Complex tasks, basic blocks

The importance of primitives in Machine Learning

Istituto Nazionale di Fisica Nucleare

SCUOLA NORMALE SUPERIORE

Francesco Vaselli

tCSC Machine Learning 2025, Malmo

Machine learning 101

The aim is to construct algorithms that are able to learn a certain response given our training data

To achieve this, the learning algorithm is usually presented some training examples that demonstrate the intended relation

Then the learner is supposed to approximate the correct behaviour, *even for examples that have not been shown during training*

$$f\_true(x) \sim f^*(x, parameters)$$

All good, right?

Machine learning 101

The aim is to construct algorithms that are able to learn a certain response given our training data

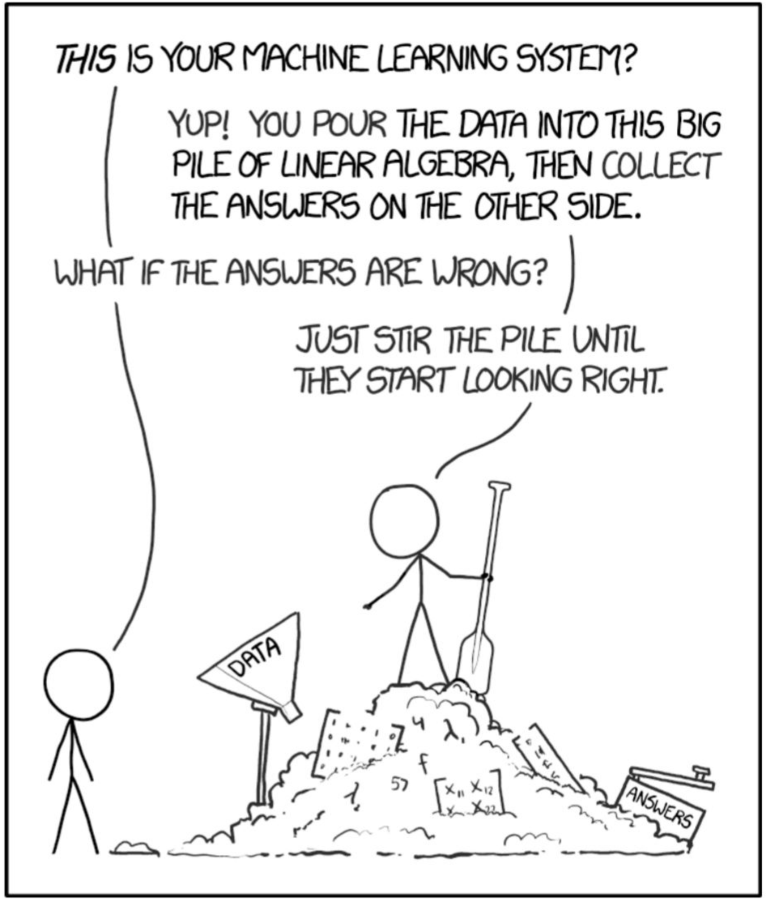
To achieve this, the learning algorithm is usually presented some training examples that demonstrate the intended relation

Then the learner is supposed to approximate the correct behaviour, *even for examples that have not been shown during training*

f_true(x) ~ f*(x, parameters)

All good, right?

Is it really that simple?


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

The need for Assumptions

Without any additional assumptions, this problem can be very hard and our solution inefficient!

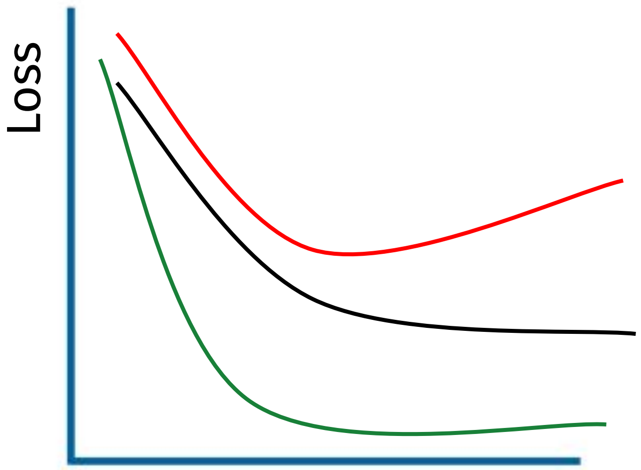
We often need to make some assumptions about the input data or the target function to approach it in a better way

For example, in statistics, if I know/assume that my data is distributed according to a quadratic law it’s much easier and efficient to fit directly a quadratic curve rather than an arbitrary n-th degree polynomial!

The need for Inductive Biases

Without any additional assumptions, this problem can be very hard and our solution inefficient! The kind of necessary assumptions about the nature of the target function are subsumed in the phrase “inductive bias”.

On the other hand, the right type of inductive bias might help the convergence and improve the results

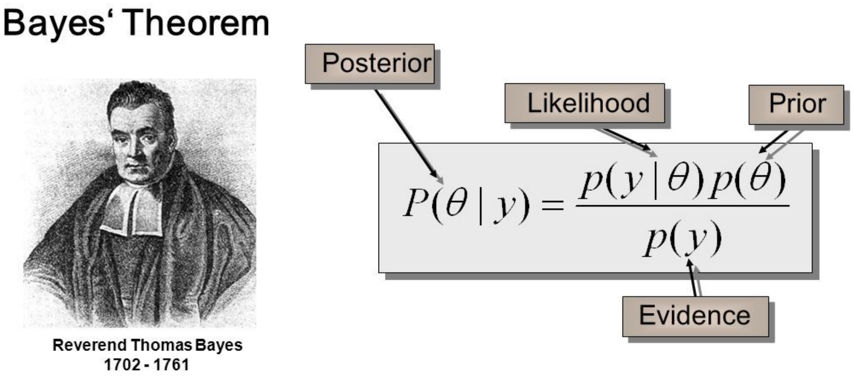


Average model
Bad bias
Good Bias

We can draw an analogy with Bayesian inference!

The Inductive Bias plays a similar role as the Prior p(θ) in Bayes theorem: we need to make some initial assumptions!

It tells which parameters are likely and which are not: good priors make for efficient inference, bad priors can make inference impossible!



"Bayes' Theorem describes, how an ideally rational person processes information."

Wikipedia

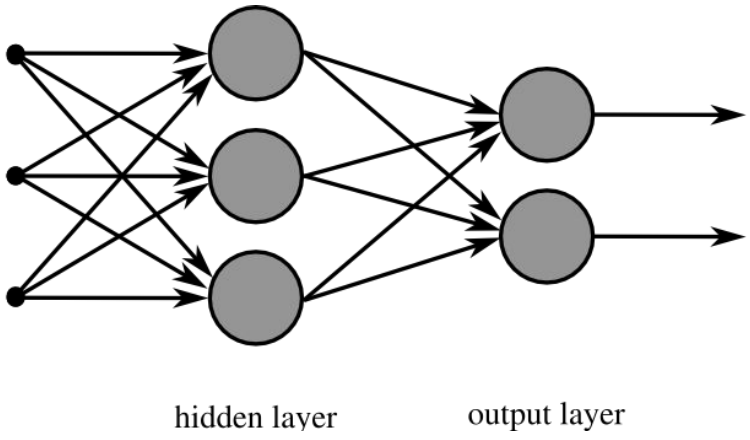
What we've mostly seen so far: feed forward NN

A stack of neurons, or MLP

Assumes all inputs are potentially related to all outputs in a dense way. In a sense, a very "relaxed" prior assumption

Good for tabular data, but…

What happens with high dimensional, structured data?



What we're going to see now

Encoding Inductive Biases for Structured Data

How to structure our networks in a way that takes the data and the domain we're working on into account: *primitives*

- Convolution
- Recursion
- Message Passing
- Attention

The focus is not on the applications, those will be illustrated better during the next lectures

What if we were to work with images?

In a Feed-forward NN we need to *flatten* the image

For a 18*16 pixels image that's already 288 input features

For a Whatsapp image ~800*800 -> 640k inputs!!

Which IB can help us in working with this input?

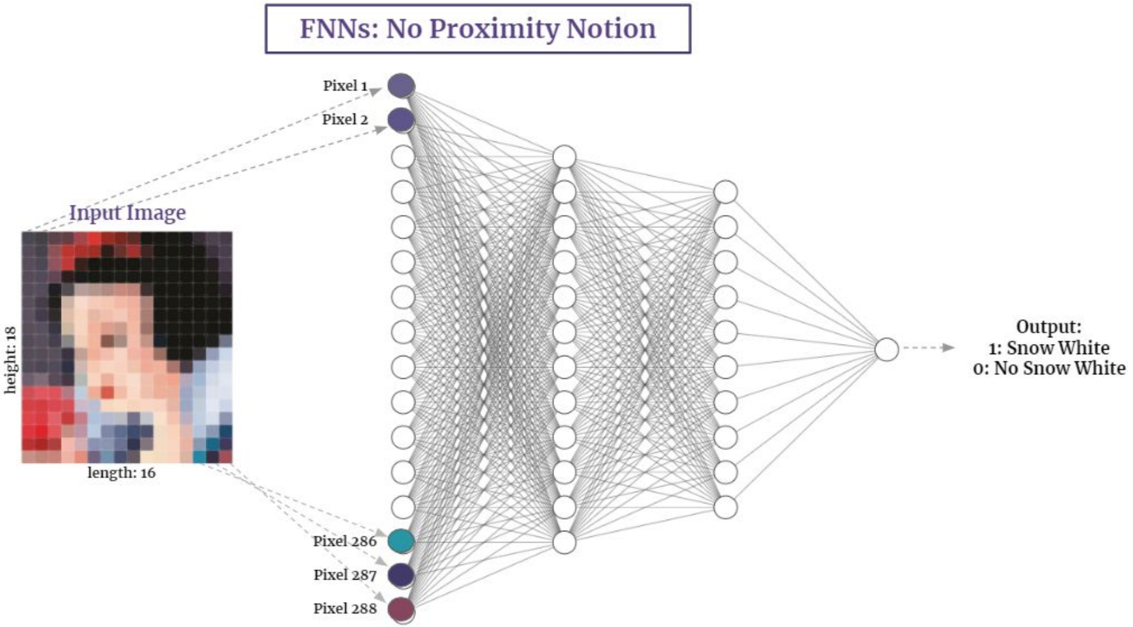


Image credit: https://www.kaggle.com/code/andradaolteanu/convolutional-neural-nets-cnns-explained

Data symmetries matter! Can we account for them in our Inductive Bias?

An eye is an eye, weather or not it’s top or bottom, right or left, small or big, etc.

How can we build a network that is invariant under the shift of input features?

So in this case:

IB = Locality + Translation invariance

We need to introduce a new *primitive*, a new building block of our NN

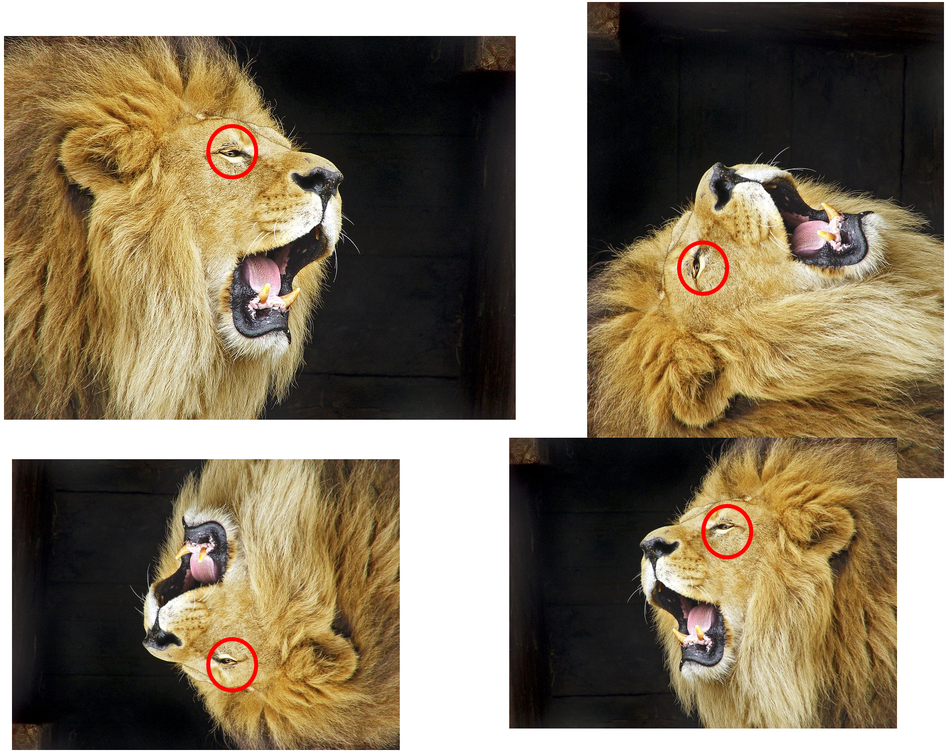
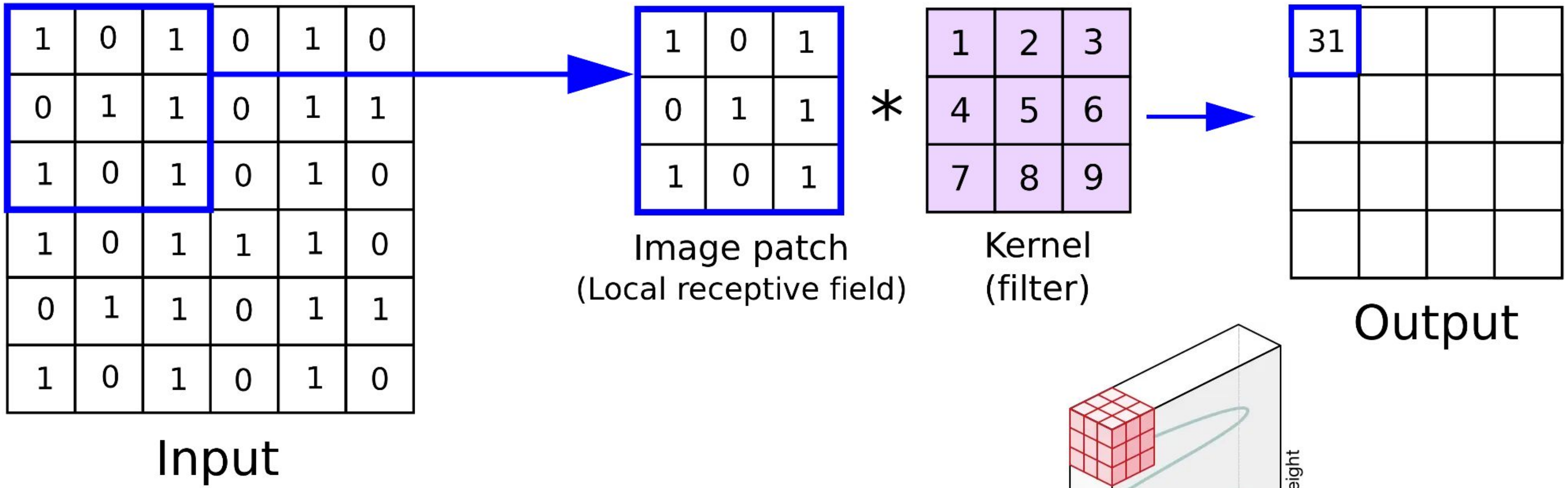


Image credit [Tambako The Jaguar](), Flickr

The filter (or *convolution*) is the new primitive!



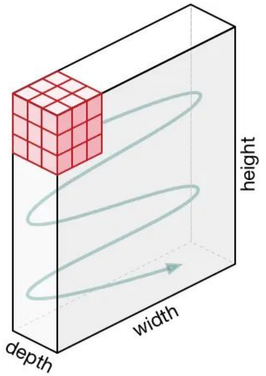


Image 1 credit
https://wikidocs.net/164365
Image 2 credit
https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

The filter is the new primitive!

Essentially

Filter == Learned matrix multiplication

By going over the image, each filter can focus on the local features

We will have filter specializing in recognizing the same element over multiple training images

A filter is still a set of parameters associated to a neuron, we just change how they go over the input data

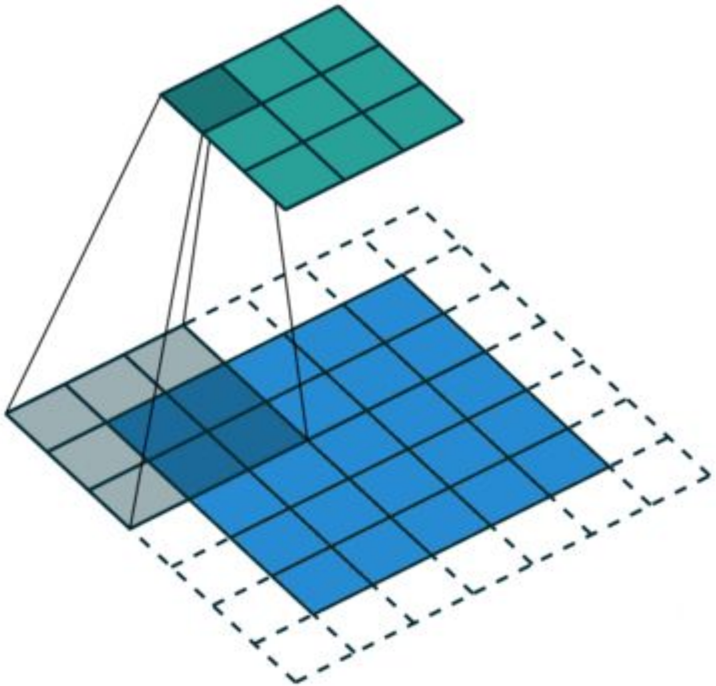


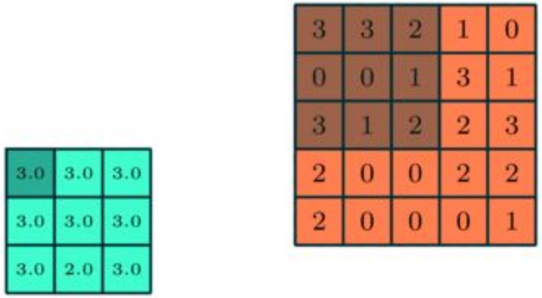
Image credit
https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

Pooling is needed to reduce complexity

After a few convolution steps we may want to aggregate the information

This helps in reducing the dimensionality of the feature grid, thus alleviating complexity

This is easily done with pooling, where we sum/average/etc. on a certain kernel size



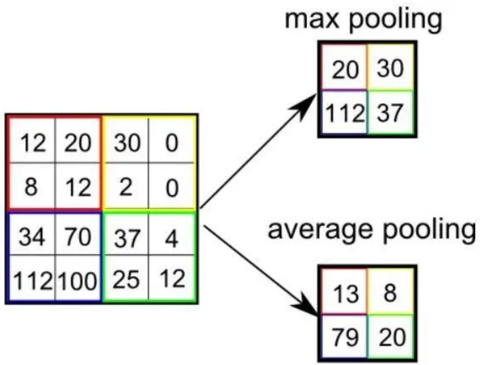
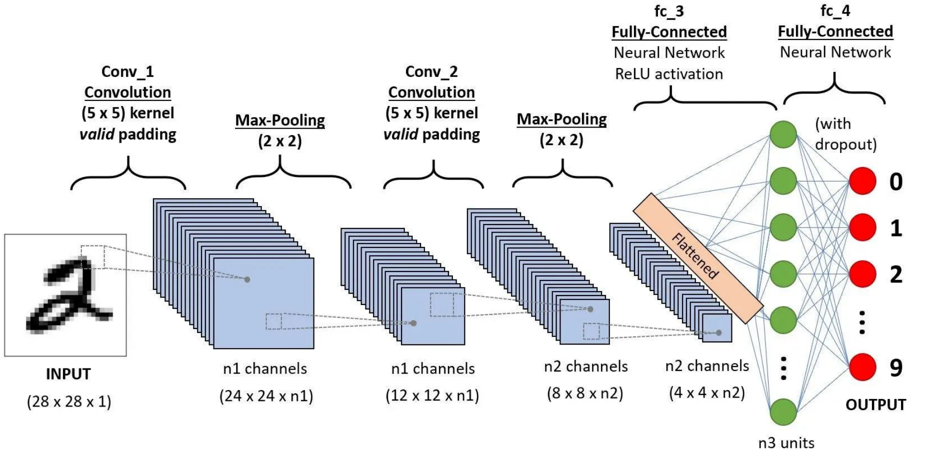


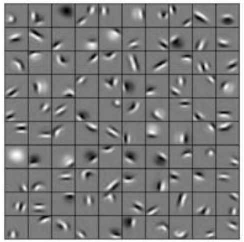
Image credit
https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

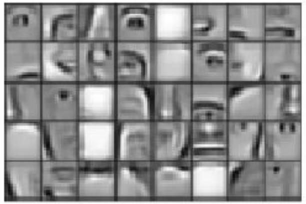
Filters in action

By going over the image, each filter can specialize on the extraction of single features and create an higher-level representation

The last part of the network is usually a feedforward NN which reasons on such a representation







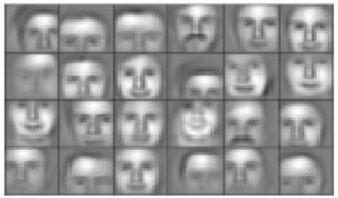


Image credit
https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/ and https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/

Improvements over FFNN

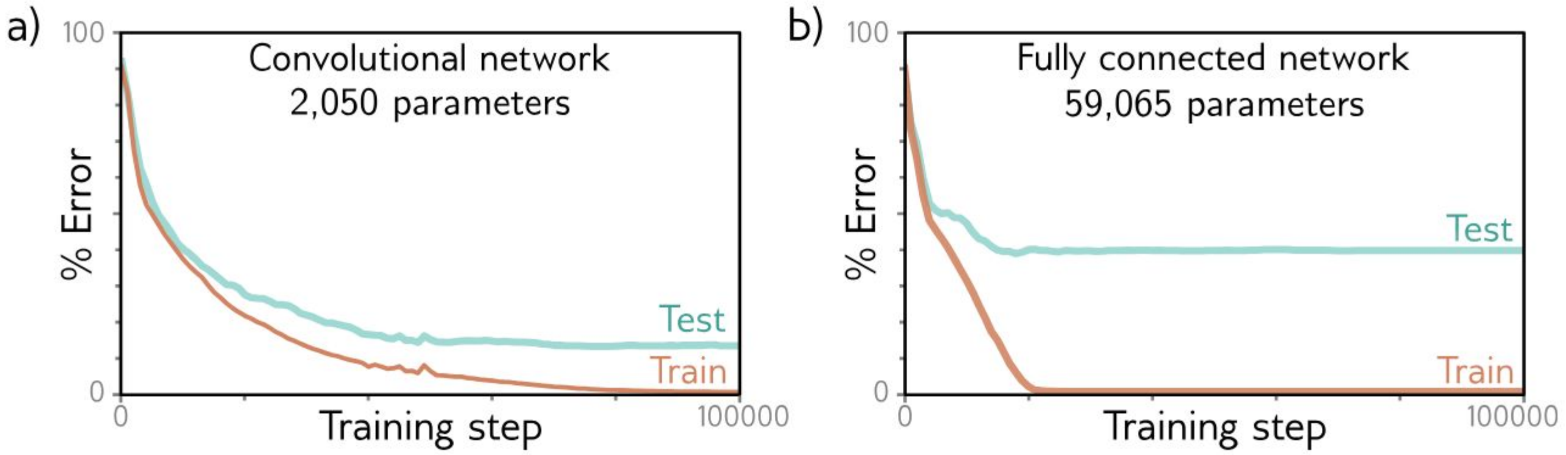


Figure 10.8 MNIST-1D results. a) The convolutional network from figure 10.7 eventually fits the training data perfectly and has ~17% test error. b) A fully connected network with the same number of hidden layers and the number of hidden units in each learns the training data faster but fails to generalize well with ~40% test error. The latter model can reproduce the convolutional model but fails to do so. The convolutional structure restricts the possible mappings to those that process every position similarly, and this restriction improves performance.

Image credit Understanding Deep Learning - Simon J.D. Prince

Data is not always tabular or in a grid!

Predicting the motion of an object knowing the current velocity and position

Predicting the next word in a sentence

Adjusting the state of an accelerator system given the previous state of a system

All domains with data coming as a stream from sensors: video, sound, *detectors*...

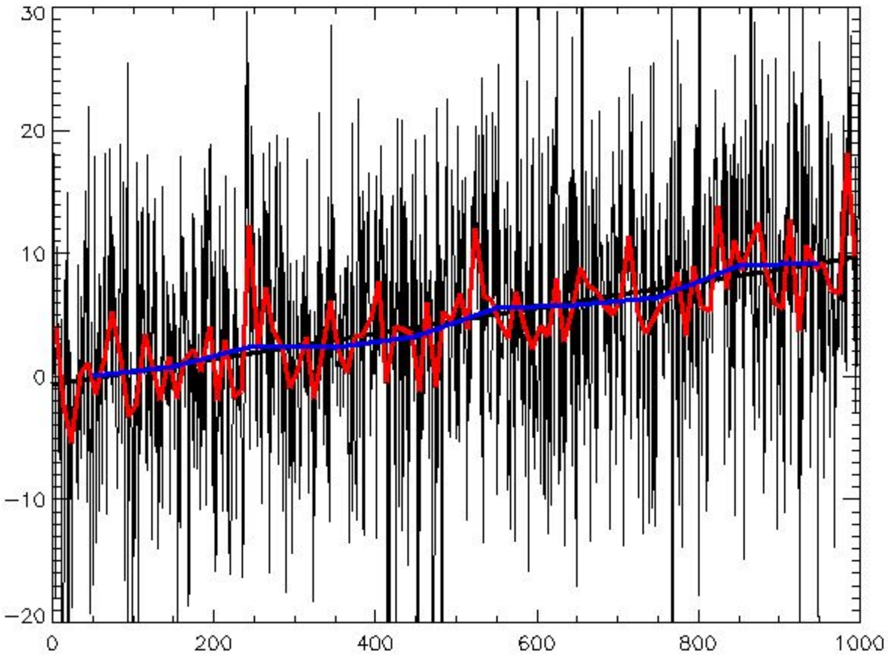


image credit
https://commons.wikimedia.org/wiki/File:Random-data-plus-trend-r2.png

We need to design our network according to this sequentiality

What if our data has an inherently sequential structure?

Inductive Bias == Sequentiality/Temporal Dependency

The order matters, past information influences present/future! We need to take this into account

We need to balance expressivity and computational constraints in our primitives

Ideally we would model $P(x_t \mid x_{t-1}, ..., x_{t-n+1})$ to capture all the correlations between the current state and the previous ones

In practice this can be very expensive and complicated to do. So the main intuition is to instead model:

$$P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid h_{t-1})$$
$$h_t = f(x_t, h_{t-1})$$

Primitive: Recurrence (hidden state carrying information through time)

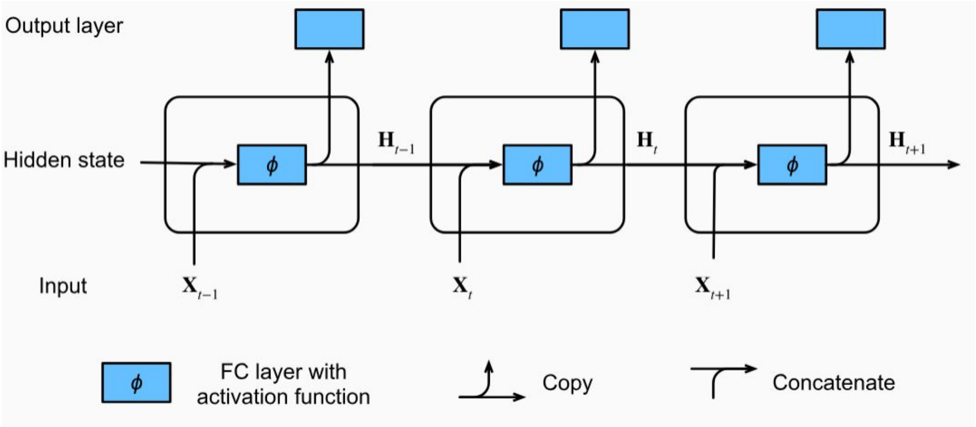
In practice: Recurrent Neural Networks

Images from https://d2l.ai/chapter_recurrent-neural-networks/rnn-scratch.html

Hidden state modeled at each timestep

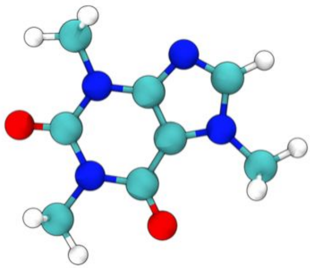
Propagated forward in time to aggregate the information from all steps

Need to handle carefully the backpropagation of gradients, can be exposed to vanishing/exploding ones



Not all our data is euclidean

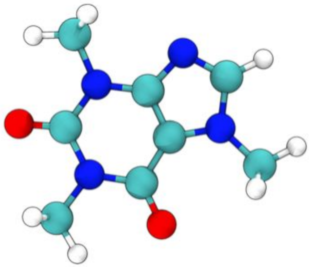
Problem: Data isn't always a grid or a sequence; it can be an arbitrary graph (social networks, molecules, particle showers...)

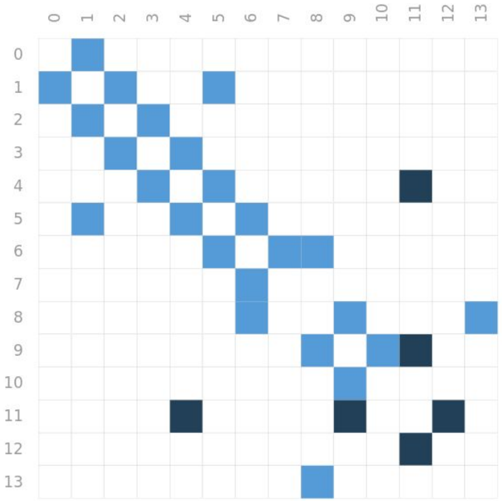


We can try to model it on a grid anyway...

We could actually model with an adjacency matrix, essentially a grid

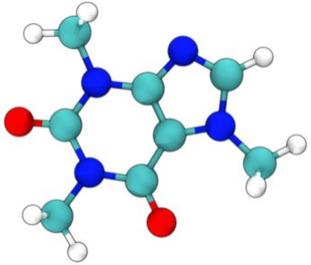
Problem: lot of sparsity, computationally inefficient, not permutation invariant

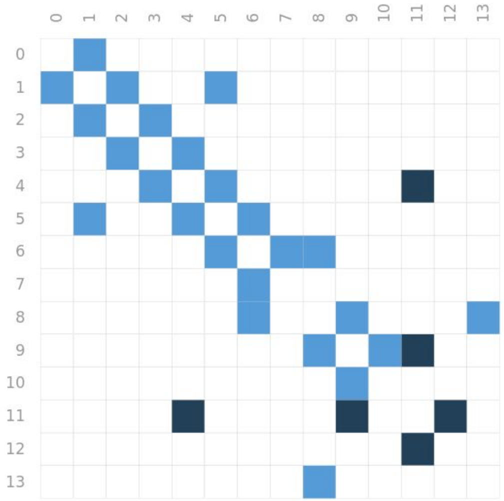


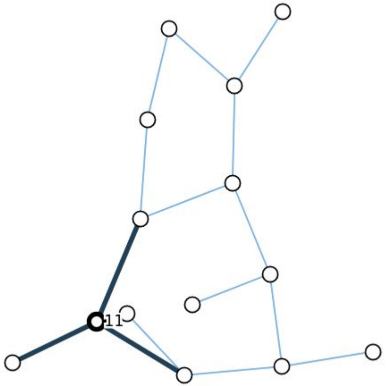


But the natural representation is through nodes and edges

We end up with the Graph representation as the most convenient and invariant representation







Images from:
https://distill.pub/2021/gnn-intro/

We want the network to encode this structure

Information propagates through connections, the specific ordering of a node's neighbors doesn't matter, only their aggregated information.

Inductive Bias == Relational Structure/Permutation Invariance of Neighbors

How can we enforce this bias? We need to have the different parts of the network communicate according to the graph structure

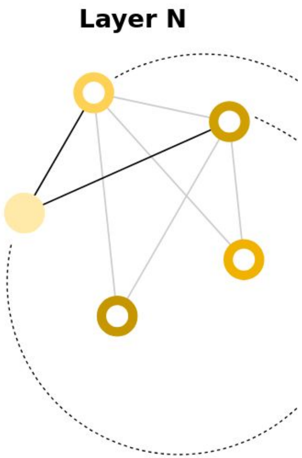
The primitive in this case is *message passing*

Message passing works in three steps:

For each node in the graph, gather all the neighboring node embeddings (or messages)

Aggregate all messages via an aggregate function (like sum).

All pooled messages are passed through an update function, usually a learned neural network



Images from:
https://distill.pub/2021/gnn-intro/

Have we handled all possible data structures?

Problem: RNNs struggle with long-range dependencies.
GNNs often aggregate locally.

How do we model relationships between any two elements in a set/sequence, especially when the "important" relationships aren't fixed by local structure?

Language is the obvious example:
“I’ve spent 3 months in France, where … and thus … Fortunately … And so I speak fluent French”

A trickier Inductive Bias is needed

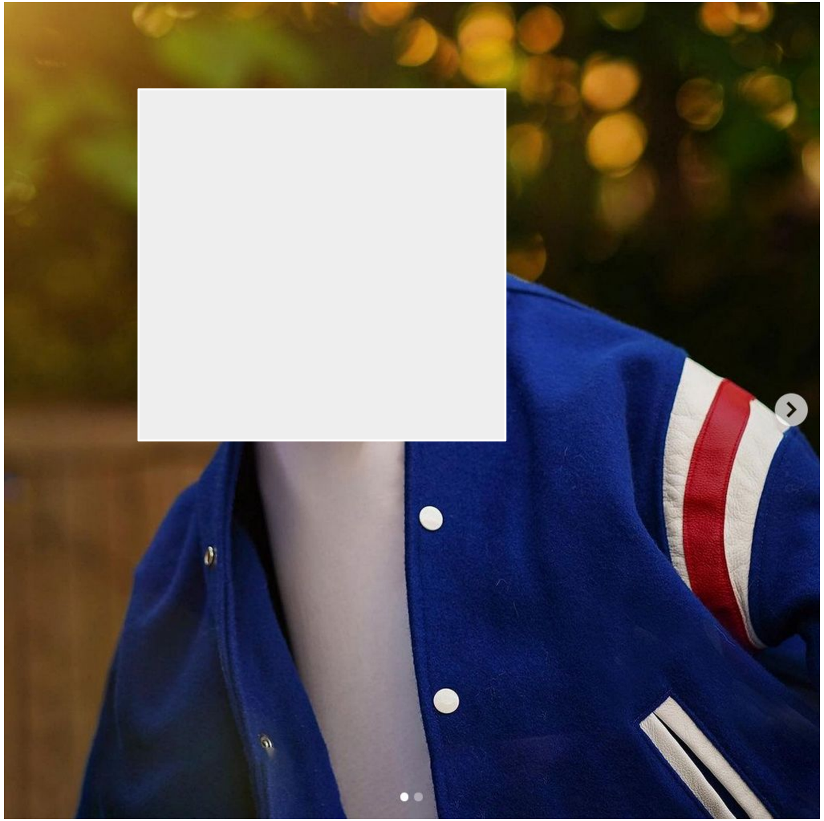
It doesn't assume fixed locality or sequentiality for determining importance, but rather learns which elements are relevant to which other elements for a given task.

Inductive Bias (or lack thereof initially, then learned) == Dynamic, Contextual Relevance.

What if we don’t know a-priori the best IB for our use case?

Can we structure the network to learn it on its own?

Is it a man or a woman?



Neither!

How was I able to fool you?

Attention is our new primitive!

Weighted sum based on learned relevance scores. This allows the model to "attend" to different parts of the input dynamically.

ML ATTENTION

==

SMART WEIGHTING of features

==

The current big thing in AI

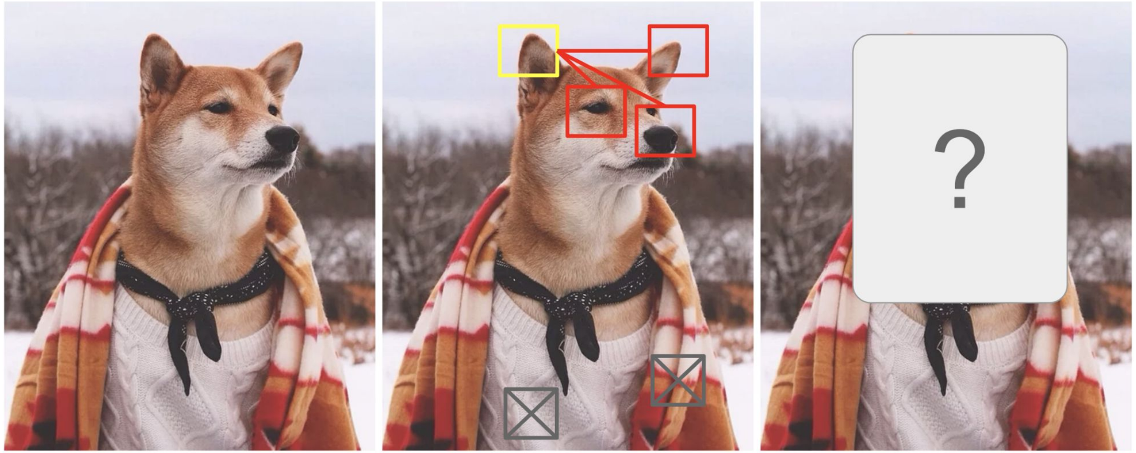


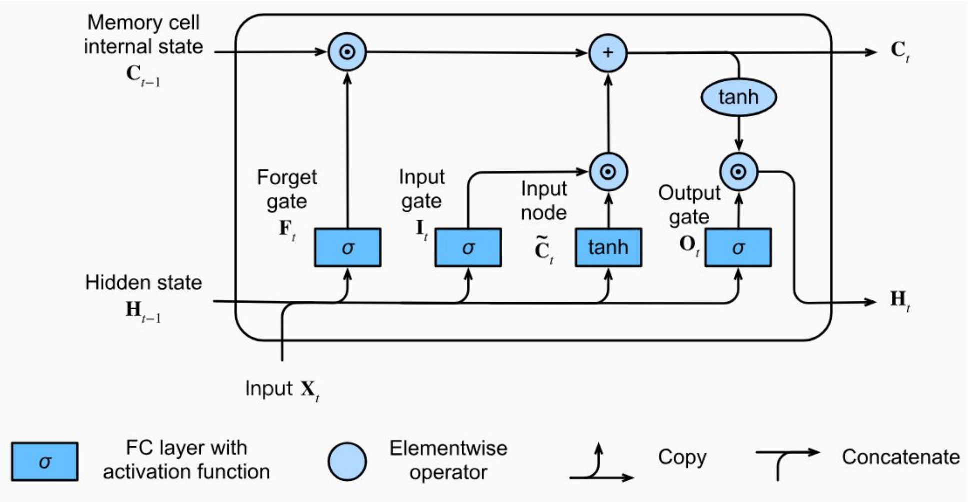
image source: Lilian Weng blog

Is it really something new?

A lot of research went into recursive model to address the problem of complex dependencies

With *Long-Short Term Memory* NN introduce a new set of internal gates to model forget functions and internal states

The real revolution was realizing that attention is enough by itself!



Assumptions behind attention

As its core, the attention mechanism assumes that the input can be divided into basic atomic units: *tokens*

tokenizer.tokenize("Don't you love 🤗 Transformers? We sure do.")

["_Don", "'", "t", "_you", "_love", "_", "🤗", "_", "Transform", "ers", "?", "_We", "_sure", "_do", "."]

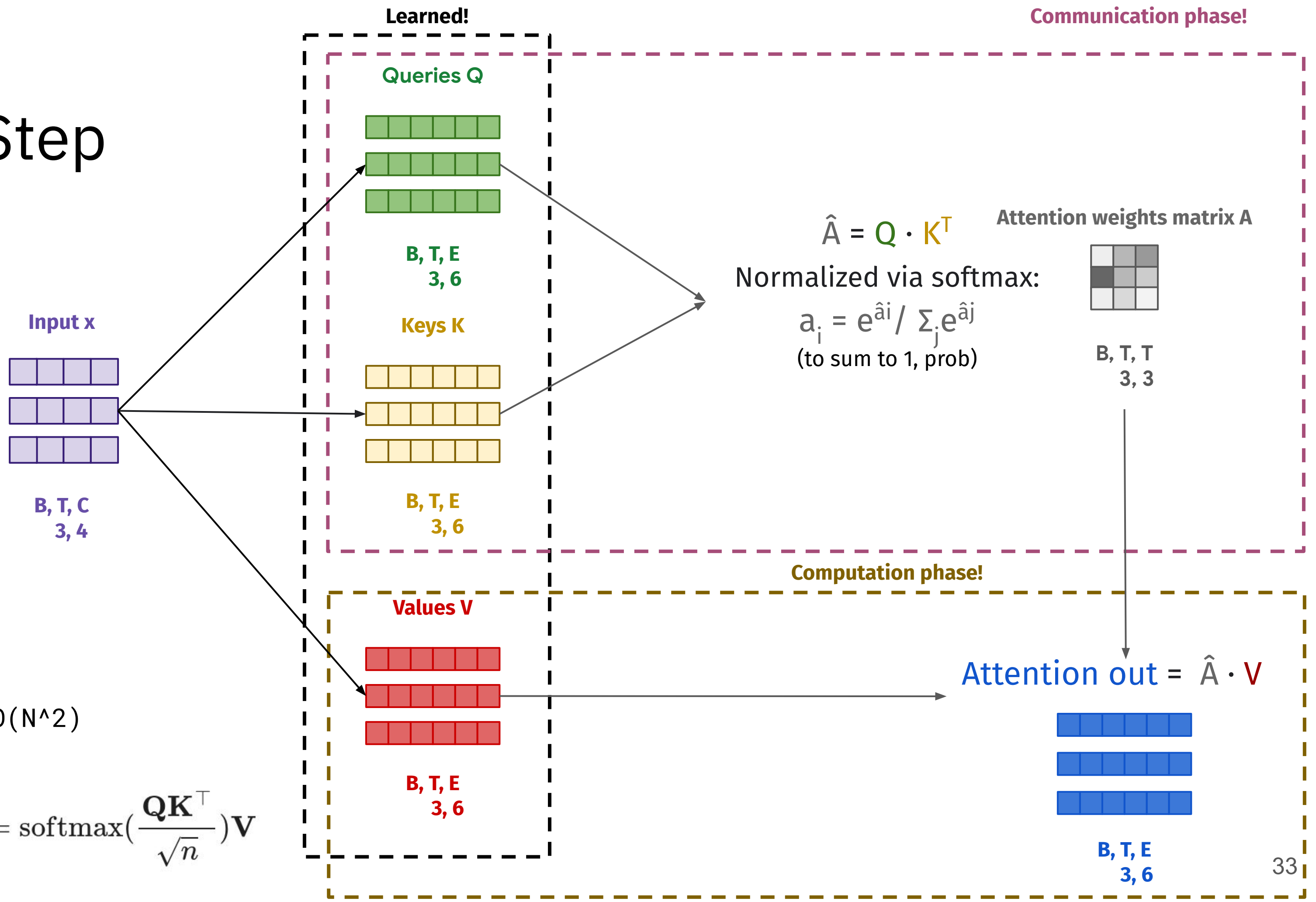
We end up with large O(100,000) vocabularies which pair each token to indexes and then to numerical values (*embedding*)

"t" ---> idx==346 ---> [18, 37, 2, ..., 129]

The size of this numerical representation is called *embedding size*

Attention Step-by-Step

"[The detective investigated]"



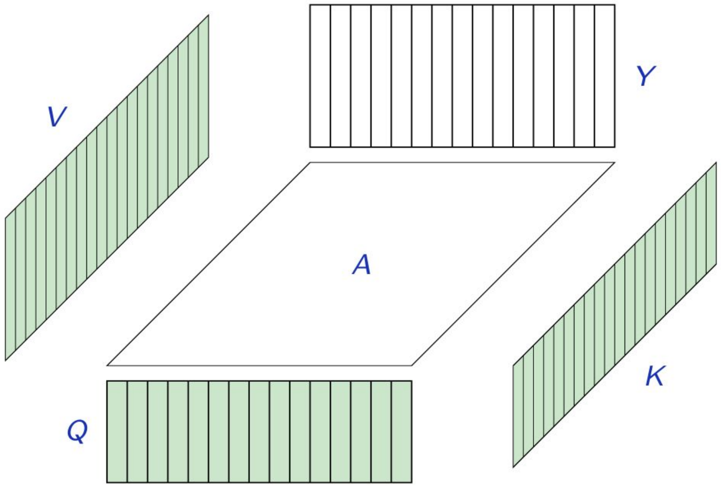
Complexity:

```
2*O(T^2 * C) ~ O(N^2)
```

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}}\right)\mathbf{V}$$

A nice visualization

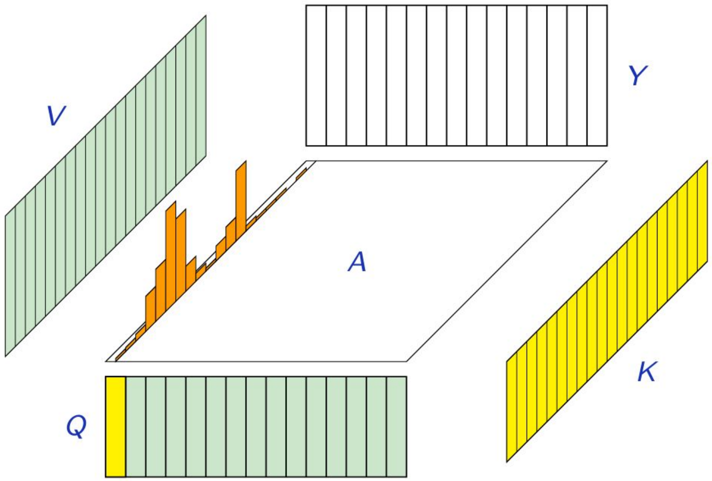
[Tensors are depicted here transposed for ease of representation.]



A nice visualization

[Tensors are depicted here transposed for ease of representation.]

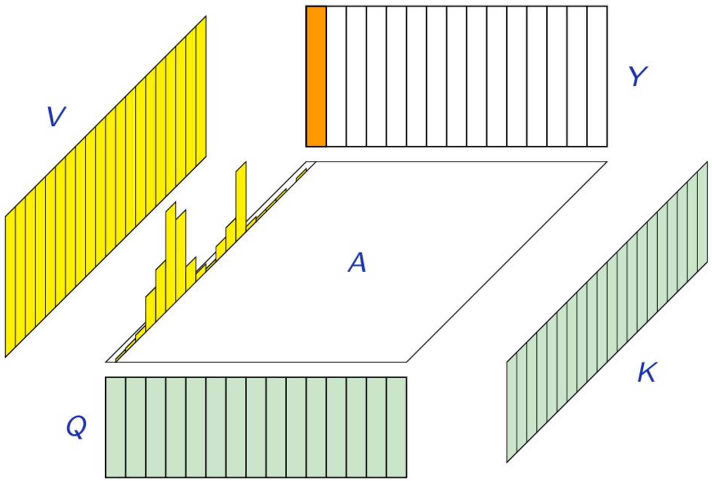
$$A_i = \text{softmax}\left(\frac{KQ_i}{\sqrt{D}}\right)$$



A nice visualization

[Tensors are depicted here transposed for ease of representation.]

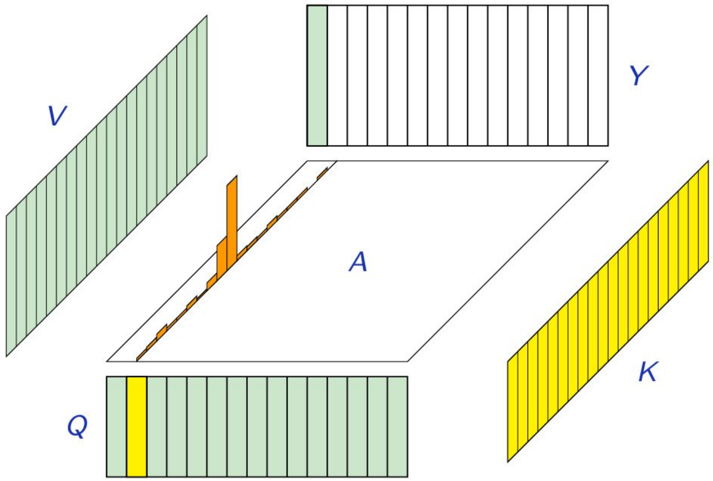
$$Y_i = V^\top A_i$$



A nice visualization

[Tensors are depicted here transposed for ease of representation.]

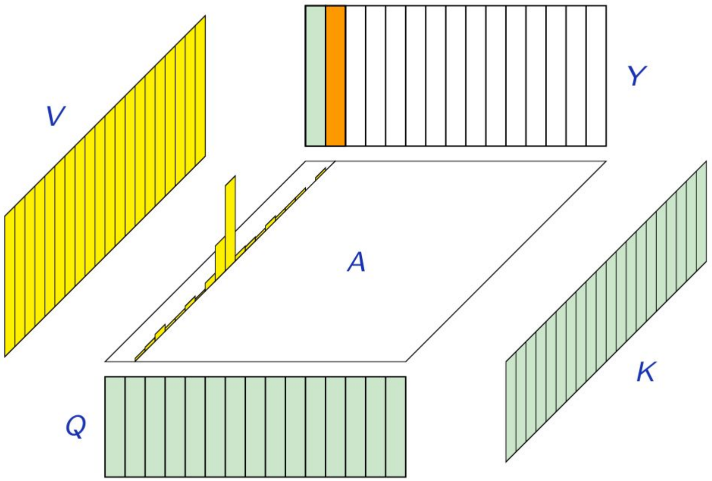
$$A_i = \text{softmax}\left(\frac{KQ_i}{\sqrt{D}}\right)$$



A nice visualization

[Tensors are depicted here transposed for ease of representation.]

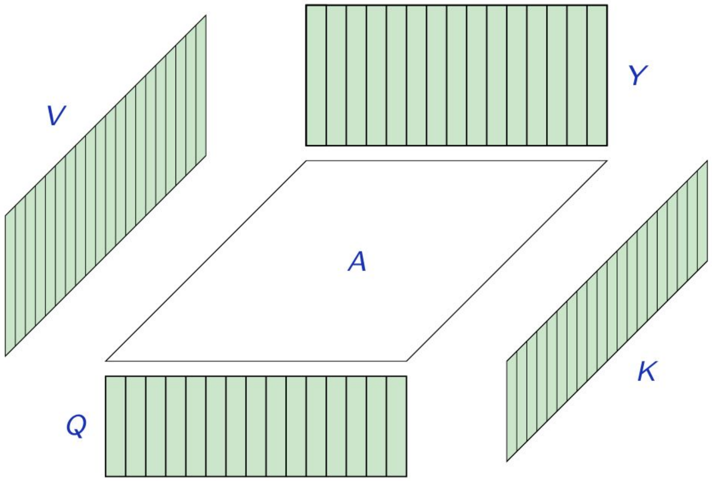
$$Y_i = V^\top A_i$$



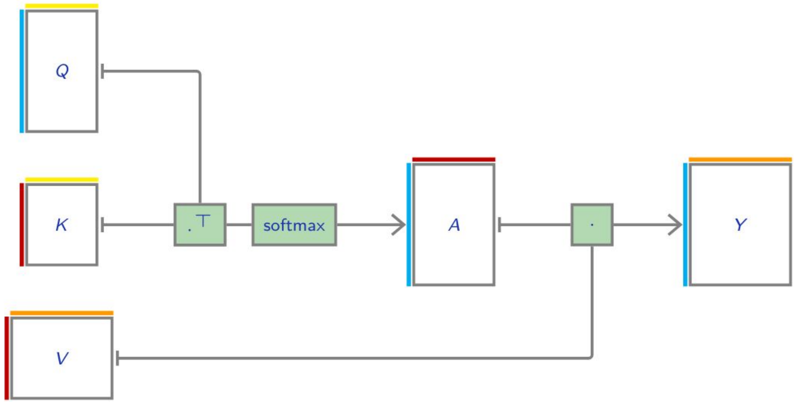
A nice visualization

[Tensors are depicted here transposed for ease of representation.]

$$A_i = \mathrm{softmax}\left(\frac{KQ_i}{\sqrt{D}}\right) \qquad Y_i = V^\top A_i$$



A nice visualization



$$A = \text{softmax}_{row}\left(\frac{QK^\top}{\sqrt{D}}\right)$$

$$Y = AV.$$

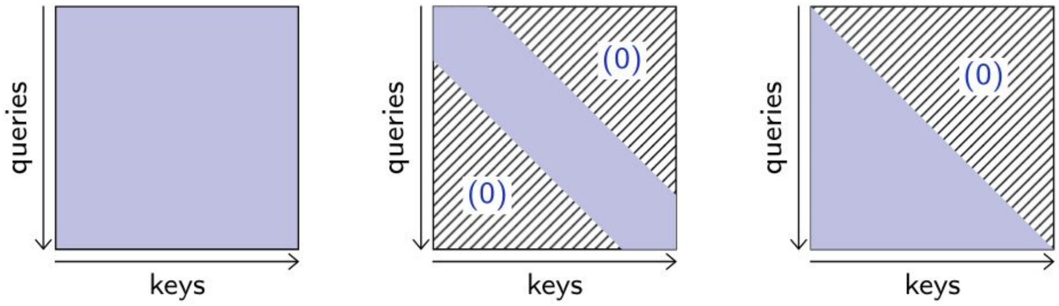
Standard attention

Attention: core take-aways

- **Attention is a communication mechanism**. Can be seen as nodes in a directed graph looking at each other and aggregating information with a weighted sum from all nodes that point to them, with data-dependent weights.
- When Q, K, V are built from the same inputs: *self-attention*, if K and V come from a different input than Q: *cross-attention*
- "Scaled" attention divides by *1/sqrt(head_size)*; so when input Q,K are unit variance the Softmax will stay diffuse and not saturate too much
- See also this notebook for a coding example

Masking is a powerful technique

It may be useful to mask the attention matrix, for instance in the case of self-attention, for computational reasons, or to make the model causal for auto-regression.



Full attention

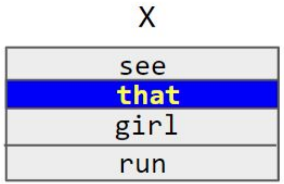
Local attention

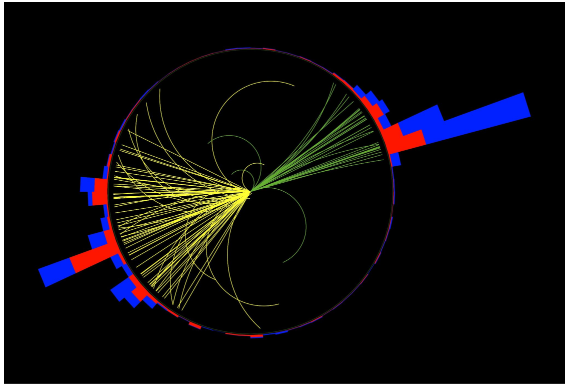
$|i-j| > \Delta \Rightarrow A_{i,j} = 0$

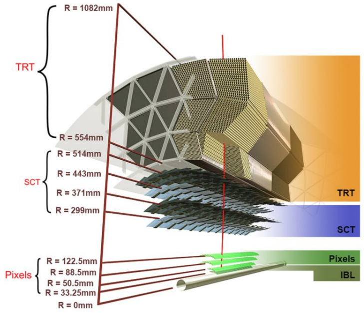
Causal attention

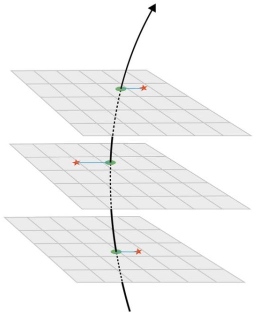
$j > i \Rightarrow A_{i,j} = 0$

Anything can be tokenized: text, waves, video, images… And more!





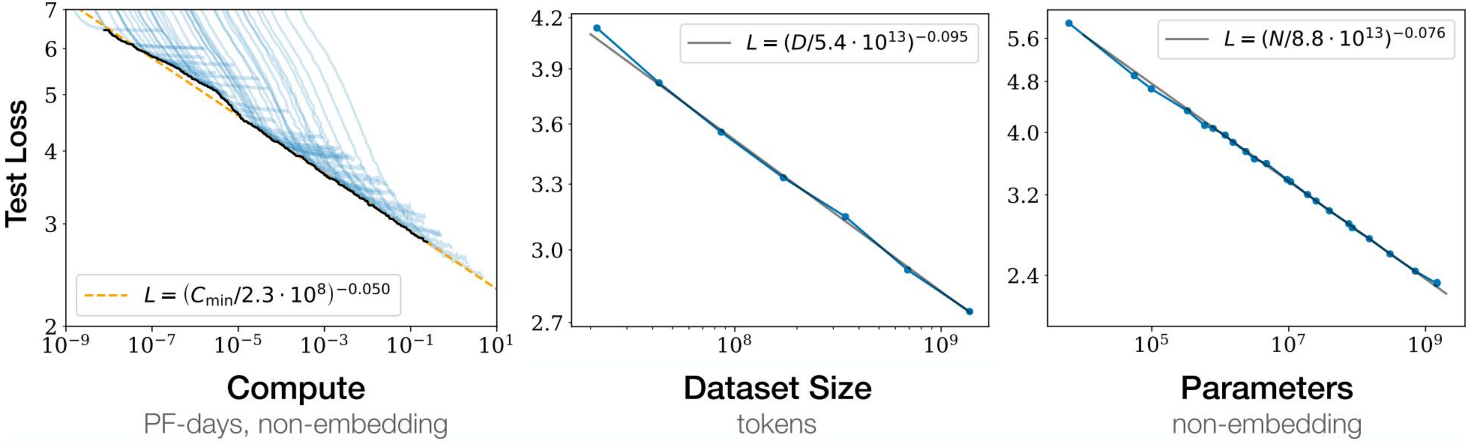




Images from Wikipedia, CMS, ATLAS

How well does it work?

Very, very well! Powering most of today's most advanced breakthroughs!
Predictable scaling laws which do not seem to be stopping: evidence of the *bitter lesson*!



Images from:
https://eliaszwang.com/paper-reviews/scaling-laws-neural-lm/

Attention as the most flexible prior possible

We return to our Bayesian analogy

It allows for finding the optimal parameter configuration, no dependency on sequence lengths or tokens shapes

Under a non-constraining prior we need a lot of data to build a constraining likelihood and achieve good performance

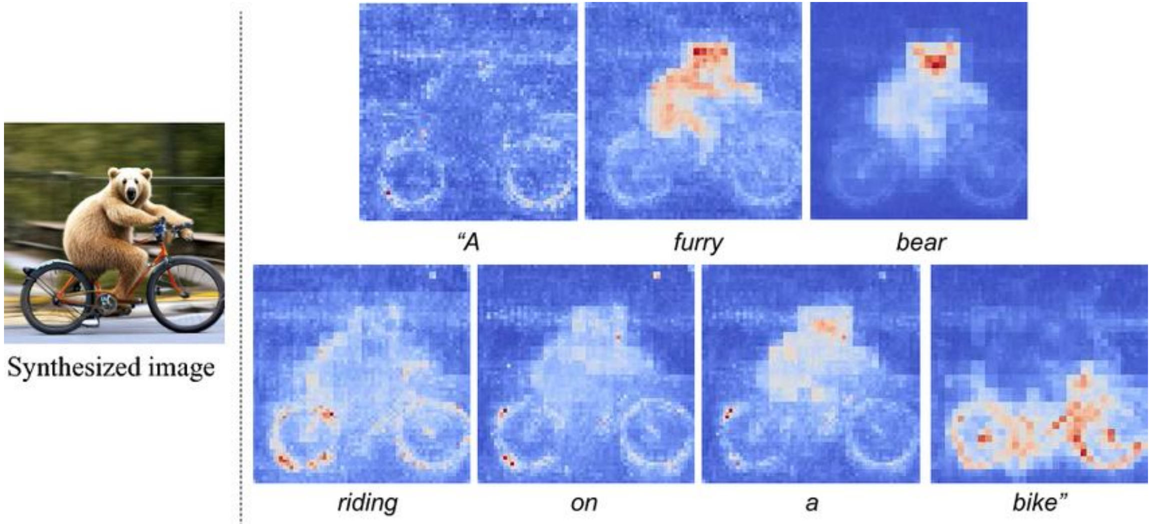
Need scale, large amount of data! No free-lunch theorem, still outperformed by appropriate IBs in low data regimes

Attention map analysis

	I	love	you
je	0.94	0.02	0.04
t'	0.11	0.01	0.88
aime	0.03	0.95	0.02

Evaluating the values of the attention scores can be used to understand which part of the input is informing which part of the output

Perfect explainability is not always possible due to the scale of these models



Visualized cross-attention maps

image credit “Masked-attention diffusion guidance for spatially controlling text-to-image generation”, Wikimedia

Attention is STILL an inductive bias

It assumes a token structure, and tokenization plays a huge role in the performance

It assumes pairwise interactions between tokens

Natively, ordering does not matter. We need to encode the position manually through specific techniques

GNNs already aggregate information from neighbors. Attention can be seen as a more flexible way to aggregate information from a set of elements, where the "neighborhood" or "relevance" is learned dynamically.

Conclusions

The right choice of primitives makes all the difference

Most primitives are motivated by data symmetries evident in our datasets

Attention emerged as the most flexible primitive/prior and has quickly replaced most other approaches

Not necessarily true at your complexity/scale! Need to consider your use case well: in the next lectures you will see how to build on these basic blocks for many successful applications in science

References

CNN: https://d2l.ai/chapter_convolutional-neural-networks/index.html

RNN: https://d2l.ai/chapter_recurrent-neural-networks/index.html

GNN: https://distill.pub/2021/gnn-intro/

Attention:

- https://lilianweng.github.io/posts/2018-06-24-attention/
- https://www.youtube.com/watch?v=kCc8FmEb1nY
- https://d2l.ai/chapter_attention-mechanisms-and-transformers/index.html
- https://www.youtube.com/watch?v=bMXqnLiVgLk

Backup

We have universal representation theorems!! Why are MLPs not enough?

Universal approximation theorems are existence theorems: They simply state that there *exists* such a sequence $\phi_1, \phi_2, \dots \to f$, and do not provide any way to actually find such a sequence. They also do not guarantee any method, such as backpropagation, might actually find such a sequence. Any method for searching the space of neural networks, including backpropagation, might find a converging sequence, or not (i.e. the backpropagation might get stuck in a local optimum).

Also, we need to make assumption on the function space of f, both theoretically and in practice (which inputs/outputs)