Introduction to Deep Learning

Anand Mishra

भारतीय प्रौद्योगिकी संस्थान जोधपुर
Indian Institute of Technology Jodhpur

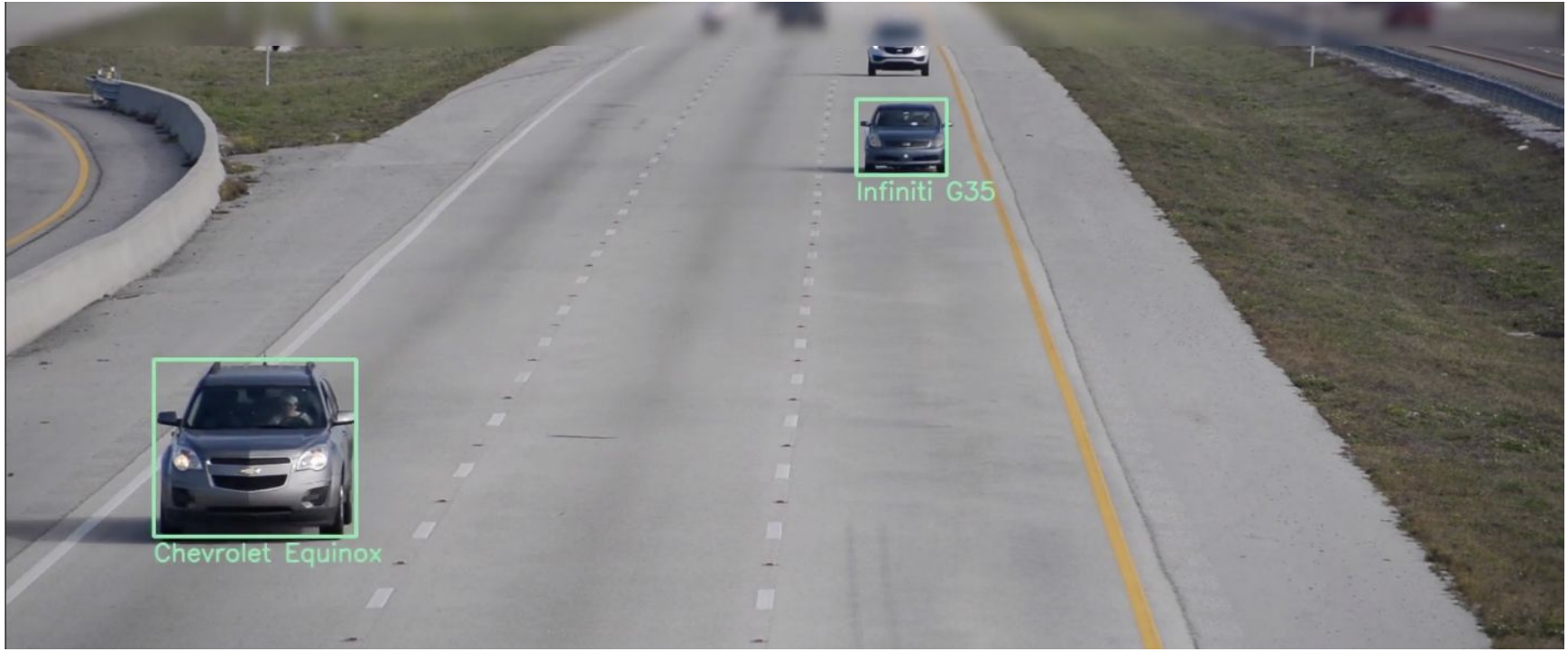
Major resources:

- 11-785 Introduction to Deep Learning (CMU): **Bhiksha Raj**
- CS7015: Deep Learning (IIT-M): **Mitesh Khapra**
- CS231n: CNNs for Visual Recognition: **Fei-Fei Lee et al.**
- http://neuralnetworksanddeeplearning.com
- https://www.deeplearningbook.org
- Duda, Hart, and Strok, **Pattern Classification**
- Neural Networks by Rojas
- …
- Few of my own examples …

Neural Networks are taking over!

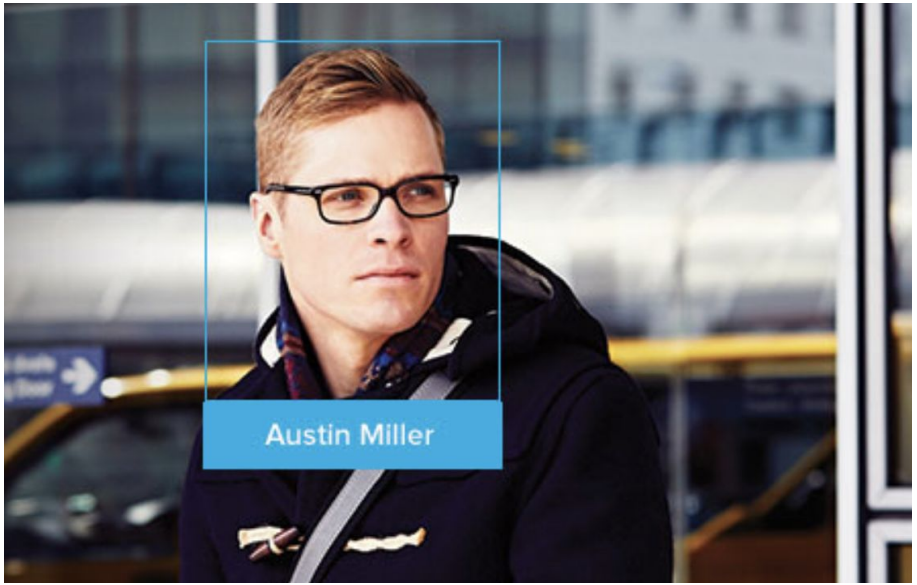
- Major thrust area
- New State of the art in many ML tasks

Visual Recognition



https://www.sighthound.com/technology/

Visual Recognition



https://www.sighthound.com/technology/

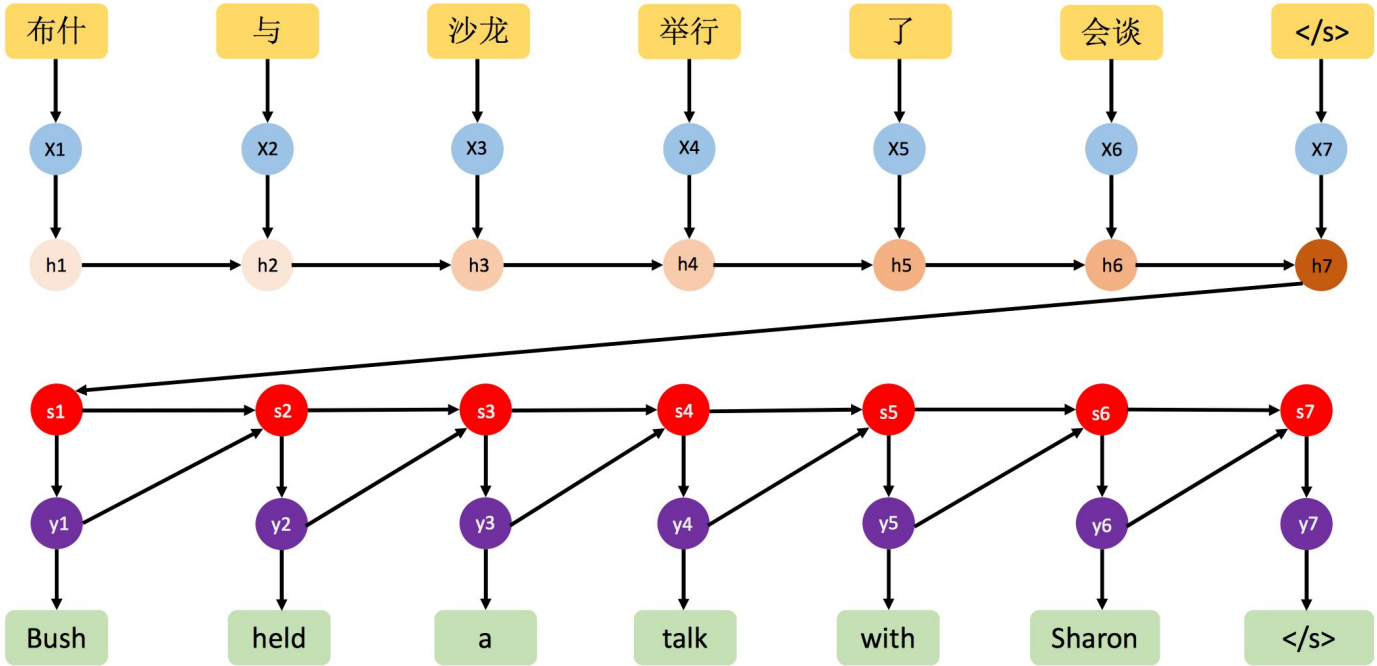
Describing Images

A child in a helmet is riding a bike.

A group of people are walking on a busy street.

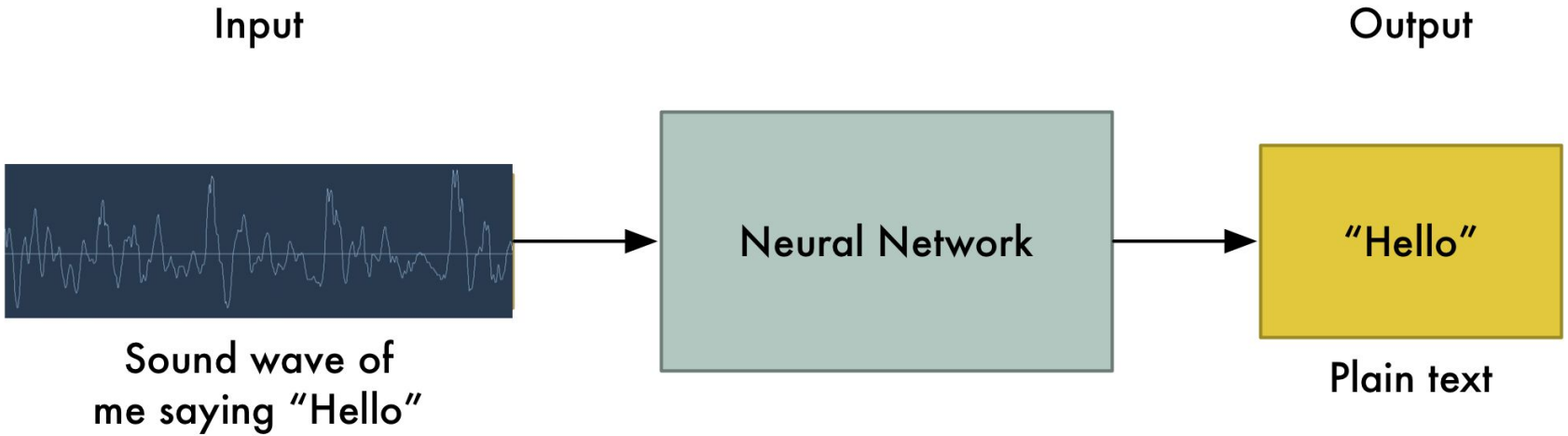
[Vinyals et al., CVPR 2015]

Translating Languages



[Sutskever et al.,NIPS 2014]

Speech Recognition



History of Neural Networks

History of Neural Networks

Breakthrows	Year	People associated
Use of term "Neuron"	1891	H. W. G. von Waldeyer-Hartz
MP Neuron (Computation model)	1943	McCullah and Pitts
Perceptron Model	1957-58	Frank Rosenbellt

History of Neural Networks

Breakthrows	Year	People associated
Multi Layer Perceptron (early deep learning model)	1965-68	Alexey Ivakhnenko
Limits of Perceptron	1969	Minsky and Papert
Gradient Descent	1986	Cauchy

History of Neural Networks

Breakthrows	Year	People associated
Universal Approximation Theorem	1989	George Cybenko
Unsupervised Pre training	2006	Hinton & Salakhutdinov
Handwriting, speech, GPUs	2009-2010	Graves, ...

History of Neural Networks

Breakthrows	Year	People associated
Visual Recognition (Alexnet): 16% error in imagenet	2012	Alex Krizhevsky, Ilya Sutskever and Hinton
Visual Recognition (VGG net)	2014	Simonyan and Zisserman
Visual Recognition (Resnet): 3.6% error in imagnet, better than human!	2016	Kaiming He et al.

History of Neural Networks

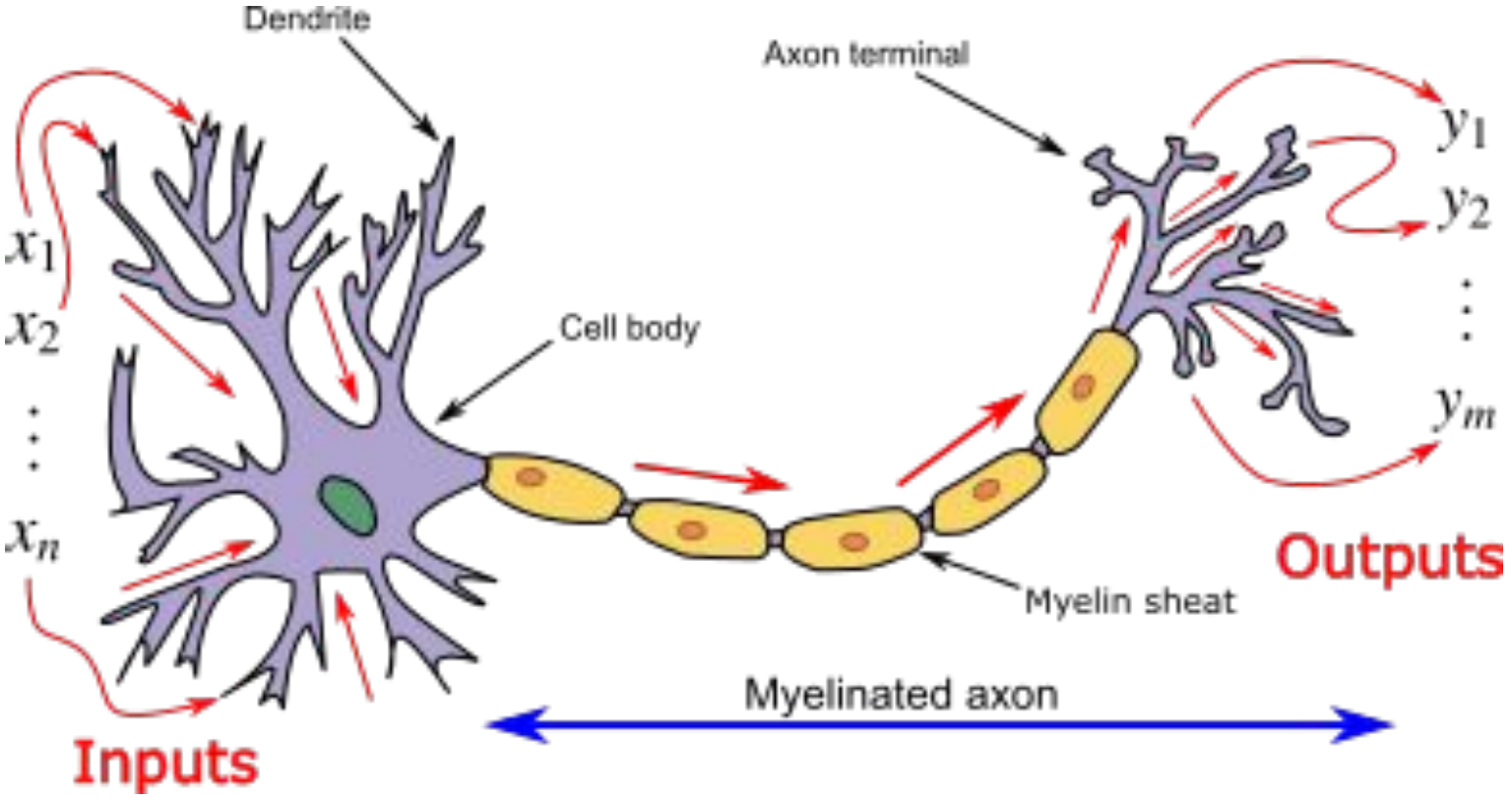
Breakthrows	Year	People associated
Generative Adversarial Networks (GANs)	2014	Ian Goodfellow et al.
Attention Mechanism	2016	Bahdanau et al.
Transformers	2017	Vaswani et al.
BERT	2018	Jacob Devlin et al.

History of Neural Networks

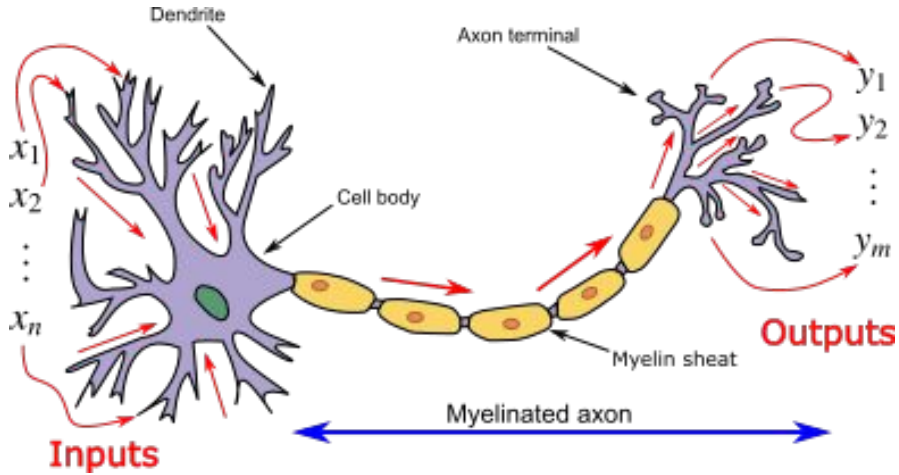
Breakthrows	Year	People associated
GPT/GPT-2/GPT-3	2018-2022	OpenAI
GPT-4 (ChatGPT)	2023	OpenAI
VIT/CLIP	2020-21	Dosovitskiy/Radford et al.
DALL-E	2021-22	OpenAI

Biological Neurons

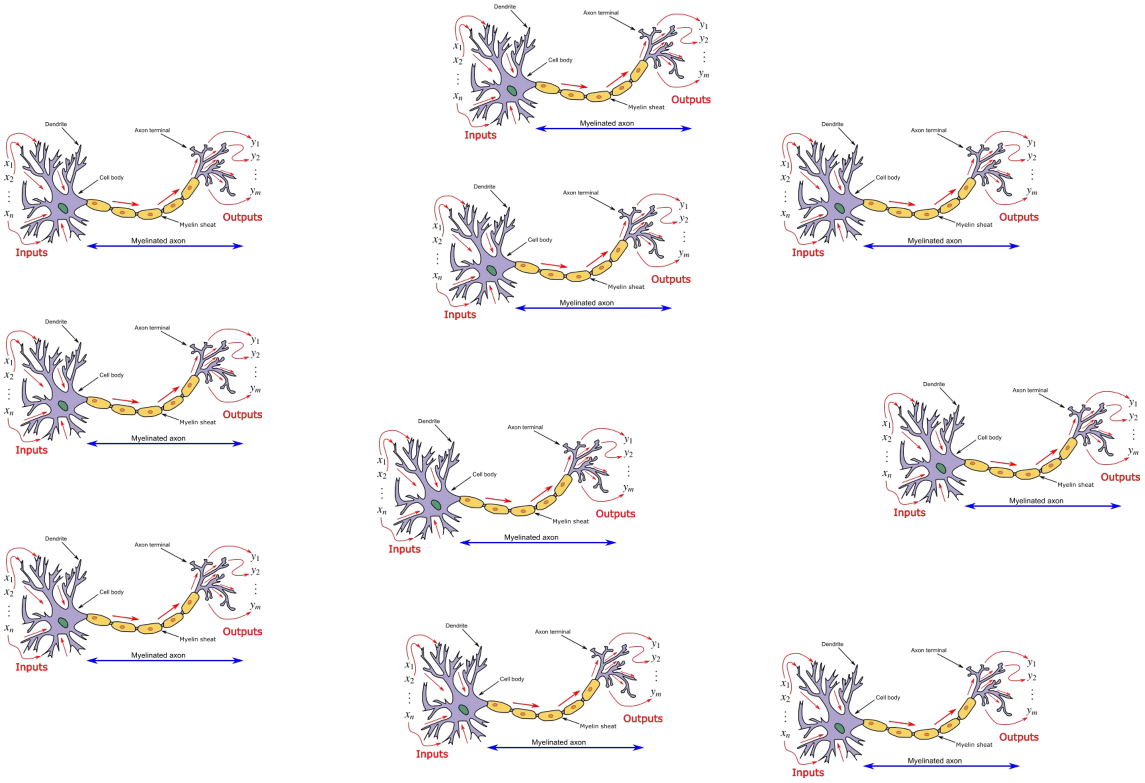
Biological Neurons



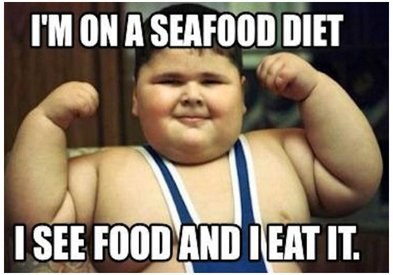
How biological neuron works?

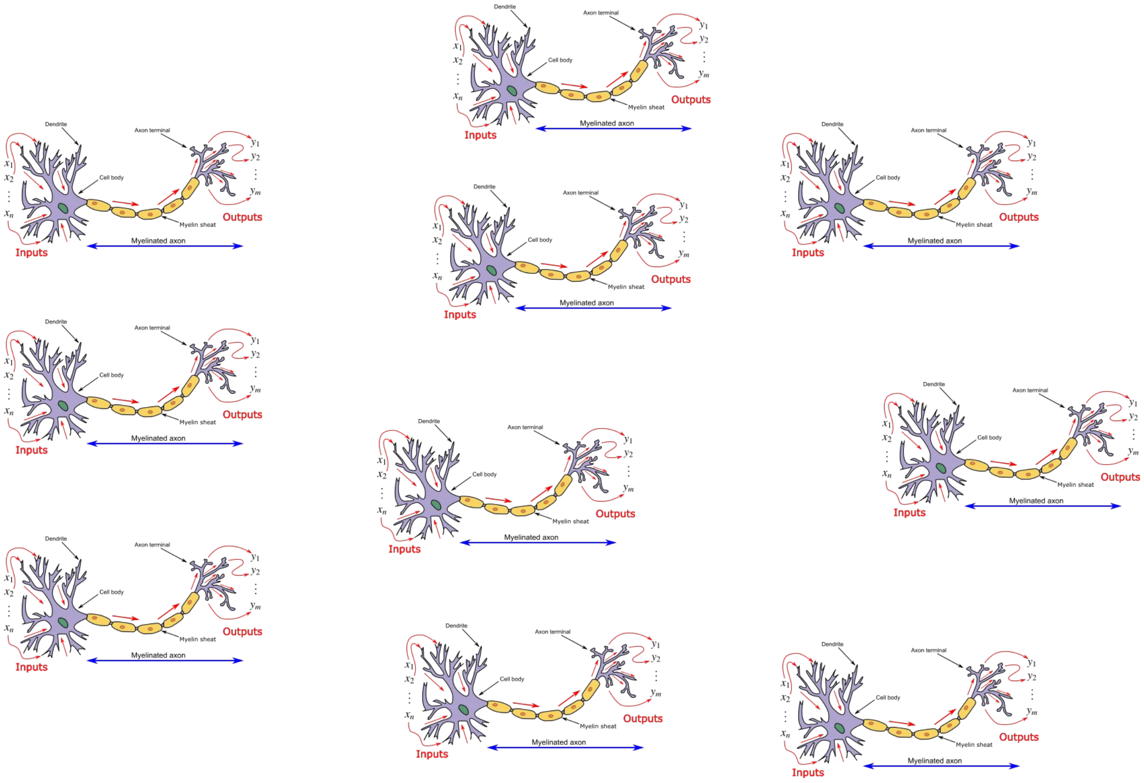


Massively Parallel Neurons

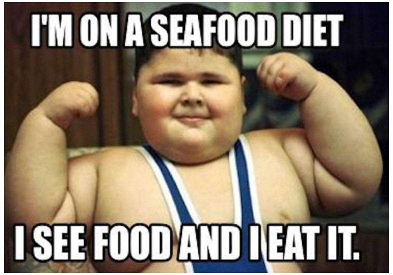


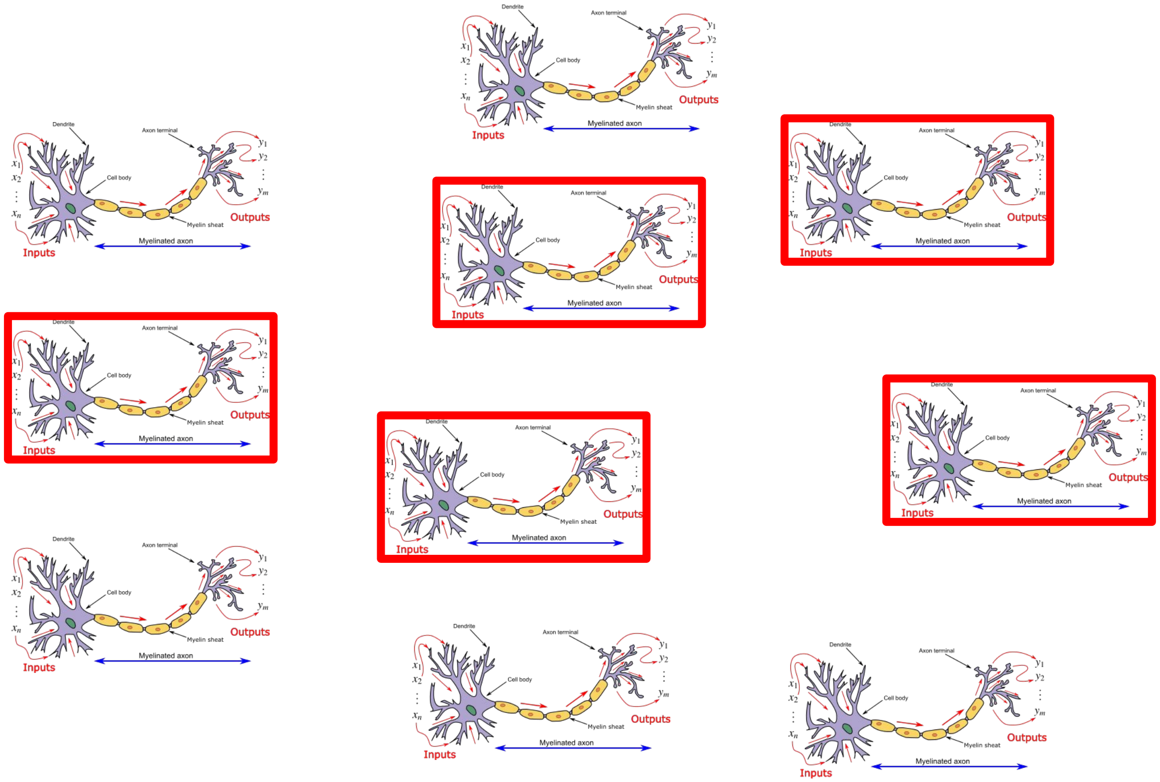
Massively Parallel Neurons



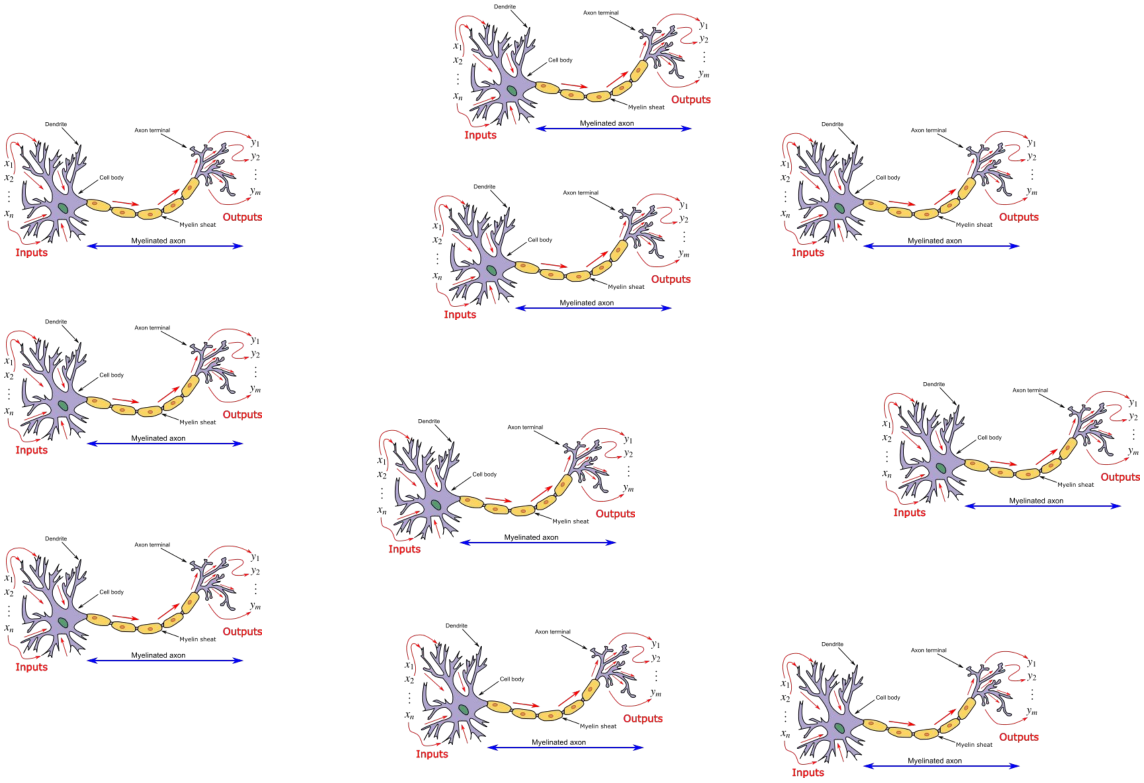


Massively Parallel Neurons

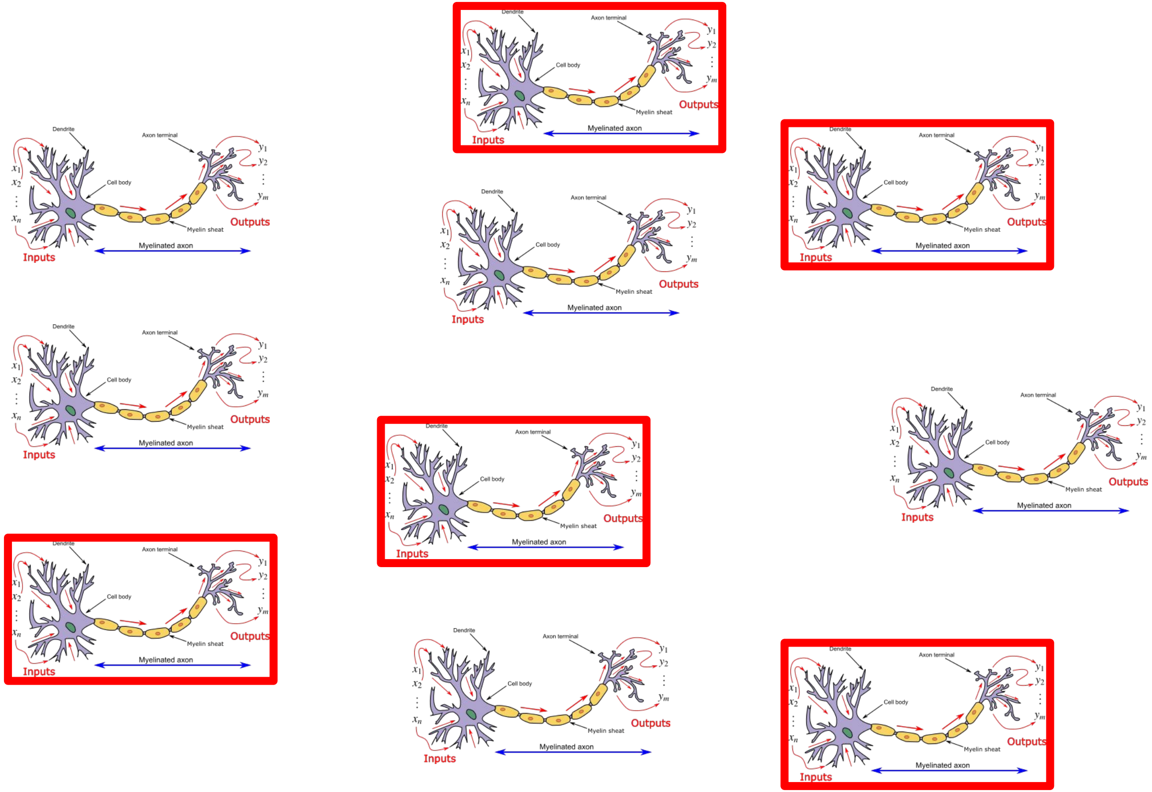




Massively Parallel Neurons

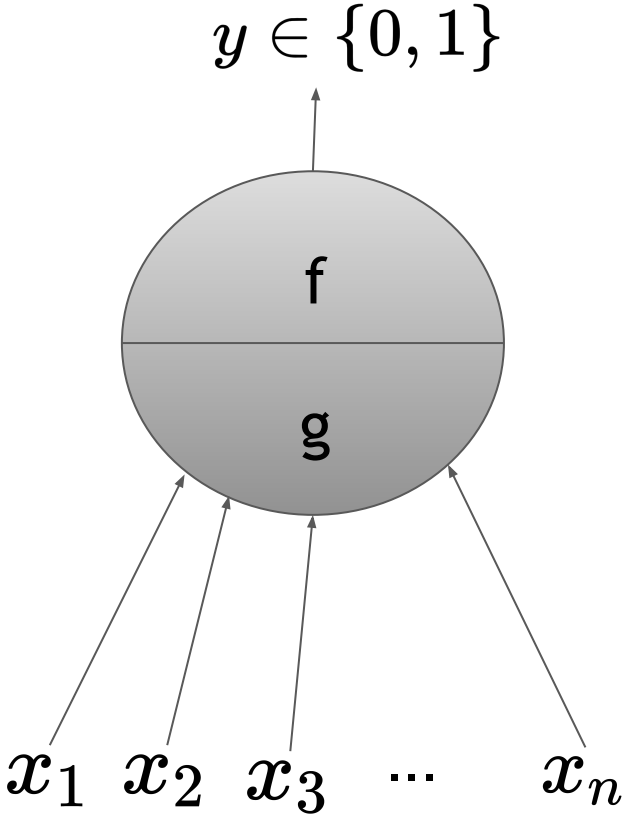


Massively Parallel Neurons



McCulloch-Pitts Neuron

Highly simplified Computation Model



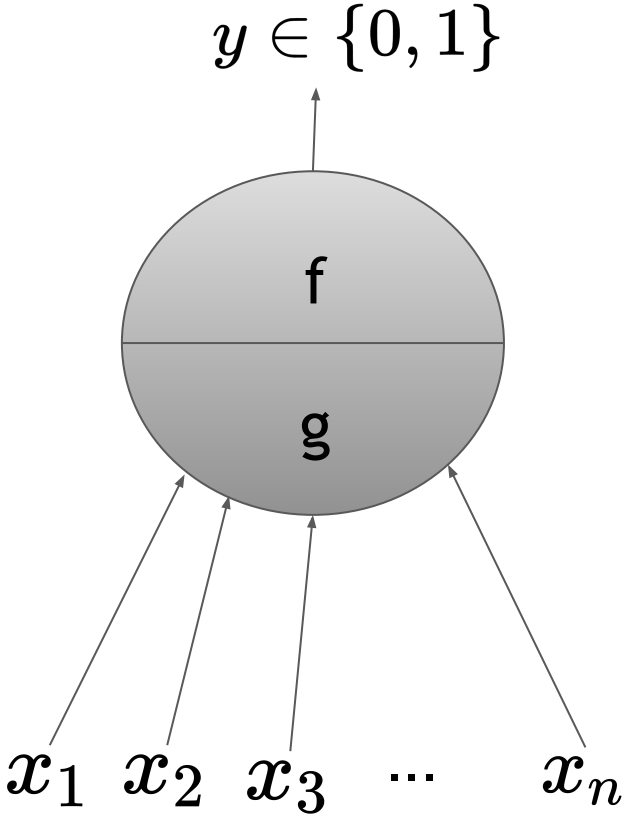
- Boolean input and Boolean output
- g aggregates input and f takes decision

$$g(\mathbf{x}) = \sum_i^n x_i$$

$$y = f(g(x)) = 1 \; if \; g(x) >= \theta$$

$$y = f(g(x)) = 0 \; if \; g(x) < \theta$$

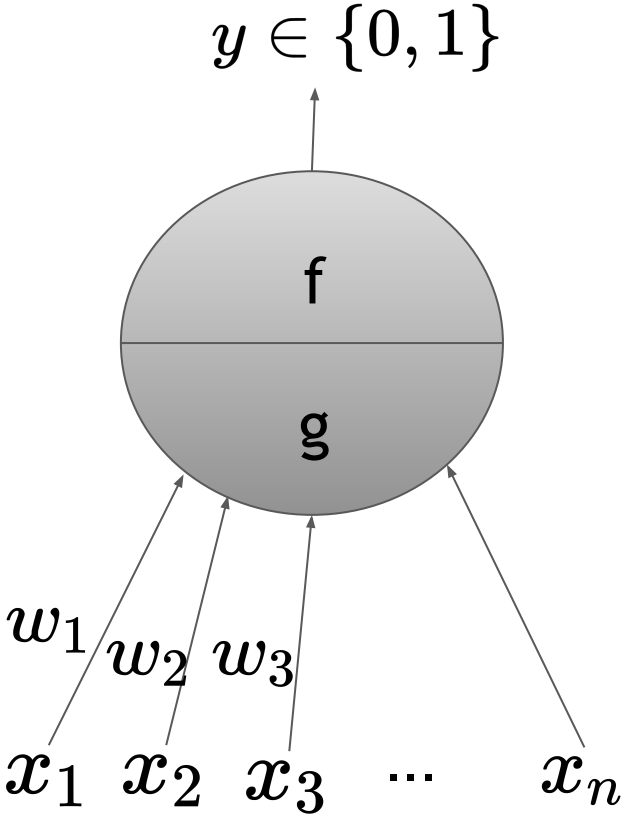
Drawbacks of M-P Neuron



- Only works for Boolean Input
- Thresholds are manually Decided
- All inputs are equal

Perceptron

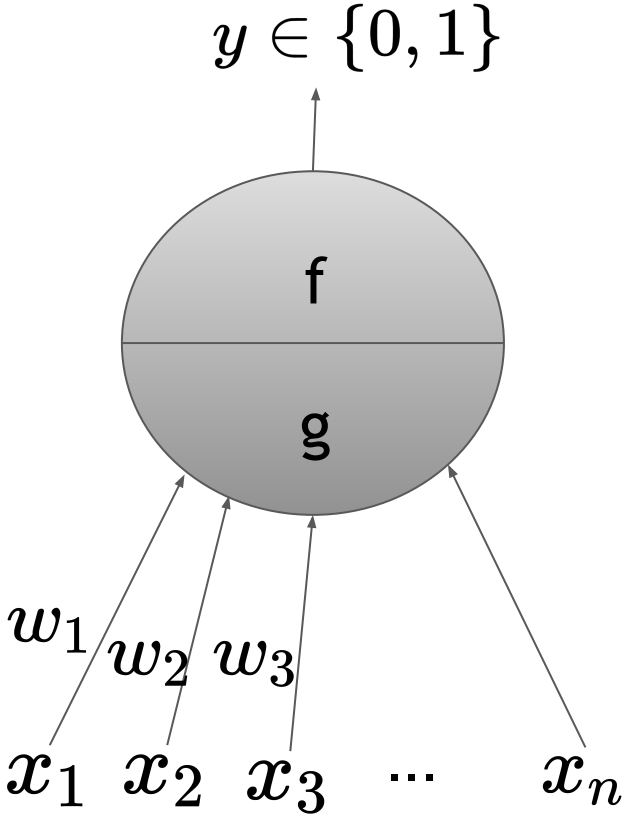
Perceptron



$$y = 1 \;\; if \;\; \sum_1^n w_i x_i >= \theta$$

$$y = 0 \;\; if \;\; \sum_1^n w_i x_i < \theta$$

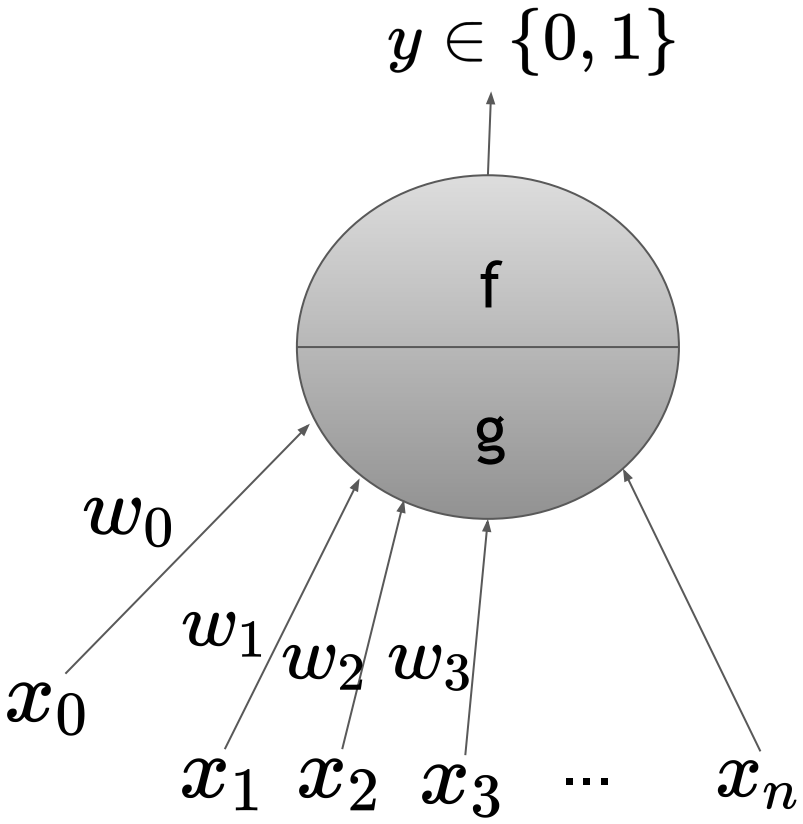
Perceptron



$$y = 1 \;\; if \;\; \sum_1^n w_i x_i - \theta >= 0$$

$$y = 0 \;\; if \;\; \sum_1^n w_i x_i - \theta < 0$$

Perceptron



$$y = 1 \ \ if \ \ \sum_1^n w_i x_i - \theta >= 0$$

$$y = 0 \ \ if \ \ \sum_1^n w_i x_i - \theta < 0$$

$$w_0 = -\theta \ \ and \ \ x_0 = 1$$

$$y = 1 \ if \ \sum_0^n w_i x_i >= 0$$

$$y = 0 \ if \ \sum_0^n w_i x_i < 0$$

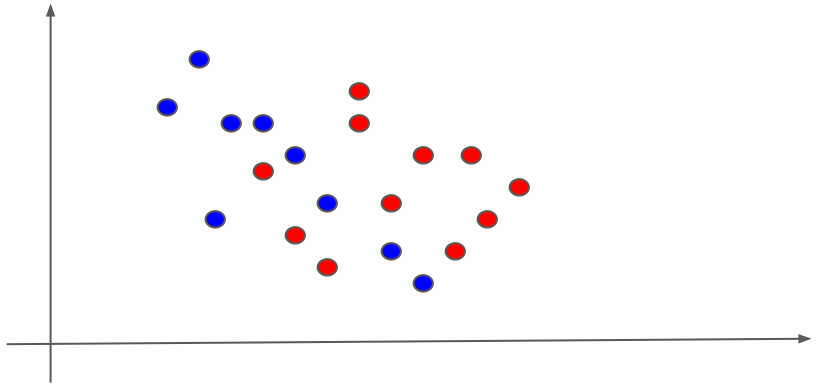
Summary so far...

- We can weight inputs.
- We can classify linearly separable samples.

What about non-linearly separable samples?

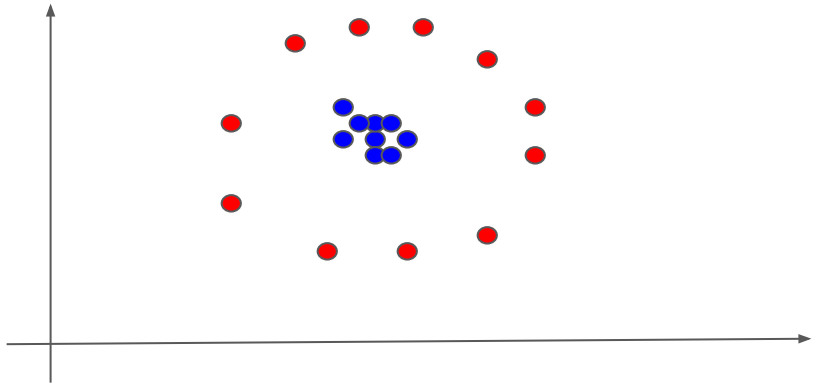
Summary so far...

- We can weight inputs.
- We can classify linearly separable samples.



Summary so far...

- We can weight inputs.
- We can classify linearly separable samples.



A network of Perceptron

Theorem

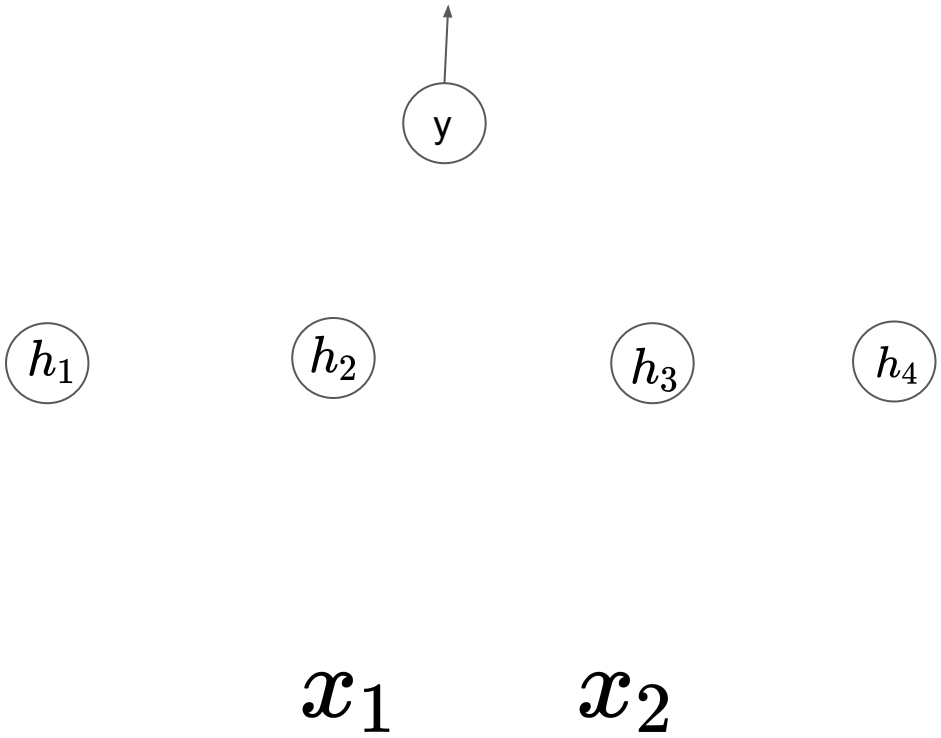
Any Boolean function of n inputs can be represented by a network of perceptron containing 1 hidden layer with 2^n perceptron and one output layer containing one perceptron.

Network of Perceptron

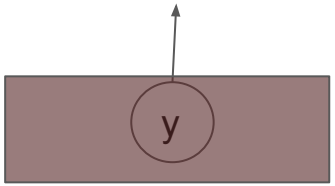
$$x_1 \in \{-1, 1\}$$
$$x_2 \in \{-1, 1\}$$

x_1 x_2

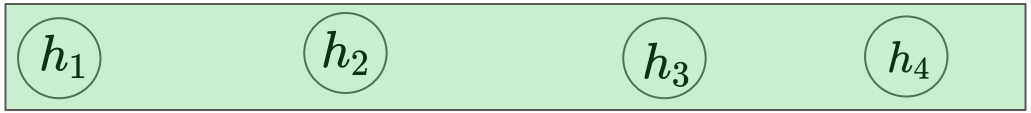
Network of Perceptron



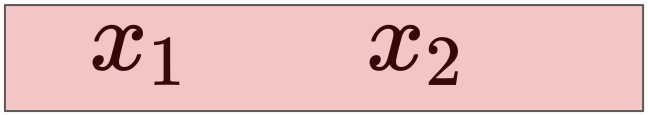
Network of Perceptron



Output Layer

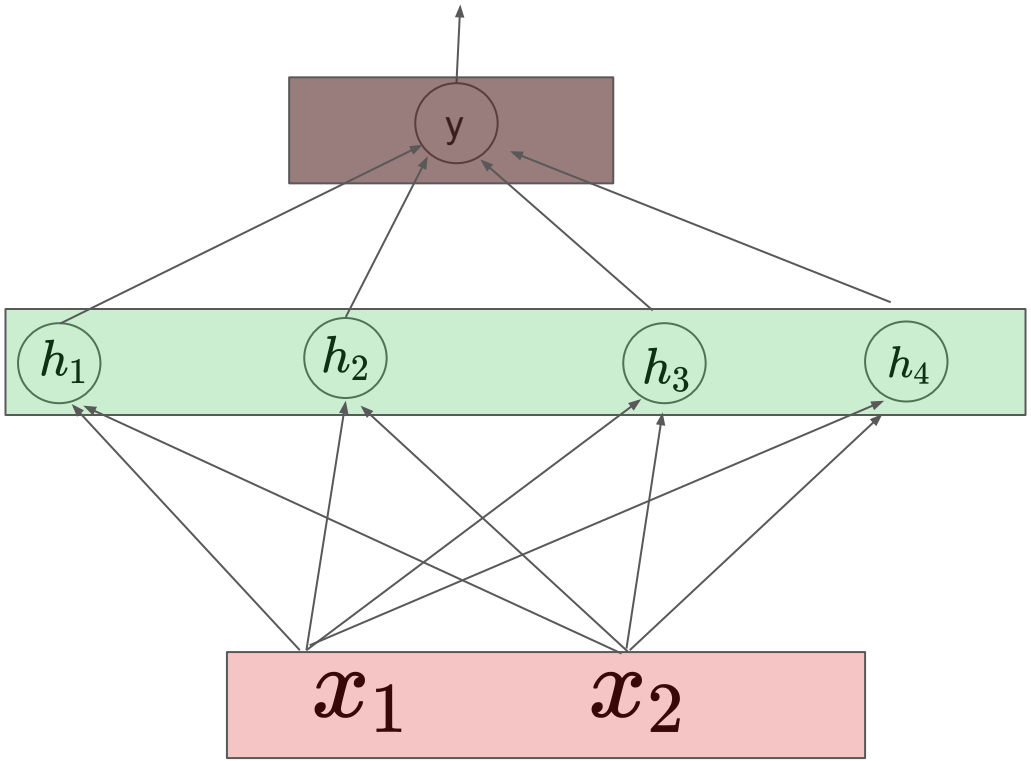


Hidden Layer



Input Layer

Network of Perceptron



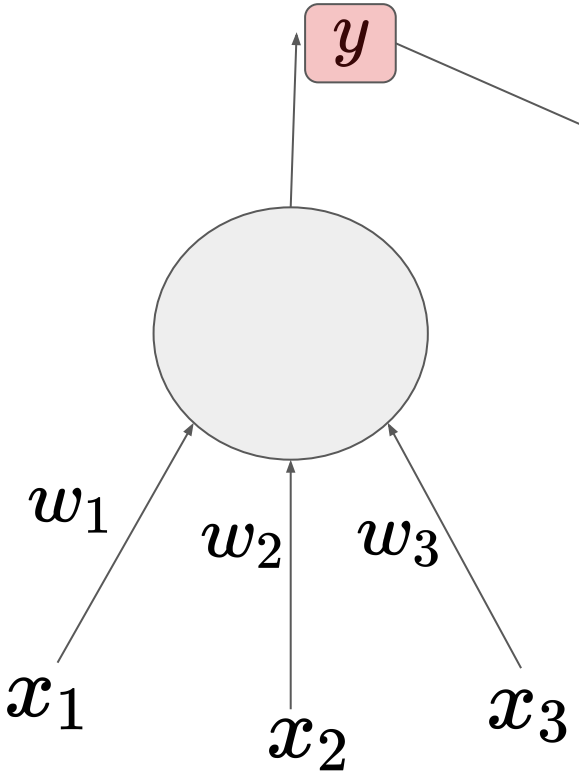
Output Layer

Hidden Layer

Input Layer

Sigmoid Neuron

So far only Boolean functions

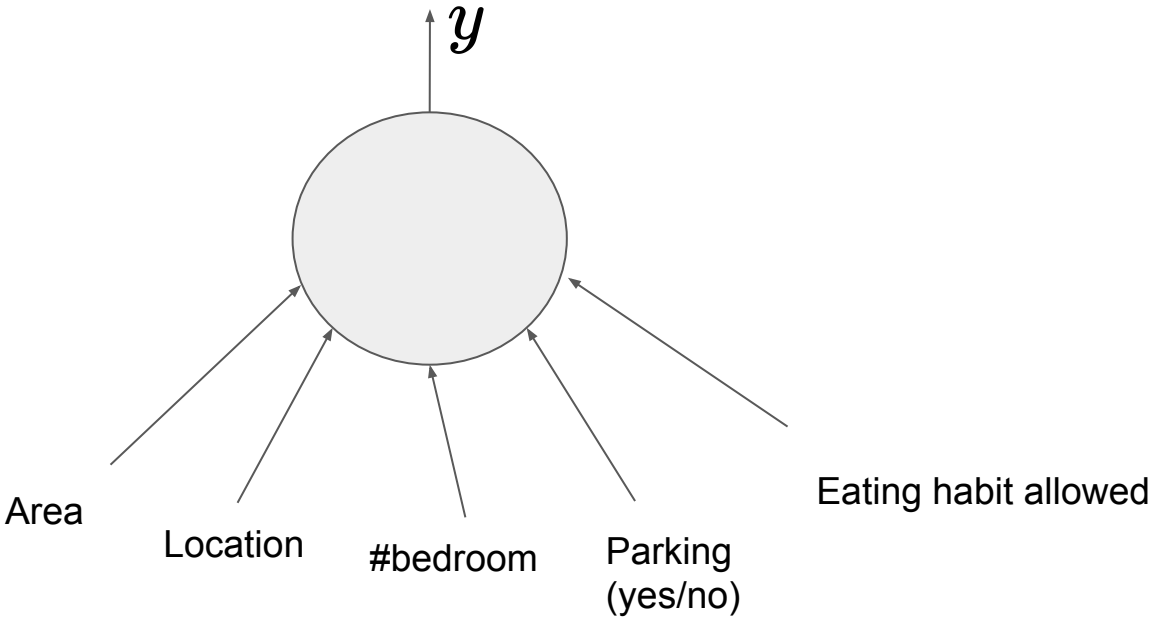


0 or 1

What if y is a real number?

Problems with perceptron

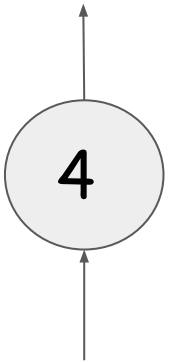
Problem 1: So far only models a Boolean Function.
Estimate the rent cost of a real-estate property?



Problems with perceptron

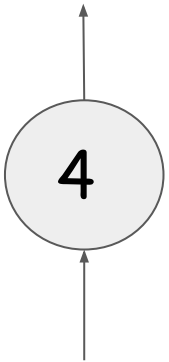
Problem 2: Hard Thresholding

You should watch movie



Average Movie Rating (4.1)

You should not watch movie

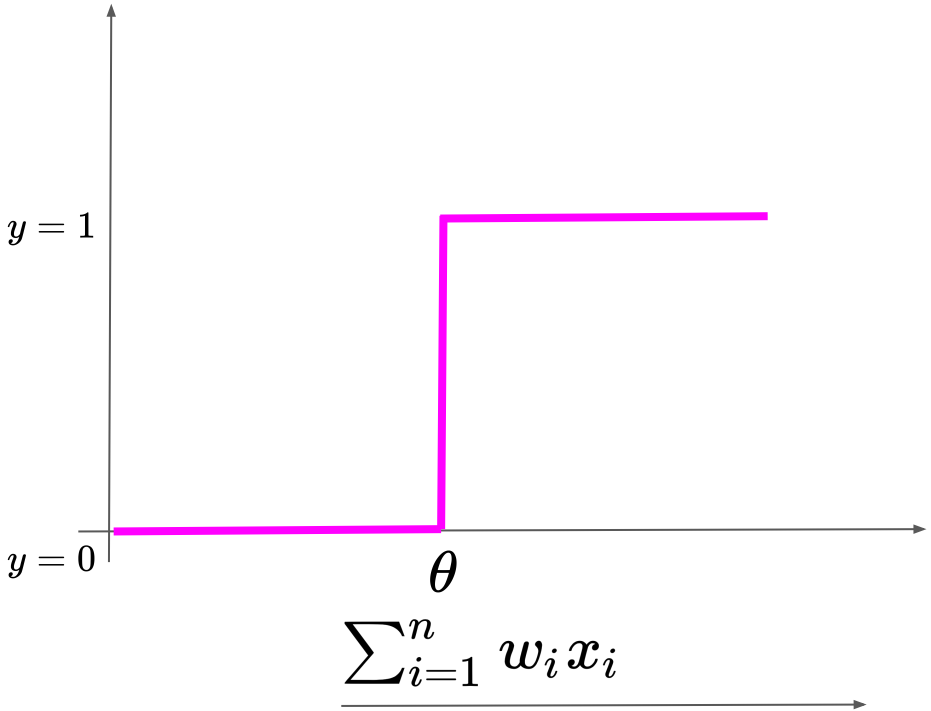


Average Movie Rating (3.9)

Decision function

$$y = 0 \;\; if \;\; \sum_{i=1}^{n} w_i x_i < \theta$$

$$y = 1 \;\; if \;\; \sum_{i=1}^{n} w_i x_i >= \theta$$

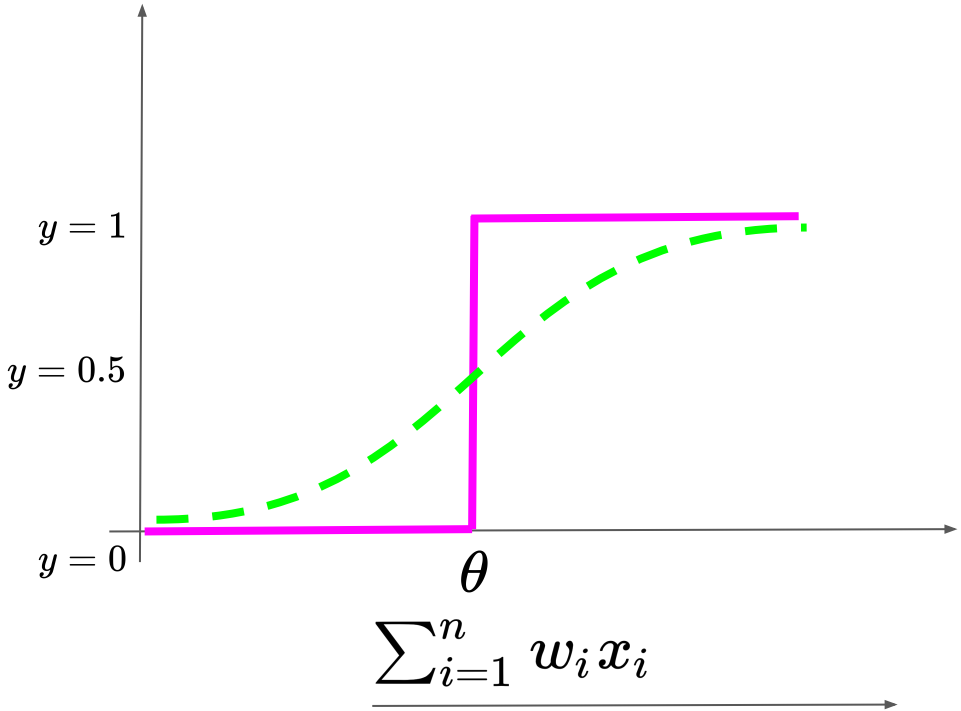


Decision function

$$y = 0 \;\; if \;\; \sum_{i=1}^{n} w_i x_i < \theta$$

$$y = 1 \;\; if \;\; \sum_{i=1}^{n} w_i x_i >= \theta$$

$$y = \frac{1}{1+e^{\sum_{i=0}^{n} w_i x_i}}$$



Decision function

Perceptron decision function

Non smooth, non-differentiable, non-continuous

Sigmoid decision function

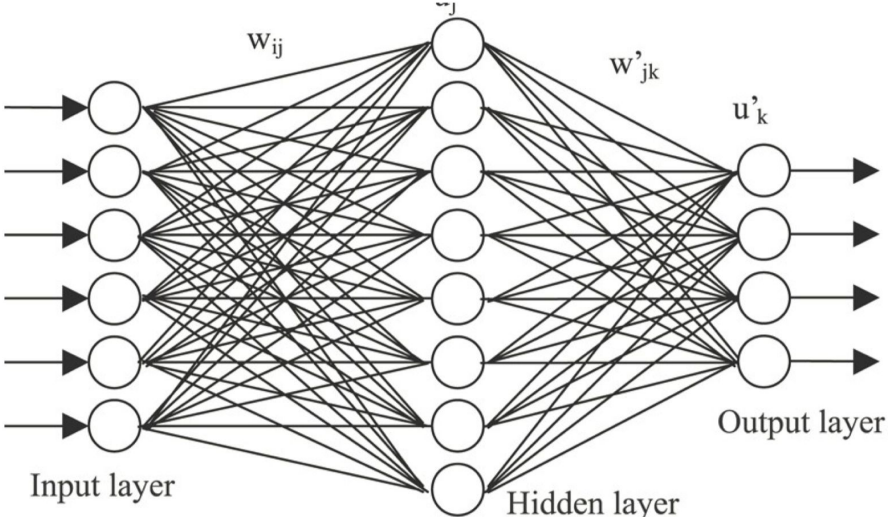
Smooth, differentiable, continuous

Summary

- Perceptrons are powerful enough to do a good job for classifying linearly-separable samples.
- There exists a network of perceptron which can classify samples correctly for a given problem. (Network of Perceptron is extremely powerful)

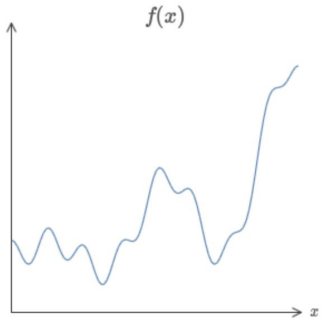
Training a neural Network - 1

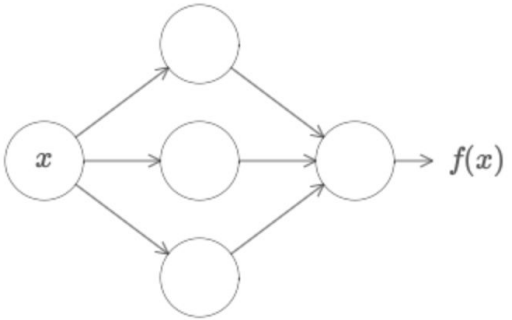
So far ...

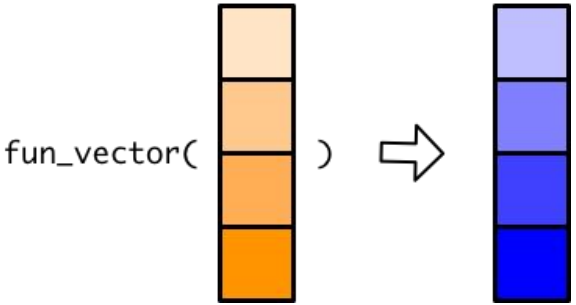


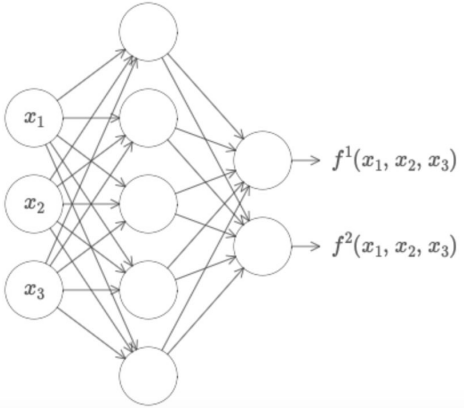
- Neural network (Network of Perceptron/MLP) can:
- model any Boolean function
- Model any decision boundary
- Model any continuous valued function (how?)

Neural Network can compute any func.



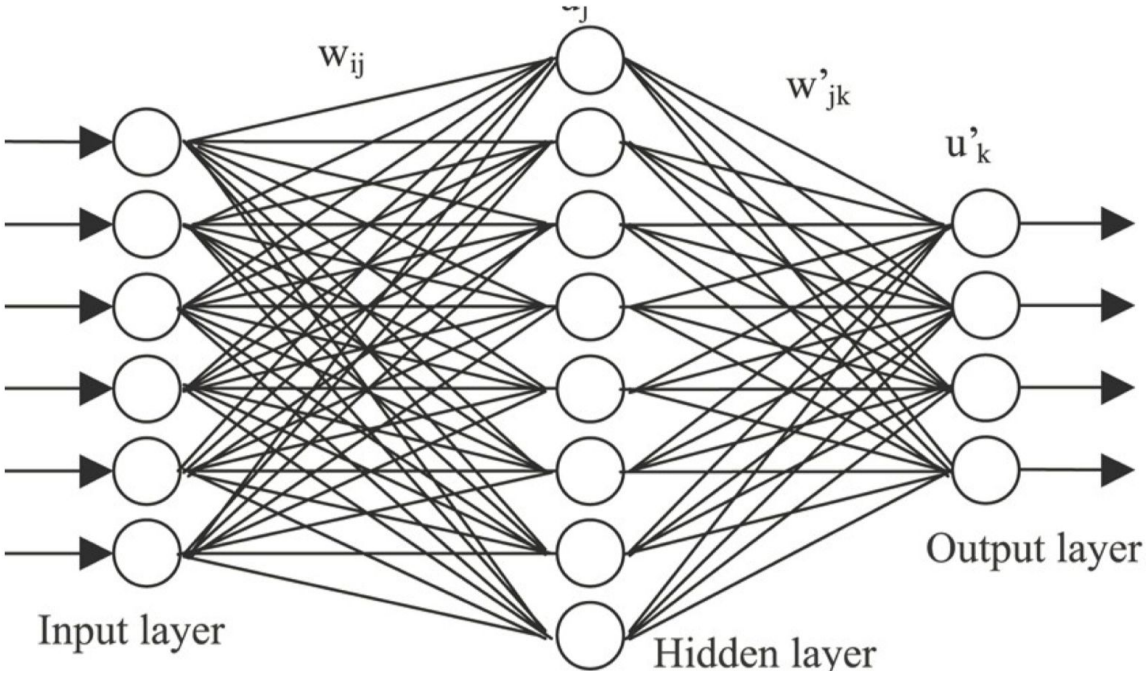






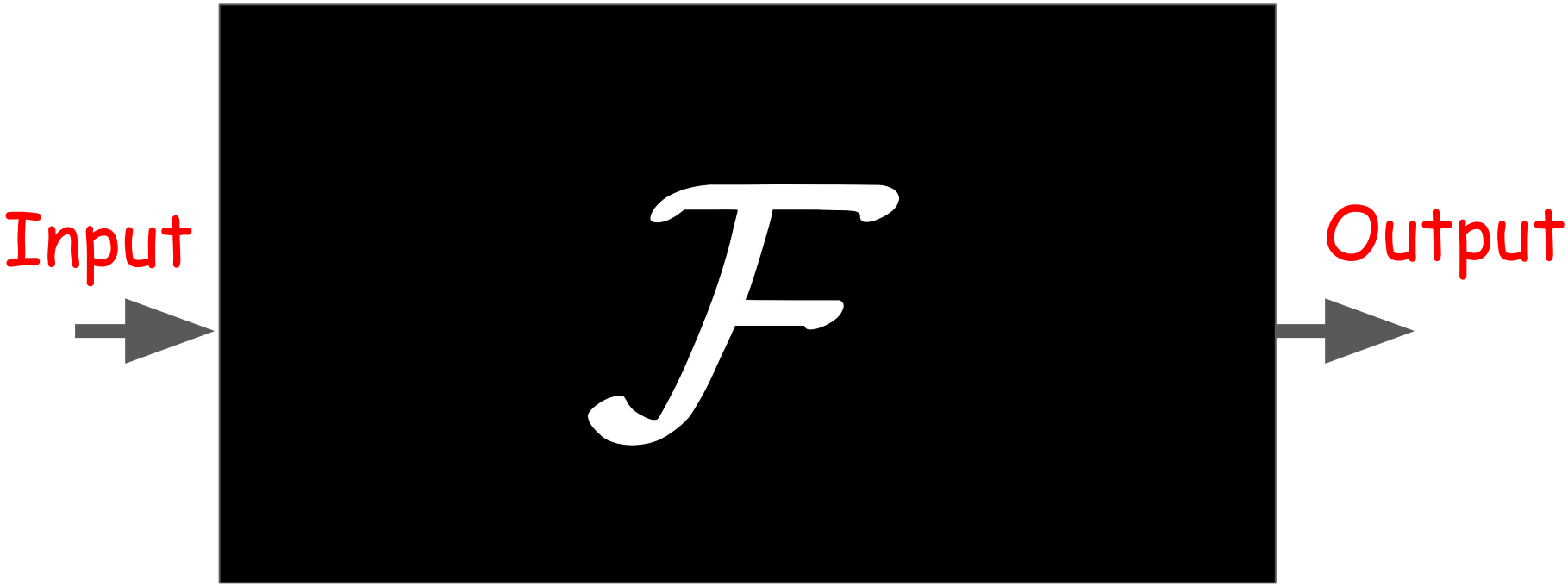
Neural Network

Input

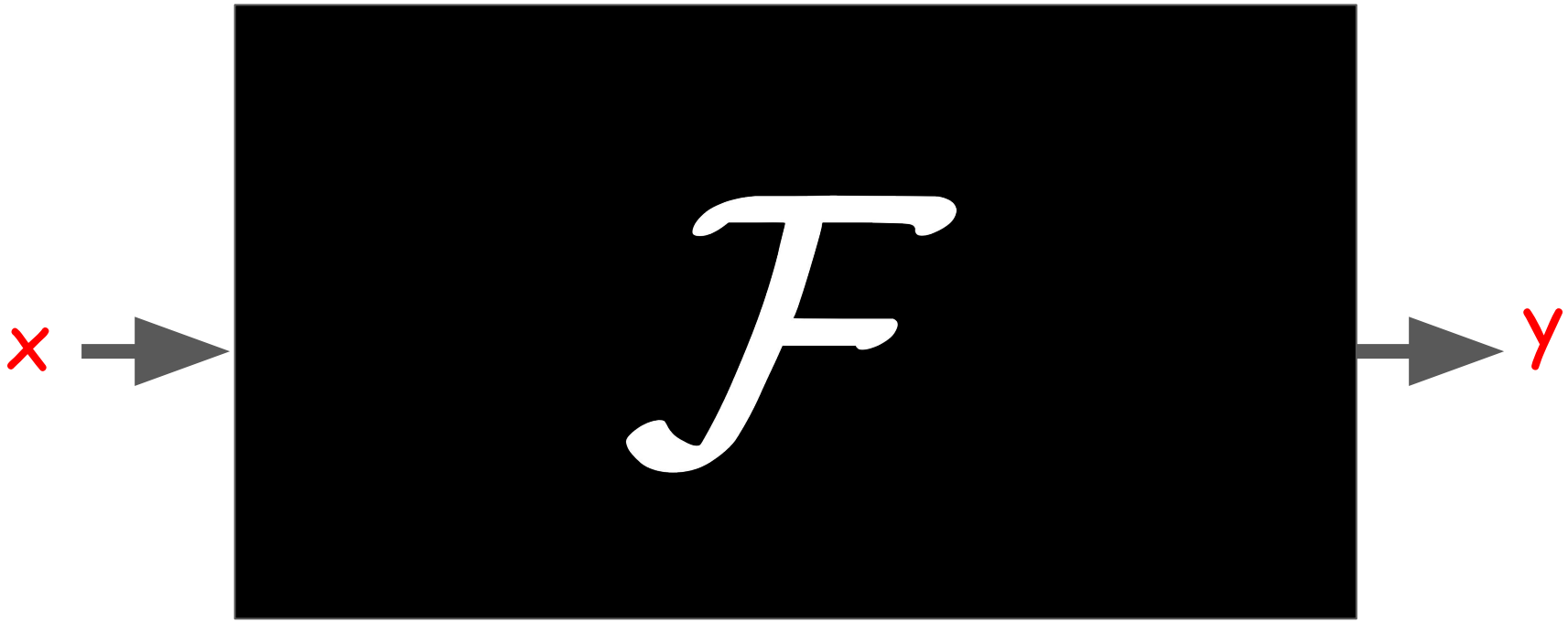


Output

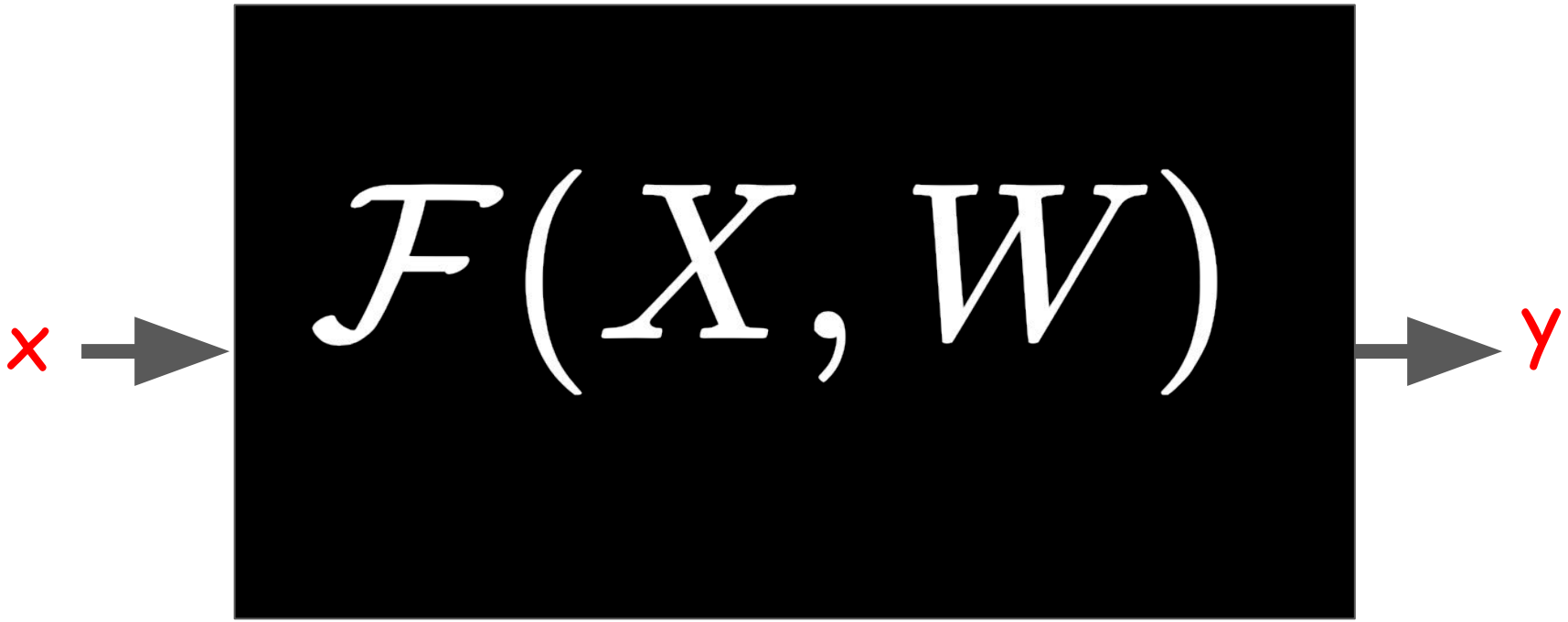
Neural Network as a function



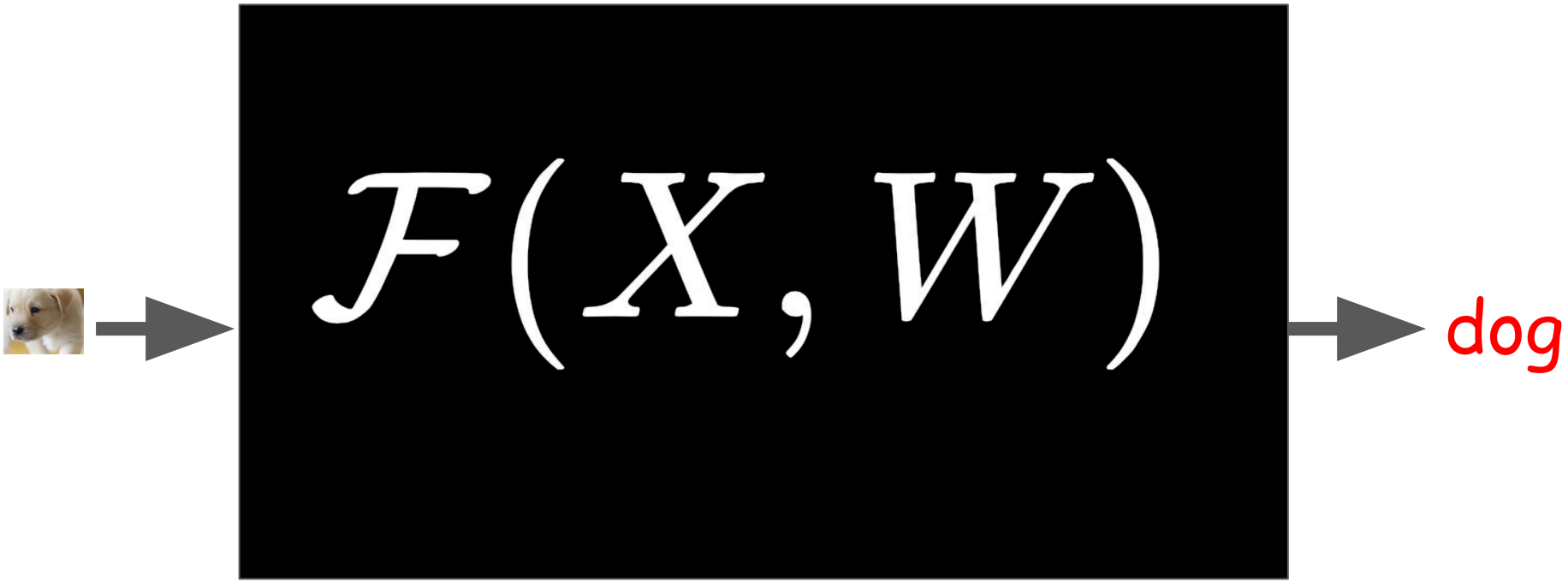
Neural Network as a function



Neural Network as a function



Neural Network as a function

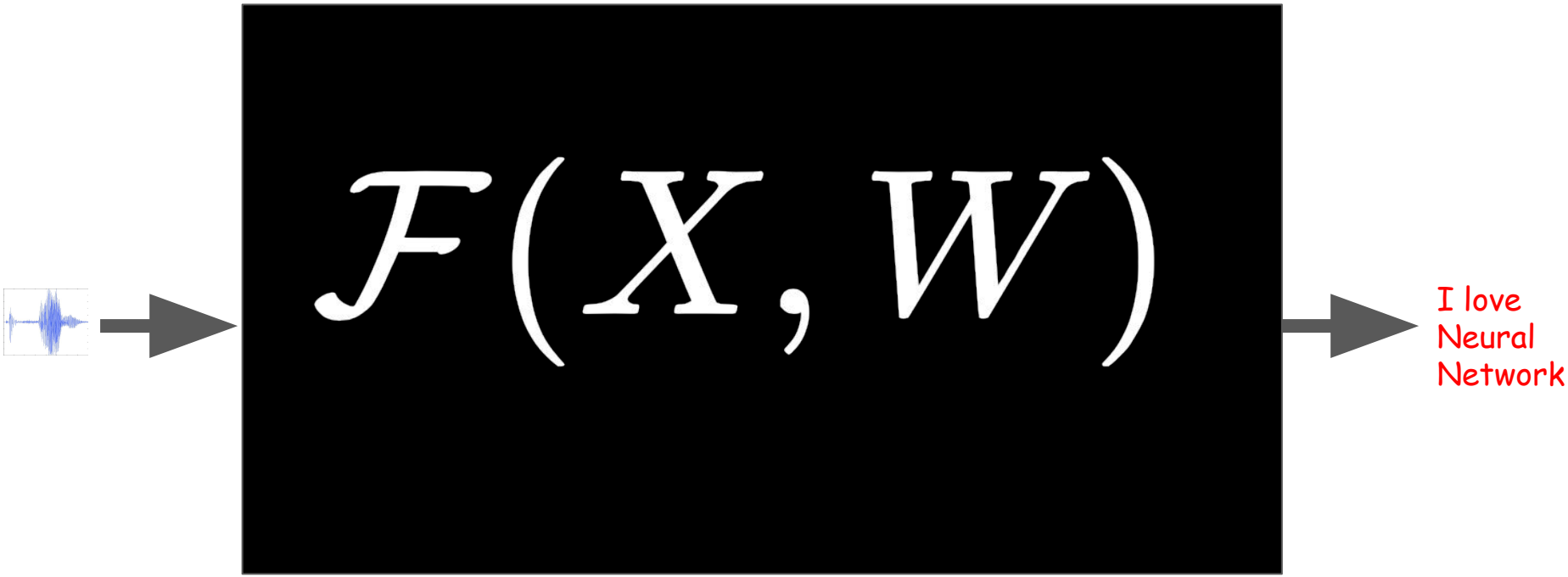


Neural Network as a function

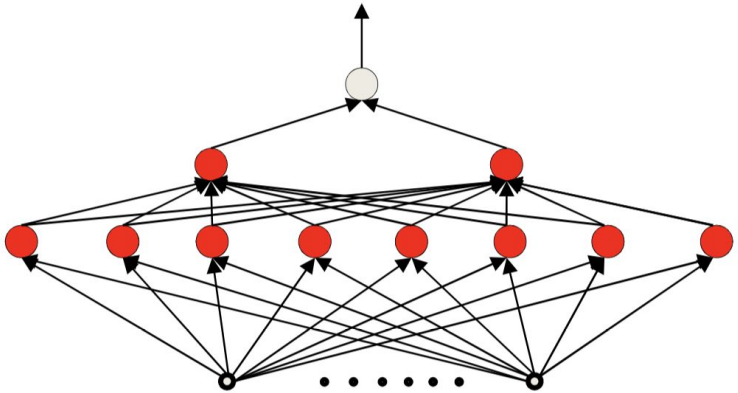
$$\mathcal{F}(X, W)$$

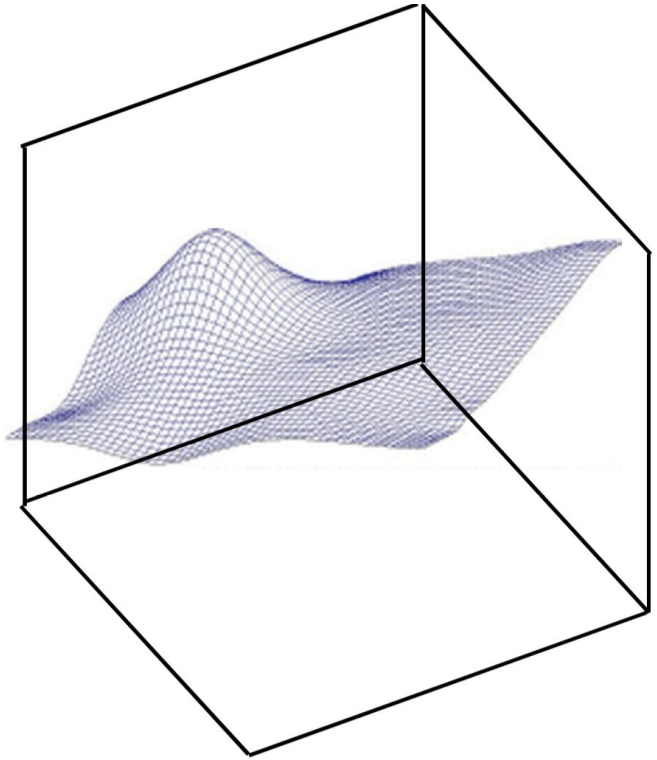
cat

Neural Network as a function

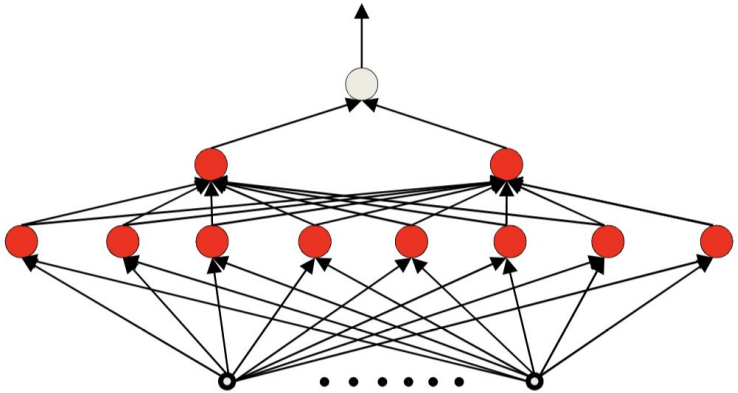


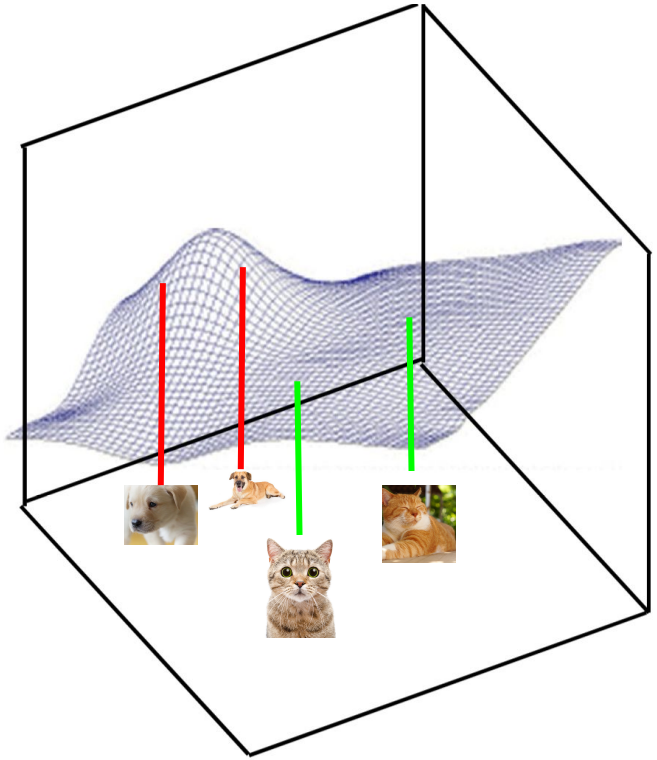
MLP Can represent anything





MLP Can represent anything





MLP Can represent anything

Option 1: Construct by hand

Option 2: Automatic estimation of an MLP

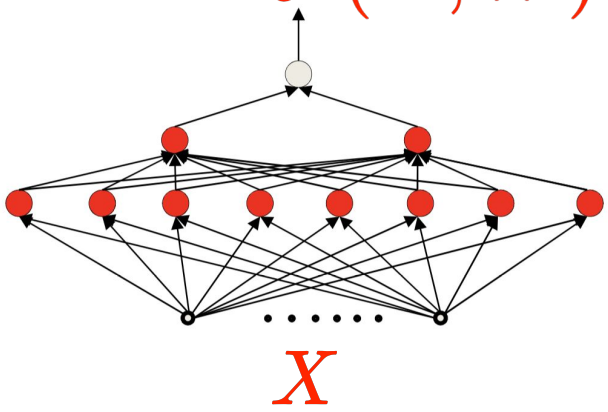
MLP Can represent anything

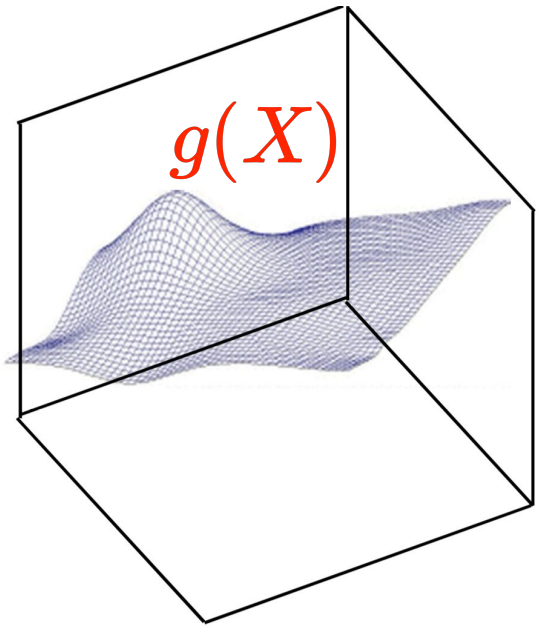
~~Option 1: Construct by hand~~

Option 2: Automatic estimation of an MLP

Automatic estimation of MLP

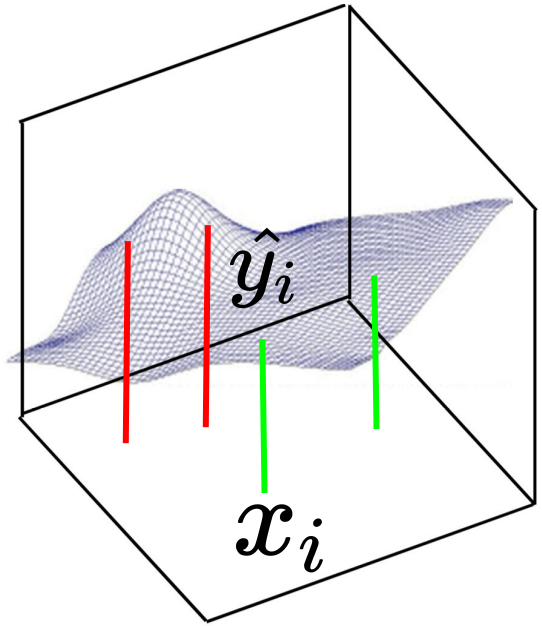
$$Y = \mathcal{F}(X, W)$$





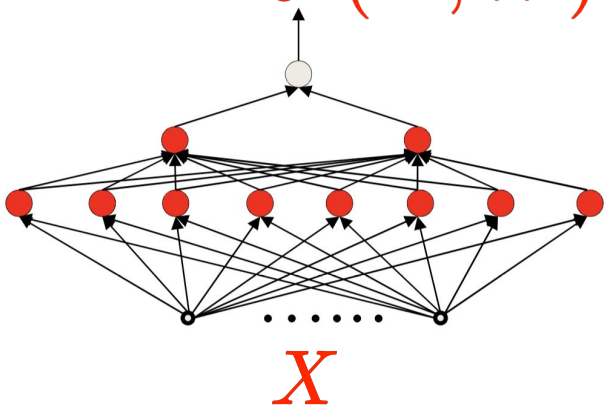
$$W^* = \arg\min_W \int_X \mathcal{DIV}\big(\mathcal{F}(X, W), g(X)\big)$$

Problem: g(x) is not known everywhere

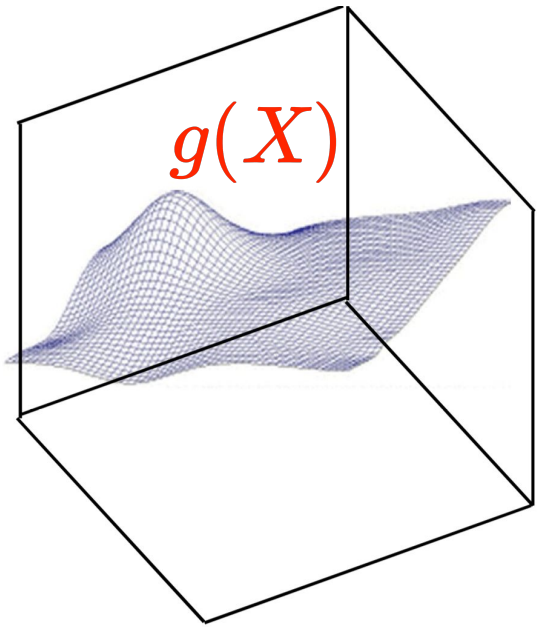


Automatic estimation of MLP

$$Y = \mathcal{F}(X, W)$$



X



$$W^* = \arg\min_W \sum_{i=1}^{n} \mathcal{DIV}\left(\mathcal{F}(x_i, W), \hat{y}_i\right)$$

Module 1: Learning Algorithm

Supervised Learning Setup

Data: $\mathcal{D} = \{\mathbf{x}_i, \hat{y}_i\}_{i=1}^{n}$, *where* $\mathbf{x}_i = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$ is a sample, and $\hat{y}_i \in \{+1, -1\}$ is a class label.

Supervised Learning Setup

Data: $\mathcal{D} = \{\mathbf{x}_i, \hat{y}_i\}_{i=1}^{n}$, *where* $\mathbf{x}_i = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$ is a sample, and $\hat{y}_i \in \{+1, -1\}$ is a class label.

$$y = \mathcal{F}(\mathbf{x}, \mathbf{w})$$

Supervised Learning Setup

Data: $\mathcal{D} = \{\mathbf{x}_i, \hat{y}_i\}_{i=1}^{n}$, *where* $\mathbf{x}_i = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$ is a sample, and $\hat{y}_i \in \{+1, -1\}$ is a class label.

$$y = \mathcal{F}(\mathbf{x}, \mathbf{w})$$

Supervised Learning Setup

Data: $\mathcal{D} = \{\mathbf{x}_i, \hat{y}_i\}_{i=1}^{n}$, *where* $\mathbf{x}_i = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$ is a sample, and $\hat{y}_i \in \{+1, -1\}$ is a class label.

Model: $y = \mathcal{F}(\mathbf{x}, \mathbf{w})$ where $\mathcal{F}$ is a neural network.

Parameters: $\mathbf{w}$ needs to be learnt.

Learning Algo: Perceptron learning algo., gradient descent

Loss function: To guide learning algorithm

The problem

$$W^* = \arg\min_W \sum_{i=1}^{n} \mathcal{DIV}\left(\mathcal{F}(x_i, W), \hat{y}_i\right)$$

The problem

$$\mathcal{L}(W) = \mathcal{DIV}(\mathcal{F}(W, x), \hat{y})$$

Divergence (intuitively a function f(a,b) which has lower value when a = b)

Example of Divergence function

- Least Mean Square

$$DIV(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

- Cross-entropy

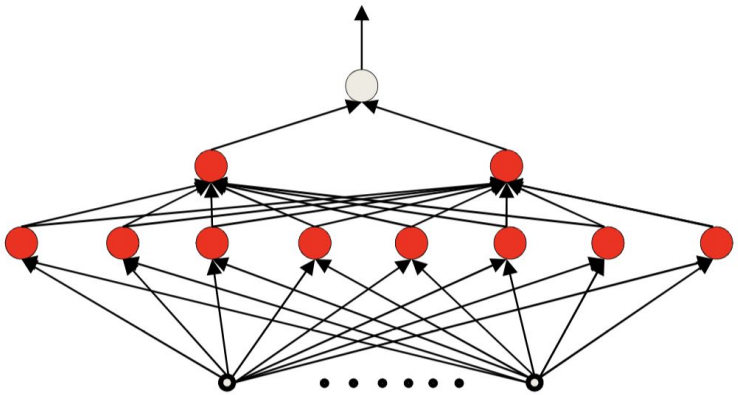
$$\mathcal{DIV}(y, \hat{y}) = -ylog(\hat{y}) - (1 - y)log(1 - \hat{y}))$$

The Problem

$$\mathcal{L}(W) = \mathcal{DIV}(\mathcal{F}(W, x), \hat{y})$$

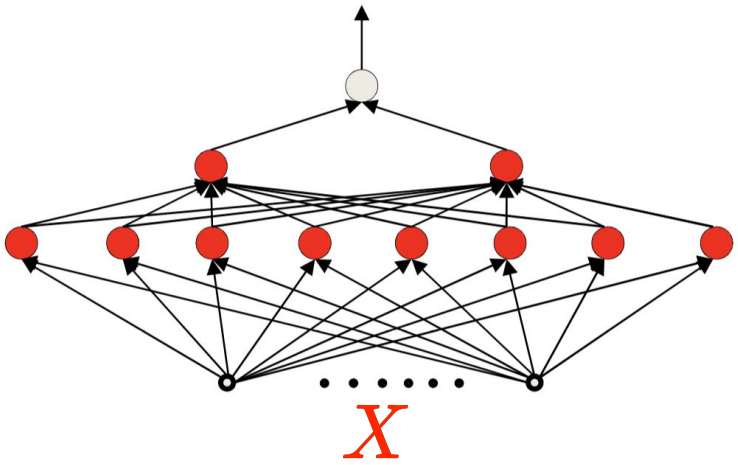
Is a neural network parameterized by W and takes input x

What is F?



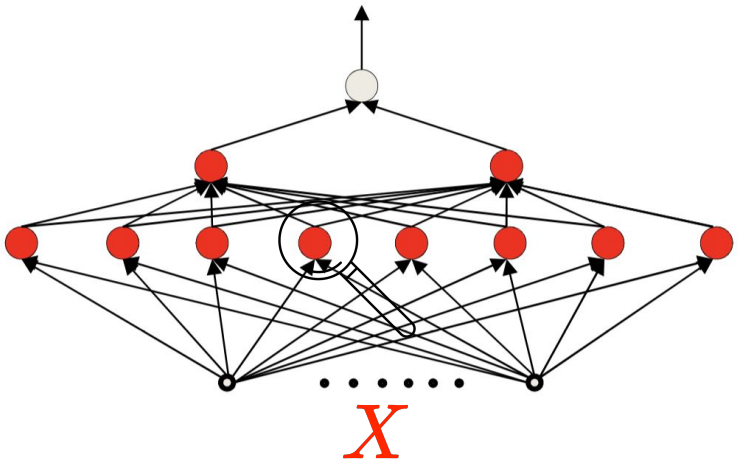
What is F?

$$Y = \mathcal{F}(X, W)$$

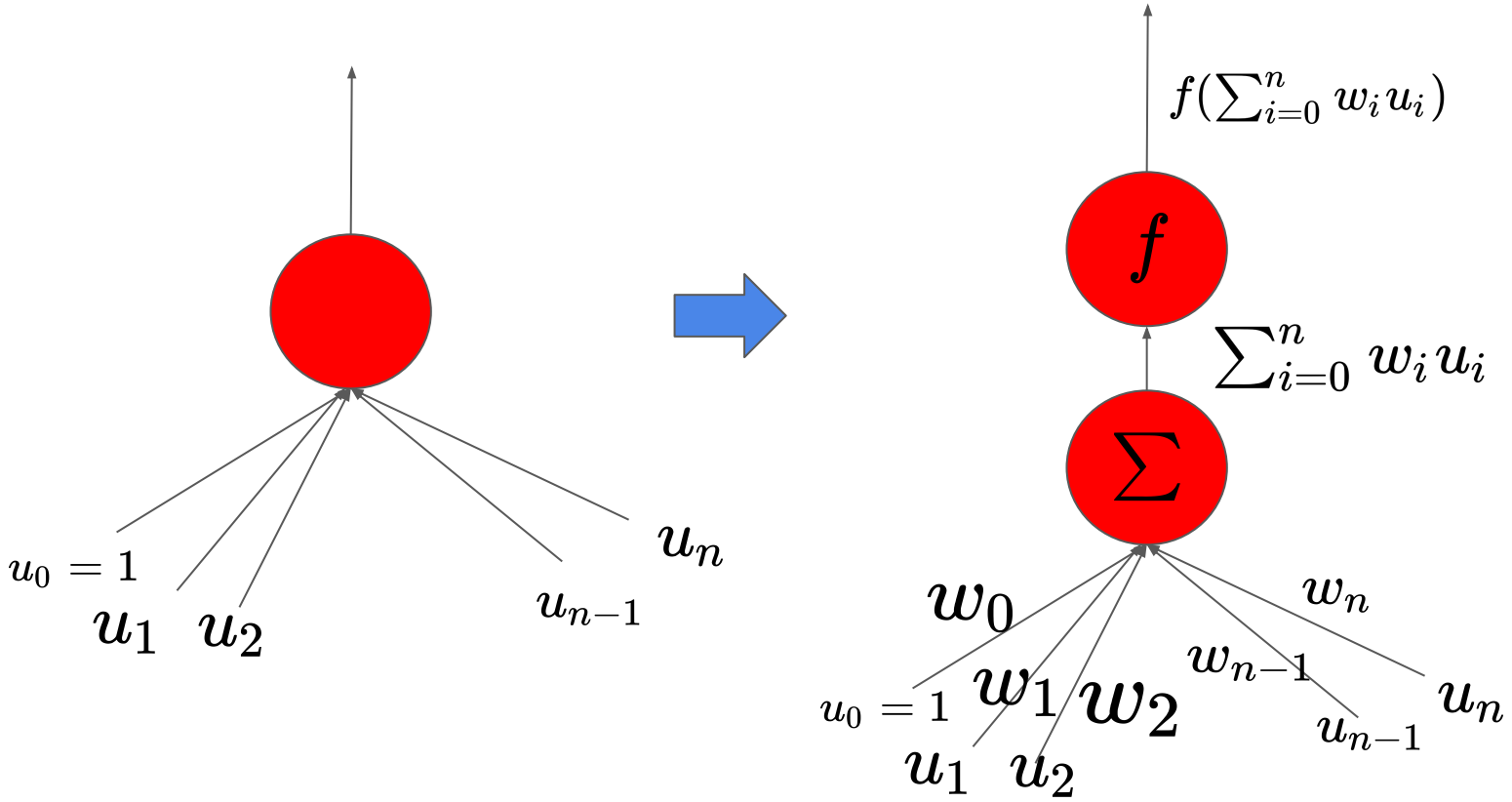


What is F?

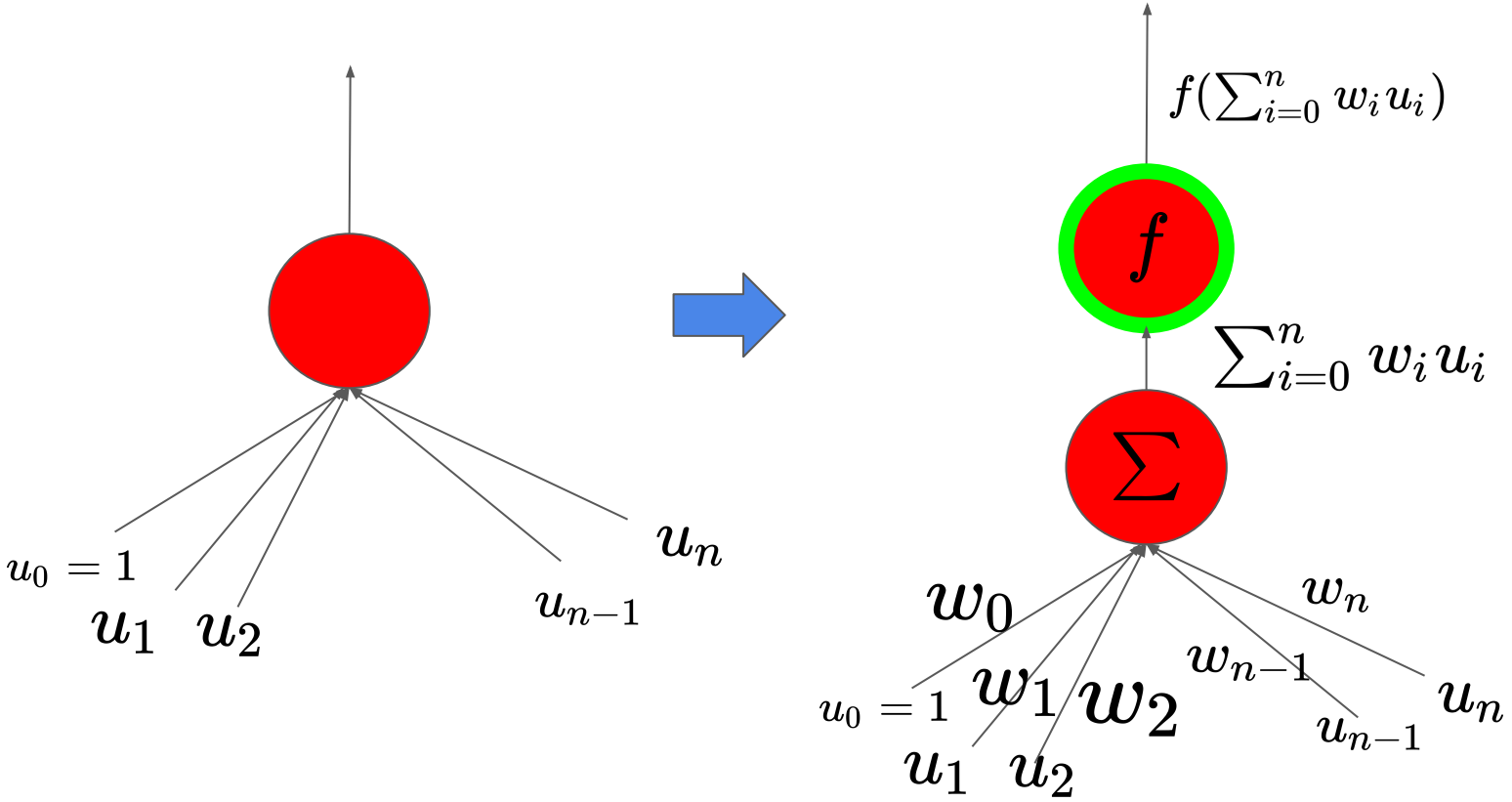
$$Y = \mathcal{F}(X, W)$$



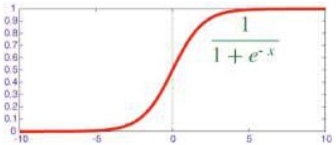
How does each neuron look?



How does each neuron look?



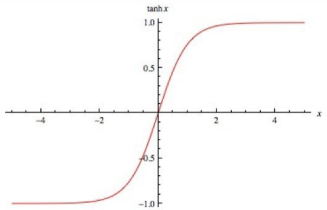
What is f ?



$$f(z) = \frac{1}{1+\exp(-z)}$$

$$f'(z) = f(z)(1-f(z))$$

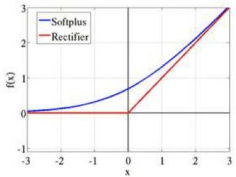
Sigmoid



$$f(z) = \tanh(z)$$

$$f'(z) = (1-f^2(z))$$

tanh



$$f(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$

[*] $$f'(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

ReLU

$$f(z) = \log(1+\exp(z))$$

$$f'(z) = \frac{1}{1+\exp(-z)}$$

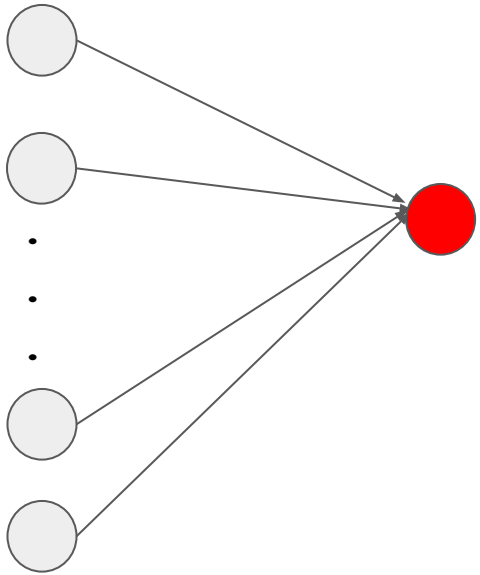
Log likelihood

Back to the problem …

$$\mathcal{L}(W) = \mathcal{DIV}(\mathcal{F}(W, x), \hat{y})$$

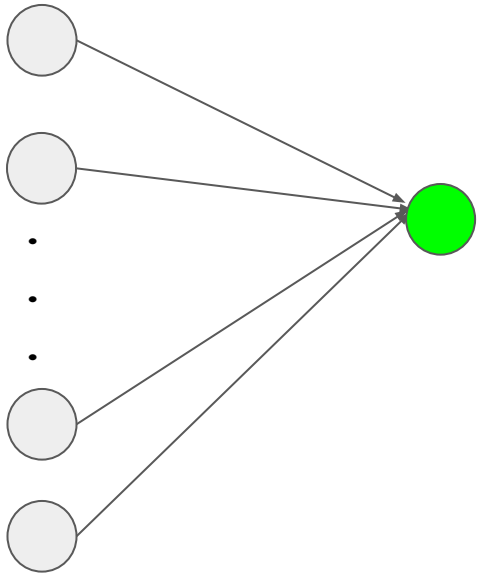
Groundtruth label

Example of $\hat{y}$: Cat vs dog Classifier



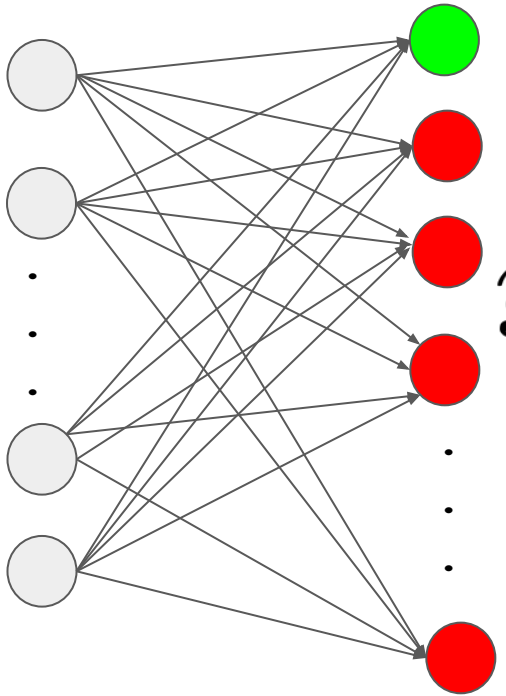
$$\hat{y} = 0$$

Example of $\hat{y}$: Cat vs dog Classifier



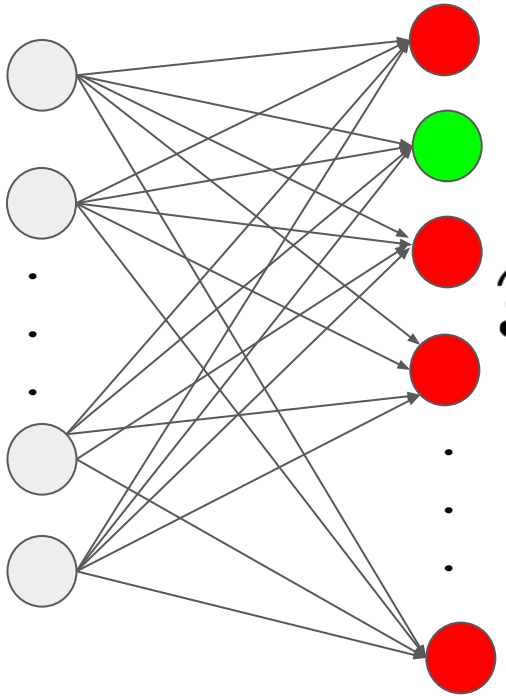
$$\hat{y} = 1$$

Example of $\hat{y}$: Digit classification



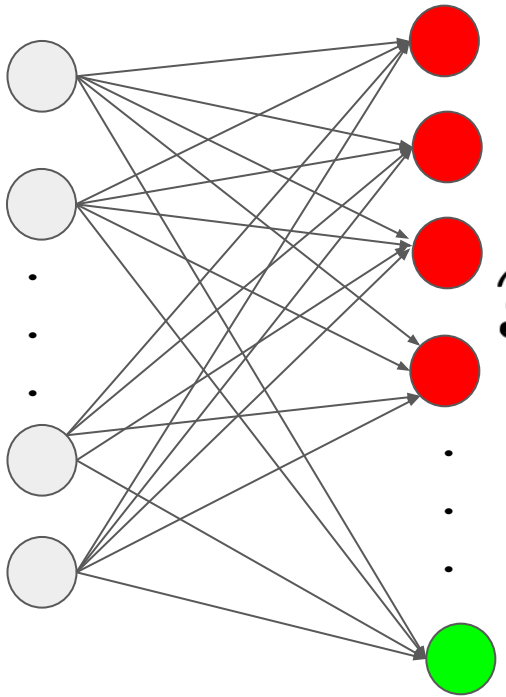
$$\hat{y} = [1, 0, 0, 0, \cdots, 0]$$

Example of $\hat{y}$: Digit classification



$$\hat{y} = [0, 1, 0, 0, \cdots, 0]$$

Example of $\hat{y}$: Digit classification



$$\hat{y} = [0, 0, 0, 0, \cdots, 1]$$

The problem

$$W^* = \arg\min_W \sum_{i=1}^{n} \mathcal{DIV}\left(\mathcal{F}(x_i, W), \hat{y}_i\right)$$

Gradient Descent

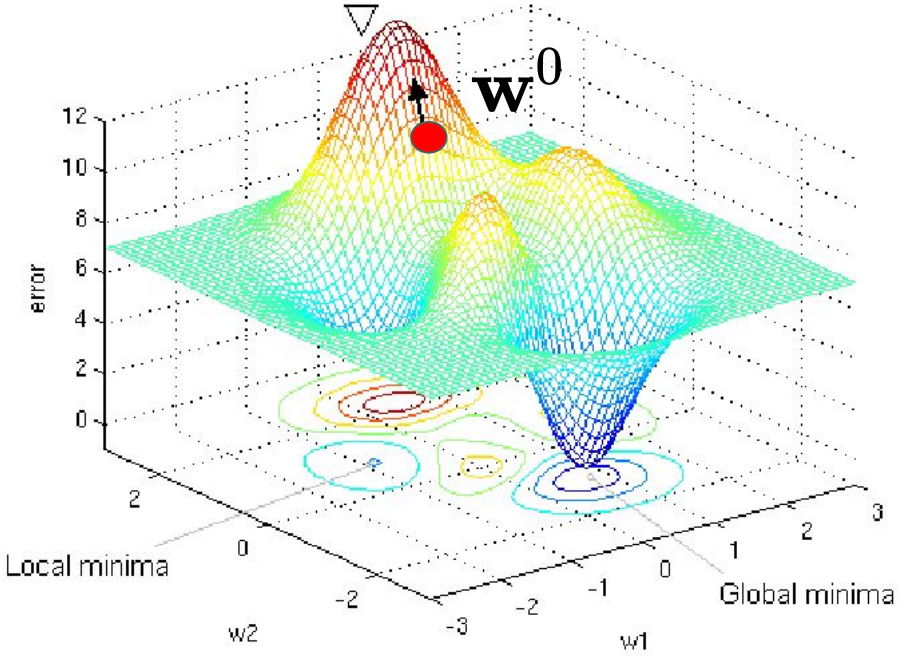
1. Initialize $iteration(t) \leftarrow 0, \eta, \mathbf{w}^0$
2. do

   (i) $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla f(\mathbf{w^t})^T$

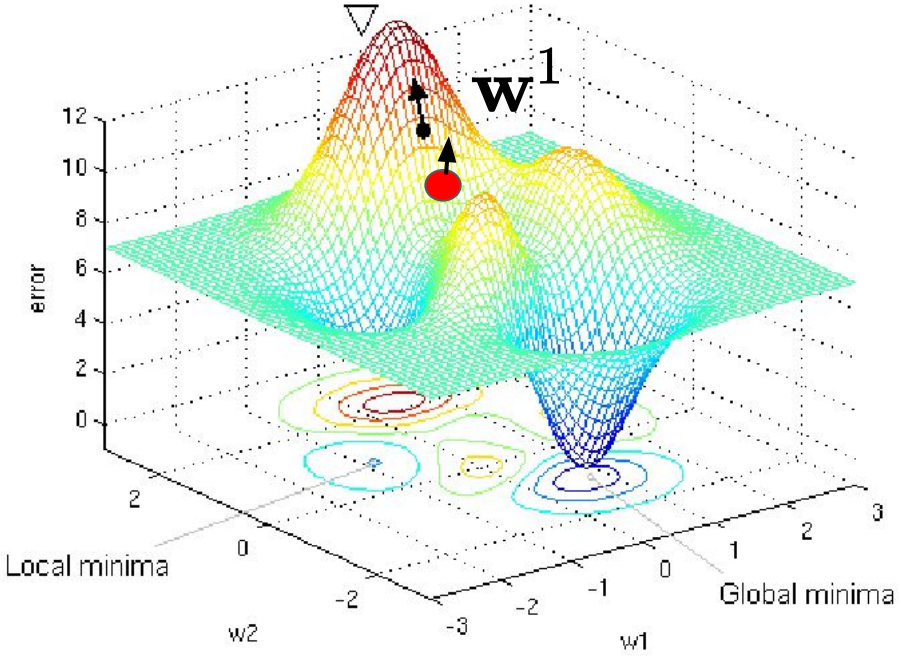
   (ii) $t = t + 1$

   While $|f(\mathbf{w}^t) - f(\mathbf{w}^{t+1})| > \epsilon$

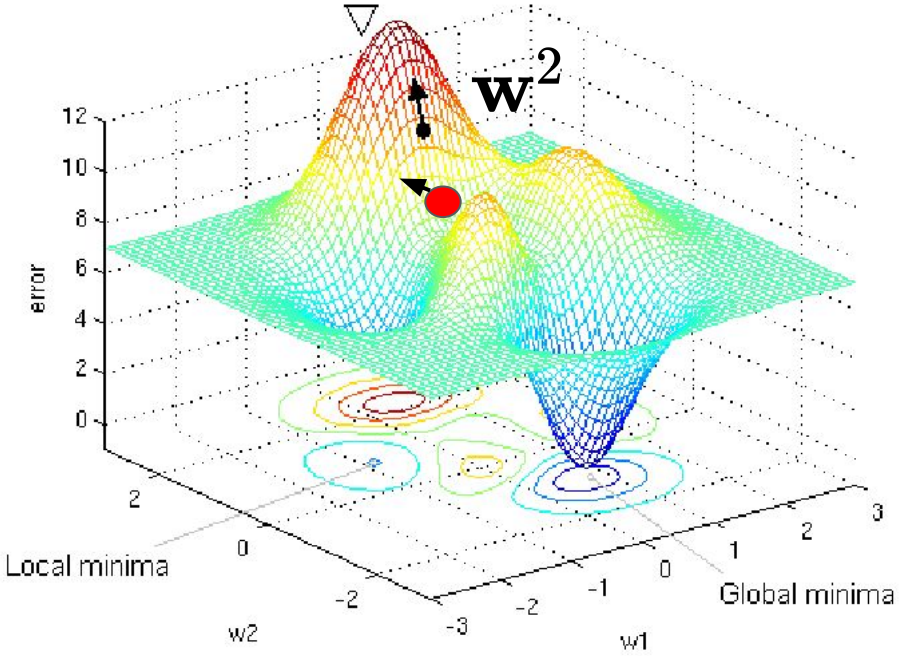
Gradient Descent in action



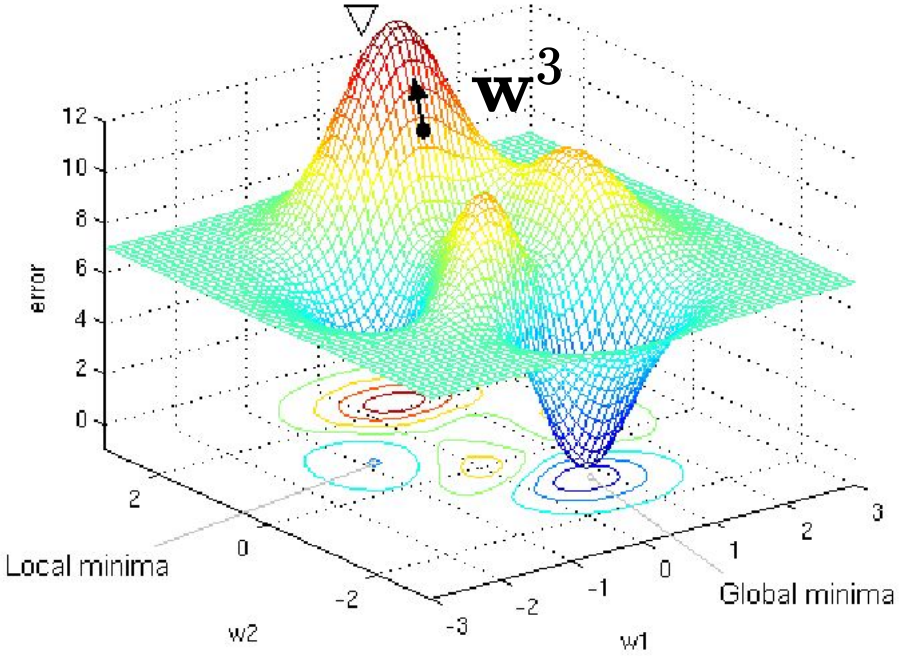
Gradient Descent in action



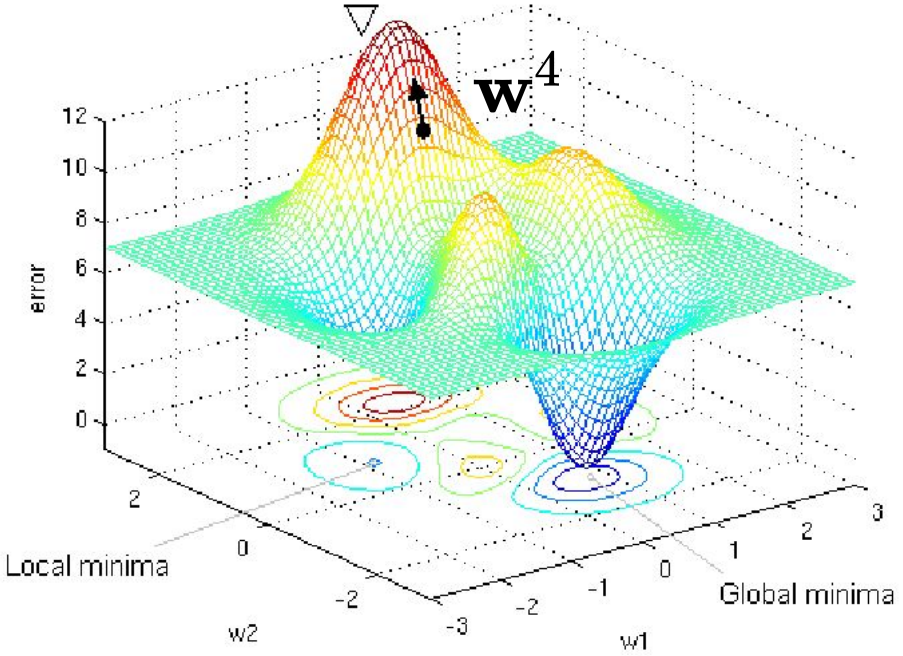
Gradient Descent in action



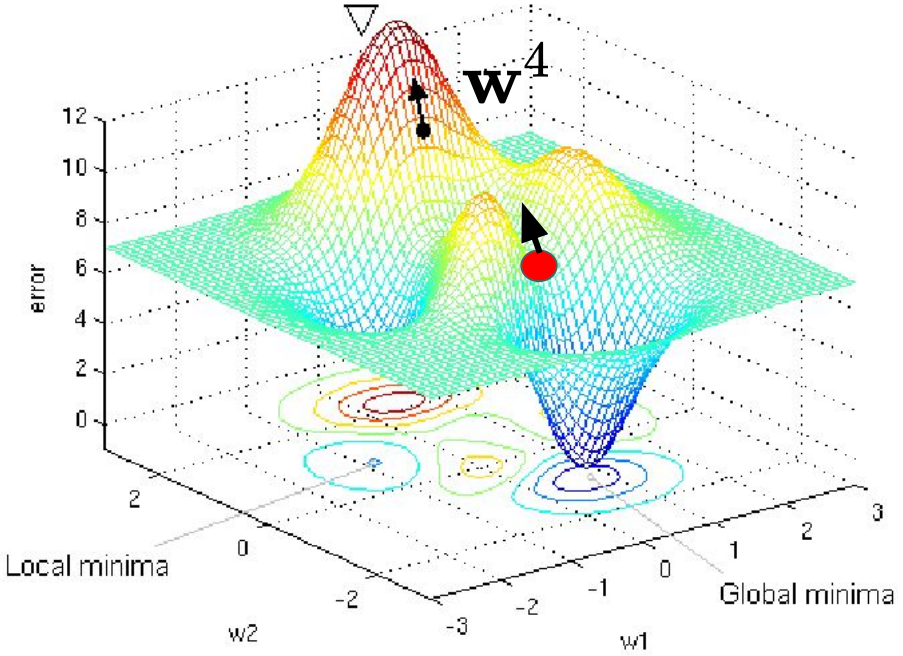
Gradient Descent in action



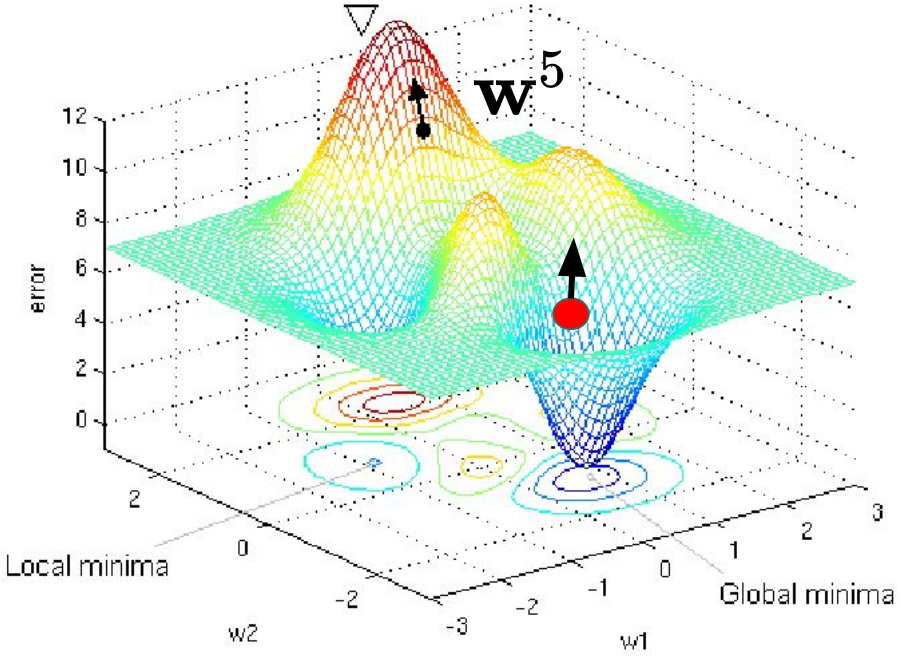
Gradient Descent in action



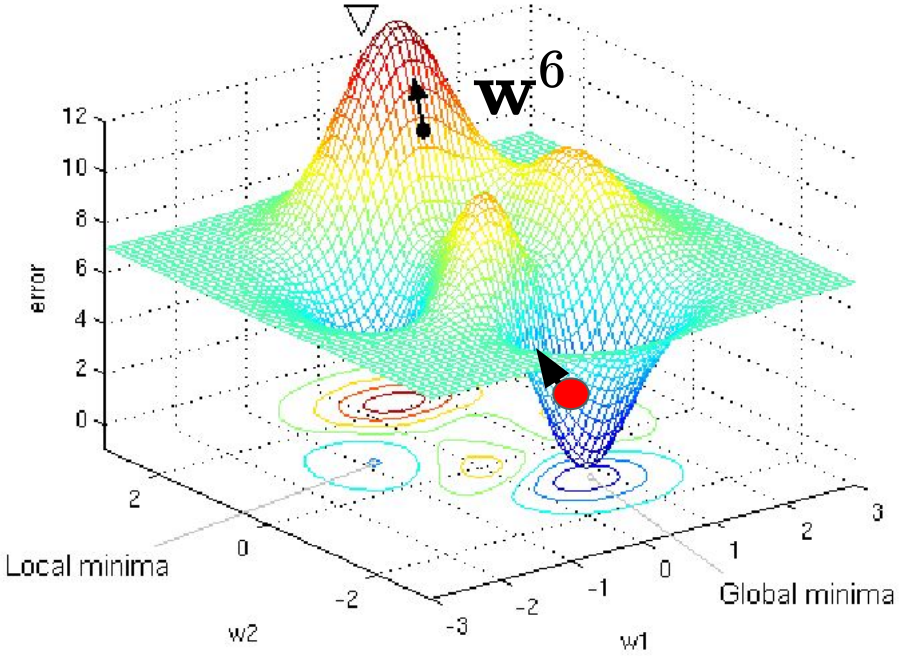
Gradient Descent in action



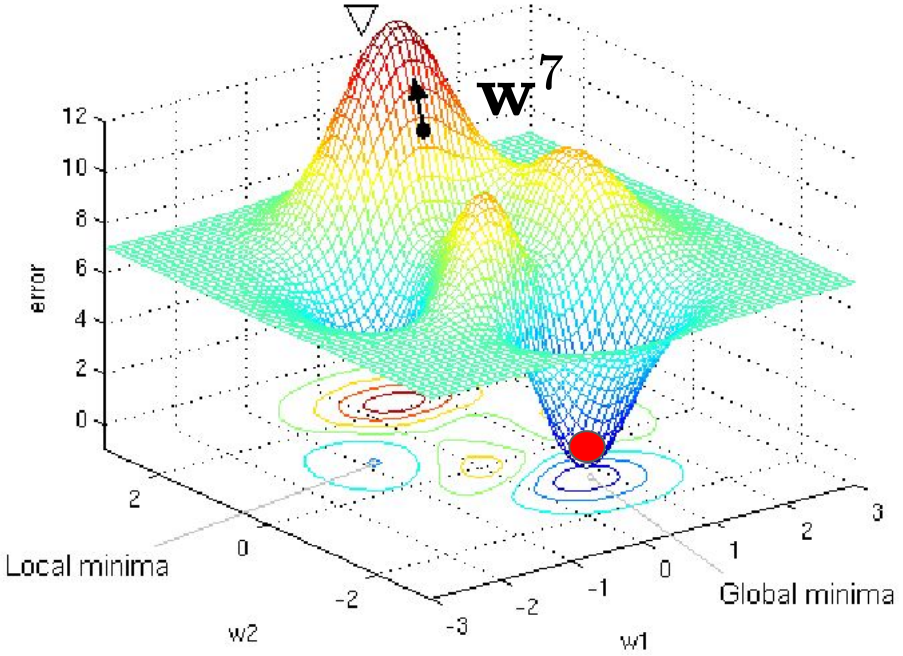
Gradient Descent in action



Gradient Descent in action

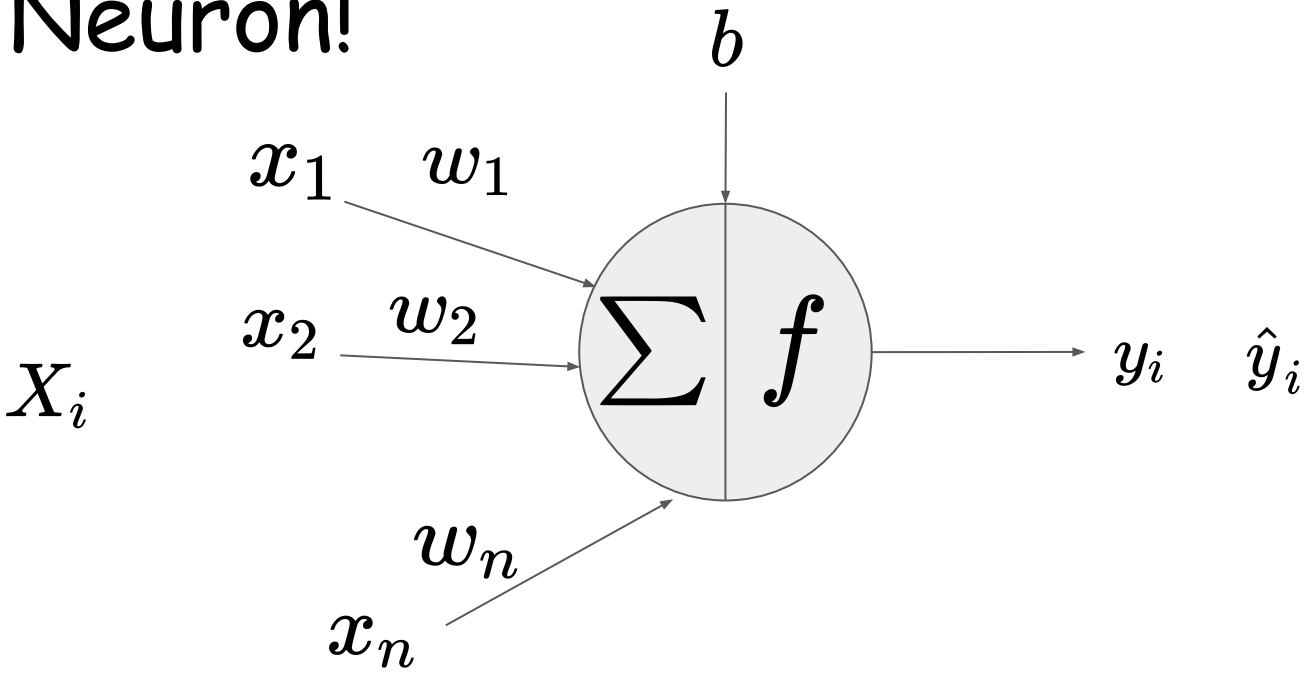


Gradient Descent in action

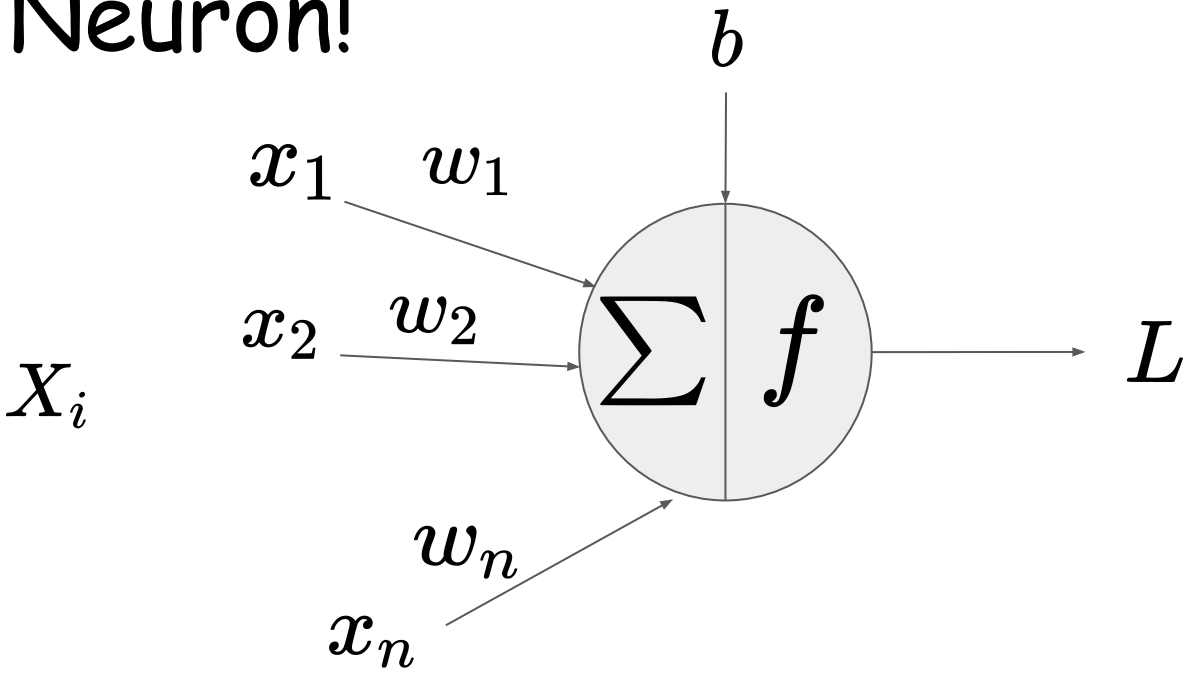


How to compute gradient?

Let us work on a simplest example, One Neuron!

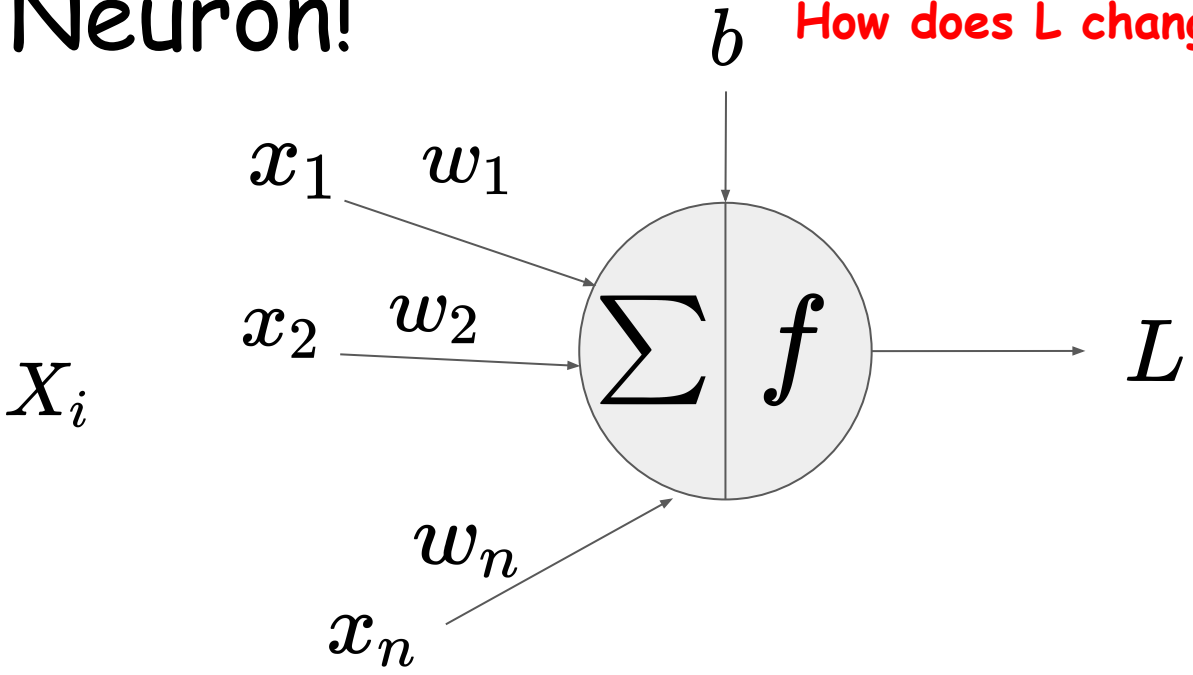


Let us work on a simplest example, One Neuron!

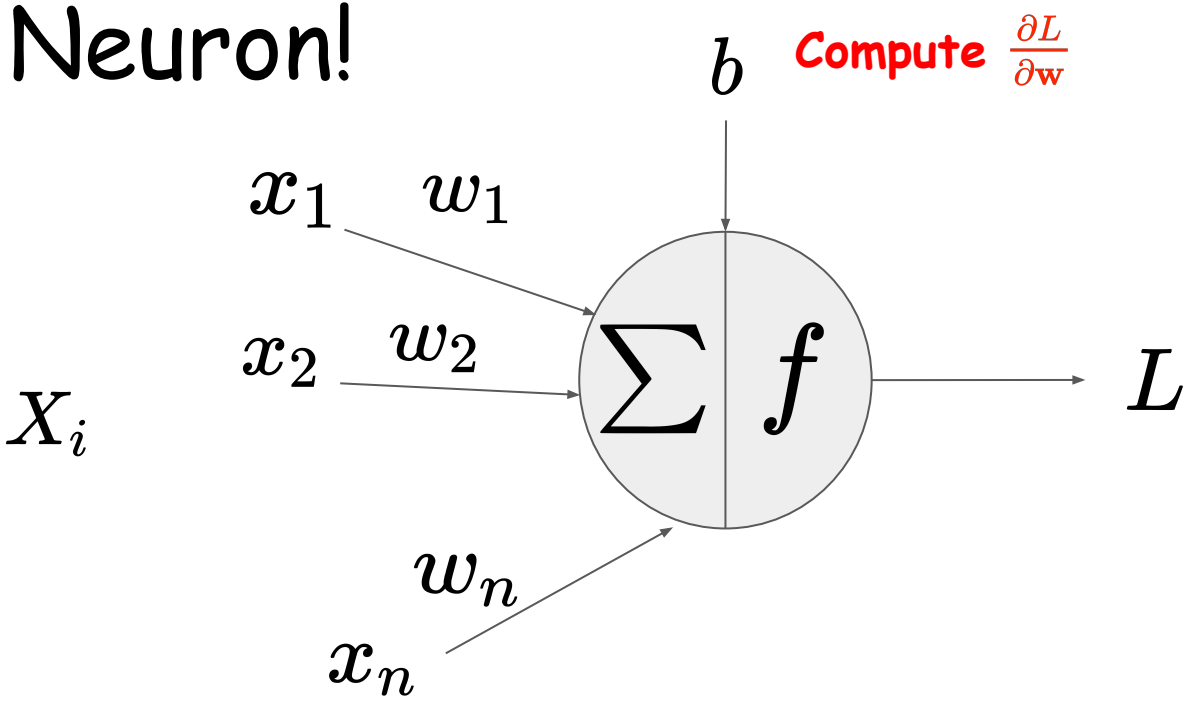


Let us work on a simplest example, One Neuron!

How does L changes if I change w a little?

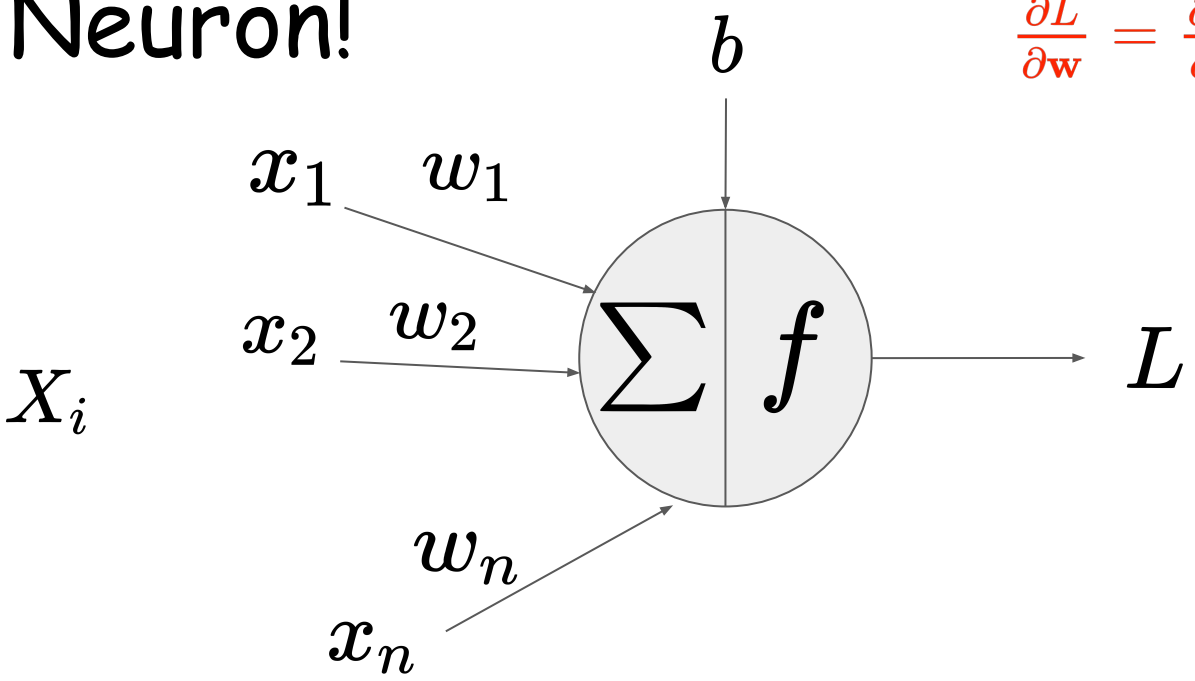


Let us work on a simplest example, One Neuron!



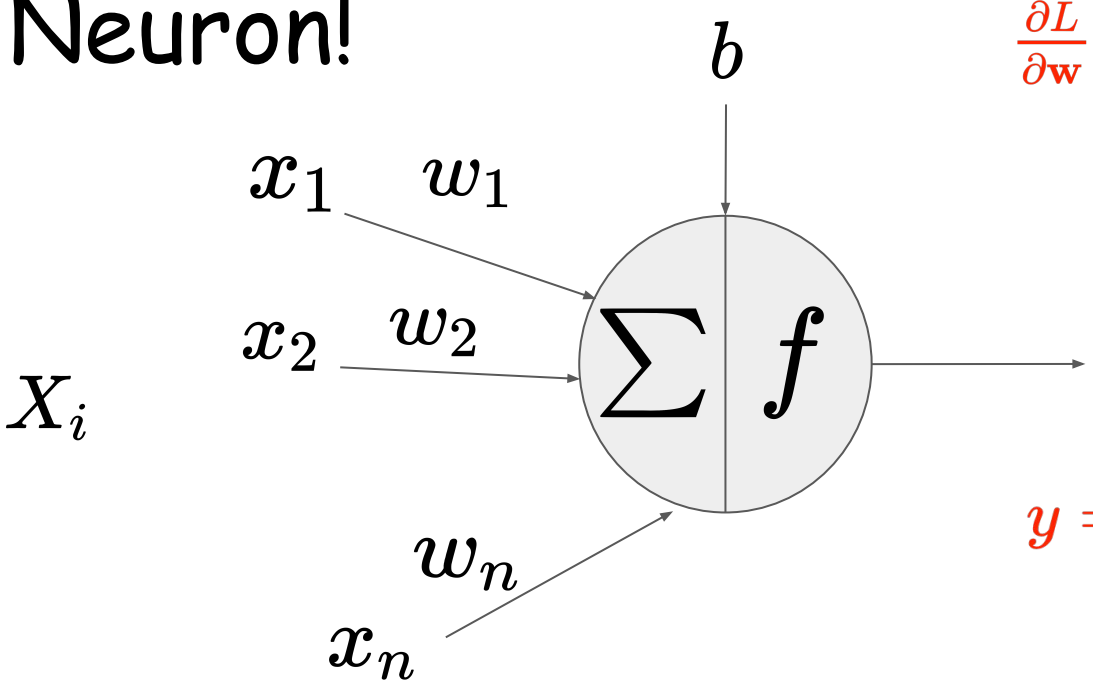
Let us work on a simplest example, One Neuron!

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{y}} \times \frac{\partial y}{\partial \mathbf{w}}$$



Let us work on a simplest example, One Neuron!

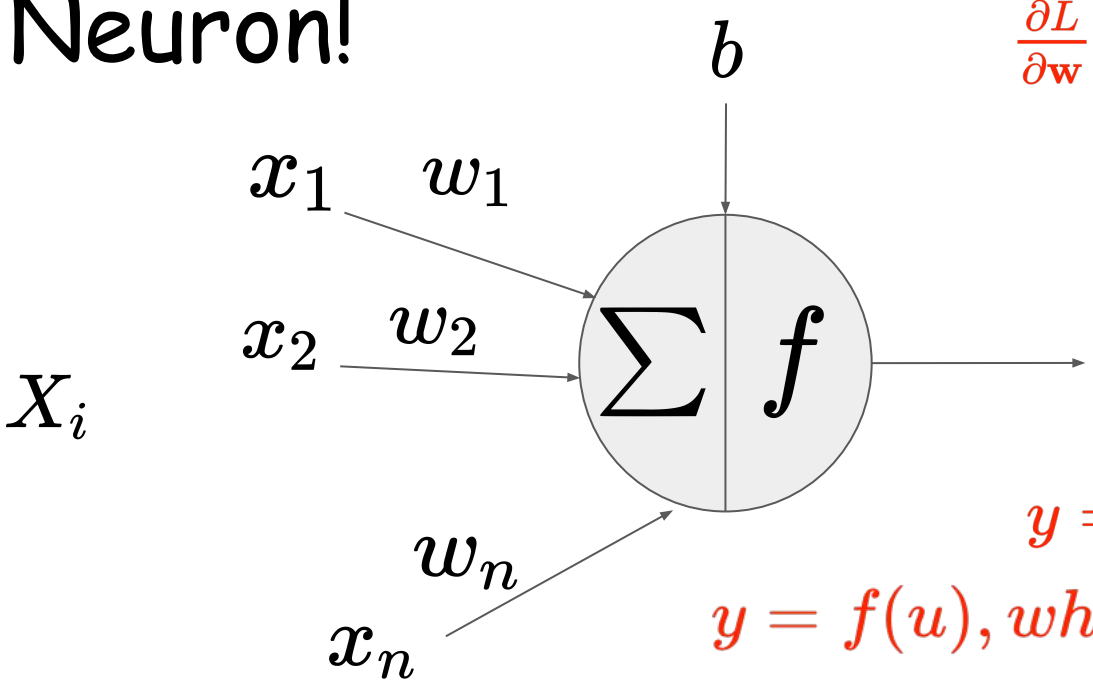
$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{y}} \times \frac{\partial y}{\partial \mathbf{w}}$$



$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

Let us work on a simplest example, One Neuron!

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{y}} \times \frac{\partial \mathbf{y}}{\partial \mathbf{w}}$$

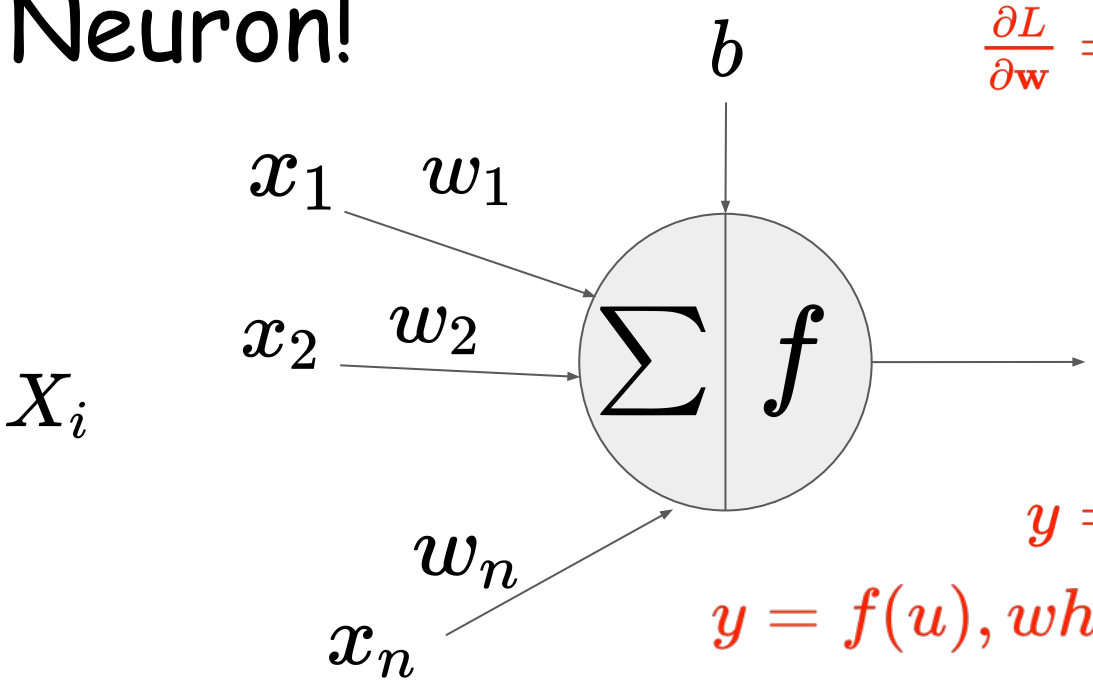


$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

$$y = f(u), where\ u = \sum_{i=1}^{n} w_i x_i + b$$

Let us work on a simplest example, One Neuron!

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{y}} \times \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \times \frac{\partial \mathbf{u}}{\partial \mathbf{w}}$$



$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

$$y = f(u), where\ u = \sum_{i=1}^{n} w_i x_i + b$$

So far ...

- MLP are universal approximators.
- Backpropagation is used to compute gradients

Implementation of Neural Network

https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

slido

Choose the statements that are TRUE?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Convolutional Neural Network (CNN)

How do we process images?

A

OR

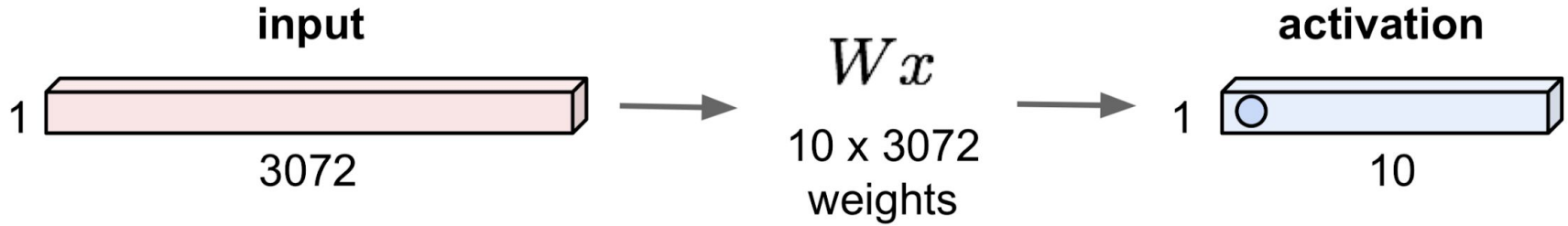
B

Slide credits:

http://cs231n.stanford.edu/syllabus.html

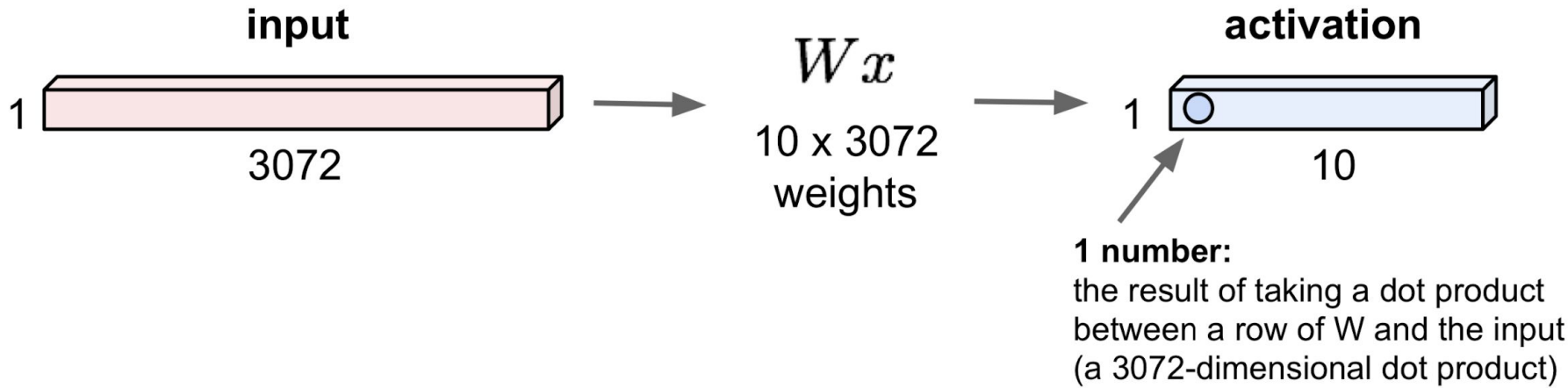
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



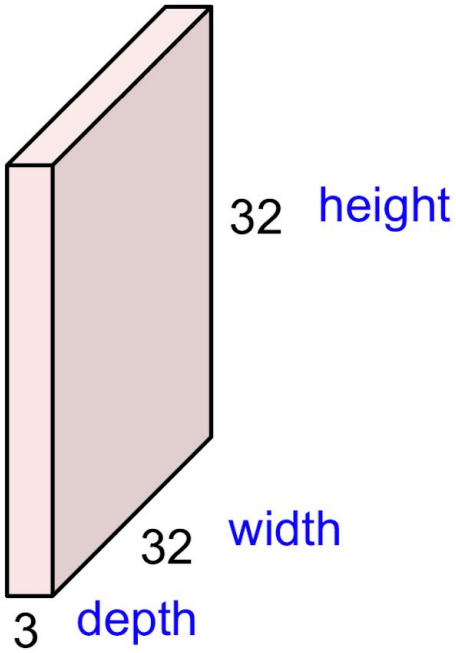
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



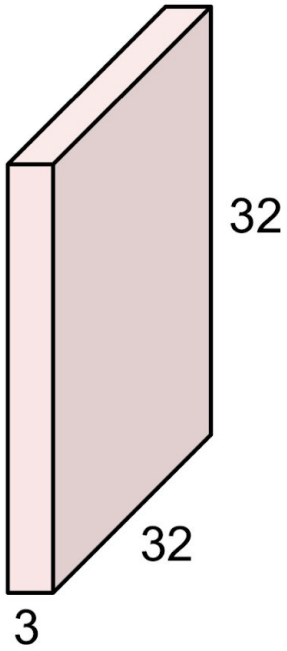
Convolution Layer

32x32x3 image -> preserve spatial structure

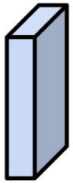


Convolution Layer

32x32x3 image



5x5x3 filter

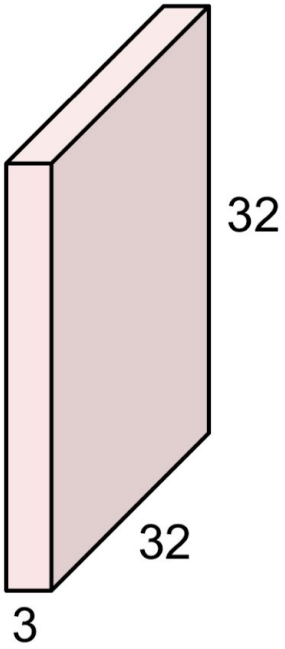


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

Filters always extend the full depth of the input volume

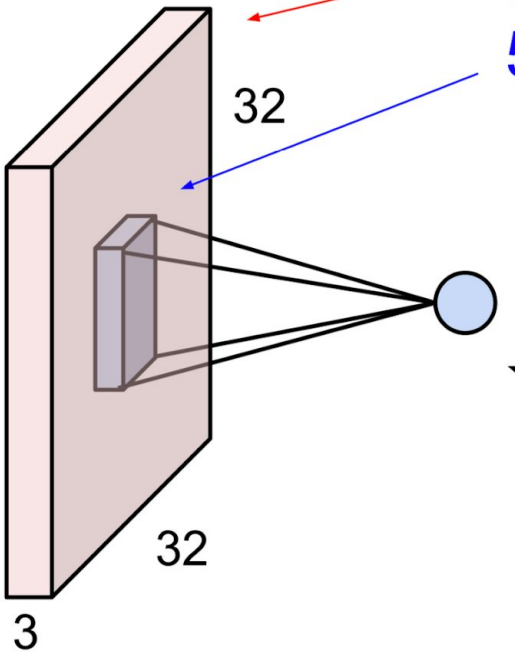
32x32x3 image



5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



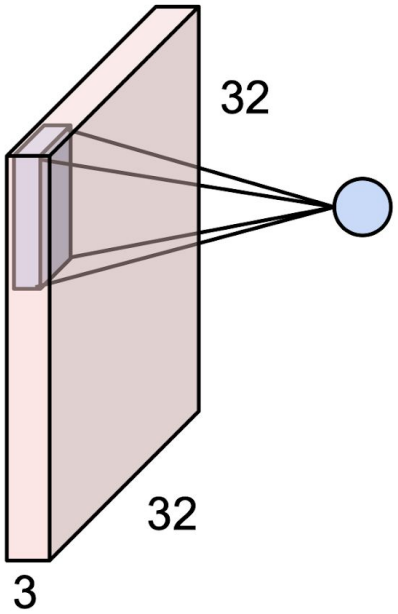
32x32x3 image

5x5x3 filter w

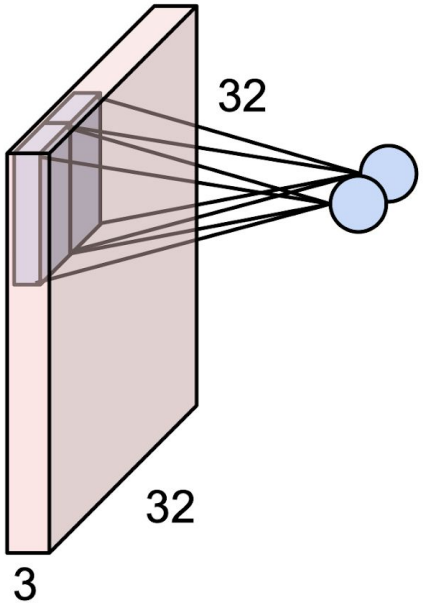
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

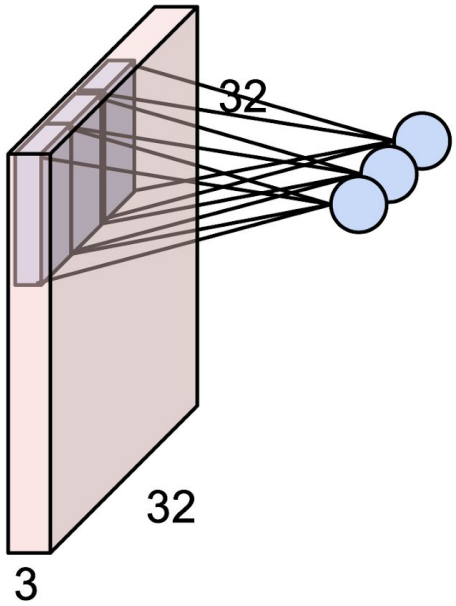
Convolution Layer



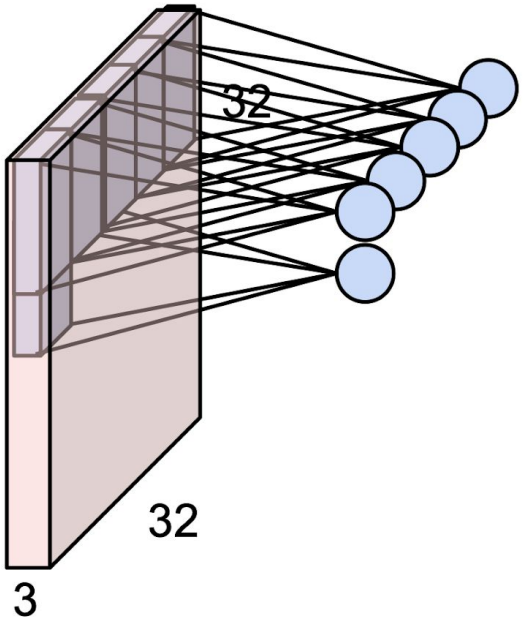
Convolution Layer



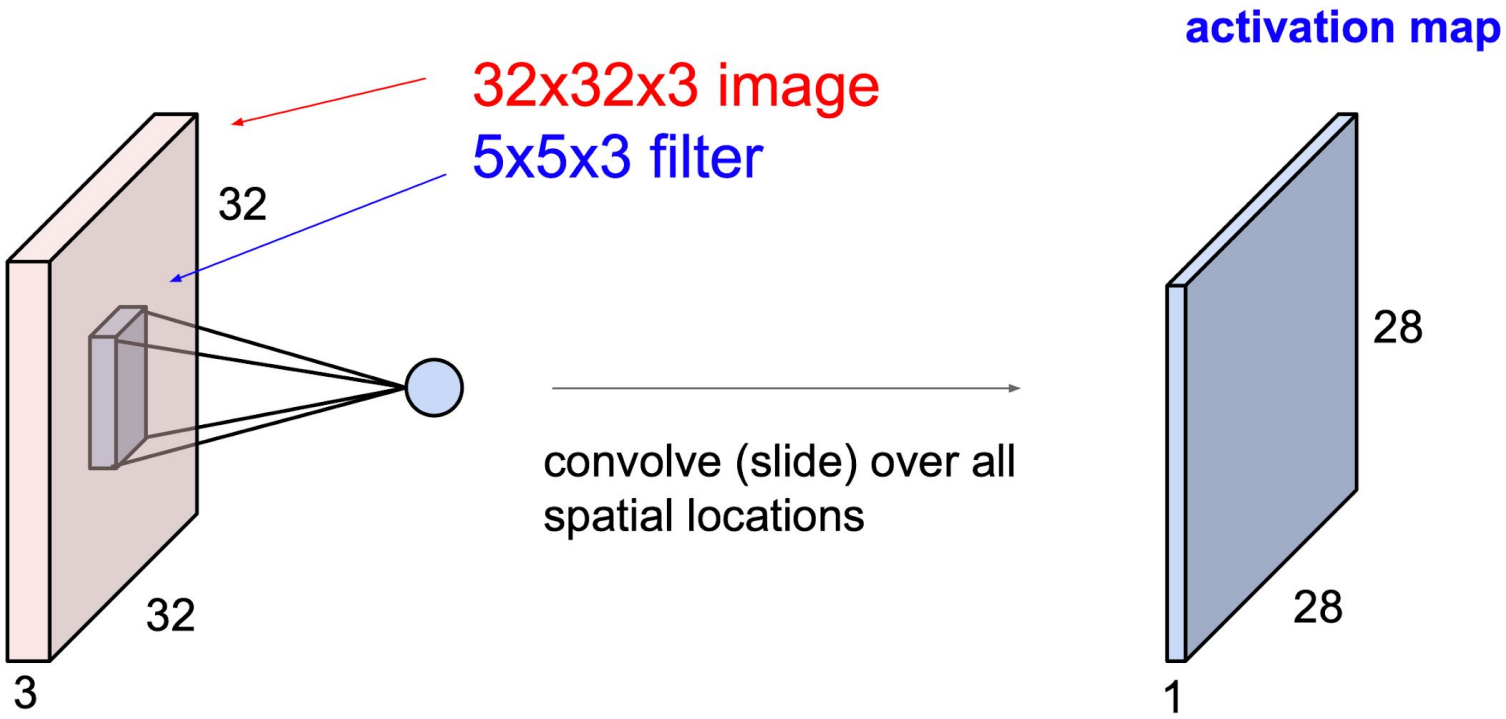
Convolution Layer



Convolution Layer

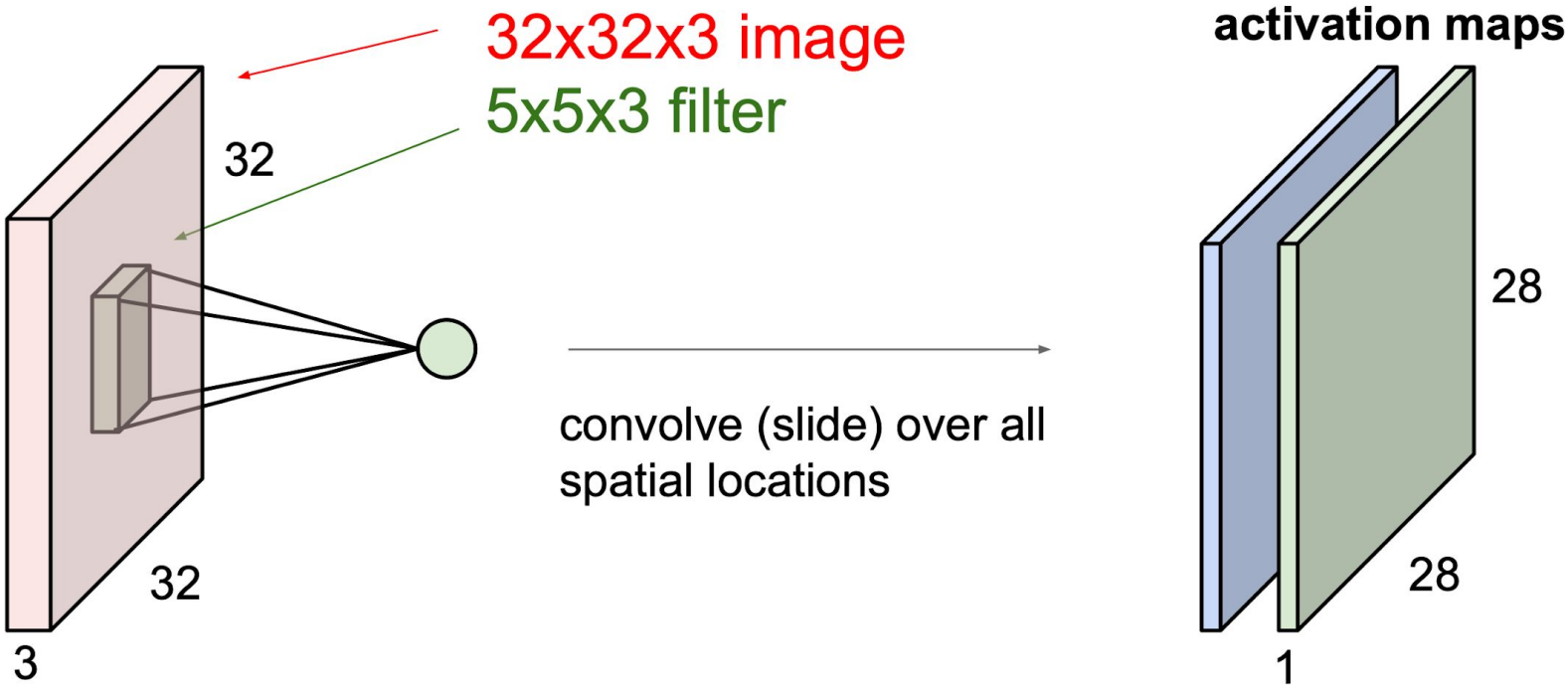


Convolution Layer

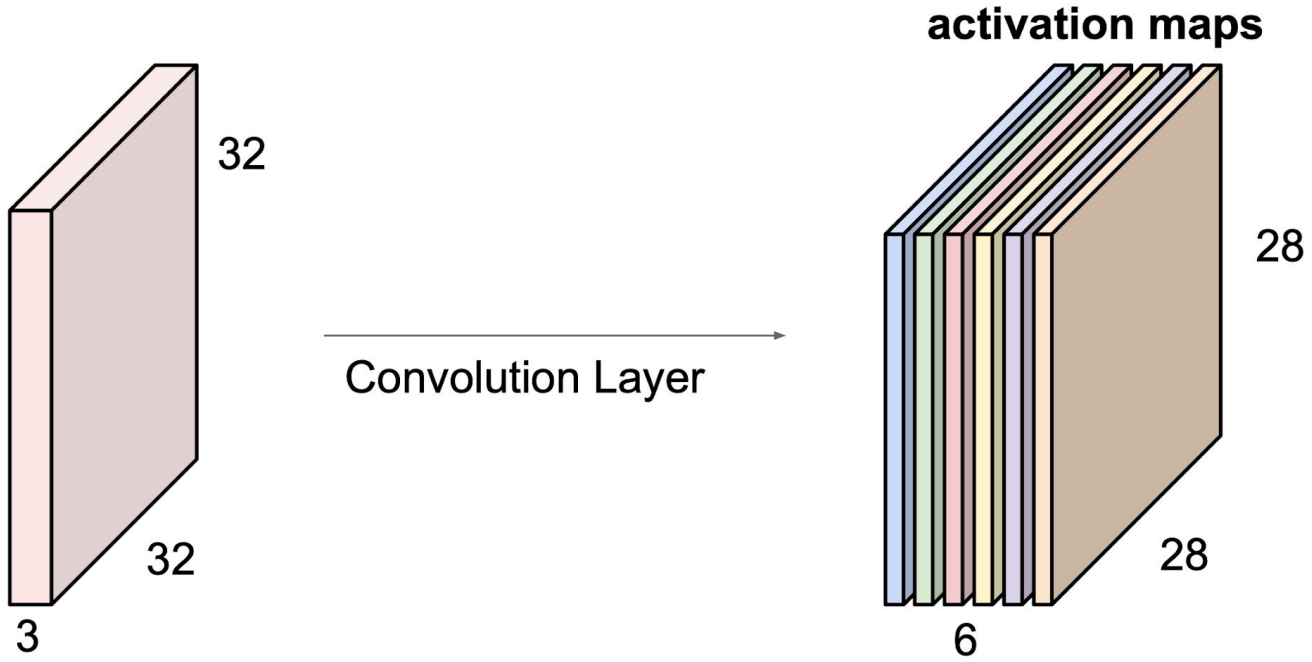


Convolution Layer

consider a second, green filter

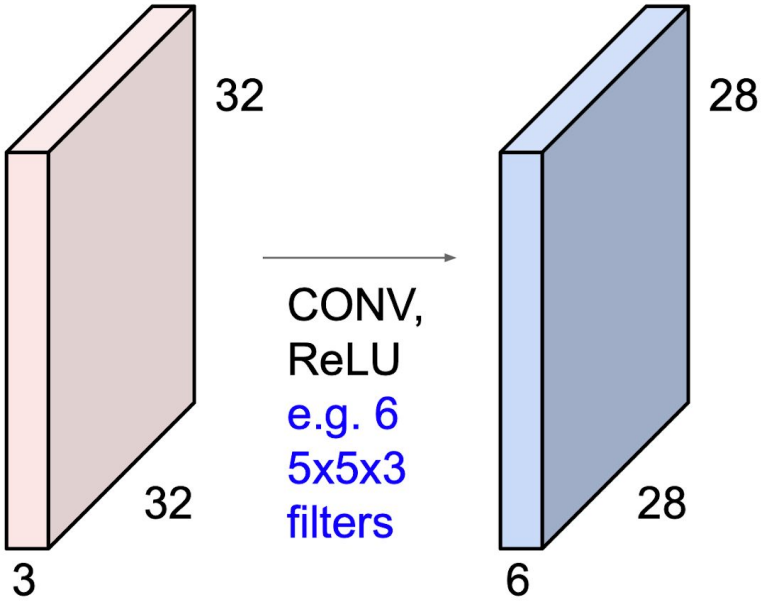


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

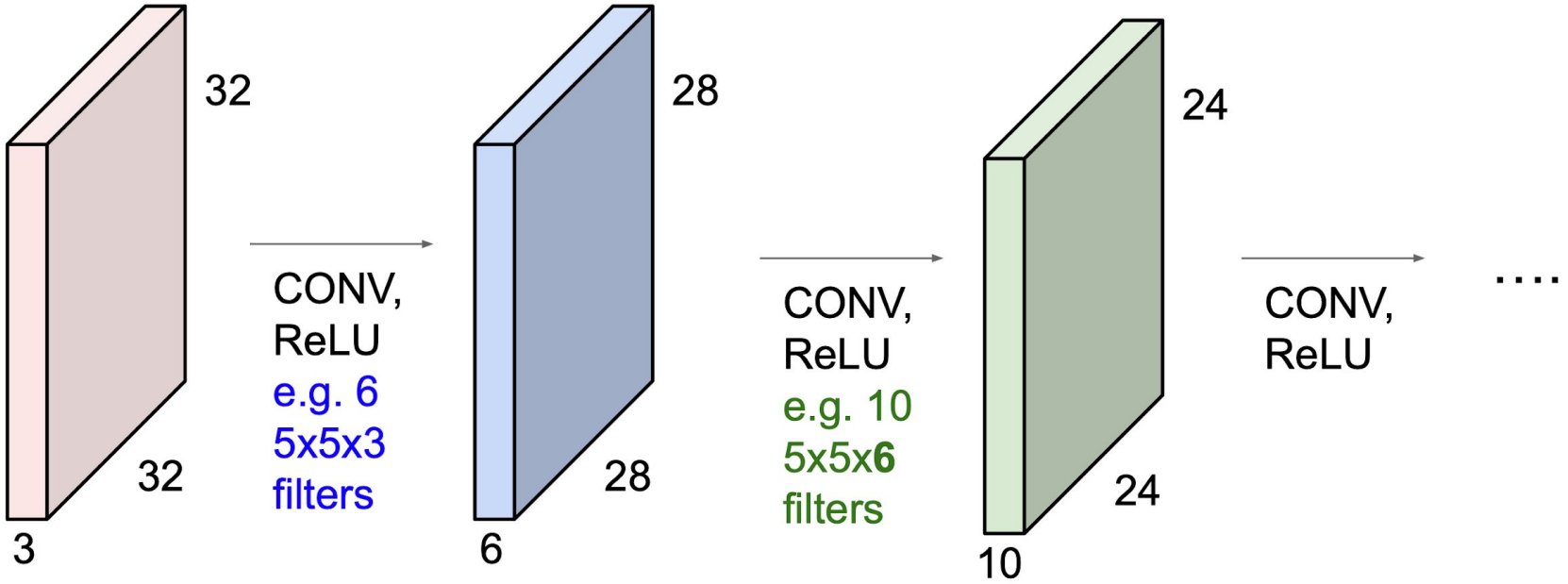


We stack these up to get a "new image" of size 28x28x6!

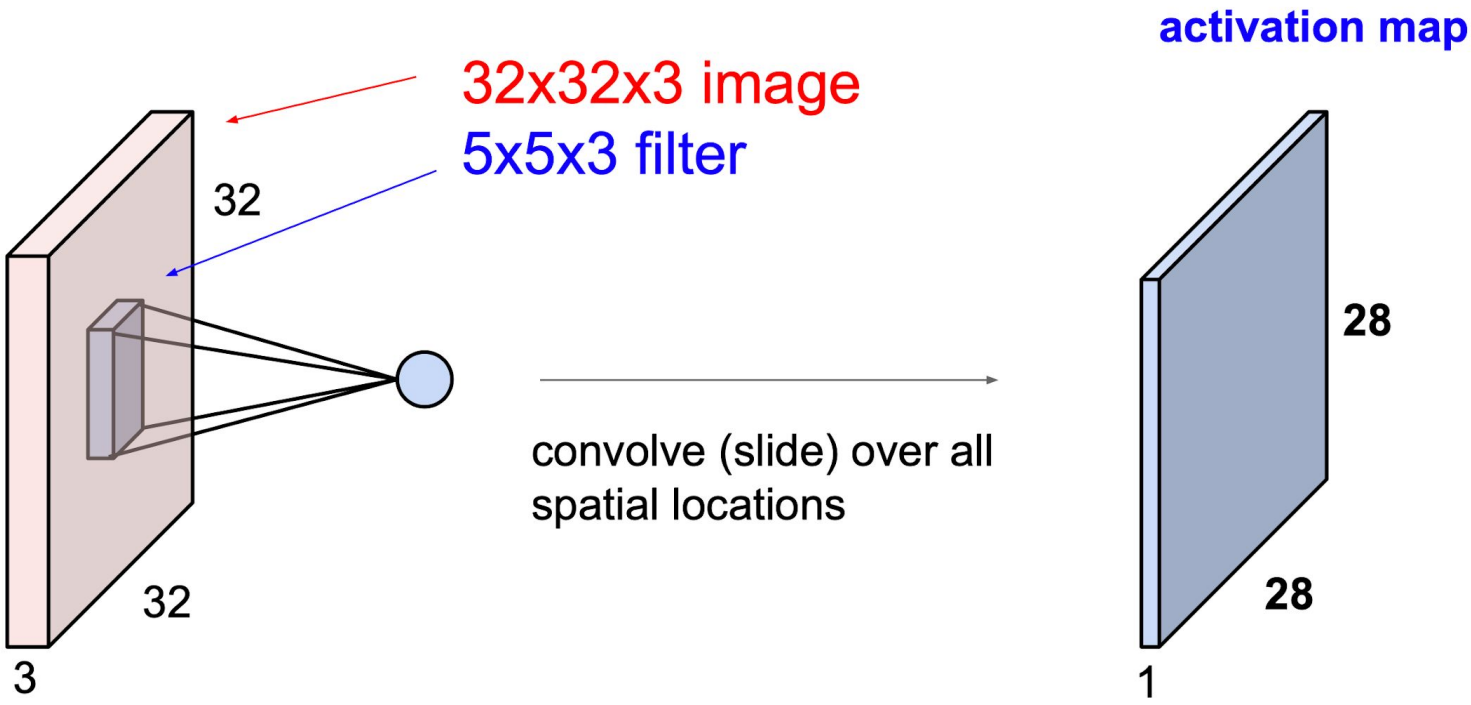
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



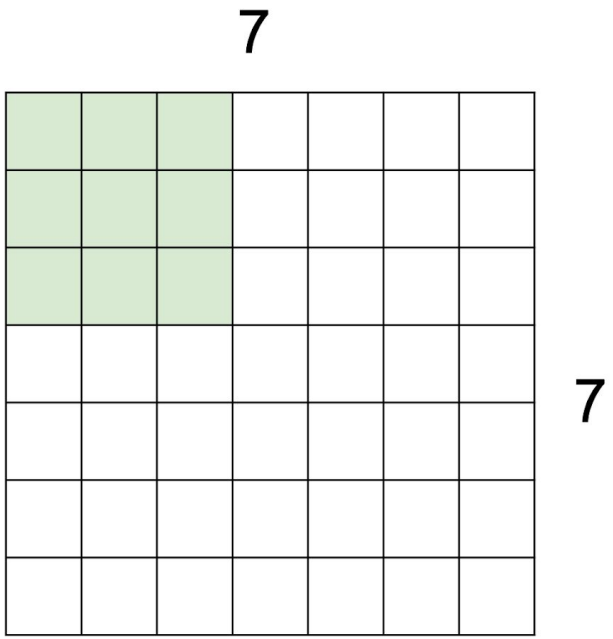
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



A closer look at spatial dimensions:

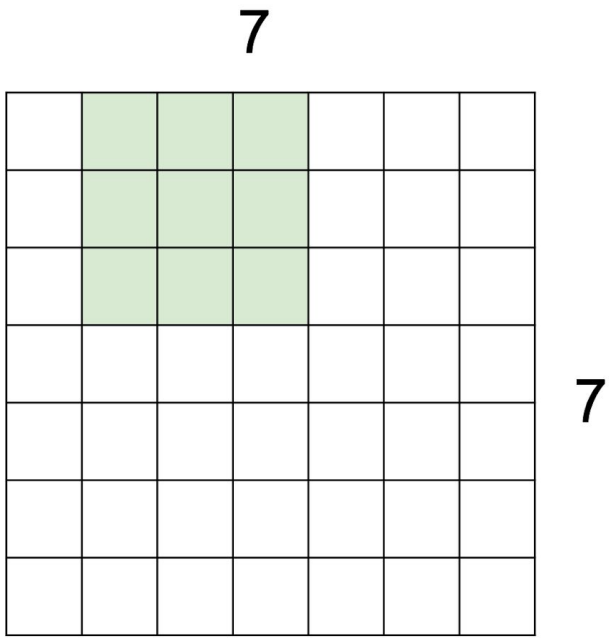


A closer look at spatial dimensions:



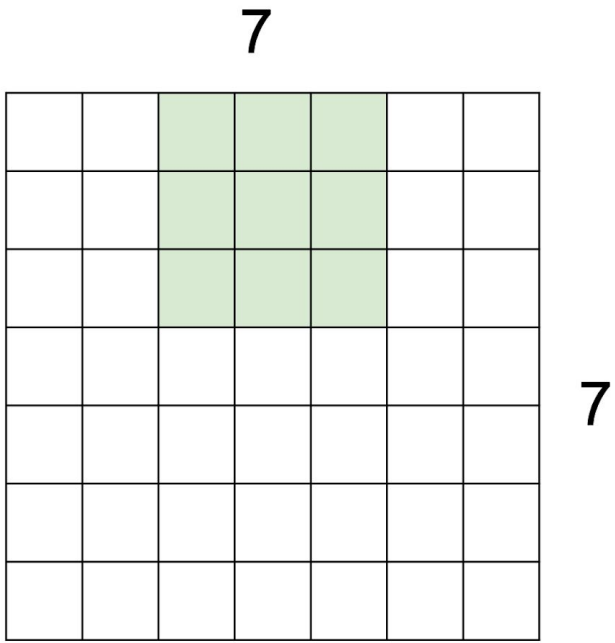
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



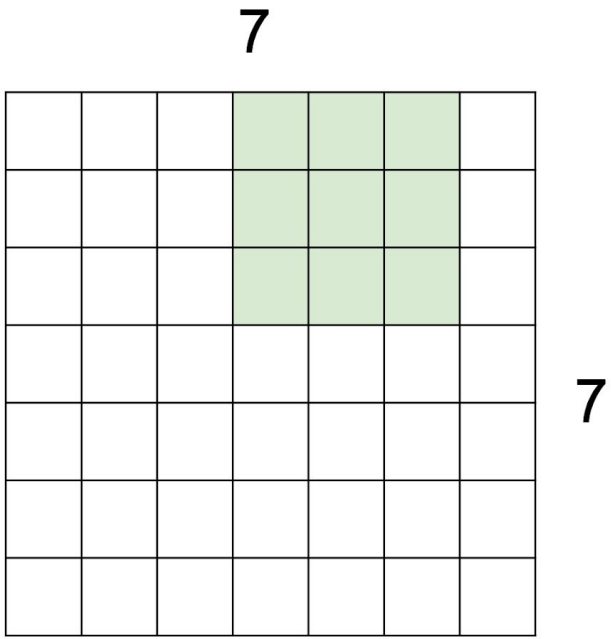
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



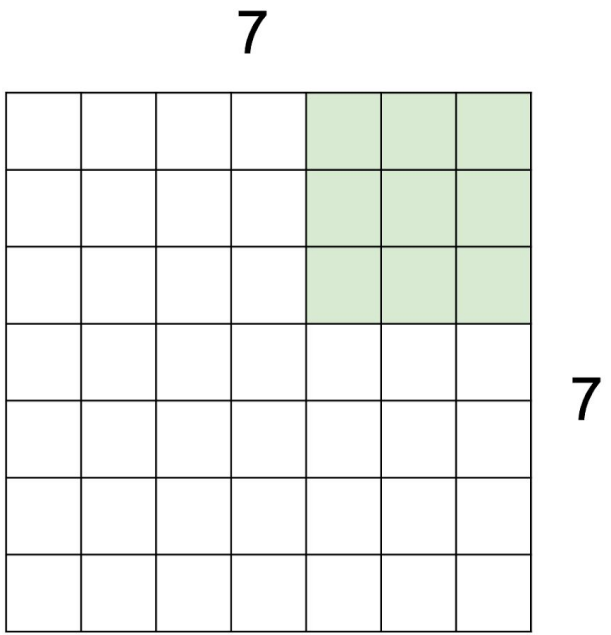
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

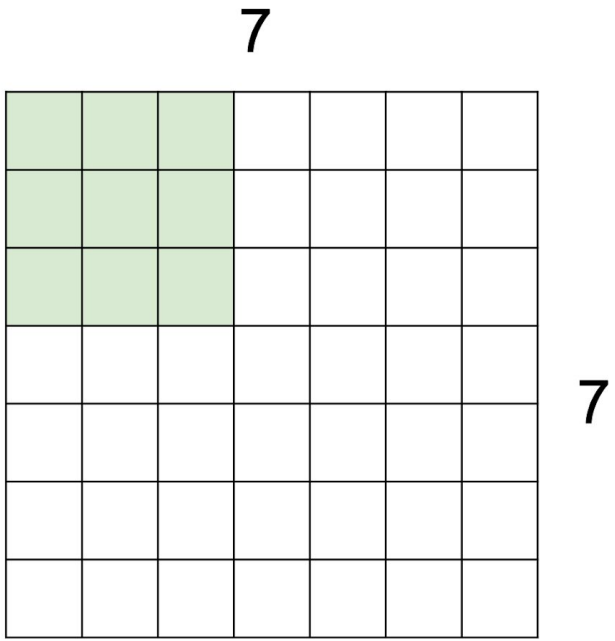
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

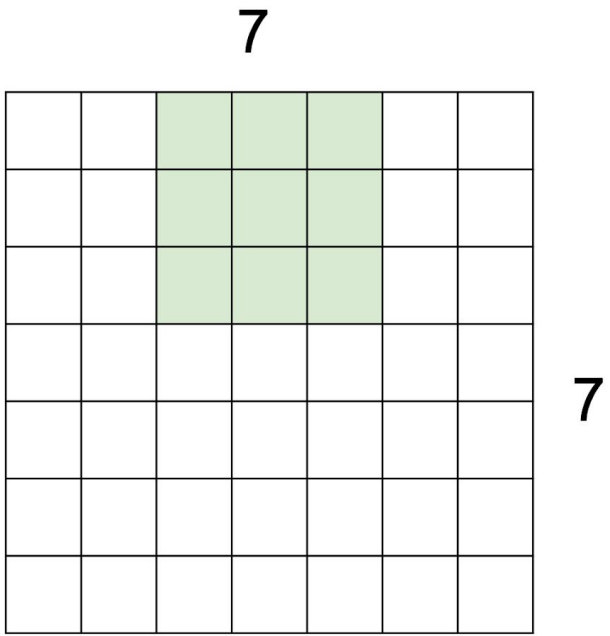
=> 5x5 output

A closer look at spatial dimensions:



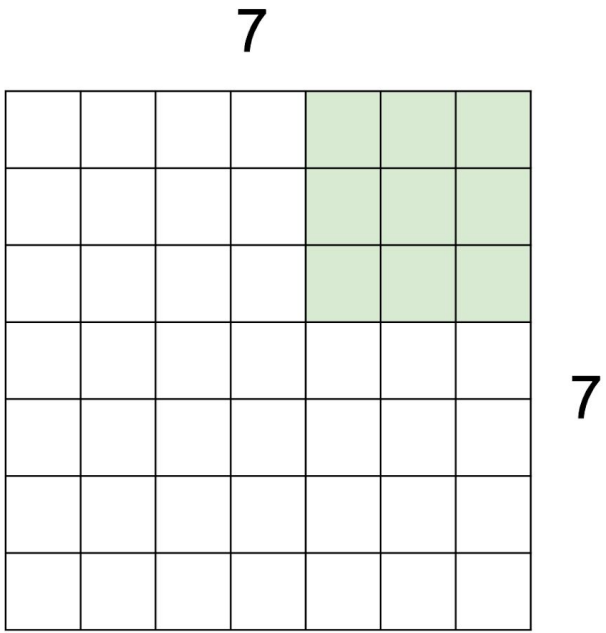
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



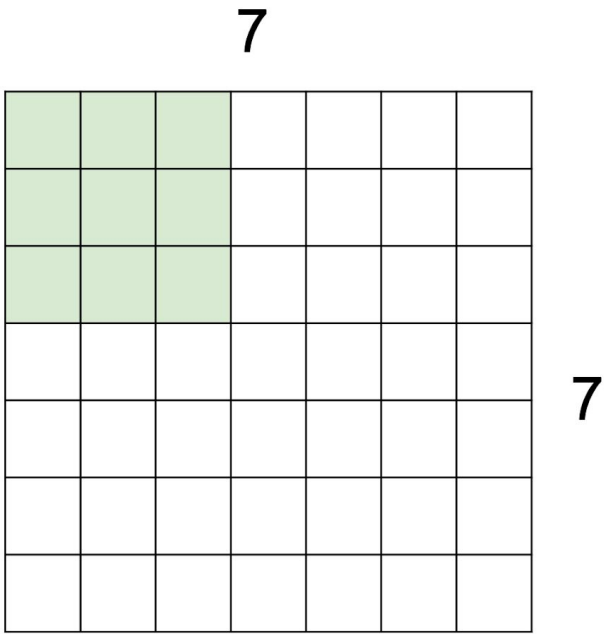
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



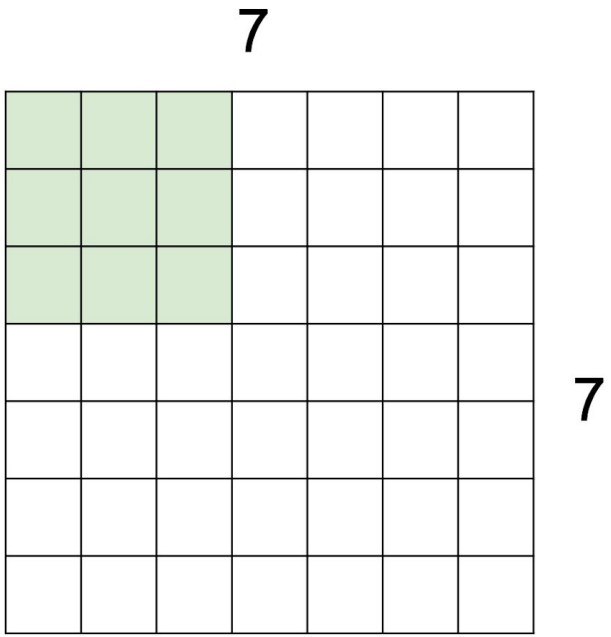
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



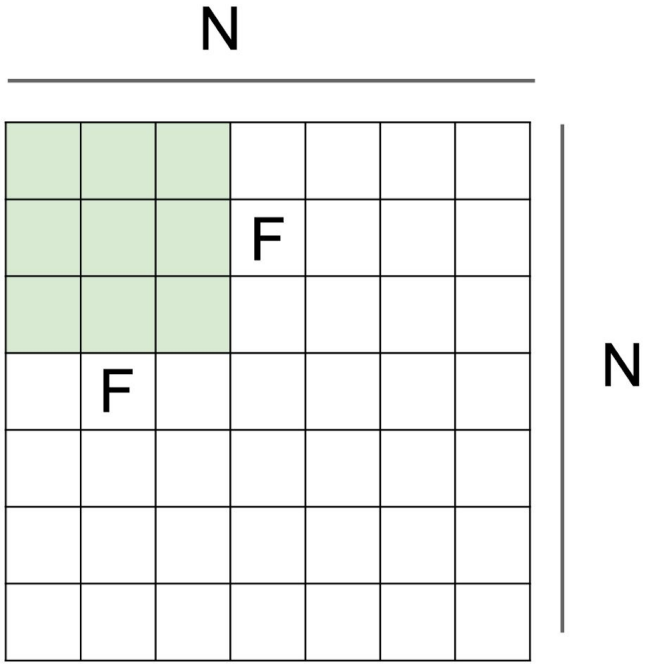
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

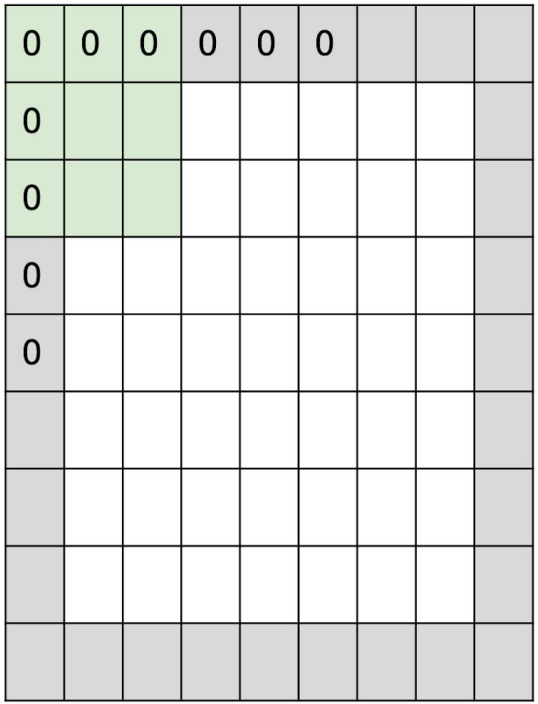
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

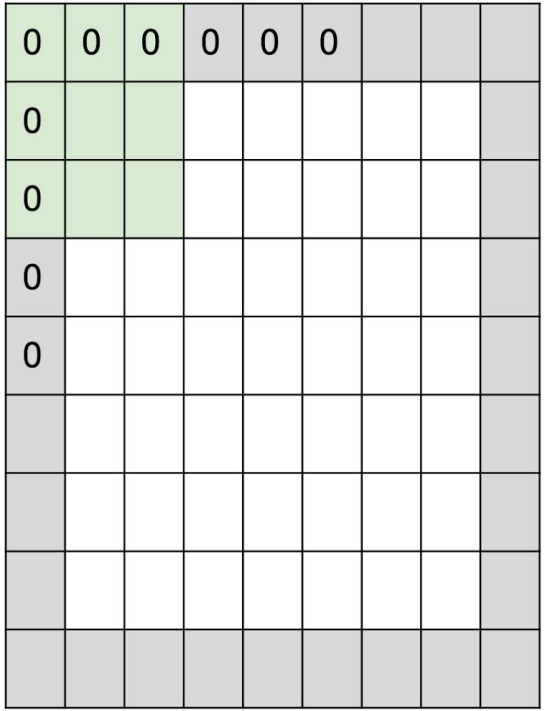
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

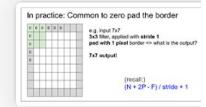
(recall:)
(N - F) / stride + 1

In practice: Common to zero pad the border

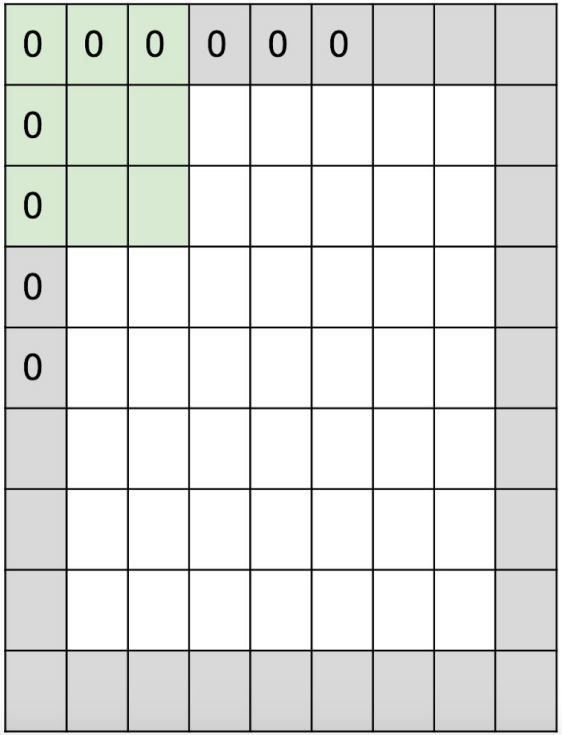


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3



In practice: Common to zero pad the border

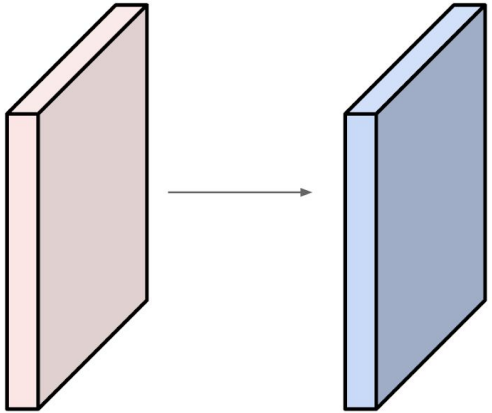


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

Examples time:

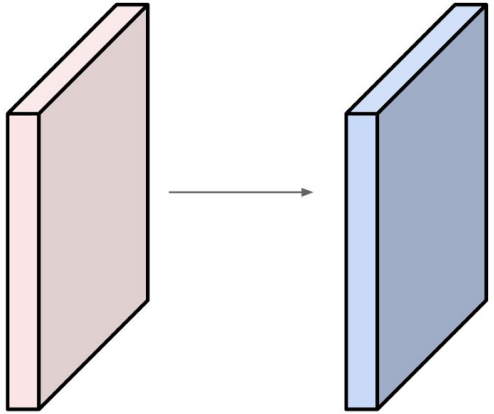


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

$$(N + 2P - F) / \text{stride} + 1$$

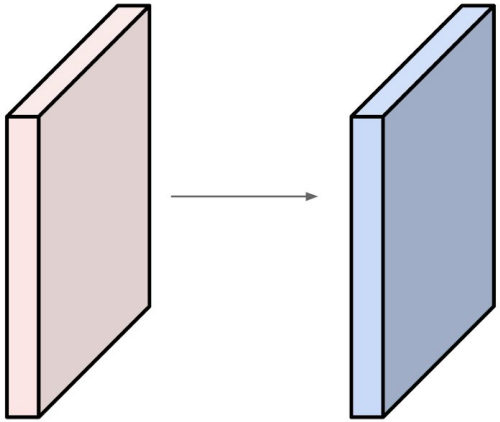
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

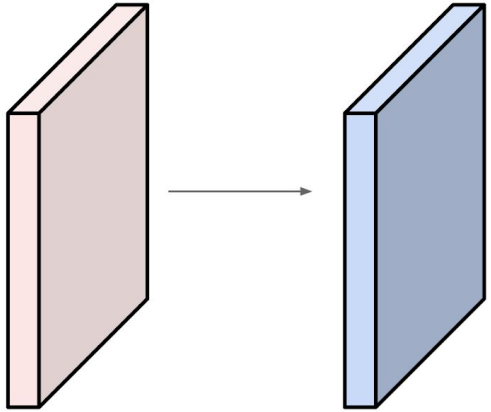
Number of parameters in this layer?

slido

Number parameters in this layer?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

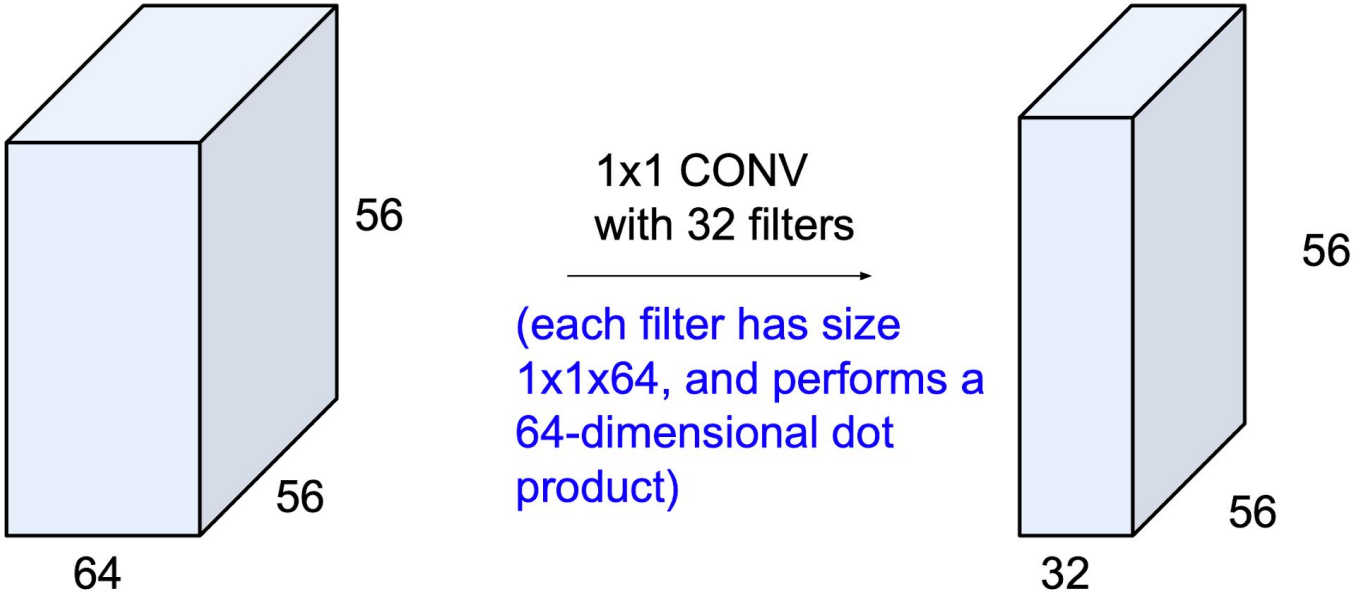
Examples time:



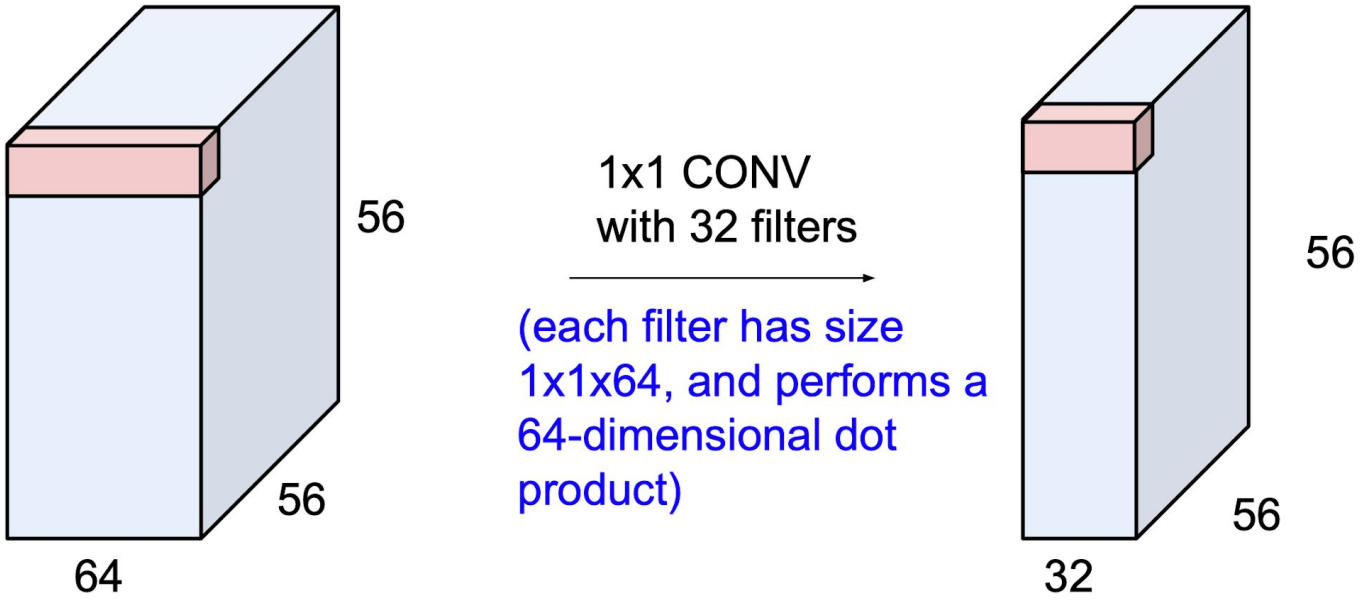
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

(btw, 1x1 convolution layers make perfect sense)

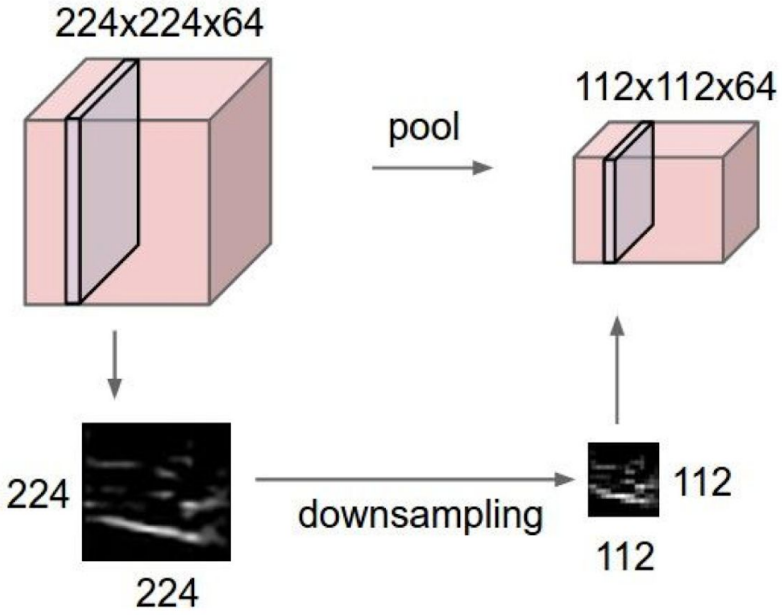


(btw, 1x1 convolution layers make perfect sense)

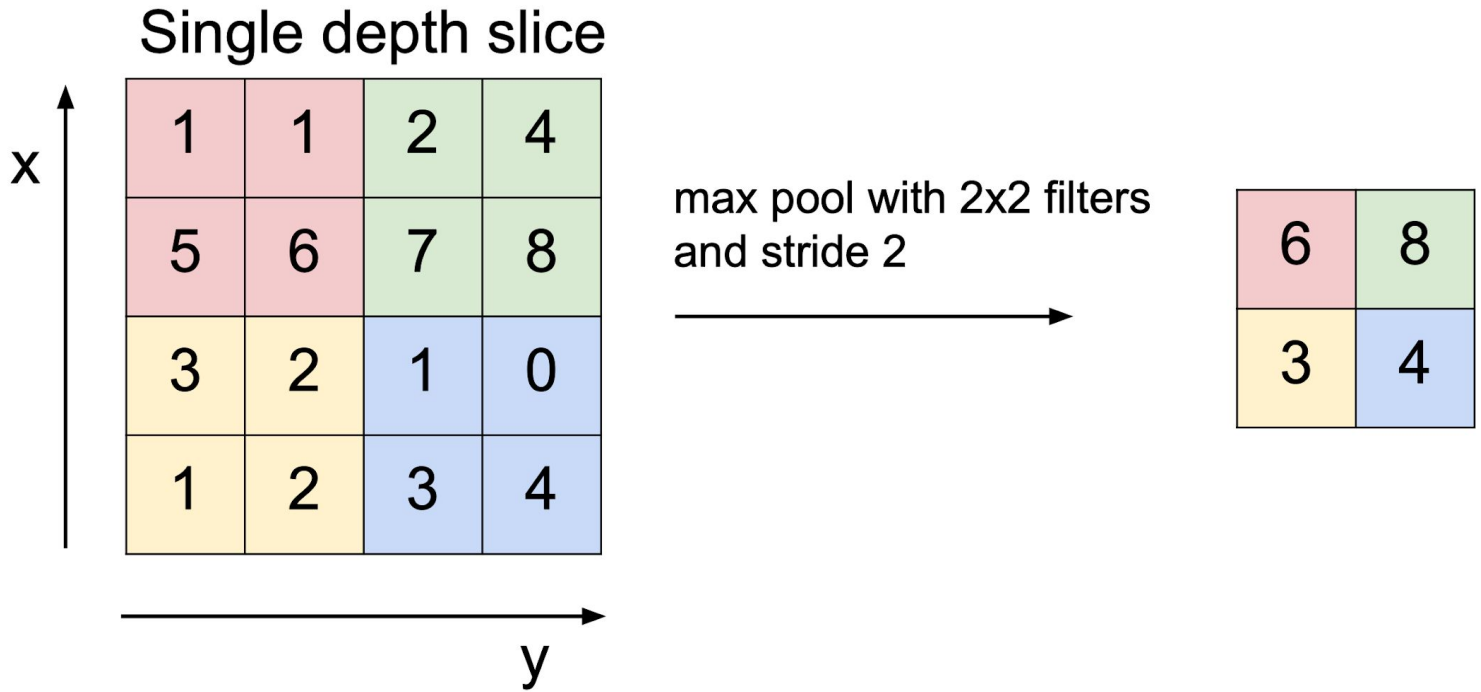


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

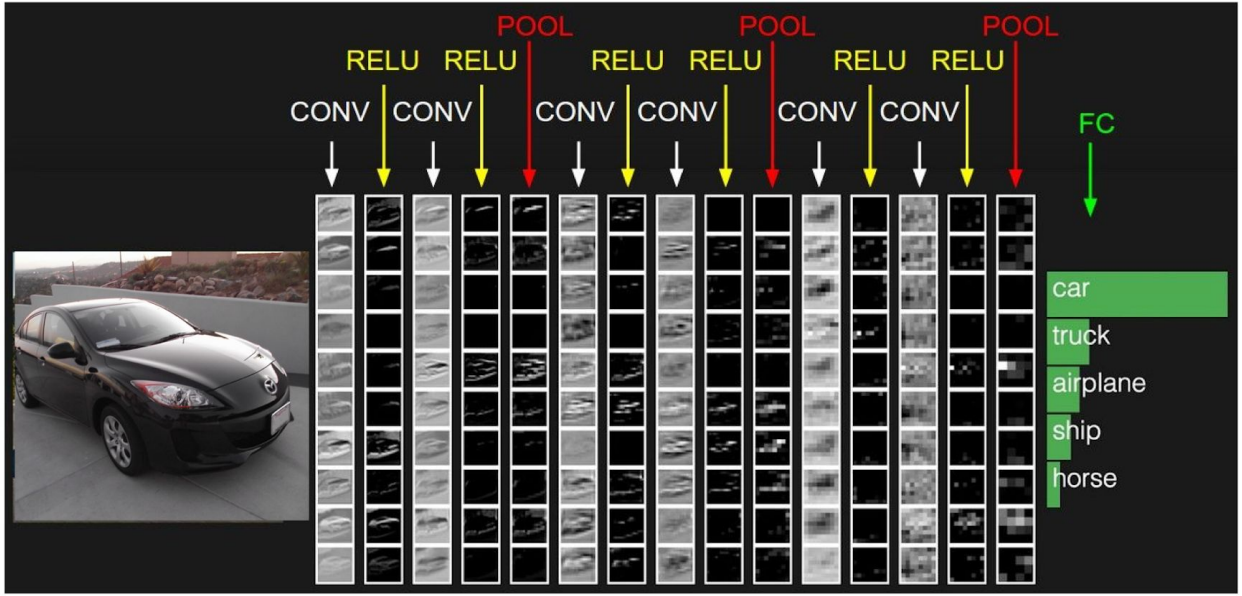


MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Different Popular CNN architectures

1. LeNet
2. Alex-net
3. VGG-net
4. Resnet
5. InceptionNet …

Implementation

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html