Course 1: Generalities about AI

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Course 1: Generalities about AI

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Welcome to the first lesson of the AI course! I'll a AI do the introduction to this course. Demo on what modern AI system are able to achieve (generative AI based on large language models: chatGPT and Dall-E)

Global overview...

What is AI?

- **Intelligence**: ability to **extract knowledge** from observations
- This knowledge is used to **solve tasks in different contexts and environments** (automation)

Old way: Memorize

- Human experts code the machines
- Goods: we know what we are doing.
- Bads: requires **explicit** solutions (not available for some problems).

Modern way: Generalize

- Let machines teach themselves how to solve a problem (**implicit**).
- Goods: universally applicable
- Bads: lack of understandability/robustness.
- Requires **training**.

Memorizing (explicit) **vs** Learning (implicit)

intelligence: ability to process information to inform future decisions. AI: focuses on building artificial algo that can do the same thing. Historically, this has been done by explicitly telling the machines (=coding algo) how to extract the required knowledge. This had the advantage of exactly knowing how the algo was working but the strong limitation of requiring explicit solution, not available for more complex tasks. The modern way to do AI, what we call ML is to teach machines how to do is without being explicitly programmed.

Global overview…

What is AI?

- **Intelligence**: ability to **extract knowledge** from observations
- This knowledge is used to **solve tasks in different contexts and environments** (automation)

Old way: Memorize

- Human experts code the machines
- Goods: we know what we are doing.
- Bads: requires **explicit** solutions (not available for some problems).

Modern way: Generalize

- Let machines teach themselves how to solve a problem (**implicit**).
- Goods: universally applicable
- Bads: lack of understandability/robustness.
- Requires **training**.

Memorizing (explicit) **vs** Learning (implicit)

intelligence: ability to process information to inform future decisions. AI: focuses on building artificial algo that can do the same thing. Historically, this has been done by explicitly telling the machines (=coding algo) how to extract the required knowledge. This had the advantage of exactly knowing how the algo was working but the strong limitation of requiring explicit solution, not available for more complex tasks. The modern way to do AI, what we call ML is to teach machines how to do is without being explicitly programmed.

Global overview…

What is AI?

- **Intelligence**: ability to **extract knowledge** from observations
- This knowledge is used to **solve tasks in different contexts and environments** (automation)

Old way: Memorize

- Human experts code the machines
- Goods: we know what we are doing.
- Bads: requires **explicit** solutions (not available for some problems).

Modern way: Generalize

- Let machines teach themselves how to solve a problem (**implicit**).
- Goods: universally applicable
- Bads: lack of understandability/robustness.
- Requires **training**.

Memorizing (explicit) **vs** Learning (implicit)

Global overview…
What is AI?
Intelligence: ability to extract knowledge from observations
This knowledge is used to solve tasks in different contexts and environments (automation)
Old way: Memorize
Human experts code the machines
Goods: we know what we are doing.
Bads: requires explicit solutions (not available for some problems).
Modern way: Generalize
Let machines teach themselves how to solve a problem (implicit).
Goods: universally applicable
Bads: lack of understandability/robustness.
Requires training.
Memorizing (explicit) vs Learning (implicit)

intelligence: ability to process information to inform future decisions. AI: focuses on building artificial algo that can do the same thing. Historically, this has been done by explicitly telling the machines (=coding algo) how to extract the required knowledge. This had the advantage of exactly knowing how the algo was working but the strong limitation of requiring explicit solution, not available for more complex tasks. The modern way to do AI, what we call ML is to teach machines how to do is without being explicitly programmed.

Global overview…

What is AI?

- **Intelligence**: ability to **extract knowledge** from observations
- This knowledge is used to **solve tasks in different contexts and environments** (automation)

Old way: Memorize

- Human experts code the machines
- Goods: we know what we are doing.
- Bads: requires **explicit** solutions (not available for some problems).

Modern way: Generalize

- Let machines teach themselves how to solve a problem (**implicit**).
- Goods: universally applicable
- Bads: lack of understandability/robustness.
- Requires **training**.

Memorizing (explicit) **vs** Learning (implicit)

Global overview…
What is AI?
Intelligence: ability to extract knowledge from observations
This knowledge is used to solve tasks in different contexts and environments (automation)
Old way: Memorize
Human experts code the machines
Goods: we know what we are doing.
Bads: requires explicit solutions (not available for some problems).
Modern way: Generalize
Let machines teach themselves how to solve a problem (implicit).
Goods: universally applicable
Bads: lack of understandability/robustness.
Requires training.
Memorizing (explicit) vs Learning (implicit)

intelligence: ability to process information to inform future decisions. AI: focuses on building artificial algo that can do the same thing. Historically, this has been done by explicitly telling the machines (=coding algo) how to extract the required knowledge. This had the advantage of exactly knowing how the algo was working but the strong limitation of requiring explicit solution, not available for more complex tasks. The modern way to do AI, what we call ML is to teach machines how to do is without being explicitly programmed.

Machine learning

- Supervised: Infer a function from inputs/outputs
- Difficulties:
  - Ill-posed problem (infinity of potential solutions)
  - Main approach: seek for particular solutions

Generalization

- Generalization refers to the ability to infer **good decisions or representations** from **examples**, **trials** and/or **interactions**.
- In computer science, we talk about **machine learning**.

Examples

- Learning to play chess through playing games,
- Learning to recognize dogs and cats in images from annotated examples . . .

Generalization and machine learning
Machine learning
Supervised: Infer a function from inputs/outputs
Difficulties:
Ill-posed problem (infinity of potential solutions)
Main approach: seek for particular solutions
Generalization
Generalization refers to the ability to infer good decisions or representations from examples, trials and/or interactions.
In computer science, we talk about machine learning.
Examples
Learning to play chess through playing games,
Learning to recognize dogs and cats in images from annotated examples . . .

The most common framework in ML is Supervised Learning: we learn a function from inputs/examples/data and their outputs/labels. It is an ill-posed problem –> impose constraints to seek for solutions that have desirable properties. One key challenge in ML is Generalization. It can be seen as the ability to generalize on unseen data (different from data (inputs/labels) the model has been trained on. In this course we will only focus on different techniques that are used to build systems that can learn on data, and generalize on unseen data.

Generalization and machine learning

Machine learning

- Supervised: Infer a function from inputs/outputs
- Difficulties:
  - Ill-posed problem (infinity of potential solutions)
  - Main approach: seek for particular solutions

Generalization

- Generalization refers to the ability to infer **good decisions or representations** from **examples**, **trials** and/or **interactions**.
- In computer science, we talk about **machine learning**.

Examples

- Learning to play chess through playing games,
- Learning to recognize dogs and cats in images from annotated examples . . .

Generalization and machine learning
Machine learning
Supervised: Infer a function from inputs/outputs
Difficulties:
Ill-posed problem (infinity of potential solutions)
Main approach: seek for particular solutions
Generalization
Generalization refers to the ability to infer good decisions or representations from examples, trials and/or interactions.
In computer science, we talk about machine learning.
Examples
Learning to play chess through playing games,
Learning to recognize dogs and cats in images from annotated examples . . .

The most common framework in ML is Supervised Learning: we learn a function from inputs/examples/data and their outputs/labels. It is an ill-posed problem –> impose constraints to seek for solutions that have desirable properties. One key challenge in ML is Generalization. It can be seen as the ability to generalize on unseen data (different from data (inputs/labels) the model has been trained on. In this course we will only focus on different techniques that are used to build systems that can learn on data, and generalize on unseen data.

Generalization and machine learning

Machine learning

- Supervised: Infer a function from inputs/outputs
- Difficulties:
  - Ill-posed problem (infinity of potential solutions)
  - Main approach: seek for particular solutions

Generalization

- Generalization refers to the ability to infer **good decisions or representations** from **examples**, **trials** and/or **interactions**.
- In computer science, we talk about **machine learning**.

Examples

- Learning to play chess through playing games,
- Learning to recognize dogs and cats in images from annotated examples . . .

Generalization and machine learning
Machine learning
- Supervised: Infer a function from inputs/outputs
- Difficulties:
  - Ill-posed problem (infinity of potential solutions)
  - Main approach: seek for particular solutions
Generalization
- Generalization refers to the ability to infer **good decisions or representations** from **examples**, **trials** and/or **interactions**.
- In computer science, we talk about **machine learning**.
Examples
- Learning to play chess through playing games,
- Learning to recognize dogs and cats in images from annotated examples . . .

The most common framework in ML is Supervised Learning: we learn a function from inputs/examples/data and their outputs/labels. It is an ill-posed problem –> impose constraints to seek for solutions that have desirable properties. One key challenge in ML is Generalization. It can be seen as the ability to generalize on unseen data (different from data (inputs/labels) the model has been trained on. In this course we will only focus on different techniques that are used to build systems that can learn on data, and generalize on unseen data.

Global formalism

Input/output

- Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
- Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.

Error/Loss

- Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.

Parameters

- $f = f_W$ contains **parameters W** to be trained,
- In most cases, an ideal f_W exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

Global formalism
Input/output
Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.
Error/Loss
Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.
Parameters
$f = f_W$ contains parameters W to be trained,
In most cases, an ideal f_W exists but is hard to find in practice,
Learning is a regression ill-posed problem.

Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exist)

Global formalism

Input/output

- Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
- Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.

Error/Loss

- Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.

Parameters

- $f = f_W$ contains **parameters W** to be trained,
- In most cases, an ideal f_W exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

Global formalism
Input/output
Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.
Error/Loss
Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.
Parameters
$f = f_W$ contains parameters W to be trained,
In most cases, an ideal f_W exists but is hard to find in practice,
Learning is a regression ill-posed problem.

Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exist)

Global formalism

Input/output

- Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
- Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.

Error/Loss

- Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.

Parameters

- $f = f_w$ contains **parameters W** to be trained,
- In most cases, an ideal f_w exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

Global formalism
Input/output
- Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
- Example: input can be an image, output a vector where the largest value indicate the category the image belongs to.
Error/Loss
- Loss $\mathcal{L}$: nonnegative measure of the discrepancy between expected output $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.
Parameters
- $f = f_w$ contains **parameters W** to be trained,
- In most cases, an ideal f_w exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

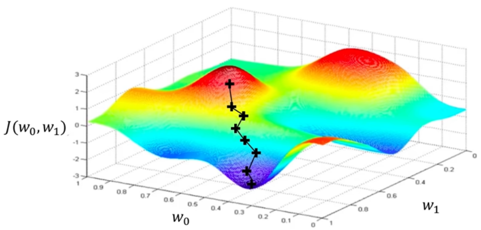
Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exist)

Global formalism

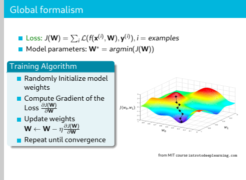
- Loss: $J(\mathbf{W}) = \sum_i \mathcal{L}(f(\mathbf{x}^{(i)}, \mathbf{W}), \mathbf{y}^{(i)}), i = examples$
- Model parameters: $\mathbf{W}^* = argmin(J(\mathbf{W}))$

Training Algorithm

- Randomly Initialize model weights
- Compute Gradient of the Loss $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Update weights $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Repeat until convergence



from MIT course `introtodeeplearning.com`

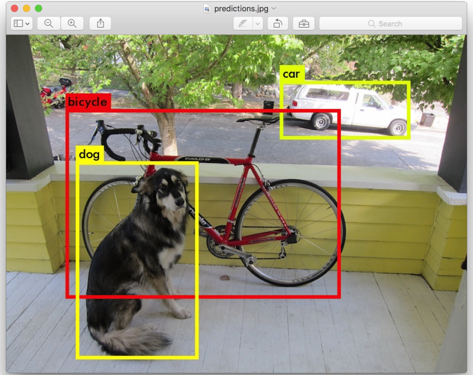


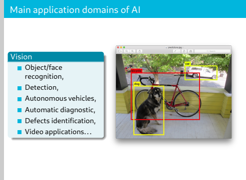
The total loss J (Empirical Risk, Objective function) is the average of Loss for each input/example and the optimal model parameters are those that minimize it. But how to find them? In other words, how to train the model? Here is a simplified description of the training algorithm at the base of modern DL, gradient descent. Repeat until reaching a local minimum (as illustrated in the figure for a simple example where we have only 2 parameters. We'll see that the function becomes much more complicated for millions of parameters -modern neural networks.)

Main application domains of AI

Vision

- Object/face recognition,
- Detection,
- Autonomous vehicles,
- Automatic diagnostic,
- Defects identification,
- Video applications…

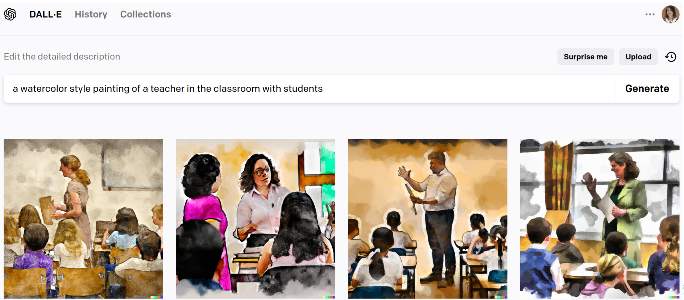


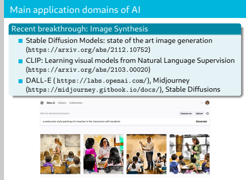


Main application domains of AI

Recent breakthrough: Image Synthesis

- Stable Diffusion Models: state of the art image generation (`https://arxiv.org/abs/2112.10752`)
- CLIP: Learning visual models from Natural Language Supervision (`https://arxiv.org/abs/2103.00020`)
- DALL-E (`https://labs.openai.com/`), Midjourney (`https://midjourney.gitbook.io/docs/`), Stable Diffusions



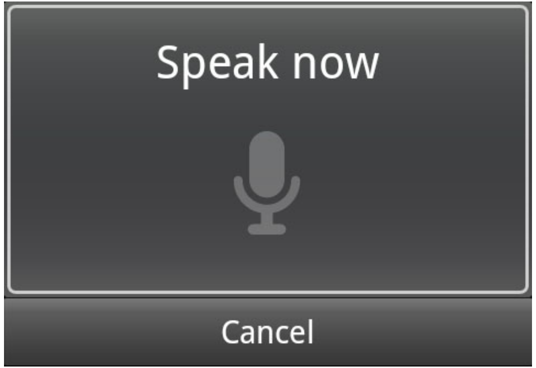


Both use diffusion models to encode and Vision Transformer

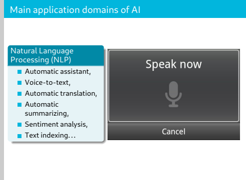
Main application domains of AI

Natural Language Processing (NLP)

- Automatic assistant,
- Voice-to-text,
- Automatic translation,
- Automatic summarizing,
- Sentiment analysis,
- Text indexing…

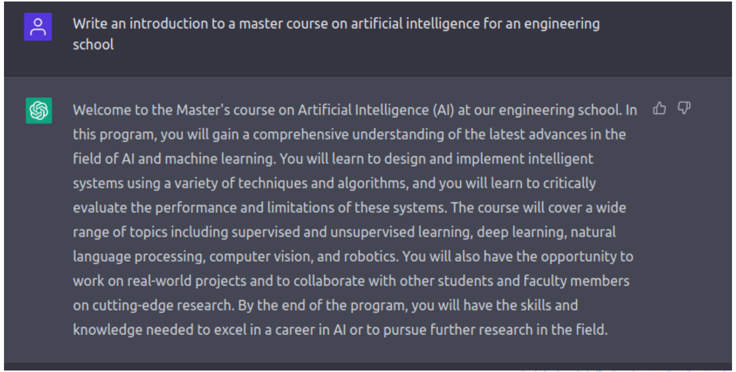


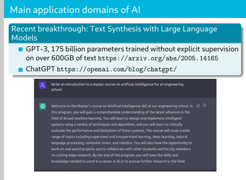
2023-02-02



Recent breakthrough: Text Synthesis with Large Language Models

- GPT-3, 175 billion parameters trained without explicit supervision on over 600GB of text `https://arxiv.org/abs/2005.14165`
- ChatGPT `https://openai.com/blog/chatgpt/`



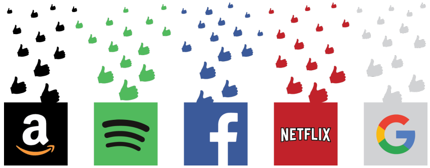


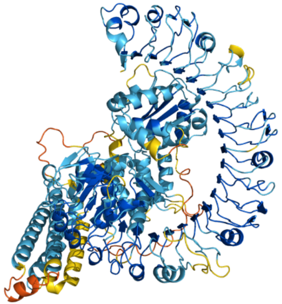
trained on CommonCrawl, Webtext, Books, Wikipedia

Main application domains of AI

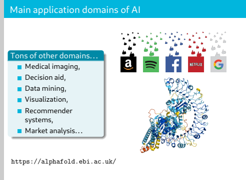
Tons of other domains…

- Medical imaging,
- Decision aid,
- Data mining,
- Visualization,
- Recommender systems,
- Market analysis…





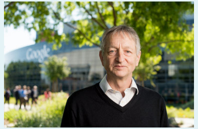
https://alphafold.ebi.ac.uk/



There is probably plenty more application domains. As an example of a breakthrough it is worth mentioning AlphaFold that has revolutioned the field of bioinformatics. The AlphaFold network directly predicts the 3D coordinates of all heavy atoms for a given protein using the primary amino acid sequence. It is an AI system that contains many "ingredients" of modern DL approaches: attention mechanisms, self-distillation,...).

The great elders of modern AI (Turing Prize 2018)

Geoffrey Hinton



- Cognitive psychologist and computer scientist,
- Prof. at University of Toronto and works for Google,
- Known for back-propagation and Boltzmann machines.

Yoshua Bengio

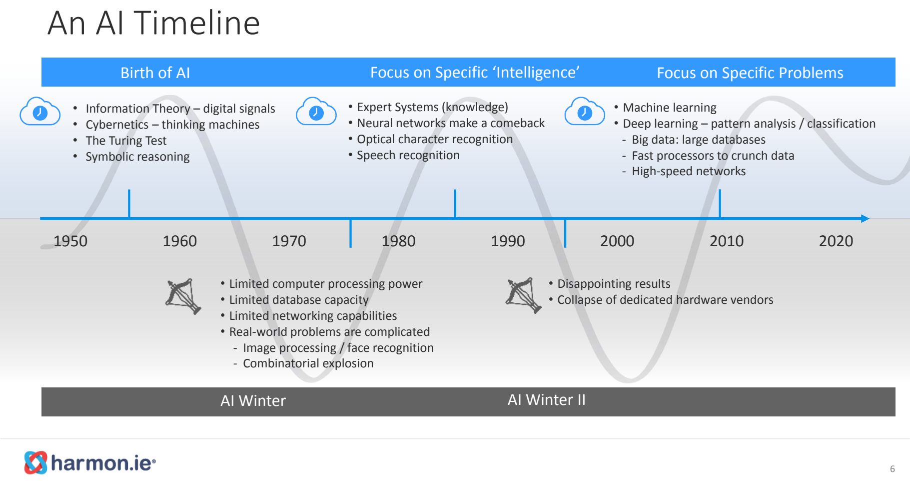
- Computer scientist,
- Prof. at Université de Montréal and head of MILA,
- Known for his work on deep learning.

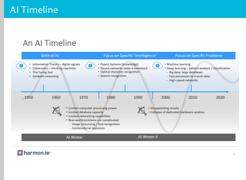
Yann le Cun

- Computer scientist,
- Prof. at New York University then he joins FAIR,
- Known for his work on back-propagation and CNNs.

The reason why we mention them is because their work has mostly enabled to get out of the last two AI winters. It is also worth noting that they won the Turing prize in 2018, which is the highest distinction in computer science.

AI Timeline

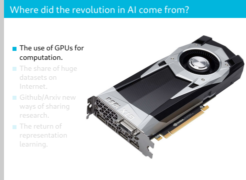




Here, we want to show that field of AI has gone through a number of "winters". This timeline shows that the basic building blocks of DL where there for decades, and algorithms to train them as well! For instance the SGD was proposed in 1952, BackPropagation in 1986. The reasons explaining the success of modern AI which is mostly based on Deep Learning are to be found in other aspects.

Where did the revolution in AI come from?

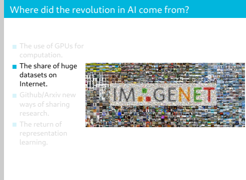
- The use of GPUs for computation.
- The share of huge datasets on Internet.
- Github/Arxiv new ways of sharing research.
- The return of representation learning.



DL training algorithms are highly parallelizable, and can benefit from modern GPU

Where did the revolution in AI come from?

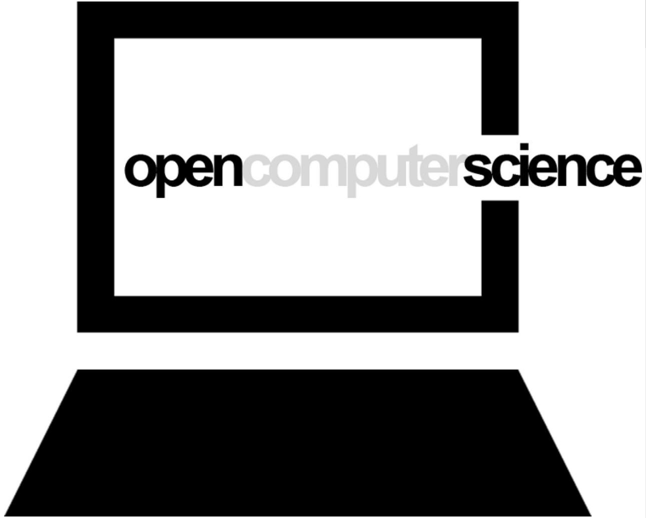
- The use of GPUs for computation.
- The share of huge datasets on Internet.
- Github/Arxiv new ways of sharing research.
- The return of representation learning.



We are in a data pervasive era, massive amount of digitale data is available and has been shared. This benefits DL algo that are extremely data hungry

Where did the revolution in AI come from?

- The use of GPUs for computation.
- The share of huge datasets on Internet.
- Github/Arxiv new ways of sharing research.
- The return of representation learning.

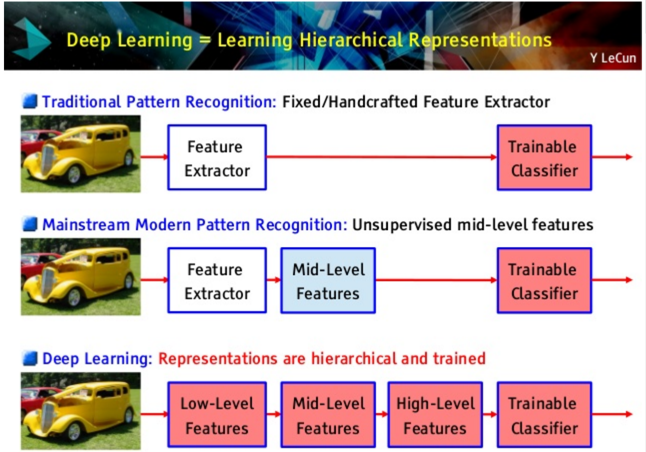


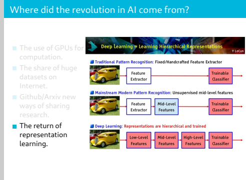
Where did the revolution in AI come from?
The use of GPUs for computation.
The share of huge datasets on Internet.
Github/Arxiv new ways of sharing research.
The return of representation learning.

The availability of open source toolboxes such as pytorch and tensorflow and the practice of sharing research content and tools makes implementing and training DL models much easier, and you will find out in this course

Where did the revolution in AI come from?

- The use of GPUs for computation.
- The share of huge datasets on Internet.
- Github/Arxiv new ways of sharing research.
- The return of representation learning.



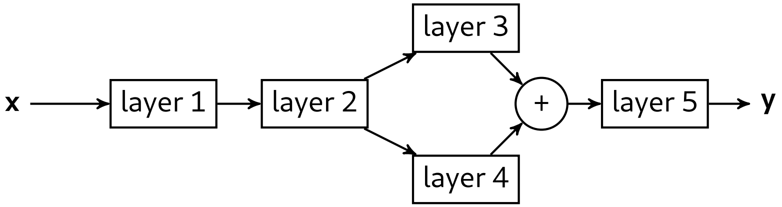


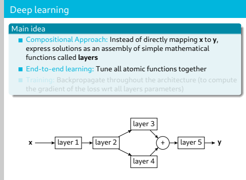
A few more explanation on Representation learning. As we will see in the next lessons, finding a good representation of the data is a very difficult thing. Deep Learning has enabled to search / decompose the data automatically to find such representations. In traditional pattern recognition, features were extracted using a priori expert knowledge on the data (eg looking for orange round objects to recognize oranges, by filter the colour orange, and extracting round objects). In mainstream pattern recognition, mathematical functions (eg wavelets) are used to automatically decompose images in sets of features that are more abstract, but still expertly chosen depending on the data. In Deep Learning: the features are trained end to end.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

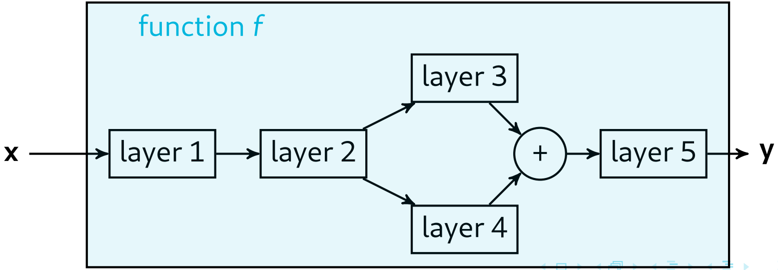




DL at is core is the ability to learn higher and higher level representations or features from data in a end to end fashion. How? By means of a compositional approach of simple mathematical functions (layers). Representation are useful to interpret data: ideally the final representation should be easy to deal with (to classify, to generate data from...). What is new about DL is that we do that in a end-to-end fashion starting from raw data. Also, in DL, we use deep architectures with hidden layers to approximate any complex function f. So the fundamental blocks of NN are layers, each layers has is own parameters (weights and bias) that need to be trained.

Main idea

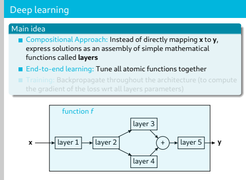
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



2023-02-02

Course 1: Generalities about AI

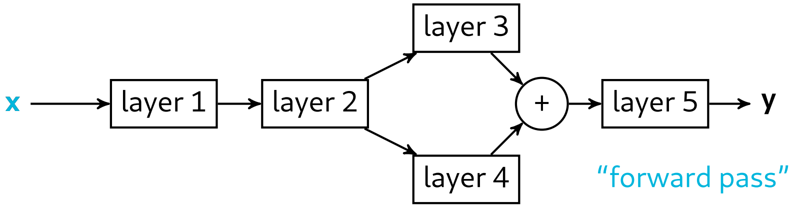
└─Deep learning



Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

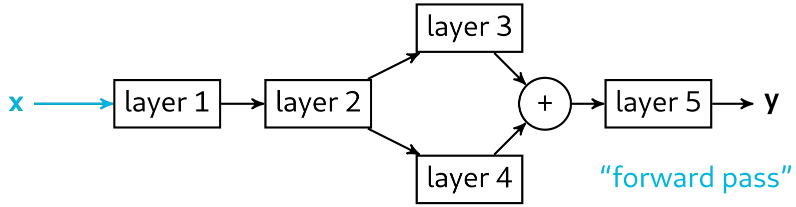


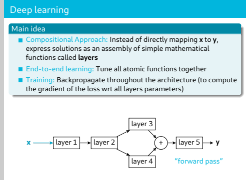
Deep learning
Main idea
Compositional Approach: Instead of directly mapping x to y, express solutions as an assembly of simple mathematical functions called layers
End-to-end learning: Tune all atomic functions together
Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



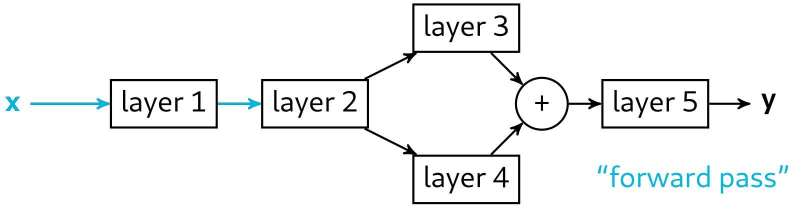


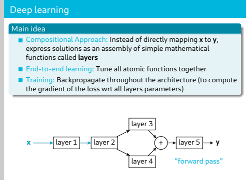
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

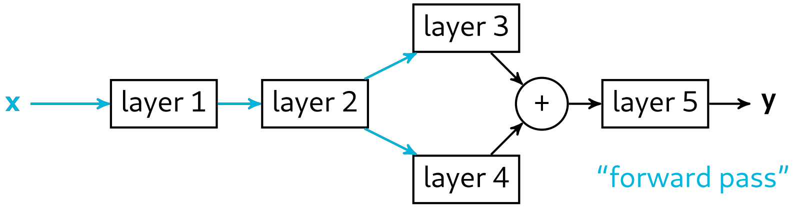


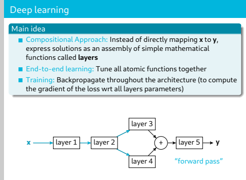


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



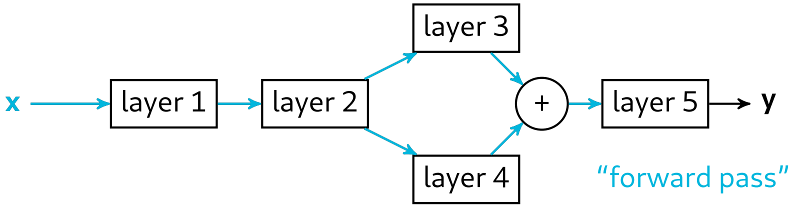


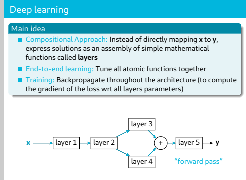
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



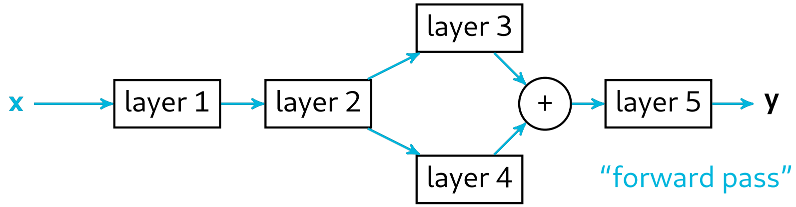


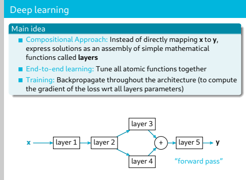
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



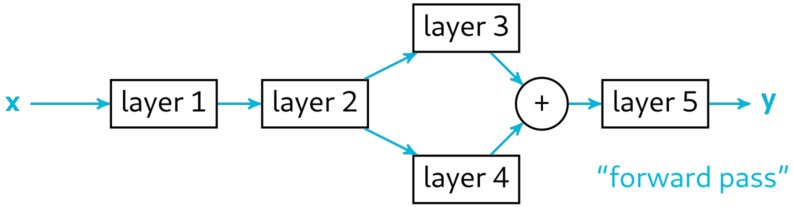


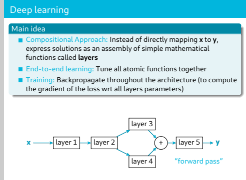
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

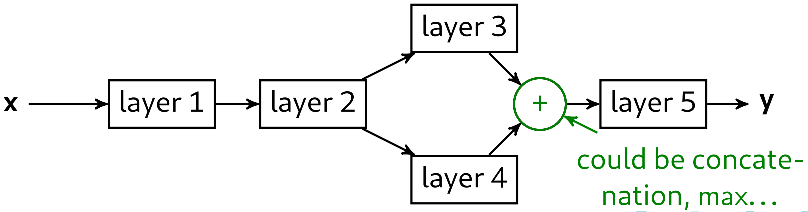


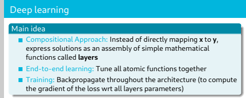


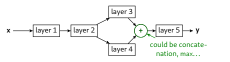
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Main idea

- Compositional Approach: Instead of directly mapping $\mathbf{x}$ to $\mathbf{y}$, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)





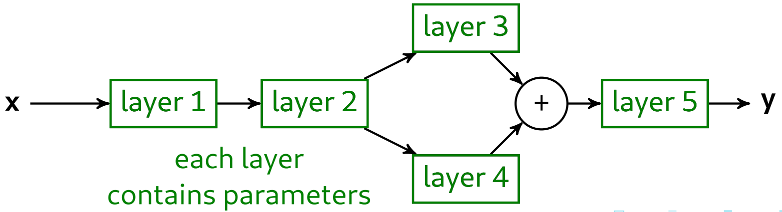


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



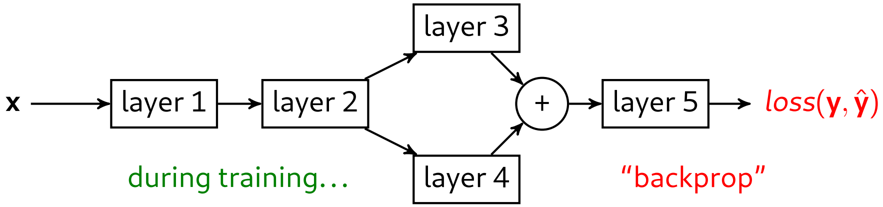
Deep learning
Main idea
Compositional Approach: Instead of directly mapping x to y, express solutions as an assembly of simple mathematical functions called layers
End-to-end learning: Tune all atomic functions together
Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

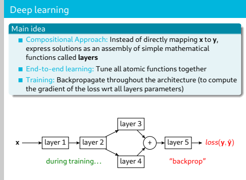
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

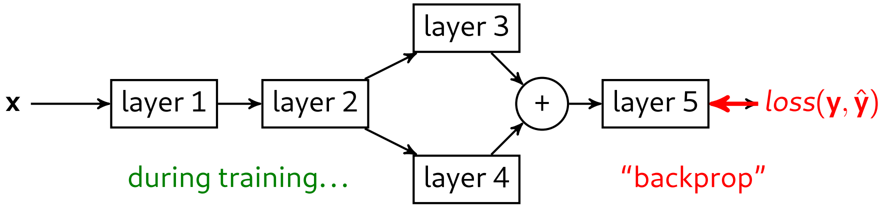


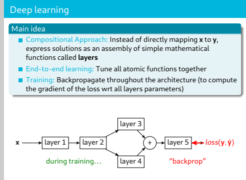


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



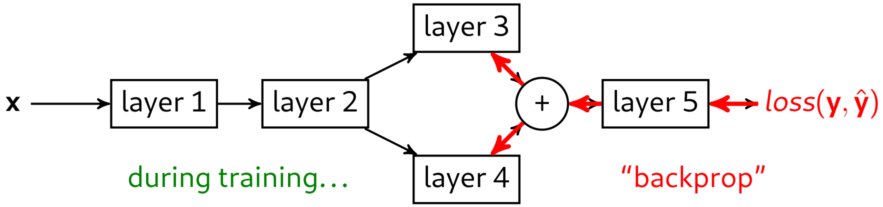


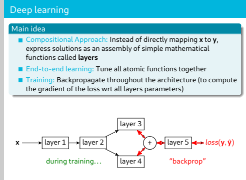
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



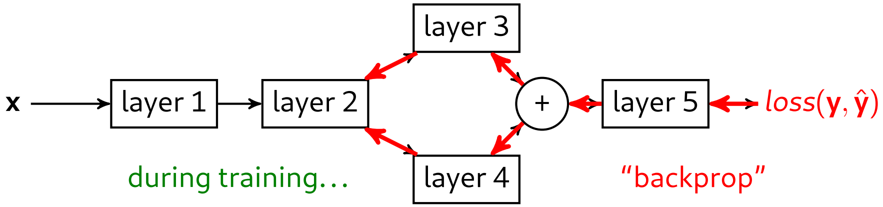


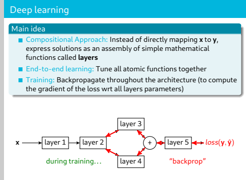
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

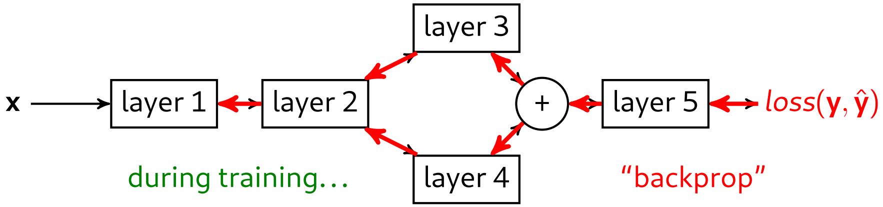




As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Main idea

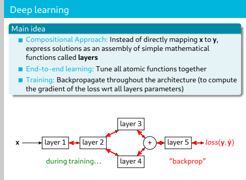
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



2023-02-02

Course 1: Generalities about AI

└─Deep learning

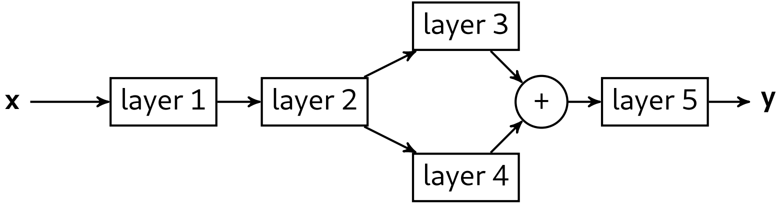


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

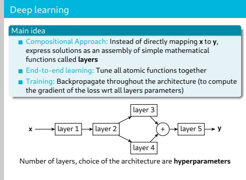
Deep learning

Main idea

- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



Number of layers, choice of the architecture are **hyperparameters**



As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The forward pass the input passes through the network and it gives an output y. The backward pass propagates the derivatives of Loss wrt the weights for each layer. Model weights are then updated with the rule we have seen $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ so that those responsible for producing the right input are increased, and the other decreased.

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: fully-connected layers,
    - Can be structure-dependent: convolutional layers.

Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive elds, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling. Le Cun 1990

Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: **fully-connected layers**,
    - Can be structure-dependent: **convolutional layers**.

Some additional details
Layers
$\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
h is a nonlinear parameterwise function (often without parameters),
$\mathbf{W}$ is a tensor:

Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive elds, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling. Le Cun 1990

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: **fully-connected layers**,
    - Can be structure-dependent: **convolutional layers**.

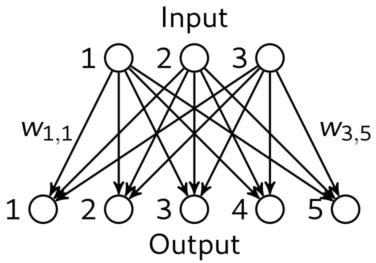
Some additional details
Layers
$\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
h is a nonlinear parameterwise function (often without parameters),
$\mathbf{W}$ is a tensor:
Can be agnostic of the structure: fully-connected layers,
Can be structure-dependent: convolutional layers.

Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive elds, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling. Le Cun 1990

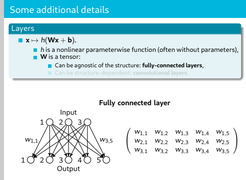
Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: **fully-connected layers**,
    - Can be structure-dependent: **convolutional layers**.

Fully connected layer



$$\begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \end{pmatrix}$$



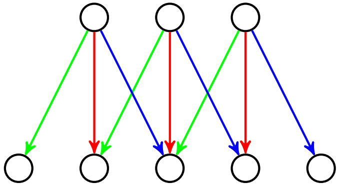
Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive elds, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling. Le Cun 1990

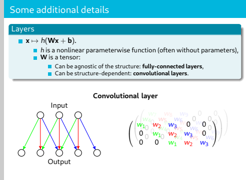
Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: **fully-connected layers**,
    - Can be structure-dependent: **convolutional layers**.

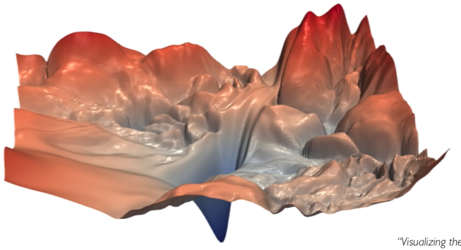
Convolutional layer





Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive elds, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling. Le Cun 1990

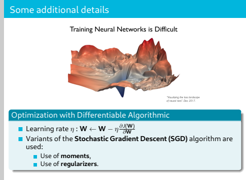
Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

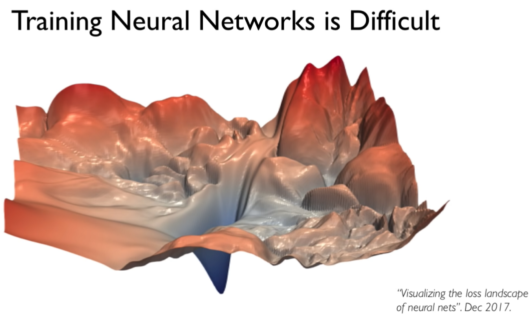
Optimization with Differentiable Algorithmic

- Learning rate η : $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Variants of the **Stochastic Gradient Descent (SGD)** algorithm are used:
  - Use of **moments**,
  - Use of **regularizers**.



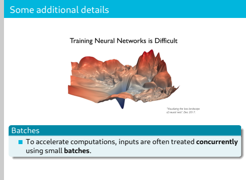
Training a NN is a challenging task: this picture represents the loss landscape of a typical DL network with million of parameters, something very different from the version we have seen before for 2 parameters, extremely complex and with many local minima. Optimization depends of different factors but one of the most crucial one is the learning rate (the fraction of the gradient that is subtracted from the loss) as it determines the convergence of the SGD: it should be large enough to avoid local minima, but small enough to converge. Most of modern implementation use an adaptive lr (increase, decrease during training): try out different adaptive schemes during the lab! Also different optimizers, all variants of SGD can be explored. To increase generalization (or in other words, avoid overfitting) different regularizations techniques are also used (dropout, early stopping). Momentum: help reduce variance: accumulates a decaying moving average of past gradients so the gradient step depends on how aligned past gradients are

Some additional details



Batches

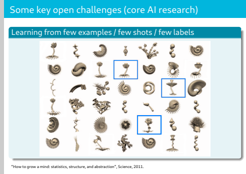
- To accelerate computations, inputs are often treated **concurrently** using small **batches**.



Backprop is computationally intensive if performed for each data example. One way to accelerate computation is the compute the gradient of batches (or small group) of training examples. This also gives a better estimate of the gradient, allows for paralellization and higher lr. Of course there is a tradeoff between higher speed (large batches) and better generalization: batch size is a hyperparameter itself. Recap: batch: gradient step, epoch: iteration over the entire dataset (ensemble of batches)

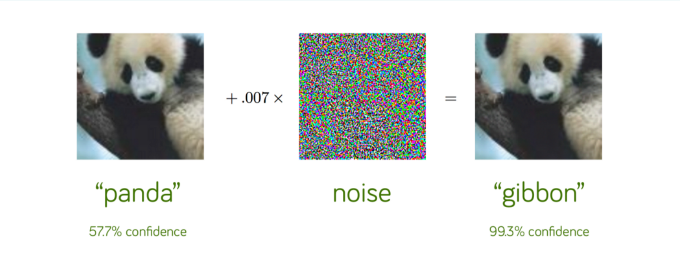
Learning from few examples / few shots / few labels

“How to grow a mind: statistics, structure, and abstraction”, Science, 2011.



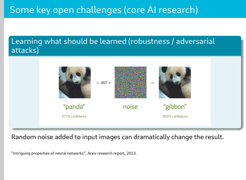
About Learning from few examples: this picture is from the science paper from Josh Tenenbaum, and is explained in the following way. It is very easy for humans with just a few examples of these three weird objects (let’s call them "tufas"), and if I ask you where are the other ones, you can tell me right away. However it is a difficult problem for current AI approach (high resolution images: very complex problems)

Learning what should be learned (robustness / adversarial attacks)



Random noise added to input images can dramatically change the result.

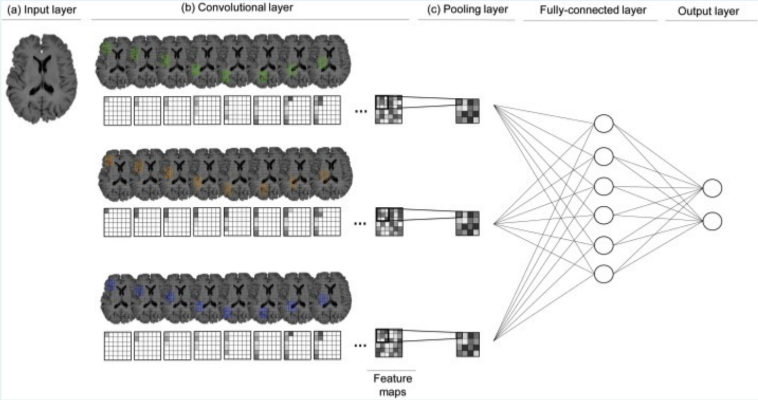
"Intriguing properties of neural networks", Arxiv research report, 2013.



Learning what should be learnt: the point here is to talk about robustness and adversarial attacks. The actual classes that is represented is "panda" and when small amount of noise is added, the algorithm erroneusly classify the image as "gibbon" with a high confidence.

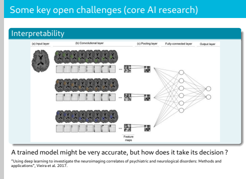
Some key open challenges (core AI research)

Interpretability



A trained model might be very accurate, but how does it take its decision ?

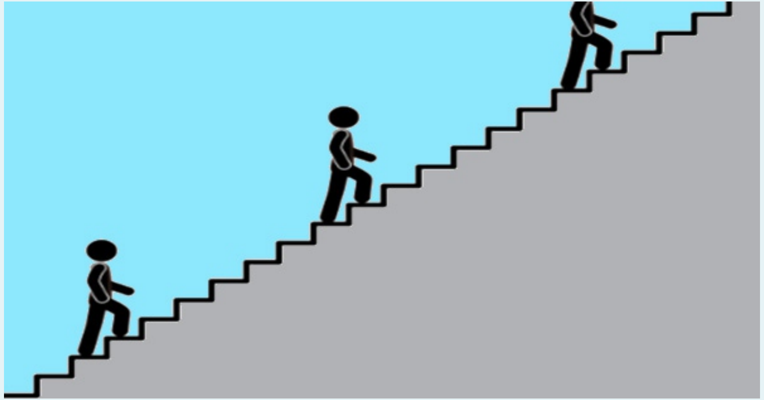
“Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications“, Vieira et al. 2017.



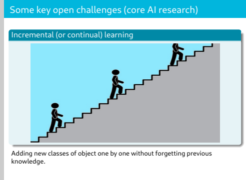
This is also somehow related to the problem of Interpretability and that we do not exactly know how the algo take decisions: the image is here to illustrate an use-case (neuroimaging) in which it can be difficult to know if the network has learnt correctly. The reason is easier to understand when comparing with a more classical image recognition task : a human can check whether an AI that learns to classify dogs versus cats is performing well by just checking the images. But with brain images (let’s say recognizing Parkinson Disease from brain structure), we can’t do that, because we don’t know necessarily what a Parkinson brain looks like.

Some key open challenges (core AI research)

Incremental (or continual) learning



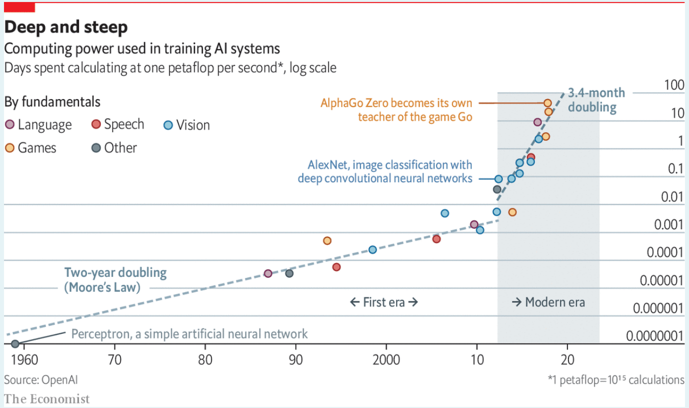
Adding new classes of object one by one without forgetting previous knowledge.



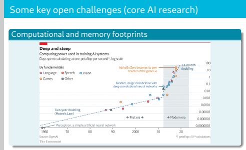
The problem of incremental learning comes from the fact that most methods can't be retrained after they have been trained. Therefore, they suffer from what is called "catastrophic forgetting", which is the fact that previously learnt knowledge is erased by the new knowledge, so the algorithm performs poorly on the newly learnt knowledge Computational and memory footprints : modern algorithms involve thousands or millions of parameters and computations. In this picture we represent the computing power used to train AI models in days spent calculating at one petaflop floating point operations) per second: we recognize the Moore law trend (number of transistors in a dense integrated circuit doubles about every two years) and the sudden increase in computational need of the modern era, also due to the GPU use.

Some key open challenges (core AI research)

Computational and memory footprints



Training a large algorithm: thousands to millions of parameters using Gigabytes of data.


Training a large algorithm: thousands to millions of parameters using Gigabytes of data.

The problem of incremental learning comes from the fact that most methods can't be retrained after they have been trained. Therefore, they suffer from what is called "catastrophic forgetting", which is the fact that previously learnt knowledge is erased by the new knowledge, so the algorithm performs poorly on the newly learnt knowledge
Computational and memory footprints : modern algorithms involve thousands or millions of parameters and computations. In this picture we represent the computing power used to train AI models in days spent calculating at one petaflop floating point operations) per second: we recognize the Moore law trend (number of transistors in a dense integrated circuit doubles about every two years) and the sudden increase in computational need of the modern era, also due to the GPU use.

Course organisation

Sessions

1. Generalities about AI (today),
2. Supervised learning,
3. Unsupervised learning,
4. Combinatorial game theory,
5. Reinforcement learning,
6. Work on final project,
7. Work on final project,
8. Ethics in AI + Challenge.

Sessions schedule

Each session has (roughly) the same structure:

- Short written exam about the previous lesson (10 min),
- Short lesson (20 min),
- Lab Session ,
- Project
- Sessions 2, 3, 4 and 8 include students' presentations

2023-02-02

Course 1: Generalities about AI

└Course organisation

Course organisation
Sessions
1 Generalities about AI (today),
2 Supervised learning,
3 Unsupervised learning,
4 Combinatorial game theory,
5 Reinforcement learning,
6 Work on final project,
7 Work on final project,
8 Ethics in AI + Challenge.
Sessions schedule
Each session has (roughly) the same structure:
Short written exam about the previous lesson (10 min),
Short lesson (20 min),
Lab Session ,
Project
Sessions 2, 3, 4 and 8 include students' presentations

Course Evaluation

Lab Sessions and Challenge

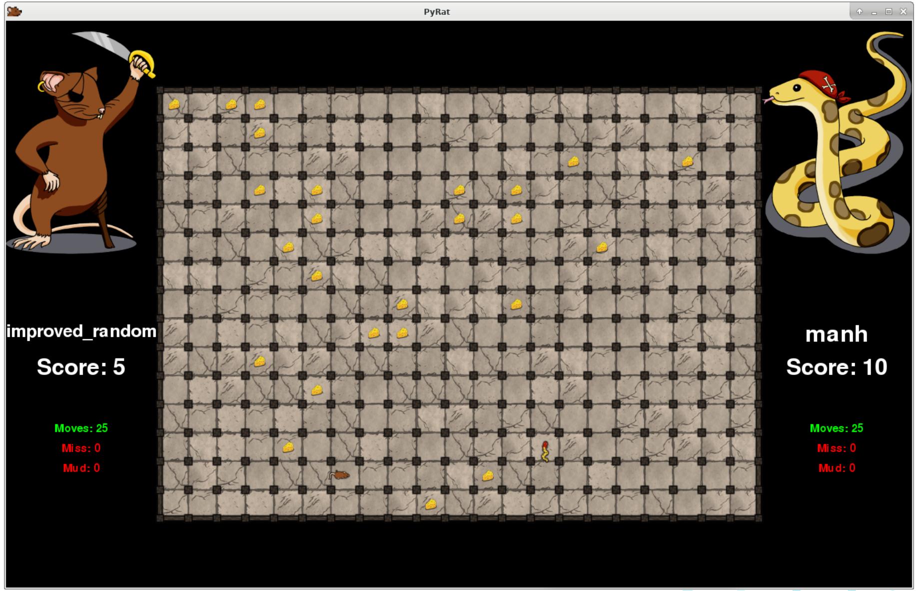
- By groups of two, you will be working on campus machines (python, deeplearningu20 environment).
- Lessons, practicals and presentations in room K01-112

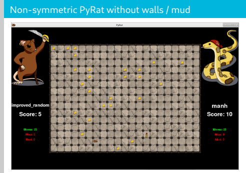
Evaluations

- QCM on moodle at 9:30 am **compulsory**
- Presentations - sessions 2,3,4 + **Final Deliverable (Challenge)** - session 8

Course Evaluation
Lab Sessions and Challenge
By groups of two, you will be working on campus machines (python, deeplearningu20 environment).
Lessons, practicals and presentations in room K01-112
Evaluations
QCM on moodle at 9:30 am **compulsory**
Presentations - sessions 2,3,4 + **Final Deliverable (Challenge)** - session 8

Non-symmetric PyRat without walls / mud





Lab Session 1 and assignment

Lab Session

- Introduction to Jupyter Notebook, numpy, scipy
- Generating Pyrat games
- Visualisation using Matplotlib

Project 0 (P0) (oral presentation)

You will choose a topic on an application of AI.
You have to prepare a 7 minutes presentation (for session 2) in which you quickly explain :

- What the topic is about
- What solutions already exist
- Examples of companies / existing products on this topic
- Example of ethical considerations related to the topic
- Current limitations and hard problems

Lab Session 1 and assignment
Lab Session
Introduction to Jupyter Notebook, numpy, scipy
Generating Pyrat games
Visualisation using Matplotlib
Project 0 (P0) (oral presentation)
You will choose a topic on an application of AI.
You have to prepare a 7 minutes presentation (for session 2) in which you quickly explain :
What the topic is about
What solutions already exist
Examples of companies / existing products on this topic
Example of ethical considerations related to the topic
Current limitations and hard problems