Introduction to Deep Learning

Bootcamp IID 2023

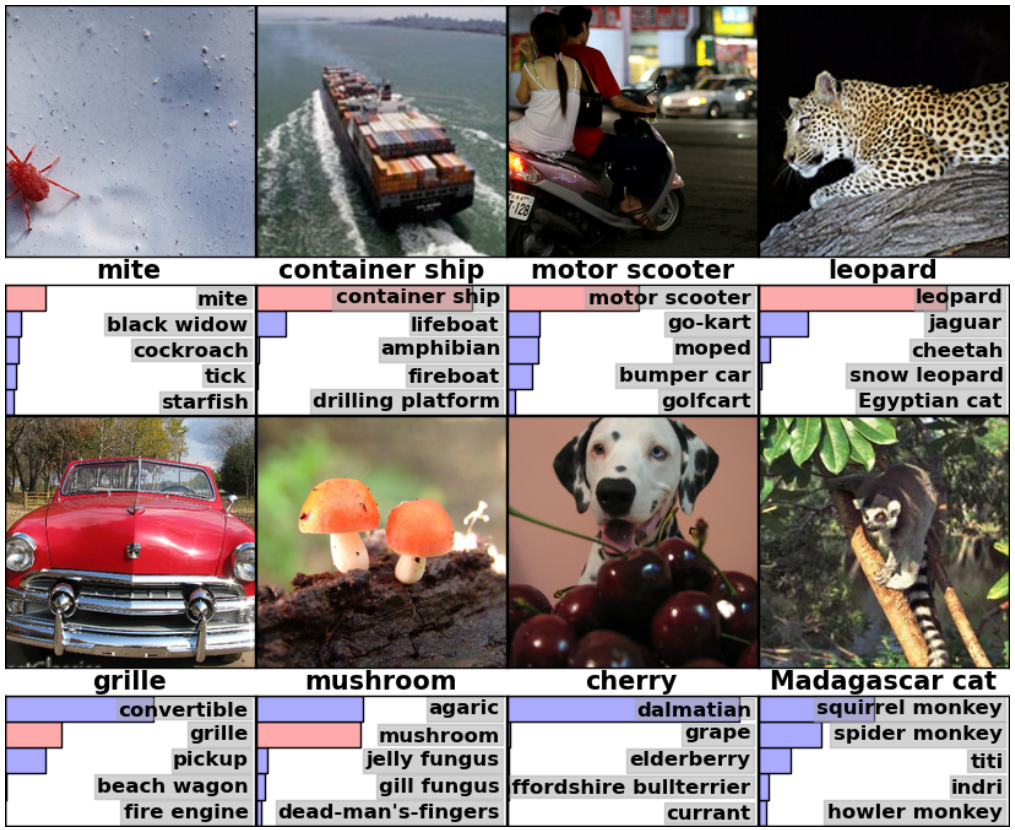
Mathieu Godbout et Alexandre Bouras

{mathieu.godbout.3, alexandre.bouras.1}@ulaval.ca

15 mai 2023

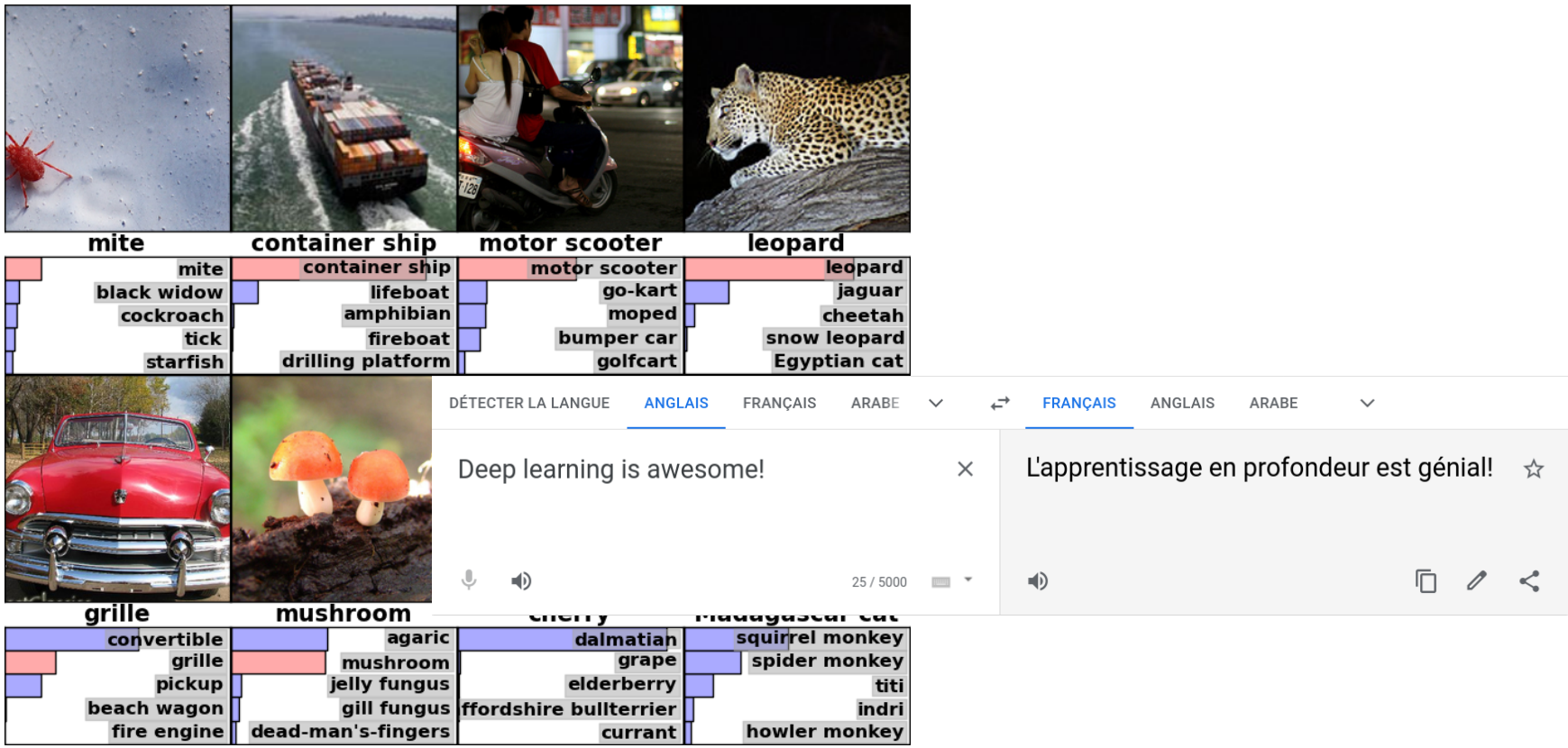
Who Are We?

Deep Neural Networks



[Krizhevsky2012ImageNetCW, Krizhevsky2012ImageNetCW]

Deep Neural Networks

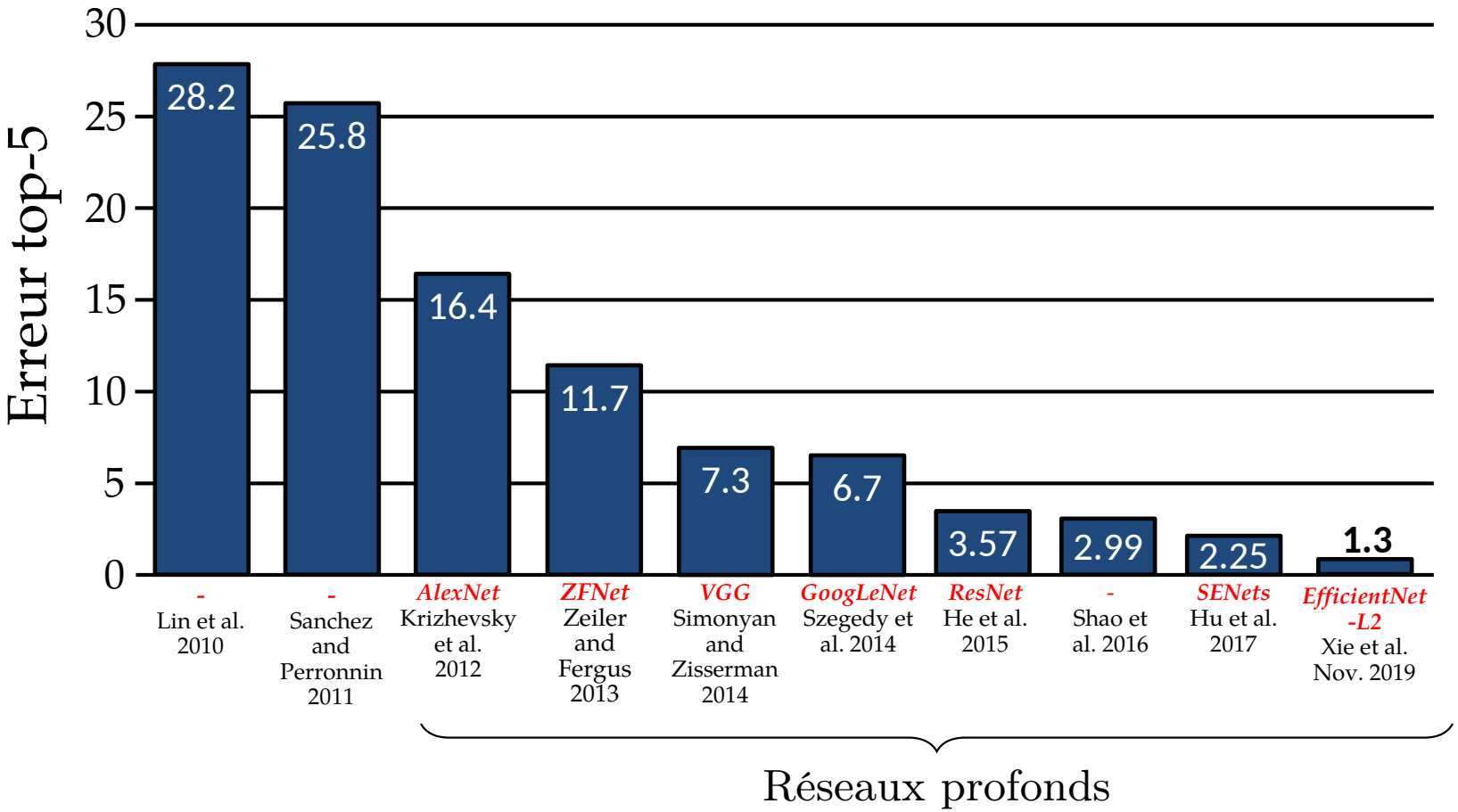


[Krizhevsky2012ImageNetCW, Krizhevsky2012ImageNetCW]

Computer Vision

Large Scale Visual Recognition Challenge (LSVRC)

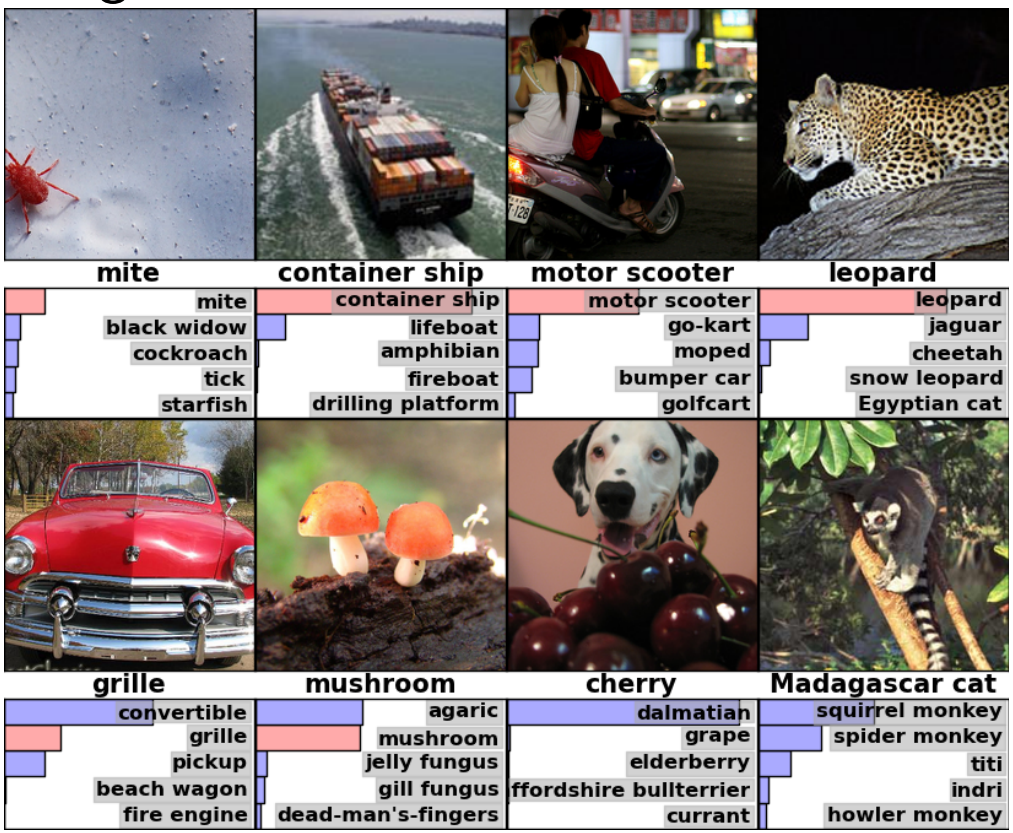
Image classification challenge on the 1,000 image classes of ImageNet



http://www2.ift.ulaval.ca/~pgiguere/cours/DeepLearning/2020/01-Introduction-2020.pdf

Computer Vision

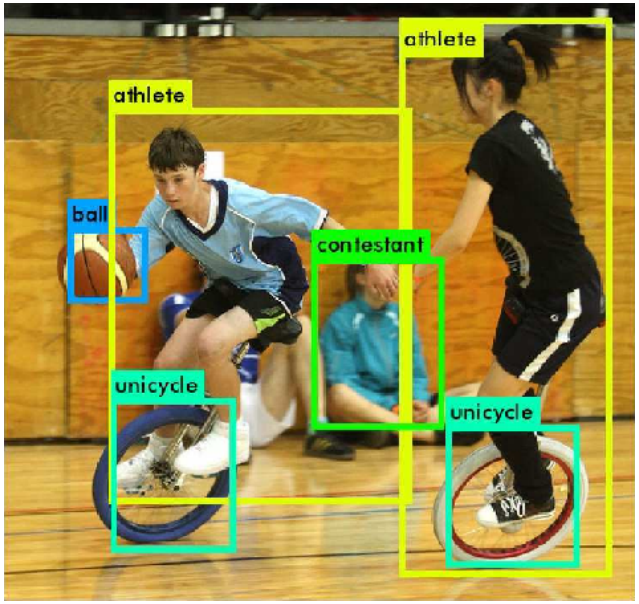
- Image classification



[Krizhevsky2012ImageNetCW, Krizhevsky2012ImageNetCW]

Computer Vision

- Image classification
- Object detection



[redmon2017yolo9000, redmon2017yolo9000]

Computer Vision

- Image classification
- Object detection
- Image segmentation

[cordts2016cityscapes, cordts2016cityscapes]

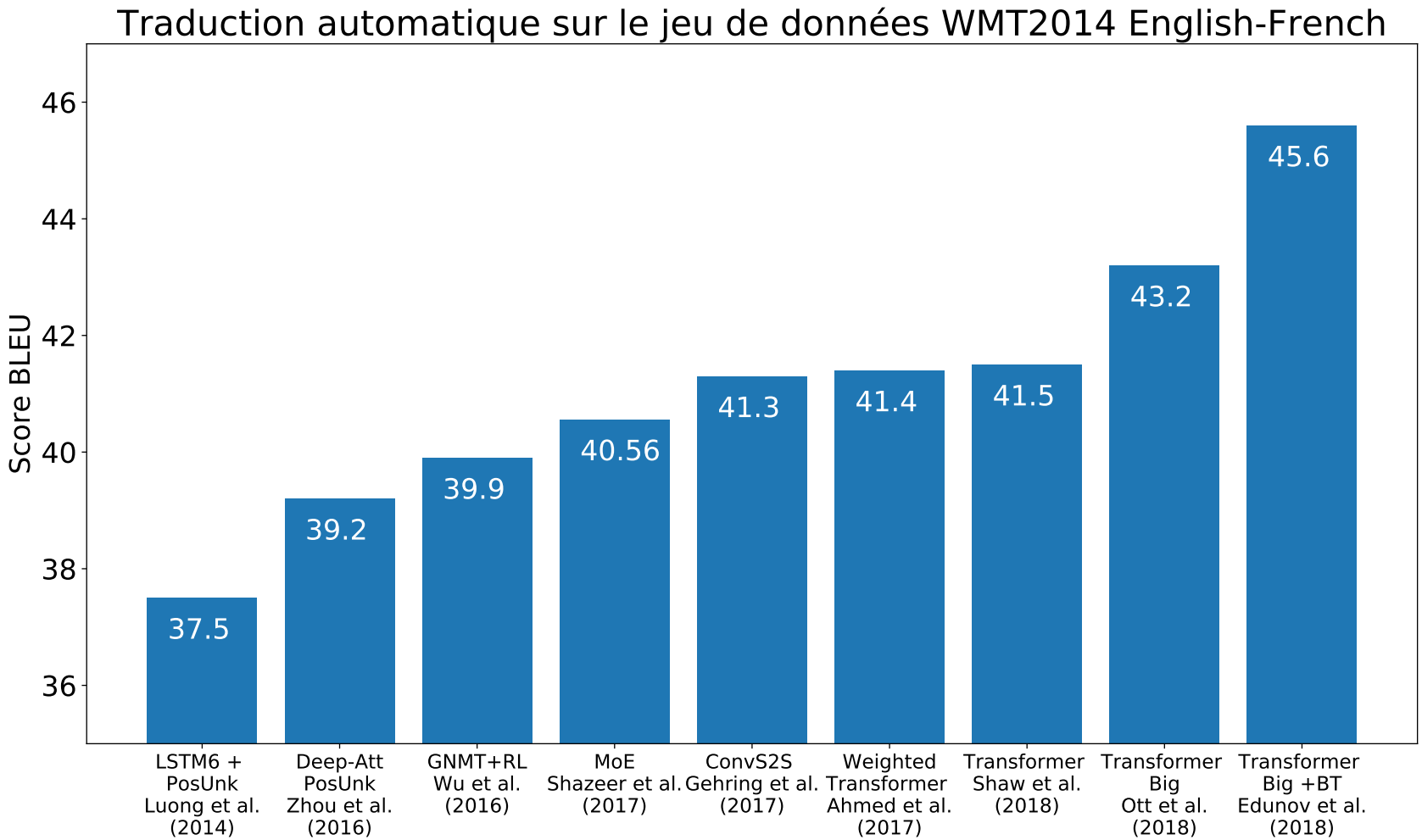
Computer Vision

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

Computer Vision

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

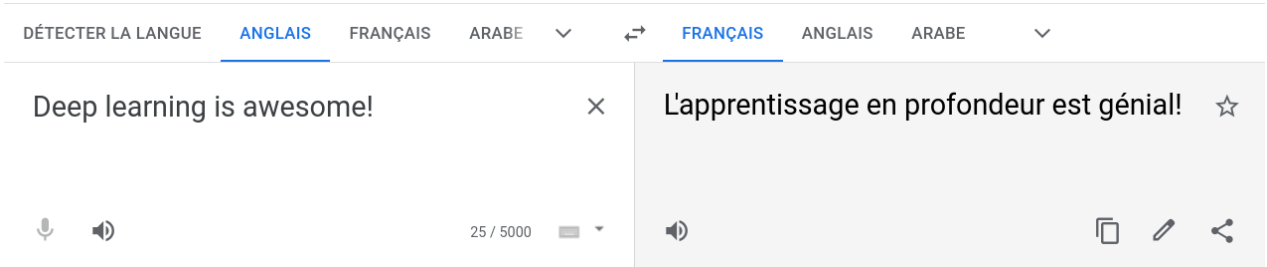
Natural Language Processing (NLP)



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

Natural Language Processing

- Translation

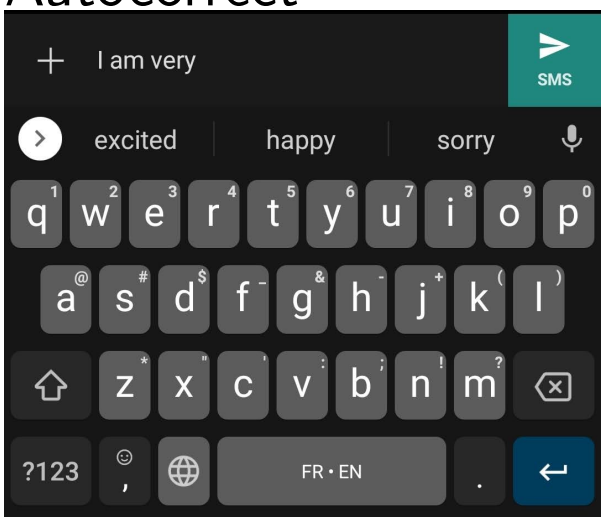


Natural Language Processing

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

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.

Machine Learning

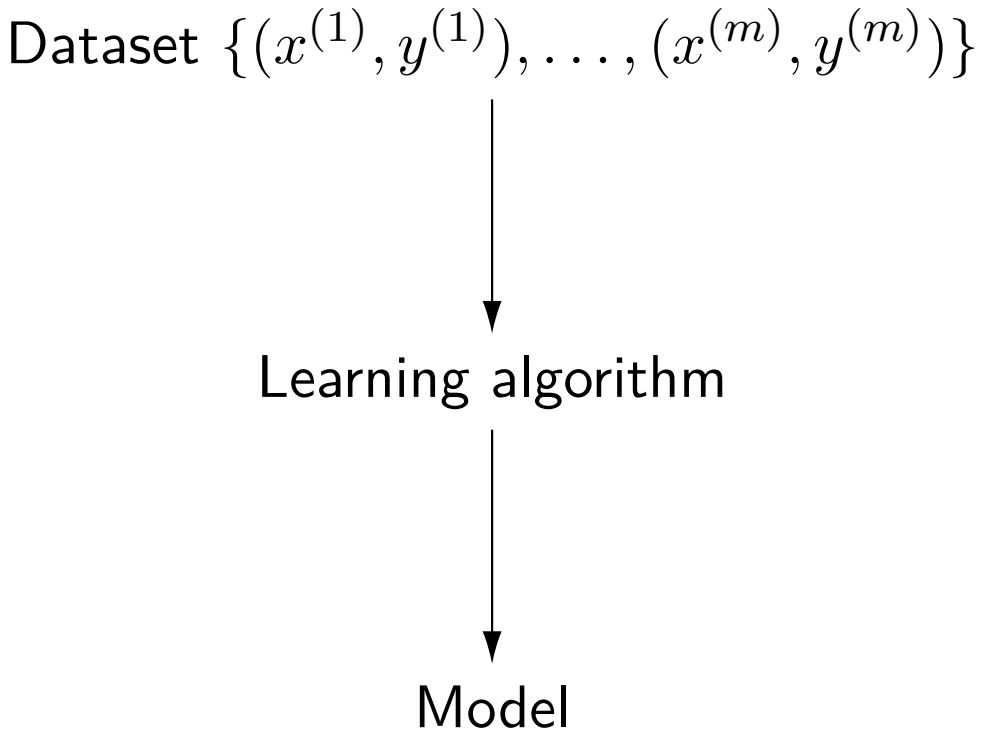
Dataset $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Machine Learning

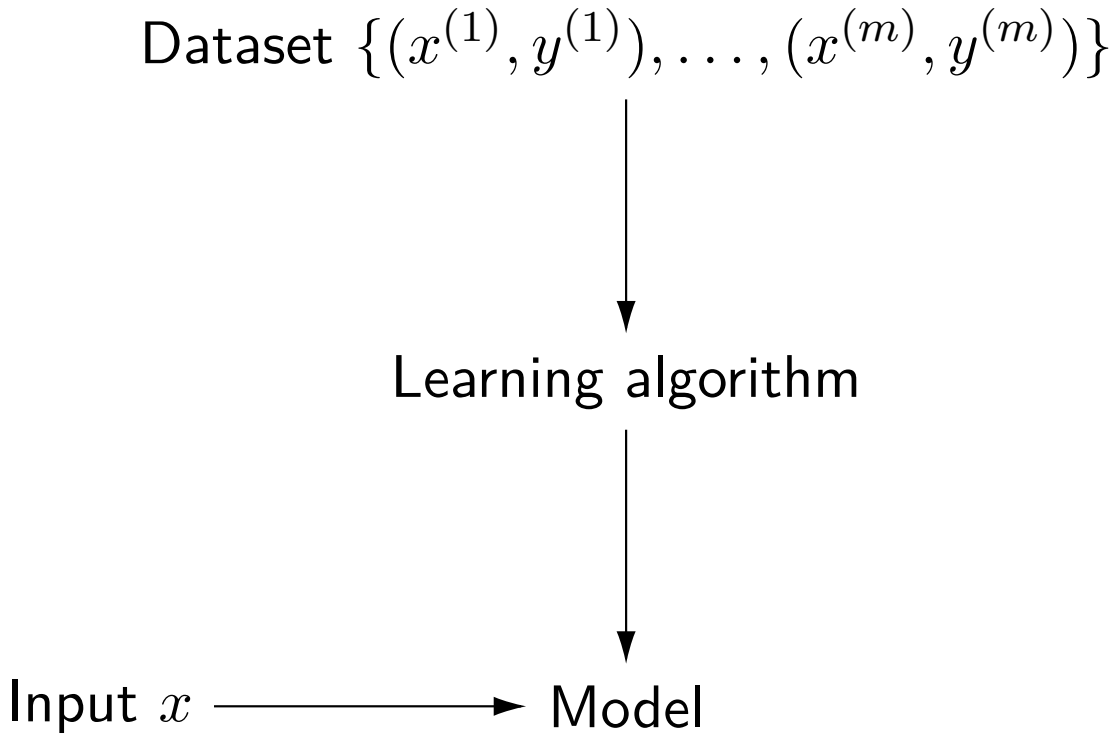
Dataset $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Learning algorithm

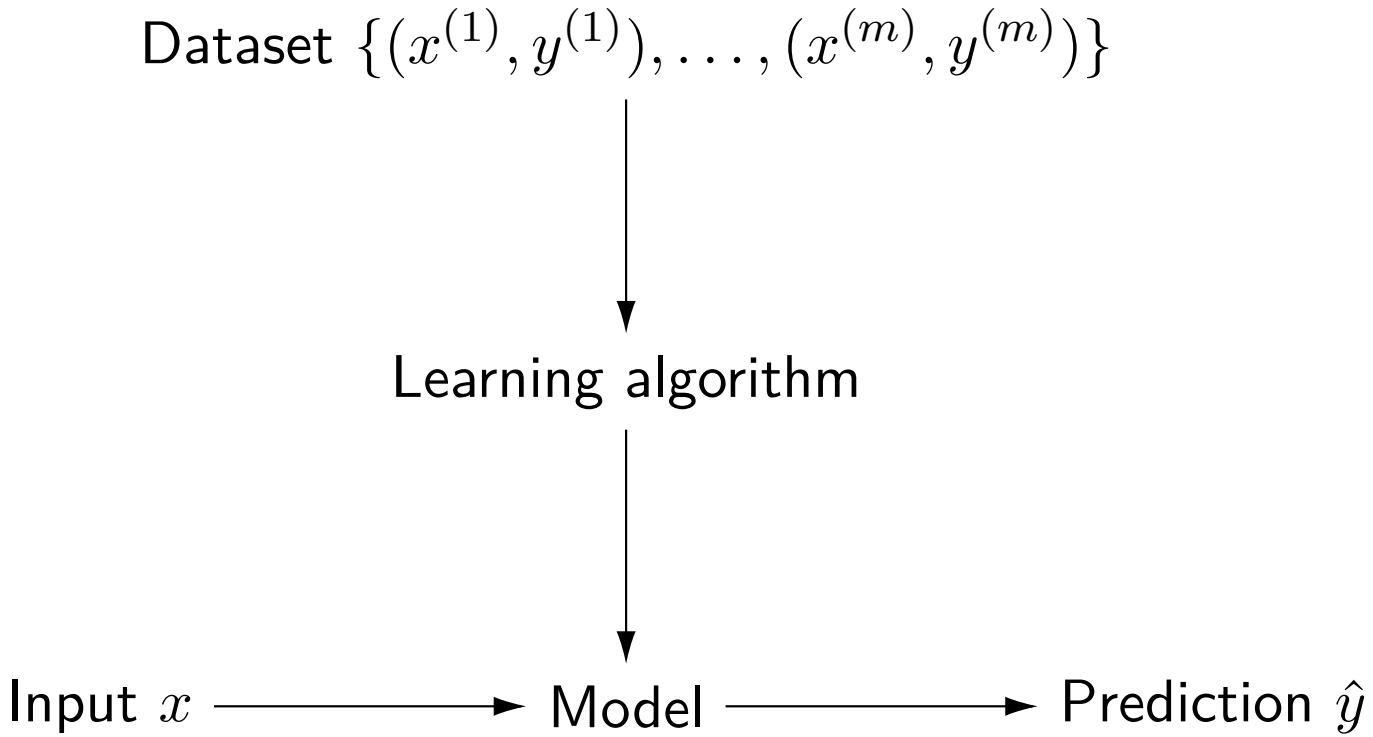
Machine Learning



Machine Learning



Machine Learning

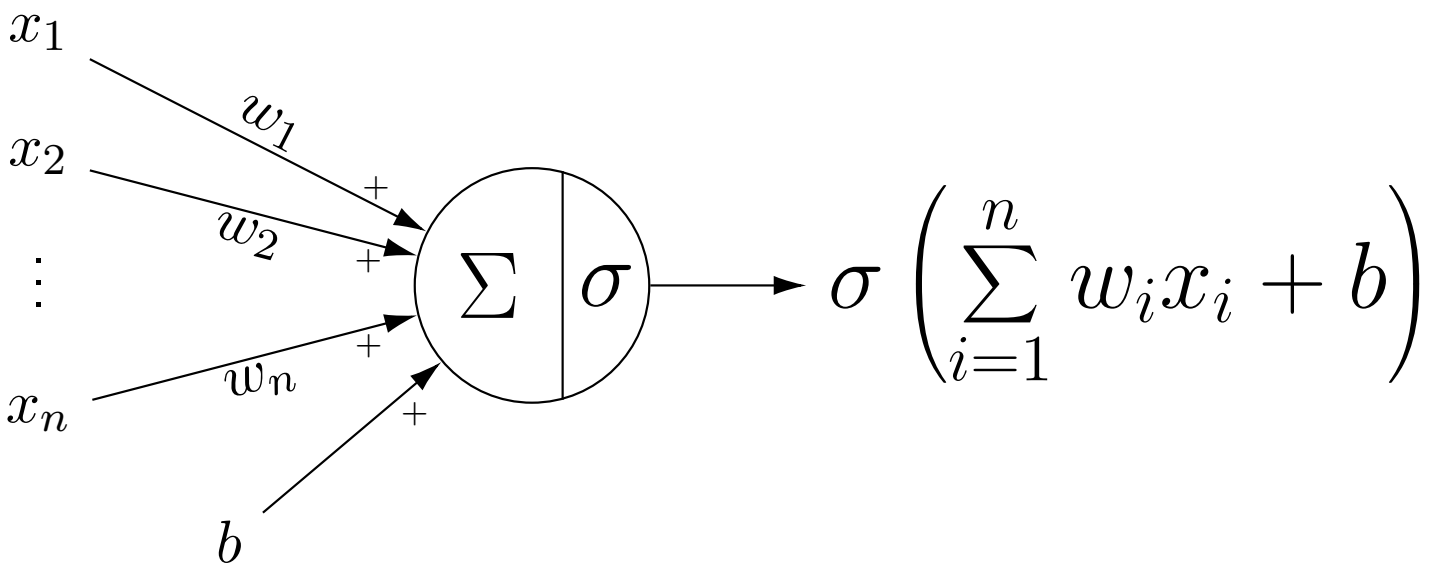


The Basis of Neural Networks: The Neuron

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

The Basis of Neural Networks: The Neuron

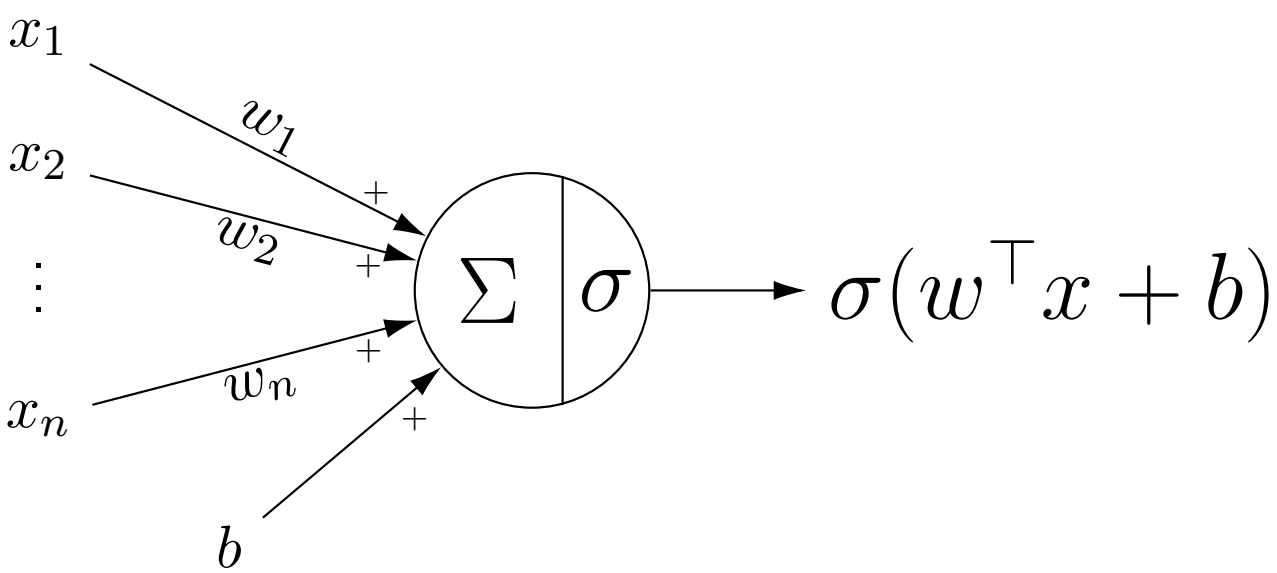
Let $x = (x_1, x_2, \ldots, x_n)$ be n features (or variables in statistics). A neuron computes the following.



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

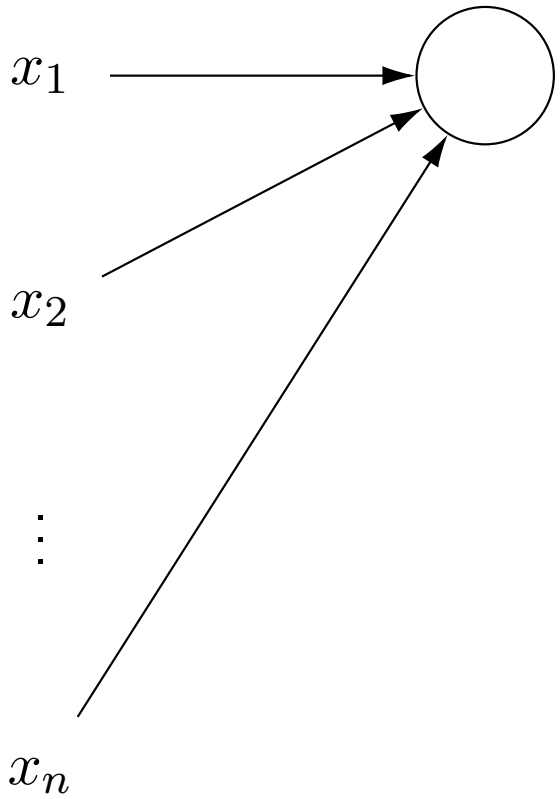
The Basis of Neural Networks: The Neuron

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

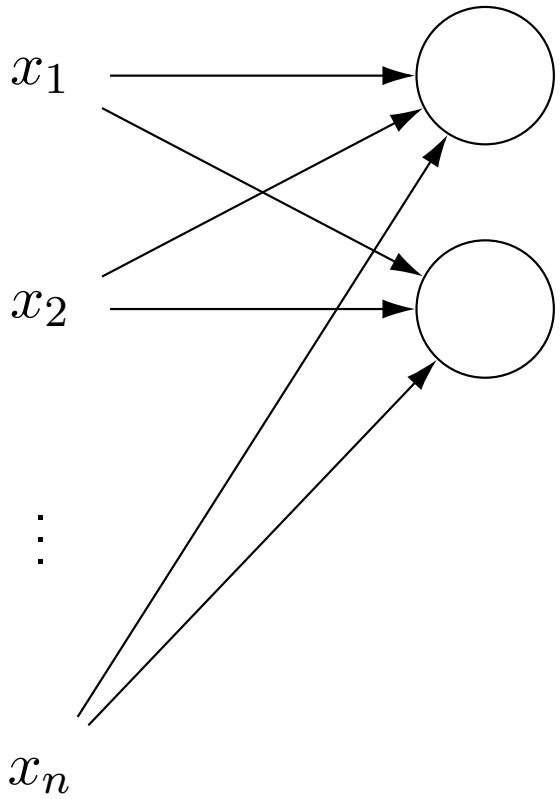


where w are called the weights of the neuron and $\sigma(\cdot)$ is a non-linear activation function.

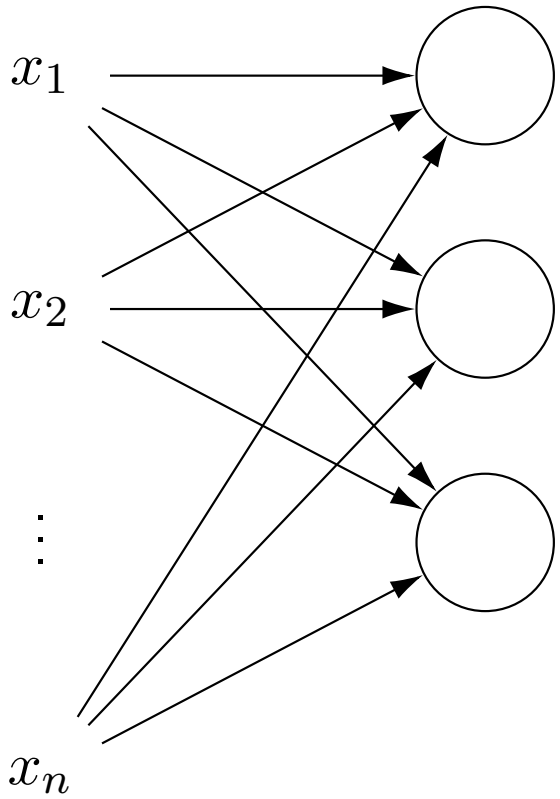
Neural Network



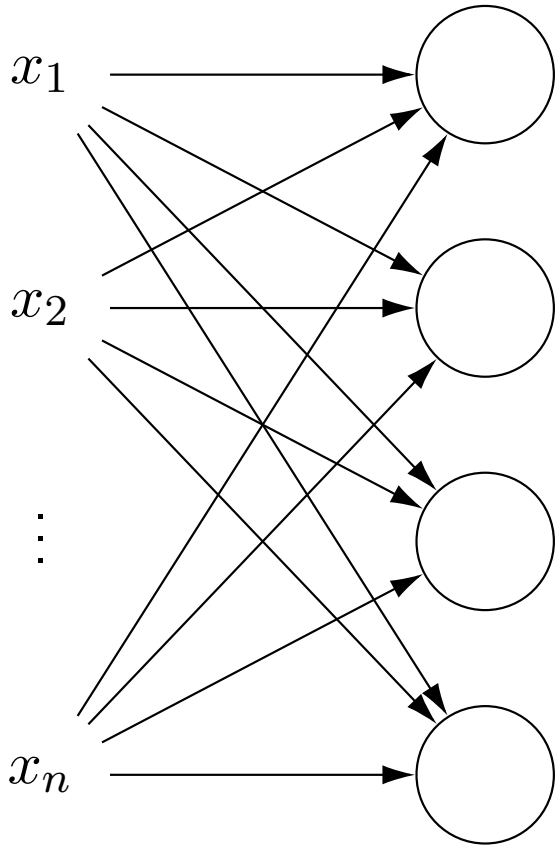
Neural Network



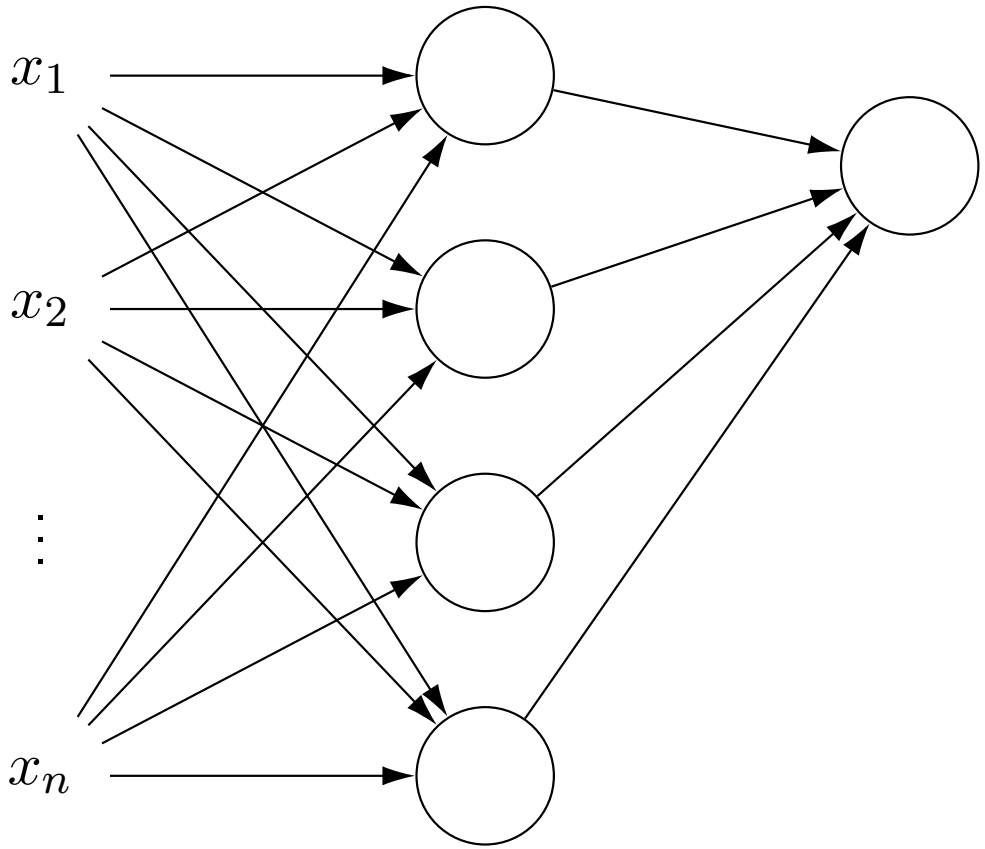
Neural Network



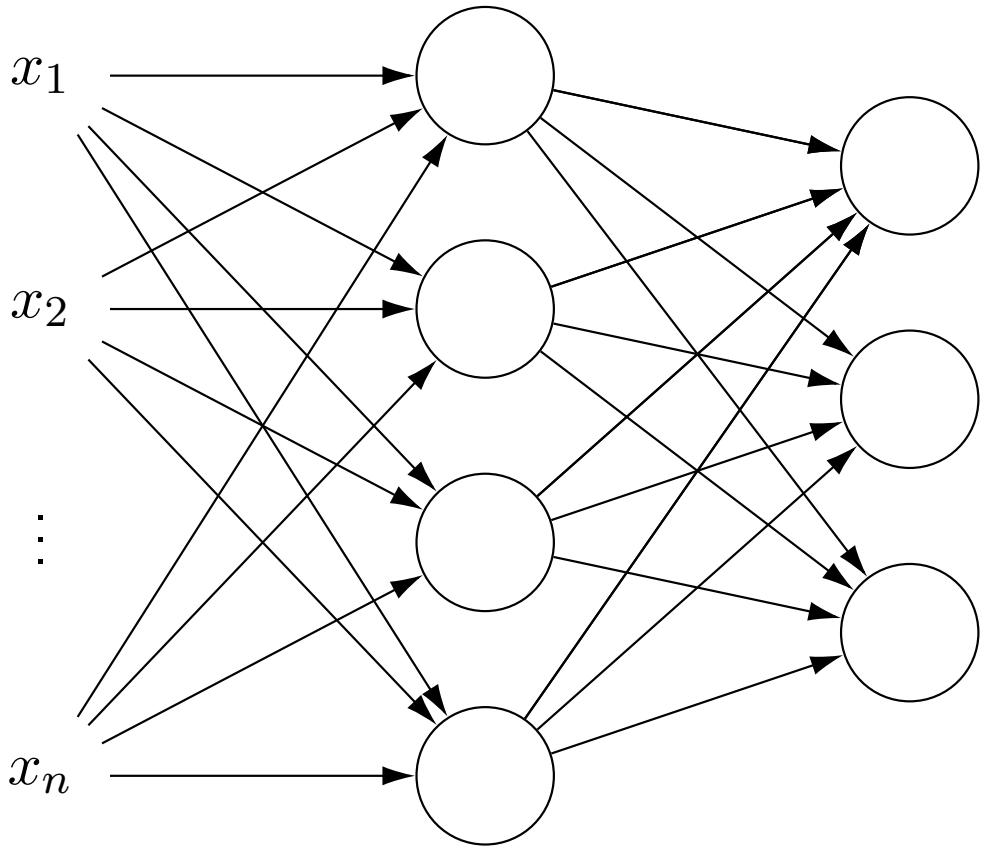
Neural Network



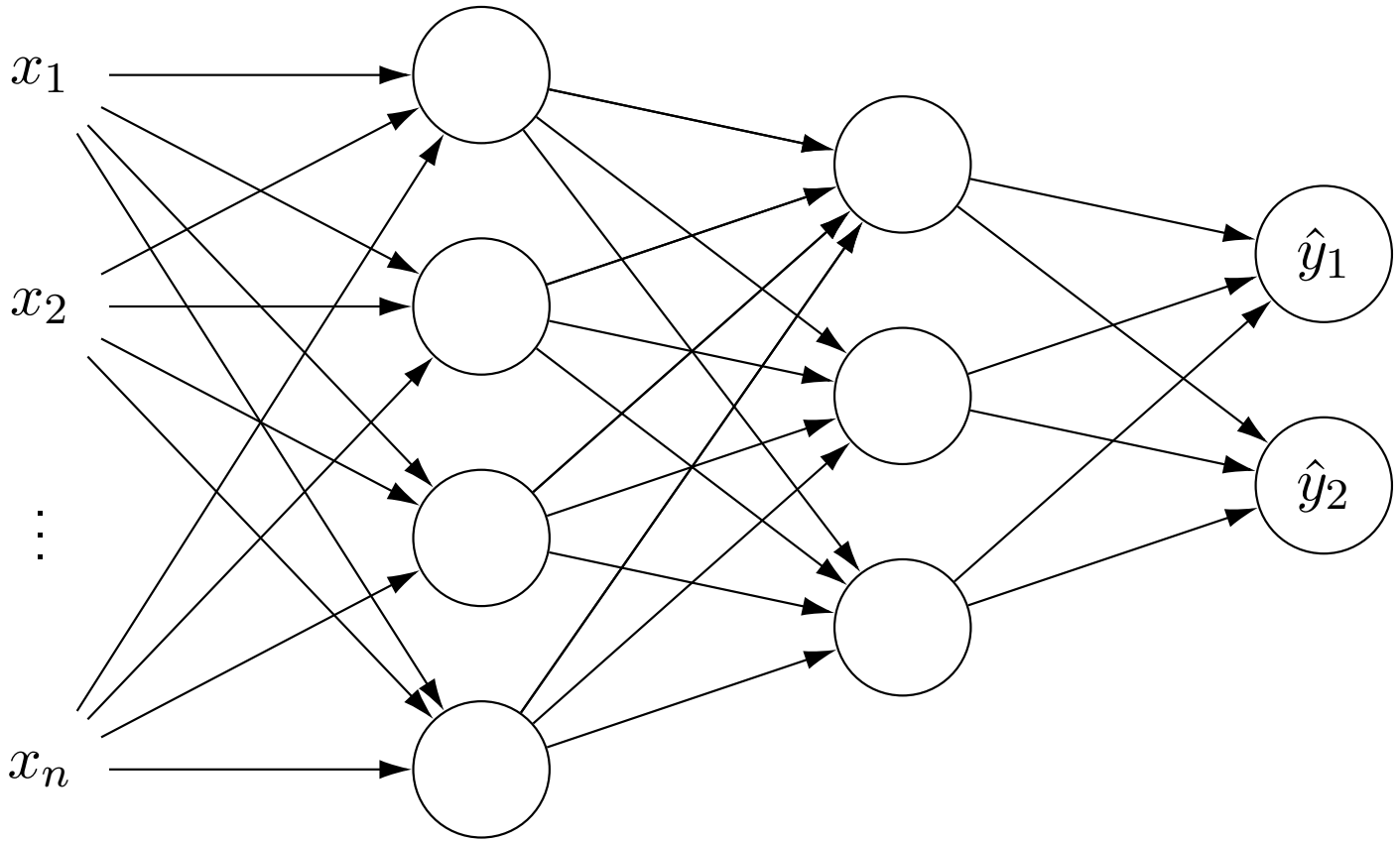
Neural Network



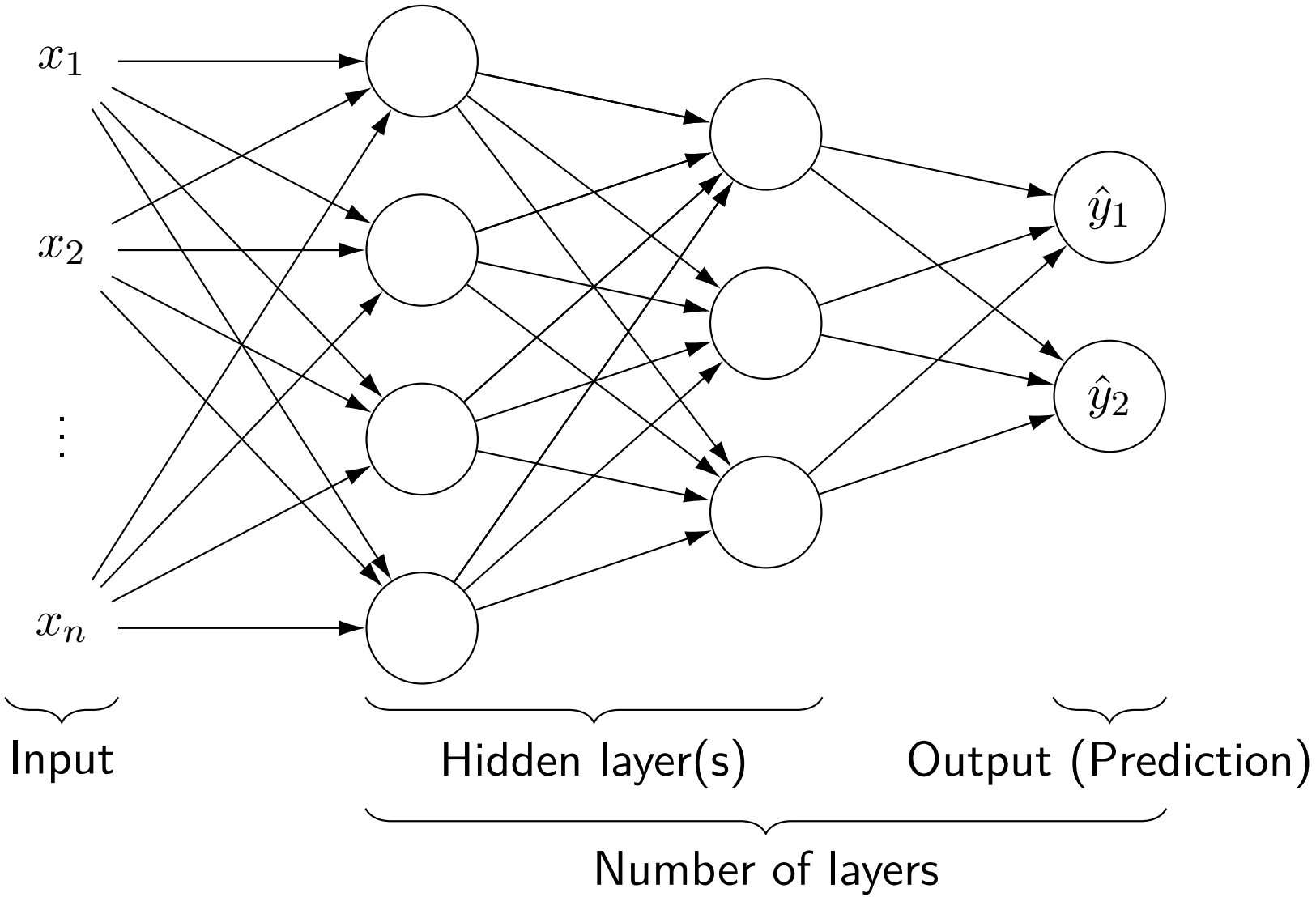
Neural Network



Neural Network



Neural Network



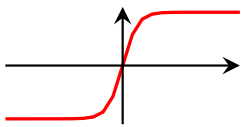
Activation Functions

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

Activation Functions

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

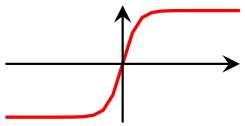
- $\tanh(z)$



Activation Functions

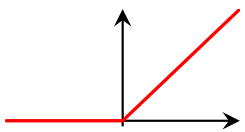
Given a non-activated output $z = w^\top x + b$, then $\sigma(z) = \ldots$

- $\tanh(z)$



- The ReLU function:

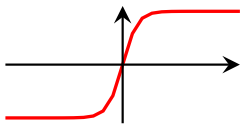
$$\text{ReLU}(z) = \max(0, z)$$



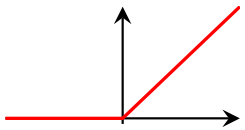
Activation Functions

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

- $\tanh(z)$

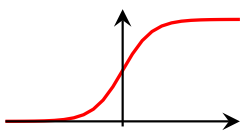


- The ReLU function:



- The logistic function (also called sigmoid function):

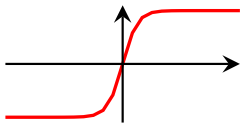
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



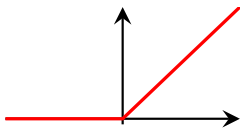
Activation Functions

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

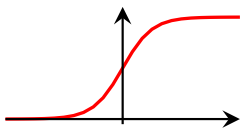
- $\tanh(z)$



- The ReLU function:



- The logistic function (also called sigmoid function):

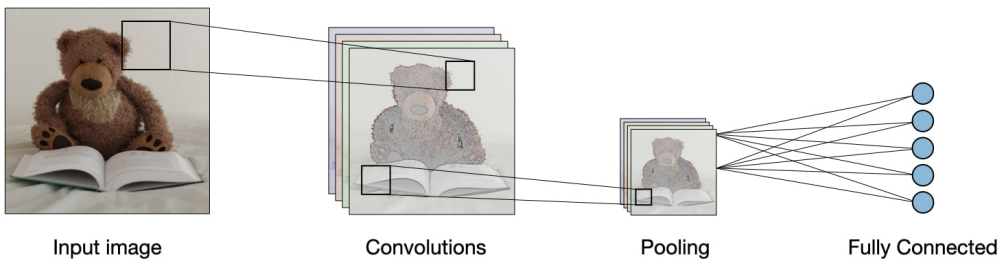


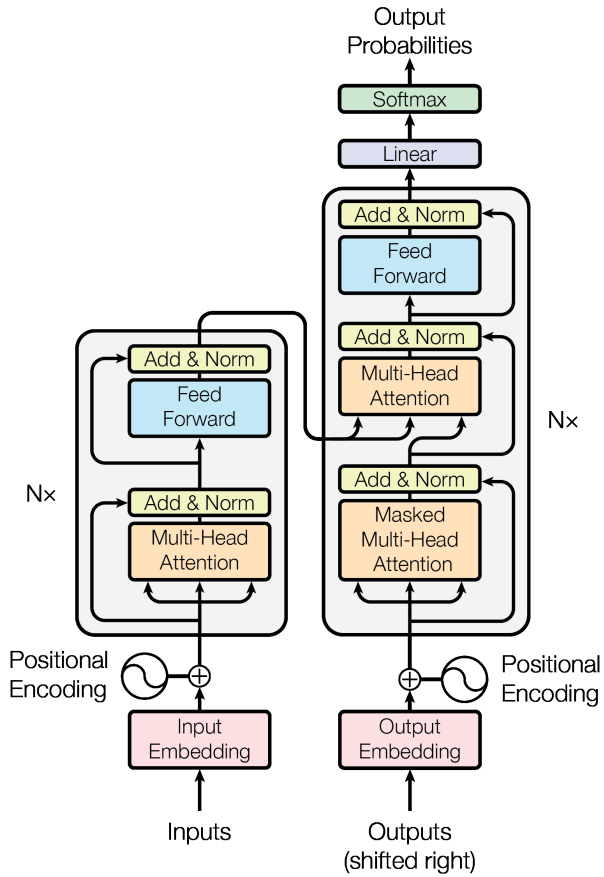
- The softmax function:

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

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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $f(x) = [f(x)_1, \ldots, f(x)_C] = \text{softmax}([z_1, \ldots, z_C])$

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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $f(x) = [f(x)_1, \ldots, f(x)_C] = \text{softmax}([z_1, \ldots, z_C])$

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

Loss Function

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

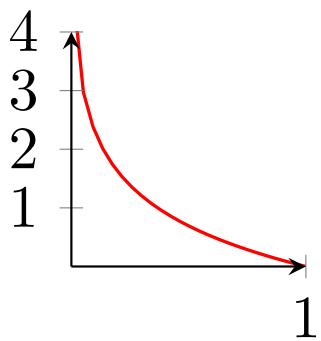
Goal: minimize the loss function!

- 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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $f(x) = [f(x)_1, \ldots, f(x)_C] = \text{softmax}([z_1, \ldots, z_C])$

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



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 θ.

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

Training Objective

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

- The only thing we control are the parameters θ.

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

Training Objective

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

- The only thing we control are the parameters θ.

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

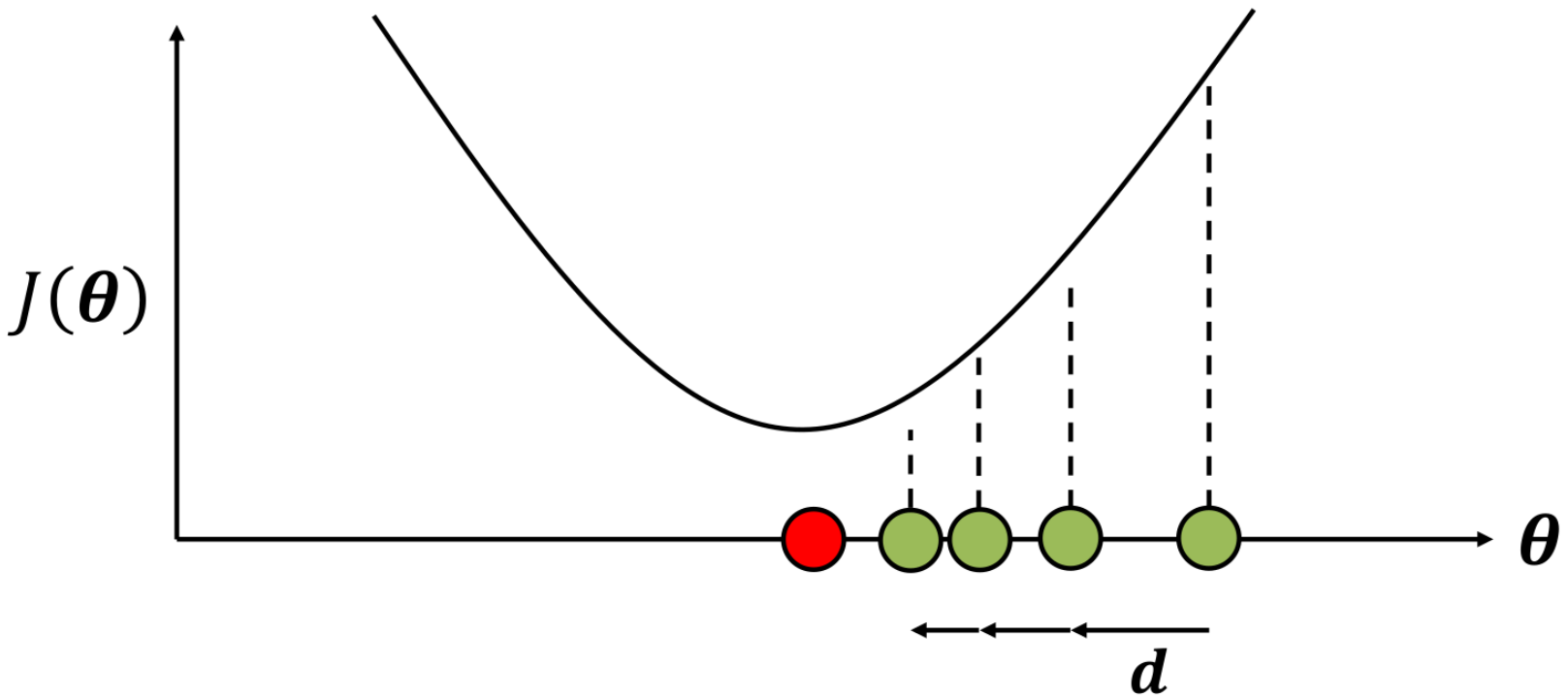
Training Objective

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

- The only thing we control are the parameters θ.

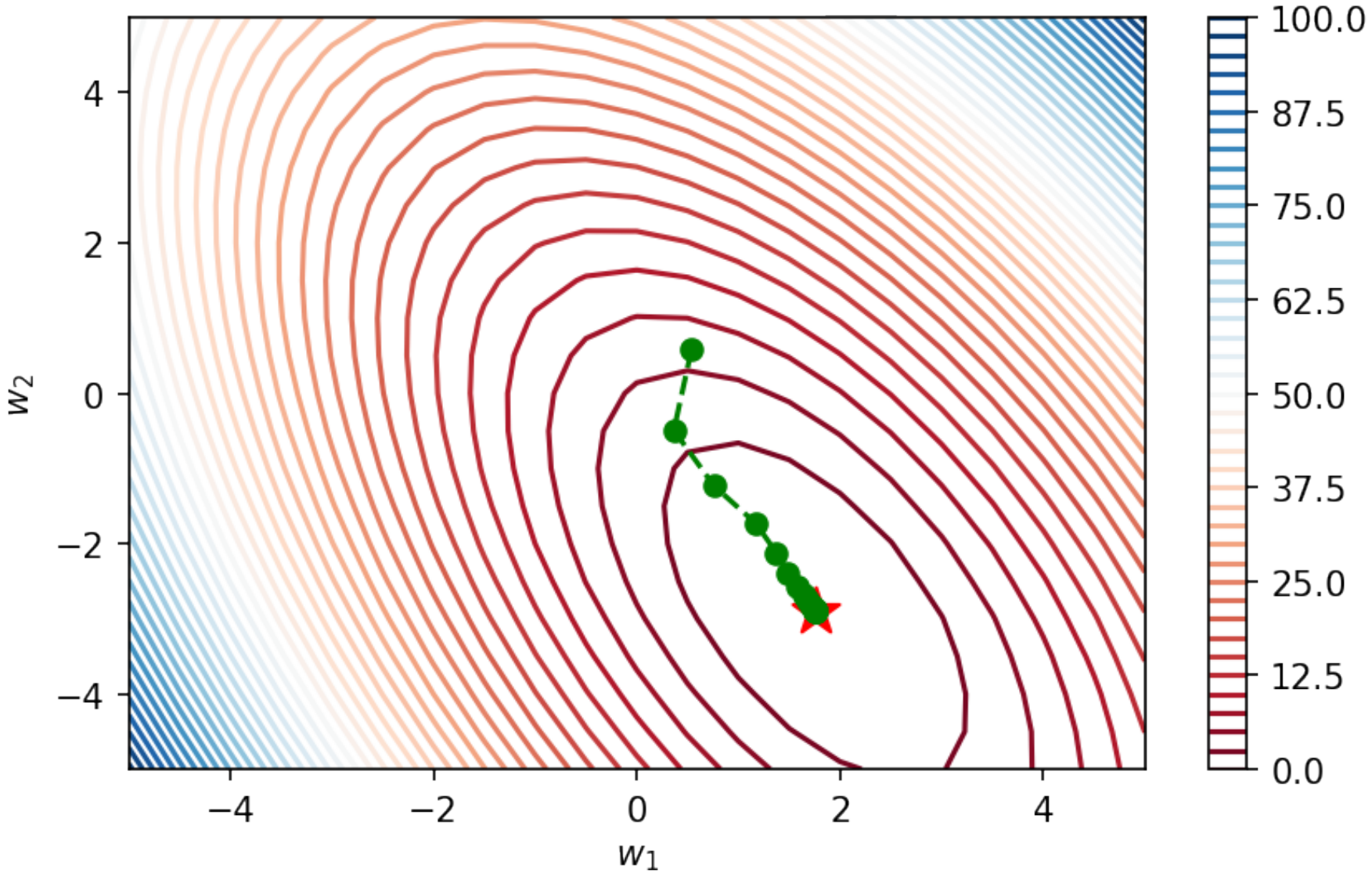
$$J(\theta) = \mathop{\mathbb{E}}_{(x,y)\in\mathcal{D}} \mathcal{L}(f(x;\theta), y)$$
$$\approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(x_i;\theta), y_i)$$

Gradient Descent



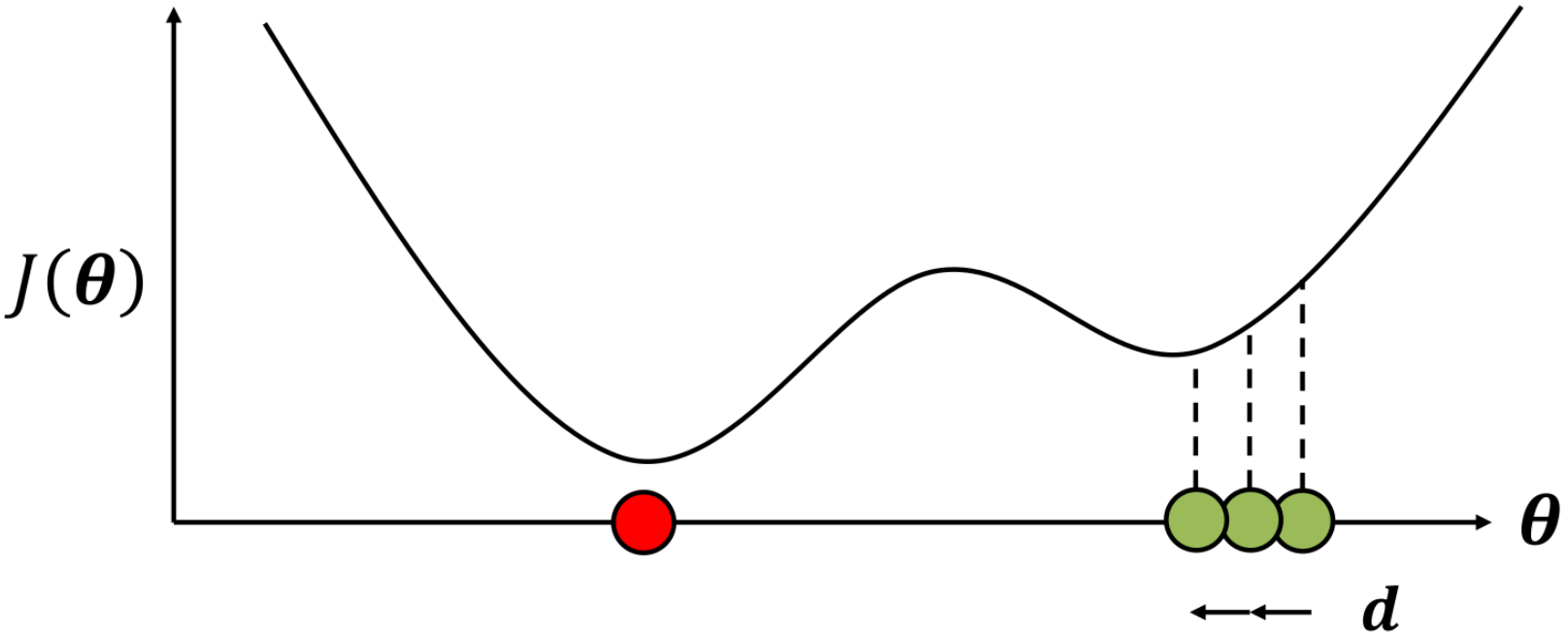
From Université Laval GLO-7030 by Ludovic Trottier

Gradient Descent



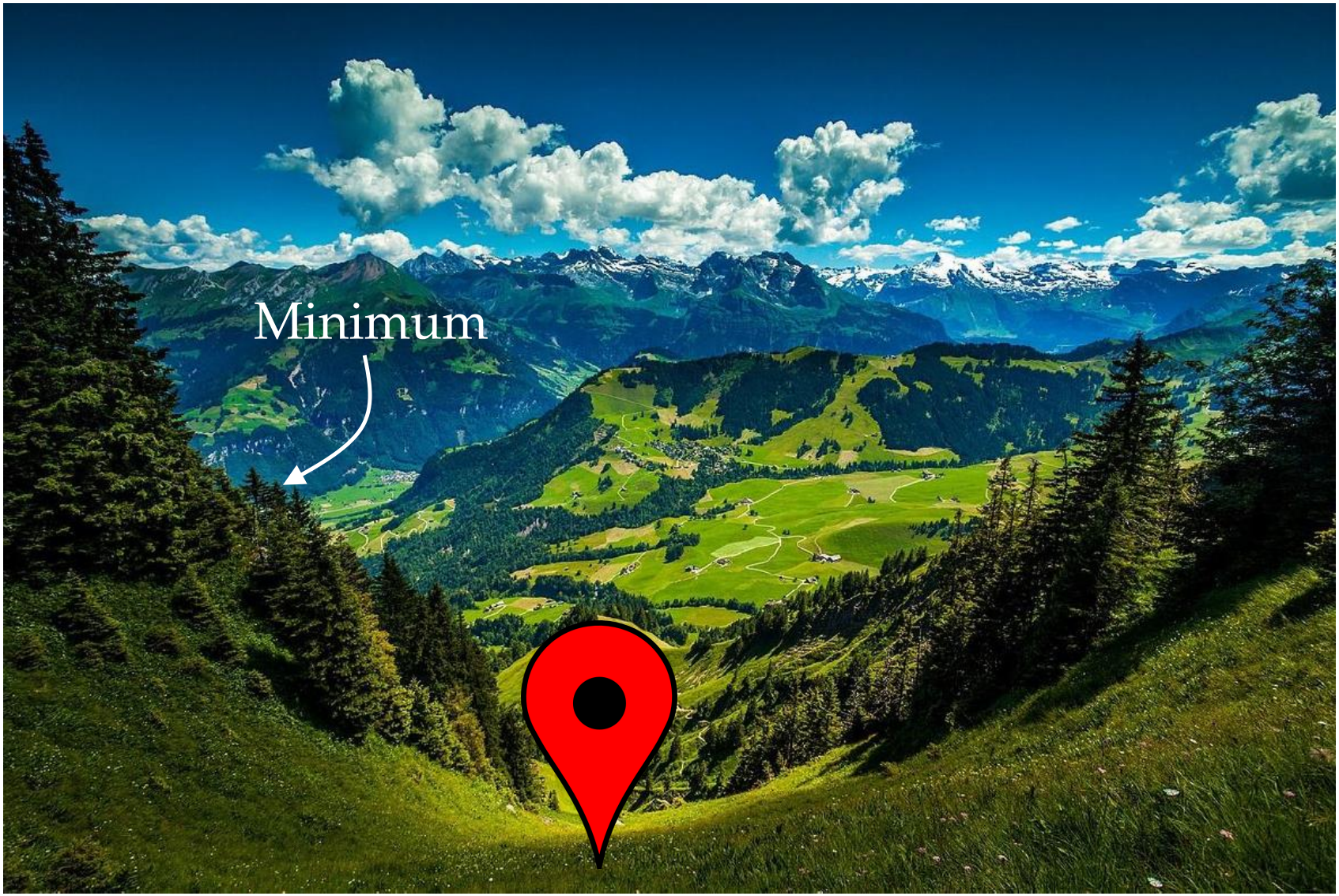
From Université Laval GLO-7030 by Pascal Germain

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

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

Computation of Gradient

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

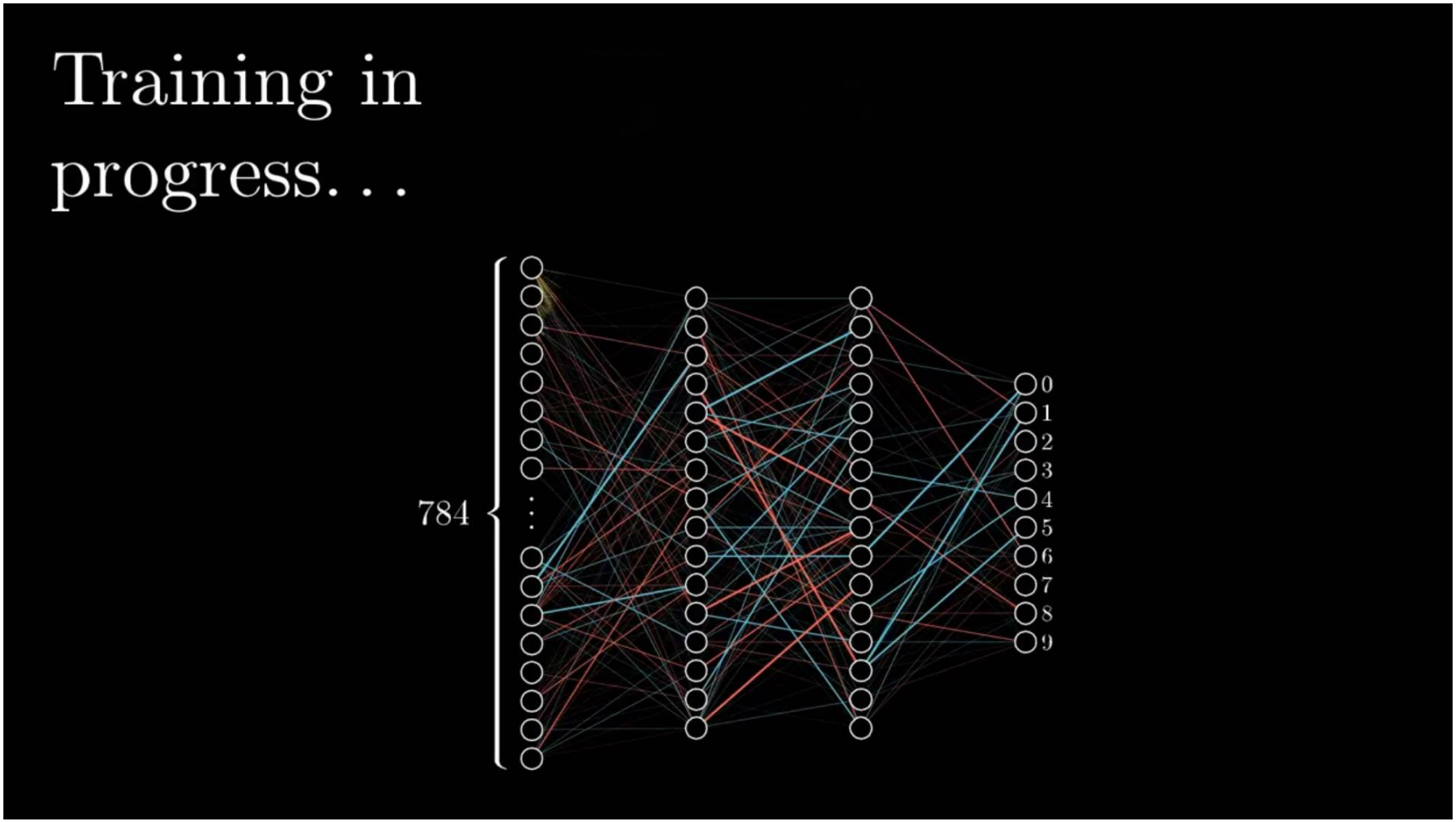
$$d = \nabla_\theta J(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_1}, \ldots, \frac{\partial J(\theta)}{\partial \theta_n}\right]$$

To update the parameters:

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

ϵ is called the learning rate.

Gradient



Extrait de `https://youtu.be/IHZwWFHWa-w` 3Blue1Brown

Optimization Algorithms

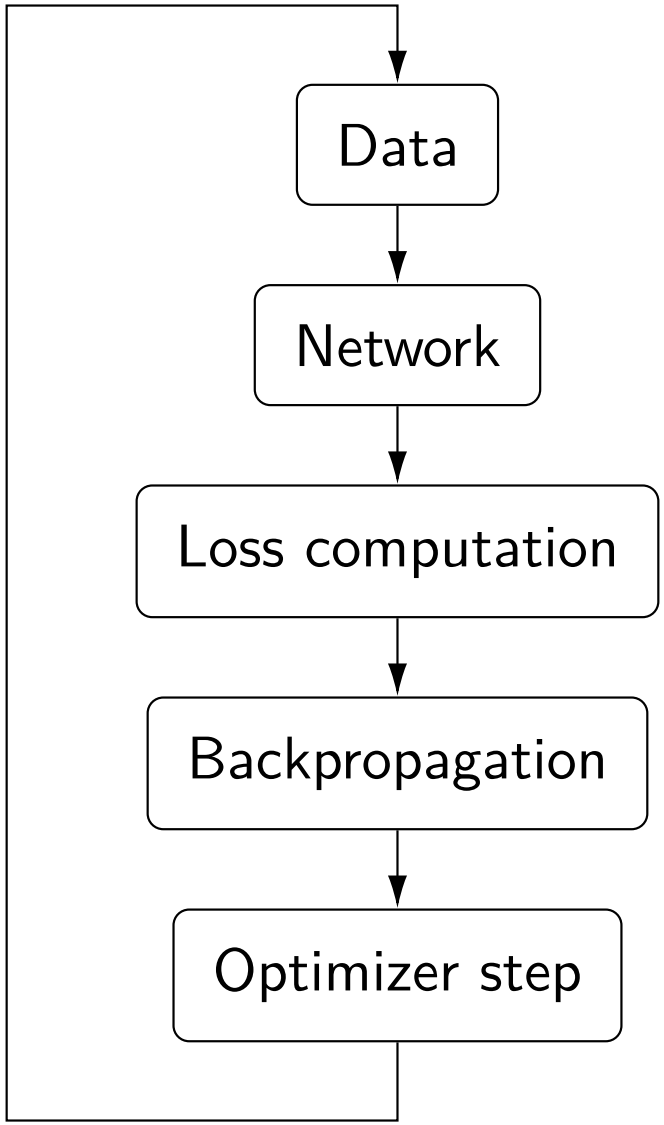
Simplest update rule:

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

Many types exist:

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

Training Procedure



Training Procedure

procedure $\textsc{Train}(f(\cdot;\theta),\ S)$

input: Neural network f parameterized by θ

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

end procedure

Training Procedure

procedure $\textsc{Train}(f(\cdot;\theta),\ S)$
    input: Neural network f parameterized by θ
    input: Dataset $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
    for n epochs **do**

    end for
end procedure

Training Procedure

procedure $\textsc{Train}(f(\cdot;\theta),\ S)$
    input: Neural network f parameterized by θ
    input: Dataset $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
    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 $\textsc{Train}(f(\cdot;\theta),\ S)$
  input: Neural network f parameterized by θ
  input: Dataset $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  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

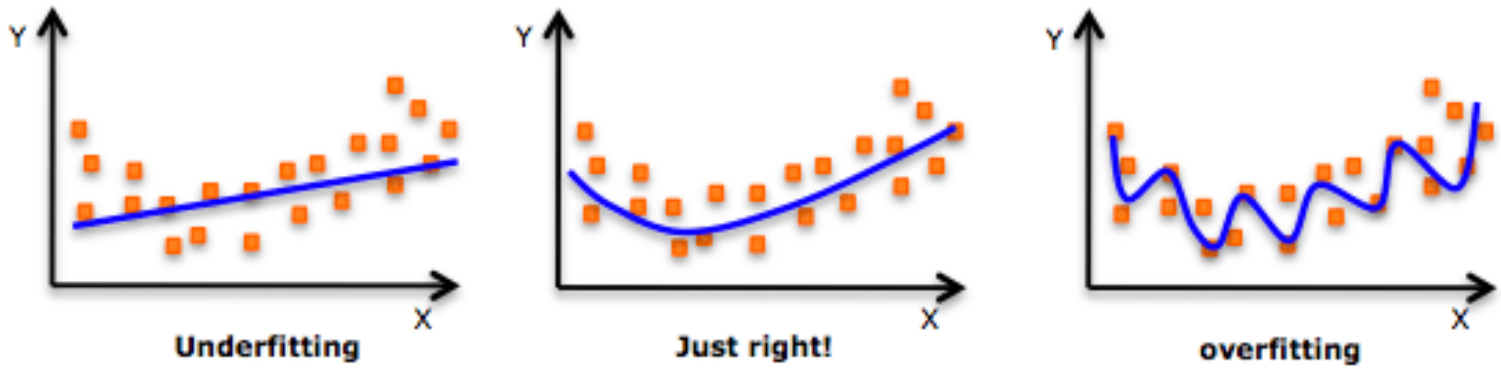
procedure $\textsc{Train}(f(\cdot;\theta),\ S)$
  input: Neural network f parameterized by θ
  input: Dataset $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  for n epochs **do**
    Let $S' = S$
    while $S' \neq \emptyset$ **do**
      Draw a batch $B \subseteq S'$ of b examples.
      $\ell = \frac{1}{b} \sum_{(x,y)\in B} \mathcal{L}(f(x;\theta), y)$
      $d = \nabla_\theta \ell$
      Update θ with d using chosen optimizer
    end while
  end for
end procedure

Neural Networks are Powerful

- Universal Approximation Theorem

Neural Networks are Powerful

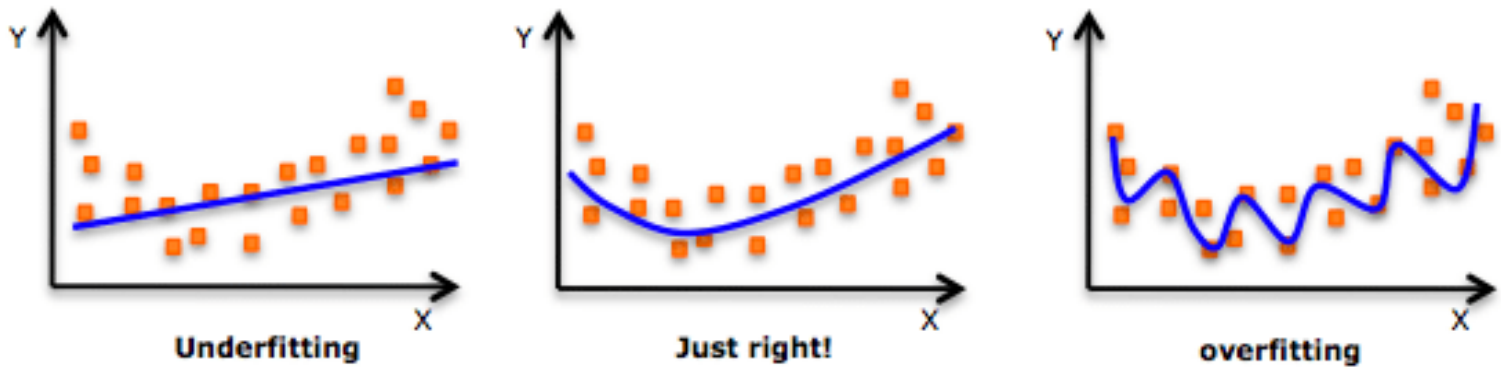
- Universal Approximation Theorem
- ... is a *blessing*!



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[1]:

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

[1] MLE stands for Maximum Likehood Estimation.

Regularization

Add a penalty term to the loss[1]:

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

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

[1] MLE stands for Maximum Likehood Estimation.

Regularization

Add a penalty term to the loss[1]:

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

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

[1] MLE stands for Maximum Likehood Estimation.

Regularization

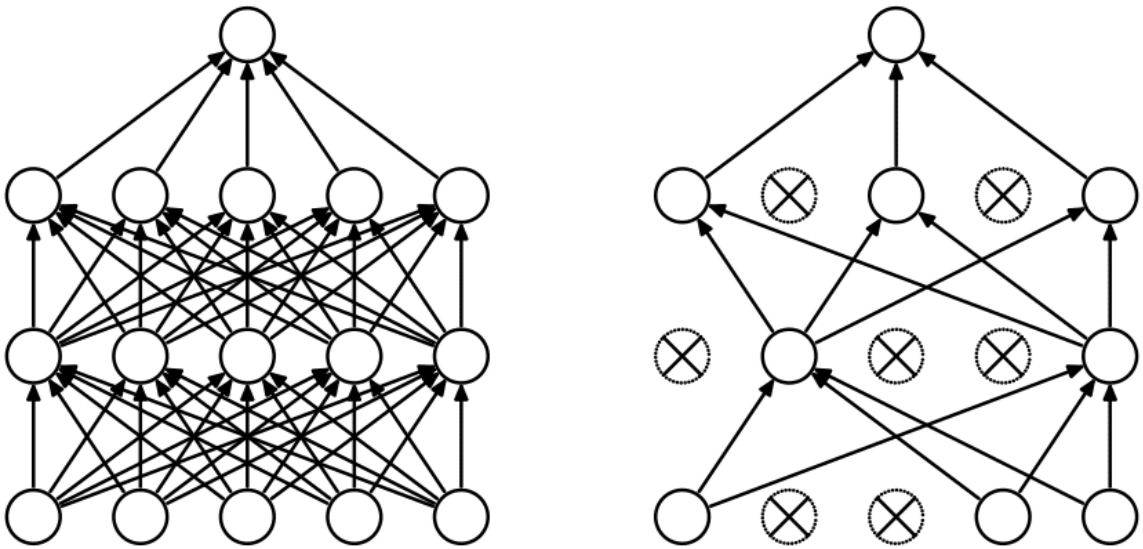
Add a penalty term to the loss[1]:

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

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

[1] MLE stands for Maximum Likehood Estimation.

Dropout

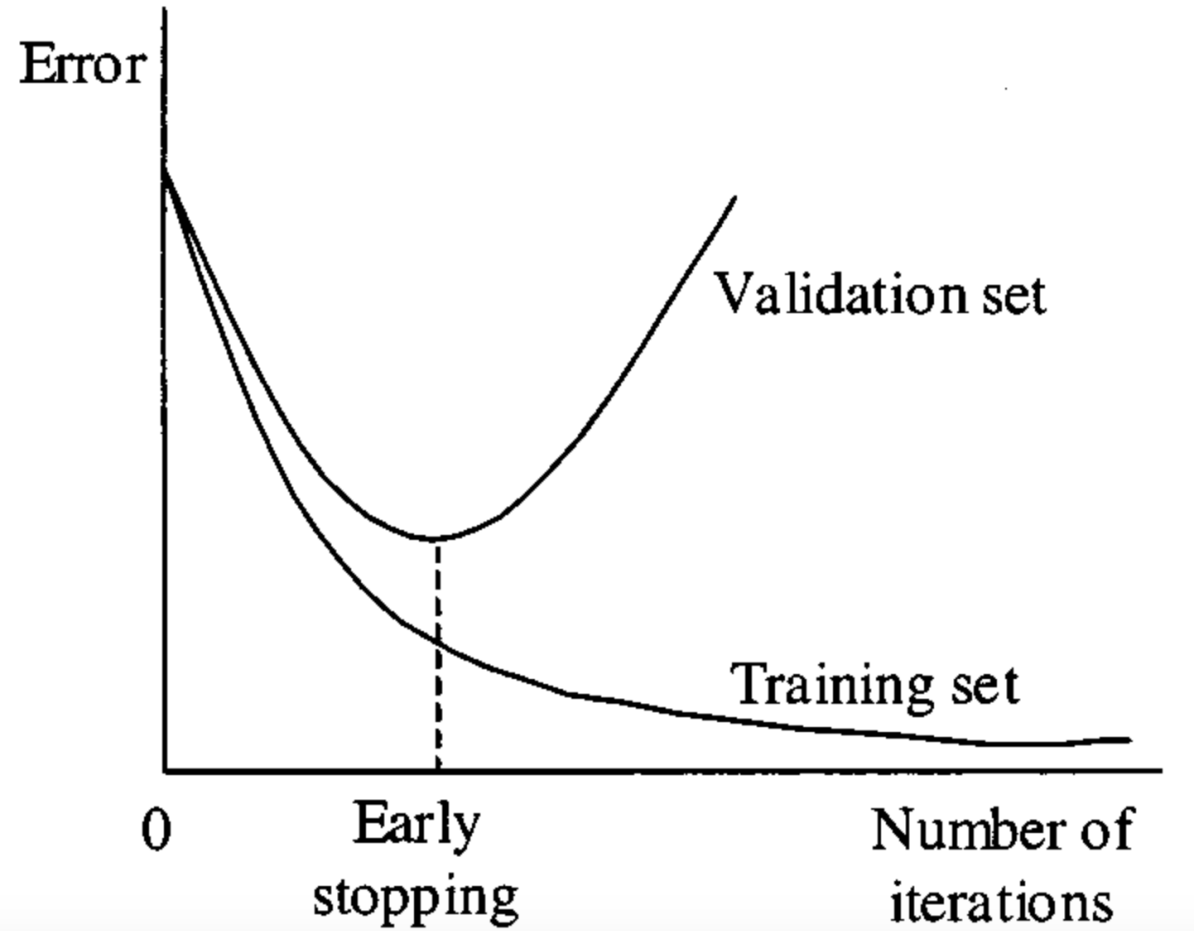


(a) Standard Neural Net (b) After applying dropout.

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 training set. Use it to assess generalization potential.



Deep Learning Libraries

TensorFlow

PyTorch

Deep Learning Libraries

TensorFlow

PyTorch

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`
- 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

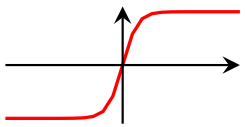
Activation Functions

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

Activation Functions

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

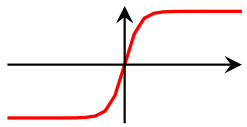
- $\tanh(z)$



Activation Functions

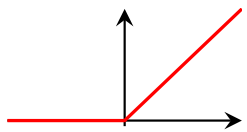
Given a non-activated output $z = w^\top x + b$, then $\sigma(z) = \ldots$

- $\tanh(z)$



- The ReLU function:

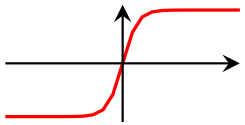
$$\text{ReLU}(z) = \max(0, z)$$



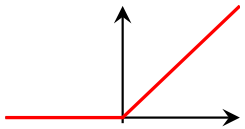
Activation Functions

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

- $\tanh(z)$

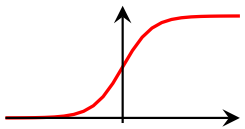


- The ReLU function:



- The logistic function (also called sigmoid function):

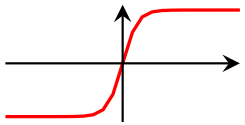
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



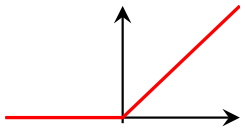
Activation Functions

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

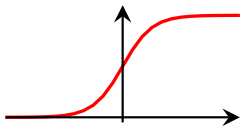
- $\tanh(z)$



- The ReLU function:



- The logistic function (also called sigmoid function):



- The softmax function:

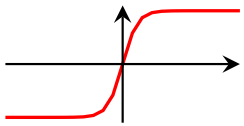
$$\text{softmax}\,(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{m} \exp(z_j)}$$

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

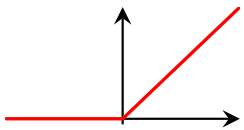
Activation Functions

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

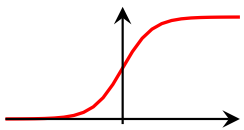
- $\tanh(z)$



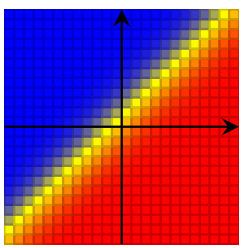
- The ReLU function:



- The logistic function (also called sigmoid function):



- The softmax function:



$\text{softmax}(z)_1 = \frac{\exp(x)}{\exp(x)+\exp(y)}$ 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 ($\mathcal{Y} = \{1, \ldots, C\}$)

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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_C] = \text{softmax}([z_1, \ldots, z_C])$

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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_C] = \text{softmax}([z_1, \ldots, z_C])$

  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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_C] = \text{softmax}([z_1, \ldots, z_C])$

  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 ($\mathcal{Y} = \{1, \ldots, C\}$)
  - $y \in \mathcal{Y}$
  - $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_C] = \text{softmax}([z_1, \ldots, z_C])$

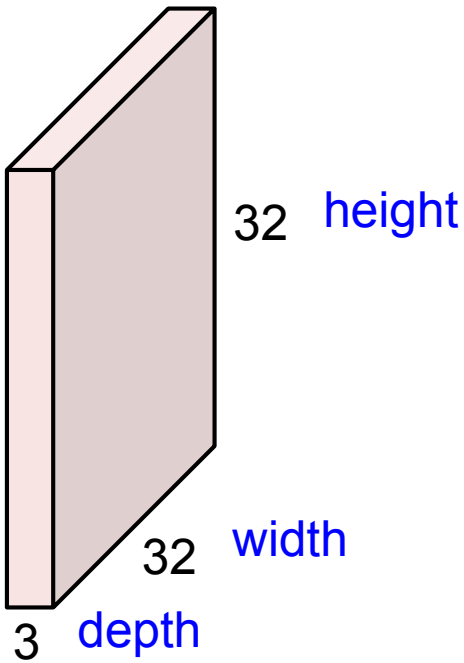
  Based on the cross entropy:

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

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

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

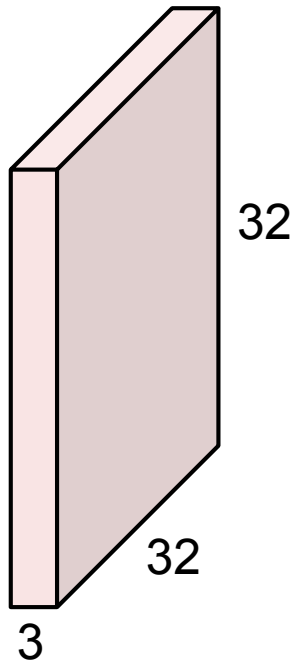
Convolution Layer



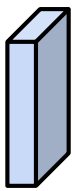
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer

32x32x3 image



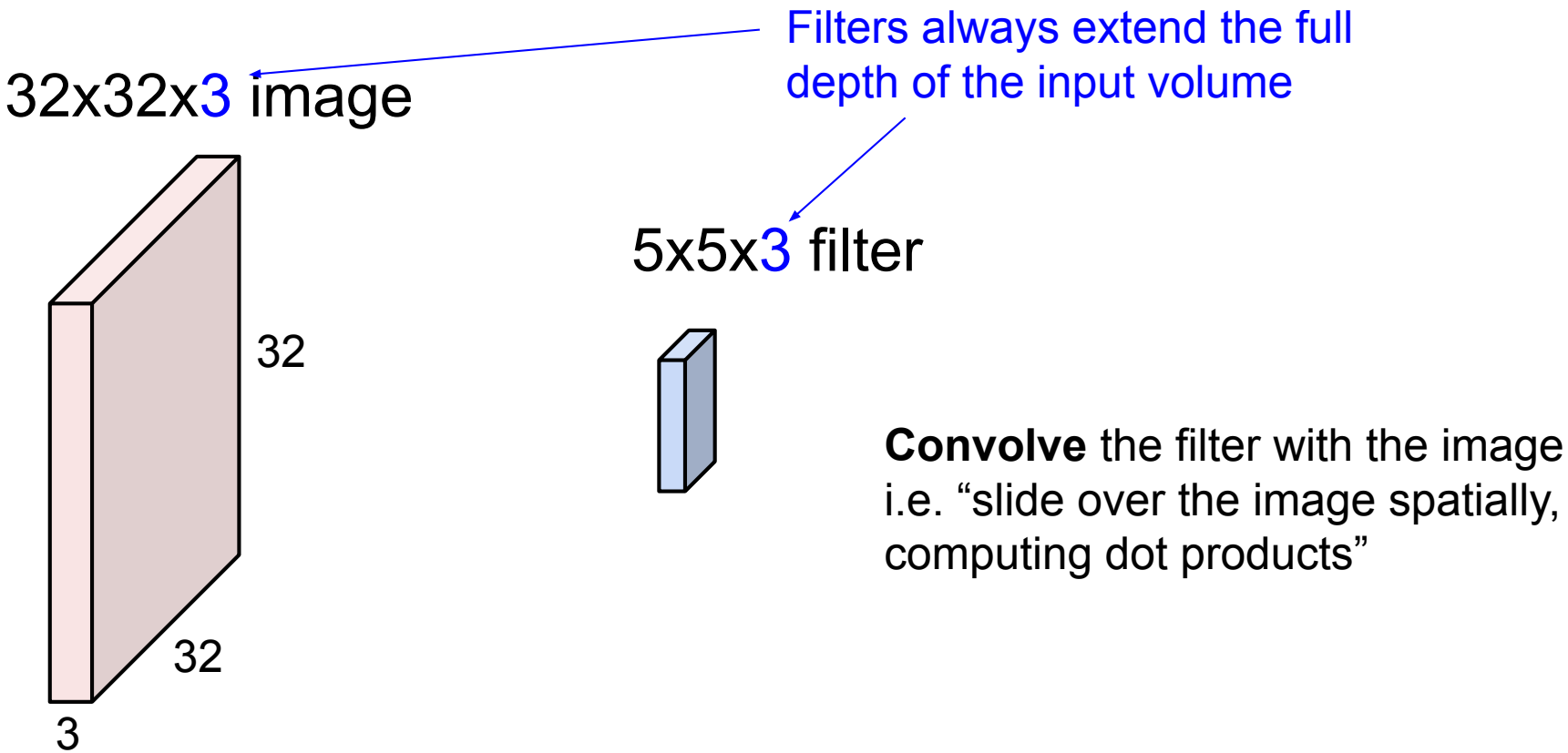
5x5x3 filter



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

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

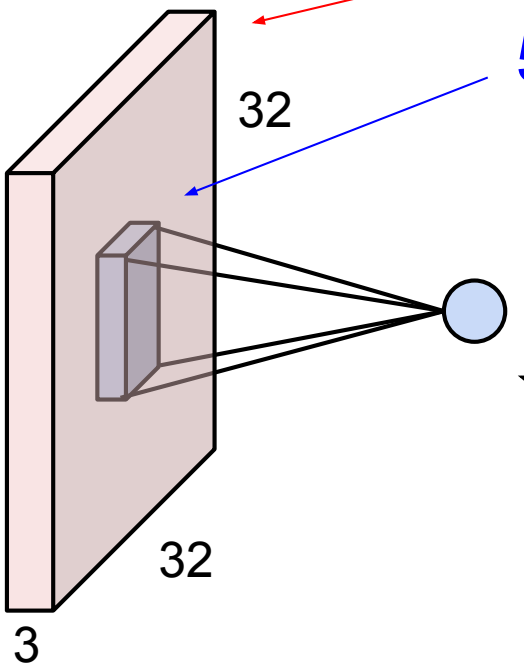
Convolution Layer



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

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

Convolution Layer

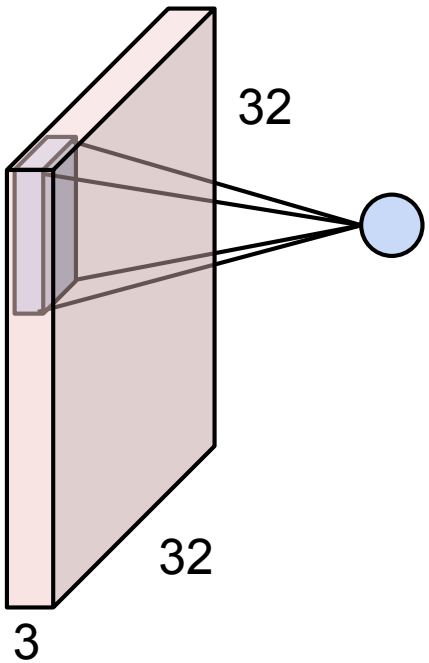


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

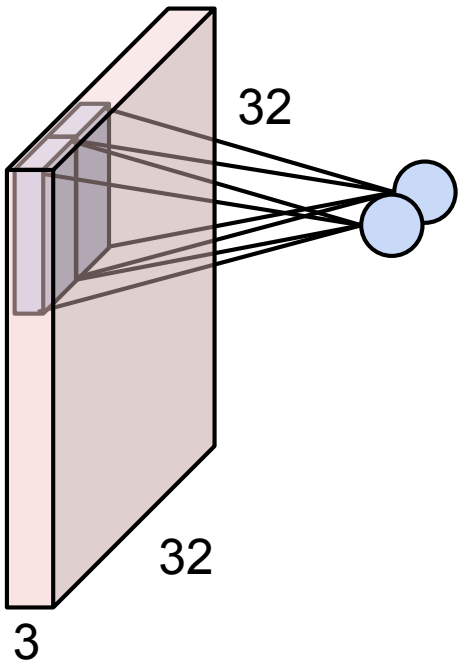
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



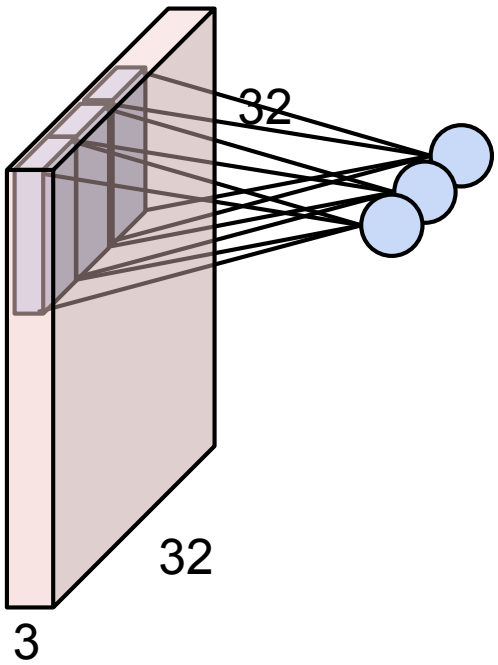
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



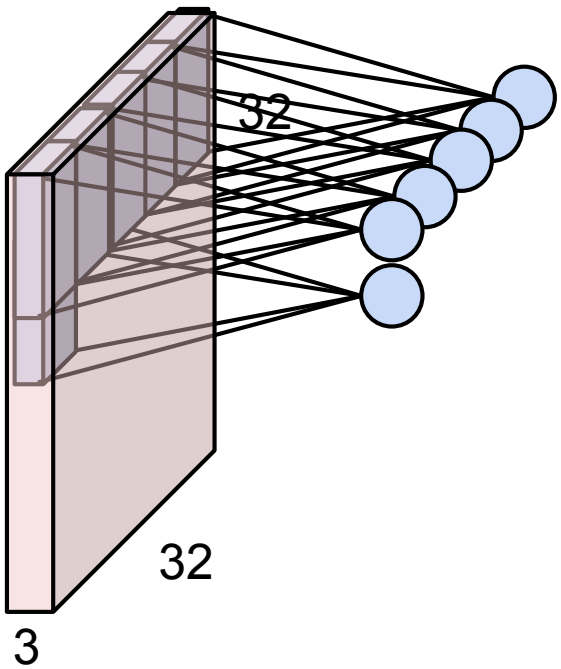
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



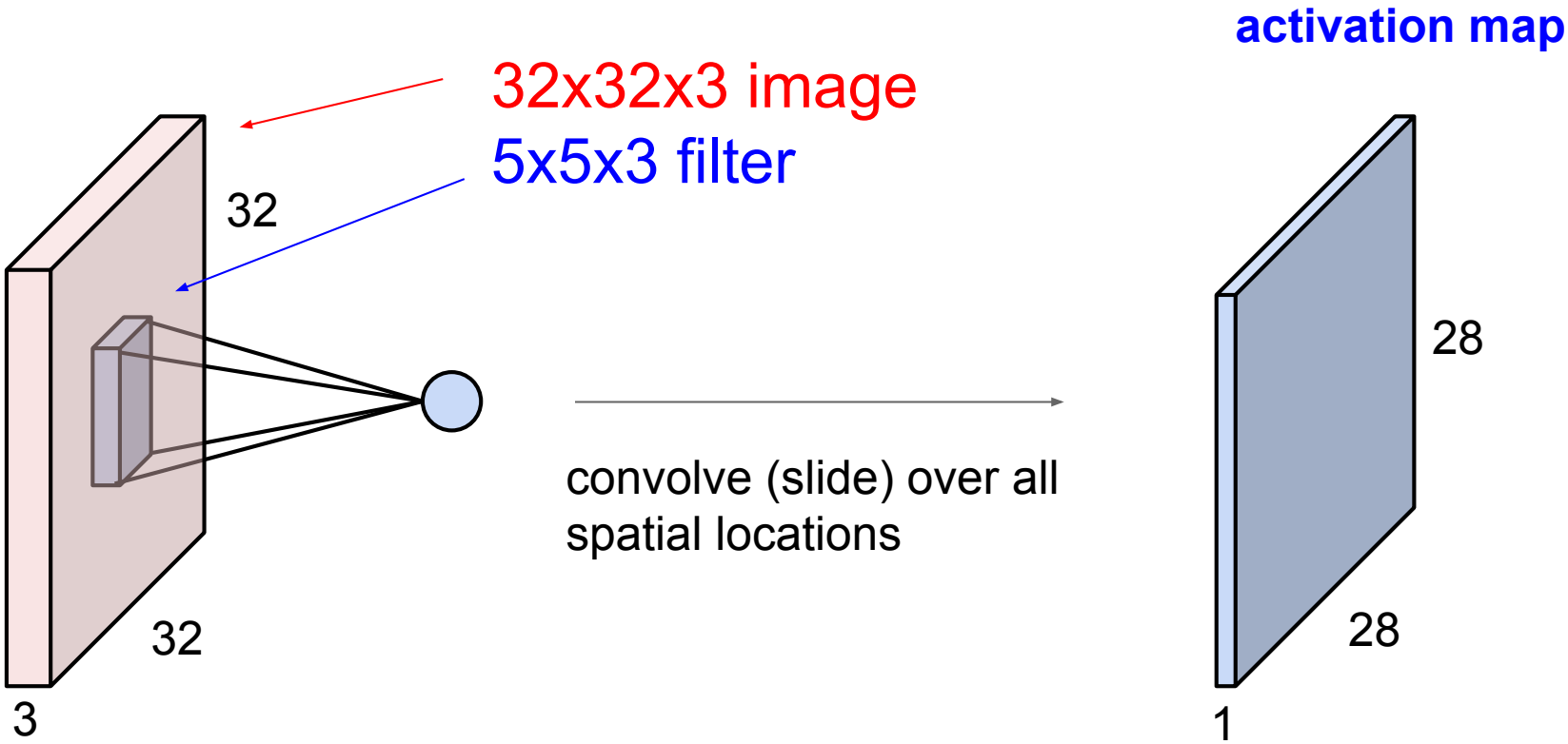
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



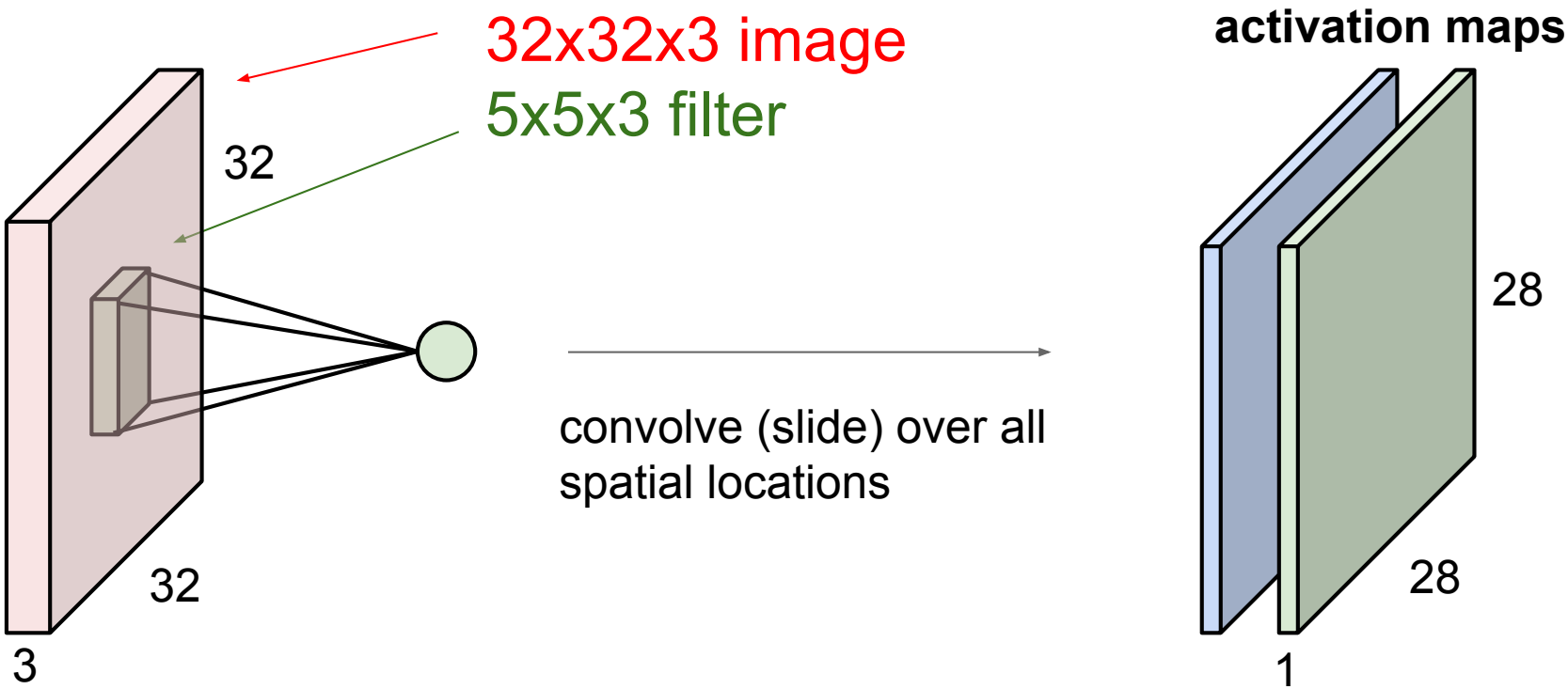
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



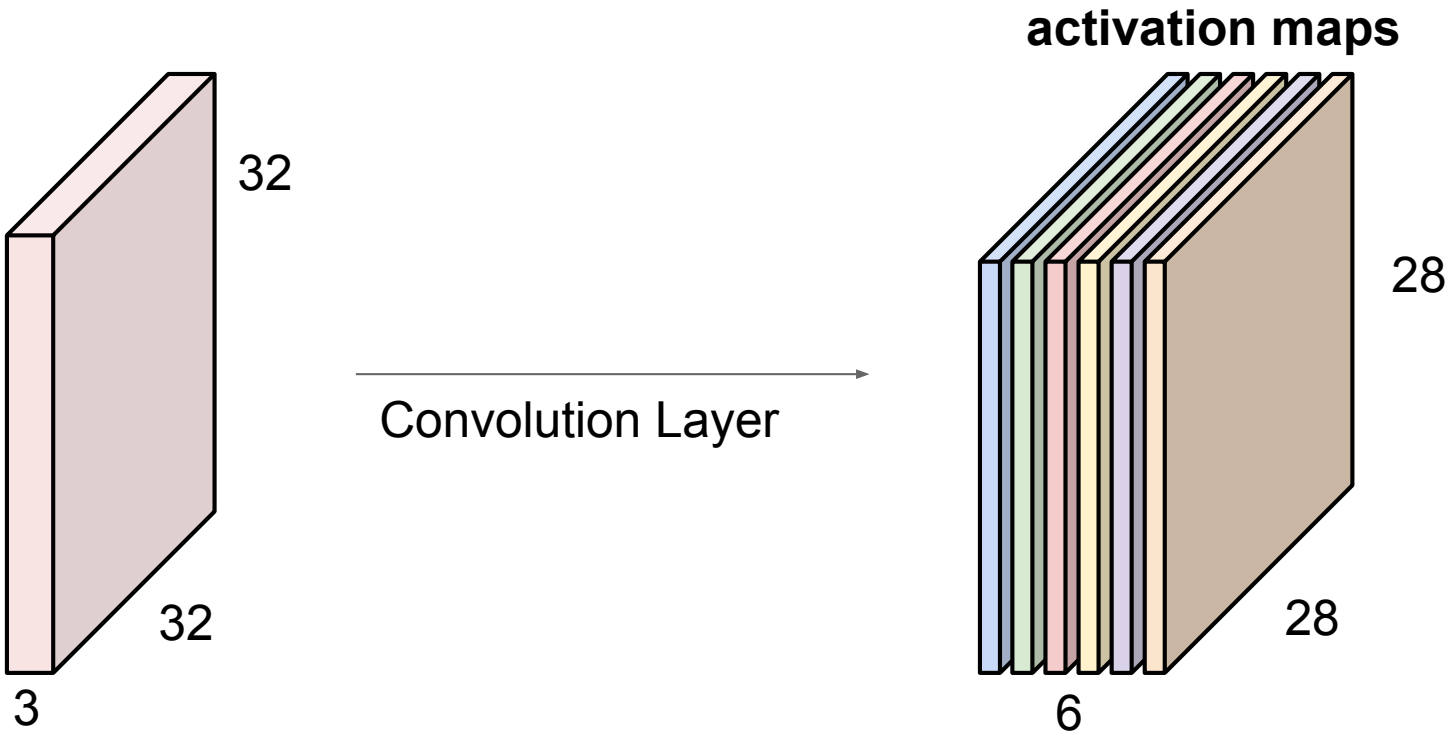
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



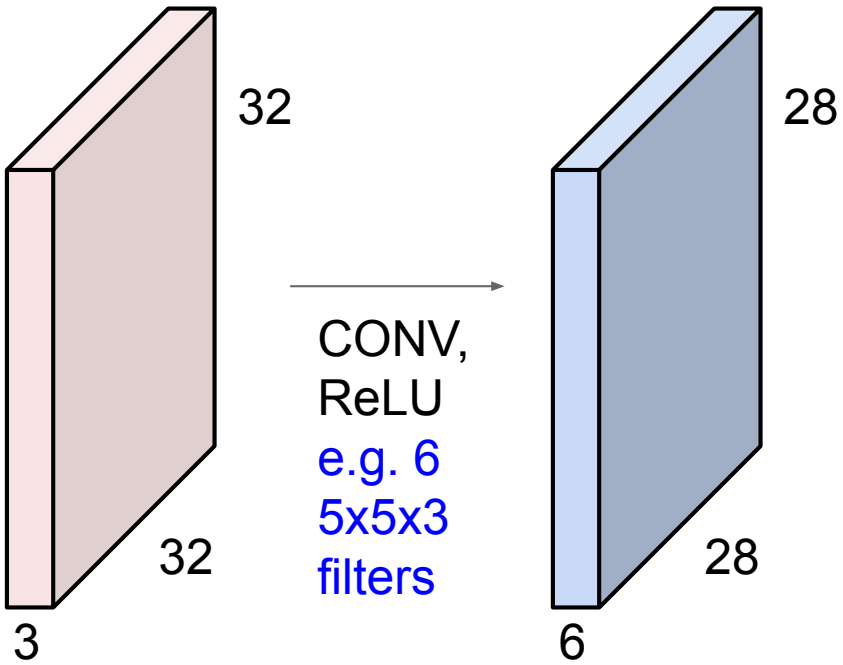
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



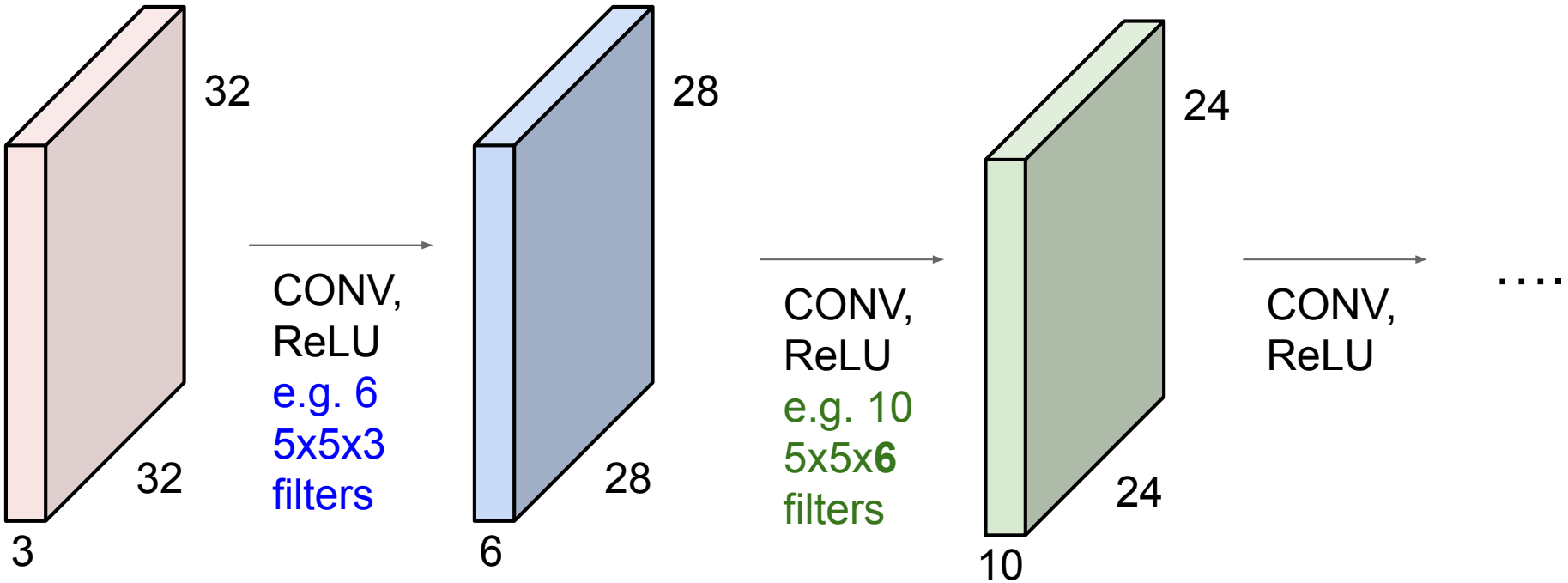
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Convolution Layer



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

Convolution Layer



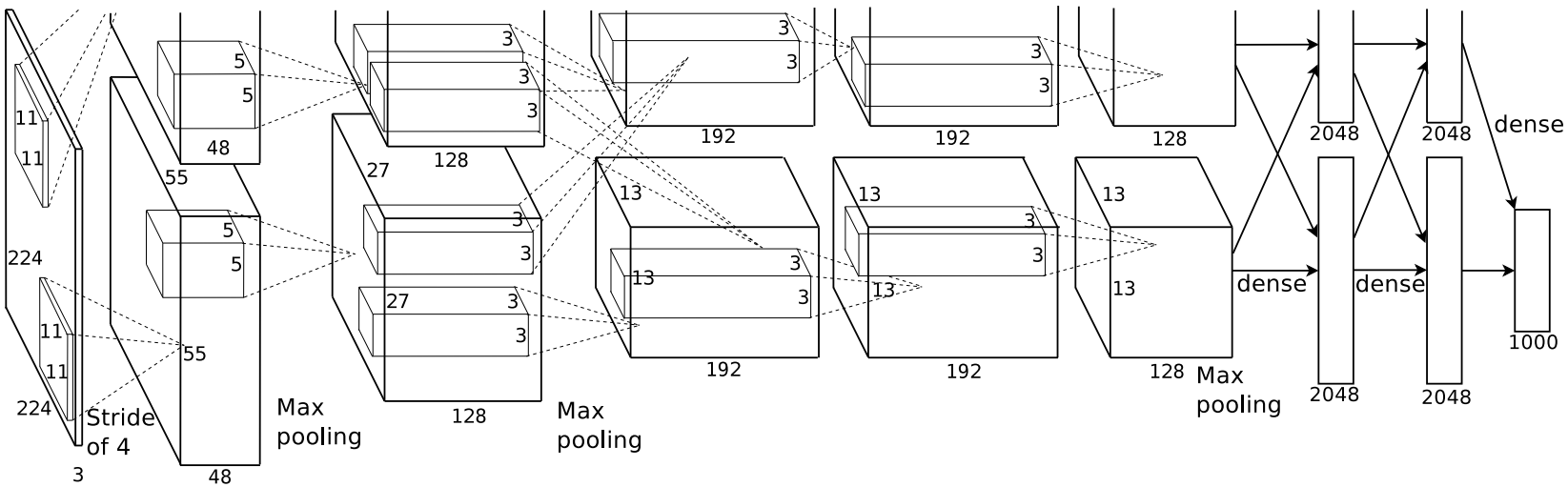
Slides of Fei-Fei Li, Ranjay Krishna, Danfei Xu; Standford CS231n.

Other Types of Layers

Many types of layers exist. Here is a few.

- Max pooling/averge pooling
- Batch normalization
- Dropout

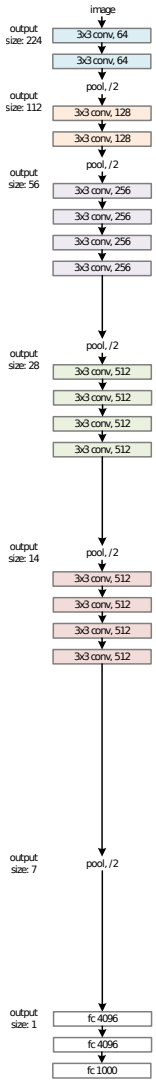
Deep Neural Network Architectures



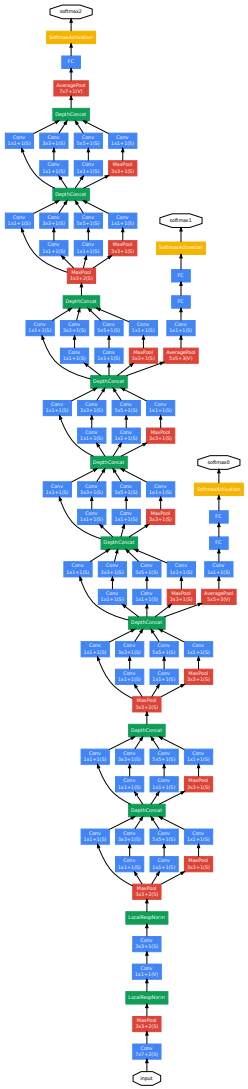
AlexNet

[**Krizhevsky2012ImageNetCW**, **Krizhevsky2012ImageNetCW**]

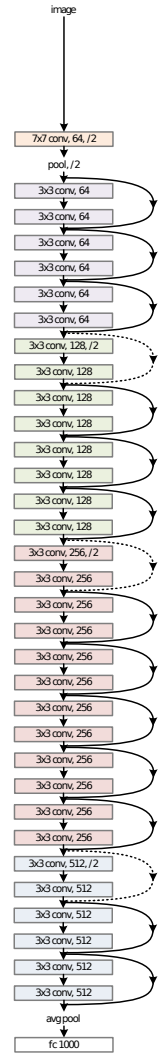
Deep Neural Network Architectures



VGG



GoogLeNet (or Inception)



ResNet

[**simonyan2014very**, **simonyan2014very**]
[**szegedy2015going**, **szegedy2015going**]
[**he2016deep**, **he2016deep**]