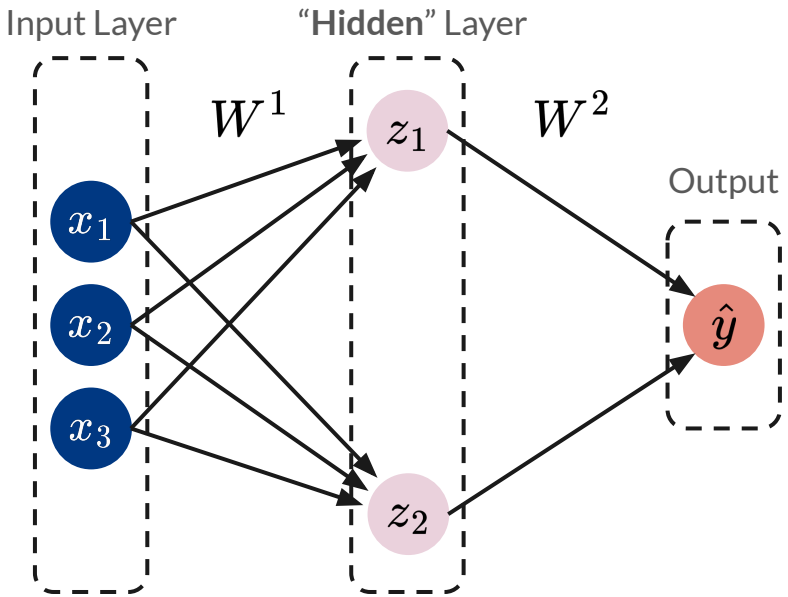
Multi Layer Perceptron



Multi Layer Perceptron



Multi Layer Perceptron



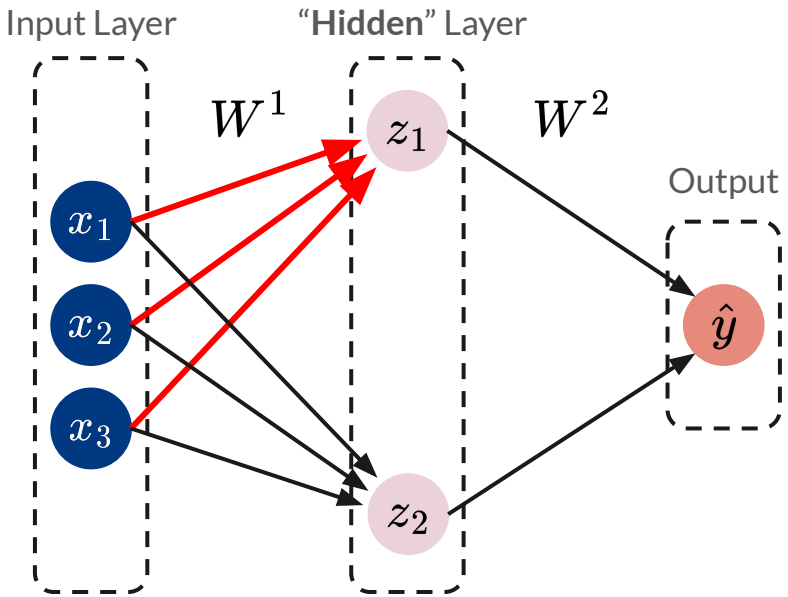
$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

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

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

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

Multi Layer Perceptron



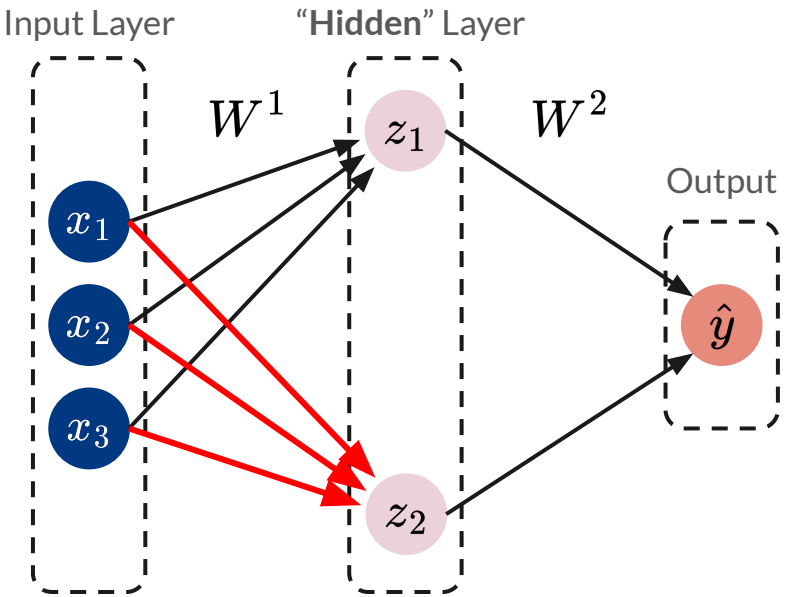
$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

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

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

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

Multi Layer Perceptron



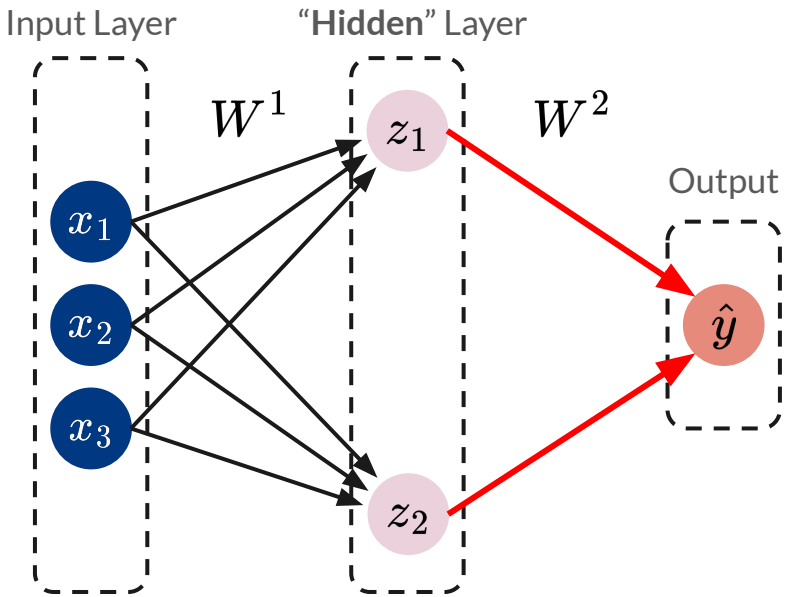
$$W^1 = \begin{bmatrix} b_0^1 & b_1^1 \\ w_{0,1}^1 & w_{1,1}^1 \\ w_{0,2}^1 & w_{1,2}^1 \\ w_{0,3}^1 & w_{1,3}^1 \end{bmatrix} \quad W^2 = \begin{bmatrix} b_0^2 \\ w_{0,1}^2 \\ w_{0,3}^2 \end{bmatrix}$$

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

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

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

Multi Layer Perceptron



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

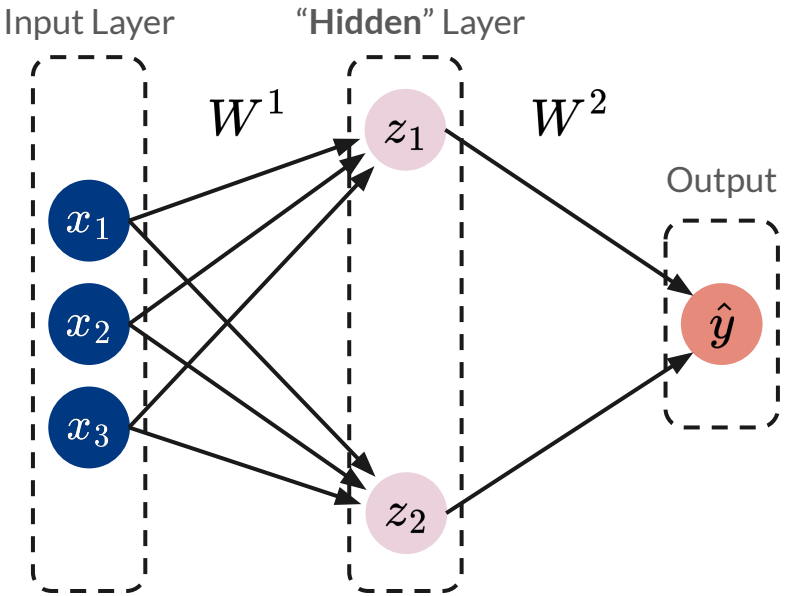
$$x = [x_1, x_2, x_3]$$

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

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

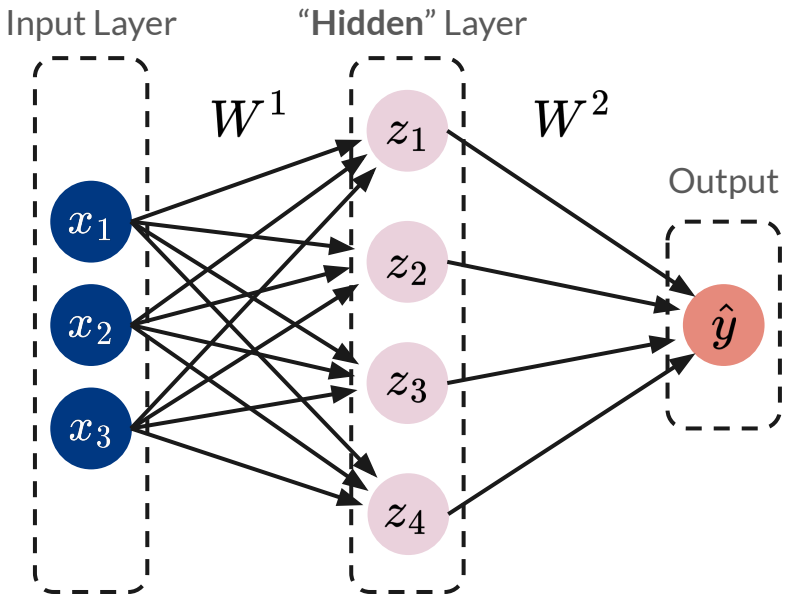
Multi Layer Perceptron

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



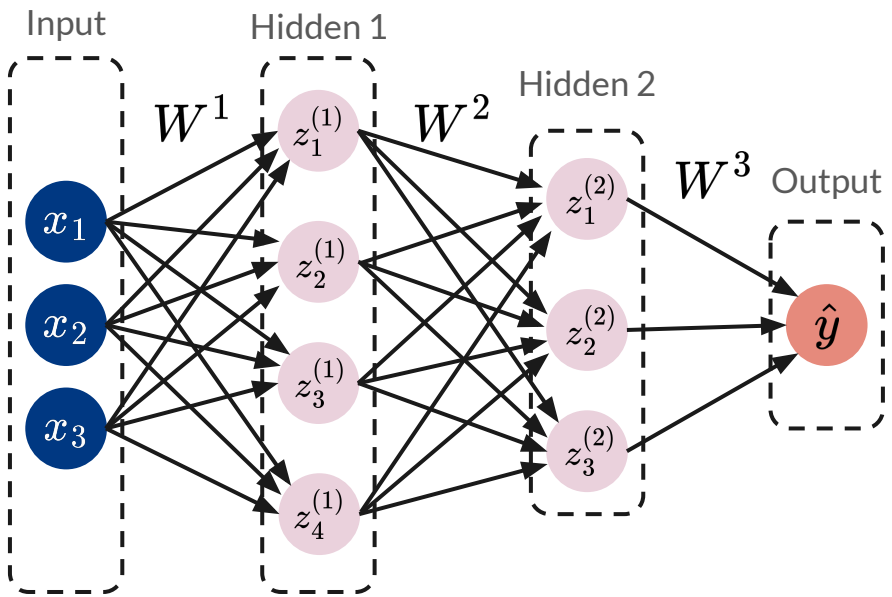
Multi Layer Perceptron

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

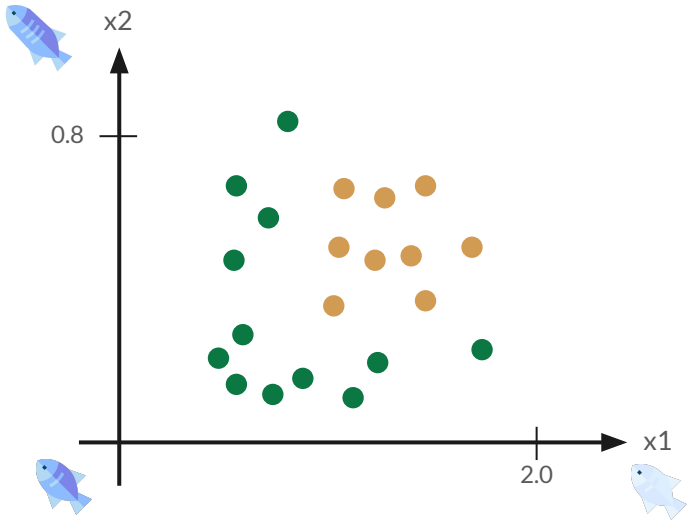


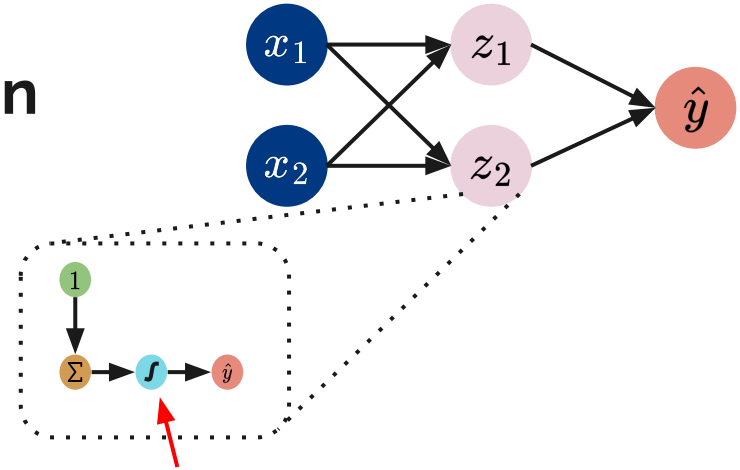
Multi Layer Perceptron

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



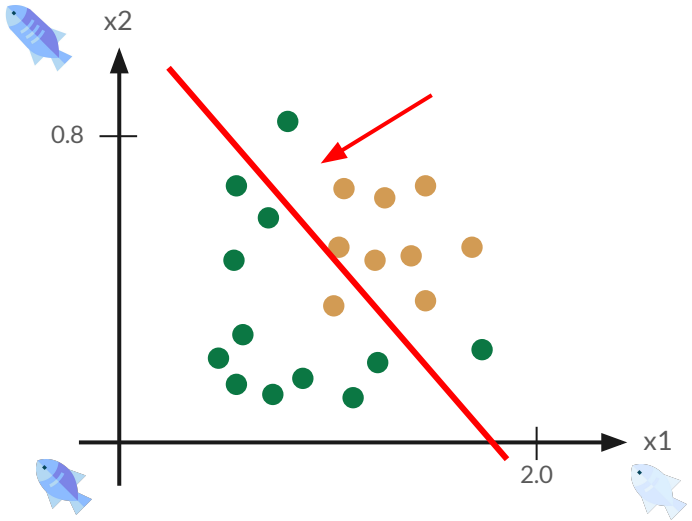
Non-Linear Activation Function

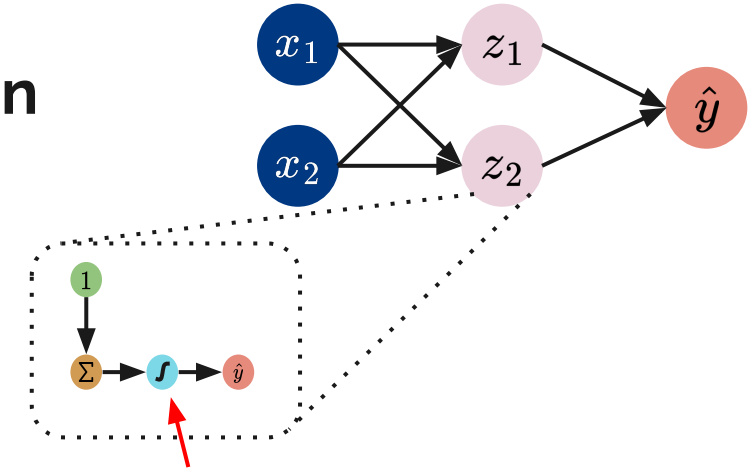




- But why do we need a **non-linear activation function**?

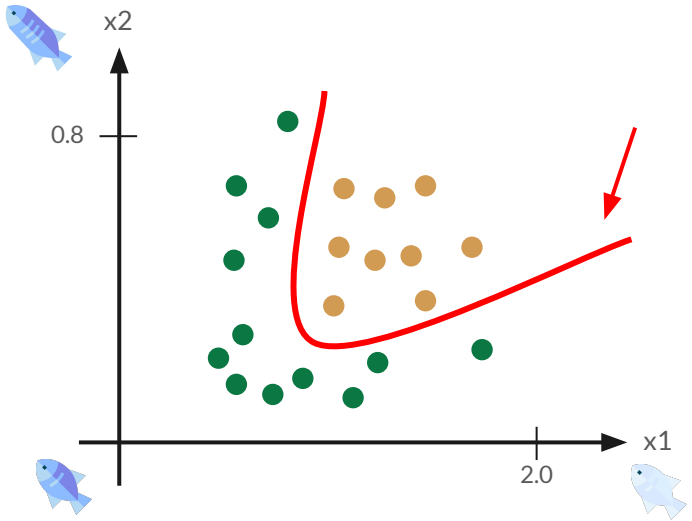
Non Linear Activation Function

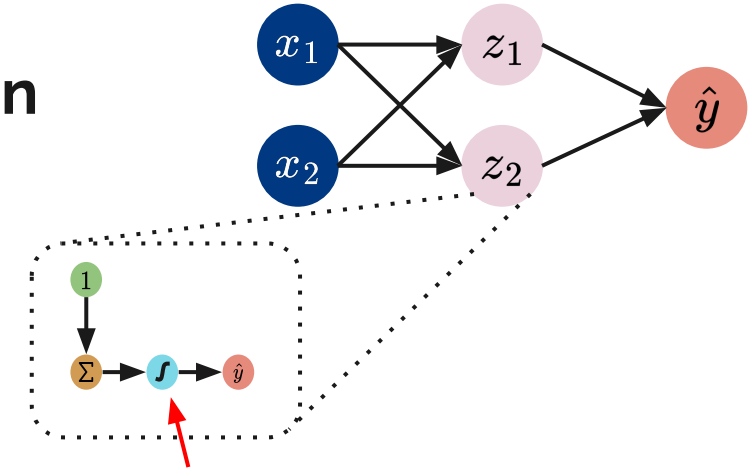




- But why do we need a **non-linear activation function**?
- We can construct **only linear classifiers** using linear activations, no matter how deep the network is!

Non Linear Activation Function



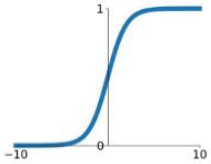


- But why do we need a **non-linear activation function**?
- We can construct **only linear classifiers** using linear activations, no matter how deep the network is!
- The non-linearity of the activation function also lets us **construct non-linear classifiers**

Activation Functions

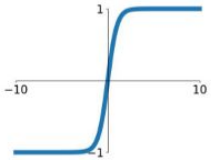
Sigmoid

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



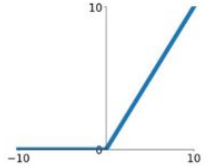
tanh

$$\tanh(x)$$



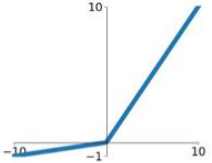
ReLU

$$\max(0, x)$$



Leaky ReLU

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

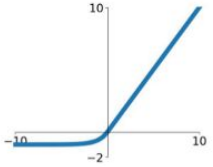


Maxout

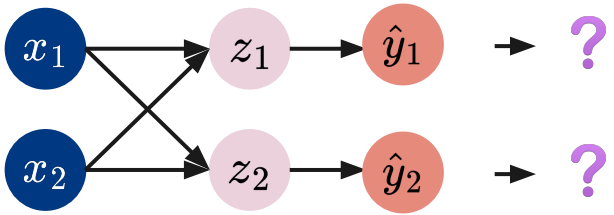
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

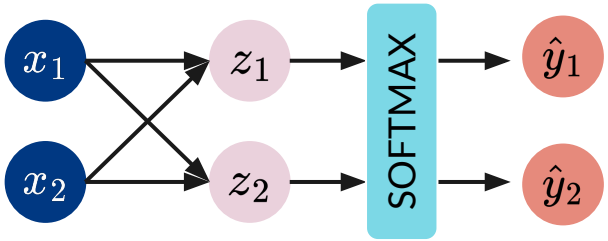


Multi-Class Problems



- 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!

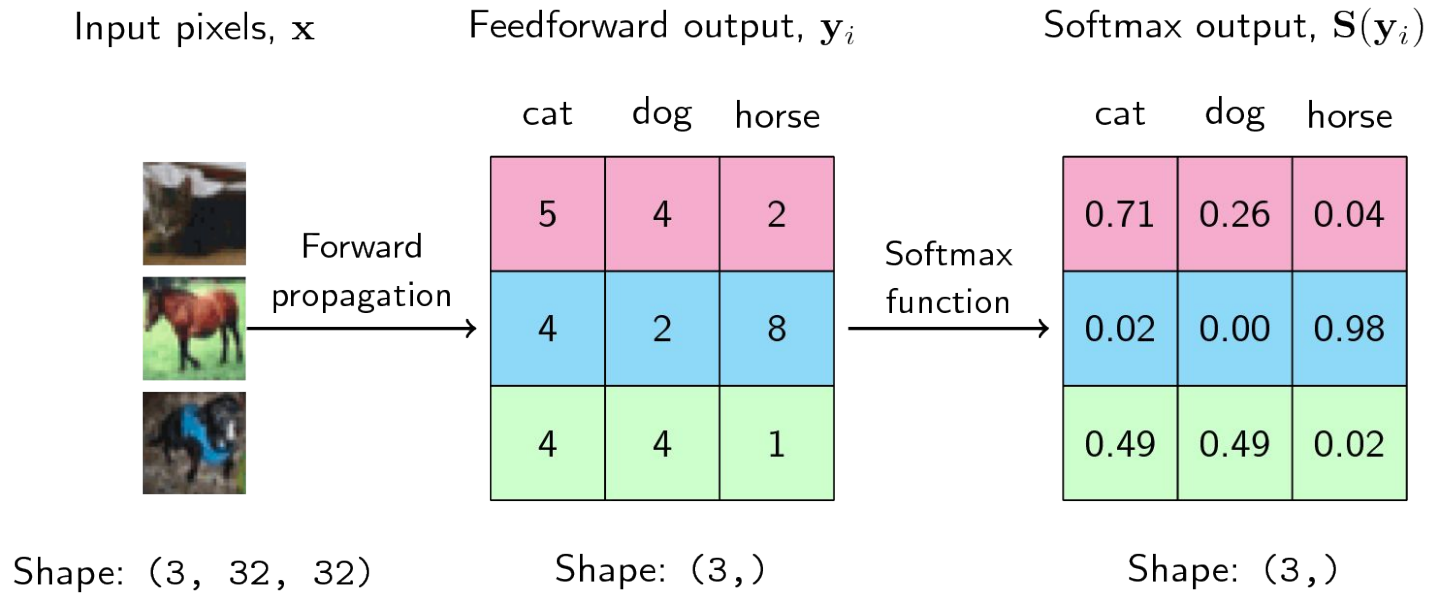
Multi-Class Problems: Softmax Function



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

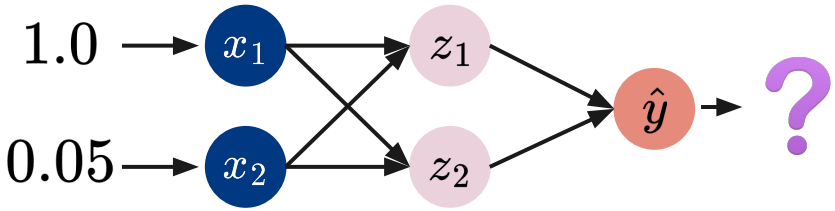
- Instead of using a single neuron we can use a neuron for each class
- But how can we obtain a probability distribution over the classes?
- We need to model a categorical distribution!

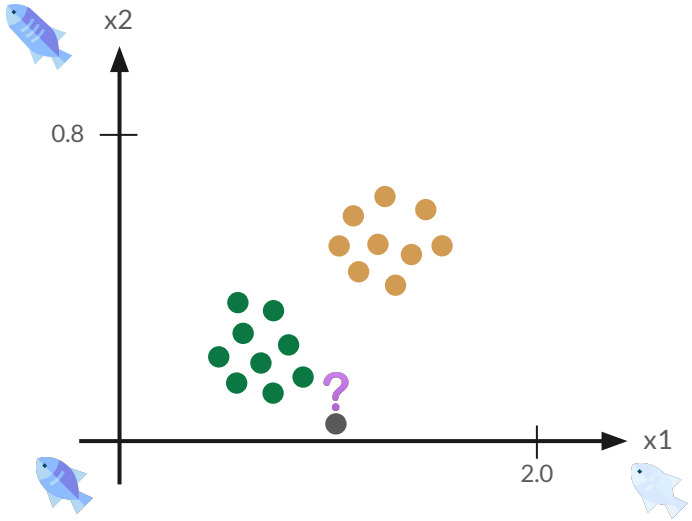
Softmax Function



Training a Neural Network

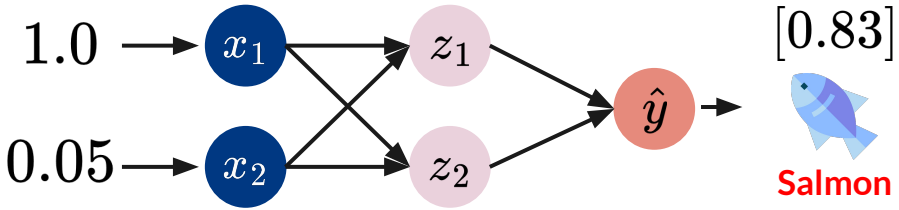
How to obtain useful weights

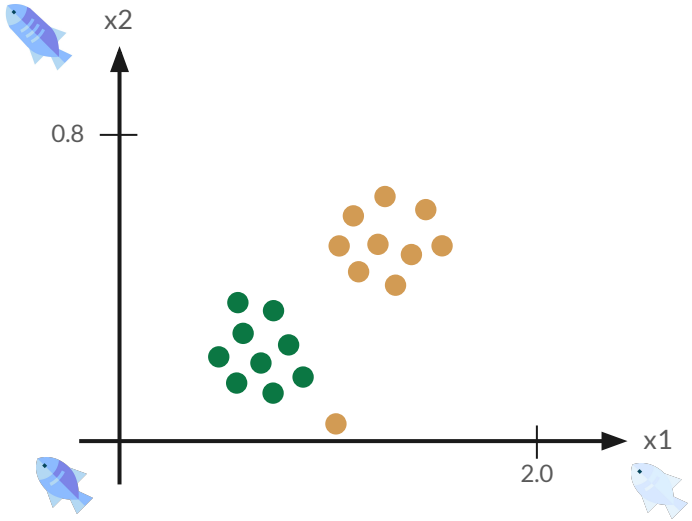




- At first the network weights are **random**

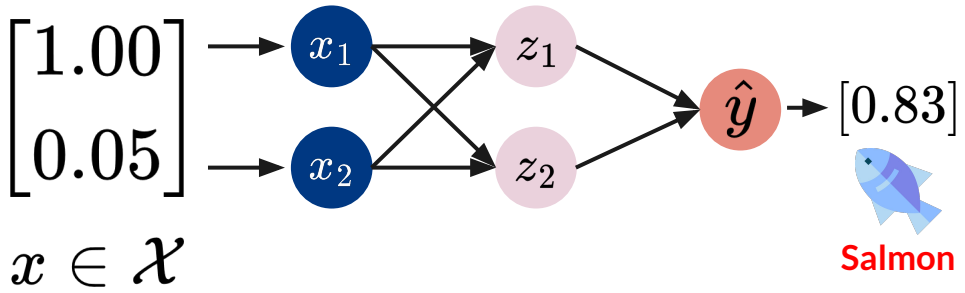
How to obtain useful weights



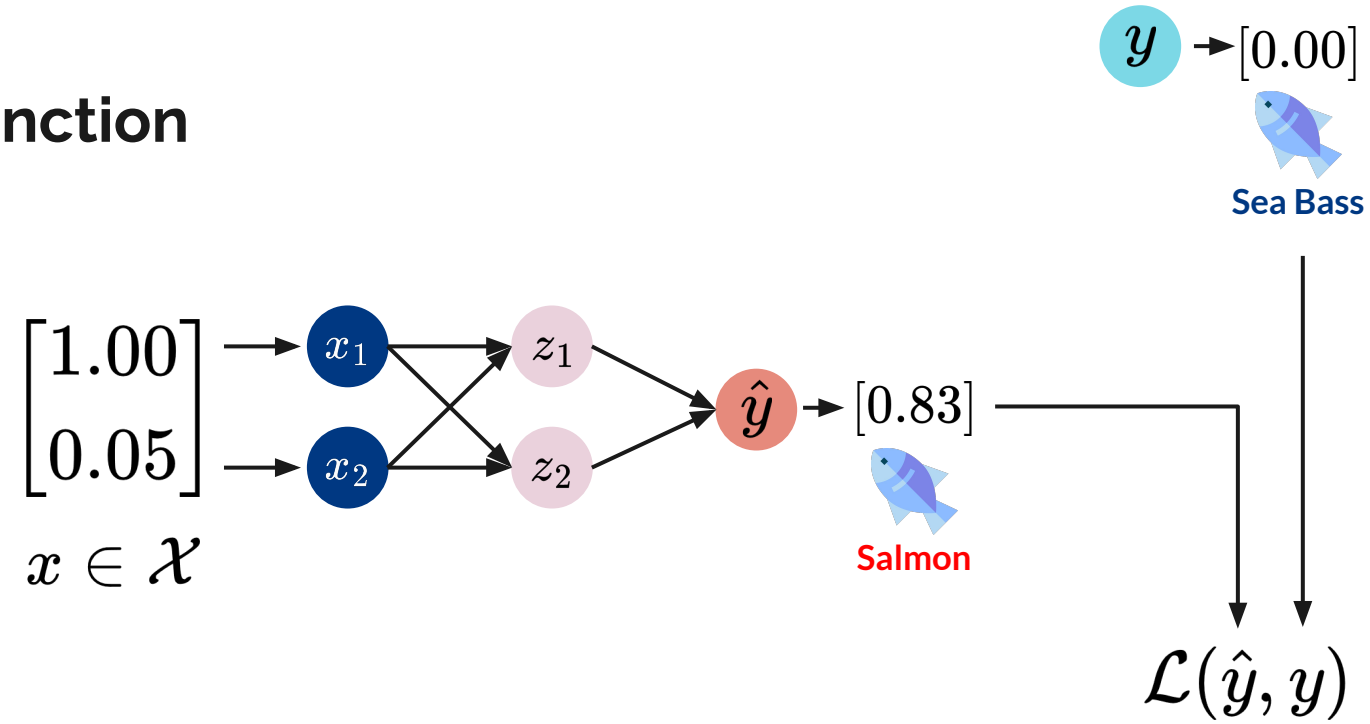


- At first the network weights are **random**
- In this case the **output is useless**!
- We should define a **learning algorithm**

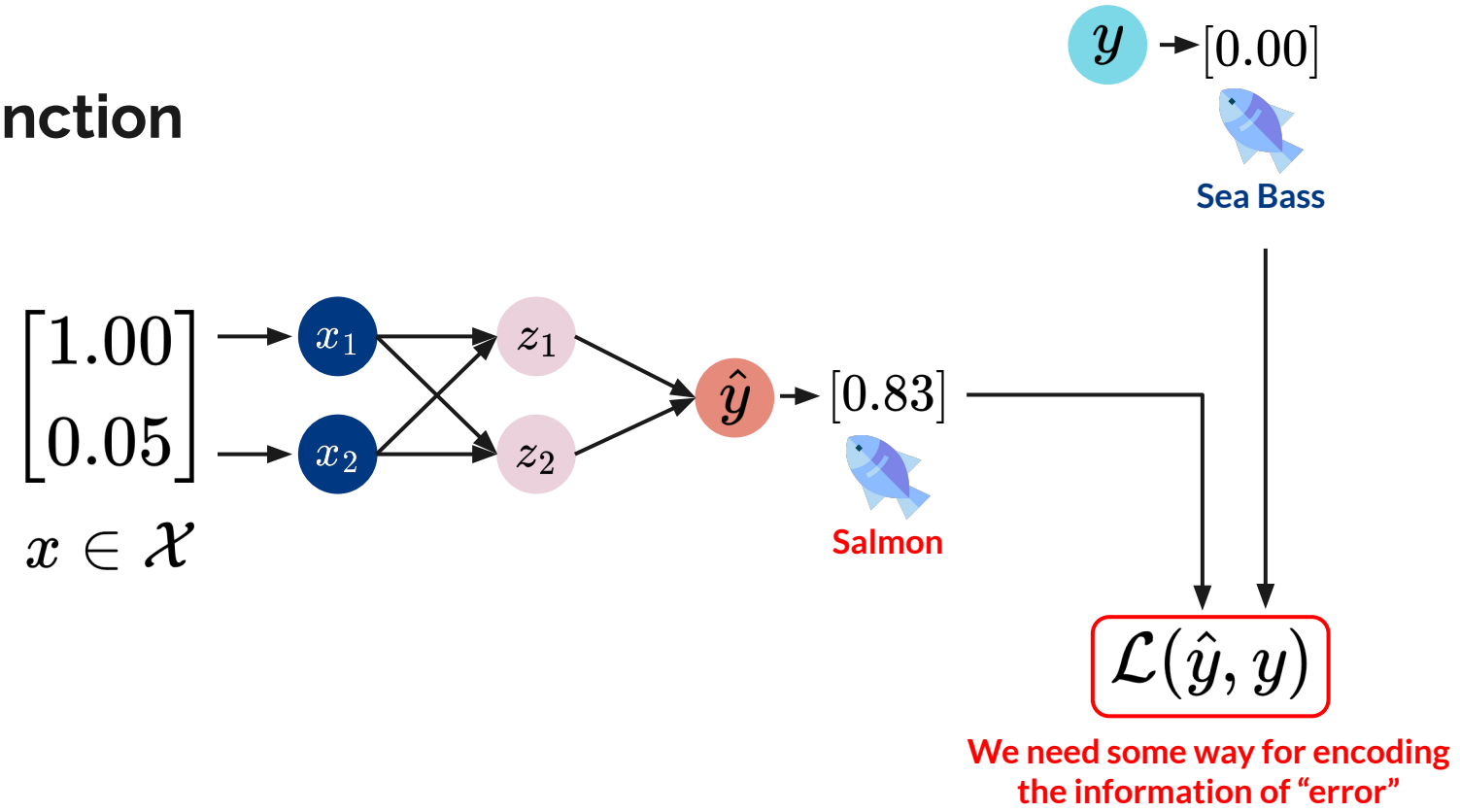
Loss Function



Loss Function



Loss Function



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

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

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

$$= -\log(0.17) = \boxed{1.77}$$

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

Output (as a probability distribution)

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

Label (as a probability distribution)

Cross Entropy - Example

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

$$H(\hat{y}, y) = -0 \log(0.21) - 1 \log(0.79)$$

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

- 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)

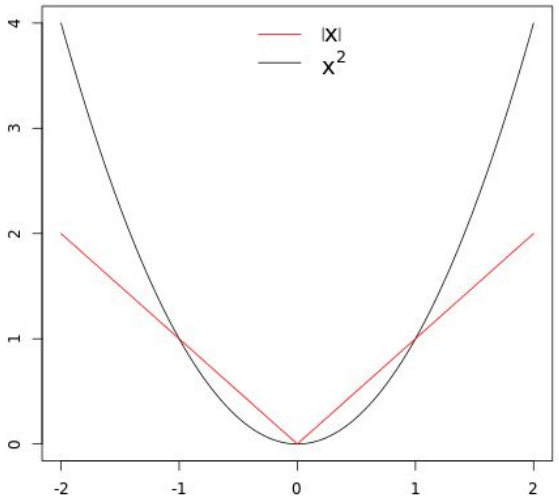
23.5

Output

25

Label

Mean Square Error

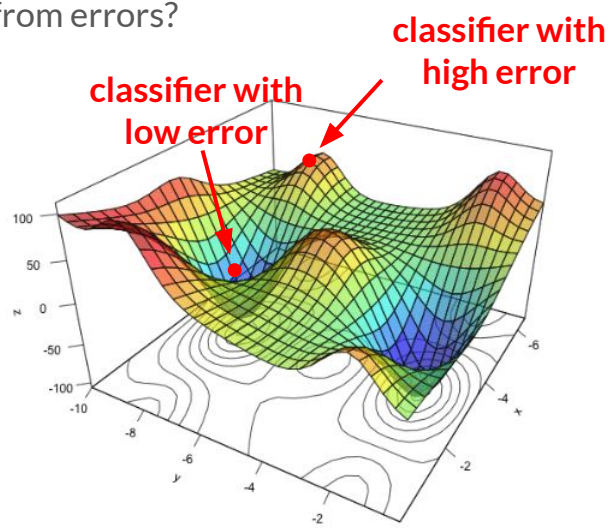


$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

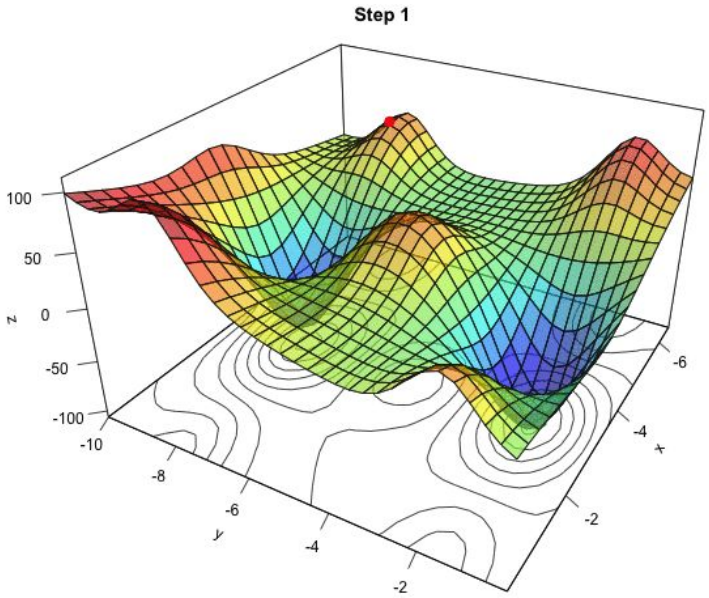
$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

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**

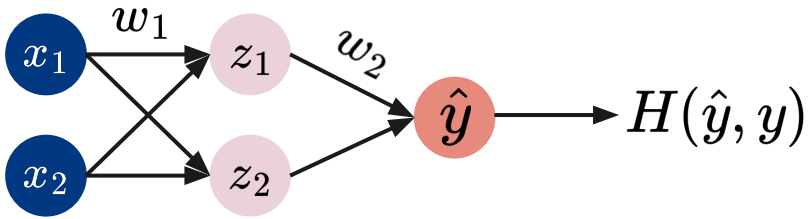


Gradient Descent



- The idea consists of "following" the **direction leading toward lower regions on the loss surface: Gradient Descent Algorithm**.
- The **gradient** of the loss function gives us information about the "**direction**" and the "**magnitude**" of the **slope of the surface**.

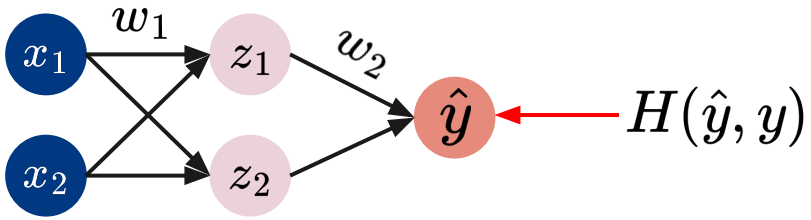
Backpropagation



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

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

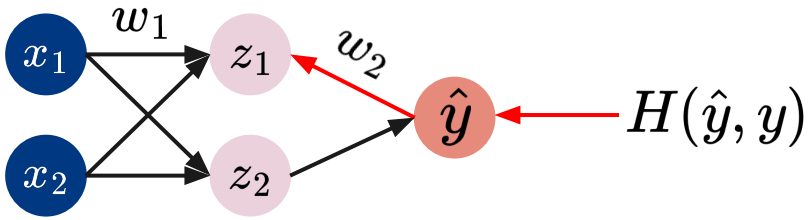
Backpropagation



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

- 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**.

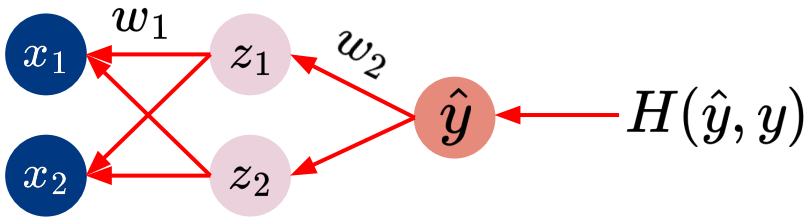
Backpropagation



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

- 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



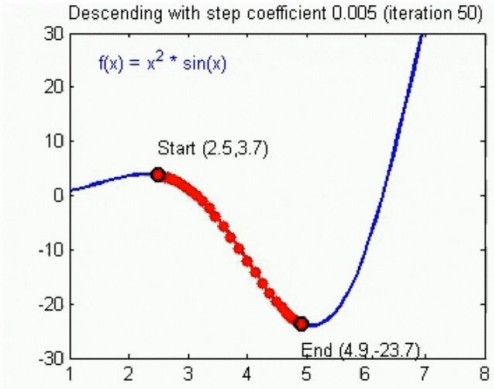
$$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial H(\hat{y}, y)}{\partial W^{(k)}}$$

Learning rate

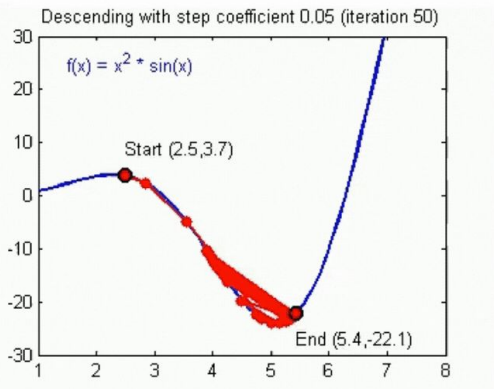
- We can use the **chain rule** to compute the **gradients** with respect to the weights.
- First of all, we compute the **derivative of the loss with respect to the output**.
- Then, we multiply the **derivative of the output with respect to the selected weight**.
- Once the derivative with respect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

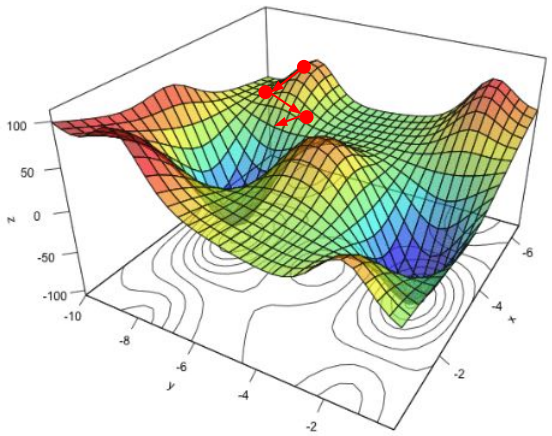


Divergence



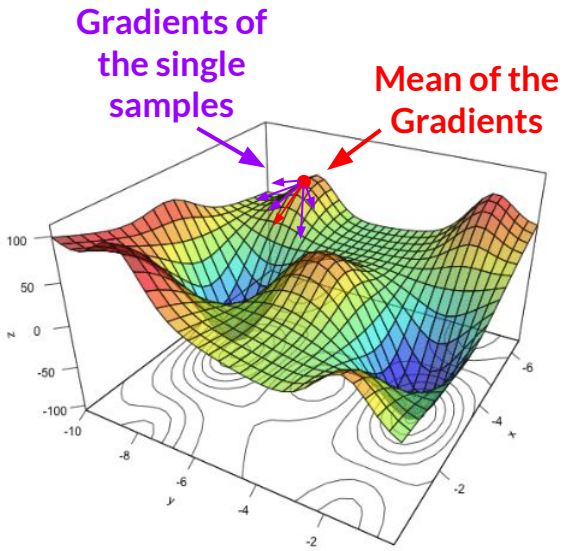
$$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial H(\hat{y}, y)}{\partial W^{(k)}}$$

Stochastic Gradient Descent



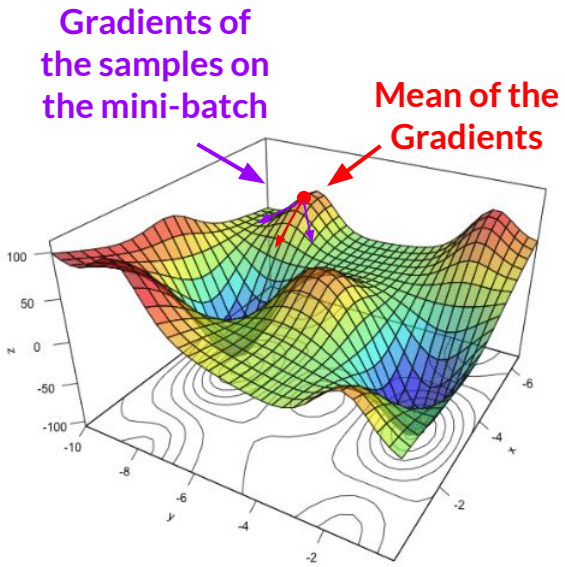
- Remember, our goal is to obtain **generalization capabilities** and not minimize the loss for a **single sample**
- A possible solution to this problem may be **randomly selecting samples from the training set** (**Stochastic Gradient Descent**)
- It is very **efficient** but leads to a **noisy descent**

Batch Gradient Descent



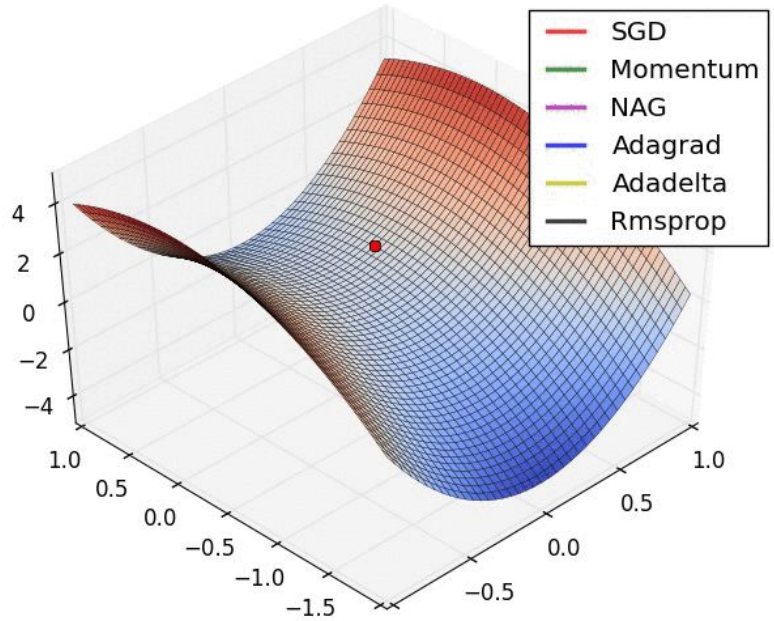
- A possible evolution of the SGD consists of using the **mean of the gradients** computed on **the whole training set** to make a single step (**Batch Gradient Descent**).
- It is a very **effective** strategy but also very **expensive**.

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).

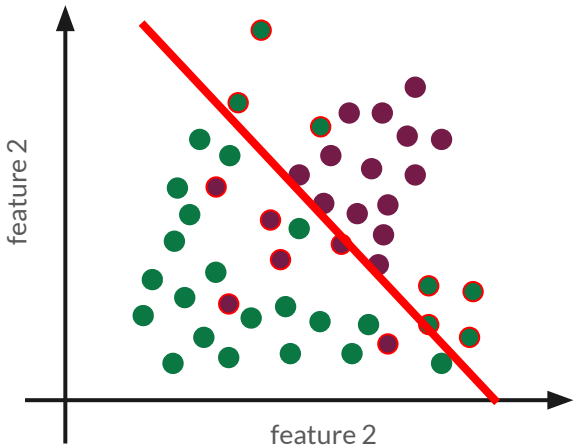
Can we do better?



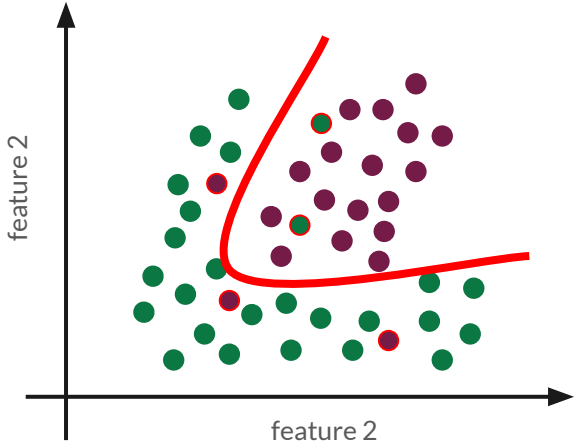
- Together with SGD, many optimizers exist that try to avoid the gradient getting stuck in local minima:
  - SGD + Momentum
  - RMSProp
  - Adam
- Standard deep-learning libraries implement many of these optimizers
- Empirically, one should **test many diverse optimizers to choose the best solution for the problem we are solving**.

Overfitting

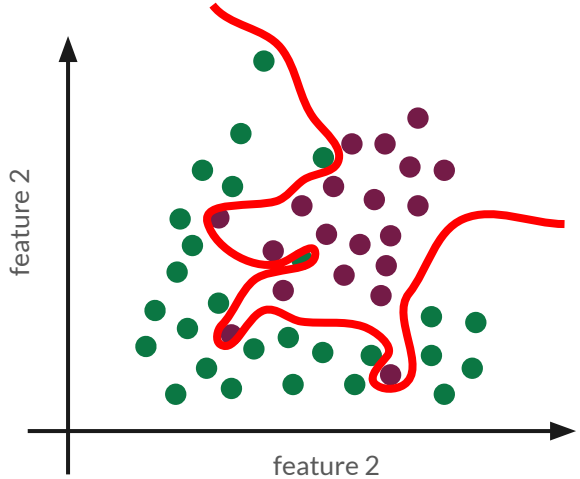
Neural Networks Suffers from high risk of Overfitting!



Underfitting



Good Fit



Overfitting

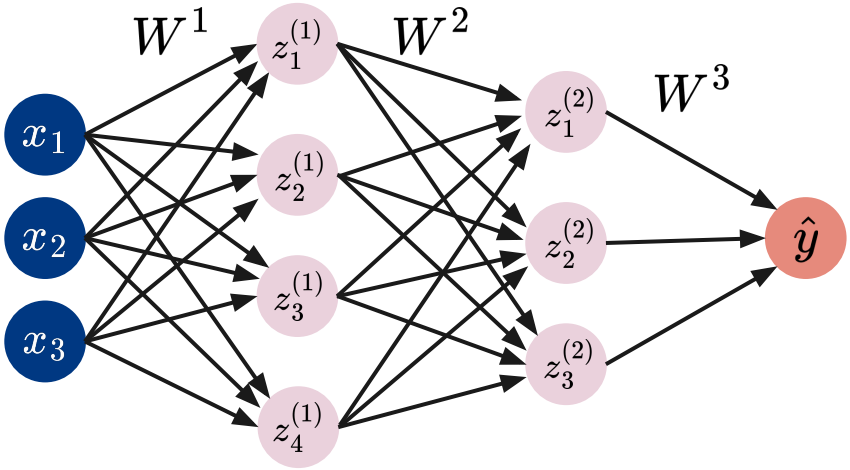
Regularization

Regularization term

$$\mathcal{L}(\hat{y}, y) = H(\hat{y}, y) + \overbrace{\|W\|}^{}_p$$

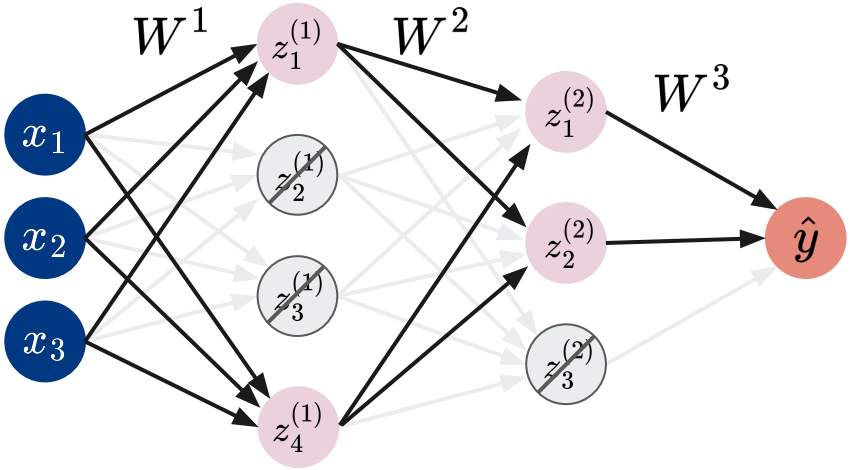
- An **L1** regularization (norm 1) **enforce sparsity** but **penalizes the magnitude** of the weights
- An **L2** regularization (norm 2) **penalizes sparsity** but **enforce the magnitude** of the weights
- In general, the desired effect is to diminish the **models' complexity**

Dropout Regularization



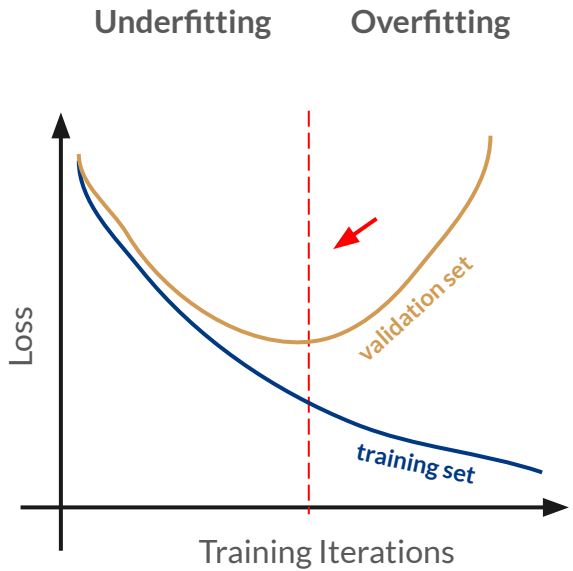
- In standard training we use all the weights during training…

Dropout Regularization



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

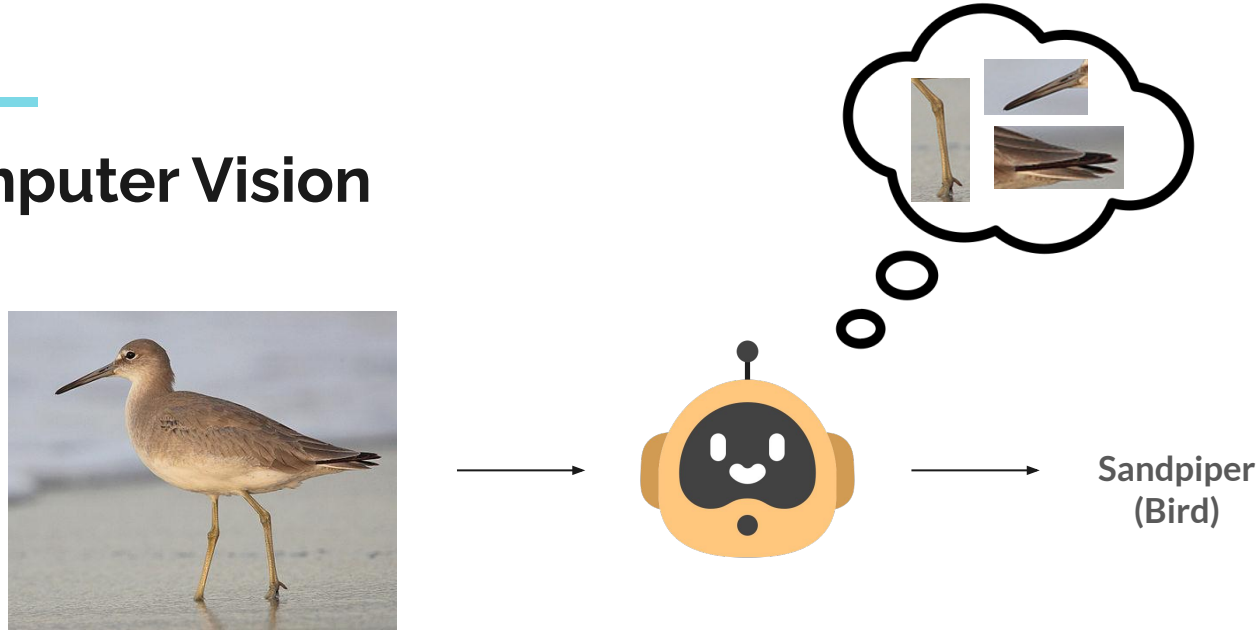
Early Stopping



- 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

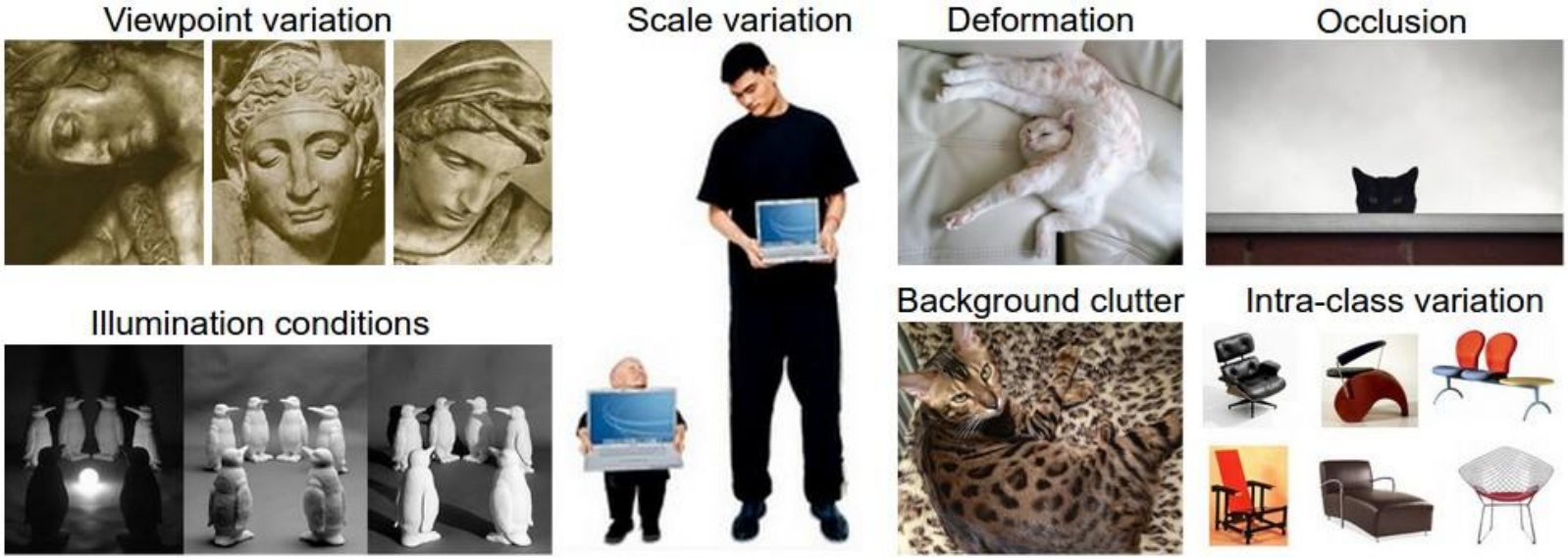
Computer Vision



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

Feature Extraction

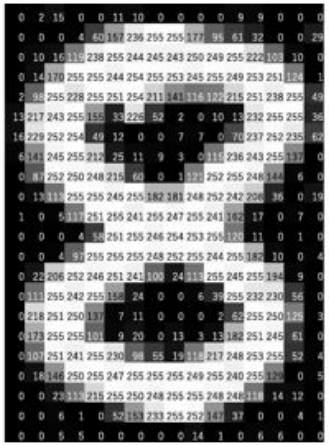
Feature Extraction



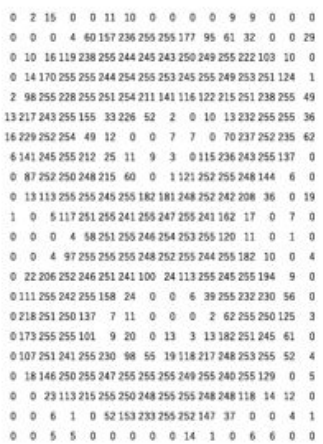
Images from the Computer Perspective

How can an algorithm recognize images?

Images from the Computer Perspective

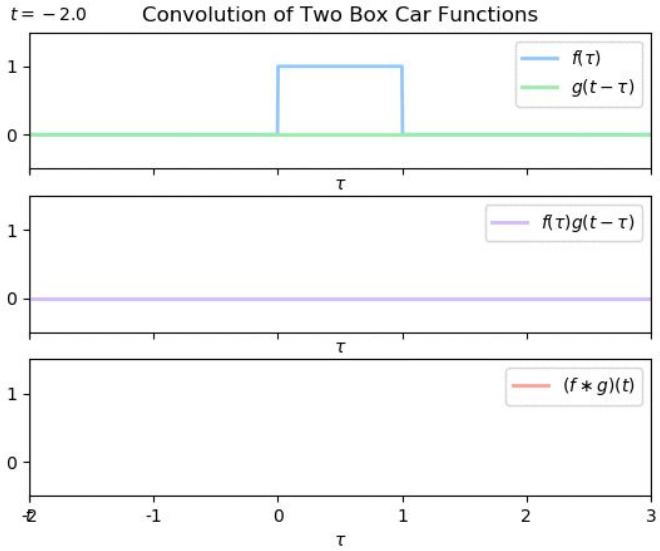


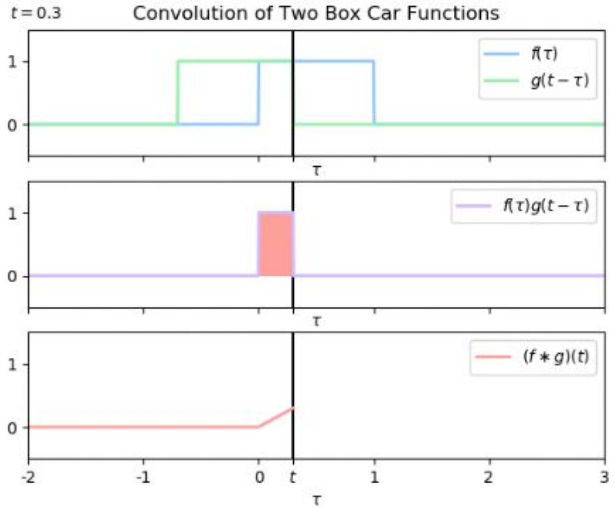
What Computer Sees



How can an algorithm recognize images?

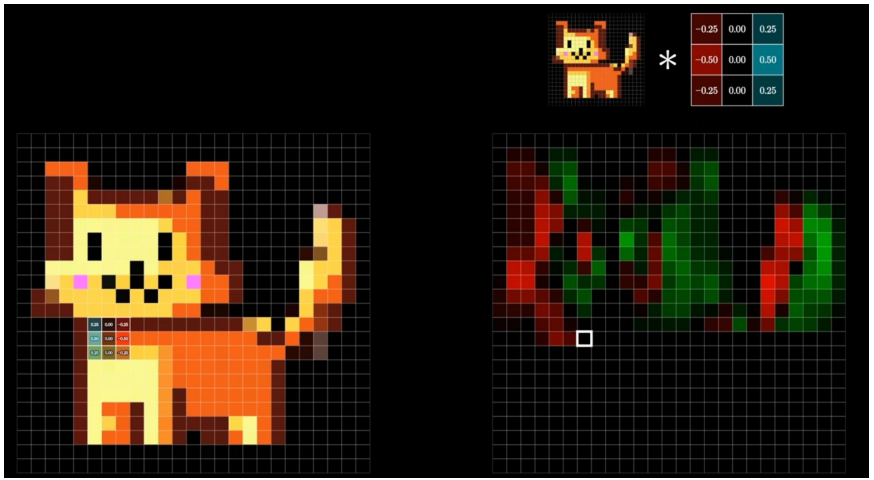
Convolutions



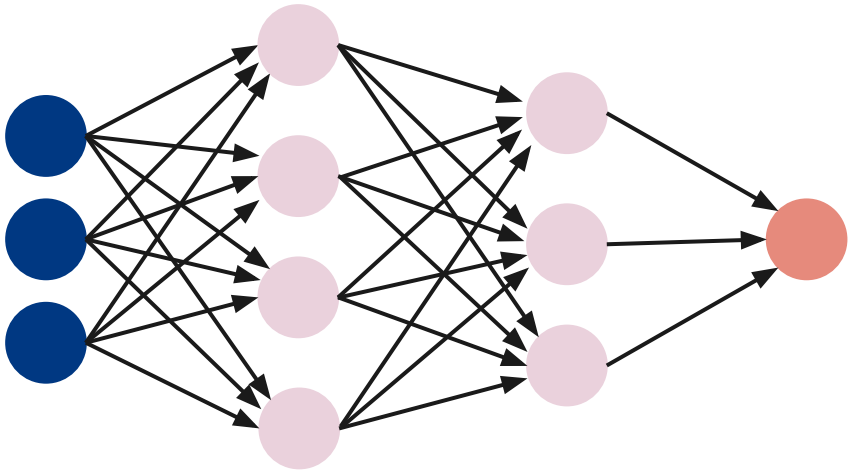


Convolutions

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



Fully Connected Layer

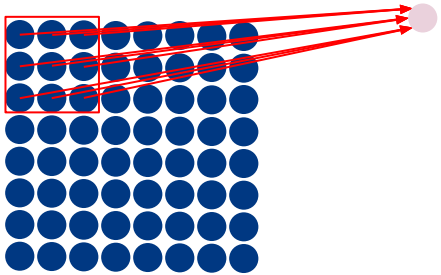


- In **fully connected** (FC) layers, each neuron of a given layer is connected to each neuron the next layer
- This would lead to a **high number of connections**
- And lacks from **spatial awareness**

Convolutional Layer

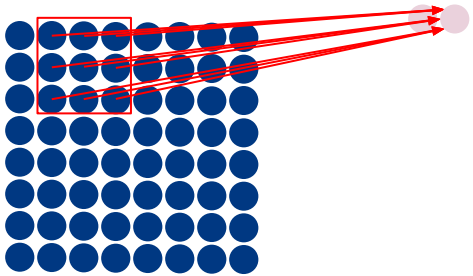
Convolutional Layer

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



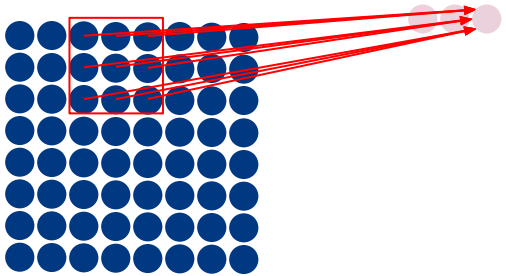
Convolutional Layer

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



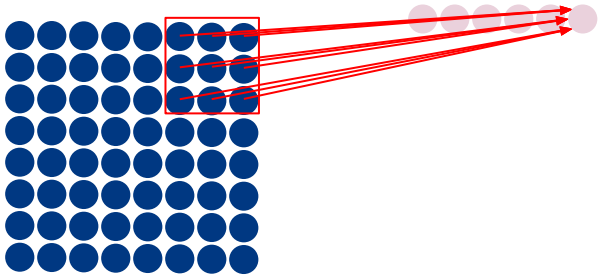
Convolutional Layer

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



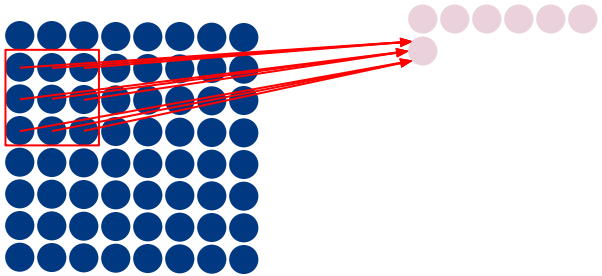
Convolutional Layer

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



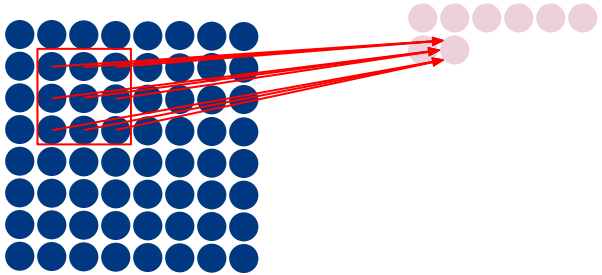
Convolutional Layer

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



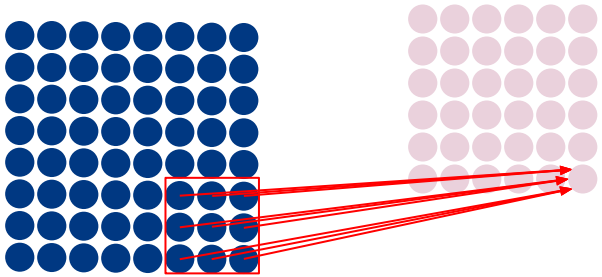
Convolutional Layer

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



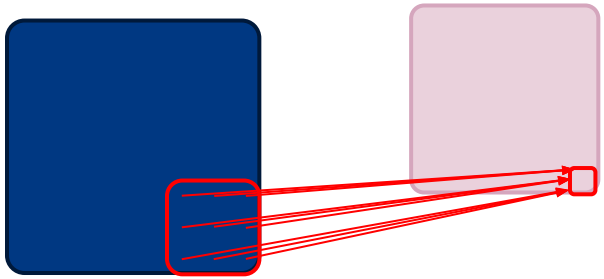
Convolutional Layer

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



Convolutional Layer

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

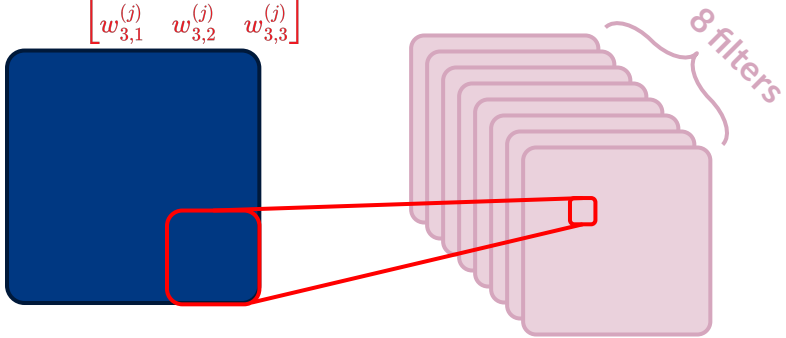


3x3 filter, replicated for each pixel

CNN: **9+1** parameters!
FC: ~**2300** parameters!

Convolutional Layer

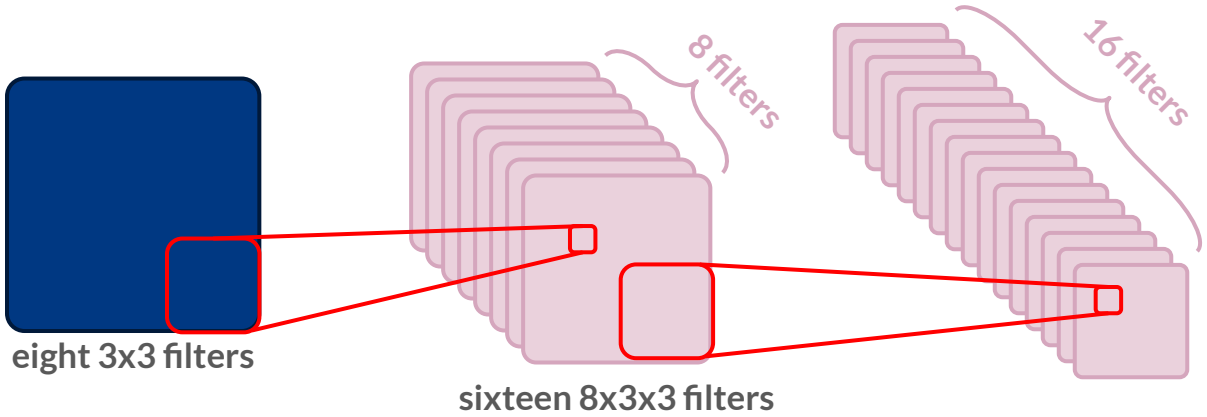
$$W_1^{(j)} = \begin{bmatrix} w_{1,1}^{(j)} & w_{1,2}^{(j)} & w_{1,3}^{(j)} \\ w_{2,1}^{(j)} & w_{2,2}^{(j)} & w_{2,3}^{(j)} \\ w_{3,1}^{(j)} & w_{3,2}^{(j)} & w_{3,3}^{(j)} \end{bmatrix}$$



eight 3x3 filters

CNN: **9+1** x 8 = **80** parameters
FC: ~**2300** x 8 ~=**18500** parameters!

Convolutional Layer



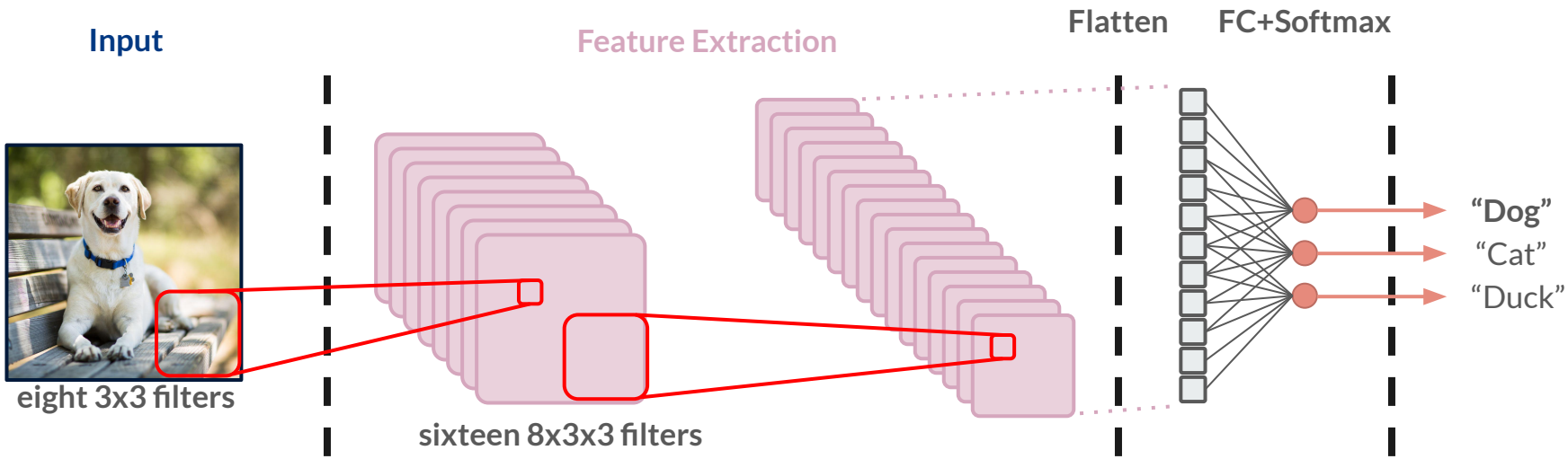
Conv Layer 1

CNN: **9+1** x 8 = **80** parameters
FC: ~**2300** x 8 ~=**18500** parameters!

Conv Layer 1

CNN: **8x3x3+1** x 16 = **1168** parameters
FC: ~**4600** x 16 ~=**73700** parameters!

Convolutional Neural Network



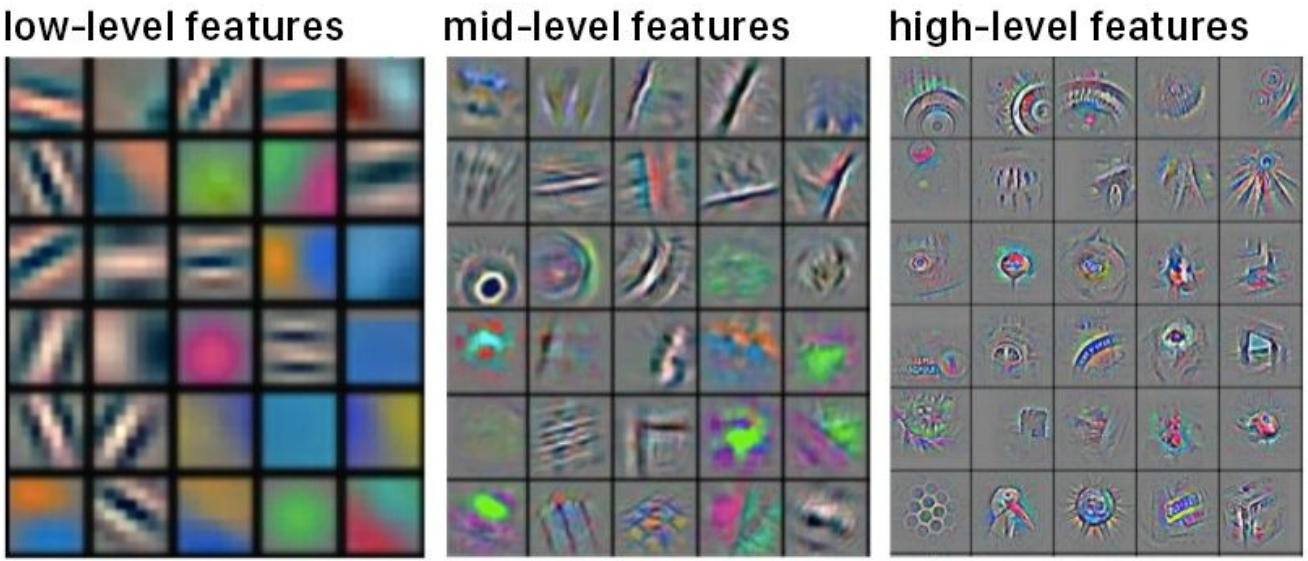
Conv Layer 1

CNN: **9+1** x 8 = **80** parameters
FC: ~**2300** x 8 ~=**18500** parameters!

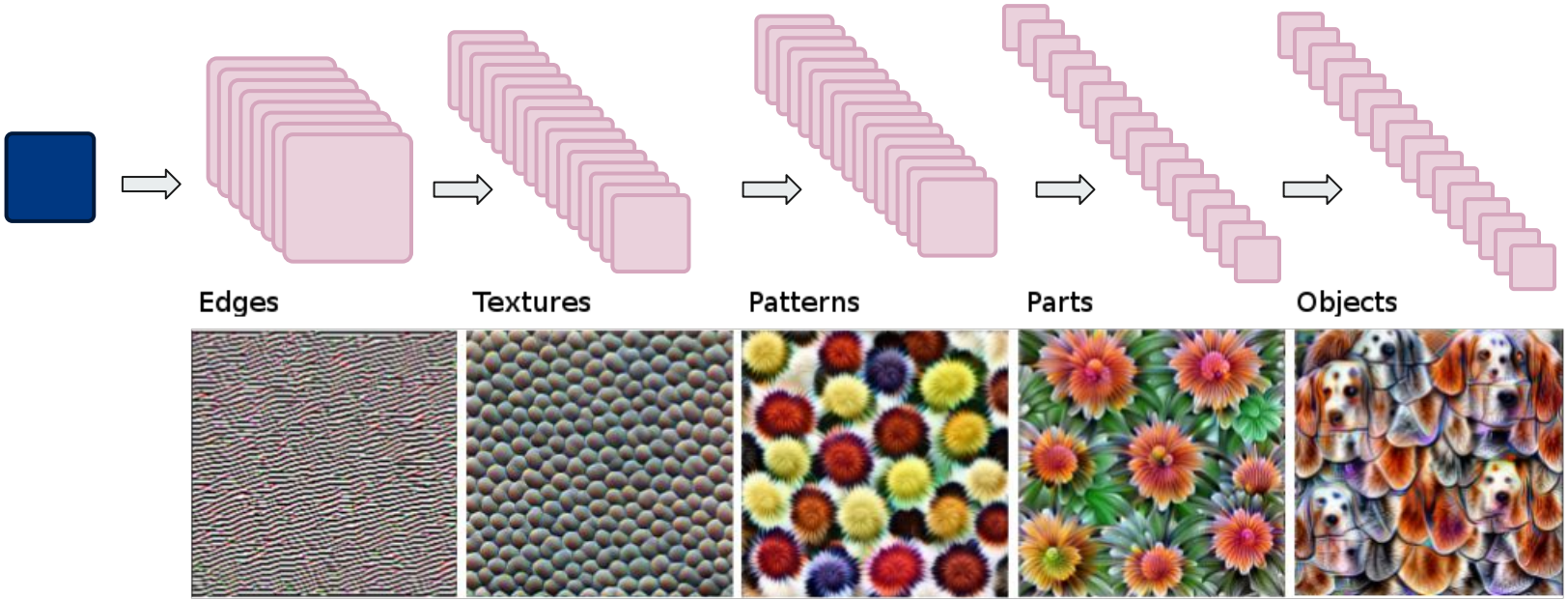
Conv Layer 1

CNN: **8x3x3+1** x 16 = **80** parameters
FC: ~**4600** x 16 ~=**73700** parameters

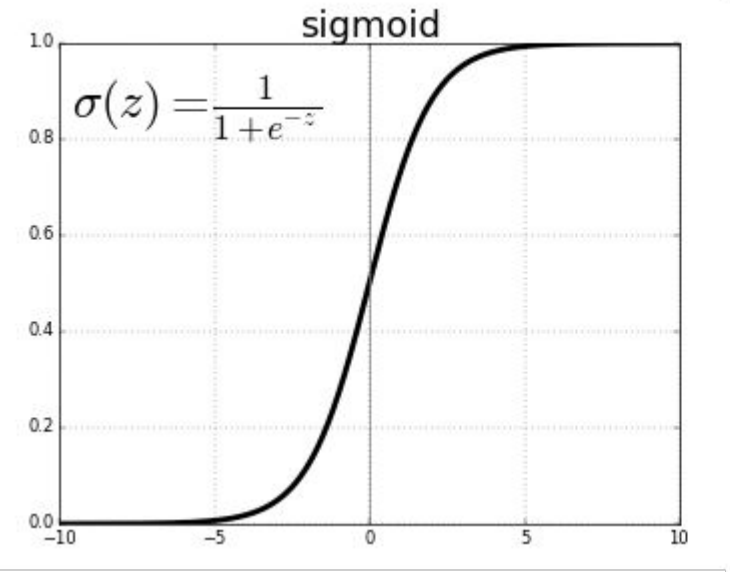
Feature Extraction using CNNs

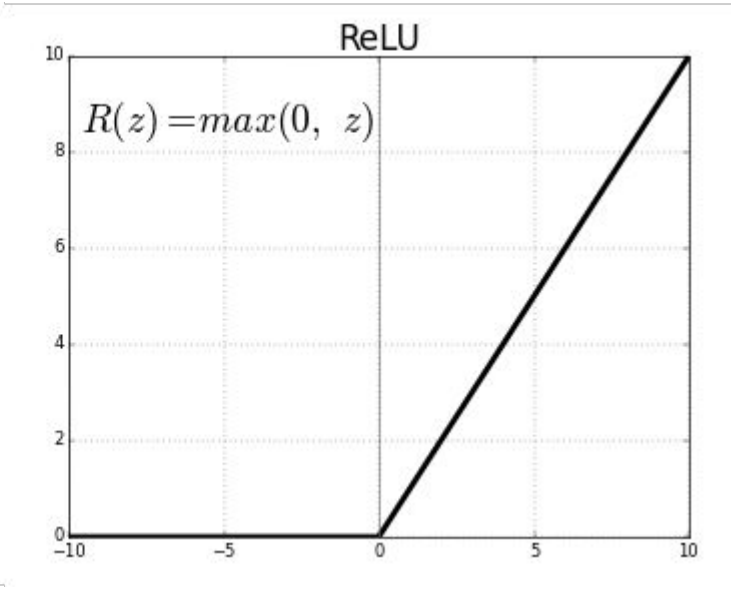


Feature Extraction using CNNs

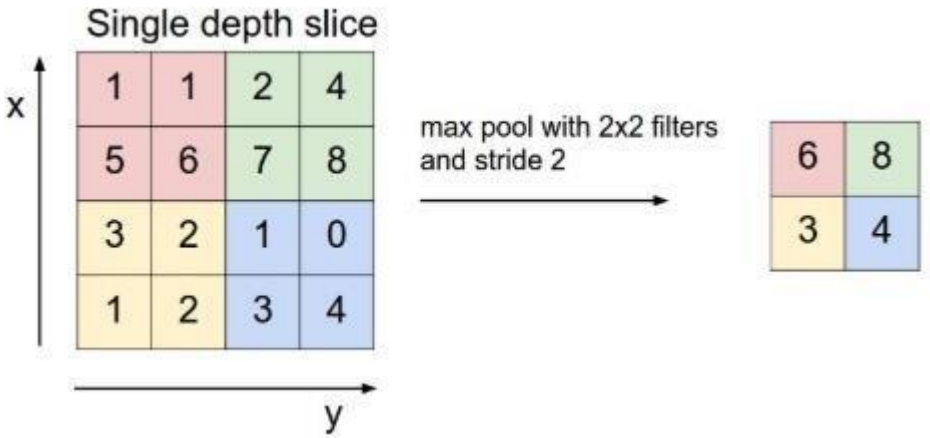


ReLU vs Sigmoid (vanishing gradients)



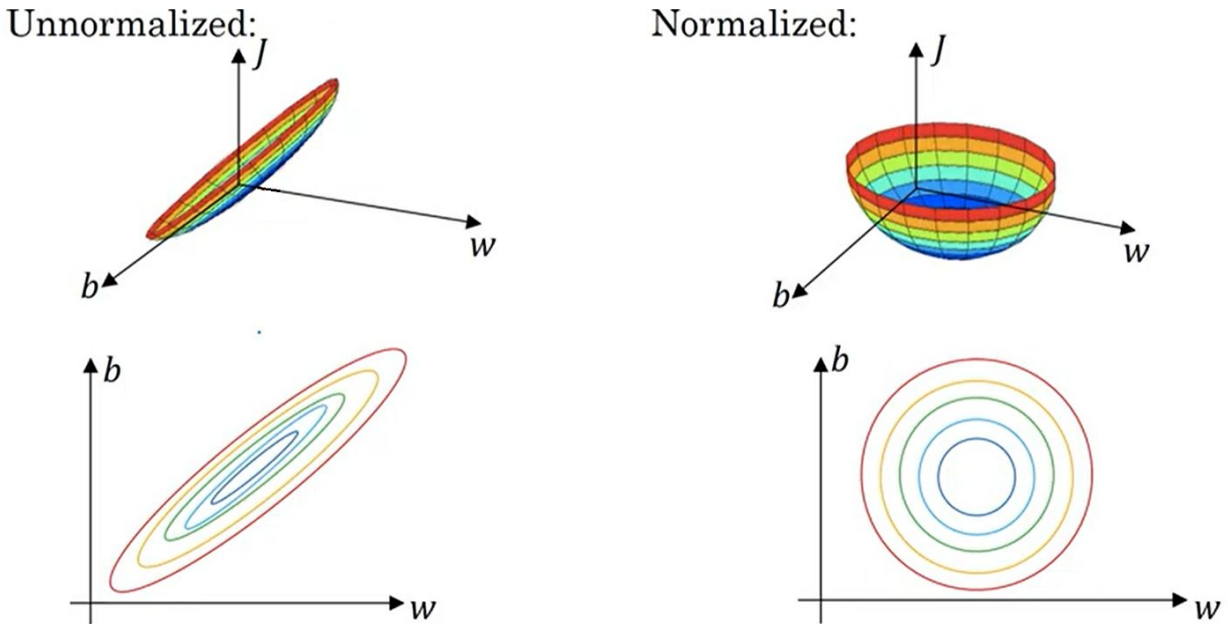


Pooling



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

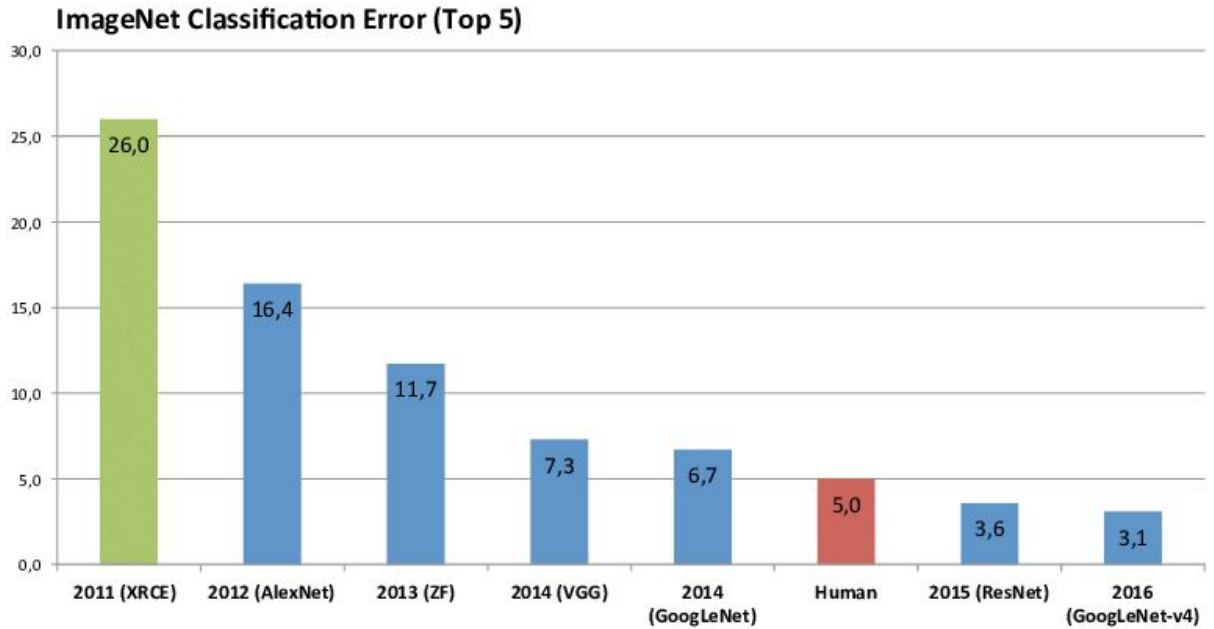
Batch Normalization



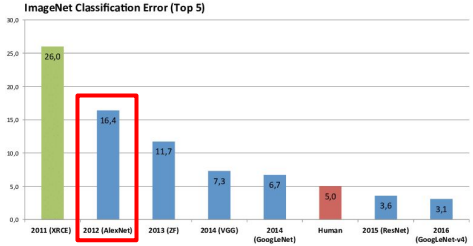
State-of-the-art CNNs

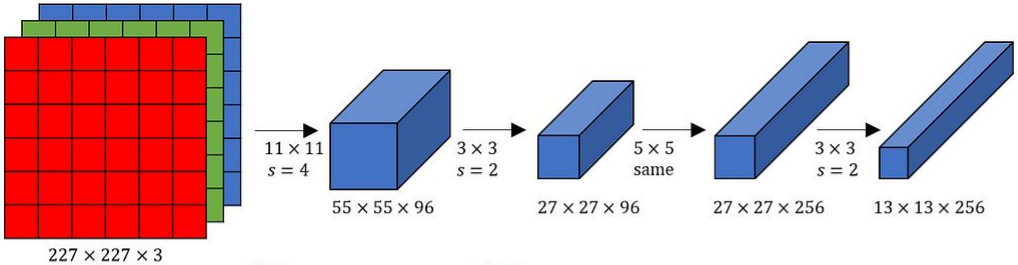
Classification

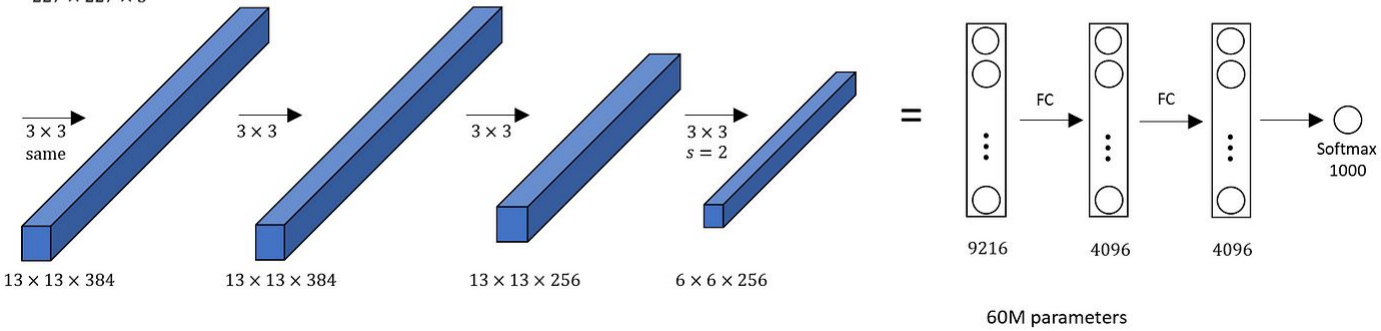
Classification



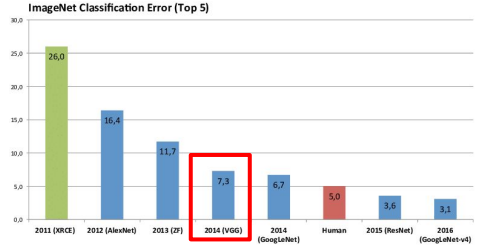
AlexNet

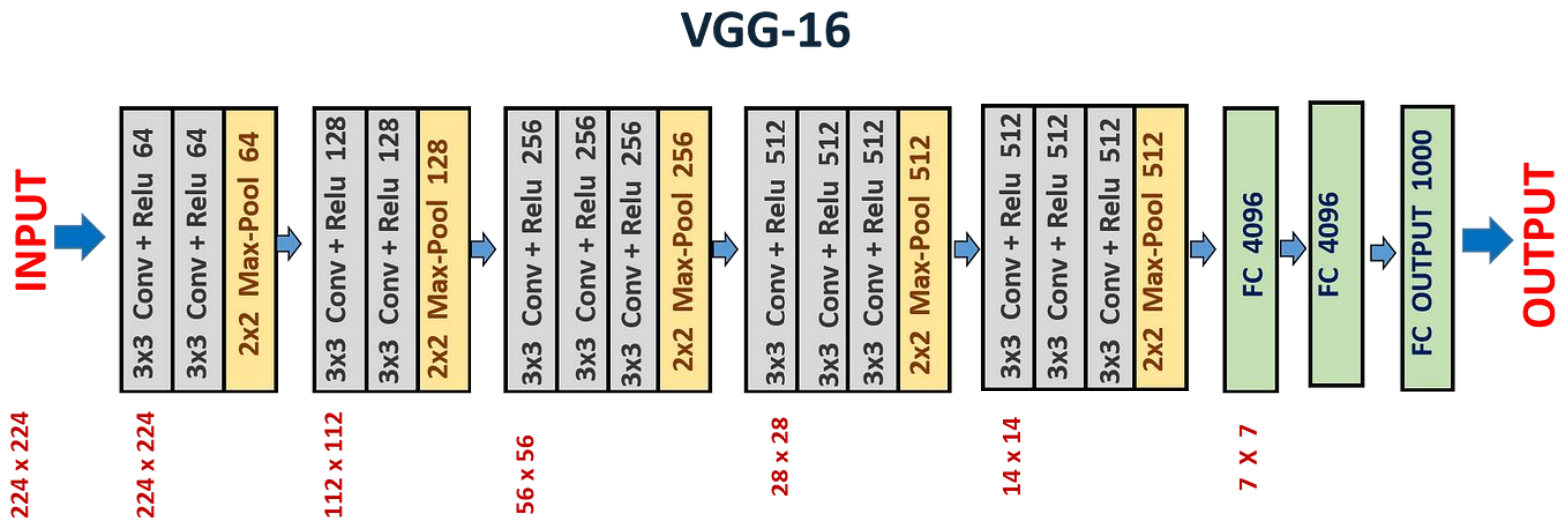

ImageNet Classification Error (Top 5)
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)


11 × 11
s = 4
3 × 3
s = 2
5 × 5
same
3 × 3
s = 2
227 × 227 × 3
55 × 55 × 96
27 × 27 × 96
27 × 27 × 256
13 × 13 × 256

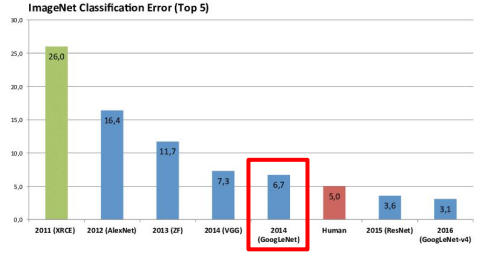

3 × 3
same
3 × 3
3 × 3
3 × 3
s = 2
=
FC
FC
Softmax
1000
13 × 13 × 384
13 × 13 × 384
13 × 13 × 256
6 × 6 × 256
9216
4096
4096
60M parameters

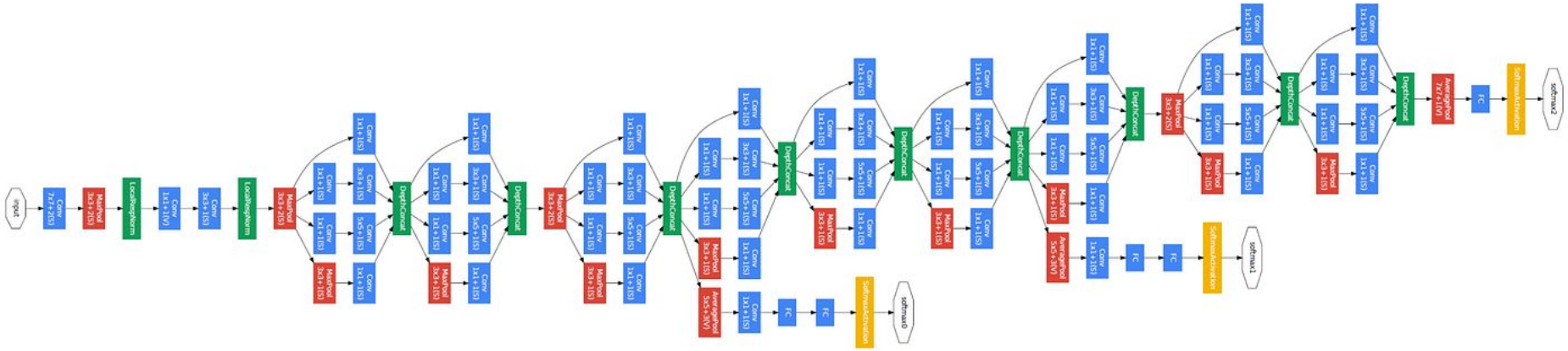
VGG


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)

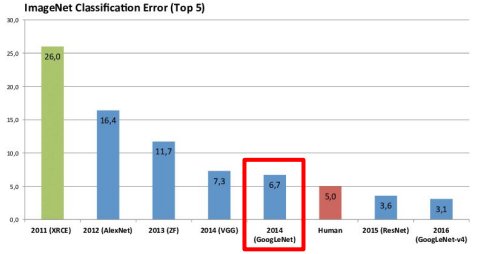

VGG-16
INPUT
224 x 224
224 x 224
3x3 Conv + Relu 64
3x3 Conv + Relu 64
2x2 Max-Pool 64
112 x 112
3x3 Conv + Relu 128
3x3 Conv + Relu 128
2x2 Max-Pool 128
56 x 56
3x3 Conv + Relu 256
3x3 Conv + Relu 256
3x3 Conv + Relu 256
2x2 Max-Pool 256
28 x 28
3x3 Conv + Relu 512
3x3 Conv + Relu 512
3x3 Conv + Relu 512
2x2 Max-Pool 512
14 x 14
3x3 Conv + Relu 512
3x3 Conv + Relu 512
3x3 Conv + Relu 512
2x2 Max-Pool 512
7 X 7
FC 4096
FC 4096
FC OUTPUT 1000
OUTPUT

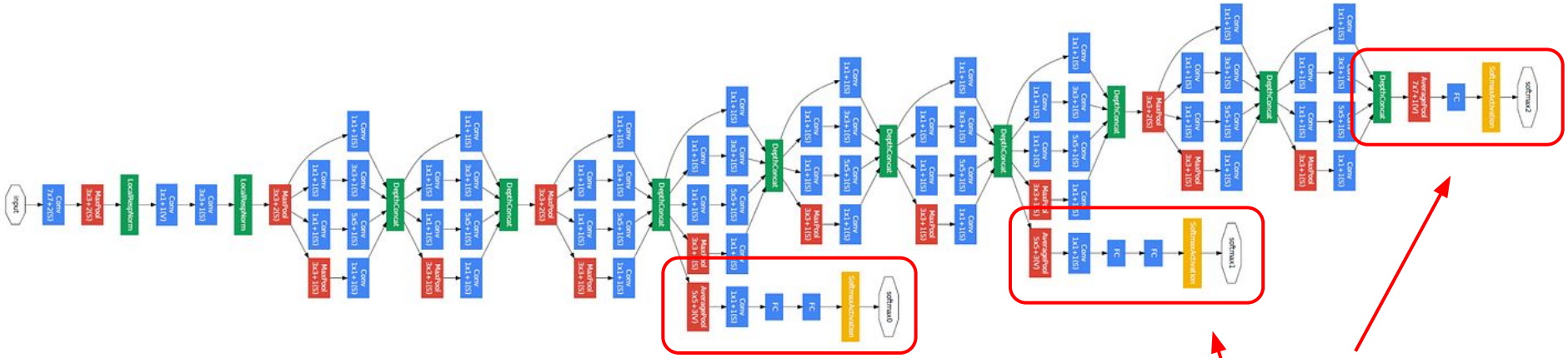
GoogLeNet


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)


input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
DepthConcat
MaxPool 3x3+2(S)
AveragePool 5x5+3(V)
FC
SoftmaxActivation
softmax0
softmax1
softmax2

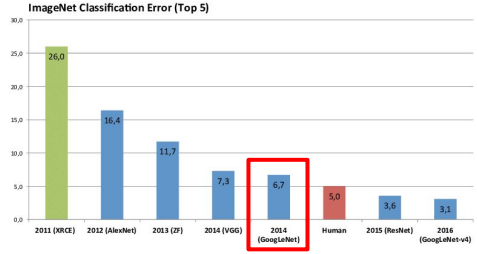
GoogLeNet

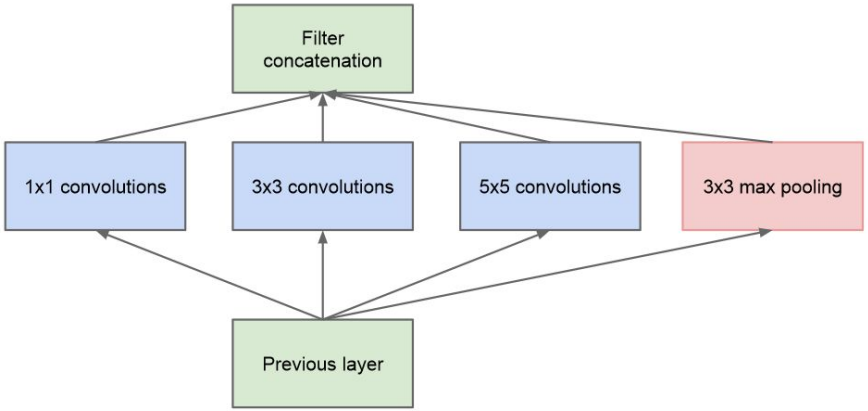


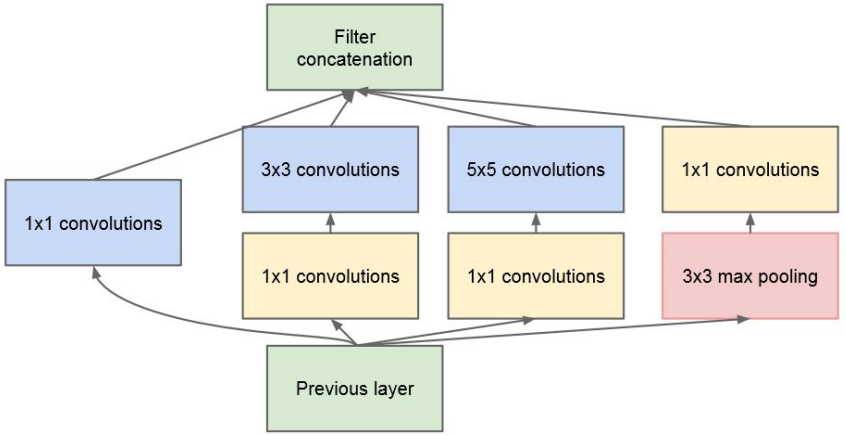


Three heads

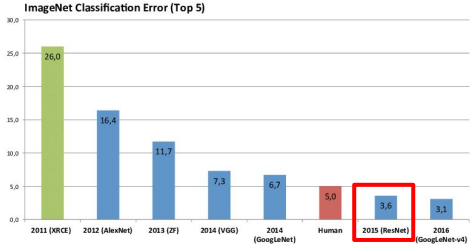
GoogLeNet - Blocks

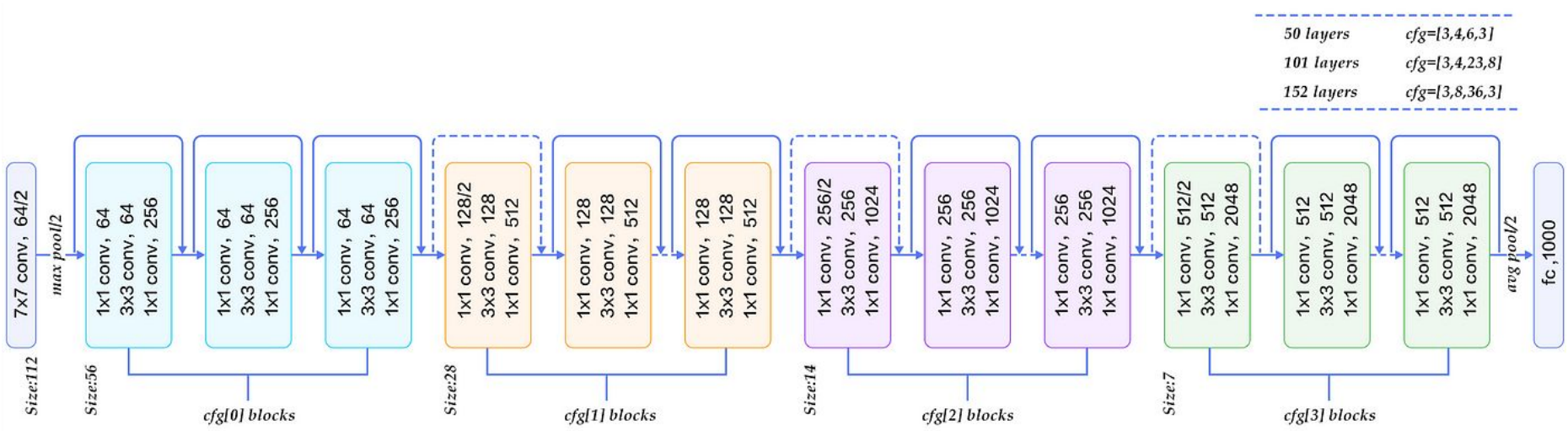




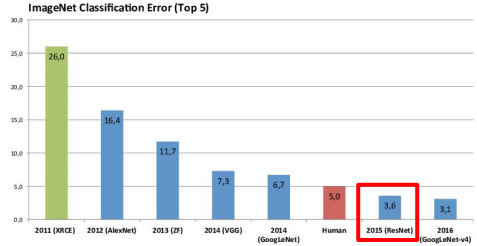


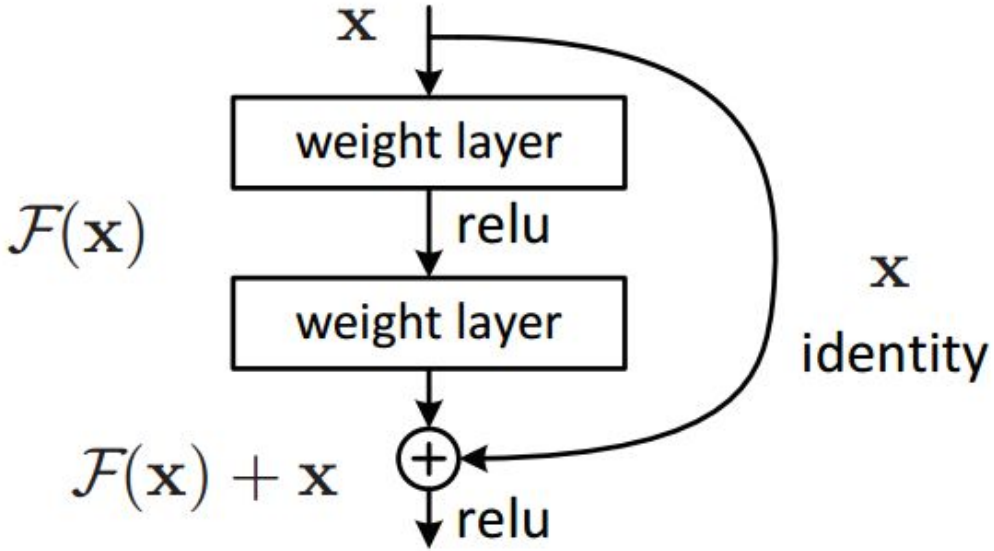
ResNet


ImageNet Classification Error (Top 5)
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)

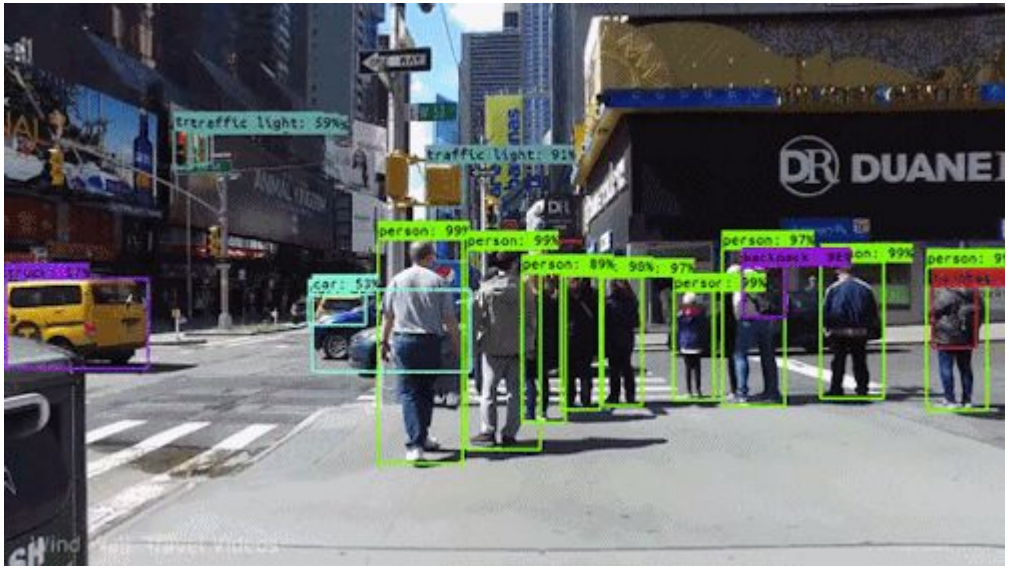

50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64
3x3 conv, 64
1x1 conv, 256
1x1 conv, 128/2
3x3 conv, 128
1x1 conv, 512
1x1 conv, 256/2
3x3 conv, 256
1x1 conv, 1024
1x1 conv, 512/2
3x3 conv, 512
1x1 conv, 2048
avg pool/2
fc ,1000
Size:112
Size:56
Size:28
Size:14
Size:7
cfg[0] blocks
cfg[1] blocks
cfg[2] blocks
cfg[3] blocks

ResNet - Residual Blocks

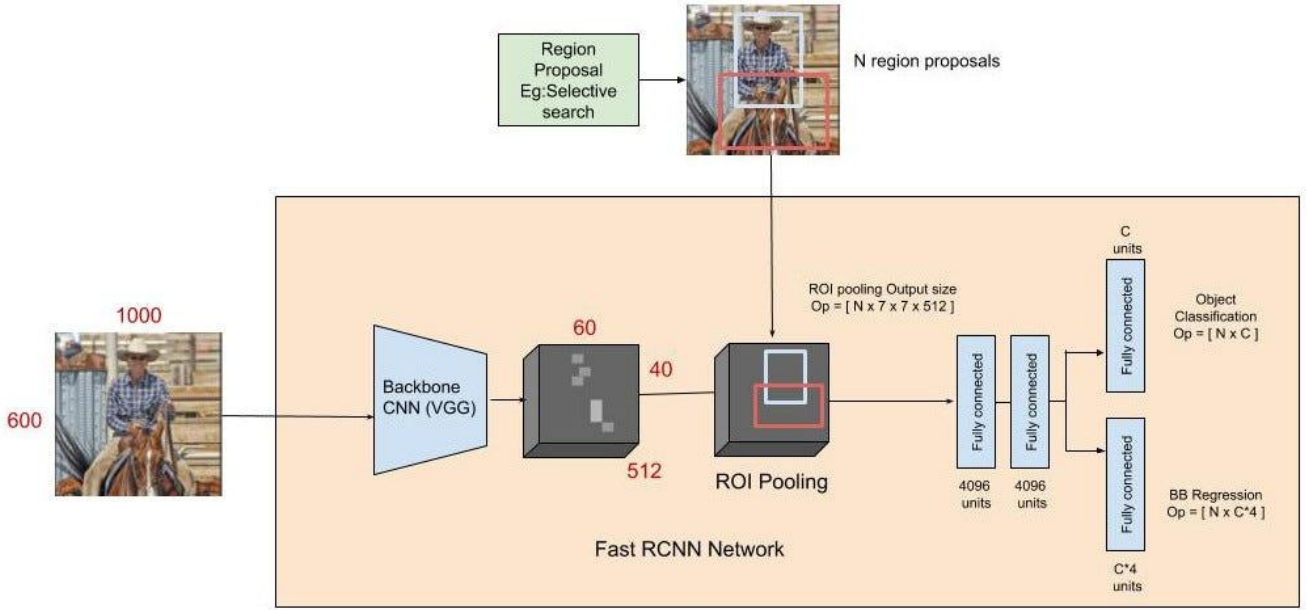




Object Detection

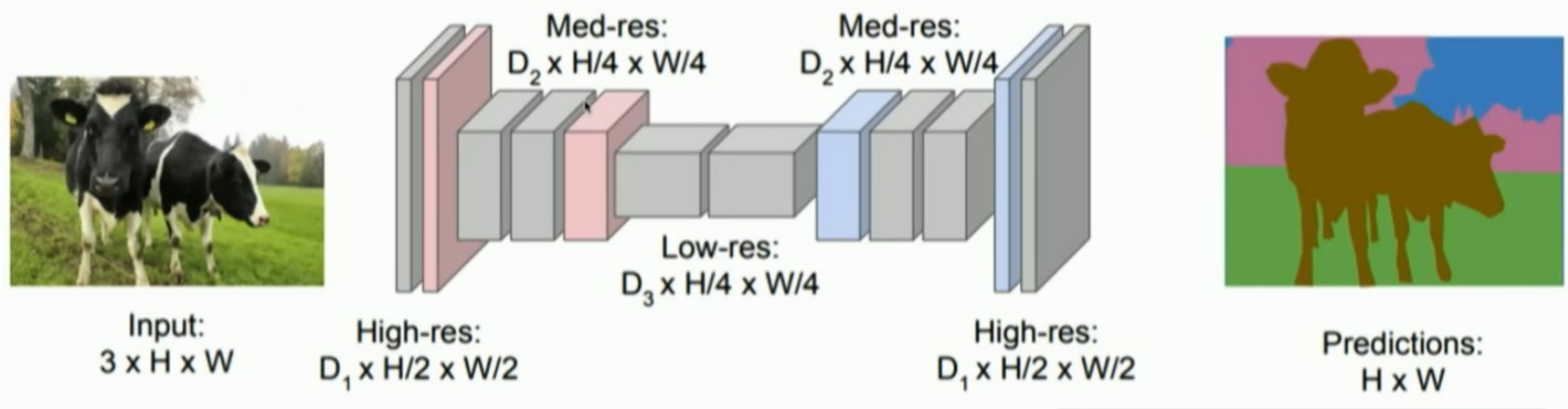


Object Detection (R-CNN)



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