Politecnico
di Torino

Deep Learning from Zero to Hero

Marco Ciccone

Politecnico di Torino

https://marcociccone.github.io/

name.surname AT polito.it

My background

- **BSc** Computer Engineering - University of Florence
- **MSc** Computer Science and Engineering - Politecnico di Milano
  - Mainly interested in Audio Signal Processing
  - End up doing thesis on Computer Vision to work on Deep Learning
- **PhD** Computer Science and Engineering - Politecnico di Milano
  - Research on Deep Learning for Computer Vision applications
    - Learning with different modalities, with few labels
  - A couple of research internships in industry (NNAISENSE, NVIDIA)
- **Postdoc** Researcher - Politecnico di Torino & UCL
  - Continual Learning
  - Federated Learning

Education

↓

Research

Disclaimers

- I will **briefly** discuss models and techniques that brought us to this exciting time in AI
- This is an **introductory lecture** to give you just an overview. Not meant to be exhaustive at all!
- If you are interested in knowing more just **ask for more** material
- I will occasionally make some examples of **applications**
- The field moves at an incredible pace. This revolution happened in only a decade!

Outline plan

- Recap on CNNs, architecture designs and training tricks
- Sequential Modeling (RNNs)
- Attention
- Transformer
- Final remarks

Machine Learning

📝

“Learning from experience how to perform a given task

that has to be automatized by a machine.”

Basics of Supervised Learning

$\mathcal{X}$ Input set $\mathcal{Y}$ Output set

ρ (unknown) probability on $\mathcal{X} \times \mathcal{Y}$

$\ell : \mathcal{Y} : \mathcal{Y} \mapsto \mathbb{R}$ Loss function

$f : \mathcal{X} \mapsto \mathcal{Y}$ input-output predictor (candidate **model**)

Classification $\mathcal{Y} = \{1, \ldots, K\}$

Regression $\mathcal{Y} \subseteq \mathbb{R}$

Goal: minimise Expected Risk

$$\mathcal{R}(f, \rho) = \mathbb{E}_\rho \, \ell(f(x), y)$$

Basics of Supervised Learning

In practice, ρ is **unknown** and it can be only accessed via finite samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

Learning Algorithm

$$\mathcal{A} : \mathcal{D} \mapsto f$$

λ Hyper-parameters

$\mathcal{H}$ Hypothesis/Inductive Bias

$$\mathcal{A}(\lambda; \mathcal{D}) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \, \mathcal{R}(f, \mathcal{D}) + \Omega_\lambda(f)$$

Empirical Risk Minimization (ERM) + Regulariser

Pretty abstract and general framework

Basics of Supervised Learning

In practice, ρ is **unknown** and it can be only accessed via finite samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

Learning Algorithm

$$\mathcal{A} : \mathcal{D} \mapsto f$$

λ Hyper-parameters

$$\mathcal{A}(\lambda; \mathcal{D}) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \quad \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \Omega_\lambda(f)$$

$\mathcal{H}$ Hypothesis / Inductive Bias / Model

Basics of Supervised Learning

In practice, ρ is **unknown** and it can be only accessed via finite samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

BackProp + **SGD**

$$\mathcal{A} : \mathcal{D} \mapsto f$$

λ Hyper-parameters

$$\mathcal{A}(\lambda; \mathcal{D}) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \quad \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) + \Omega_\lambda(f)$$

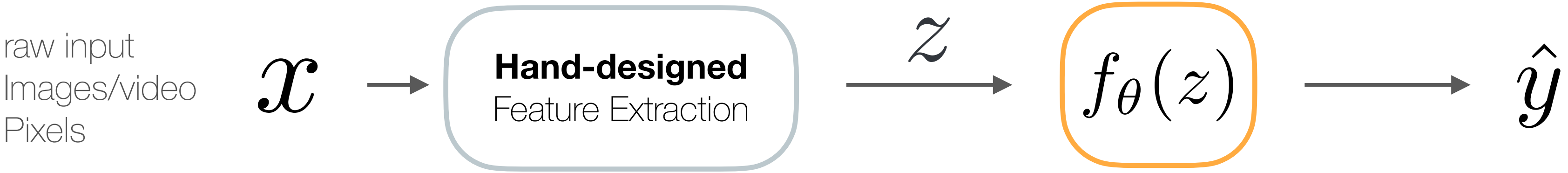
$\mathcal{H}$ **Deep Neural Networks**

Before: classification pipeline

— Learned
— Fixed

Input set $\mathcal{X} \subseteq \mathbb{R}^{h \times w \times c}$

Output set $\mathcal{Y} = \{1, \ldots, K\}$



raw input
Images/video
Pixels

$x \rightarrow$ **Hand-designed** Feature Extraction $\xrightarrow{z}$ $f_\theta(z)$ $\longrightarrow \hat{y}$

- Features are **not** learned (e.g. HOG, SIFT, bag of features)
- Designed to have nice properties from domain experts

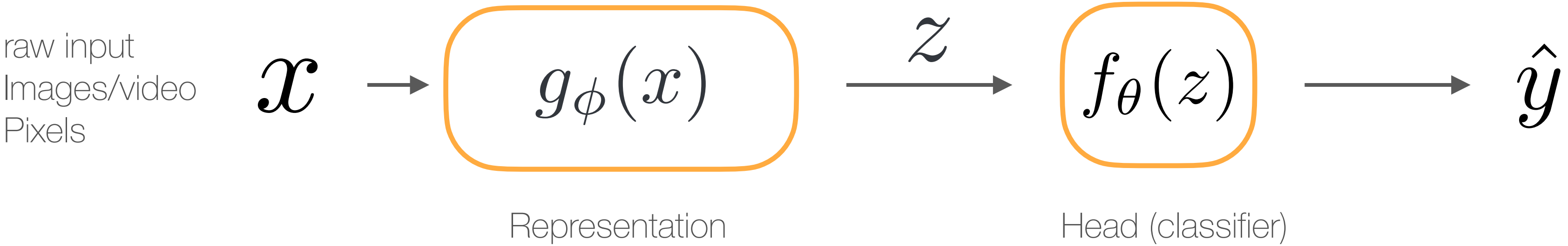
After: classification pipeline

Input set $\mathcal{X} \subseteq \mathbb{R}^{h \times w \times c}$

Output set $\mathcal{Y} = \{1, \ldots, K\}$

— Learned

— Fixed



raw input
Images/video
Pixels

$x \rightarrow \boxed{g_\phi(x)} \xrightarrow{z} \boxed{f_\theta(z)} \longrightarrow \hat{y}$

Representation

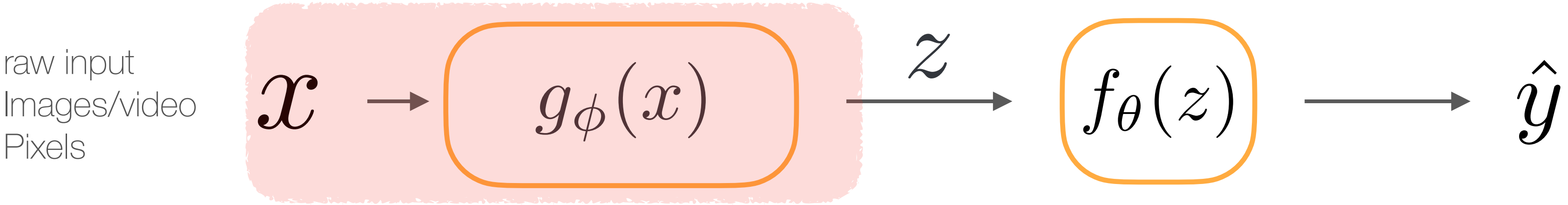
Head (classifier)

After: classification pipeline

Input set $\mathcal{X} \subseteq \mathbb{R}^{h \times w \times c}$

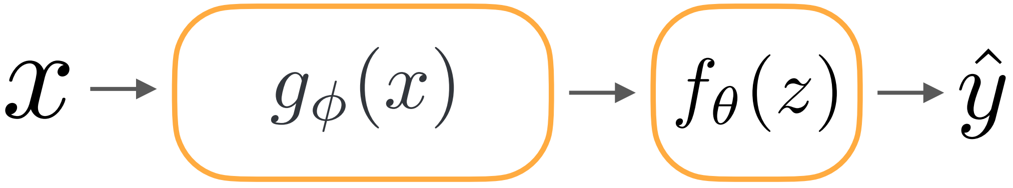
Output set $\mathcal{Y} = \{1, \ldots, K\}$

🟧 Learned
⬜ Fixed



Representations learning via **Deep Neural Networks**
uses back-propagation and (stochastic) gradient descent

Prediction as Translation

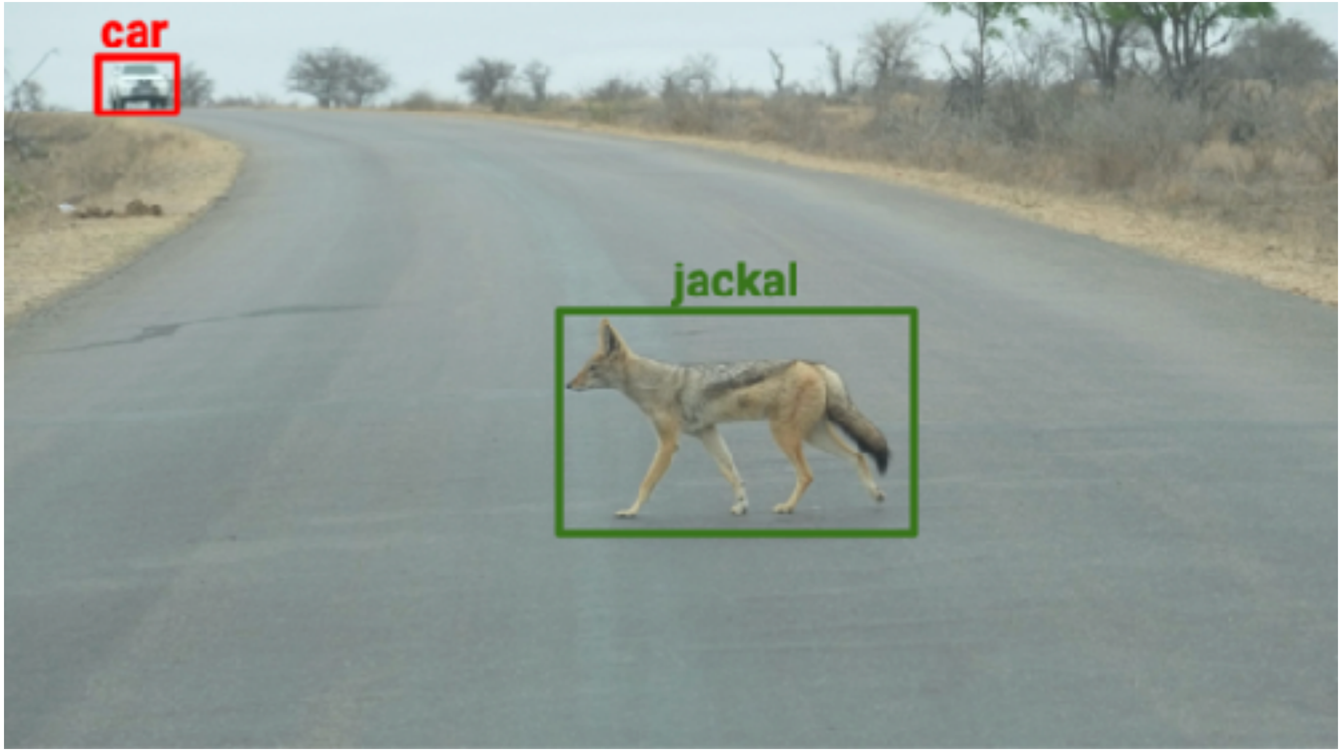


Translate a raw signal to another space which has a **semantic**

(End-to-end)

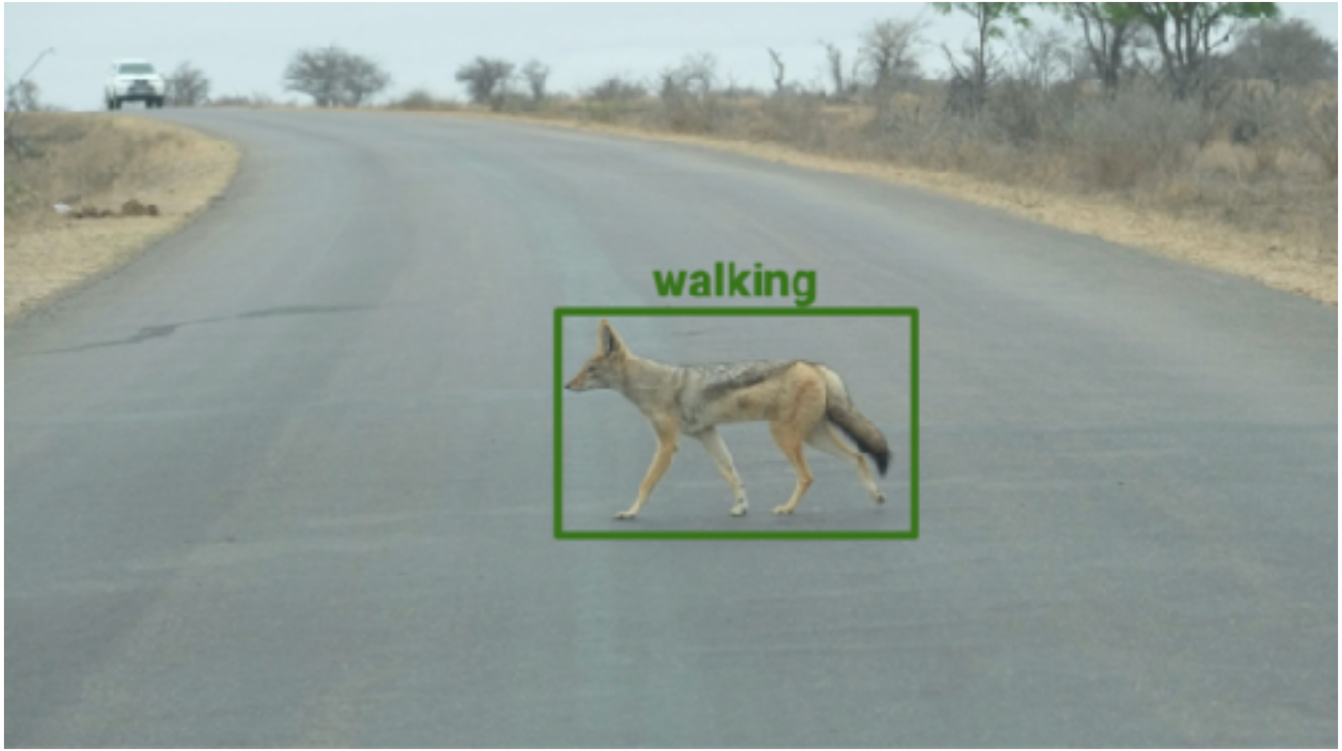
Vision Tasks

Object Detection



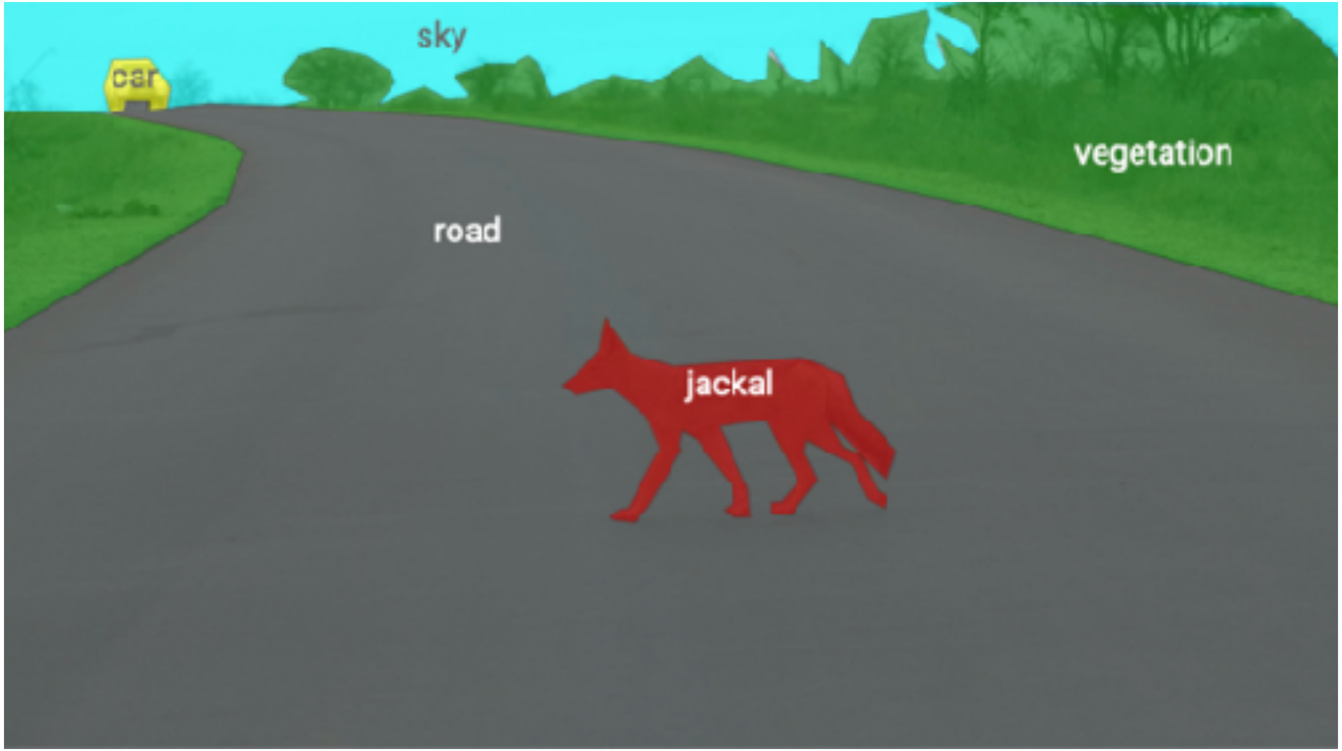
Credits Naila Murray

Activity recognition



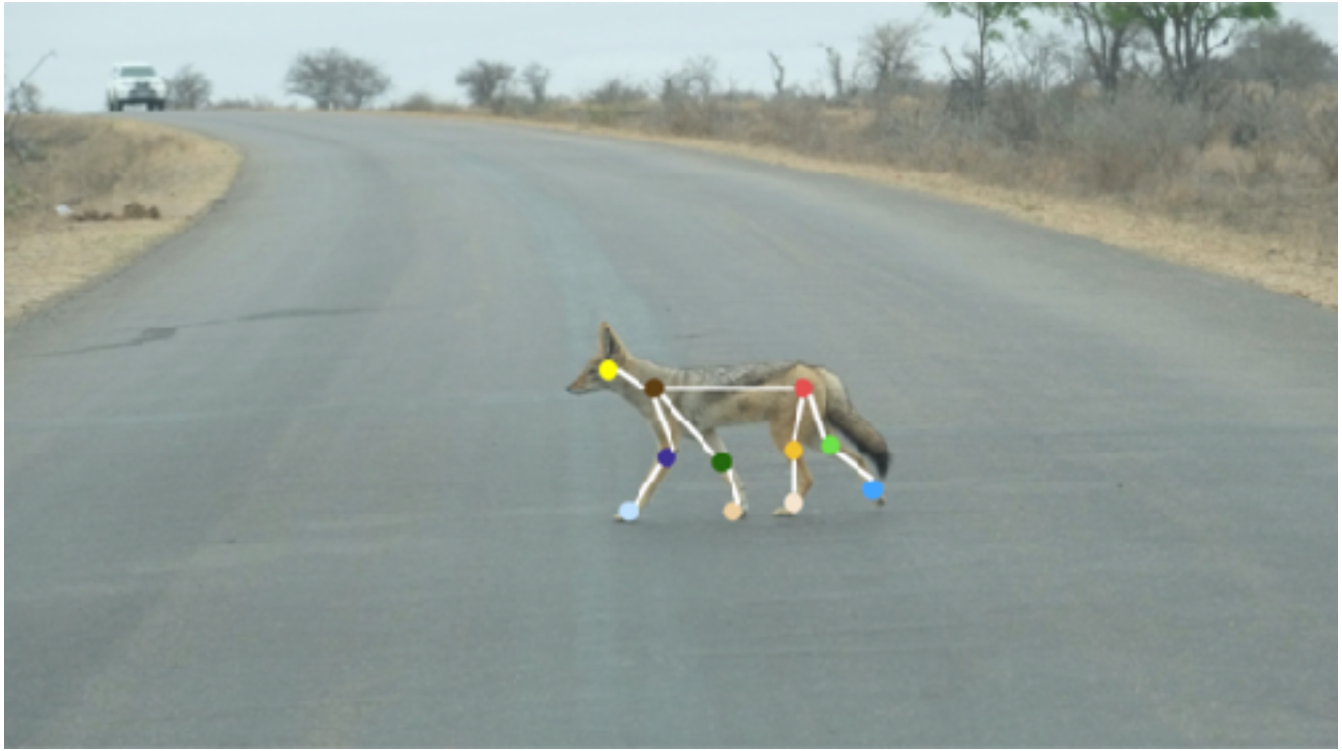
Credits Naila Murray

Semantic Segmentation



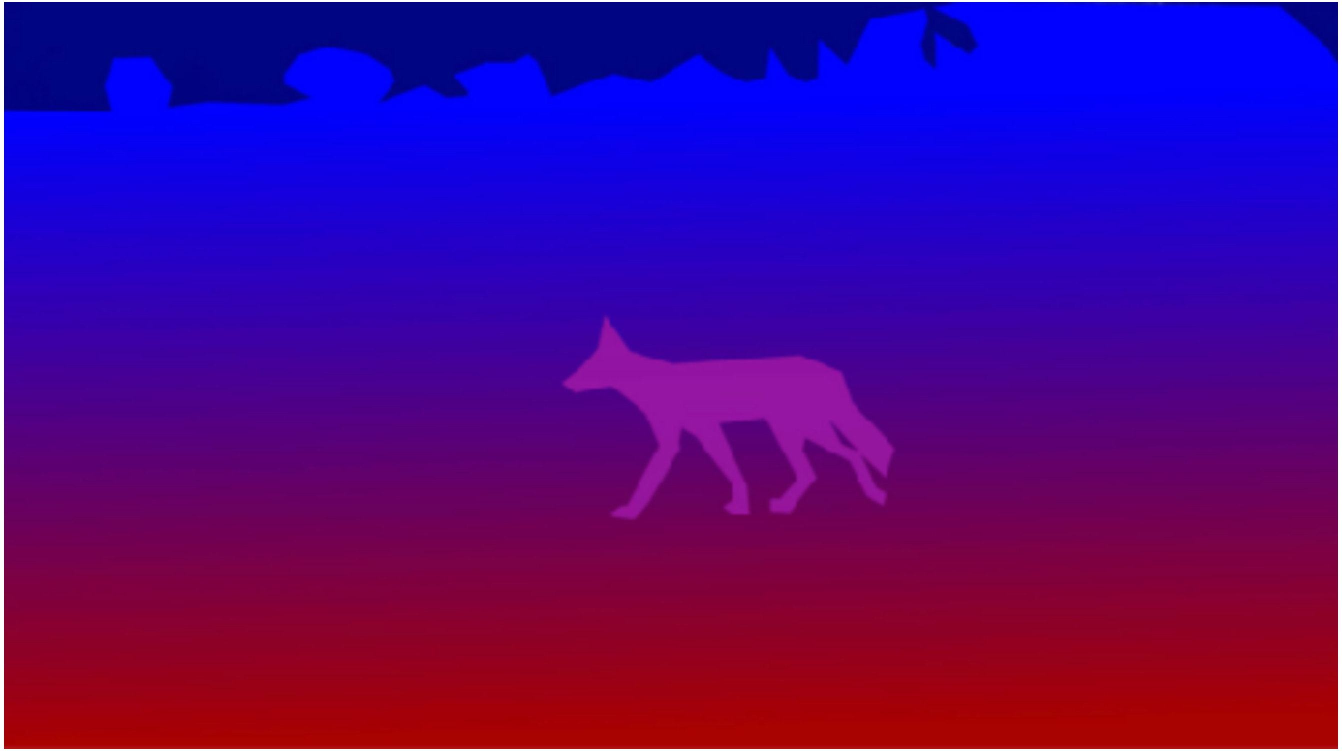
Credits Naila Murray

Pose Estimation



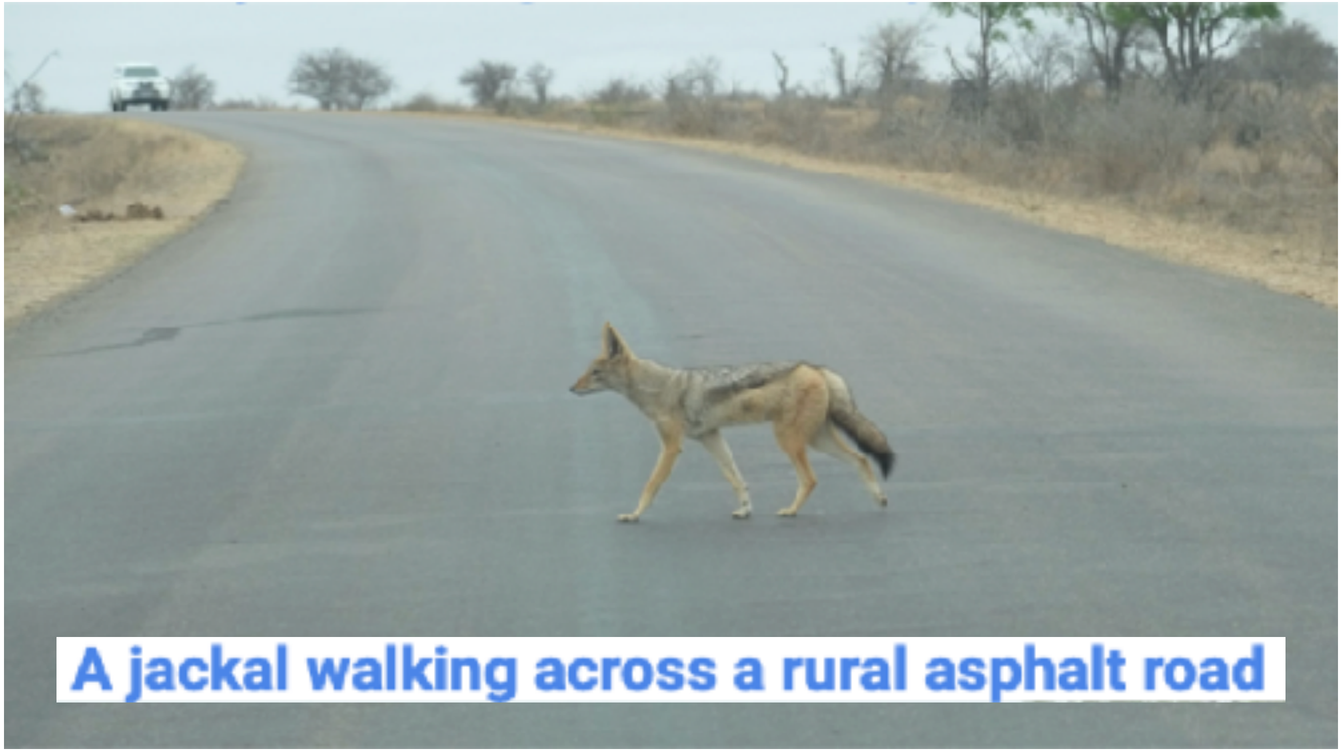
Credits Naila Murray

Depth Estimation



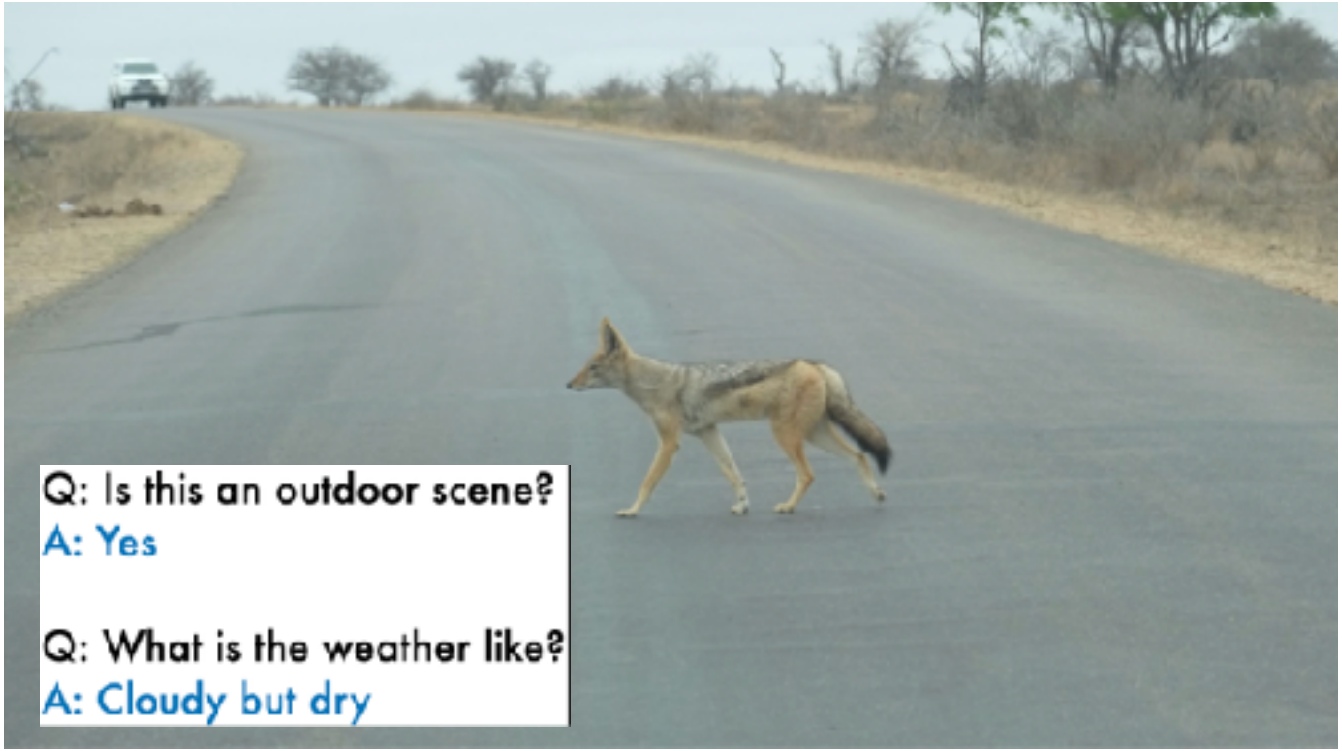
Credits Naila Murray

Captioning



Credits Naila Murray

Visual Question Answering



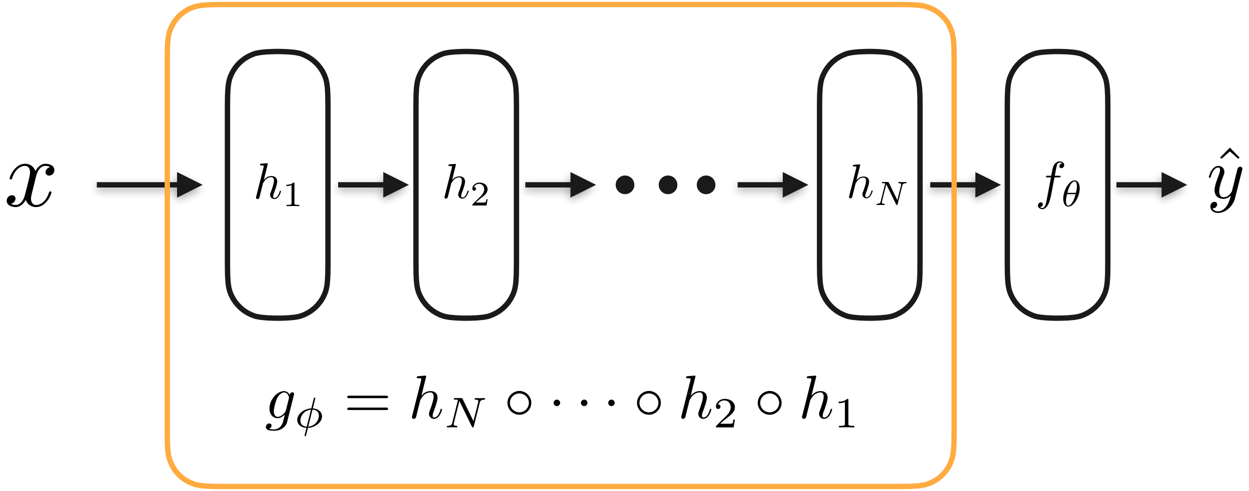
Q: Is this an outdoor scene?
A: Yes

Q: What is the weather like?
A: Cloudy but dry

Credits Naila Murray

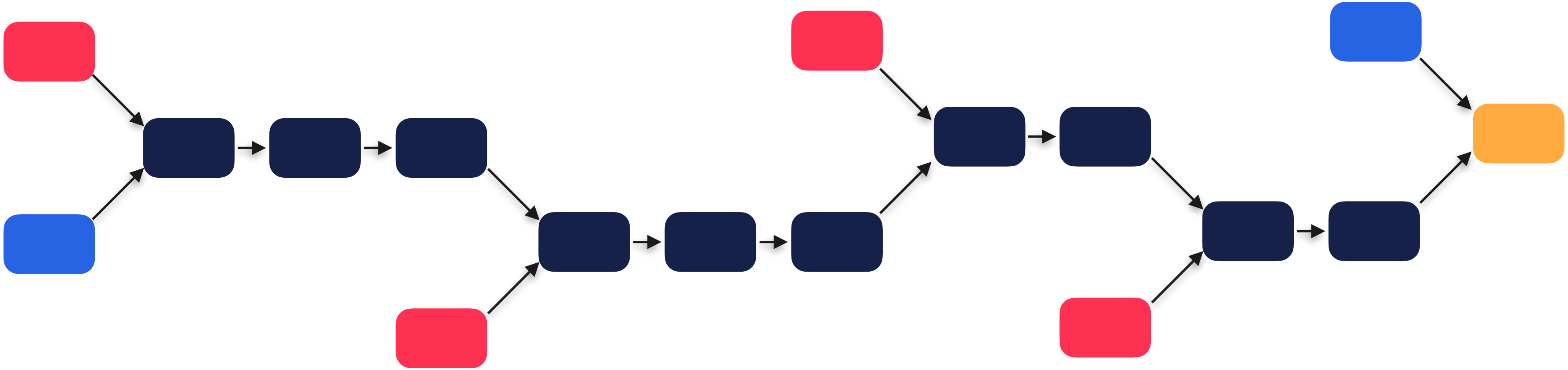
Representation learning a.k.a. Deep Learning

Deep Neural Networks are **composition** of **nonlinear functions,** also called **layers.**



Representation: feature extractor, **learned** from data

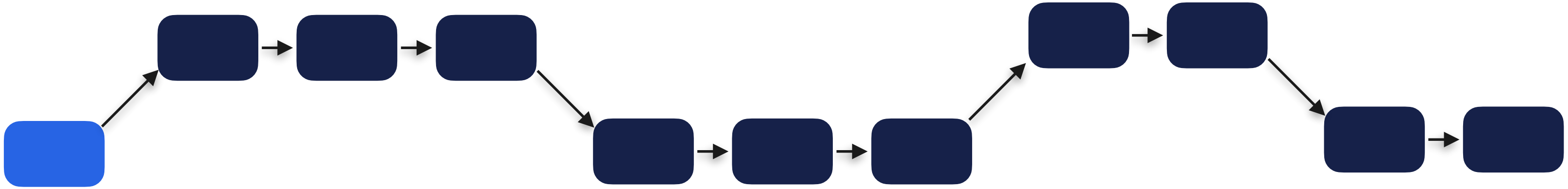
Neural Networks as computational graphs

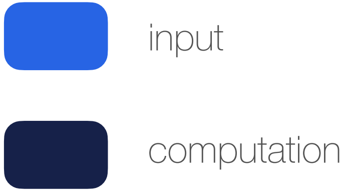


🟦	input	🟧	loss
⬛	computation	🟥	parameters

Credits DeepMind x UCL Lectures - Sander Dieleman

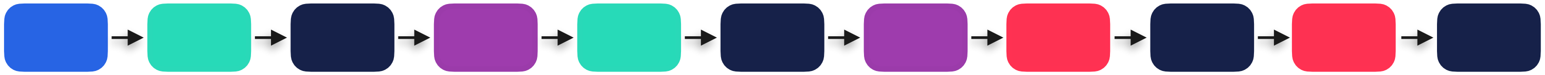
Simplified diagram: implicit parameters and loss





Credits DeepMind x UCL Lectures - Sander Dieleman

Building blocks of convnets



🟥 fully-connected layer

🟦 input

🟩 convolution layer

⬛ nonlinearity

🟪 pooling layer

and several other tricks…

Credits DeepMind x UCL Lectures - Sander Dieleman

What is a layer?

Typically is just **matrix multiplication!** But the function can take many forms.

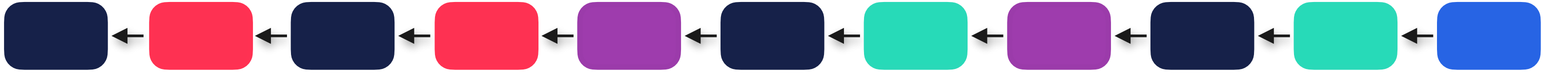
- 🟥 **Fully connected** layer
- 🟩 **Convolutional** layer
- 🟪 **Pooling** layer (e.g., Max Pooling, Avg Pooling)
- ⬛ **Non-linearity** (e.g., ReLU, GeLU, Tanh, Sigmoid…)
- 🟧 **Normalization** layer
- 🟦 **Attention** layer

Backprop

$$x_N = (h_N \circ h_{N-1} \circ \cdots \circ h_2 \circ h_1)(x_0)$$

The only requirement for layers is to be **differentiable** to be optimised with gradient descent

Backprop



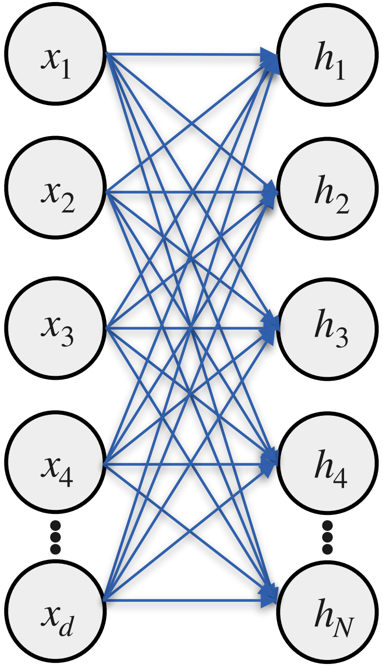
$$x_N = (h_N \circ h_{N-1} \circ \cdots \circ h_2 \circ h_1)(x_0)$$

$$\frac{dx_n}{dx_0} = \frac{dh_N}{dh_{N-1}} \times \frac{dh_{N-1}}{dh_{N-2}} \times \cdots \times \frac{dh_1}{dx_0}$$

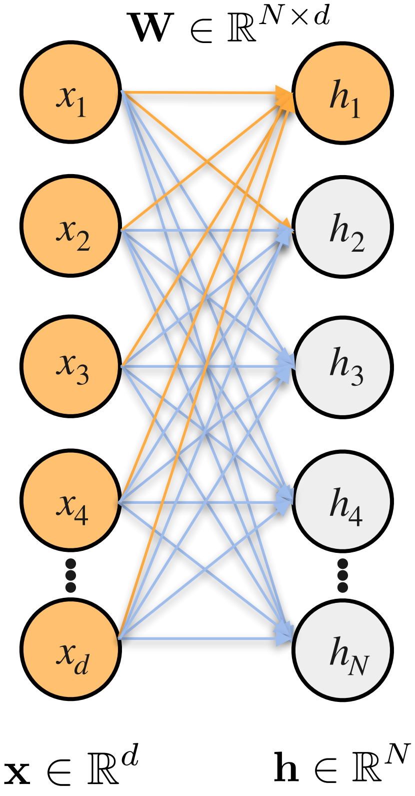
The only requirement for layers is to be **differentiable** to be optimised with gradient descent

Building blocks

Fully-connected Dense Layers



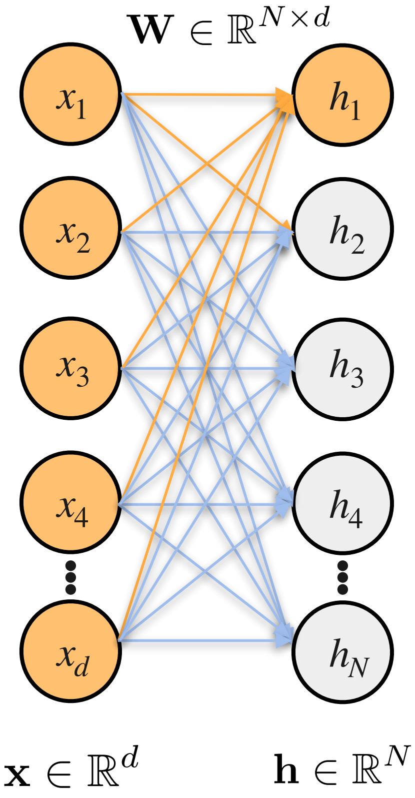
Fully-connected (Dense) Layers


$\mathbf{W} \in \mathbb{R}^{N \times d}$
x_1
x_2
x_3
x_4
x_d
h_1
h_2
h_3
h_4
h_N
$\mathbf{x} \in \mathbb{R}^d$
$\mathbf{h} \in \mathbb{R}^N$

$$h_i = \sum_{i=1}^{d} w_{ij} x_j + b_i$$

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Fully-connected (Dense) Layers

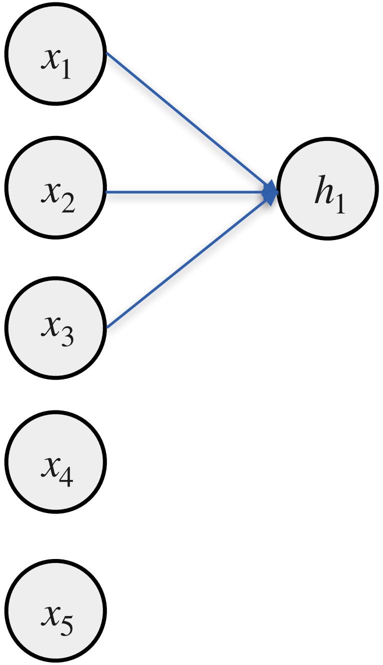

$\mathbf{W} \in \mathbb{R}^{N \times d}$
x_1
x_2
x_3
x_4
x_d
h_1
h_2
h_3
h_4
h_N
$\mathbf{x} \in \mathbb{R}^d$
$\mathbf{h} \in \mathbb{R}^N$

$$h_i = \sum_{i=1}^{d} w_{ij} x_j + b_i$$

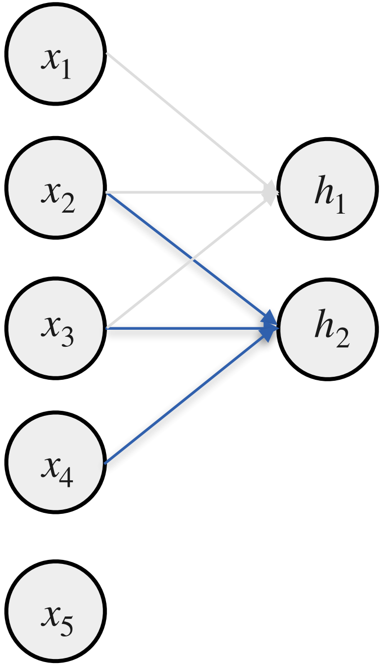
$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{h} = W\mathbf{x} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{Nd} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & \cdots & h_N \end{bmatrix}$$

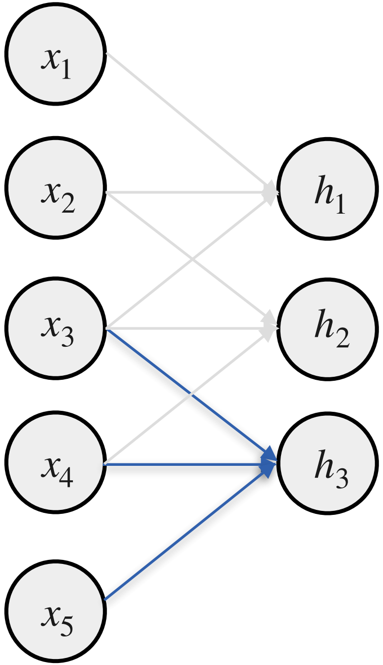
Convolutional Layer



Convolutional Layer



Convolutional Layer



Convolution operator

=

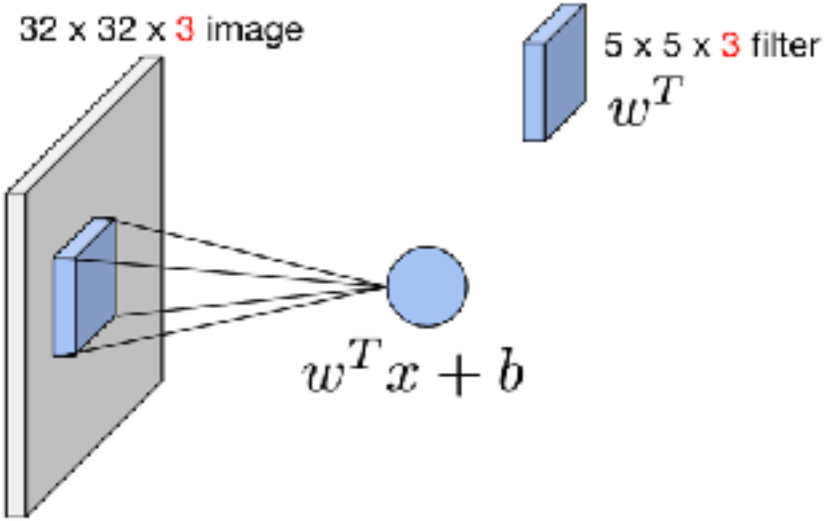
Local connectivity + **Parameter Sharing**

Apply a set of weights (filter or kernel) to extract local features
Spatially share filters to reduce the number of parameters

Translation Equivariance

Extract **same features** at every location

Convolutional Layer

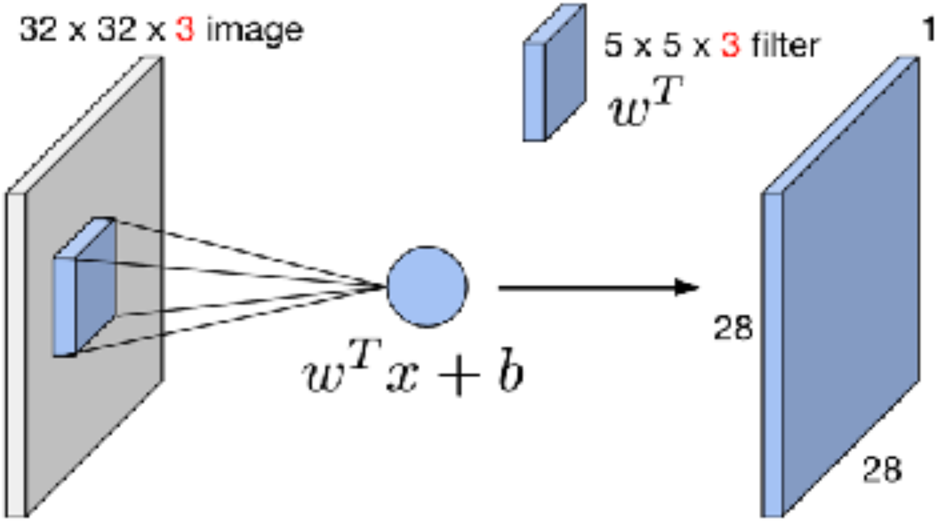


We **Convolve** the “*kernel*” or “*filter*” with the image

i.e. slide the filter over all the input locations and compute the dot product

This is a **local** linear combination of the input features.

Convolutional Layer



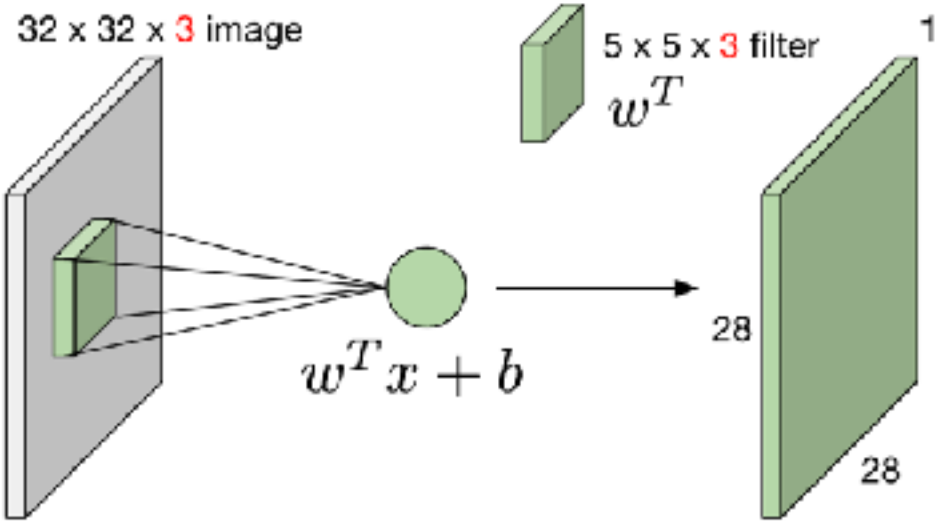
The result of the convolution between the input and the kernel is called

“feature map” or **“activation map”**

Depending on the type of convolution the feature map will have a different size:

- VALID (n - k + 1)
- SAME
- FULL (n + k -1)

Convolutional Layer

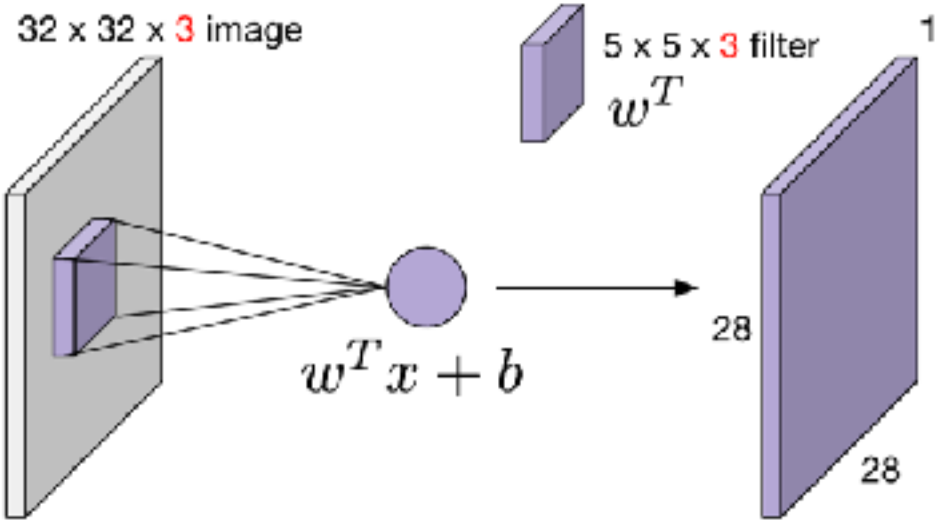


Each color corresponds to **different filters:**

- **Sharing** parameters among all the locations of the input
- And extracting features **independently on the location** in which is applied

The result of the convolution between the input and a kernel gives a feature map.

Convolutional Layer

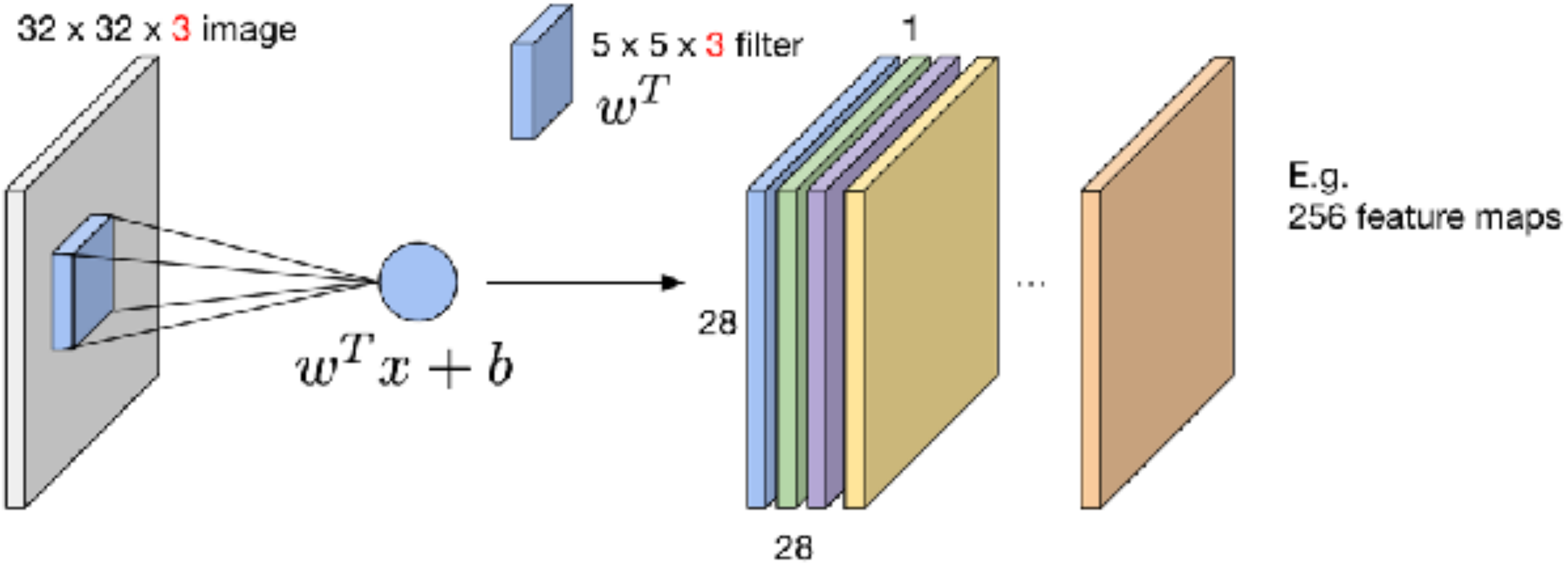


Each color corresponds to **different filters:**

- **Sharing** parameters among all the locations of the input
- And extracting features **independently on the location** in which is applied

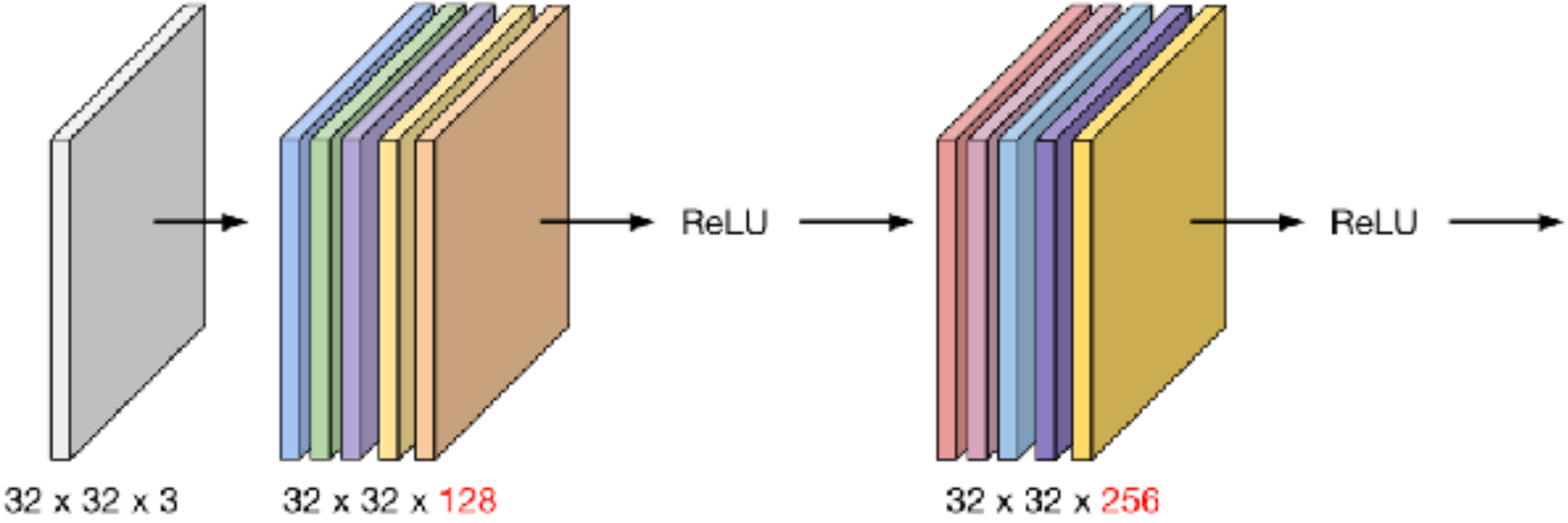
The result of the convolution between the input and a kernel gives a feature map.

Convolutional Layer



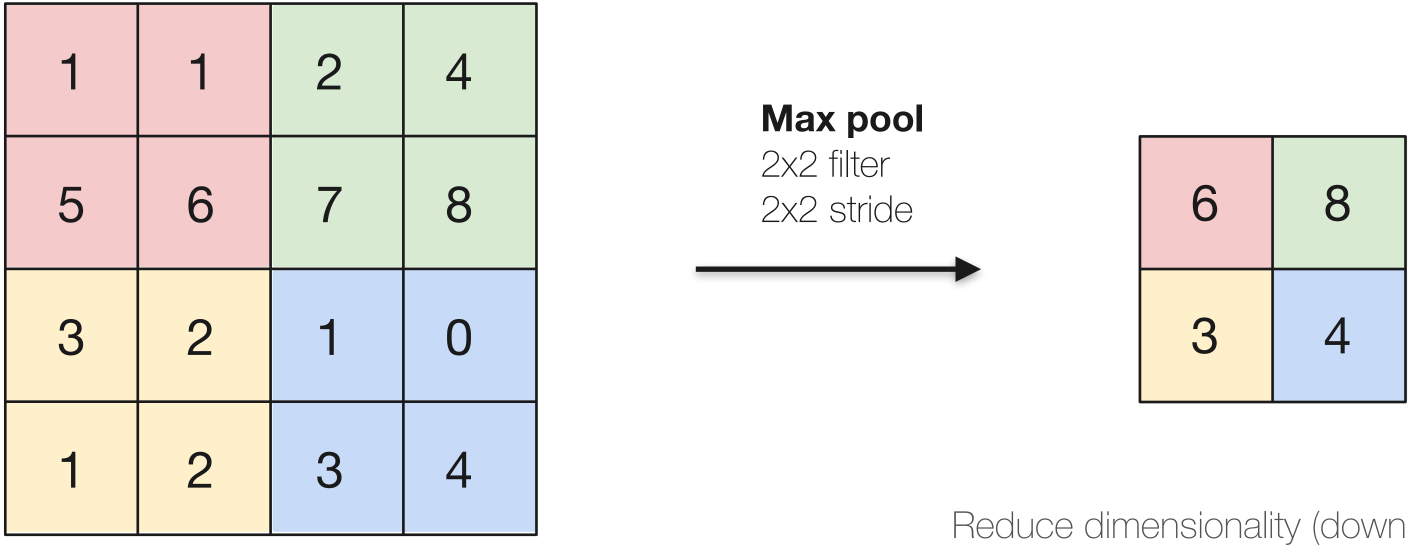
In a CNN we want to learn many different filters,
each one specialized on extracting a specific feature in the image.
So we are going to have **several feature maps**!

Convolutional Layer



CNNs constraint the hypothesis space to functions with only **local interactions** and **translation equivariance**

Spatial Max Pooling



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

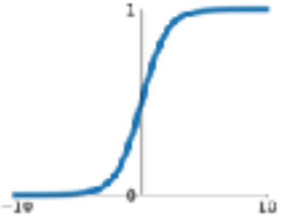
Max pool
2x2 filter
2x2 stride

6	8
3	4

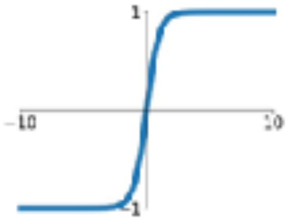
Reduce dimensionality (downsampling)
Preserve spatial invariance

Nonlinearities (Activation functions)

$$f(x) = \frac{1}{1+e^{-x}}$$

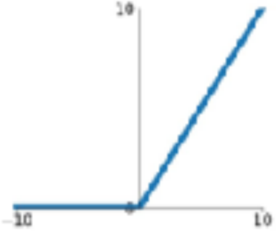


$$f(x) = \tanh(x)$$

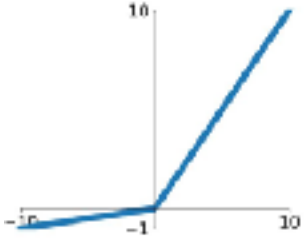


Some are better than others for training, but each has a specific role.

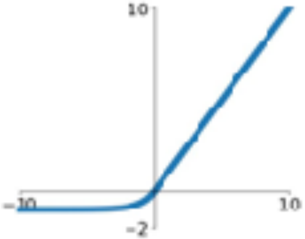
$$f(x) = \max(0, x)$$



$$f(x) = \begin{cases} x & \text{if x} \geq 0 \\ \alpha x & \text{if x} < 0 \end{cases}$$

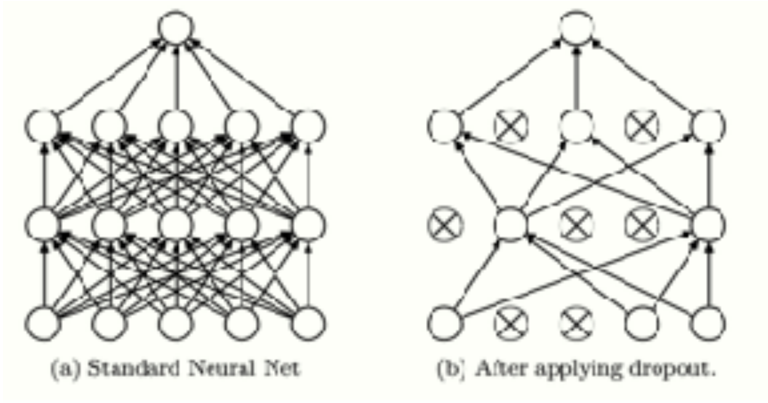


$$f(x) = \begin{cases} x & \text{if x} \geq 0 \\ \alpha e^{x} & \text{if x} < 0 \end{cases}$$



Dropout regularization

Idea: Randomly turn off some neurons of the network



(a) Standard Neural Net

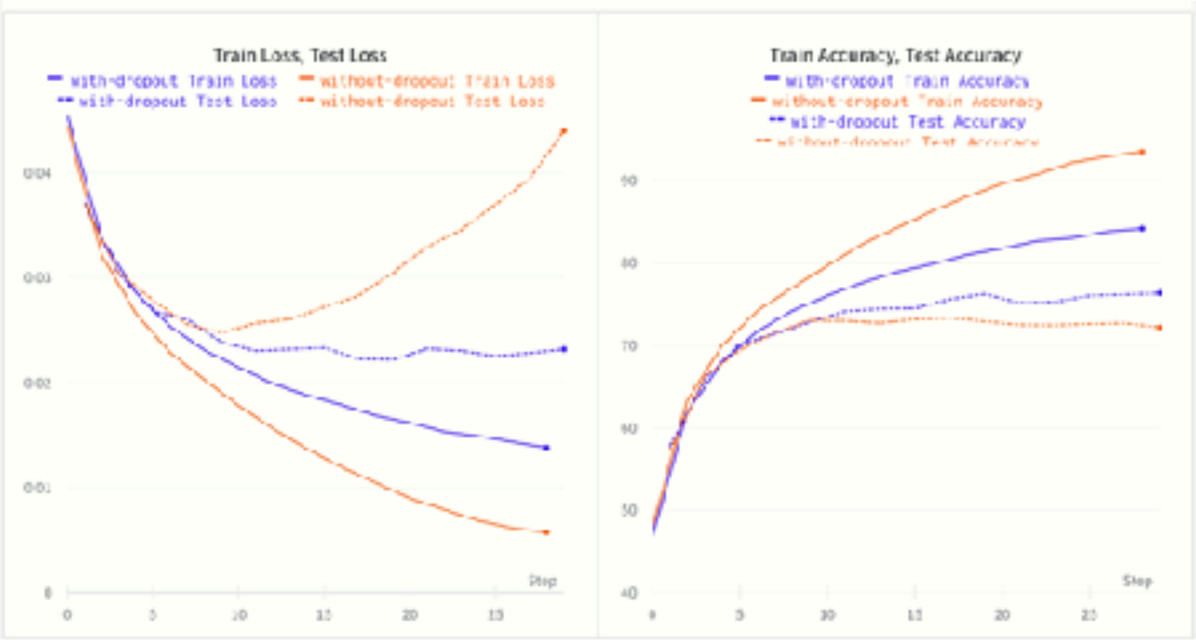
(b) After applying dropout.

$$
\begin{aligned}
m_j^{\ell} &\sim \mathrm{Bernoulli}(p) \\
\tilde{\mathbf{y}}^{\ell} &= \mathbf{m}^{\ell} * \mathbf{y}^{\ell} \\
z_i^{\ell+1} &= \mathbf{w}_i^{\ell+1} \tilde{\mathbf{y}}^{\ell} + b_i^{\ell+1} \\
y_i^{\ell+1} &= \sigma\left(z_i^{\ell+1}\right).
\end{aligned}
$$

- Each hidden unit is set to zero with p probability
- Each layer can have a different prob

Force weights to learn independent features by preventing hidden units to rely on other units (co-adaptation).

Dropout regularization



Reduce overfitting!

Data Augmentation

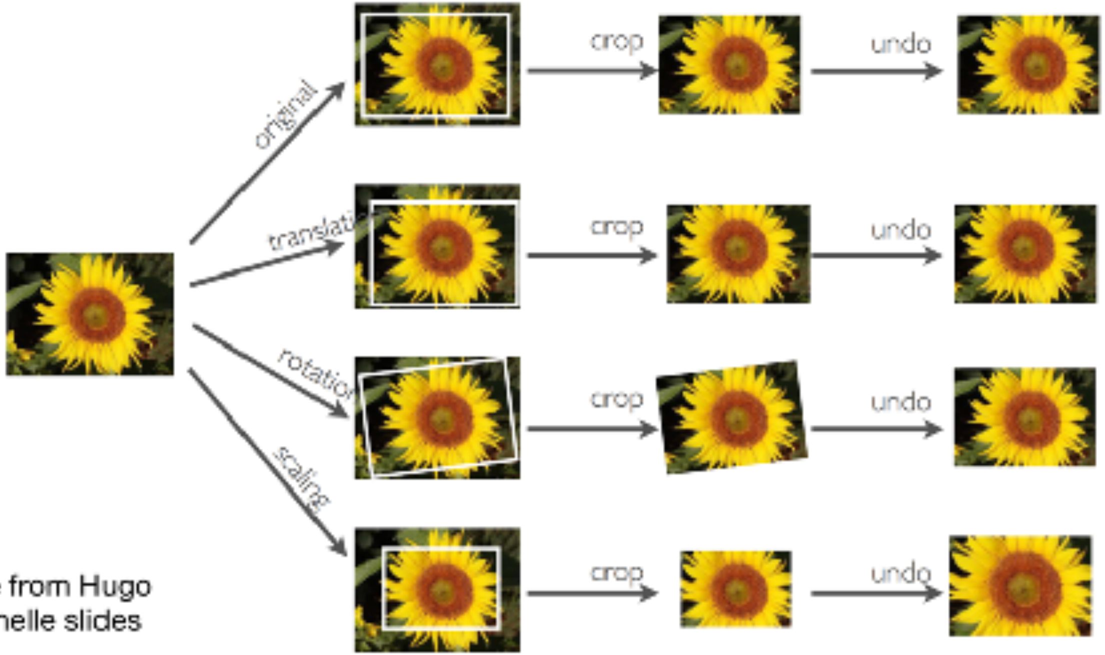
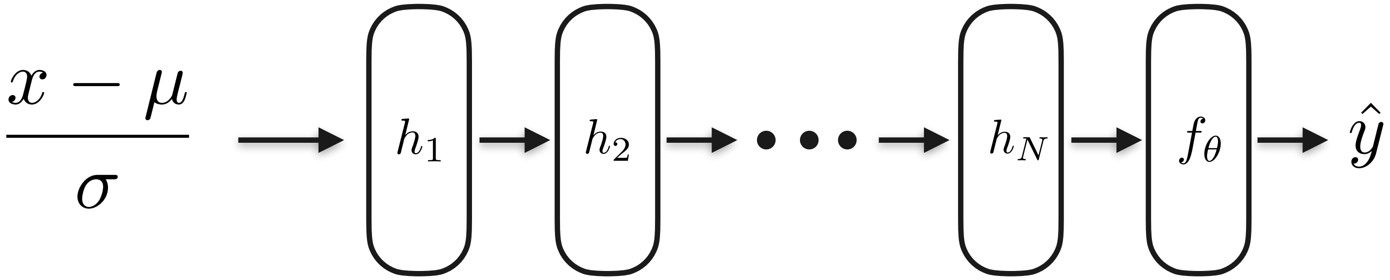


Figure from Hugo Larochelle slides

Augment the dataset with several transformation
It can be done online at every minibatch
to improve network robustness and generalisation

Input normalization

e.g., standardize input to have **zero-mean** and **unitary variance**

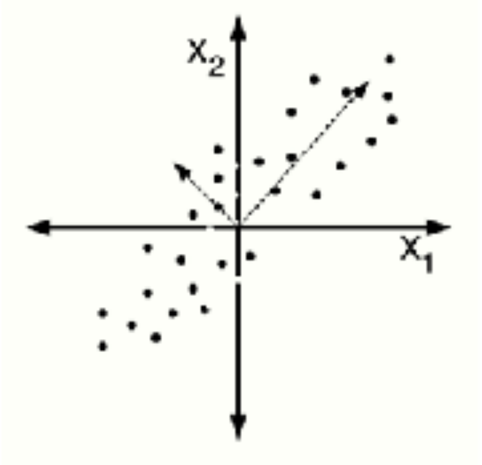


Improve convergence of gradient descent!

Lecun et al, 1998 Efficient Backprop

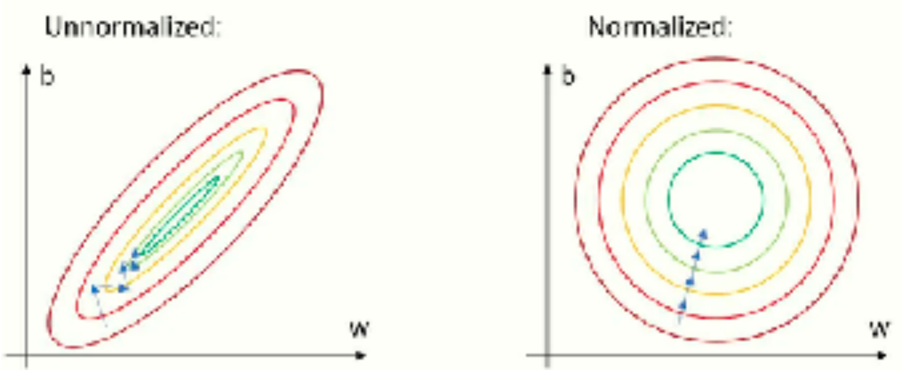
Input normalization (linear example)

$$\mathcal{L}(W) = \frac{1}{2N}\sum_{i=0}^{N}(\mathbf{y}_i - \mathbf{W}^\top \mathbf{x}_i)^2$$



Convergence speed of gradient descent depends on the **condition number** of the Hessian matrix $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$

Eigenvalues measure the **spread** of the input in its dimensions

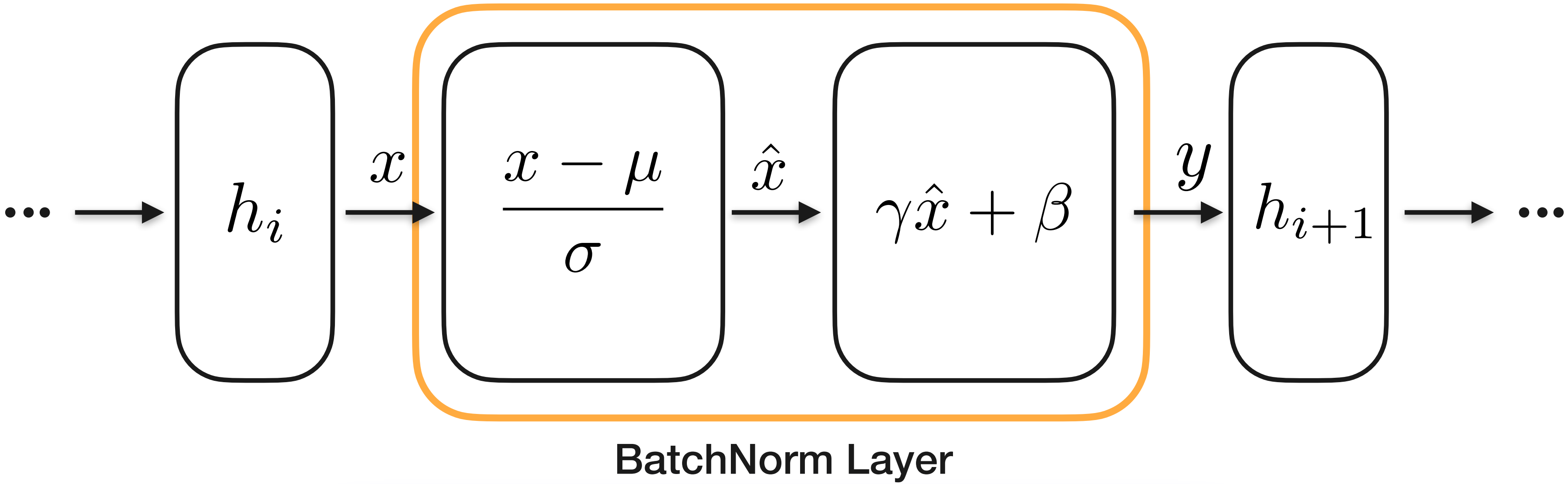


$$H = \nabla^2 \mathcal{L} = \frac{1}{N}\sum_{i=0}^{N}\mathbf{x}_i\mathbf{x}_i^\top$$

Hessian = Input Covariance matrix
(in this linear example)

Can we use normalisation for intermediate computations?

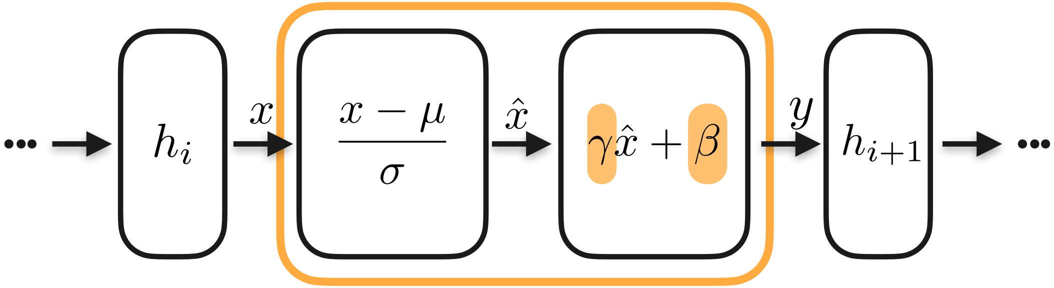
Batch Normalization (BN)



BatchNorm Layer

Normalize **input of each hidden layer** and learn how much to normalize to be more expressive

Batch Normalization



Train mode
Use **batch statistics** to normalize activations

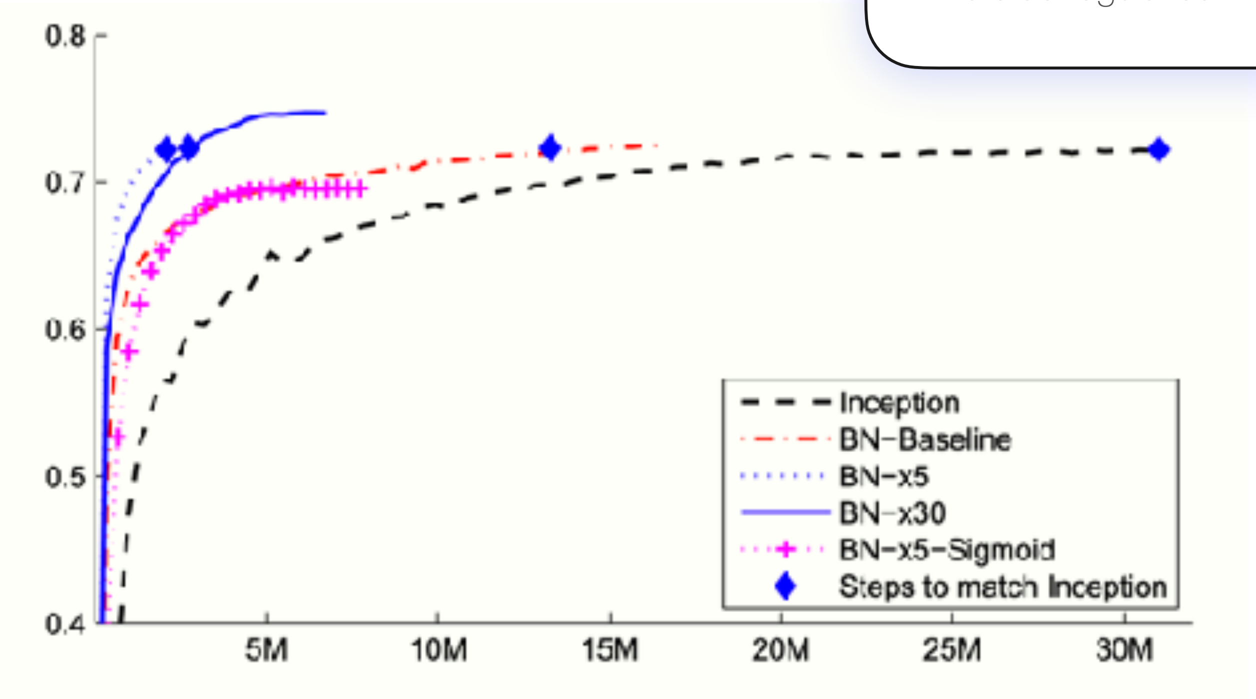
≠

Inference mode
Use dataset statistics, Collected

Be sure to be in the correct mode!
Frequent cause of bugs…

Batch Normalization

- Faster convergence
- Less sensitive to hyper parameters
- Acts as regulariser



CNNs are simply a composition of
conv layers, nonlinearities, normalisation and other blocks

Everything is differentiable, so backprop for learning parameters!

CNNs were not invented over night

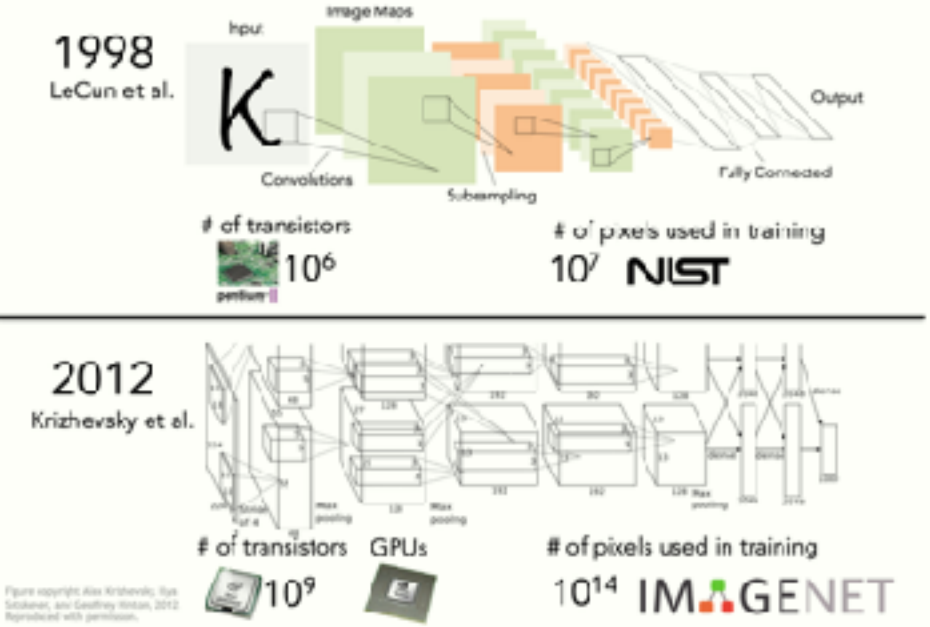
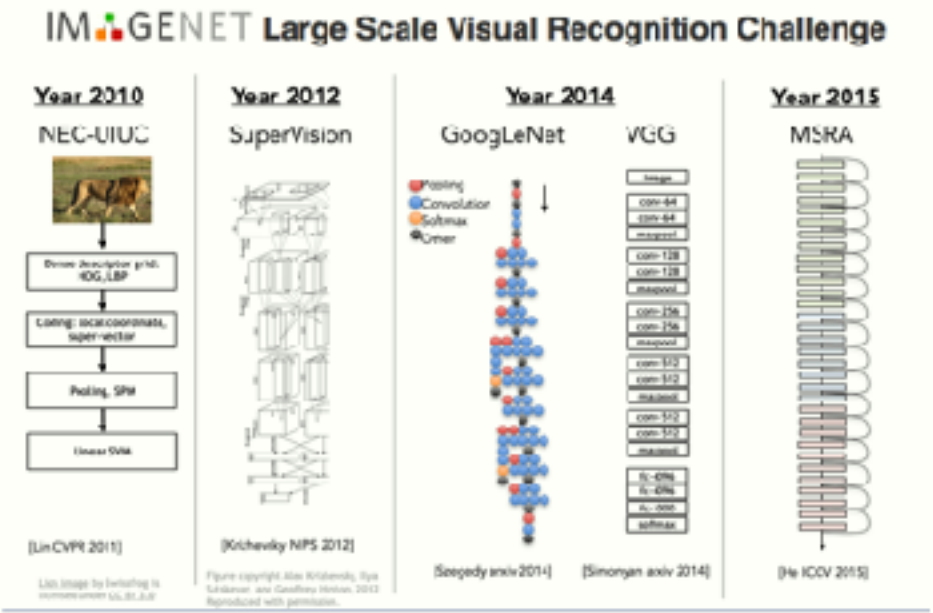


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.



[Lin CVPR 2011]

[Krizhevsky NIPS 2012]

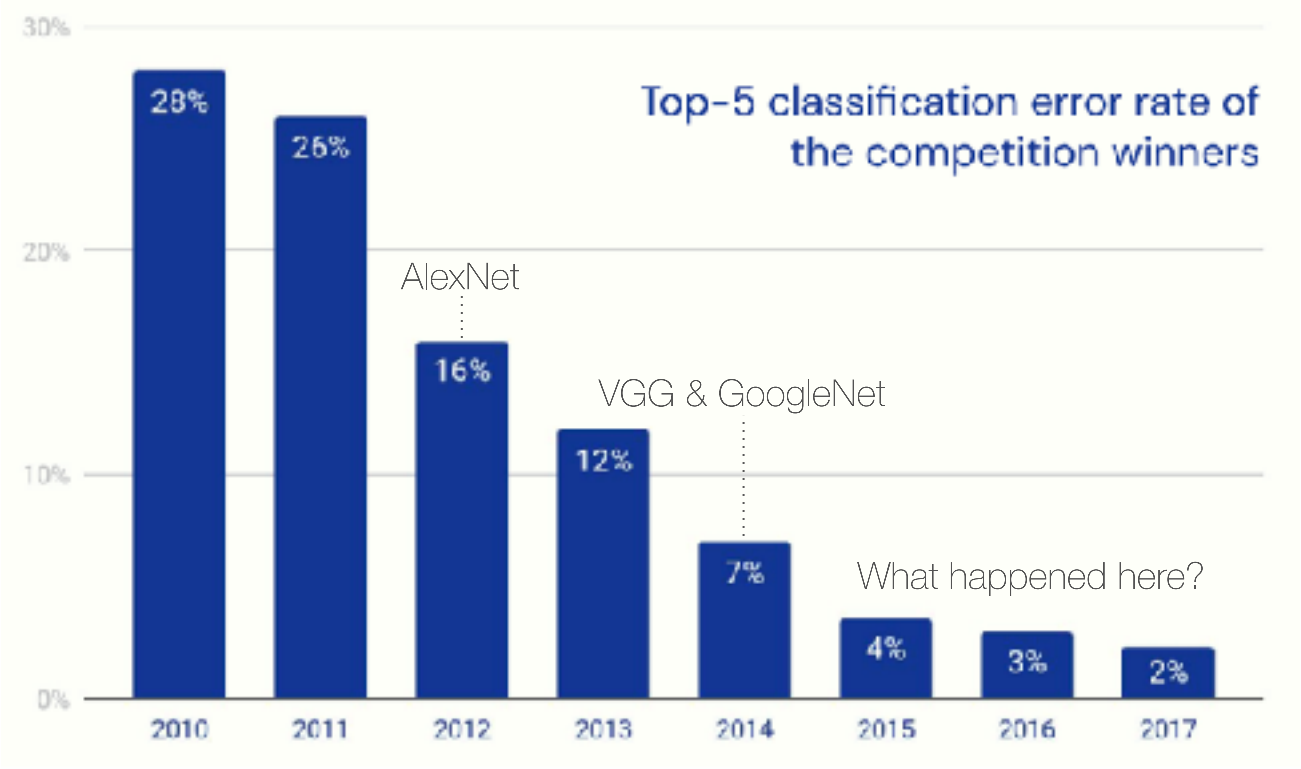
[Szegedy arxiv 2014]

[Simonyan arxiv 2014]

[He ICCV 2015]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Progress on ImageNet



AlexNet

VGG & GoogleNet

What happened here?

That’s it? Should we just stack more layers?

Expressivity & Trainability

* great blog https://blog.evjang.com/2017/11/exp-train-gen.html

Slides inspired by Orhan Firat M2L Lecture

Expressivity & Trainability

$$\text{quality} = f(\mathcal{D}, \theta, \lambda)$$

What functions your model can approximate

How good can you train your model (optimization)

$\mathcal{D}$ = Data

θ = Model

λ = Loss, optimizer, hyperparams

* great blog https://blog.evjang.com/2017/11/exp-train-gen.html

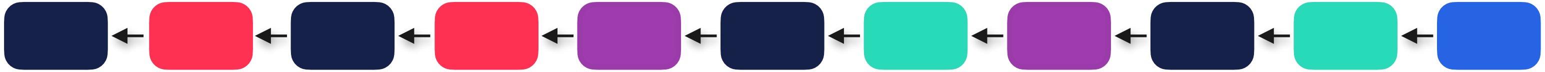
Slides inspired by Orhan Firat M2L Lecture

Credit Assignment Problem in Machine Learning

1) Evaluate all the intermediate computations that the predictor has made
   - understand what could have been done differently to obtain a better outcome
   - assign the **proper credit** to each **decision (or unit)** involved in the prediction
2) Update the system accordingly

Representations learning via Neural Networks
uses back-propagation and gradient descent

Vanishing gradient problem



$$x_N = (h_N \circ h_{N-1} \circ \cdots \circ h_2 \circ h_1)(x_0)$$

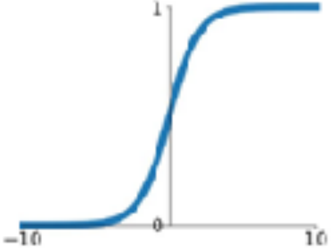
$$\frac{dx_n}{dx_0} = \frac{dh_N}{dh_{N-1}} \times \frac{dh_{N-1}}{dh_{N-2}} \times \cdots \times \frac{dh_1}{dx_0}$$

Earlier units do net get any credit for their computation (bad or good prediction) because the **gradients vanishes (or even explode) over depth** because of successive multiplication

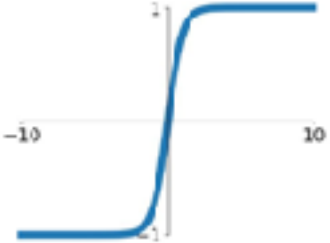
Choosing the correct nonlinearity

Saturating units

$$f(x) = \frac{1}{1+e^{-x}}$$



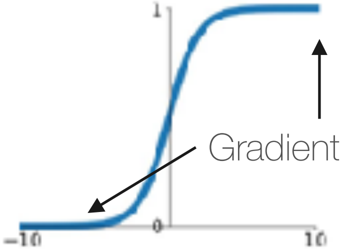
$$f(x) = \tanh(x)$$



Choosing the correct nonlinearity

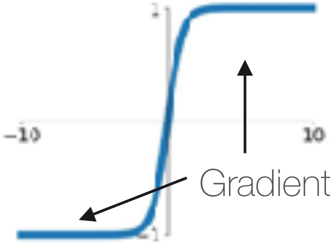
Saturating units

$$f(x) = \frac{1}{1 + e^{-x}}$$



Gradient goes to zero

$$f(x) = \tanh(x)$$

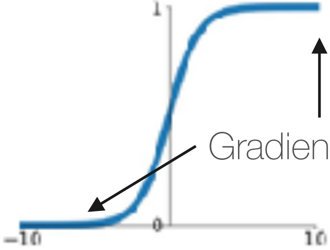


Gradient goes to zero

Choosing the correct nonlinearity

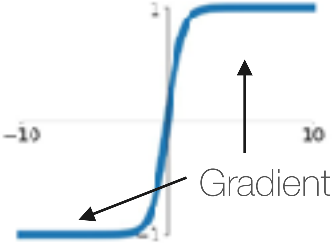
Saturating units

$$f(x) = \frac{1}{1 + e^{-x}}$$



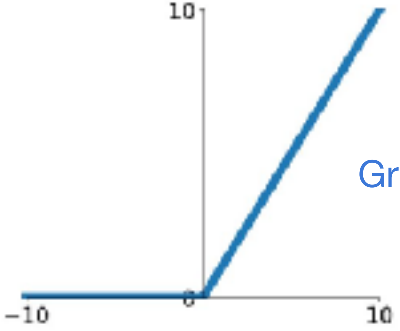
Gradient goes to zero

$$f(x) = \tanh(x)$$



Gradient goes to zero

Non-saturating units

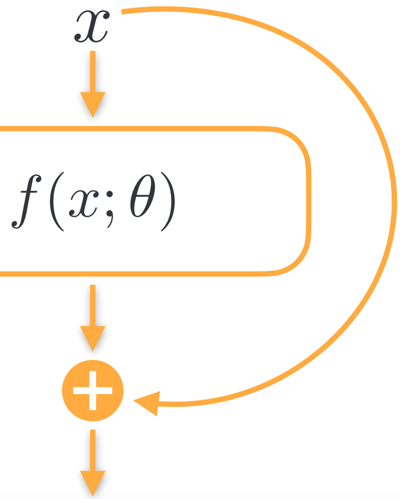


Gradient is 1

$$f(x) = \max(0, x)$$

Residual and Highway Networks

Redesign neural networks to make them **easier to optimize even for very large depths** (solving the vanishing gradient problem)



“Shortcut or skip connection”

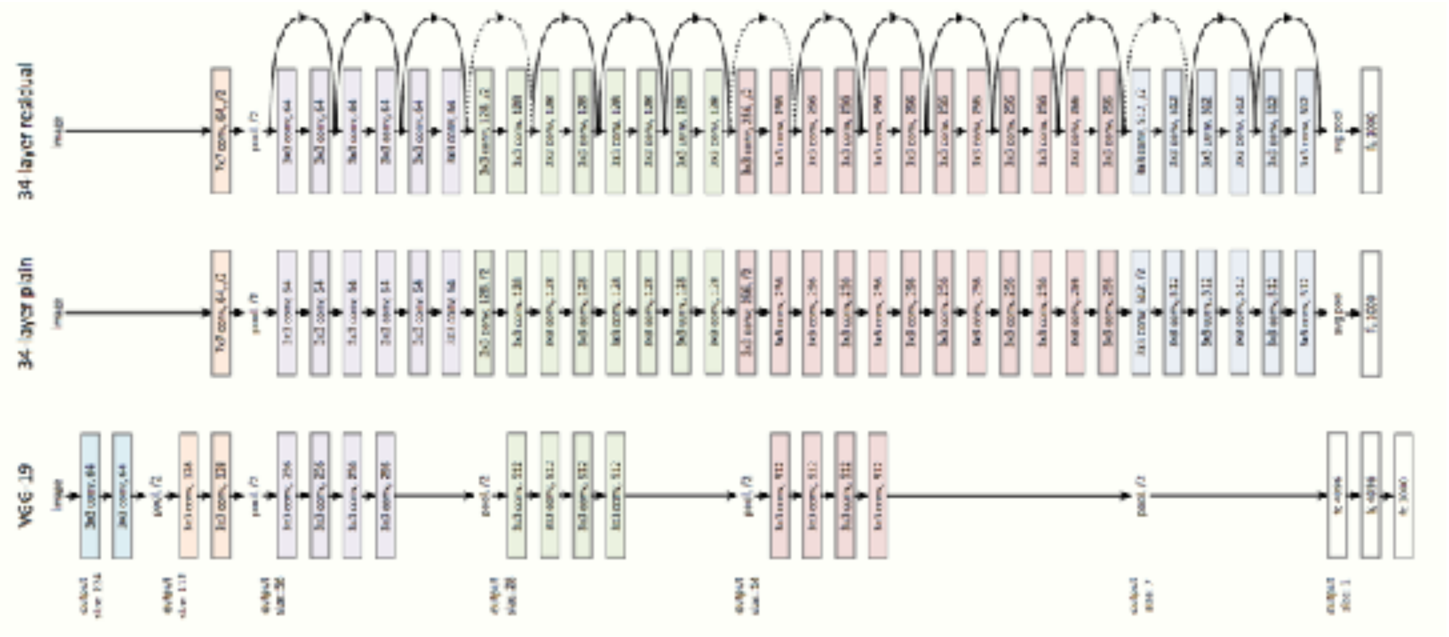
The **gradient can skip layers** of computation to assign credit to initial units.

$$y = x + f(x;\theta)$$

$$y = x \cdot \underbrace{C(x;\theta_C)}_{\text{Carry gate}} + f(x;\theta) \cdot \underbrace{T(x;\theta_T)}_{\text{Transform gate}}$$

He et al. (2015)
Srivastava et al. (2015)

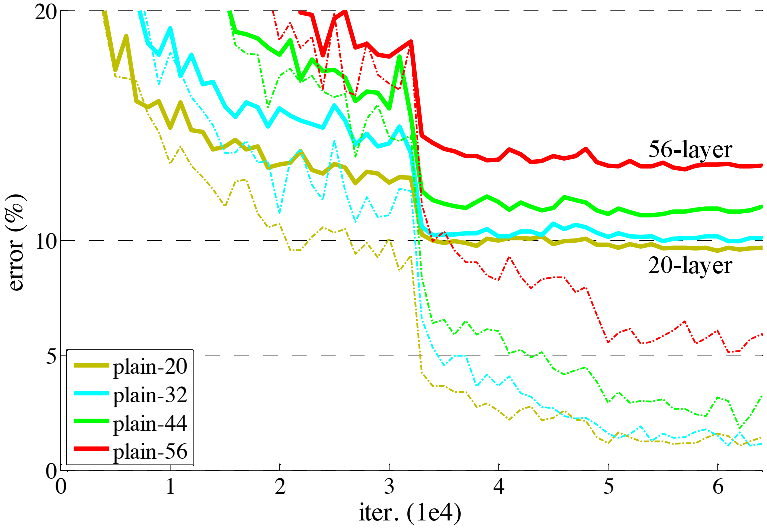
ResNets



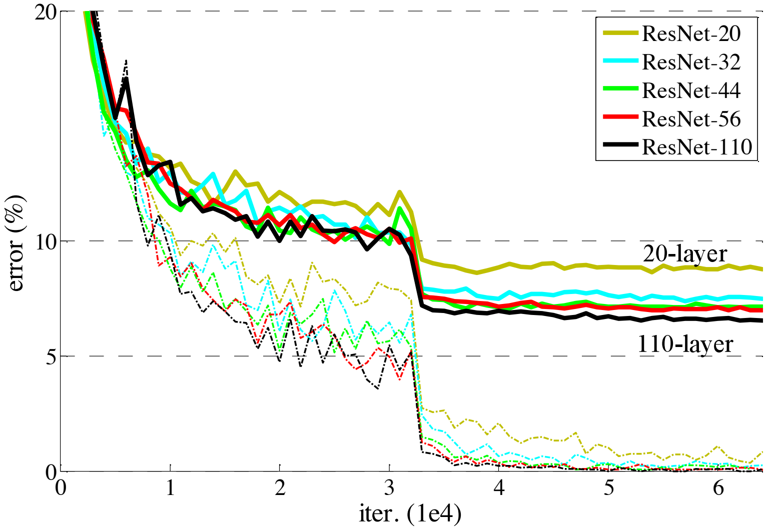
Deep Residual Learning for Image Recognition, He et al (2015)

Residual and Highway Networks

Plain Networks



Residual Networks



He et al. (2015)

Additive Compositional Layers

Most of the successfully trained **very deep architectures** share a **core building block** to compute a vector representation at **layer $k+1$,** parametrised by $\theta(k)$:

$$x(k+1) = x(k) + f(x(k), \theta(k))$$

Previous Representation

Additive Non-Linear Transformation

Inductive bias: Iterative inference and features refinement

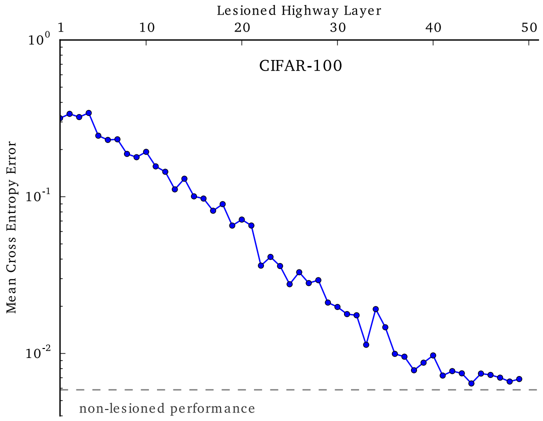
Iterative inference in ResNets and Highway Networks

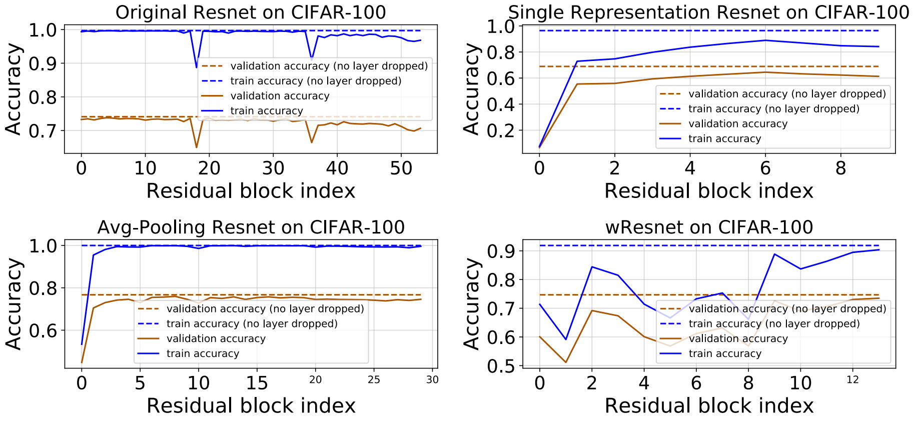
Lower residual blocks learn **hierarchical representations**

(each block discovers a different representation)

Higher residual blocks learn to perform **iterative inference** and **feature refinement.**

(keep the semantics of the representation of the previous layer)

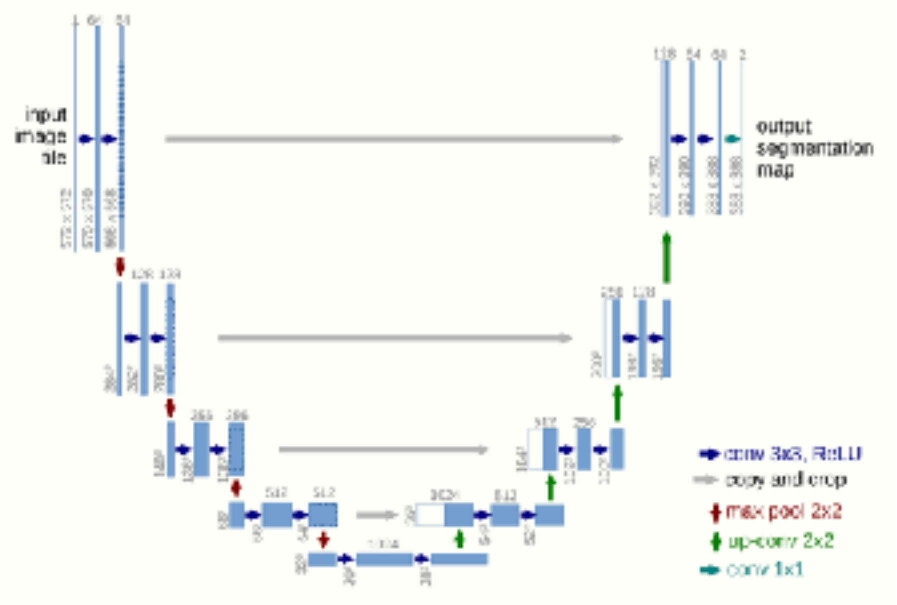




Greff et al. (2017)

Jastrzebski et al. (2018)

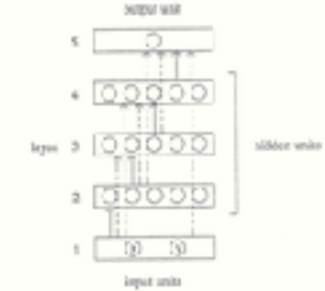
U-Net

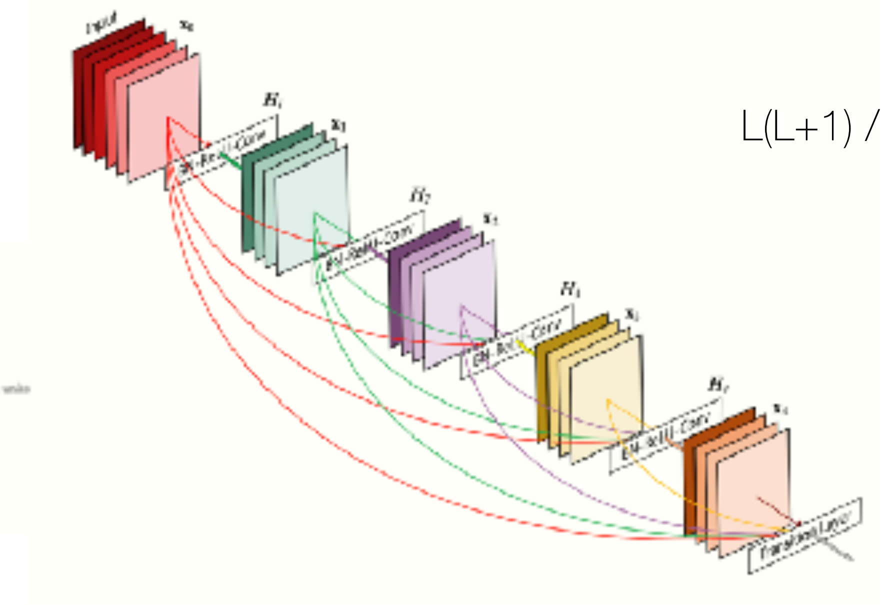


Skip connections from the encoder to the decoder for better gradient flow and details reconstruction

U-Net: Convolutional Networks for Biomedical Image Segmentation, Ronneberger et al 2015

DenseNet





L(L+1) / 2 direct connections

Lang, K. J., and Witbrock. M. (1988). “Learning to Tell Two Spirals Apart.” In Proc. of 1988 Connectionist Models Summer School

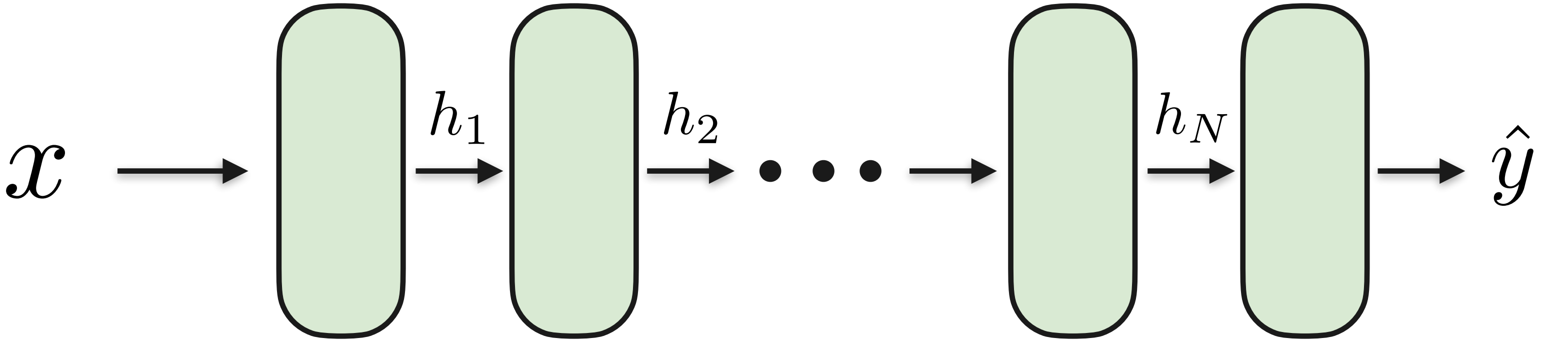
Densely Connected Convolutional Networks, Huang et al 2016

So far…

- **CNNs** are powerful models for image/video representation learning
  - Deeper and larger models need some **tricks (skip connections & normalisation)** to be trained because of **vanishing gradients** and **instabilities**

Sequence Modelling

Feed-forward Networks



$x \in \mathbb{R}^m$

$\hat{y} \in \mathbb{R}^n$

Information only **flows forward**

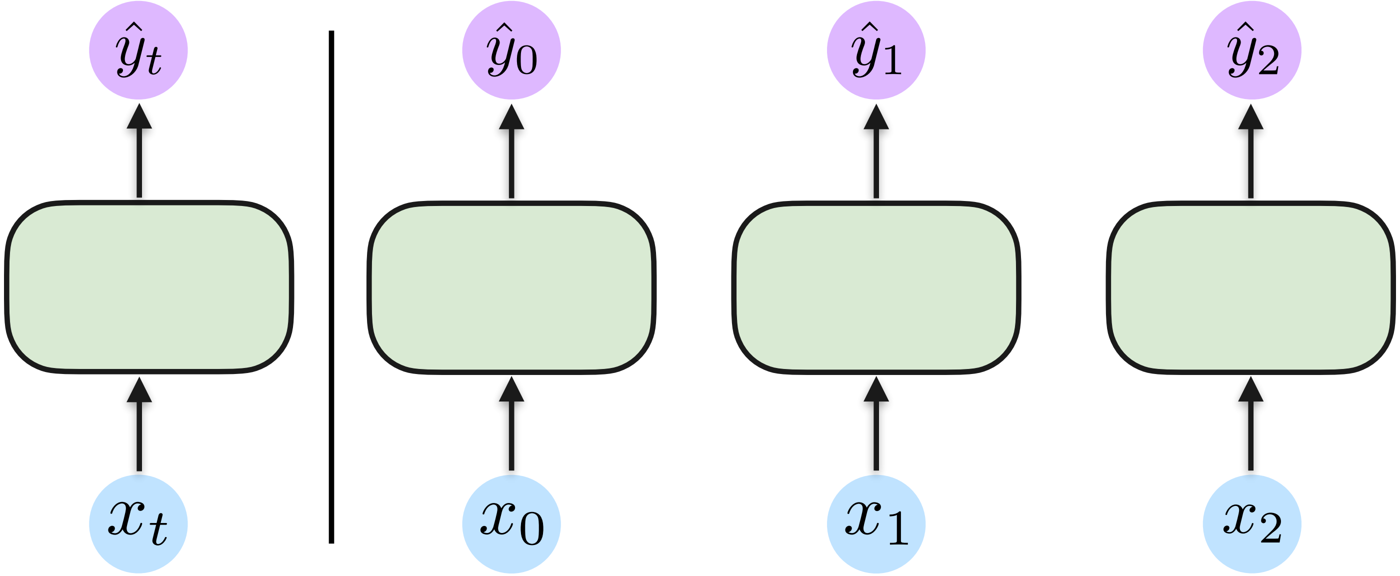
Handling sequences

$$\{x_0, x_1, \cdots, x_T\}$$

Correlated inputs with a **time dependence**

Memory-less models: naive

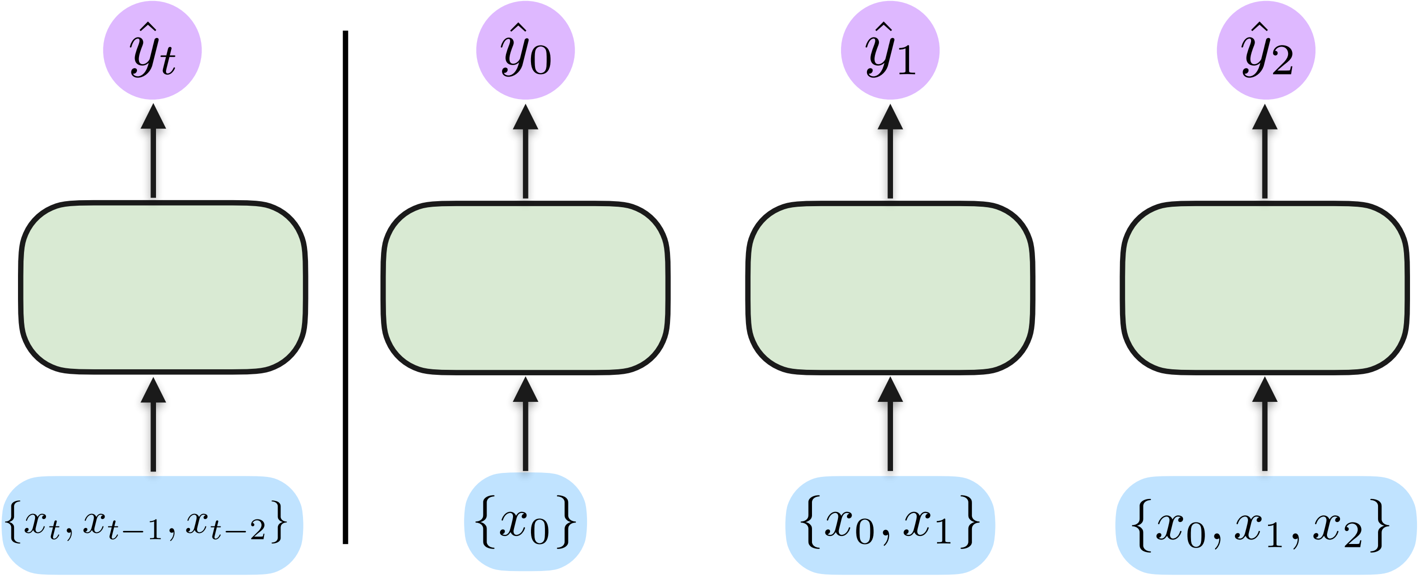
$\{x_0, x_1, \cdots, x_T\}$ Correlated inputs with a time dependence



output only depends on current time

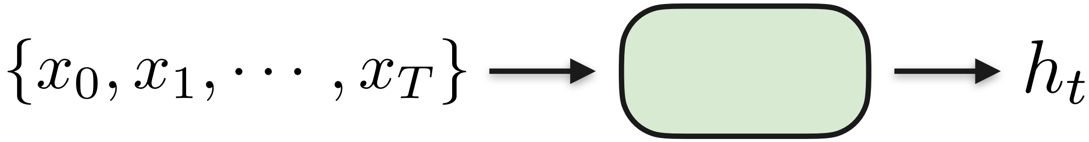
Memory-less models: fixed context window

$\{x_0, x_1, \cdots, x_T\}$ Correlated inputs with a time dependence



output conditioned on a **fixed window** of k past elements

Models with memory

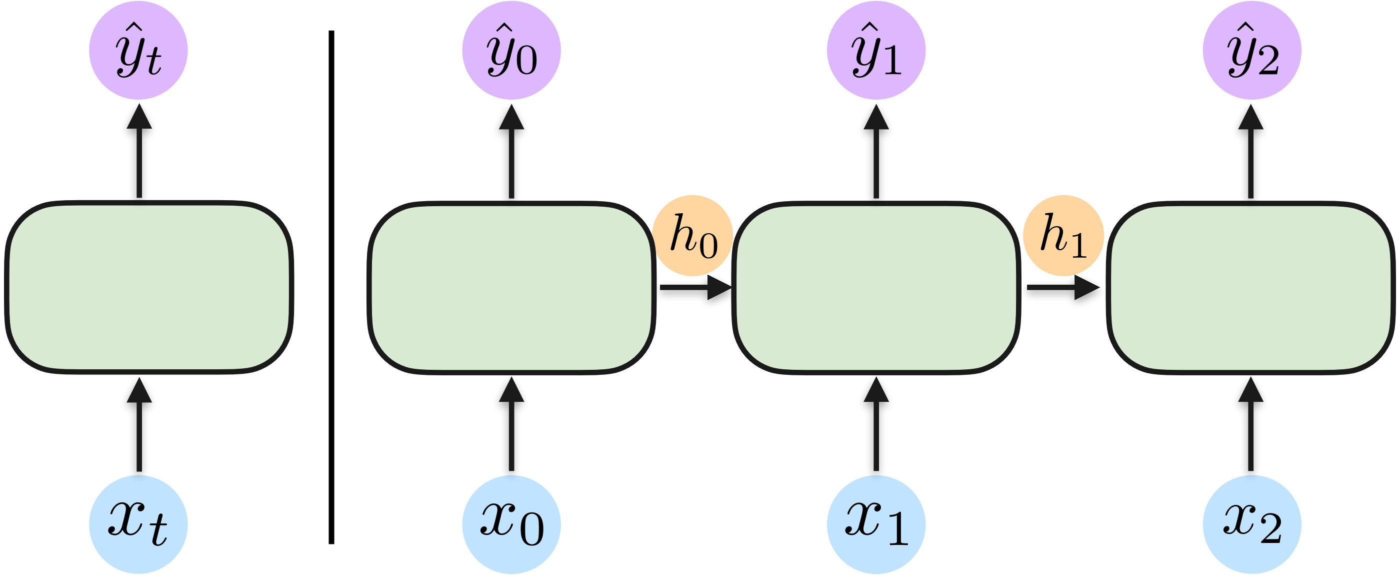


Learn a function that **summarises the context** and creates a persistent state (**memory**)

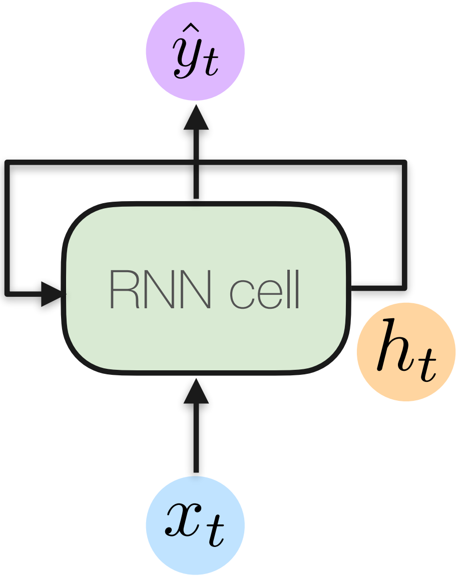
$$p(x_t | x_1, \ldots, x_{t-1}) \approx p(x_t | h_t)$$

Models with memory

$\{x_0, x_1, \cdots, x_T\}$ Correlated inputs with a time dependence



Recurrent Neural Networks (RNNs)



Apply **recurrent relation** at every time step to process a sequence:

$$h_t = f(x_t, h_{t-1})$$

The **same function** and set of parameters are used at every time step

General formulation!
The **recurrent cell** can be anything (differentiable)

RNNs in a nutshell

What are they?

- Very powerful sequence models (in theory)
- Non-Markovian (Infinite-Context) Sequence Modelling
- Directly model original conditional probabilities

$$p(x_1, x_2, \cdots, x_T) = \prod_{t=1}^{T} p(x_t | x_1, \cdots, x_{t-1})$$

Expressivity (Representational Capacity)

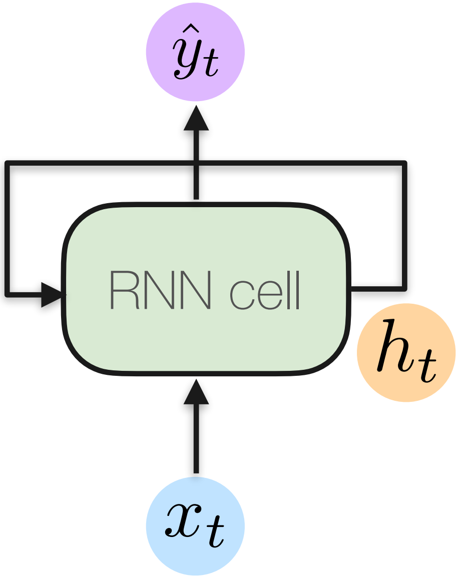
- FFNN are universal function approximators

  [Hornik, 1989; Cybenko,1989]
- RNN are universal dynamical system approximators

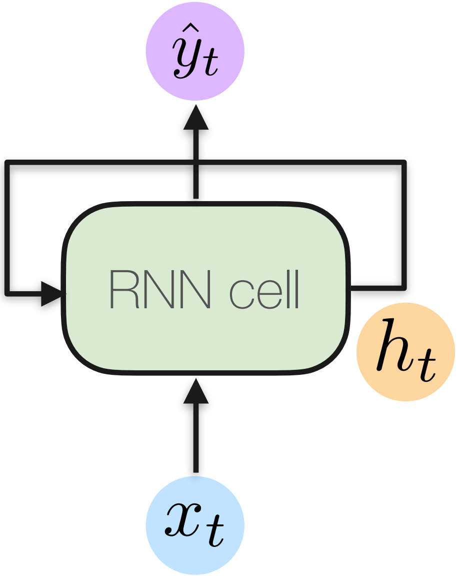
  [Schäfer, 2006]
- Turing complete

  Compute anything a Turing machine can compute (learn to program)

  [Siegelmann, 1991; Graves, 2014]
- Strong connection with Finite State Machines [Horne, 1998]



Vanilla RNN



Output vector

$$\hat{y}_t = W_y^\top h_t$$

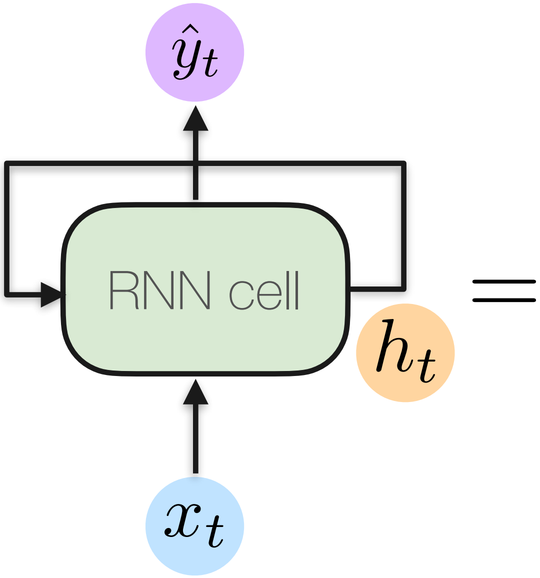
Hidden State update

$$h_t = \tanh(W_h^\top h_{t-1} + W_x^\top x_t)$$

Input vector

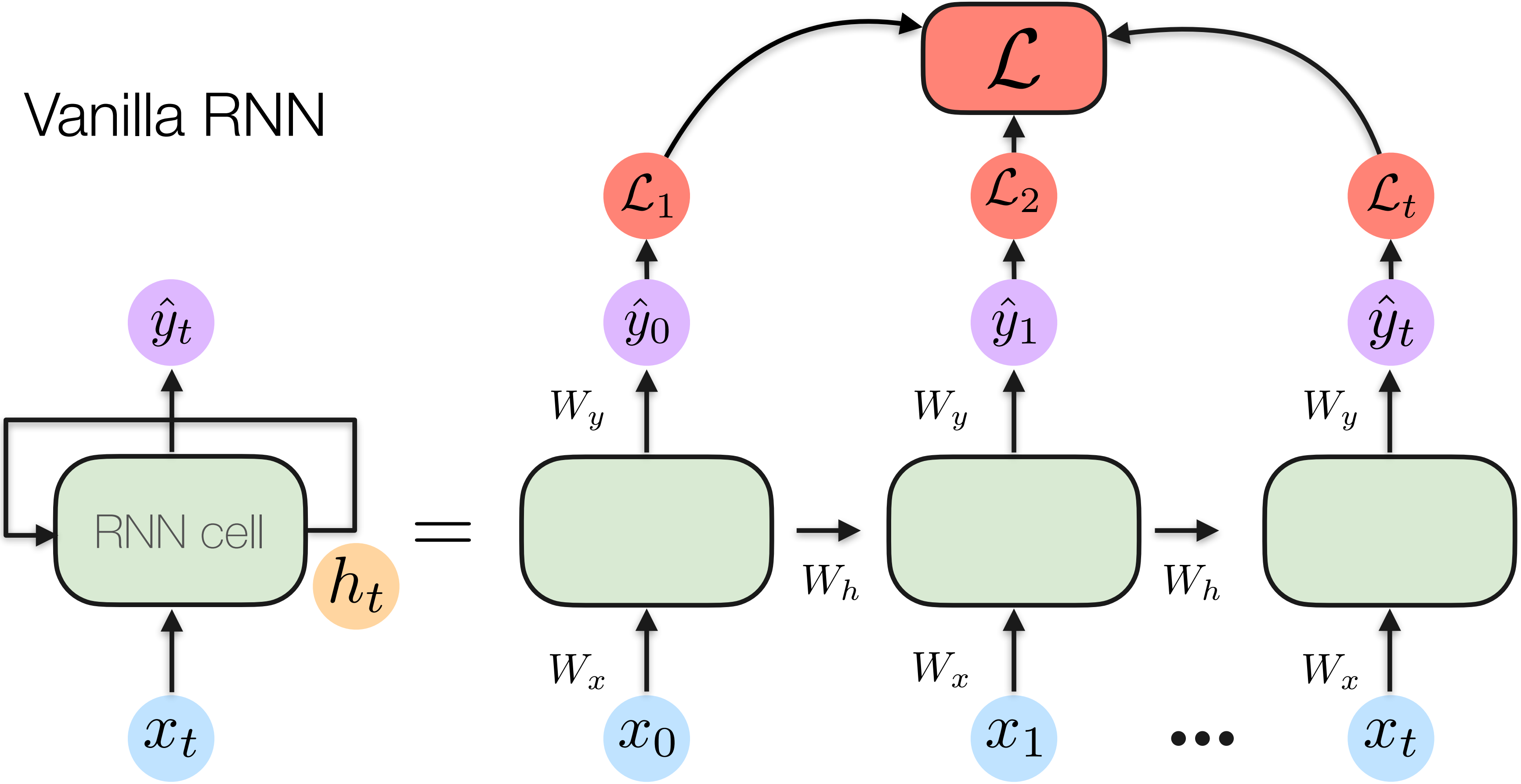
$$x_t$$

Vanilla RNN



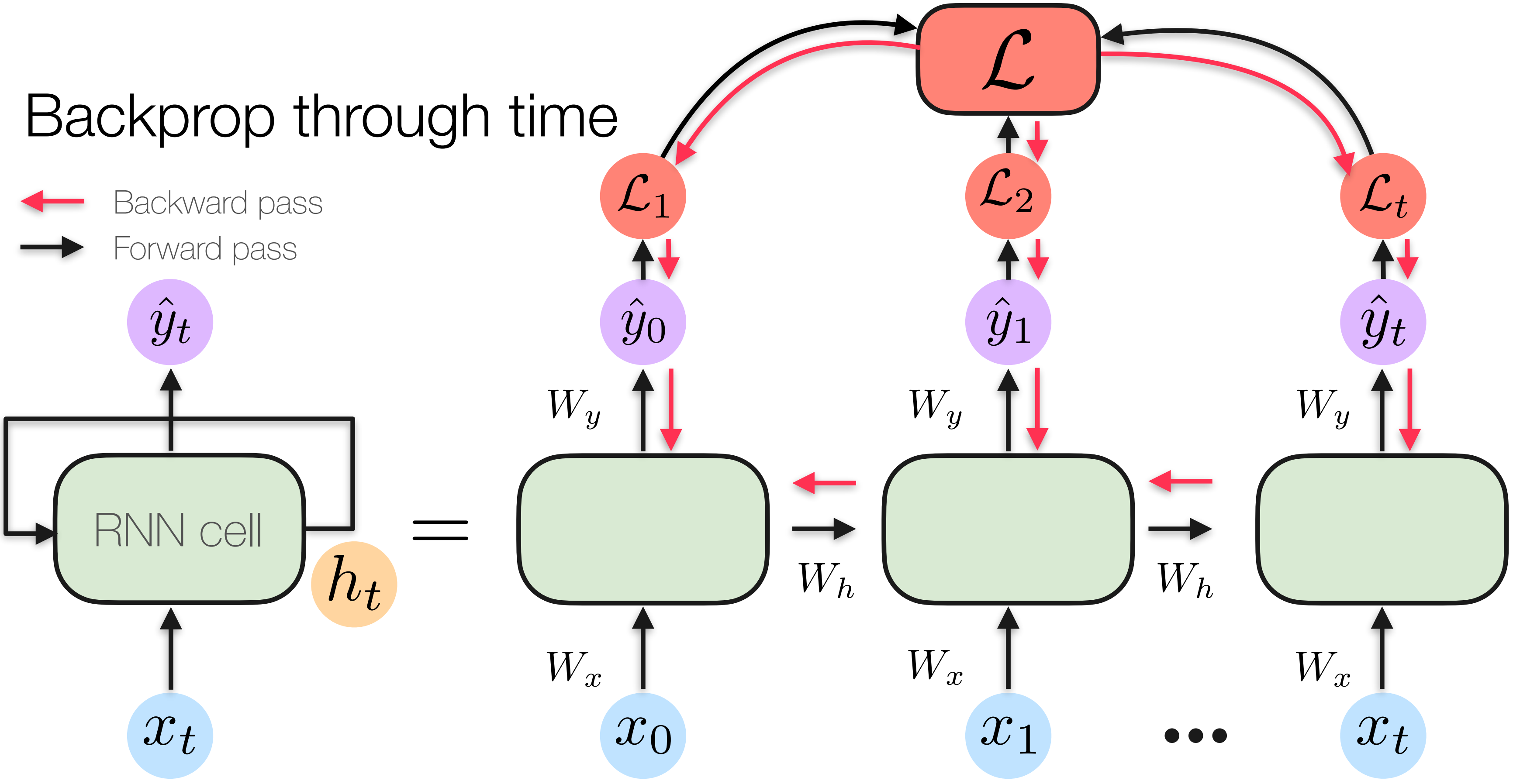
RNNs as computational graph unrolled across time

Vanilla RNN



Re-use the **same weight matrices** at every time-step

Backprop through time



Re-use the **same weight matrices** at every time-step

Expressivity & Trainability II

$$\text{quality} = f(\mathcal{D}, \theta, \lambda)$$

What functions your model can approximate

How good can you train your model (optimization)

$\mathcal{D}$ = Data

θ = Model

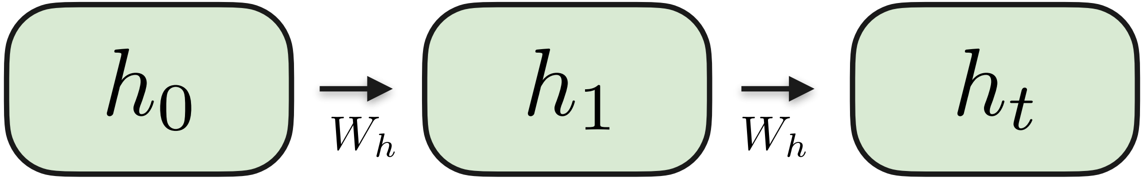
λ = Loss, optimizer, hyperparams

* great blog https://blog.evjang.com/2017/11/exp-train-gen.html

Slides inspired by Orhan Firat [M2L Lecture]()

Vanilla RNN gradient flow

A simple example



$$h_t = W_h h_{t-1} = (W_h)^t h_0$$

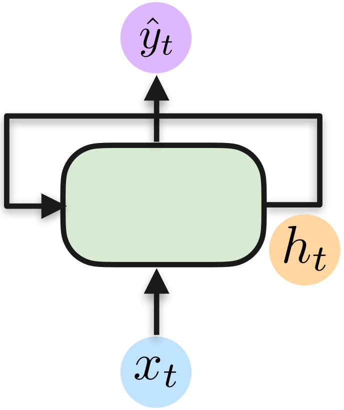
$$h_t \to \infty \text{ if } ||\mathbf{W}_h|| > 1$$ Exploding Gradients

$$h_t \to 0 \text{ if } ||\mathbf{W}_h|| < 1$$ Vanishing Gradients

Credits DeepMind x UCL Lectures - Marta Garnelo

Vanilla RNN gradient flow

$$h_t = \tanh(W_h^\top h_{t-1} + W_x^\top x_t)$$



$$\frac{\partial \mathcal{L}_{\theta,t}}{\partial \mathbf{W}_h} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{\mathbf{y}}_t}\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_h^\top \mathrm{diag}[f'(h_{i-1})] = \mathbf{W}_h^{(t-k)^\top} \prod_{i=k+1}^{t} \mathrm{diag}[f'(h_{i-1})]$$

Bounding the norms of these terms using the power iteration method:

$$\left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \left\| \mathbf{W}_h^\top \right\| \left\| \mathrm{diag}\left(f'\left(\mathbf{h}_{i-1}\right)\right) \right\| \leq \gamma \sigma_{\max}$$

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| \leq \prod_{i=k+1}^{t} \left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \left\| \gamma \sigma_{\max} \right\|^{(t-k)}$$

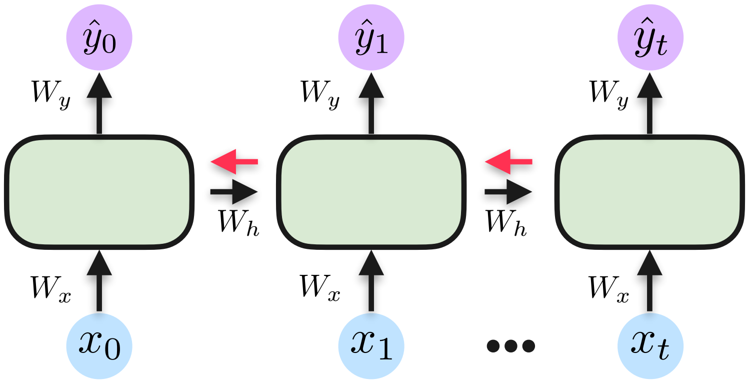
Largest singular value

Bound on nonlinearity

$\gamma \sigma_{\max} > 1$ Exploding grads

$\gamma \sigma_{\max} < 1$ Vanishing grads

Vanilla RNN gradient flow



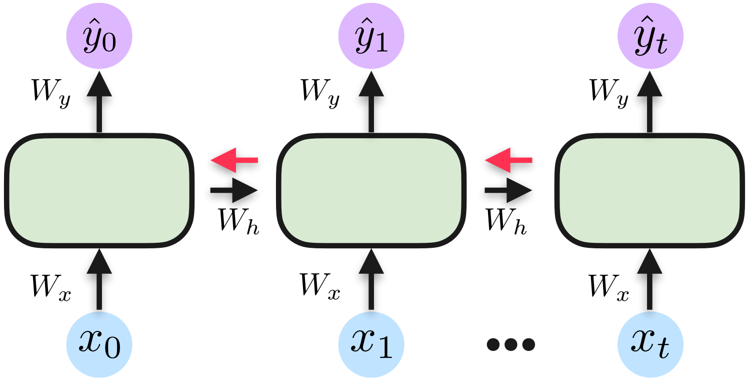
Exploding Gradients

Gradient clipping
to scale large gradients

Vanishing Gradients

1. Activation functions
2. Weight initialization
3. **Network architecture**

The problem of Long-Term Dependencies



RNNs can model **sequences of variable length** and can be trained via back-propagation

However, the vanishing **gradients problem** **stops them from capturing long-term dependencies.**

Long-Term Dependencies are important

… Finally, Tim was planning to visit France on the final week of his journey. He was quite excited to try the local delicacies and had lots of recommendations for good restaurants and exhibitions. His first stop was, of course, the capital where he would meet his long-time Friend Jean-Pierre. In order to arrive for breakfast he took the early 5 AM train from London to …

PARIS!

Credits DeepMind x UCL Lectures - Marta Garnelo

How can we model long-term dependencies?

Long-Short Term Memory (LSTM)

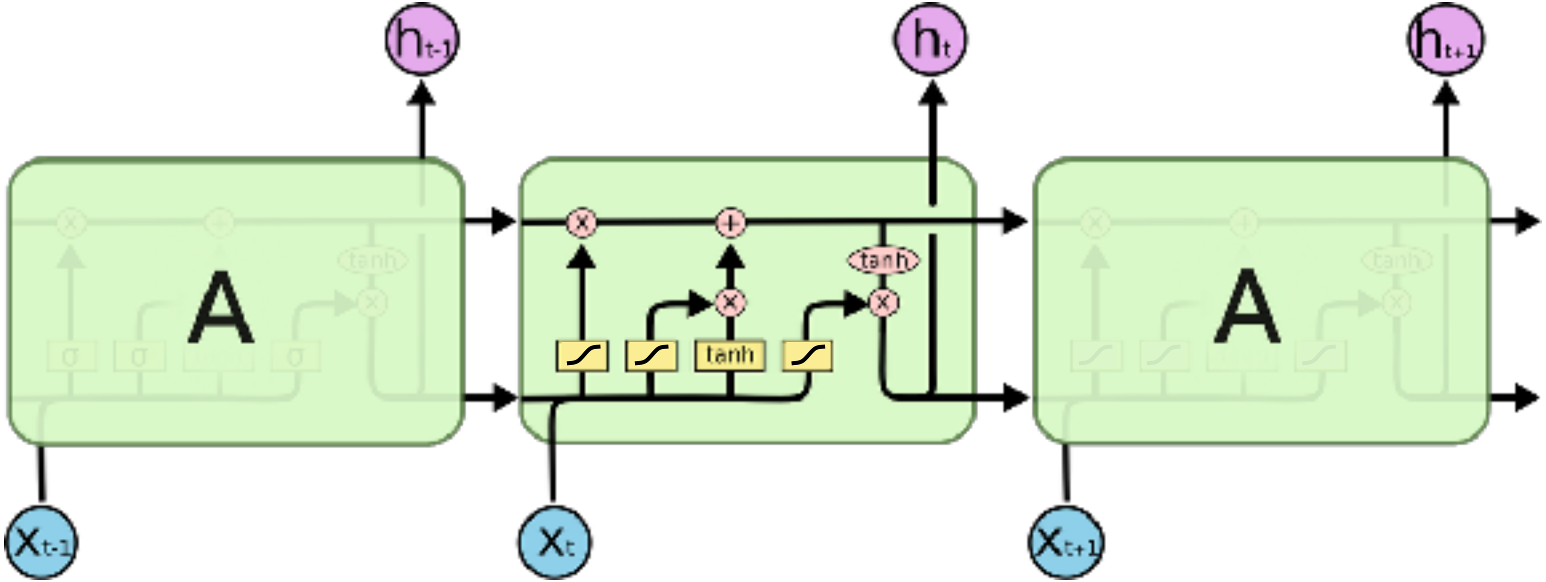
▭ Affine transformation and nonlinearity

○ Elementwise Operation

→ Vector Transfer

≻→ Concatenation

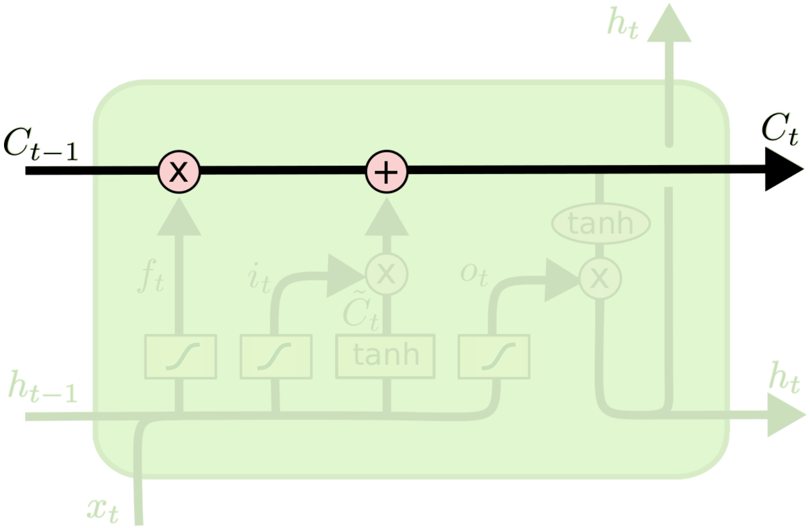
–< Copy



The LSTM cell contains four interacting modules

From https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Cell Memory



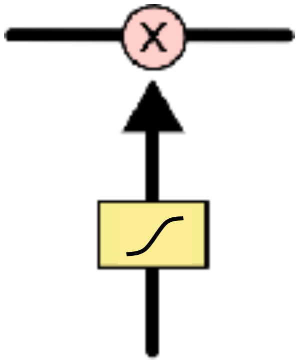
The core idea of LSTM is to have a **cell memory** with only some minor linear interactions.

It's very easy for information to just flow along it unchanged.

Gates: regulating information flow

The LSTM cell has the ability to remove or add information to the cell state, regulated by modules called **gates**.

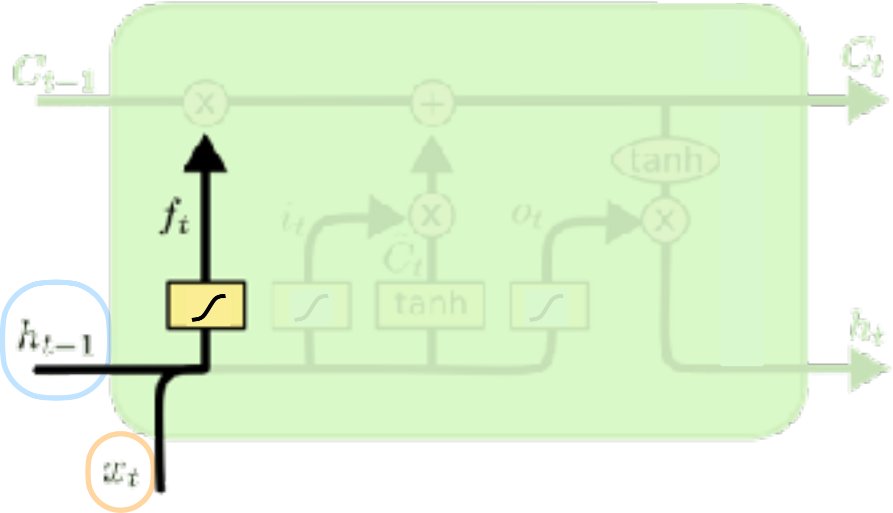
sigmoid + affine transformation (matrix multiplication)



A value of zero means "let nothing through," while a value of one means "let everything through!"

Gates are a way to optionally let information through conditioned on the current input and the past state.

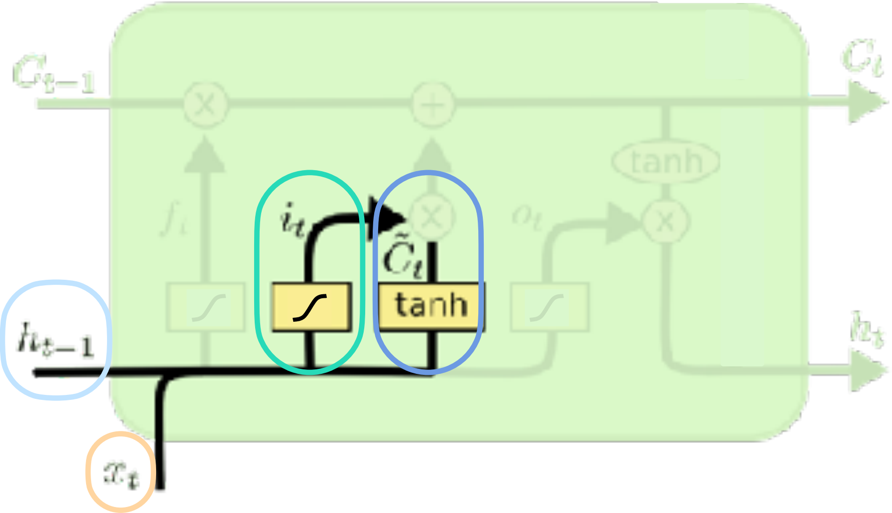
Forget Gate



$$f_t = \sigma(W_f^\top [h_{t-1}, x_t] + b_f)$$

Decide what information to throw away from the cell state

Input Gate

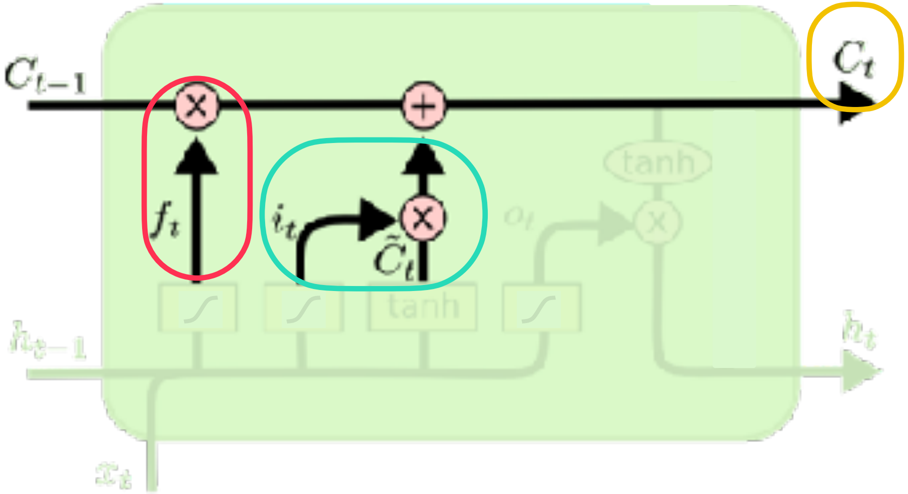


$$i_t = \sigma(W_i^\top [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c^\top [h_{t-1}, x_t] + b_c)$$

Decide what new information to store in the cell state

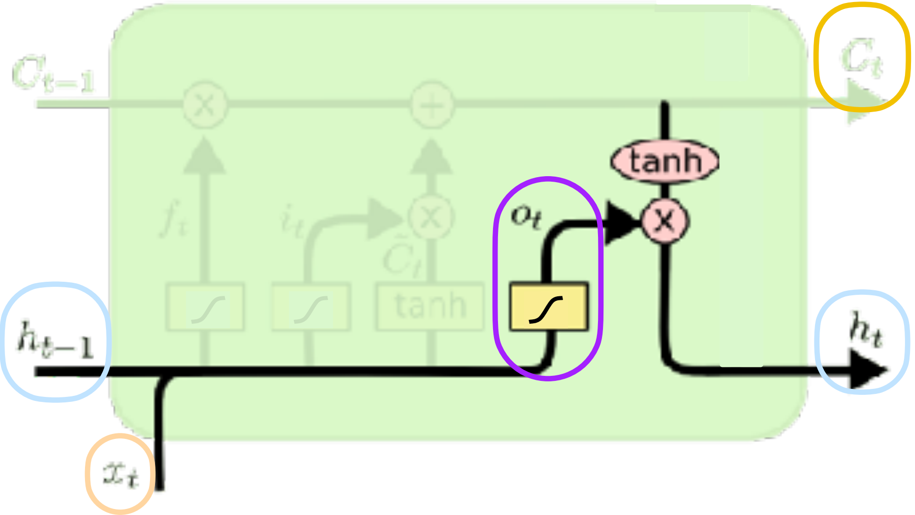
Memory cell update



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Update cell state removing old information and storing new one

Output Gate



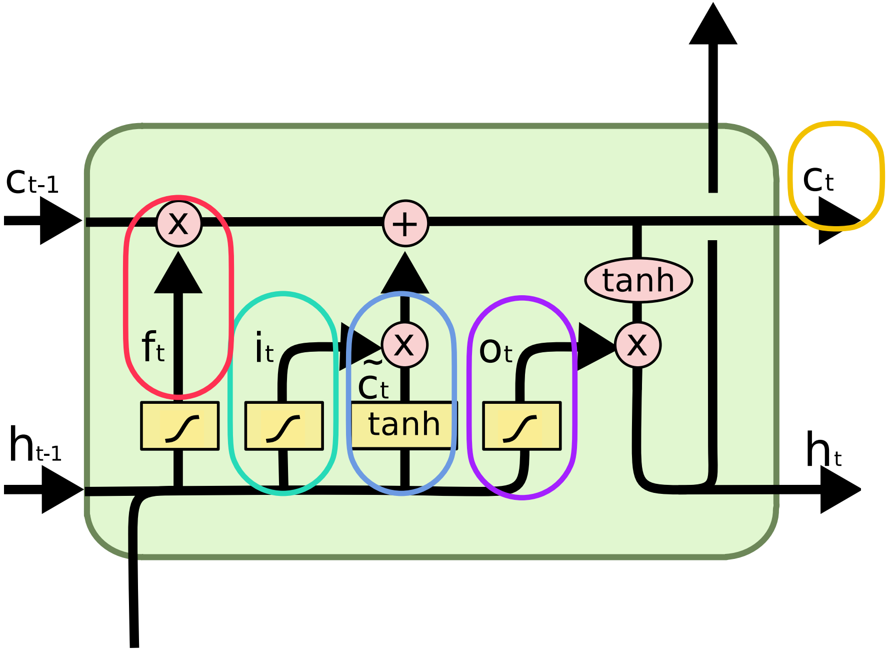
$$o_t = \sigma(W_o^\top [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Decide what to output

LSTM cell



$$f_t = \sigma(W_f^\top [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t] + b_o)$$

$$\tilde{C}_t = \tanh(W_c^\top [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

LSTM can be seen as differentiable hardware register with “**load**”, “**save**” and “**reset**” operations.

More details on https://d-nb.info/1082034037/34

How LSTM solves vanishing gradient?

Within the LSTM cell, the **gradient of the memory with respect to its immediate predecessor** is the only module where gradients flow through time

$$\begin{aligned}\frac{\partial c_t}{\partial c_{t-1}} &= \frac{\partial \phi\left(f_t, c_{t-1}\right)}{\partial c_{t-1}} \\ &= \frac{\partial \sigma\left(f_t\right) c_{t-1}}{\partial c_{t-1}} \\ &= c_{t-1} \underbrace{\frac{\partial \sigma\left(f_t\right)}{\partial c_{t-1}}}_{=0} + \underbrace{\frac{\partial c_{t-1}}{\partial c_{t-1}}}_{=1} \sigma\left(f_t\right) \\ &= \sigma\left(f_t\right)\end{aligned}$$

LSTM $$\frac{\partial c_{t'}}{\partial c_t} = \prod_{k=1}^{t'-t} \sigma\left(f_{t+k}\right)$$

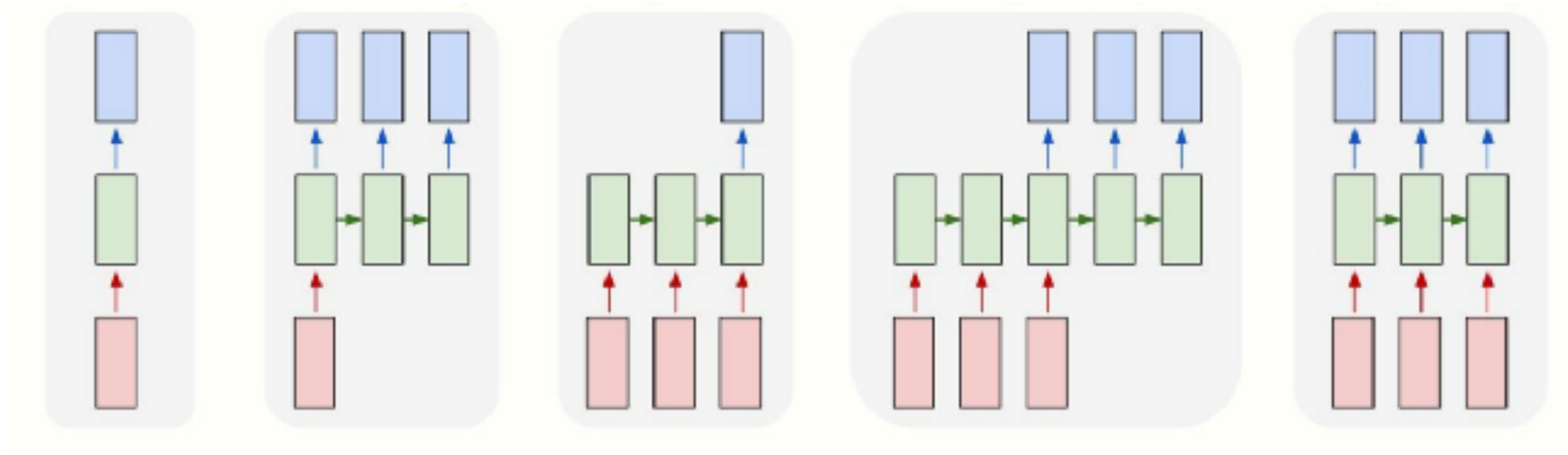
- No exponential decay for very large time lags t' >> t
- Safe from exploding, bounded by 1 thanks to sigmoid

RNN $$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \mathbf{W}_h^{(t-k)^\top} \prod_{i=k+1}^{t} \operatorname{diag}[f'(h_{i-1})]$$

(Copied from past slide, sorry for the indexes)

N.B. The whole gradient might still explode due to the other parts of the total derivative, but **we can now learn long-term dependencies!**

Sequential Data Problems



One to one

Fixed-sized input to fixed-sized output (e.g. image classification)

One to many

Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

Many to one

Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

Many to many

Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

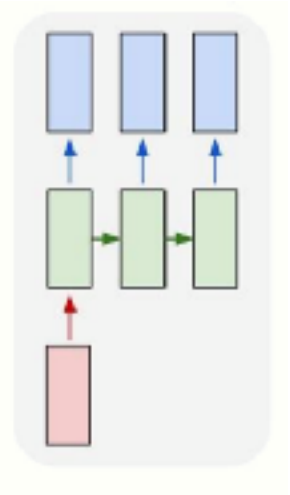
Many to many

Synced sequence input and output (e.g. video classification where we wish to label each frame of the video)

Sequence Learning examples

Image Captioning: input a single image and get a series or sequence of words as output which describe it. The image has a fixed size, but the output has varying length.

One to many



A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

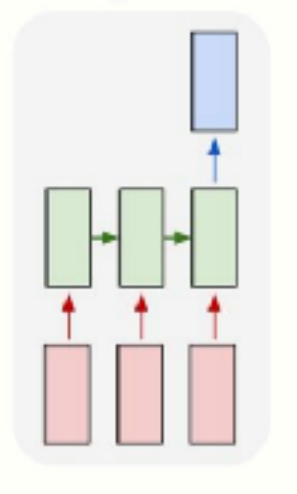
A group of young people playing a game of frisbee.

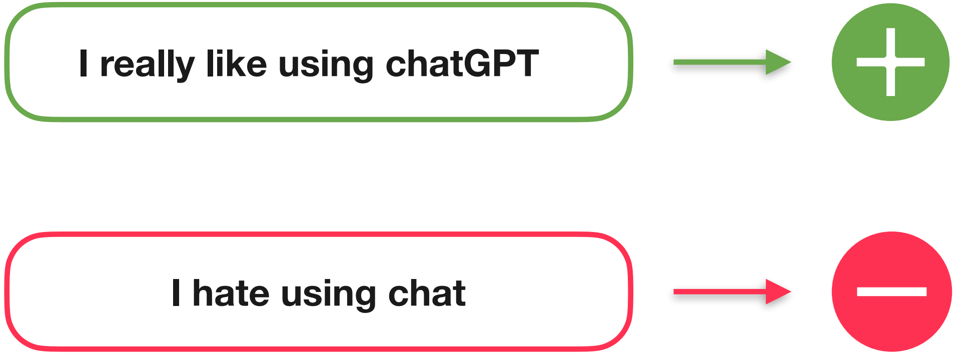
Two hockey players are fighting over the puck.

Sequence Learning examples

Sentiment Analysis: input a sequence of characters or words, e.g., a tweet, and classify the sequence into positive or negative sentiment. Input has varying lengths; output is of a fixed type and size

Many to one





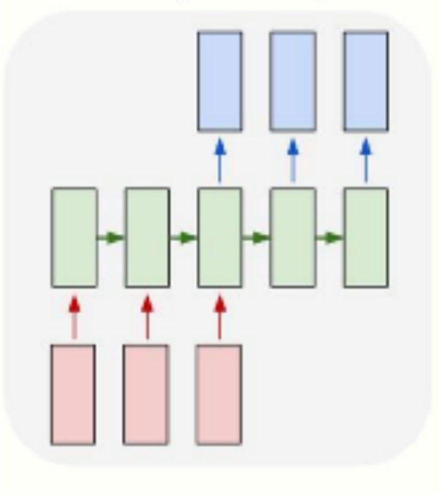
I really like using chatGPT → ＋

I hate using chat → −

Sequence Learning examples

Language Translation: having some text in a particular language, e.g., English, we wish to translate it in another, e.g., Italian. Each language has it's own semantics and it has varying lengths for the same sentence.

Many to many



ChatGPT is a cutting-edge language model based on the GPT-3.5 architecture and trained by OpenAI. It is designed to generate high-quality, human-like responses to natural language inputs, making it a valuable tool for a variety of applications, including chatbots, virtual assistants, and text generation. With its advanced capabilities in language understanding and generation, ChatGPT is helping to revolutionize the way we communicate and interact with technology.

↓

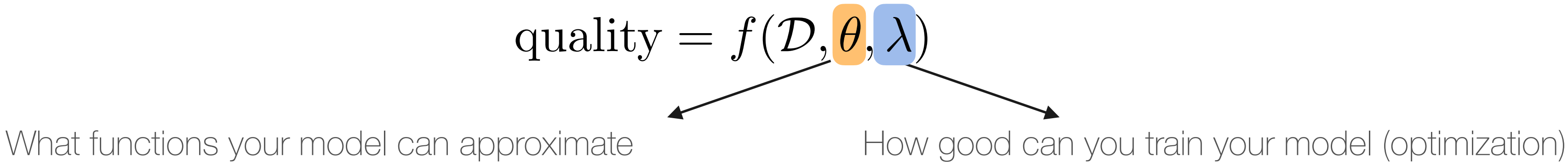
ChatGPT è un modello linguistico all'avanguardia basato sull'architettura GPT-3.5 e addestrato da OpenAI. È progettato per generare risposte simili a quelle umane di alta qualità agli input del linguaggio naturale, rendendolo uno strumento prezioso per una varietà di applicazioni, inclusi chatbot, assistenti virtuali e generazione di testo. Con le sue capacità avanzate nella comprensione e generazione del linguaggio, ChatGPT sta contribuendo a rivoluzionare il modo in cui comunichiamo e interagiamo con la tecnologia.

So far…

- **CNNs** are powerful models for image/video representation learning
  - Deeper and larger models need some **tricks (skip connections & normalisation)** to be trained because of **vanishing gradients** and **instabilities**

- **RNNs** can be used to model sequence data
  - They suffer even more of vanishing gradient

Expressivity & Trainability III



$$\text{quality} = f(\mathcal{D}, \theta, \lambda)$$

What functions your model can approximate

How good can you train your model (optimization)

$\mathcal{D}$ = Data

θ = Model

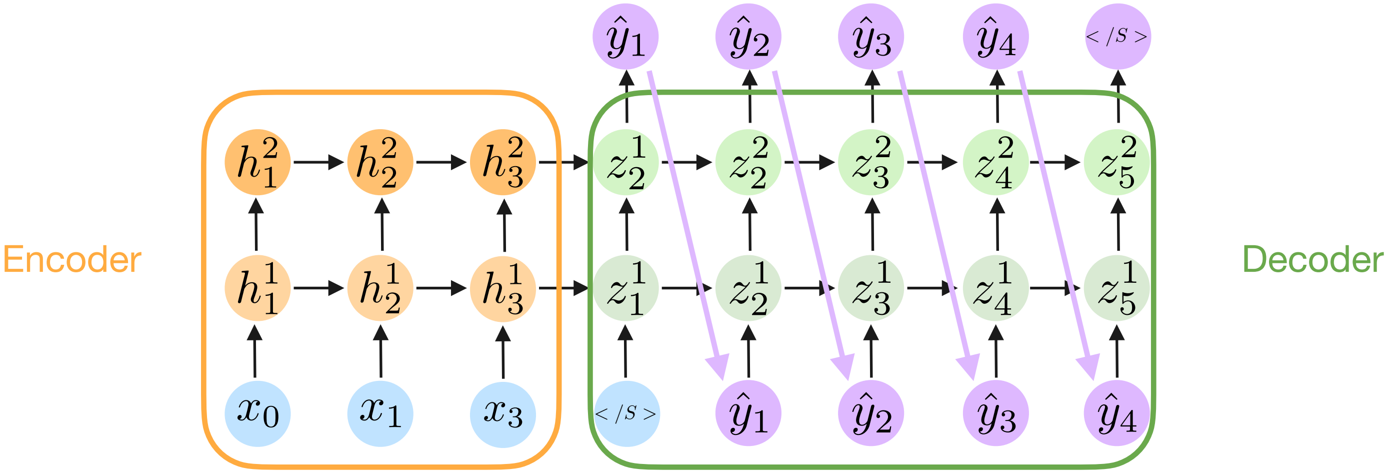
λ = Loss, optimizer, hyperparams

* great blog https://blog.evjang.com/2017/11/exp-train-gen.html

Slides inspired by Orhan Firat M2L Lecture

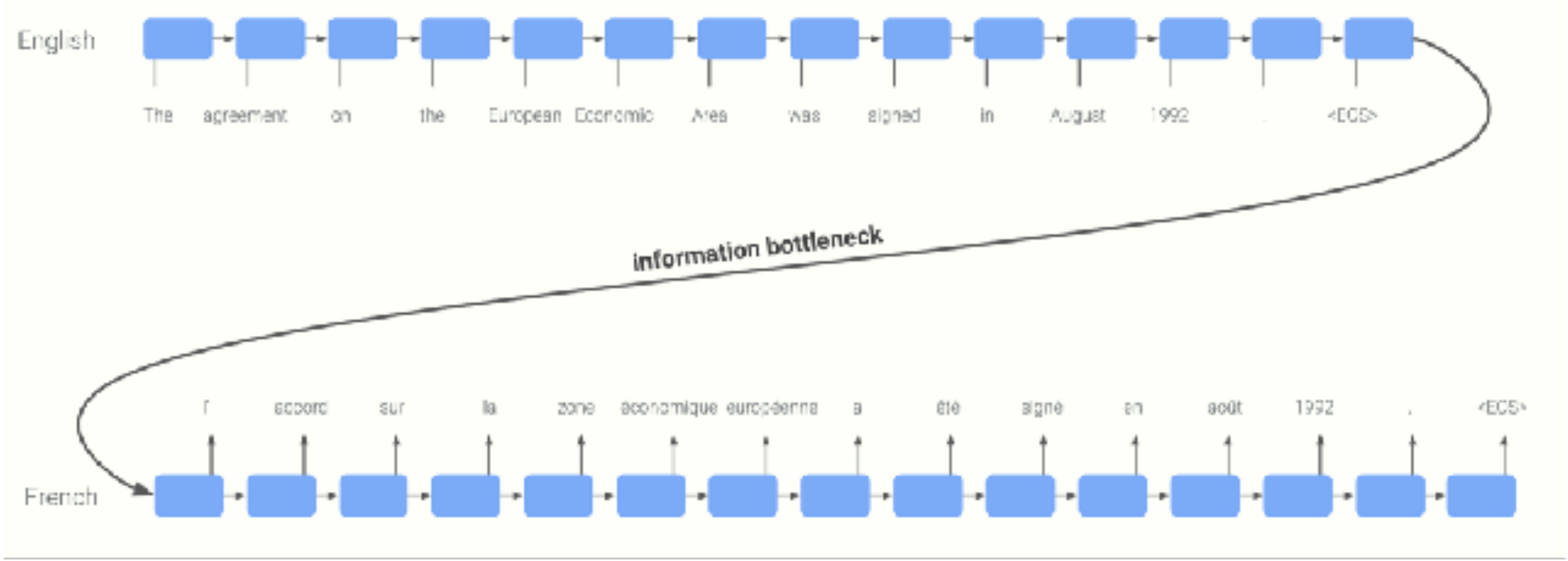
Sequence to Sequence models (Sutsekerver et al 2014, Cho et al 2014)

Generate the output sequence, one token at the time (autoregressively)



Seq2Seq models learn the conditional probability p(y | x)

The encoder-decoder bottleneck



Slide credit: Ilharco et al. High Performance NLP, EMNLP'20

Example derived from Bahdanau, et al. 2014

Seq2Seq with attention (Bahdanau et al. 2015)

Encoder

Attention

Decoder

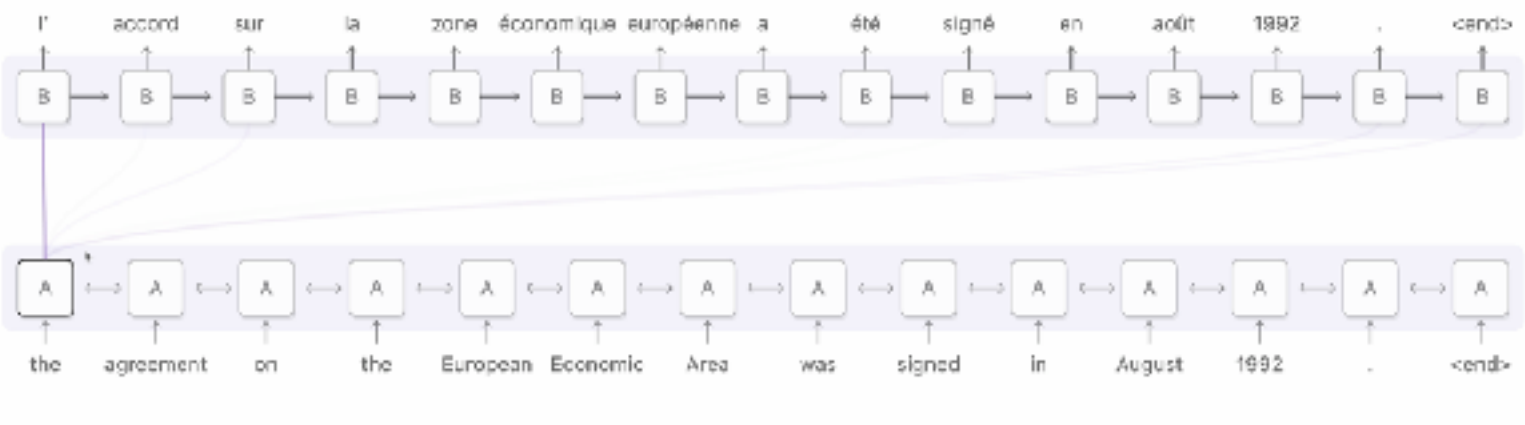

l' accord sur la zone économique européenne a été signé en août 1992 . <end>
B B B B B B B B B B B B B B B
A A A A A A A A A A A A A A A
the agreement on the European Economic Area was signed in August 1992 . <end>

Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

Attention gives **weights** [importance / similarity / relevance]
of the source states/tokens **for** the target state/token

Solves the information bottleneck problem!

https://distill.pub/2016/augmented-rnns/

Seq2Seq with attention (Bahdanau et al. 2015)

Encoder

Attention

Decoder

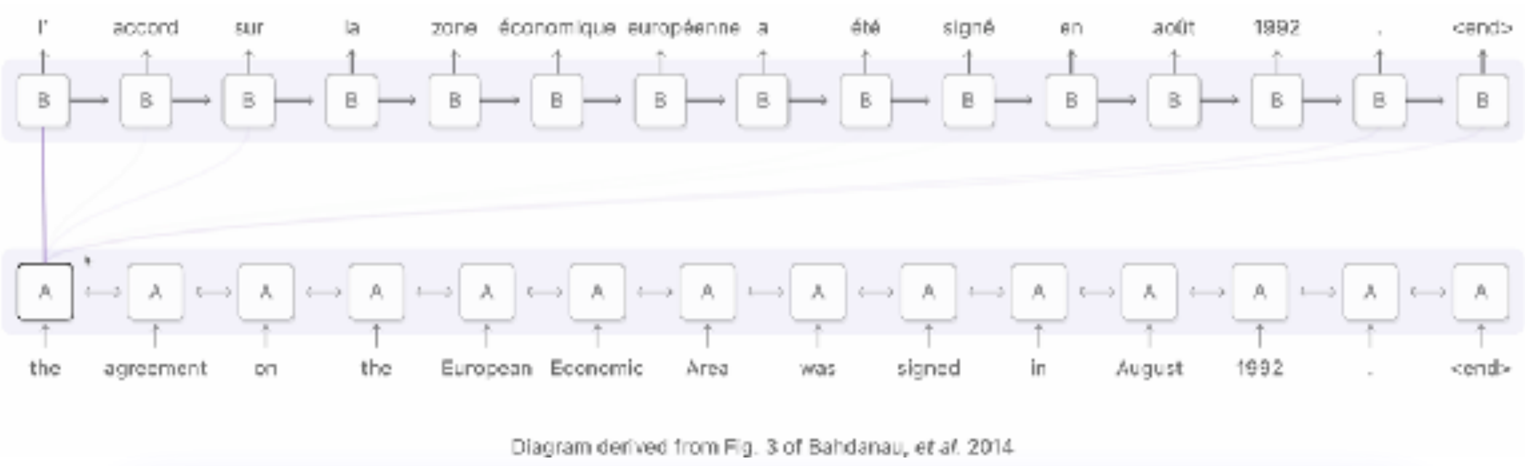
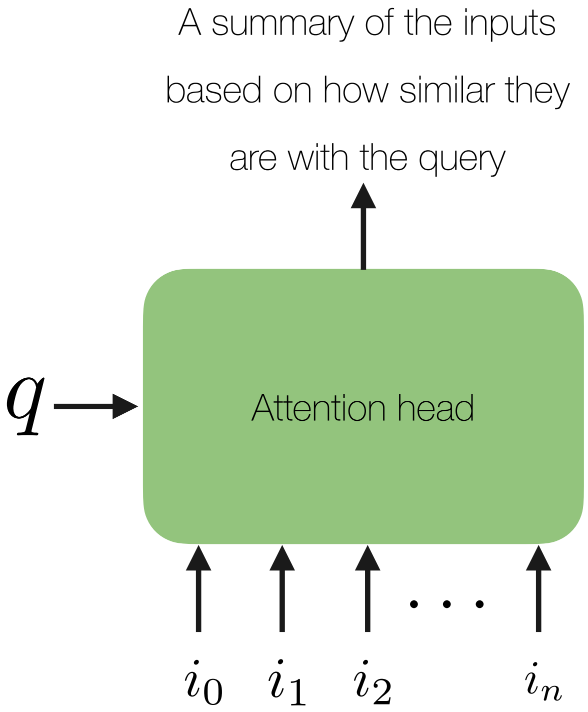


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

The problem is reduced to an **alignment problem**,
where we aim at **(soft) aligning two sets of vectors**
every state in the target sequence with every state in the source sequence

https://distill.pub/2016/augmented-rnns/

Attention module (simplified)

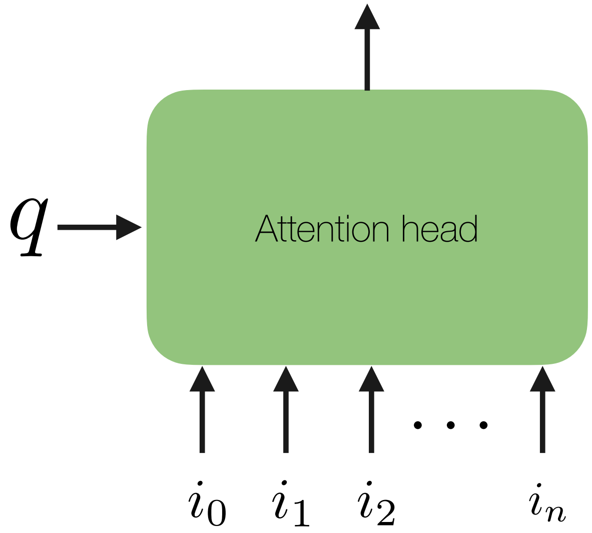

A summary of the inputs based on how similar they are with the query
q
Attention head
i_0
i_1
i_2
$\ldots$
i_n

Slide credit: Ilharco et al. High Performance NLP, EMNLP'20

Attention module (simplified)

Weighted combination of the inputs

$$\sum_{k=0}^{n} a_k i_k$$

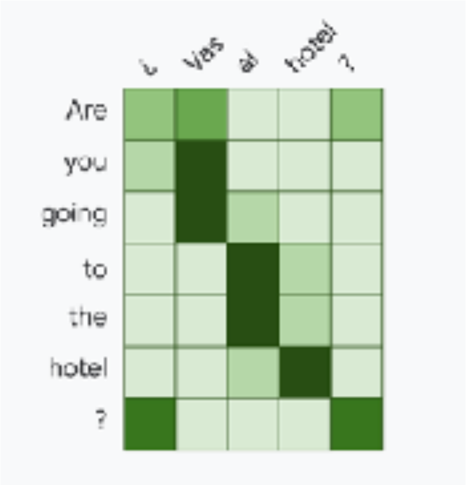


Softmax normalisation

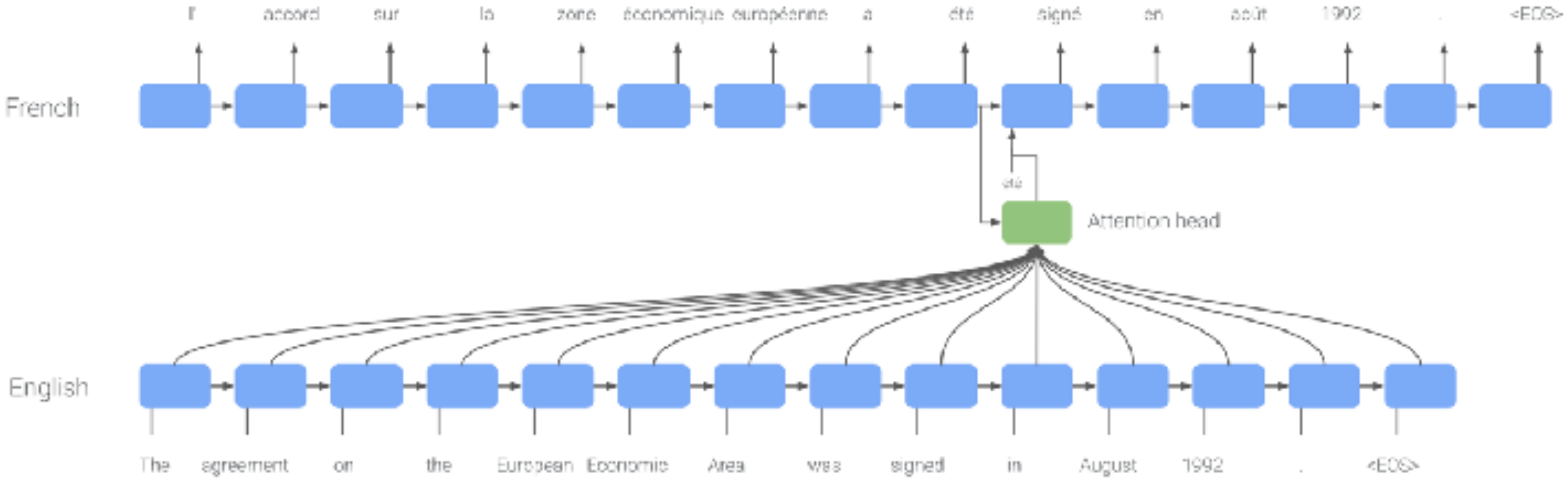
$$\phi\left(q, i_k\right) = \exp\left(q^\top i_k\right)$$

$$a_k = \frac{\phi\left(q, i_k\right)}{\sum_{j=0}^{n} \phi\left(q, i_j\right)}$$

Attention matrix



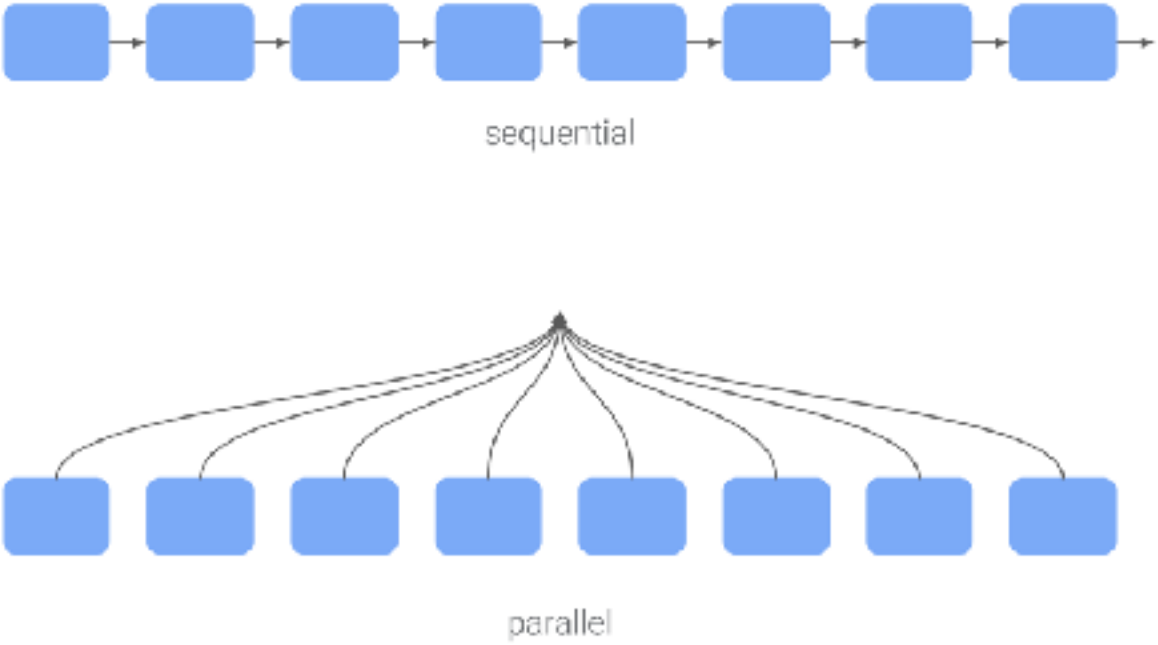
Slide credit: Ilharco et al. High Performance NLP, EMNLP'20



Example derived from Bahdanau, et al. 2014 (https://arxiv.org/pdf/1409.0473.pdf)

Slide credit: Ilharco et al. High Performance NLP, EMNLP'20

Attention is all you need?

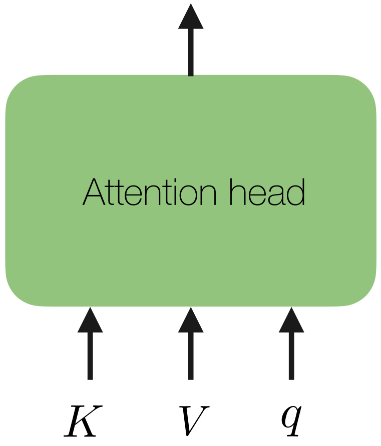


Slide credit: Ilharco et al. High Performance NLP, EMNLP'20

Scaled Dot-Product Attention

A summary of **values**
based on how similar their
corresponding **keys** are
with the **query**

(jargon used in transformers)



Attention head

K V q

Slide credit: Ilharco et al. [High Performance NLP, EMNLP'20](#)

Scaled Dot-Product Attention

Attention is a function similar to a **soft (differentiable) key-value dictionary lookup**

1. Attention weights are query-key similarities, normalized by a softmax

$$\hat{\mathbf{a}}_i = \mathbf{q} \cdot \mathbf{k}_i \qquad \mathbf{a}_i = \frac{e^{\hat{\mathbf{a}}_i}}{\sum_j e^{\hat{\mathbf{a}}_i}}$$

(Single query point)

2. Output is an attention-weighted average of **values**

$$\mathbf{z} = \sum_i \mathbf{a}_i \mathbf{v}_i = \mathbf{a} \cdot \mathbf{v}$$

3. Usually **k** and **v** are derived from the same **input x**

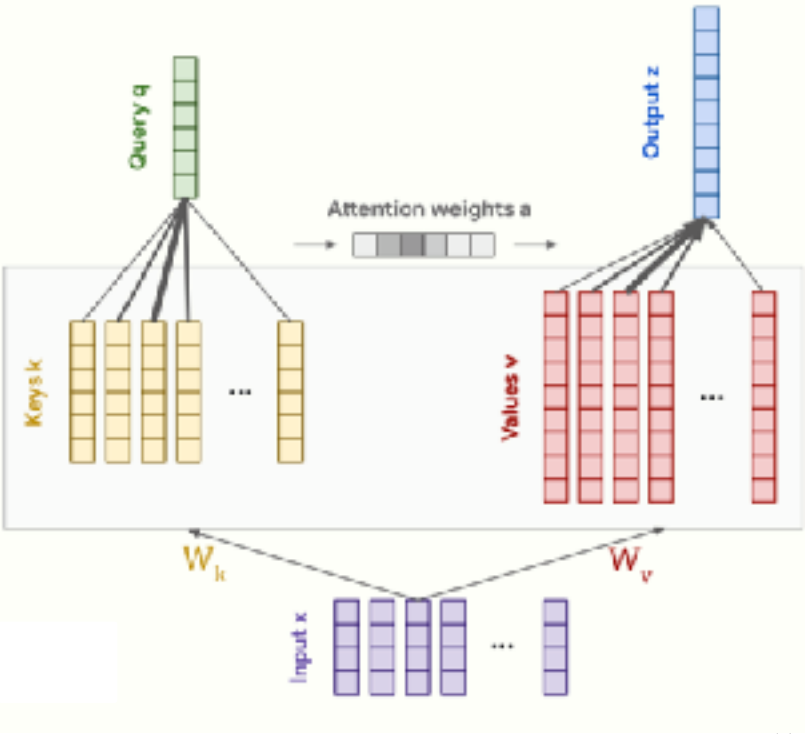
$$\mathbf{v} = \mathbf{W_v} \cdot \mathbf{x} \qquad \mathbf{k} = \mathbf{W_k} \cdot \mathbf{x}$$

The query **q** can come from a **separate input y**

$$\mathbf{q} = \mathbf{W_q} \cdot \mathbf{y}$$

Or from the same input x. Then we call it **"self-attention"**

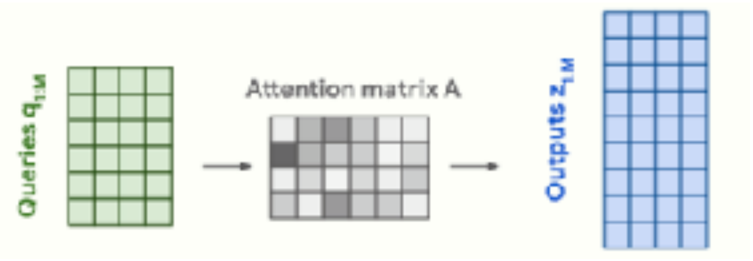
$$\mathbf{q} = \mathbf{W_q} \cdot \mathbf{x}$$



Slide credit Luca Beyer M2L 2022

Vectorization

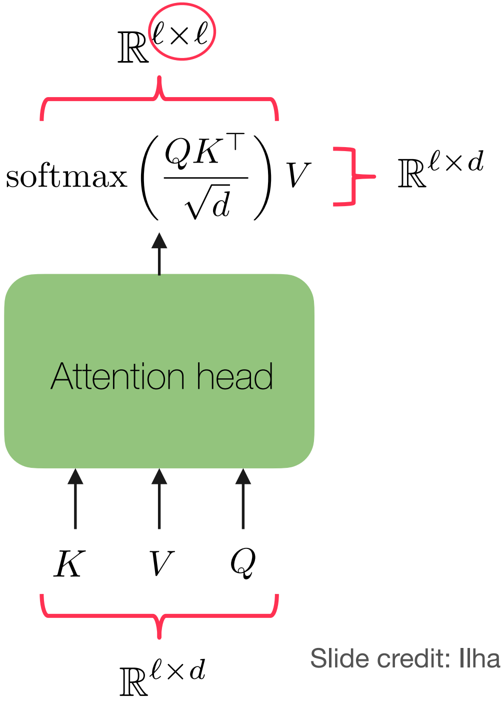
1. We can compute attention for multiple queries in parallel thanks to dot product. Stacking queries lead to Attention Matrix and many outputs



Attention is quadratic in sequence length!

ℓ = Sequence length

d = Feature dim



$$\overbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)}^{\mathbb{R}^{\ell\times\ell}} V \quad \Big\} \ \mathbb{R}^{\ell\times d}$$

$$\underbrace{K \quad V \quad Q}_{\mathbb{R}^{\ell\times d}}$$

Slide credit: Ilharco et al. High Performance NLP, EMNLP’20

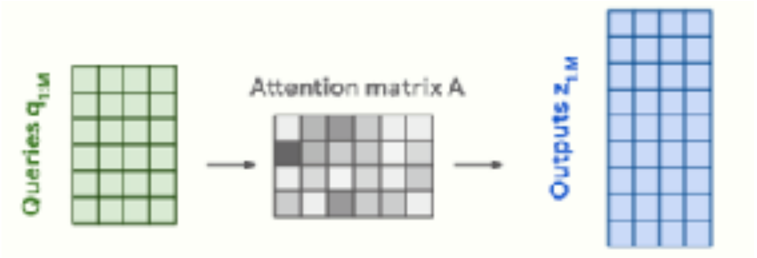
Multi-head attention

2. We usually use “multi-head” attention. This means the operation is repeated **h times** and the results are concatenated along the feature dimension. We use different parameters W.

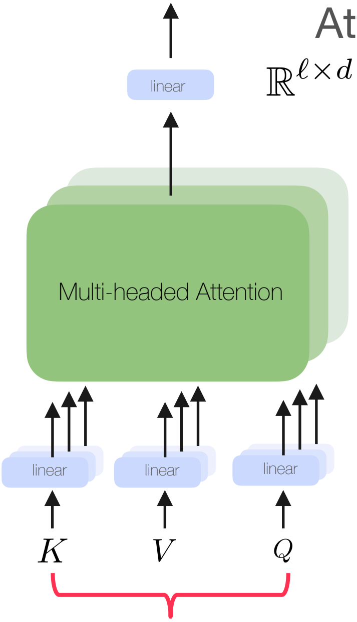
ℓ = Sequence length

d = Feature dim

h = # attention head



Attention is quadratic in sequence length!



$\mathbb{R}^{\ell \times d}$

Each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

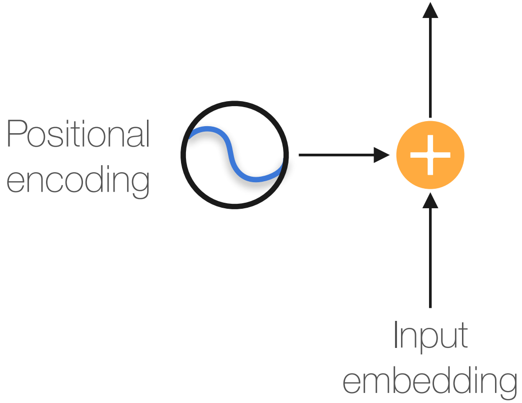
Each head: $\mathbb{R}^{\ell \times \frac{d}{h}}$

$W_i^K, W_i^V, W_i^Q \in \mathbb{R}^{\frac{d}{h}}$

$\mathbb{R}^{\ell \times d}$

Slide credit: Ilharco et al. High Performance NLP, EMNLP’20

Positional encodings

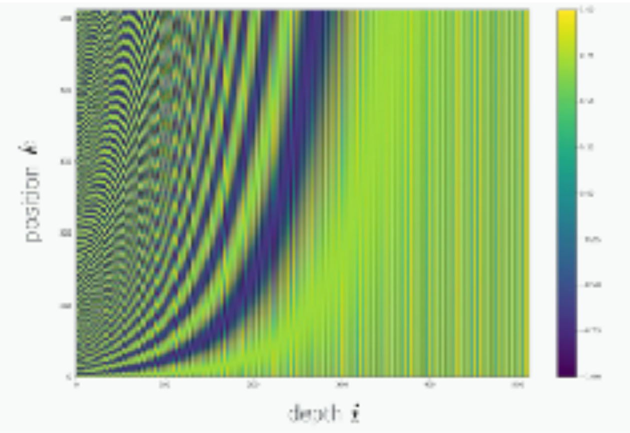


So far, attention has been a set operator, meaning that is **permutation invariant**.

As we want to deal with sequences, we we to add a positional information!

This can be either **fixed** or **learned.**

$$E_{ki} = \begin{cases} \sin\left(k/10000^{\frac{i}{N}}\right) & \text{if } i \text{ is even} \\ \cos\left(k/10000^{\frac{i-1}{N}}\right) & \text{if } i \text{ is odd} \end{cases}$$

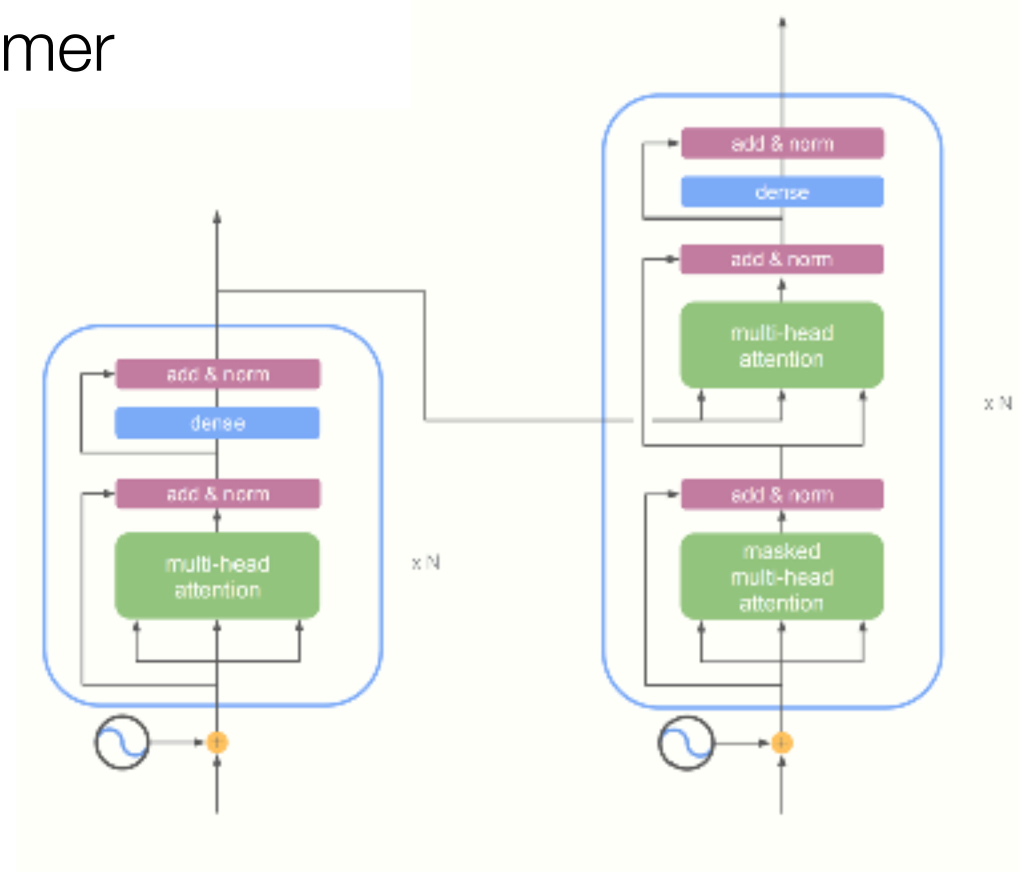


example: fixed for a position **k** in the sequence and **i** in the feature space

Slide credit: Ilharco et al. High Performance NLP, EMNLP'20

The Transformer

Encoder



Decoder

Slide credit: Ilharco et al. [High Performance NLP, EMNLP'20](#)

So far…

- **CNNs** are powerful models for image/video representation learning
  - Deeper and larger models need some **tricks (skip connections & normalisation)** to be trained because of **vanishing gradients** and **instabilities**

- **RNNs** can be used to model sequence data
  - They suffer even more of vanishing gradient

- **Attention** can be used to solve the information bottleneck problem

- **Attention** can be used to replace recurrent computation

One revolution at the time

First revolution ~2010-2017

theano TensorFlow

PyTorch JAX

Hugging Face

1. **Large-scale supervised datasets**

e.g. **ImageNet 1K** (1.2M imgs)

subset of 21K (14M imgs)

2. **GPUs Parallelisation**

larger models & more data

3. **Autodiff** frameworks

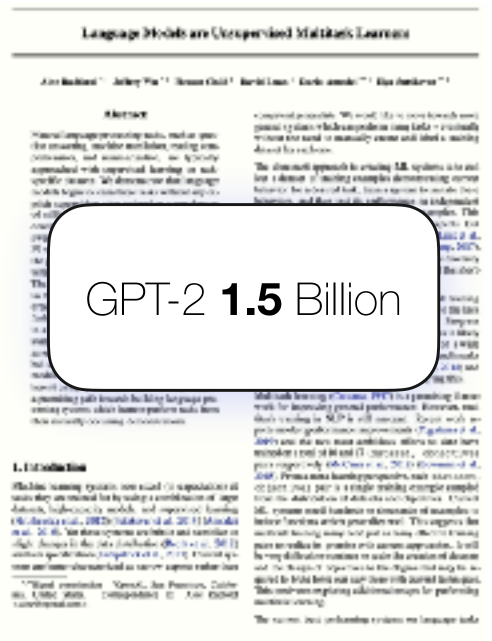
&

communities

Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." 2009

Second revolution ~2017-now : Attention & Scale is all you need

Transformers are scalable and parallelizable!
Let's use **larger models** and **more data**



Language Models are Unsupervised Multitask Learners

GPT-2 **1.5** Billion

Language Models are Few-Shot Learners

GPT-3 **175** Billion

GPT-4 Technical Report

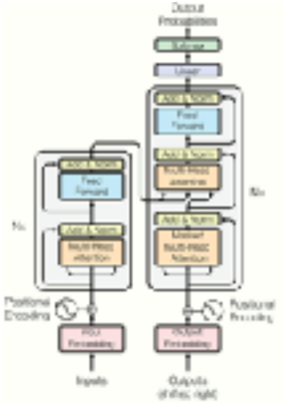
???

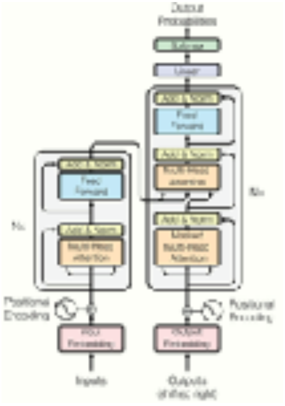
Easter egg :) Look who won the best paper award the same year as GPT-3

Transformers taking over one field at the time

Vision: CNNs (+ ResNet)

NLP: RNNs (+ LSTM)



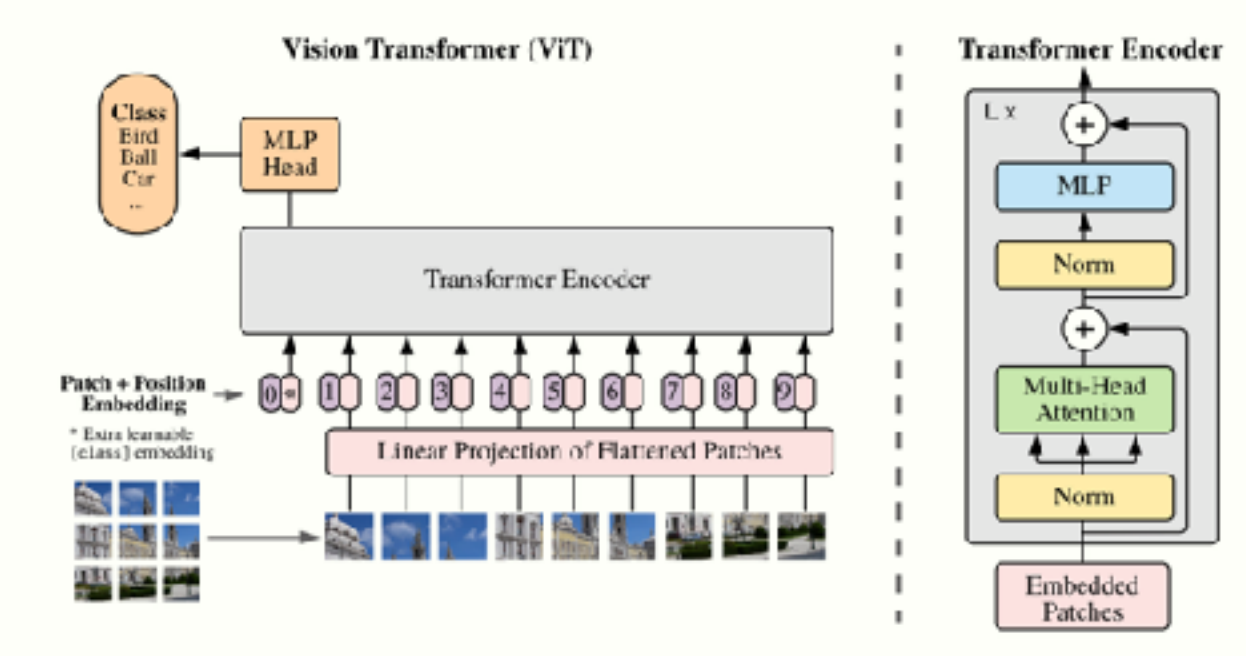


Audio

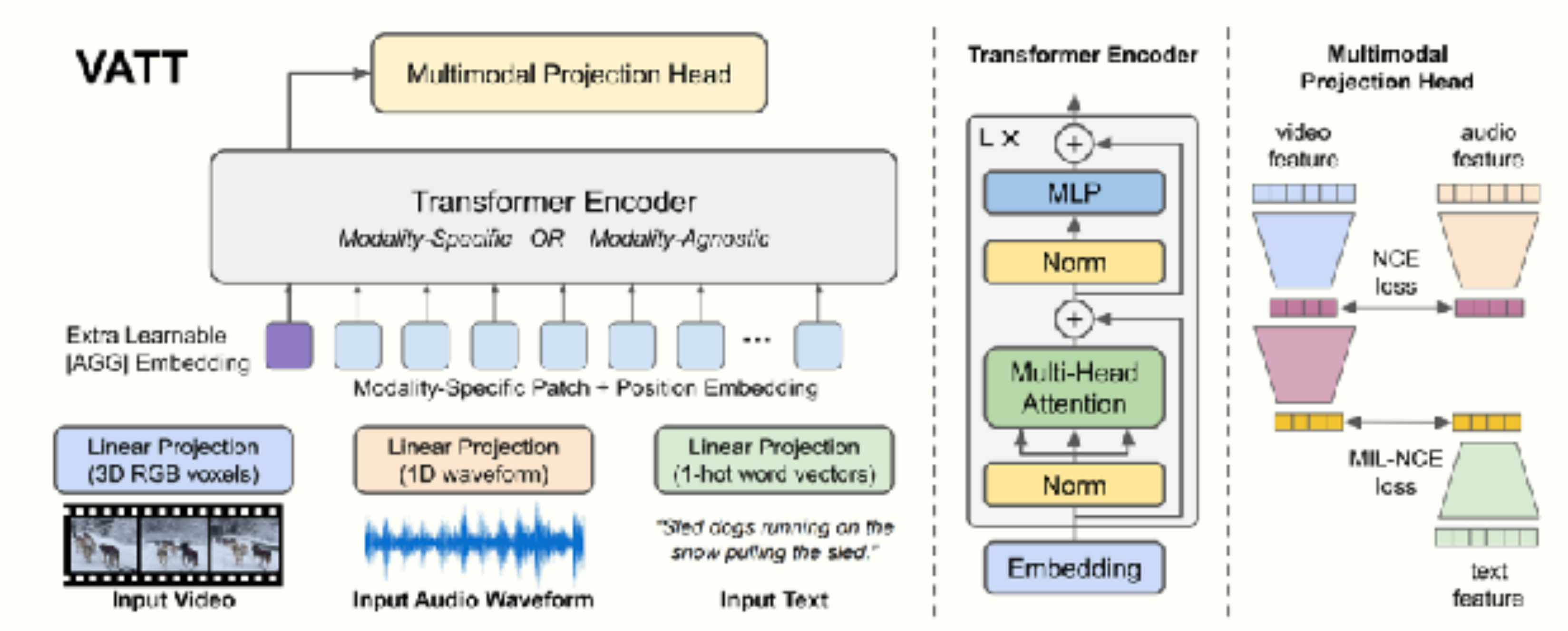
Reinforcement Learning

Cons: you need far more data. Especially in vision, as you don’t have any model inductive bias

Anything you can tokenize, you can feed to Transformer



Anything you can tokenize, you can feed to Transformer



VATT
Multimodal Projection Head
Transformer Encoder
Modality-Specific OR Modality-Agnostic
Extra Learnable [AGG] Embedding
...
Modality-Specific Patch + Position Embedding
Linear Projection (3D RGB voxels)
Linear Projection (1D waveform)
Linear Projection (1-hot word vectors)
"Sled dogs running on the snow pulling the sled."
Input Video
Input Audio Waveform
Input Text
Transformer Encoder
L ×
+
MLP
Norm
+
Multi-Head Attention
Norm
Embedding
Multimodal Projection Head
video feature
audio feature
NCE loss
MIL-NCE loss
text feature

So far…

- **CNNs** are powerful models for image/video representation learning
  - Deeper and larger models need some **tricks (skip connections & normalisation)** to be trained because of **vanishing gradients** and **instabilities**
- **RNNs** can be used to model sequence data
  - They suffer even more of vanishing gradient
- **Attention** can be used to solve the information bottleneck problem
- **Attention** can be used to replace recurrent computation
- **Scale & Attention** is all you need (we will see…)

Questions?