

# Introduction to Deep Learning

## Bootcamp IID 2025

Jonas Ngnawé

{jonas.ngnawe.1}@ulaval.ca

07 mai 2025

## **1** Introduction

## **2** Neural Networks

## **3** Training

## **4** Overfitting

# Who Am I?



**Jonas Ngnawé**



Ph.D. Student at Université Laval  
(IID & Mila)

[jonasngnawe](#)

Slides Credit: Frédéric Paradis, Mathieu Godbout et Alexandre Bouras

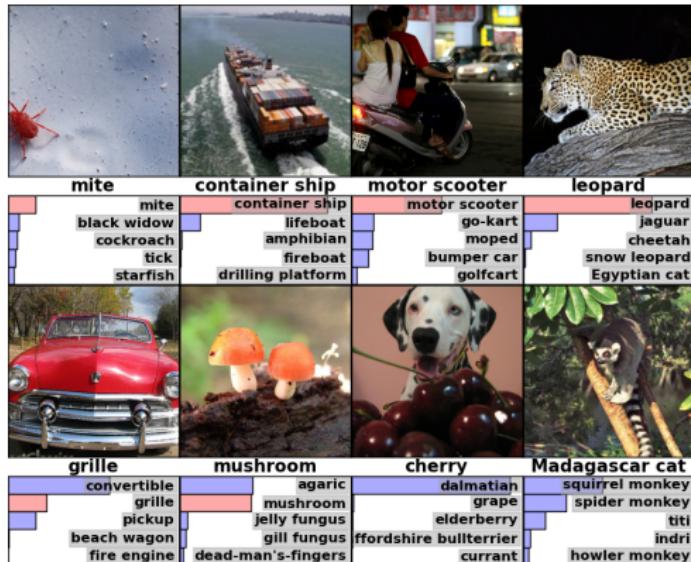
## 1 Introduction

## 2 Neural Networks

## 3 Training

## 4 Overfitting

# Deep Neural Networks



[Krizhevsky et al. 2012, "ImageNet Classification with Deep Convolutional Neural Networks"]

# Deep Neural Networks

The image shows a user interface for a deep learning model. At the top, there are four small images with their predicted labels below them:

- A red mite: mite, black widow, cockroach, tick, starfish
- A container ship: container ship, lifeboat, amphibian, fireboat, drilling platform
- A motor scooter: motor scooter, go-kart, moped, bumper car, golfcart
- A leopard: leopard, jaguar, cheetah, snow leopard, Egyptian cat

Below these are two more images with labels:

- A red classic car: grille, convertible, grille, pickup, beach wagon, fire engine
- Two orange mushrooms: mushroom, agaric, mushroom, jelly fungus, gill fungus, dead-man's-fingers

At the bottom, there is a text input field with the placeholder "Deep learning is awesome!" and a translation section:

DÉTECTOR LA LANGUE ANGLAIS FRANÇAIS ARABE ▾ FRANÇAIS ANGLAIS ARABE ▾

Deep learning is awesome! × L'apprentissage en profondeur est génial! ★

25 / 5000

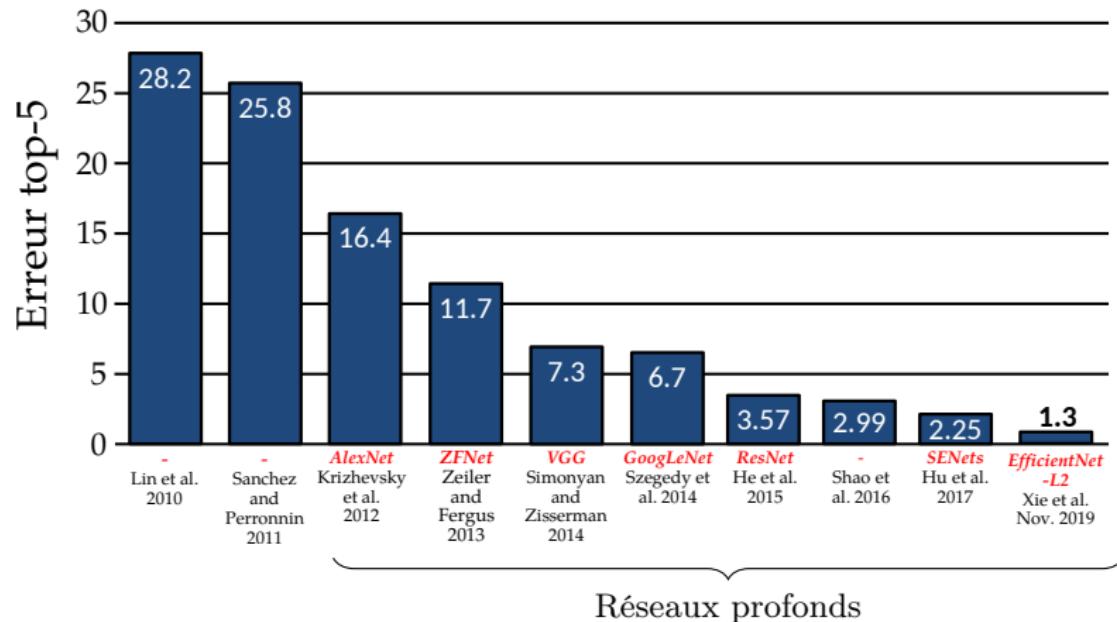
grille, convertible, agaric, dalmatian, squirrel monkey, grape, spider monkey, elderberry, titi, ffordshire bullterrier, currant, howler monkey, indri

[Krizhevsky et al. 2012, "ImageNet Classification with Deep Convolutional Neural Networks"]

# Computer Vision

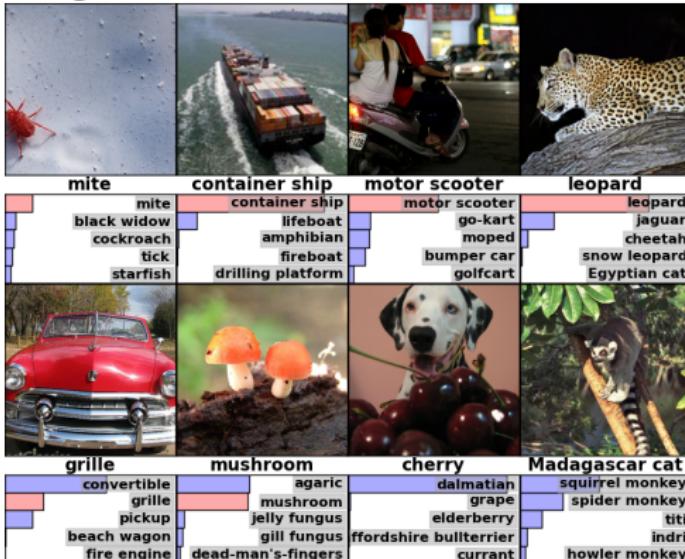
## Large Scale Visual Recognition Challenge (LSVRC)

Image classification challenge on the 1,000 image classes of ImageNet



# Computer Vision

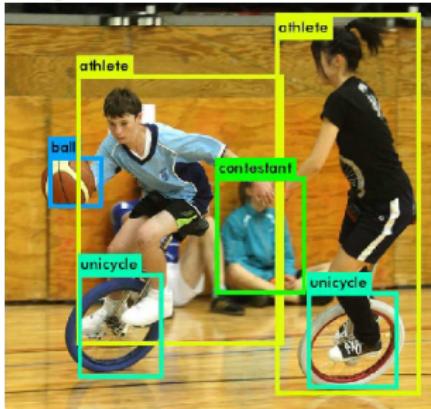
## ■ Image classification



[Krizhevsky et al. 2012, "ImageNet Classification with Deep Convolutional Neural Networks"]

# Computer Vision

- Image classification
- Object detection



[Redmon and Farhadi 2017, "YOLO9000: better, faster, stronger"]

# Computer Vision

- Image classification
- Object detection
- Image segmentation



[Cordts et al. 2016, "The cityscapes dataset for semantic urban scene understanding"]

# Computer Vision

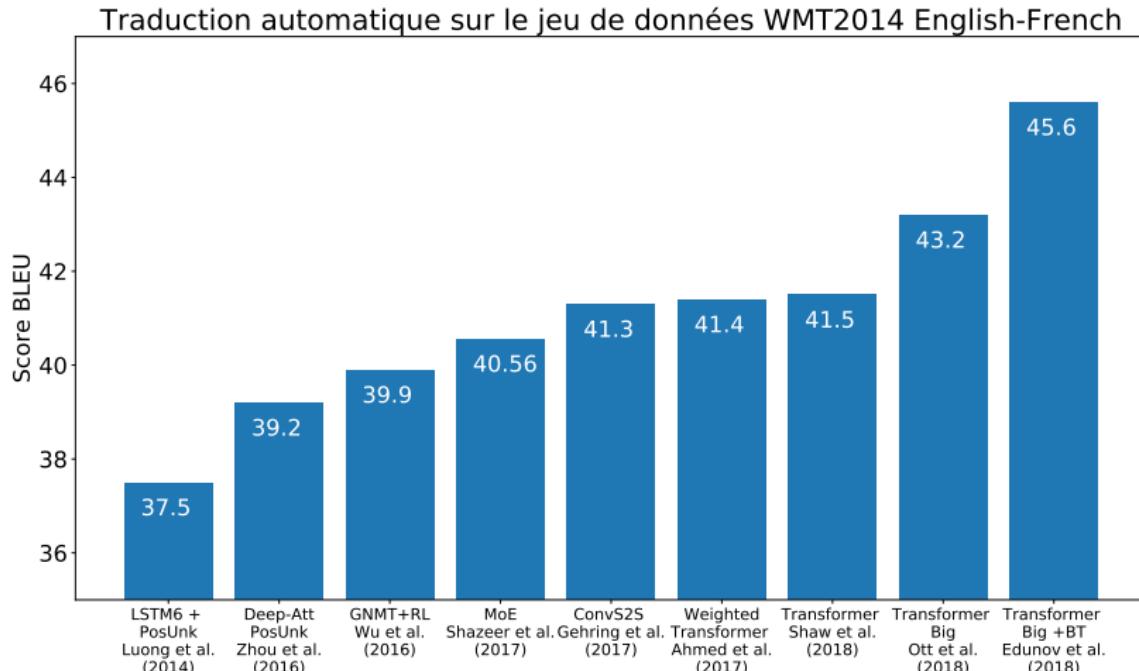
- Image classification
- Object detection
- Image segmentation
- Image generation



# Computer Vision

- Image classification
- Object detection
- Image segmentation
- Image generation
- etc.

# Natural Language Processing (NLP)



<https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french>

# Natural Language Processing

## ■ Translation

The screenshot shows a translation interface with two main sections. The left section is for input, and the right section is for output. Both sections have language selection menus at the top.

**Input Section:**

- Label: DÉTECTOR LA LANGUE
- Selected Language: ANGLAIS
- Text Input: Deep learning is awesome!
- Close button: X
- Bottom controls: A microphone icon, a speaker icon, and a progress bar indicating 25 / 5000.

**Output Section:**

- Label: FRANÇAIS
- Selected Language: FRANÇAIS
- Text Output: L'apprentissage en profondeur est génial! ☆
- Bottom controls: A microphone icon, a speaker icon, and a set of icons for copy, edit, and share.

# Natural Language Processing

- Translation
- Text classification (e.g. topic, sentiment)



John Doe

★★★★★ **Awesome product**

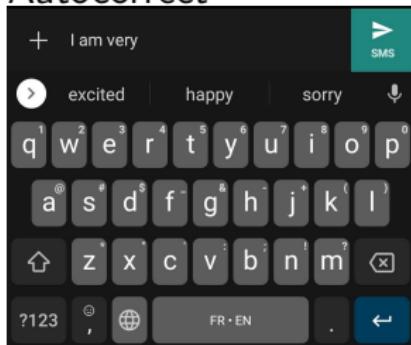
Reviewed in Canada on May 04, 2021

This is an awesome product. It couldn't be better!!!

16 people found this helpful

# Natural Language Processing

- Translation
- Text classification (e.g. topic, sentiment)
- Autocorrect



# Natural Language Processing

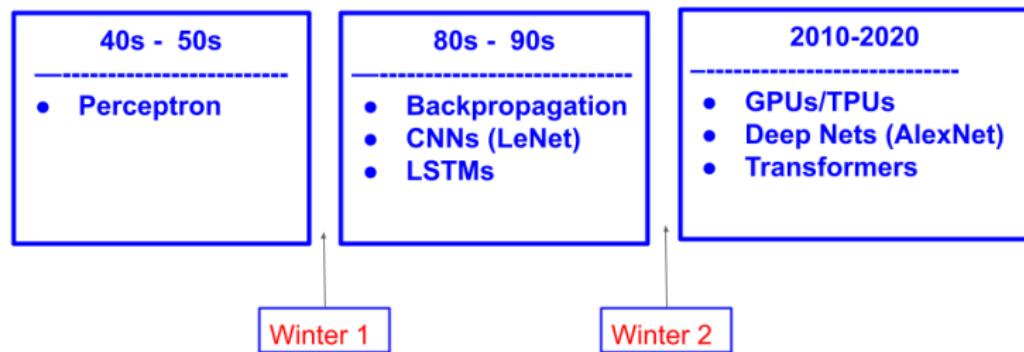
- Translation
- Text classification (e.g. topic, sentiment)
- Autocorrect
- Chatbot



# Natural Language Processing

- Translation
- Text classification (e.g. topic, sentiment)
- Autocorrect
- Chatbot
- etc.

# History Overview



## **1** Introduction

## **2** Neural Networks

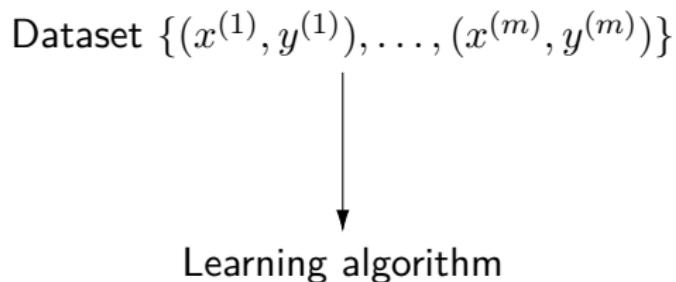
## **3** Training

## **4** Overfitting

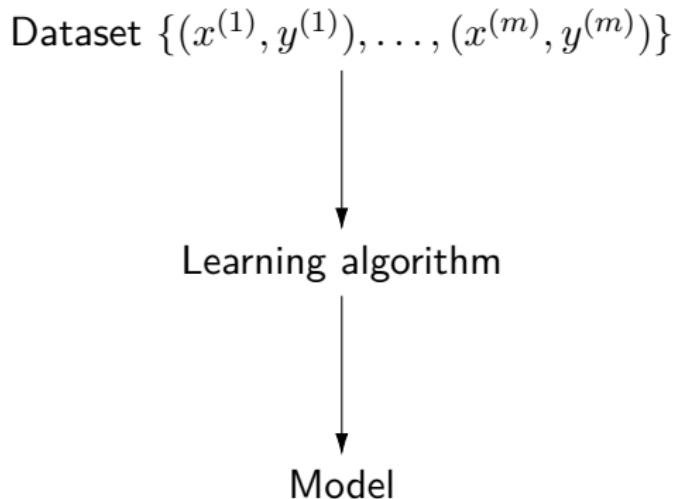
# Machine Learning

Dataset  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

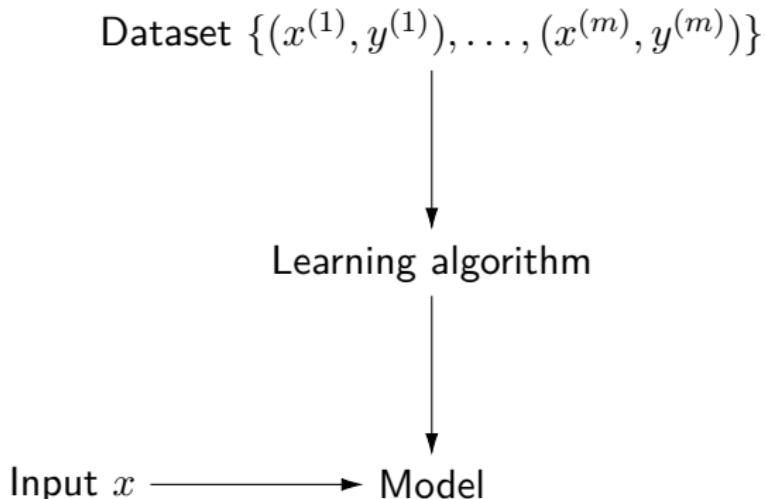
# Machine Learning



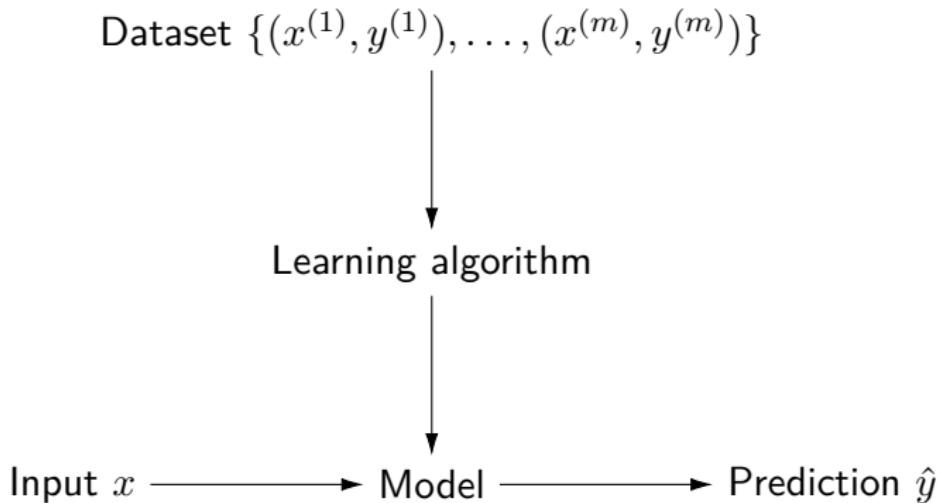
# Machine Learning



# Machine Learning



# Machine Learning

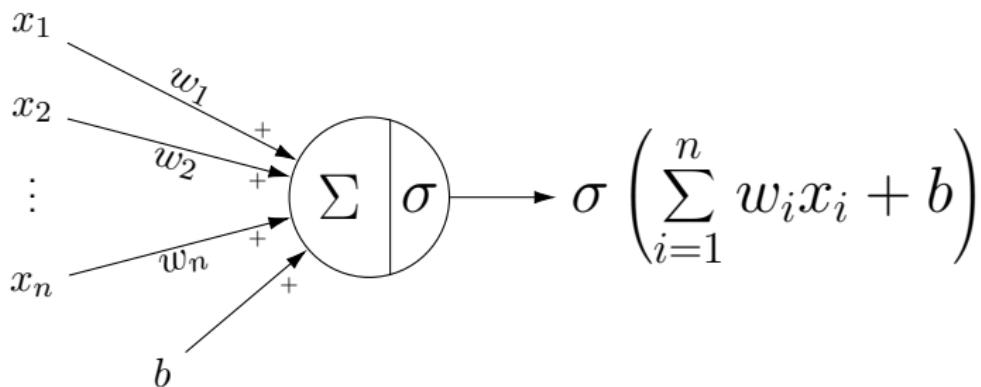


# The Basis of Neural Networks: The Neuron

Let  $x = (x_1, x_2, \dots, x_n)$  be  $n$  features (or variables in statistics).

# The Basis of Neural Networks: The Neuron

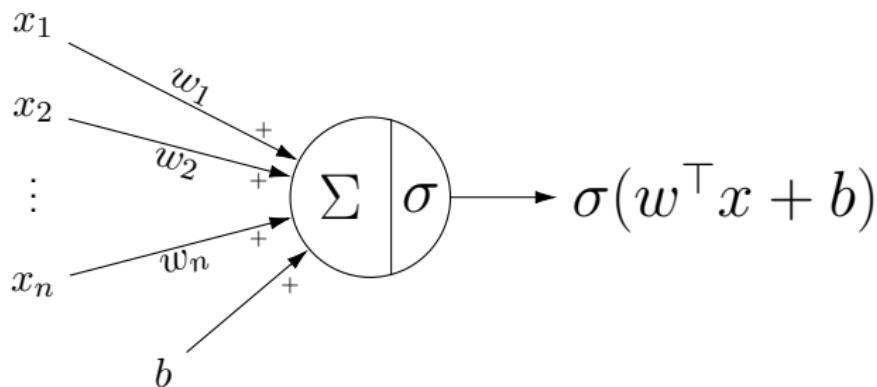
Let  $x = (x_1, x_2, \dots, x_n)$  be  $n$  features (or variables in statistics).  
A neuron computes the following.



where  $w$  are called the weights of the neuron and  $\sigma(\cdot)$  is a non-linear activation function.

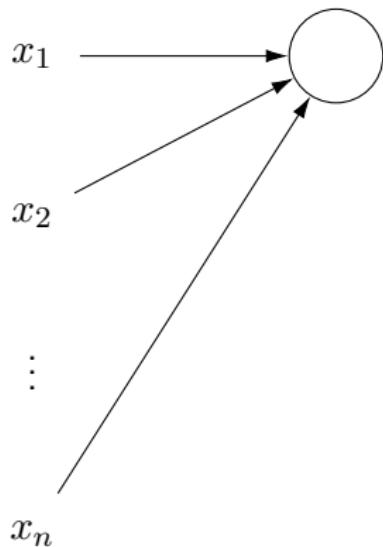
# The Basis of Neural Networks: The Neuron

Let  $x = (x_1, x_2, \dots, x_n)$  be  $n$  features (or variables in statistics). A neuron computes the following.

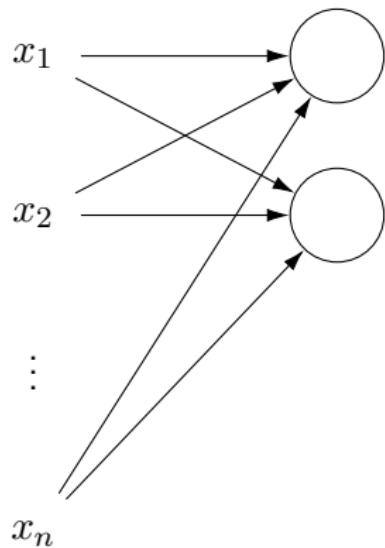


where  $w$  are called the weights of the neuron and  $\sigma(\cdot)$  is a non-linear activation function.

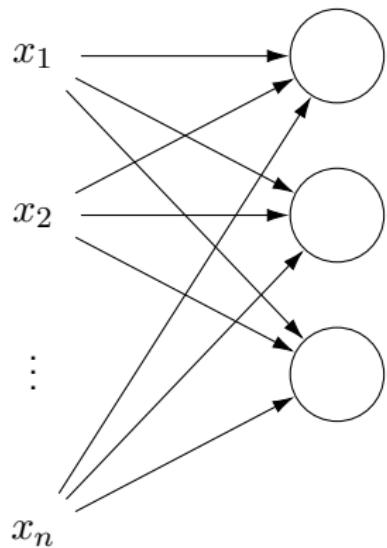
# Neural Network



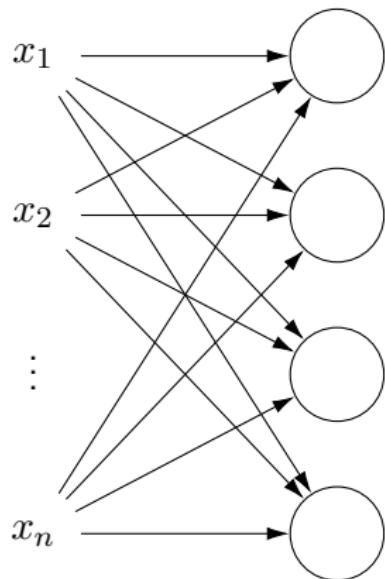
# Neural Network



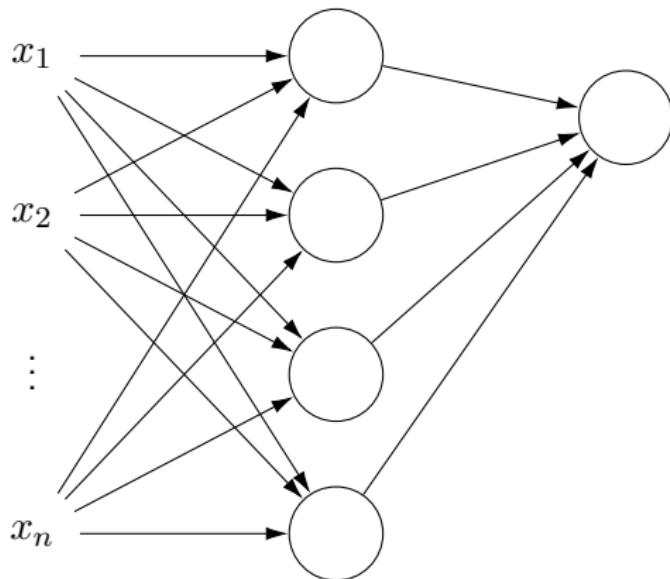
# Neural Network



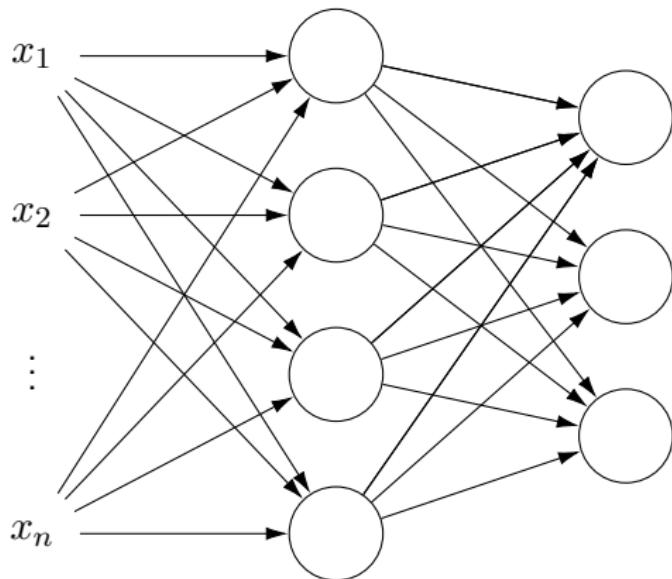
# Neural Network



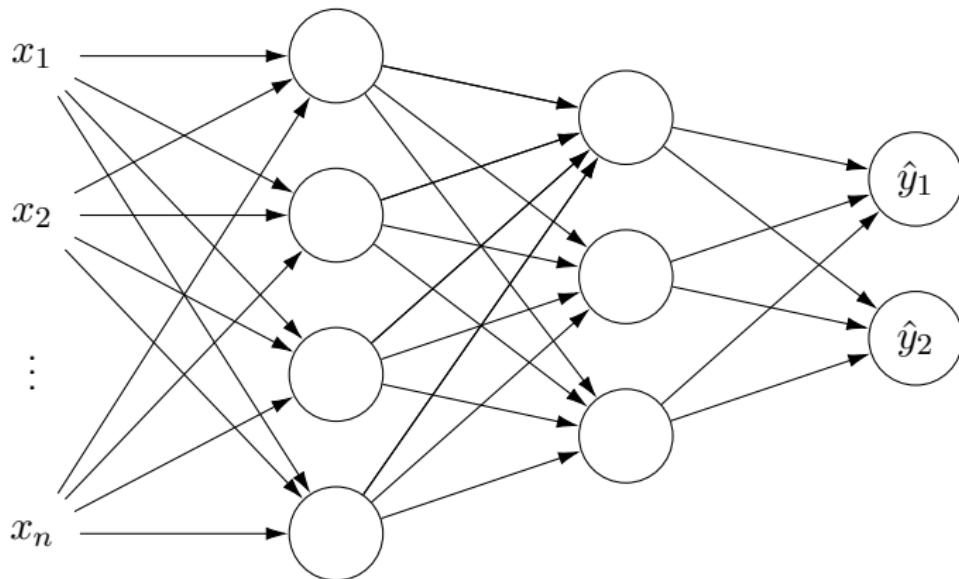
# Neural Network



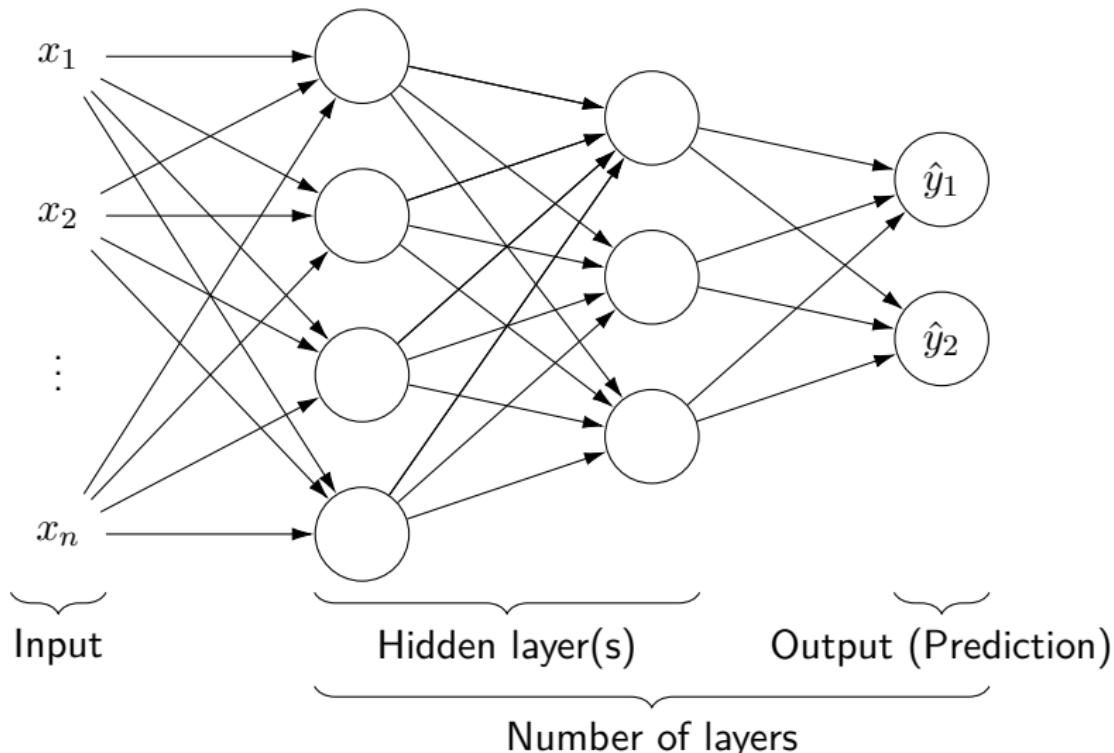
# Neural Network



# Neural Network



# Neural Network



$$y = f_L \circ f_{L-1} \circ \dots \circ f_2 \circ f_1(x), \quad f_\ell = \sigma(W_\ell f_{\ell-1} + b_\ell), \quad f_0 = x$$

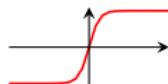
## Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

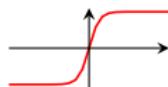
- $\tanh(z)$



# Activation Functions

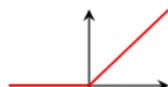
Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

- $\tanh(z)$



- The ReLU function:

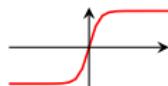
$$\text{ReLU}(z) = \max(0, z)$$



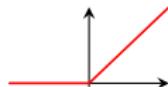
# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

- $\tanh(z)$

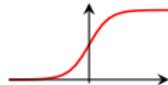


- The ReLU function:



- The logistic function (also called sigmoid function):

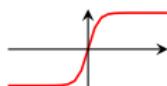
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



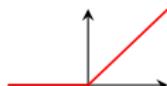
# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

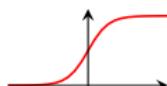
- $\tanh(z)$



- The ReLU function:



- The logistic function (also called sigmoid function):

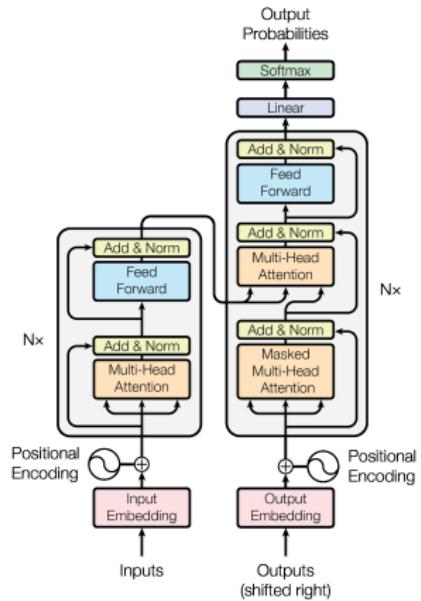
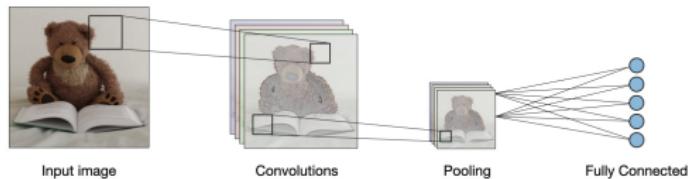


- The softmax function:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^m \exp(z_j)}$$

for  $z = Wx + b \in \mathbb{R}^m$ .

# Other Architectures



## Loss Function

Measure of error between a prediction  $f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

## Loss Function

Measure of error between a prediction  $f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Mean squared error for regression
  - $y, f(x) \in \mathbb{R}$

$$\mathcal{L}_{\text{MSE}}(f(x), y) = (y - f(x))^2$$

# Loss Function

Measure of error between a prediction  $f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Mean squared error for regression
  - $y, f(x) \in \mathbb{R}$

$$\mathcal{L}_{\text{MSE}}(f(x), y) = (y - f(x))^2$$

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )
  - $y \in \mathcal{Y}$
  - $f(x) = [f(x)_1, \dots, f(x)_C] = \text{softmax}([z_1, \dots, z_C])$

# Loss Function

Measure of error between a prediction  $f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Mean squared error for regression

- $y, f(x) \in \mathbb{R}$

$$\mathcal{L}_{\text{MSE}}(f(x), y) = (y - f(x))^2$$

- Cross-entropy loss for classification with  $C$  classes

$$(\mathcal{Y} = \{1, \dots, C\})$$

- $y \in \mathcal{Y}$

- $f(x) = [f(x)_1, \dots, f(x)_C] = \text{softmax}([z_1, \dots, z_C])$

$$\mathcal{L}_{\text{CE}}(f(x), y) = -\log f(x)_y$$

# Loss Function

Measure of error between a prediction  $f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Mean squared error for regression

- $y, f(x) \in \mathbb{R}$

$$\mathcal{L}_{\text{MSE}}(f(x), y) = (y - f(x))^2$$

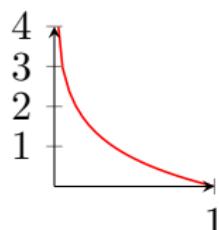
- Cross-entropy loss for classification with  $C$  classes

$$(\mathcal{Y} = \{1, \dots, C\})$$

- $y \in \mathcal{Y}$

- $f(x) = [f(x)_1, \dots, f(x)_C] = \text{softmax}([z_1, \dots, z_C])$

$$\mathcal{L}_{\text{CE}}(f(x), y) = -\log f(x)_y$$



## **1** Introduction

## **2** Neural Networks

## **3** Training

## **4** Overfitting

## Training Objective

Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

$$\mathcal{L}(f(x), y)$$

## Training Objective

Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

- The only thing we control are the parameters  $\theta$ .

$$\mathcal{L}(f(x; \theta), y)$$

## Training Objective

Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

- The only thing we control are the parameters  $\theta$ .

$$\mathbb{E}_{(x,y) \in \mathcal{D}} \mathcal{L}(f(x; \theta), y)$$

## Training Objective

Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

- The only thing we control are the parameters  $\theta$ .

$$J(\theta) = \mathbb{E}_{(x,y) \in \mathcal{D}} \mathcal{L}(f(x; \theta), y)$$

## Training Objective

Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

- The only thing we control are the parameters  $\theta$ .

$$\begin{aligned} J(\theta) &= \mathbb{E}_{(x,y) \in \mathcal{D}} \mathcal{L}(f(x; \theta), y) \\ &\approx \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f(x_i; \theta), y_i) \end{aligned}$$

# Training Objective

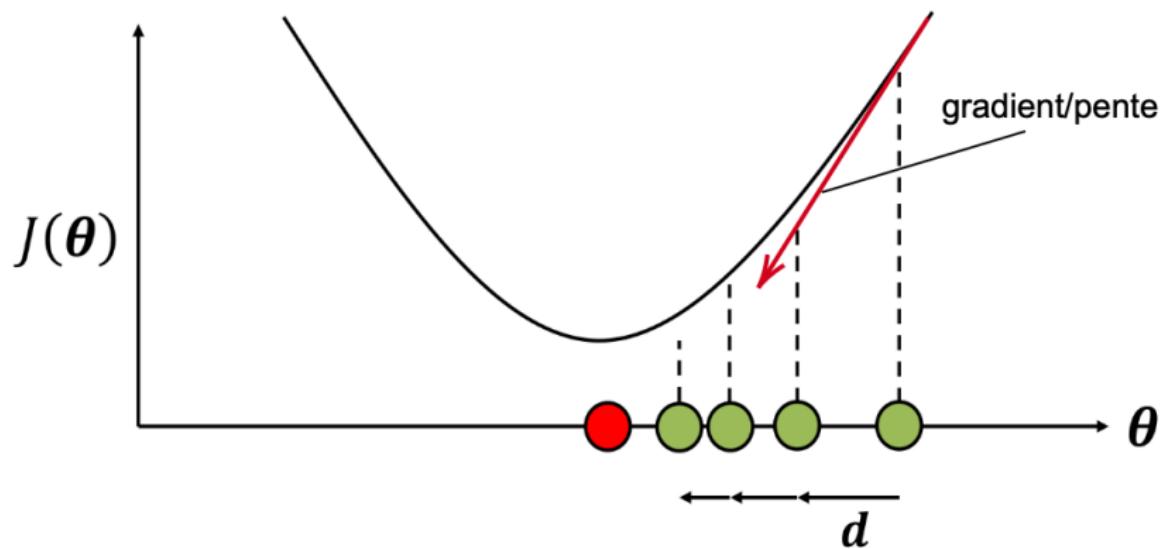
Goal: minimize the loss function for all examples (pairs  $(x, y)$ )!

- The only thing we control are the parameters  $\theta$ .

$$\begin{aligned} J(\theta) &= \underset{(x,y) \in \mathcal{D}}{\mathbb{E}} \mathcal{L}(f(x; \theta), y) \\ &\approx \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f(x_i; \theta), y_i) \end{aligned}$$

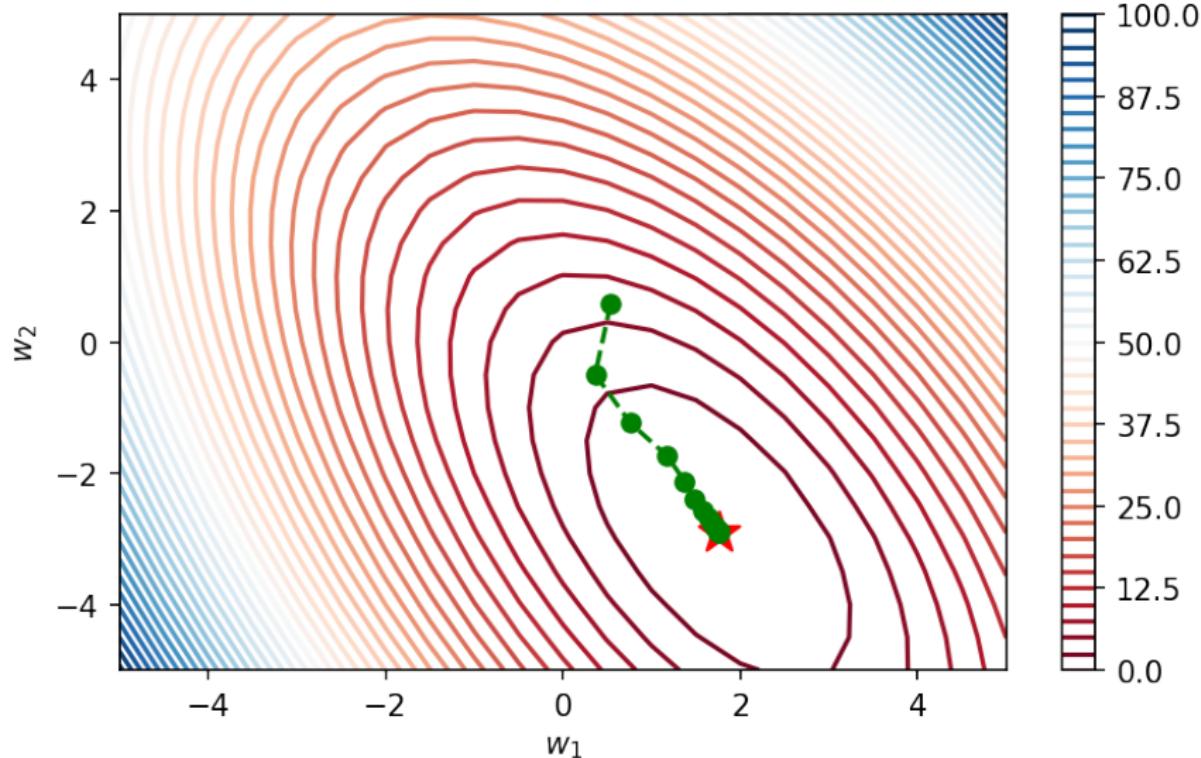
$$\theta^* = \operatorname{argmin}_{\theta \in \Theta} J(\theta)$$

# Gradient Descent

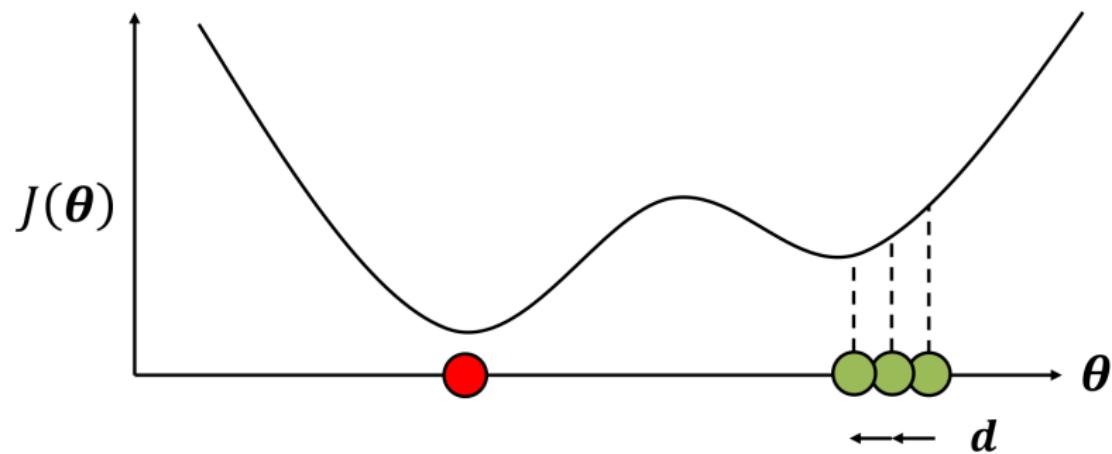


From Université Laval GLO-7030 by Ludovic Trottier

# Gradient Descent



# Gradient Descent For Deep Neural Networks



From Université Laval GLO-7030 by Ludovic Trottier

# Gradient Descent For Deep Neural Networks



Adapted by Philippe Giguère for Université Laval GLO-7030 from Standford CS231N

# Gradient Descent For Deep Neural Networks

Réalité : trouver le fond de la vallée embrumée, à tâtons



## Computation of Gradient

Using "**backpropagation**", we compute the **partial derivative** of **the loss** with respect to **each parameter**. The vector containing all partial derivatives is called the **gradient**.

$$d = \nabla_{\theta} J(\theta) = \left[ \frac{\partial J(\theta)}{\partial \theta_1}, \dots, \frac{\partial J(\theta)}{\partial \theta_n} \right]$$

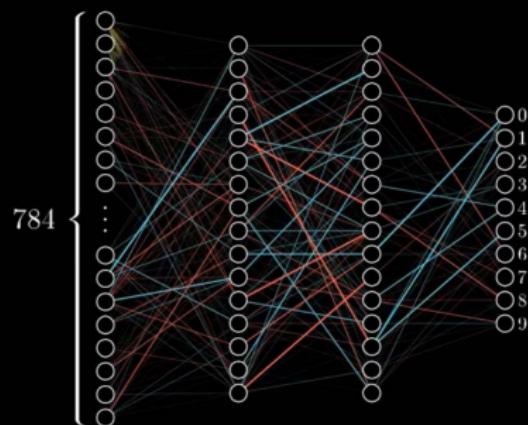
To update the parameters:

$$\theta \leftarrow \theta - \epsilon d$$

$\epsilon$  is called the learning rate.

# Gradient

Training in  
progress. . .



Extrait de <https://youtu.be/IHZwWFHwa-w> 3Blue1Brown

# Optimization Algorithms

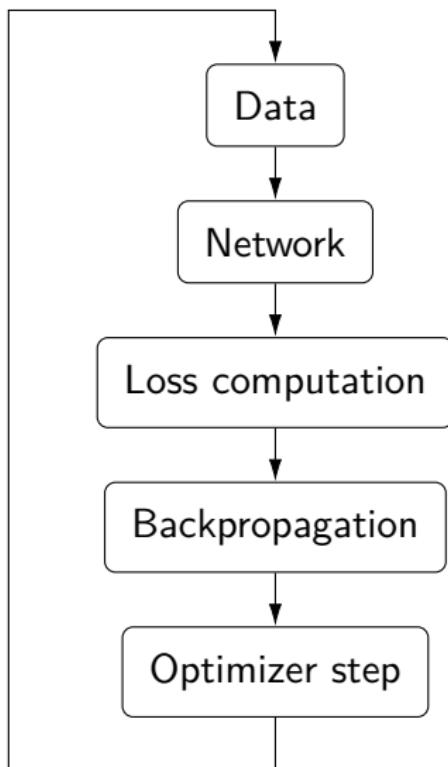
Simplest update rule:

$$\theta \leftarrow \theta - \epsilon d$$

Many types exist:

- **SGD**
- SGD with momentum
- SGD with momentum Nesterov
- Adagrad
- RMSprop
- **Adam**

# Training Procedure



# Training Procedure

**procedure** TRAIN( $f(\cdot; \theta)$ ,  $S$ )

**input:** Neural network  $f$  parameterized by  $\theta$

**input:** Dataset  $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$

**end procedure**

# Training Procedure

```
procedure TRAIN( $f(\cdot; \theta)$ ,  $S$ )
    input: Neural network  $f$  parameterized by  $\theta$ 
    input: Dataset  $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$ 
    for  $n$  epochs do
        end for
end procedure
```

## Training Procedure

**procedure** TRAIN( $f(\cdot; \theta)$ ,  $S$ )

**input:** Neural network  $f$  parameterized by  $\theta$

**input:** Dataset  $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$

**for**  $n$  epochs **do**

        Let  $S' = S$

**while**  $S' \neq \emptyset$  **do**

            Draw a batch  $B \subseteq S'$  of  $b$  examples.

**end while**

**end for**

**end procedure**

# Training Procedure

**procedure** TRAIN( $f(\cdot; \theta)$ ,  $S$ )

**input:** Neural network  $f$  parameterized by  $\theta$

**input:** Dataset  $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$

**for**  $n$  epochs **do**

        Let  $S' = S$

**while**  $S' \neq \emptyset$  **do**

            Draw a batch  $B \subseteq S'$  of  $b$  examples.

$$\ell = \frac{1}{b} \sum_{(x,y) \in B} \mathcal{L}(f(x; \theta), y)$$

**end while**

**end for**

**end procedure**

# Training Procedure

**procedure** TRAIN( $f(\cdot; \theta)$ ,  $S$ )

**input:** Neural network  $f$  parameterized by  $\theta$

**input:** Dataset  $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$

**for**  $n$  epochs **do**

        Let  $S' = S$

**while**  $S' \neq \emptyset$  **do**

            Draw a batch  $B \subseteq S'$  of  $b$  examples.

$$\ell = \frac{1}{b} \sum_{(x,y) \in B} \mathcal{L}(f(x; \theta), y)$$

$$d = \nabla_{\theta} \ell$$

            Update  $\theta$  with  $d$  using chosen optimizer

**end while**

**end for**

**end procedure**

**1** Introduction

**2** Neural Networks

**3** Training

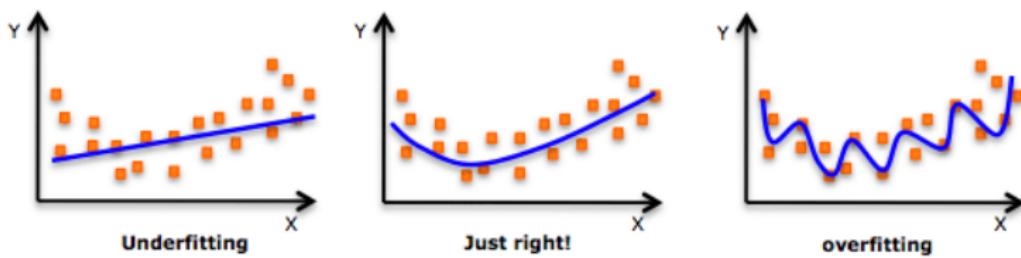
**4** Overfitting

# Neural Networks are Powerful

- Universal Approximation Theorem

# Neural Networks are Powerful

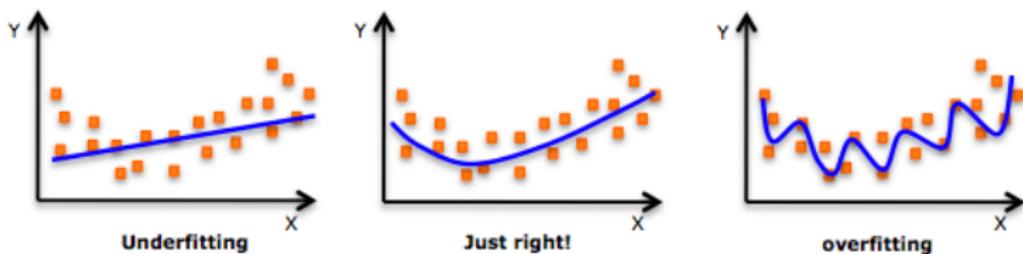
- Universal Approximation Theorem
- ... is a *blessing!*



An example of overfitting, underfitting and a model that's "just right!"

# Neural Networks are Powerful

- Universal Approximation Theorem
- ... is a *blessing!*
- ... and a *curse!*



An example of overfitting, underfitting and a model that's "just right!"

# Neural Networks are Powerful

- Universal Approximation Theorem
- ... is a *blessing*!
- ... and a *curse*!
- How can we avoid overfitting?

# Regularization

Add a penalty term to the loss<sup>1</sup>:

$$\mathcal{L}(f(x), y) = \mathcal{L}_{\text{MLE}}(f(x), y) + \lambda \Omega(f)$$

---

<sup>1</sup>MLE stands for Maximum Likelihood Estimation.

# Regularization

Add a penalty term to the loss<sup>1</sup>:

$$\mathcal{L}(f(x), y) = \mathcal{L}_{\text{MLE}}(f(x), y) + \lambda \Omega(f)$$

- L2 regularization:  $\Omega(f) = \|\theta\|_2^2$ , where  $\theta$  are the learnable parameters of the neural network  $f$ .

---

<sup>1</sup>MLE stands for Maximum Likelihood Estimation.

# Regularization

Add a penalty term to the loss<sup>1</sup>:

$$\mathcal{L}(f(x), y) = \mathcal{L}_{\text{MLE}}(f(x), y) + \lambda \Omega(f)$$

- L2 regularization:  $\Omega(f) = \|\theta\|_2^2$ , where  $\theta$  are the learnable parameters of the neural network  $f$ .
- L1 regularization (or LASSO):  $\Omega(f) = \|\theta\|_1$

---

<sup>1</sup>MLE stands for Maximum Likelihood Estimation.

# Regularization

Add a penalty term to the loss<sup>1</sup>:

$$\mathcal{L}(f(x), y) = \mathcal{L}_{\text{MLE}}(f(x), y) + \lambda \Omega(f)$$

- L2 regularization:  $\Omega(f) = \|\theta\|_2^2$ , where  $\theta$  are the learnable parameters of the neural network  $f$ .
- L1 regularization (or LASSO):  $\Omega(f) = \|\theta\|_1$
- etc.

---

<sup>1</sup>MLE stands for Maximum Likelihood Estimation.

# Dropout

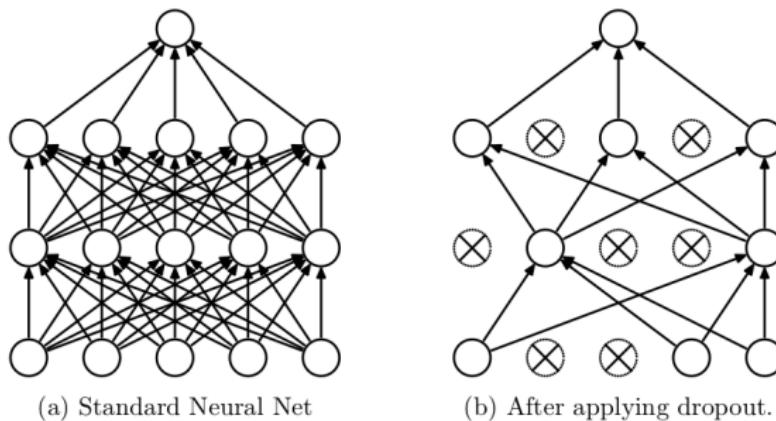
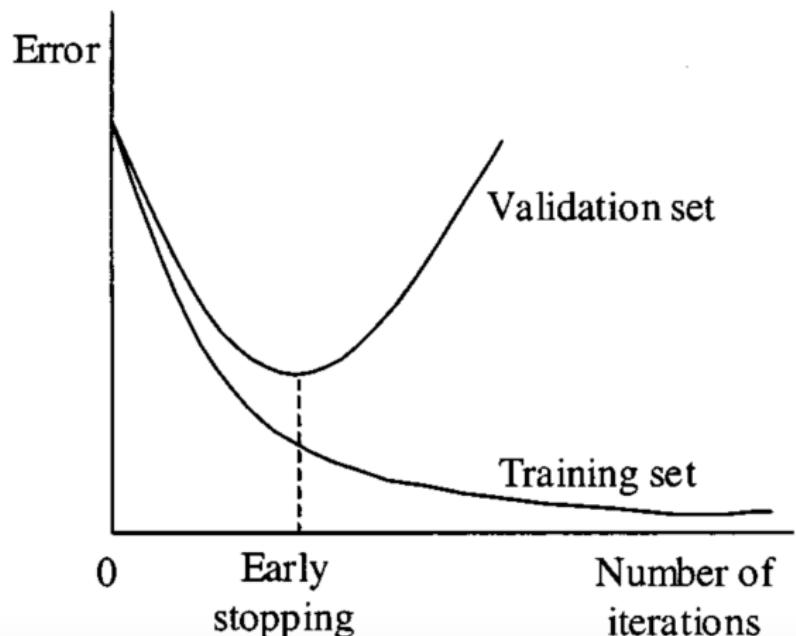


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

## Early Stopping

Extract a validation set  $\mathcal{V}$  from the training set. Use it to assess generalization potential.



## More ways to regularize

- Control the model capacity (e.g.:depth)
- More Data
- Data Augmentation

## Deep Learning Libraries



TensorFlow

## Deep Learning Libraries



TensorFlow



# PyTorch Demo

## Liens de référence

- Cours GLO-4030/7030 Apprentissage par réseaux de neurones profonds:
  - Slides: <https://ulaval-damas.github.io/glo4030/>
  - Laboratoire:  
<https://github.com/ulaval-damas/glo4030-labs>
- Vidéos du cours CS231n:  
<https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk>
- Blogpost: A Recipe for Training Neural Networks. Andrej Karpathy
- Tutoriels et documentation de PyTorch
  - <https://pytorch.org/tutorials/> (pas tout le temps les meilleures pratiques)
  - <https://pytorch.org/docs/stable/index.html>
- Documentation de Poutyne: <https://poutyne.org/>

The End.

Questions?

# Bibliography I

-  Cordts, Marius et al. (2016). "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223.
-  He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.
-  Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*.
-  Redmon, Joseph and Ali Farhadi (2017). "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271.
-  Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

## Bibliography II

-  Szegedy, Christian et al. (2015). “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

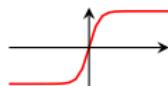
## Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

## Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

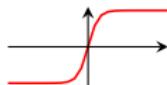
- $\tanh(z)$



## Activation Functions

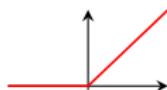
Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

- $\tanh(z)$



- The ReLU function:

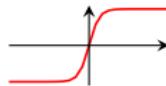
$$\text{ReLU}(z) = \max(0, z)$$



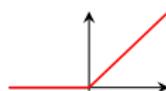
# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

- $\tanh(z)$

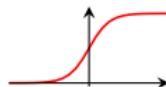


- The ReLU function:



- The logistic function (also called sigmoid function):

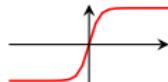
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



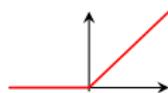
# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

- $\tanh(z)$



- The ReLU function:



- The logistic function (also called sigmoid function):



- The softmax function:

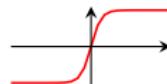
$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^m \exp(z_j)}$$

for  $z = Wx + b \in \mathbb{R}^m$ .

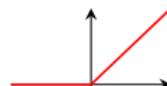
# Activation Functions

Given a non-activated output  $z = w^\top x + b$ , then  $\sigma(z) = \dots$

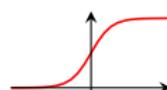
- $\tanh(z)$



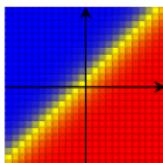
- The ReLU function:



- The logistic function (also called sigmoid function):



- The softmax function:



$$\text{softmax}(z)_1 = \frac{\exp(x)}{\exp(x)+\exp(y)} \text{ for } z = [x, y]$$

## Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Squared error (SE) for regression:

$$\mathcal{L}_{\text{SE}}(\hat{y}, y) = (\hat{y} - y)^2$$

for  $\hat{y}, y \in \mathbb{R}$ .

## Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )

## Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \dots, \hat{y}_C] = \text{softmax}([z_1, \dots, z_C])$

## Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \dots, \hat{y}_C] = \text{softmax}([z_1, \dots, z_C])$

Based on the cross entropy:

$$H(q, p) = - \sum_{c \in \mathcal{Y}} p(c) \log q(c)$$

# Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \dots, \hat{y}_C] = \text{softmax}([z_1, \dots, z_C])$

Based on the cross entropy:

$$H(q, p) = - \sum_{c \in \mathcal{Y}} p(c) \log q(c)$$

where  $p(c) = \mathbb{1}(y = c)$  and  $q(c) = \hat{y}_c$ .

## Loss Function

Measure of error between a prediction  $\hat{y} = f(x)$  and a ground truth  $y$ .

Goal: minimize the loss function!

- Cross-entropy loss for classification with  $C$  classes ( $\mathcal{Y} = \{1, \dots, C\}$ )
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \dots, \hat{y}_C] = \text{softmax}([z_1, \dots, z_C])$

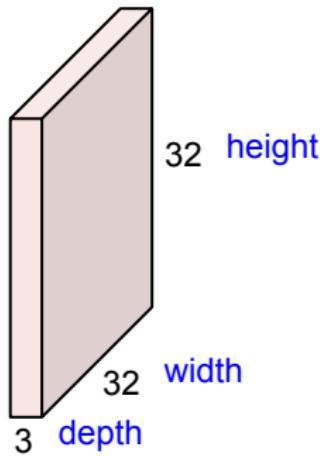
Based on the cross entropy:

$$H(q, p) = - \sum_{c \in \mathcal{Y}} p(c) \log q(c)$$

where  $p(c) = \mathbb{1}(y = c)$  and  $q(c) = \hat{y}_c$ . This simplifies to:

$$\mathcal{L}_{\text{CE}}(\hat{y}, y) = - \sum_{c=1}^C \mathbb{1}(y = c) \log \hat{y}_c = - \log \hat{y}_y$$

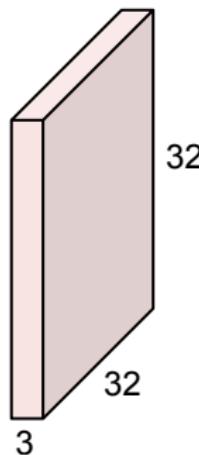
# Convolution Layer



Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer

32x32x3 image

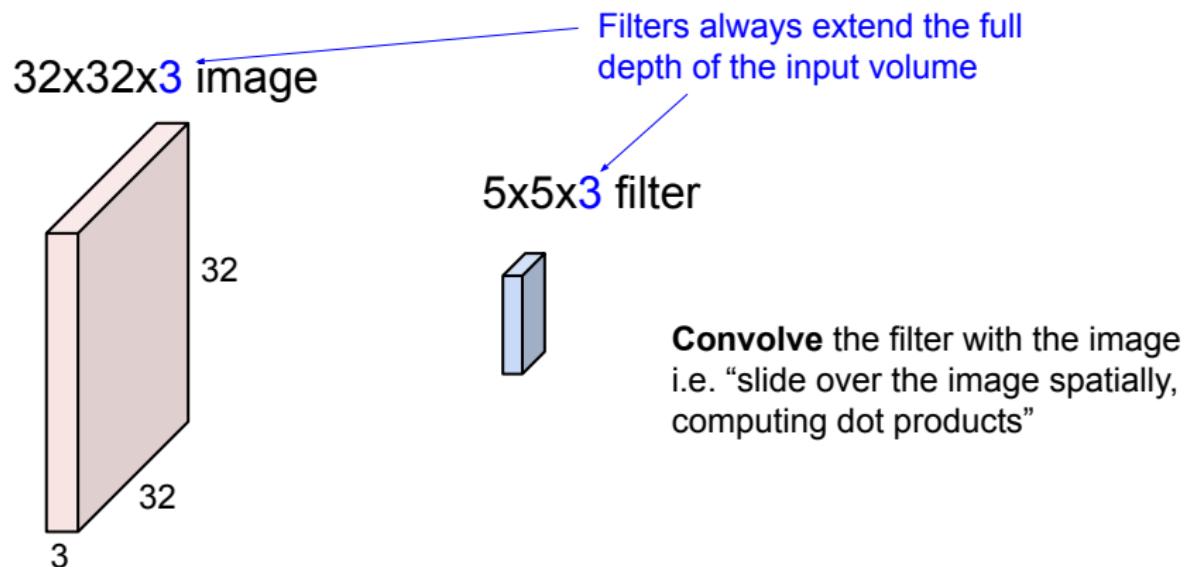


5x5x3 filter

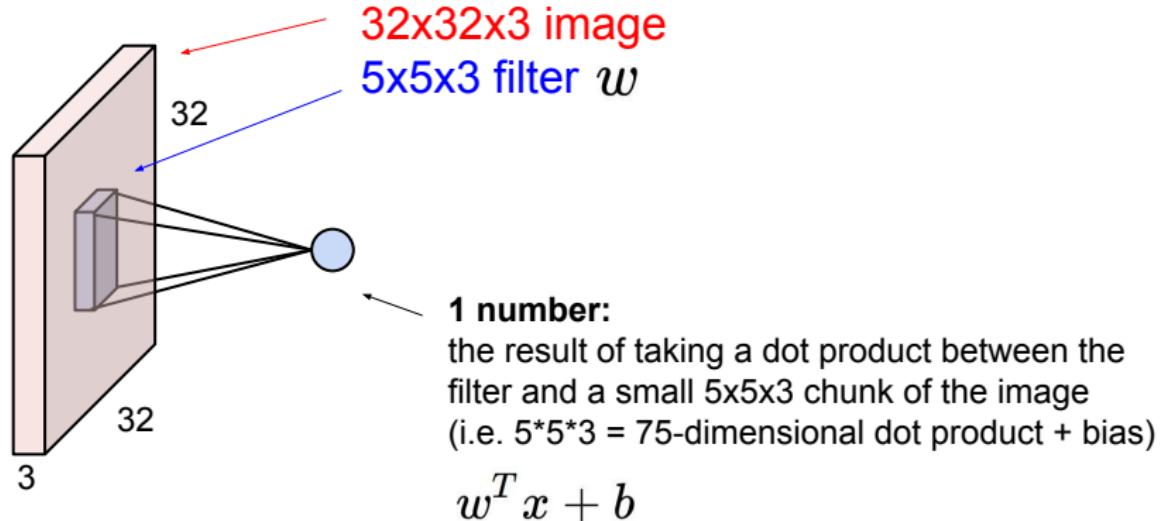


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

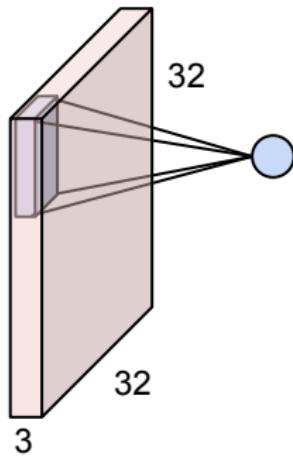


# Convolution Layer



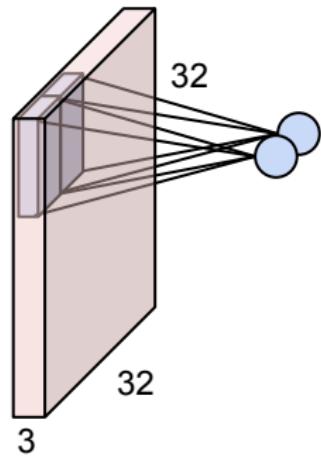
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



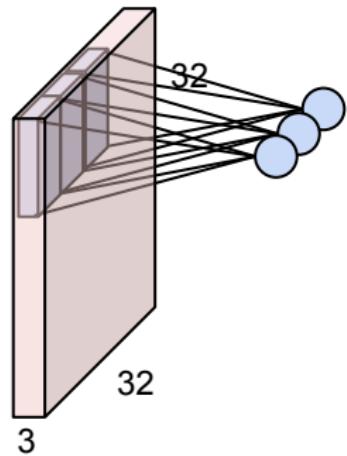
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



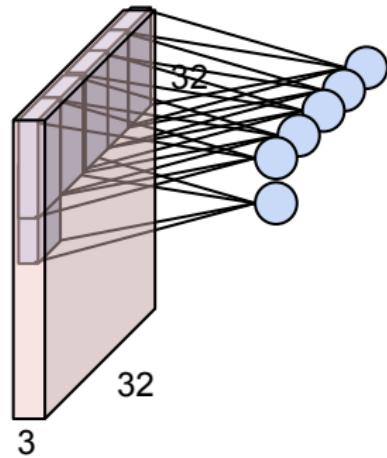
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



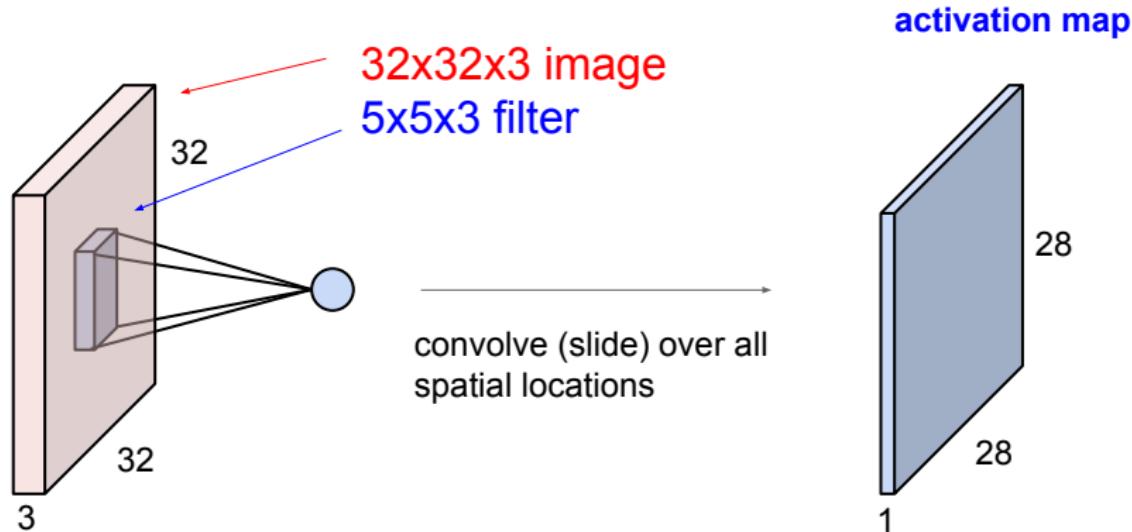
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



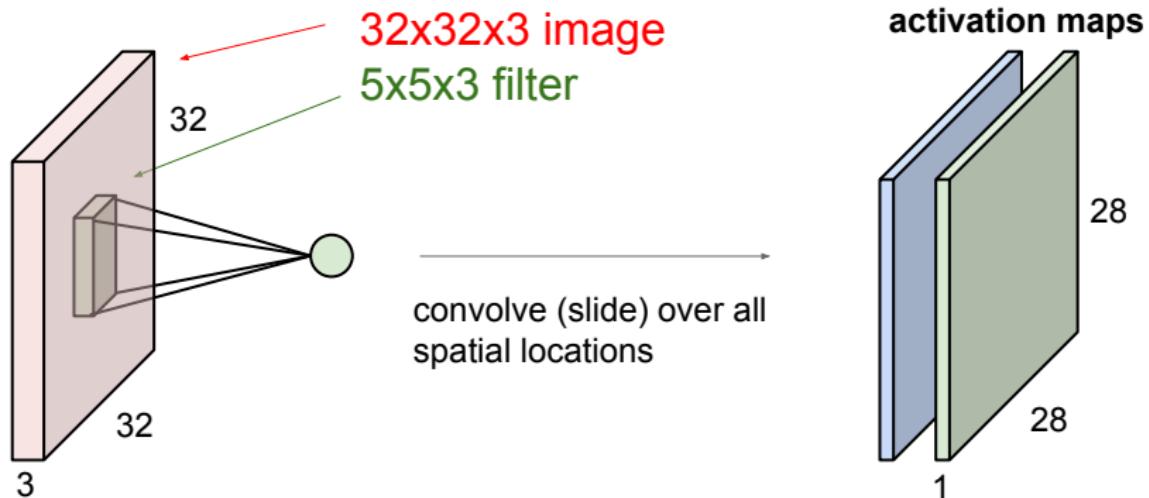
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



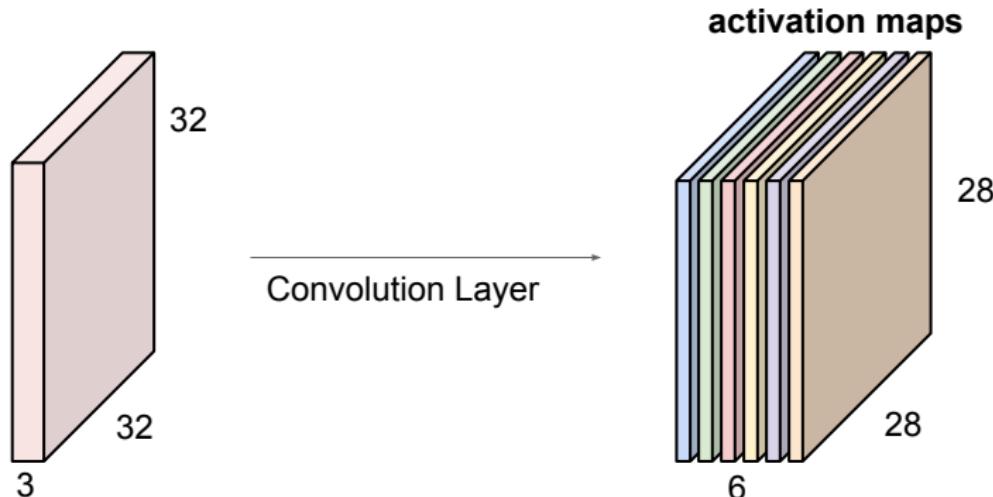
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



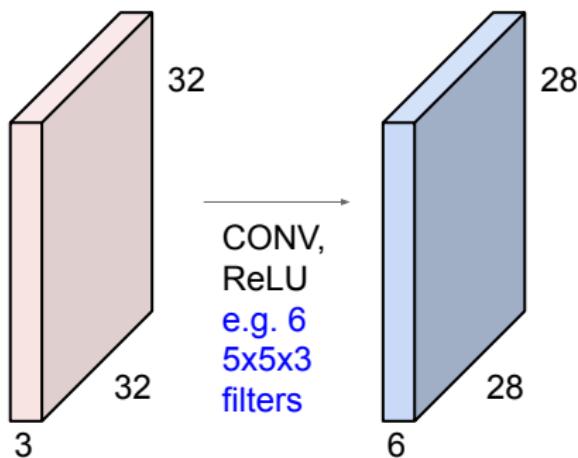
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



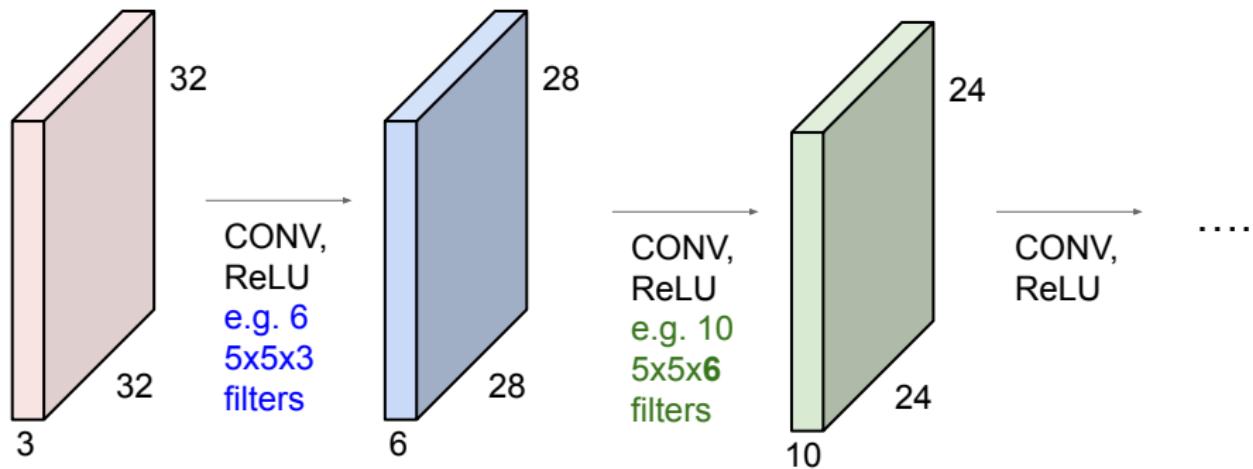
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

# Convolution Layer



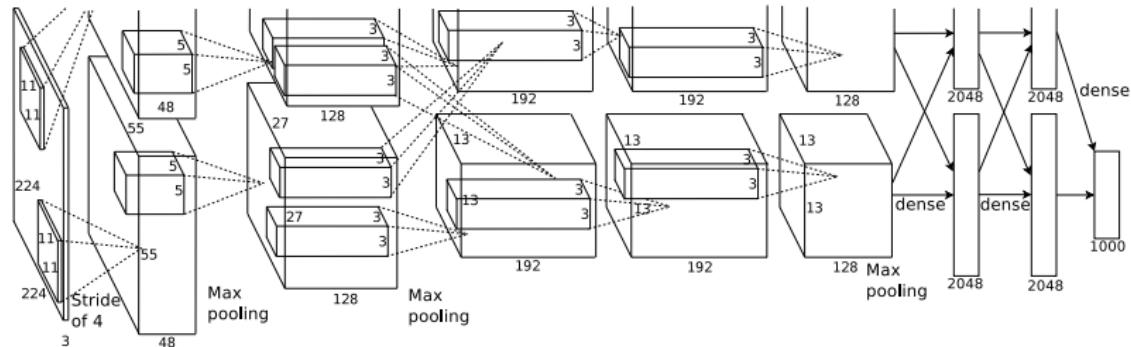
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

## Other Types of Layers

Many types of layers exist. Here is a few.

- Max pooling/average pooling
- Batch normalization
- Dropout

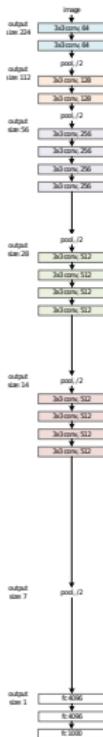
# Deep Neural Network Architectures



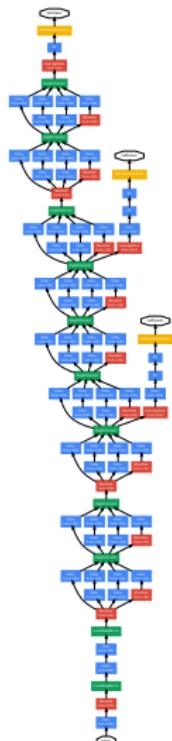
AlexNet

[Krizhevsky et al. 2012, "ImageNet Classification with Deep Convolutional Neural Networks"]

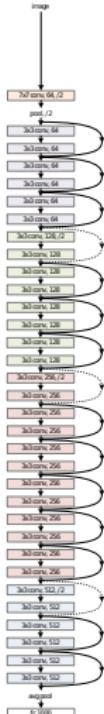
# Deep Neural Network Architectures



VGG



GoogLeNet (or Inception)



ResNet

[Simonyan and Zisserman 2014, "Very deep convolutional networks for large-scale image recognition"]

[Szegedy et al. 2015, "Going deeper with convolutions"]

[He et al. 2016, "Deep residual learning for image recognition"]