Challenging deep learning models in real-world applications

Learning with few or no data and looking for explainability
Introduction

Céline Hudelot

Ecole d'été de Peyresq 2022

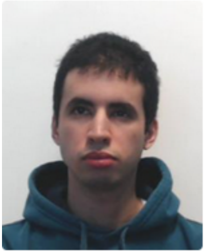
Foreword

Prof. Céline Hudelot, Computer Science
Head of the MICS Laboratory
Research on semantic data interpretation
https://scholar.google.fr/citations?user=gFlAh6MAAAAJ&hl=fr

Foreword

Other contributors of this course

Dr. Victor Bouvier
Research Scientist, Dataiku

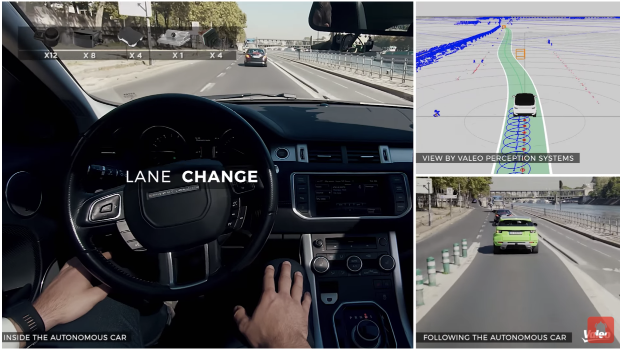


Yassine Ouali
PhD, MICS

The Deep Learning breakthrough

Some impressive applications of AI

Self-driving cars in the road of Paris



See :`https://www.youtube.com/watch?v=9mBL16JuvsM`

Some impressive applications of AI

Realistic data generation

Figure 1: Test it here : `https://thispersondoesnotexist.com/`

Some impressive applications of AI

Able to play music

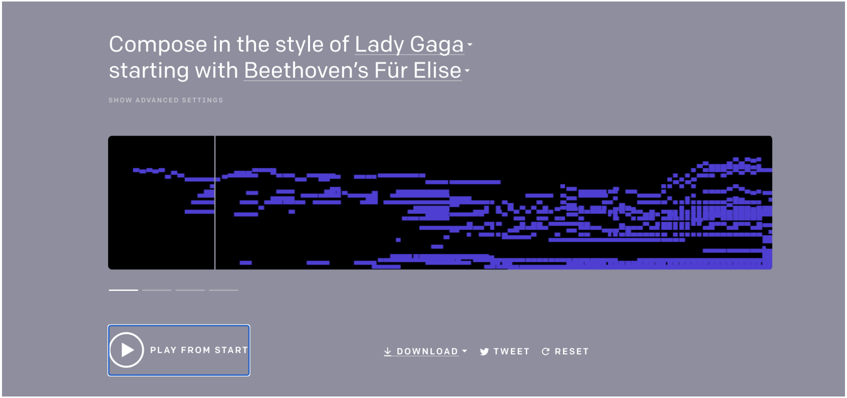


Figure 2: Test it here : `https://openai.com/blog/musenet/`

Some impressive applications of AI

Able to create art

Figure 3: The next Rembrandt: `https://www.nextrembrandt.com/`

This raises problems of intellectual property: who is the author? (`https://cacm.acm.org/magazines/2020/7/245693-ai-authorship/fulltext`).

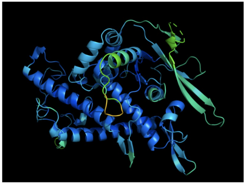
Some impressive applications of AI

Advancing knowledge: predicting the structure of proteins from their amino acid sequence

'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures

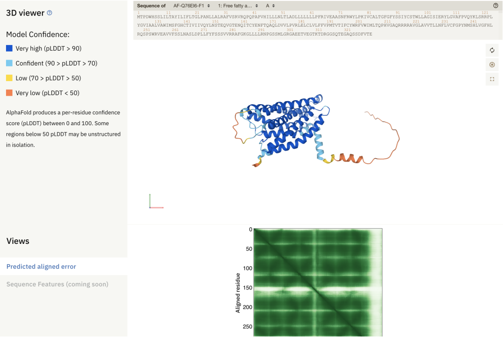
Google's deep-learning program for determining the 3D shapes of proteins stands to transform biology, say scientists.

Ewen Callaway



A protein's function is determined by its 3D shape. Credit: DeepMind

An artificial intelligence (AI) network developed by Google AI offshoot DeepMind has made a gargantuan leap in solving one of biology's grandest challenges — determining a protein's 3D shape from its amino-acid sequence.



Able to predict the structure of proteins from their amino acid sequence

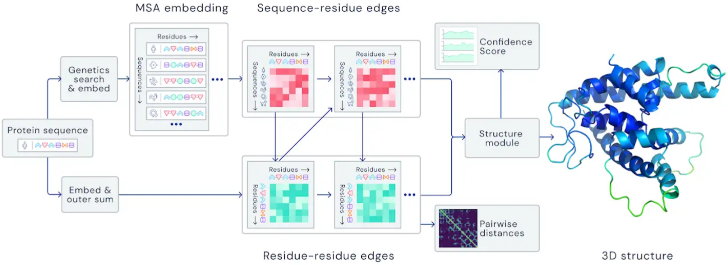
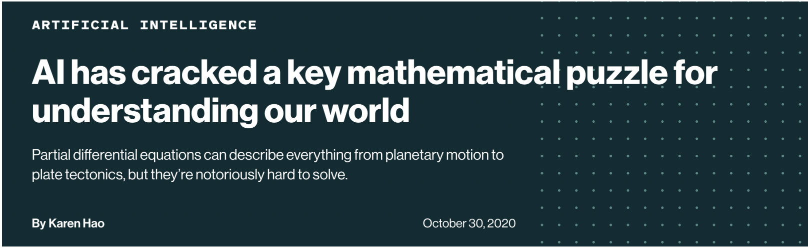


Figure 4: AlphaFold

Some impressive applications of Deep Learning

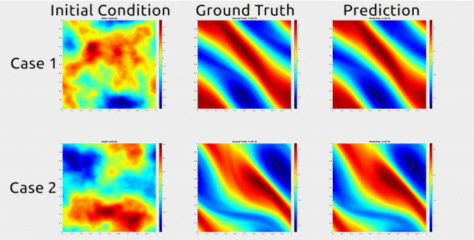
Able to solve PDE!


ARTIFICIAL INTELLIGENCE

AI has cracked a key mathematical puzzle for understanding our world

Partial differential equations can describe everything from planetary motion to plate tectonics, but they're notoriously hard to solve.

By Karen Hao October 30, 2020



The Deep Learning breakthrough

Which AI ?

Two major antagonistic approaches[1]

Two different assumptions

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.
  - Statistic or data-centric AI - Connectionist approaches - Learning from data.

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.
    - Statistic or data-centric AI - Connectionist approaches - Learning from data.
    - Exploitation of the past experience represented by annotated data, building calibrated predictive models from it.

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.
  - Statistic or data-centric AI - Connectionist approaches - Learning from data.
  - Exploitation of the past experience represented by annotated data, building calibrated predictive models from it.
- Human reasoning can be captured, even if partially incomplete: explicit representation of knowledge (using symbols rather that statistics to represent the world).

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.
  - Statistic or data-centric AI - Connectionist approaches - Learning from data.
  - Exploitation of the past experience represented by annotated data, building calibrated predictive models from it.
- Human reasoning can be captured, even if partially incomplete: explicit representation of knowledge (using symbols rather that statistics to represent the world).
  - Symbolic AI - Based on the modeling of logical reasoning, on formalisms for knowledge representation and reasoning.

[1] D. Cardon et al - La Revanche des neurones - `https://hal.archives-ouvertes.fr/hal-02005537/document`

Two major antagonistic approaches[1]

Two different assumptions

- Human reasoning and knowledge are complex: knowledge implicitly in data.
  - Statistic or data-centric AI - Connectionist approaches - Learning from data.
  - Exploitation of the past experience represented by annotated data, building calibrated predictive models from it.
- Human reasoning can be captured, even if partially incomplete: explicit representation of knowledge (using symbols rather that statistics to represent the world).
  - Symbolic AI - Based on the modeling of logical reasoning, on formalisms for knowledge representation and reasoning.

[1] D. Cardon et al - La Revanche des neurones - https://hal.archives-ouvertes.fr/hal-02005537/document

Data-driven Artificial Intelligence

Exploits the past experience represented by labeled data to build calibrated predictive models from **labeled** data.

Data-driven Artificial Intelligence

Exploits the past experience represented by labeled data to build calibrated predictive models from **labeled** data.
At the origin of the recent AI explosion, thanks to:

- increased availability of **data**, 'big data'.
- improvement of processing methods and **algorithms** (in particular deep neural networks)
- increasing of the **Computing** capacity.

Data-driven Artificial Intelligence

Exploits the past experience represented by labeled data to build calibrated predictive models from **labeled** data.
At the origin of the recent AI explosion, thanks to:

- increased availability of **data**, 'big data'.
- improvement of processing methods and **algorithms** (in particular deep neural networks)
- increasing of the **Computing** capacity.

Principle

We want to predict Y from X, as for instance:

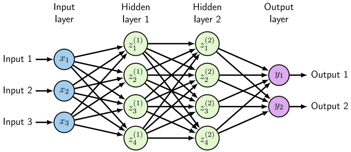
- X: radiology image, Y: presence of a tumor?
- X: sensor and monitoring data Y: rule life prediction of the system ?

Determination of the fonction ψ (model) such that $Y = \psi(X)$, and ψ is estimated from **labeled data**:
N situations in which one knows at the same time X and Y: $(X_i, Y_i)_{1 \le i \le N}$

Data-driven Artificial Intelligence

Deep neural networks

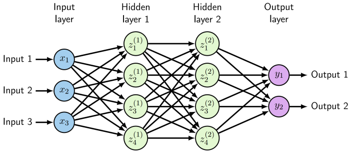
$Y = \psi(X)$, with $\psi(X) = h_M \circ g_M \circ \ldots \circ h_1 \circ g_1(X)$ where h_i some non-linear transformations and g_i some affine transformations.



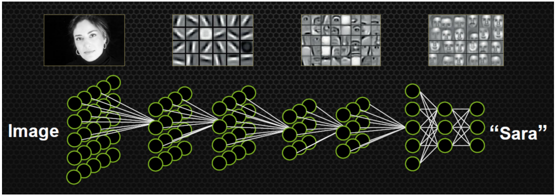
Data-driven Artificial Intelligence

Deep neural networks

$Y = \psi(X)$, with $\psi(X) = h_M \circ g_M \circ \ldots \circ h_1 \circ g_1(X)$ where h_i some non-linear transformations and g_i some affine transformations.



Representation learning



The **Deep** layers capture complex features in the image to extract the most relevant information for the prediction task [Lee et al., 2009].

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 ▷ *or a loss to minimize* ℓ,
3. Specify a model ▷ $(f_\theta)_{\theta \in \Theta}$

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

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

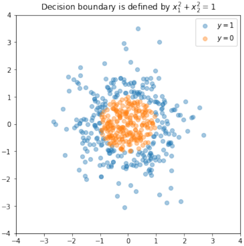
Features engineering

Features engineering

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** $x \sim \mathcal{N}(0, 1)$

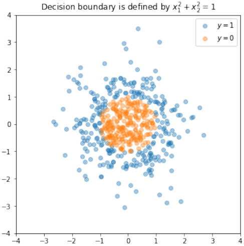


Features engineering

Features engineering
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/1NugMhZ9VEE3Mwt5OavgFV1hPWXl7N5Wo?usp=sharing`)

Representation

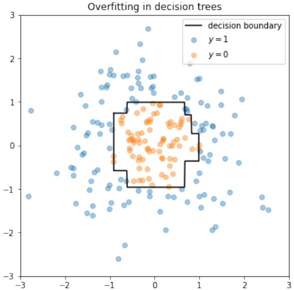
- Features engineering (representation) is mandatory in two cases:

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.*

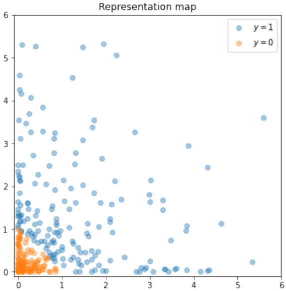
Representation

- 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.*



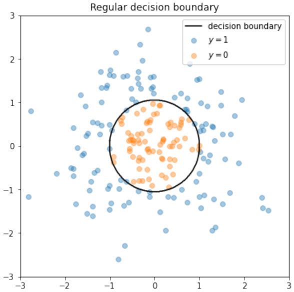
Representation

- 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 ▷ $\varphi(x) := (x_1^2, x_2^2)$



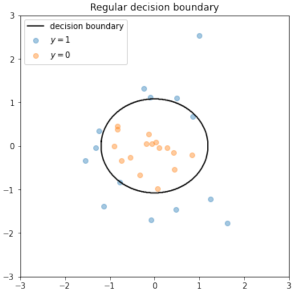
Representation

- 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?

Model benefits from a strong regularity...
A representation encodes our inductive bias $\triangleright$
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?

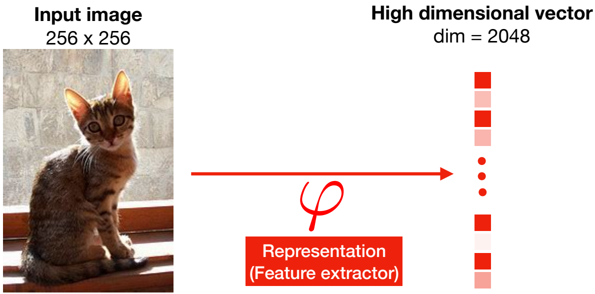
- Does the data scientist know *a priori* the shape of the solution?
- Does it exists a representation that can be reasonably hand-crafted?

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

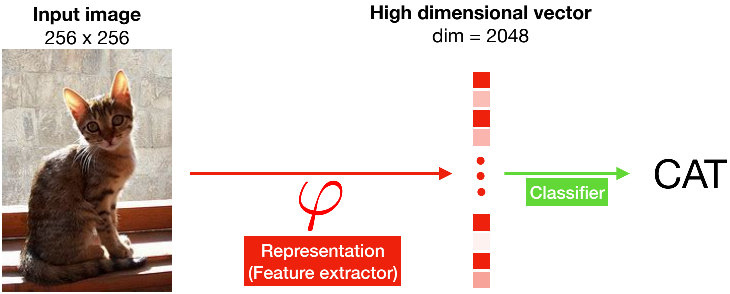
Learning representations

Input image
256 x 256

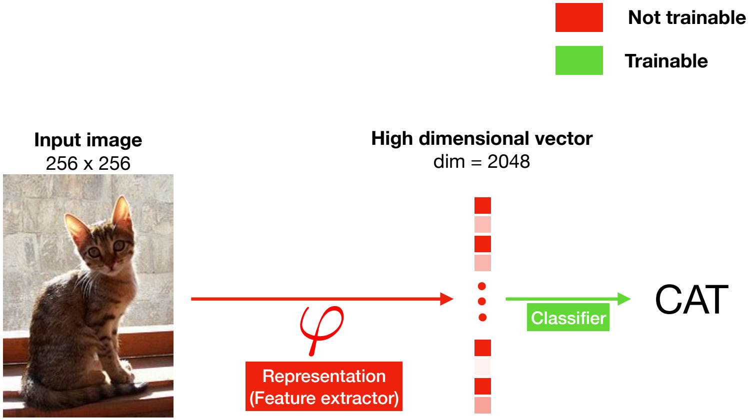
Learning representations

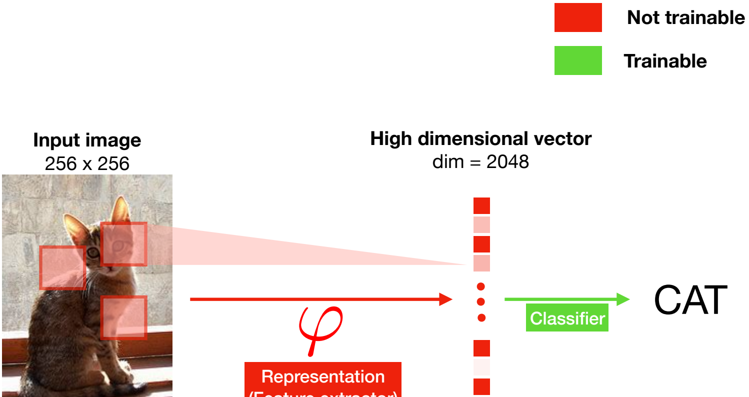


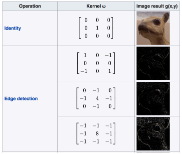
Learning representations



Learning representations







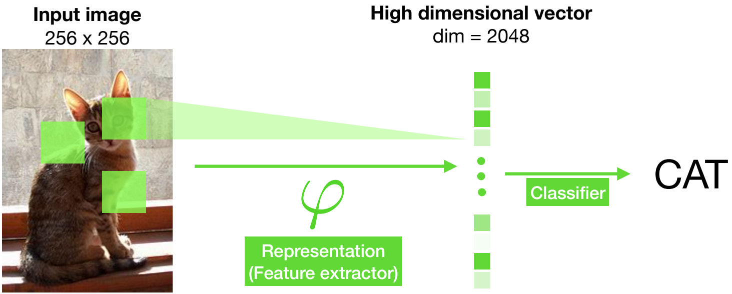
Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1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

Source wikipedia

Not trainable

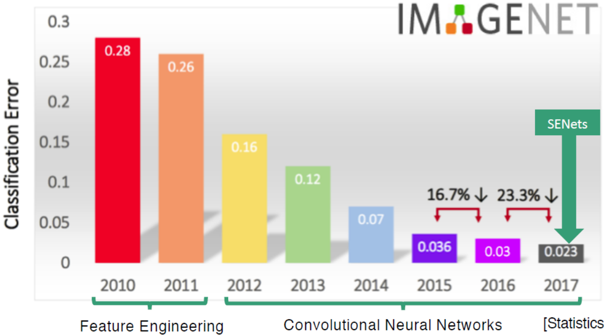
Trainable

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…)

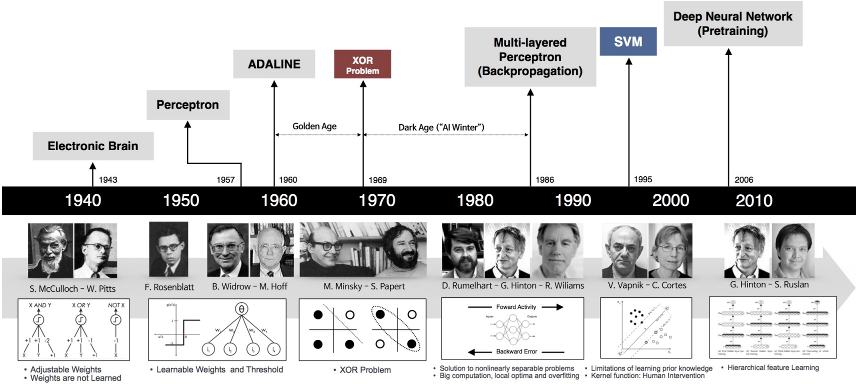
Learning representations



[Statistics provided by ILSVRC]

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

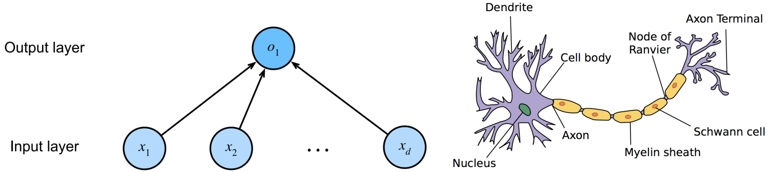


Figure 5: (Left): Illustration of a linear neural network from Rosentblatt. (Right) Biological inspiration of artifical neurons from Warren McCulloch and Walter Pitts. From `d2l.ai/d2l-en.pdf`.

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

with $\theta = (w, b)$. ▷ 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:** $\ell(y, f_\theta(x)) := -y \log(f_\theta(x)) - (1 - y) \log(1 - f_\theta(x))$

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

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

Linear Neural Network

Linear Neural Network

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

with $\theta = (w, b)$. ▷ 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:** $\ell(y, f_\theta(x)) := -y \log(f_\theta(x)) - (1-y)\log(1 - f_\theta(x))$

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

▷ 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 f as follows:

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

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

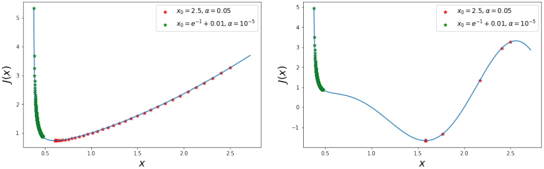


Figure 6: (Left) GD can minimize arbitrary complicated functions, here $J(x) = x^2/(1 + \log(x))$. (Right)Gradient descent with different initialization may lead to different minima, here $J(x) = x^2 \sin(\pi x)/(1 + \log(x))$

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).

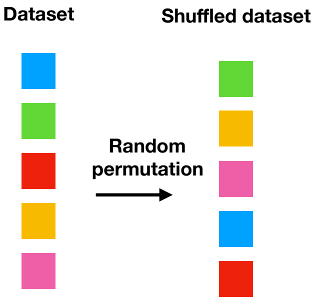
- **Input:** Θ parameters, η learning rate, b batch size, $\mathcal{D}$ dataset.
- Initialize $\theta_0 \in \Theta$.
- For a given number of iterations:
  - Sample $\mathcal{B} \sim \mathcal{D}$ such that $|\mathcal{B}| = b$.
  - Compute the batch loss $\mathcal{L}^{\mathcal{B}}(\theta) := \frac{1}{b}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

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

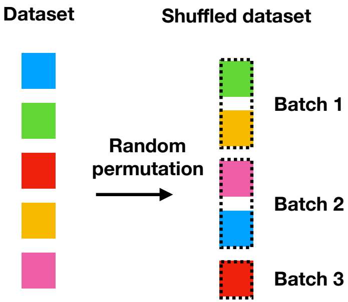
Stochastic Gradient Descent

Dataset

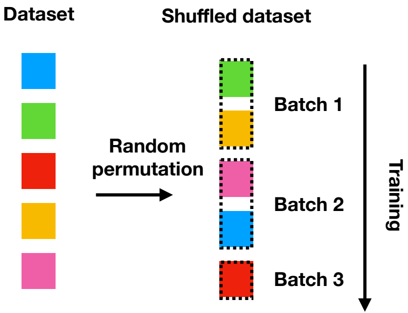
Stochastic Gradient Descent



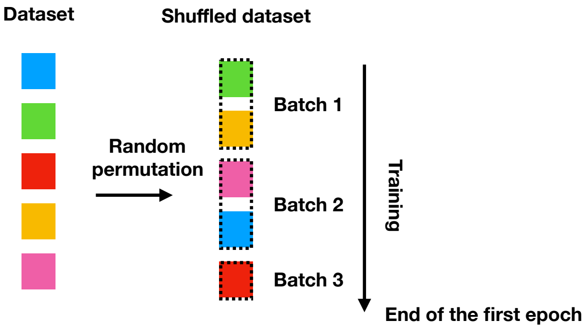
Stochastic Gradient Descent



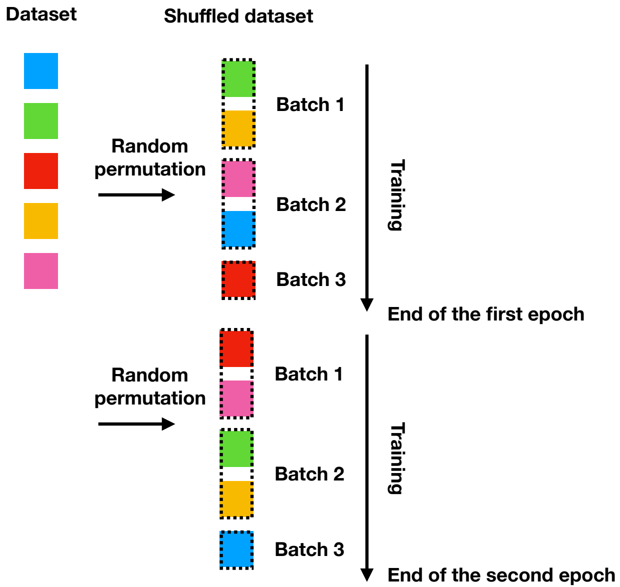
Stochastic Gradient Descent



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

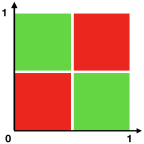
- Linear neural network: $f_\theta(x) := a(x^\top w + b)$
- w are weights, and b is the bias of the layer, a is an activation function.
- SGD allows to learn $\theta := (w, b)$ for arbitrary loss.

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



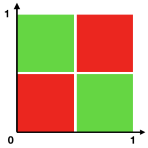
- $(0,0) \mapsto 0$, $(1,1) \mapsto 0$, $(1,0) \mapsto 1$ and $(0,1) \mapsto 1$.
- A linear neural network can not learn the XOR function.

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

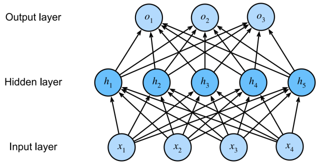


Figure 7: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

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

One-hidden layer Neural Network

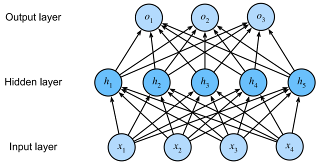


Figure 7: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

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

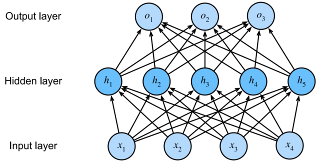


Figure 7: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

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

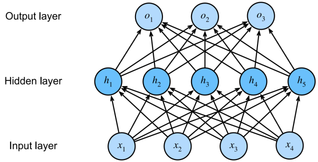


Figure 7: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

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

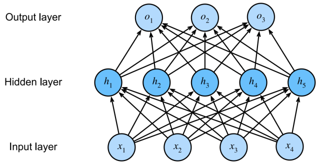


Figure 7: A MLP with one-hidden layer with five units. From `d2l.ai/d2l-en.pdf`.

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. ▷ 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"*.

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:

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 $\triangleright$ $(f_\theta)_{\theta\Theta}$

Overview of usual steps for training a neural network

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

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

Overview of usual steps for training a neural network

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

Overview of usual steps for training a neural network

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

Overview of usual steps for training a neural network

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

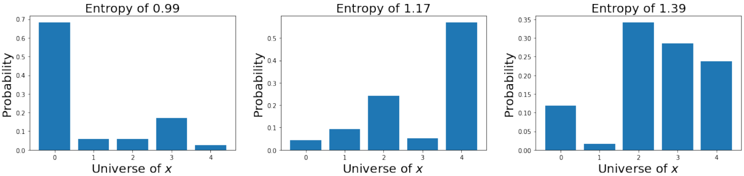
Training a neural network

Defining a loss

Basics of Information theory

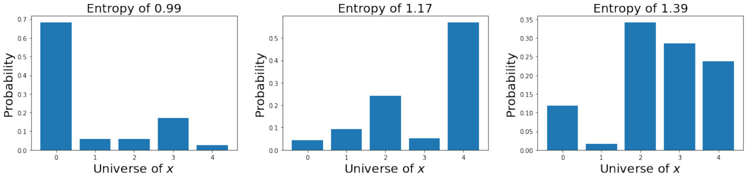
Let p and q two distributions.

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

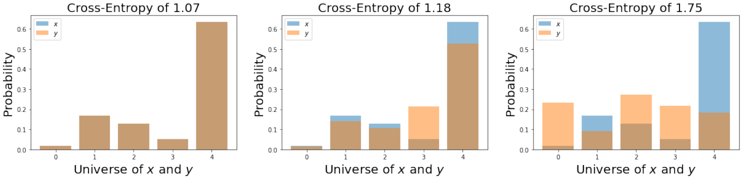


Let p and q two distributions.

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



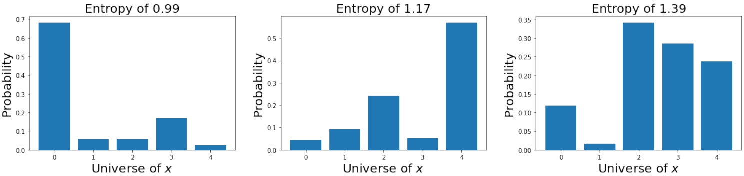
- **Cross-Entropy:** $H(p, q) := \mathbb{E}_{x \sim p}[-\log q(x)]$



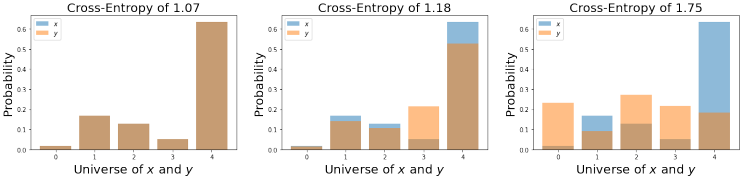
Basics of Information theory

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:
  - $f_\theta(x) \in [0, 1]$ ▷ *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:
  - $f_\theta(x) \in [0, 1]$ ▷ *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 ▷ *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 ▹ *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:

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

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:

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

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:

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

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.

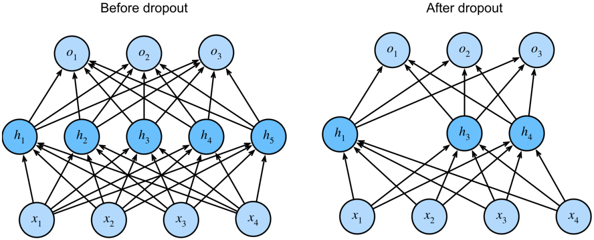


Figure 8: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

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

Dropout: Ensembling of neural networks

Dropout consists in randomly deleting some units during training.

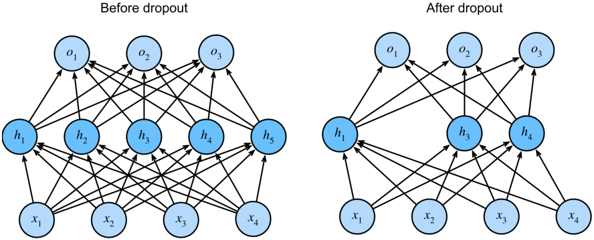


Figure 8: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

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

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

Dropout: Ensembling of neural networks

Dropout consists in randomly deleting some units during training.

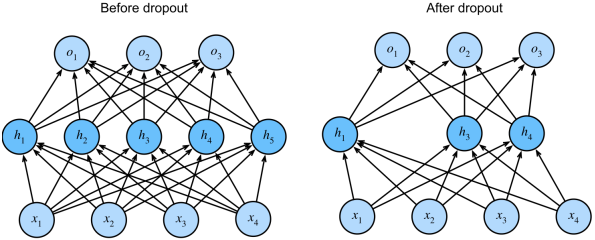


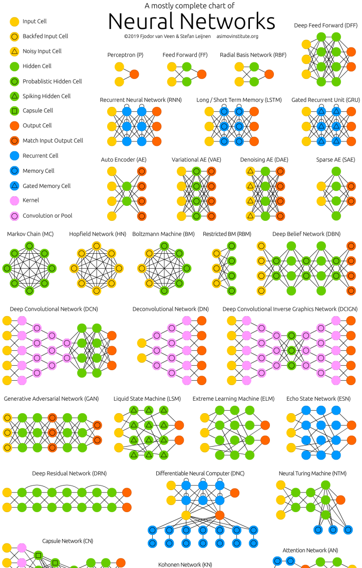
Figure 8: MLP before and after dropout. From `d2l.ai/d2l-en.pdf`.

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

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

Deep neural model zoo

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

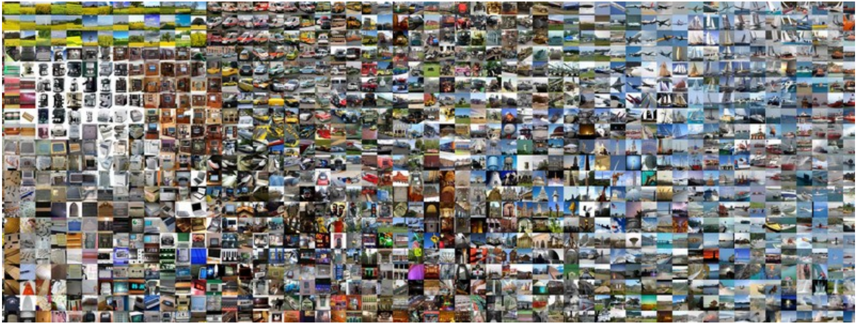


Training a neural network

The Needs

Deep Learning : Needed ressources

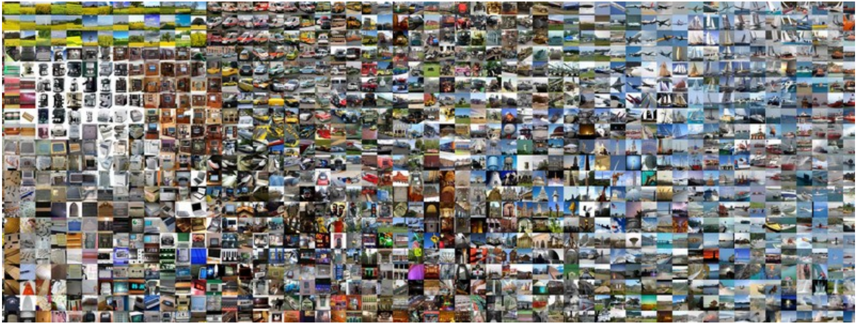
Huge annotated data



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

Deep Learning : Needed ressources

Huge annotated data

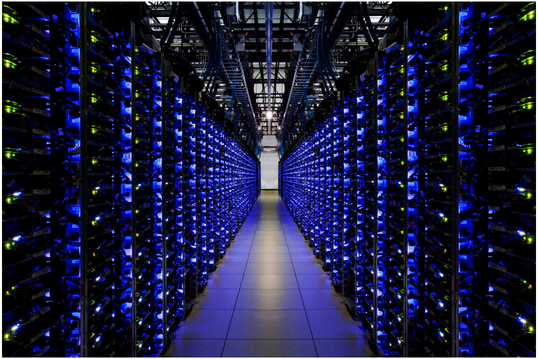


But annotation not prevent from bias in data [2],

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

Deep Learning : Needed ressources

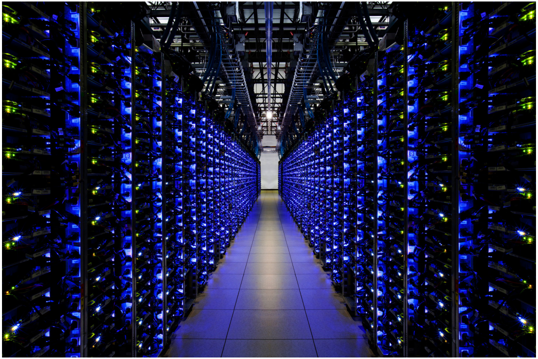
Computing and storage ressources



[3] 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

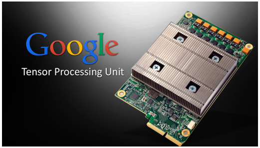


To a green AI [3].

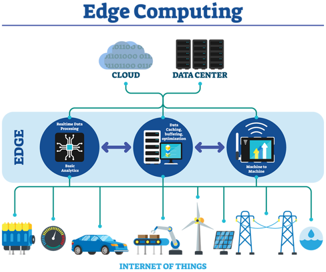
[3] https://www.technologyreview.com/f/614056/ai-research-has-an-environment-climate-toll/?utm_campaign=site_visitor.unpaid.engagement&utm_source=twitter&utm_medium=tr_social

Deep Learning : Needed ressources

Computer science and IT.



The Big Data technology stack is evolving rapidly



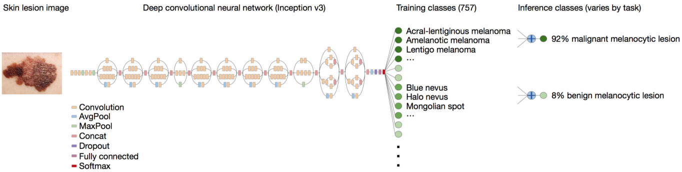
Deep Learning : where are we ?

Deep Learning : where are we ?

Highly performant deep Learning :
More performant than humans?

Let's look at a model that claims it!

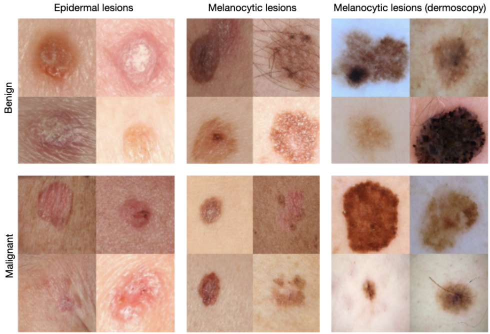
Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]



Let's look at a model that claims it!

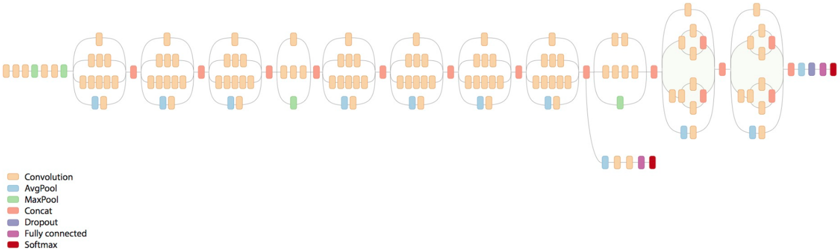
The task

Skin cancer detection : a fine-grained visual recognition task (2,032 different diseases, fine-grained variability) but evaluation is done on two binary classification tasks (*keratinocyte carcinomas versus benign seborrheic keratoses; and malignant melanomas versus benign nevi*).



Let's look at a model that claims it!

The model
Inception v3 CNN architecture, pre-trained on ImageNet and fine-tuned on the target dataset[4]

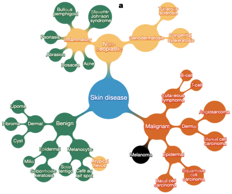


[4] https://ai.googleblog.com/2016/03/train-your-own-image-classifier-with.html

Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]

The data

- 129,450 clinical images, including 3,374 dermoscopy images, **annotated by dermatologists**.
- Images are organized in a tree-structured taxonomy of 2,032 diseases, derived by dermatologists using a bottom-up procedure: individual diseases, initialized as leaf nodes, were merged based on clinical and visual similarity, until the entire structure was connected.
- Training dataset ; 127,463 training and validation images and testing dataset : 1,942 biopsy-labelled test images with no-overlap (same lesion, multiple viewpoints)

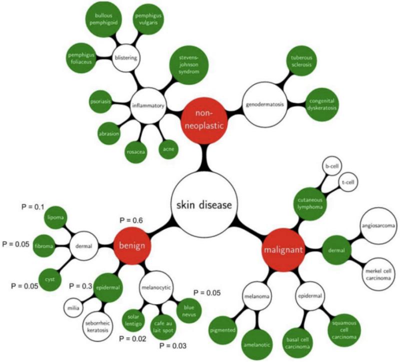


Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]

Drom Disease to training classes
Disease partitioning algorithm : partitions individual diseases into training classes whose individual diseases are clinically and visually similar and with contraints on the size of the class (maxClassSize = 1,000) : **disease partition of 757 classes**.

Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]

From training classes to inference classes.



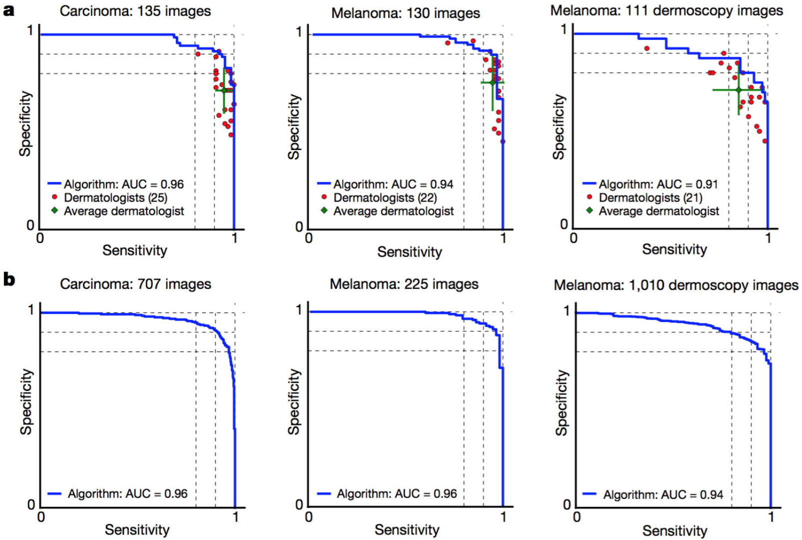
$$P[u] = \sum_{v \in C(u)} P[v]$$

Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]

Experimental protocol

- Test against 21 board-certified dermatologists on biopsy-proven clinical images.
- Two critical binary classification use cases:
  - malignant carcinomas versus benign seborrheic keratoses: identification of the most common cancers
  - malignant melanomas versus benign nevi : identification of the deadliest skin cancer

Dermatologist-level classification of skin cancer with deep neural networks [Esteva et al, 17]



Results

- The richness of the approach is in the building or the database and the ontology-based annotation.
- The classification task is simple.

Performant but what about the confiance on the decision ?

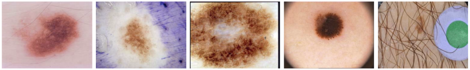
Skin images

- The ISIC database is a database of annotated dermoscopic images.
- Use in different challenges.
- Deep neural networks (AUC=71%) have better results than dermatologists (AUC=67%)

(Bissoto el al, 2019) (De)Constructing Bias on Skin Lesion Datasets (`https://openaccess.thecvf.com/content_CVPRW_2019/papers/ISIC/Bissoto_DeConstructing_Bias_on_Skin_Lesion_Datasets_CVPRW_2019_paper.pdf`)

Performant but what about the confiance on the decision ?

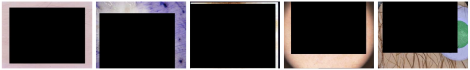
Black boxes pose the problem of bias in the data.



(a) Traditional images

(b) Only Skin images

(c) Bbox images



(d) Bbox70 images

Performant but what about the confiance on the decision ?

Black boxes pose the problem of bias in the data.

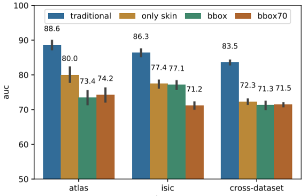


Figure 3: Models' performance over the disturbed datasets. We first remove all the pixel colors inside the lesion (*only skin*), proceeding to remove border information (*bbox*), and finally, removing the size (diameter) of the lesion (*bbox70*). Surprisingly, even when we destruct all clinical-meaningful information, the network finds a way to learn to classify skin lesion images much better than chance.

Deep Learning : where are we ?

Trustworthy?

Not trustworthy !

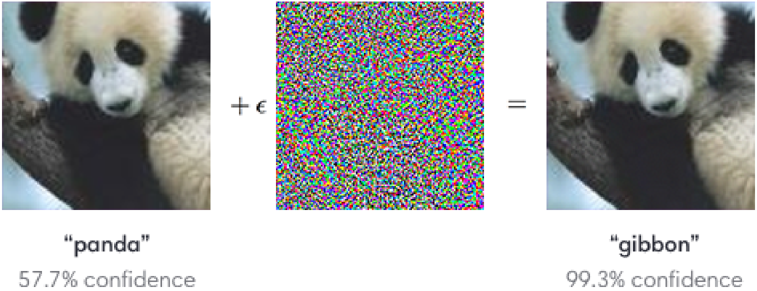


Figure 9: Goodfellow el al, Explaining and Harnessing adversarial examples. See `https://arxiv.org/pdf/1412.6572.pdf`

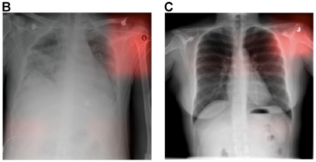
Performant AI systems are often **black box models**, i.e. models, whose internals are either unknown to the observer or they are known but uninterpretable by humans[5]. And **they can be fooled**.

[5]Guidotti et al - A Survey of Methods for Explaining Black Box Models `https://dl.acm.org/doi/pdf/10.1145/3236009`

Not trustworthy !

CNNs learn to predict pneumonia by detecting hospital which took the image

- Study on detecting pneumonia using 158,323 chest radiographs
- CNNs robustly identified hospital system and department within a hospital
- CNN has learned to detect a metal token that radiology technicians place on the patient in the corner of the image



Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study.
Zech JR1, Badgeley MA2, Liu M2, Costa AB3, Titano JJ4, Oermann EK3. https://www.ncbi.nlm.nih.gov/pubmed/30399157

Slide credit : K. Saenko

Not trustworthy !

Many accidents in AI systems !


CSET CENTER for SECURITY and EMERGING TECHNOLOGY

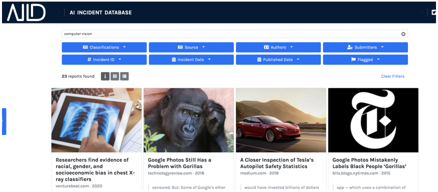
Analysis

AI Accidents: An Emerging Threat

What Could Happen and What to Do

Zachary Arnold and Helen Toner

July 2021



https://cset.georgetown.edu/publication/ai-accidents-an-emerging-threat/
https://incidentdatabase.ai/?lang=en

Not trustworthy !

Concrete problems for AI safety !

- **Avoiding negative side effects** : *Can we transform an RL agent's reward function to avoid undesired effects on the environment?* E.g, build a robot that move an object while avoiding knocking or breaking anything over, without programming a penalty for each possible bad behavior?
- **Safe exploration** : *Can reinforcement learning (RL) agents learn about their environment without executing catastrophic actions?* E.g, RL agent learn to navigate an environment without ever falling off a ledge?
- **Robustness to distributional shift** : *Can machine learning systems be robust to changes in the data distribution, or at least fail gracefully?* E.g, build image classifiers that indicate appropriate uncertainty when shown new kinds of images, instead using inapplicable learned model?
- **Avoiding "reward hacking" and "wireheading"** : *Can we prevent agents from "gaming" their reward functions, such as by distorting their observations?* E.g, train an RL agent to minimize the number of dirty surfaces, without looking for dirty surfaces or creating new dirty surfaces to clean up?
- **Scalable oversight.** : *Can RL agents efficiently achieve goals for which feedback is very expensive?* E.g, build agents that try to clean a room in the way the user would be happiest with, even though feedback is very rare and cheap approximations during training?

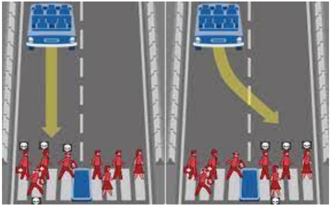
Ethics and dilemna

Autonomous cars and moral decisions [6]
An autonomous car is an intelligent agent capable of perceiving and acting on its environment while moving with little or no human intervention. For the vehicle to move safely and understand its environment, a huge amount of data must be captured by a multitude of different sensors in the car at any given time. This data is then processed by the vehicle's autonomous driving system.

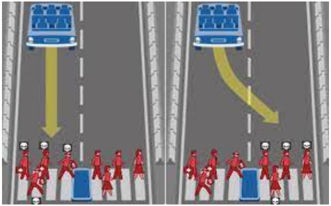
Figure 10: Source :Shutterstock.com/Senha

[6] `https://www.youtube.com/watch?v=HzYG56HLxbI&feature=youtu.be`

Autonomous cars and moral decisions

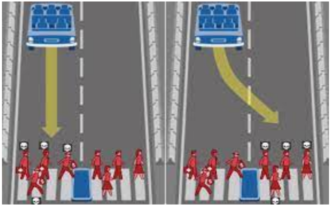


Autonomous cars and moral decisions



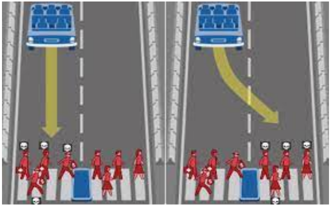
- From a **ethics dilemna** : what should a car do or not do in a specific scenario?

Autonomous cars and moral decisions



- From a **ethics dilemna** : what should a car do or not do in a specific scenario?
- to a **social dilemna**: how to make the company accept and apply the compromises that suit it?

Autonomous cars and moral decisions



- From a **ethics dilemna** : what should a car do or not do in a specific scenario?
- to a **social dilemna**: how to make the company accept and apply the compromises that suit it?
- AI is not just a technical, economical or legislative problem, it is also a problem of society's **cooperation**. `https://www.moralmachine.net/`

Quelles décisions morales les voitures sans conducteur devraient-elles prendre ?

Take the time to look at the TED conference of Iyad Rahwan

`https://www.ted.com/talks/iyad_rahwan_what_moral_decisions_should_driverless_cars_make/transcript`

Requirements for AI adoption

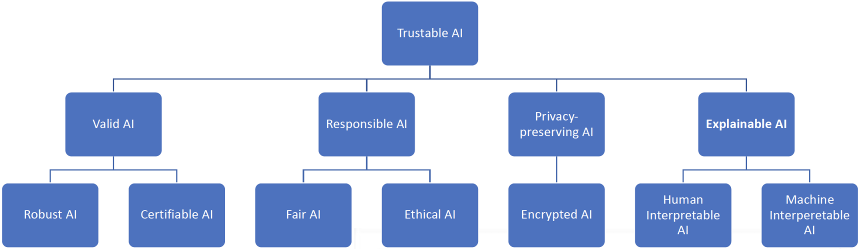


Figure 11: Source : `https://xaitutorial2019.github.io/`

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**