Intro to Deep Learning

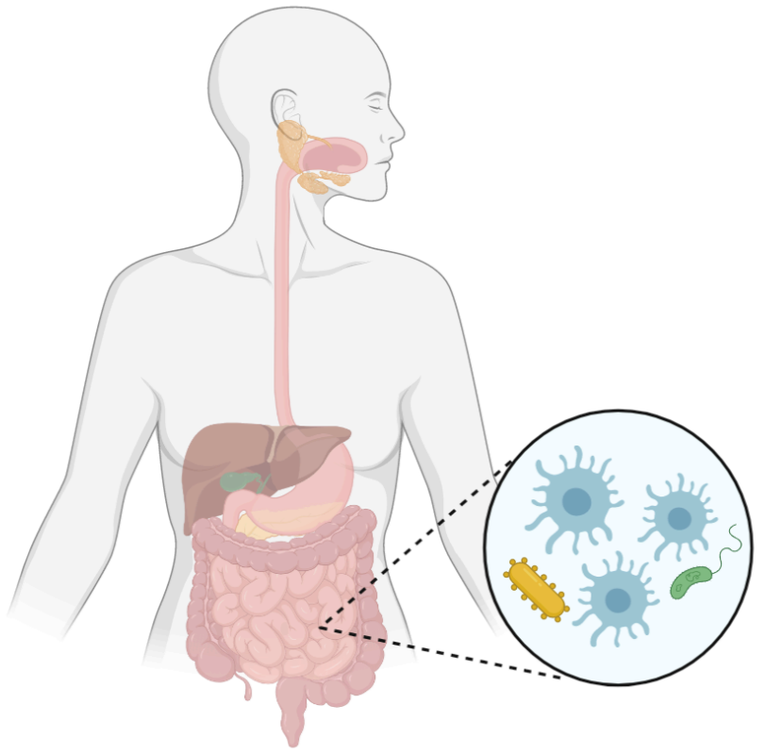
Romain Menegaux - *Inria Grenoble*

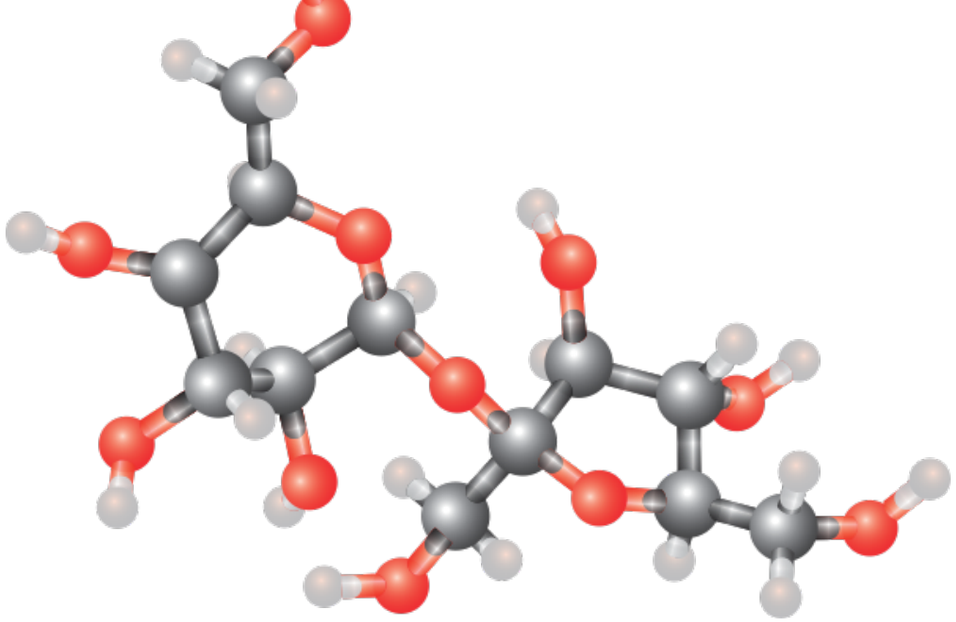
Sacl-AI 4 Science Workshop - July 9th, 2024

(with material adapted from Thomas Moreau, Julien Mairal, Mathurin Massias 🙏)

Who am I ?

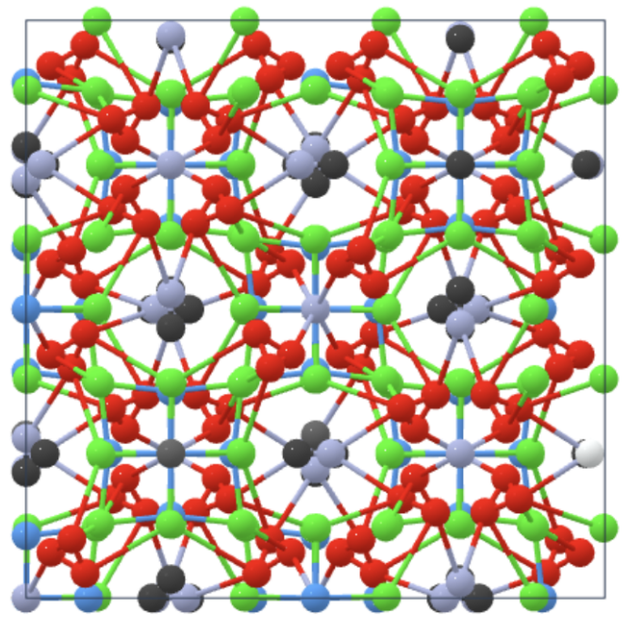
- PhD in Paris, applying NLP neural networks on DNA
- PostDoc at Inria Grenoble: Graph Neural Networks and molecule property prediction



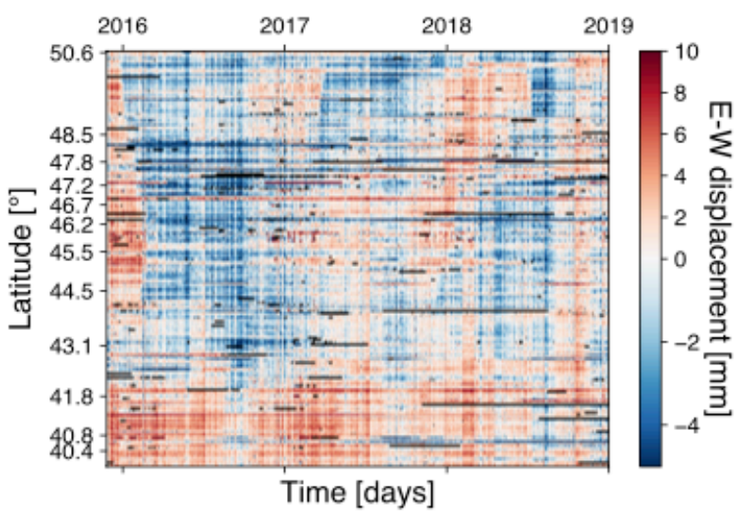


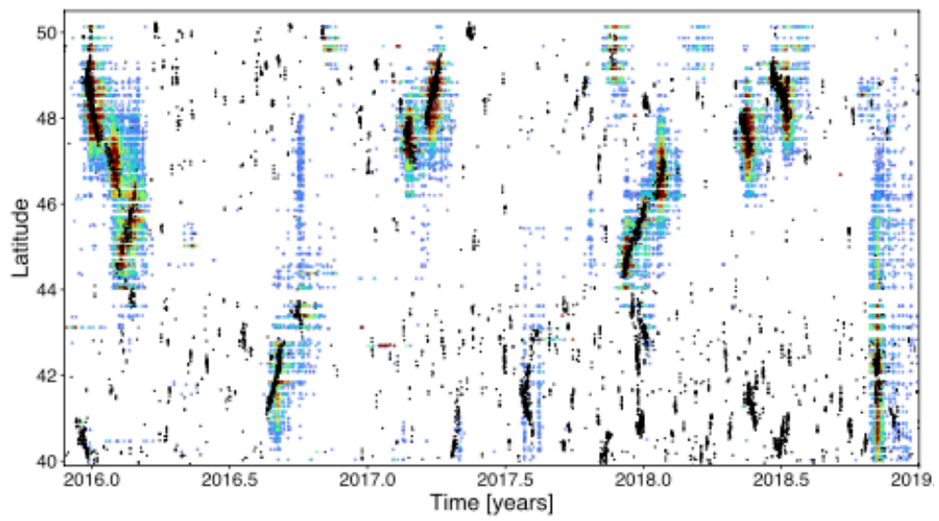
Who am I ?

- PhD in Paris, applying NLP neural networks on DNA
- PostDoc at Inria Grenoble: Graph Neural Networks and molecule property prediction
- Current projects:



Crystal generation, flow matching





Denoising seismic data with transformers

Outline

- Introduction
  - **Why** should you care
  - Machine learning overview
- **How** Deep Learning works, Inner workings
  - Basic Neural Net, backprop
  - Model Architectures
  - Training Tips and Strategies
- **Where**/**when** to apply it, problem definition

Context and problem setting

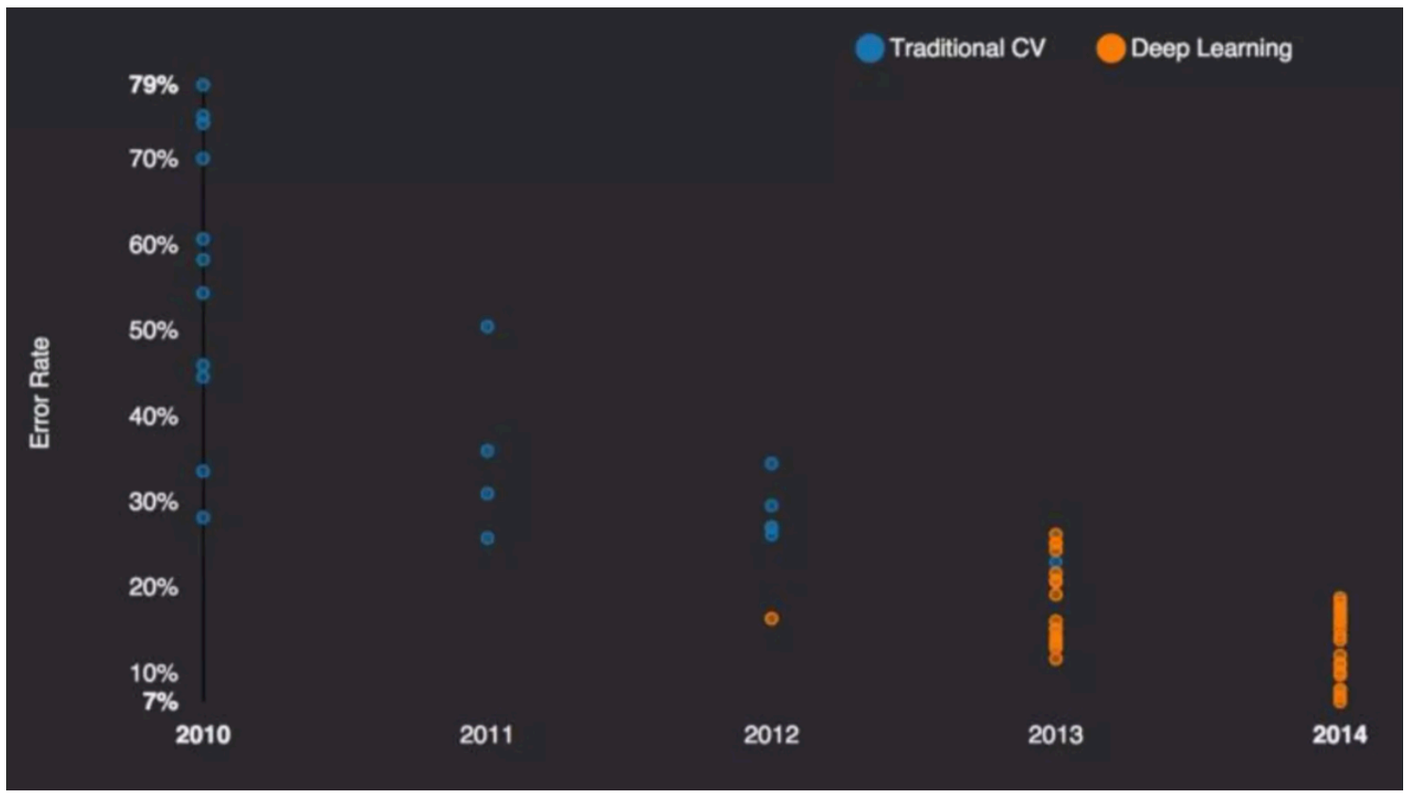
Why Deep Learning ?

- For many fields there has been a before and after
- Computer vision
- Language
- Games
- Molecular biology (AlphaFold)

If you can’t beat them, join them

Vision

[Stanford 2017]



[ImageNet 2010-2014]

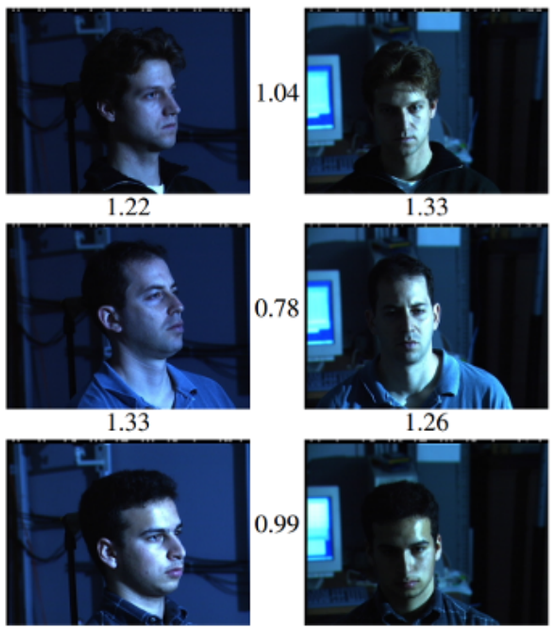
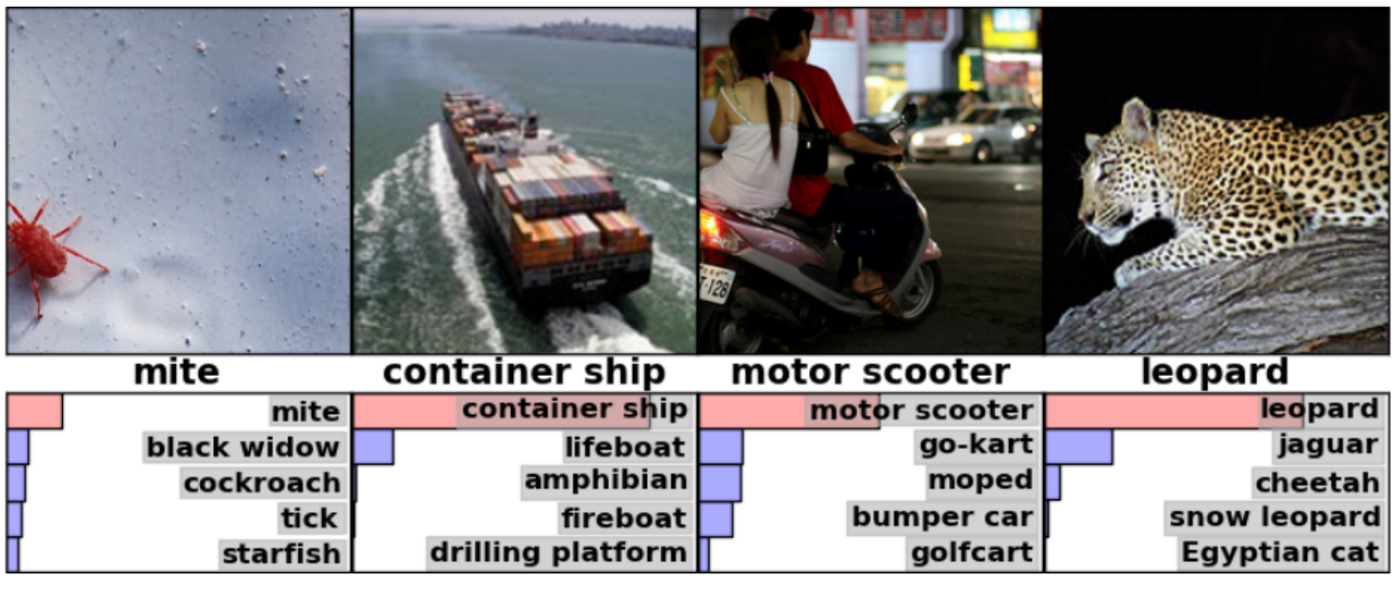


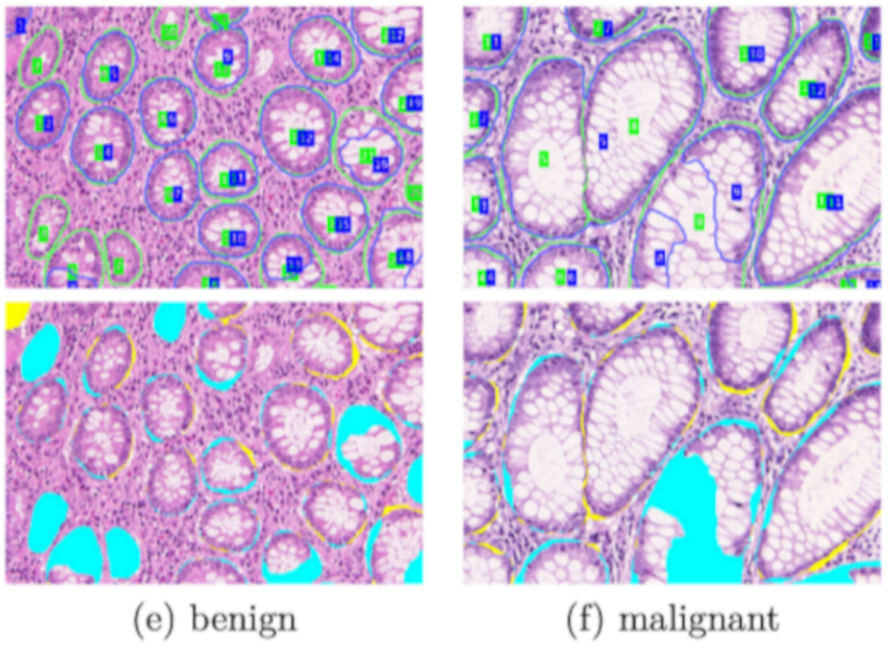
Figure 1. **Illumination and Pose invariance.**

[FaceNet - Google 2015]



[Krizhevsky 2012]

Vision

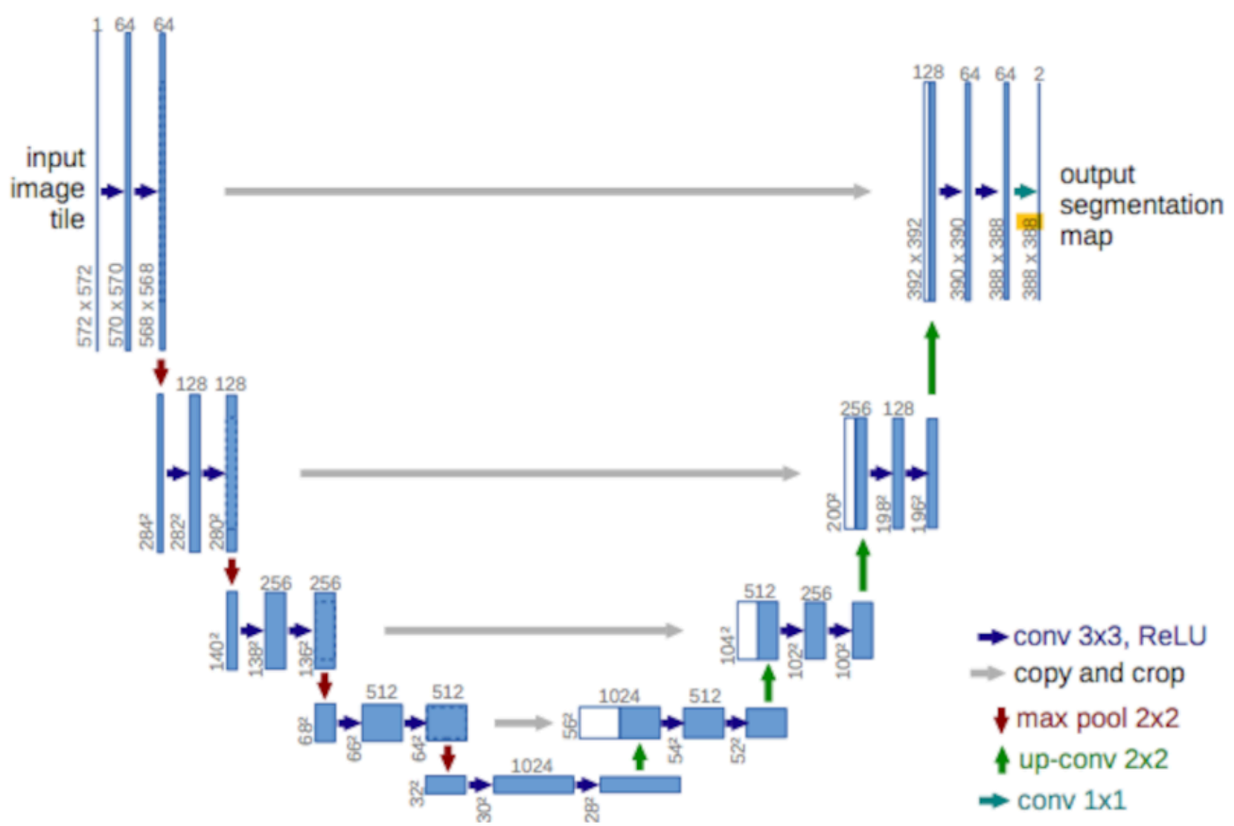


(e) benign (f) malignant

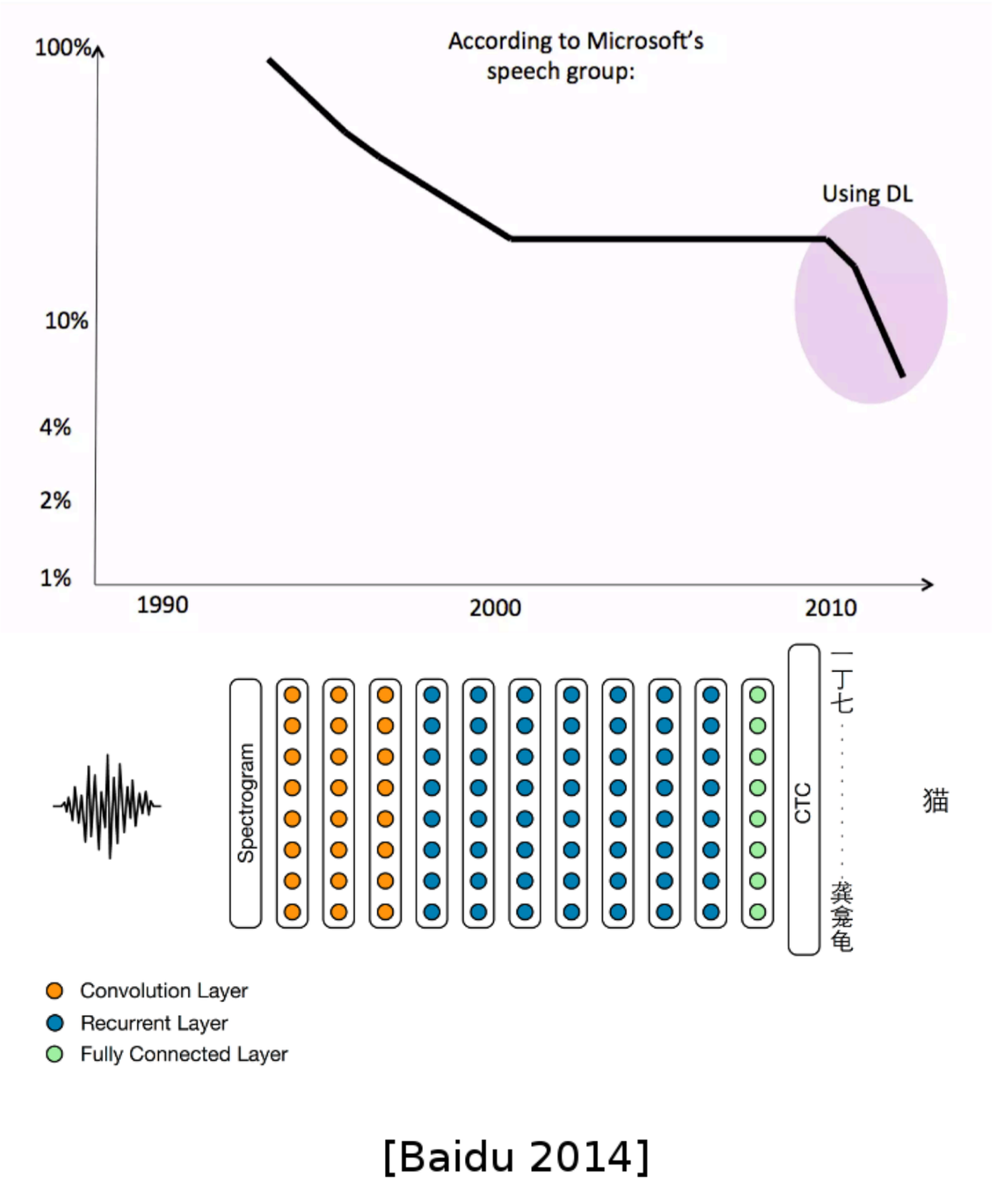
[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

U-Net
Ronneberger et al.
2015

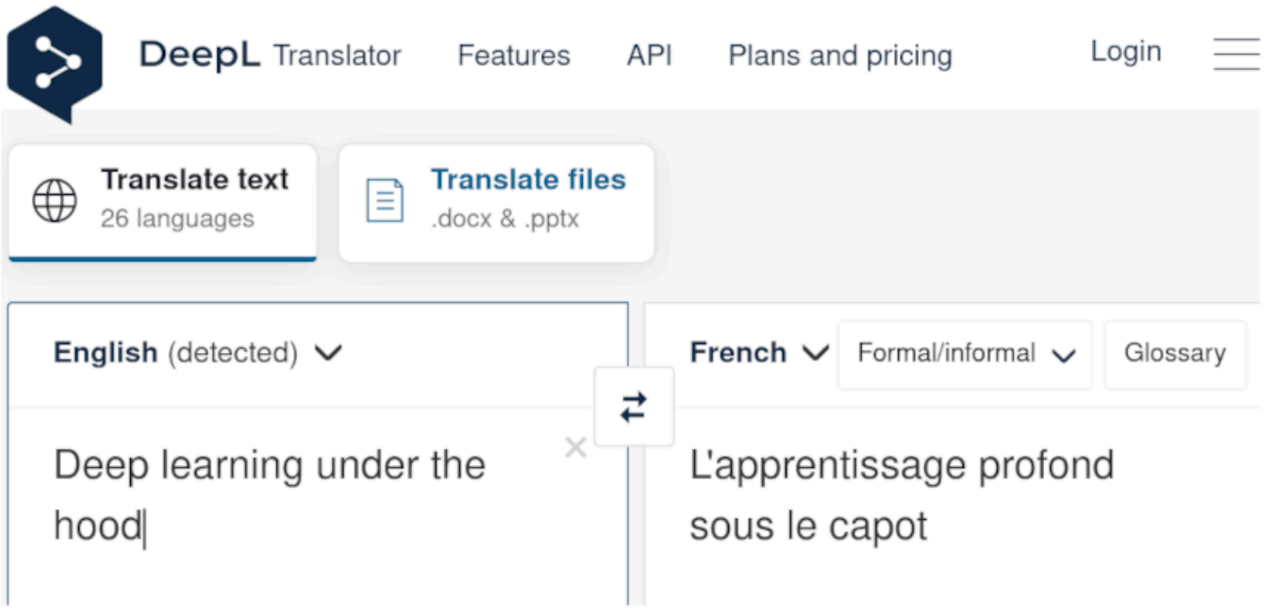


Speech to Text



[Baidu 2014]

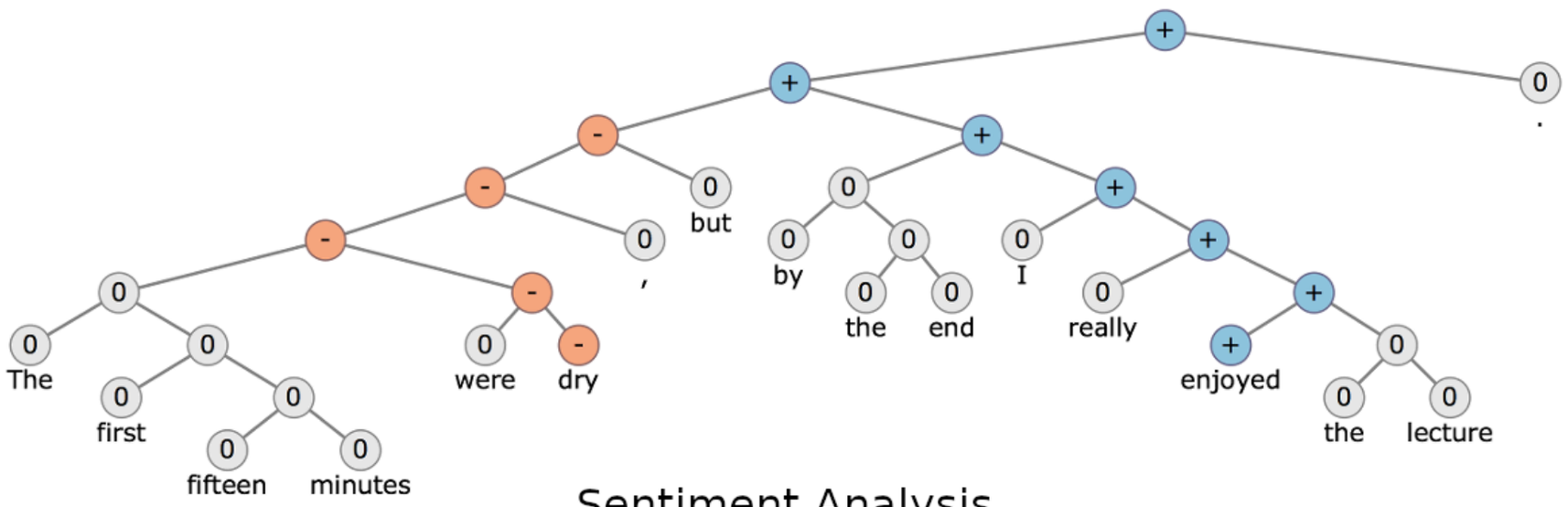
NLP



[DeepL tranlation 2017]

Transformers

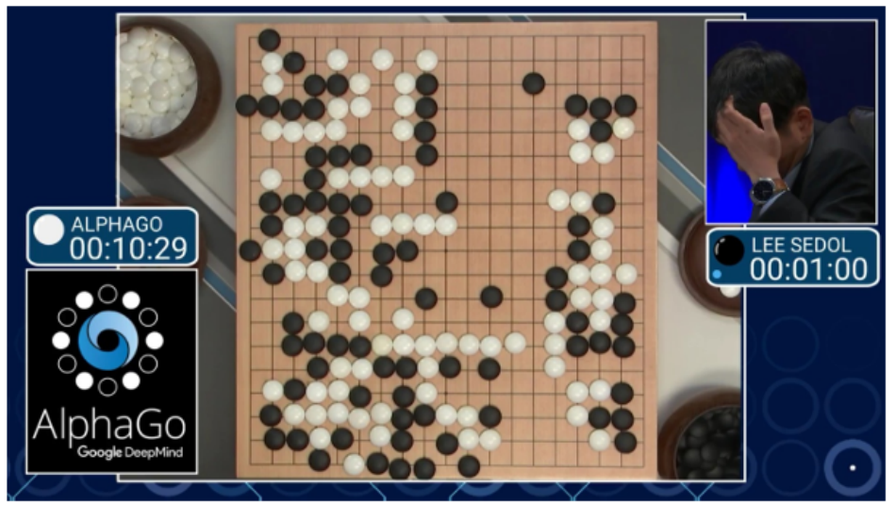
[Vaswani et al. 2017]

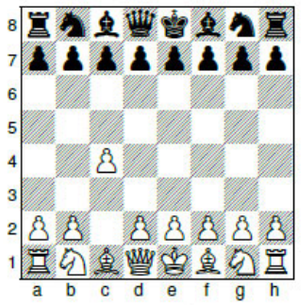


Sentiment Analysis

[Socher 2015]

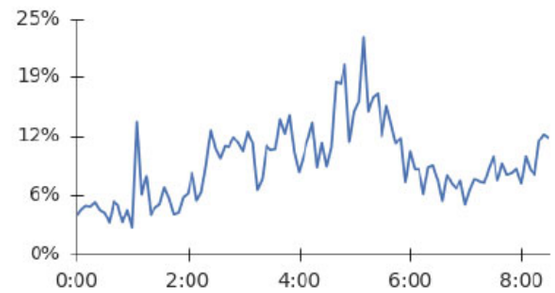
AI in games





w 20/30/0, b 8/40/2

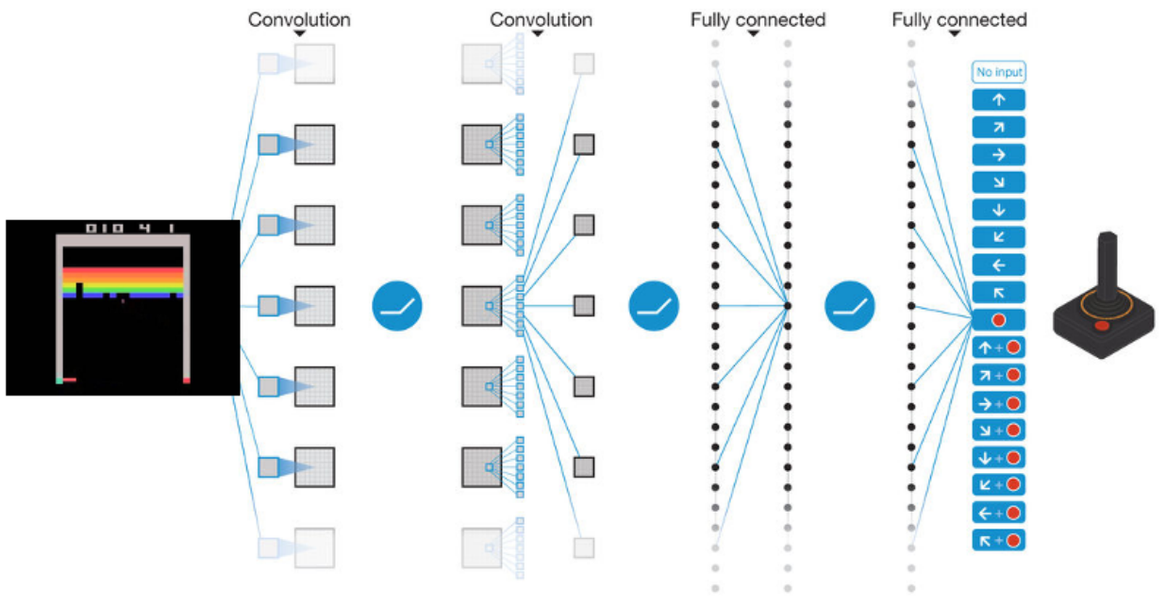
A10: English Opening



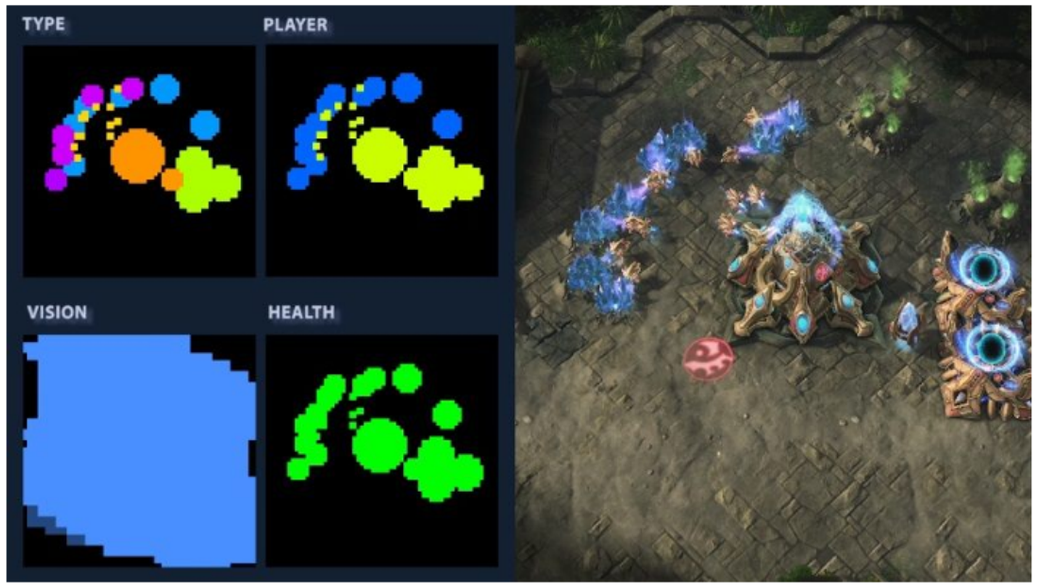
1...e5 g3 d5 cxd5 ♘f6 ♗g2 ♘xd5 ♘f3

[Deepmind AlphaGo / Zero 2017]

AlphaGo/Zero: Monte Carlo Tree Search, Deep Reinforcement Learning, self-play

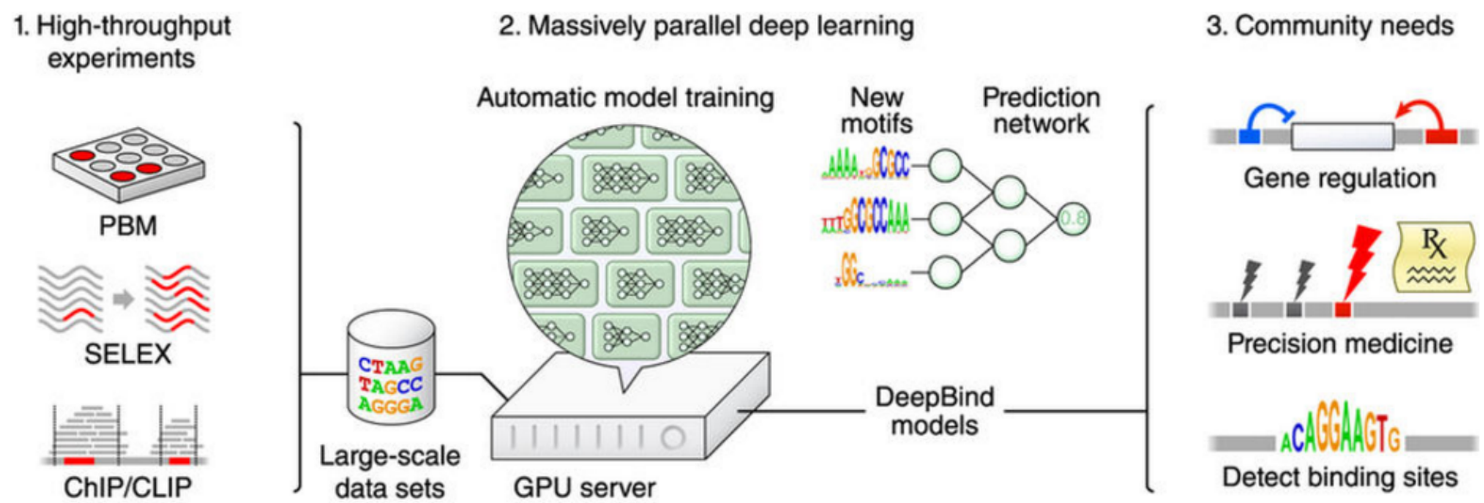


[Atari Games - DeepMind 2016]

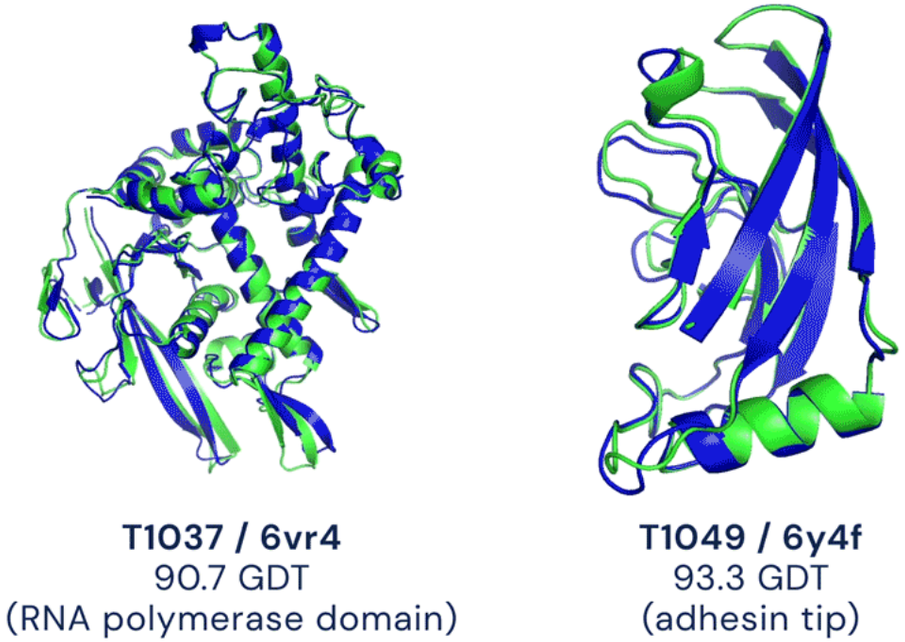


[Starcraft 2 for AI research]

In sciences



[Deep Genomics 2017]

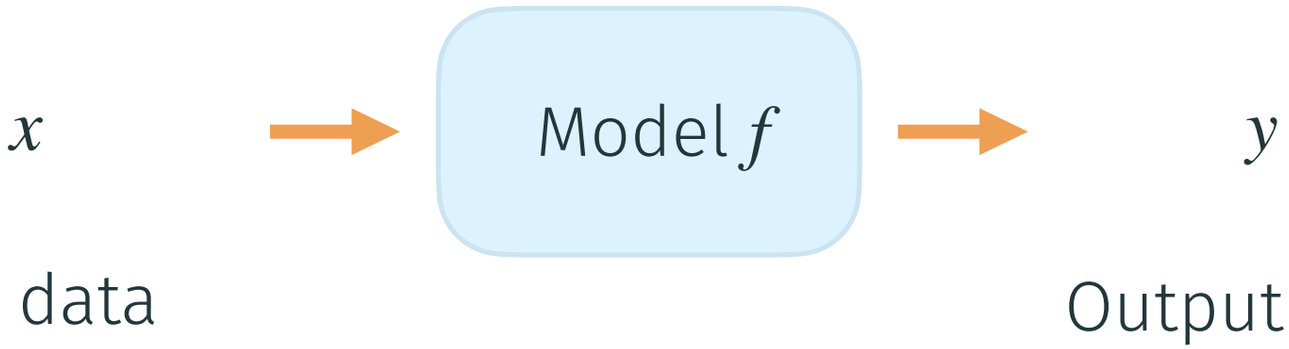


[AlphaFold 2017]

Machine learning, brief recap

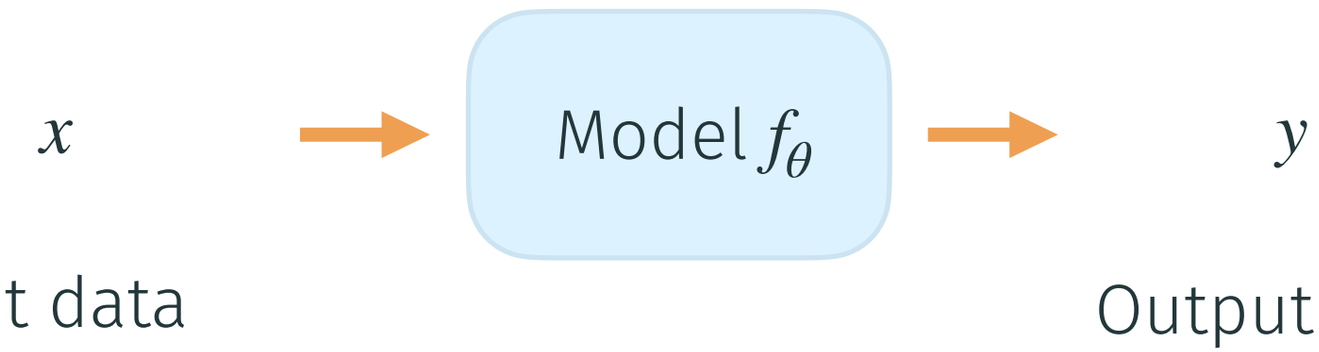
Supervised Learning - Setting

- The goal is to learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ given labeled training samples $(x_i, y_i)_{i=1,\dots,n}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$:



Supervised Learning - Setting

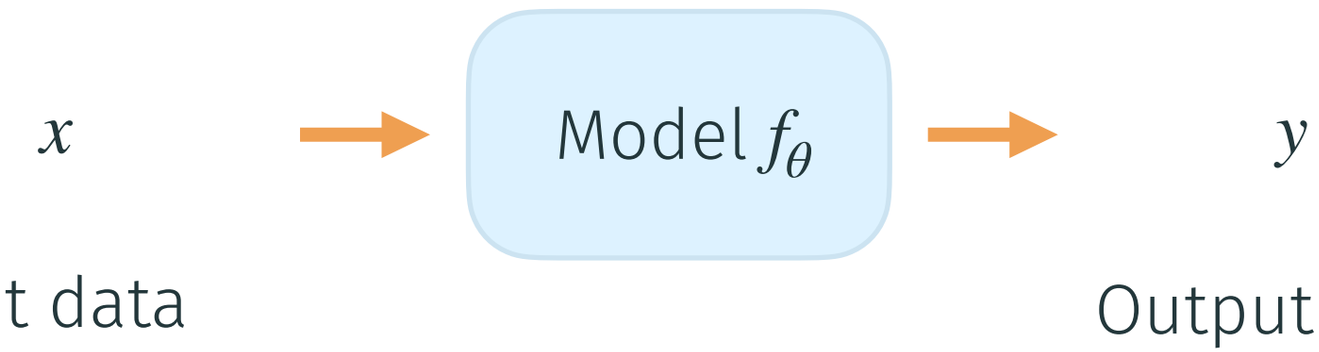
- The goal is to learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ given labeled training samples $(x_i, y_i)_{i=1,\dots,n}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$:



- Parametrize the family by a parameter vector vector $\theta \in \mathbb{R}^N$

Supervised Learning - Setting

- The goal is to learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ given labeled training samples $(x_i, y_i)_{i=1,\dots,n}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$:



- Parametrize the family by a parameter vector vector $\theta \in \mathbb{R}^N$

Objective: Find the optimal parameter θ^*, that gives the best fit $f_{\theta^*}(x) \sim y$

Supervised Learning - Setting

- The "best" fit on samples S is measured with a **loss** function $L(\theta, S)$
- Empirical Risk Minimisation:

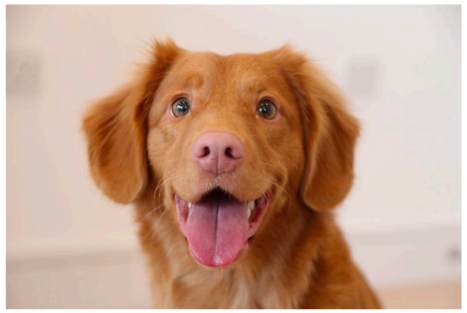
$$L(\theta, S) = \sum_{(x,y) \in S} \ell(f_\theta(x), y)$$

- Example: Mean-Squared-Error loss: $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- The optimal θ^* for samples S:

$$\theta^* = \text{argmin}_\theta \, L(\theta, S)$$

Quality of Fit

Plane



Dog

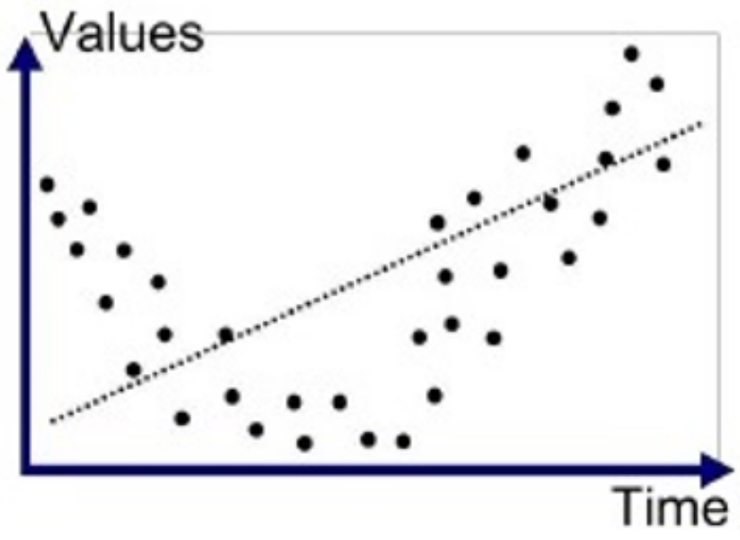
Table

Cat

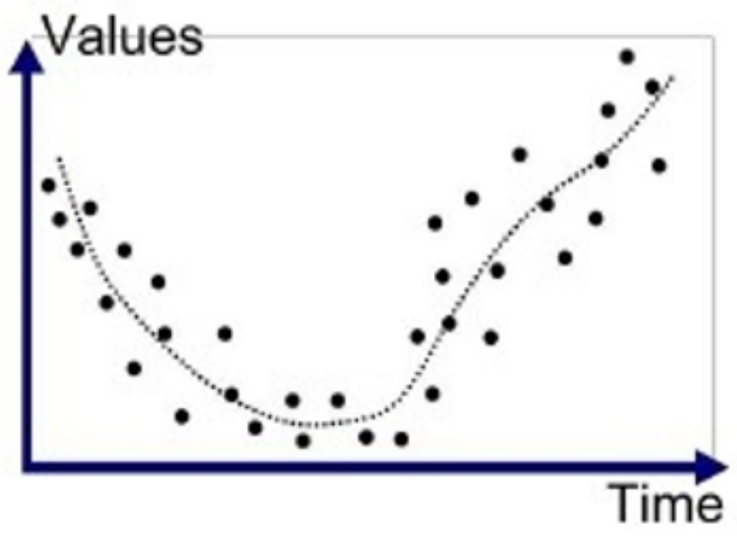
Plane!

Slide from Xavier Alameda Pineda

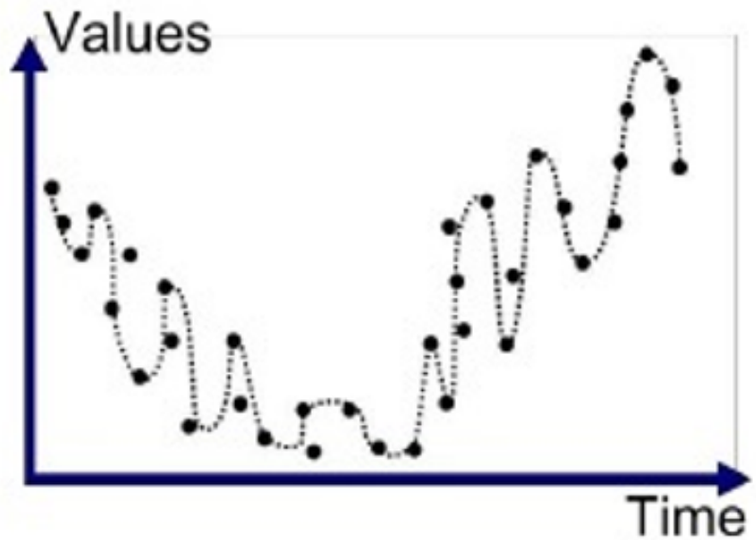
Quality of Fit



Underfitted



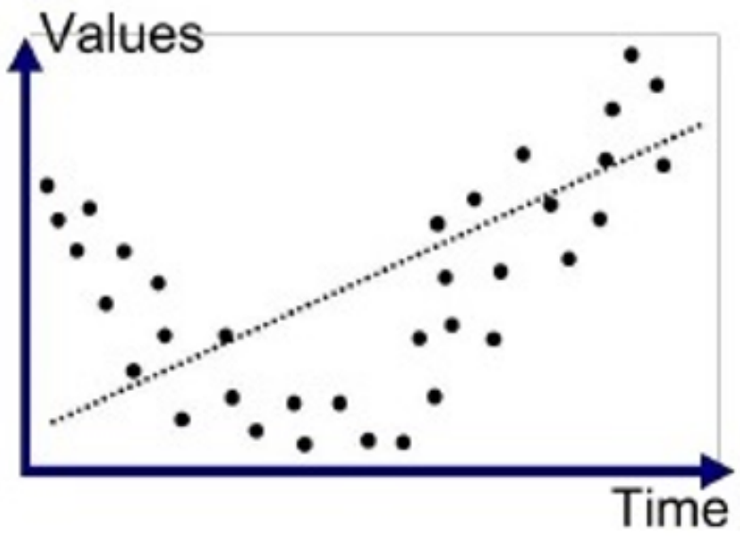
Good Fit/Robust



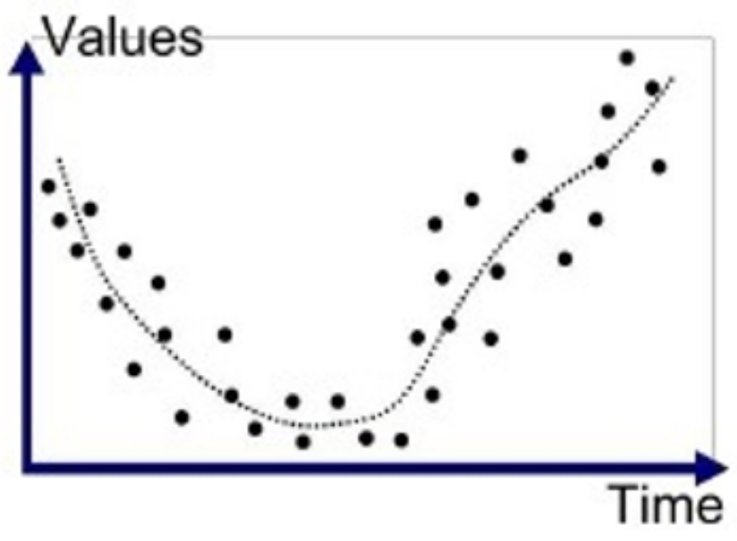
Overfitted

Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

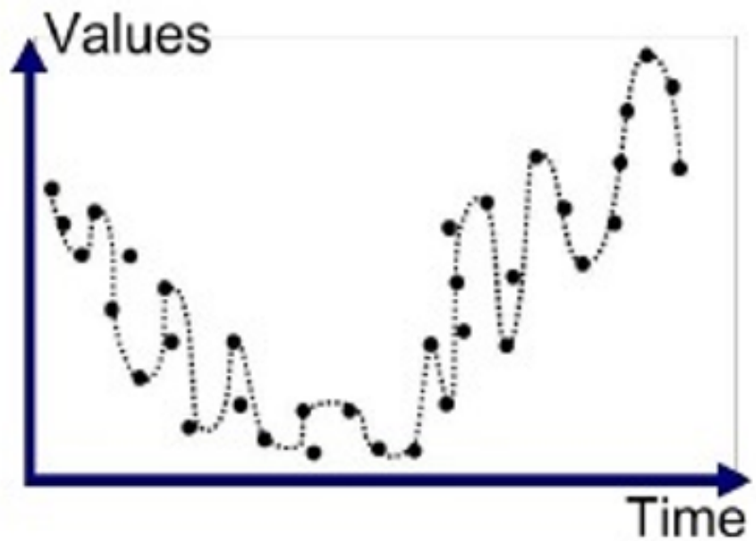
Quality of Fit



Underfitted



Good Fit/Robust



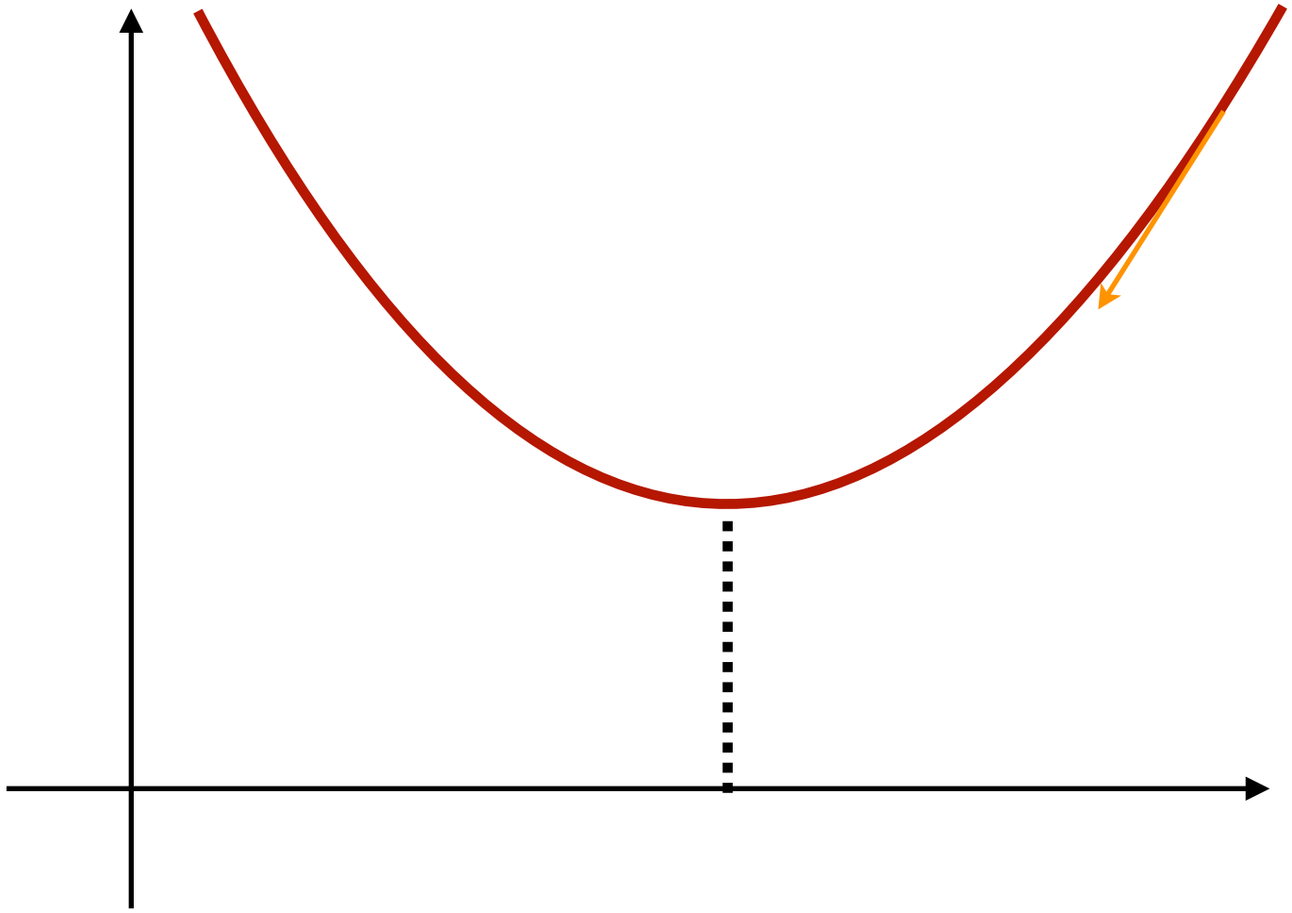
Overfitted

Regularisation: Add a term that penalises variance of f_θ

$$L(\theta, S) = \sum_{(x,y)\in S} \ell(f_\theta(x), y) + \Omega(f_\theta)$$

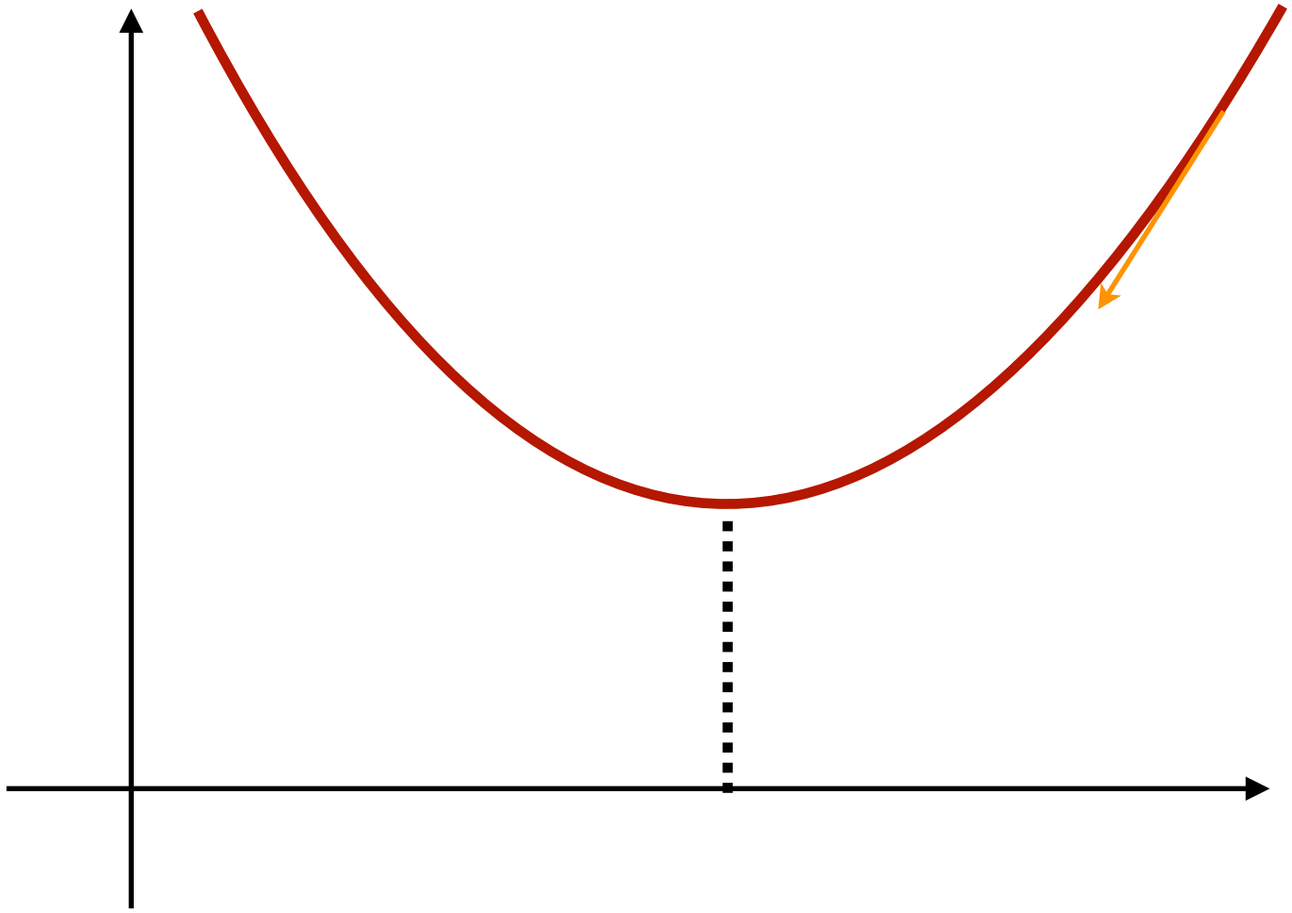
What is Deep Learning ?

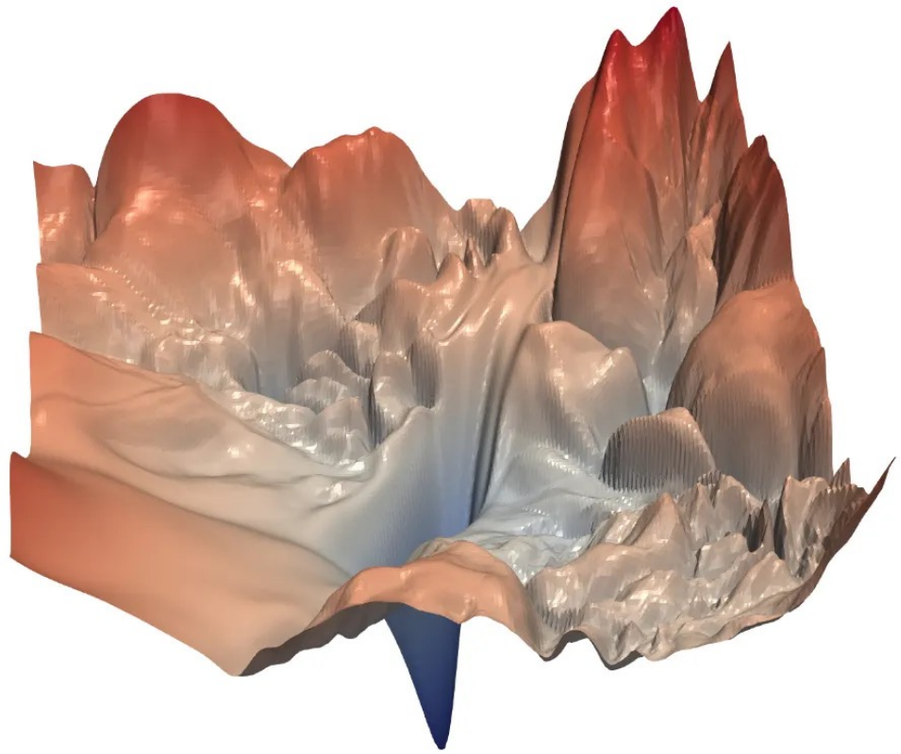
- How do we optimise the loss L?



What is Deep Learning ?

- How do we optimise the loss L?



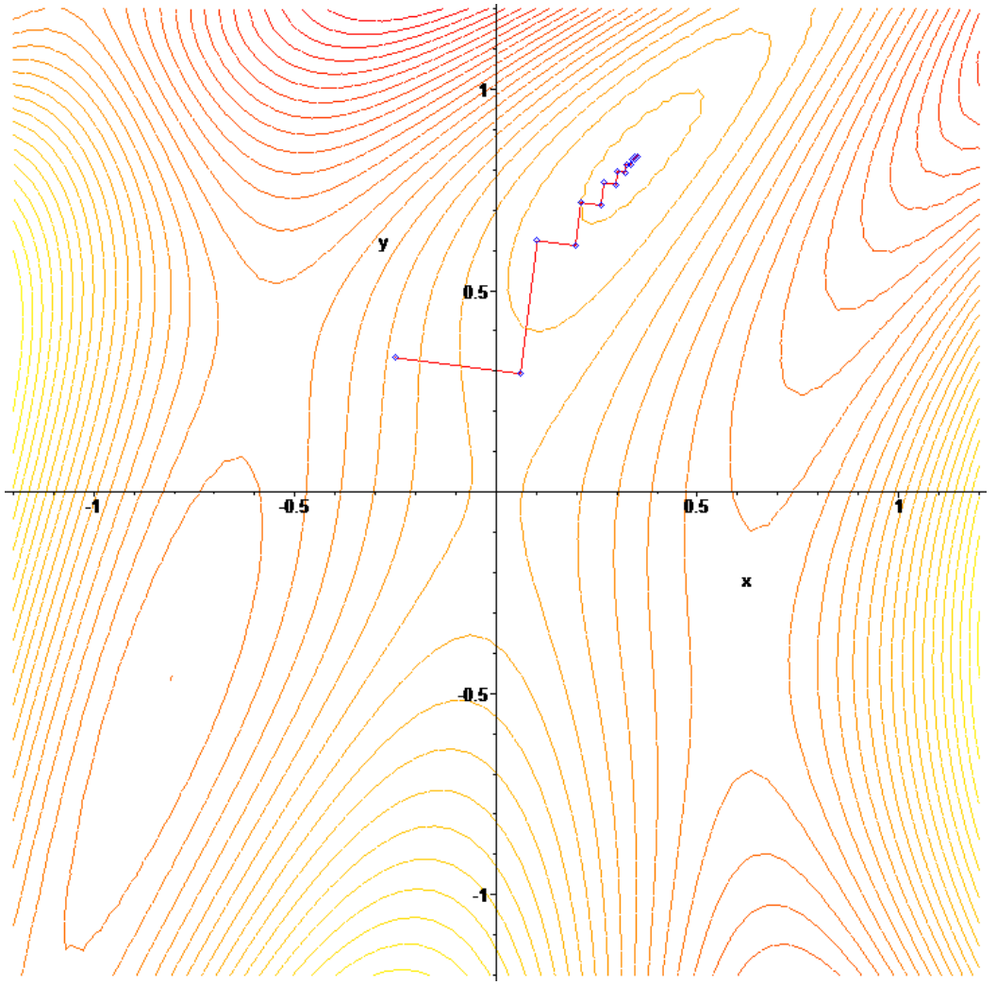


Loss landscape can be horrendously complex!

Gradient Descent

- Initialize θ randomly
- For E epochs perform:
  - Compute gradients: $\Delta = \nabla_\theta L(\theta, S)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$

η is called the *learning rate*



By user:Joris Gillis - Created with Maple 10, Public Domain, https://commons.wikimedia.org/w/index.php?curid=521419

Stochastic Gradient Descent

- Too expensive to compute the whole gradient

$$\nabla_\theta L(\theta, S) = \sum_{s \in S} \nabla_\theta L(\theta, s)$$

- For modern networks S can have billions of samples!

Stochastic Gradient Descent

- Too expensive to compute the whole gradient

$$\nabla_\theta L(\theta, S) = \sum_{s \in S} \nabla_\theta L(\theta, s)$$

- For modern networks S can have billions of samples!

- Optimal batch-size: trade-off between speed and generalization performance

Stochastic Gradient Descent

- Initialize θ randomly
- For E epochs perform:
  - Randomly select a small batch of samples $B \subset S$
  - Compute gradients: $\Delta = \nabla_\theta L(\theta, B)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$

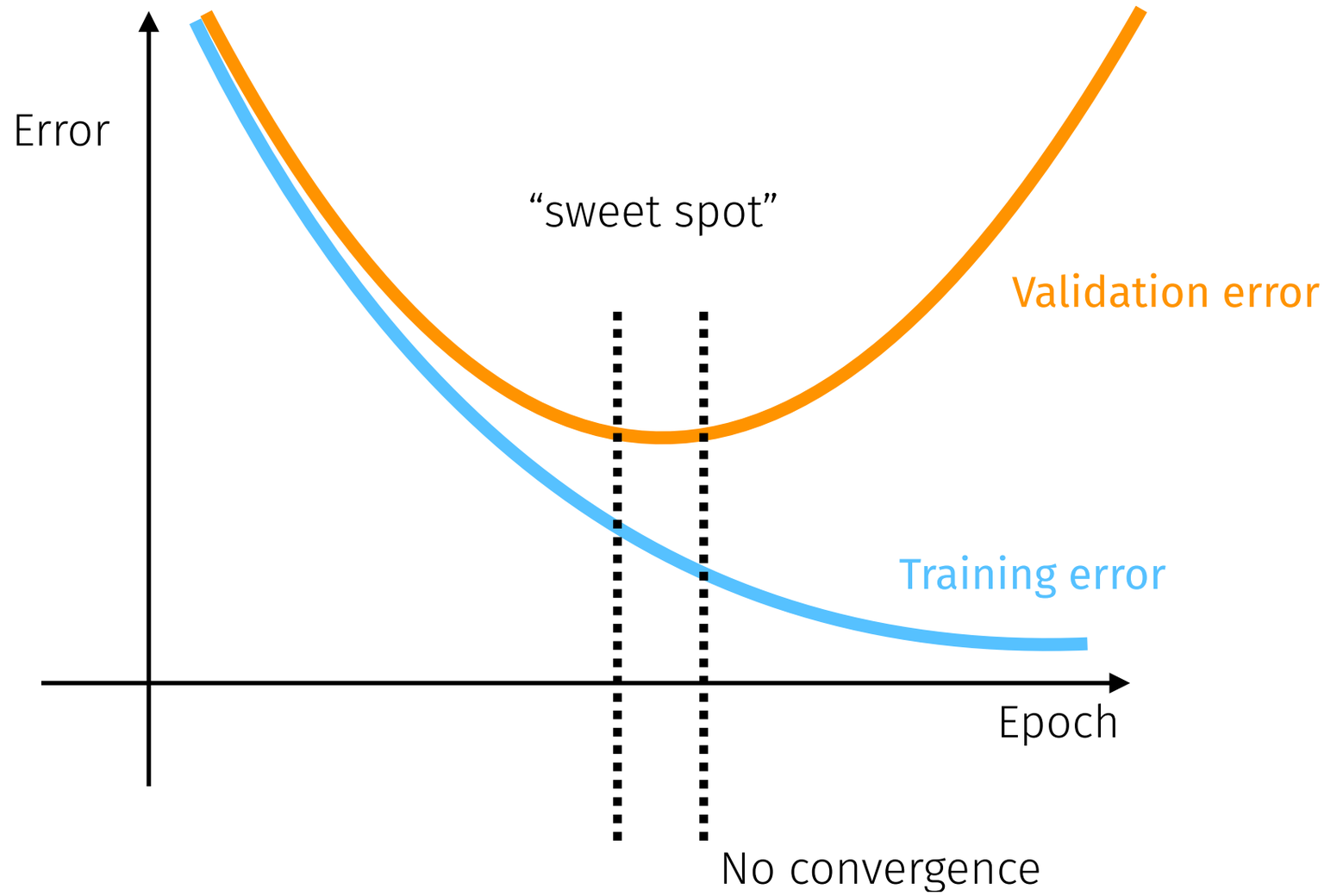
η *learning rate*

- Core idea: good direction on average:

$$\mathbb{E}_{B \subset S}[\nabla_\theta L(\theta, B)] = \nabla_\theta L(\theta, S)$$

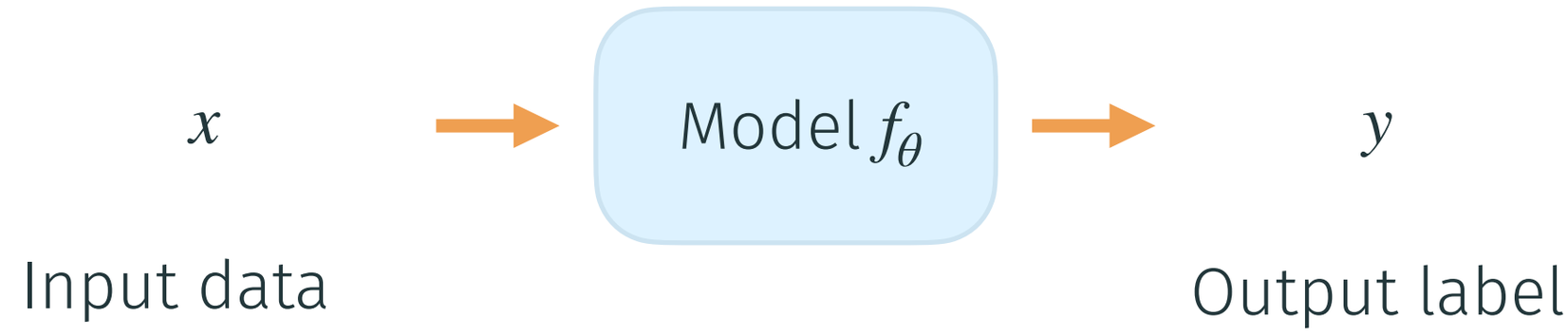
Early Stopping

- Stop when no progress on validation loss after *patience* epochs
- Output model with best validation error
- Reduce risk of over-fitting



Supervised Learning - Setting

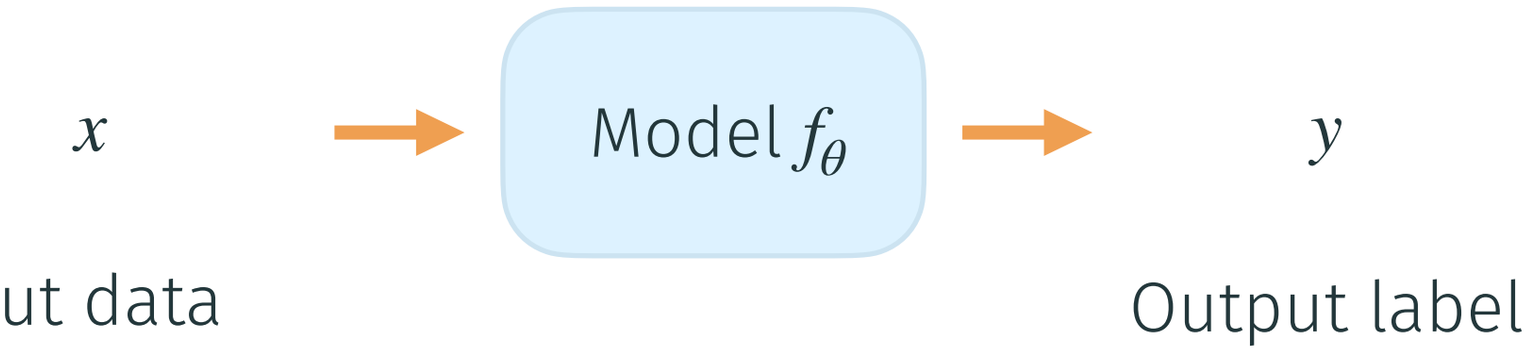
- Reminder: our model f_θ:



- Powerful (able to model complex relations)
- Scalable. Fast at inference

Supervised Learning - Setting

- Reminder: our model f_θ:

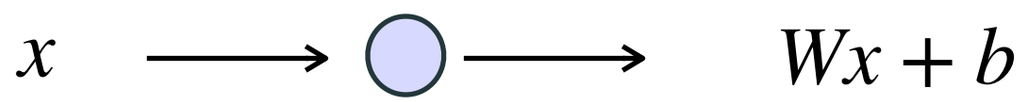


- Powerful (able to model complex relations)
- Scalable. Fast at inference
- **Easily computable gradient** $\nabla_\theta f_\theta$

How, Opening the black box

Intuition, Single Neuron

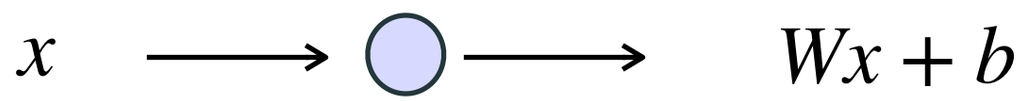
- Reminder: we want a function that is **scalable**, **powerful**, easy to **optimise**
- **Composition** of elementary functions:
  - Multi-variable linear, $x \rightarrow Wx + b$



A single "*neuron*"

Intuition, Single Neuron

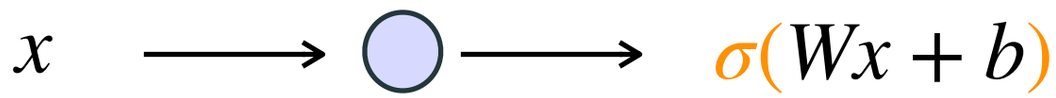
- Reminder: we want a function that is **scalable**, **powerful**, easy to **optimise**
- **Composition** of elementary functions:
  - Multi-variable linear, $x \rightarrow Wx + b$



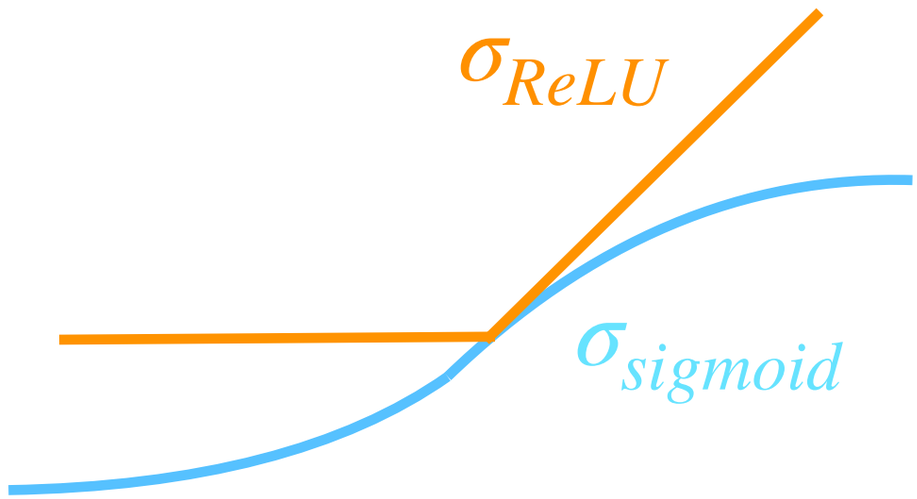
A single "*neuron*"

Intuition, Single Neuron

- Reminder: we want a function that is **scalable**, **powerful**, easy to **optimise**
- **Composition** of elementary functions:
  - Multi-variable linear, $x \rightarrow Wx + b$
  - Elementwise continuous functions (non-linearity) $x \rightarrow \sigma(x)$



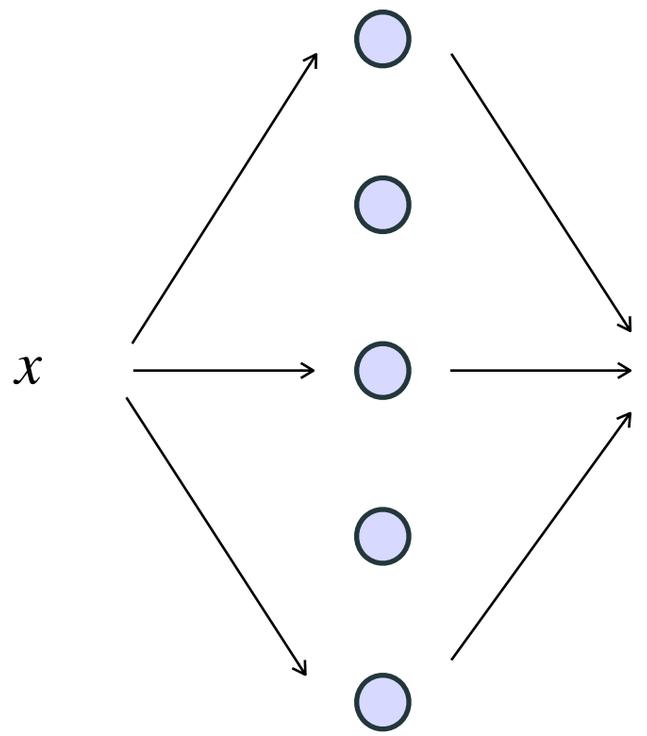
Non-linearities:



Universal Approximation Theorem

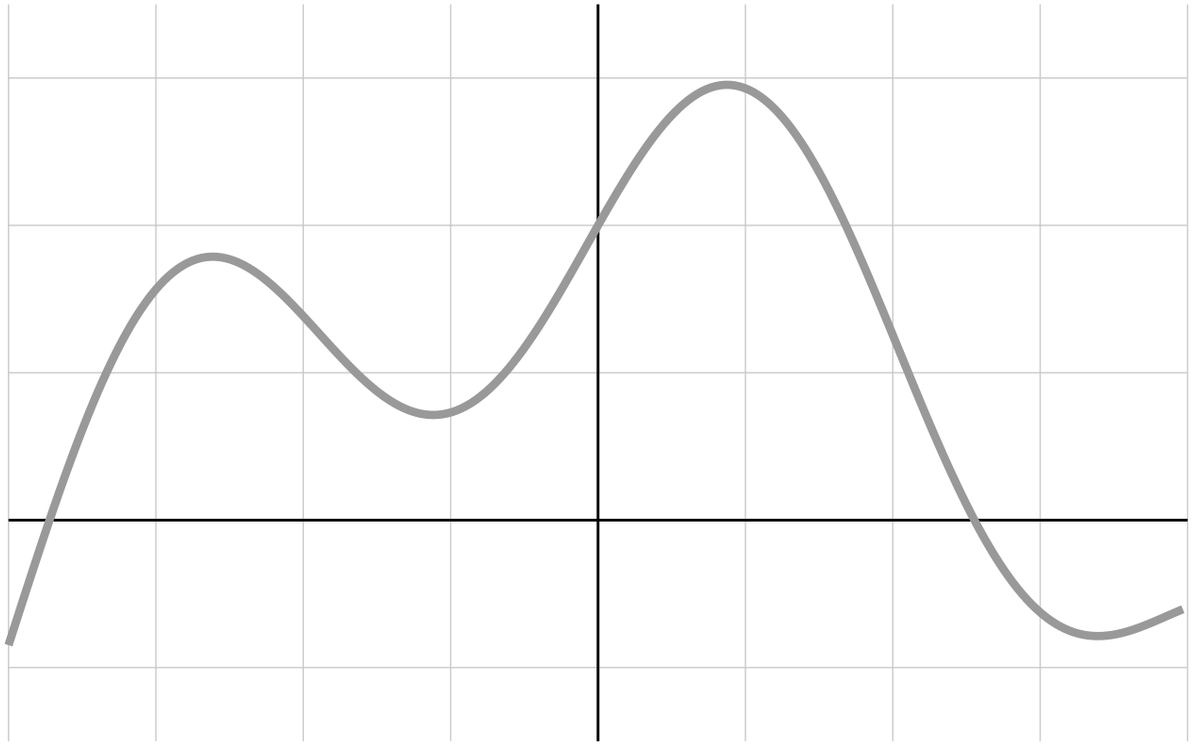
- Neural *network:* stacked neurons:

- With enough neurons, one can approximate any continuous function!



$$\sum_i \alpha_i \sigma(W_i x + b_i)$$

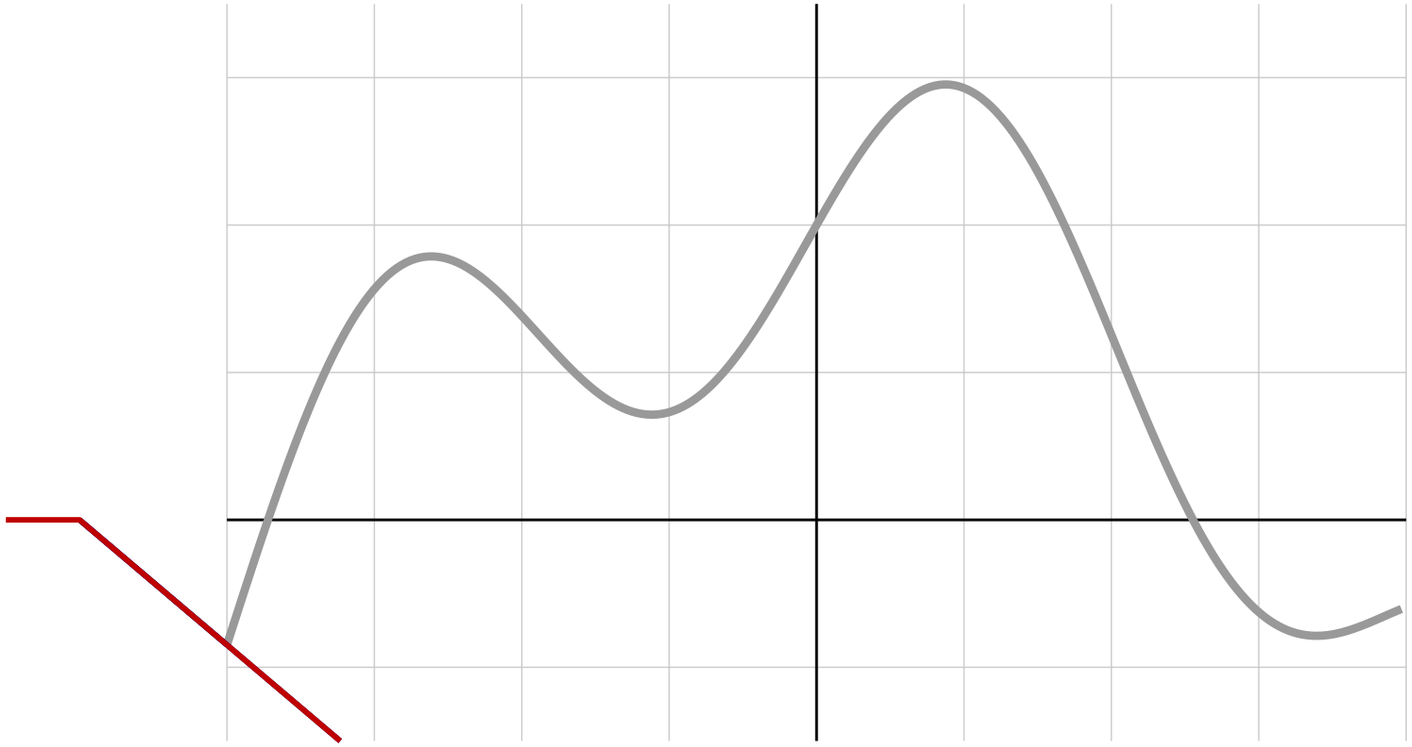
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

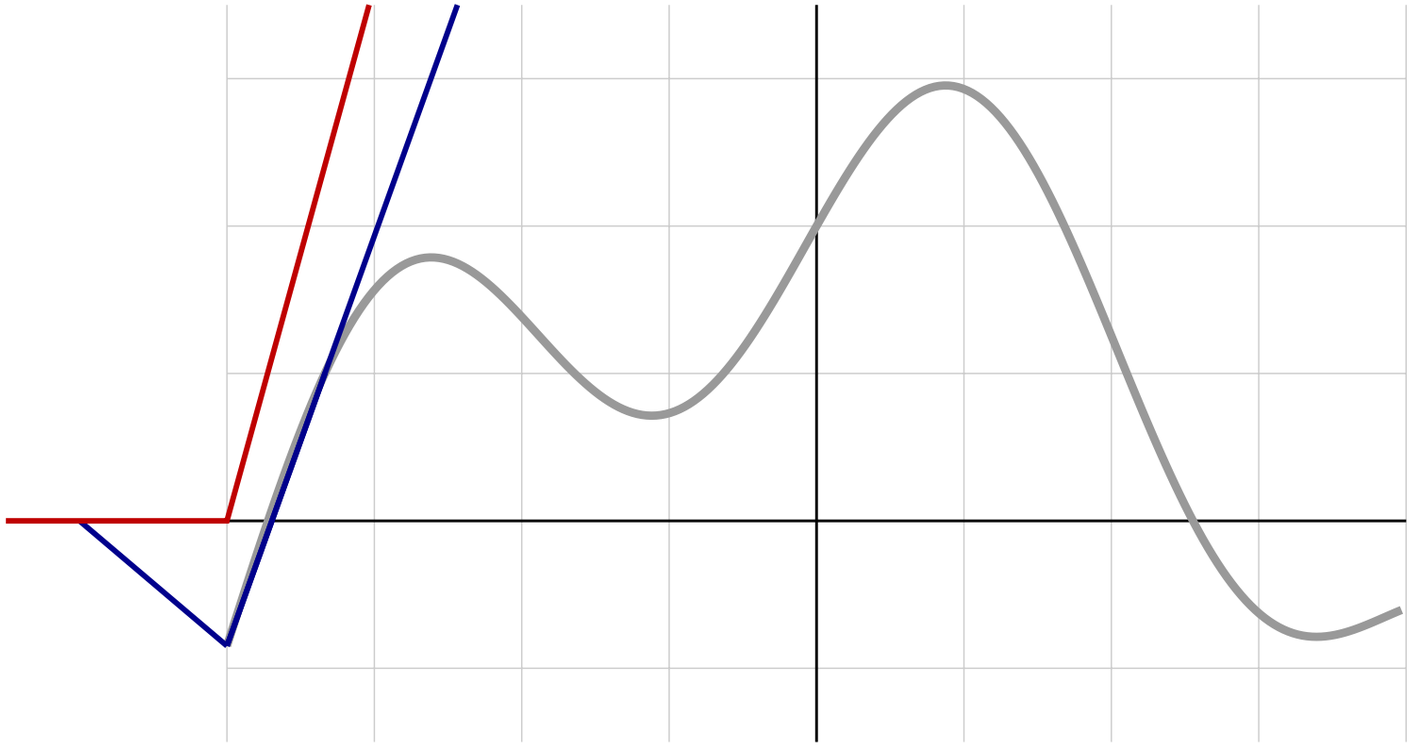
$$f(x) = \sigma(w_1 x + b_1)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

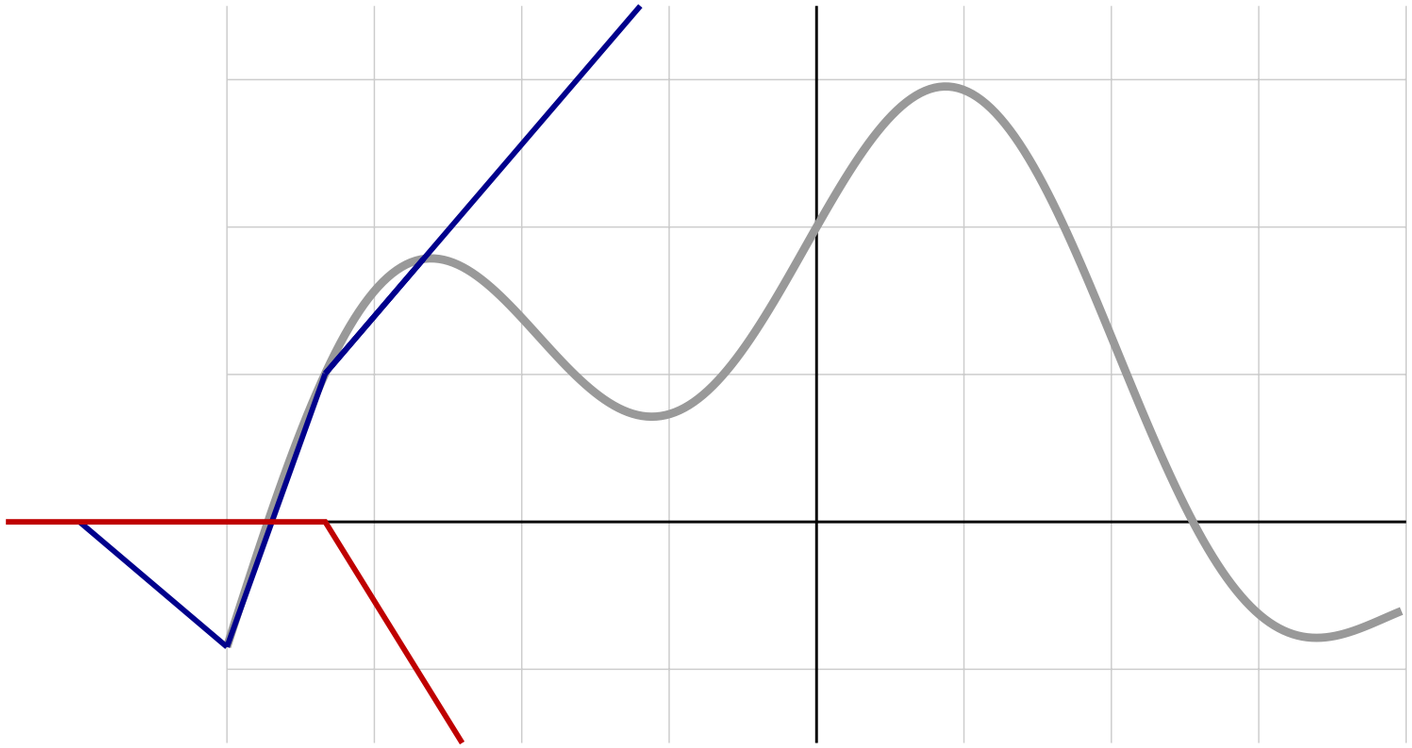
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

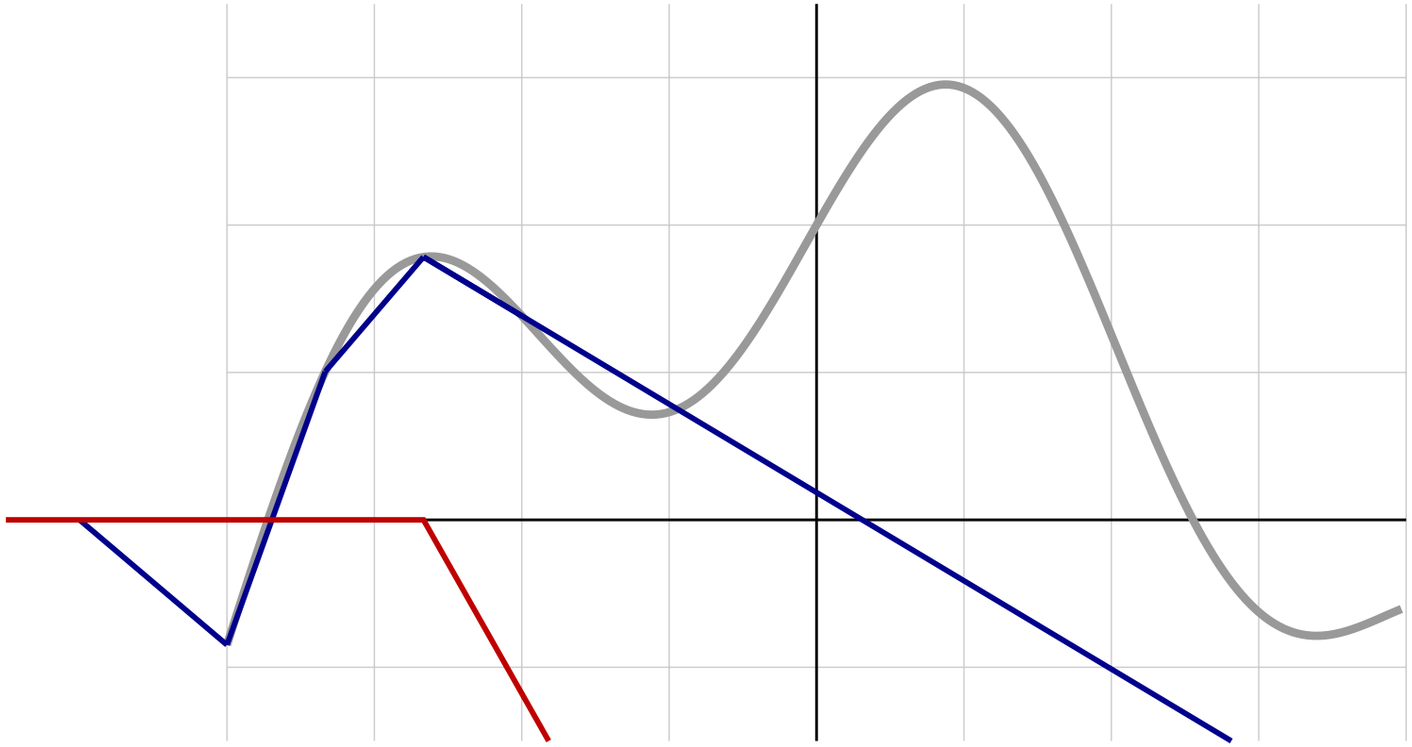
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

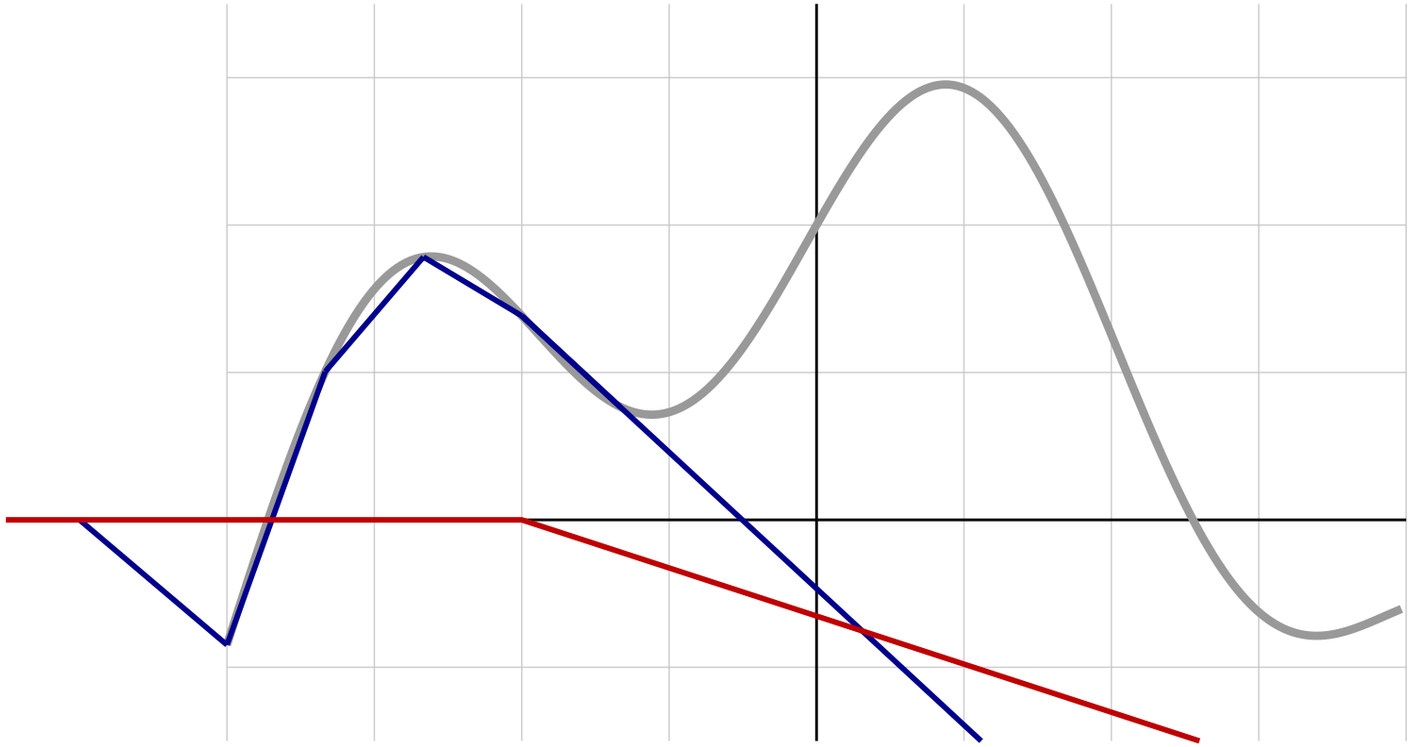
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

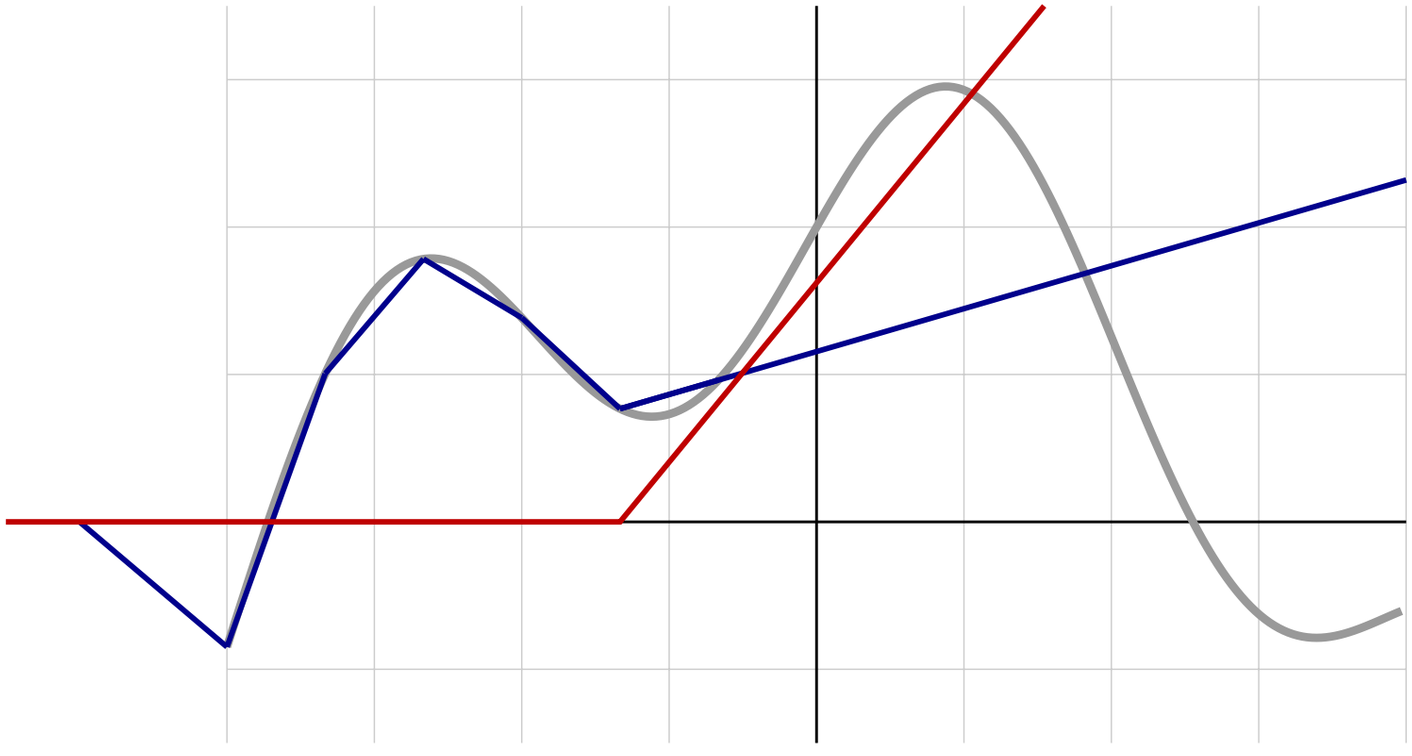
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

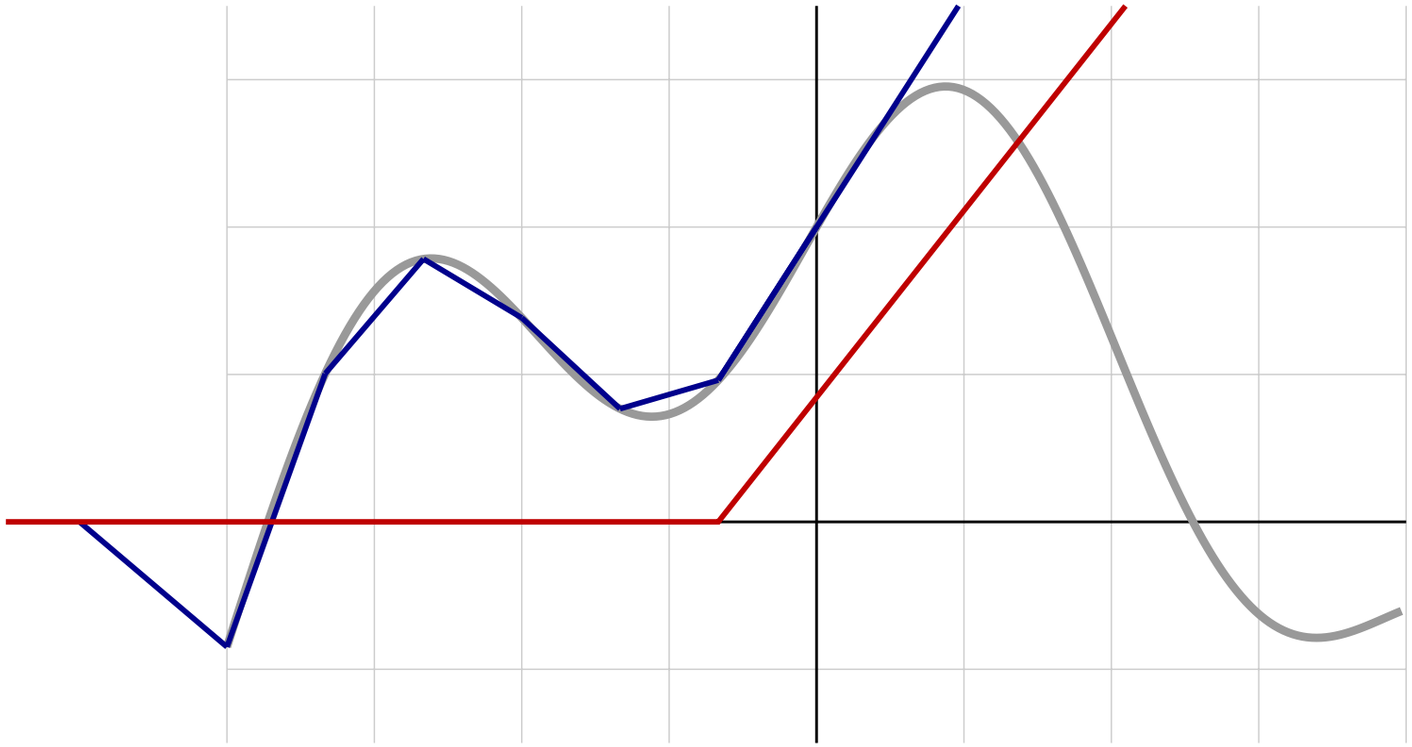
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

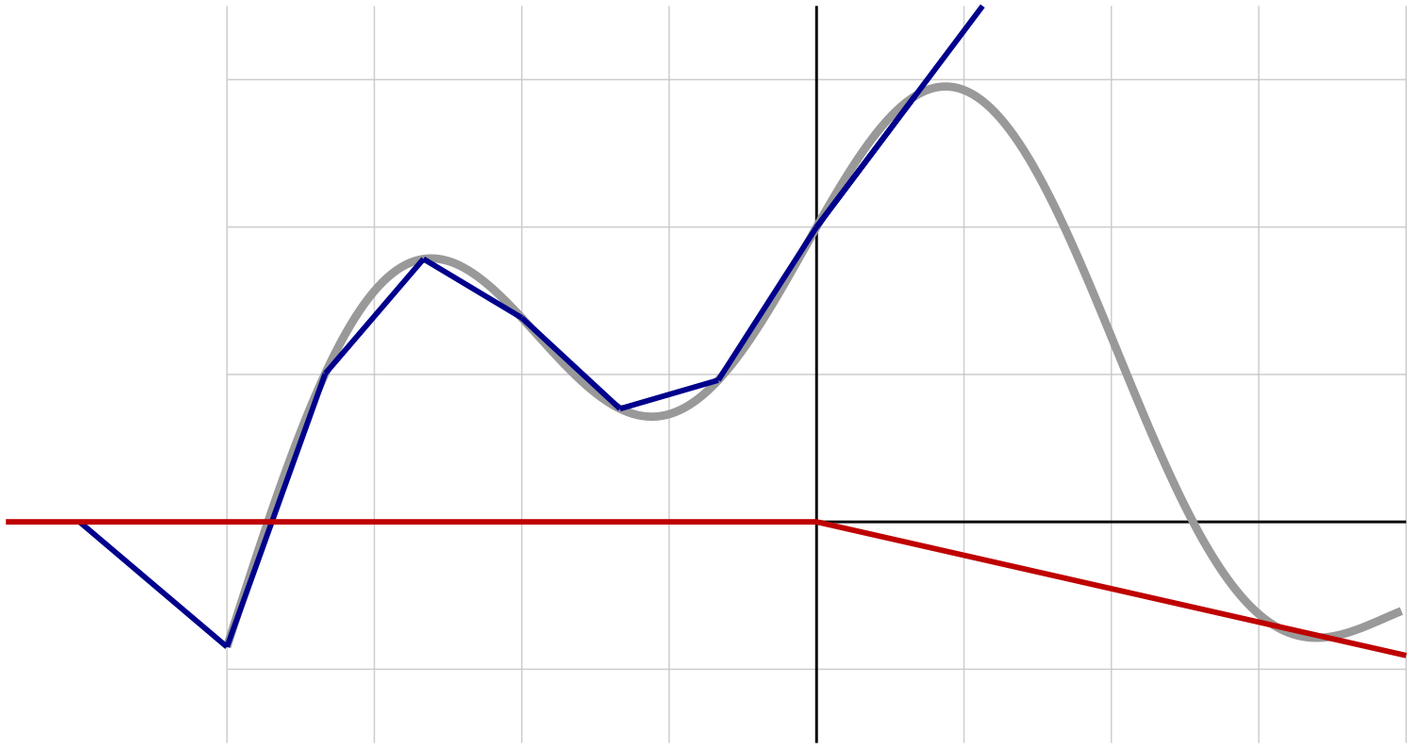
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

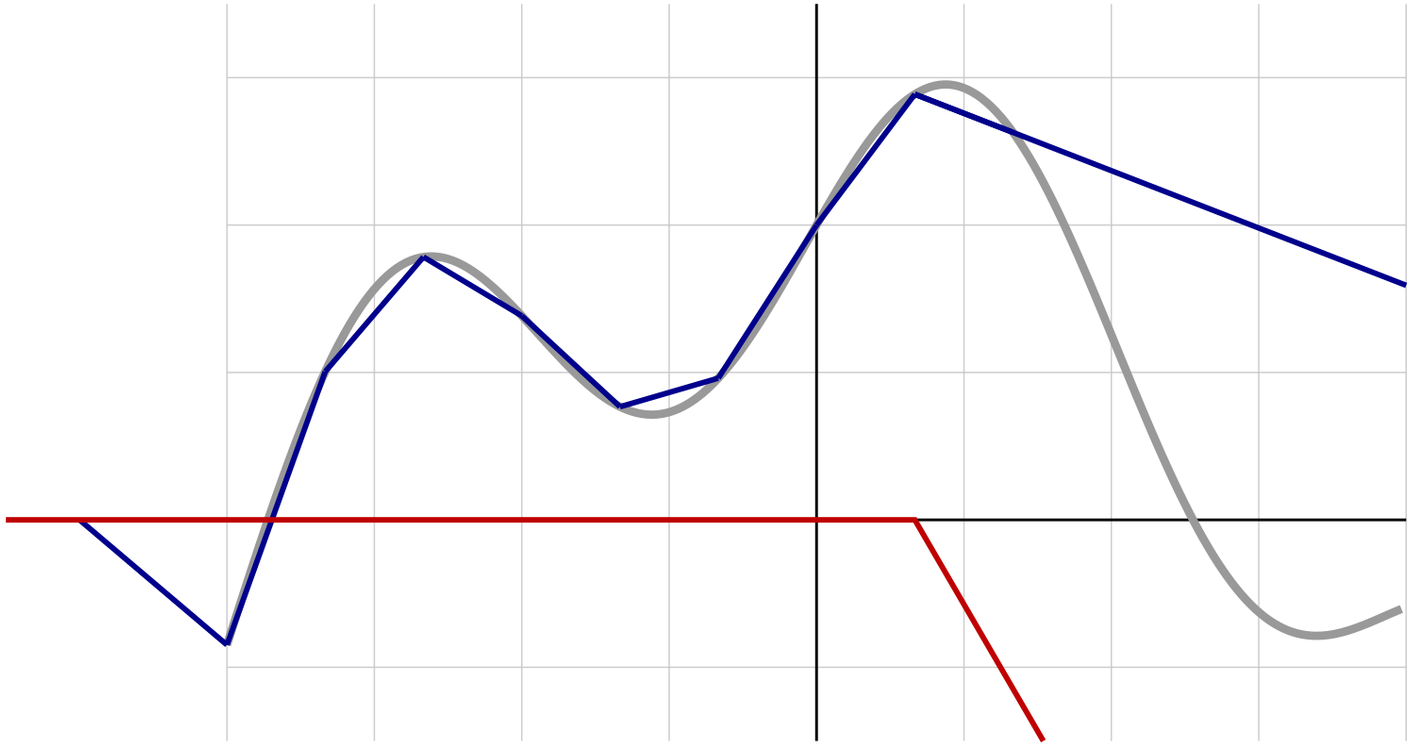
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

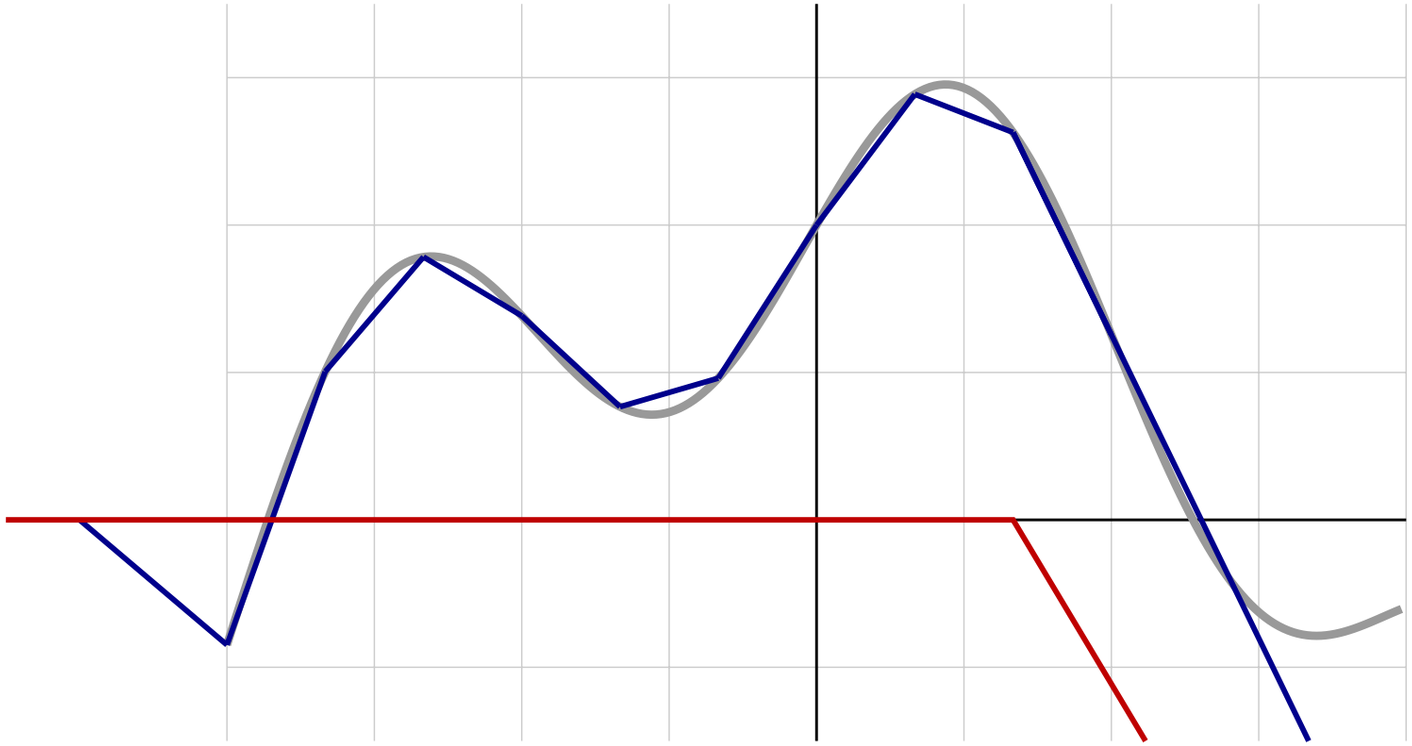
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

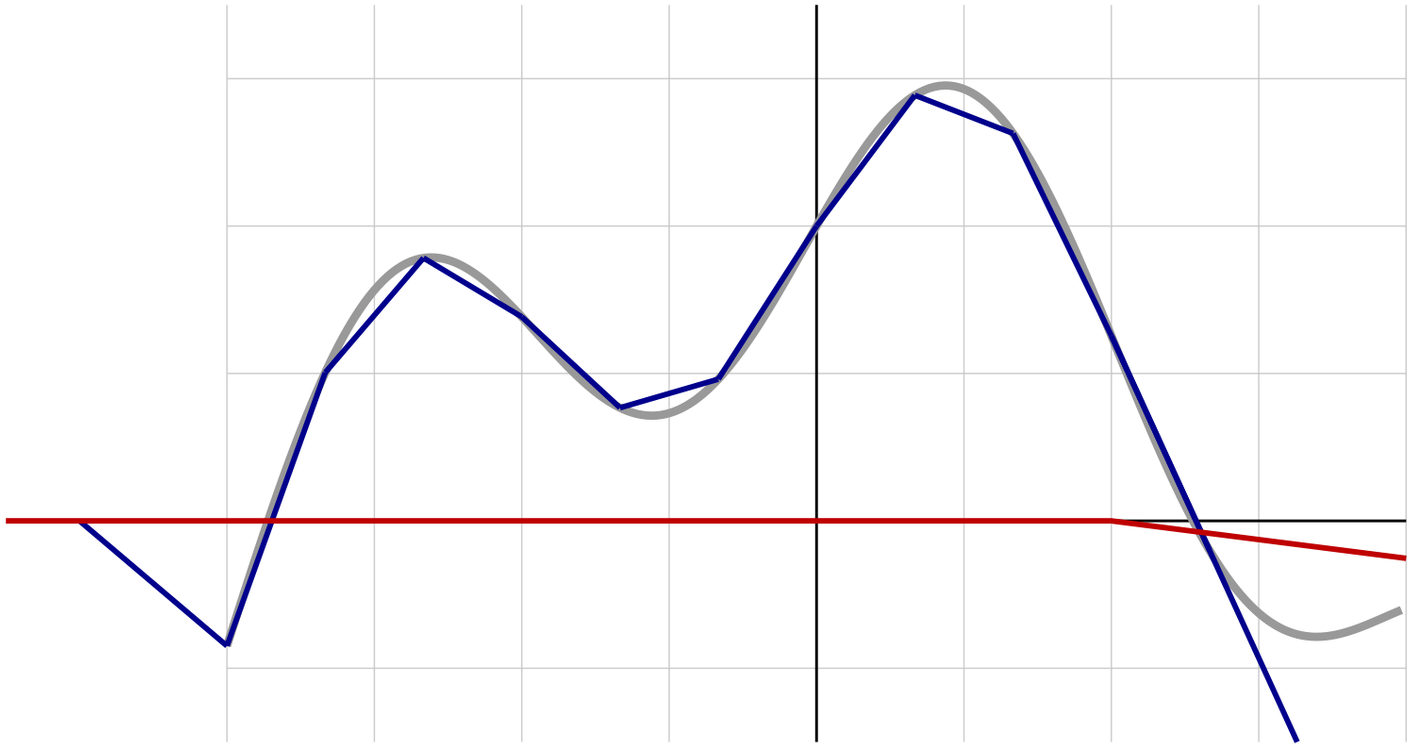
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

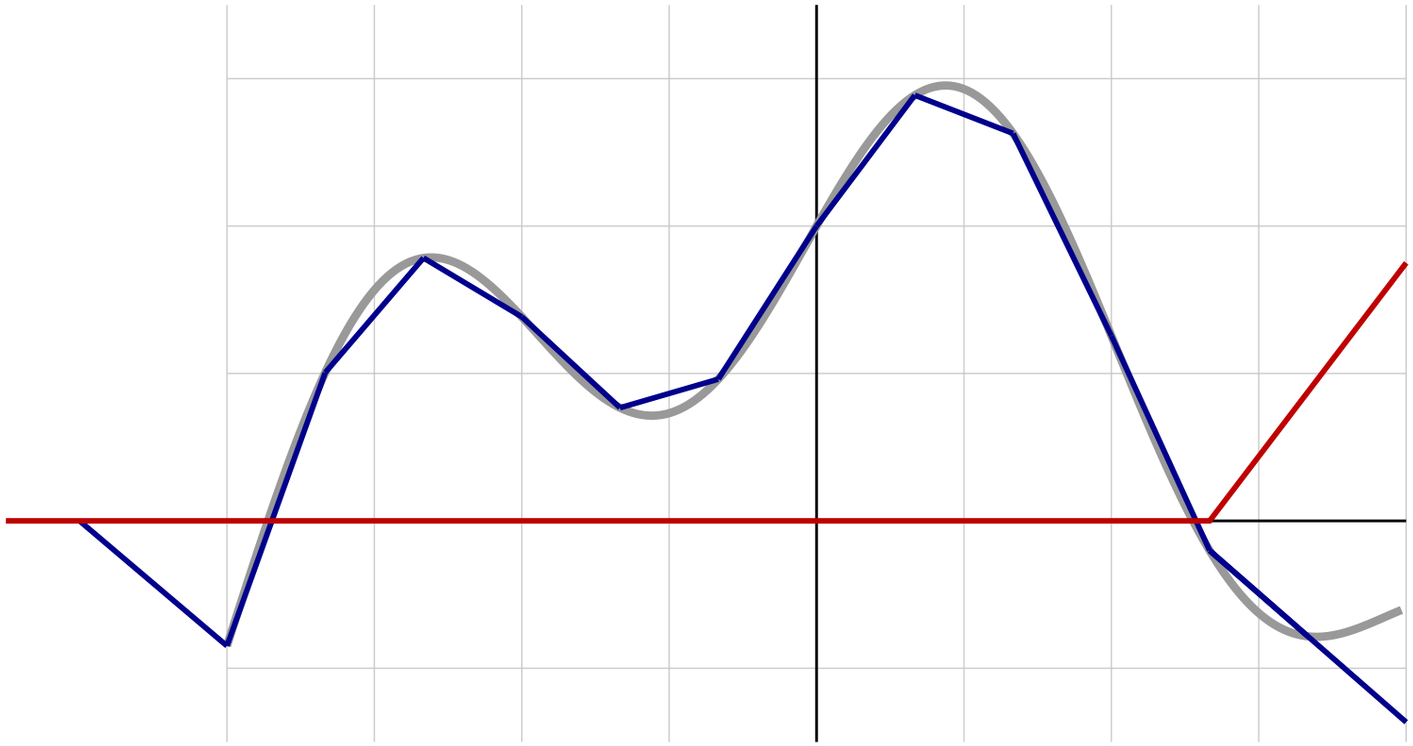
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

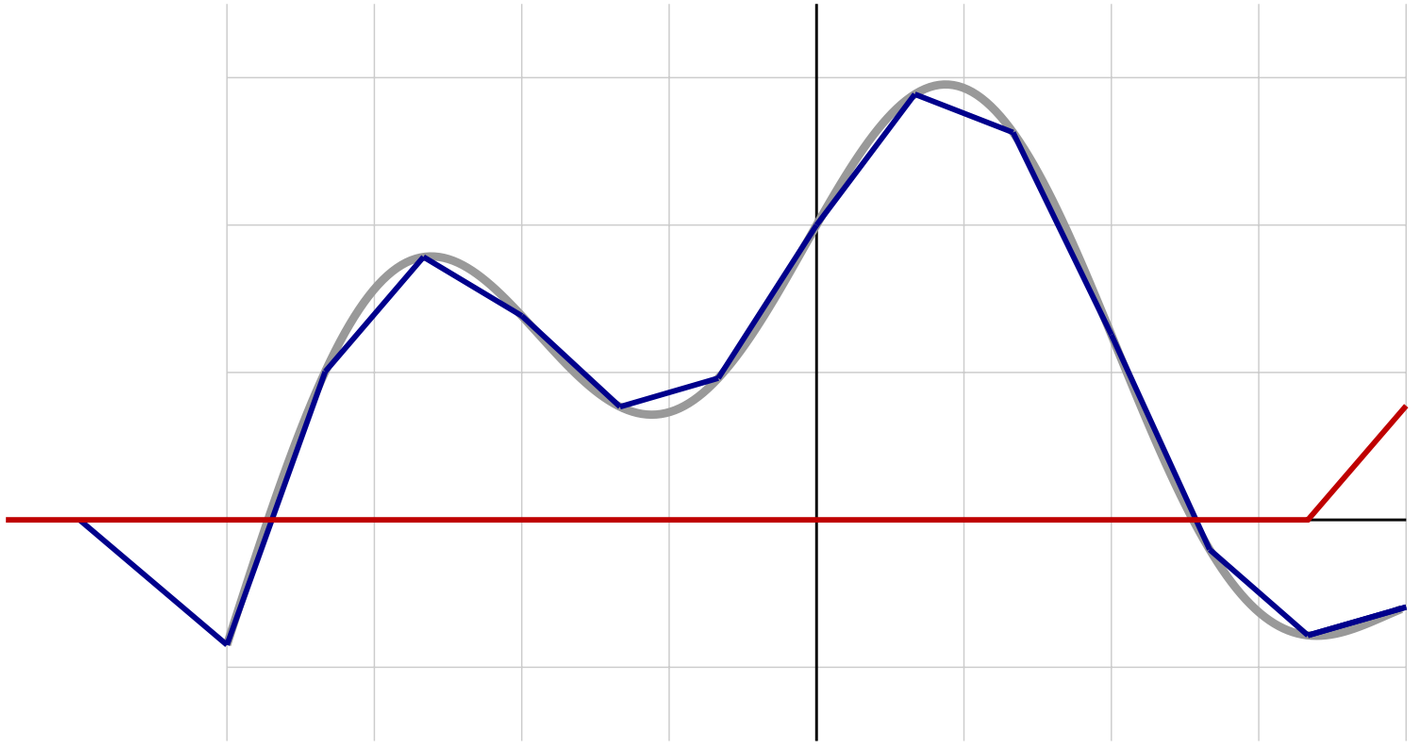
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

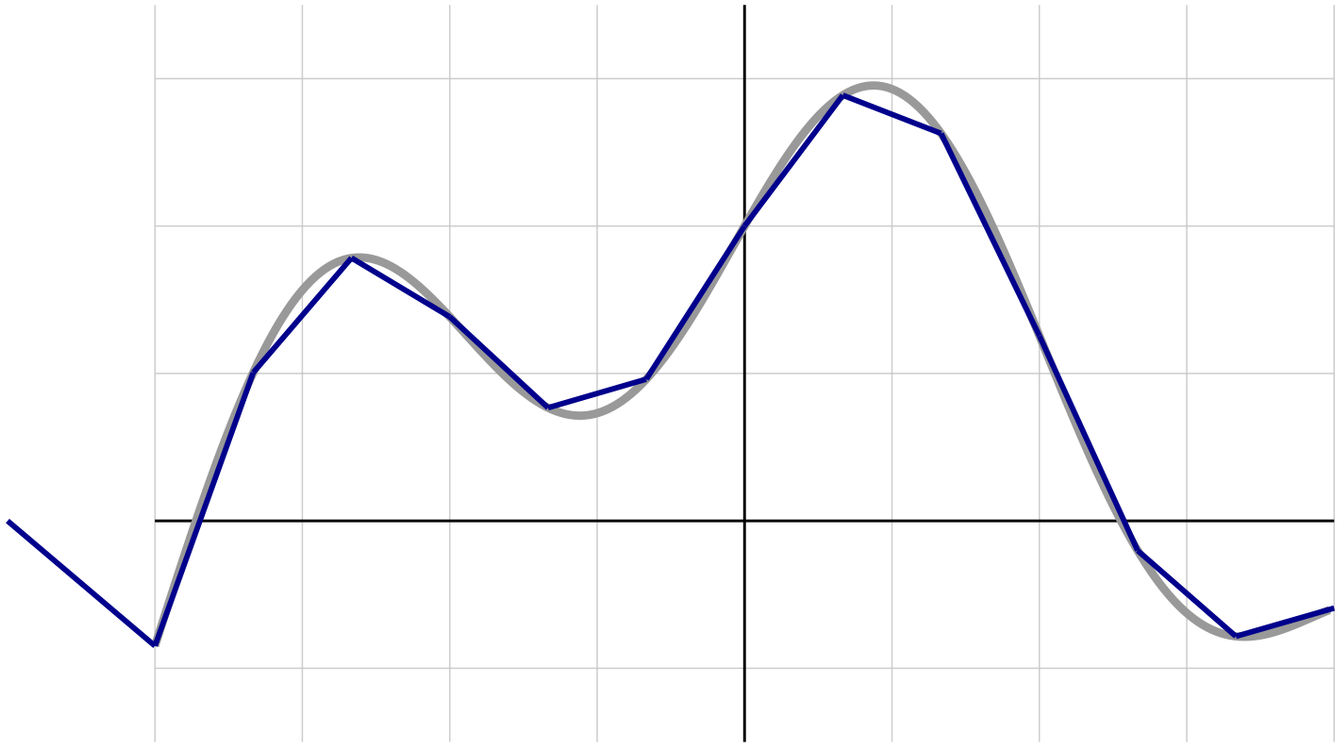
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

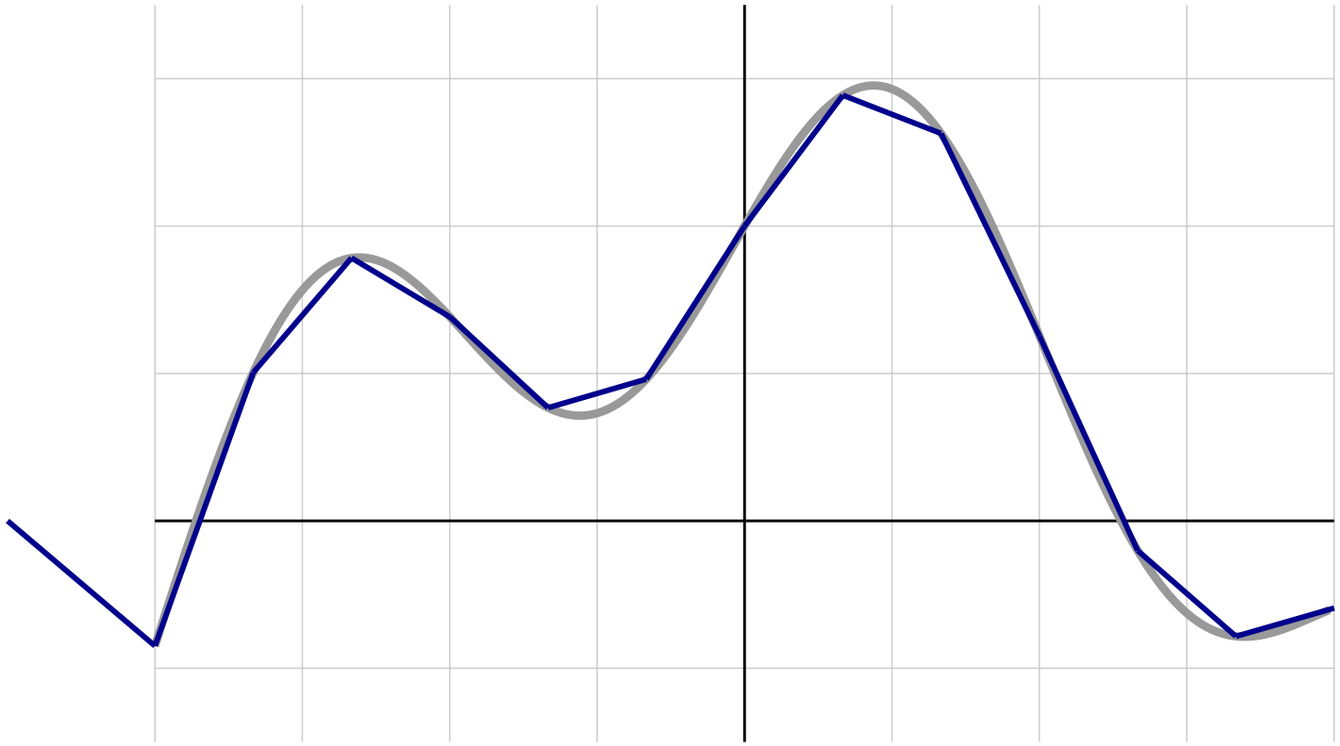
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



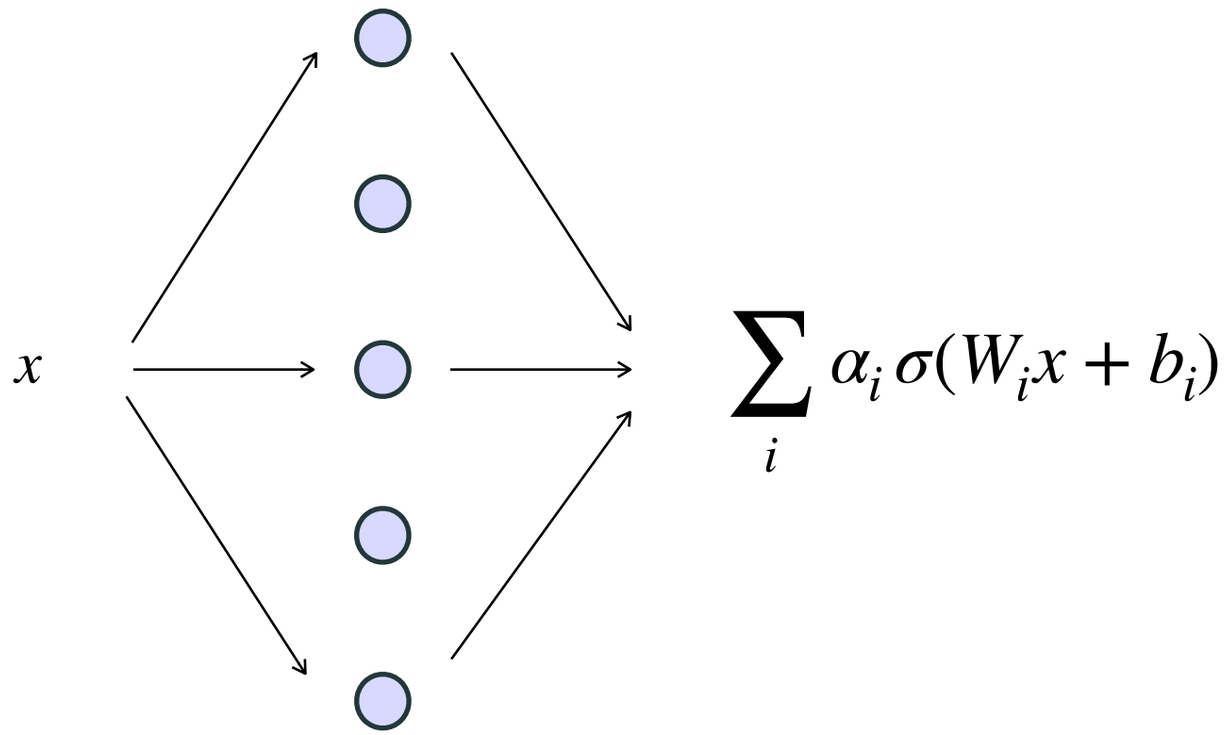
This is true for other activation functions under mild assumptions.

Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

Universal Approximation Theorem

- Neural *network:* stacked neurons:

- With enough neurons, one can approximate any continuous function!



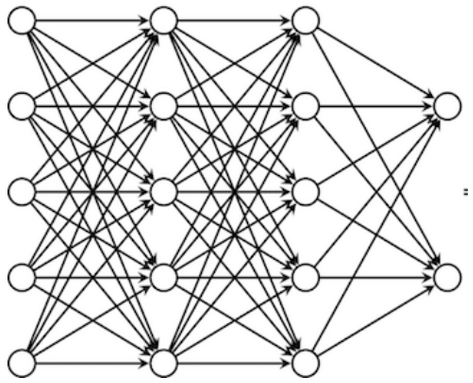
- However, expressive power does not scale well with number of neurons

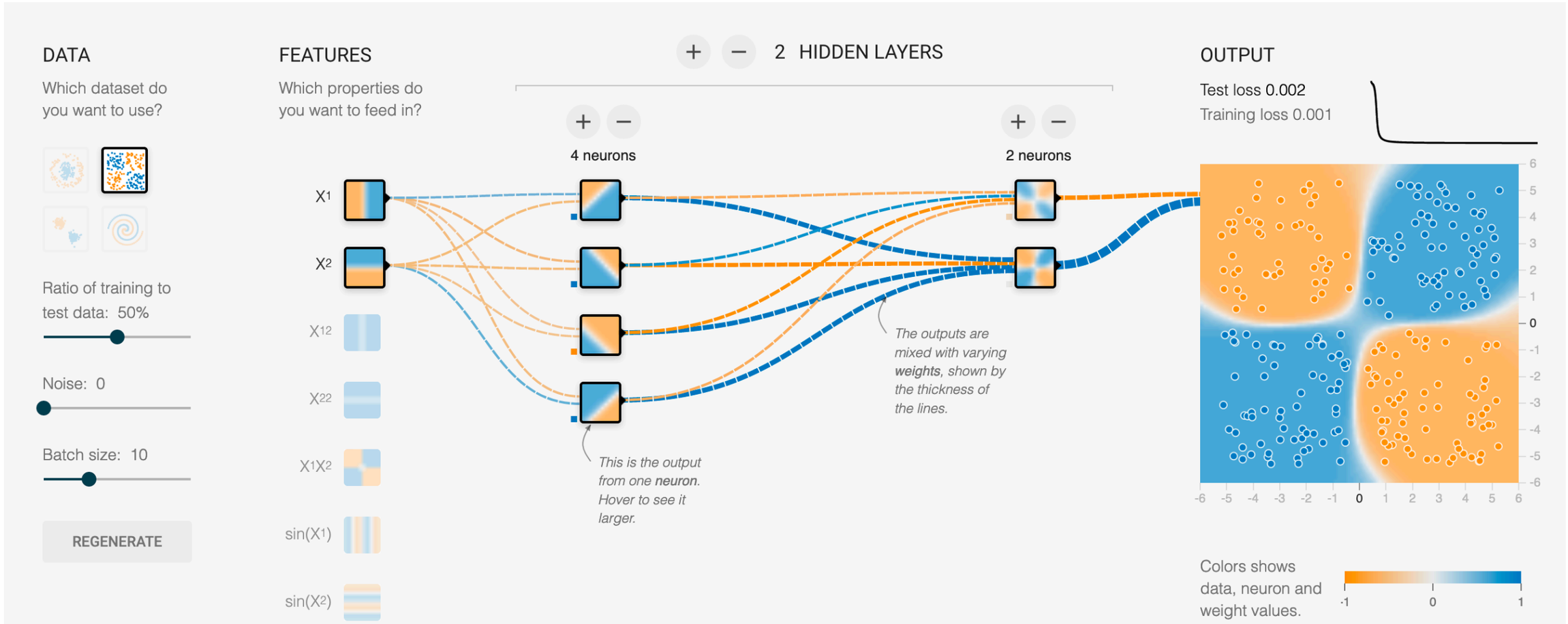
Intuition, MLP

- Reminder: we want a function that is **scalable**, **powerful**, easy to **optimise**
- Composition of elementary functions:
  multi-variable linear, x -> Wx + b
  element wise continuous functions (non-linearity) x -> sigma(x)

Intuition, MLP

- Go deeper!
- More complexity with less neurons

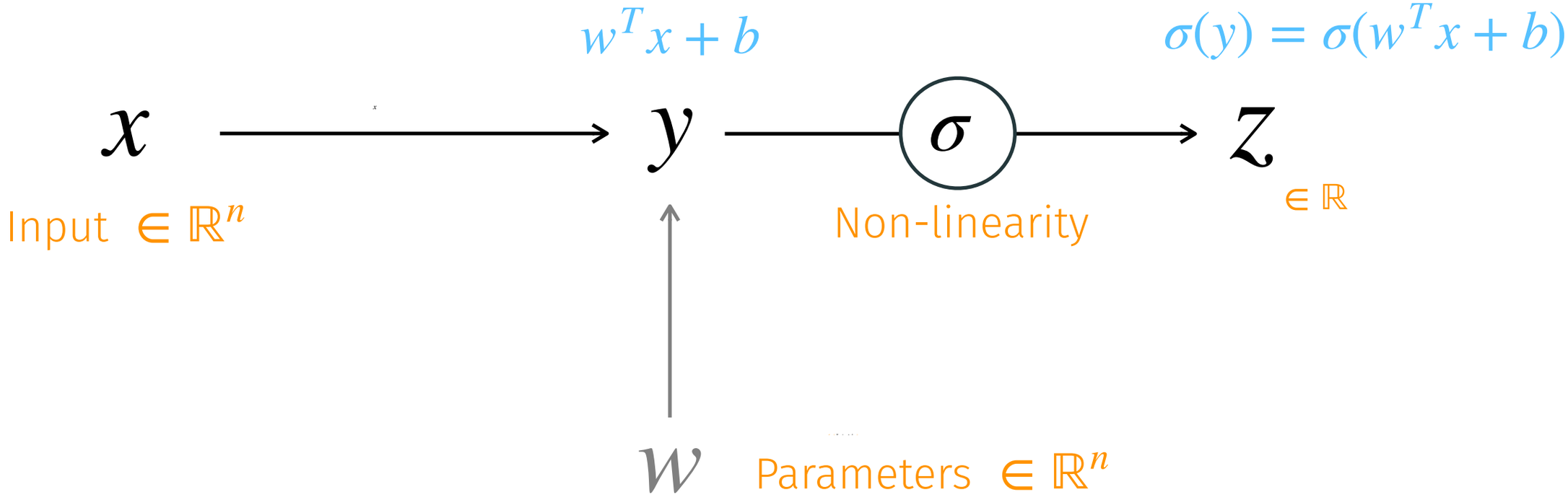




https://playground.tensorflow.org

Backpropagation

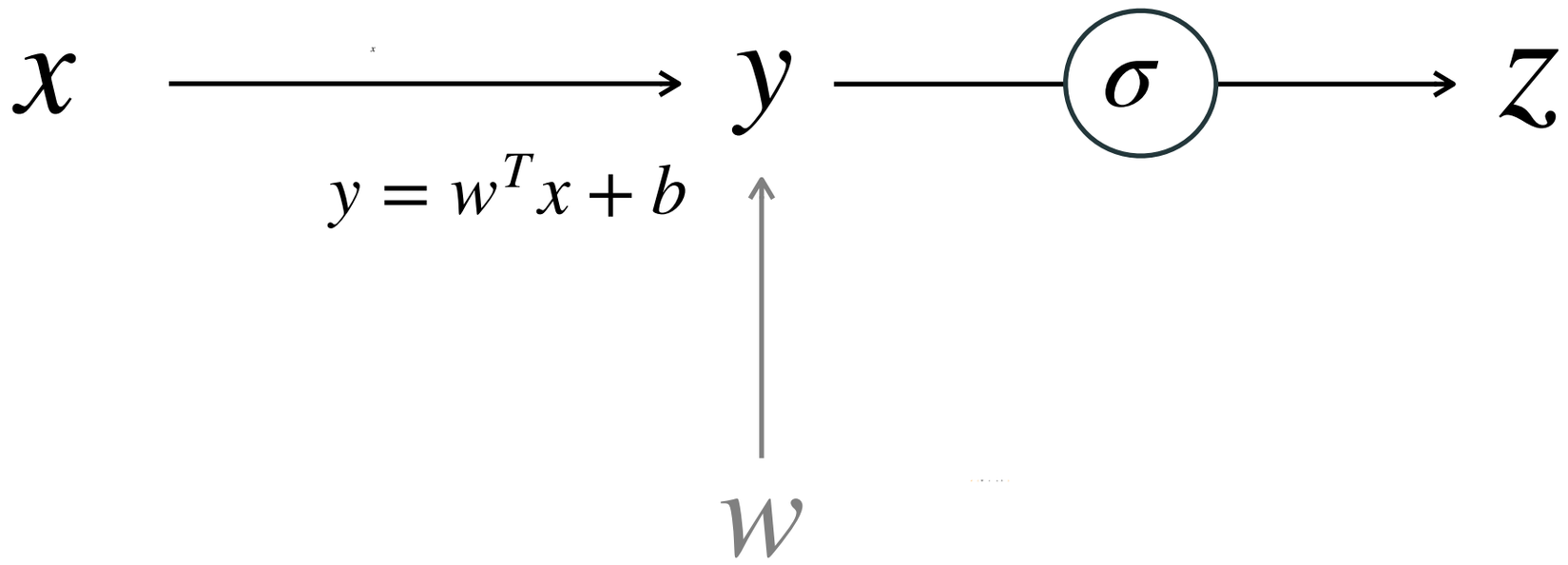
- Computation Graph: Single Neuron
- To update parameters w, need to compute $\frac{\partial z}{\partial w}$



Backpropagation

- Computation Graph: Single Neuron
- To update parameters w, need to compute $\frac{\partial z}{\partial w}$

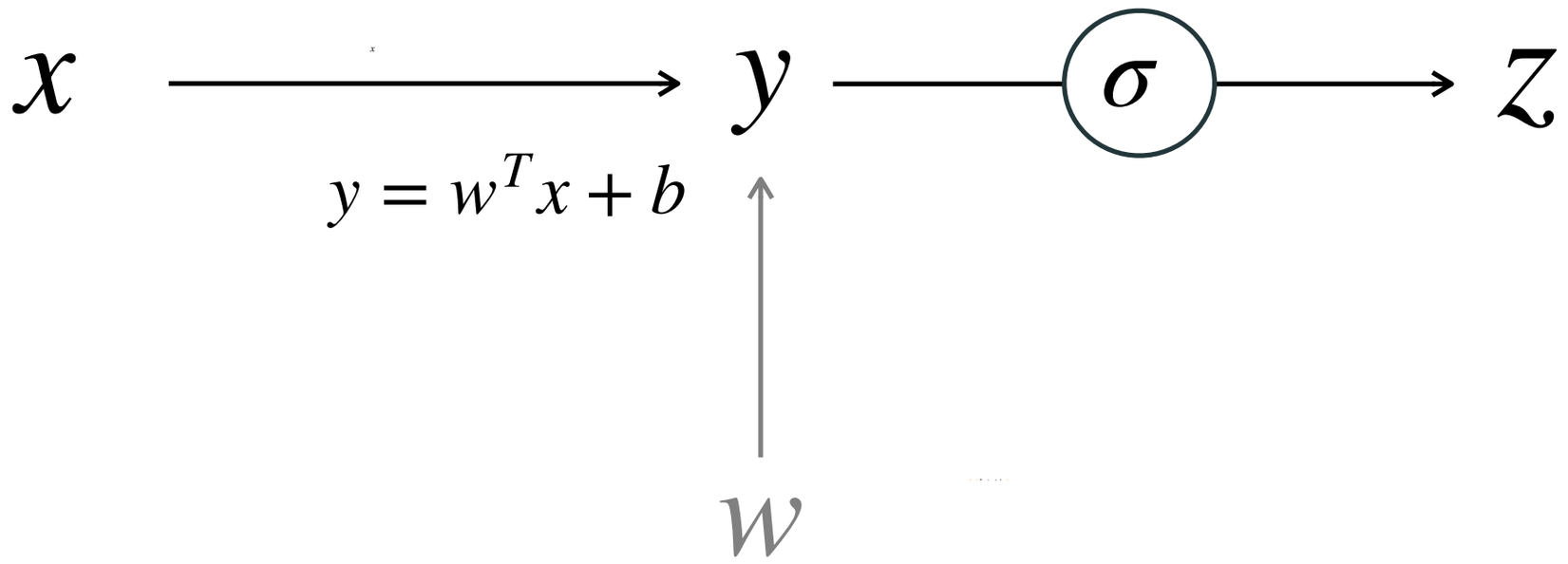
$$\textbf{Chain rule}: \frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial w}$$



Backpropagation

- Computation Graph: Single Neuron
- To update parameters w, need to compute $\frac{\partial z}{\partial w}$

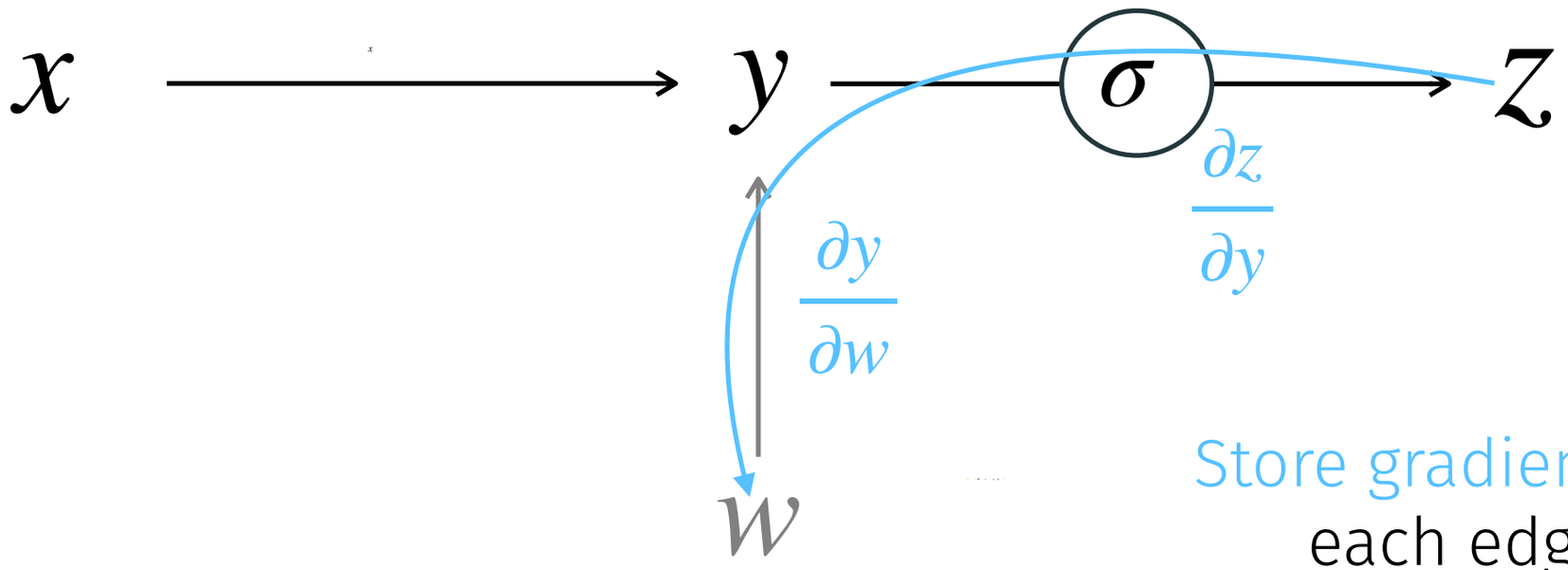
$$\textbf{Chain rule}: \frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial w} = \sigma'(y)\, x$$



Backpropagation

- Computation Graph: Single Neuron
- To update parameters w, need to compute $\frac{\partial z}{\partial w}$

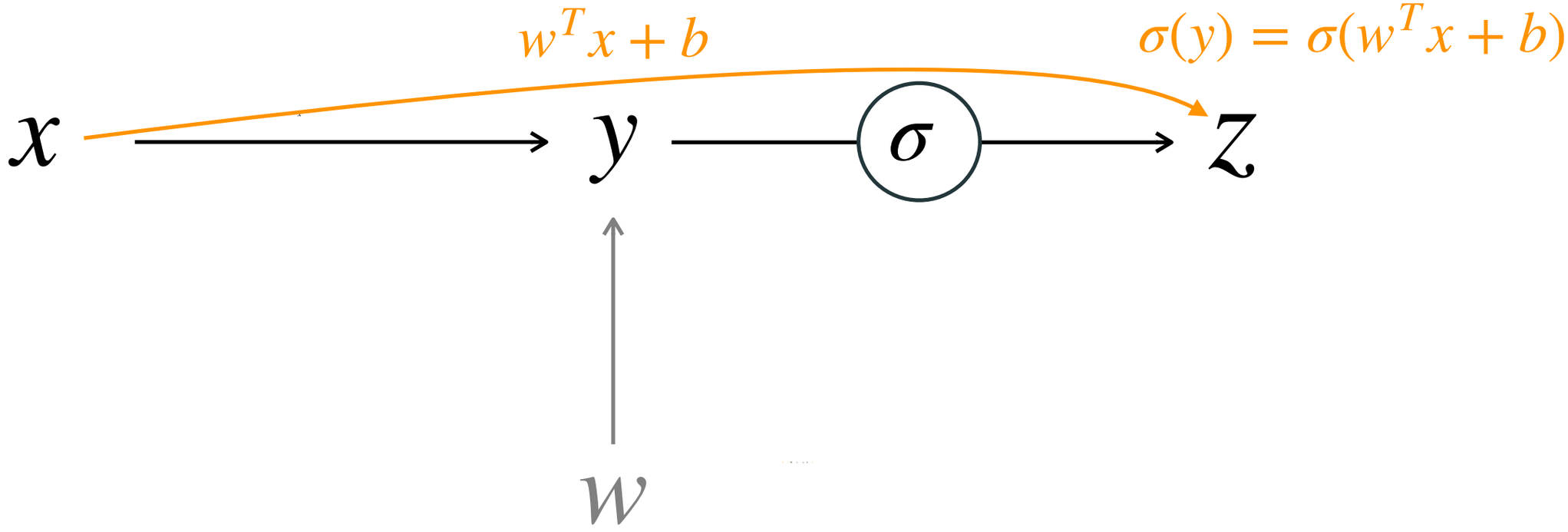
$$\text{Chain rule: } \frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial w} = \sigma'(y)\, x$$



Store gradients on each edge!

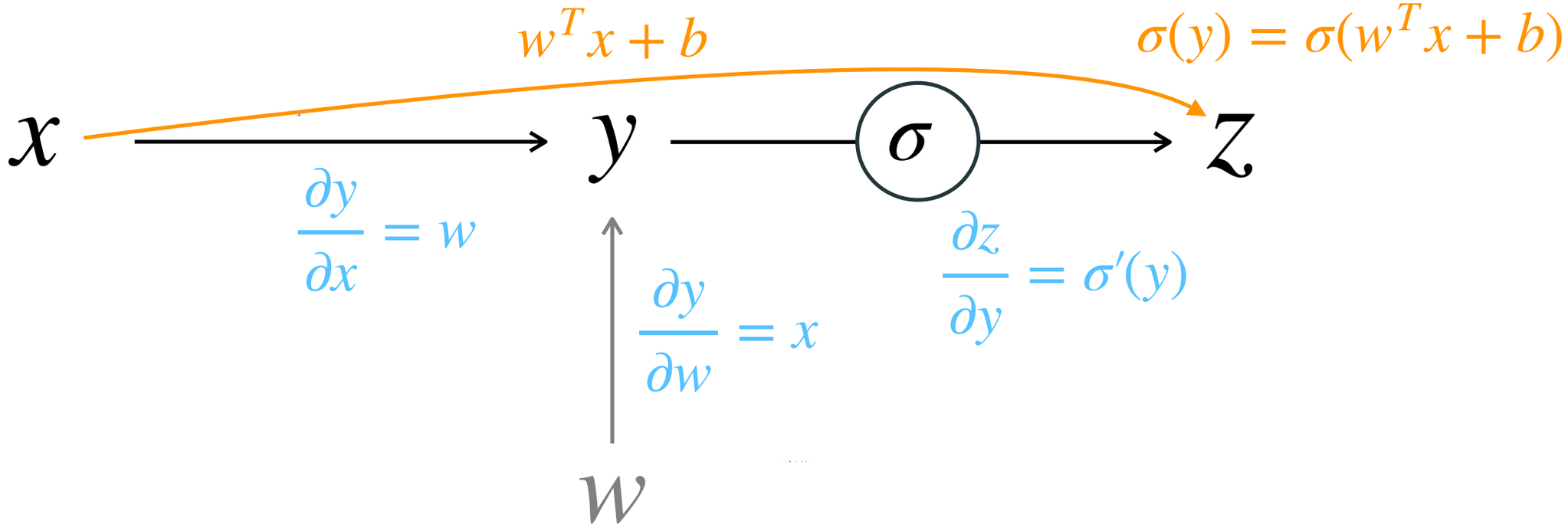
Backpropagation

- Forward pass:
  - compute model output



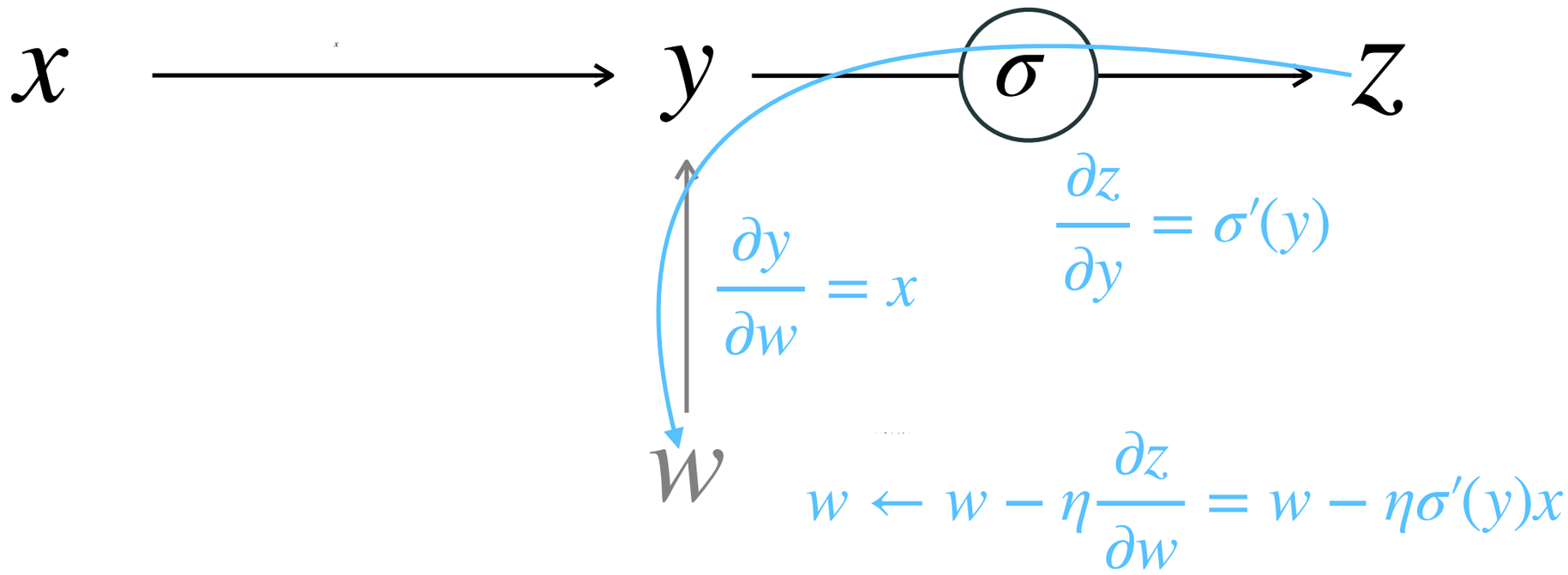
Backpropagation

- Forward pass:
  - compute model output
  - store gradients



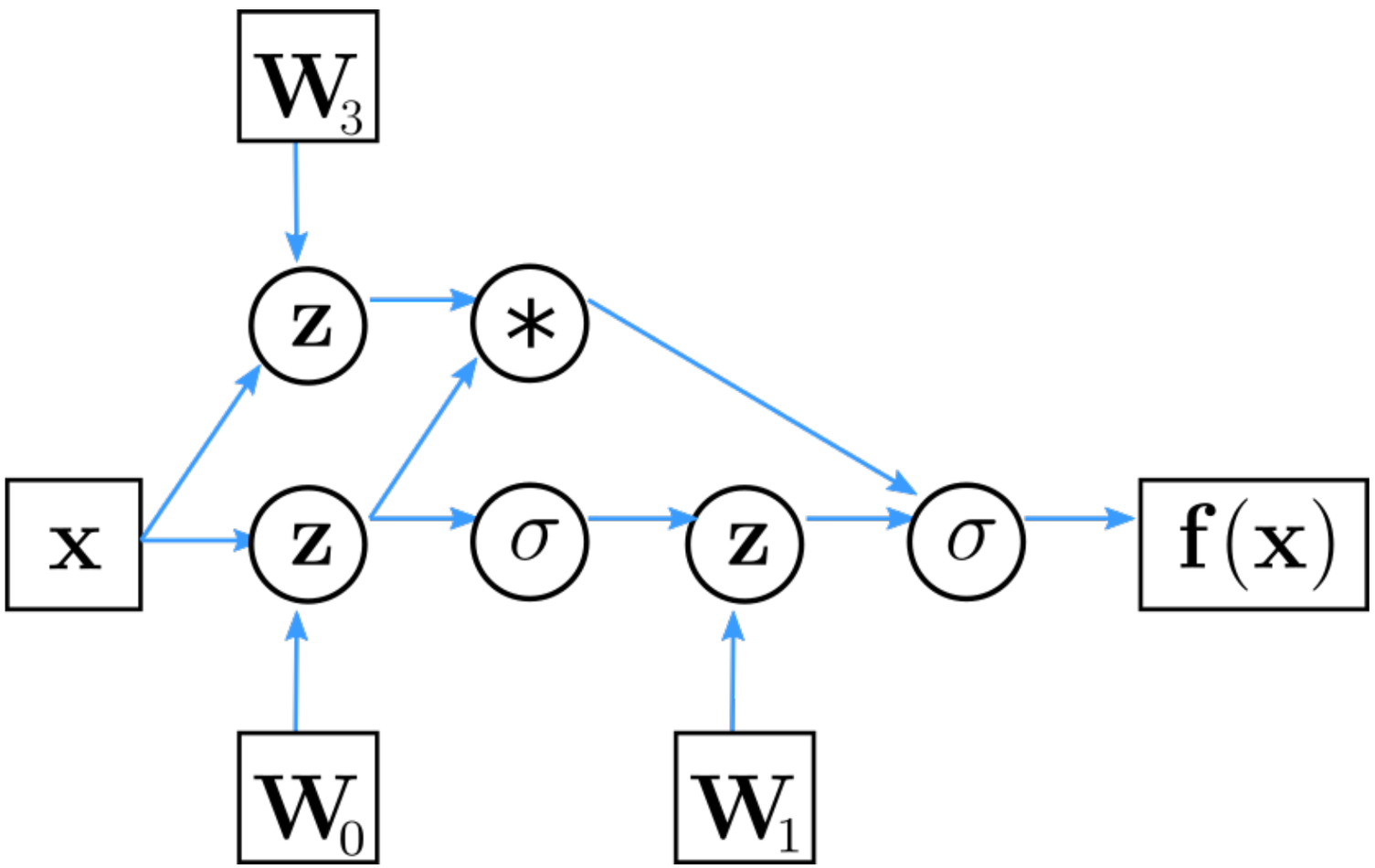
Backpropagation

- Forward pass:
  - compute model output
  - store gradients
- **Backward** pass: Chain gradients, update parameters



Backpropagation

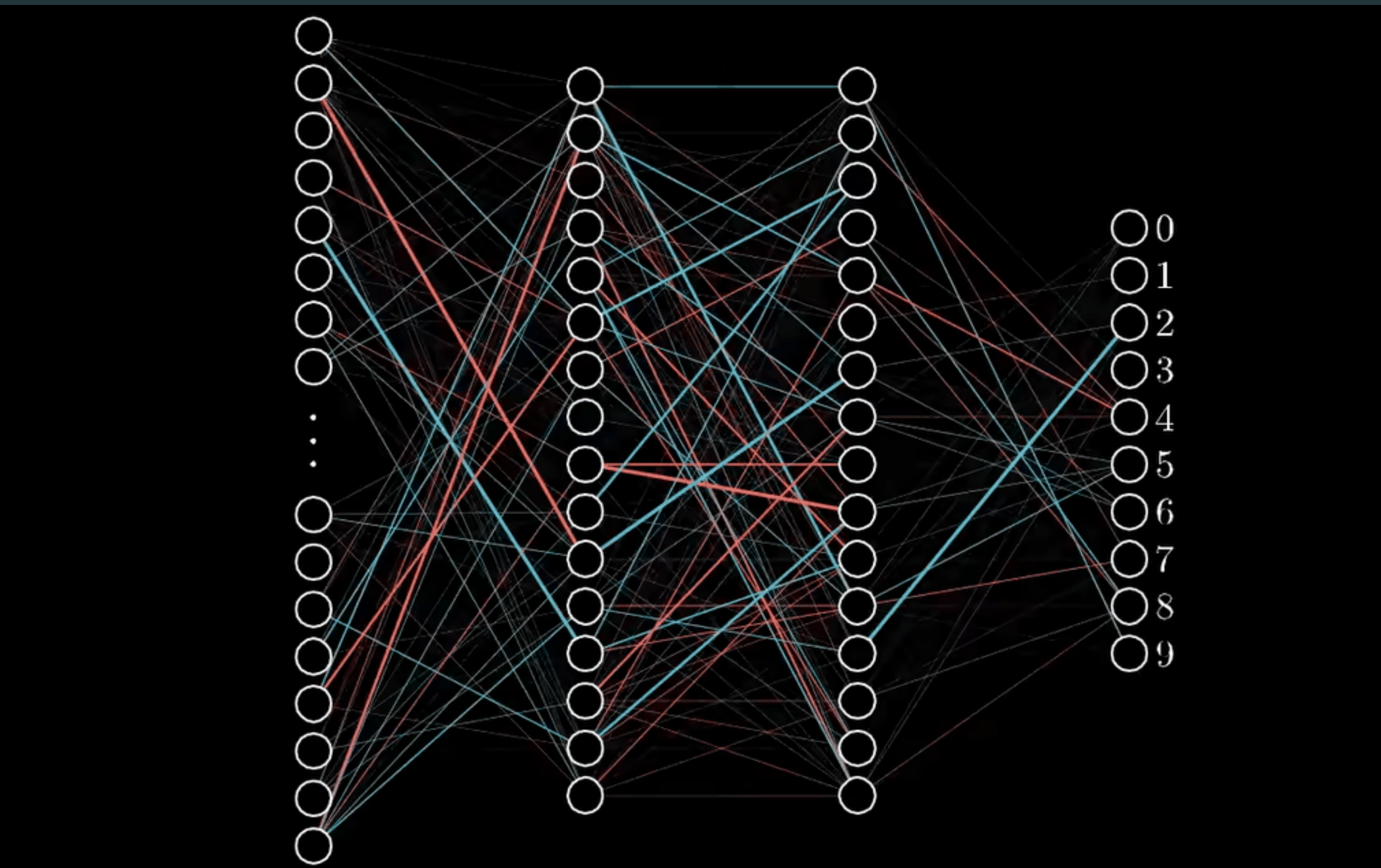
- Works for arbitrarily complicated computation graph, not only sequential !



Backpropagation

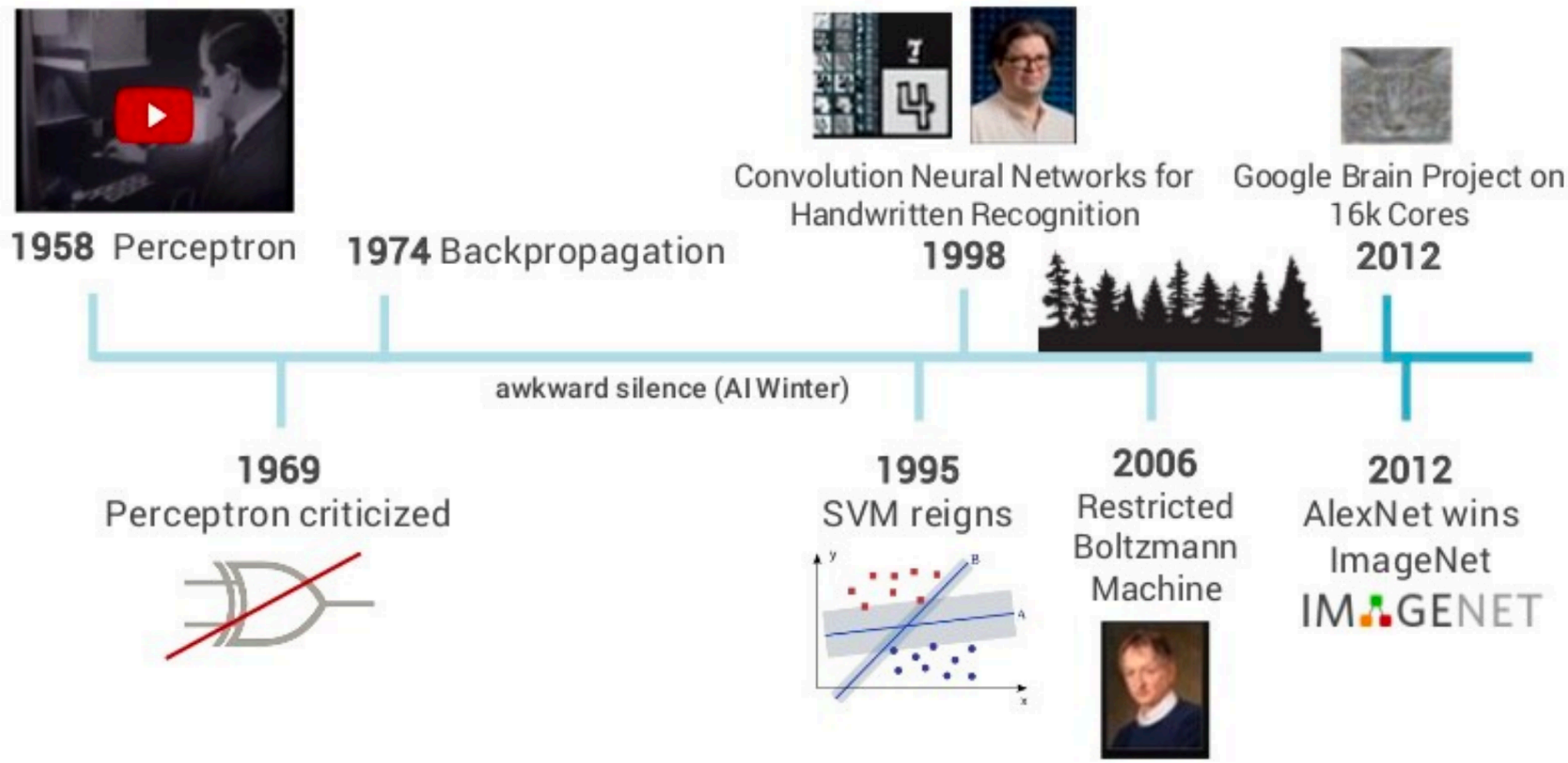
- Works for arbitrarily complicated computation graph, not only sequential !
- Libraries, framework for auto-differentiation:

Backpropagation



Aparté: Why now ?

- We have known Neural Networks for a long time, why did it take so long to break through?

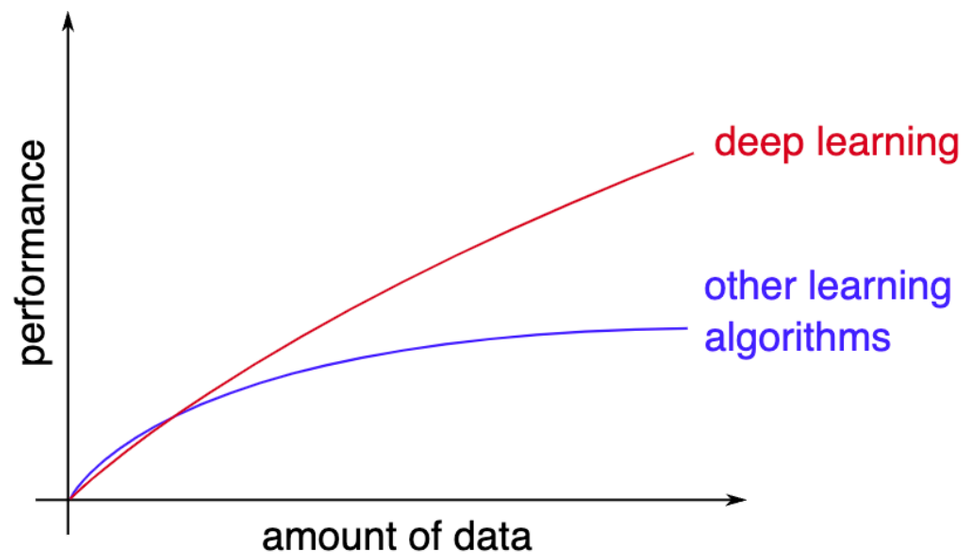


Aparté: Why now ?

Cascading effect: AVAILABILITY / EASE OF USE
now there are many open-source libraries (PyTorch, TensorFlow-> JAX), lots of documentation, online computation resources (AWS, google cloud, Microsoft Azure).

- Need a lot of (labeled) data
- Computing power

  (GPUs, TPUs, distributed)



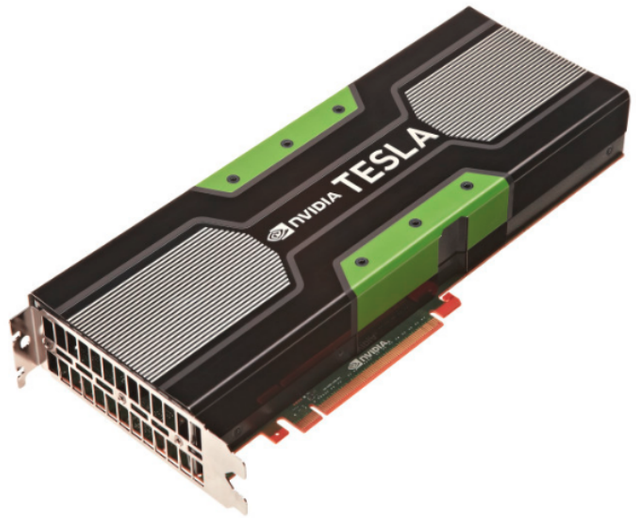
Aparté: Why now ?

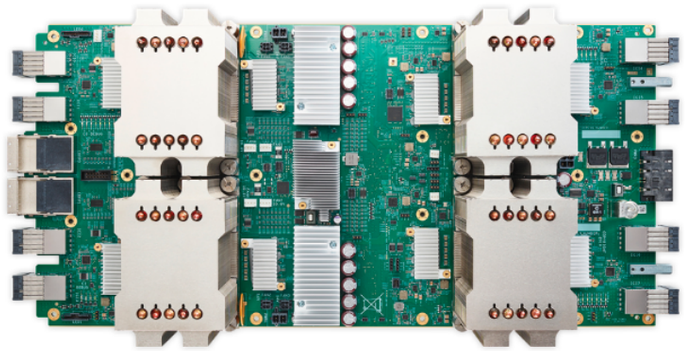
Cascading effect: AVAILABILITY / EASE OF USE
now there are many open-source libraries (PyTorch, TensorFlow-> JAX), lots of documentation, online computation resources (AWS, google cloud, Microsoft Azure).

- Need a lot of (labeled) data

- Computing power

  (GPUs, TPUs, distributed)





Aparté: Why now ?

- Need a lot of (labeled) data
- Computing power (GPUs, TPUs, distributed)
- Algorithms/Libraries



Deep Learning Toolbox:
Modern Architectures and Training Tips

Which Architecture should I choose

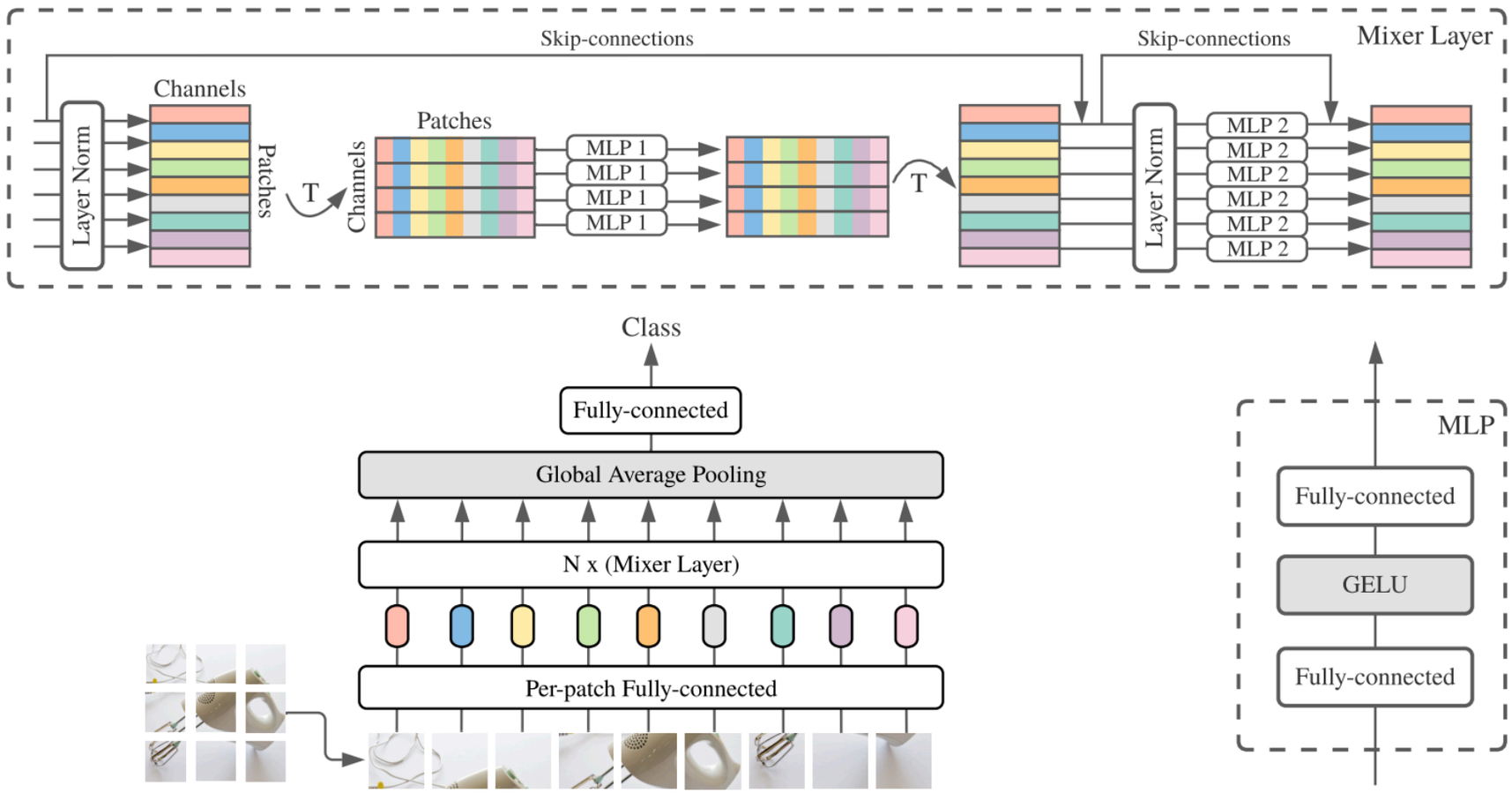
- Many flavours of deep neural networks:
- MLP, CNN, RNN, GNN, Transformers
- Which is the best suited for your task?

Intuition, Multi-Layer Perceptron

- MLP: Solid baseline, when input is not too high-dimensional
- In fact, decent performance on small images, with data-augmentation

Intuition, Multi-Layer Perceptron

- MLP: Solid baseline, when input is not too high-dimensional
- In fact, decent performance on small images, with data-augmentation
- Can be competitive with some extra mixing.



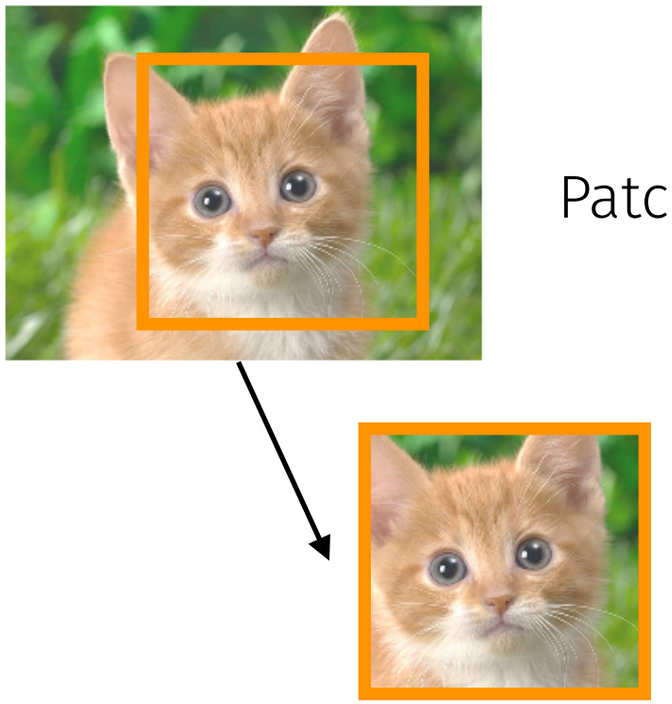
MLP-Mixer (2021)

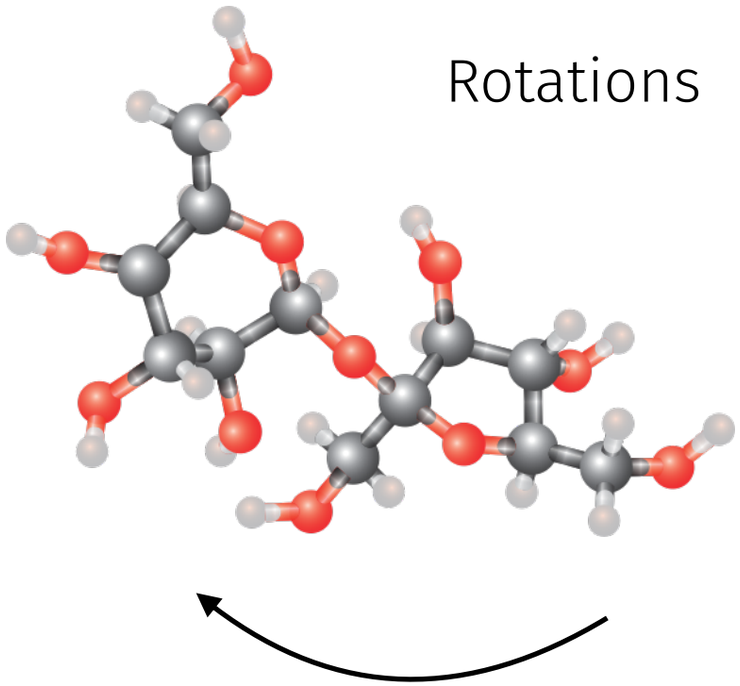
Handling Invariances in the data

- Many types of data have invariances / equivariances g
  - Invariance to transformation g: $f(g \cdot x) = f(x)$
  - Equivariance to transformation g: $f(g \cdot x) = g \cdot f(x)$

Handling Invariances in the data

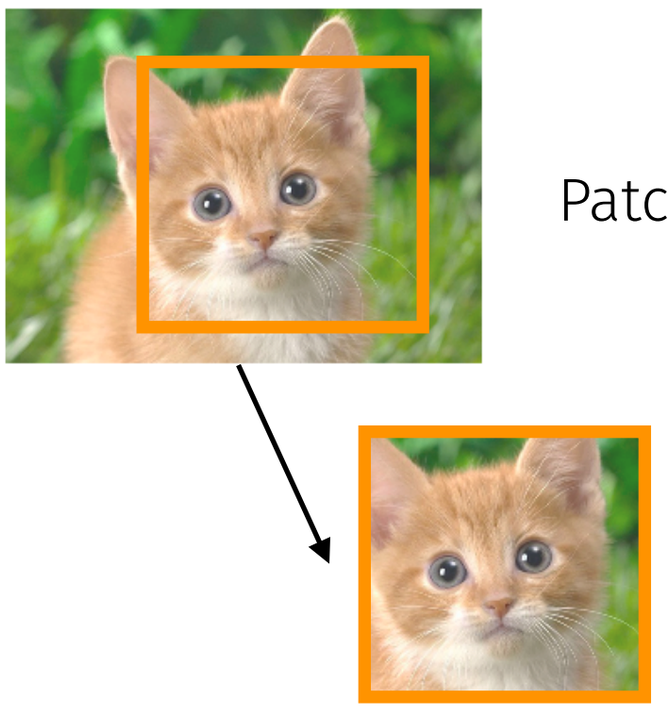
- Many types of data have invariances / equivariances g
  - Invariance to transformation g: $f(g \cdot x) = f(x)$
  - Equivariance to transformation g: $f(g \cdot x) = g \cdot f(x)$

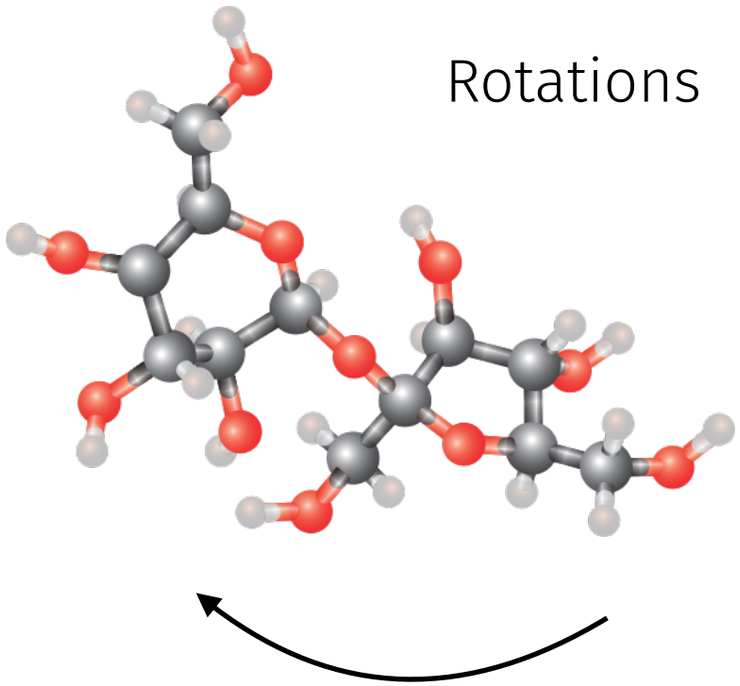




Handling Invariances in the data

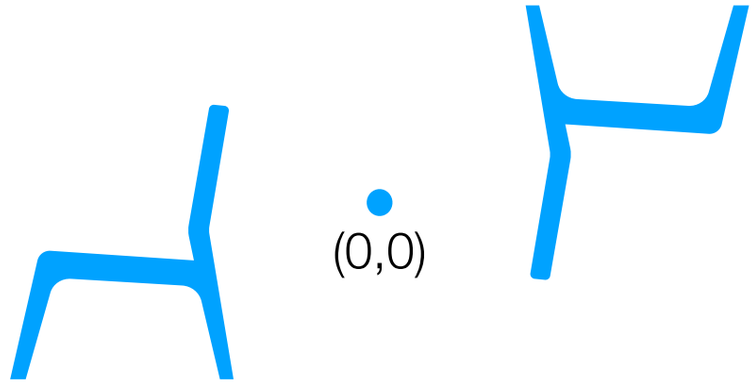
- Many types of data have invariances / equivariances g
  - Encode the invariance in the network
  - Perform data augmentation





Handling Invariances in the data

- Many types of data have invariances / equivariances
  - Encode the invariance in the network
  - Perform data augmentation



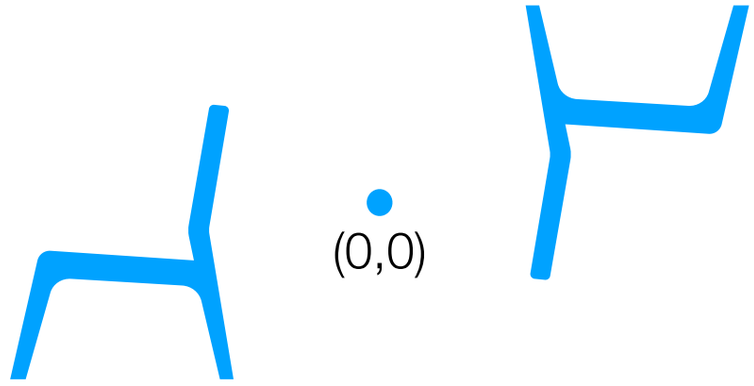
- Example: *point symmetry* $g \cdot x = -x$

  How would you encode invariance to g?

    - in the network:
    - In the data:

Handling Invariances in the data

- Many types of data have invariances / equivariances
  - Encode the invariance in the network
  - Perform data augmentation



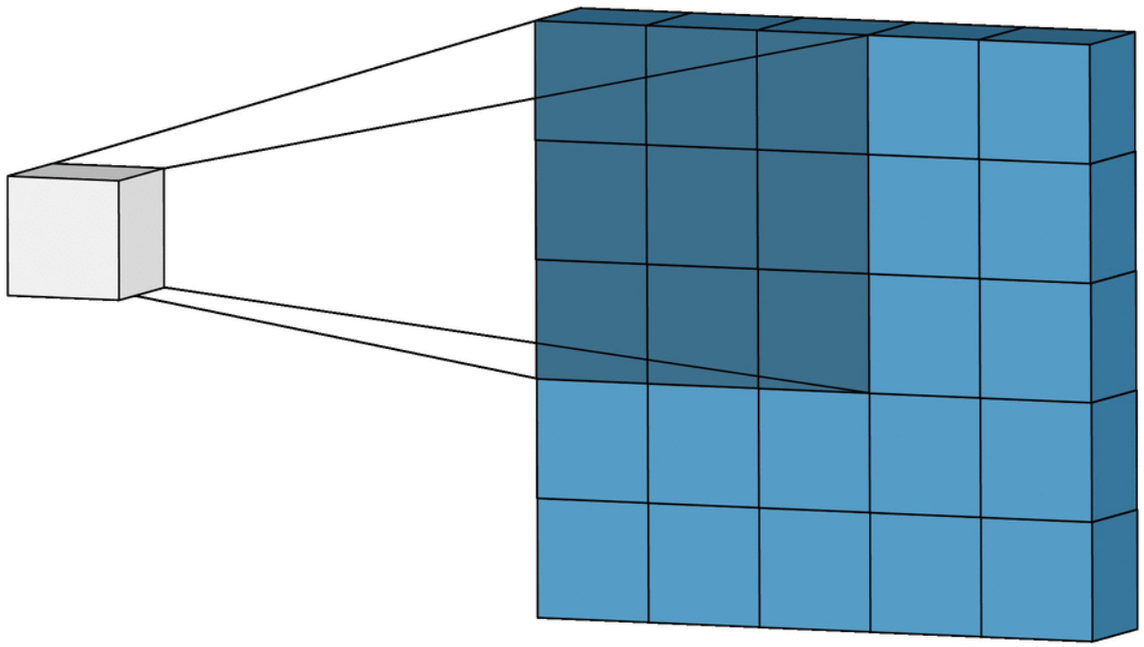
- Example: *point symmetry* $g \cdot x = -x$

  How would you encode invariance to g?

  - in the network: $f(x) + f(-x)$
  - In the data: randomly flip sign of x

Convolutional Networks

- Constraints over the linear layers, which apply over spatial regions
  -



Convolution operation

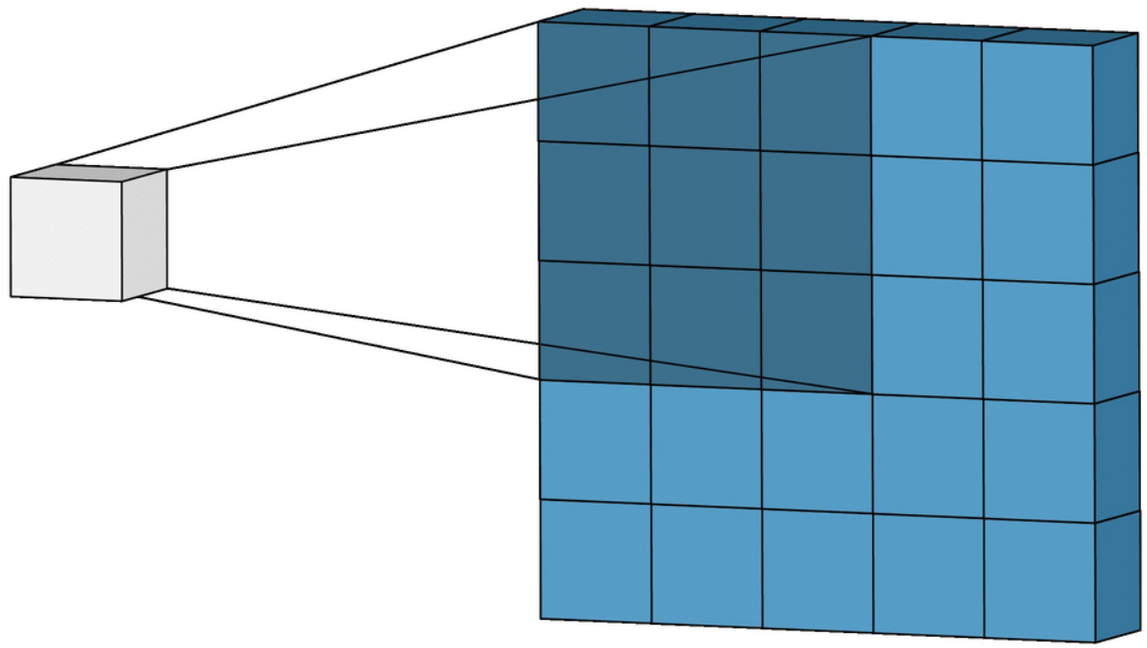
(Animation from towardsdatascience post)

Convolutional Networks

- Constraints over the linear layers, which apply over spatial regions

- For long the go-to baseline on images

- Works for all types of spatial patterns:
  - time series
  - videos
  - sequences
  - audio signal

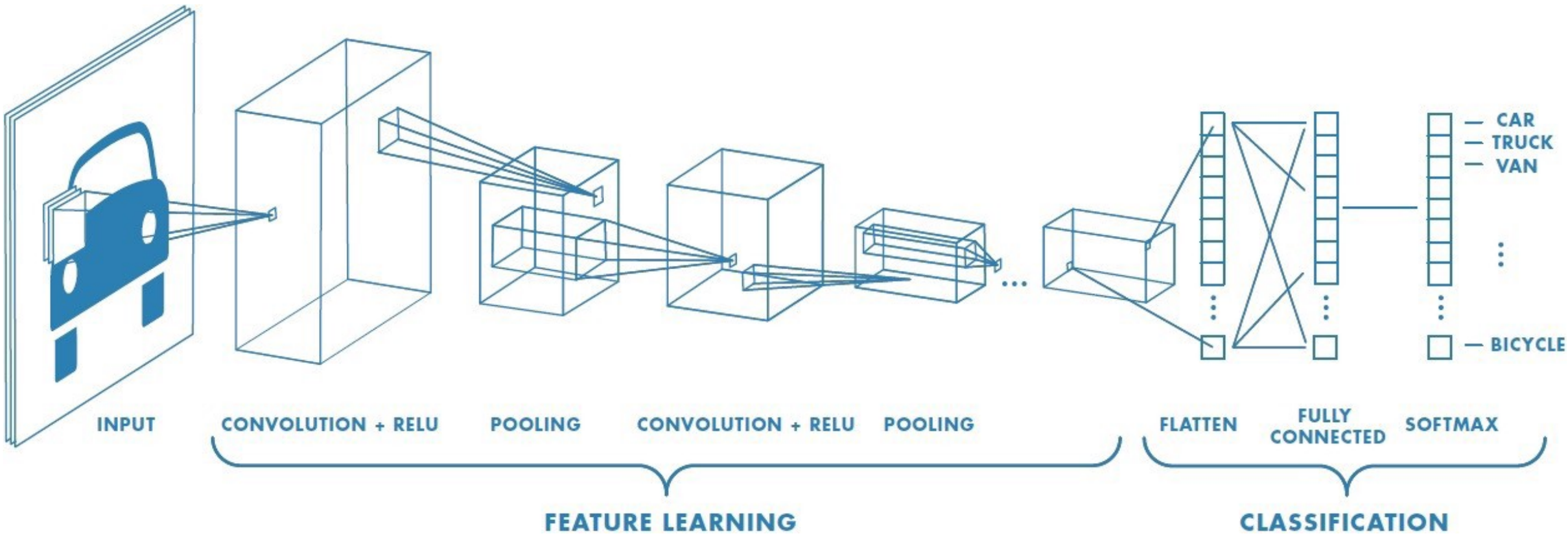


Convolution operation

(Animation from towardsdatascience post)

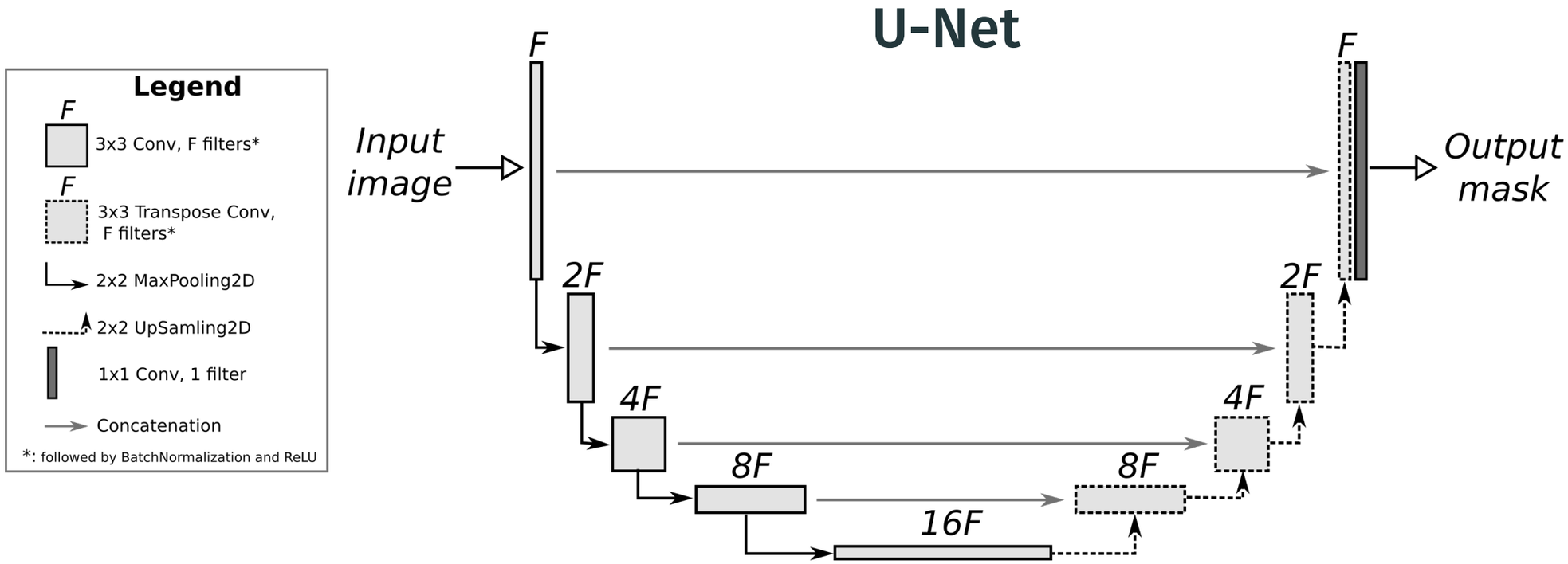
Convolutional Networks

- Constraints over the linear layers, which apply over spatial regions



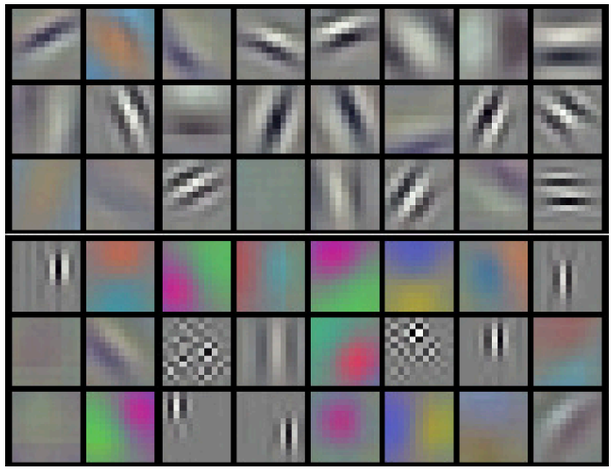
Convolutional Networks

- Constraints over the linear layers, which apply over spatial regions
- Variant with U-Net, useful when output is an image:
- for segmentation, image generation...

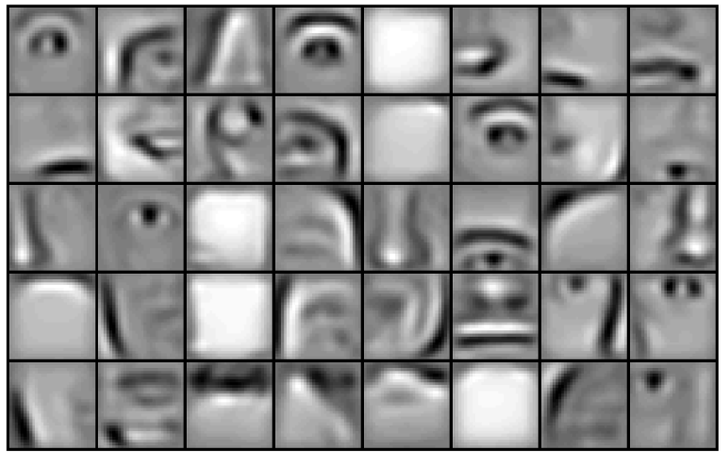


Convolutional Networks

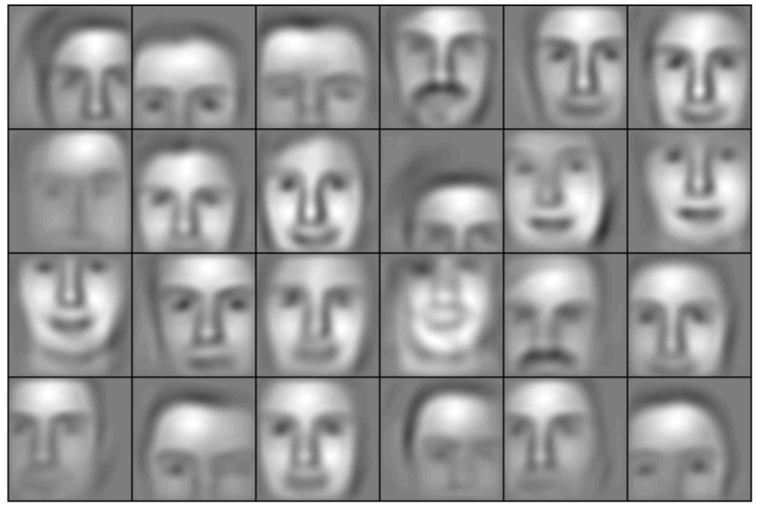
- Compose simple operations
- Get a rich representation



First layer



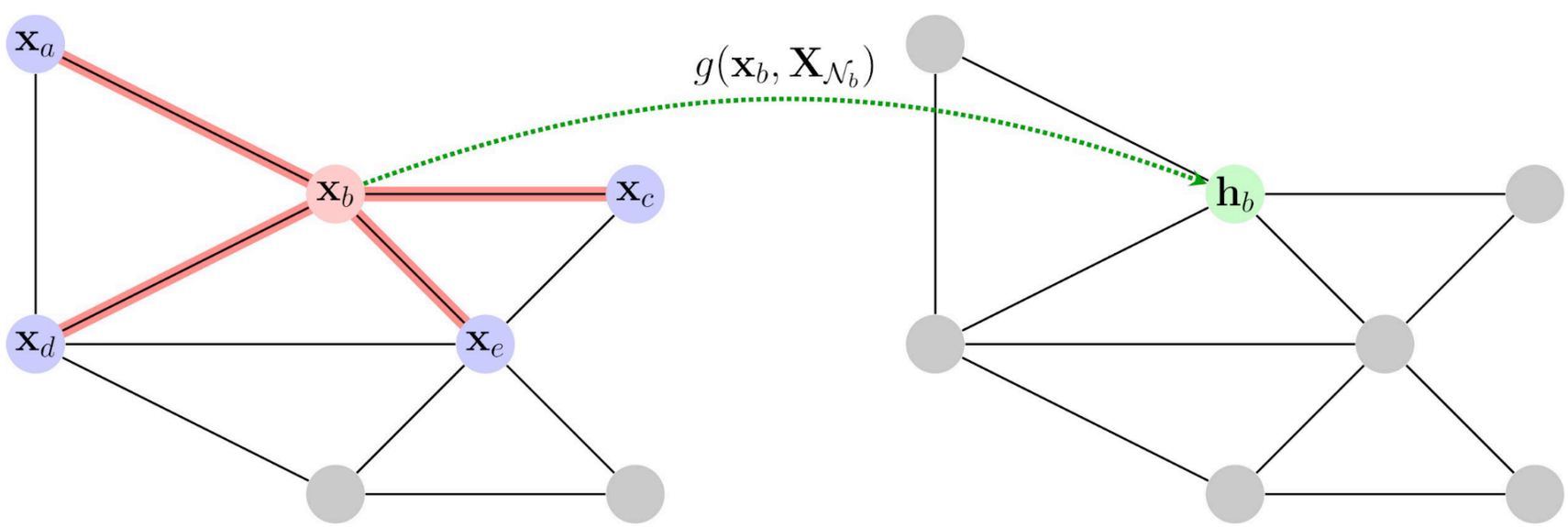
Intermediate layer



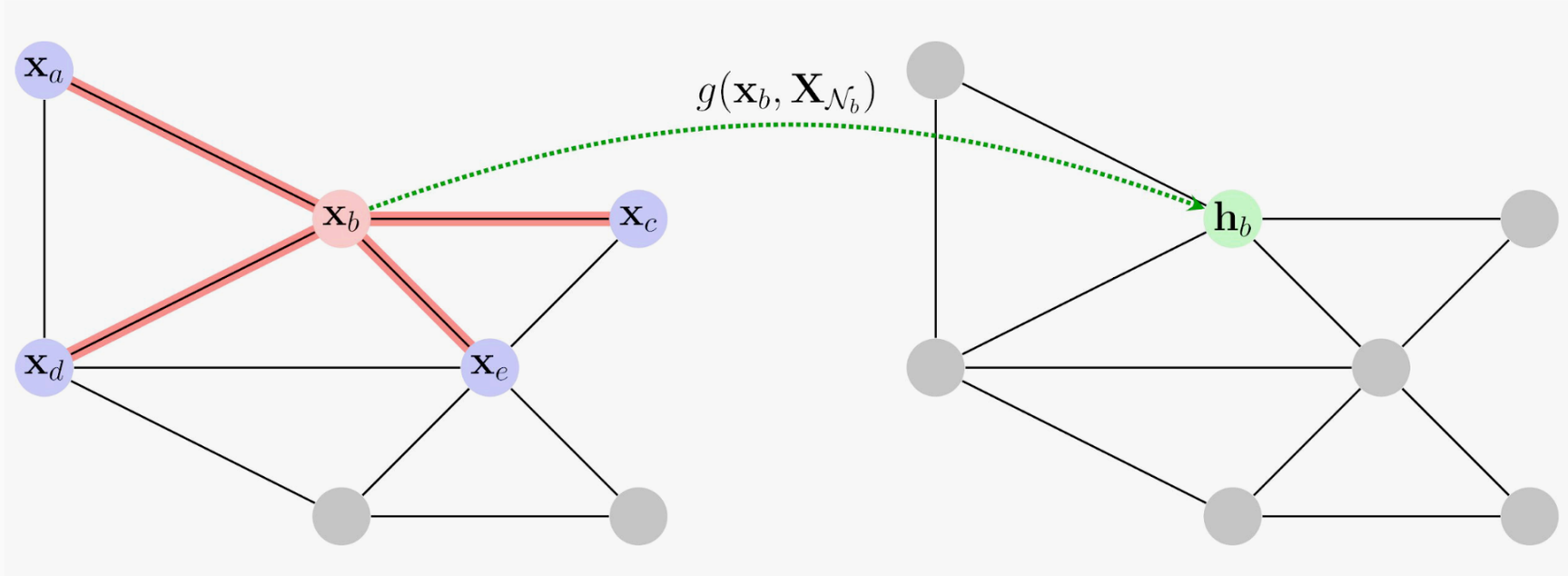
Final layer

Convolutional Networks

- Limitations: long-range dependencies, sparse irregular structures.
- Natural extension to graphs: GNNs



Message passing NN

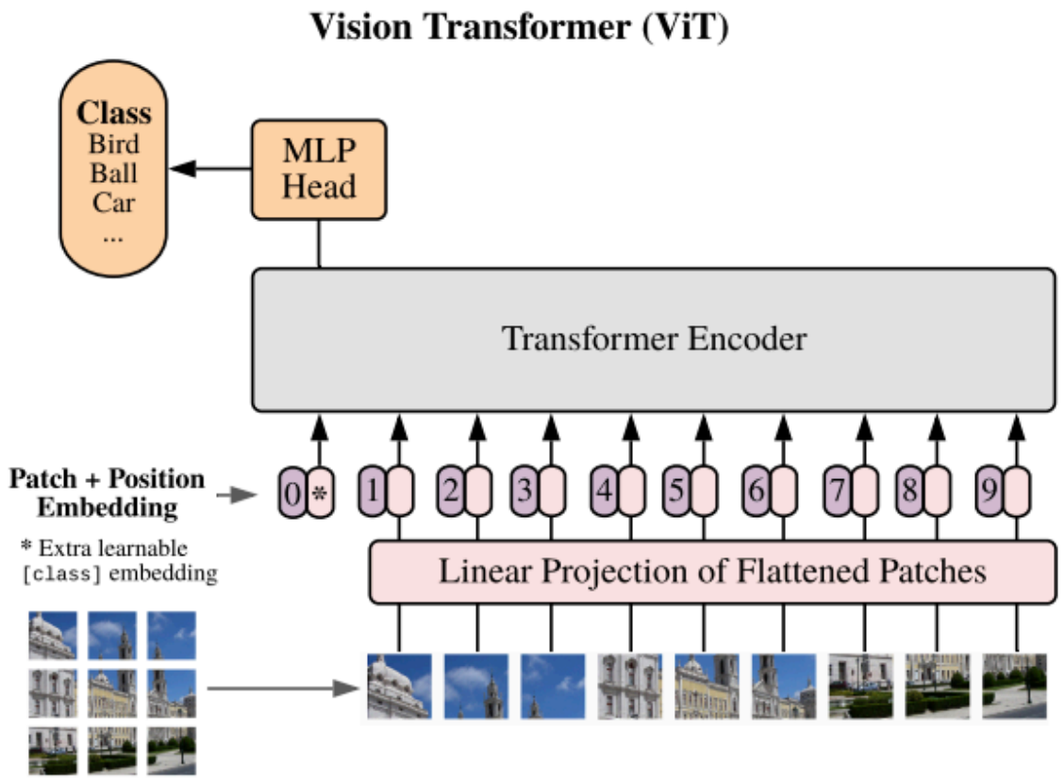


- Each node is aware of its (1-hop) neighborhood.
  As in CNNs, adding layers enlarges the receptive field.
- Both theoretically grounded and performant on real datasets.
- Popular variants: GCN, GAT, GIN

Kipf, T. N., & Welling, M. (ICLR'2017). Semi-supervised classification with graph convolutional networks.
Veličković et al. (ICLR'2018). Graph attention networks.
Xu et al. (ICLR'2019). How powerful are graph neural networks?

Transformers

- Ubiquitous



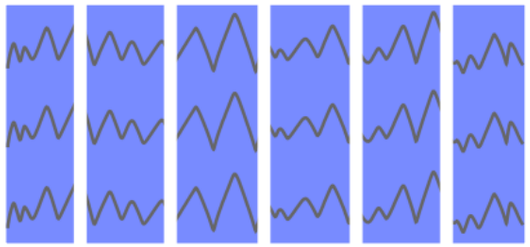
Transformers
[Vaswani et al. 2017]

Transformers

- Break input data into *tokens* $h_1, \ldots, h_n$

Image patches

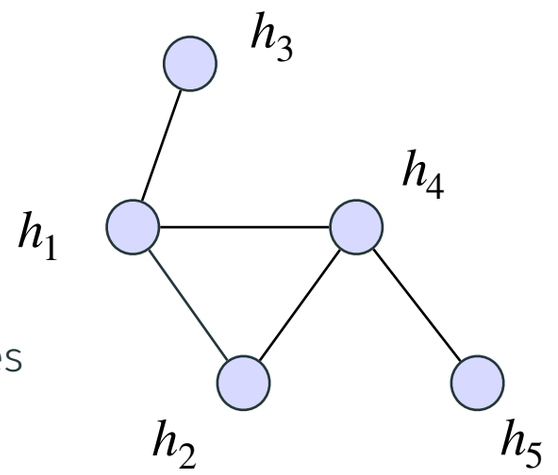
Time series fragments



Word pieces

```
[`_Sacl`, `AI`, `_Work`, `shop`]
```

Nodes and edges in graphs

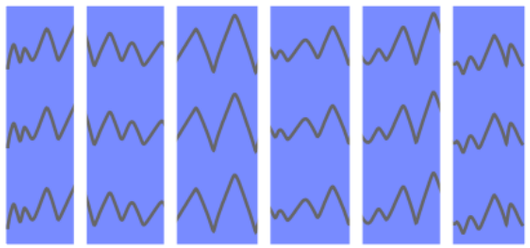


Transformers

- Break input data into *tokens* $h_1, \ldots, h_n$

Image patches

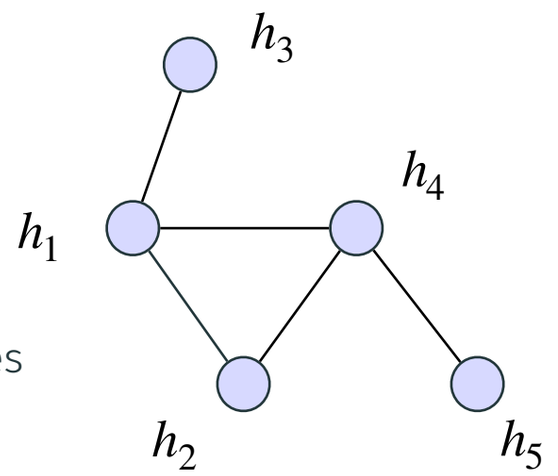
Time series fragments



Word pieces

```
[`_Sacl`, `AI`, `_Work`, `shop`]
```

Nodes and edges in graphs



- Alternate:
  - Processing tokens independently $h_i \leftarrow f(h_i)$
  - Mixing tokens $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, h_j$

Self-Attention, general form

- Update in a Transformer layer: $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, V(h_j)$

- The *attention score* α_{ij} is a similarity score between nodes **i** and **j**:

$$\tilde{\alpha}_{ij} = \exp\left(Q(h_i) \cdot K(h_j)\right) \qquad \text{normalized: } \alpha_{ij} = \frac{\tilde{\alpha}_{ij}}{\sum_k \tilde{\alpha}_{ik}}$$

Q, K and V are linear functions ("queries", "keys", "values").

Vaswani et al. (NIPS'2017). Attention Is All You Need.

Self-Attention, general form

- Update in a Transformer layer: $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, V(h_j)$

- The *attention score* α_{ij} is a similarity score between nodes **i** and **j :**

$$\tilde{\alpha}_{ij} = \exp\left(Q(h_i) \cdot K(h_j)\right) \qquad \text{normalized: } \alpha_{ij} = \frac{\tilde{\alpha}_{ij}}{\sum_k \tilde{\alpha}_{ik}}$$

- Positional information (ordering) is *lost:*

How do we keep **structural information?**

The mouse ate the cat

?

The cat ate the mouse

Vaswani et al. (NIPS'2017). Attention Is All You Need.

Positional Encodings (PEs)

- **Absolute** positional encodings: add a vector p to node features

$$h \leftarrow h \,\|\, p$$

Schematically, the attention score can be rewritten:

$$score(i,j) = \exp\left(h_i \cdot h_j\right) \exp\left(p_i \cdot p_j\right)$$

Semantic — Structural/Positional

Positional Encodings (PEs)

- **Absolute** positional encodings: add a vector p to node features

$$h \leftarrow h \,\|\, p$$

Schematically, the attention score can be rewritten:

$$score(i,j) = \exp\left(h_i \cdot h_j\right) \exp\left(p_i \cdot p_j\right)$$

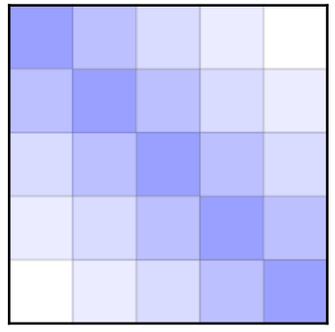
Semantic — Structural/Positional

- p_i should encode the position of element i in the sentence/image/graph

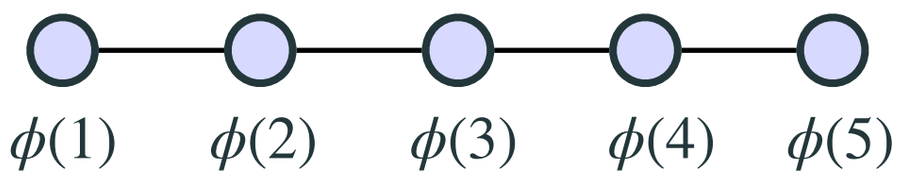
Positional Encodings - Sequences

- In NLP, the standard sinusoidal PE [1] is
  $p_i = (\dots, \cos(\omega_n i), \dots, \sin(\omega_n i), \dots)$
  - Unique
  - Independent of sequence length
  - $p_i \cdot p_j$ only depends on the relative distance $|i - j|$

$p^T \cdot p$



- Other choices include *learnable PEs*, possible because of *natural ordering* in sequences.



[1] Vaswani et al. (NIPS'2017). Attention Is All You Need.

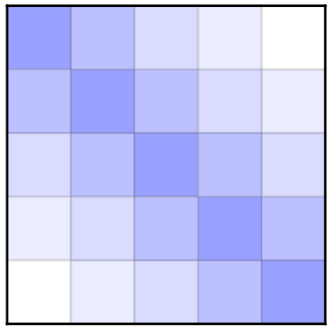
Relative Positional Encodings (RPEs)

- **Relative PE**: Modulate the attention matrix directly

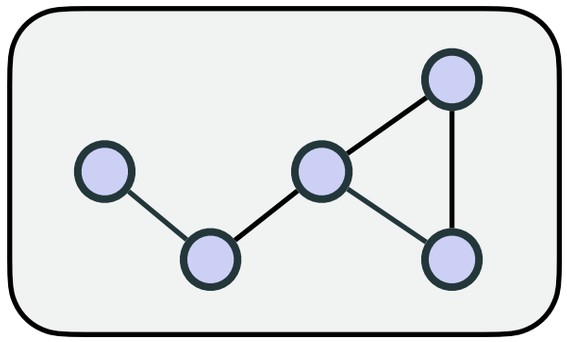
$$score(i,j) = \exp\left(h_i \cdot h_j\right) \; bias\left(i,j\right)$$

Semantic — Structural/Positional

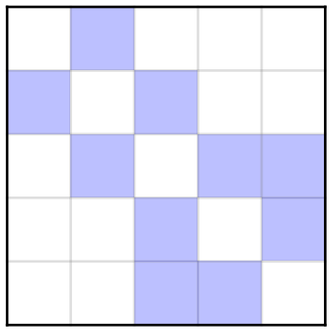
Ex for sequences: $bias(i,j) = e^{-\lambda|i-j|}$



Example with graphs

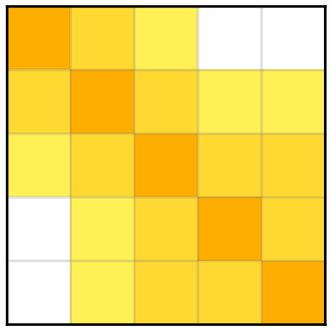


Edge Prior Info



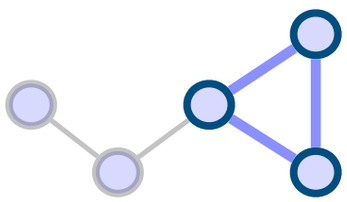
Semantic edge features
(Bond types)

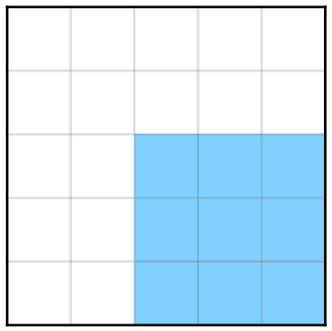
+



Structural Encodings
(Based on random walks or
shortest paths)

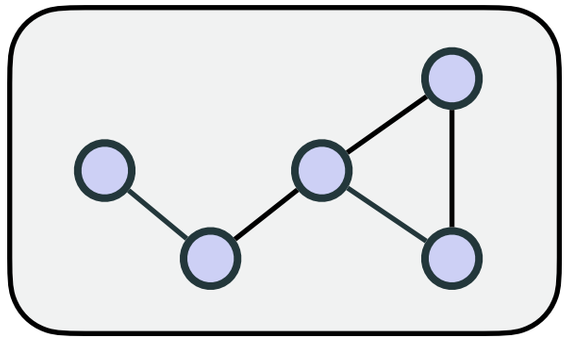
+



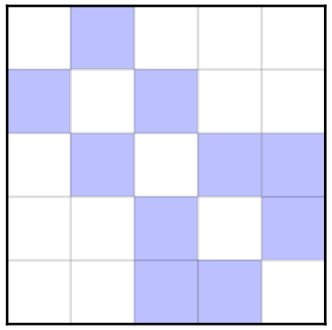


Topology Encoding
(Example w rings)

Example with graphs

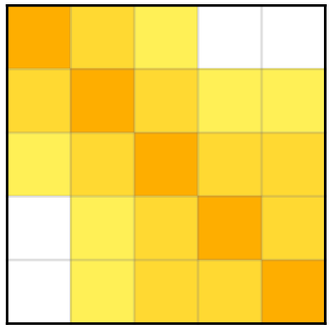


Edge Prior Info



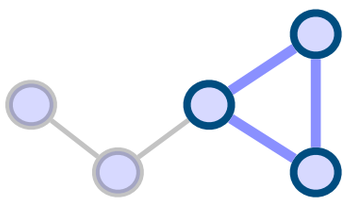
Semantic edge features
(Bond types)

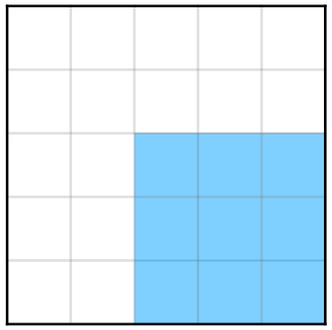
+



Structural Encodings
(Based on random walks or
shortest paths)

+



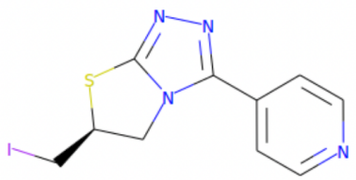


Topology Encoding
(Example w rings)

Transformers

- Can introduce heterogeneous data
  - Class token
  - Time: $f(t, x)$
- Interpretable attention maps
- Massively parallelizable. (GPU/TPU)

Attention maps

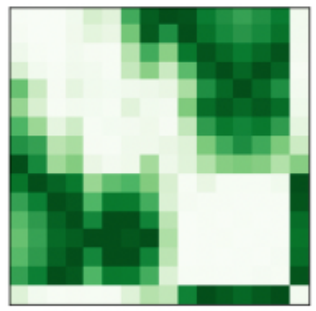


Full Attention Matrix

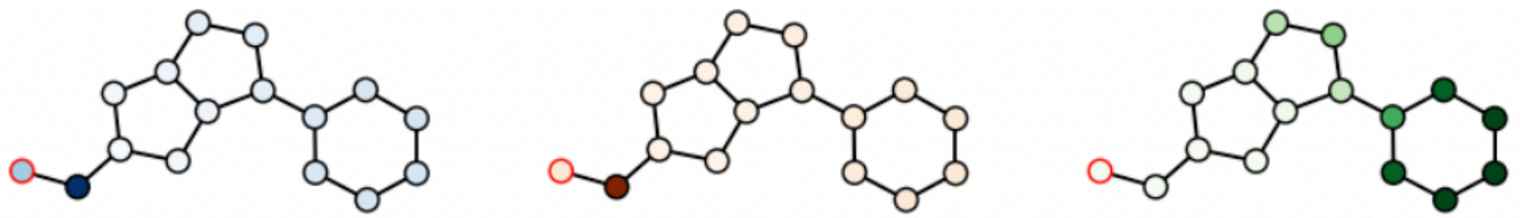
Channel 1: Focus on **neighbors**

Channel 7: Focus on **rings**

Channel 12: Focus on **faraway** nodes



Node 1



Node 3

Node 5

Node 13

Self-Attention, general form

- Update in a Transformer layer: $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, V(h_j)$

- The *attention score* α_{ij} is a similarity score between nodes **i** and **j**:

$$\tilde{\alpha}_{ij} = \exp\left(Q(h_i) \cdot K(h_j)\right) \qquad \text{normalized: } \alpha_{ij} = \frac{\tilde{\alpha}_{ij}}{\sum_k \tilde{\alpha}_{ik}}$$

Q, K and V are linear functions ("queries", "keys", "values").

Vaswani et al. (NIPS'2017). Attention Is All You Need.

Self-Attention, general form

- Update in a Transformer layer: $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, V(h_j)$

- The *attention score* α_{ij} is a similarity score between nodes **i** and **j**:

$$\tilde{\alpha}_{ij} = \exp\left(Q(h_i) \cdot K(h_j)\right) \qquad \text{normalized: } \alpha_{ij} = \frac{\tilde{\alpha}_{ij}}{\sum_k \tilde{\alpha}_{ik}}$$

Q, K and V are linear functions ("queries", "keys", "values").

Vaswani et al. (NIPS'2017). Attention Is All You Need.

Self-Attention, general form

- Update in a Transformer layer: $h_i \leftarrow \Sigma_j \, \alpha_{ij} \, V(h_j)$

- The *attention score* α_{ij} is a similarity score between nodes **i** and **j**:

$$\tilde{\alpha}_{ij} = \exp\left(Q(h_i) \cdot K(h_j)\right) \qquad \text{normalized: } \alpha_{ij} = \frac{\tilde{\alpha}_{ij}}{\sum_k \tilde{\alpha}_{ik}}$$

- Positional information (ordering) is *lost:*

How do we keep **structural information?**

The mouse ate the cat

?

The cat ate the mouse

Vaswani et al. (NIPS'2017). Attention Is All You Need.

Positional Encodings (PEs)

- **Absolute** positional encodings: add a vector p to node features

$$h \leftarrow h \,\|\, p$$

Schematically, the attention score can be rewritten:

$$score(i,j) = \exp\left(h_i \cdot h_j\right) \exp\left(p_i \cdot p_j\right)$$

Semantic — Structural/Positional

Positional Encodings (PEs)

- **Absolute** positional encodings: add a vector p to node features

$$h \leftarrow h \,\|\, p$$

Schematically, the attention score can be rewritten:

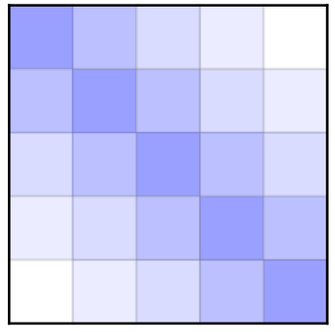
$$score(i,j) = \underbrace{\exp\left(h_i \cdot h_j\right)}_{\text{Semantic}} \underbrace{\exp\left(p_i \cdot p_j\right)}_{\text{Structural/Positional}}$$

- p_i should encode the position of element i in the sentence/image/graph

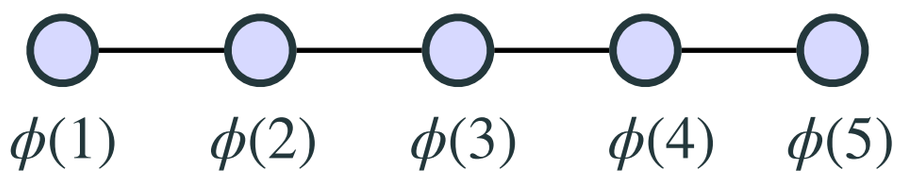
Positional Encodings - Sequences

- In NLP, the standard sinusoidal PE [1] is
  $p_i = (\dots, \cos(\omega_n i), \dots, \sin(\omega_n i), \dots)$
  - Unique
  - Independent of sequence length
  - $p_i \cdot p_j$ only depends on the relative distance $|i - j|$

$p^T \cdot p$



- Other choices include *learnable PEs*, possible because of *natural ordering* in sequences.



[1] Vaswani et al. (NIPS'2017). Attention Is All You Need.

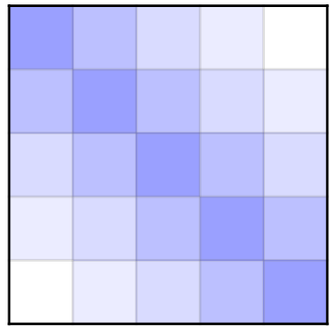
Relative Positional Encodings (RPEs)

- **Relative PE**: Modulate the attention matrix directly

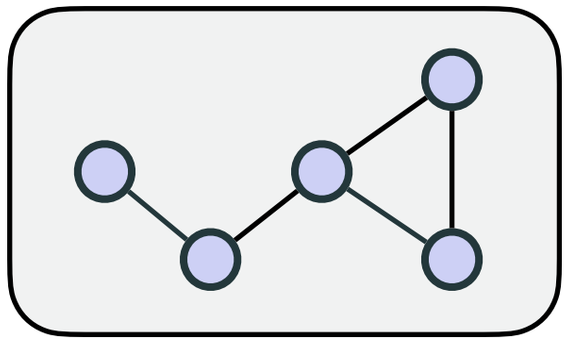
$$score(i,j) = \exp\left(h_i \cdot h_j\right) \; bias\left(i,j\right)$$

Semantic — Structural/Positional

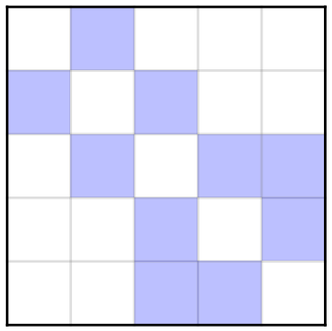
Ex for sequences: $bias(i,j) = e^{-\lambda|i-j|}$



Example with graphs

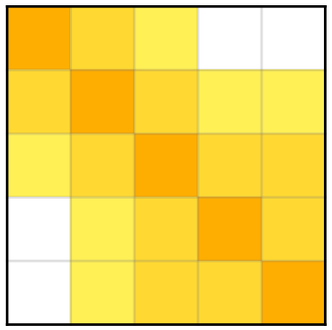


Edge Prior Info



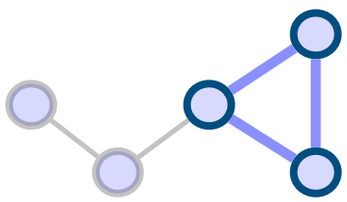
Semantic edge features
(Bond types)

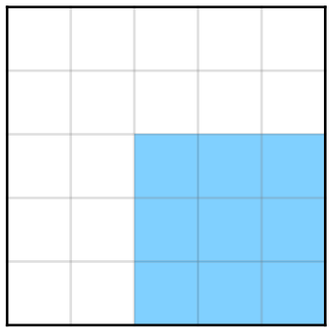
+



Structural Encodings
(Based on random walks or
shortest paths)

+





Topology Encoding
(Example w rings)

Deep Learning Toolbox:
Modern Architectures and Training Tips

II. Training Tips

Data side

- **Normalization**: Ideally input should be standard gaussian, try to get close
- Data Augmentation
- Feature Engineering (Graphs, positional encodings)

Neural Network side

- Regularization:
  - Dropout
  - Weight Decay
- Residual connections
- Batch/Layer Normalisation
- Weight Initialisation

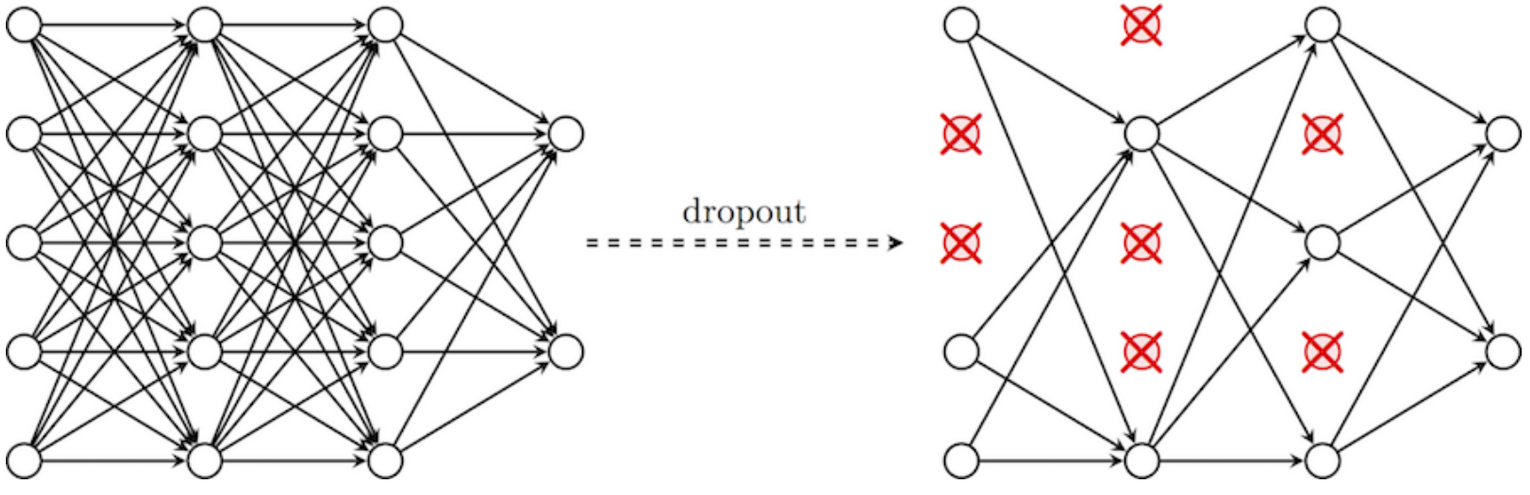


Illustration by Xavier Alameda-Pineda

Neural Network side

- Regularization:
  - Dropout
  - Weight Decay
- Residual connections
- Batch/Layer Normalisation
- Weight Initialisation

$$h^{(l+1)} \leftarrow h^{(l)} + Layer^{(l)}(h^{(l)})$$

[ResNet 2015]

Neural Network side

- Regularization:
  - Dropout
  - Weight Decay
- Residual connections
- Batch/Layer Normalisation
- Weight Initialisation

Want activations to be more or less **standard**:

$$h^{(l)} \leftarrow \frac{h^{(l)} - \mu^{(l)}}{\sigma^{(l)}}$$

Neural Network side

- Regularization:
  - Dropout
  - Weight Decay
- Residual connections
- Batch/Layer Normalisation
- Weight Initialisation

Want activations to be more or less **standard**:

Optimisation side

- Batch Gradient Descent
- Optimizers: Adam, RMSProp
- Learning Rate scheduling

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamP/SGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

ResNet18 on CIFAR10

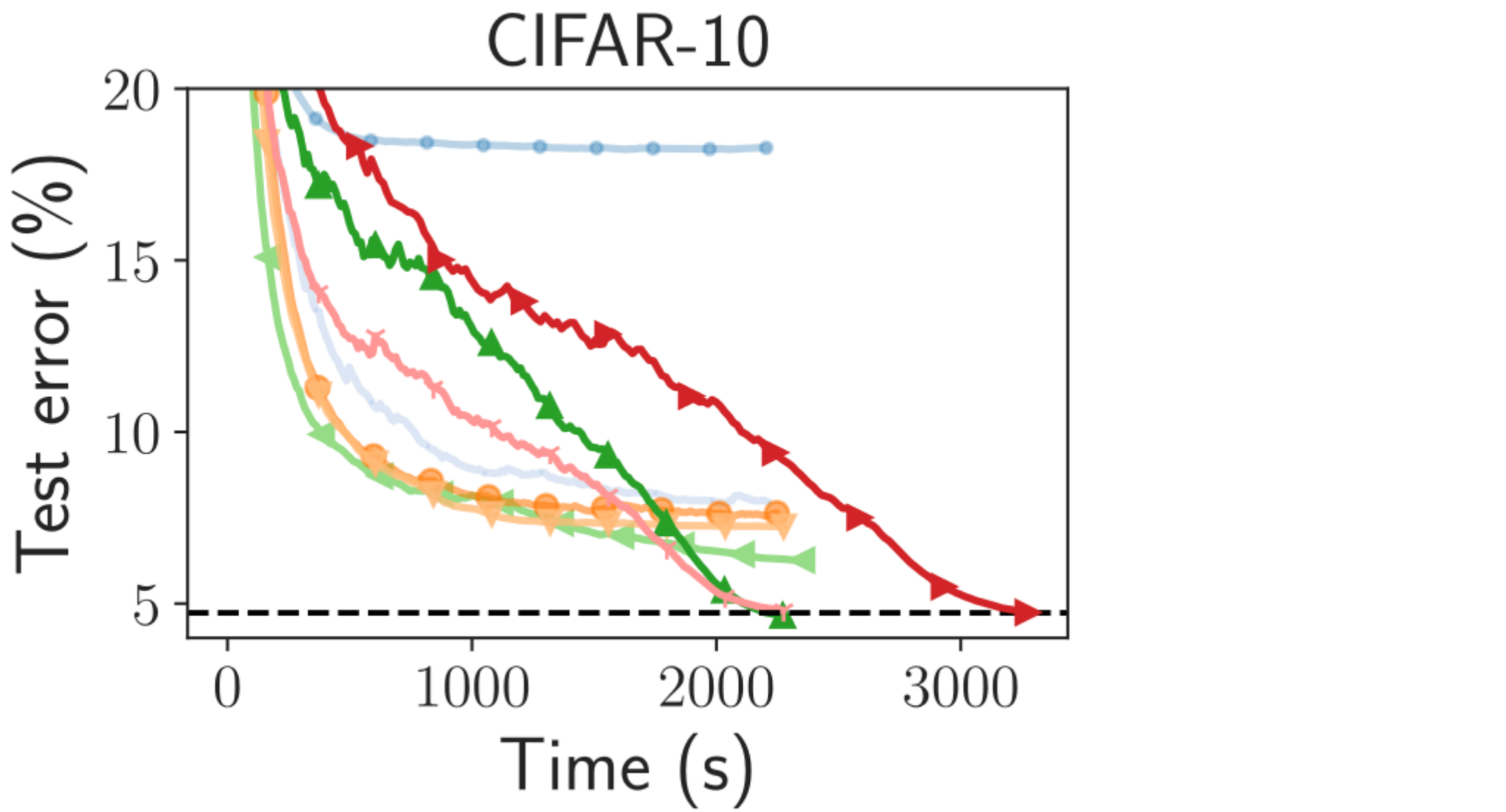
CIFAR10: Image classification of 32×32images from 10 classes.

ResNet18: *strong* baseline for this task.

However, replicating SOTA for this architecture is actually quite hard!

- Optimizer: SGD, Adam, RMSProp, ...
- Learning rate: fixed, exponential, cosine annealing, ...
- Data augmentation: invariances are for the model
- Weight decay?

ResNet18 on CIFAR10



Typical pipeline

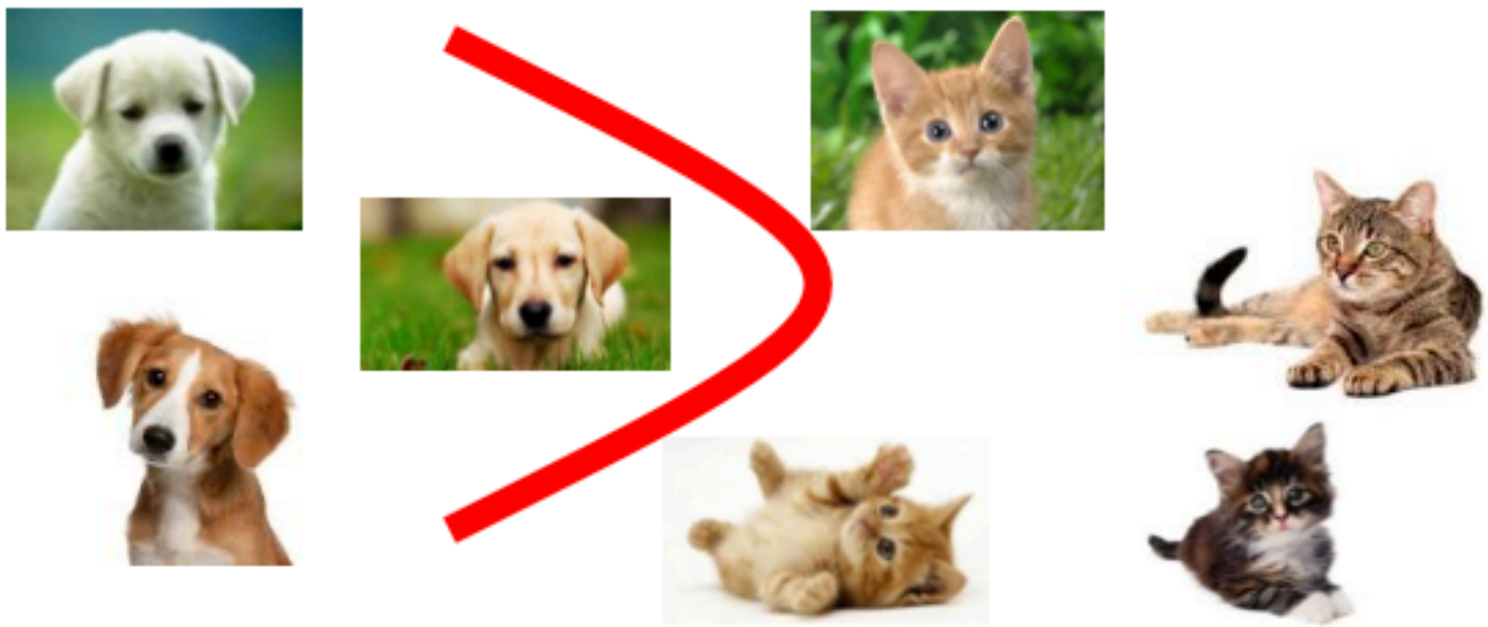
- Problem design
- Data Engineering
- Model Design
- Model Training

I have the key in my hand,
All I need to find is the lock

Supervised Learning

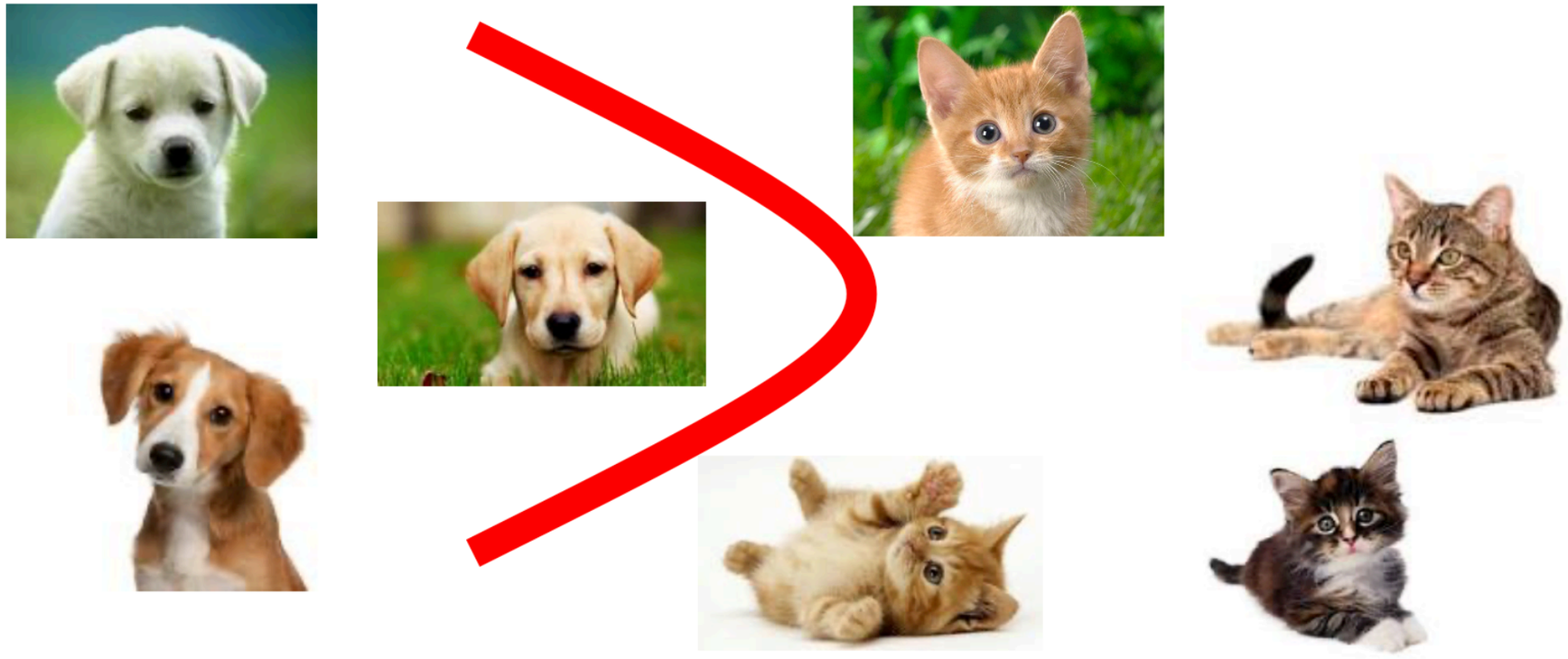
- The goal is to learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ given labeled training samples $(x_i, y_i)_{i=1,\dots,n}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}} .$$



Supervised Learning

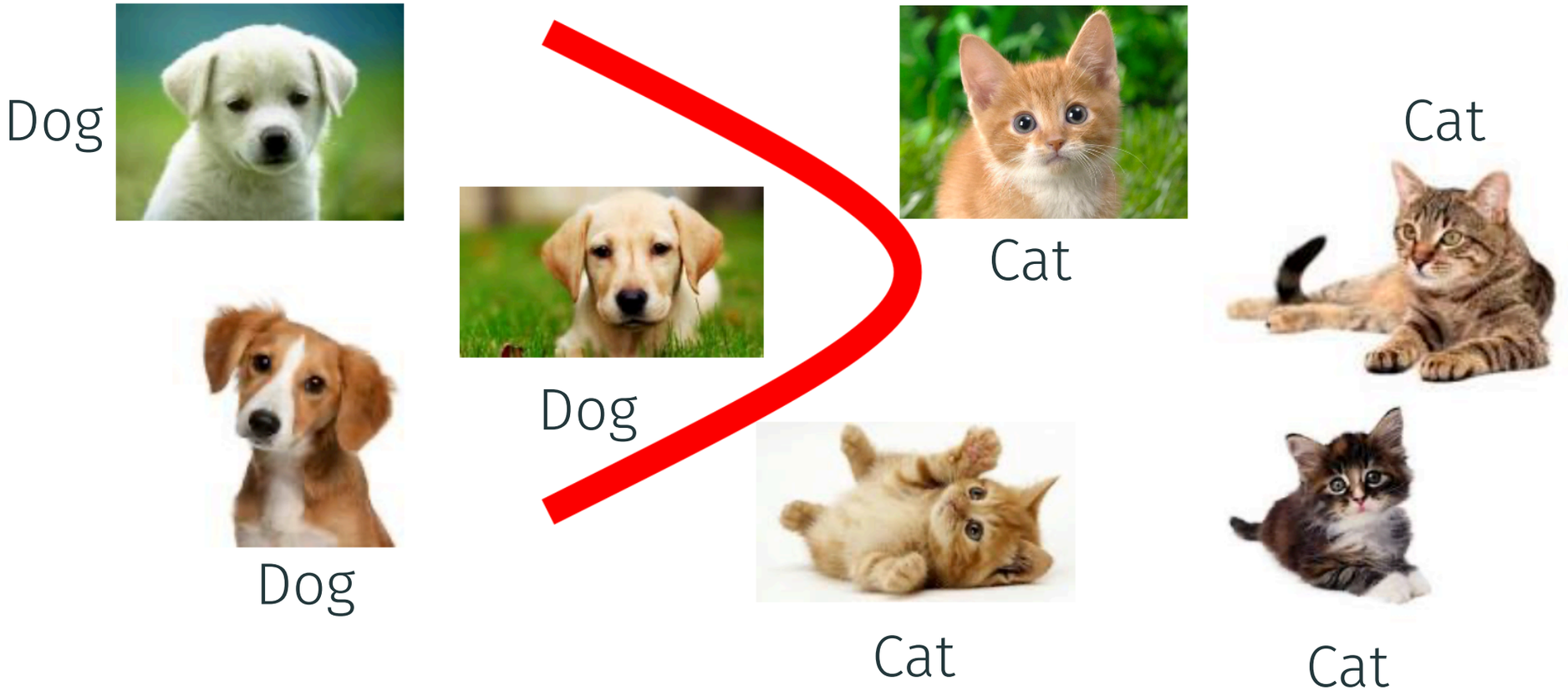
- A lot of the problems we encounter are instances of supervised learning.
- What are the drawbacks?



Supervised Learning

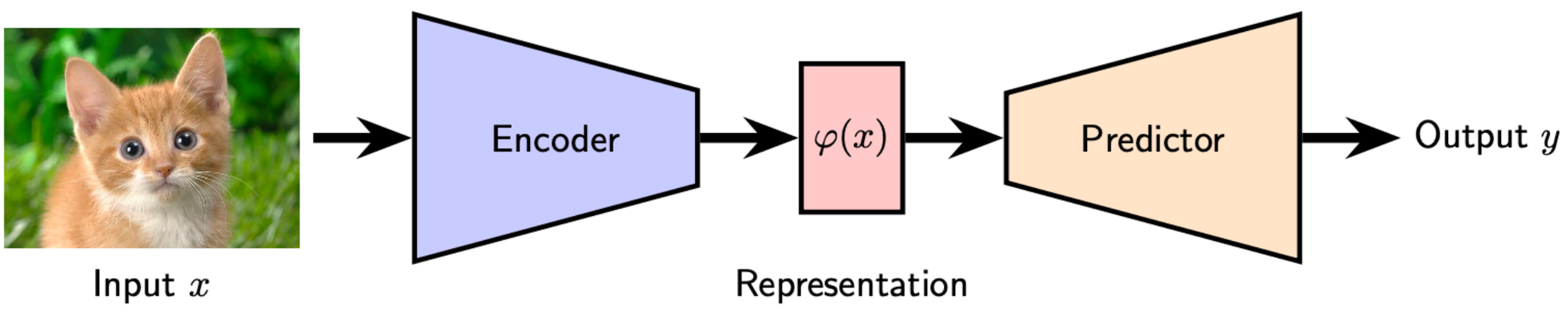
- A lot of the problems we encounter are instances of supervised learning.
- What are the drawbacks?

Need massive amounts of **labeled** data!



Transfer Learning

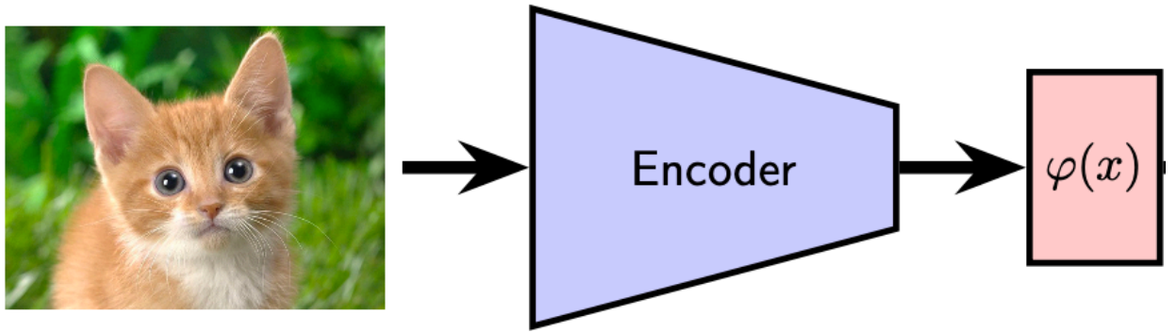
- One of the ways to circumvent this is to re-use existing models



- Often the predictor is simple (a couple of linear layers)
- But the representation is rich: just retrain the predictor
- Many models are available for free and easily accessible

Self-Supervised Learning

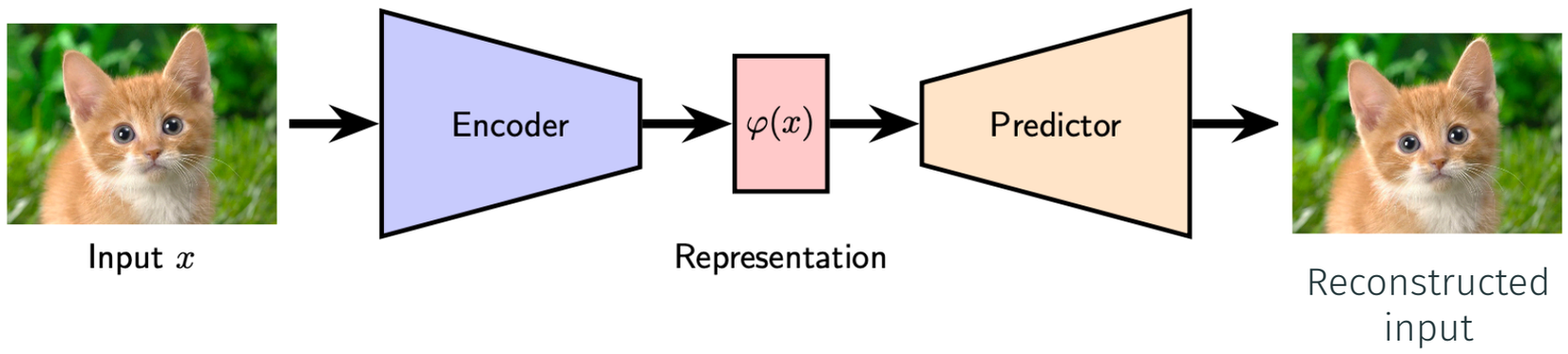
- If you have a lot of un-annotated data
- Try to learn a meaningful representation: self-supervised learning



Design a pretext task

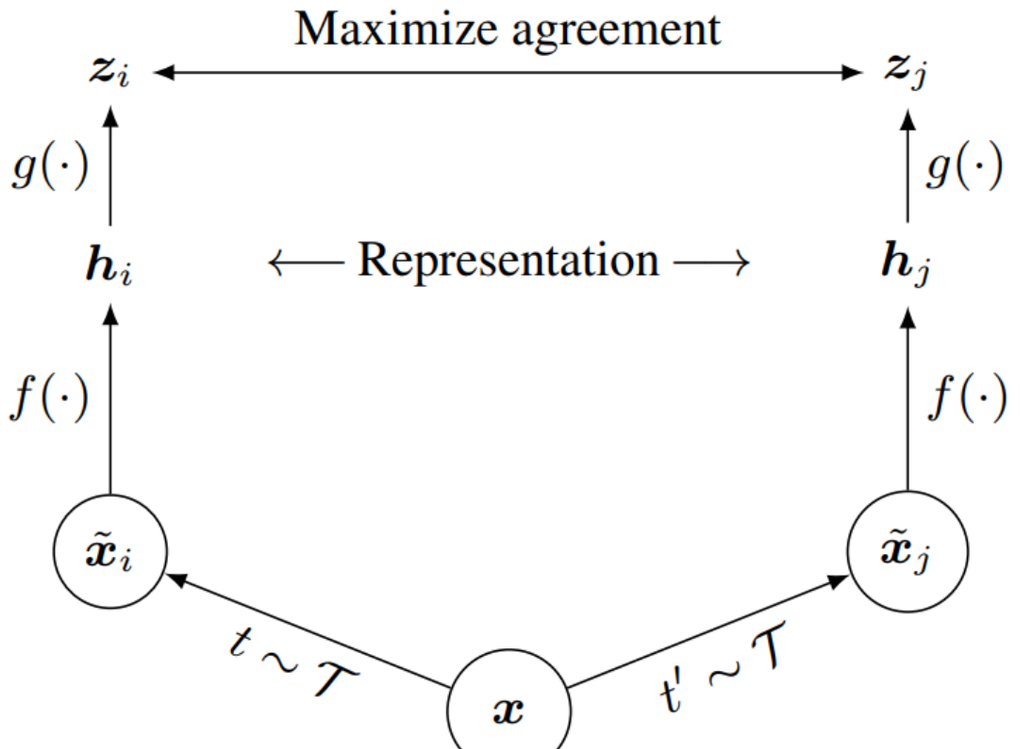
Self-Supervised Learning: Pretext tasks

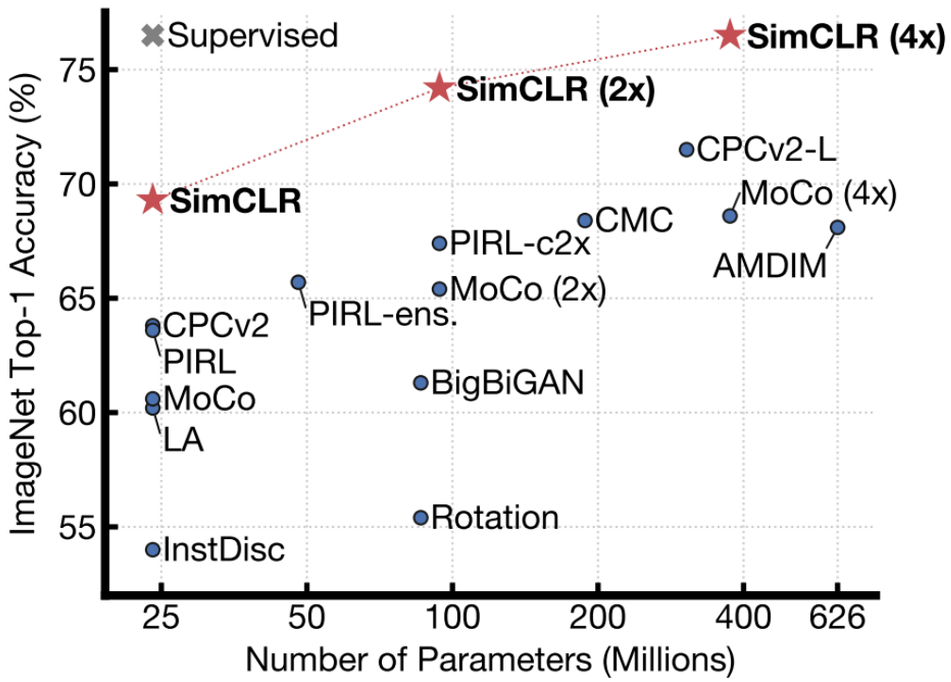
- Reconstruction: AutoEncoders
- Contrastive learning
- Masked Auto-Encoders
- Distillation



Self-Supervised Learning: Pretext tasks

- Reconstruction: AutoEncoders
- Contrastive learning
- Masked Auto-Encoders
- Distillation

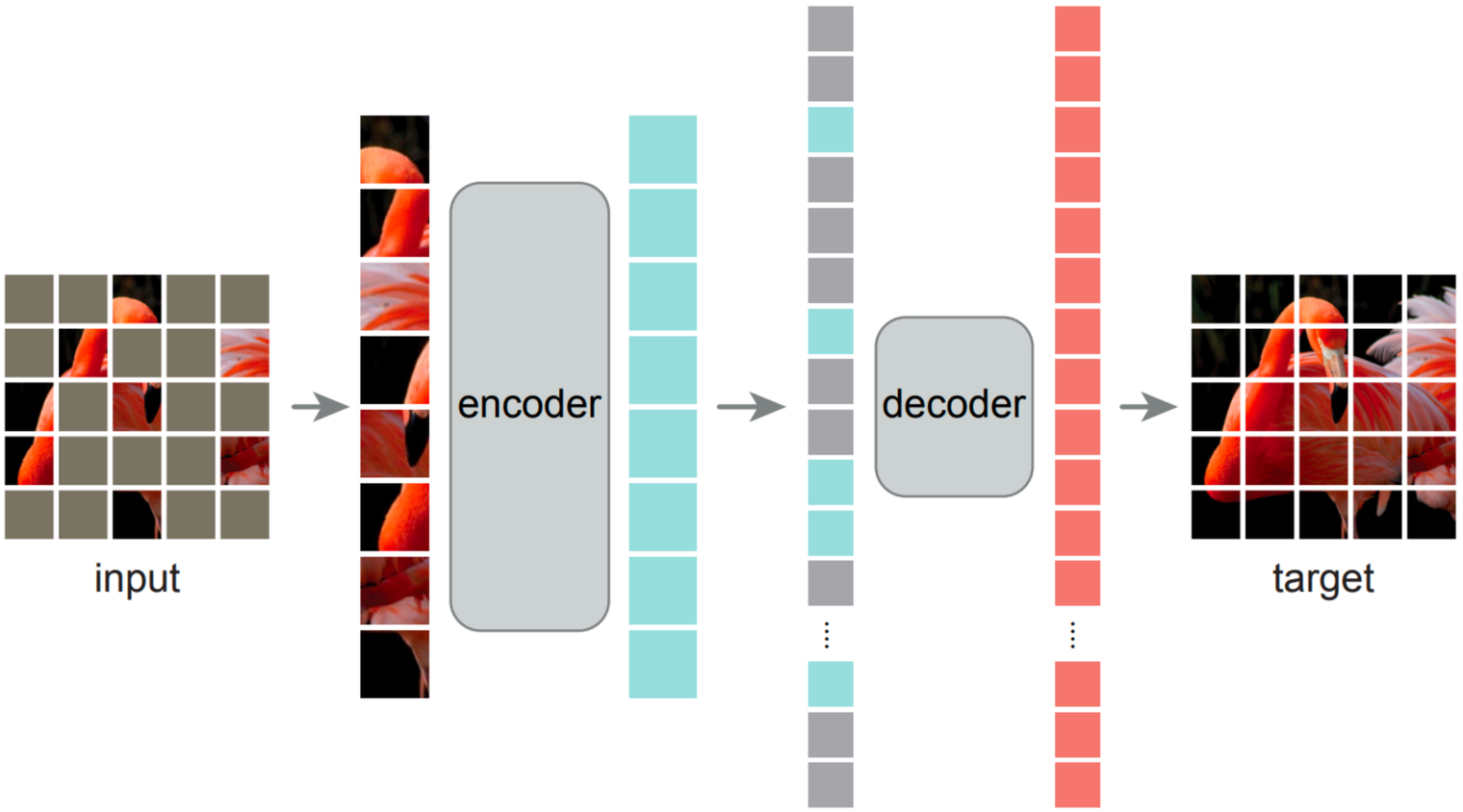




SimCLR (2020)

Self-Supervised Learning: Pretext tasks

- Reconstruction: AutoEncoders
- Contrastive learning
- Masked Auto-Encoders
- Distillation

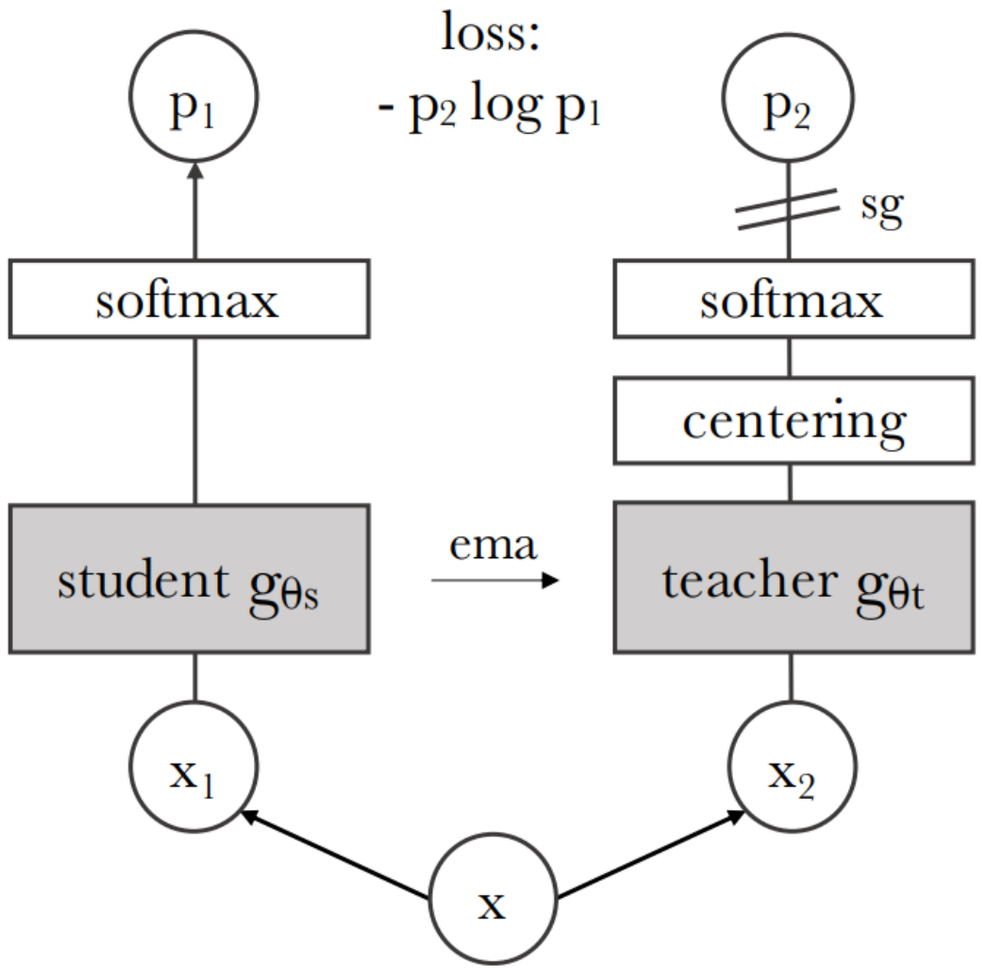


Picture from He et al 2022

Predict next word

Self-Supervised Learning: Pretext tasks

- Reconstruction: AutoEncoders
- Contrastive learning
- Masked Auto-Encoders
- Self-distillation



AlphaZero (2017)

Dino (2021)

Self-Supervised Learning: Pretext tasks

Dino (2021)

Example training times

- It would take 355 years to train GPT-3 on a single NVIDIA Tesla V100 GPU.
- Microsoft (using Azure DCs) built a supercomputer with 10,000 V100 GPUs exclusively for OpenAI.
- Estimated that it cost around $5M in compute time to train GPT-3.
- DINOv2: 3 days of training with 96 A100 GPUs

Recap

Recap

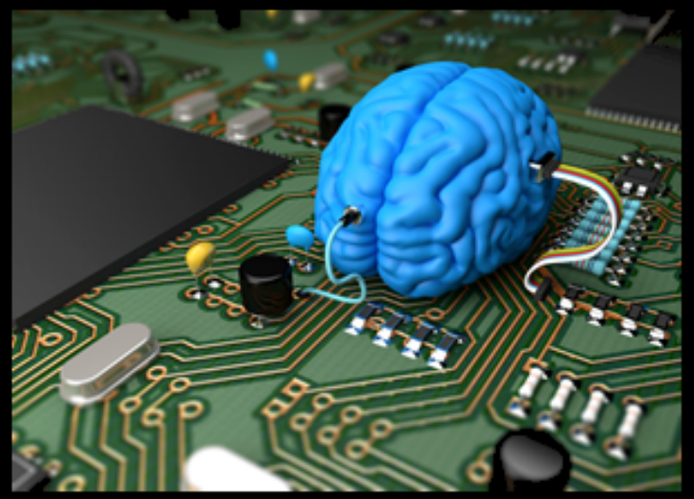
- Based on data type, availability:
  - Define the problem (supervised, self (weakly)-supervised)
  - For complex data: feature engineering.
  - Design data augmentations
  - Model design: MLP, CNN, GNN, Transformer
- Tune the learning rate !
- Not suited to : *little heterogeneous or sparse data. Other approaches may be best*

On to the practical session

Deep Learning

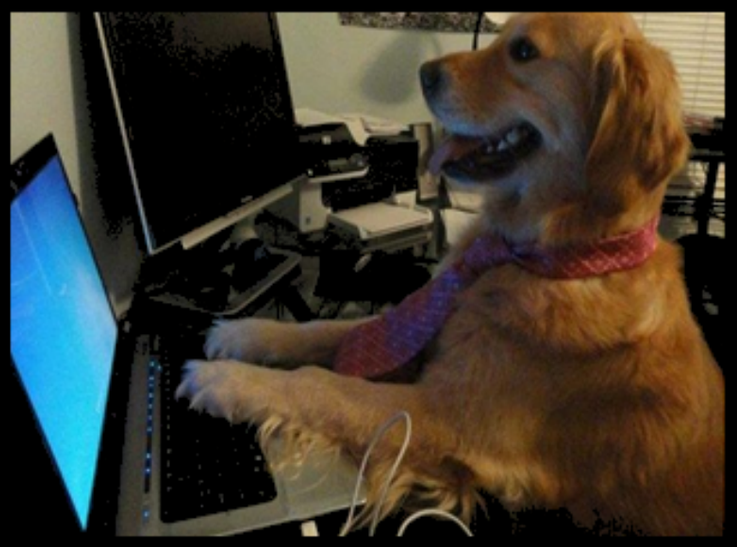
https://www.kdnuggets.com/2017/08/first-steps-learning-deep-learning-image-classification-keras.html

What society thinks I do



What my friends think I do

What other computer scientists think I do



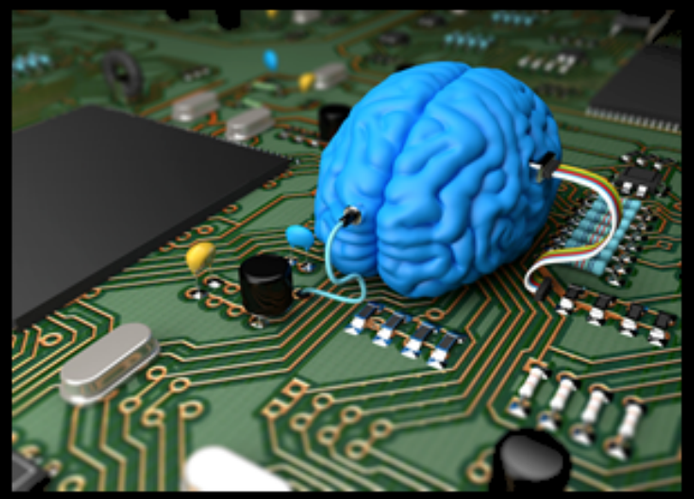
What mathematicians think I do

What I think I do

On to the practical session

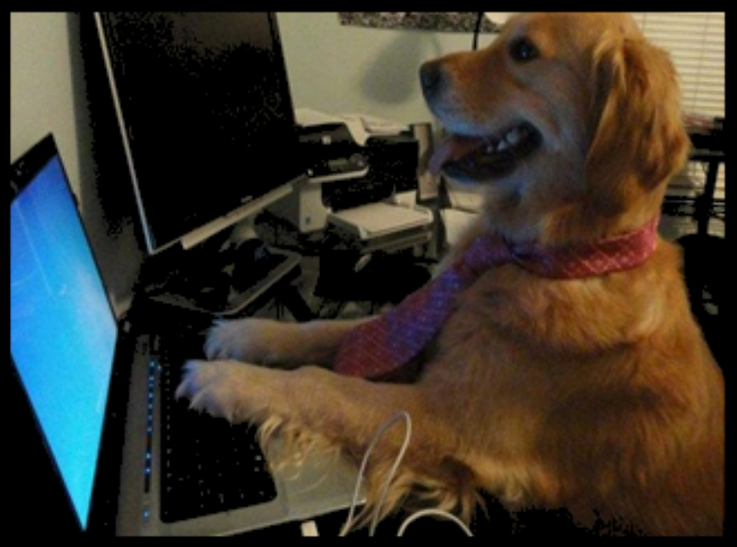
Deep Learning

What society thinks I do



What my friends think I do

What other computer scientists think I do



What mathematicians think I do

What I think I do

```
In [1]:
import keras
Using TensorFlow backend.
```

What I actually do

Thank you for listening!

Recommended reading

- deeplearningbook.org: Math and main concepts
- Francois Chollet's book: Keras programming
- Aurélien Géron's book: Generic Machine Learning with Scikit-learn and Deep Learning with TF/Keras

Backpropagation

TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES

View more or edit prompt ↓

TEXT PROMPT

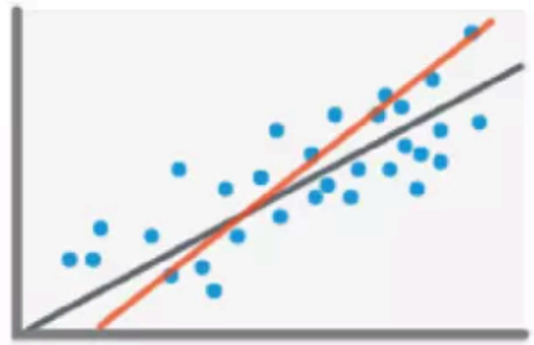
a store front that has the word ‘openai’ written on it [...]

AI-GENERATED IMAGES

View more or edit prompt ↓

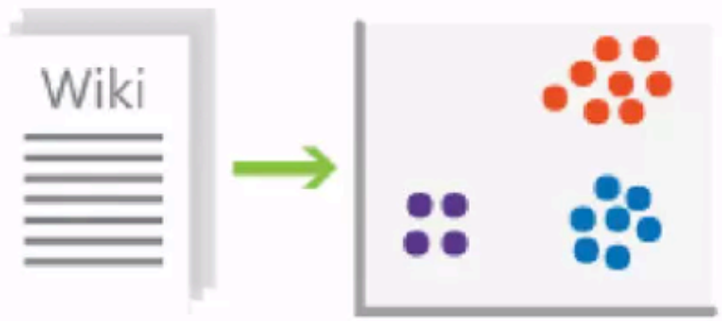
What is Deep Learning ?

Classification
(supervised – predictive)

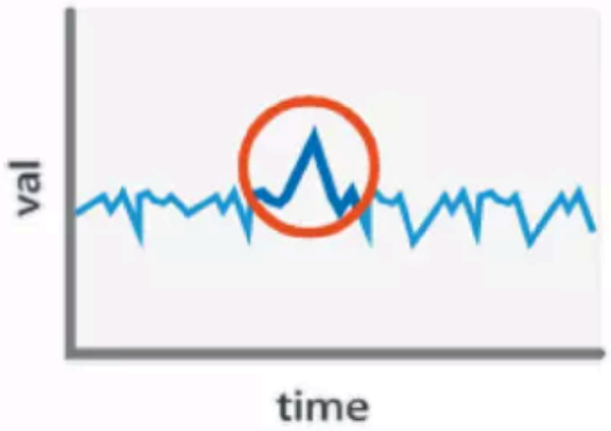


Regression
(supervised – predictive)

https://www.slideshare.net/slideshow/deep-learning-a-visual-introduction/55857150#9



Clustering
(unsupervised – descriptive)



Anomaly Detection
(unsupervised– descriptive)

In sciences

Stanford | News

Search Stanford news...

Home Find Stories For Journalists Contact

JANUARY 25, 2017

Deep learning algorithm does as well as dermatologists in identifying skin cancer

In hopes of creating better access to medical care, Stanford researchers have trained an algorithm to diagnose skin cancer.

BY TAYLOR KUBOTA

It's scary enough making a doctor's appointment to see if a strange mole could be cancerous. Imagine, then, that you were in that situation while also living far away from the nearest doctor, unable to take time off work and unsure you had the money to cover the cost of the visit. In a scenario like this, an option to receive a diagnosis through your smartphone could be lifesaving.

Universal access to health care was on the minds of computer scientists at Stanford when they set out to create an artificially intelligent diagnosis algorithm for skin cancer. They made a database of nearly 130,000 skin disease images and trained their algorithm to visually diagnose potential cancer. From the very first test, it performed with inspiring accuracy.