Architectures.

Mainly based on

-Kaplan's notes
-"A high-bias, low-variance introduction to Machine Learning for physicists"
-https://towardsdatascience.com/transformers-141e32e69591
-Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (Aurélien Géron)

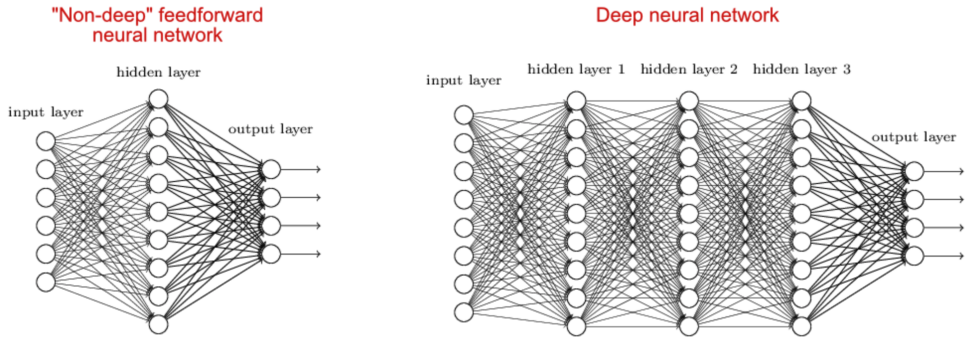
Philipp Englert

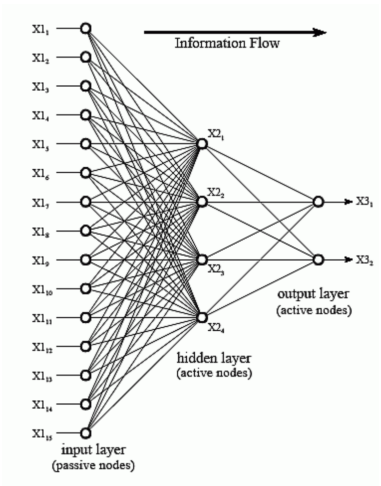
Workshop Seminar 2020/2021

Hamburg, 26 January 2021

How to design a Neural Network?

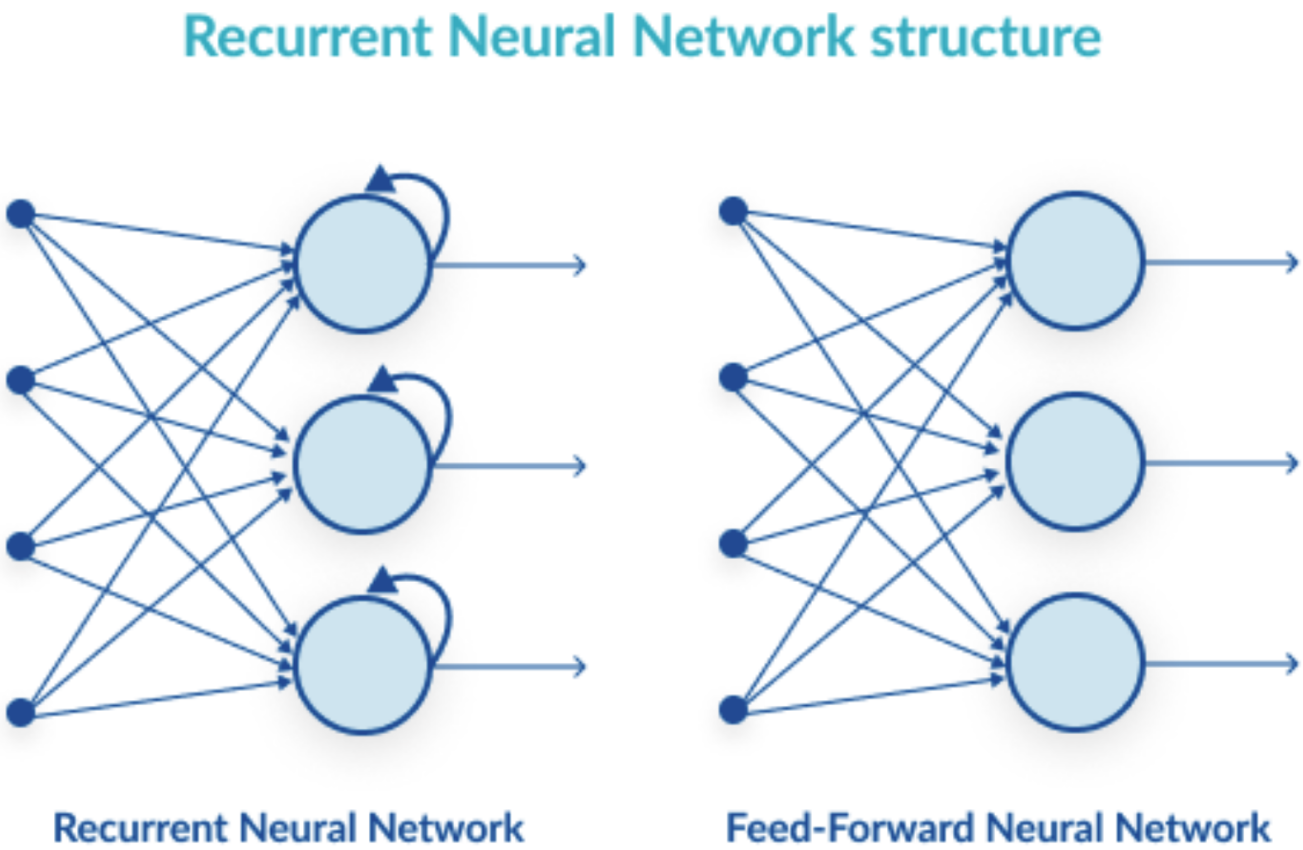
- So far: mostly had fully connected, feed-forward NN in mind
- Number of layers? Number of nodes per layer? Activation functions?





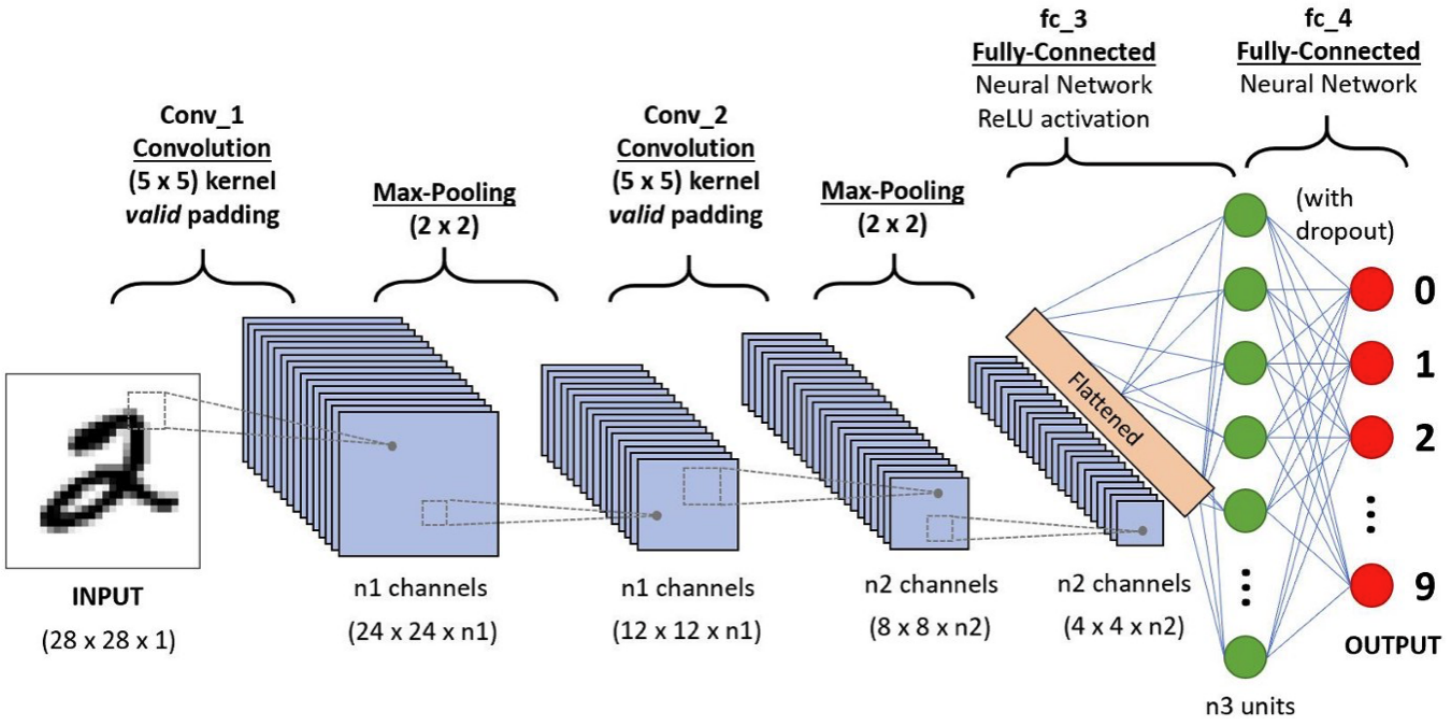
How to design a Neural Network?

- There's more to it than just that!
- For example:



How to design a Neural Network?

- There's more to it than just that!
- For example:



Design Considerations

DESY

Important design considerations

Can we represent the function we want to learn with this architecture?

- necessary but not sufficient for choosing a suitable architecture
- usually not a big concern
- Universal Approximation Theorem: NN with > 0 hidden layers can approximate any continuous, multi-input/multi-output function with arbitrary accuracy

Important design considerations

Will the relevant information be able to propagate through the whole network?

- especially problematic for deep networks
- gradients can vanish or explode
- to help the information flow:
  - choose suitable activation function
  - initialize weights properly
  - 'whiten' data, i.e. decorrelate and normalize, such that variance=1
  - residual connections
- does the information propagate in a small number of steps?

Important design considerations

Does the network have suitable symmetries for the problem of interest?

- choice of architecture biases the distribution of possible NN parameters
- choose a suitable architecture for the problem right away!
- otherwise: waste of computational capacity by forcing the NN to learn this information

DESY

Important design considerations

Other considerations

- can the network efficiently 'forget' irrelevant information?
- can the Neural Network be parallelized?
- ...

Information propagation

DESY

Information Propagation

Activation functions

- relatively little is understood about this subject
- specific choice of activation function is not that important, as long as it's not 'pathological'
- for deep NN's the activation functions in intermediate layers should not saturate on both sides to avoid vanishing gradients

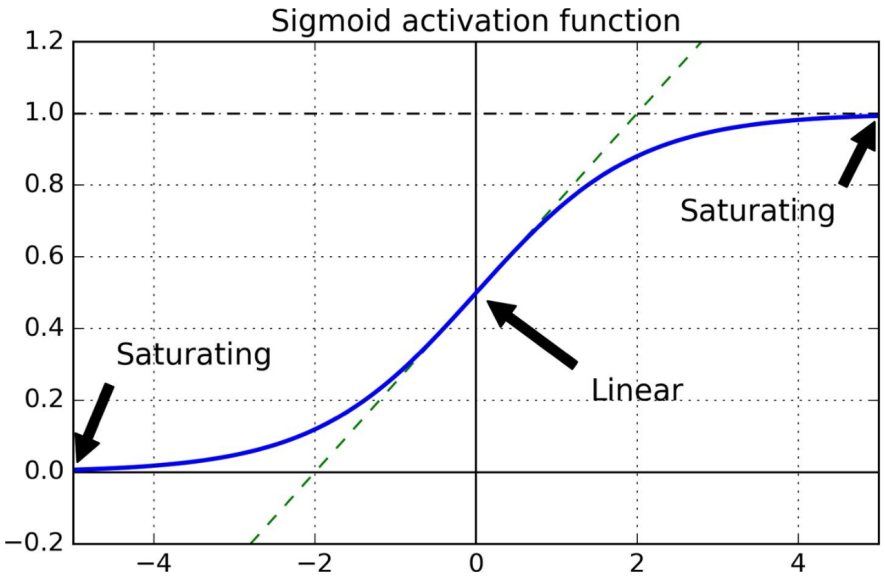
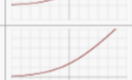


Figure 11-1. Logistic activation function saturation

Information Propagation

Activation functions

- abundance of options:

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$

- default choice: ReLU (no saturation on one side, cheap to compute)
- leaky ReLU is not very common and does not seem to help much with training
- identity if you just want to perform a linear transformation
- tanh, sigmoid, etc. if you want outputs in a fixed range

Information Propagation

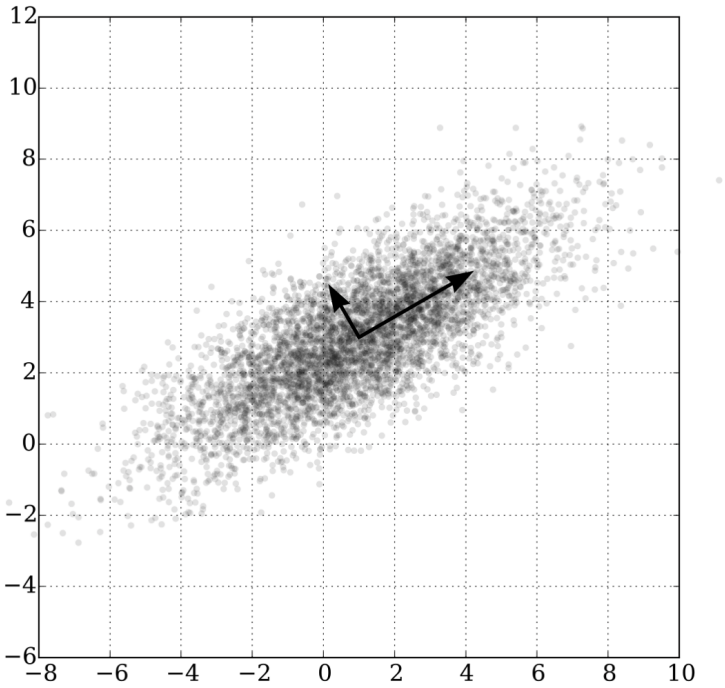
Whitening

- for better information propagation: decorrelate the input data and normalize it ('whitening')
- decorrelation e.g. with Principal Component Analysis (PCA)
- normalization with e.g. Layer Norm or Batch Norm
- smoothens loss landscape $\rightarrow$helps with training

Information Propagation

Whitening - PCA

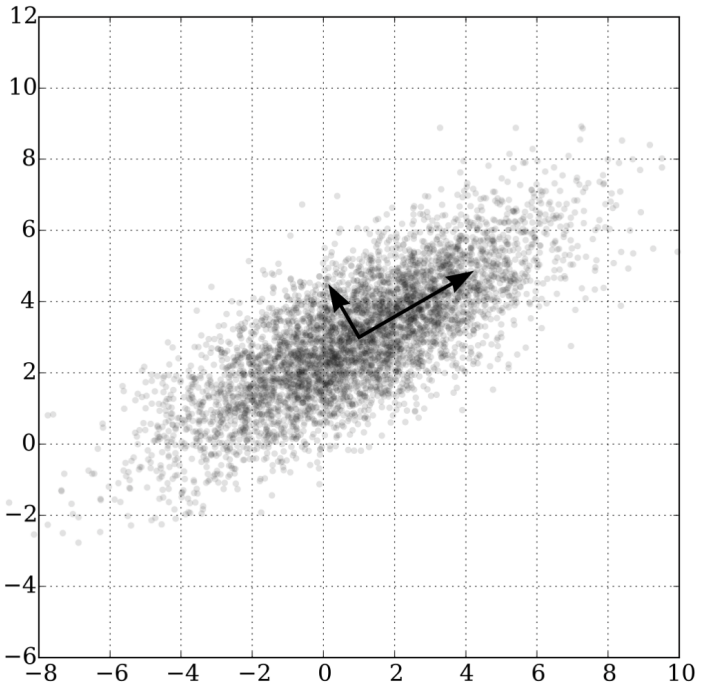
- i-th principal component: direction of a line that best fits the data, while being orthogonal to the first $i-1$ principal components



Information Propagation

Whitening - PCA

- decorrelation: decompose data in terms of principal components
- can reduce dimensionality by only taking the first n principal components



Information Propagation

Whitening - Normalization

- also important: activations should follow "reasonable" distribution (order 1 mean and standard deviation) $\rightarrow$ normalize weights or activation functions
- more common: normalizing activation functions

DESY

Information Propagation

Whitening - Normalization

- Activation i in Layer n: $X_n^i = ReLU(W^{ij} X_{n-1,j} + b^i)$
- Layer norm:
  - mean: $\mu_n = \frac{1}{L_n} \sum_i X_n^i$, L_n: number of neurons in layer n
  - variance: $\sigma_n^2 = \frac{1}{L_n} \sum_i (X_n^i - \mu_n)^2$
  - normalize: $X_n^i \to \hat{X}_n^i = \frac{X_n^i - \mu_n}{\sigma_n}$
  - modified definition of the layer
- Batch norm:
  - average over samples in batch for each neuron i
  - changes during training
  - only fixed after freezing the model (after training)

Information Propagation

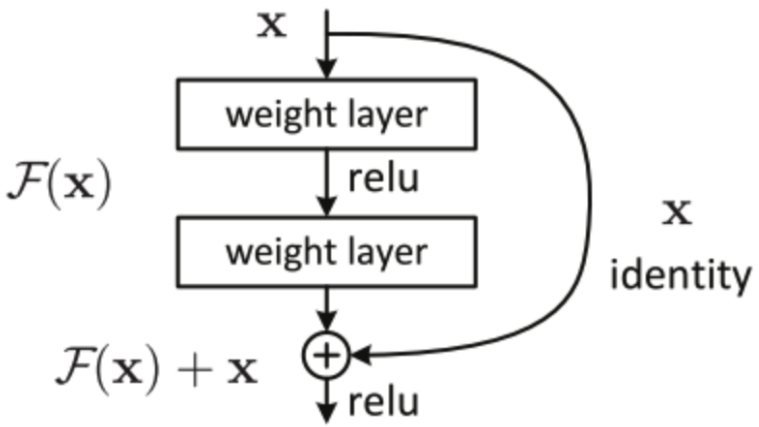
Initialization

- need to initialize weights randomly (following some distribution)
- risk: $\mathrm{Var}(X_n^i)$ somehow depends on $\mathrm{Var}(W^{ij}) \rightarrow$if $\mathrm{Var}(X_n^i)$ is too large or too low, gradients could blow up/vanish already in the beginning of training
- need to choose variance of weight initialization wisely
- common and good choices: $\mathrm{Var}(W^{ij}) = \frac{2}{L_n}$, $\mathrm{Var}(W^{ij}) = \frac{2}{L_{n-1}}$

Information Propagation

Residual Connections

- we can let the layers compute $X_{n+1} = X_n + F(X_n)$ instead of $X_{n+1} = F(X_n)$, where F is some non-trivial function (e.g. matrix multiplication & activation function)
- can help the information to reach later layers, even if earlier layers are learning slowly
- helps learning functions close to the identity
- we can also let the information skip several layers, e.g.



Example Architectures

-

Convolutional Neural Networks

DESY

Convolutional Neural Networks (CNN)

The human visual cortex (naively)

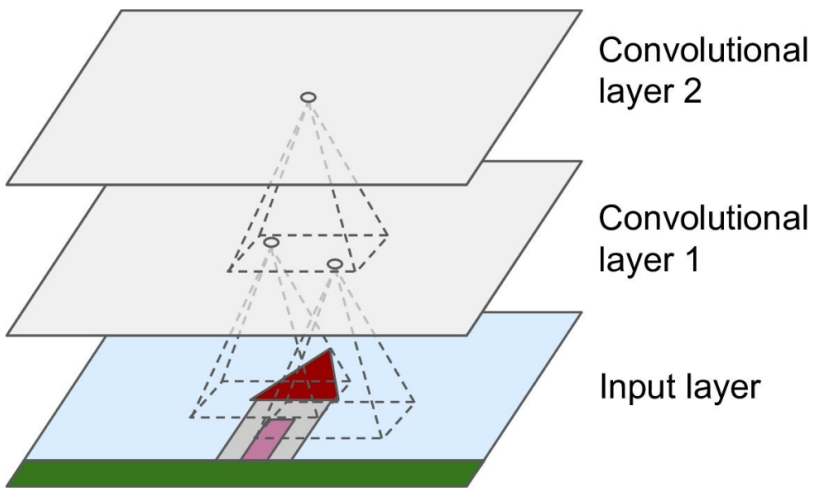
- many neurons in the human visual cortex have a *small local* receptive field
- receptive fields may overlap
- they collectively cover the whole visual field
- some neurons only react to higher level features, e.g. horizontal lines or vertical lines
- neurons with the same receptive field can react to different features
- some neurons react to even more complex features (combinations of lower level features)

$\rightarrow$ this inspired Convolutional Neural Networks

Convolutional Neural Networks (CNN)

Convolutional Layers

- neurons in the first convolutional layer are only connected to a few pixels in a local receptive field
- same goes for the second layer, etc.
- this allows the NN to focus on low level features in the first layer and on higher level features in the deeper hidden layers

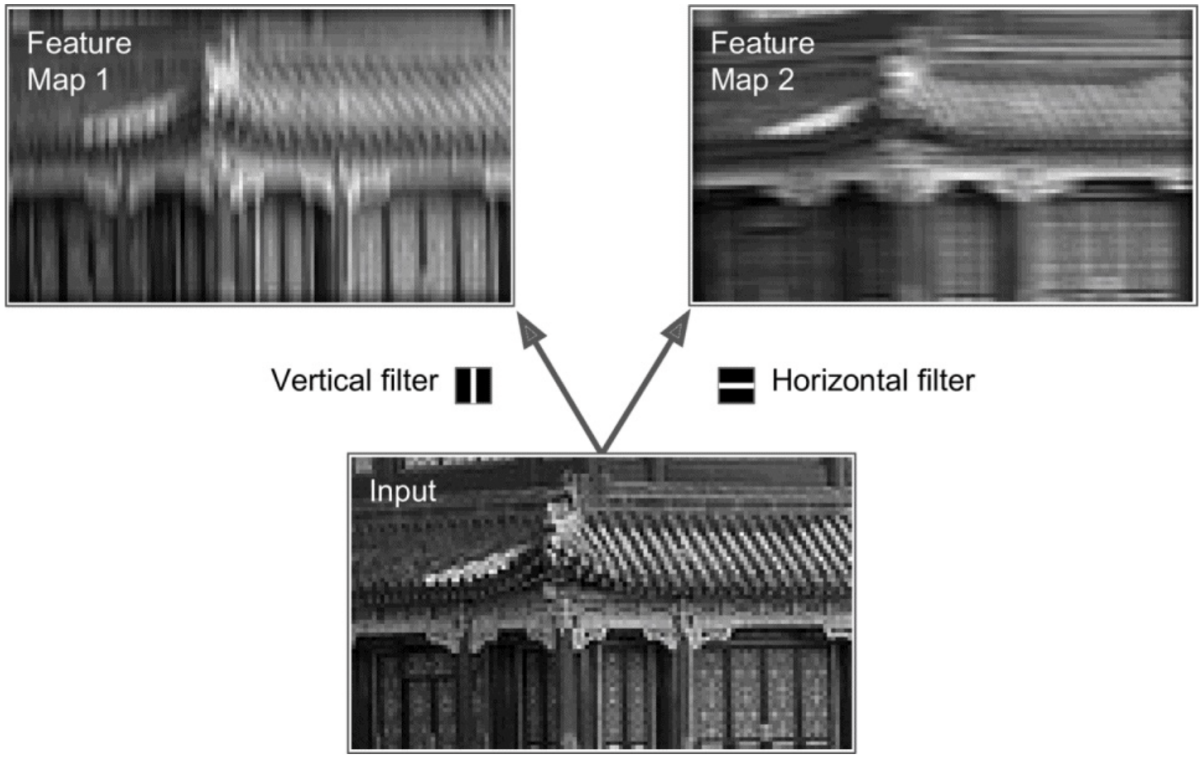


DESY

Convolutional Neural Networks (CNN)

Convolutional Layers

- each neuron in a specific layer has the same weights
- these weights are called *convolutional kernel* or *filter* of the layer



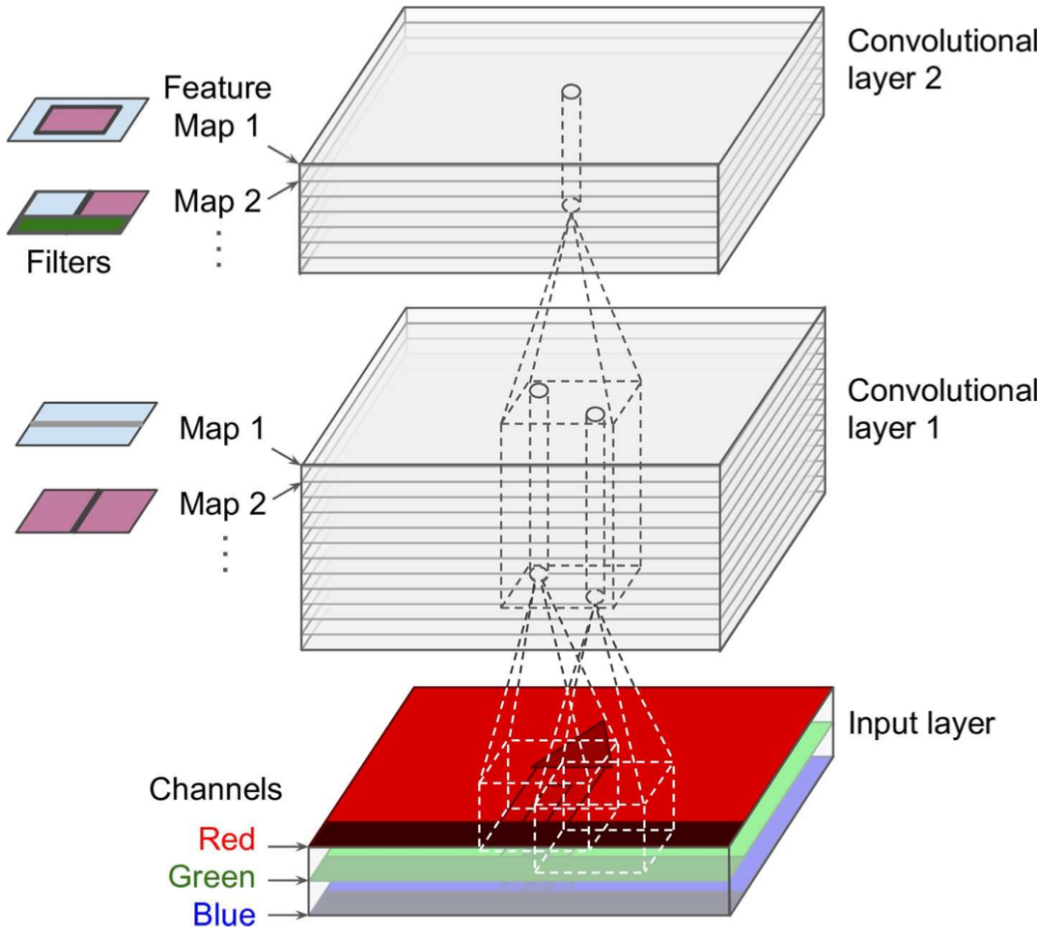
Convolutional Neural Networks (CNN)

Convolutional Layers

- filters are learned during training
- a layer of neurons with the same weights is called *feature map*
- a convolutional layer can consist of several feature maps
- a neuron in a specific convolutional layer is connected to the neurons in its receptive field in all the feature maps of the previous convolutional layer
- input layers can also consist of several sublayers (e.g. RGB)

Convolutional Neural Networks (CNN)

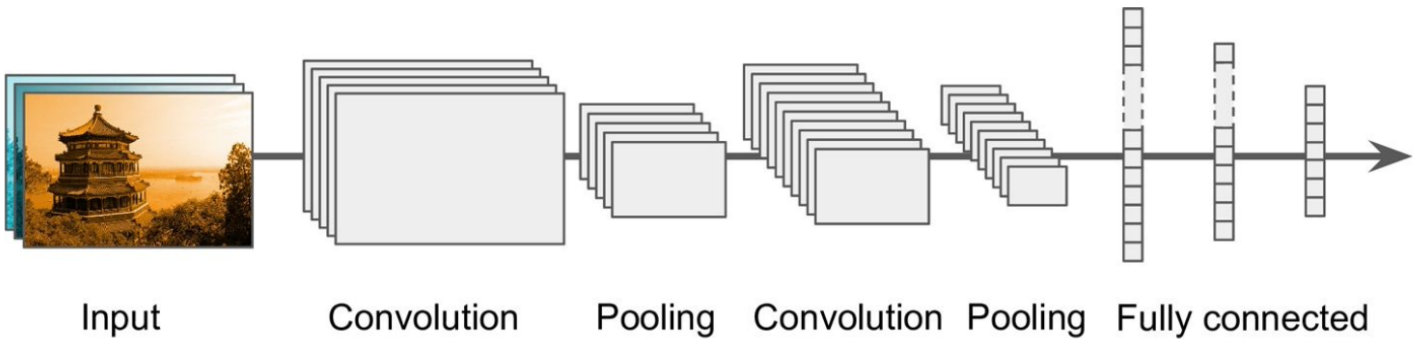
Convolutional Layers



Convolutional Neural Networks (CNN)

Pooling Layers

- typical CNN: alternates convolutional layers and pooling layers
- a pooling layer is like a convolutional layer that just aggregates its input neurons
- aggregation functions: e.g. maximum or mean of the input neurons
- typical CNN-architecture:



Example Architectures

-

Recurrent Architectures

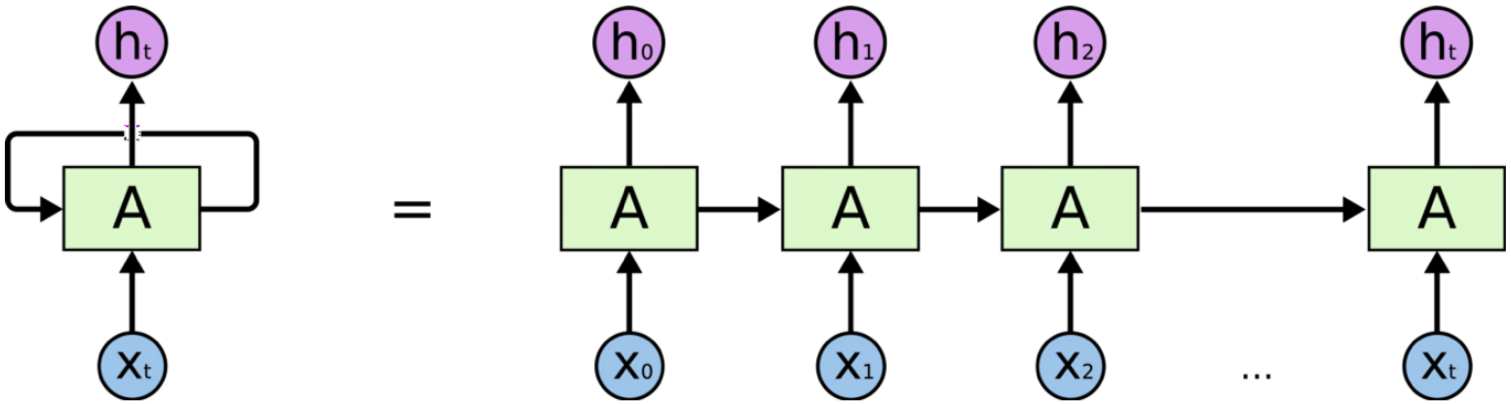
Recurrent Architectures

- some tasks require some sort of memory, e.g. sequence transduction ($\supset$ language translation, text-to-speech transformation, ...)
- e.g. when translating a word in a sentence, we need to remember how we translated the previous words
- solutions in Machine Learning:
  - $\rightarrow$ recurrent architectures (RNN, LSTM, ...)
  - $\rightarrow$ Convolutional Neural Networks (CNN)
  - $\rightarrow$ Attention and the Transformer
- **now**: recurrent architectures

Recurrent Architectures

Recurrent Neural Networks (RNN)

- RNNs contain loops that pass a hidden state ('memory') to the next iteration of the network:

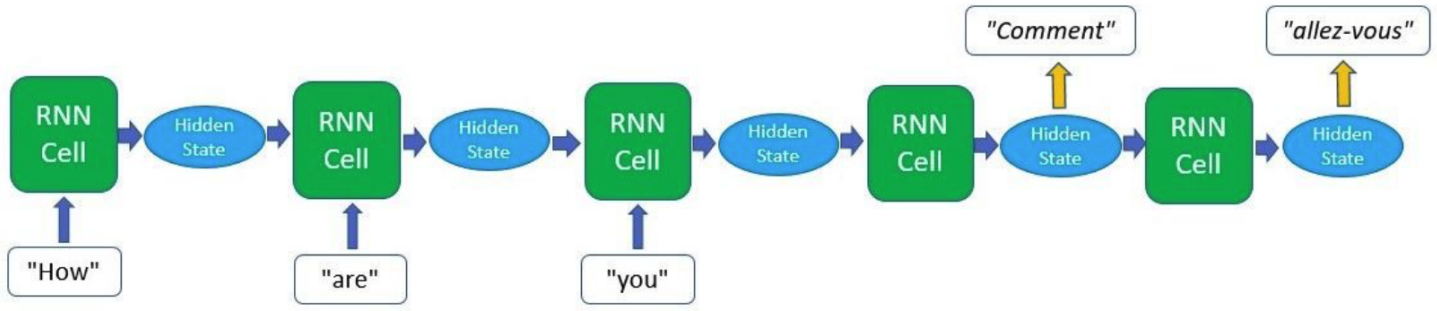


- x_t are the inputs of the network A, h_t are the outputs
- naive intuition: in a text, x_t could be the t-th word in the original language and h_t the t-th translated word

Recurrent Architectures

Recurrent Neural Networks (RNN)

- common implementation: sequence-to-sequence translation:
  1. encode original sentence
  2. decode to translated sentence



- side note:
  - usually in applications use embedding matrices to reduce dimensionality of the vocabulary
  - project the vocabulary to lower dimensional embedding space (vector space that still captures a lot of the meaning and semantic information, e.g. "king - man + woman = queen")

Recurrent Architectures

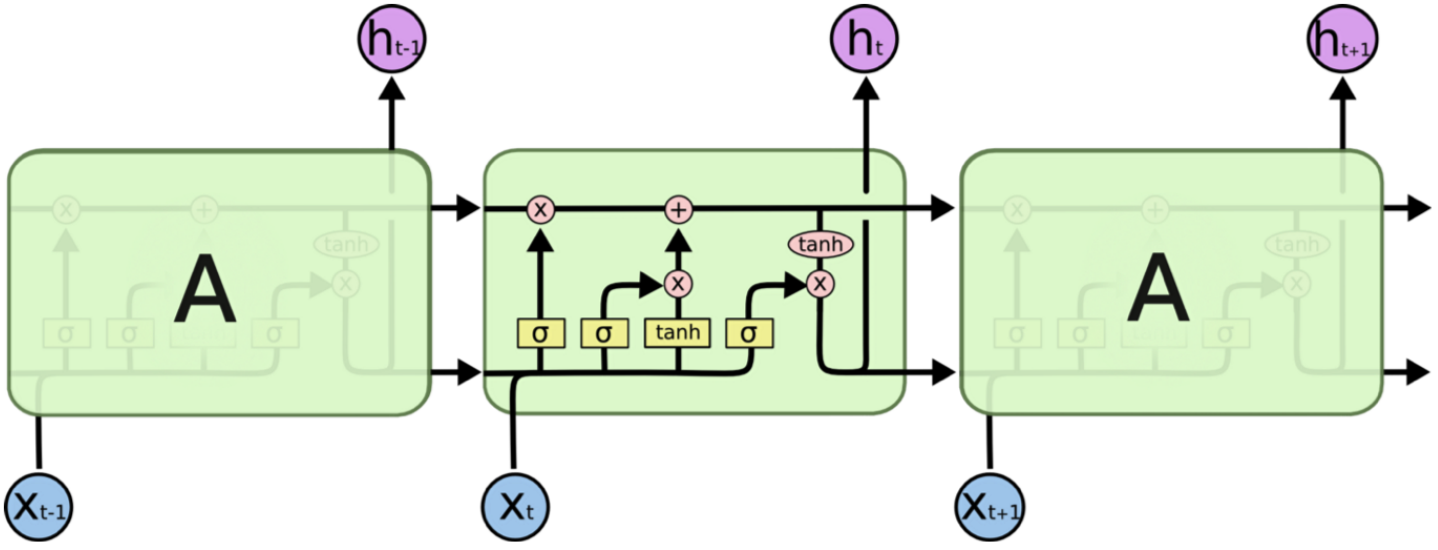
Recurrent Neural Networks (RNN)

Problems:

- RNN's are not very good at dealing with long-term dependencies (i.e. if the relevant information is far from the point where it is needed)
- all the 'memory' is passed in one hidden state at each step
- the longer the chain, the more likely it is the relevant information is lost along the chain
- improvement over regular RNN's: Long Short-Term Memory (LSTM)

Recurrent Architectures

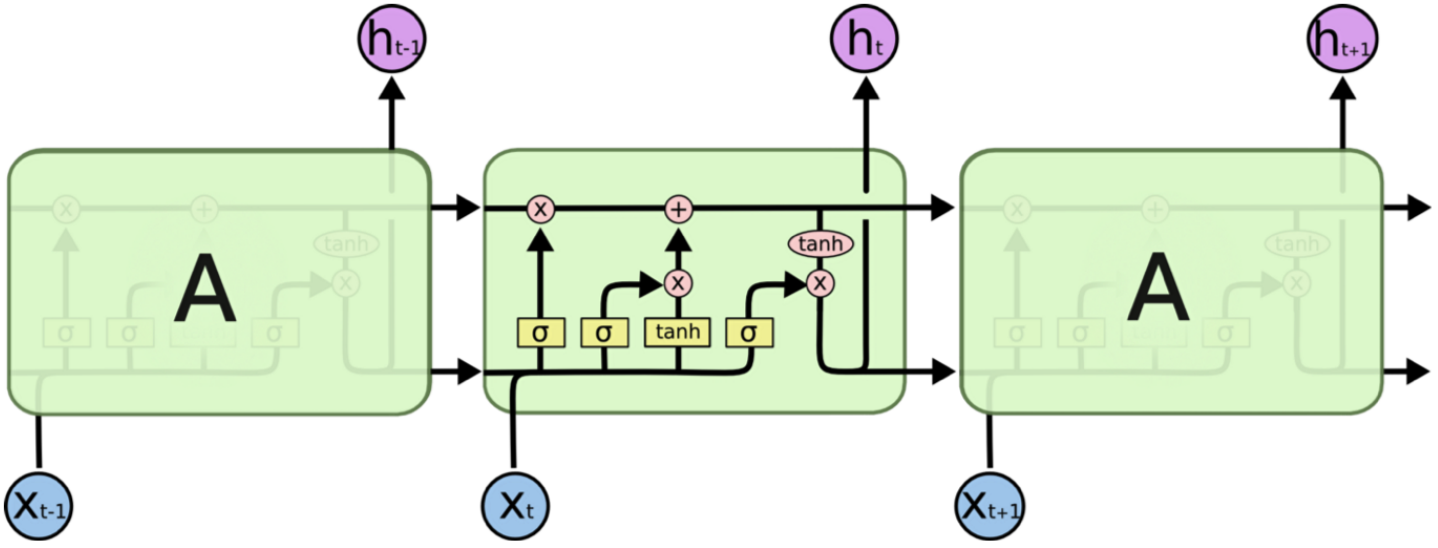
Long Short-Term Memory (LSTM)



- LSTM's have an additional hidden state, the cell-state
- series of operations between hidden state and cell-state
  $\rightarrow$ selectively remember important and forget unimportant information

Recurrent Architectures

Long Short-Term Memory (LSTM)



LSTM's still have some problems:

- if the sentences are too long, important information may still be lost
- sequential computation $\rightarrow$ no parallelization

Example Architectures

-

Attention and the Transformer

DESY

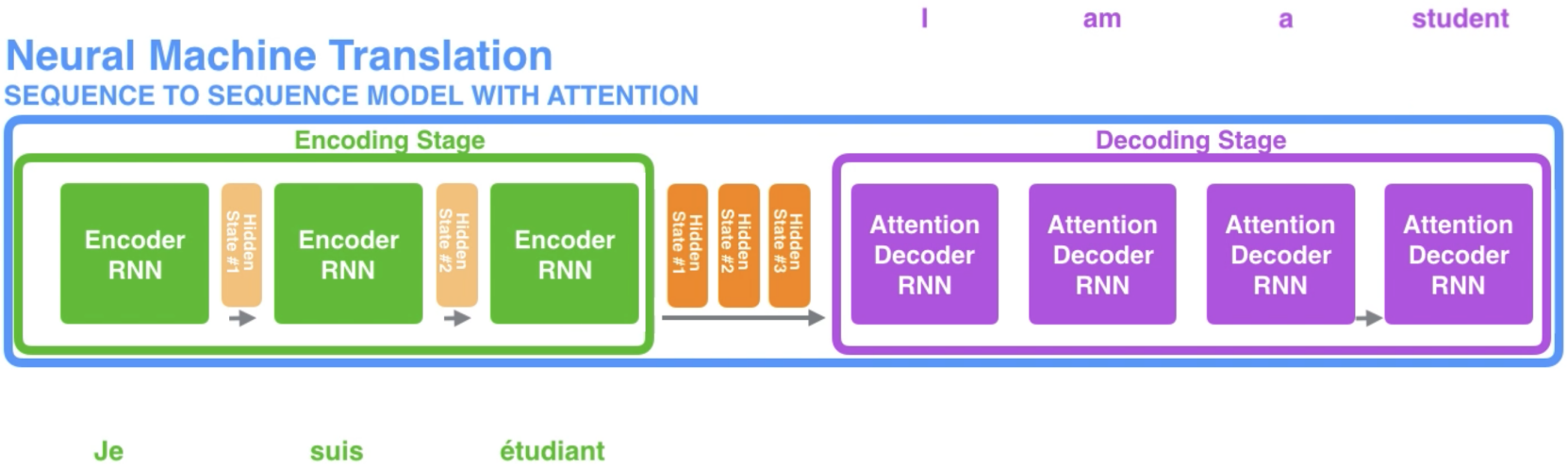
Attention

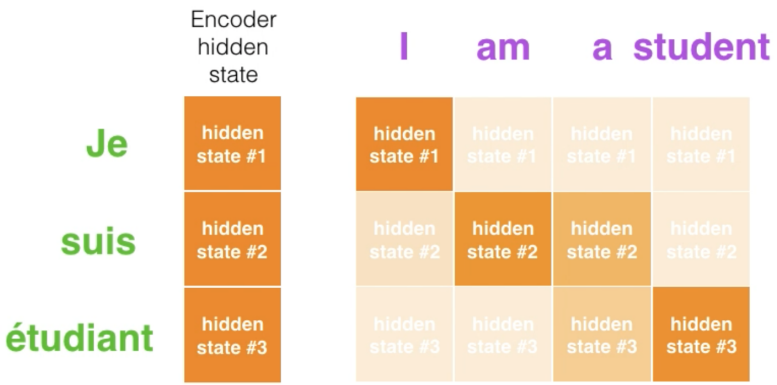
- the problem of lost long-term information can be addressed using *attention*
- idea: a word in a translated sentence could depend on any word in the original sentence
  $\rightarrow$ keep all of them as an input for each translated word but weight them by their importance:
  $x_i \rightarrow \sigma_i(Q, K)x_i$, where
  x_i: input words, $\sigma_i(Q, K)$: weights, K: key, Q: query
- e.g. for the translation of a verb (query), consider the gender of the subject and the verb in the original language (keys match) and ignore the other words (keys don't match)
- concept of attention is much more general than just for language translation

DESY

Attention

Example: attention in sequence-to-sequence models





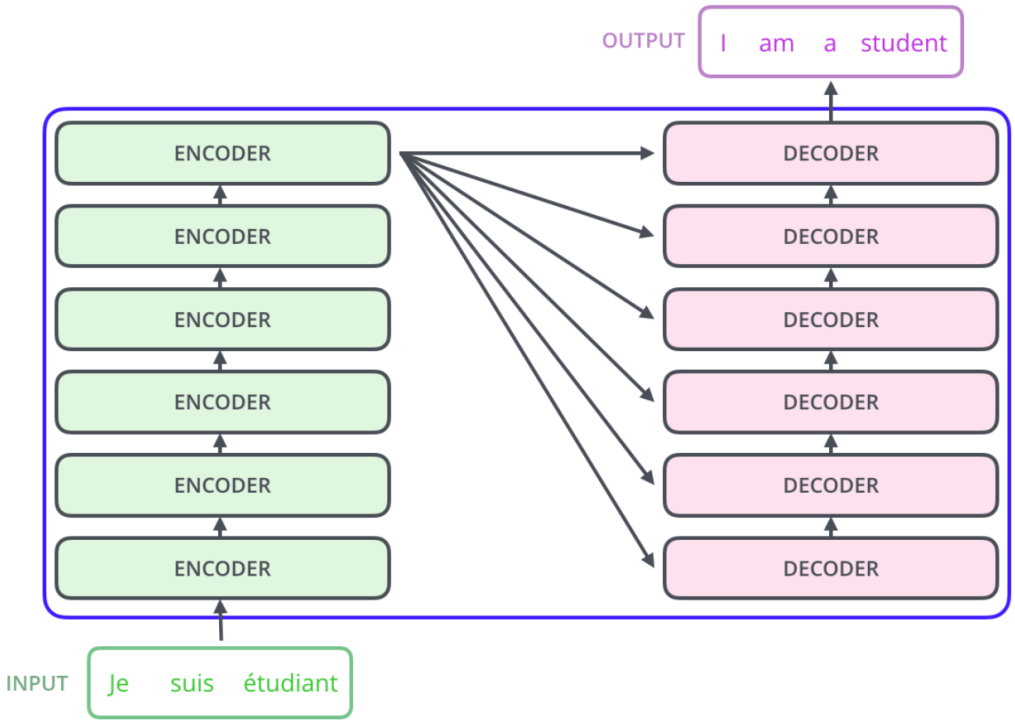
Self-Attention

- Self-Attention: keys K_i and queries Q_i are determined by the input x_i: $K_i = W_K x_i$, $Q_i = W_Q x_i$, $V_i = W_V x_i$,
  note: x_i are vectors in the embedding space, $W_{K,Q,V}$ matrices that map the x_i to an even lower dimensional space
- output: $z_i = \text{softmax}_j(Q_i \cdot K_j/\sqrt{d})V_j$, where
  $\text{softmax}_j(\alpha_{ij}) = \frac{e^{\alpha_{ij}}}{\sum_k e^{\alpha_{ik}}}$, d: $\dim(V_i) = \dim(K_i) = \dim(Q_i)$

DESY

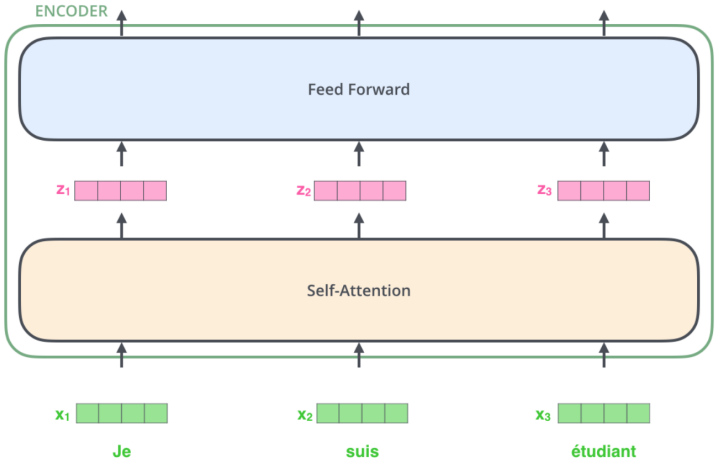
The Transformer

- consists of several encoders and decoders that each have multi-headed attention, i.e. they can pay attention to different kinds of information

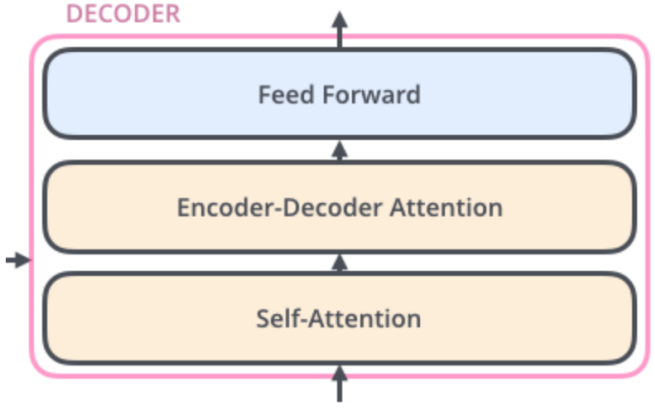


The Transformer

- encoder:



- decoder:



The Transformer

- attention mechanism solves the problem of long-term dependencies
- the feed forward NN's are independent of each other $\rightarrow$ can be parallelized
- this is the state-of-the-art technique that is used for sequence transduction

Attention

Thank you for your *Attention*

DESY