Une brève introduction à l'Intelligence Artificielle

Apprentissage profond - Quelques ingrédients

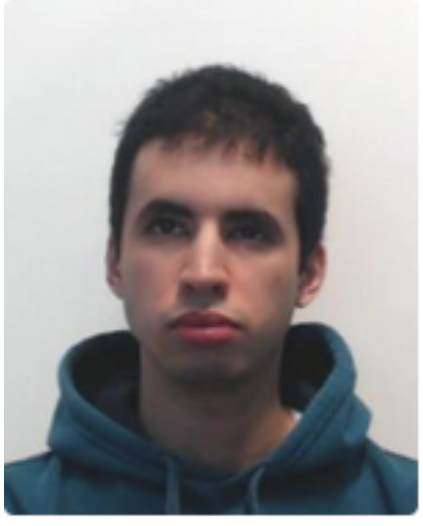
Céline Hudelot

September 7, 2023

Foreword

Other contributors of this course

Dr. Victor Bouvier
Research Scientist, Dataiku



Yassine Ouali
PhD, MICS

Motivations of Deep Learning

Motivations of Deep Learning

Limits of 'traditional' Machine Learning

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,
2. Define an objective (a metric) to maximize $\triangleright$ *or a loss to minimize* ℓ,

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,
2. Define an objective (a metric) to maximize $\triangleright$ *or a loss to minimize* ℓ,
3. Specify a model $\triangleright$ $(f_\theta)_{\theta \in \Theta}$

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,
2. Define an objective (a metric) to maximize $\triangleright$ *or a loss to minimize* ℓ,
3. Specify a model $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
4. Split the data into a train and a test set,

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,
2. Define an objective (a metric) to maximize $\triangleright$ *or a loss to minimize* ℓ,
3. Specify a model $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
4. Split the data into a train and a test set,
5. Train the model $\triangleright$ $\hat{\theta} := \arg\min_\theta \frac{1}{n} \sum_{(x,y) \in \text{train}} \ell(f_\theta(x), y)$

Road map for building a ML model

Business case

- You are working in some company where a support team helps users.
- The boss finds that his support team spends a lot of time dealing spam mails rather than dealing with real user requests.
- You are asked to build a model that detect spam mails.

What do you do?

1. Define the task and collect data,
2. Data cleaning and feature engineering $\triangleright$ 90% of your time! Why..?
3. Define an objective (a metric) to maximize $\triangleright$ *or a loss to minimize* ℓ,
4. Specify a model $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
5. Split the data into a train and a test set,
6. Train the model $\triangleright$ $\hat{\theta} := \arg\min_\theta \frac{1}{n} \sum_{(x,y) \in \text{train}} \ell(f_\theta(x), y)$
7. Evaluate the model according to a metric.

Motivations of Deep Learning

Features engineering: How to 'represent' your data?

Features engineering

Features engineering

Features engineering is the process that consists in transforming the features such that learning the task from the latter is easier (or leads to better generalization) compared to the former.

Features engineering

Features engineering

Features engineering is the process that consists in transforming the features such that learning the task from the latter is easier (or leads to better generalization) compared to the former.

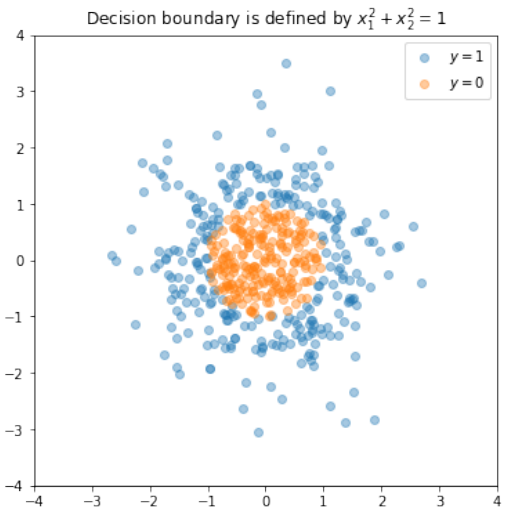
▷ Examples of features engineering

Features engineering

Features engineering is the process that consists in transforming the features such that learning the task from the latter is easier (or leads to better generalization) compared to the former.

▷ Examples of features engineering

Unit circle: $y := (||x||^2 > 1)$ **where** $x \sim \mathcal{N}(0, 1)$

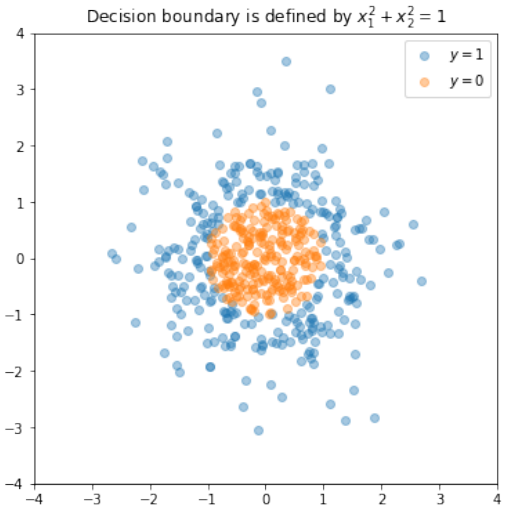


Features engineering

Features engineering is the process that consists in transforming the features such that learning the task from the latter is easier (or leads to better generalization) compared to the former.

▷ Examples of features engineering

Unit circle: $y := (||x||^2 > 1)$ **where** $x \sim \mathcal{N}(0, 1)$



▷ How do you adress this problem? (Notebook session: colab.research.google.com/drive/1NugMhZ9VEE3Mwt50avgFV1hPWXl7N5Wo?usp=sharing)

Representation

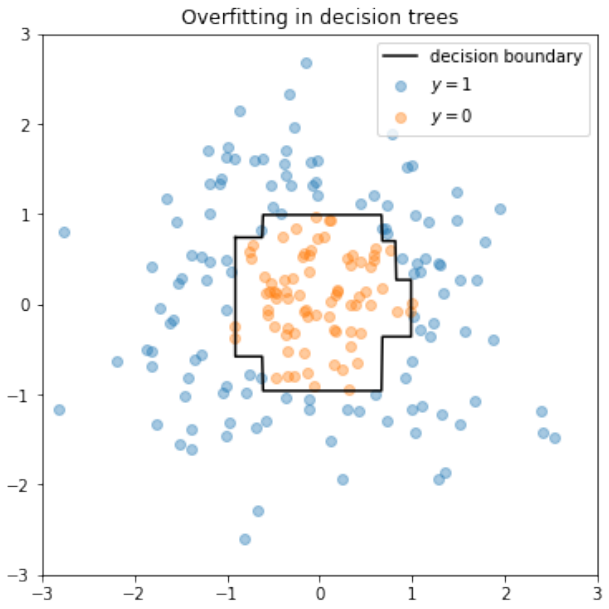
- Features engineering (representation) is mandatory in two cases:

Representation

- Features engineering (representation) is mandatory in two cases:
  1. Our model has not enough capacity for separating the data ▷ *Logistic Regression for the unit circle.*

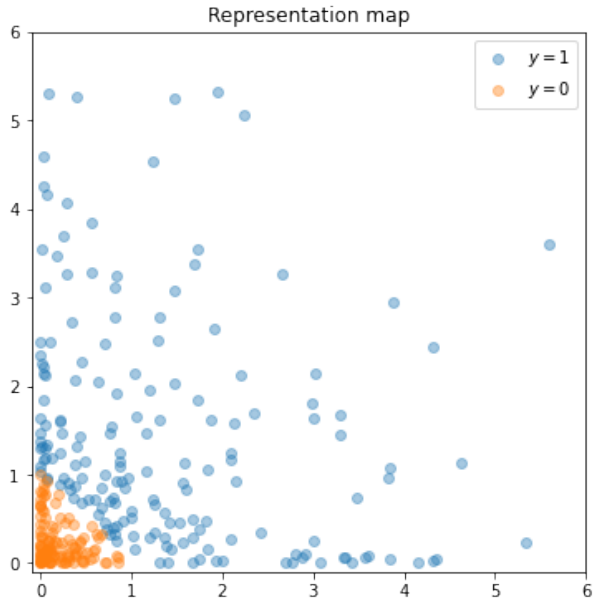
Representation

- Features engineering (representation) is mandatory in two cases:
  1. Our model has not enough capacity for separating the data ▷ *Logistic Regression for the unit circle.*
  2. Our model has too much capacity and overfit the data ▷ *Decision Tree for the unit circle.*



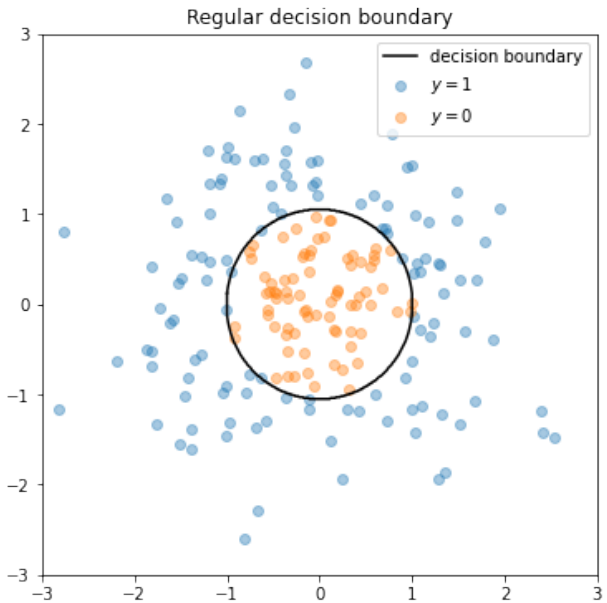
Representation

- Features engineering (representation) is mandatory in two cases:
  1. Our model has not enough capacity for separating the data ▷ *Logistic Regression for the unit circle.*
  2. Our model has too much capacity and overfit the data ▷ *Decision Tree for the unit circle.*
- If provided with a suitable representation ▷ $\varphi(x) := (x_1^2, x_2^2)$



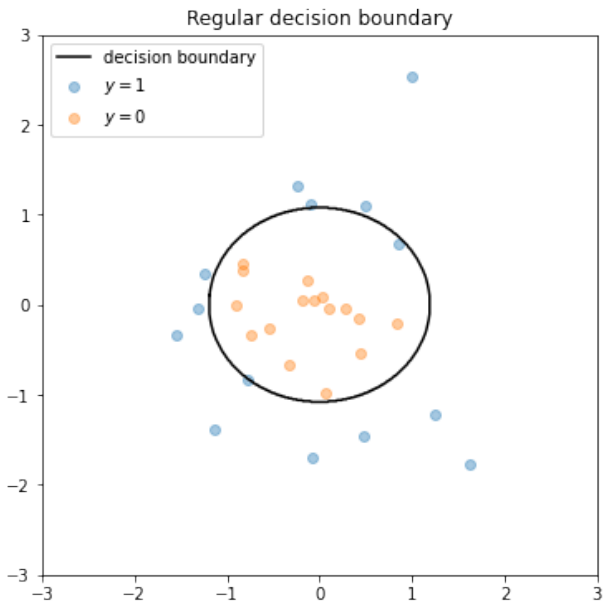
Representation

- Features engineering (representation) is mandatory in two cases:
  1. Our model has not enough capacity for separating the data ▷ *Logistic Regression for the unit circle.*
  2. Our model has too much capacity and overfit the data ▷ *Decision Tree for the unit circle.*
- If provided with a suitable representation ▷ $\varphi(x) := (x_1^2, x_2^2)$
  - model benefits from a strong regularity ▷ *circle shape of the decision boundary for the unit circle*.



Representation

- Features engineering (representation) is mandatory in two cases:
  1. Our model has not enough capacity for separating the data ▷ *Logistic Regression for the unit circle.*
  2. Our model has too much capacity and overfit the data ▷ *Decision Tree for the unit circle.*
- If provided with a suitable representation ▷ $\varphi(x) := (x_1^2, x_2^2)$
  - model benefits from a strong regularity ▷ *circle shape of the decision boundary for the unit circle*.
  - model can generalize with few samples ▷ $\sim$ 30 are enough!



Representation

Model benefits from a strong regularity...

A representation encodes our inductive bias ▷

the hypothesis space is biased to solutions we found 'plausible'.

Representation

Model benefits from a strong regularity...

A representation encodes our inductive bias ▷

the hypothesis space is biased to solutions we found 'plausible'.

Is it easy to define a good representation?

- Does the data scientist know *a priori* the shape of the solution?
- Does it exists a representation that can be reasonably hand-crafted?

Representation

Model benefits from a strong regularity...

A representation encodes our inductive bias ▷

the hypothesis space is biased to solutions we found 'plausible'.

Is it easy to define a good representation?

- Does the data scientist know *a priori* the shape of the solution?
- Does it exists a representation that can be reasonably hand-crafted?

Maybe, you have already work with data representation

Representation

Model benefits from a strong regularity...
A representation encodes our inductive bias ▷

the hypothesis space is biased to solutions we found 'plausible'.

Is it easy to define a good representation?

- Does the data scientist know *a priori* the shape of the solution?
- Does it exists a representation that can be reasonably hand-crafted?

Maybe, you have already work with data representation ▷ Kernel trick in SVM!

Learning representations

Input image
256 x 256

Learning representations

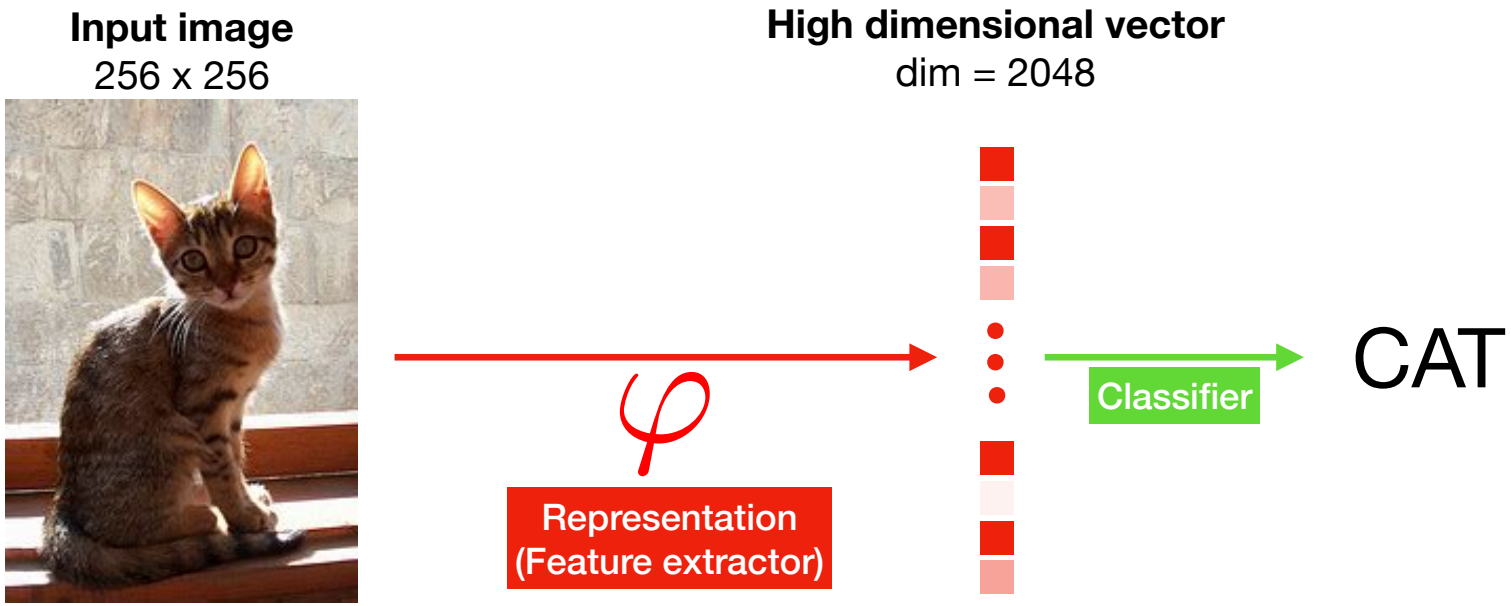
Input image
256 x 256

φ

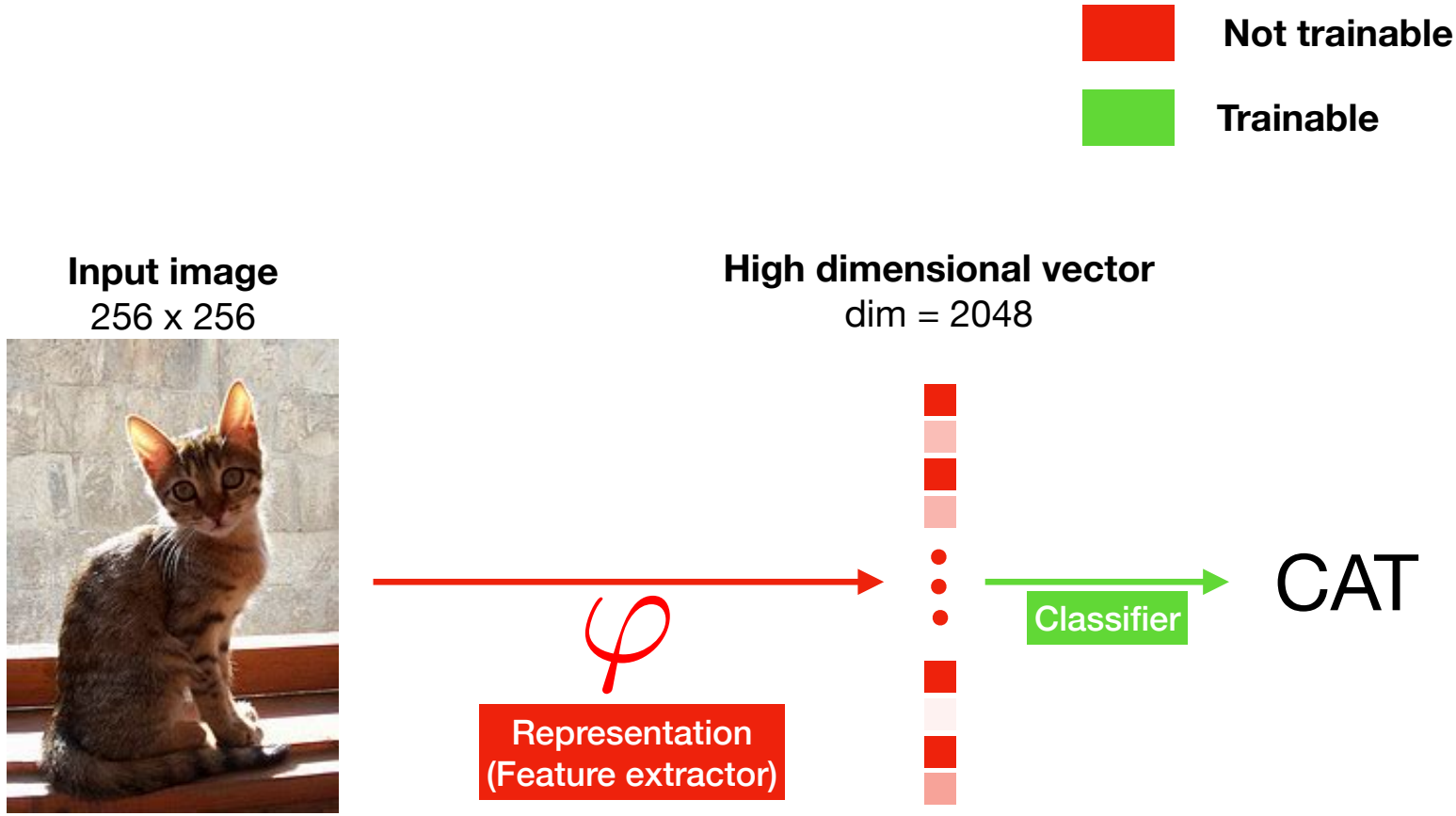
Representation
(Feature extractor)

High dimensional vector
dim = 2048

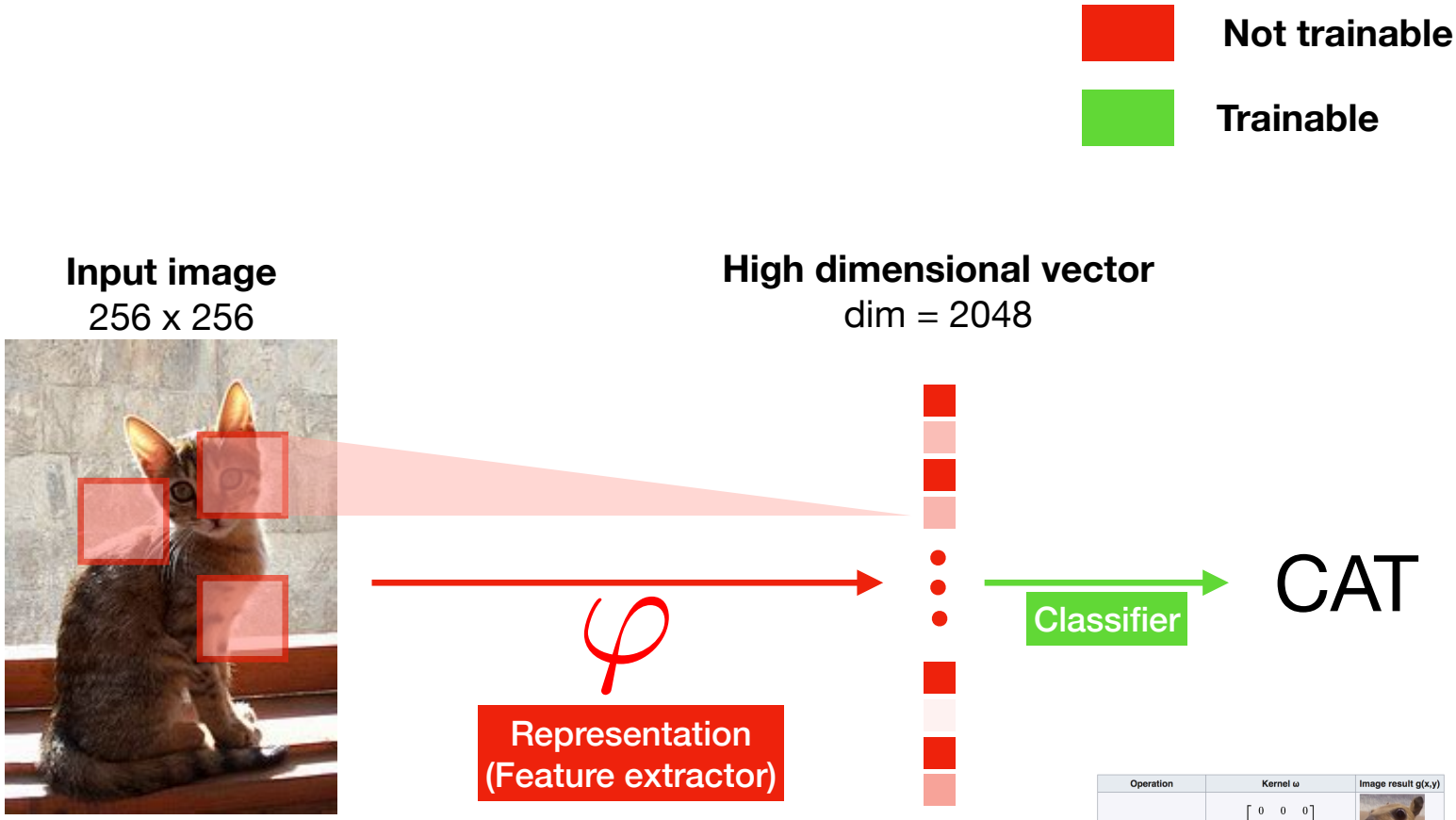
Learning representations



Learning representations



Learning representations

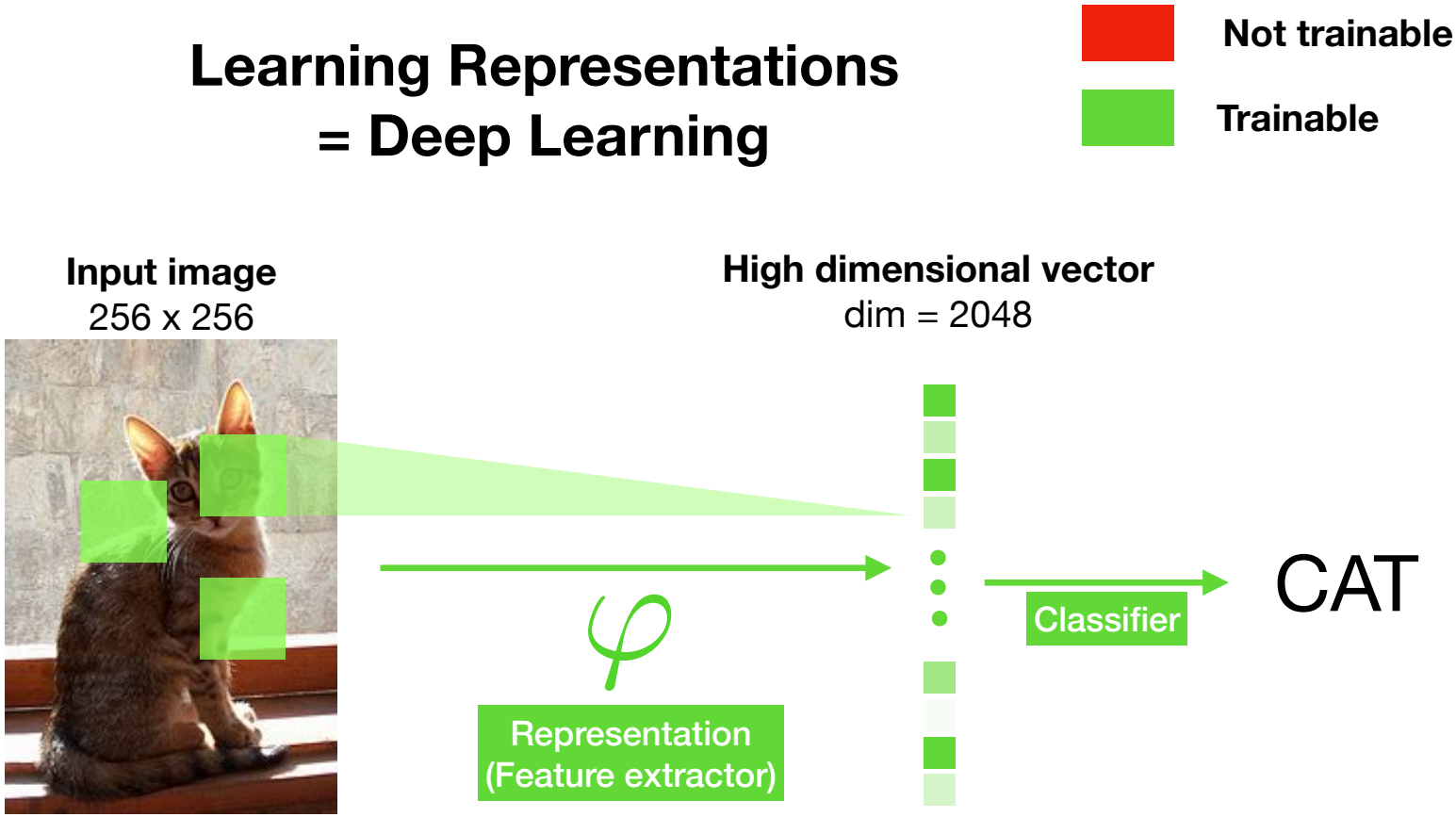


Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Source wikipedia

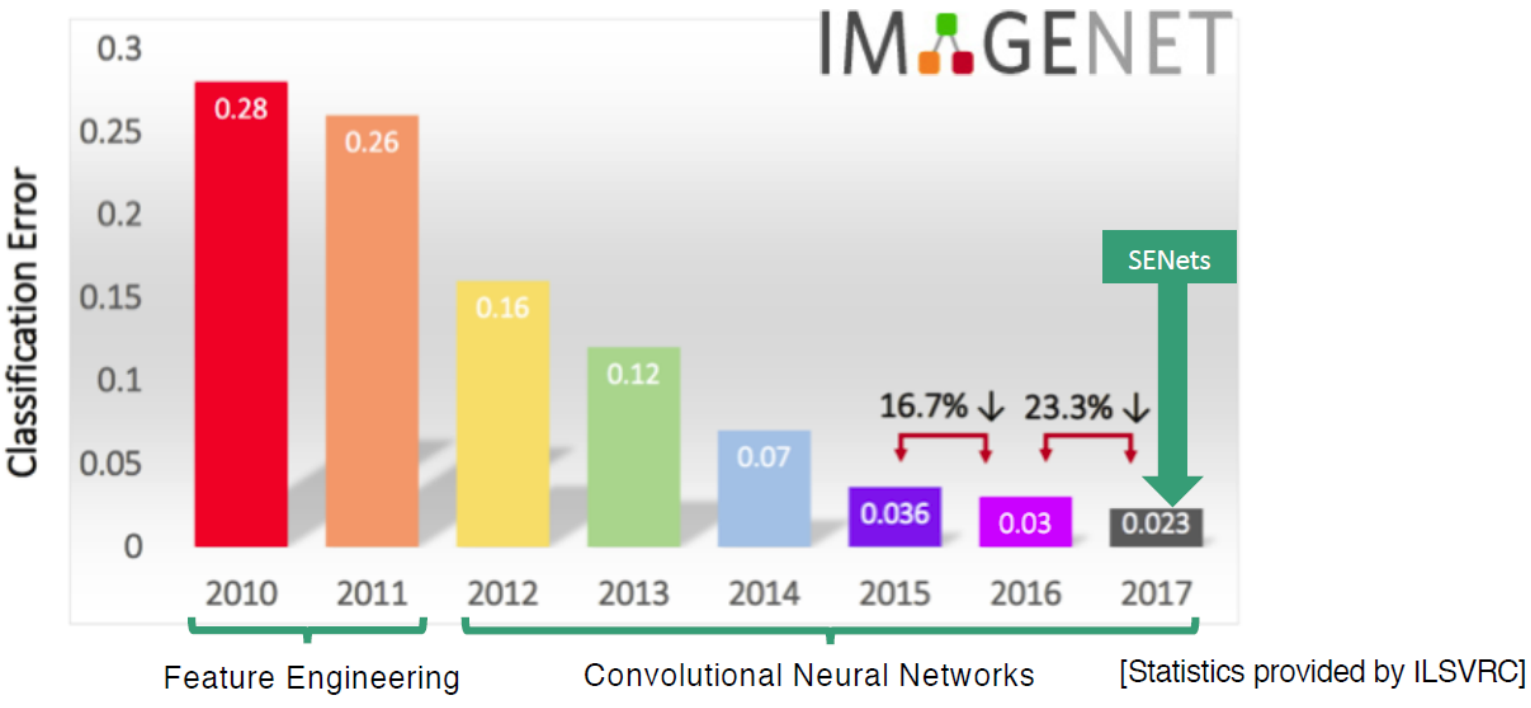
Learning representations

Learning Representations = Deep Learning



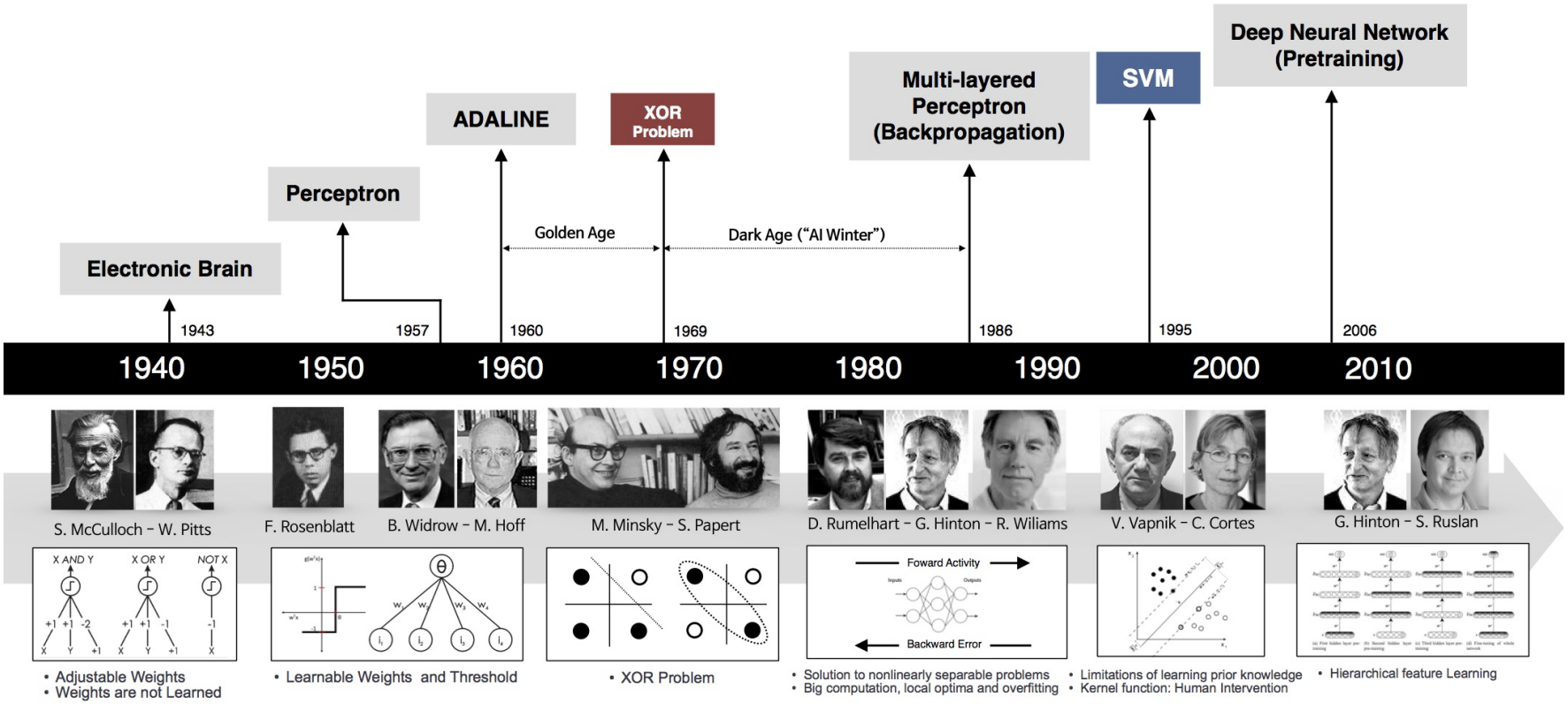
- Representation is a function from the input space to the features space
- Defined by a large numbers of parameters
- Deep Learning is finding strong inductive bias for learning a good representation (convolutional neural network, recurrent neural network, transformers…)

Learning representations



The Deep Learning timeline

The Deep Learning timeline



beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.
- 'Deep' refers to the process of composing (stacking) simple functions in order to build an over-parametrized representation.

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.
- 'Deep' refers to the process of composing (stacking) simple functions in order to build an over-parametrized representation.
- Inductive bias (assumptions made about the shape of the learner) reduces the number of parameters.

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.
- 'Deep' refers to the process of composing (stacking) simple functions in order to build an over-parametrized representation.
- Inductive bias (assumptions made about the shape of the learner) reduces the number of parameters.

To go deeper in deep learning

1. Linear network (Rosenblatt's perceptron)

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.
- 'Deep' refers to the process of composing (stacking) simple functions in order to build an over-parametrized representation.
- Inductive bias (assumptions made about the shape of the learner) reduces the number of parameters.

To go deeper in deep learning

1. Linear network (Rosenblatt's perceptron)
2. Multi-layers perceptron.

What we have learnt so far

- Deep Learning is a new Machine Learning approach where the representation (features extractor) is learned.
- Representation is (most of the time) an over-parametrized mapping.
- 'Deep' refers to the process of composing (stacking) simple functions in order to build an over-parametrized representation.
- Inductive bias (assumptions made about the shape of the learner) reduces the number of parameters.

To go deeper in deep learning

1. Linear network (Rosenblatt's perceptron)
2. Multi-layers perceptron.
3. Training a neural network.

Linear network

Linear Neural Network

Two examples

- **Regression:** Linear regression $\triangleright$ $y = x^\top \theta + \mathcal{N}(0, \sigma^2)$
- **Classification:** Logistic regression $\triangleright$ $p(y|x) = \sigma(x^\top \theta + \mathcal{N}(0, \sigma^2))$ where $\sigma(x) := 1/(1 + \exp(-x))$.

Linear Neural Network

Two examples

- **Regression:** Linear regression ▷ $y = x^\top \theta + \mathcal{N}(0, \sigma^2)$
- **Classification:** Logistic regression ▷ $p(y|x) = \sigma(x^\top \theta + \mathcal{N}(0, \sigma^2))$ where $\sigma(x) := 1/(1 + \exp(-x))$.

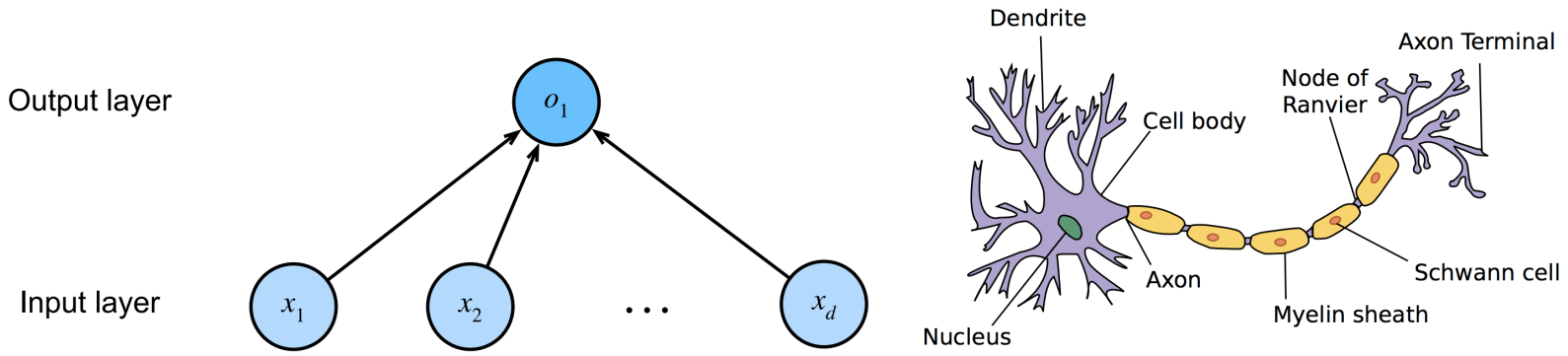


Figure 1: (Left): Illustration of a linear neural network from Rosentblatt. (Right) Biological inspiration of artifical neurons from Warren McCulloch and Walter Pitts. From `d2l.ai/d2l-en.pdf`.

Linear Neural Network

Linear Neural Network

- **Regression:** Linear regression $\triangleright$ $f_\theta(x) := x^\top w + b$
- **Classification:** Logistic regression $\triangleright$ $f_\theta(x) := \sigma(x^\top w + b)$.

with $\theta = (w, b)$.

Linear Neural Network

Linear Neural Network

- **Regression:** Linear regression $\triangleright$ $f_\theta(x) := x^\top w + b$
- **Classification:** Logistic regression $\triangleright$ $f_\theta(x) := \sigma(x^\top w + b)$.

with $\theta = (w, b)$. $\triangleright$ Learning is finding the optimal θ.

Defining a loss using Maximum Likelihood Estimation

- **Regression:** $\ell(y, f_\theta(x)) := (y - f_\theta(x))^2$
- **Classification:** $\ell(y, f_\theta(x)) := -y \log(f_\theta(x)) - (1 - y) \log(1 - f_\theta(x))$

$$\mathcal{L}(\theta) := \frac{1}{n} \sum_{(x,y)} \ell(y, f_\theta(x))$$

Linear Neural Network

Linear Neural Network

- **Regression:** Linear regression $\triangleright$ $f_\theta(x) := x^\top w + b$
- **Classification:** Logistic regression $\triangleright$ $f_\theta(x) := \sigma(x^\top w + b)$.

with $\theta = (w, b)$. $\triangleright$ Learning is finding the optimal θ.

Defining a loss using Maximum Likelihood Estimation

- **Regression:** $\ell(y, f_\theta(x)) := (y - f_\theta(x))^2$
- **Classification:** $\ell(y, f_\theta(x)) := -y \log(f_\theta(x)) - (1 - y) \log(1 - f_\theta(x))$

$$\mathcal{L}(\theta) := \frac{1}{n} \sum_{(x,y)} \ell(y, f_\theta(x))$$

$\triangleright$ Learning is $\hat{\theta} := \arg\min_\theta \mathcal{L}(\theta)$

Linear Neural Network

Linear Neural Network

- **Regression:** Linear regression $\triangleright$ $f_\theta(x) := x^\top w + b$
- **Classification:** Logistic regression $\triangleright$ $f_\theta(x) := \sigma(x^\top w + b)$.

with $\theta = (w, b)$. $\triangleright$ Learning is finding the optimal θ.

Defining a loss using Maximum Likelihood Estimation

- **Regression:** $\ell(y, f_\theta(x)) := (y - f_\theta(x))^2$
- **Classification:** $\ell(y, f_\theta(x)) := -y \log(f_\theta(x)) - (1 - y) \log(1 - f_\theta(x))$

$$\mathcal{L}(\theta) := \frac{1}{n} \sum_{(x,y)} \ell(y, f_\theta(x))$$

$\triangleright$ Learning is $\hat{\theta} := \arg\min_\theta \mathcal{L}(\theta)$

We need more flexible learning process (optimization procedure)

Linear network

Gradient Descent

Gradient Descent

Gradient Descent (GD)

Let J a function from $\mathbb{R}^d \to \mathbb{R}$, Gradient Descent consists in localizing a **local minimum** of J as follows:

- Initialize $x_0 \in \mathbb{R}^d$.
- For a given number of iterations:

$$x_{t+1} \leftarrow x_t - \alpha(\nabla_x J)(x_t)$$

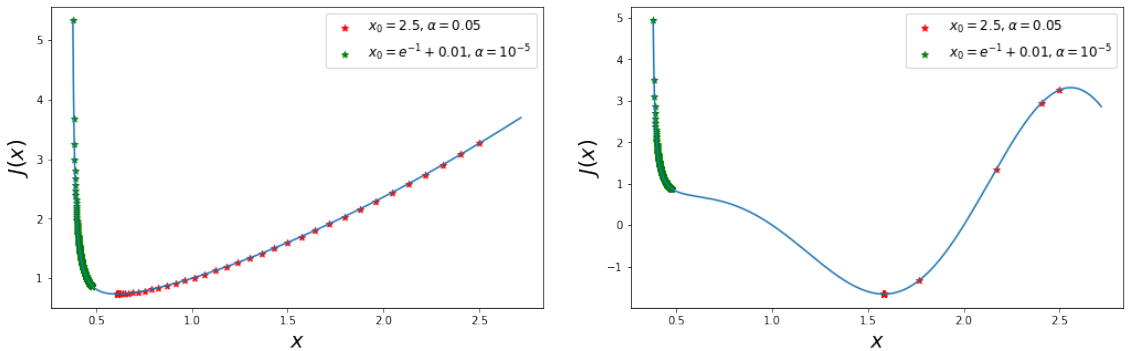


Figure 2: (Left) GD can minimize arbitrary complicated functions, here $J(x) = x^2/(1 + \log(x))$. (Right)Gradient descent with different initialization may lead to different minima, here $J(x) = x^2 \sin(\pi x)/(1 + \log(x))$

From Gradient Descent to Learning

Momentum

$$x_{t+1} \leftarrow x_t - \alpha(t)(\nabla_x J)(x_t)$$

- The choice of momentum $\alpha(t)$ is often a trade-off between **speed** and **accuracy** when finding a minimum.
- `ruder.io/optimizing-gradient-descent`

From Gradient Descent to Learning

Momentum

$$x_{t+1} \leftarrow x_t - \alpha(t)(\nabla_x J)(x_t)$$

- The choice of momentum $\alpha(t)$ is often a trade-off between **speed** and **accuracy** when finding a minimum.
- `ruder.io/optimizing-gradient-descent`

Learning by Gradient Descent

$$\theta_{t+1} \leftarrow \theta_t - \eta\alpha(t)(\nabla_\theta \mathcal{L})(\theta_t)$$

- η is the **learning rate**.
- Each iteration needs a **full-scan** of the dataset

From Gradient Descent to Learning

Momentum

$$x_{t+1} \leftarrow x_t - \alpha(t)(\nabla_x J)(x_t)$$

- The choice of momentum $\alpha(t)$ is often a trade-off between **speed** and **accuracy** when finding a minimum.
- `ruder.io/optimizing-gradient-descent`

Learning by Gradient Descent

$$\theta_{t+1} \leftarrow \theta_t - \eta\alpha(t)(\nabla_\theta \mathcal{L})(\theta_t)$$

- η is the **learning rate**.
- Each iteration needs a **full-scan** of the dataset ▷ Stochastic Gradient Descent.

Linear network

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic Gradient Descent is Gradient Descent where each iteration is performed on a random subset of the dataset (typically of size between 16 and 256 samples).

- **Input:** Θ parameters, η learning rate, b batch size, $\mathcal{D}$ dataset.
- Initialize $\theta_0 \in \Theta$.
- For a given number of iterations:

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic Gradient Descent is Gradient Descent where each iteration is performed on a random subset of the dataset (typically of size between 16 and 256 samples).

- **Input:** Θ parameters, η learning rate, b batch size, $\mathcal{D}$ dataset.
- Initialize $\theta_0 \in \Theta$.
- For a given number of iterations:
  - Sample $\mathcal{B} \sim \mathcal{D}$ such that $|\mathcal{B}| = b$.

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic Gradient Descent is Gradient Descent where each iteration is performed on a random subset of the dataset (typically of size between 16 and 256 samples).

- **Input:** Θ parameters, η learning rate, b batch size, $\mathcal{D}$ dataset.
- Initialize $\theta_0 \in \Theta$.
- For a given number of iterations:
  - Sample $\mathcal{B} \sim \mathcal{D}$ such that $|\mathcal{B}| = b$.
  - Compute the batch loss $\mathcal{L}^{\mathcal{B}}(\theta) := \frac{1}{b} \sum_{(x,y)\in\mathcal{B}} \ell(y, f_\theta(x))$.

Stochastic Gradient Descent

Stochastic Gradient Descent

Stochastic Gradient Descent is Gradient Descent where each iteration is performed on a random subset of the dataset (typically of size between 16 and 256 samples).

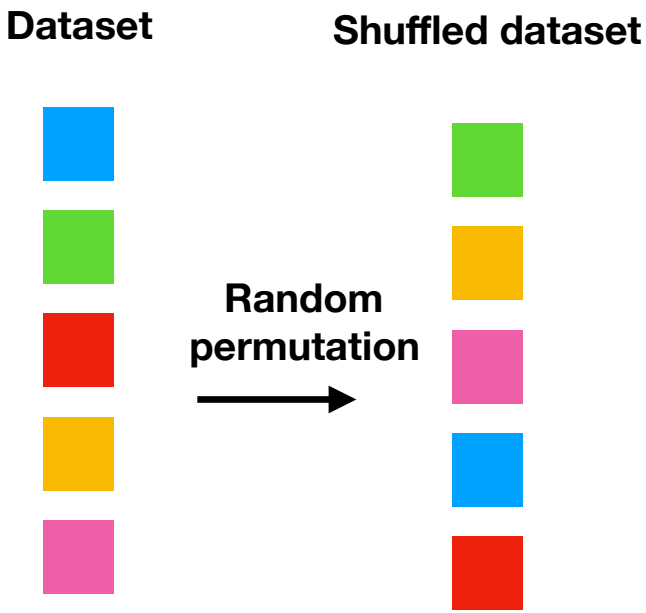
- **Input:** Θ parameters, η learning rate, b batch size, $\mathcal{D}$ dataset.
- Initialize $\theta_0 \in \Theta$.
- For a given number of iterations:
  - Sample $\mathcal{B} \sim \mathcal{D}$ such that $|\mathcal{B}| = b$.
  - Compute the batch loss $\mathcal{L}^{\mathcal{B}}(\theta) := \frac{1}{b} \sum_{(x,y)\in\mathcal{B}} \ell(y, f_\theta(x))$.
  - Update parameters according to:

$$\theta_{t+1} \leftarrow \theta_t - \eta\alpha(t)(\nabla_\theta \mathcal{L}^{\mathcal{B}})(\theta_t)$$

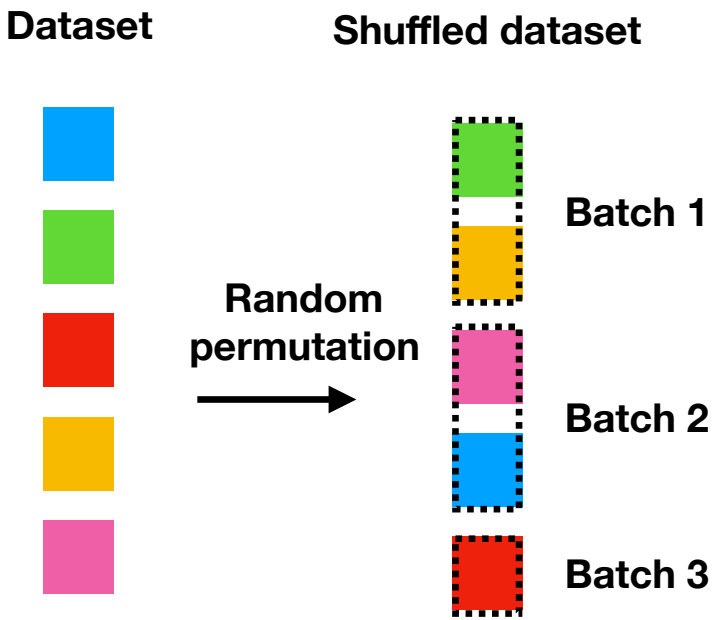
Stochastic Gradient Descent

Dataset

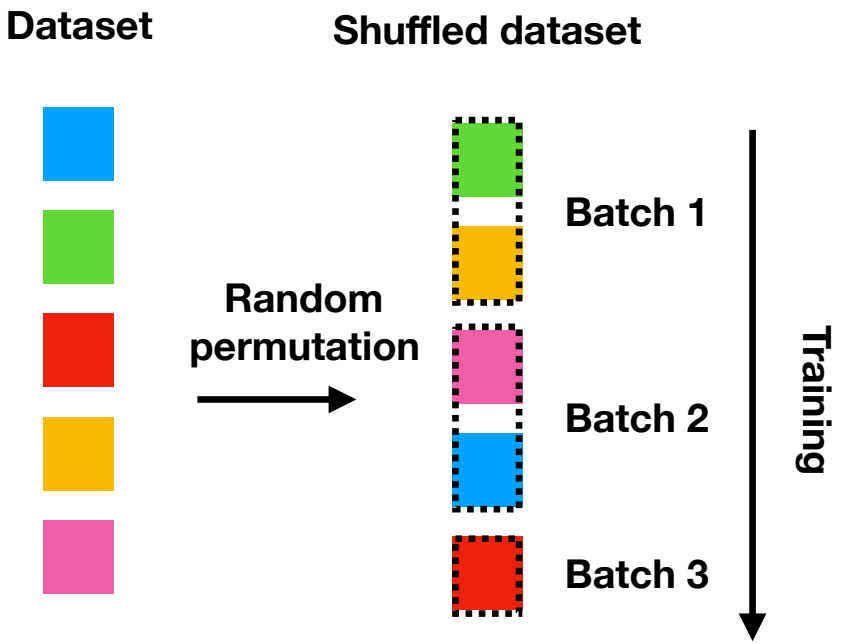
Stochastic Gradient Descent



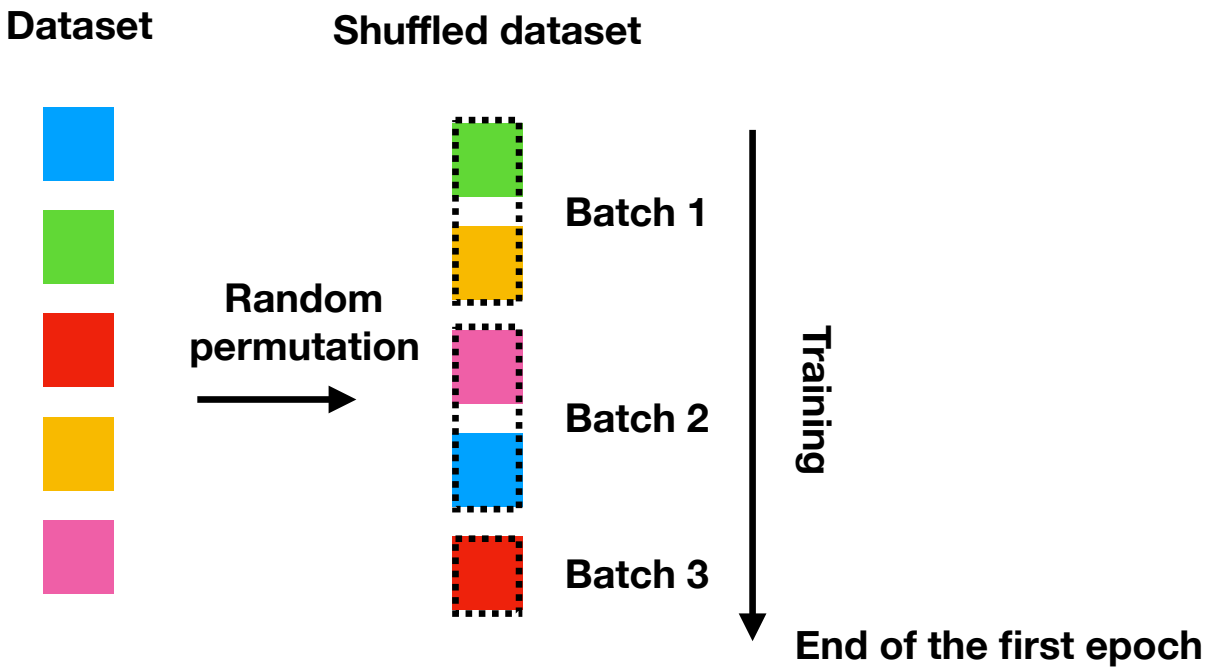
Stochastic Gradient Descent



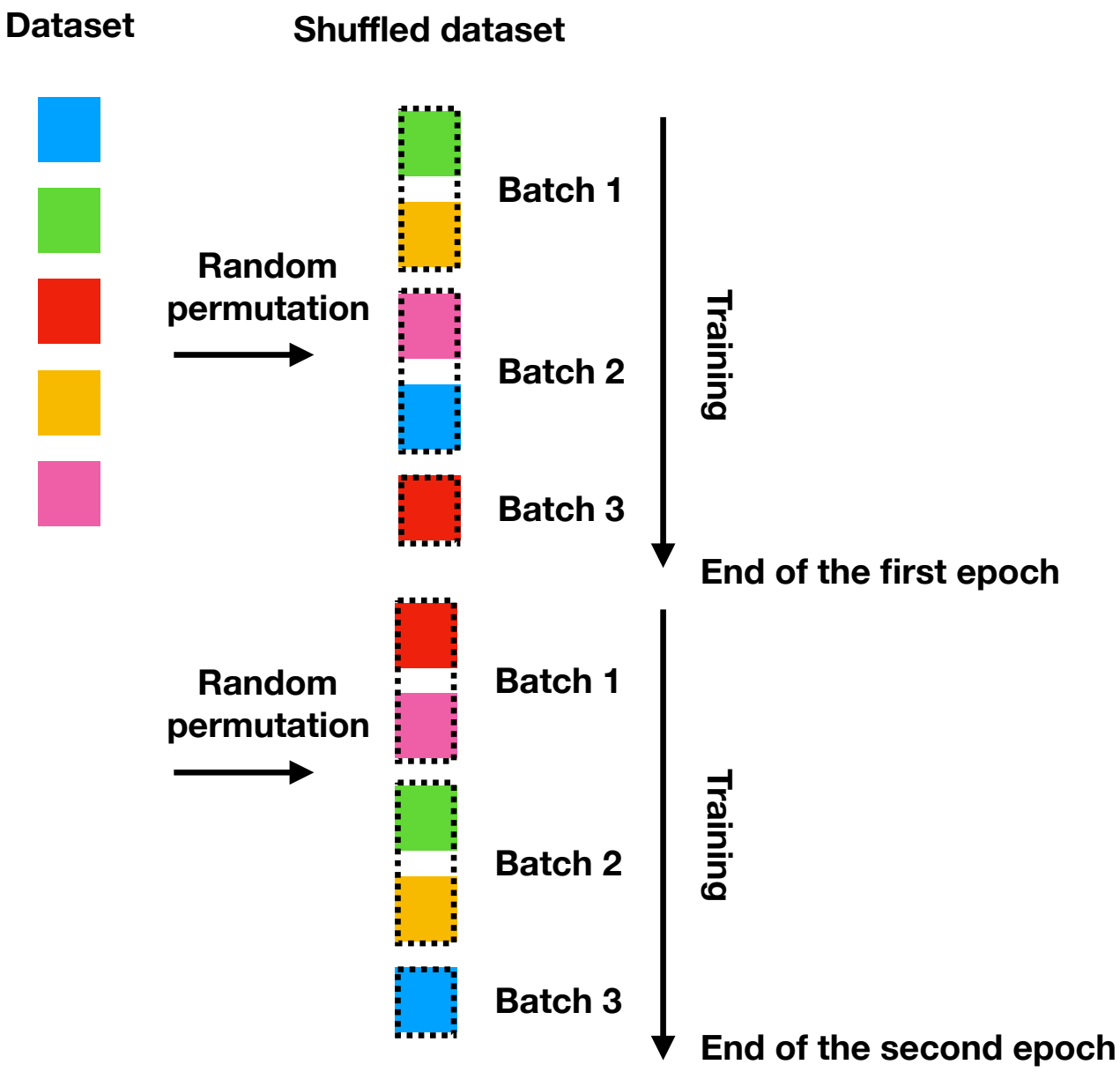
Stochastic Gradient Descent



Stochastic Gradient Descent



Stochastic Gradient Descent



Linear network

Limitations

Limitations of the Linear Neural Network

Quick summary

- Linear neural network: $f_\theta(x) := a(x^\top w + b)$

Limitations of the Linear Neural Network

Quick summary

- Linear neural network: $f_\theta(x) := a(x^\top w + b)$
- w are weights, and b is the bias of the layer, a is an activation function.

Limitations of the Linear Neural Network

Quick summary

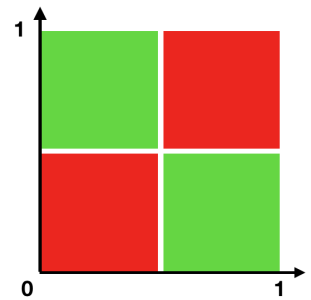
- Linear neural network: $f_\theta(x) := a(x^\top w + b)$
- w are weights, and b is the bias of the layer, a is an activation function.
- SGD allows to learn $\theta := (w, b)$ for arbitrary loss.

Limitations of the Linear Neural Network

Quick summary

- Linear neural network: $f_\theta(x) := a(x^\top w + b)$
- w are weights, and b is the bias of the layer, a is an activation function.
- SGD allows to learn $\theta := (w, b)$ for arbitrary loss.

Limitations: the overkill XOR function



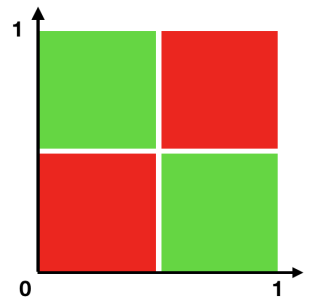
- $(0,0) \mapsto 0$, $(1,1) \mapsto 0$, $(1,0) \mapsto 1$ and $(0,1) \mapsto 1$.
- A linear neural network can not learn the XOR function.

Limitations of the Linear Neural Network

Quick summary

- Linear neural network: $f_\theta(x) := a(x^\top w + b)$
- w are weights, and b is the bias of the layer, a is an activation function.
- SGD allows to learn $\theta := (w, b)$ for arbitrary loss.

Limitations: the overkill XOR function



- $(0,0) \mapsto 0$, $(1,1) \mapsto 0$, $(1,0) \mapsto 1$ and $(0,1) \mapsto 1$.
- A linear neural network can not learn the XOR function.

▷ Because we need a representation layer!

Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP)

One-hidden layer Neural Network

One-hidden layer Neural Network

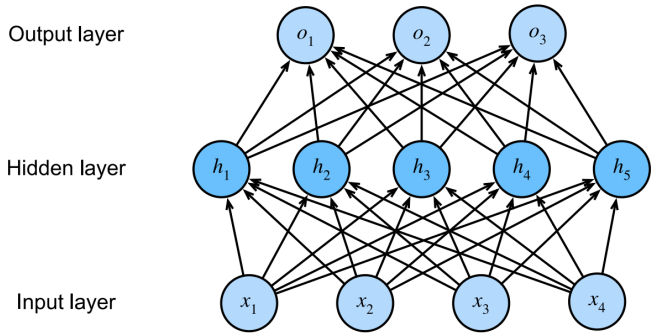


Figure 3: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

Forward pass: $x \longrightarrow h \longrightarrow o$

One-hidden layer Neural Network

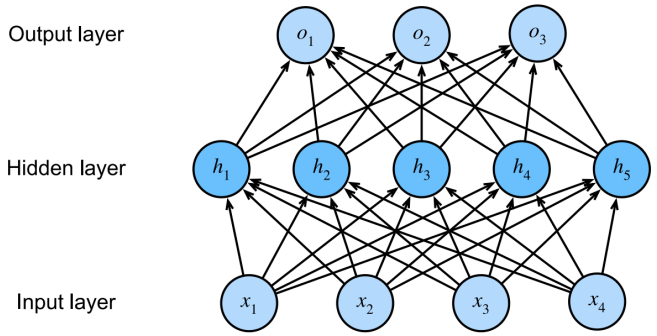


Figure 3: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

Forward pass: $x \longrightarrow h \longrightarrow o$

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$

One-hidden layer Neural Network

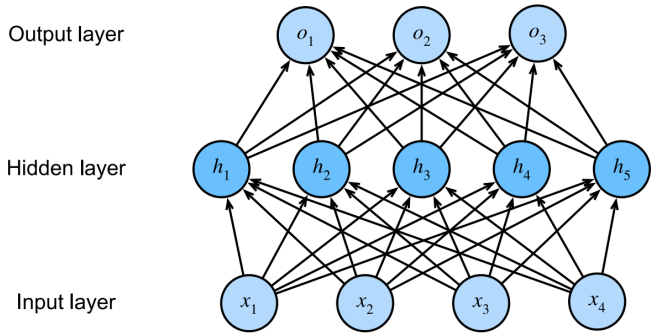


Figure 3: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

Forward pass: $x \longrightarrow h \longrightarrow o$

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$

One-hidden layer Neural Network

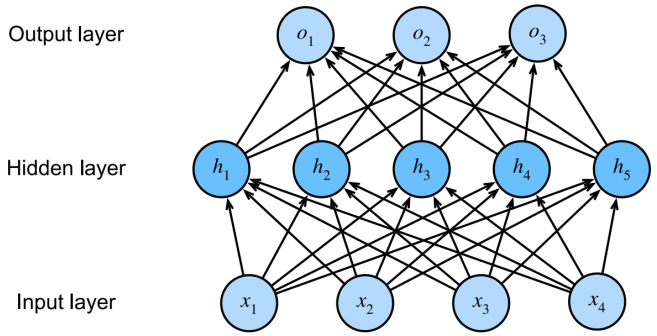


Figure 3: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

Forward pass: $x \longrightarrow h \longrightarrow o$

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$
- $\theta := (w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)})$ defines $o = f_\theta(x)$

One-hidden layer Neural Network

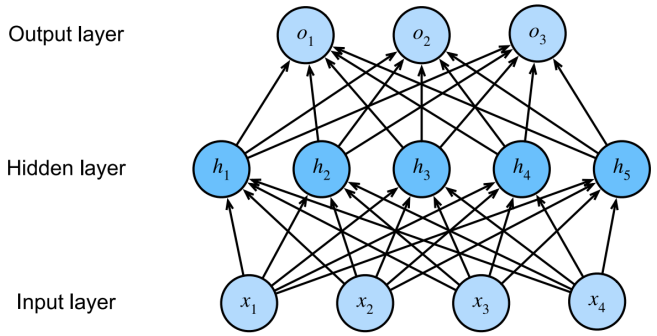


Figure 3: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

Forward pass: $x \longrightarrow h \longrightarrow o$

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$
- $\theta := (w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)})$ defines $o = f_\theta(x)$

$\triangleright$ h is the (hidden) representation of x!

Universal approximation theorem

Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$

Universal approximation theorem

Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$

The role of non-linearity

If $a^{(1)}$ is the identity function $\Rightarrow$ the linear network. Typical $a^{(1)}$:

- Sigmoid: $\sigma(x) := \frac{1}{1+\exp(-x)}$
- Tangeante-hyperbolic: $\mathrm{Tanh}(x) := \frac{e^{-x}-e^{x}}{e^{-x}+e^{x}}$
- Rectified Linear Unit: $\mathrm{ReLU}(x) := \max(0, x)$

Universal approximation theorem

Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$

The role of non-linearity

If $a^{(1)}$ is the identity function $\Rightarrow$ the linear network. Typical $a^{(1)}$:

- Sigmoid: $\sigma(x) := \frac{1}{1+\exp(-x)}$
- Tangeante-hyperbolic: $\mathrm{Tanh}(x) := \frac{e^{-x}-e^{x}}{e^{-x}+e^{x}}$
- Rectified Linear Unit: $\mathrm{ReLU}(x) := \max(0, x)$

Universal approximation theorem

For any continuous function on a compact, it exists a One-hidden layer network with continuous, bounded, non-constant activation, which achieves uniformly an arbitrary small error on the compact.

Universal approximation theorem

Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)} + b^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)} + b^{(2)}\right)$

The role of non-linearity

If $a^{(1)}$ is the identity function $\Rightarrow$ the linear network. Typical $a^{(1)}$:

- Sigmoid: $\sigma(x) := \frac{1}{1+\exp(-x)}$
- Tangeante-hyperbolic: $\mathrm{Tanh}(x) := \frac{e^{-x}-e^{x}}{e^{-x}+e^{x}}$
- Rectified Linear Unit: $\mathrm{ReLU}(x) := \max(0, x)$

Universal approximation theorem

For any continuous function on a compact, it exists a One-hidden layer network with continuous, bounded, non-constant activation, which achieves uniformly an arbitrary small error on the compact. $\triangleright$ Increases the number of units (dimension) of h!

Multi-Layer Perceptron (MLP)

Training a MLP by Back-propagation

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers ▷ That's why we *"Back-propagate errors"*.

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers ▷ That's why we *"Back-propagate errors"*.
- Usually we distinguish two passes:

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers ▷ That's why we *"Back-propagate errors"*.
- Usually we distinguish two passes:
  - **forward pass:** From inputs to outputs (network as a function)

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers ▷ That's why we *"Back-propagate errors"*.
- Usually we distinguish two passes:
  - **forward pass:** From inputs to outputs (network as a function)
  - **backward pass:** Gradient from outputs to inputs (chain rule)

Back-propagation

Backpropagation demystified

- Backprop is a memory efficient algorithm for computing gradients of MLP's parameters.
- It is based on the chain rule for computing derivatives.
- Roughly, for computing the gradient of a layer, we use gradient of upper layers ▷ That's why we *"Back-propagate errors"*.
- Usually we distinguish two passes:
  - **forward pass:** From inputs to outputs (network as a function)
  - **backward pass:** Gradient from outputs to inputs (chain rule)

Chain rule

Let the computational graph $x \longrightarrow y \longrightarrow z$

$$\frac{\partial z}{\partial x} = \mathrm{prod}\left(\frac{\partial z}{\partial y}, \frac{\partial y}{\partial x}\right)$$

where prod is the multiplication if variables are real, matrix product if vectors, ...

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

- **Gradient wrt** $w^{(2)}$: $\frac{\partial \ell}{\partial w^{(2)}}$

$$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial w^{(2)}} = \underbrace{\frac{\partial \ell}{\partial o}}_{G_o} h^\top \underbrace{a^{(2)\prime}(h^\top w^{(2)} + b)}_{G_2} = G_o(h^\top G_2)$$

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

- **Gradient wrt** $w^{(2)}$: $\frac{\partial \ell}{\partial w^{(2)}}$

$$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial w^{(2)}} = \underbrace{\frac{\partial \ell}{\partial o}}_{G_o} h^\top \underbrace{a^{(2)\prime}(h^\top w^{(2)} + b)}_{G_2} = G_o(h^\top G_2)$$

- **Gradient wrt** $w^{(1)}$: $\frac{\partial \ell}{\partial w^{(1)}}$

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

- **Gradient wrt** $w^{(2)}$: $\frac{\partial \ell}{\partial w^{(2)}}$

$$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial w^{(2)}} = \underbrace{\frac{\partial \ell}{\partial o}}_{G_o} h^\top \underbrace{a^{(2)\prime}(h^\top w^{(2)} + b)}_{G_2} = G_o(h^\top G_2)$$

- **Gradient wrt** $w^{(1)}$: $\frac{\partial \ell}{\partial w^{(1)}}$

$$\frac{\partial \ell}{\partial w^{(1)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial h}\frac{\partial h}{\partial w^{(1)}} = G_o\left(a^{(2)\prime}(h^\top w^{(2)} + b)^\top w^{(2)}\right)\frac{\partial h}{\partial w^{(1)}}$$

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

- **Gradient wrt** $w^{(2)}$: $\frac{\partial \ell}{\partial w^{(2)}}$

$$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial w^{(2)}} = \underbrace{\frac{\partial \ell}{\partial o}}_{G_o} h^\top \underbrace{a^{(2)\prime}(h^\top w^{(2)} + b)}_{G_2} = G_o(h^\top G_2)$$

- **Gradient wrt** $w^{(1)}$: $\frac{\partial \ell}{\partial w^{(1)}}$

$$\frac{\partial \ell}{\partial w^{(1)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial h}\frac{\partial h}{\partial w^{(1)}} = G_o\left(a^{(2)\prime}(h^\top w^{(2)} + b)^\top w^{(2)}\right)\frac{\partial h}{\partial w^{(1)}}$$

$$\frac{\partial \ell}{\partial w^{(1)}} = G_o(G_2^\top w^{(2)})(x^\top G_1)$$

Backprop in a One-hidden layer MLP

(Bias free) Forward pass

- 1st layer: $h = a^{(1)}\left(x^\top w^{(1)}\right)$
- 2nd layer: $o = a^{(2)}\left(h^\top w^{(2)}\right)$
- Loss: $\ell(o)$

- **Gradient wrt** $w^{(2)}$: $\frac{\partial \ell}{\partial w^{(2)}}$

$$\frac{\partial \ell}{\partial w^{(2)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial w^{(2)}} = \underbrace{\frac{\partial \ell}{\partial o}}_{G_o} h^\top \underbrace{a^{(2)\prime}(h^\top w^{(2)} + b)}_{G_2} = G_o(h^\top G_2)$$

- **Gradient wrt** $w^{(1)}$: $\frac{\partial \ell}{\partial w^{(1)}}$

$$\frac{\partial \ell}{\partial w^{(1)}} = \frac{\partial \ell}{\partial o}\frac{\partial o}{\partial h}\frac{\partial h}{\partial w^{(1)}} = G_o\left(a^{(2)\prime}(h^\top w^{(2)} + b)^\top w^{(2)}\right)\frac{\partial h}{\partial w^{(1)}}$$

$$\frac{\partial \ell}{\partial w^{(1)}} = G_o(G_2^\top w^{(2)})(x^\top G_1)$$

▷ Just one additional gradient to compute $G_1 := \frac{\partial h}{\partial w^{(1)}}$

Training a neural network

Training a neural network

Overview

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \in \Theta}$

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
2. Define your dataset $\triangleright$ *Normalization*, *Augmentation*, ...

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
2. Define your dataset $\triangleright$ *Normalization*, *Augmentation*, ...
3. Define your loss

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
2. Define your dataset $\triangleright$ *Normalization*, *Augmentation*, ...
3. Define your loss
4. Regularize your network

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \in \Theta}$
2. Define your dataset $\triangleright$ *Normalization*, *Augmentation*, ...
3. Define your loss
4. Regularize your network
5. Define your optimizer $\triangleright$ *What kind of momentum you want to use*

Overview of usual steps for training a neural network

1. Define your neural network $\triangleright$ $(f_\theta)_{\theta \Theta}$
2. Define your dataset $\triangleright$ *Normalization*, *Augmentation*, ...
3. Define your loss
4. Regularize your network
5. Define your optimizer $\triangleright$ *What kind of momentum you want to use*
6. Perform SGD until you have reached a stopping criterion (Callback)

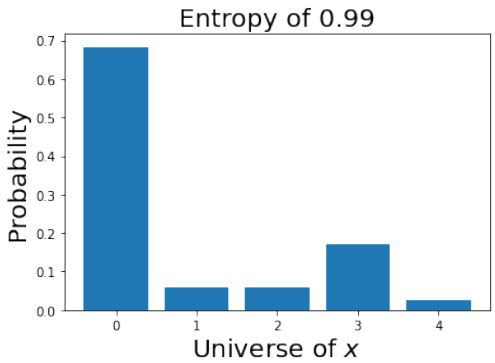
Training a neural network

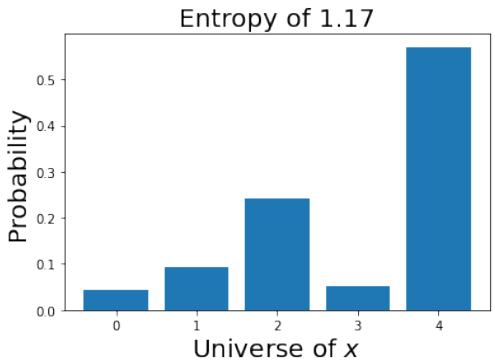
Defining a loss

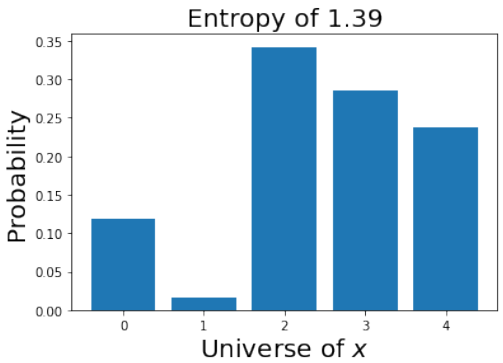
Basics of Information theory

Let p and q two distributions.

- **Entropy:** $H(p) := \mathbb{E}_{x \sim p}[-\log p(x)]$



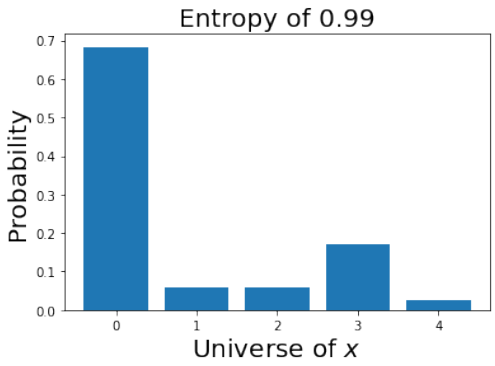


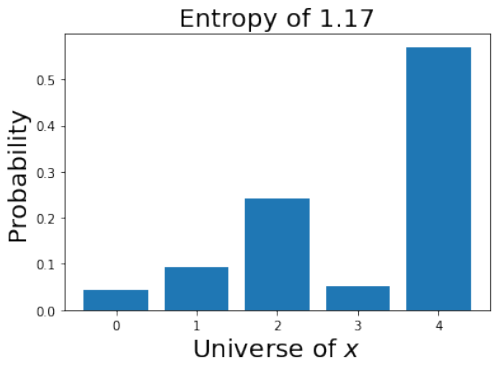


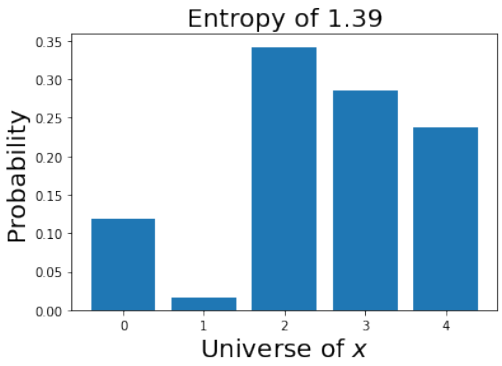
Basics of Information theory

Let p and q two distributions.

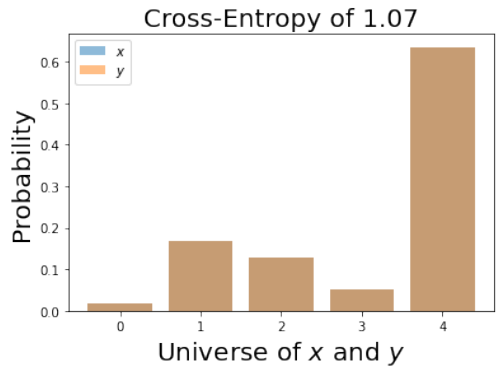
- **Entropy:** $H(p) := \mathbb{E}_{x\sim p}[-\log p(x)]$

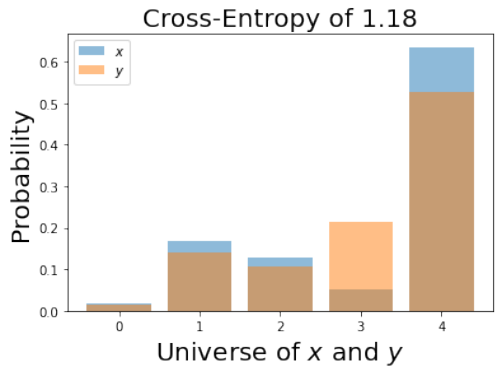


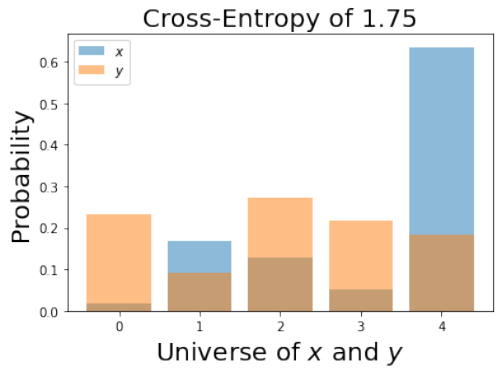




- **Cross-Entropy:** $H(p, q) := \mathbb{E}_{x\sim p}[-\log q(x)]$



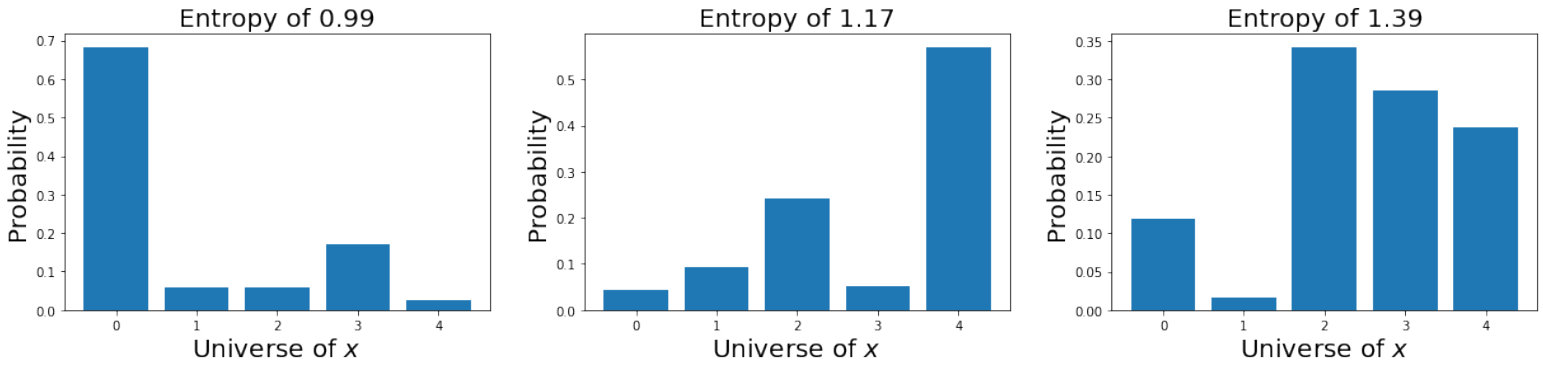




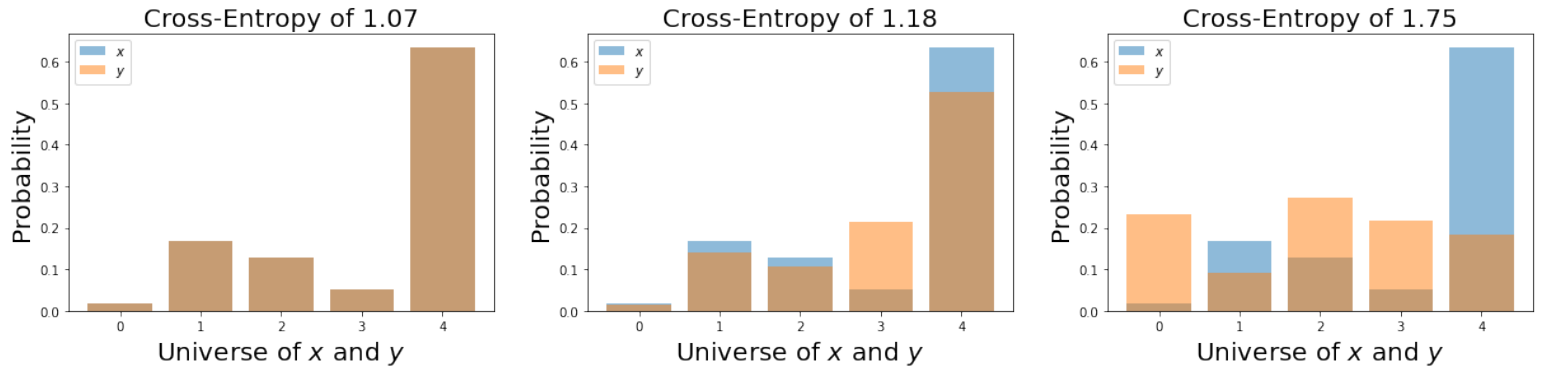
Basics of Information theory

Let p and q two distributions.

- **Entropy:** $H(p) := \mathbb{E}_{x\sim p}[-\log p(x)]$



- **Cross-Entropy:** $H(p, q) := \mathbb{E}_{x\sim p}[-\log q(x)]$



▷ Given p, the cross-entropy is minimal when $q = p$

Binary Cross-Entropy

Binary cross-entropy

- Particular case when the universe is binary (positive and negative)

$$p := p(X = 1), q := q(X = 1)$$

Binary Cross-Entropy

Binary cross-entropy

- Particular case when the universe is binary (positive and negative)

$$p := p(X = 1), q := q(X = 1)$$

- Given p and q, the (binary-)cross-entropy is:

$$H(p, q) = -p \log q - (1 - p) \log(1 - q)$$

Binary Cross-Entropy

Binary cross-entropy

- Particular case when the universe is binary (positive and negative)

$$p := p(X = 1), q := q(X = 1)$$

- Given p and q, the (binary-)cross-entropy is:

$$H(p, q) = -p \log q - (1 - p) \log(1 - q)$$

- f_θ is a neural net such that:
  - $f_\theta(x) \in [0, 1] \triangleright$ *uses* $\sigma(\cdot)$ *for outputs.*

Binary Cross-Entropy

Binary cross-entropy

- Particular case when the universe is binary (positive and negative)

$$p := p(X = 1), q := q(X = 1)$$

- Given p and q, the (binary-)cross-entropy is:

$$H(p, q) = -p \log q - (1 - p) \log(1 - q)$$

- f_θ is a neural net such that:
  - $f_\theta(x) \in [0, 1] \triangleright$ *uses* $\sigma(\cdot)$ *for outputs.*
  - $f_\theta(x)$ is design to fit $p(Y = 1|X = x)$ by minimizing:

$$\ell(y, f_\theta(x)) := -y \log f_\theta(x) - (1 - y) \log(1 - f_\theta(x))$$

(Categorical) Cross-Entropy

Categorical cross-entropy

- Let (x, y) a training data point.
- y is a categorical variable $\triangleright$ *cat*, *dog*, *cow*, *plane*, ...

(Categorical) Cross-Entropy

Categorical cross-entropy

- Let (x, y) a training data point.
- y is a categorical variable $\triangleright$ *cat*, *dog*, *cow*, *plane*, ...
- Let $\mathcal{C}$ the number of categories and $c = \underbrace{\{0, ..., 0, 1, 0..., 0\}}_{\mathcal{C} \text{ dimensions}}$ is a **one-hot vectorization** of y.

(Categorical) Cross-Entropy

Categorical cross-entropy

- Let (x, y) a training data point.
- y is a categorical variable $\triangleright$ *cat*, *dog*, *cow*, *plane*, ...
- Let $\mathcal{C}$ the number of categories and $c = \underbrace{\{0, ..., 0, 1, 0..., 0\}}_{\mathcal{C} \text{ dimensions}}$ is a **one-hot vectorization** of y.
- $f_\theta(x) \rightarrow [0, 1]^{\mathcal{C}}$ maps x to $\mathcal{C}-$dimensional space.

(Categorical) Cross-Entropy

Categorical cross-entropy

- Let (x, y) a training data point.
- y is a categorical variable $\triangleright$ *cat*, *dog*, *cow*, *plane*, ...
- Let $\mathcal{C}$ the number of categories and $c = \underbrace{\{0, ..., 0, 1, 0..., 0\}}_{\mathcal{C} \text{ dimensions}}$ is a **one-hot vectorization** of y.
- $f_\theta(x) \to [0, 1]^{\mathcal{C}}$ maps x to $\mathcal{C}-$dimensional space.
- $f_\theta(x)_c$ is design to fit $p(c|X = x)$ i.e. x belongs to the $c-$th class by minimizing:

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

- Maps an arbitrary dimensional vector $z \in \mathbb{R}^{\mathcal{C}}$ into a probability distribution over the dimensions,

Softmax-layer

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

- Maps an arbitrary dimensional vector $z \in \mathbb{R}^{\mathcal{C}}$ into a probability distribution over the dimensions,
- with higher probabilities to higher values of z,

Softmax-layer

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

- Maps an arbitrary dimensional vector $z \in \mathbb{R}^{\mathcal{C}}$ into a probability distribution over the dimensions,
- with higher probabilities to higher values of z,
- ultimately, the higher coordinate has a probability close to 1 and others are pushed to 0,

Softmax-layer

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

- Maps an arbitrary dimensional vector $z \in \mathbb{R}^{\mathcal{C}}$ into a probability distribution over the dimensions,
- with higher probabilities to higher values of z,
- ultimately, the higher coordinate has a probability close to 1 and others are pushed to 0,
- it is differentiable approximation of the $\arg\max$:

Softmax-layer

$$\ell(y, f_\theta(x)) := -\log f_\theta(x)_{c(y)}$$

Softmax-layer

- Maps an arbitrary dimensional vector $z \in \mathbb{R}^{\mathcal{C}}$ into a probability distribution over the dimensions,
- with higher probabilities to higher values of z,
- ultimately, the higher coordinate has a probability close to 1 and others are pushed to 0,
- it is differentiable approximation of the arg max:

$$\mathrm{Softmax}(z) := \frac{1}{\sum_c^{\mathcal{C}} e^{z_c}} (e^{z_1}, ..., e^{z_c})$$

Training a neural network

Regularizing neural networks

L^2 = Ridge= Weight decay = Tikhonov

The L^2 penalty is the most common regularization of models:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda||\theta||^2$$

Linear Regression

- $X \in \mathbb{R}^{n \times p}$: `(samples, features)`, $Y \in \mathbb{R}^{n \times 1}$: `(samples, value)`,
- **No-regularization:** $||Y - X^\top \theta||^2$

L^2 = Ridge= Weight decay = Tikhonov

The L^2 penalty is the most common regularization of models:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda||\theta||^2$$

Linear Regression

- $X \in \mathbb{R}^{n \times p}$: `(samples, features)`, $Y \in \mathbb{R}^{n \times 1}$: `(samples, value)`,
- **No-regularization:** $||Y - X^\top \theta||^2$

$$\hat{\theta} = (X^\top X)^{-1}(X^\top Y)$$

L^2 = Ridge= Weight decay = Tikhonov

The L^2 penalty is the most common regularization of models:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda||\theta||^2$$

Linear Regression

- $X \in \mathbb{R}^{n \times p}$: `(samples, features)`, $Y \in \mathbb{R}^{n \times 1}$: `(samples, value)`,
- **No-regularization:** $||Y - X^\top \theta||^2$

$$\hat{\theta} = (X^\top X)^{-1}(X^\top Y)$$

- **L^2-regularization:** $||Y - X^\top \theta||^2 + \lambda||\theta||^2$

L^2 = Ridge= Weight decay = Tikhonov

The L^2 penalty is the most common regularization of models:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda||\theta||^2$$

Linear Regression

- $X \in \mathbb{R}^{n \times p}$: `(samples, features)`, $Y \in \mathbb{R}^{n \times 1}$: `(samples, value)`,
- **No-regularization:** $||Y - X^\top \theta||^2$

$$\hat{\theta} = (X^\top X)^{-1}(X^\top Y)$$

- **L^2-regularization:** $||Y - X^\top \theta||^2 + \lambda||\theta||^2$

$$\hat{\theta} = (X^\top X + \lambda I)^{-1}(X^\top Y)$$

Training a neural network

Dropout regularization

Dropout: Ensembling of neural networks

Dropout consists in randomly deleting some units during training.

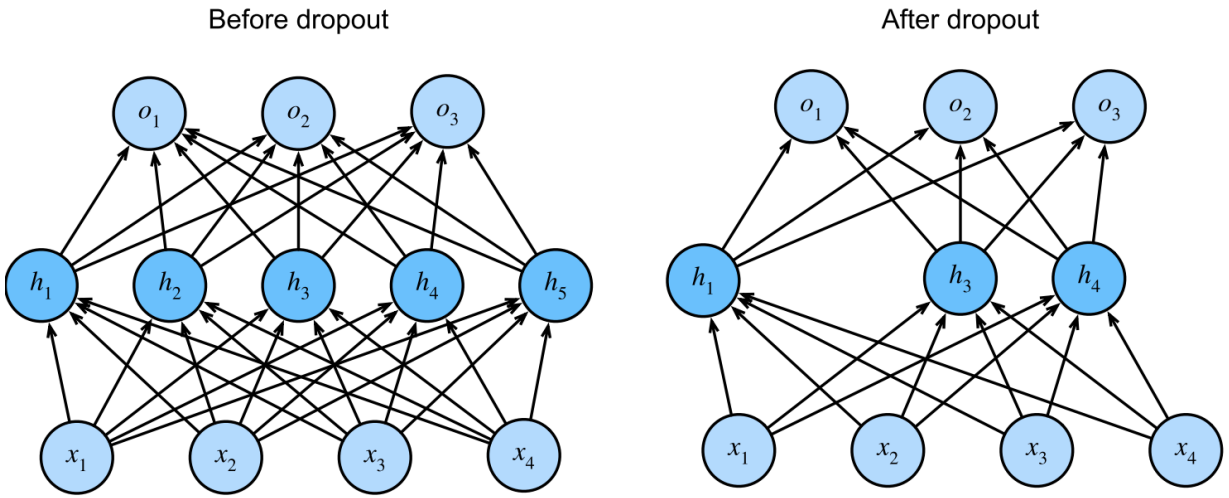


Figure 4: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

$$\text{Dropout}_p(h)_i = \begin{cases} 0 & \text{with probability } p \\ \frac{h_i}{1-p} & \text{with probability } 1-p \end{cases}$$

Dropout: Ensembling of neural networks

Dropout consists in randomly deleting some units during training.

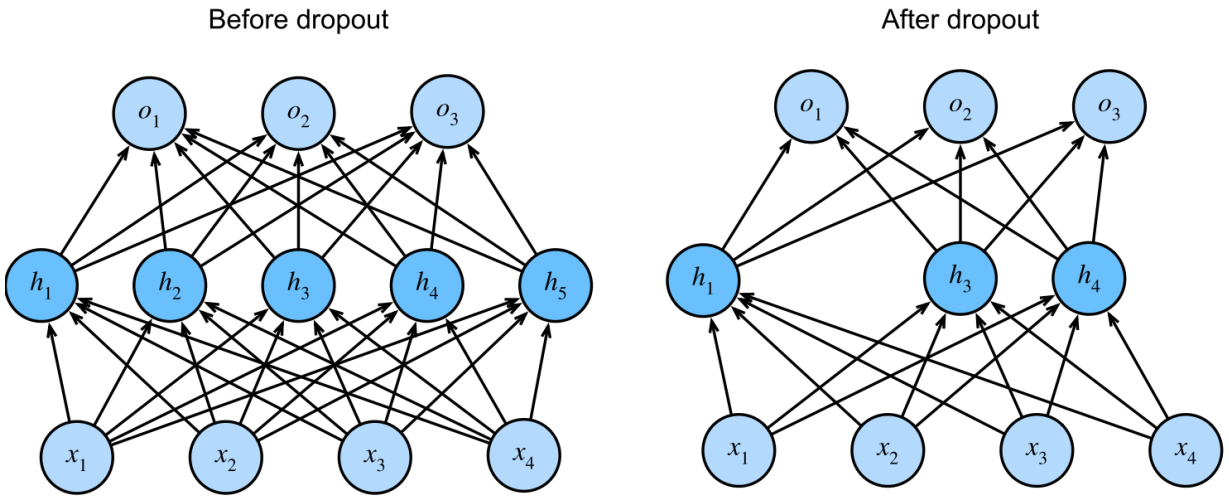


Figure 4: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

$$\text{Dropout}_p(h)_i = \begin{cases} 0 & \text{with probability } p \\ \frac{h_i}{1-p} & \text{with probability } 1-p \end{cases}$$

- Ensures an unbiased layer *i.e.*, $h = \mathbb{E}_p[\text{Dropout}_p(h)]$

Dropout: Ensembling of neural networks

Dropout consists in randomly deleting some units during training.

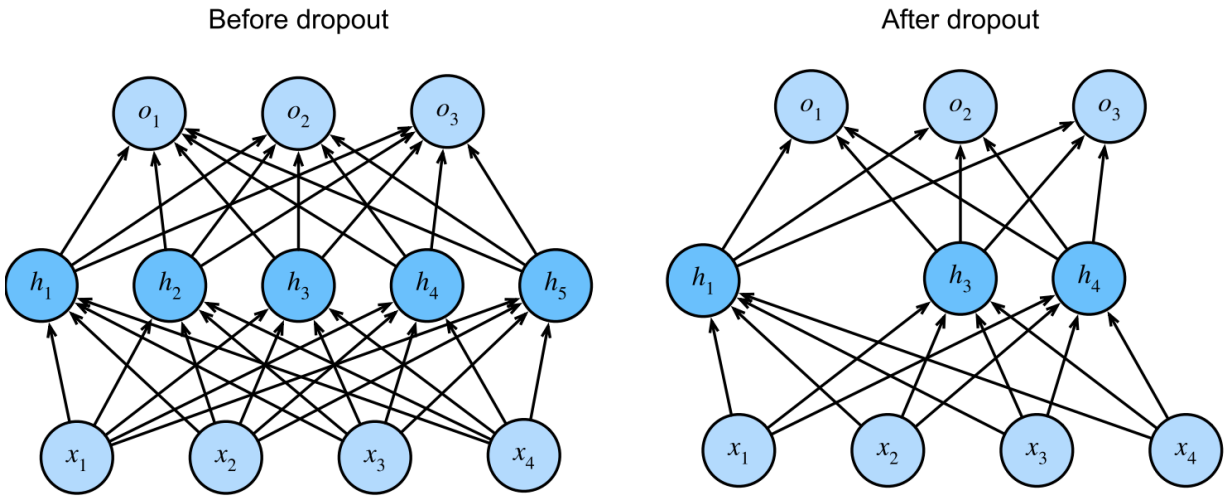


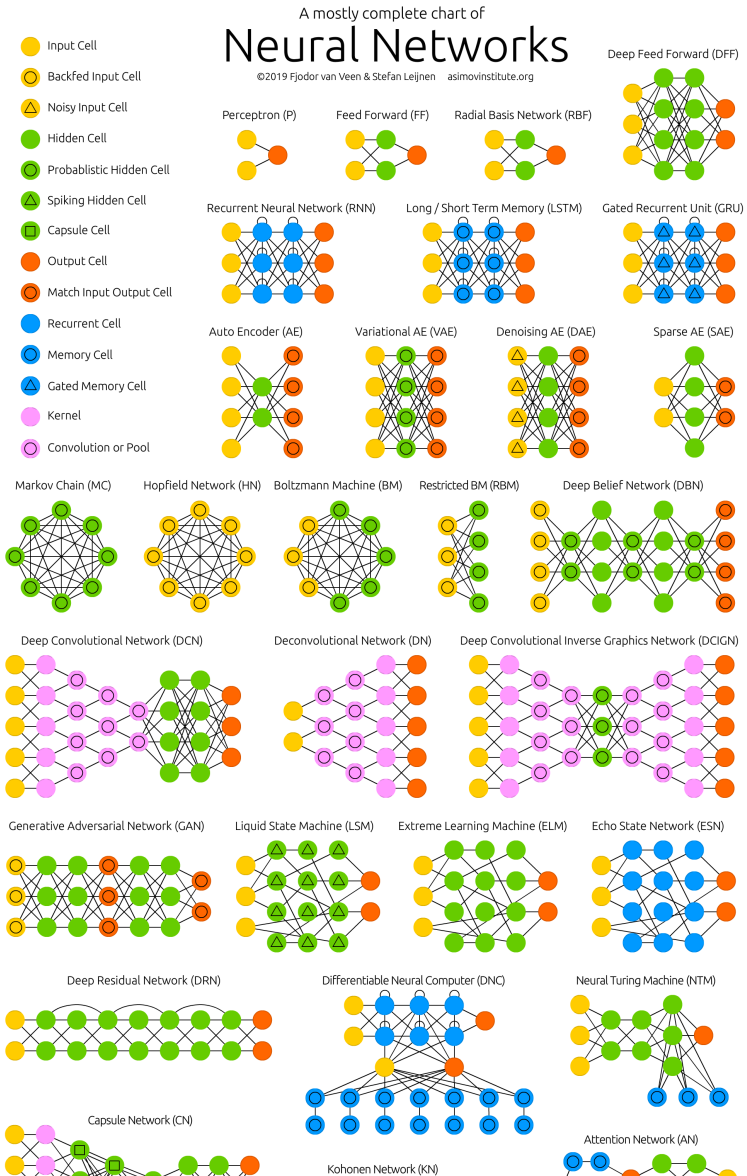
Figure 4: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

$$\text{Dropout}_p(h)_i = \begin{cases} 0 & \text{with probability } p \\ \frac{h_i}{1-p} & \text{with probability } 1-p \end{cases}$$

- Ensures an unbiased layer *i.e.*, $h = \mathbb{E}_p[\text{Dropout}_p(h)]$
- **At test-time,** $p = 0$**.**

Deep neural model zoo

https://www.asimovinstitute.org/neural-network-zoo/



Training a neural network

The Needs

Deep Learning : Needed ressources

Huge annotated data

[1]Unbiased Look at Dataset Bias :
http://people.csail.mit.edu/torralba/research/bias/

Deep Learning : Needed ressources

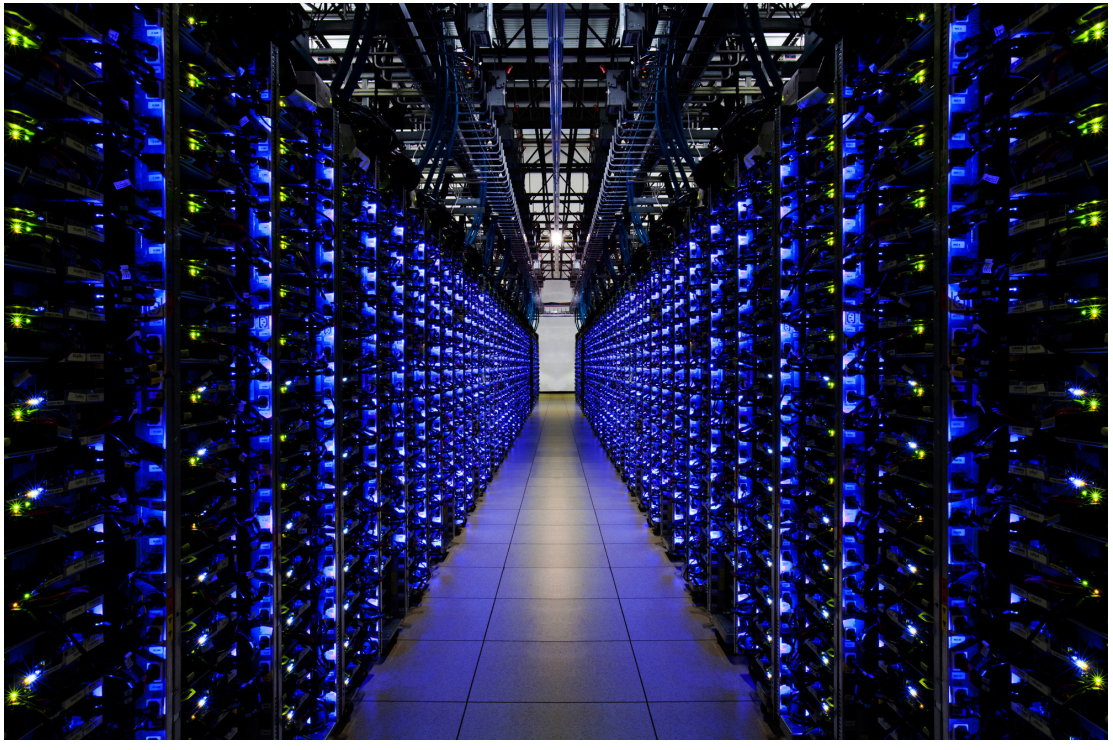
Huge annotated data

But annotation not prevent from bias in data [1],

[1] Unbiased Look at Dataset Bias :
http://people.csail.mit.edu/torralba/research/bias/

Deep Learning : Needed ressources

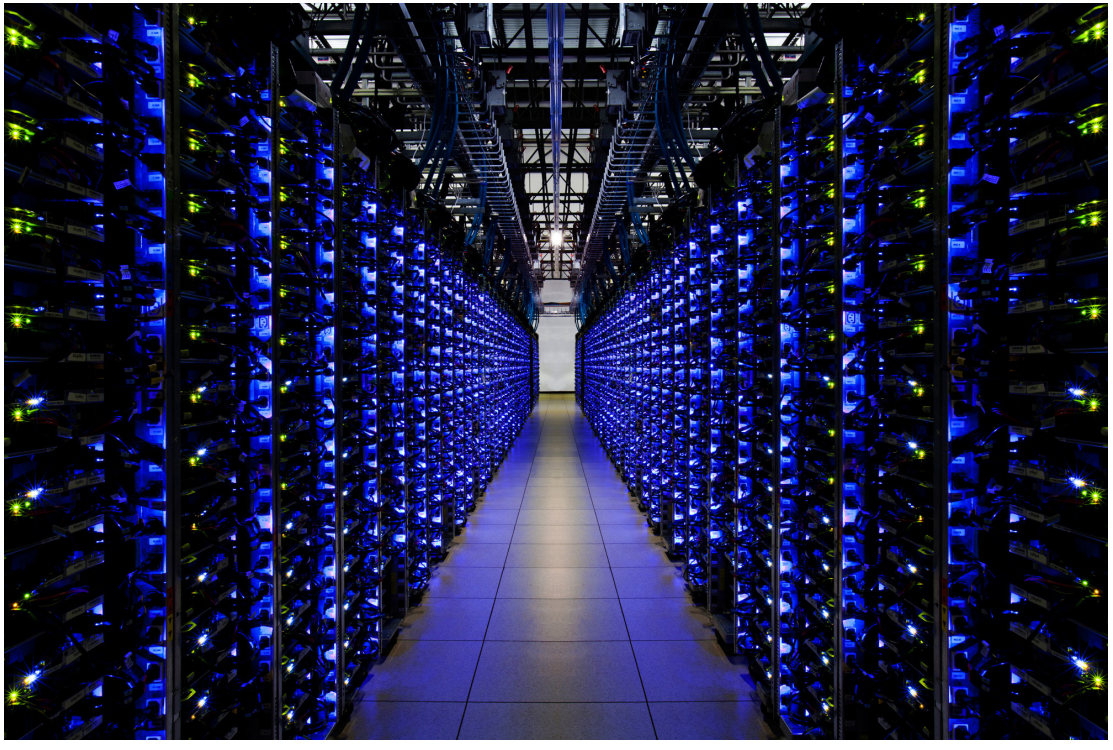
Computing and storage ressources



[2] https://www.technologyreview.com/f/614056/ai-research-has-an-environment-climate-toll/?utm_campaign=site_visitor.unpaid.engagement&utm_source=twitter&utm_medium=tr_social

Deep Learning : Needed ressources

Computing and storage ressources

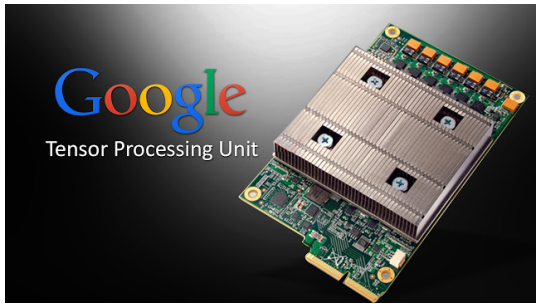


To a green AI [2].

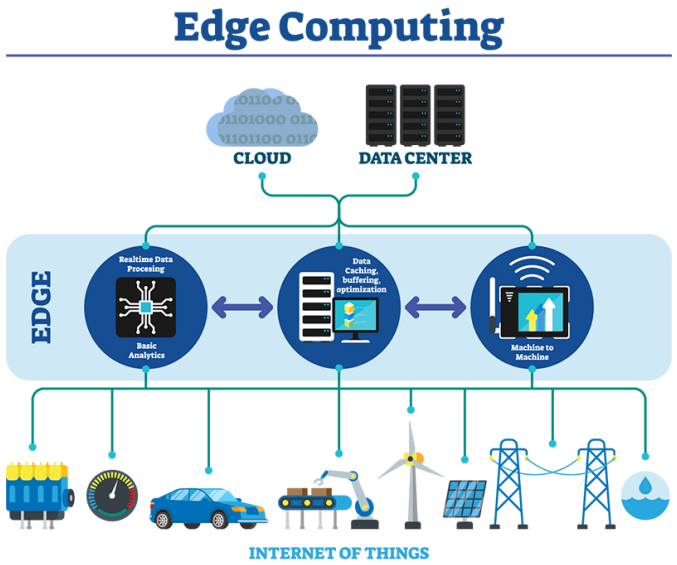
[2] https://www.technologyreview.com/f/614056/ai-research-has-an-environment-climate-toll/?utm_campaign=site_visitor.unpaid.engagement&utm_source=twitter&utm_medium=tr_social

Deep Learning : Needed ressources

Computer science and IT.



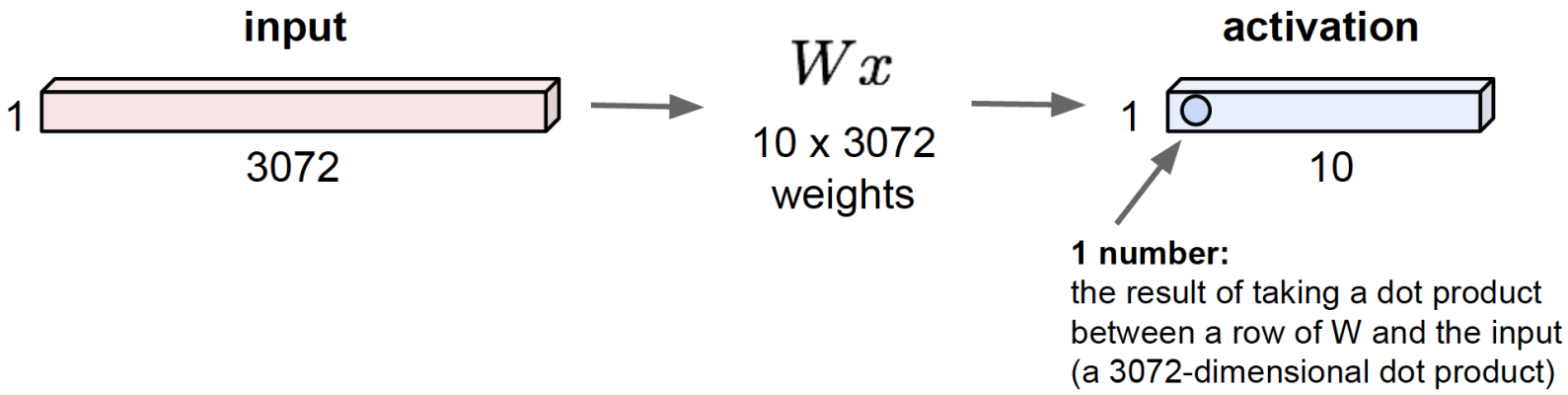
The Big Data technology stack is evolving rapidly



Convolutional neural networks

Fully Connected Layers

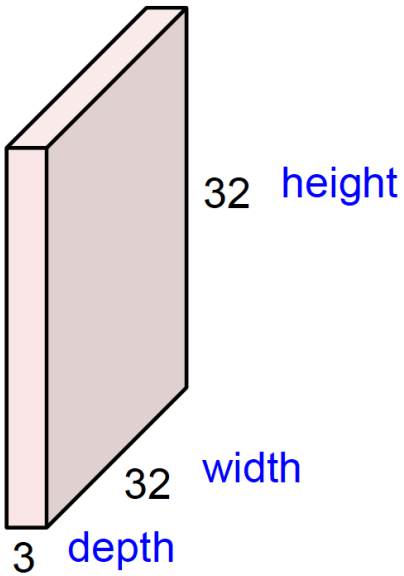
32x32x3 image -> stretch to 3072 x 1



Source : Stanford CS 231n course

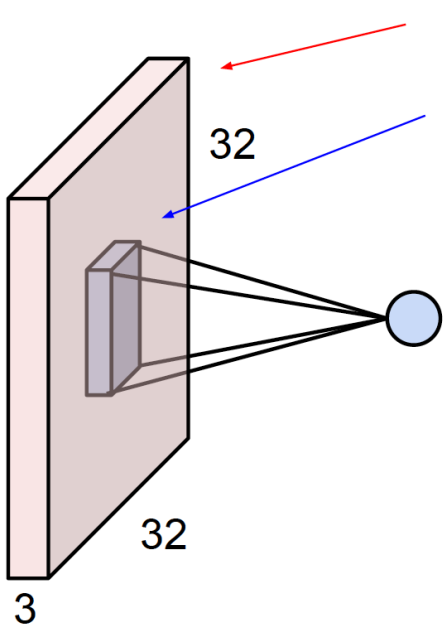
Convolutional Layer

32x32x3 image -> preserve spatial structure



Source : Stanford CS 231n course

Convolutional Layer

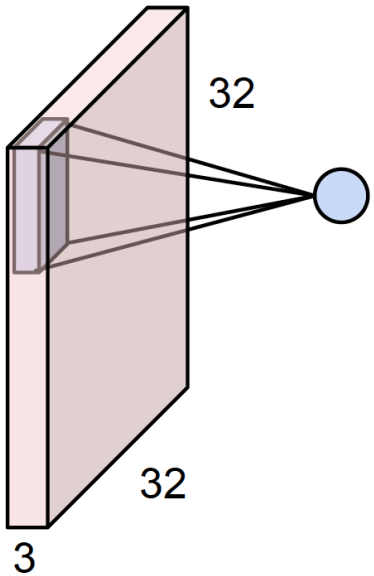


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

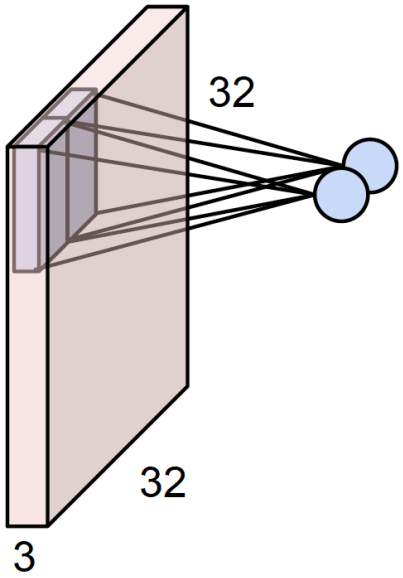
Source : Stanford CS 231n course

Convolutional Layer



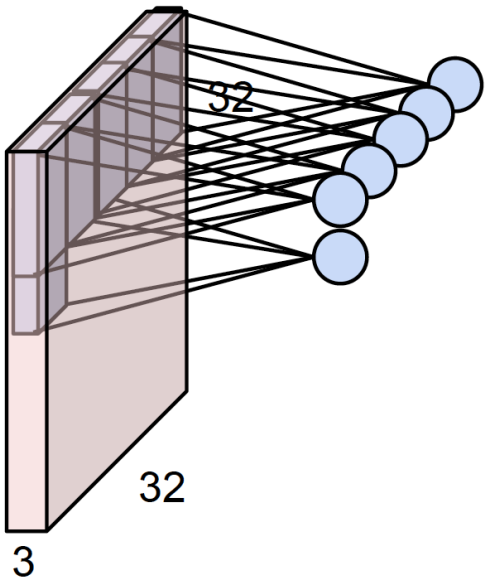
Source : Stanford CS 231n course

Convolutional Layer



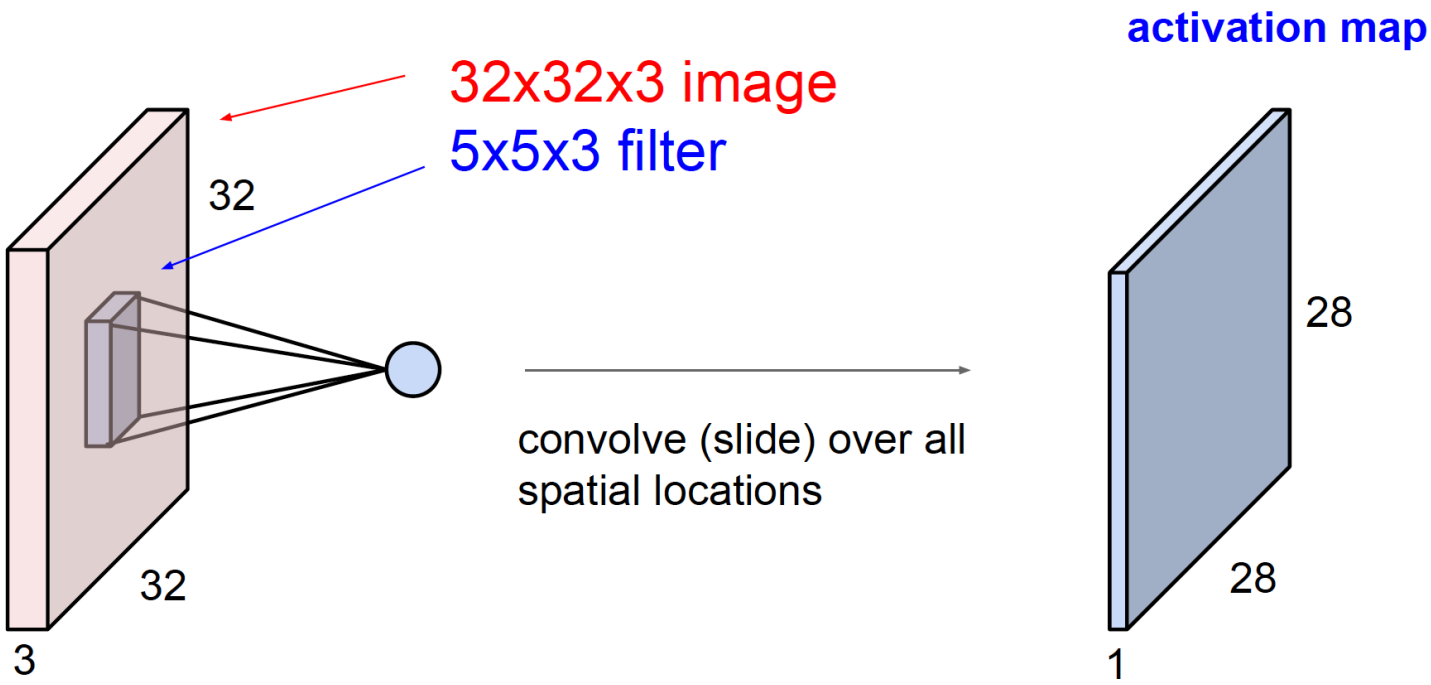
Source : Stanford CS 231n course

Convolutional Layer



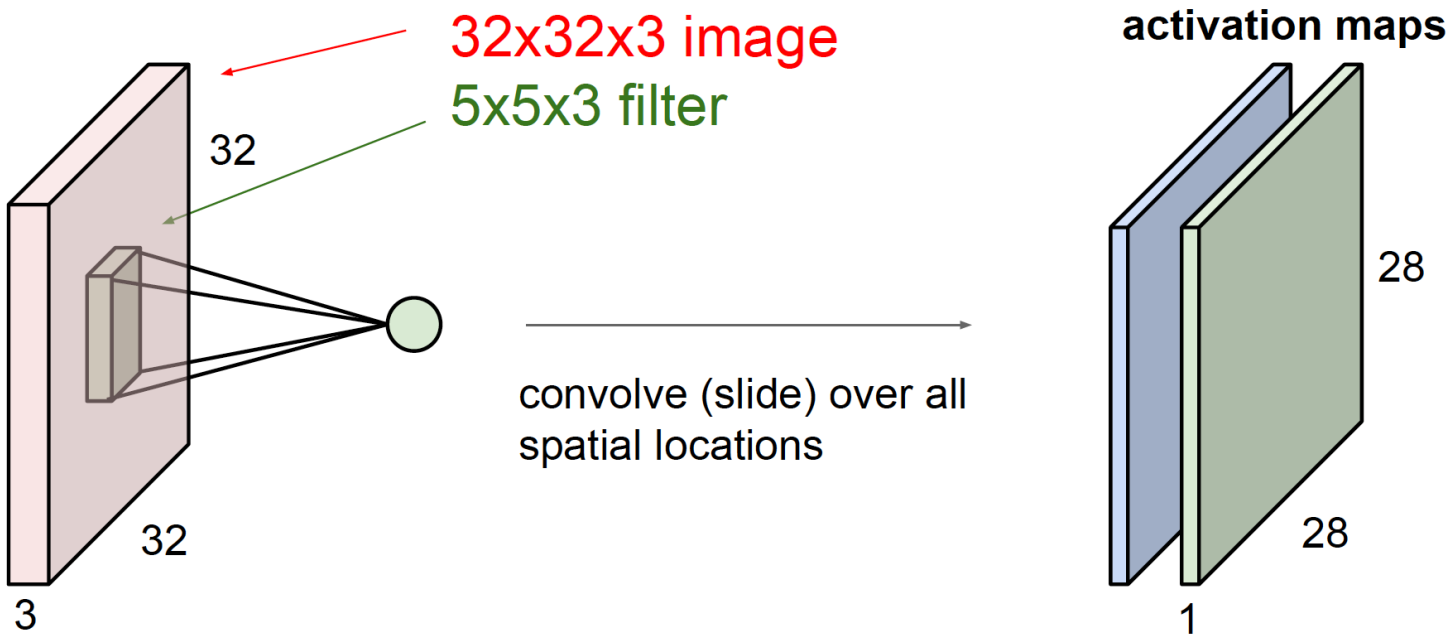
Source : Stanford CS 231n course

Convolutional Layer



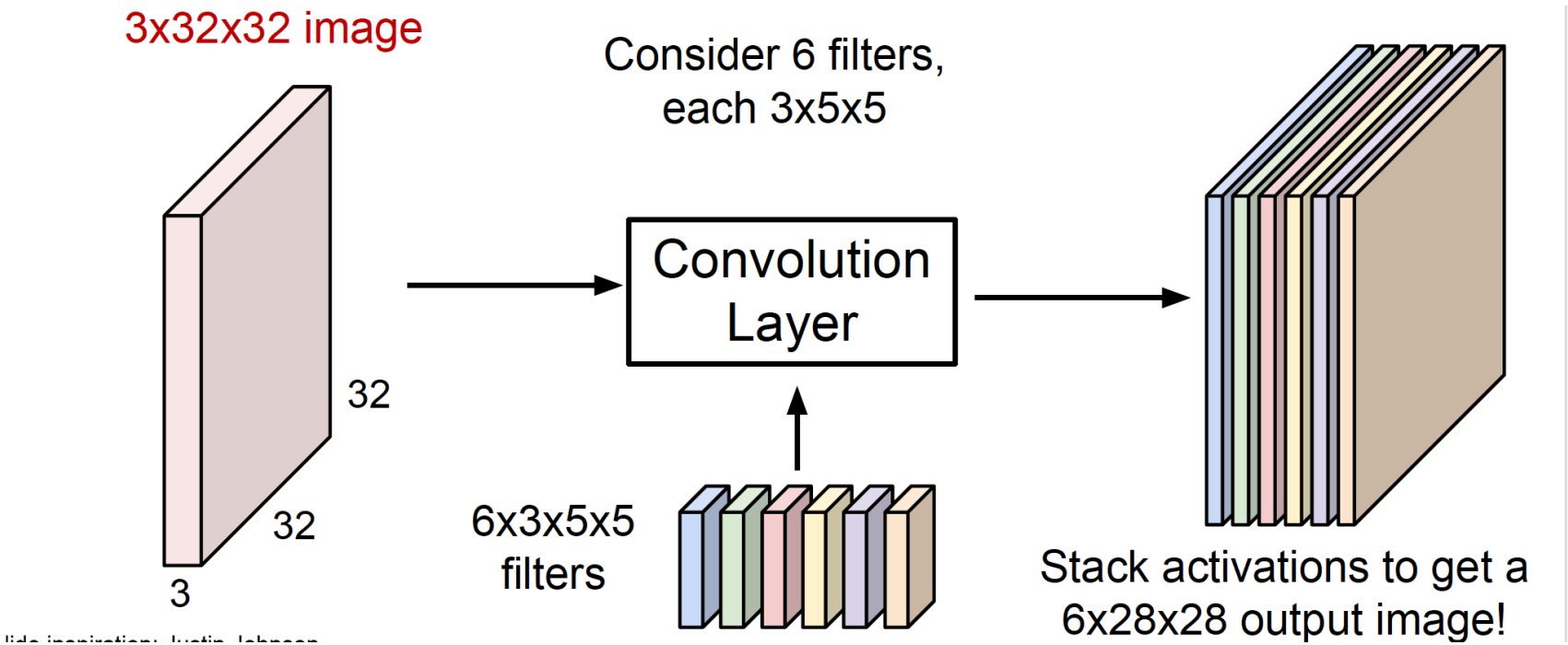
Source : Stanford CS 231n course

Convolutional Layer



Source : Stanford CS 231n course

Convolutional Layer



Source : Stanford CS 231n course

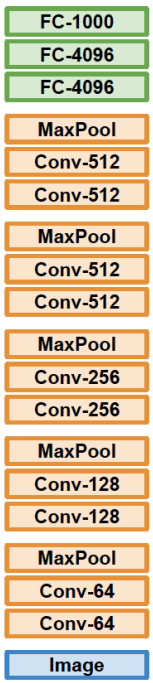
CNNs

To understand : `https://poloclub.github.io/cnn-explainer/`

Transfer Learning

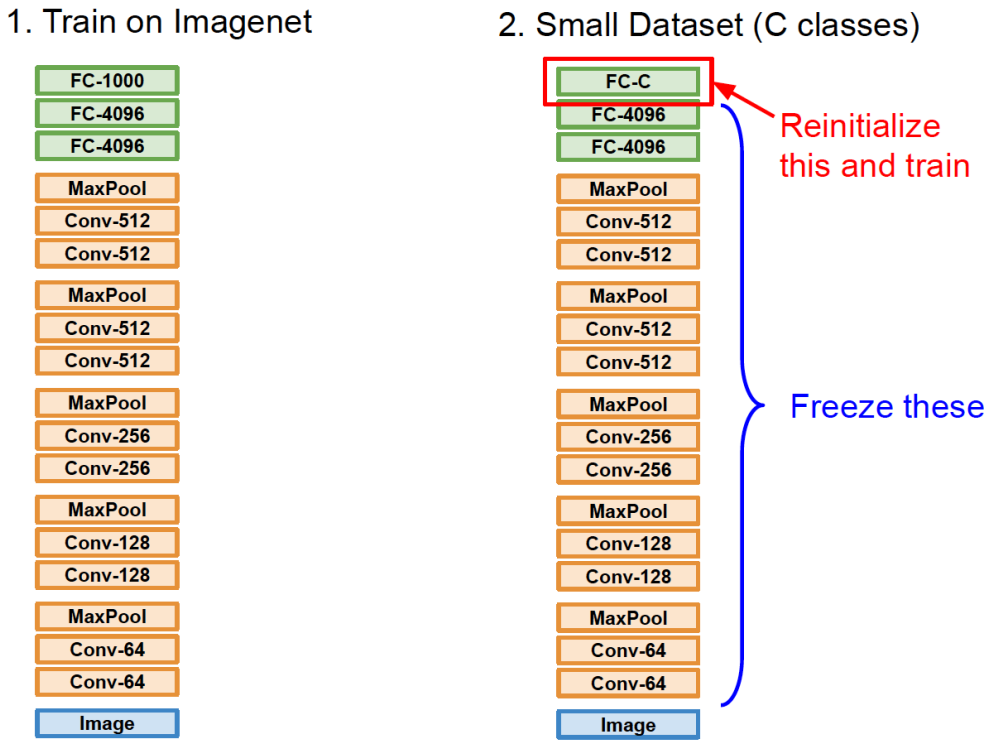
Transfer Learning

1. Train on Imagenet



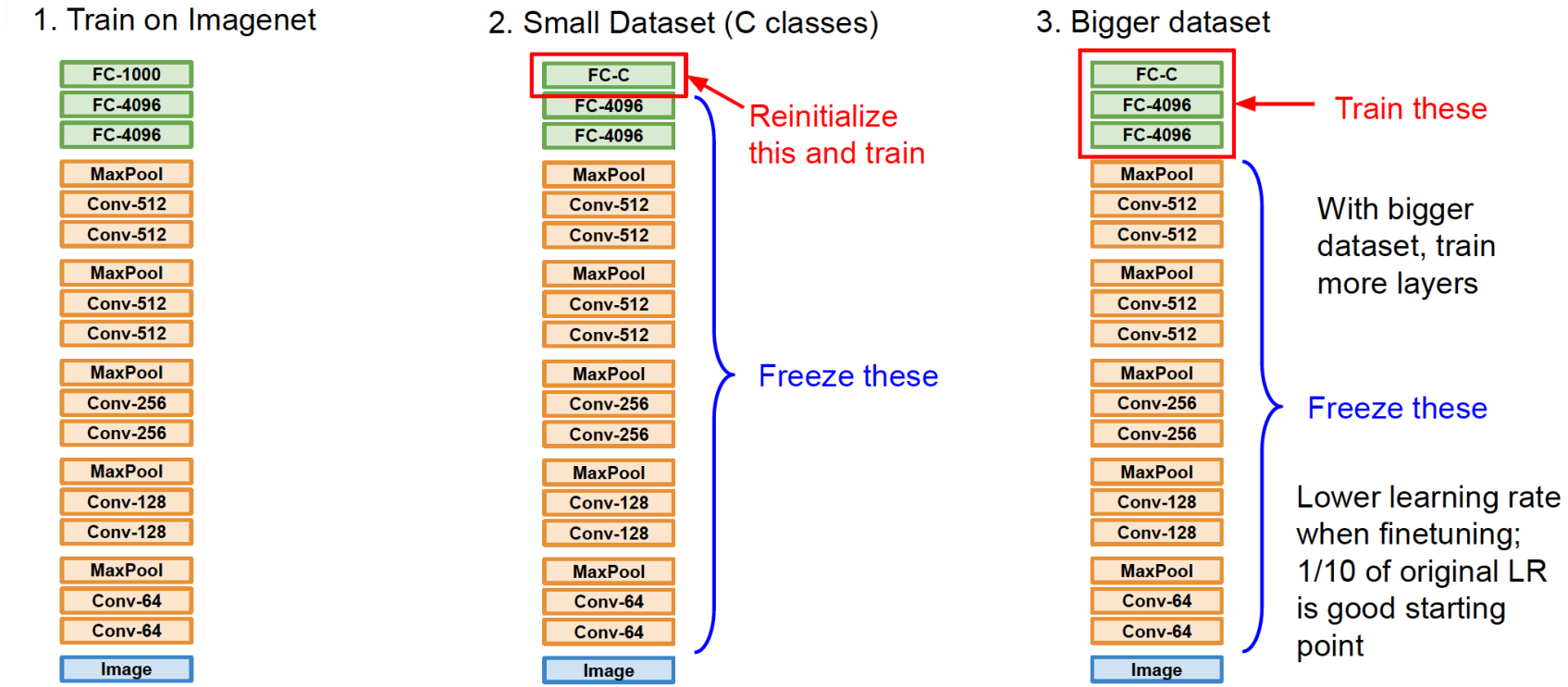
Source : Stanford CS 231n course

Transfer Learning



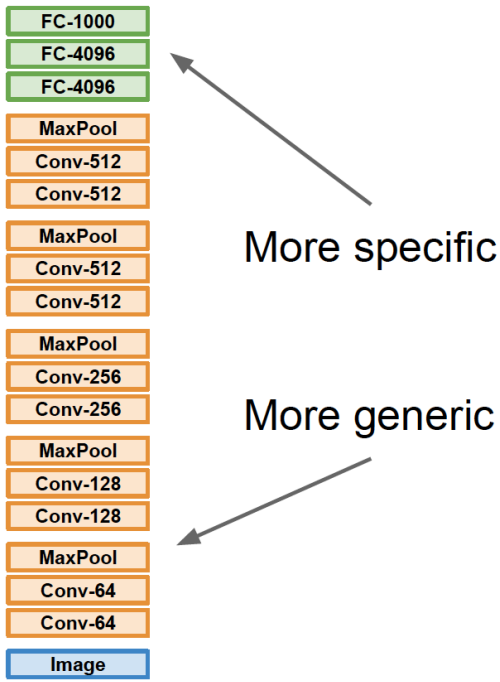
Source : Stanford CS 231n course

Transfer Learning



Source : Stanford CS 231n course

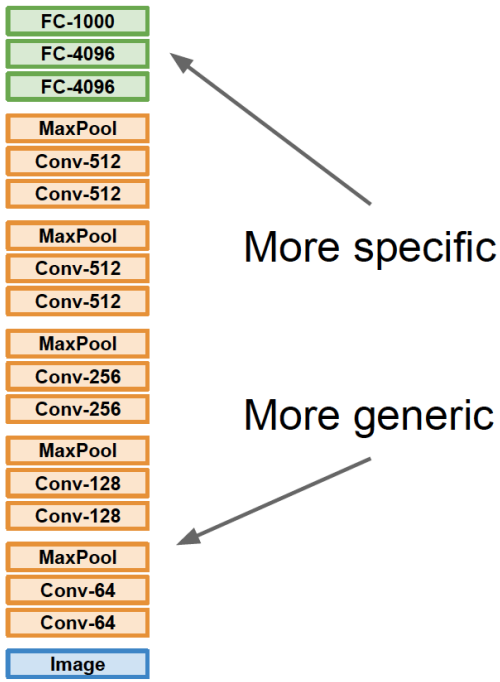
Transfer Learning



	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?

Source : Stanford CS 231n course

Transfer Learning



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Source : Stanford CS 231n course

Conclusion

Conclusion

- Deep learning is a key element of the recent success of AI.
- Performance of deep learning models is highly correlated to the availability of huge high-quality annotated datasets.
  - But, this availability assumption is not realistic: **deep learning in low data regime**
- Deep learning and more generally AI is face to methodological issues to tackle :
  - Robustness (to shifts)
  - Safety
  - Privacy
  - Ethics
  - **Explainability**

Foundations models

Foundation models

A foundation model is a model trained at broad scale that can adapted to a wide range of downstream tasks

Examples: BERT GPT-3 CLIP

The technology is not new:

Self-supervised learning with neural networks

What is new?

Scale and the ability to perform tasks beyond training

Emergence and homogenisation

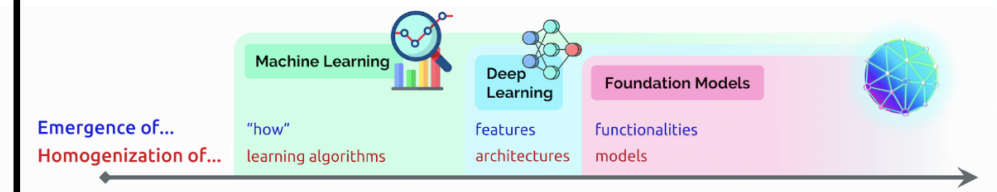
Two key ideas underpin the significance of foundation models:

Emergence

- system behaviour is implicitly induced rather than explicitly constructed
- cause of scientific excitement and anxiety of unanticipated consequences

Homogenisation

- consolidation of methodology for building machine learning system across many applications
- provides strong leverage for many tasks, but also creates single points of failure



Foundations models

Machine learning

Modern systems targeting AI tend to use machine learning

The ideas behind machine learning (ML) have been discussed for a long time (Turing, 1948; Samuel, 1959)

Machine learning really began to rise in popularity in 1990s

It represented a shift in how AI systems were built

Machine learning does not specify how to solve a task

Instead, the "how" emerges from the learning process

Machine learning also represents a step towards homogenisation:

Many applications can be powered by the same learning algorithm

Complex tasks in NLP/computer vision still required domain experts

Feature engineering (e.g. SIFT) needed to achieve good performance

Deep Learning

Slightly more than a decade ago, there was a resurgence of Deep Learning

Beyond the original algorithms, key factors included:

- GPUs
- Increased data availability

These produced breakthrough results like AlexNet

Deep learning also represented a shift towards homogenisation:

Instead of hand-crafting features, the same architecture could be used widely

References

A. M. Turing, "Intelligent Machinery" (1948)
A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of R&D (1959)
D. Lowe, "Object recognition from local scale-invariant features" ICCV (1999)
Y. LeCun et al., "Deep learning", Nature (2015)
J. Schmidhuber, "Deep learning in neural networks: An overview", Neural networks (2015)
A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks", NeurIPS (2012)

Foundations models

Foundation models: origins

Foundation models are enabled by transfer learning (Bozinovski, 1976)

Take knowledge from one task (e.g. object recognition in images) apply it to another task (e.g. activity recognition in videos)

In deep learning, the dominant paradigm is pretraining:

- train a model on a surrogate task
- adapt by fine-tuning on the task of interest

Foundation models are powerful transfer learners due to their scale

Ingredients of scaling:

- computer hardware improvements (e.g. GPUs and memory)
- Transformer architectures (leverage parallelism, expressivity)
- Availability of training data

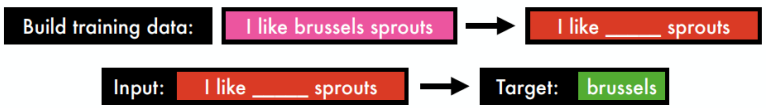
Foundation models: self-supervision

Transfer learning saw initial success from supervised pretraining (e.g. ImageNet)

However, annotation cost limited its benefits

In self-supervised learning, the pretraining task is derived from the data

Example: BERT is trained to predict a masked word from its context



Self-supervised tasks are more scalable than supervised tasks (no labelling cost)

Also potentially a richer learning signal: the model predicts part of inputs (which can be very diverse), rather than a label space (which is typically more limited)

References

S. Bozinovski et al., "The influence of pattern similarity and transfer of learning upon training of a base perceptron B2" (original in Croatian, 1976)
(Transformers) A. Vaswani et al., "Attention is all you need", NeurIPS (2017)
(ImageNet) O. Russakovsky et al., "Imagenet large scale visual recognition challenge", IJCV (2015)
(BERT) J. Devlin et al., "Bert: Pre-training of deep bidirectional transformers for language understanding", NAACL-HLT (2019)