DEEP LEARNING FOR GENOMICS

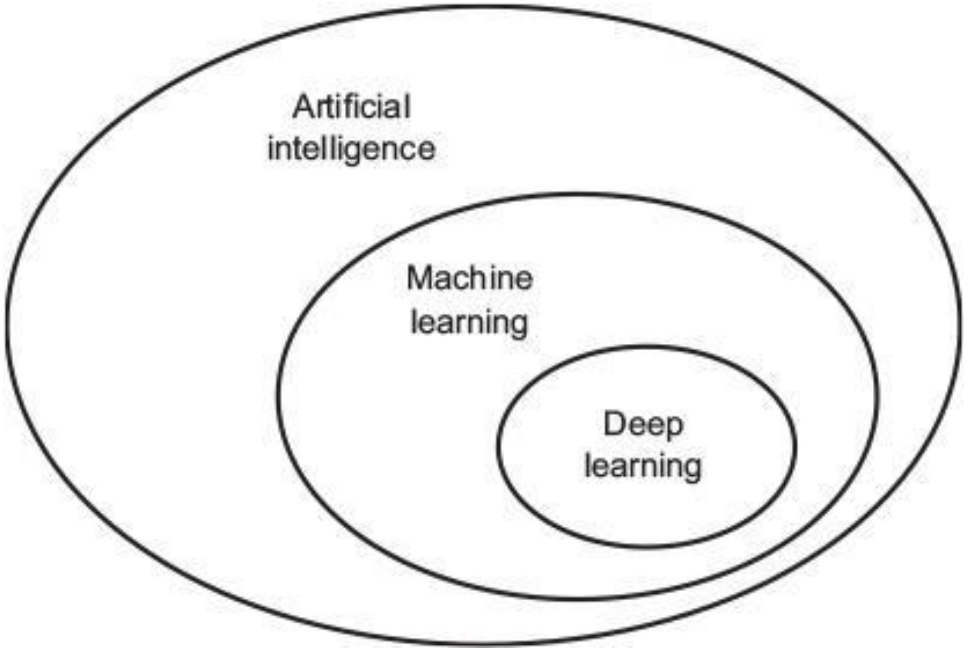
Raphaël MOURAD, Associate Prof.
MIAT INRAe
Université Paul Sabatier, Toulouse III

INTRODUCTION TO DEEP LEARNING

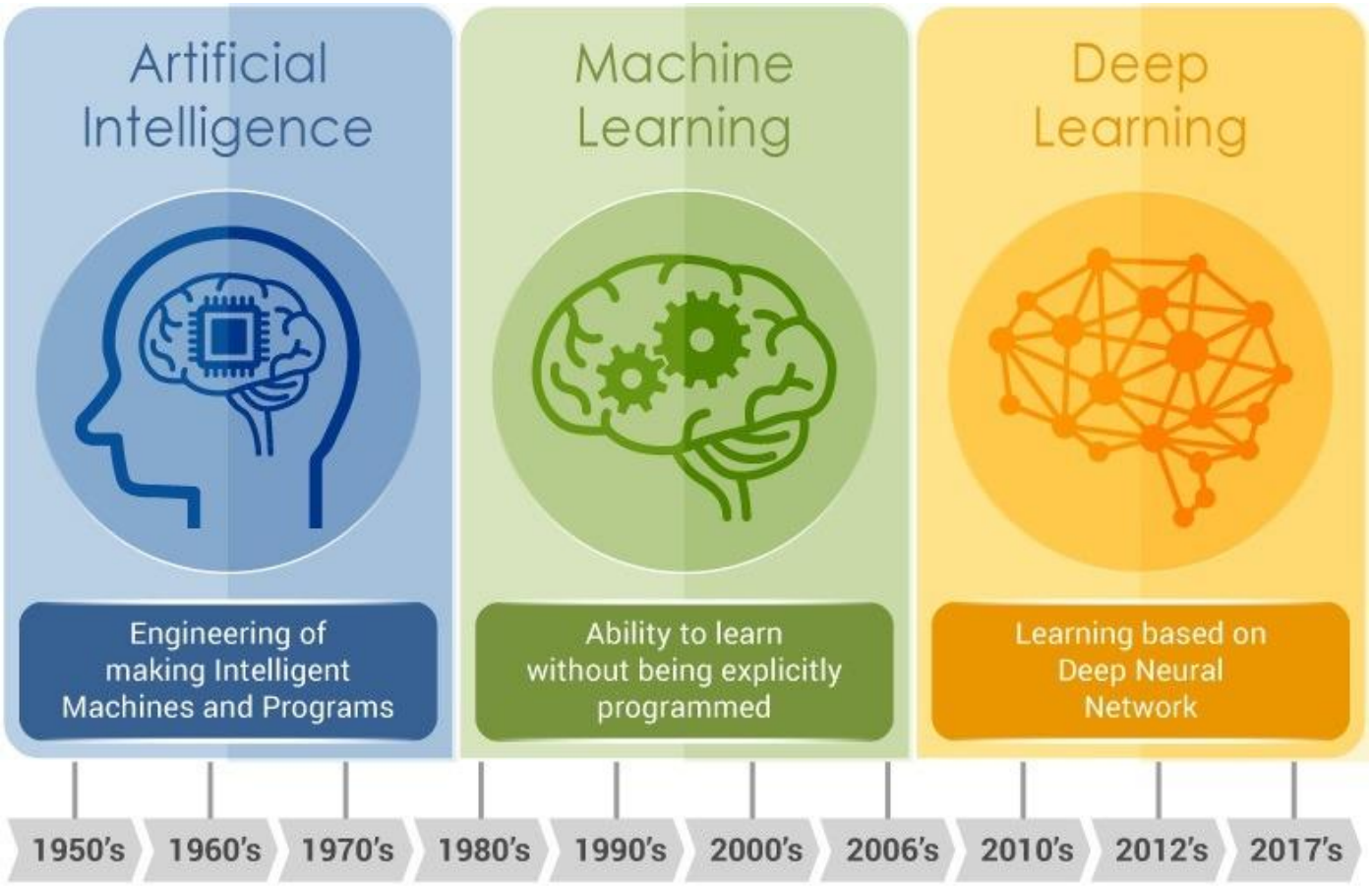
Deep learning as a branch of AI

- Deep learning is a branch of machine learning and AI, which has been very successful in the past years (since 2012).

- Deep learning relies on:
  - new algorithms,
  - new GPUs
  - access to big data.

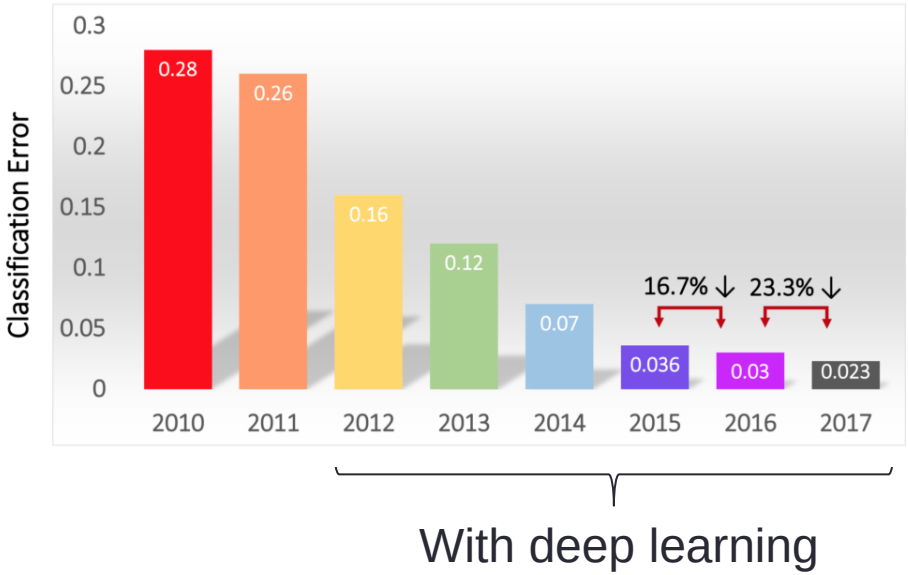


Deep learning as a branch of AI



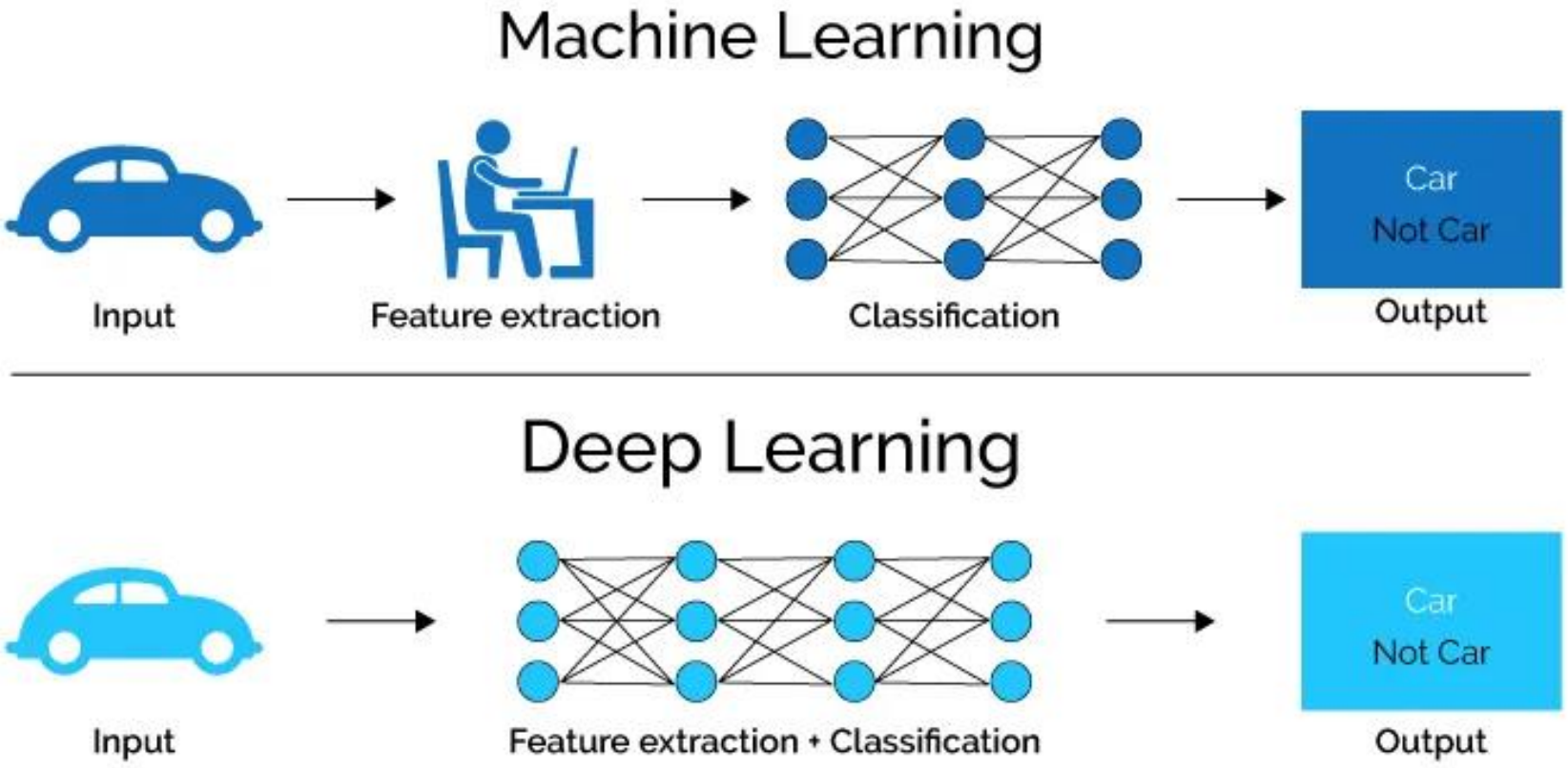
Success of deep learning since 2012: Example of computer vision

Image classification
(ImageNet challenge)

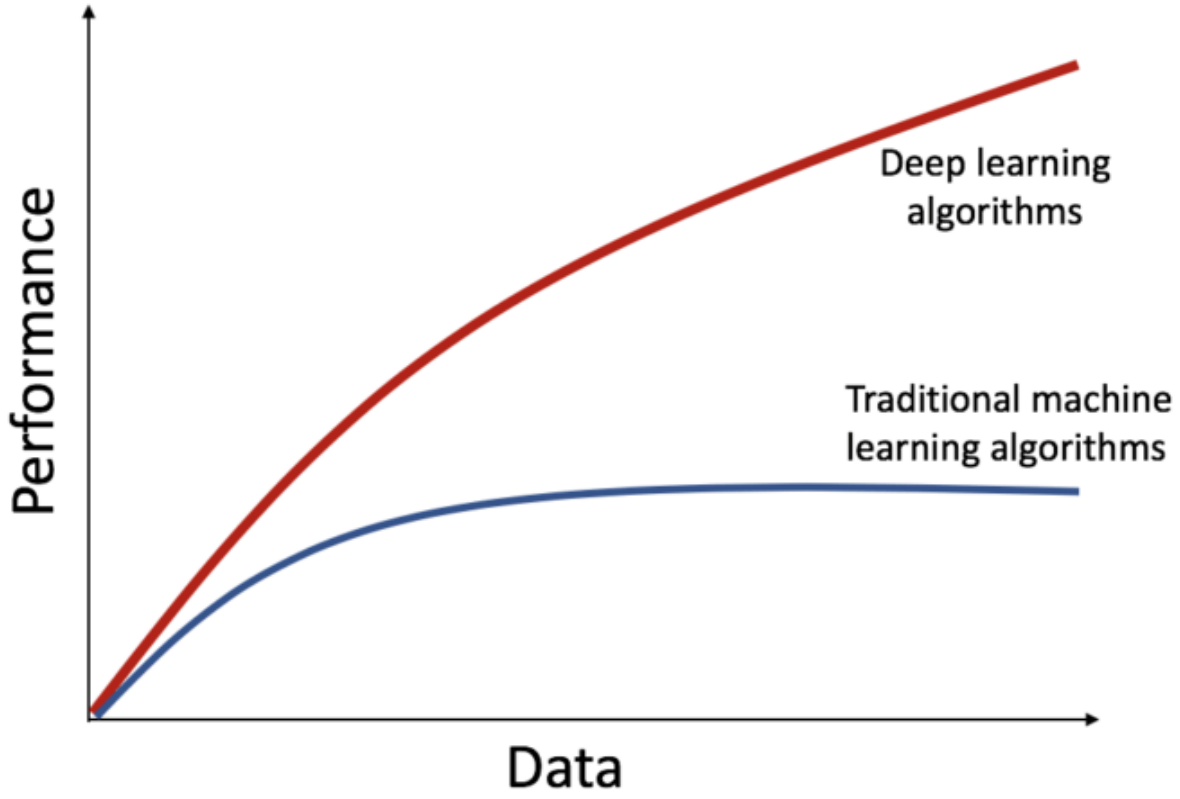


- 2012: AlexNet (convNet)
- 2013: ZFNet
- 2014:
  - VGGNet (deeper, simpler)
  - InceptionNet (faster)
- 2015: ResNet (deeper)
- 2016: Ensemble networks

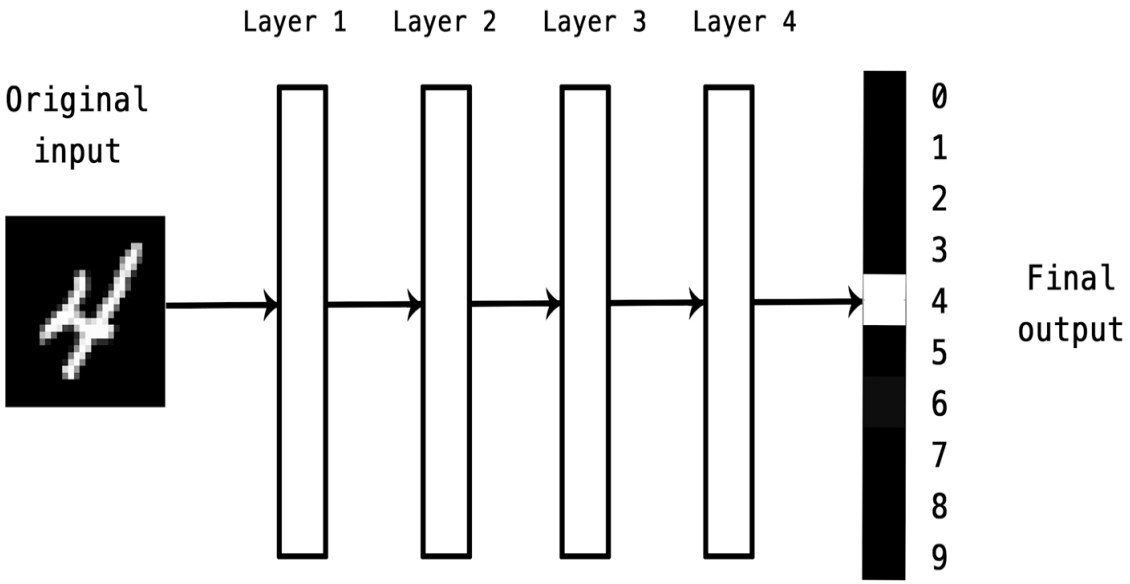
Difference between machine and deep learning



Difference between machine and deep learning

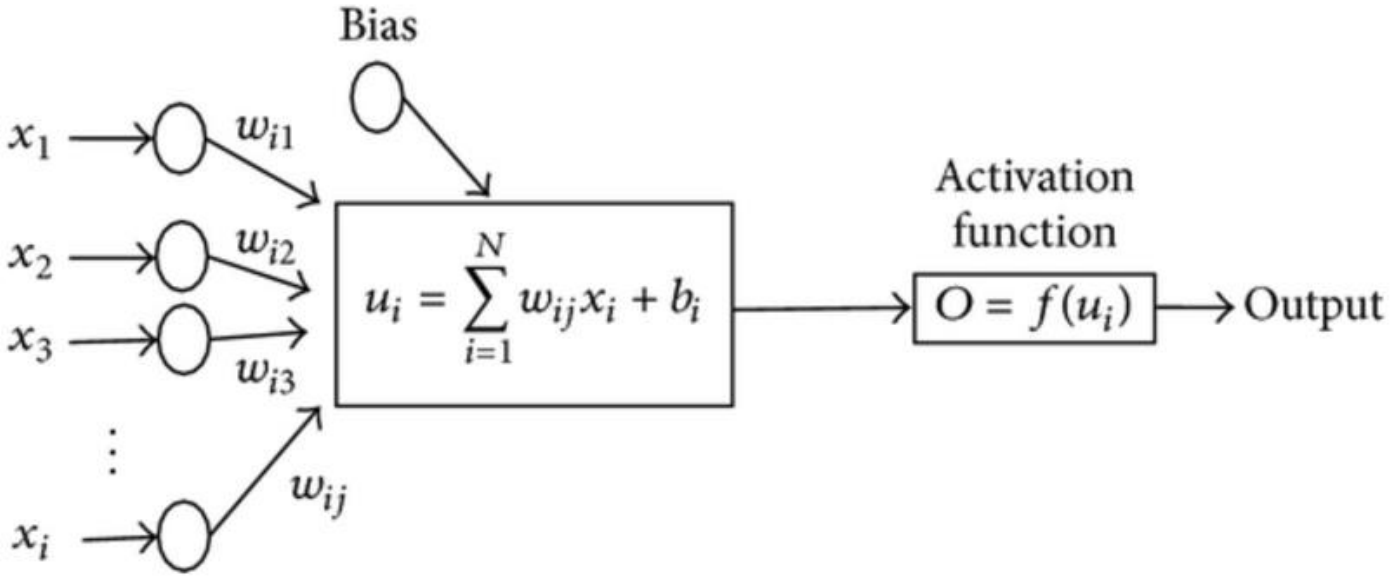


Deep learning as neural networks



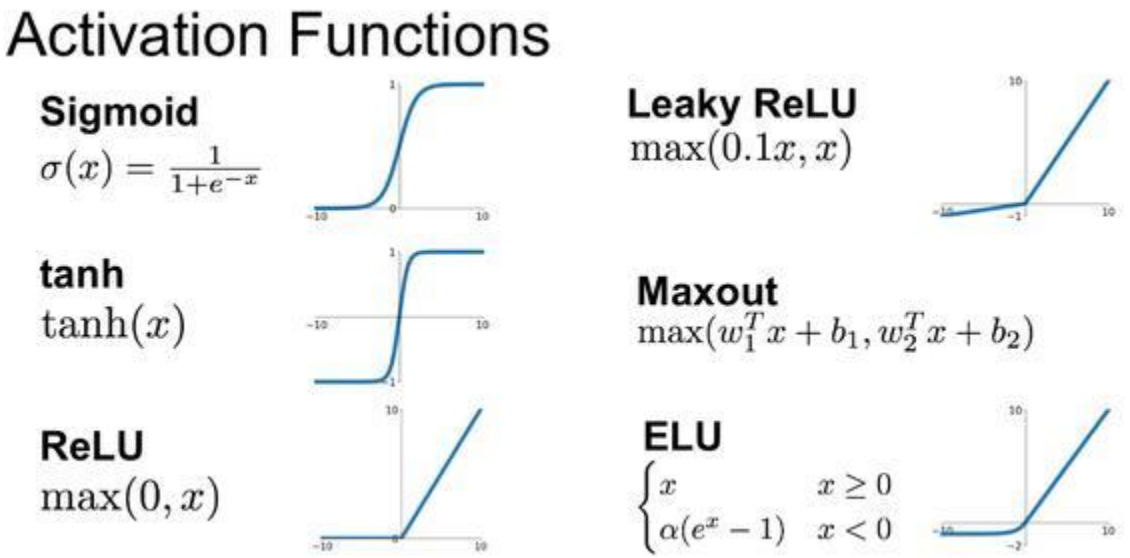
- Deep learning is based on a **deep** neural network which is the stacking of different neuronal layers to predict a final output.

Neural networks



- In a neural network, multiple inputs x_i are combined through a linear combination (with weights w_i), and then an activation function is used for a non-linear transformation to obtain the output.

Activation function



- The activation function allows to obtain a non-linear output from a linear input.
- NB: the linear activation function also exists (A = cx).

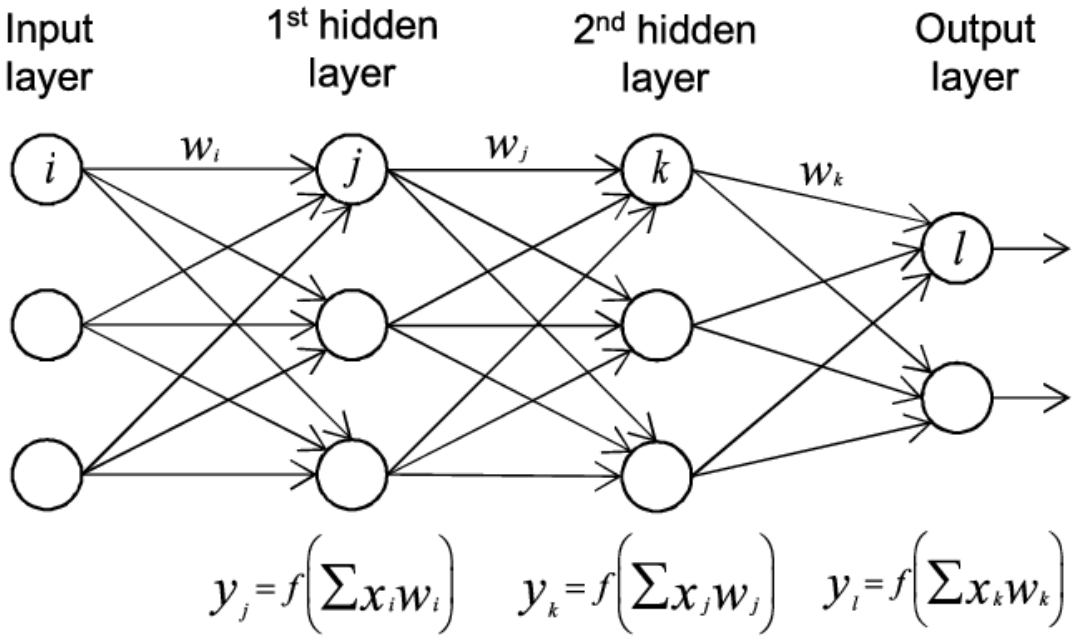
Activation function

- For regression with counts (0, 1, 2, …, n):

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x))$$

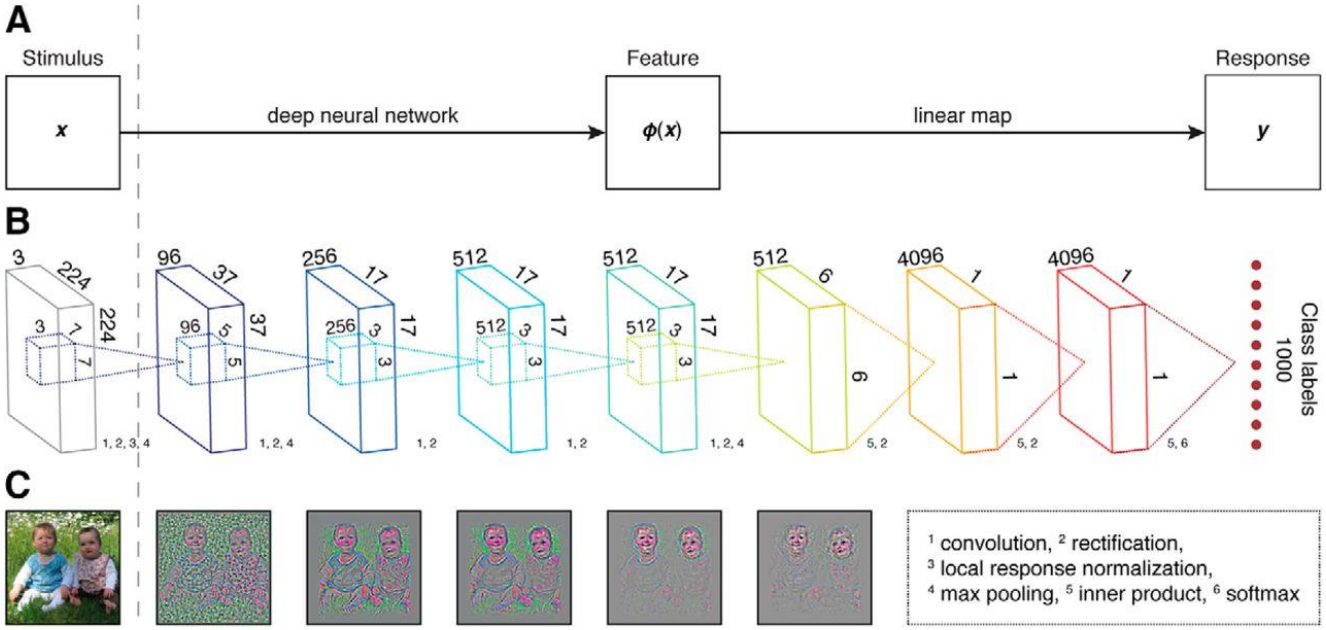
- By default: β = 1

Deep neural networks



- A deep neural network (DNN) is an neural network (NN) with multiple layers between the input and output layers. Each hidden layer linearly combines the output from the previous layer and then does a non-linear transformation.

Principle of deep learning : stacking many different sorts of layers



- Building a deep neural network is like assembling a lego toy where every lego brick is a layer.

Loss functions

- For regression (continuous outputs):
  - Mean Square Error:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

  - Mean Absolute Error:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{x}_i - \boldsymbol{x}|$$

Loss functions

- For regression with counts (0, 1, 2, …, n):
  - Poisson loss:

$$\text{Loss} = \frac{1}{n} \sum_{i=1}^{n} (\tilde{y}_i - y_i \log(\tilde{y}_i))$$

Loss functions

- For classification (categorical outputs):
  - Cross entropy:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Gradient descent

- The loss function to minimize:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

Where w are the model parameters and n the number of samples.

- Batch gradient descent:

$$w := w - \eta \nabla Q(w) = w - \frac{\eta}{n}\sum_{i=1}^{n} \nabla Q_i(w),$$

where η is a step size (sometimes called the *learning rate* in machine learning).

Other gradient descents

- Stochastic gradient descent:

$$w := w - \eta \nabla Q_i(w).$$

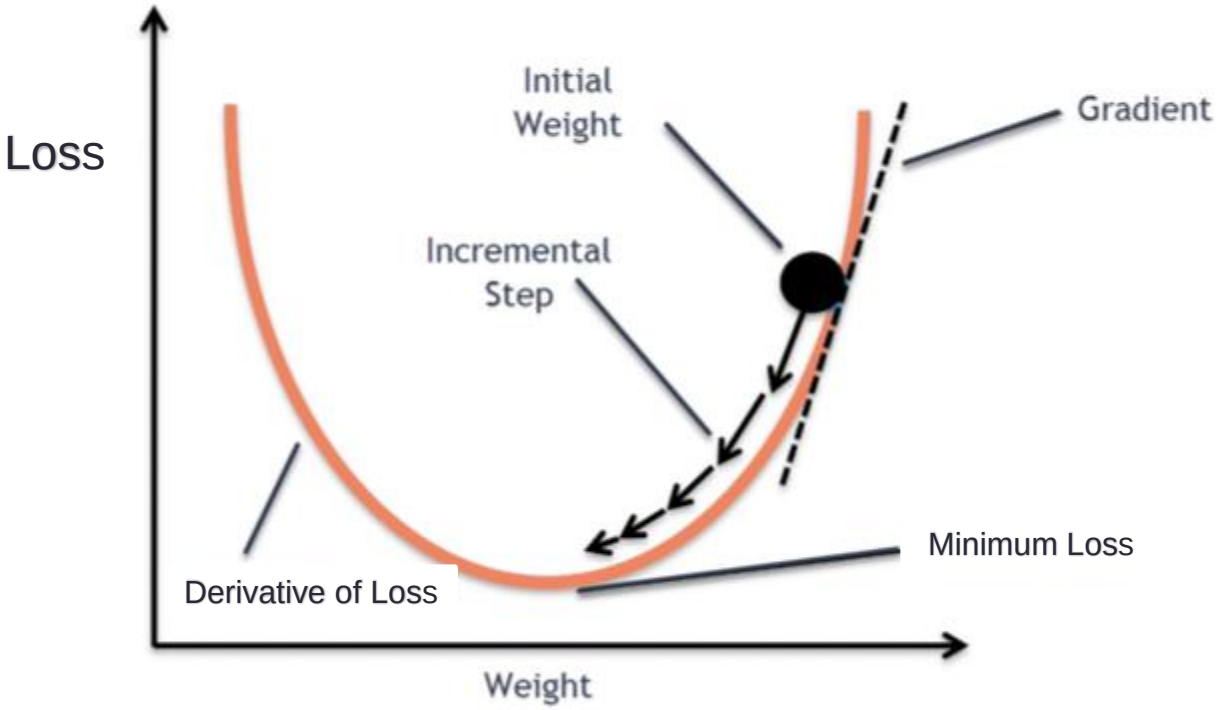
At each step, the parameters are updated by the gradient of only one sample randomly drawn.

- Mini-batch gradient descent (most used):

Same as stochastic gradient descent except that the parameters are updated by the gradient of a number x of samples (usually 32, 64 or 128 samples).

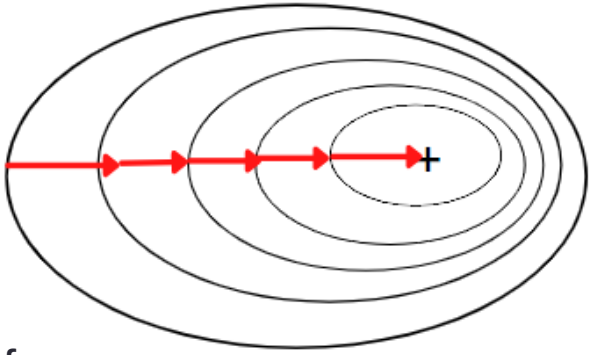
=> The most used approach!

Illustration of gradient descent



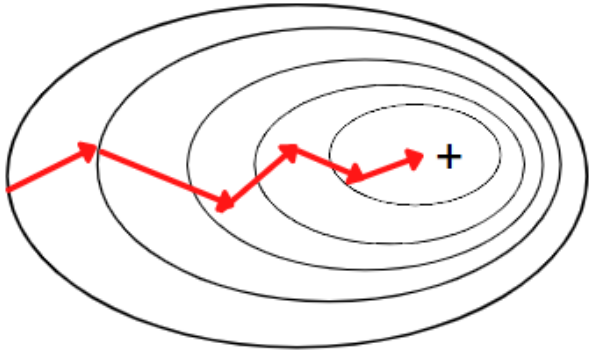
Comparison of different gradient descents

Batch Gradient Descent



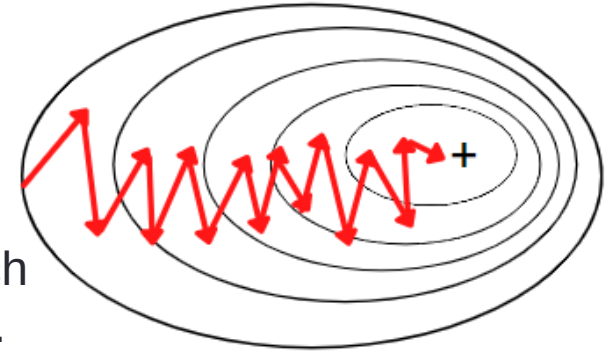
Good for convex problem. But here it gets trapped in local minimina easily.

Mini-Batch Gradient Descent



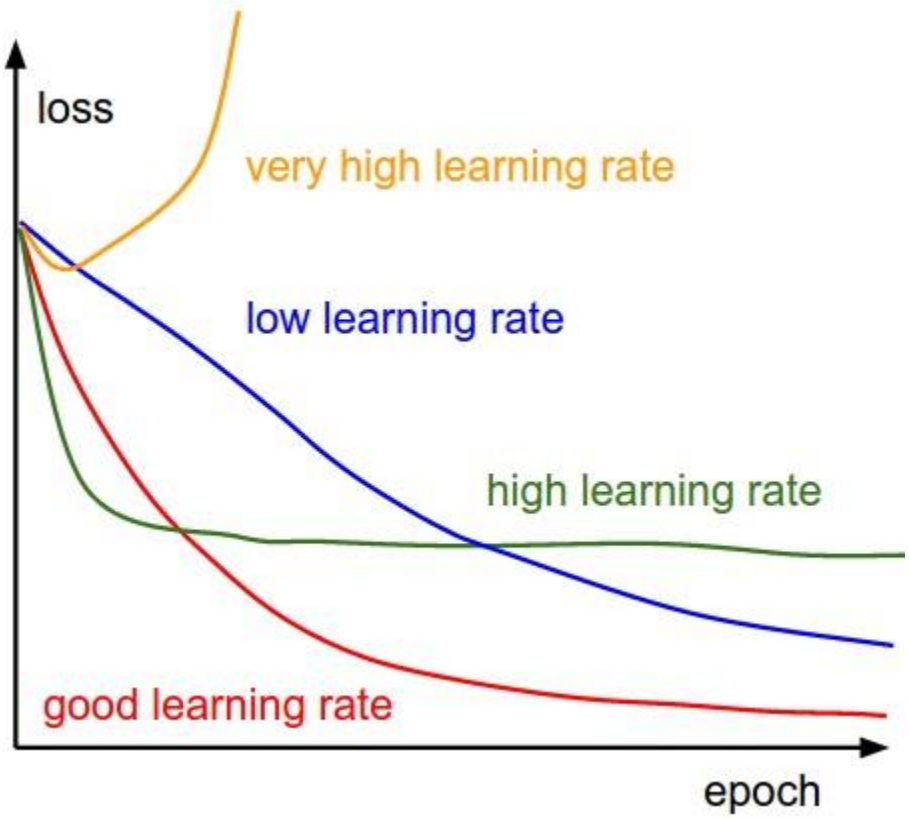
Good trade-off.

Stochastic Gradient Descent



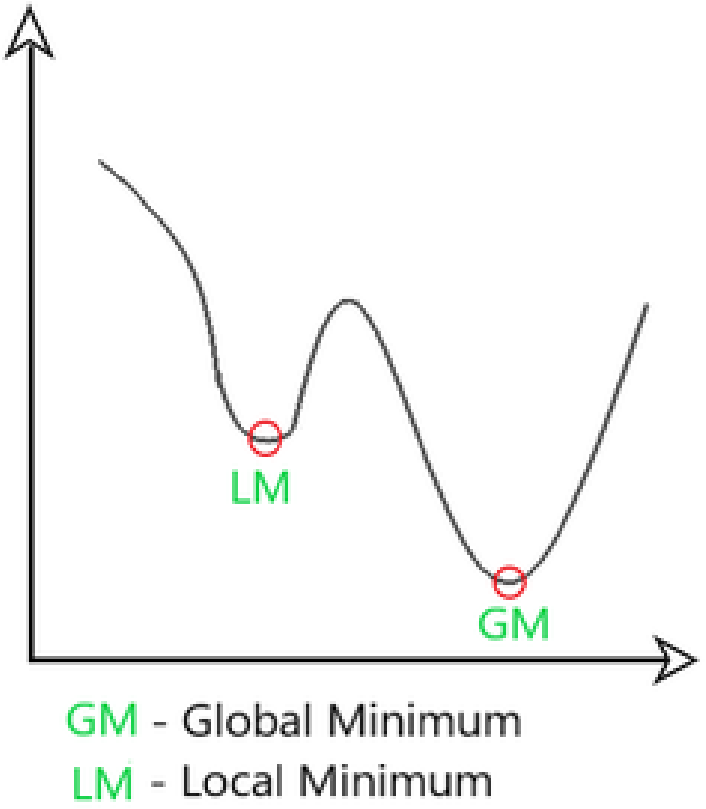
Too noisy. Don’t reach local minimina easily.

Careful choice of the learning rate η

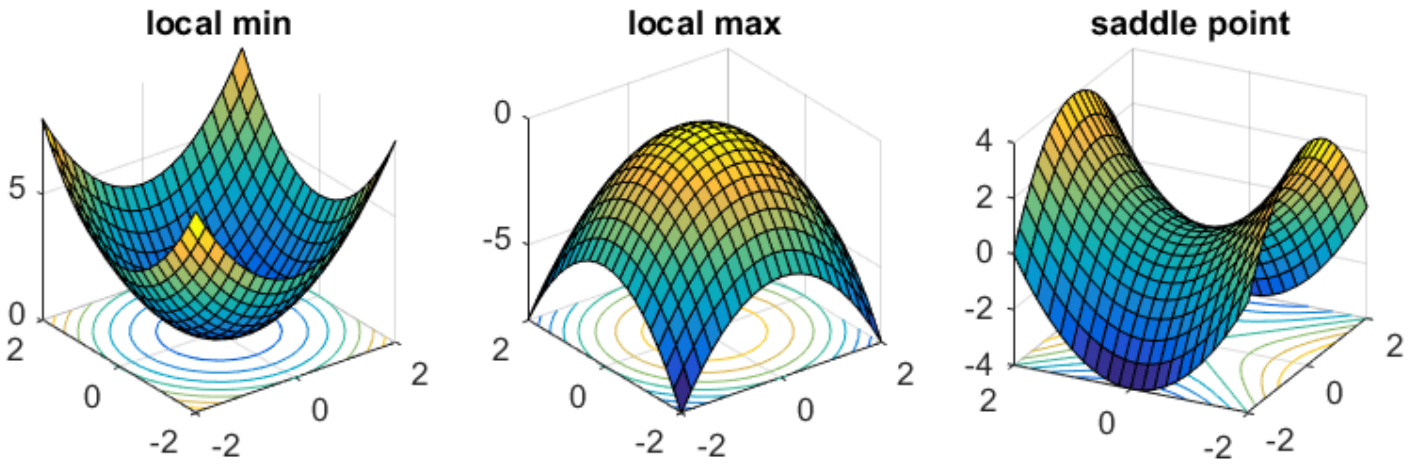


Difference between global and local minimum

- As the loss function is not convex, there is no guarantee to identify the global minimum (best parameters).



Saddle point



- An optimizer gets often trapped into a saddle point.
- The noise during training with stochastic gradient descent helps to escape from the saddle point in practice.

Use of momentum during learning

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment — learning rate — weight gradient

Without momentum

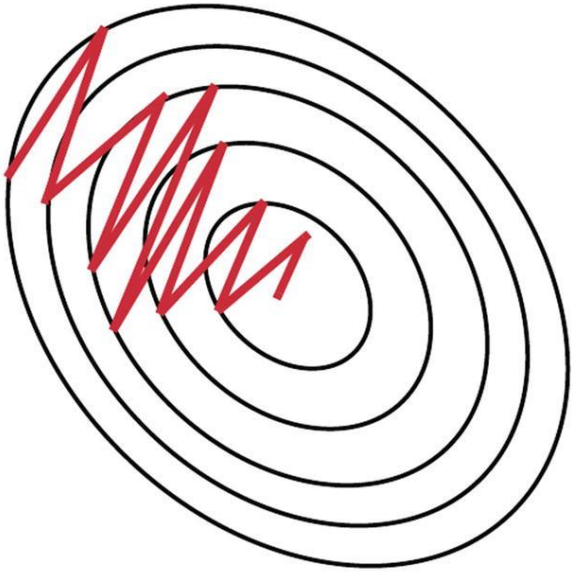
$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor — weight increment, previous iteration

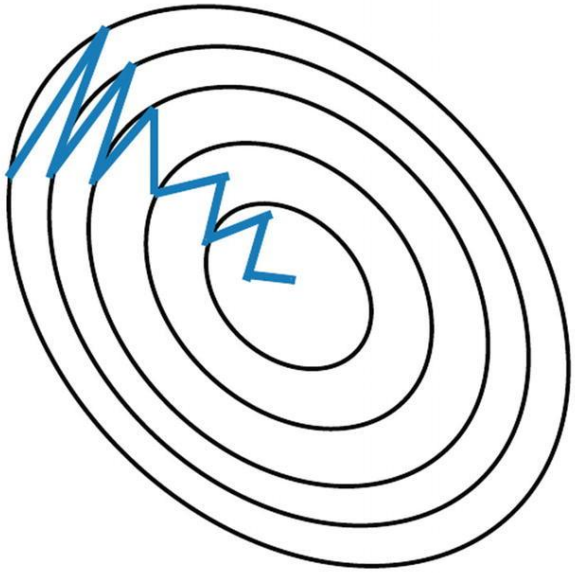
With momentum

E : loss

Using momentum enable faster convergence



Stochastic Gradient Descent **without** Momentum



Stochastic Gradient Descent **with** Momentum

ADAM optimizer

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*

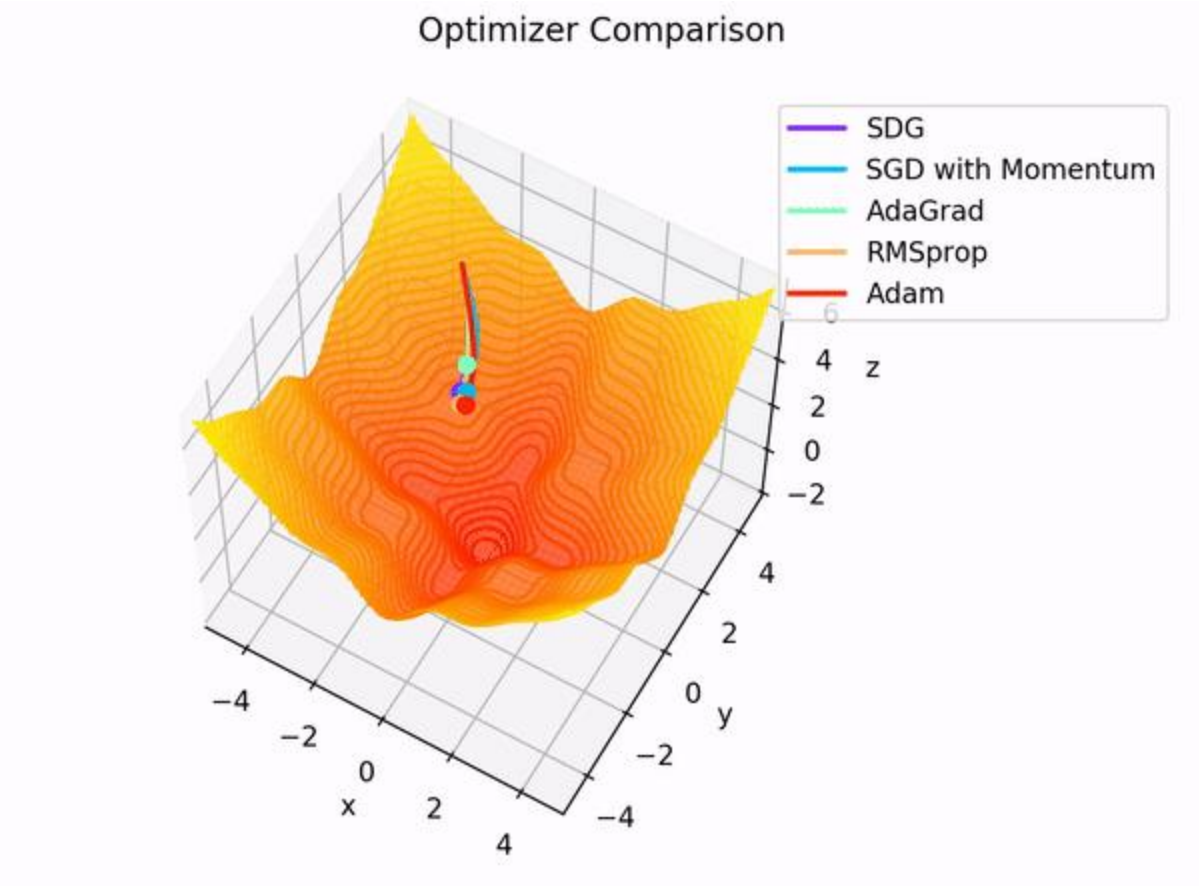
g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

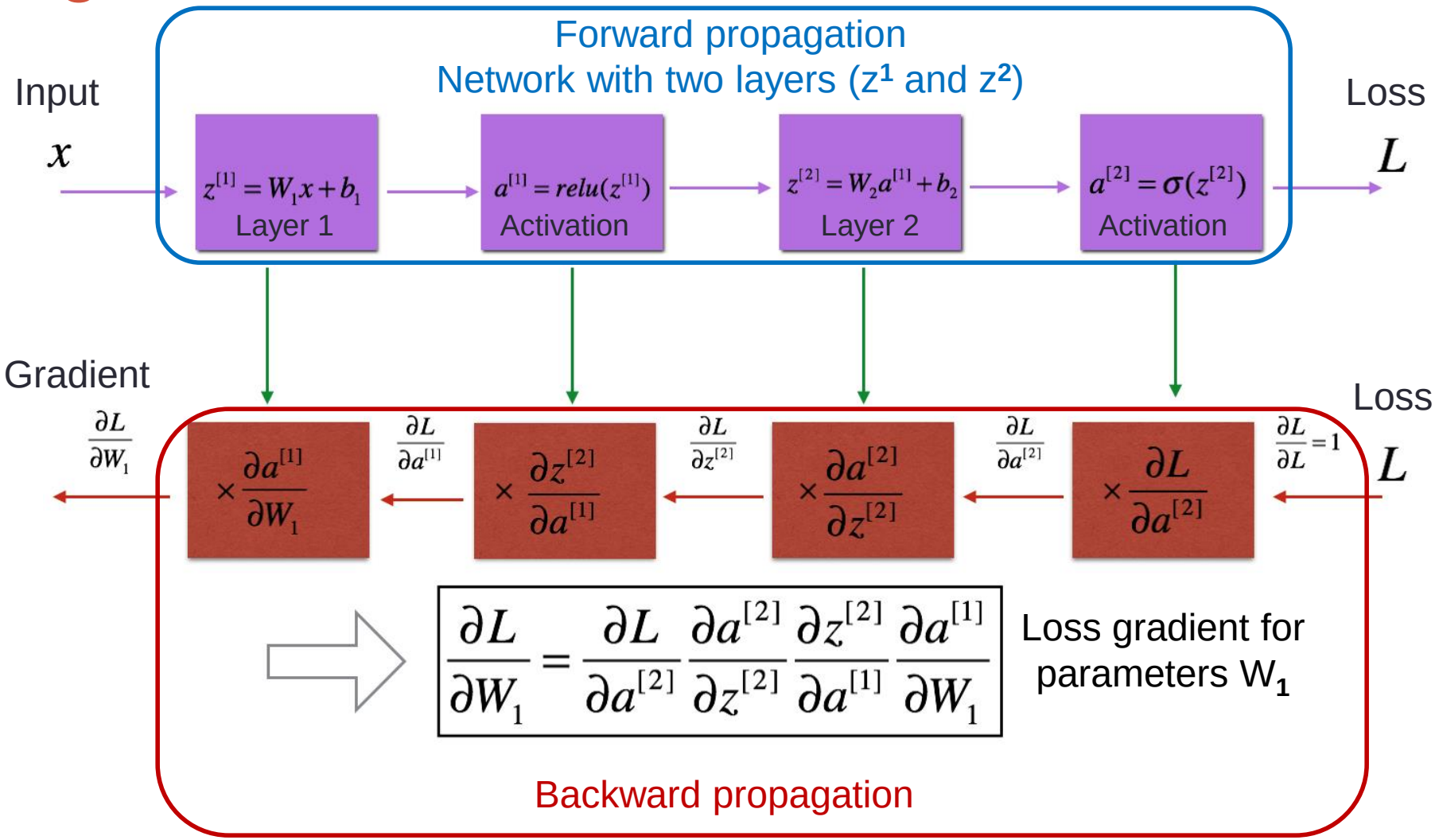
s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

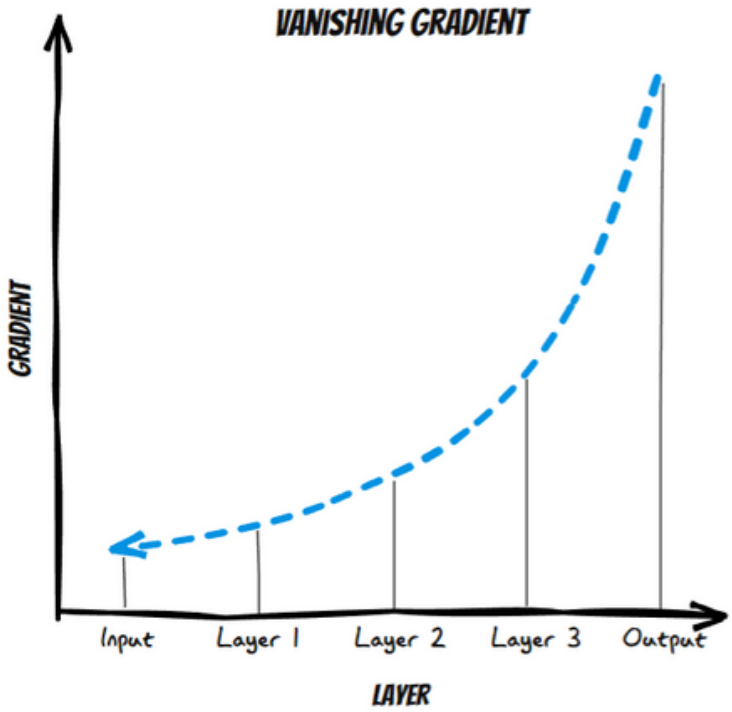
ADAM optimizer

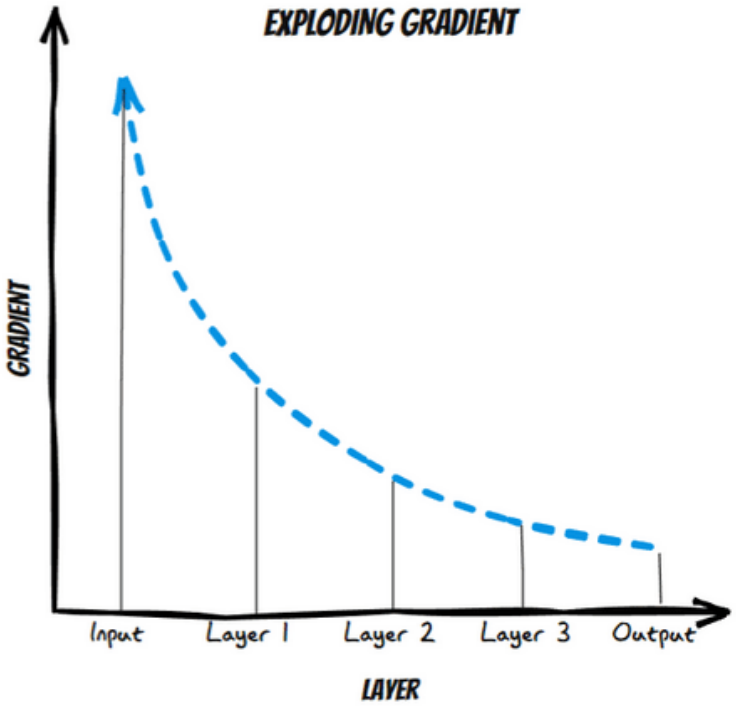


Backward propagation to compute the gradient



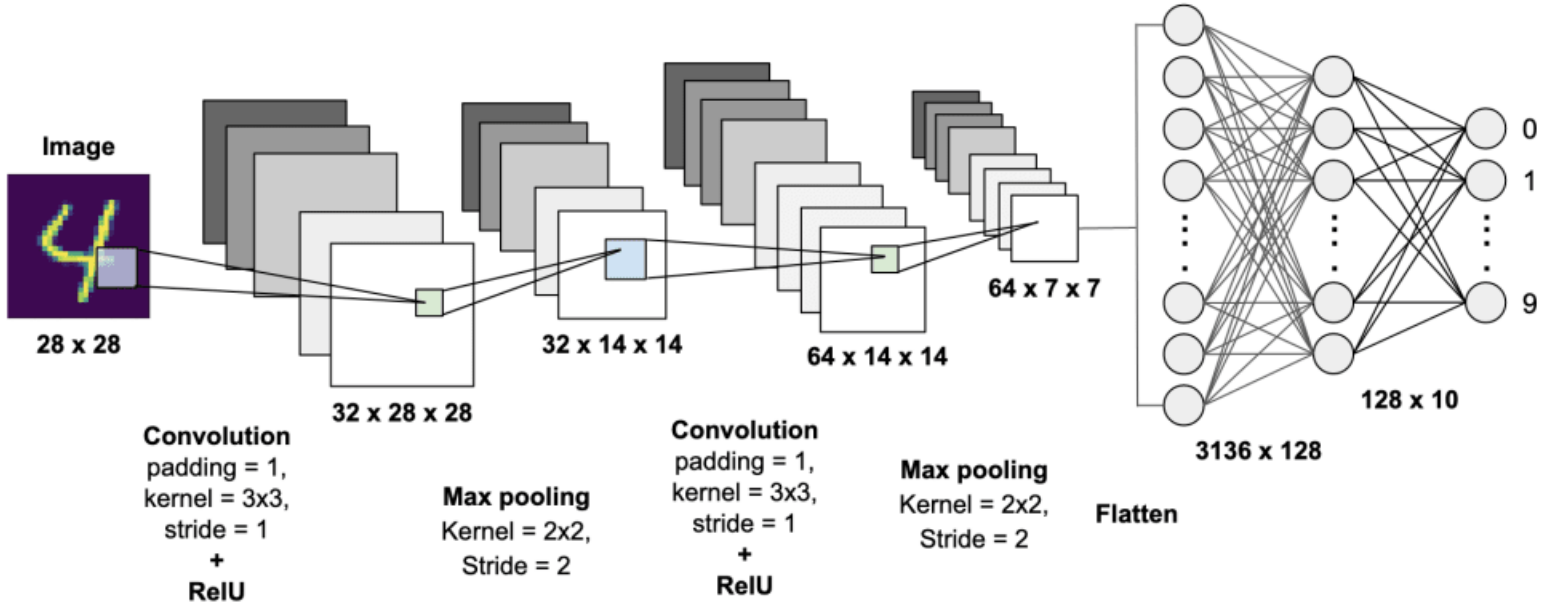
Problems related with deep neural network





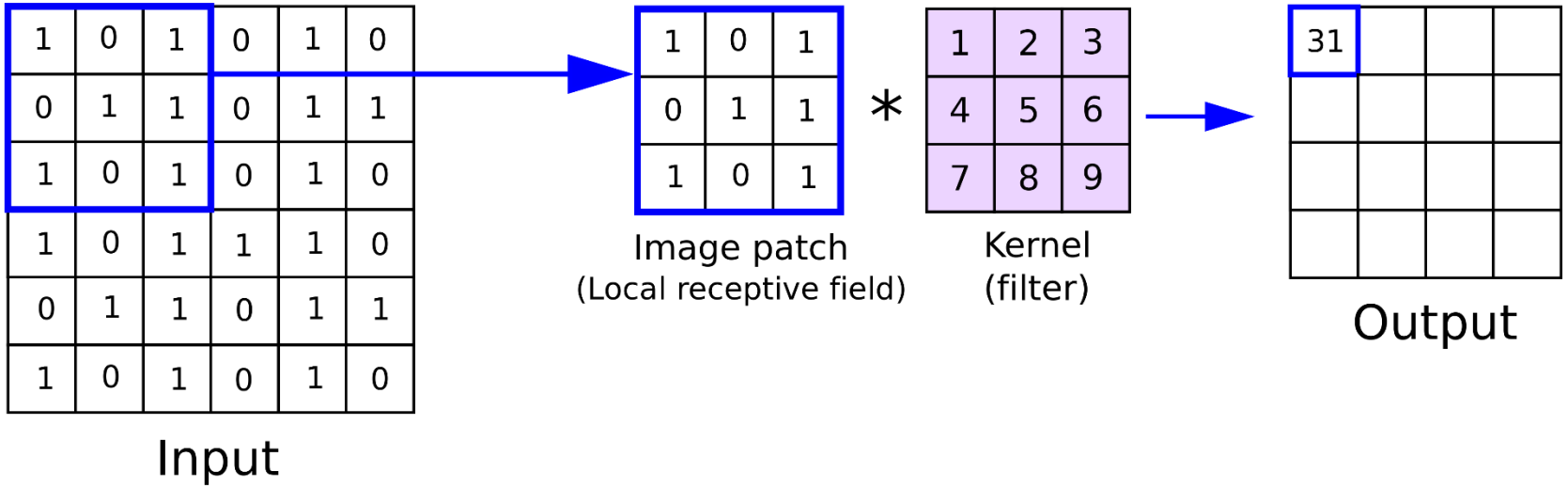
- The multiplication of a very large number of terms (in a deep network) will either lead to a gradient with values close to zero (vanishing gradient) or to a gradient with values toward infinite (exploding gradient).
- Problem: very small or large values cannot be handled by a computer.

Convolutional neural networks (CNNs)



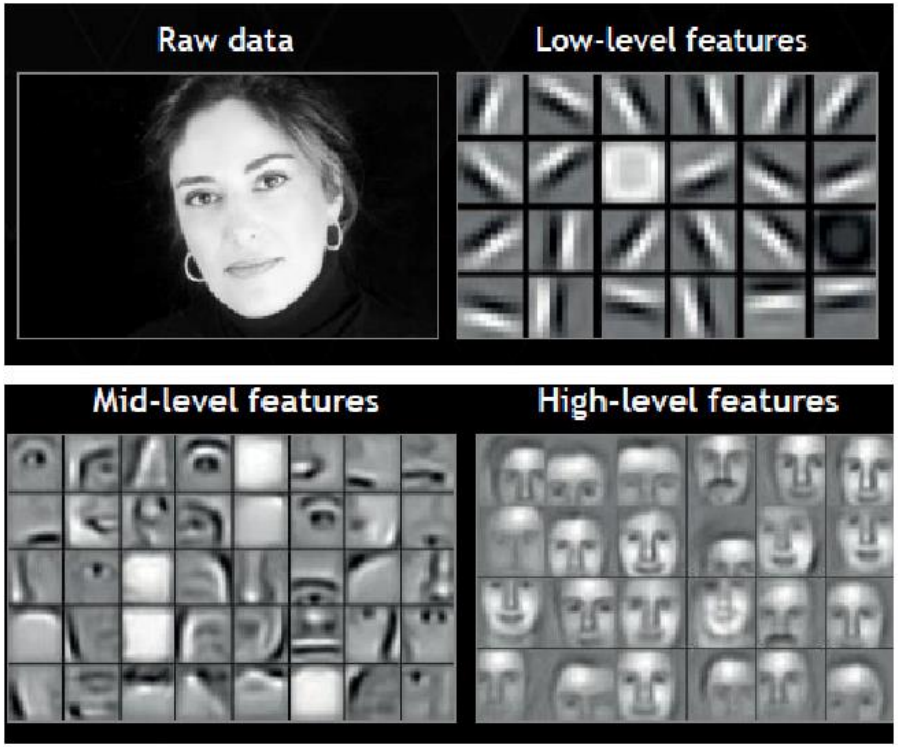
- A CNN is based on the stacking of one or more convolutional layers. A convolutional layer is based on the shared-weight architecture of the convolution kernels (or filters) that slide along input features and provide translation equivariant responses known as feature maps.

Convolutional layer



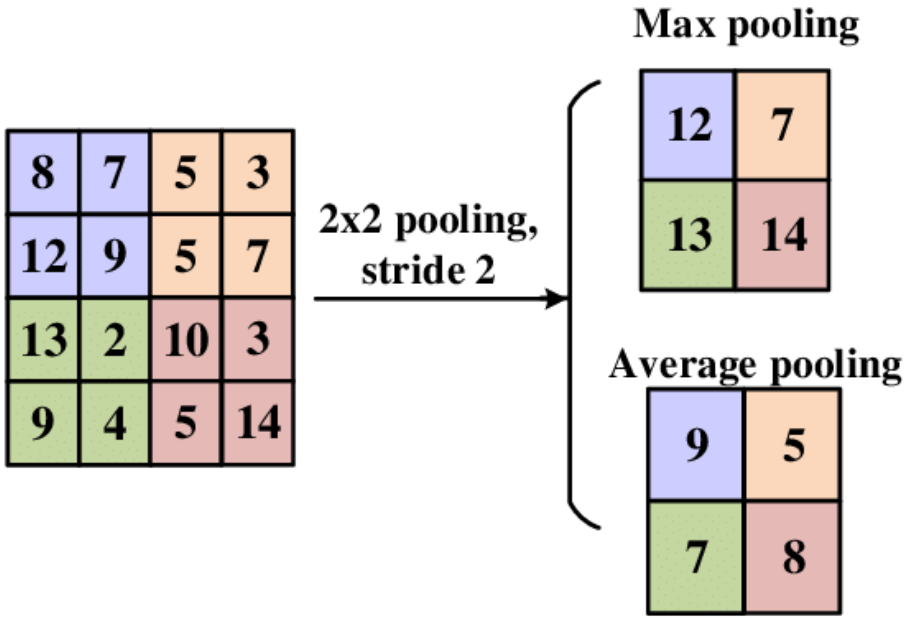
- A patch (submatrix) in the input matrix is multiplied by a kernel (or filter) to obtain an output value. This operation is done for every patch to obtain every output value.

What do the kernels represent in the convolutional layer(s)?



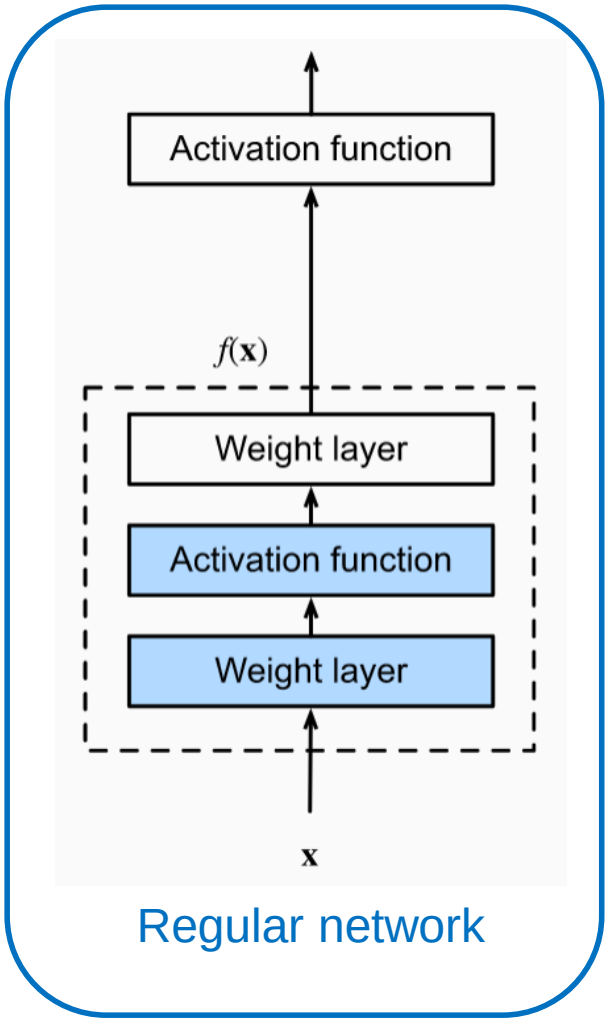
- In the first conv layer, the kernels correspond to low-level features (often edges). In the middle conv layers, the kernels correspond to mid-level features (parts of an object). In the last conv layers, the kernels correspond to high-level features (often objects).

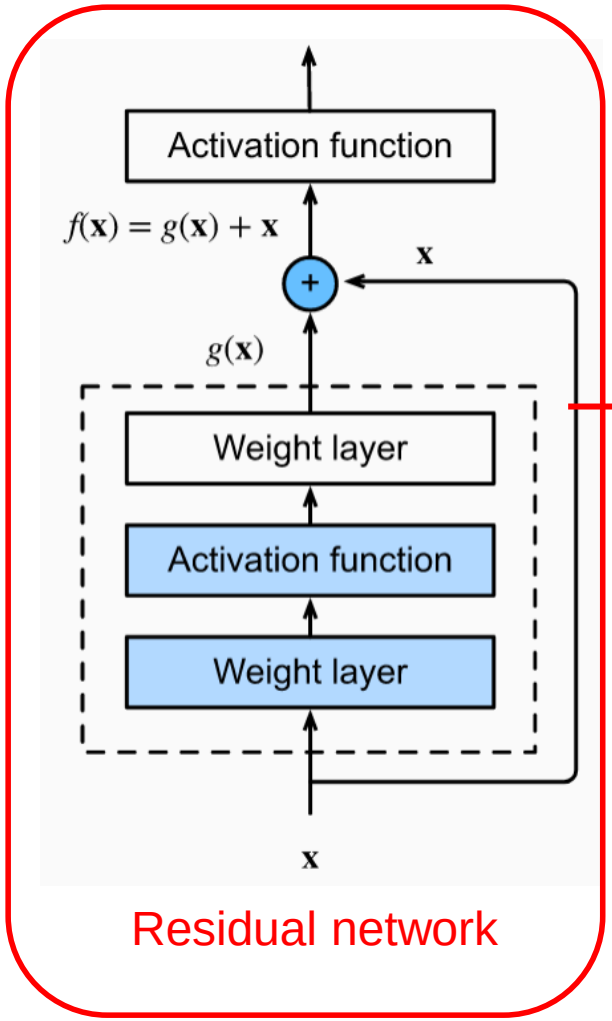
Pooling layer



- The pooling layer aims to reduce the dimension of the input matrix in order to summarize the underlying information.

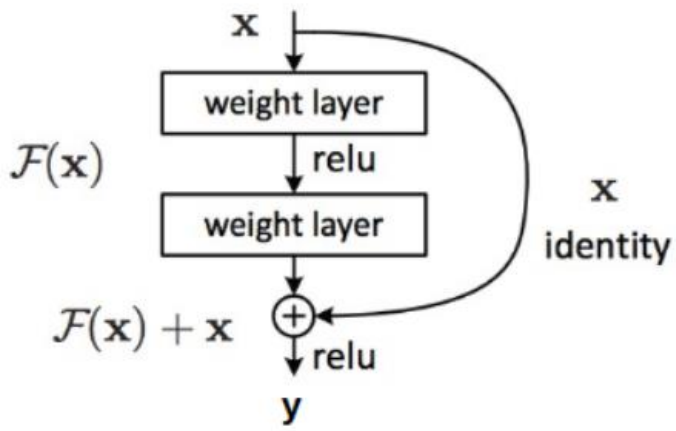
Residual network





Residual connection

Residual network as a way to reduce vanishing gradient problem



$$y = x + F(x)$$

$$\frac{\delta E}{\delta x} = \frac{\delta E}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta E}{\delta y} * (1 + F'(x))$$

$$= \frac{\delta E}{\delta y} + \frac{\delta E}{\delta y} * F'(x)$$

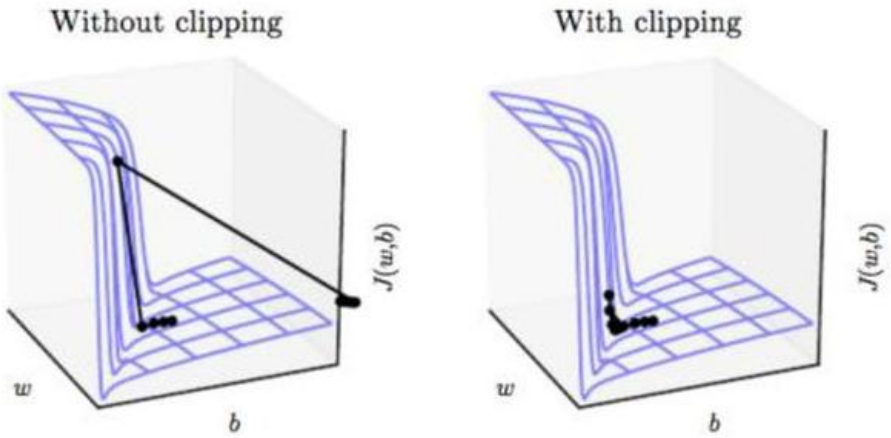
- Assuming E is the loss, the gradient dE/dy is added to the gradient dE/dy*F'(x).

  [*the derivate of a sum of functions is the sum of function derivates*]

- This way, the gradient cannot converge to zero (vanishing gradient).

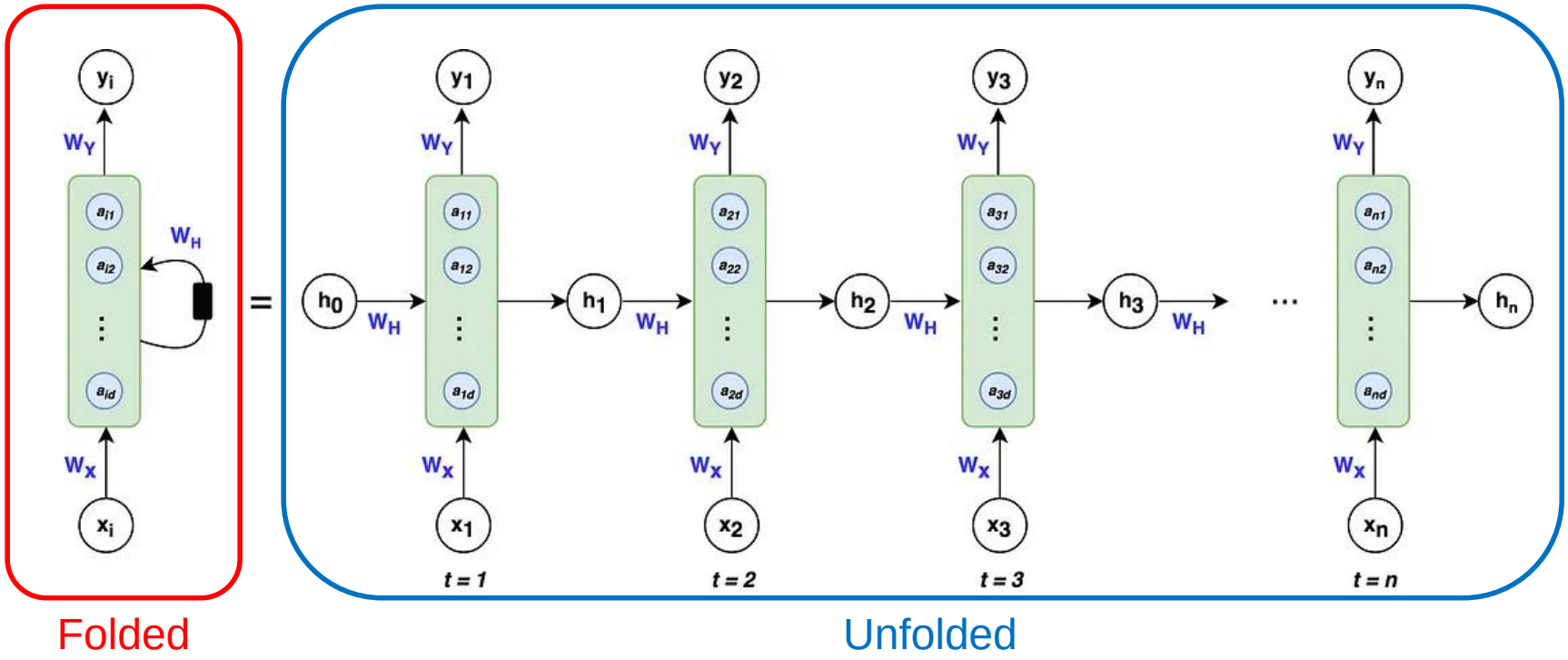
Gradient clipping to reduce exploding gradient

$$\mathbf{g} \leftarrow \frac{\eta \mathbf{g}}{\|\mathbf{g}\|}$$



- When the gradient g becomes too large (the norm of the gradient), gradient clipping normalizes the gradient g by its norm ||g|| given some constant η.

Recurrent neural networks (RNN)



- Recurrent neural networks are used to model temporal (or sequential) dependencies.

Simple RNN

- Elman network:

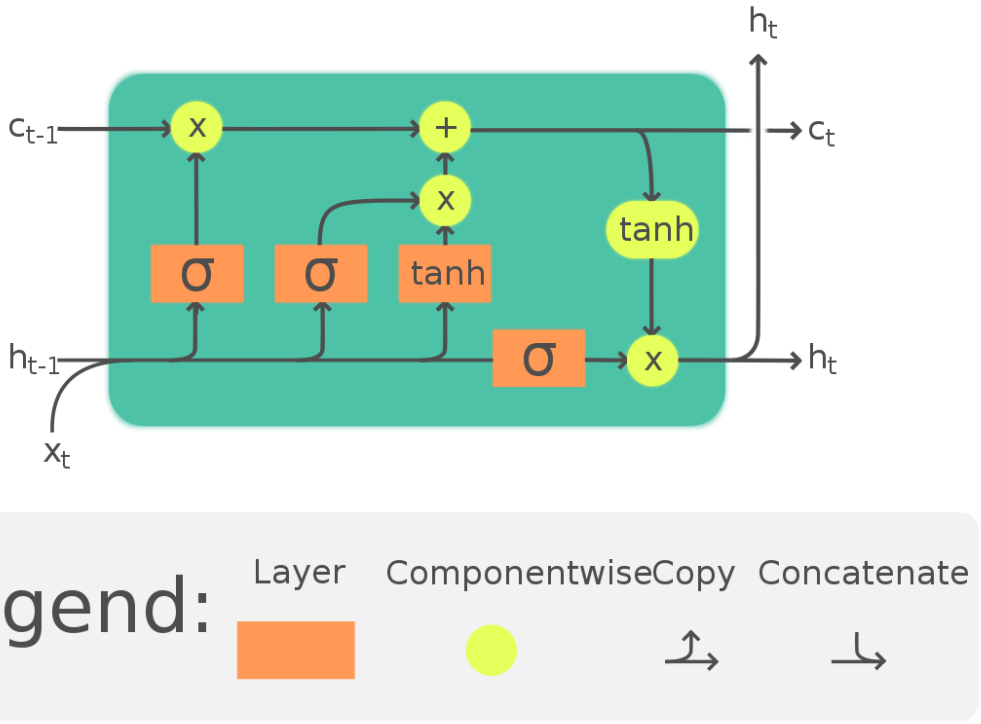
$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Variables and functions

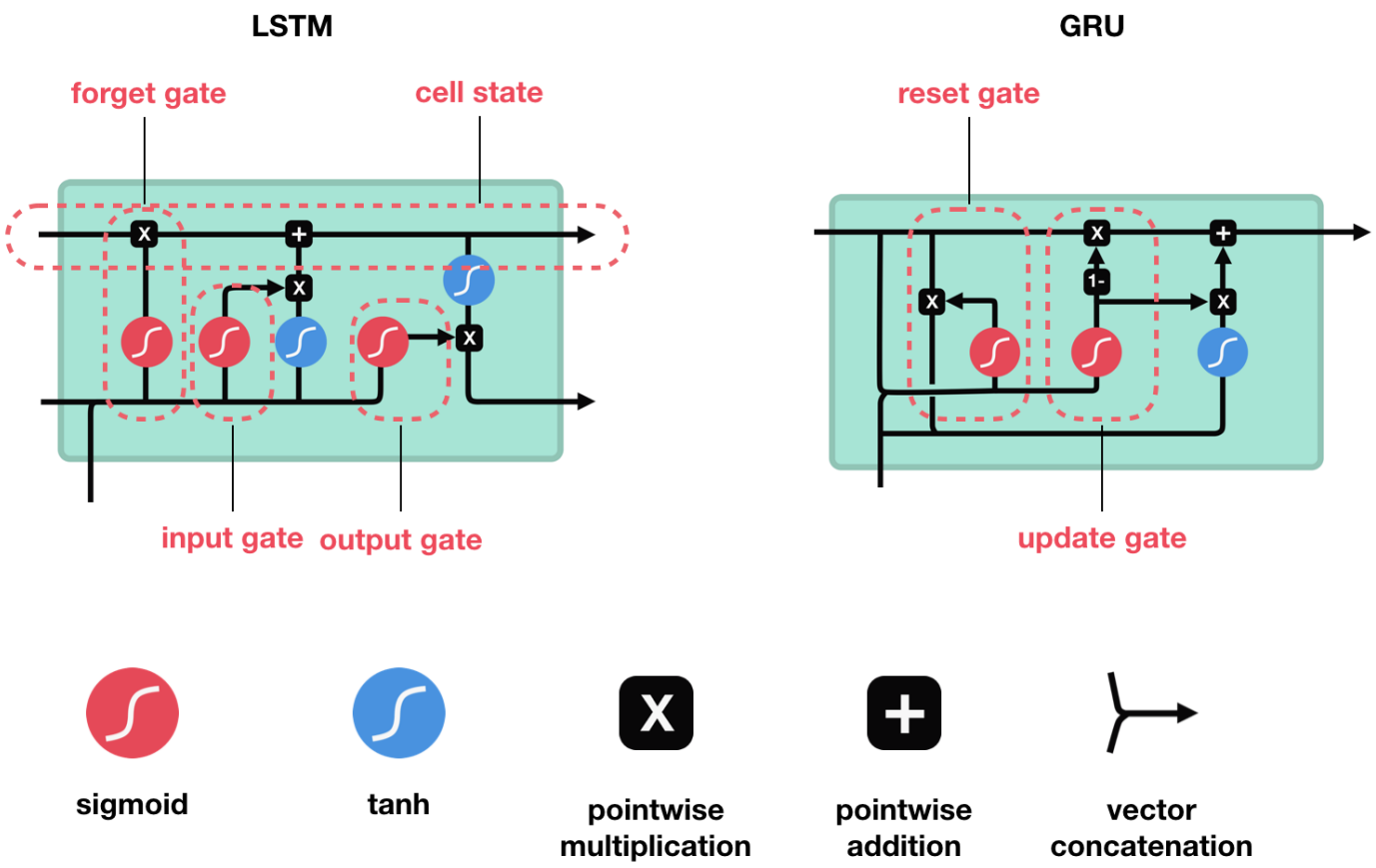
- x_t: input vector
- h_t: hidden layer vector
- y_t: output vector
- W, U and b: parameter matrices and vector
- σ_h and σ_y: Activation functions

LSTM (long short-term memory)

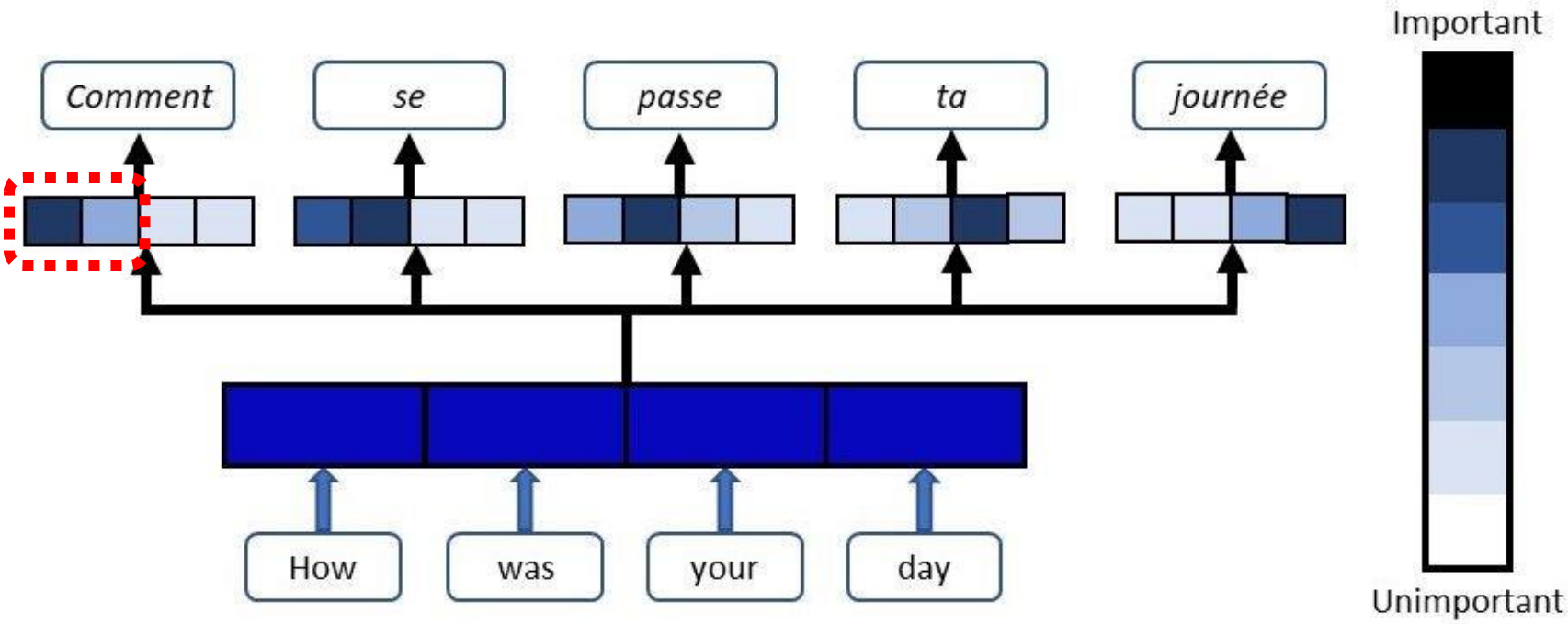
- Reduces the vanishing gradient problem found in RNNs.



GRU and comparison with LSTM



Attention



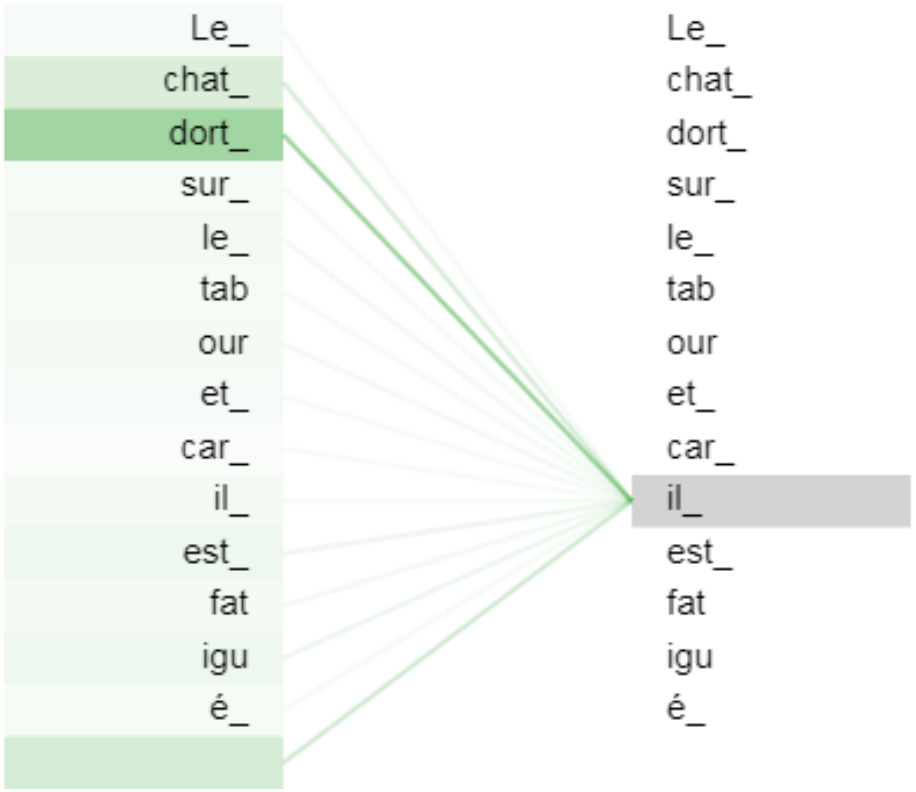
- Here, attention is used to weight different words from English to translate into French.
- For instance, to translate “how” to “comment”, you don’t only need the word “comment” (high weight) but you need other words such as the word “was” (moderate weight).

Self-attention in transformer model

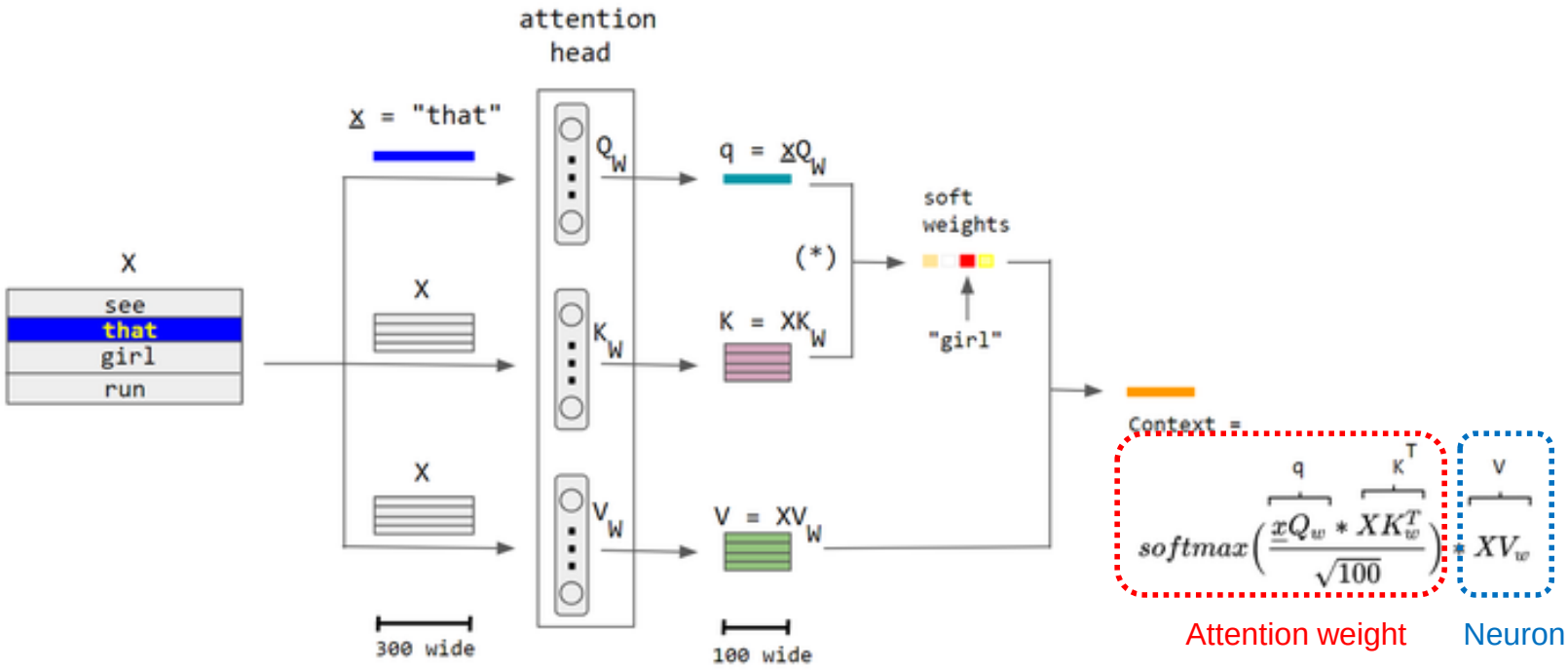
Self-attention is similar to attention, except that it is applied to the input sequence itself.

Self-attention allows to model long-range dependencies between any word in a sequence.

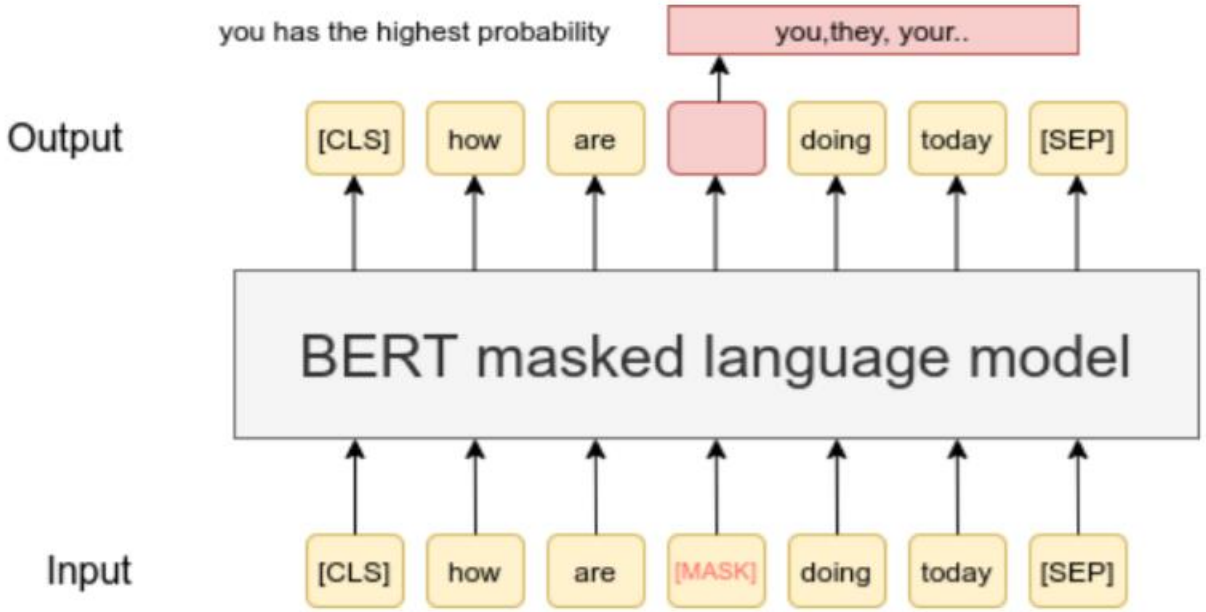
Self-attention is not directional as compared to RNN (LSTM or GRU), allowing parallel computing.



How to compute self-attention weights



Masked language model



- In a masked language model, the task is to predict some words that were masked using the context of the words.
- Together with self-attention, it was used for the BERT model (Bidirectional Encoder Representations from Transformers).

GPT-1 model (Masked language model)

- GPT-1 implements the transformer architecture (self-attention).

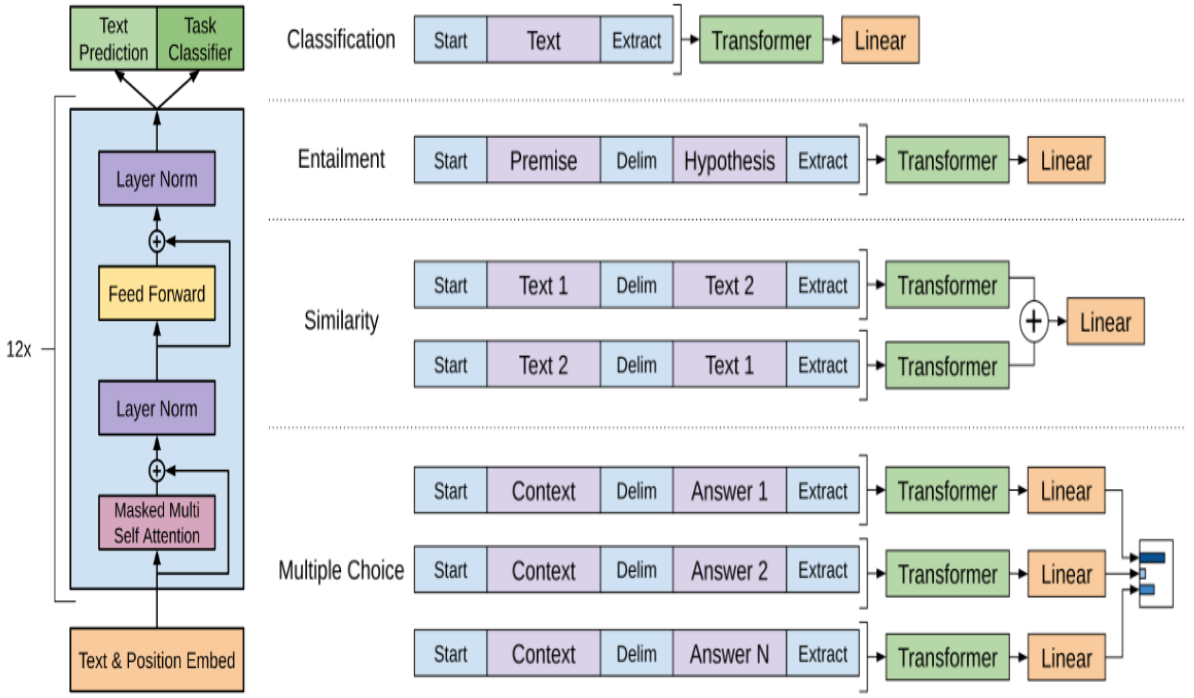


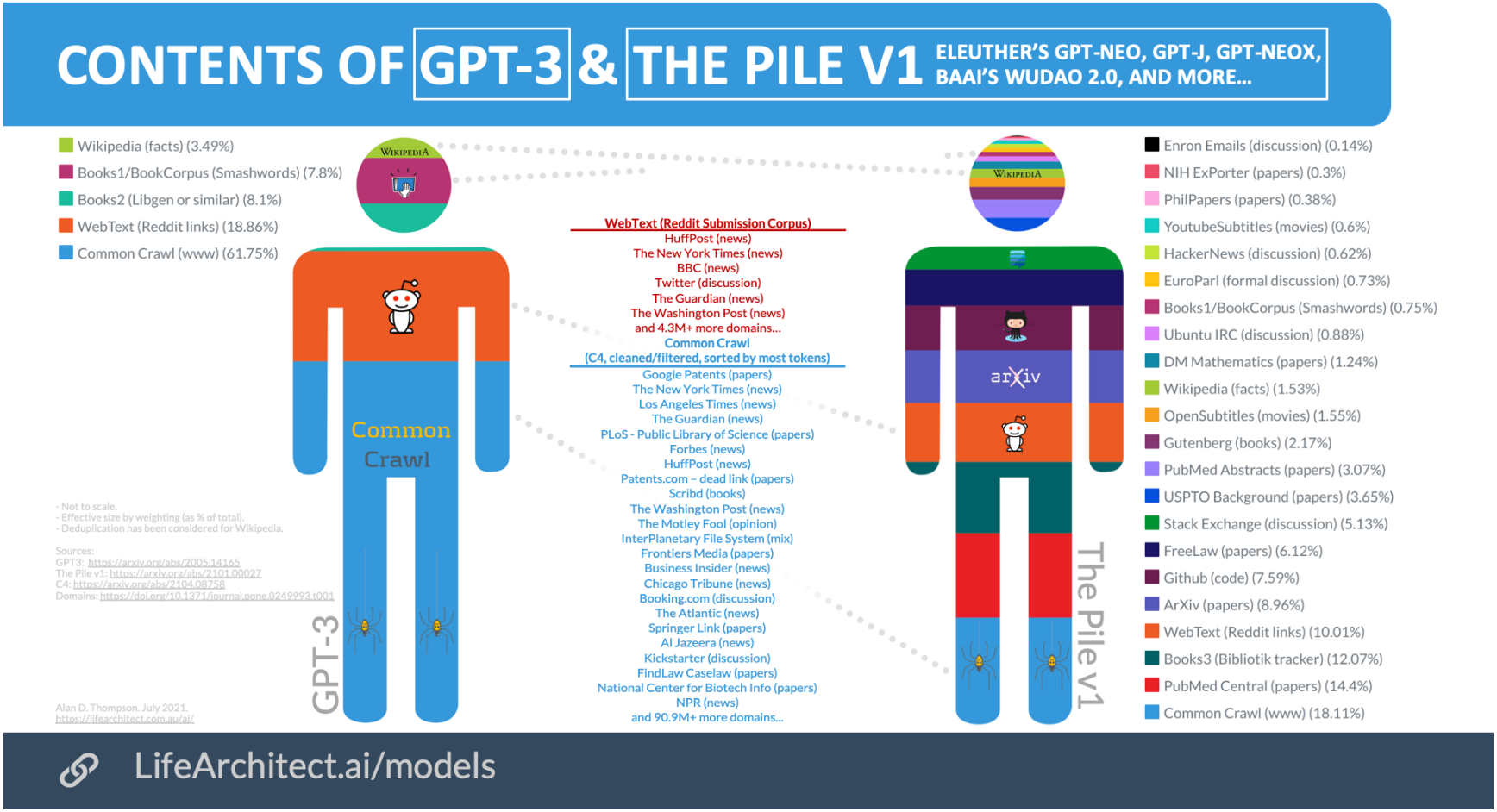
Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT-1, GPT-2 and GPT-3

	GPT-1	GPT-2	GPT-3
Parameters	117 million	1.5 billion	175 billion
Decoded Layers	12	48	96
Hidden Layer	768	1600	12288
Context Token size	512	1024	2048

- Number of parameters increasing over time.

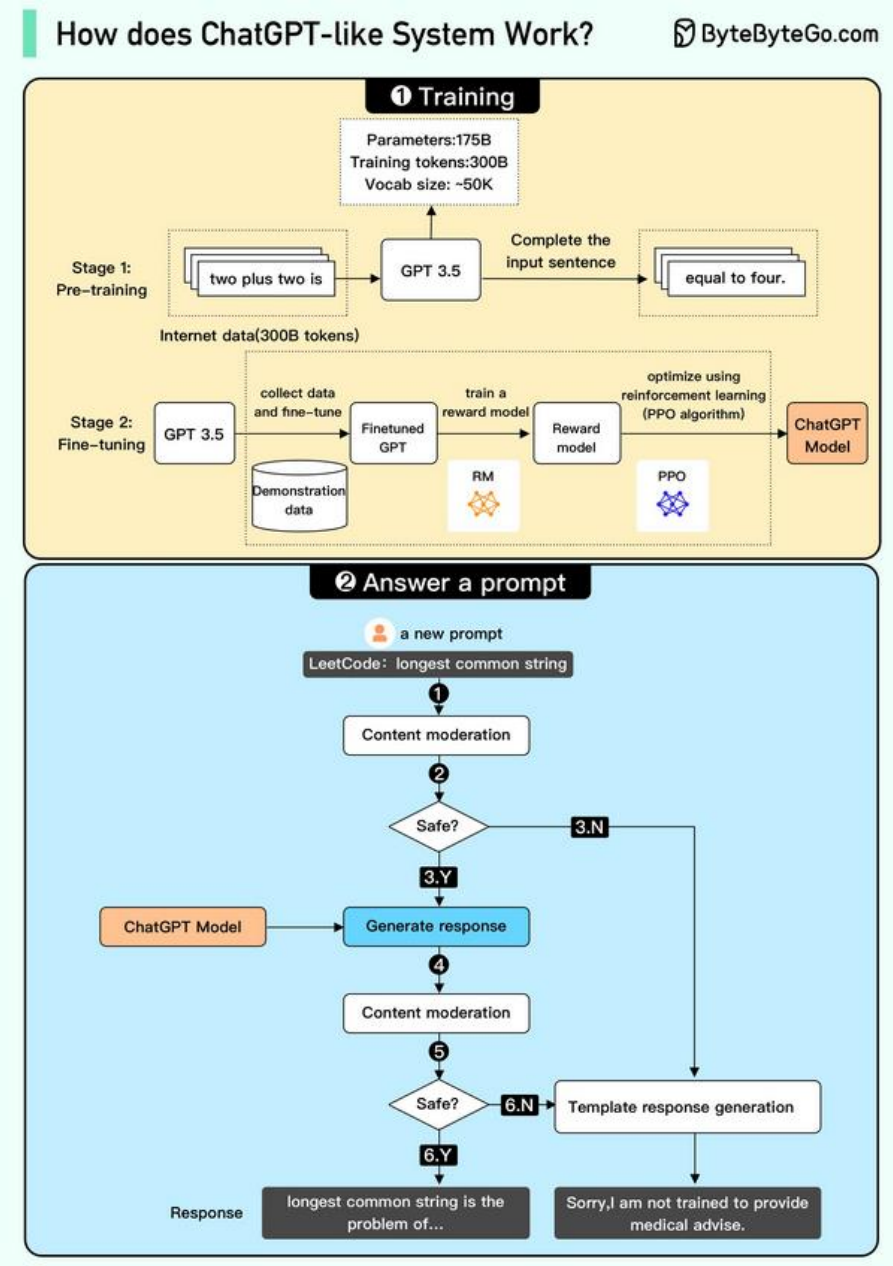
GPT-3 training data



LifeArchitect.ai/models

chatGPT

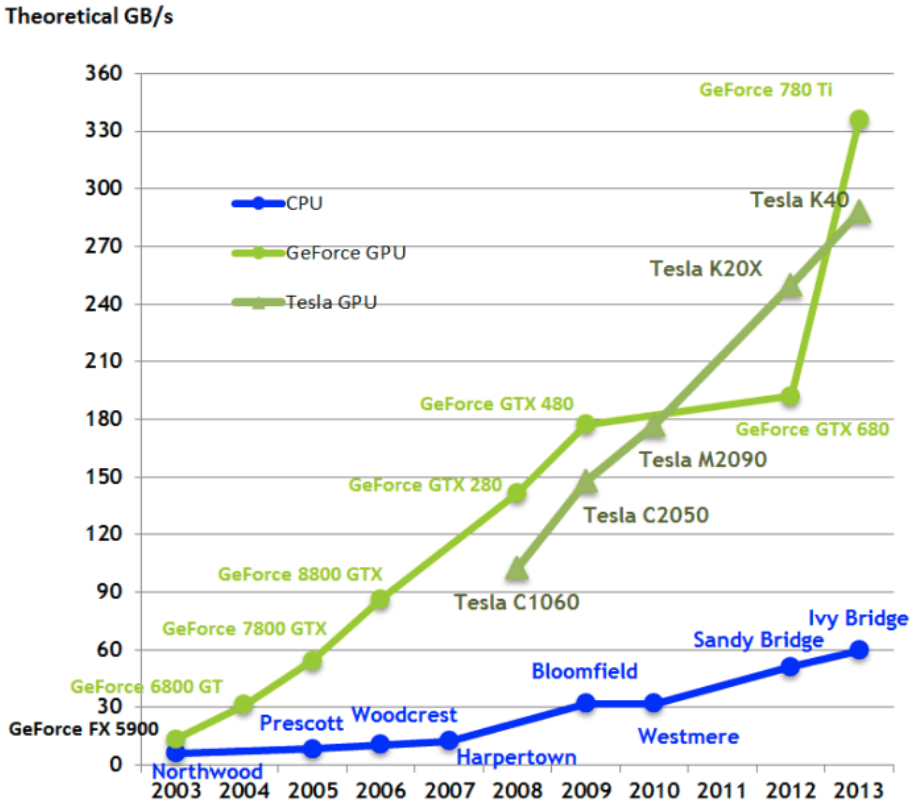
- Based on gpt3.5 (pretraining), with fine-tuning (stage 2) and reinforcement learning with human feedback (in blue).



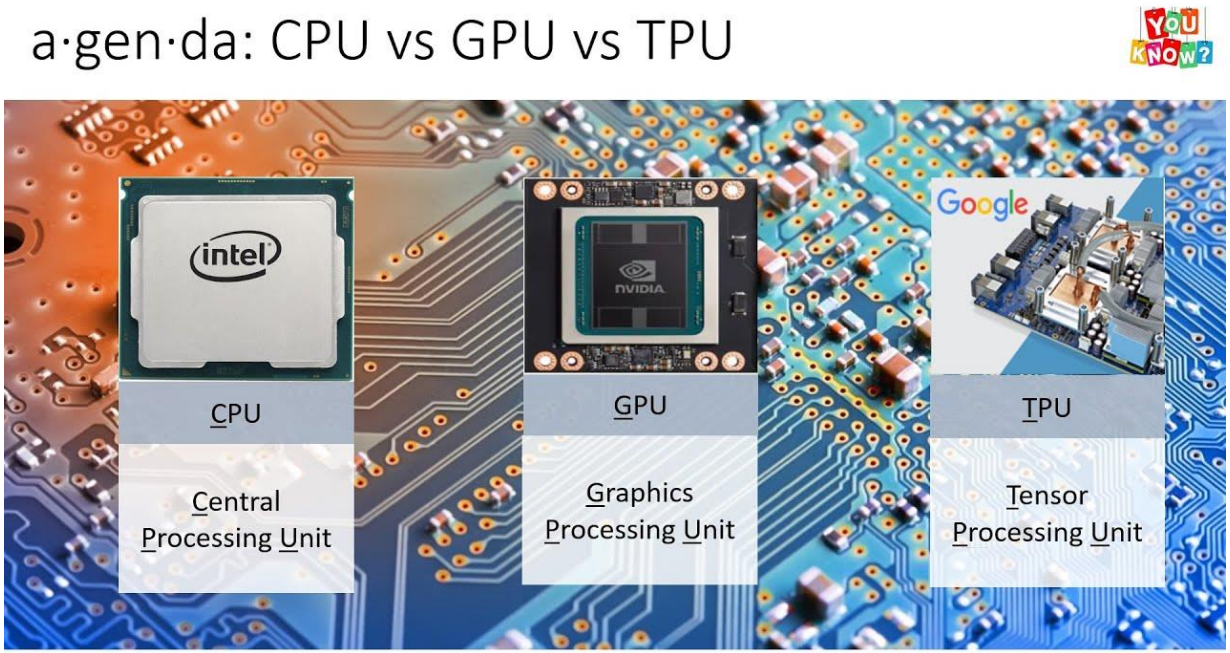
NEW COMPUTING HARDWARES

CPUs are not performing well for deep learning

- CPU is the main processor of a computer.
- CPU are not designed for machine learning applications.
- Machine learning needs highly parallel computing.

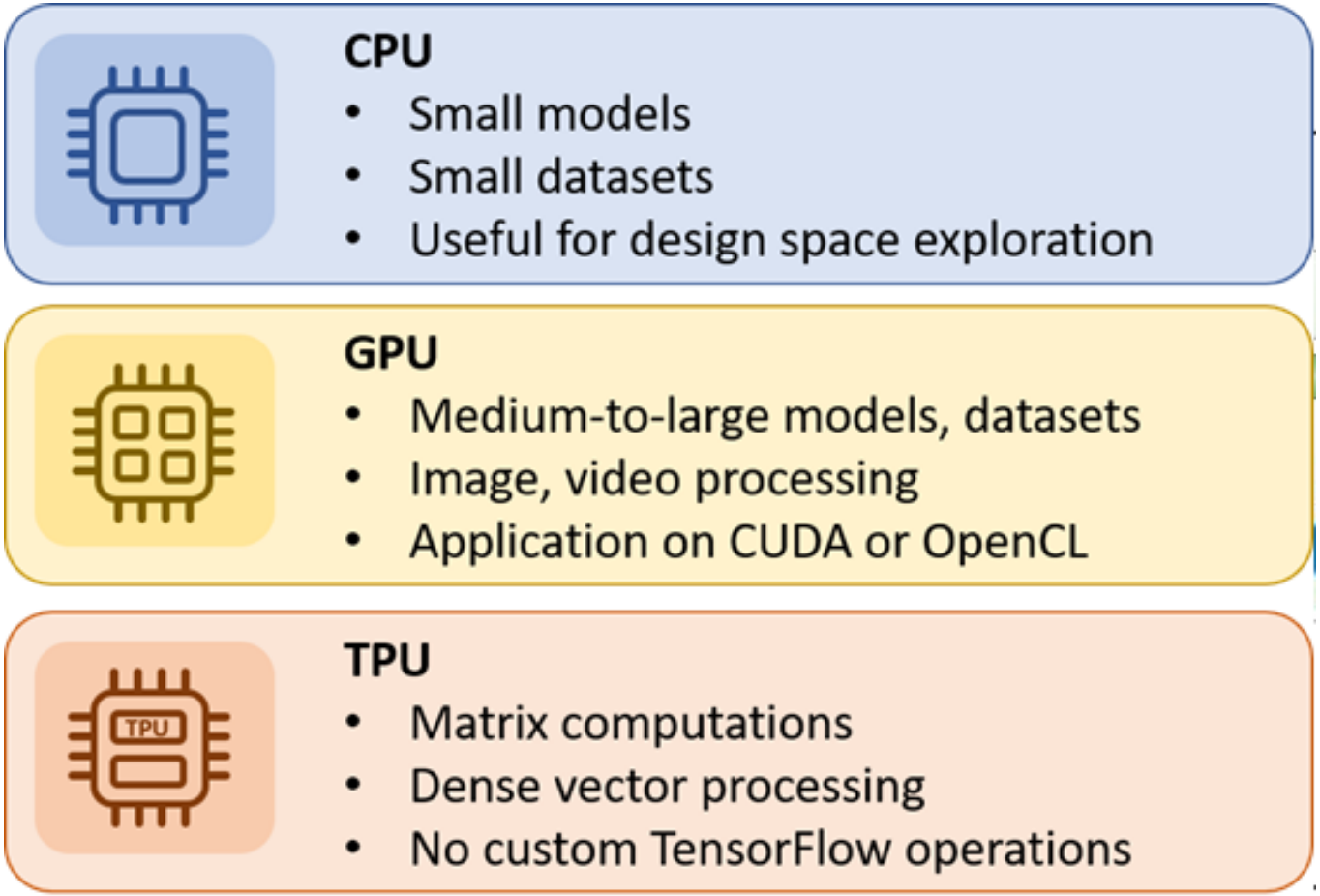


GPUs and TPUs are better suited



- GPUs were originally designed for video games, but repurposed for deep learning.
- New processors (TPUs) were specifically designed for deep learning.

Comparison of CPU vs GPU vs TPU

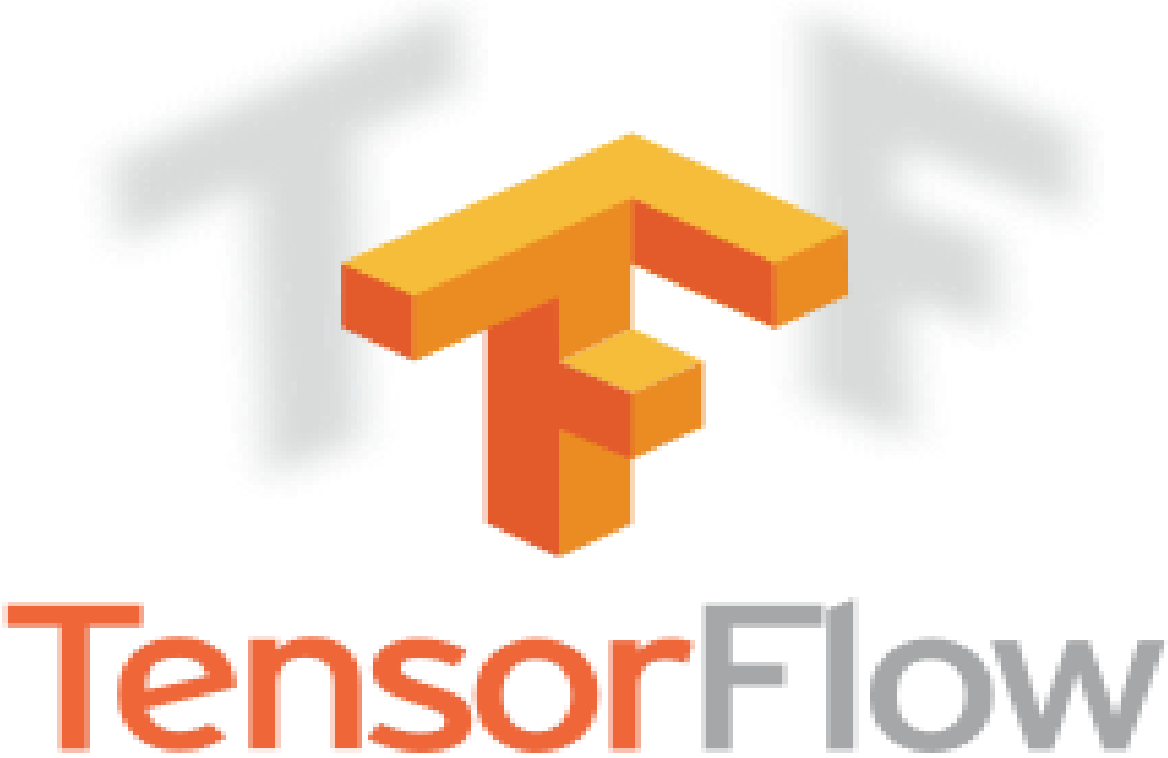


DEEP LEARNING LIBRAIRIES

Tensorflow

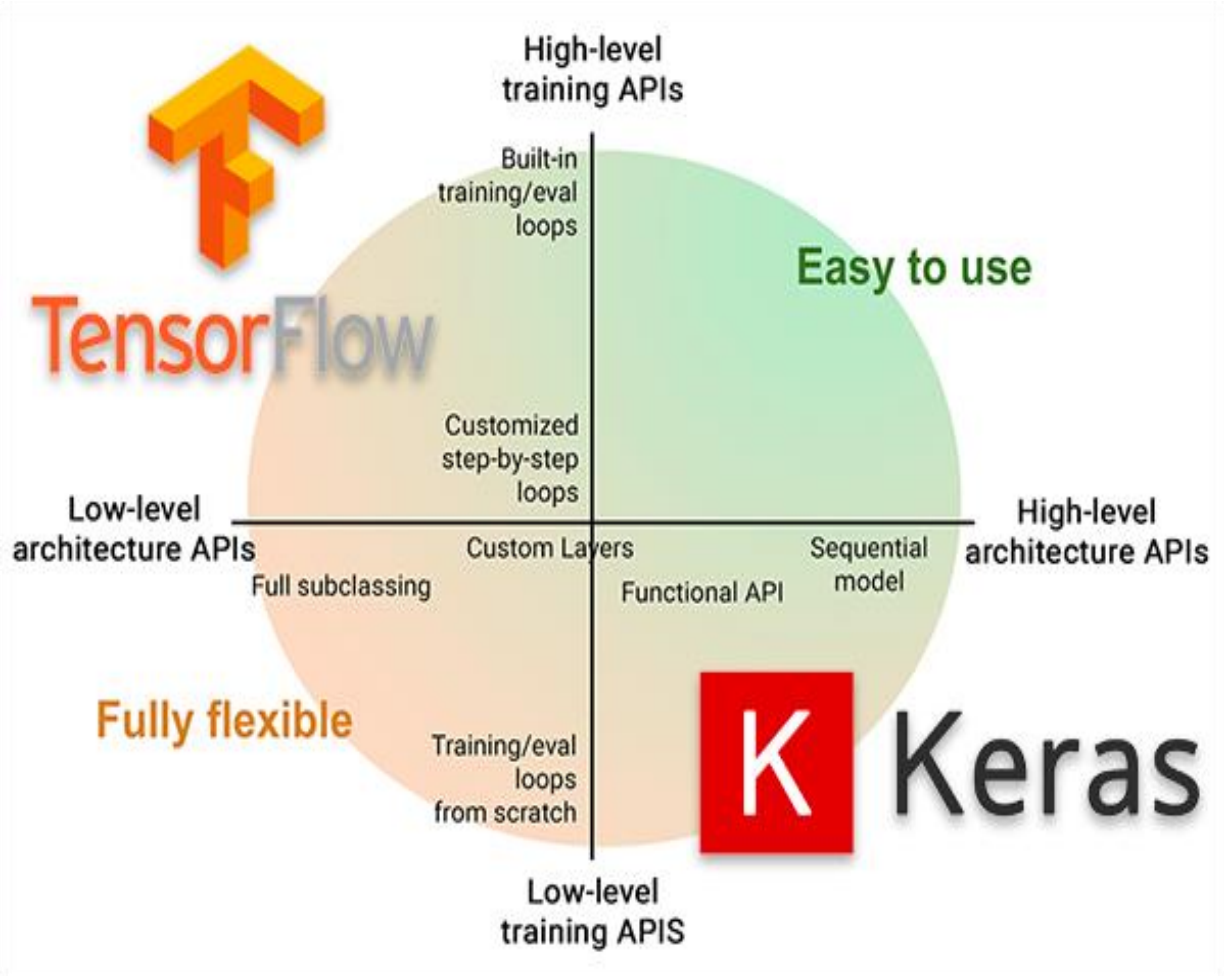
- Library developed by Google to develop and train deep neural networks. Version 1.0.0 was released on February 11, 2017.
- Run on multiple CPUs and GPUs (or now on TPUs).
- Python library, but can be used in other languages such as R.
- Very fast and memory efficient library.
- Very flexible library: you can implement any function and make your own layers for deep learning.

Keras as a simplification of Tensorflow



Keras

- Easy to implement a model in a just a few lines of code.



Keras: two modes

- tf.keras.Sequential() class:
  - Every line corresponds to a layer.
  - Easy to code.
  - Cannot deal with complex models with multiple inputs and outputs!

- tf.keras.Model() class:
  - More complex models with multiple inputs and outputs.
  - Can be used to implement any model.
  - Not for beginners!

TF and keras installation

- For CPU installation (computation is slow), installation is easy!

- For GPU installation (computation is optimized and uses GPU), installation is quite tricky!
  - Need to install python, CUDA, Cudnn, tensorflow, Keras, with versions that are compatible with your graphic card.

Keras R

Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

Simple MNIST convnet

- Setup
- Prepare the data
- Build the model
- Train the model
- Evaluate the trained model

Keras R

R interface to Keras

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.* Keras has the following key features:

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

See the package website at https://tensorflow.rstudio.com for complete documentation.

Other library: Pytorch

```
library(keras)
```

Let's start by loading and preparing the MNIST dataset. The values of thee pixels are integers between 0 and 255 and we will convert them to floats between 0 and 1.

```
mnist <- dataset_mnist()
mnist$train$x <- mnist$train$x/255
mnist$test$x <- mnist$test$x/255
```

Now, let's define the a Keras model using the sequential API.

```
model <- keras_model_sequential() %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(0.2) %>%
  layer_dense(10, activation = "softmax")
```

Note that when using the Sequential API the first layer must specify the `input_shape` argument which represents the dimensions of the input. In our case, images 28x28.

After definning the model, you can see information about layers, number of parameters, etc with the `summary` function:

```
summary(model)
```

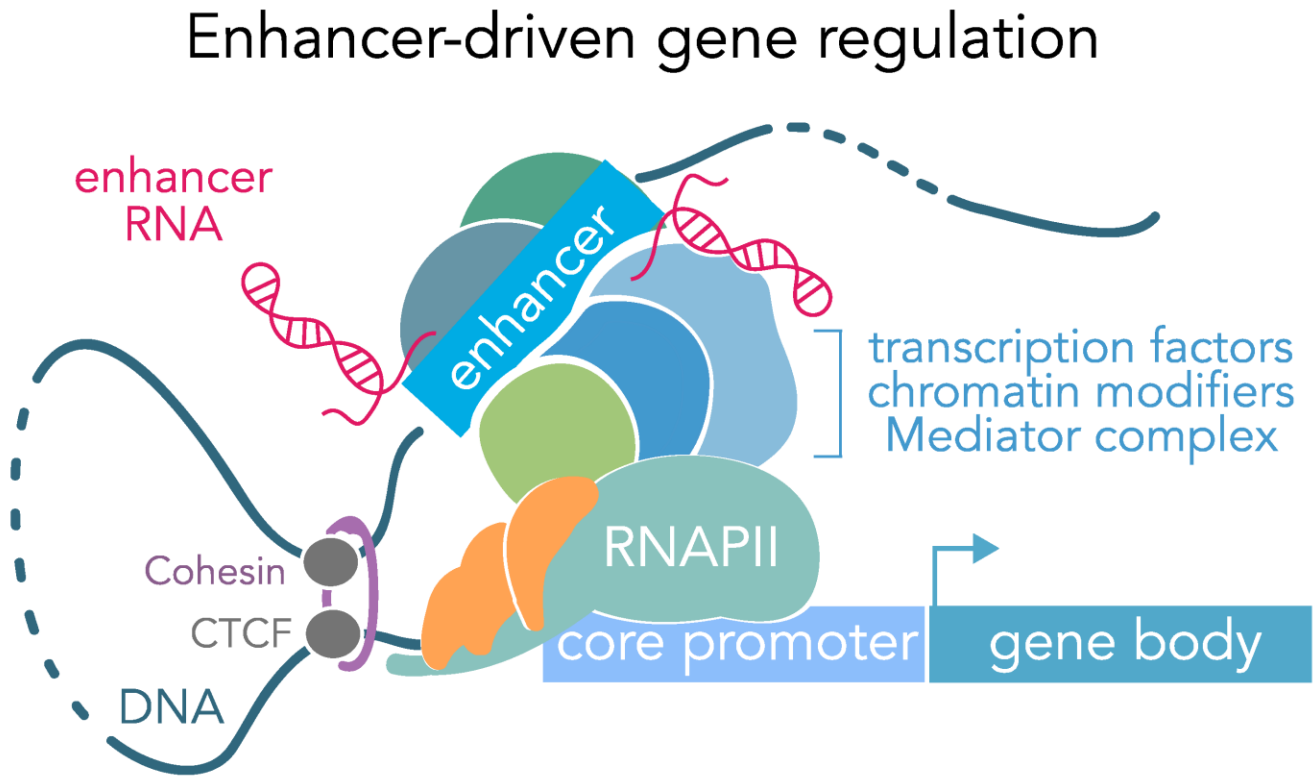
Comparison Keras vs PyTorch

	Keras	PyTorch
API	High-Level	Low-Level
Coding	Python	Python
Architecture	Simple	Complex
Speed	Slow	Fast
Data Sets	Small	Large

>hackr.io

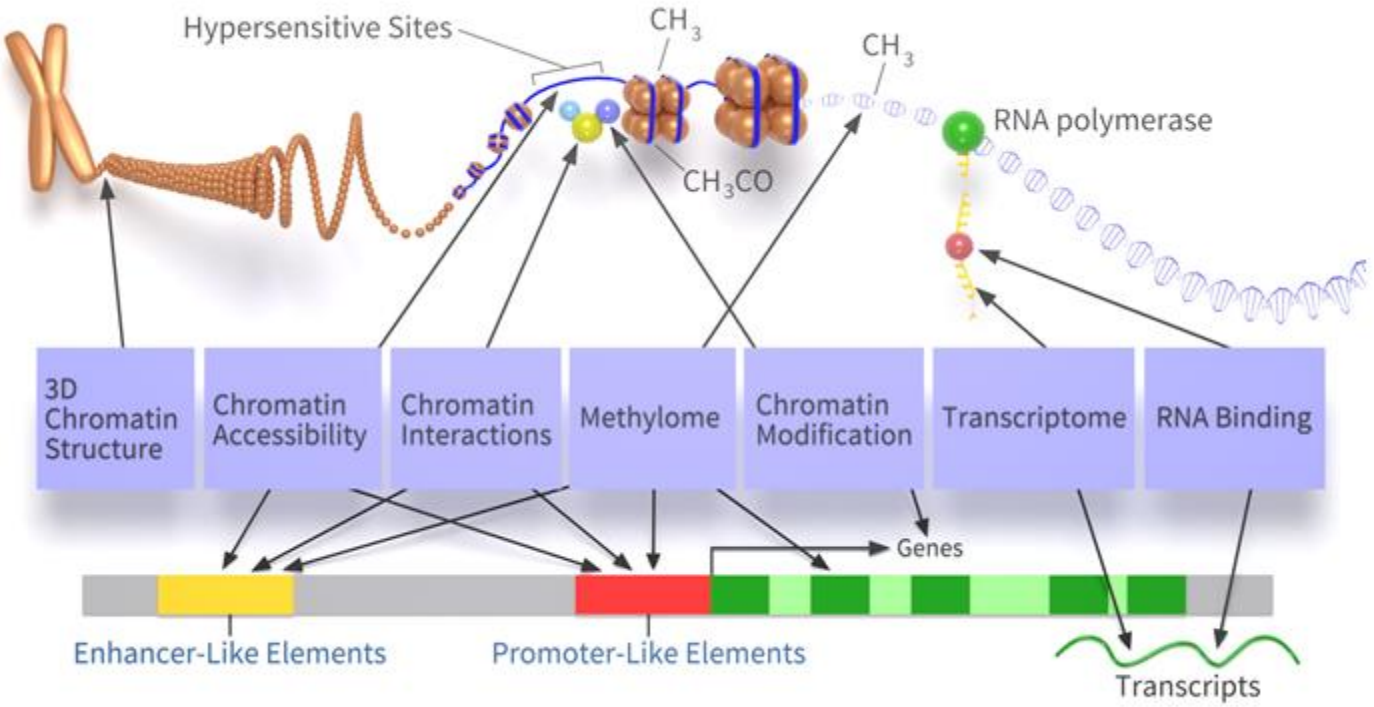
REGULATORY GENOMICS

Regulation of gene expression in Eucaryotes



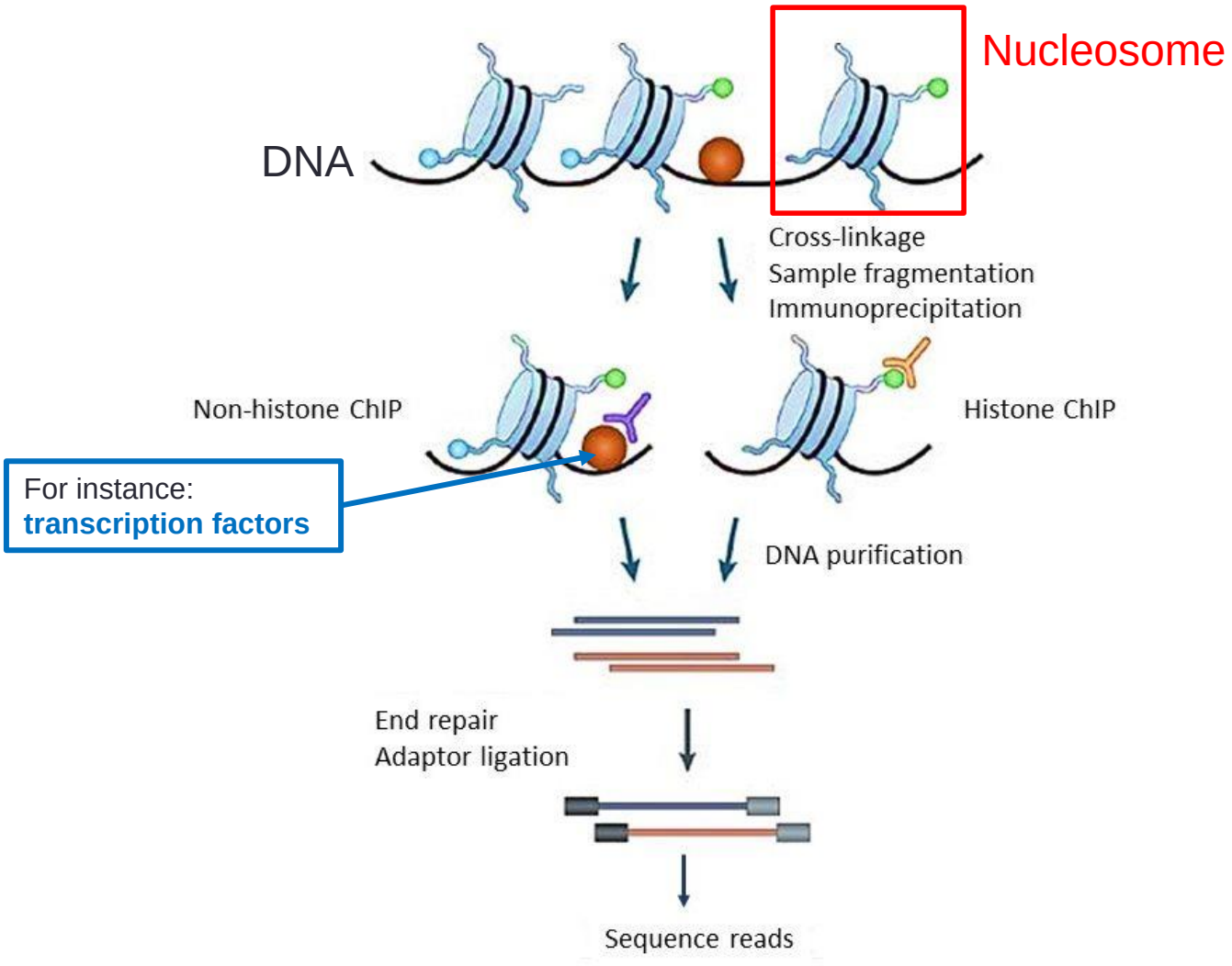
- Regulatory regions (promoters, enhancers, insulators, …) are non-coding DNA sequences that control the expression of target genes.

How to map regulatory regions?

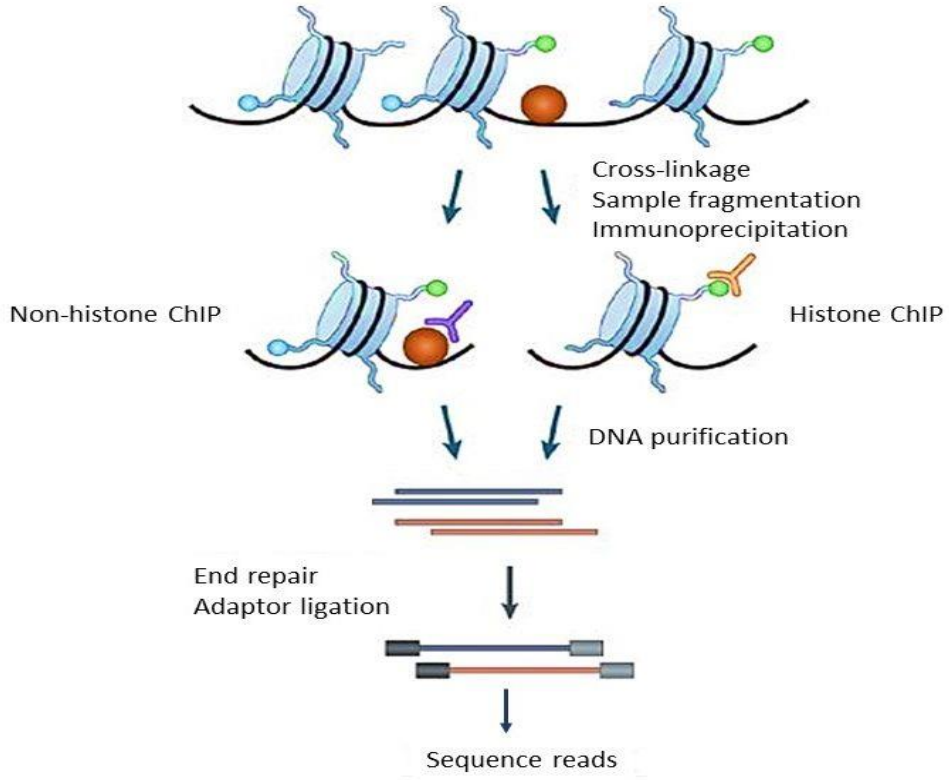


- Regulatory regions were mapped during the last decade using techniques such as ChIP-seq, ATAC-seq, Hi-C, methyl-seq...

ChIP-seq experiment

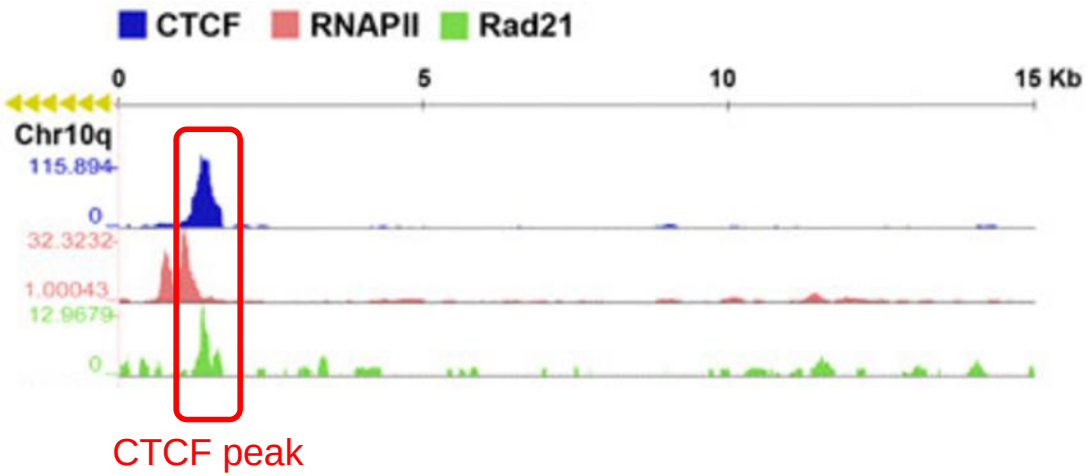


CTCF: a key chromatin protein



CTCF is a key protein involved in chromatin loops with another protein called cohesin, which controls the regulation of gene expression. CTCF binds to specific sites on DNA.

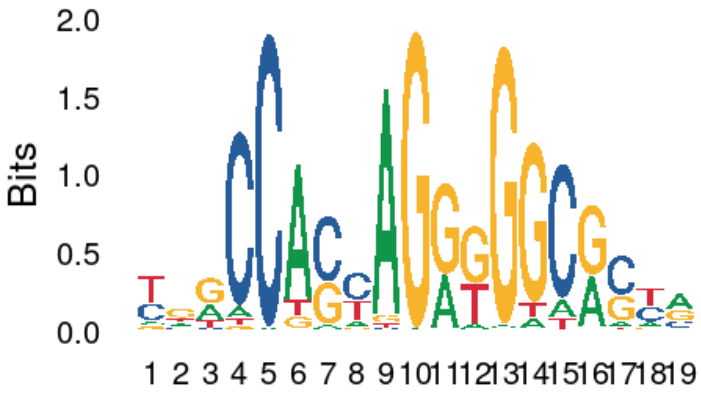
CTCF ChIP-seq peaks as examples



- We extract the DNA sequences of the CTCF ChIP-seq peaks to identify a DNA motif.

CTCF binds to a specific DNA motif

- If we run a motif search (using MEME for instance), we will observe the CTCF motif MA0139.1:



CTCF motif

CLASSIFICATION OF DNA SEQUENCES AS A MACHINE LEARNING PROBLEM

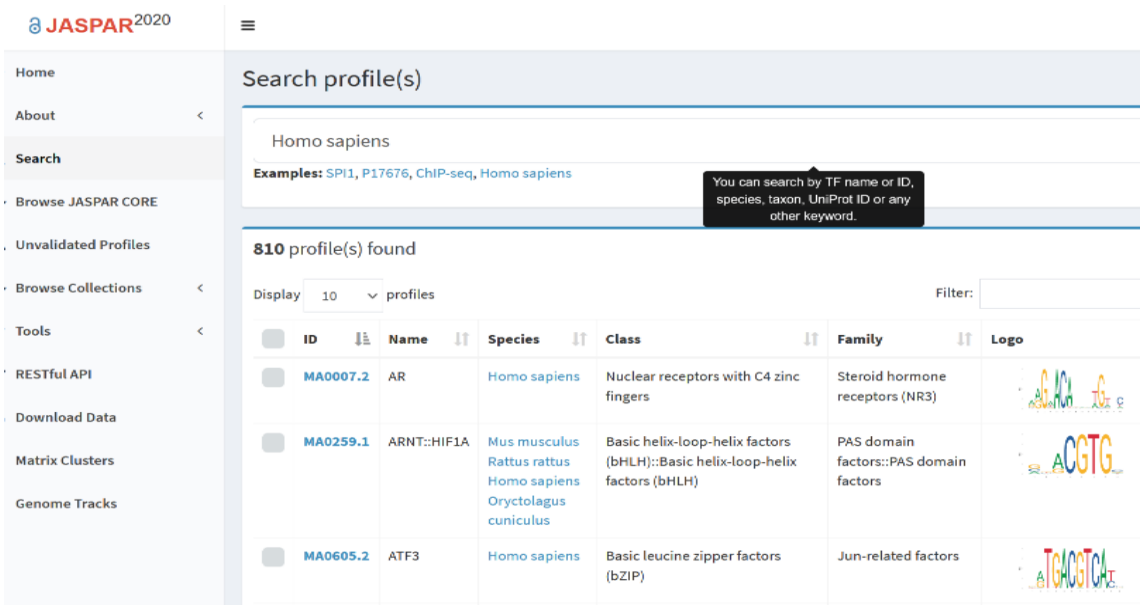
Classifier to distinguish CTCF bound sequences from other sequences

- One can build a simple classifier that distinguish between CTCF bound sequences and other sequences by looking for the occurrence of the motif MA0139.1.

- NB: other sequences (negative sequences) can be:
  - Shuffled bound sequences while conserving dinucleotides (ie 2nd order markov model) as implemented in MEME suite.
  - Real unbound sequences drawn from the genome with GC content, length and mappability similar to the bound sequences as implemented in gkmSVM R package.

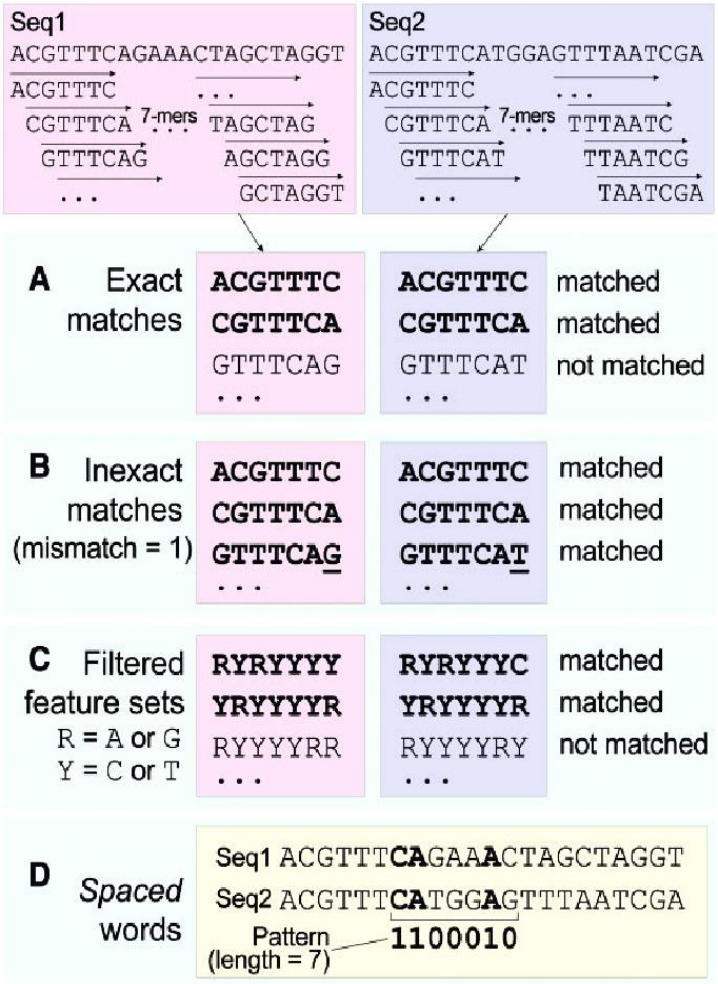
How to obtain features from the DNA sequences?

- Known transcription factor motifs (JASPAR, HOCOMOCO databases): motif occurrence can be used.
- Very easy but you will miss features that are important predictors but are not in the database!



How to obtain features from the DNA sequences?

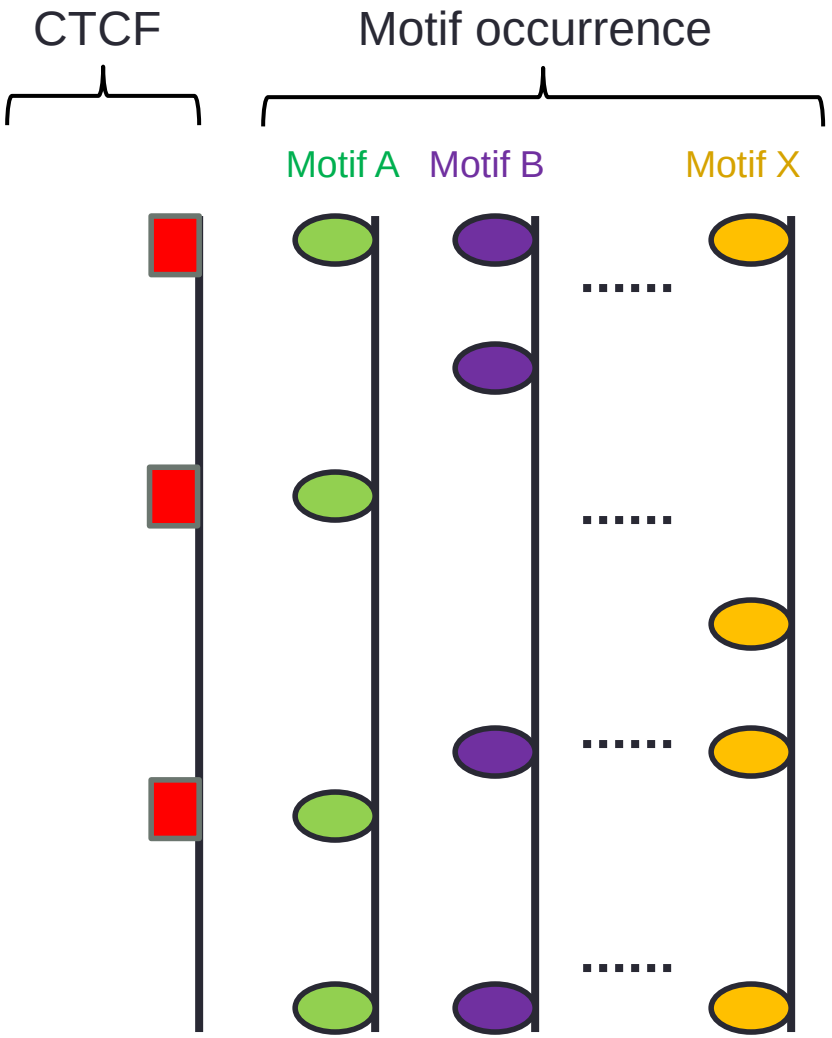
- K-mers (for instance: ATCTC or ATTTC, …): very powerfull approach since almost no prior information (except k) is needed to build the features. Then you can count the occurrence of every k-mer.



How to obtain features from the DNA sequences?

- Many other possible features:
  - Di/trinucleotide composition
  - DNA shape
  - Microsatellites: any kind of repeats
  - DNA conservation among species (PhastCons score)
  - Any other motif from the litterature.

Data table



Model variables

Y	X1	X2	…	Xp
1	2	1	…	5
0	0	0	…	0
0	0	1	…	0
0	0	0	…	0
1	1	0	…	0
0	0	0	…	0
0	0	0	…	2
0	0	0	…	0
0	0	3	…	7
1	1	0	…	0
0	0	0	…	0
0	0	0	…	0
1	1	2	…	1

Simple model: logistic regression

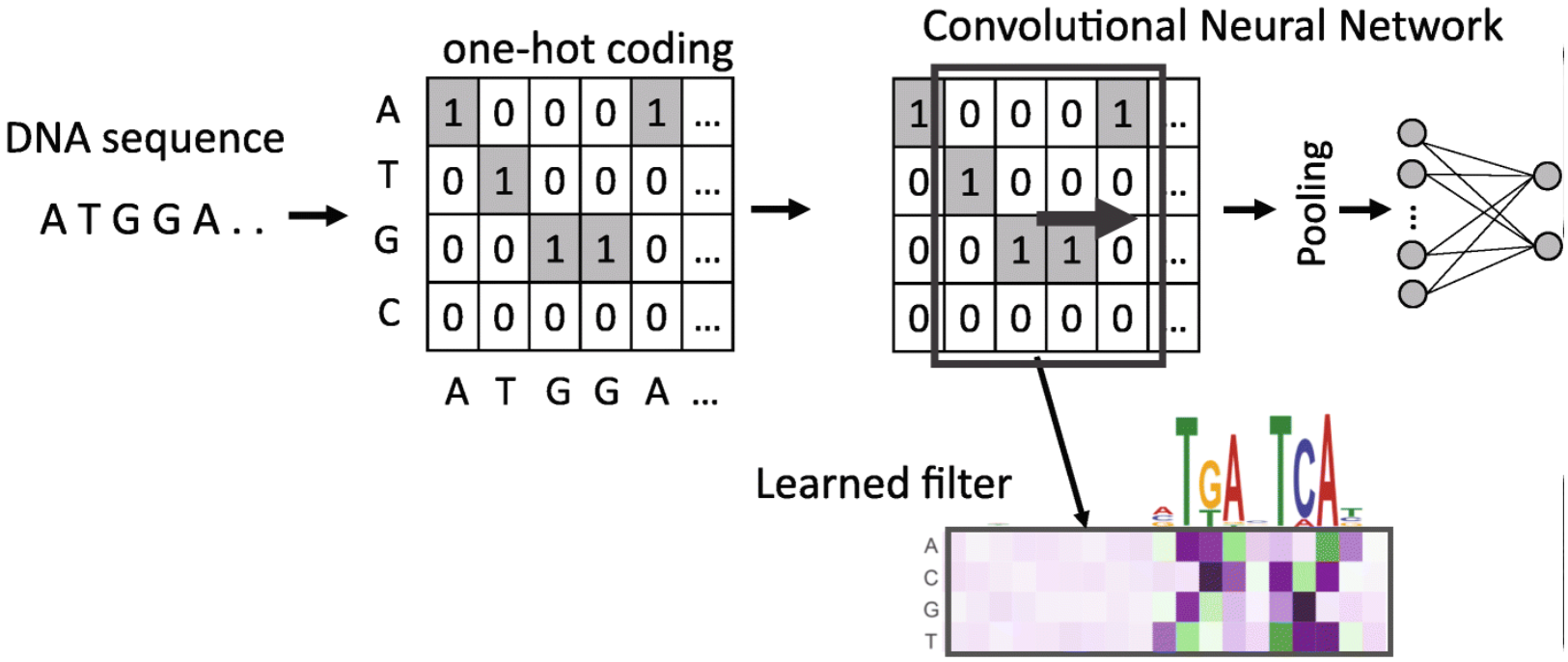
Logistic regression

$$ln\frac{Prob(Y=1|X)}{1-Prob(Y=1|X)}=\beta_0+\beta X$$

- One of the most basic classifier.
- Assumes additivity effect of predictors (each predictor contributes independently).
- Can add lasso penalty to improve predictions.

DEEP LEARNING FOR GENOMICS

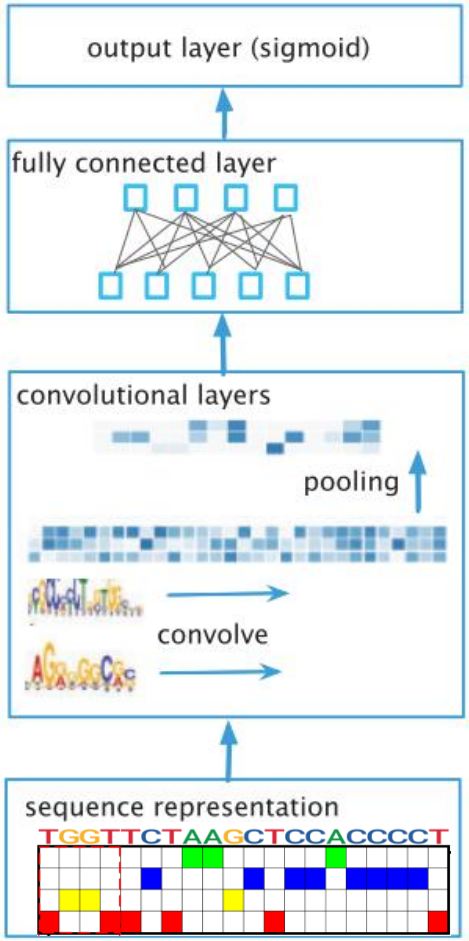
Meaning of 1D-convolution for DNA sequences



- Based on kernels (filters) that are matrices of weights for each base of a DNA motif.
- Kernels = Position Weight Matrices (given some transformation).

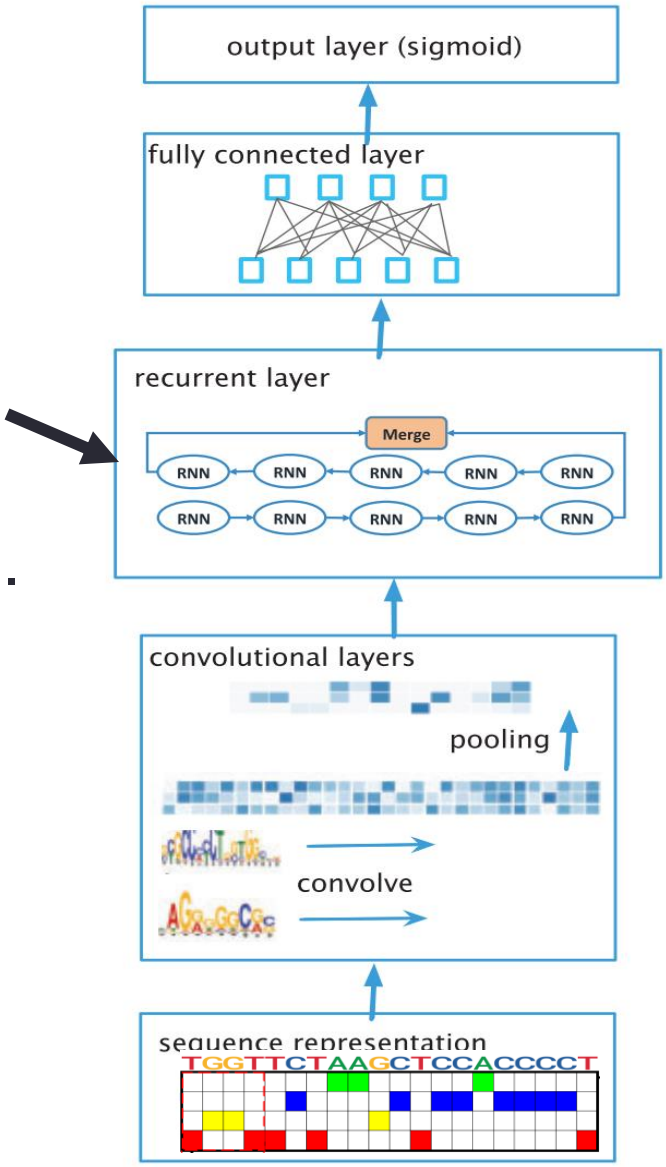
CNN for classifying DNA sequences

- Binary output
- Standard fully connected layer
- 1-dimension convolution
  = DNA motif scanning
- One-hot encoding of DNA:
  Colored cells = 1; white cells = 0.

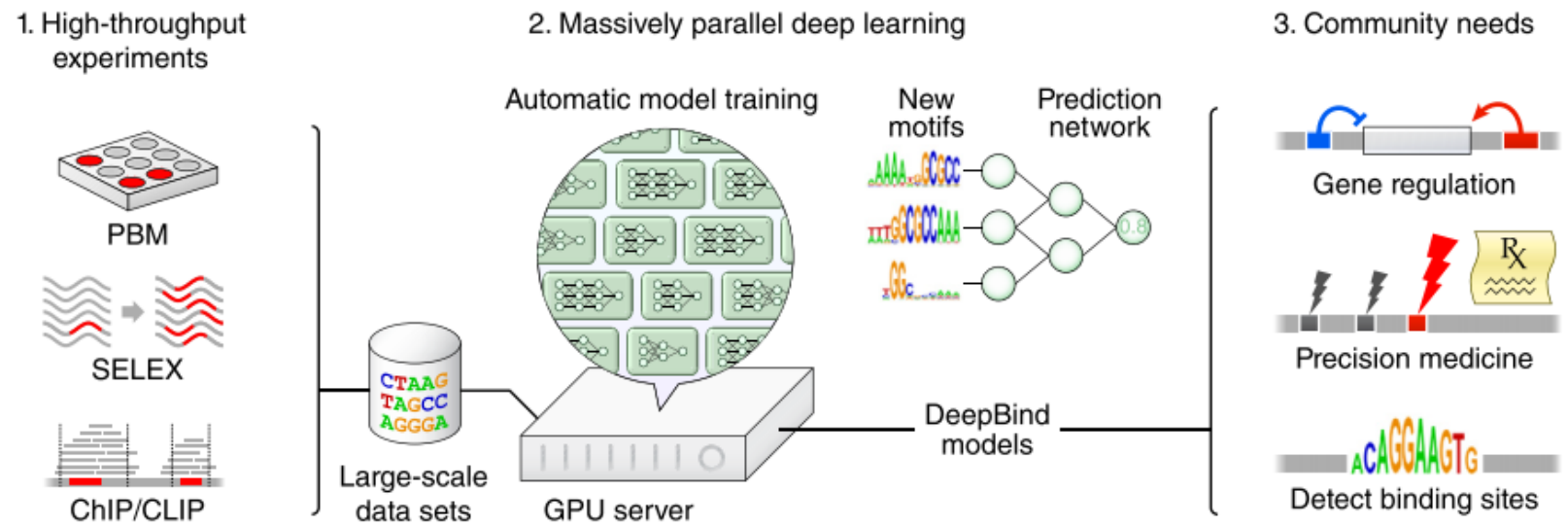


CNN + RNN model

- One can simply add an RNN (or GRU or LSTM or bi-LSTM) layer on the top of the CNN layer in order to capture long-range dependencies between the motifs of the sequence.



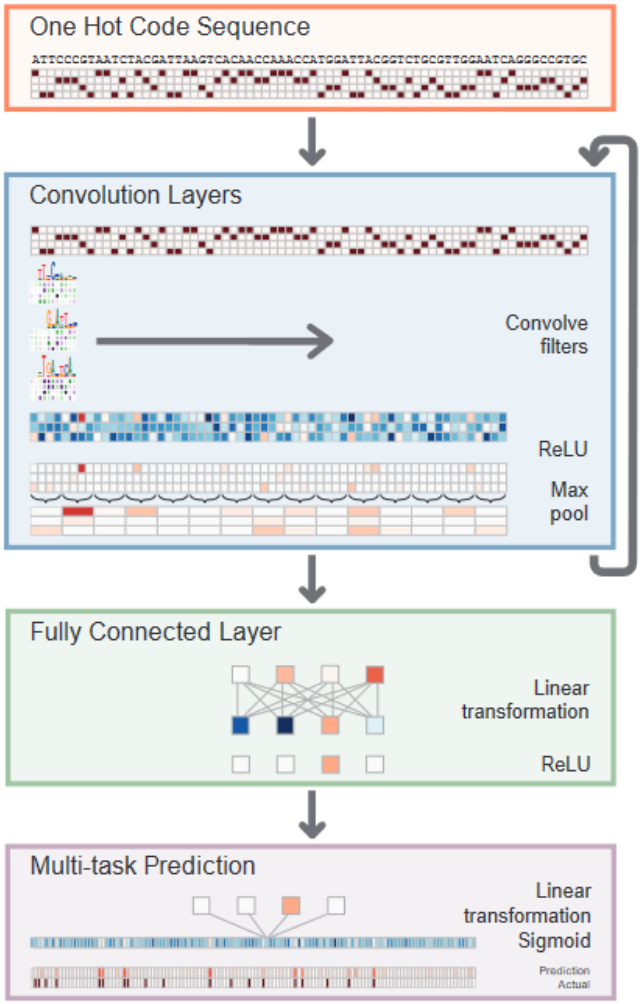
Example: Deepbind



- Predict binding proteins to DNA given the DNA sequence.

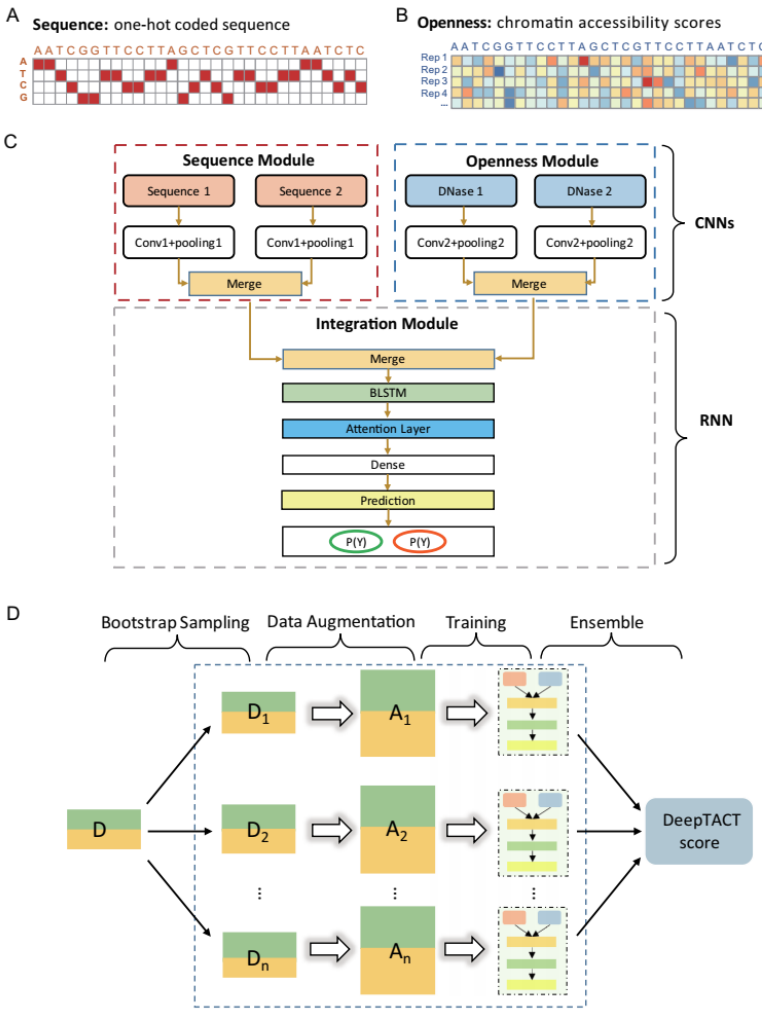
Example: Basset

- Predict chromatin accessibility (DNase-seq) from DNA sequences.

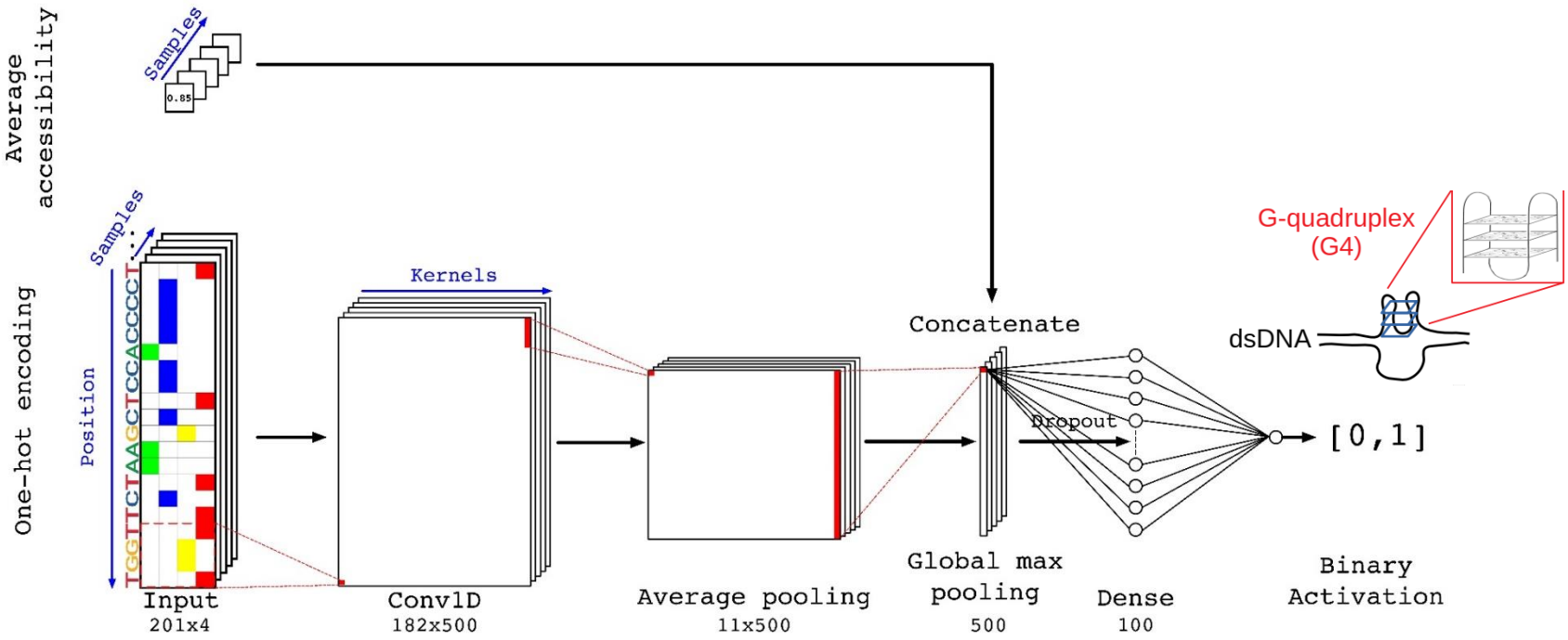


Example: DeepTact

- Predict long-range contacts (Hi-C) from DNA sequences and chromatin accessibility.



Example: DeepG4



- Predict cell-type specific G-quadruplex structures given the DNA sequence and chromatin accessibility.

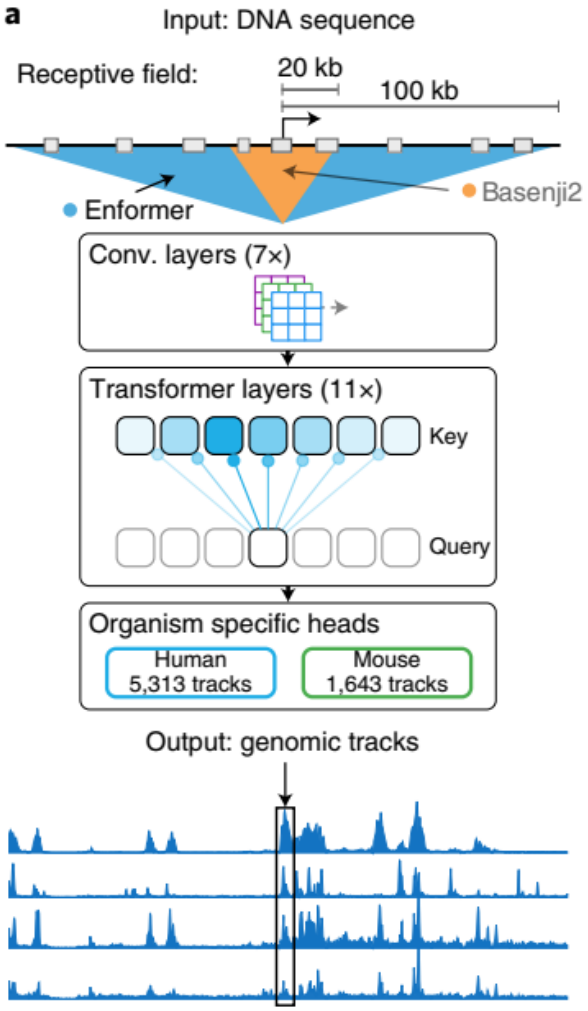
Rocher, Genais, Nassereddine and Mourad. PLOS Comp Bio 2021.

Example: Enformer

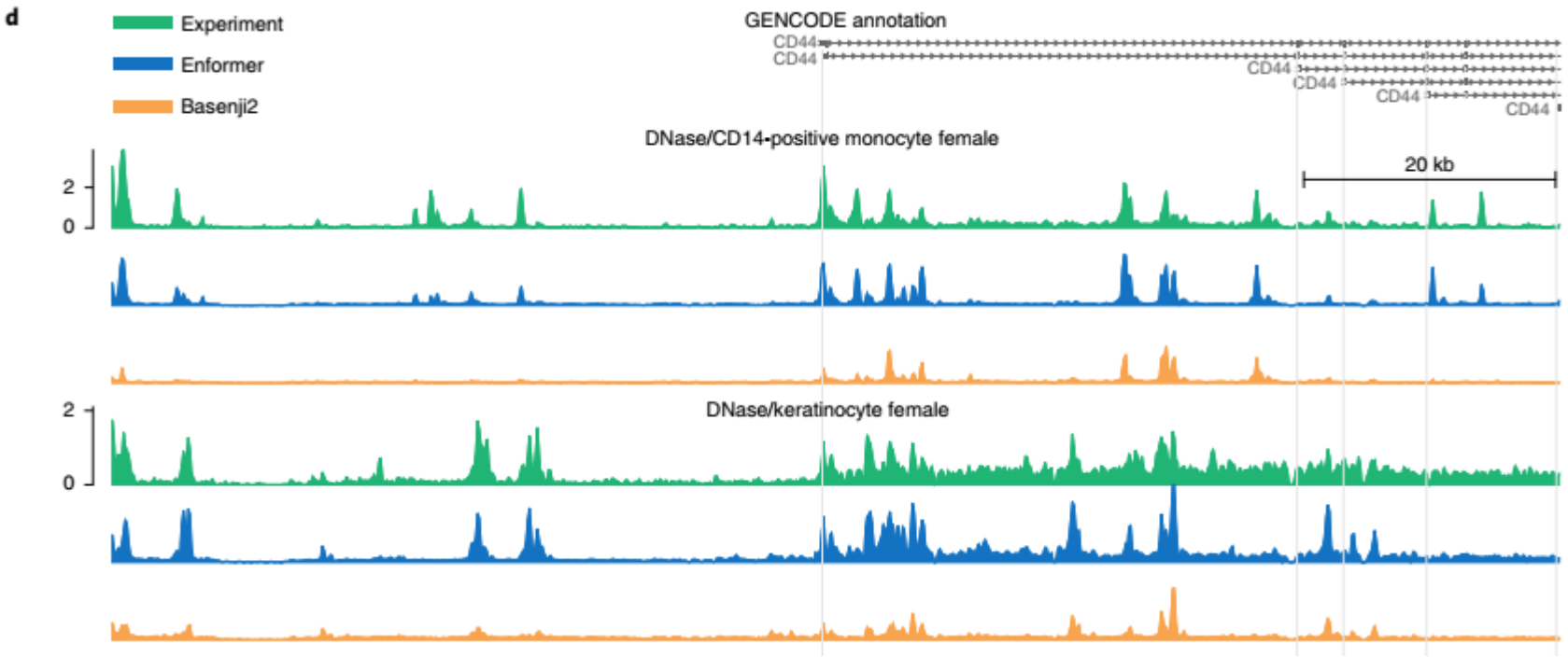
- Enformer takes a sequence of 200 kb, and predicts 5313 human experiments and 1643 mouse experiments!

- Big model:

>200 million parameters!

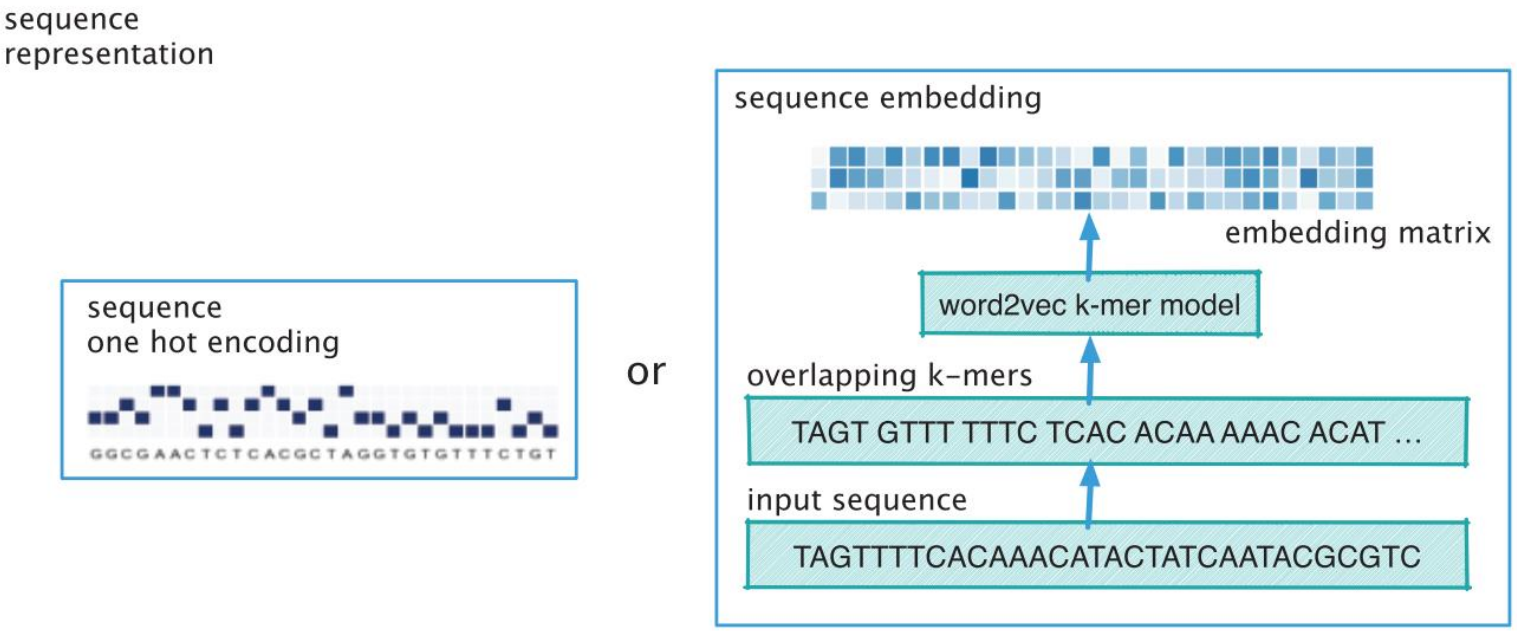


Example: Enformer performance



- Very good predictions of CAGE-seq experiments (gene expression) only from DNA sequence.

Two options for sequence representations

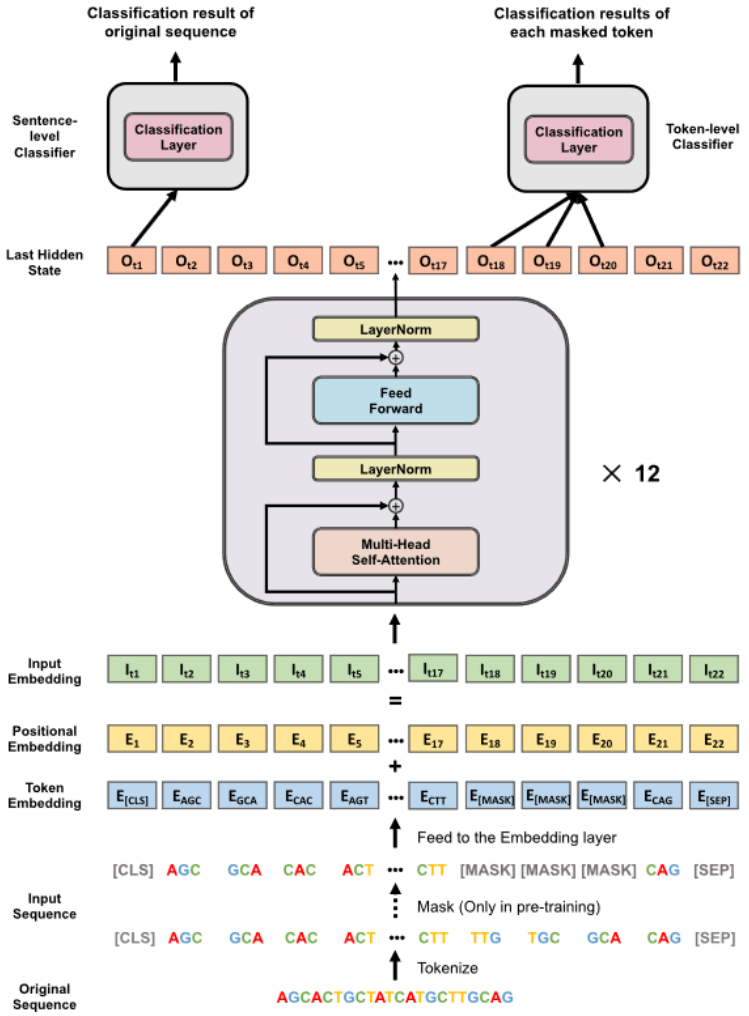


- 2 options:
  - One hot encoding followed by 1D-convolution,
  - Overlapping k-mers followed by an embedding (as word2vec).

Example: DNABERT

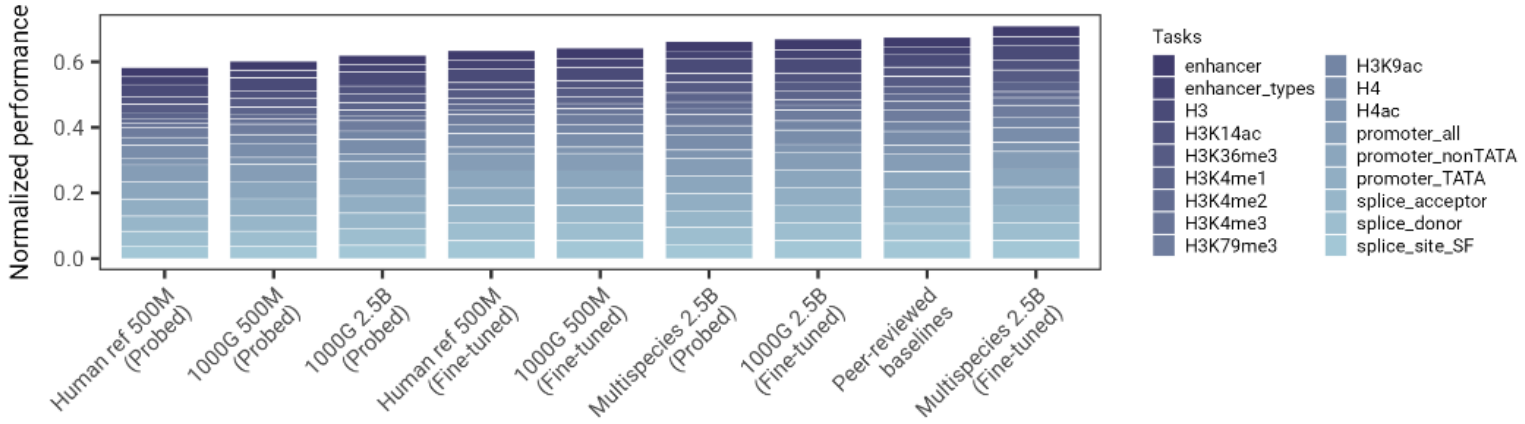
- The self-attention model DNABERT is trained by masking some kmers in the DNA sequence and then by trying to predict them using the other k-mers in the DNA sequence (context).

- At the end, the model provides features that encode DNA sequences in a very efficient way for any predictive task.



Example: Nucleotide Transformer

- BERT model trained not only from the human genome (3.3 Gb) like DNABERT, but trained from thousands of human and animal genomes!



- The prediction performance increases with the number of genomes used during training.

Assessing the impact of a SNP

- Easy way to assess the impact of a SNP:
  - First train a model on data (RNA-seq, ChIP-seq, ATAC-seq)
  - Predict the value for a sequence with the reference allele **C**:
    - ATGTAGTGGGTACC**C**TGTGTAGAAGCCA
  - Predict the value for a sequence with the reference allele **T**:
    - ATGTAGTGGGTACC**T**TGTGTAGAAGCCA

Assessing the impact of a SNP : classification setting

- $\Delta score = \big(p(seq\ alt) - p(seq\ ref)\big) * max(0, p(seq\ alt), p(seq\ ref))$

Where $p(seq\ alt)$ is the predicted probability of binding for the sequence with the alternative allele, $p(seq\ ref)$ is the predicted probability of binding for the sequence with the reference allele.

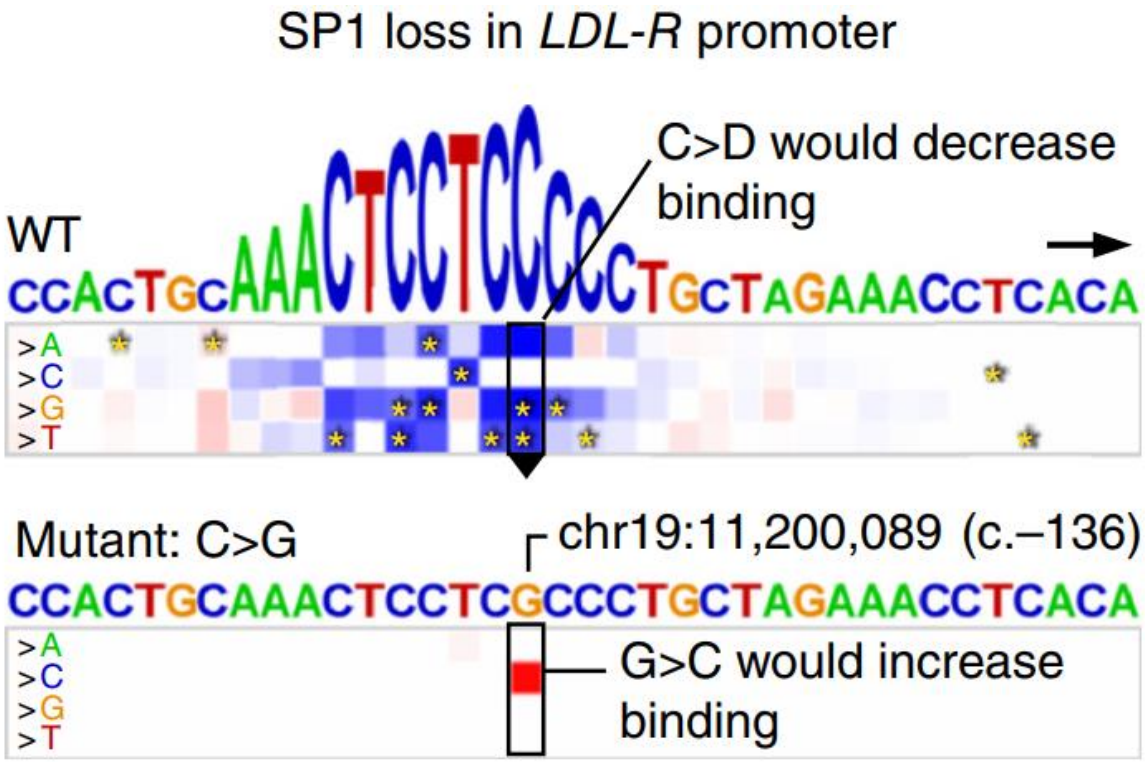
$\Delta score$>0 means that the alternative allele increases the binding, whereas $\Delta score$ <0 means that the alternative allele decreases the binding.

Assessing the impact of a SNP : regression setting

- $\Delta score = \left(score(seq\ alt) - score(seq\ ref)\right)$

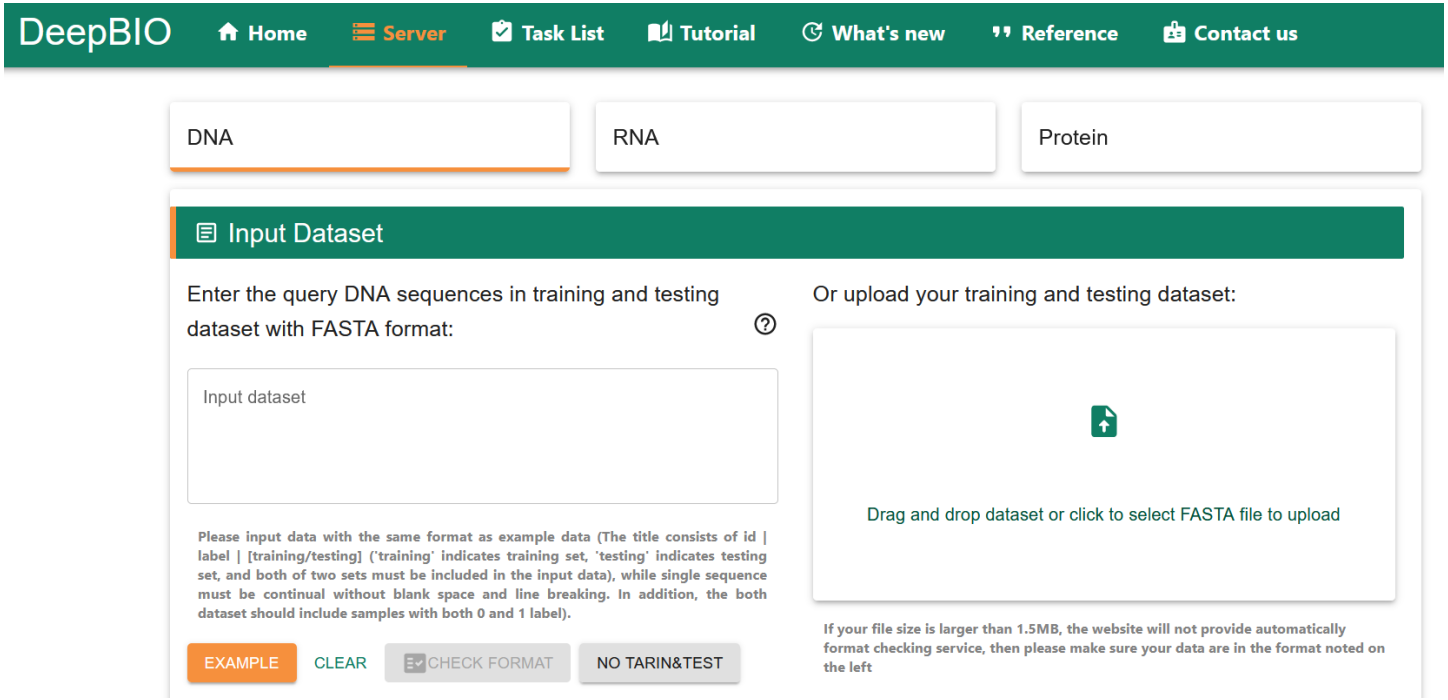
- In a regression setting, we simply look at the difference between the two scores.

Mutation map



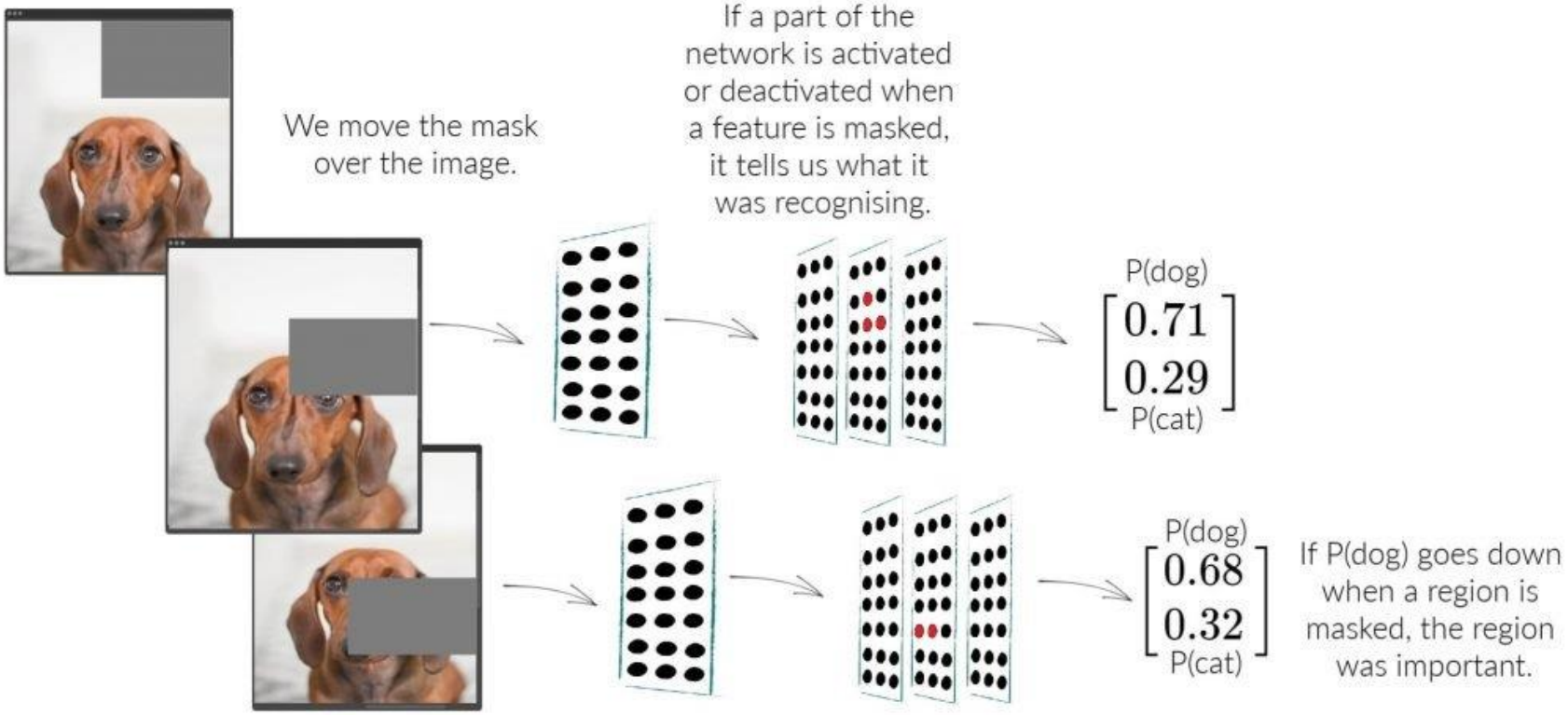
- For a given sequence, all possible mutations can be done and their impact ($\Delta score$) can be measured and then visualized.

Train deep learning models using a webserver



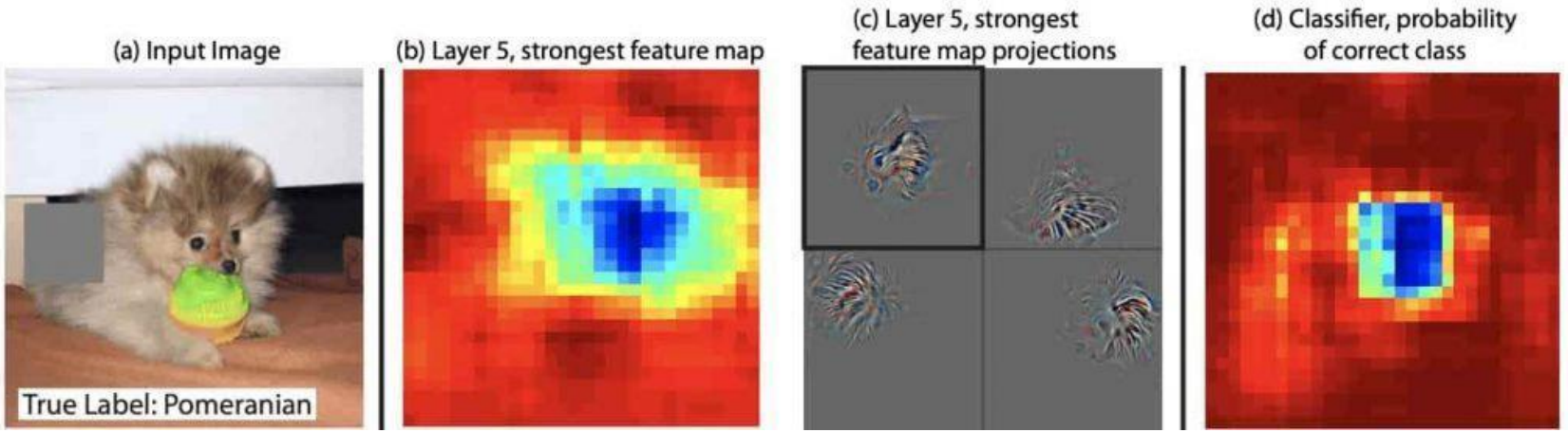
- Copy-paste fasta files containing sequens for training and testing a deep learning model.

Model interpretability for images



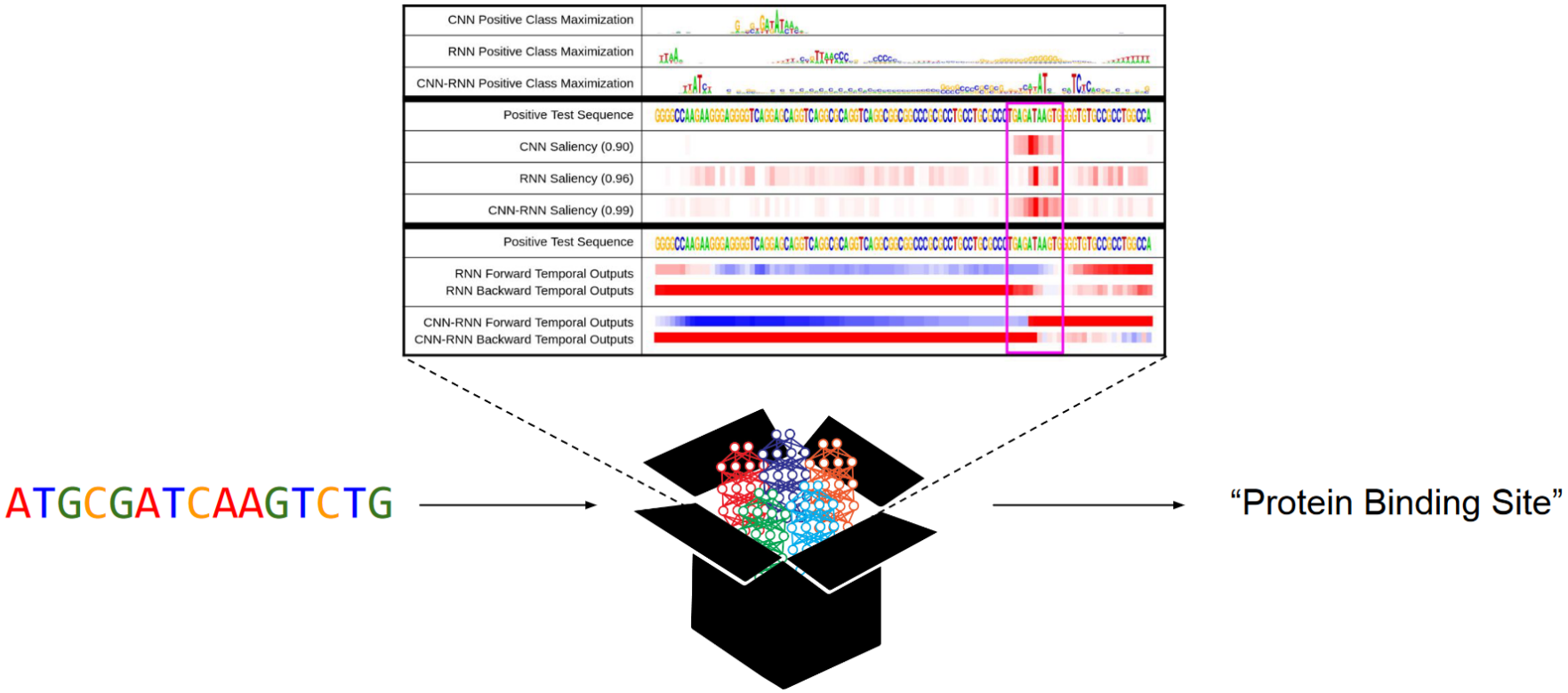
- Perturbing the input (for instance with a mask) can reveal from the output (here P(dog) is decreasing) which part of the image is important.

Model interpretability for images



- Another way is to look at some deep layer and a given neuron, where values are high.
- Here the neuron is « activated » at the face of the Pomeranian. This means the face is the most important feature to predict the dog race.

Model interpretability: saliency map



- Saliency maps are used to identify with the DNA sequence which parts are the most important predictors.

How to compute saliency

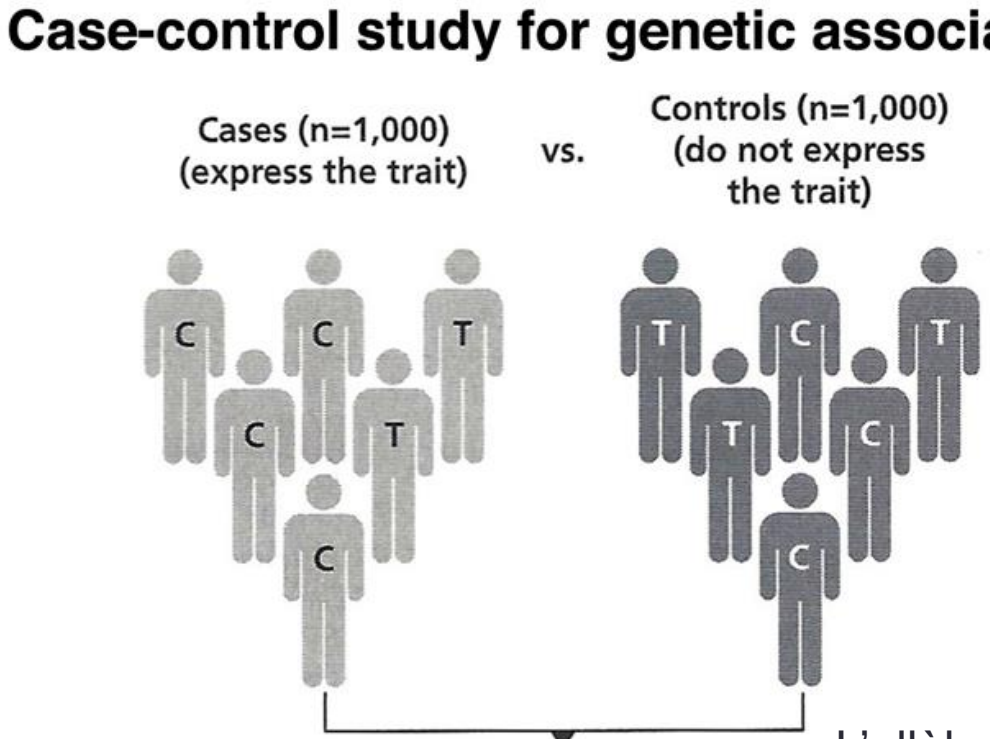
- For a given DNA sequence base:
  - $Saliency = \frac{d\ output}{d\ input}$

  - For a classification network, the output is a probability.
  - For a regression network, the output is the number of reads (quantitative signal).

APPLICATIONS

Predicting the impact of SNPs using Deep learning

Principe des Genome-wide association studies (GWAS)



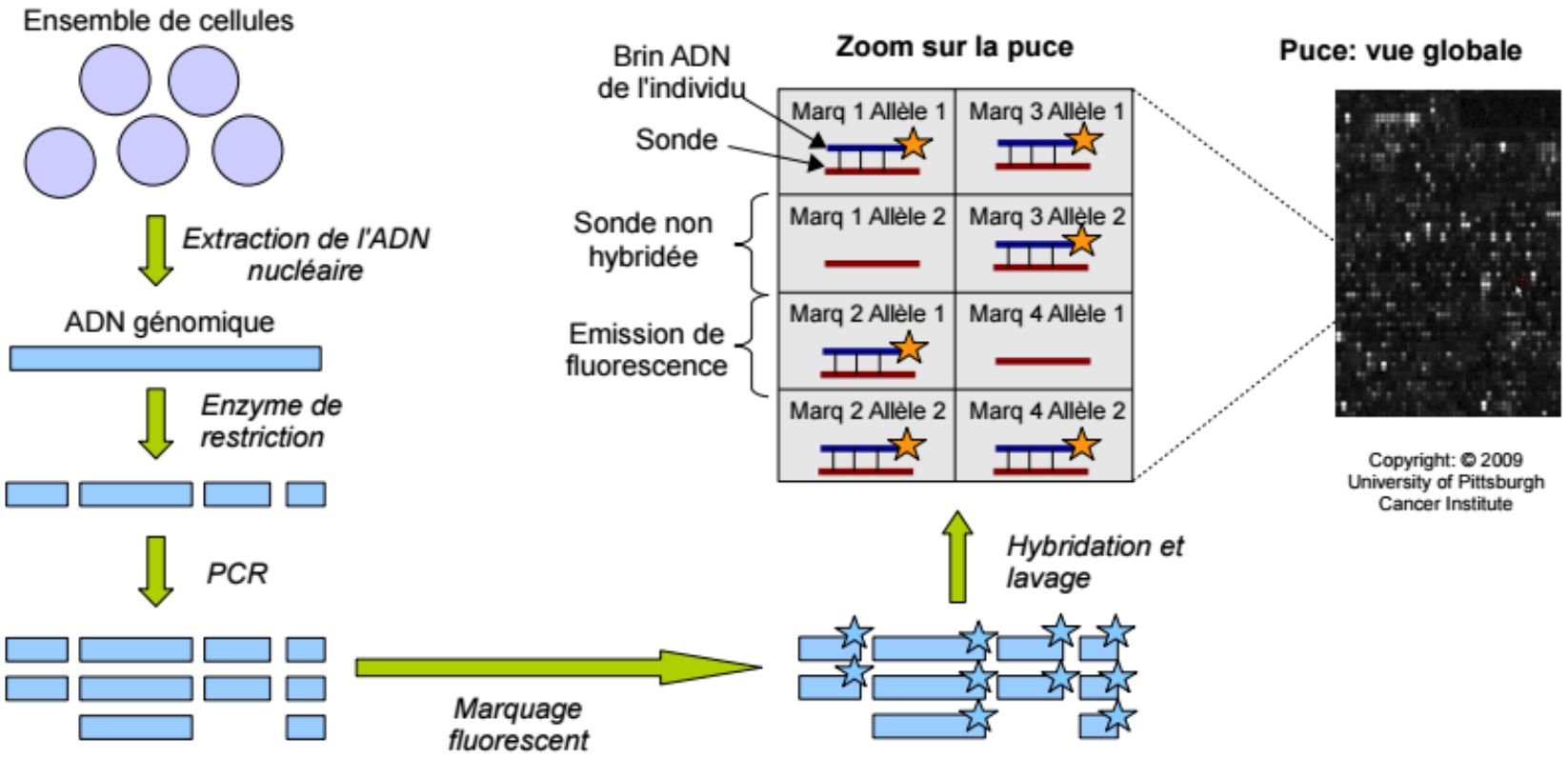
	C	T
Cases	62%	38%
Controls	49%	51%

L'allèle C est associée à la maladie (le trait).
On parle **d'association statistique**.

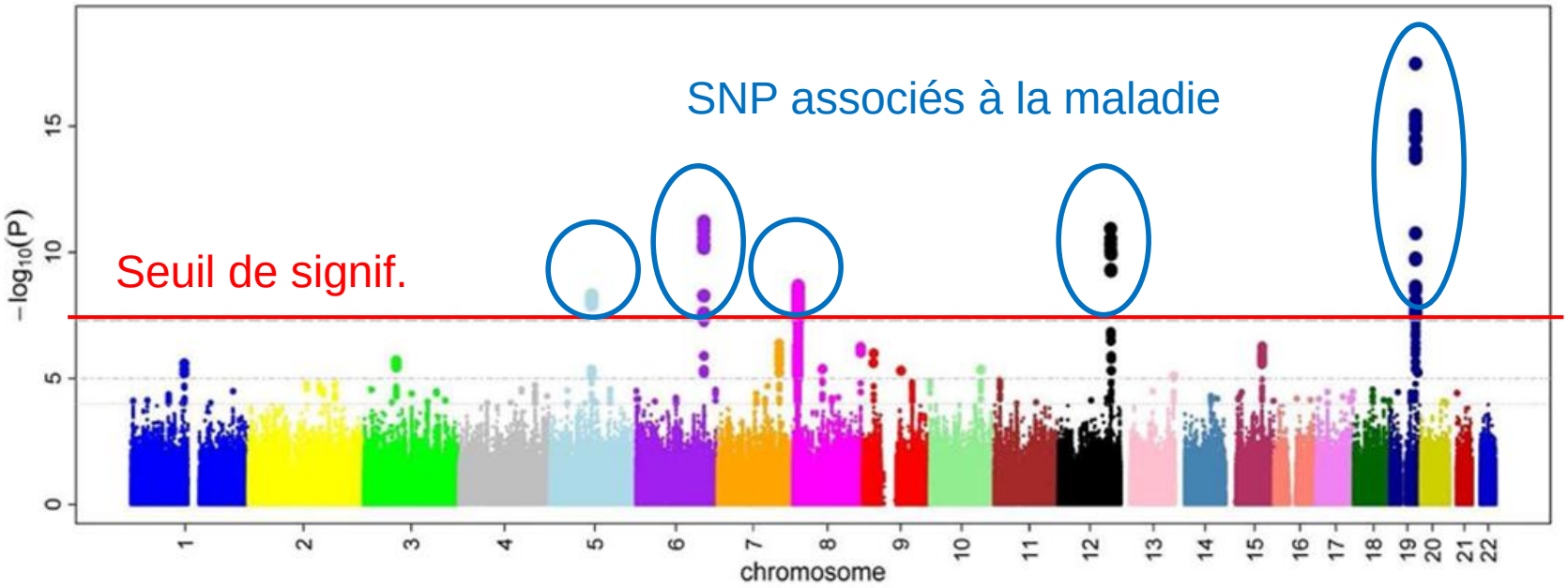
Single nucleotide polymorphism (SNP)

- Intérêt d'étudier les SNP:
  - Mutation fréquente: 1 tous les kb, ce qui permet cartographier de façon précise le génome;
  - Un individu en possède quelques millions;
  - Les SNP permettent de capturer la majorité de la variation du génome, grâce au déséquilibre de liaison;
  - Facile à génotyper de façon automatique (puce à ADN, NGS);
  - Coût de génotypage très faible;

Puce à ADN pour SNP



Résultats d'une GWAS



- Chaque point est un SNP, l'abscisse est la coordonnée génomique et l'ordonnée est le niveau d'association -log10(p-valeur). Résultats d'une GWAS sur la microcirculation, les points les plus hauts sont les SNP les plus souvent trouvés chez les individus atteints de constriction des petits vaisseaux sanguins.

SNPs affecting molecular phenotypes : gene expression, protein binding, …

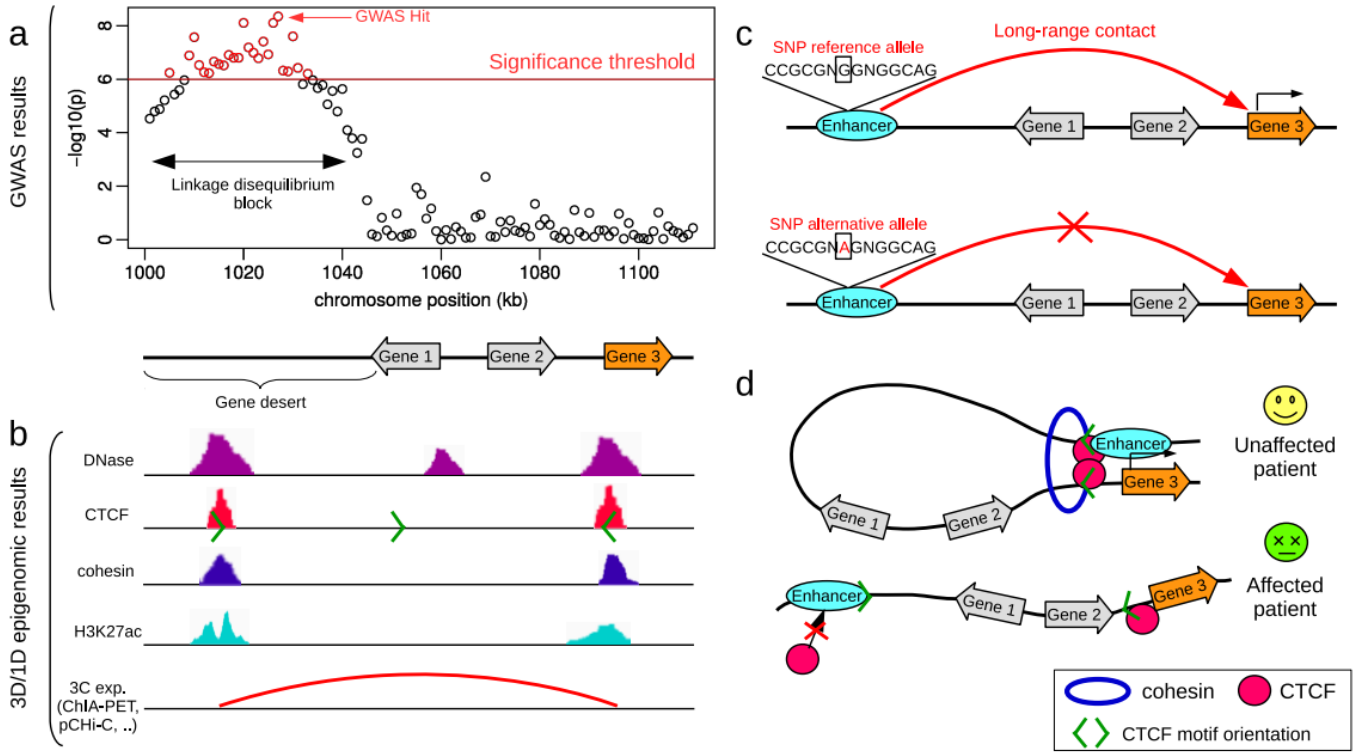
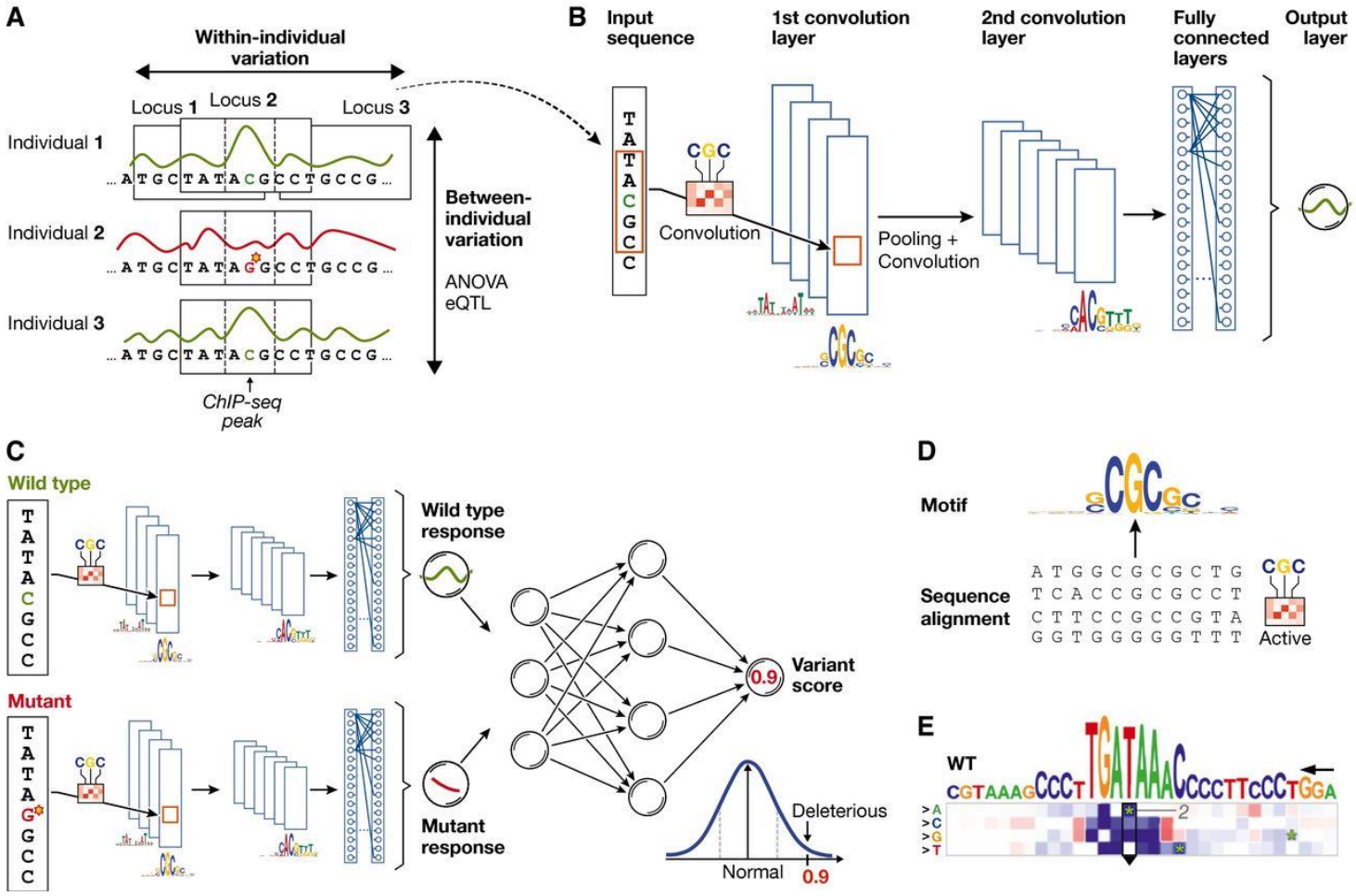


Figure 4. Post-GWAS analysis of risk loci using 3D genome information. a) GWAS mapping of single nucleotide polymorphisms (SNPs) associated with a complex disease. b) Mapping of epigenomic features including DNase, CTCF, cohesin, H3K27ac and 3C-based experiments. Green arrows represent CTCF motif orientations. c) 3C-based experiment results depending of the SNP allele. d) Functional model explaining the non-coding SNP effect.

Predict the impact of mutations

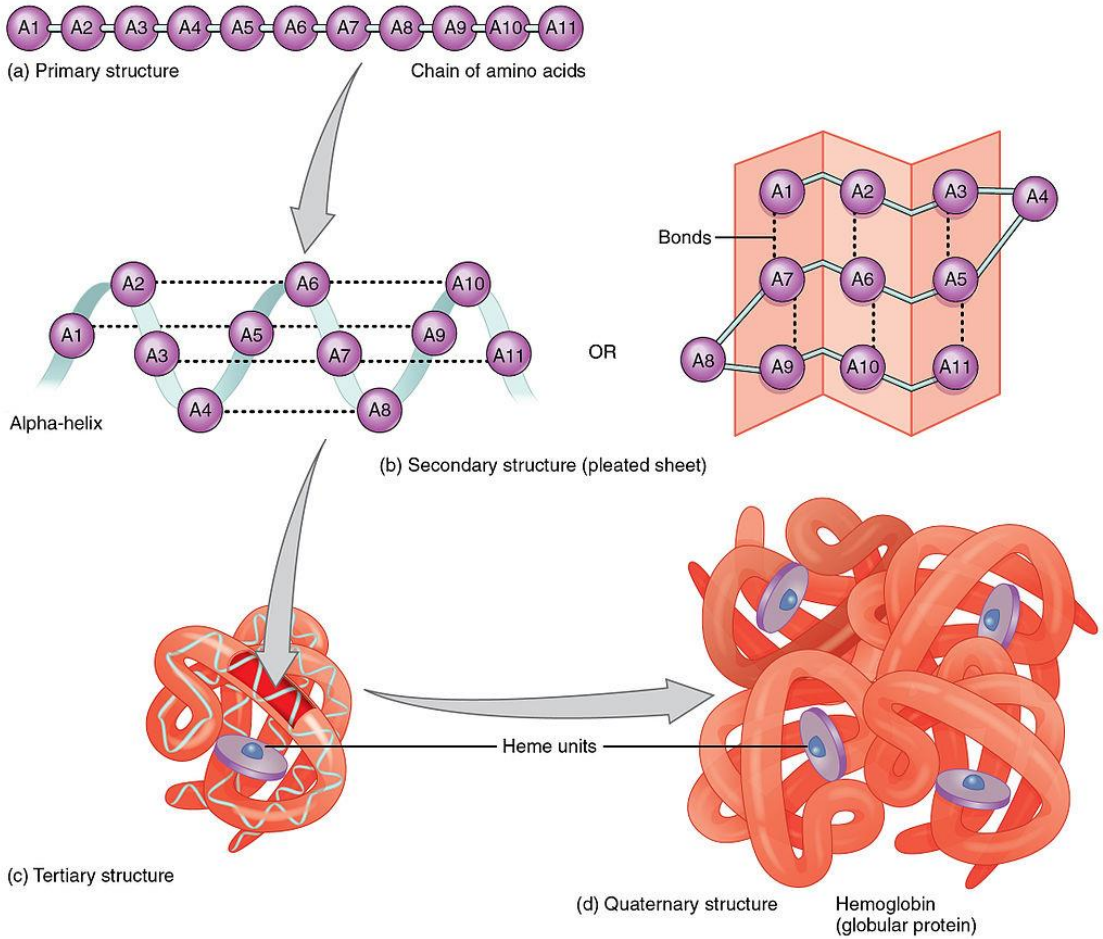


OTHER APPLICATION OF DEEP LEARNING

Prediction of protein 3d structures

Protein folding as a central problem in biology

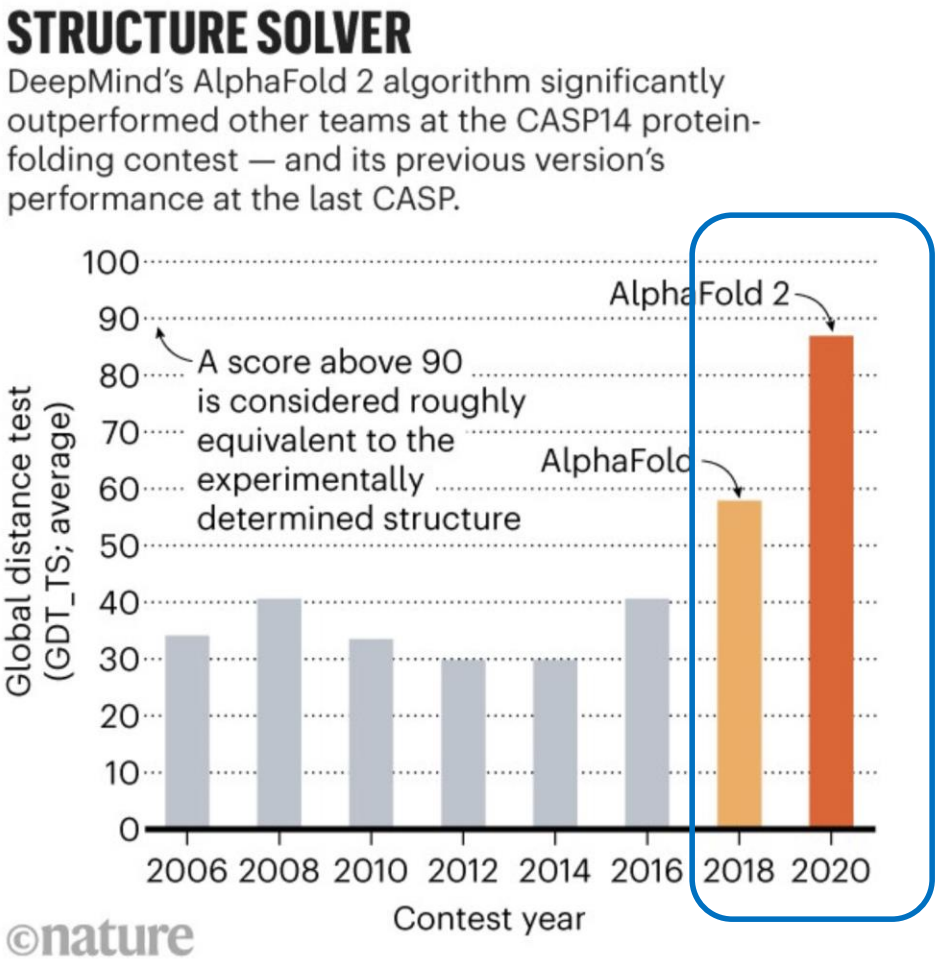
- Protein folding is the physical process by which a protein chain is translated to its native 3D structure, by which the protein becomes biologically functional.



Protein folding as (NP) hard problem to solve

- Levinthal's paradox:
  - Trying all possible 3D conformations for a protein would take an astronomical amount of time to do so!
  - Thus, heuristics have been developed for decades to predict the folding of a protein in short time.

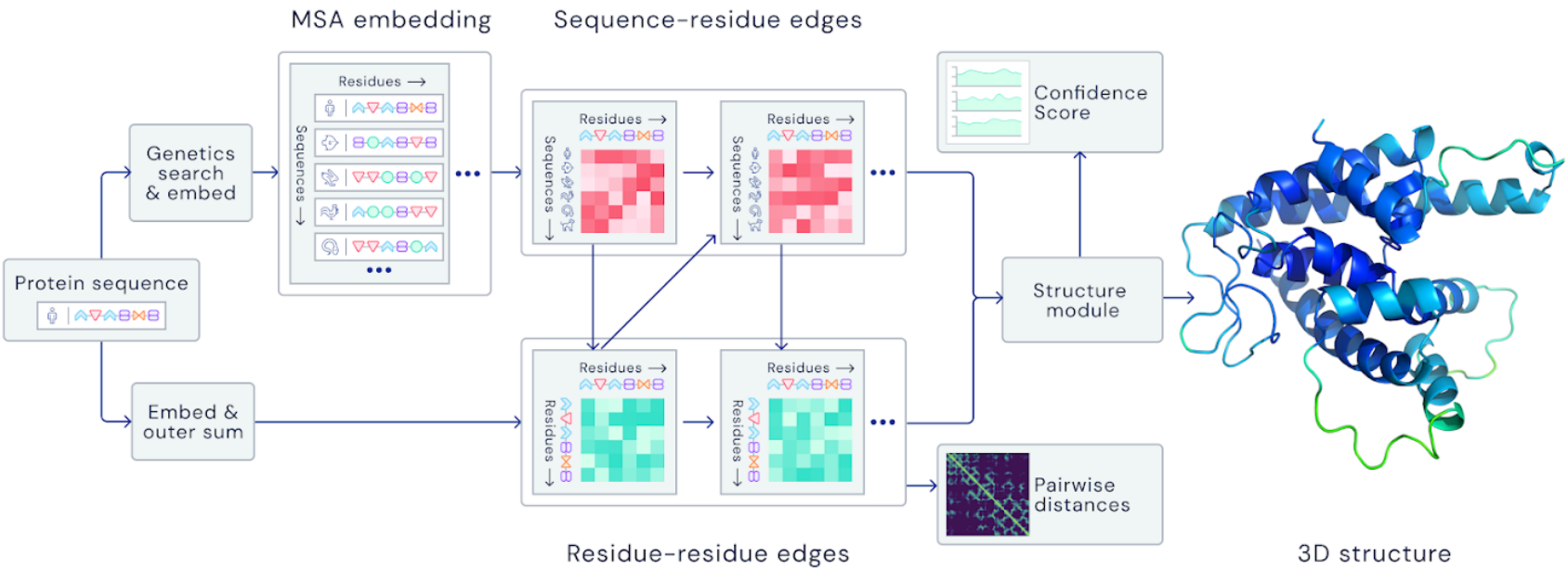
Success of deep learning to improve prediction performance of protein 3D structures



Deep learning with AlphaFold1 and 2

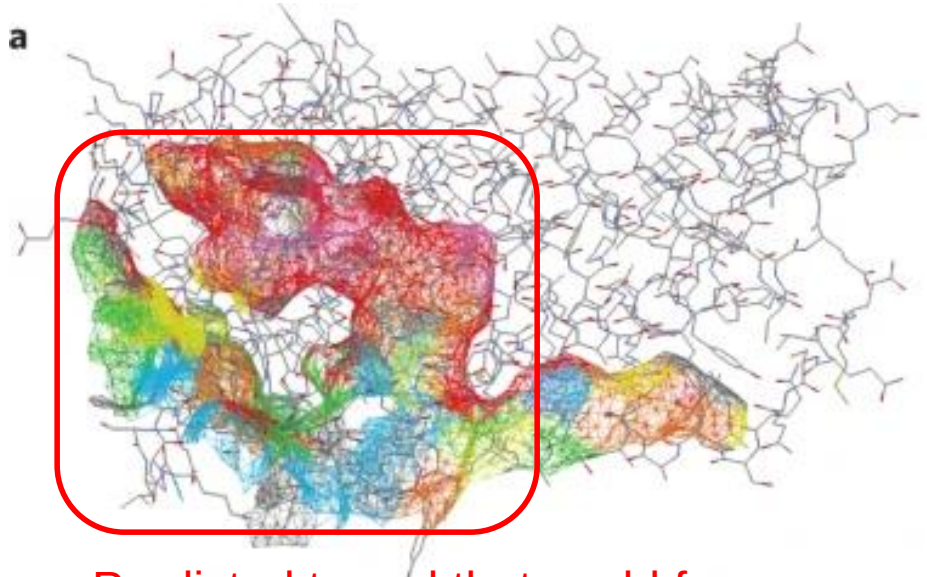
Example: AlphaFold2

- AlphaFold 2 from DeepMind (Nature, 2021).

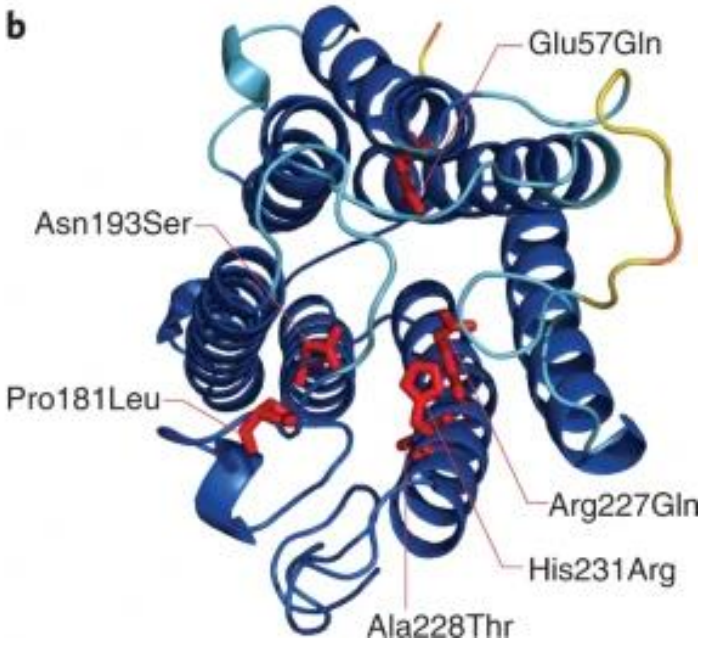


- AlphaFold2 is thought to have resolved the folding of most proteins (70%).

Applications in medicine



Predicted tunnel that could form the basis for further drug design



- A) Predicted structure of a protein targeted by many drugs.
- B) Known disease-associated mutations of the protein lie in the predicted tunnel.

OTHER APPLICATION OF DEEP LEARNING

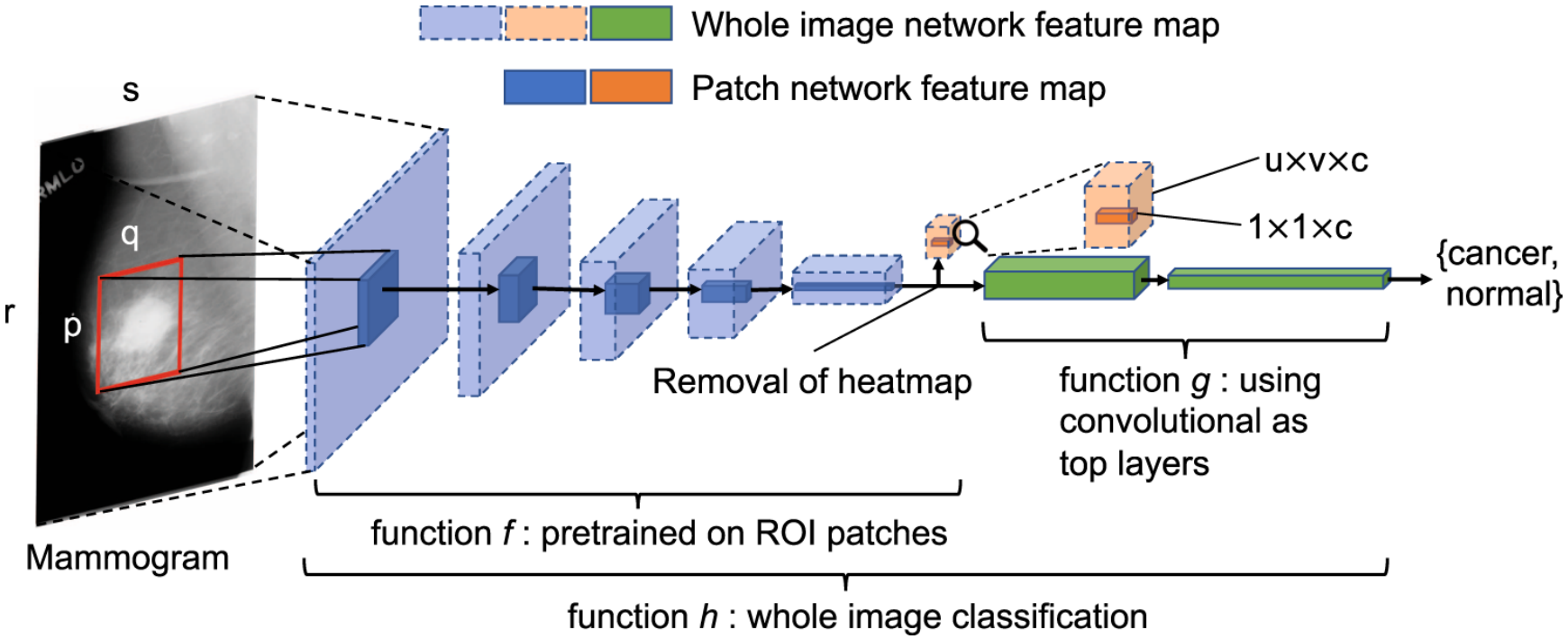
Image Classification for cancer

Imaging allows to identify potential breast cancer

- Breast cancer is the second leading cause of death for women, so accurate early detection can help decrease breast cancer mortality rates.

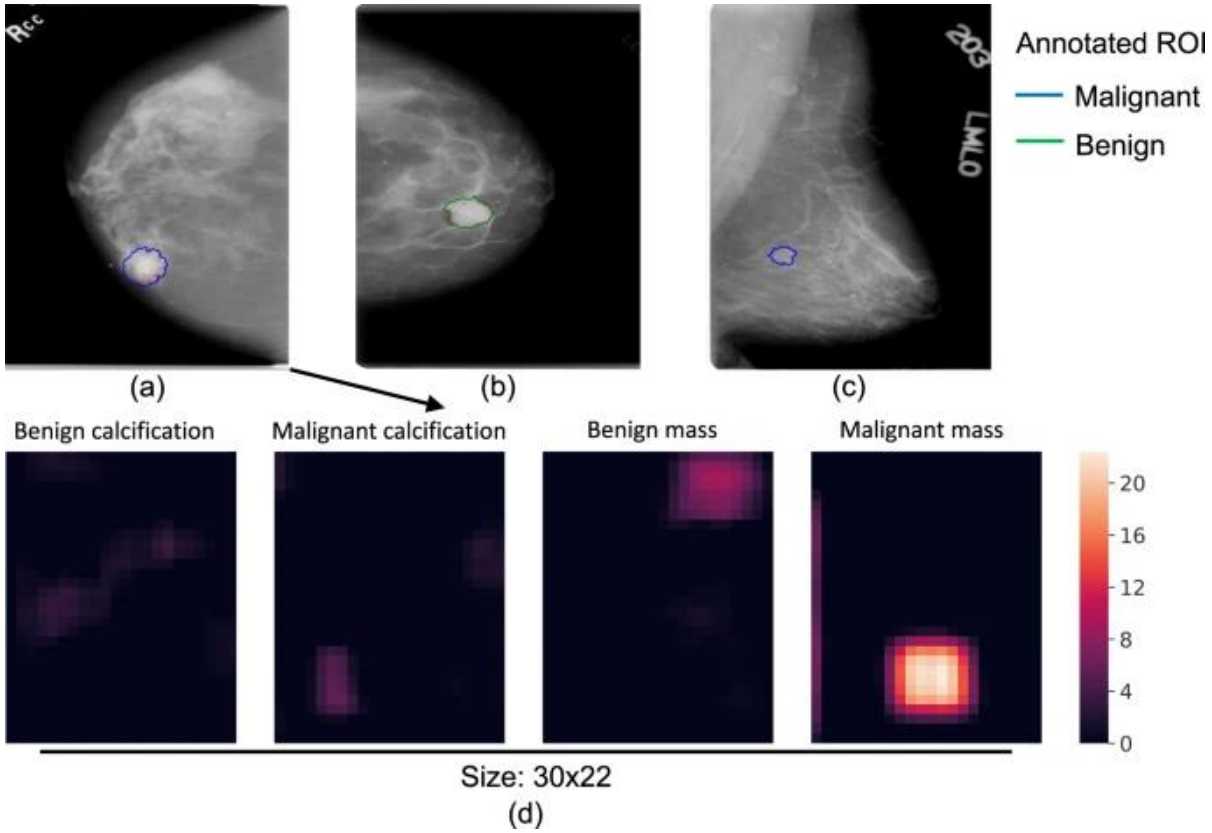
- Medical images are sources of information relevant to the detection and diagnosis of various diseases and abnormalities.

Detection of breast cancer from imaging



- Convolutional network to predict breast cancer.
- Li Shen et al. Scientific Report 2019, 29;9(1):12495.

Deep learning can detect the exact region of the tumor



Deep learning can distinguish between malignant and begnin turmours.