

# Machine Learning for Networks: Neural Networks

**Andrea Araldo**

*December 8, 2020*



	ML task	Linear Regression	Logistic Regression	Tree-based learning	Neural Networks	$k$ -Nearest Neighbors
Supervised	Regression Classification	x	x	x	x	
Unsupervised	Clustering Dimensionality reduction Anomaly detection			x	x	x
	Recommender Systems			x		

- Structure of NNs
- Training (backpropagation)
- Design choices and hyper-parameters

## Section 1

### **Introduction**

BULLETIN OF  
MATHEMATICAL BIOPHYSICS  
VOLUME 5, 1943

## A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

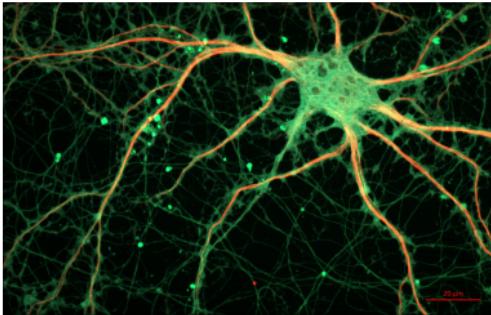
FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,  
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,  
AND THE UNIVERSITY OF CHICAGO

18K citations!

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.

# Neural Network - Human brain

5 / 60



- By ZEISS Microscopy from Germany (Cultured Rat Hippocampal Neuron) [CC BY 2.0 (<http://creativecommons.org/licenses/by/2.0>)], via Wikimedia Commons

- <https://pixabay.com/en/neurons-brain-cells-brain-structure-1739997/>



- *Walter Pitts*: logician
- *Warren McCulloch*: neurophysiologist

CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>), via Wikimedia Commons



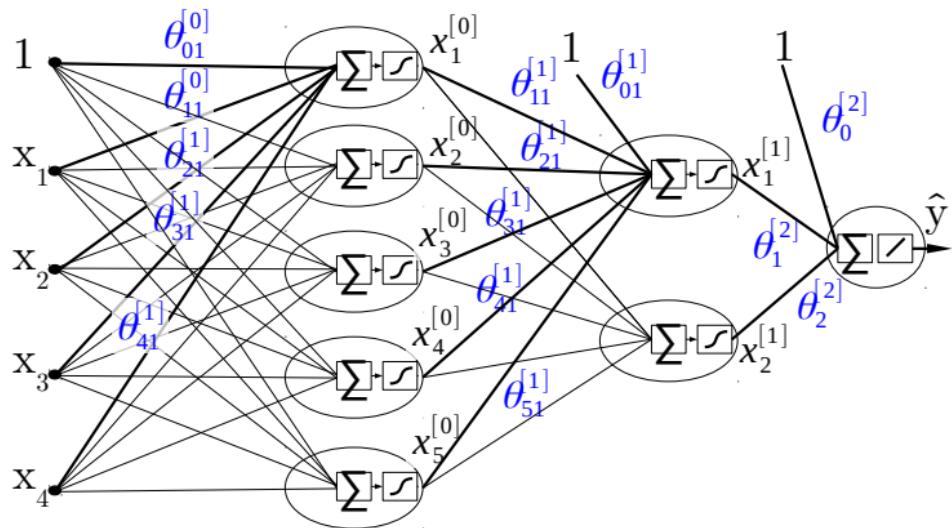
Walter Pitts:

- When he was 12, he criticized Principia Mathematica from Bertrand Russel.
- Russel invited him to Cambridge University and Pitts refused.

source: Wikipedia

# Neural Network - Multi-Layer Perceptron (MLP)

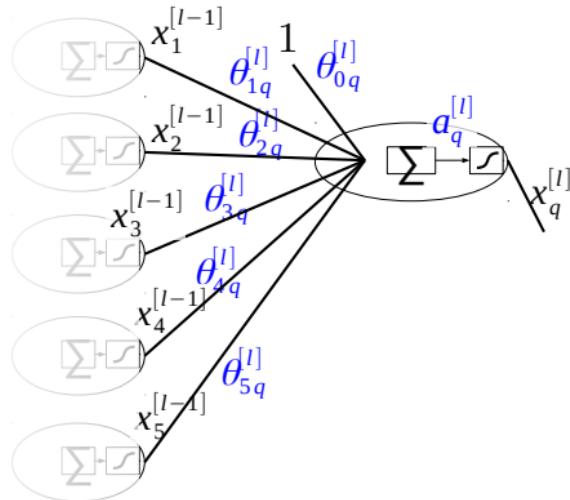
7 / 60



# Neural Network - Single neuron

8 / 60

Let us look at the  $q$ -th neuron in the  $l$ -layer.  
Do you recognize it?



- Output from the previous layer:

$$\mathbf{x}^{[l-1]} = (1, x_1^{[l-1]}, x_2^{[l-1]}, \dots)$$

- Weights:

$$\boldsymbol{\theta}_q^{[l]} = (\theta_{0q}^{[l]}, \theta_{1q}^{[l]}, \dots)$$

- Weighted input

$$a_q^{[l]} = \boldsymbol{\theta}_q^{[l]T} \cdot \mathbf{x}^{[l-1]}$$

- Activation function  $\sigma(\cdot)$

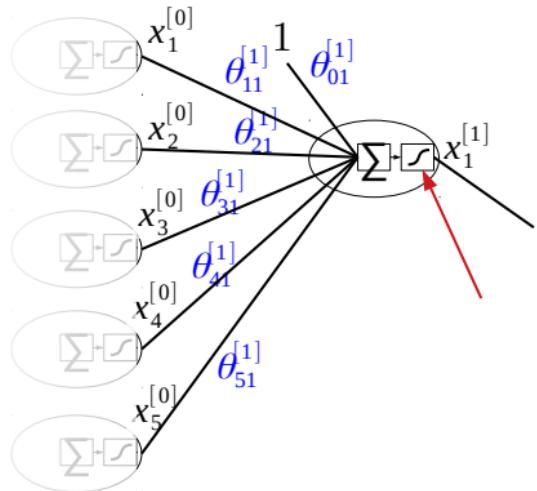
- Output:

$$x_q^{[l]} = \sigma(a_q^{[l]})$$

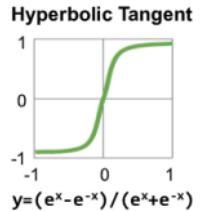
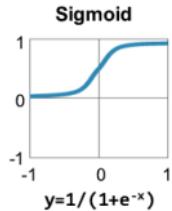
This can be fed to further neurons.

# Activation Functions

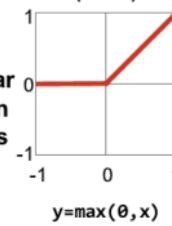
9 / 60



Traditional  
Non-Linear  
Activation  
Functions

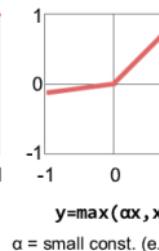


Rectified Linear Unit  
(ReLU)

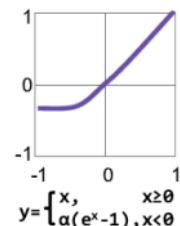


Modern  
Non-Linear  
Activation  
Functions

Leaky ReLU



Exponential LU

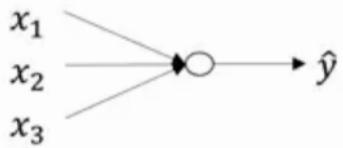


$\alpha$  = small const. (e.g. 0.1)

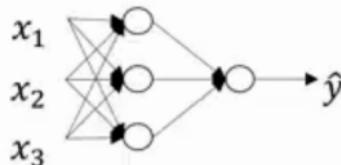
Figure from [SCYE17].

# Depth of a NN

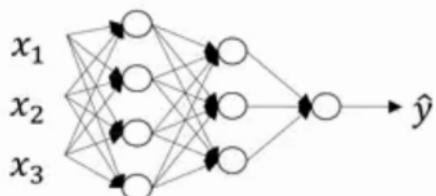
10 / 60



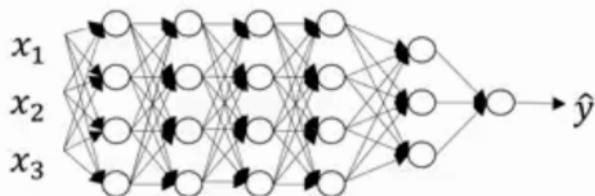
logistic regression



1 hidden layer



2 hidden layers



5 hidden layers

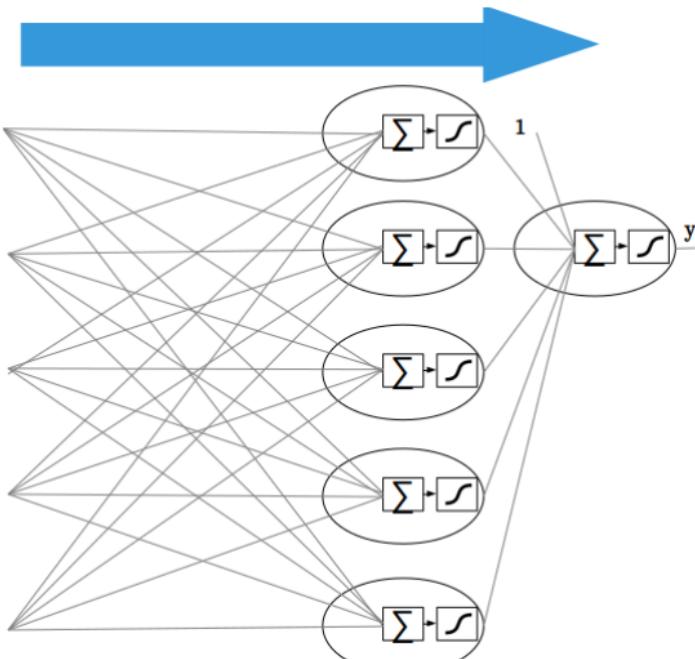
Andrew Ng

Source: Andrew Ng, Deep Neural Networks

Deep NN: NN with many hidden layers.

# Prediction with neural networks

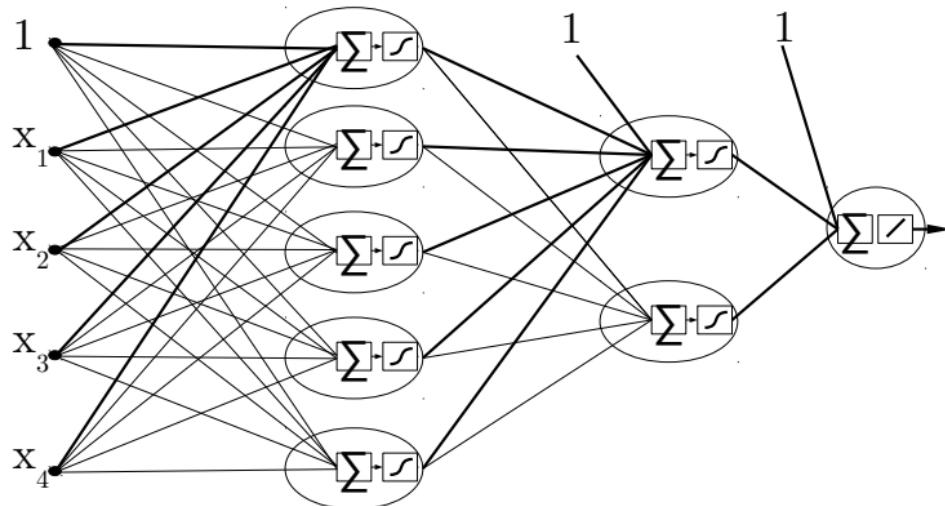
11 / 60



Information is processed from left to right (forward propagation)

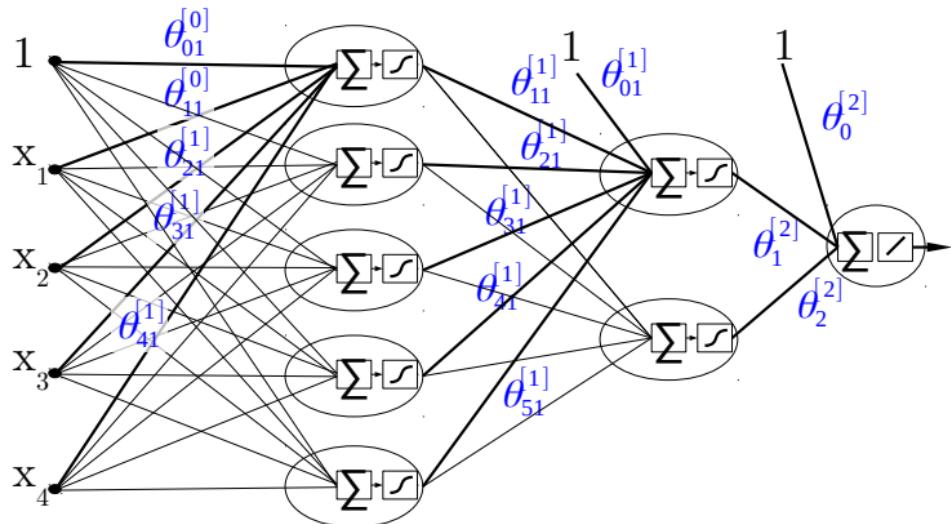
# NN for regression

12 / 60



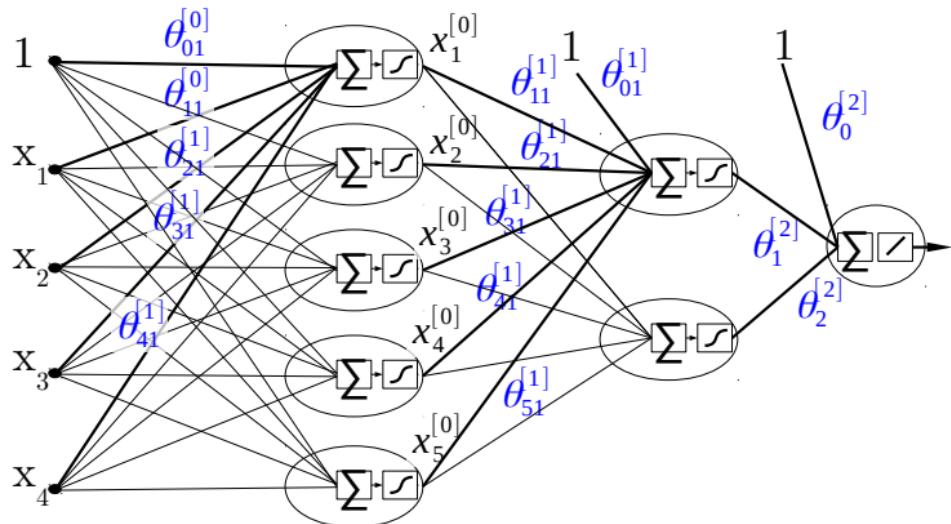
# NN for regression

12 / 60



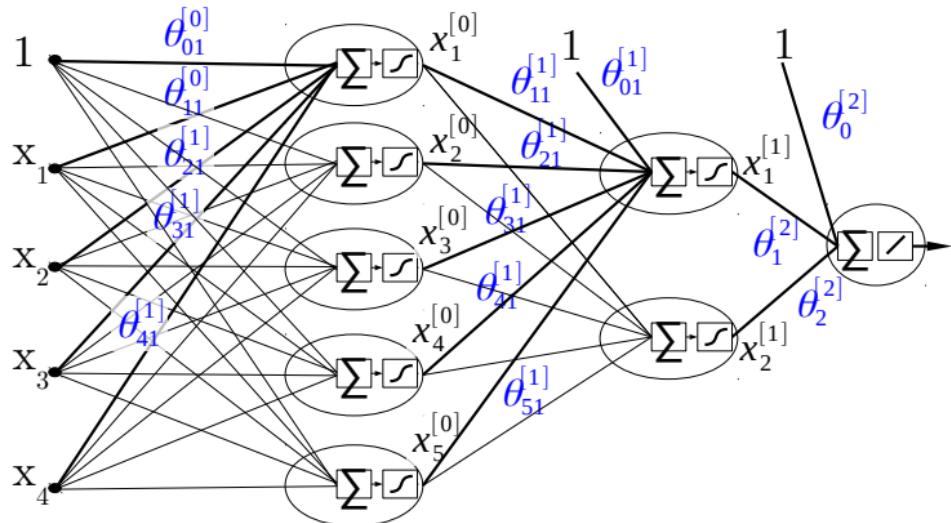
# NN for regression

12 / 60



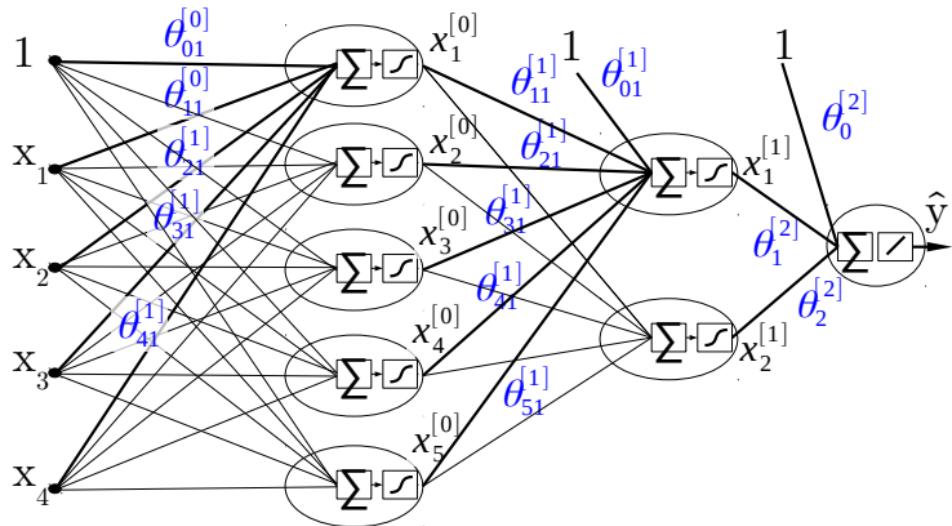
# NN for regression

12 / 60



# NN for regression

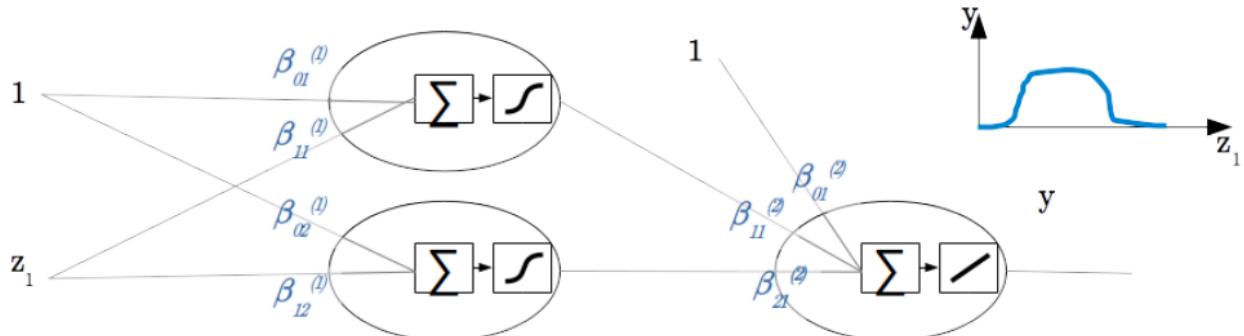
12 / 60



$$\hat{y} = h_{\theta}(\mathbf{x})$$

# Universal Approximator

13 / 60

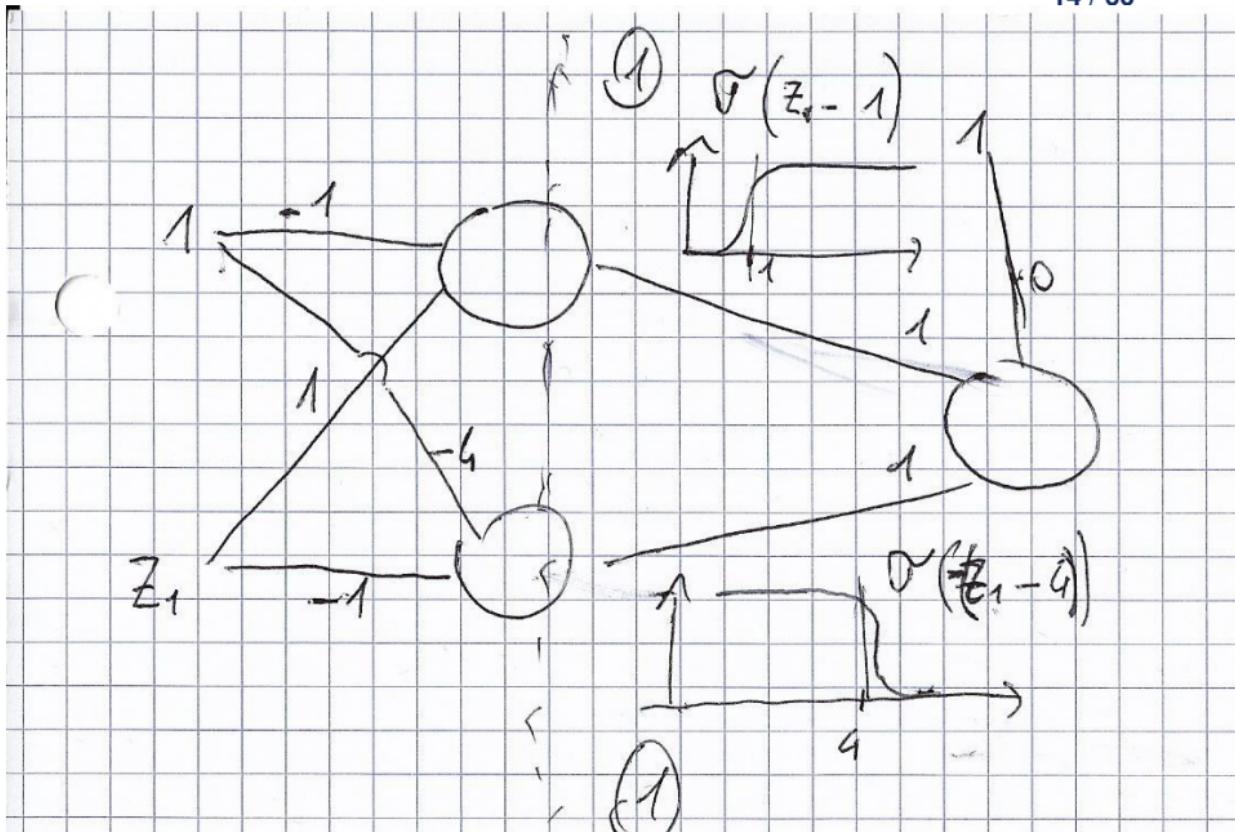


A single hidden layer neural network is a *universal approximator*: any continuous function can be approximated to arbitrary accuracy, provided that there are enough neurons.

Let's write the weights to approximate the function above ....

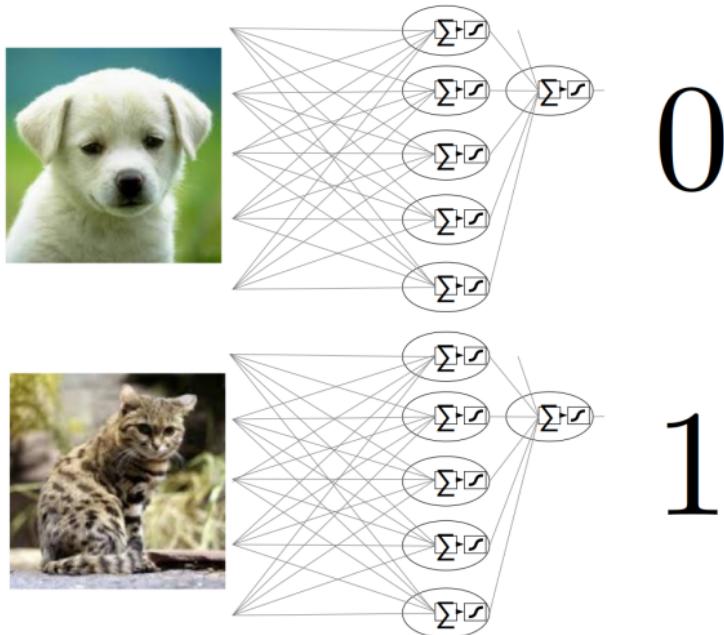
# Solution

14 / 60



# Binary Classification: Desiderata

15 / 60



Source: Google

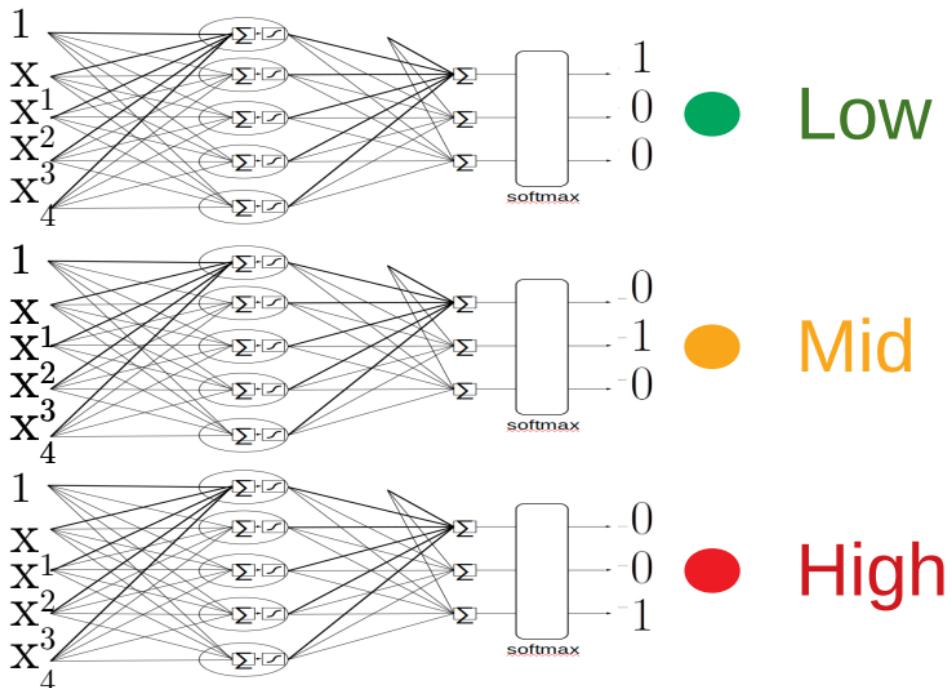
# Multiclass Classification: Desiderata

16 / 60



Ex.: **Telstra Kaggle Competition** [Tai17]:

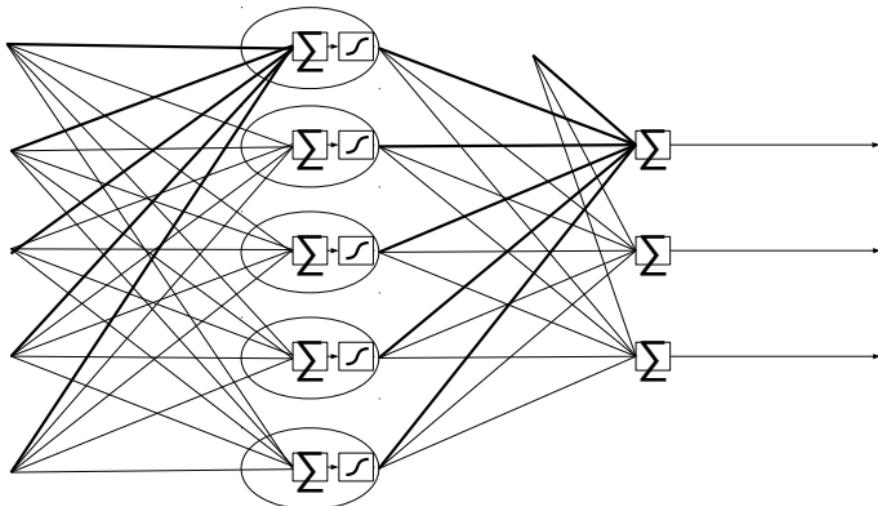
- Features: description of events (location, resource involved, type of event)
- Predict: the severity of fault



A class is *coded* in a string with one 1.

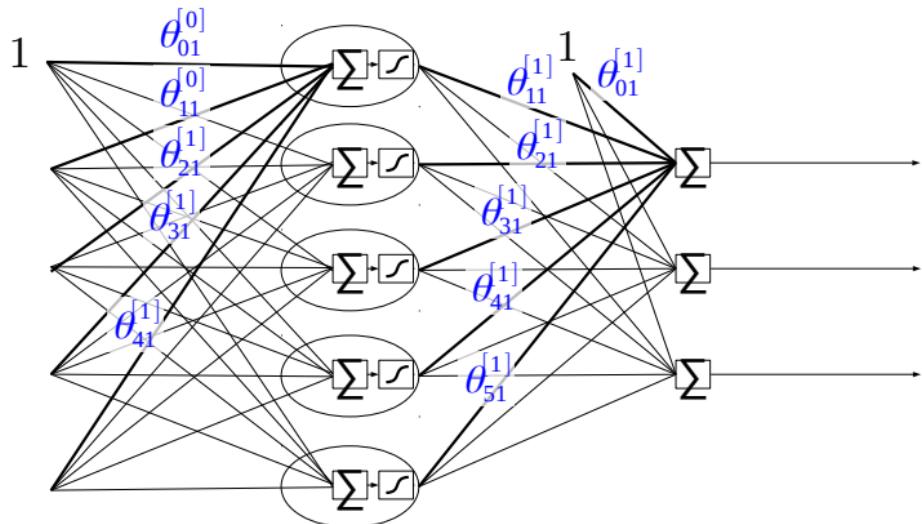
# NN for classification

17 / 60



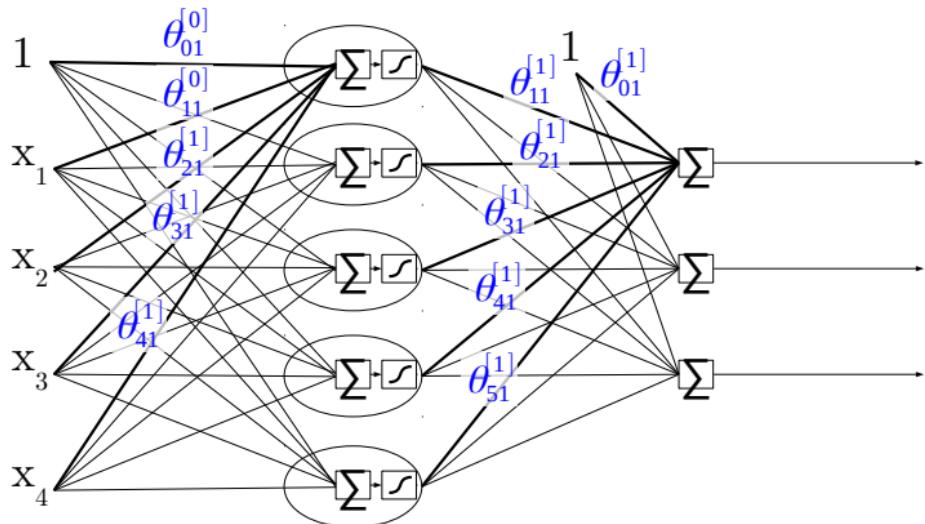
# NN for classification

17 / 60



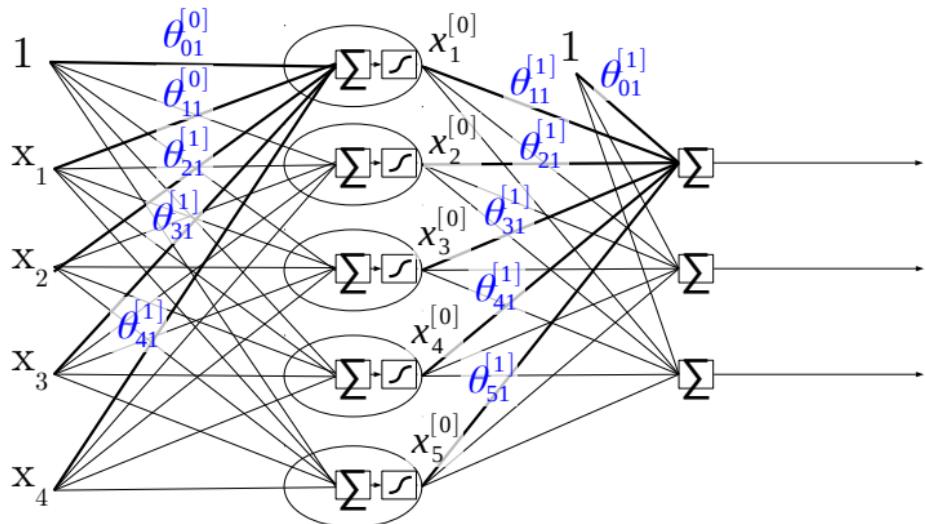
# NN for classification

17 / 60



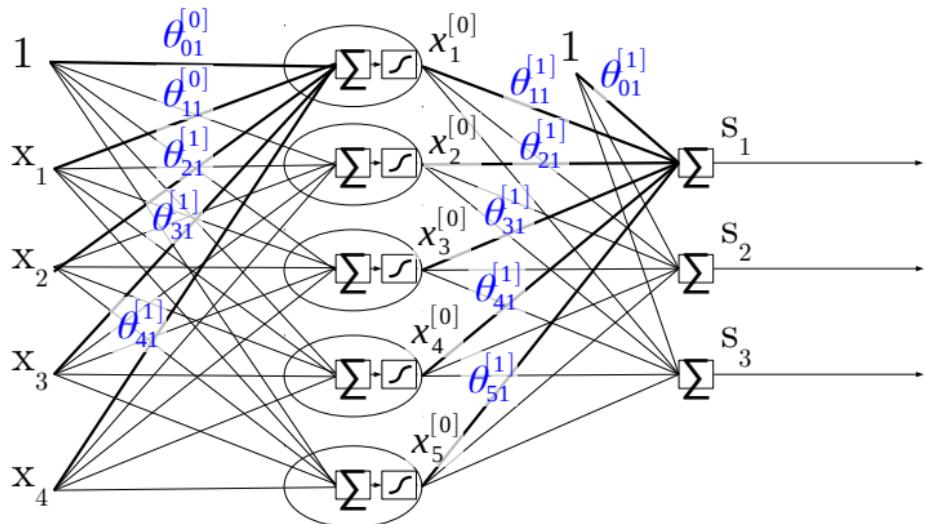
# NN for classification

17 / 60



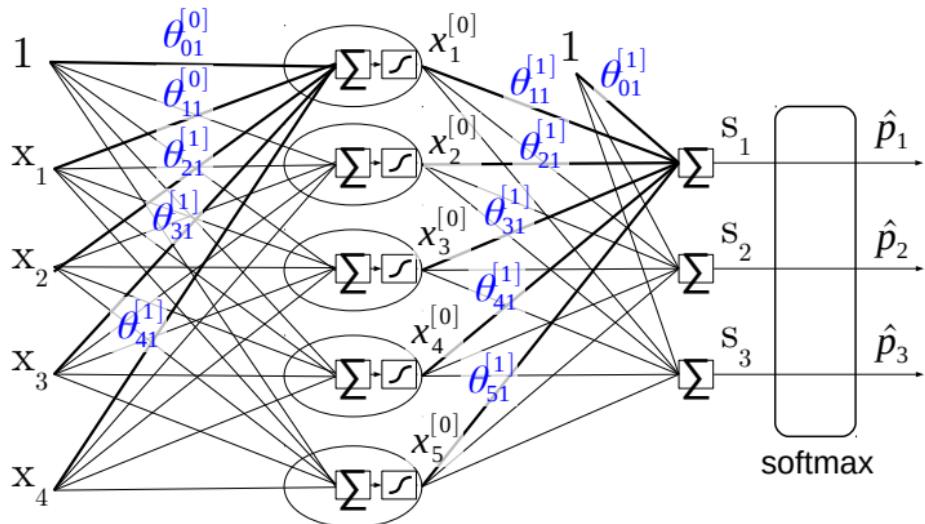
# NN for classification

17 / 60



# NN for classification

17 / 60

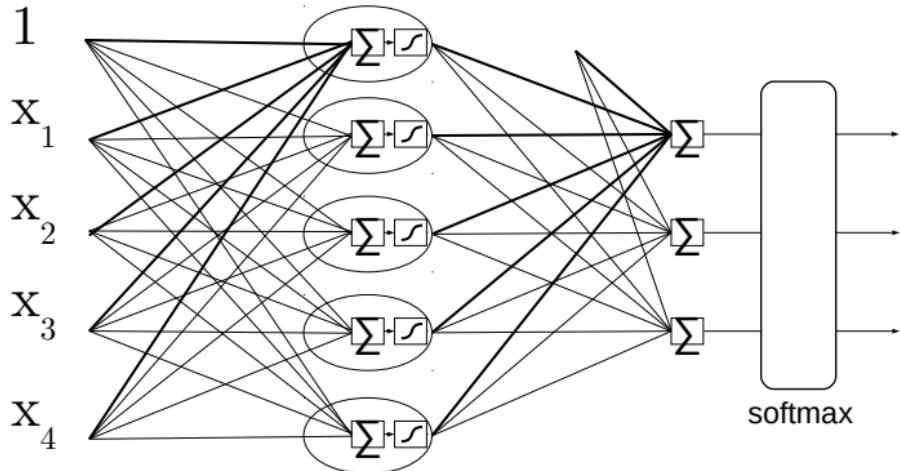


$$\hat{p}_k^{(i)} = \text{softmax}(s_k(\mathbf{x})) = \frac{\exp s_k(\mathbf{x})}{\sum_{z=1}^K \exp s_z}$$

$$k^* = \arg \max_k \hat{p}_k^{(i)}$$

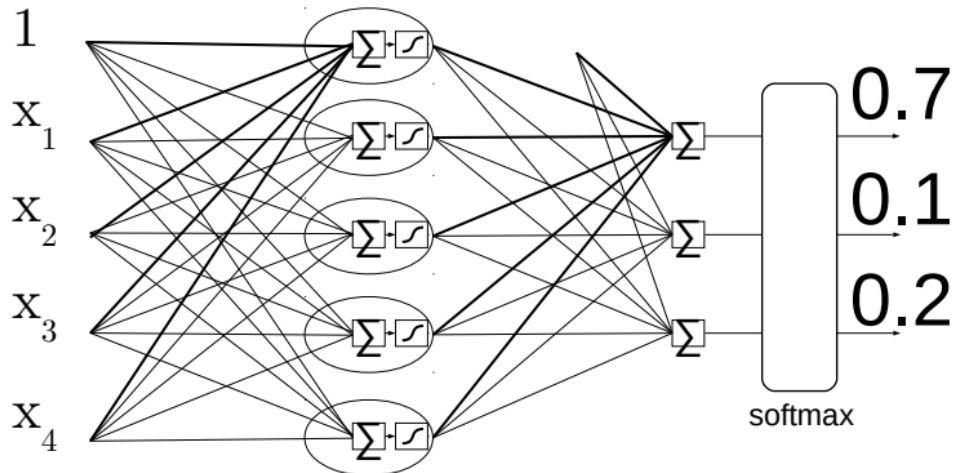
# Multiclass Classification: Prediction

18 / 60



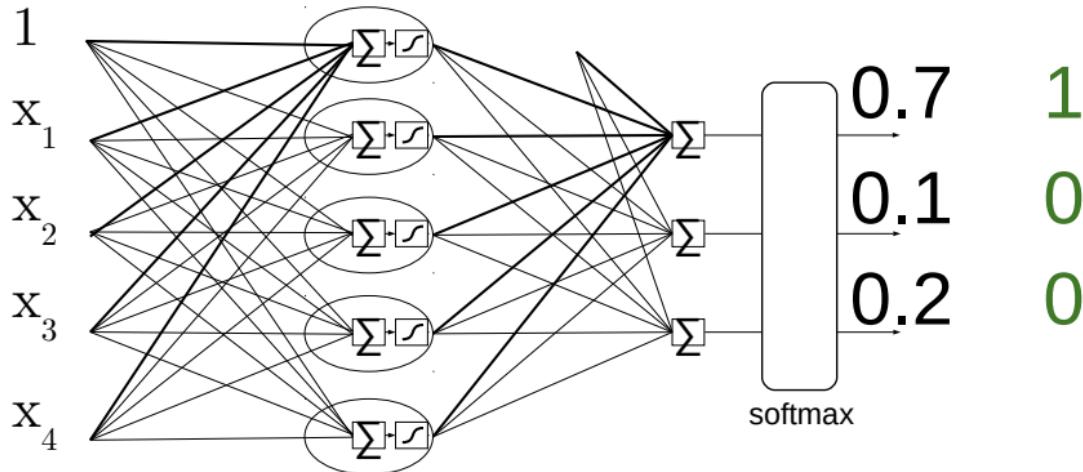
# Multiclass Classification: Prediction

18 / 60



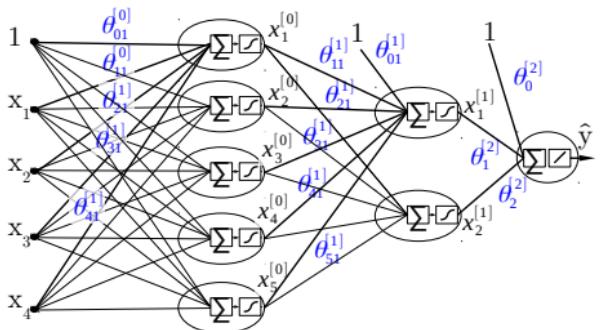
# Multiclass Classification: Prediction

18 / 60

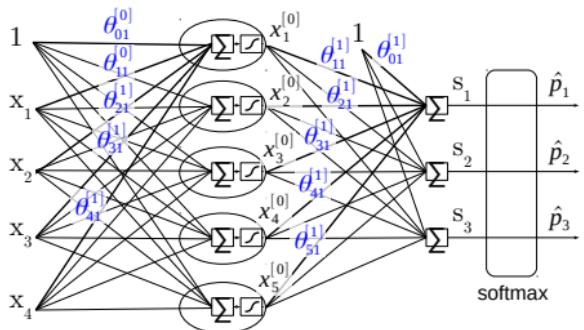


# Activation functions in the output layer

19 / 60



Regression: no activation function



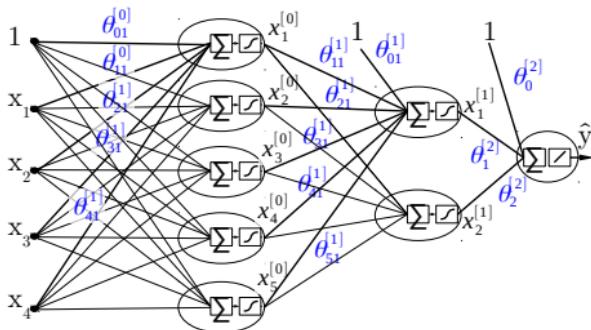
Classification: softmax

## Section 2

### **Training (backpropagation)**

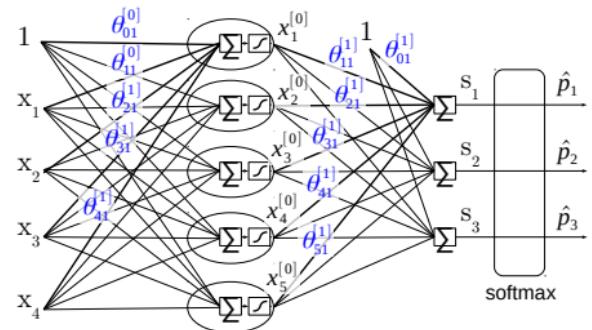
- A NN is completely specified by the weight matrix  $\Theta$
- Training: Given a training set of  $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ , find the “best” matrix

$$\Theta^* \triangleq \arg \min_{\Theta} J(\Theta, \mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}) = \arg \min_{\Theta} \sum_i J(\Theta, \mathbf{x}^{(i)}, y^{(i)})$$



Regression:

$$J(\Theta, \mathbf{x}^{(i)}, y^{(i)}) = \text{MSE} = (y^{(i)} - \hat{y}^{(i)})^2$$



Classification:

$$J(\Theta, \mathbf{x}^{(i)}, y^{(i)}) = \text{cross-entropy} = -\ln \hat{p}_{k(i)}^{(i)}$$

where  $k(i)$  is the true class of sample  $i$ .

- Objective:  $\min_{\theta} J(\theta, \mathbf{X}, \mathbf{y})$ 
  - where  $(\mathbf{X}, \mathbf{y})$  is the training dataset.
- Initialize  $\theta$  randomly
  - ... but wisely (see page 333-4 of [Ge19] for initialization techniques)
- Gradient Descent: at each iteration

$$\theta := \theta - \eta \cdot \nabla_{\theta} J(\theta, \mathbf{X}, \mathbf{y})$$

- $\eta$ : learning rate
- Gradient:  $\nabla J(\theta, \mathbf{X}, \mathbf{y}) \triangleq \left( \frac{\partial}{\partial \theta_{qv}^{[l]}} J(\theta, \mathbf{X}, \mathbf{y}) \right)_{qvl}$ 
  - $\theta_{qv}^{[l]}$ : weight of layer  $l$  connecting neuron  $q$  to neuron  $v$ .

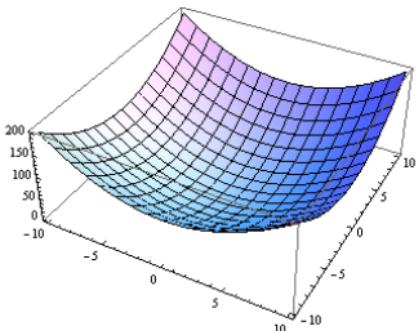
# Non-convexity

23 / 60

Logistic Regression:

$J(\theta, \mathbf{X}, \mathbf{y})$  derivable and convex  
(unique minimum)

⇒ Convergence to minimum guaranteed.



By JackB09 [Public domain], via Wikimedia Commons

Neural Network:

$J(\theta, \mathbf{X}, \mathbf{y})$  derivable but not convex (local minima)

⇒ Gradient descent may be trapped in local minima

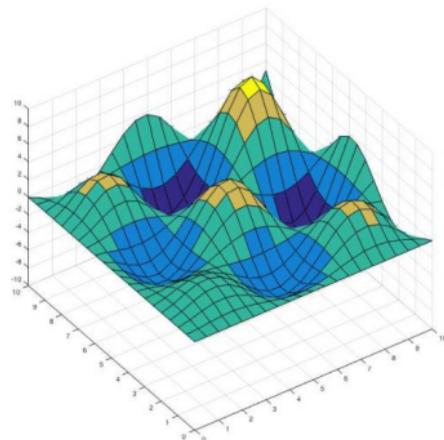
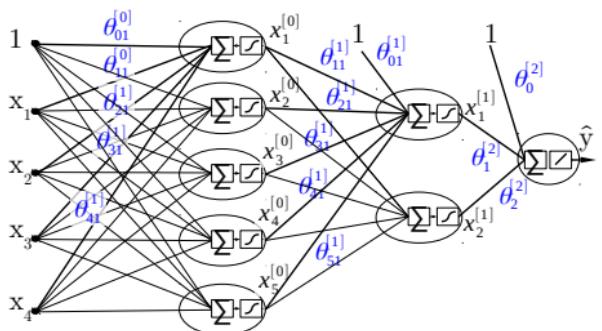


Figure from Bauso, Dario & Gao, Jian & Tembine, Hamidou. (2017). Distributionally Robust Games: f-Divergence and Learning.

# Derivatives in the last layer $L$



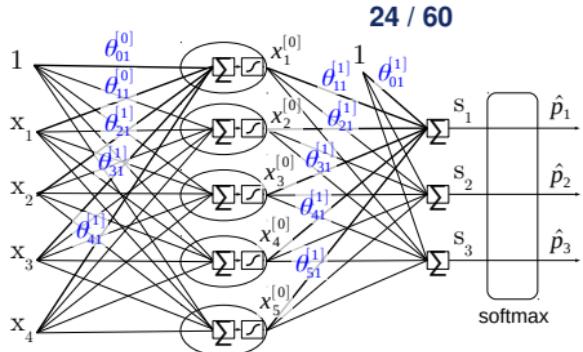
Regression:

$$J(\theta, \mathbf{x}^{(i)}, y^{(i)}) = (y^{(i)} - \hat{y}^{(i)})^2$$

$$= \left( \underbrace{y^{(i)} - \sum_q \theta_q^{[L]} \cdot x_q^{(i)[L-1]}}_{\varepsilon^{(i)}} \right)^2$$

$$\Rightarrow \frac{\partial}{\partial \theta_q^{[L]}} J(\theta, \mathbf{x}^{(i)}, y^{(i)}) = -2 \cdot \varepsilon^{(i)} \cdot x_q^{(i)[L-1]}$$

\*\*\*\*\*



Classification:

$$J(\theta, \mathbf{x}^{(i)}, y^{(i)}) = -\ln \hat{p}_{k(i)}^{(i)}$$

$$= -\ln \frac{\exp \left( \theta_{k(i)}^{[L]T} \cdot \mathbf{x}^{(i)[L-1]} \right)}{\sum_{z=1}^K \exp \left( \theta_z^{[L]T} \cdot \mathbf{x}^{(i)[L-1]} \right)}$$

$k(i)$ : true class of sample  $i$ .

$\theta_z^{[L]}$ : vector of weights of the  $z$ -th exit.

We can compute  $\frac{\partial}{\partial \theta_{qz}^{[L]}} J(\theta, \mathbf{x}^{(i)}, y^{(i)})$

## Backpropagation (i.e. $\nabla_{\theta} J()$ computation)

25 / 60

Let us compute  $\frac{\partial J}{\partial \theta_{vq}^{[l]}}(\theta, \mathbf{x}, y)$ ,  $\forall$  training sample  $(\mathbf{x}, y)$ :

- Consider the  $q$ -th neuron of intermediary layer  $l$ .
- *Weighted input* (to the neuron):

$$a_q^{[l]} = \theta_q^{[l]T} \cdot \mathbf{x}^{[l-1]} = \sum_z \theta_{zq}^{[l]} \cdot x_z^{[l-1]}$$

$$\bullet \quad \frac{\partial J}{\partial \theta_{vq}^{[l]}} = \underbrace{\frac{\partial J}{\partial a_q^{[l]}}}_{\triangleq \text{error } \delta_q^{[l]}} \cdot \underbrace{\frac{\partial a_q^{[l]}}{\partial \theta_{vq}^{[l]}}}_{x_v^{[l]}} = \delta_q^{[l]} \cdot x_v^{[l-1]}$$

- **Multivariable chain rule** of derivation:

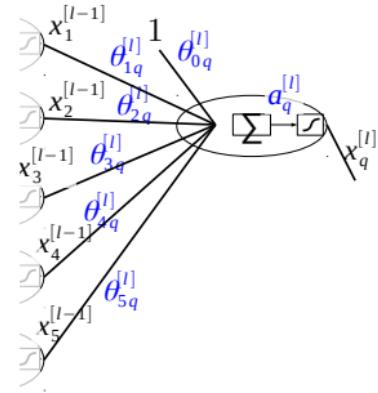
$$\delta_q^{[l]} \triangleq \frac{\partial J}{\partial a_q^{[l]}} = \sum_z \frac{\partial J}{\partial a_z^{[l+1]}} \cdot \frac{\partial a_z^{[l+1]}}{\partial a_q^{[l]}} = \sum_z \delta_z^{[l+1]} \cdot \frac{\partial a_z^{[l+1]}}{\partial a_q^{[l]}}$$

- Recall that

$$a_z^{[l+1]} = \sum_{z'} \theta_{z'z}^{[l+1]} \cdot x_{z'}^{[l]} = \sum_{z'} \theta_{z'z}^{[l+1]} \cdot \sigma(a_{z'}^{[l]})$$

$$\bullet \implies \frac{\partial a_z^{[l+1]}}{\partial a_q^{[l]}} = \theta_{qz}^{[l+1]} \cdot \sigma'(a_q^{[l]})$$

$$\bullet \implies \delta_q^{[l]} = \sigma'(a_q^{[l]}) \cdot \sum_z \delta_z^{[l+1]} \cdot \theta_{qz}^{[l+1]}$$



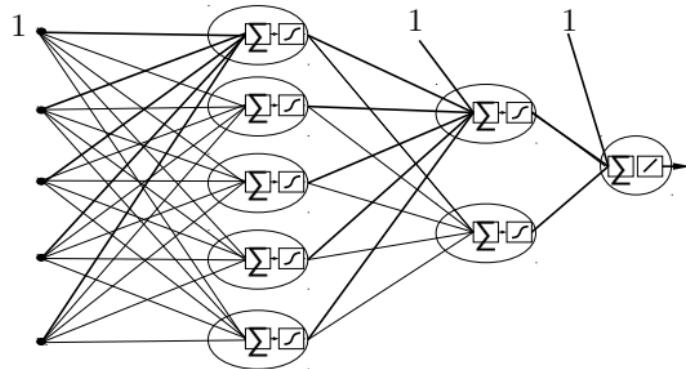
The errors  $\delta_z^{[l+1]}$  propagate back to layer  $l$ .

In NN for regression, in the last layer  $L$  there is only one neuron and

$$\delta^{[L]} = 1$$

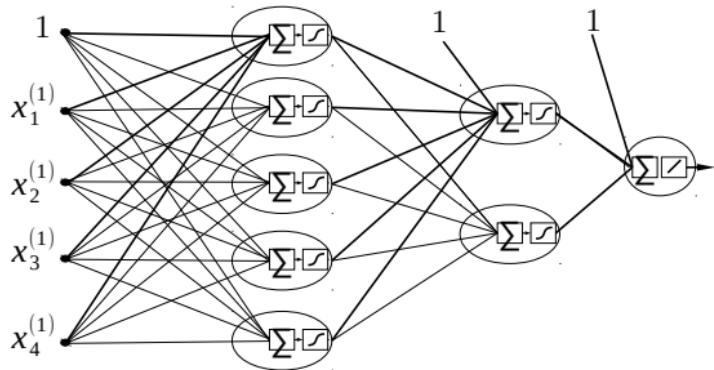
# Forward and Backward Propagation

26 / 60



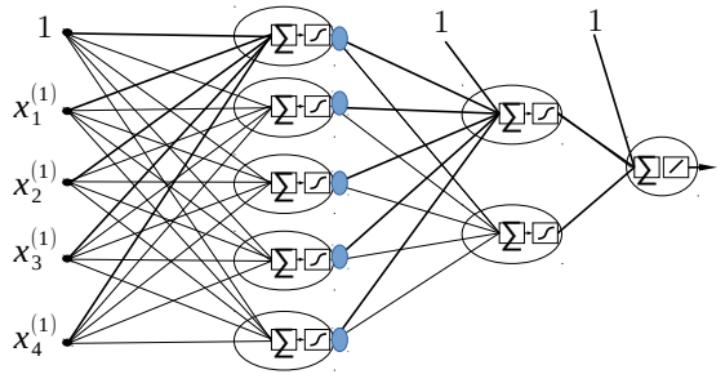
# Forward and Backward Propagation

26 / 60



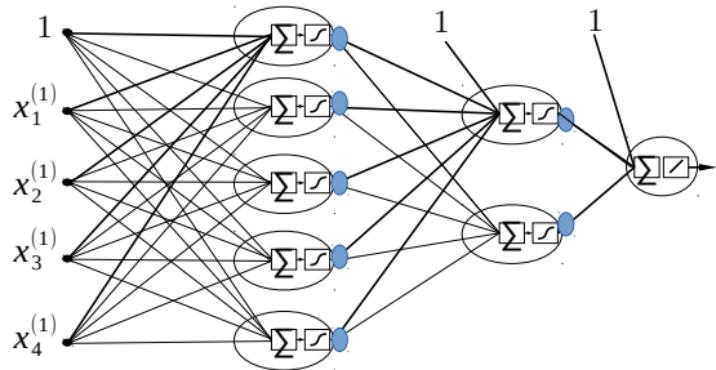
# Forward and Backward Propagation

26 / 60



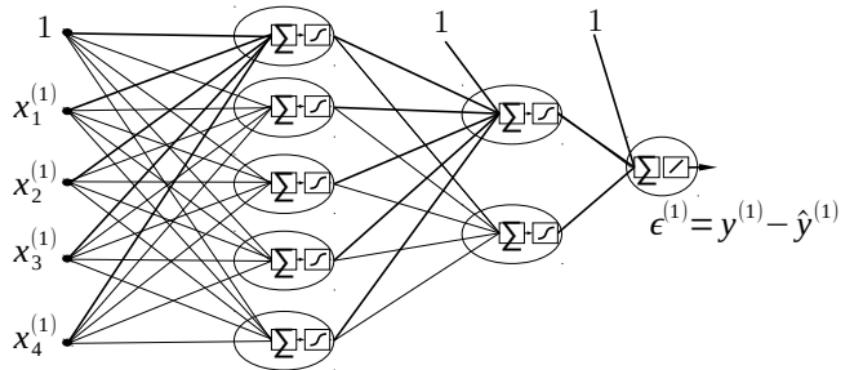
# Forward and Backward Propagation

26 / 60



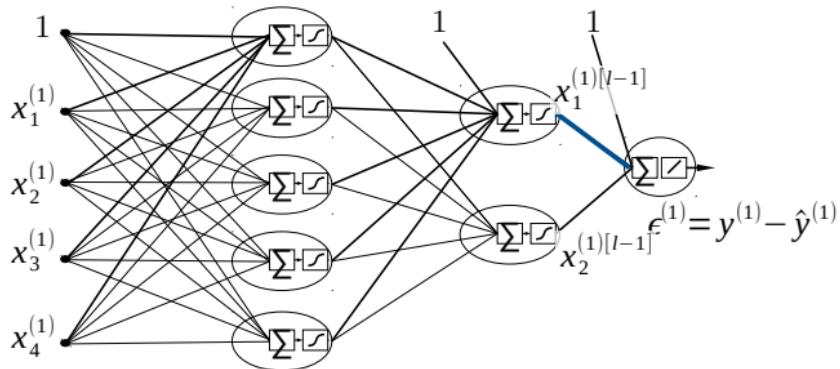
# Forward and Backward Propagation

26 / 60



# Forward and Backward Propagation

26 / 60



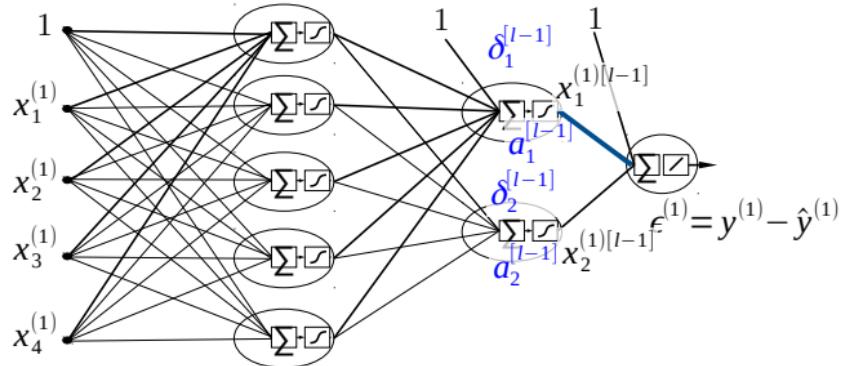
- Compute

$$\frac{\partial}{\partial \theta_q^{[L]}} J(\theta, \mathbf{x}^{(1)}, y^{(1)}) = \boldsymbol{\varepsilon}^{(1)} \cdot x_q^{(1)[L-1]}$$

for all  $\theta_q^{[L]}$  in the last layer

# Forward and Backward Propagation

26 / 60

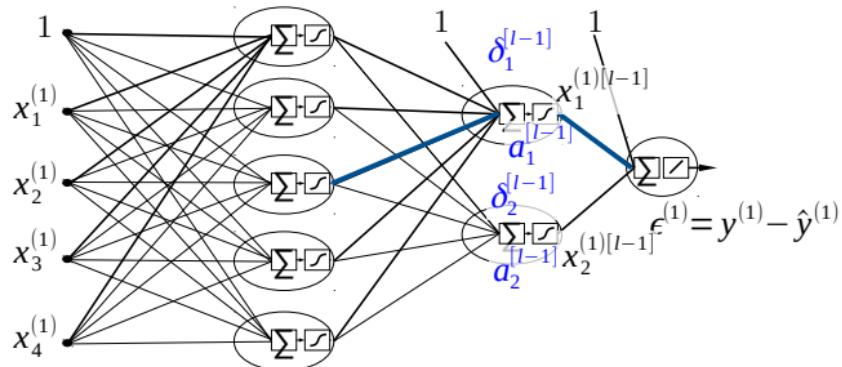


- Compute

$$\delta_q^{[l-1]} = \sigma'(a^{[l-1]}) \cdot \sum_z \delta_z^{[l]} \cdot \theta_{qz}^{[l]}$$

# Forward and Backward Propagation

26 / 60



- Compute

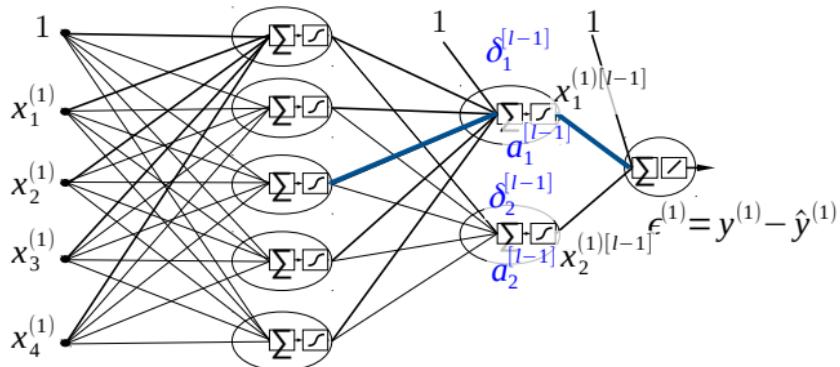
$$\delta_q^{[l-1]} = \sigma'(a^{[l-1]}) \cdot \sum_z \delta_z^{[l]} \cdot \theta_{qz}^{[l]}$$

- and

$$\frac{\partial}{\partial \theta_{vq}^{[l-1]}} J = \delta_q^{[l-1]} \cdot x_v^{(1)[l-1]}$$

# Forward and Backward Propagation

26 / 60



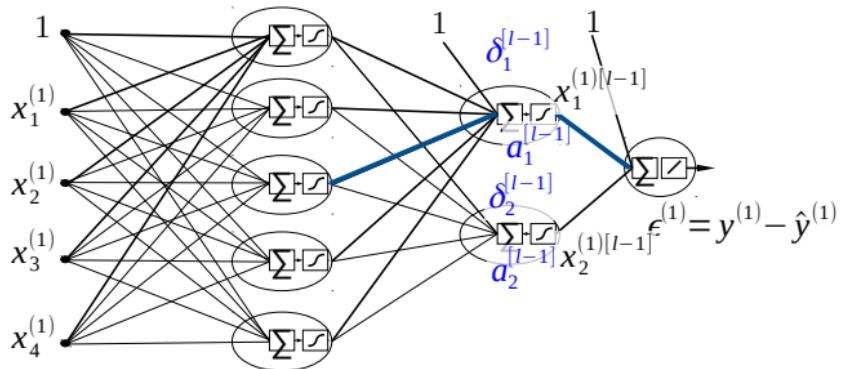
- Do the same for all the weights, backward, to compute

$$\frac{\partial}{\partial \theta_{vq}^{[l-1]}} J = \delta_q^{[l-1]} \cdot x_v^{(1)[l-1]}$$

for all weights  $\theta_{vq}^{[l]}$

# Forward and Backward Propagation

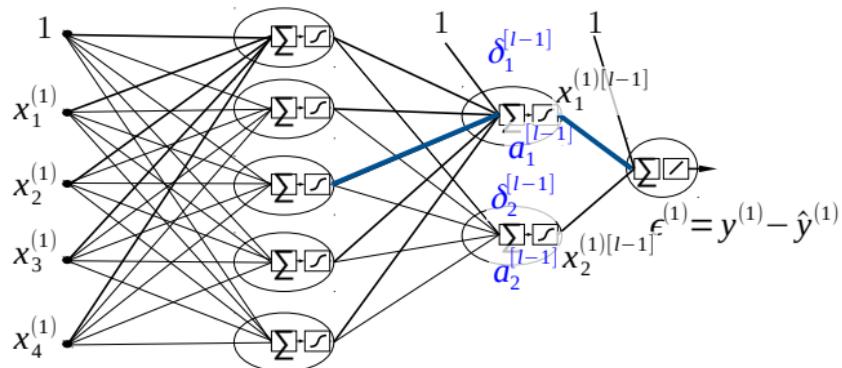
26 / 60



- We thus obtain  $\nabla J(\Theta, \mathbf{x}^{(i)}, \mathbf{y}^{(i)})$

# Forward and Backward Propagation

26 / 60



- Do the same for all the samples
- Finally

$$\nabla J(\theta, \mathbf{X}, \mathbf{y}) = \frac{1}{M} \sum_{i=1}^M \nabla J(\theta, \mathbf{x}^{(i)}, y^{(i)})$$

And update

$$\theta := \theta - \eta \cdot \nabla J(\theta, \mathbf{X}, \mathbf{y})$$

## 1. Full Gradient Descent

- Predict  $\hat{y}^{(i)}$  for all  $\mathbf{x}^{(i)}$  in  $\mathcal{D}^{\text{train}}$
- $\nabla J(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = \frac{1}{M} \sum_{i=1}^M \nabla J(\boldsymbol{\theta}, \mathbf{x}^{(i)}, y^{(i)})$
- Update weights  $\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \cdot \nabla J(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y})$

## 2. Stochastic Gradient Descent

(update parameters at each sample)

- For each sample  $\mathbf{x}^{(i)}$ 
  - Predict  $\hat{y}^{(i)}$
  - Compute  $J(\boldsymbol{\theta}, \mathbf{x}^{(i)}, y^{(i)})$
  - Assume  $\nabla J(\boldsymbol{\theta}, \mathbf{x}^{(i)}, y^{(i)}) \simeq \nabla J(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y})$
  - Update weights  
$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \cdot \nabla J(\boldsymbol{\theta}, \mathbf{x}^{(i)}, y^{(i)})$$

## 3. Batch Gradient Descent

- Divide  $\mathcal{D}^{\text{train}}$  in batches
- Update the parameters after predicting each batch.

- **Epoch:** Sequence of predictions on the entire  $\mathcal{D}^{\text{train}}$
- How many parameter updates per-epoch (using the 3 strategies)?
- Usually **many epochs** are needed

# Multi-Layer Perceptron Implementation

28 / 60



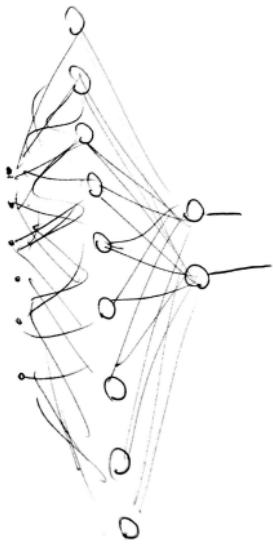
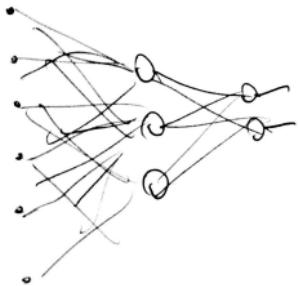
On the right: weights  $\theta$  (potentiometers adjust via motors)

$$\theta := \theta - \eta \cdot \nabla J(\theta, \mathbf{x}^{(i)}, y^{(i)})$$

Figure from [Bis06]

## Section 3

### **Design of NNs**



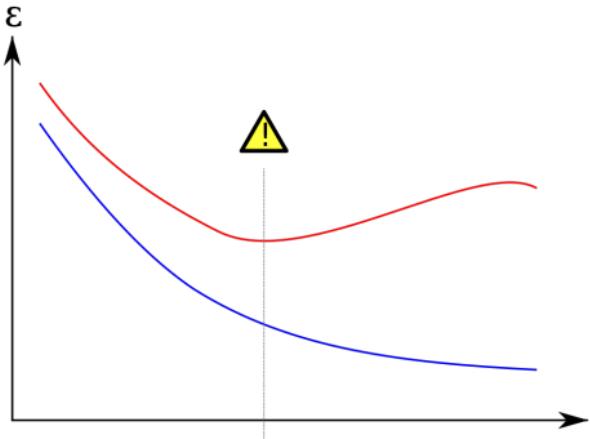
Compression vs. Augmentation.

Some authors use the same number of neurons per layer - pag. 324 of [Ger19]

# Model complexity and Overfitting

31 / 60

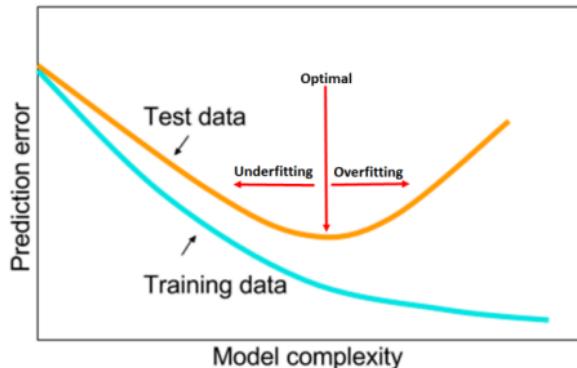
## Overfitting



From User:Gringer, Wikipedia

## Solution:

- Use smaller architectures
- Regularize
- Early Stopping:  
stop training when the test error  
does not improve for some  
consecutive epochs



From [Smi18]

- NN with many parameters are **too flexible**: they can approximate weird functions
- To avoid **overfitting** the training data, we must reduce their flexibility
- **Regularization**
- The loss function to minimize is
  - For regression

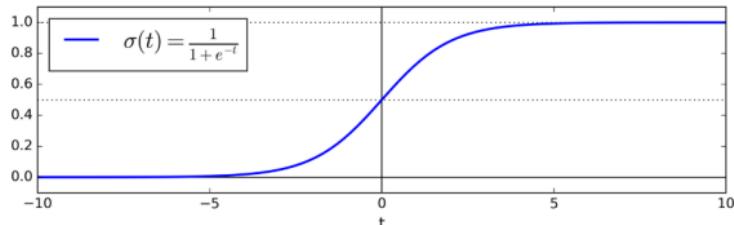
$$J(\theta, \mathbf{X}, \mathbf{y}) = \underbrace{\frac{1}{M} \sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)})^2}_{\text{Error term}} + \underbrace{\alpha \|\theta\|^2}_{\text{Regularization term}}$$

- For classification

$$J(\theta, \mathbf{X}, \mathbf{y}) = \underbrace{-\frac{1}{M} \sum_{i=1}^M \ln \hat{p}_{k(i)}^{(i)}}_{\text{Error term}} + \underbrace{\alpha \|\theta\|^2}_{\text{Regularization term}}$$

where  $k(i)$  is the true class of sample  $i$

- Activation functions like sigmoids are intended to get values in a small range, otherwise they *saturate*.



Ex. If we enter to the neuron 8, 10 the output is practically the same.

- Scaling is also needed because we regularize NNs
- ⇒ Always scale the dataset (StandardScaler or MinMaxScaler)

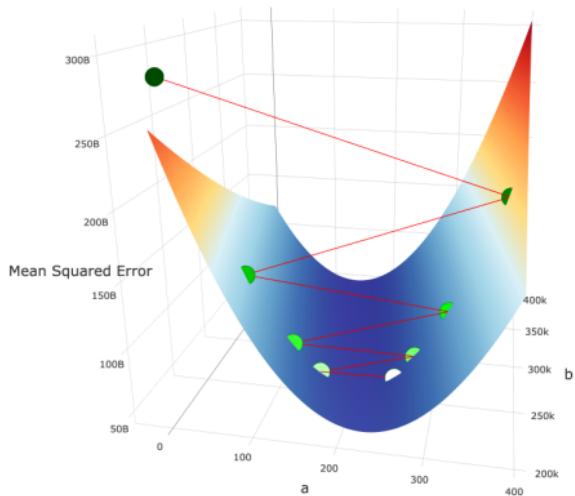
- Hyper-parameters:
  - Architecture
    - Layers? (start with few, increase if needed)
    - Neurons per layer?
  - Learning rate  $\eta$ : too high: noise; too low: slow to converge.
  - How many epochs?
  - Regularization weight  $\alpha$
  - Weight initialization.
  - Batch size.
  - Activation Functions.
- Strategies for tuning
  - Grid search (time consuming)
  - Random search, Bayesian Optimization, Design Space Exploration (time consuming) - see pagg.320-323 of [Ger19]
  - Trial and error, experience (people with less money need to be smarter)

# Fixed learning rate

35 / 60

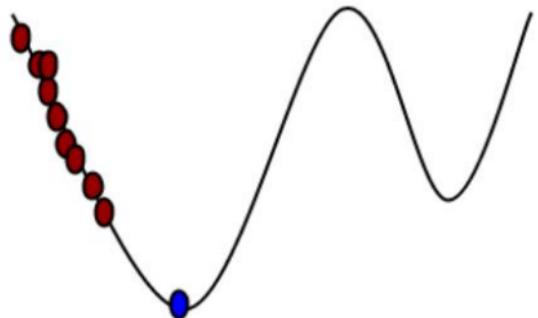
$$\theta := \theta - \eta \cdot \nabla J(\theta, \mathbf{X}, \mathbf{y})$$

$\eta$  too large

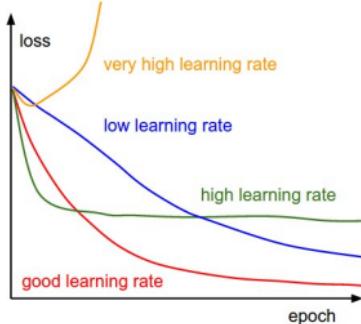


From S. Harrington [blog](#)

$\eta$  too small

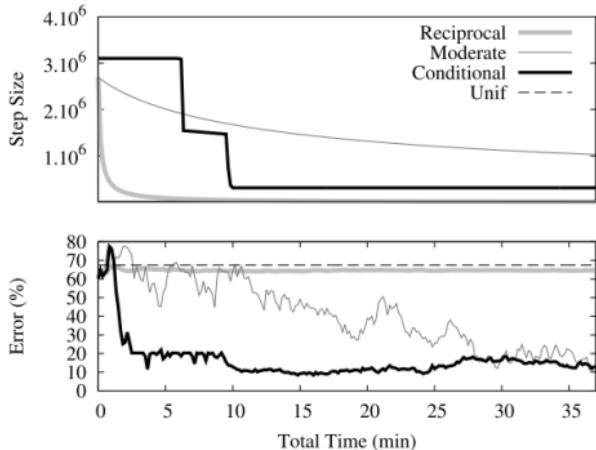
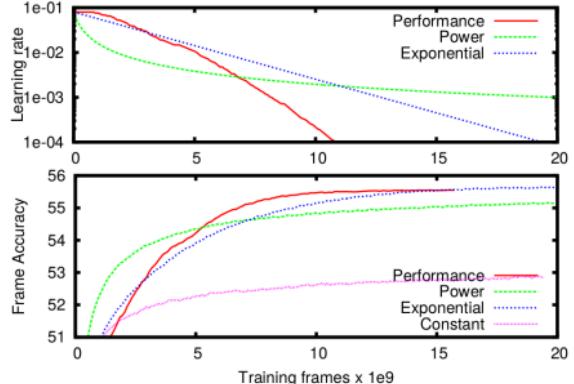


From ProgrammerSought [blog](#).



From CS231 [class](#) at  
Stanford

Start with large learning rates and reduce them after parameter updates



From [Sen13] and [ADR18]

See pagg.359-364 of [Ger19] to know more.

Gradient descent is

$$\theta := \theta - \eta \cdot \nabla_{\theta} J(\theta, \mathbf{X}, \mathbf{y})$$

Other optimizers use a different parameter update equation, using gradient in a smarter way.

Most popular: Adaptive Moment Estimation (Adam )

[Animated comparison](#) of optimizers.

See pagg. 351-359 of [Ger19] to know more.

*“Friends, don’t let friends use  
mini-batches larger than 32 ”*  
Yann LeCun tweet, 2018



From Wikipedia

Yann LeCun (Facebook, New York University, ACM Turing Award)

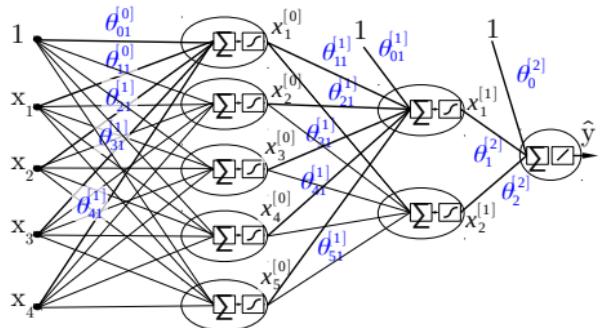
## Large minibatches

- Allow to use GPU parallelization
- Risk of instability in loss minimization

# Activation functions

39 / 60

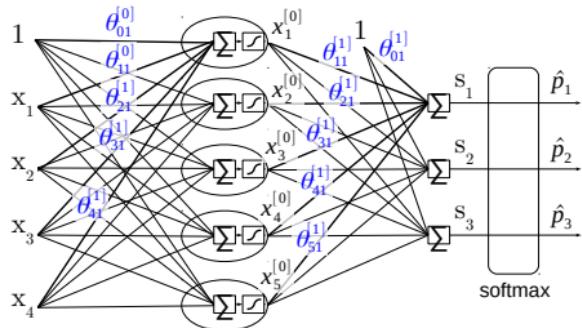
In the last layer



Regression: no activation function

In the hidden layer:

- The most popular is **relu**
- Sigmoid: old school, it has some problems (p 335 of [Ger19])



Classification: softmax

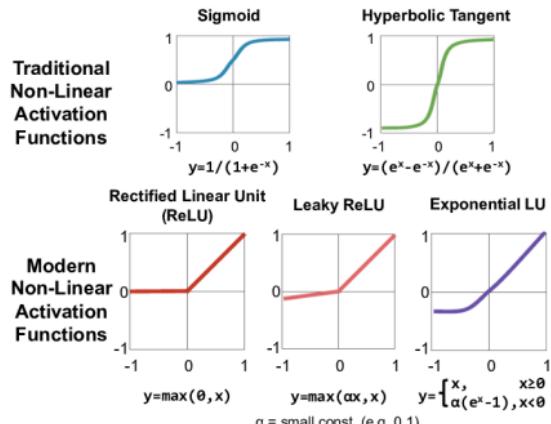
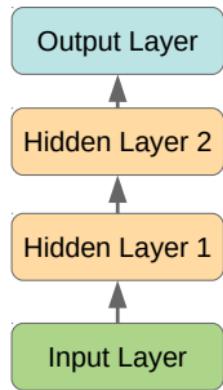


Figure from [SCYE17].

## Section 4

### **Complex architectures**

Multi-layer perceptron



Other architectures

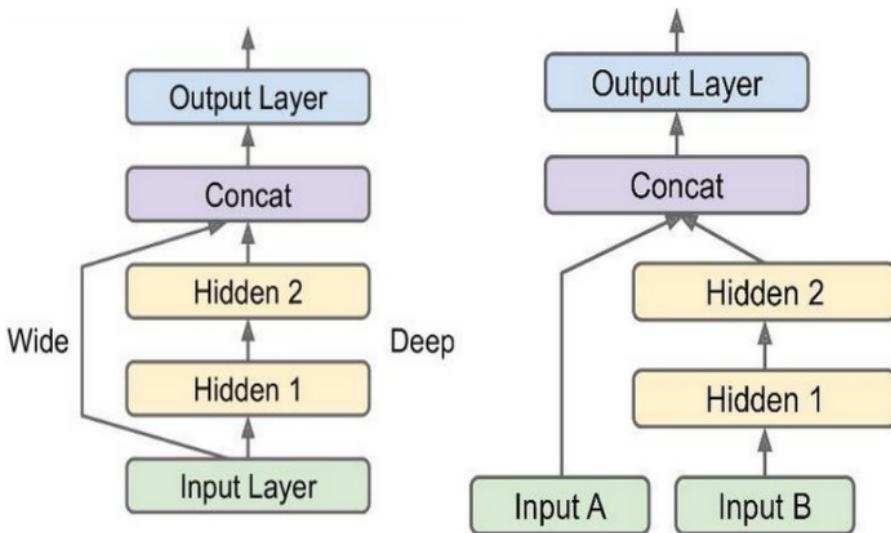


Figure from [Ger19].

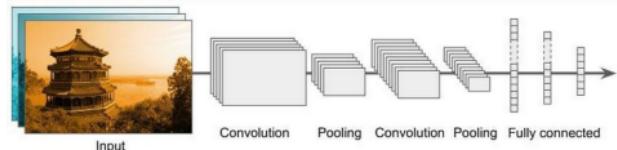
- Limit of Multi-Layer Perceptron: all input data are “deformed” by hidden layers.
- Other architectures are able to *bypass* some hidden layer
- Feel free to experiment with them in your project (pagg.308-313 of [Ger19]).

# Notable deep Neural Networks

42 / 60

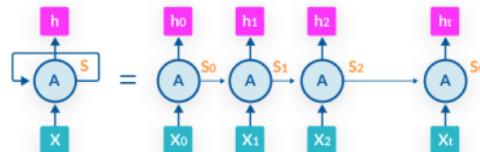
Convolutional NN:

- Image processing



Recurrent NN:

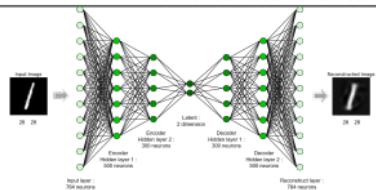
- Time series, language modeling



Unrolled Recurrent Neural Network

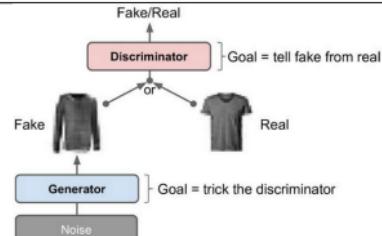
Autoencoder (AE):

- Dimensionality reduction and anomaly detection



Generative Adversarial Networks (GAN):

- To generate images or sounds



Figures from [Ger19], missinglink.ai, medium.com, [Ger19]

# How to choose the right Neural Network?

43 / 60

No standard procedure  $\Rightarrow$  Need for intuition, experience and, more importantly, trial and test  $\Rightarrow$  Neural Networks are an art!

However, some rough guidelines are:

- Image in input  $\Rightarrow$  Convolutional Neural Networks (slide 42)
- Time series in input  $\Rightarrow$  Recurrent Neural Networks (slide 42)
- The output layer depends on the task (regression or classification - slide 19)
- Size: start with a small neural network (few layers, few neurons per layer) and check the test result. Improve this result via Early Stopping and Regularization (slide 31). The result will be your reference baseline. Then, try with bigger architectures and compare the test error with the reference baseline (slide 42)
- Activation function and optimizers: use the latest findings from research (e.g., relu as activation function and Adam as optimizer - slide 9)
- If you have a lot of servers and a lot of time: automatically train several neural networks and get the best after some days / weeks! (grid search, randomized search)

Note that guidelines are continuously broken/replaced, as deep learning progresses!



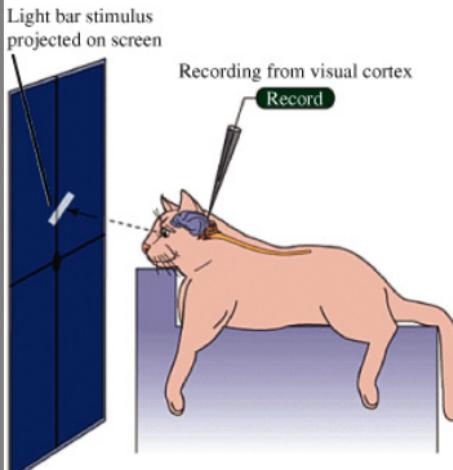
Go to notebook 04.neural-networks.ipynb

# Convolutional NN

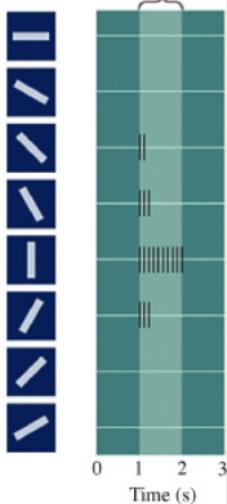
45 / 60

A celebrity among cats: Hubel and Wiesel (Nobel prize '81) cat (Harvard)

A Experimental setup



B Stimulus orientation      Stimulus presented

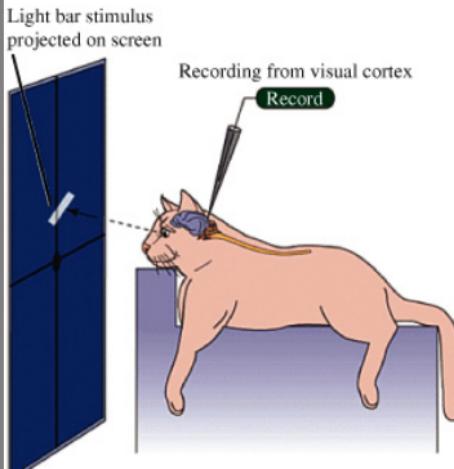


Source: Purves, Brains: How They Seem to Work

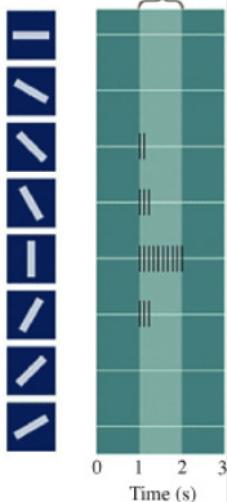
Source: youtube

A celebrity among cats: Hubel and Wiesel (Nobel prize '81) cat (Harvard)

**A** Experimental setup



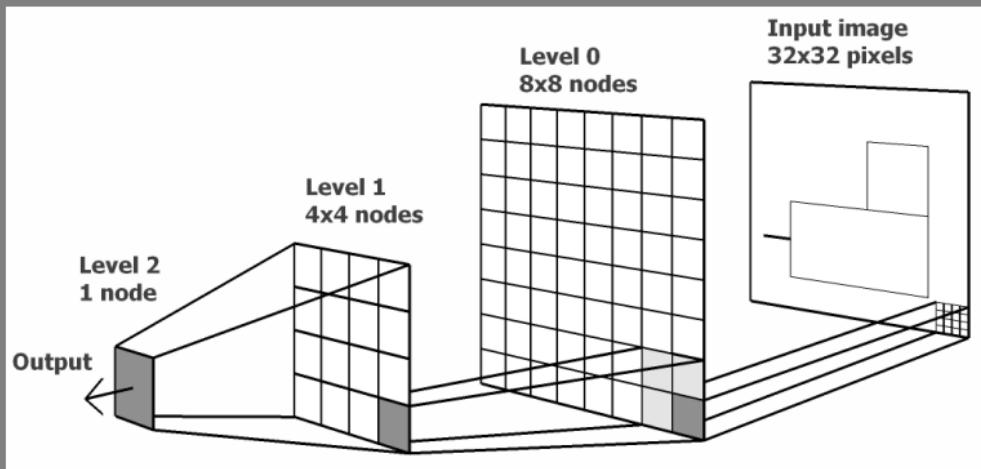
**B** Stimulus orientation      Stimulus presented



Source: Purves, Brains: How They Seem to Work

Source: youtube

- Instead of representing neurons stacked in columns, for image recognition it is easier to imagine them organized in matrices
- No difference in terms of mathematics



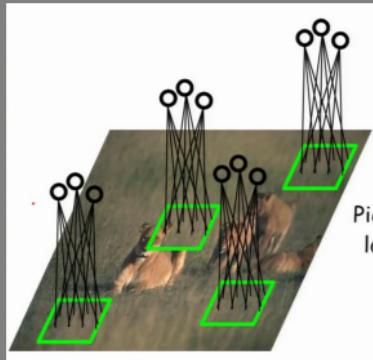
Source: Saulius Garalevicius 2010

## Main idea

- In “classic” NN, we let it learn “wild” by
  - drawing all weights
  - let the weights take any value
- Can we learn from the cat?
  - Add structure to the architecture of the NN
  - Add constraints to the values of the weights
  - Do this by taking inspiration from the way vision works in living beings

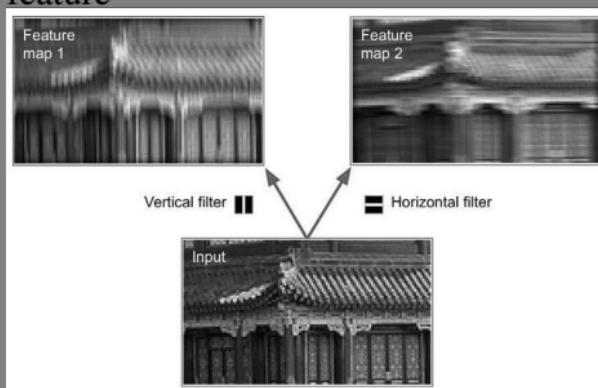
# 1st Hidden Layer: Feature Maps

48 / 60



- Each neuron in the 1st layer is only connected to a *patch* (e.g., 5px X 5px) of pixels.
- Several neurons (3 in the example) are attached to the same patch, each looking for a different *feature*. Output  $\sim 0$  or  $\sim 1$ .
- *Feature map*: Set of neurons looking for the same feature

Source: Nando de Freitas, Lectures Machine Learning, University of British Columbia

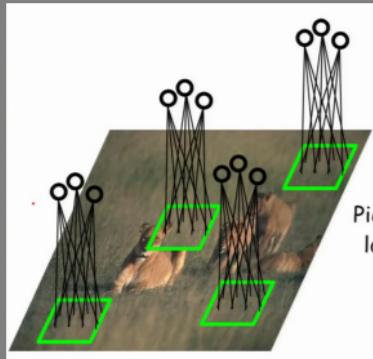


From [Ger19]

- A neuron “implements” a vertical filter when its weights are 1 in the center line and 0 elsewhere.

# 1st Hidden Layer: Feature Maps

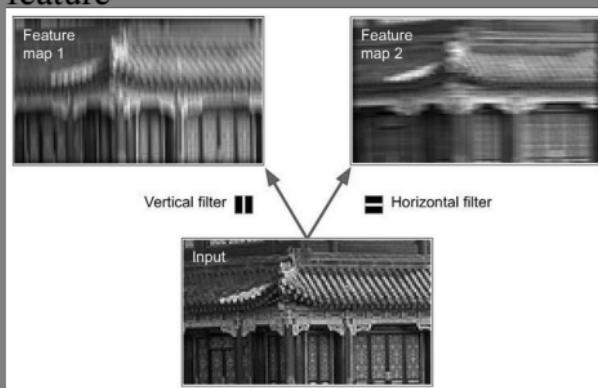
48 / 60



Source: Nando de Freitas, Lectures Machine Learning, University of British Columbia

Who decides the filters?

- Each neuron in the 1st layer is only connected to a *patch* (e.g., 5px X 5px) of pixels.
- Several neurons (3 in the example) are attached to the same patch, each looking for a different *feature*. Output  $\sim 0$  or  $\sim 1$ .
- *Feature map*: Set of neurons looking for the same feature

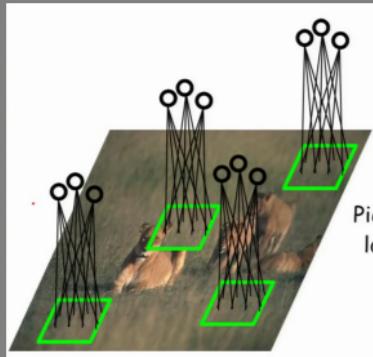


From [Ger19]

- A neuron “implements” a vertical filter when its weights are 1 in the center line and 0 elsewhere.

# 1st Hidden Layer: Feature Maps

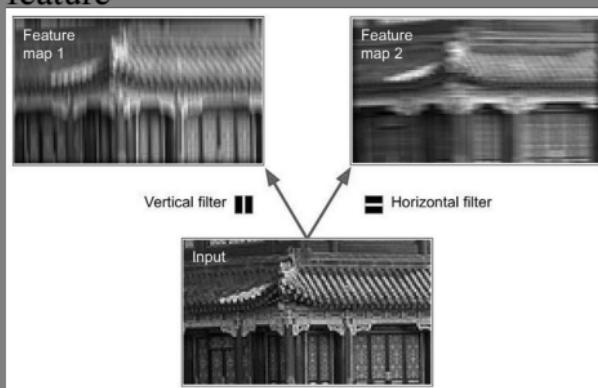
48 / 60



Source: Nando de Freitas, Lectures Machine Learning, University of British Columbia

Who decides the filters?  
*Gradient descent*

- Each neuron in the 1st layer is only connected to a *patch* (e.g., 5px X 5px) of pixels.
- Several neurons (3 in the example) are attached to the same patch, each looking for a different *feature*. Output  $\sim 0$  or  $\sim 1$ .
- *Feature map*: Set of neurons looking for the same feature

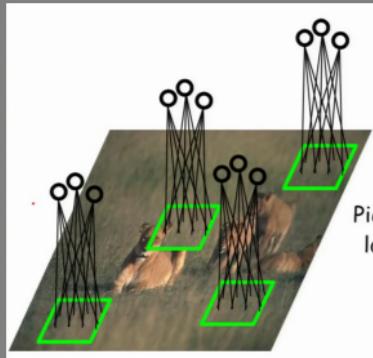


From [Ger19]

- A neuron “implements” a vertical filter when its weights are 1 in the center line and 0 elsewhere.

# 1st Hidden Layer: Feature Maps

48 / 60

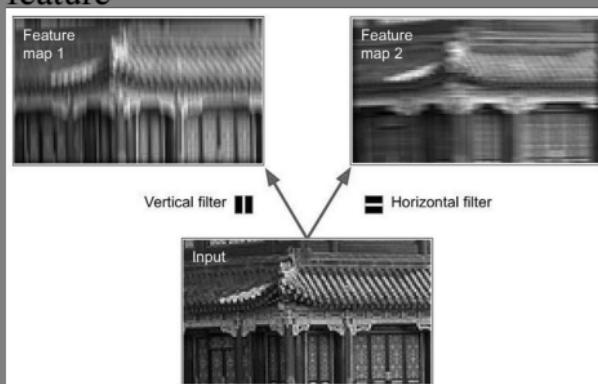


Source: Nando de Freitas, Lectures Machine Learning, University of British Columbia

Who decides the filters?  
*Gradient descent*

How to force the neurons of a feat.map to look for the same feature?

- Each neuron in the 1st layer is only connected to a *patch* (e.g., 5px X 5px) of pixels.
- Several neurons (3 in the example) are attached to the same patch, each looking for a different *feature*. Output  $\sim 0$  or  $\sim 1$ .
- *Feature map*: Set of neurons looking for the same feature

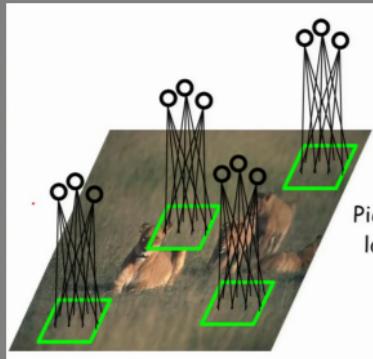


From [Ger19]

- A neuron “implements” a vertical filter when its weights are 1 in the center line and 0 elsewhere.

# 1st Hidden Layer: Feature Maps

48 / 60

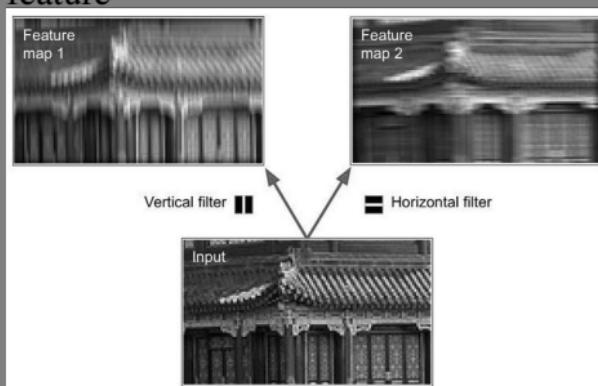


Source: Nando de Freitas, Lectures Machine Learning, University of British Columbia

Who decides the filters?  
*Gradient descent*

How to force the neurons of a feat.map to look for the same feature?  
*Shared weights*

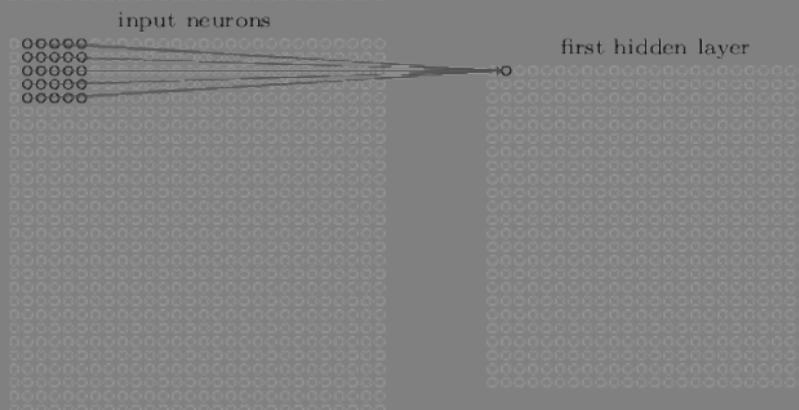
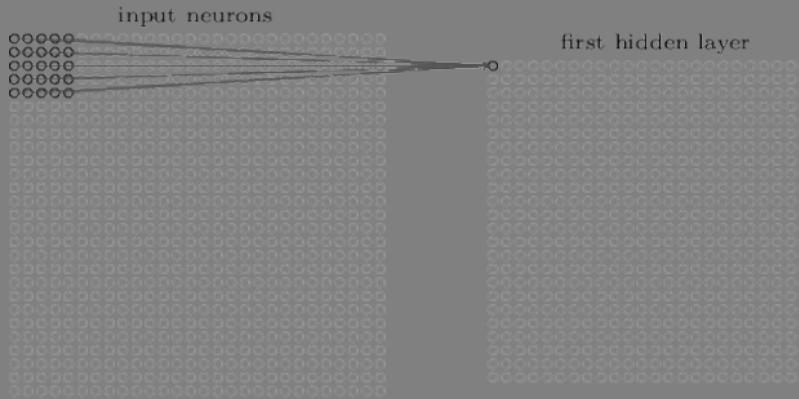
- Each neuron in the 1st layer is only connected to a *patch* (e.g., 5px X 5px) of pixels.
- Several neurons (3 in the example) are attached to the same patch, each looking for a different *feature*. Output  $\sim 0$  or  $\sim 1$ .
- *Feature map*: Set of neurons looking for the same feature



From [Ger19]

- A neuron “implements” a vertical filter when its weights are 1 in the center line and 0 elsewhere.

# 1st Hidden Layer: Feature Map

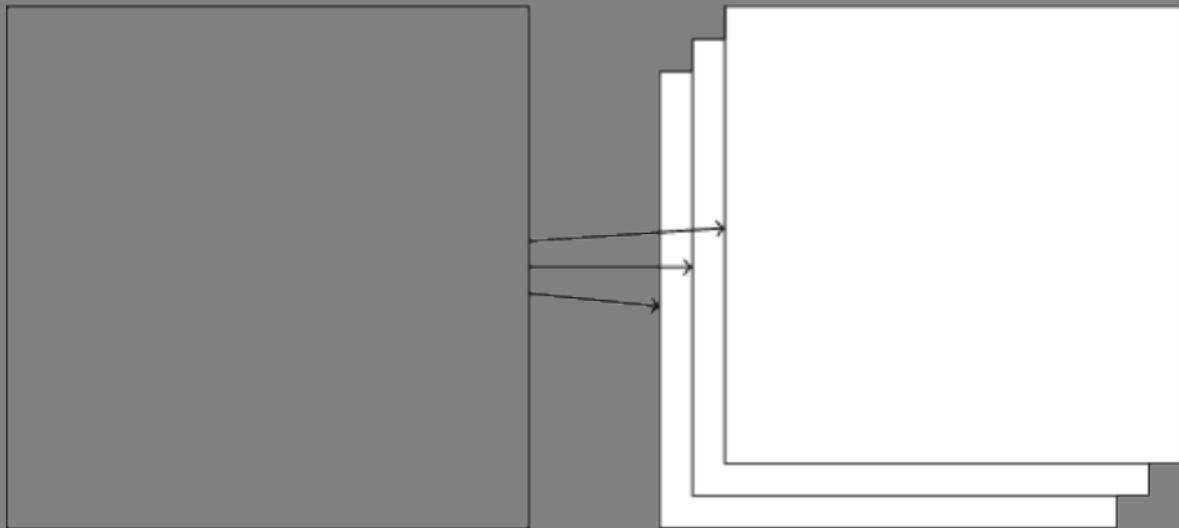


- Patches “seen” by neurons on the same feature map often overlaps
- Hyperparameter: *stride length* (by how much we slide the patch.)
- Sliding patch

Source: M. Nielsen - Neural Networks and Deep Learning

## 1st Hidden Layer: Feature Map

50 / 60

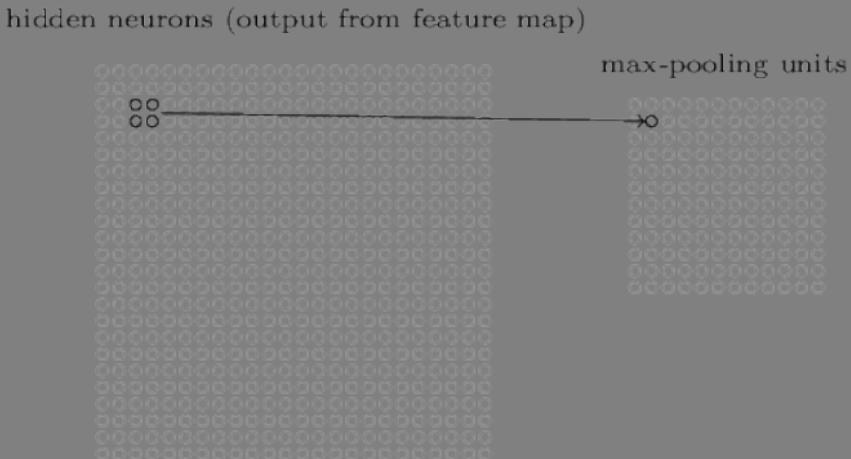


- We can visually organize the 1st layer as a set of *feature maps*

Source: M. Nielsen - Neural Networks and Deep Learning

## 2nd Hidden Layer: Pooling

51 / 60

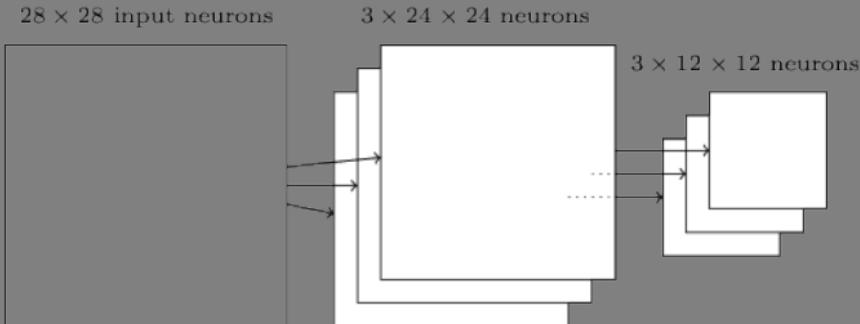


- Max (or other function - hyperparameter!) of the output of a patch in a feature map
- Meaning: is the feature present in a region of the image?
- No weights to learn here

Source: M. Nielsen - Neural Networks and Deep Learning

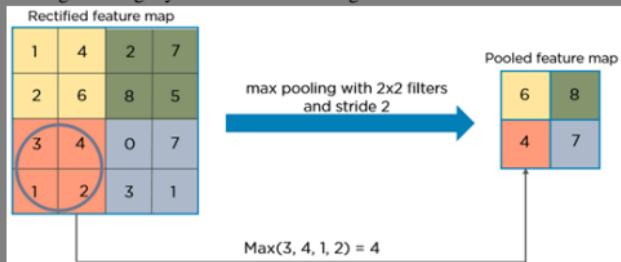
## 2nd Hidden Layer: Pooling

52 / 60

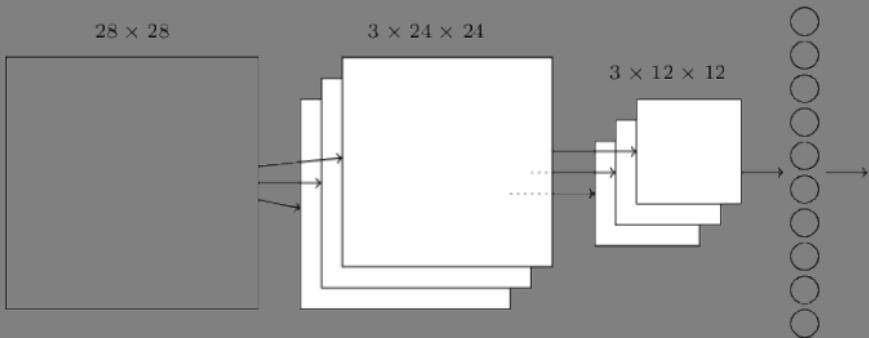


Source: M. Nielsen - Neural Networks and Deep Learning. Pooling layer is the one on the right.

- One pool per feature map
- Similar to convolutional layer, but no weights
  - Just take the average or the max of the patch
- Goal: summarizing features



From [video](#) on Simplilearn.

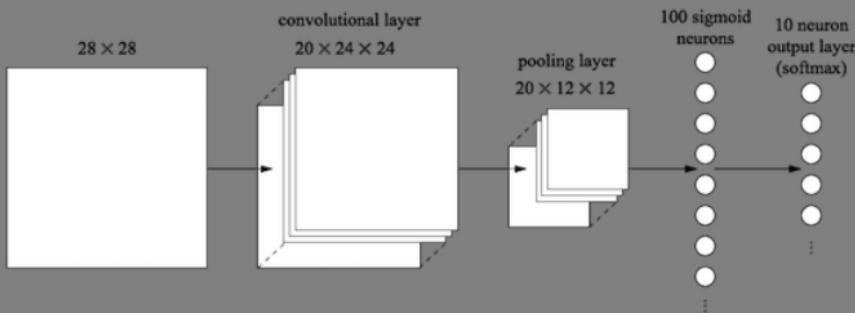


- One category per output neuron
- Ex: bus, car, truck, etc.

Source: M. Nielsen - Neural Networks and Deep Learning

# Many possible architectures for convolutional NN

54 / 60



- Ex: Add another hidden layer to summarize information further, before classification

Source: M. Nielsen - Neural Networks and Deep Learning

- In all the NN seen so far, all neurons take input that depends on the current sample forward-propagated
- In Recurrent NN, samples are submitted in sequences
- Some neurons is connected to previous samples
- What is this model aimed for?
- Language processing, Speach recognition
- Zaremba (NUY) and Sutskever (Google), “Learning to execute”
  - Their NN takes the words, one by one, of a (very simple) python scripts
  - It learns to predict the output!

- In all the NN seen so far, all neurons take input that depends on the current sample forward-propagated
- In Recurrent NN, samples are submitted in sequences
- Some neurons are connected to previous samples
- What is this model aimed for?
- Language processing, Speech recognition
- Zaremba (NYU) and Sutskever (Google), “Learning to execute”
  - Their NN takes the words, one by one, of a (very simple) python scripts
  - It learns to predict the output!

- Generative models
- Training: Find the weights submitting many images of cats
- Use: Give it a random input and get a synthetic cat image
- “Few” neurons in the hidden layer. Why?
- Serious use: drug discovery, music generation

In this lesson

- Structure of NNs
- Training (backpropagation)
- Design choices and hyper-parameters

In next lesson

- Random Forests

-  Andrea Araldo, György Dán, and Dario Rossi, *Caching Encrypted Content via Stochastic Cache Partitioning*, IEEE/ACM Transactions on Networking **26** (2018), no. 1, 548–561.
-  Christopher M. Bishop, *Pattern Recognition and Machine Learning*, 2006.
-  Aurélien Geron, *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*, O'Reilly, 2019.
-  Vivienne Sze, Yu Hsin Chen, Tien Ju Yang, and Joel S. Emer, *Efficient Processing of Deep Neural Networks: A Tutorial and Survey*, Proceedings of the IEEE **105** (2017), no. 12, 2295–2329.
-  Andrew Senior, *An Empirical Study of the Learning Rates in Deep Neural Networks for Speech Recognition*, IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

-  Leslie N. Smith, *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*, Tech. report, US Naval Research Laboratory, 2018.
-  D. S. Taimanov, *Network disruption prediction based on neural networks*, 2017.

- Convolutional Neural Networks for Computer Vision (Ch.11 of [Ger19])
- Christopher M. Bishop. Pattern Recognition and Machine Learning, Springer - Sections 5.1, 5.2, 5.3
- <http://neuralnetworksanddeeplearning.com>