Université de Rennes

istic Informatique Électronique

UMR IRISA

Deep Learning Bases
(P1_3)

Elisa Fromont
M2 SIF DLV

"When you're fundraising, it's AI. When you're hiring, it's ML. When you're implementing, it's logistic regression."
—everyone on Twitter ever

What is deep learning?

= process of learning the parameters of composed (complex) functions

- NN can be deep (many layers)
- CNN can be deep (see soon)
- Other models are also deep (hierachical models, etc.)

« any » composition of differentiable functions can be optimized with gradient descent (➔ « deep » makes sens for NN)

nature International weekly journal of science

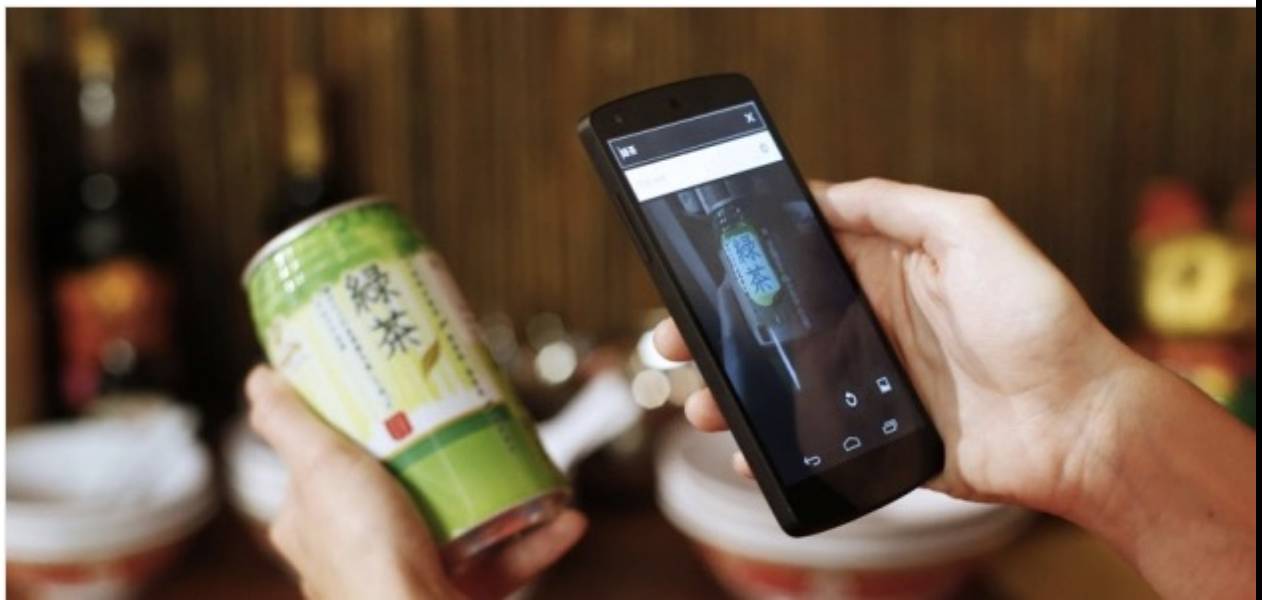
NATURE | NEWS

Deep learning boosts Google Translate tool

Internet giant's latest service employs neural networks to cut error rate by 60%, the company says.

Davide Castelvecchi

27 September 2016



nature International weekly journal of science

NATURE | NEWS

Google AI algorithm masters ancient game of Go

Deep-learning software defeats human professional for first time.

Elizabeth Gibney

27 January 2016

NEWS FEATURE | 25 July 2023

ChatGPT broke the Turing test — the race is on for new ways to assess AI

Large language models mimic human chatter, but scientists disagree on their ability to reason.

Celeste Biever

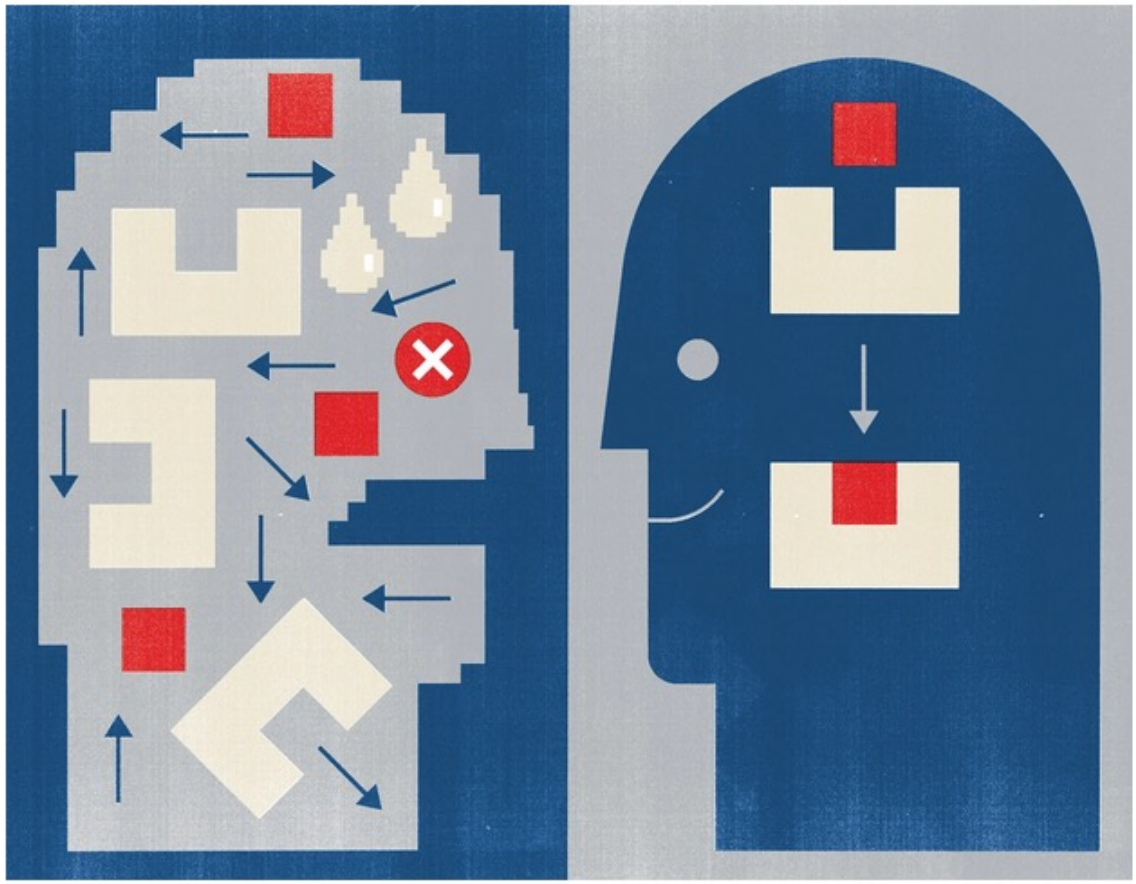


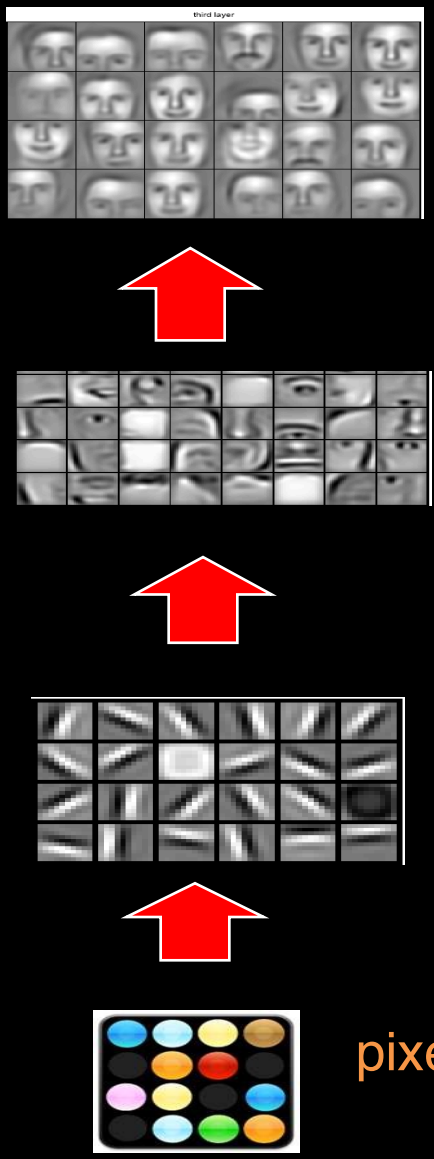
Illustration by The Project Twins

Turing Award 2018

ACM named Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

Why Deep Learning?

- Biological Plausibility – e.g. Visual Cortex
- Hastad proof - Problems which can be represented with a polynomial number of nodes with *k* layers, may require an exponential number of nodes with *k*-1 layers (e.g. parity)
- Highly varying functions can be efficiently represented with deep architectures
  - Less weights/parameters to update than a less efficient shallow representation
- Sub-features created in deep architecture can potentially be shared between multiple tasks
  - Type of Transfer/Multi-task learning



Difficulties of supervised training of deep networks

- Early layers of MLP do not get trained well
  - Diffusion of Gradient – error attenuates as it propagates to earlier layers
  - Leads to slow training
  - Exacerbated since top couple layers can usually learn any task "pretty well" and thus the error to earlier layers drops quickly as the top layers "mostly" solve the task– lower layers never get the opportunity to use their capacity to improve results
  - Need a way for early layers to do effective work
- Often not enough labeled data available while there may be lots of unlabeled data
  - Can we use unsupervised/semi-supervised approaches to take advantage of the unlabeled data
- Deep networks tend to have more local minima problems than shallow networks during supervised training

In practice: which deep neural network?

1. Deep but with very constrained architectures ➔ convolutional neural networks
   ➔ recurrent neural networks

2. Deep but with unsupervised weight initialization ➔ layer-wise training with e.g. auto-encoders

3. Deep with really good computers ➔ GPU/TPU

CONVOLUTIONAL NEURAL NETWORKS (CNN)

A particular architechture: CNN

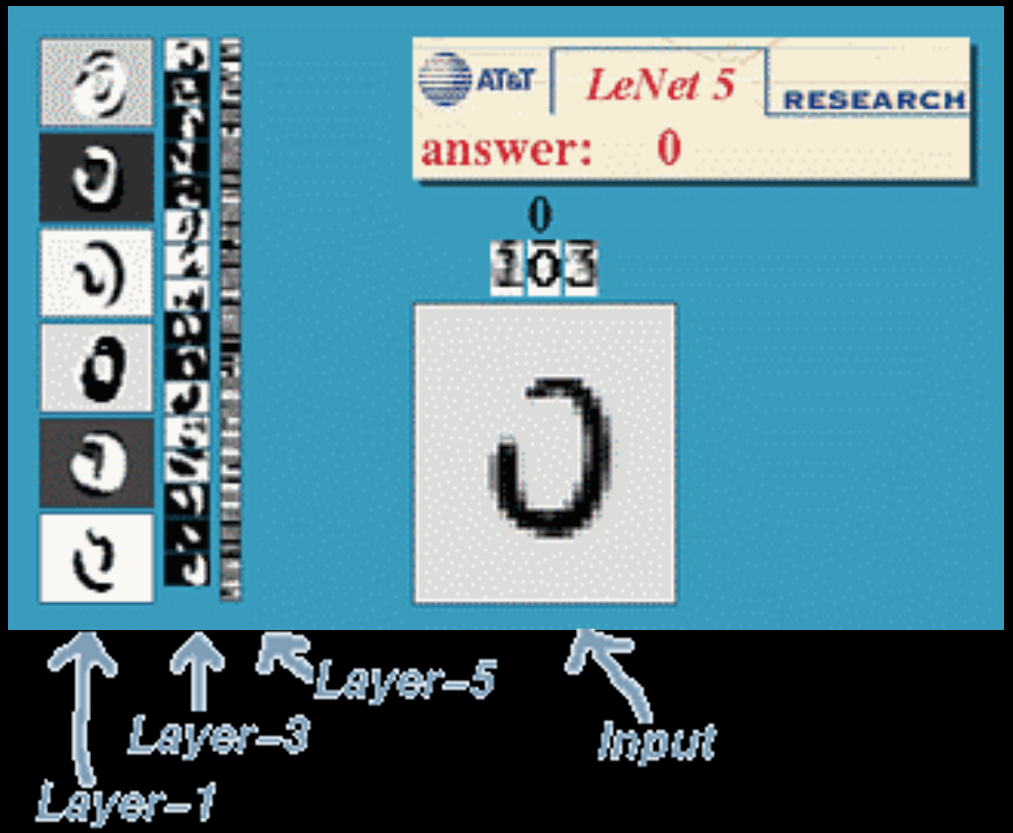
- A special kind of multi-layer neural networks.
- Implicitly extract relevant features.
- A feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks CNNs are trained with a version of the back-propagation algorithm.
- Particularly suitable for signal processing applications (for example computer vision, speech recognition)
  - ex: digit recognition, image classification…

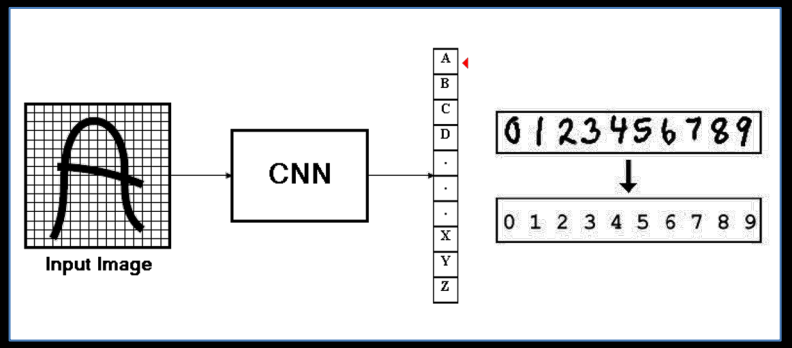
History

Yann LeCun

Yoshua Bengio

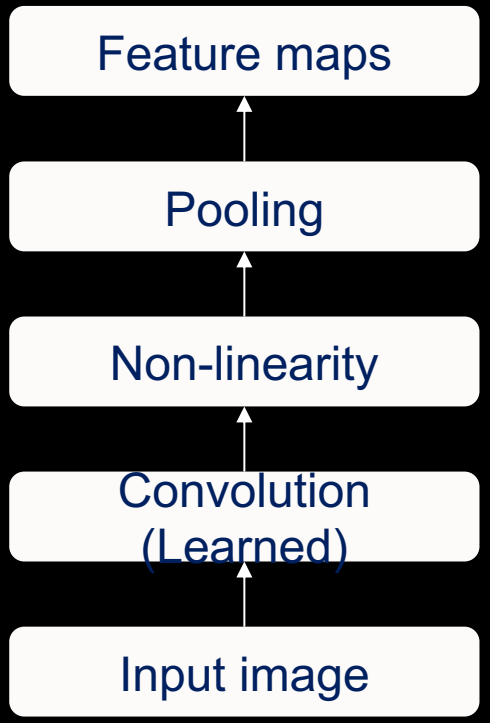
In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

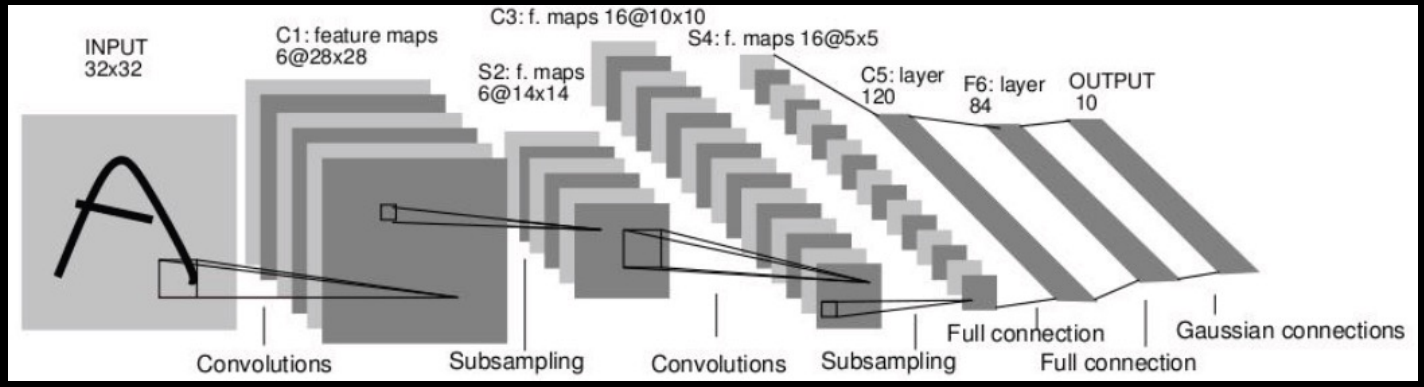




CNN: overview

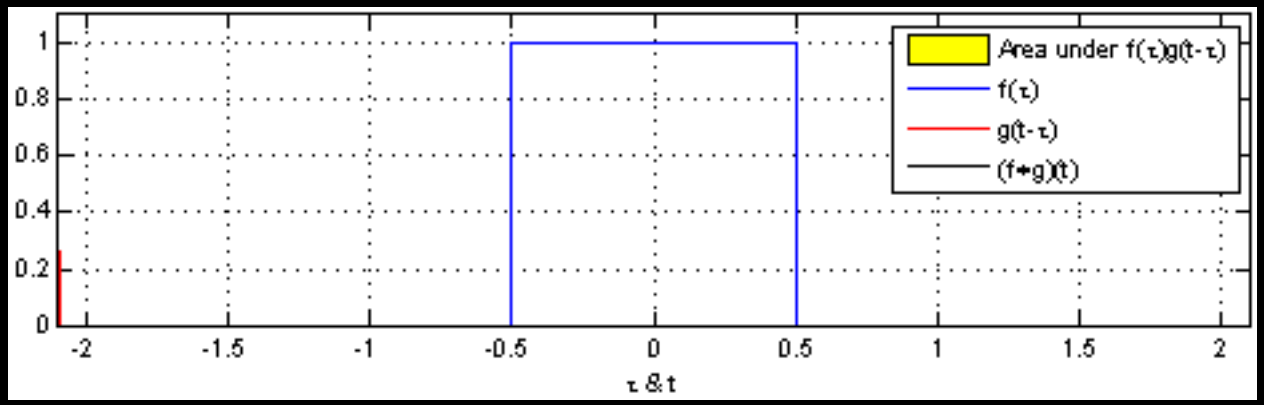
- Neural network with specialized connectivity structure
- Feed-forward:
  - Convolve input = pattern detectors
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating the learning error

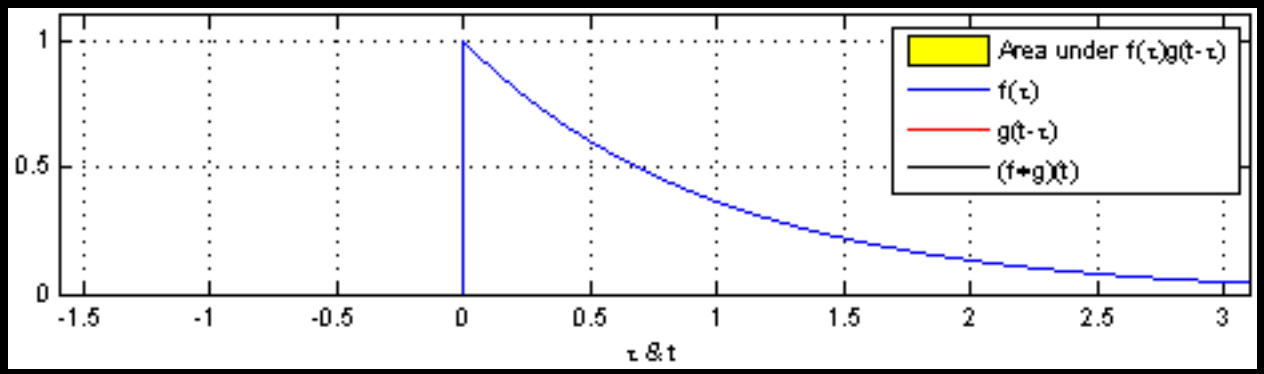




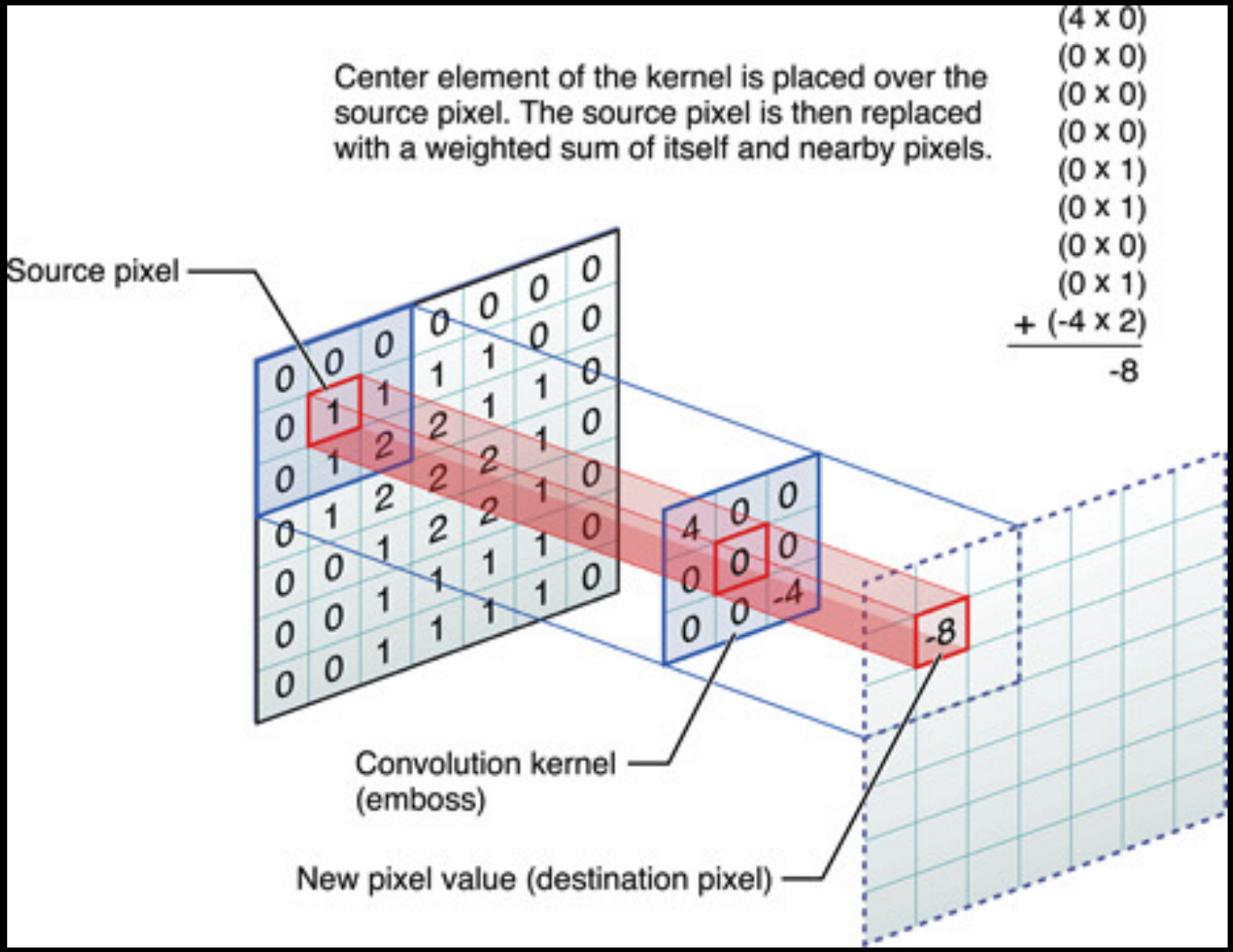
Convolution

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t) \times g(x-t)\,dt$$

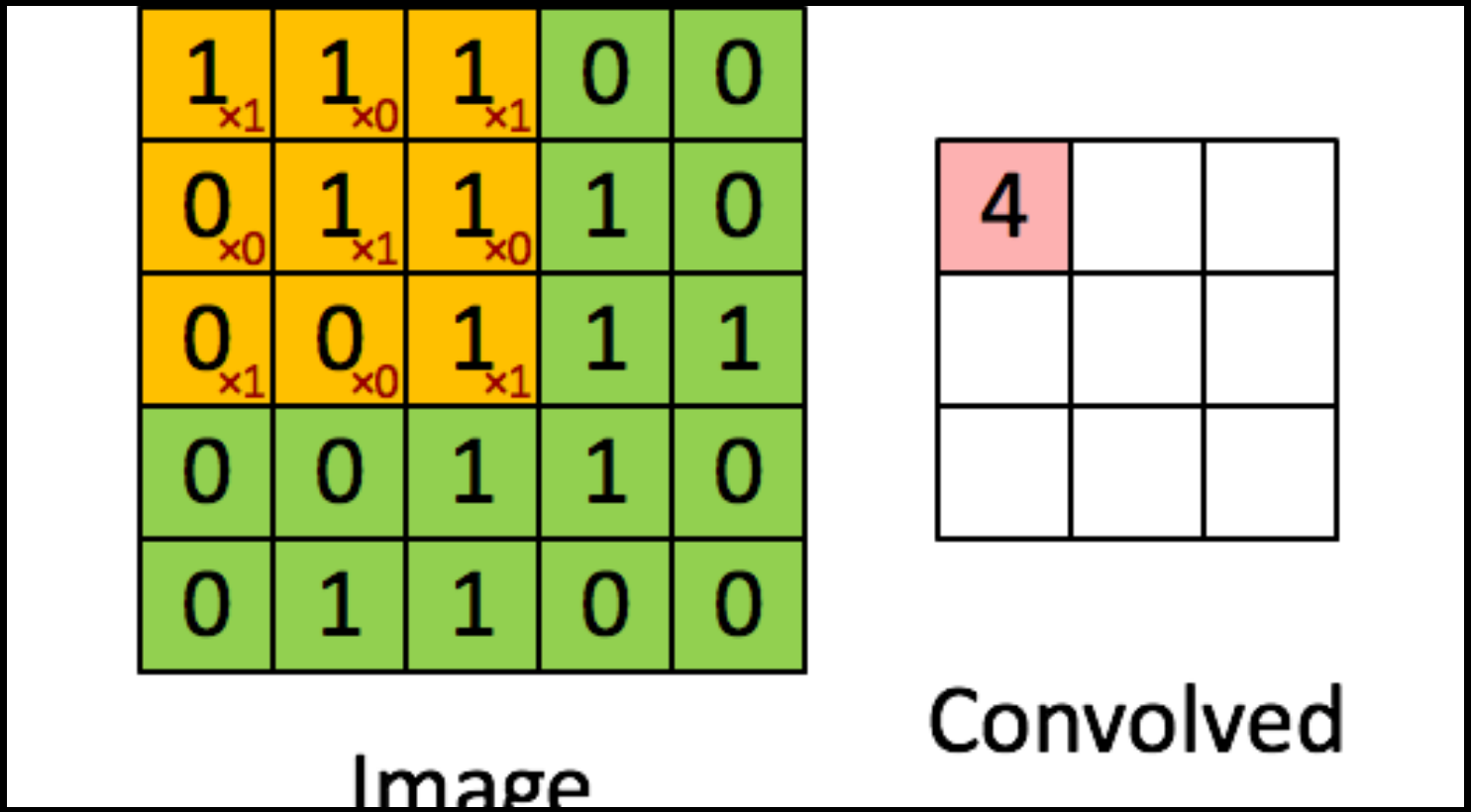




Discrete Convolution



Ex: Image convolution



http://www.songho.ca/dsp/convolution/convolution2d_example.html

Effect of the convolution Mask

Blur

1/9	**1/9**	**1/9**
1/9	1/9	1/9
1/9	1/9	1/9

Embosse

2	**0**	**0**
0	-1	0
0	0	-1

Laplacian gaussian

0	**0**	**-1**	**0**	**0**
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Gaussian Blur

0	**0**	**0**	**5**	**0**	**0**	**0**
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Effect of different average filter sizes

1/9	**1/9**	**1/9**
1/9	1/9	1/9
1/9	1/9	1/9

3X3 5X5 7X7 Average Filter

3X3 5X5 7X7 Average Filter

Effect of different gaussian filter sizes

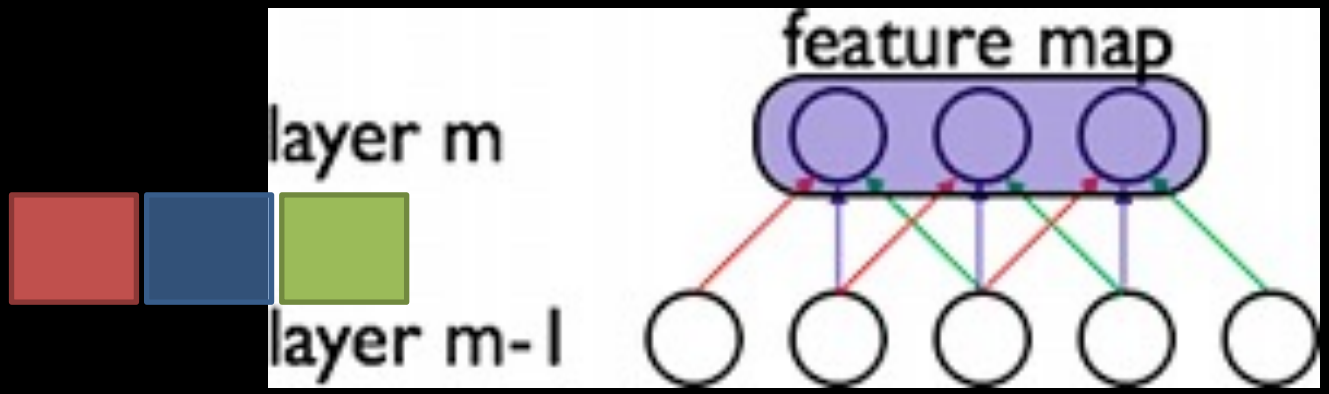
3X3 5X5 7X7 Sigma = 2 Gaussian Filter

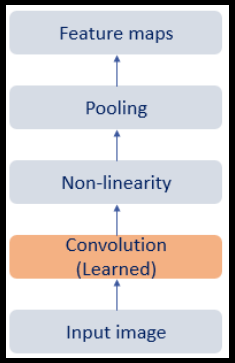
0.7	**0.8**	**0.7**
0.8	1	0.8
0.7	0.8	0.7

3X3 5X5 7X7 Sigma = 2 Gaussian Filter

Try it yourself: http://beej.us/blog/data/convolution-image-processing/

About Learning Convolution



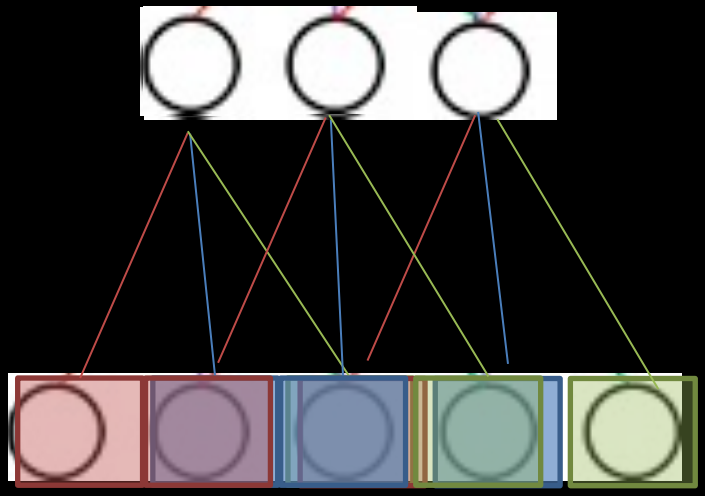


- The size of the convolution kernel is fixed depending on the application.

- The parameters of the kernel (the weights = the type of filter) correspond to the connexions from one layer to another (this is what is learned !). The resulted convolved image is called a feature map.
- The **same** convolution kernel is slid over the entire image so the weights to construct each convoluted pixels are shared (colors “red, green, blue”) in the above image)
- The smaller the convolution kernel, the less parameters to learn
- Depending of the size of the filter, the image size can be reduced

  ((dim image – dim kernel) / (stepsize for sliding)) + 1

Convolution example



Convolution kernel/mask

Exercise 4

1. What is the result of applying the given convolution kernel (on the right) to this image ?

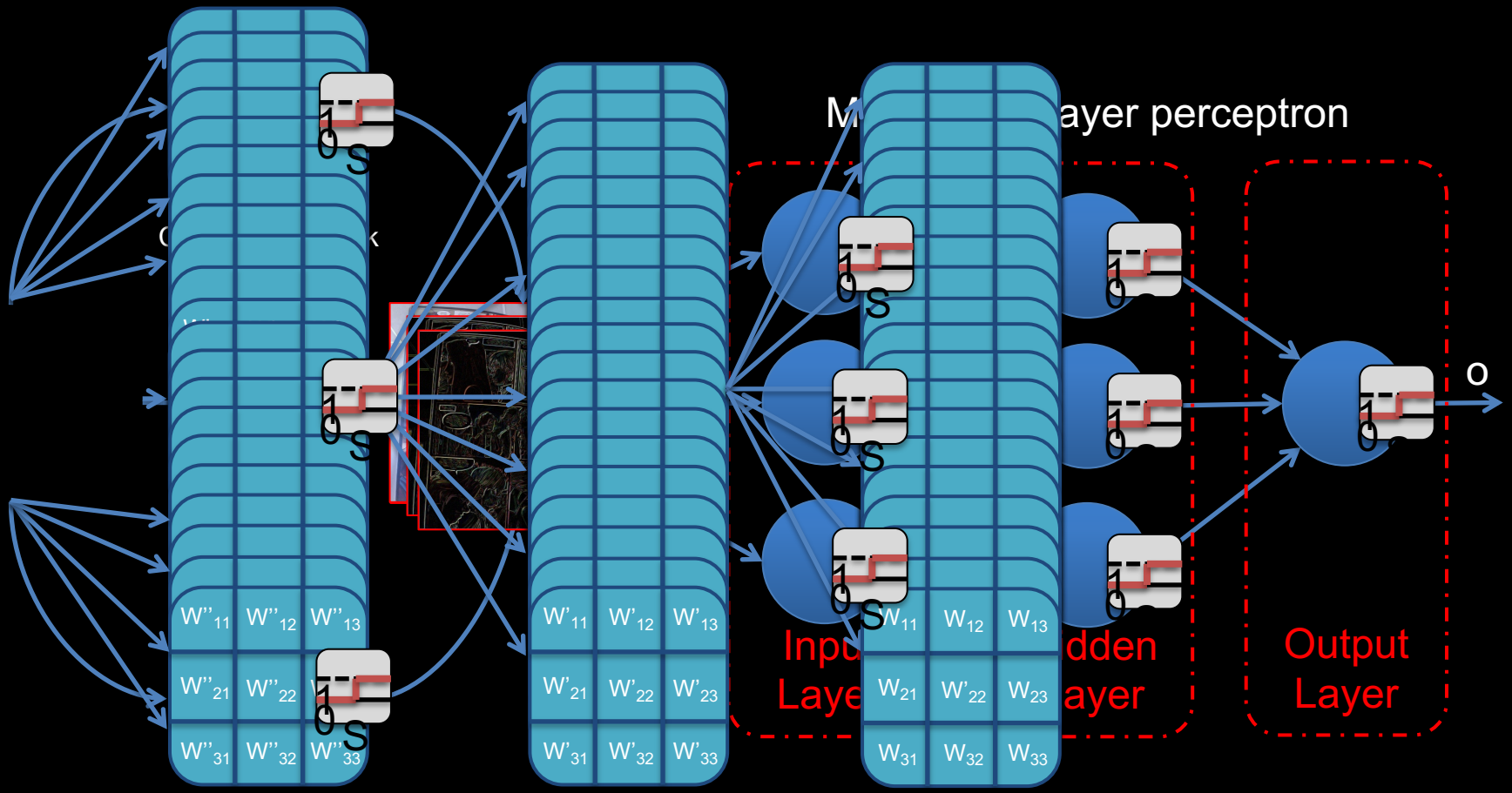
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	0	0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0

0	**-1**	**1**

2. What does the kernel do ?

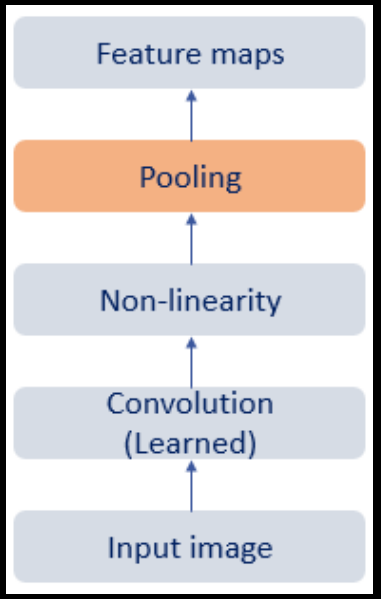
Try it yourself: http://beej.us/blog/data/convolution-image-processing/

Convolutional Neural Network

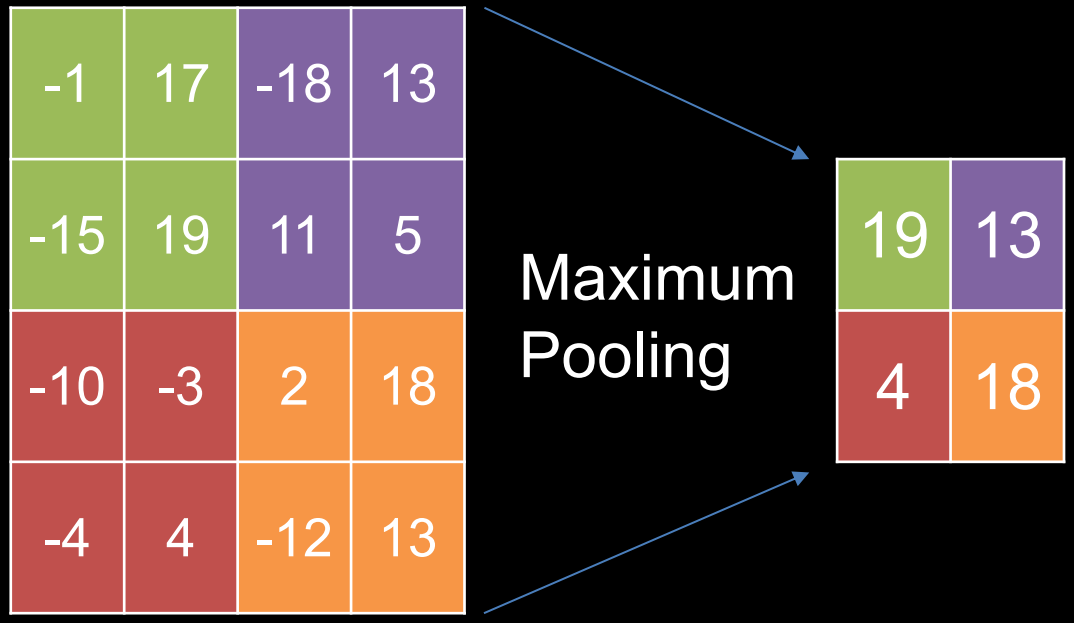


Why Pooling? (subsampling)

1. In general terms, the objective of pooling is to transform the joint feature representation into a new, more usable one that preserves important information while discarding irrelevant detail, the crux of the matter being to determine what falls in which category.
2. Achieving **invariance** to changes in position or lighting conditions, robustness to clutter, and compactness of representation, are all common goals of pooling.
3. Speed up the process (smaller feature maps = less parameters in the last layers)



Ex of Pooling



Effect:

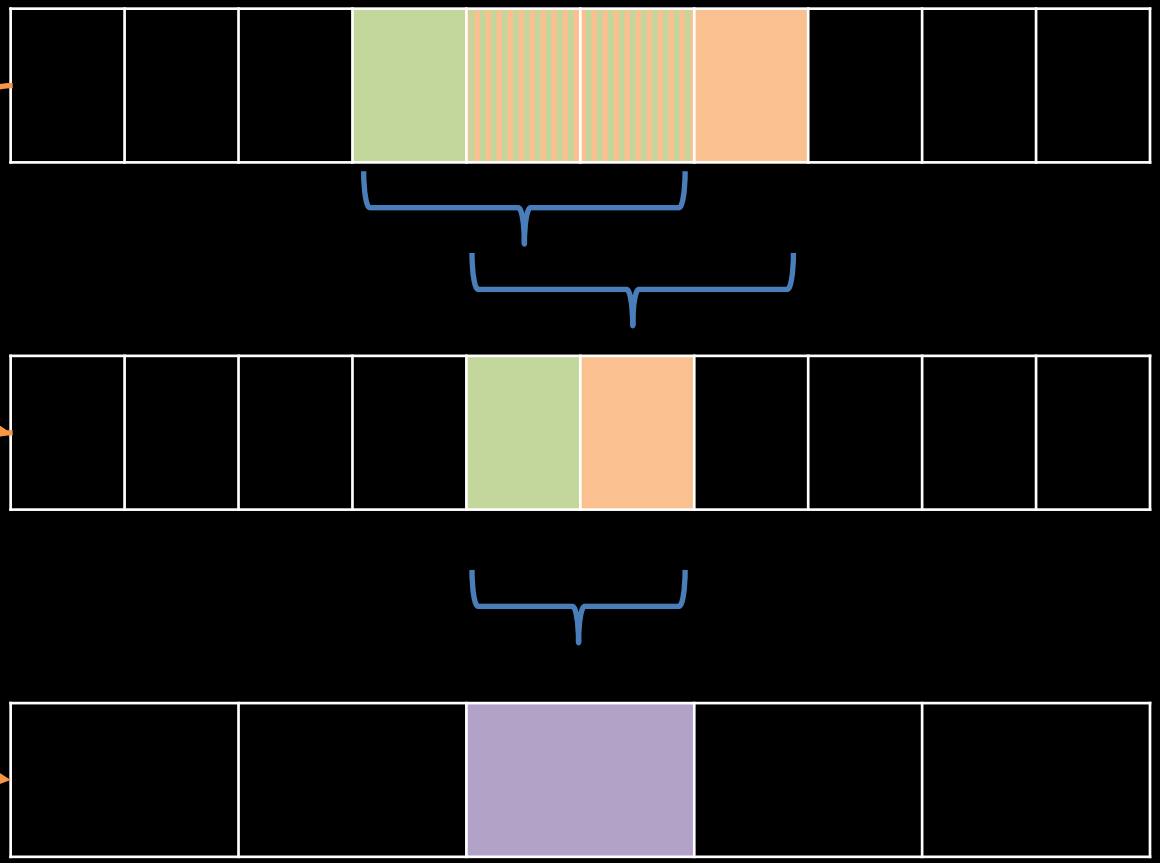
- Reduces the feature map's size
- Increases the field of view

- Average pooling
- Sum pooling
- Stochastic pooling
- Etc …

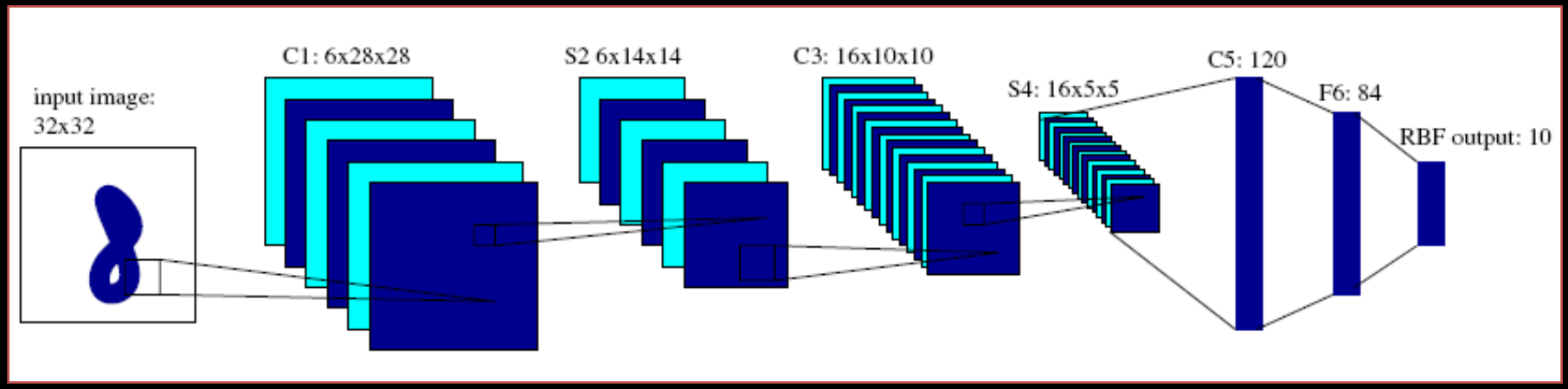
Field of view

Convolution with mask size = 3

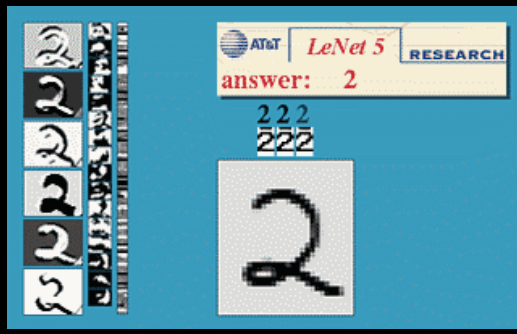
Pooling with mask size = 2



Example: LeNet5



- C1,C3,C5 : Convolutional layers (**5 × 5 × nbniputchannels)** convolution kernels (2D size given)
- S2 , S4 : Subsampling layer. (by factor 2)
- F6 : Fully connected layer.
- Nb of feature maps (6, 16, 120 and then 84) is given



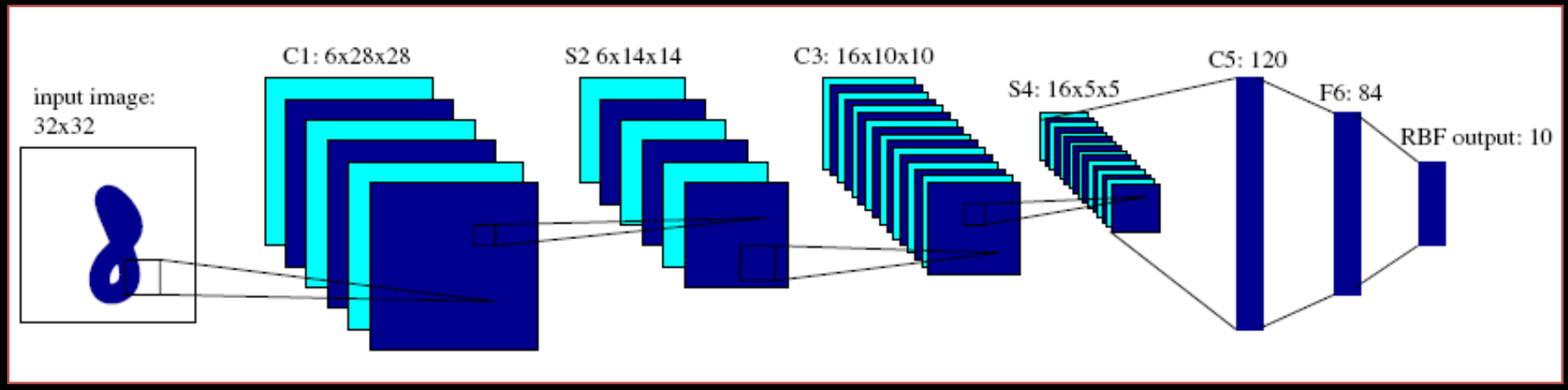
Try it out: http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

LeNet5 layers

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(6, kernel_size=5, strides=1,
activation='tanh', input_shape=train_x[0].shape, padding='same'),
#C1
    keras.layers.AveragePooling2D(), #S2
    keras.layers.Conv2D(16, kernel_size=5, strides=1,
activation='tanh', padding='valid'), #C3
    keras.layers.AveragePooling2D(), #S4
    keras.layers.Flatten(), #Flatten
    keras.layers.Dense(120, activation='tanh'), #C5
    keras.layers.Dense(84, activation='tanh'), #F6
    keras.layers.Dense(10, activation='softmax') #Output layer
])
```

- Convolution #1. Input = 32x32x1. Output = 28x28x6 conv2d
- SubSampling #1. Input = 28x28x6. Output = 14x14x6. SubSampling is simply Average Pooling so we use avg_pool
- Convolution #2. Input = 14x14x6. Output = 10x10x16 conv2d
- SubSampling #2. Input = 10x10x16. Output = 5x5x16 avg_pool
- Flatten + FC (5*5*16 = 400 ➔ 120) or Convolution #3. Input = 5x5x16. Output = 120x1x1 conv2d
- Fully Connected #1. Input = 120. Output = 84
- Fully Connected #2. Input = 84. Output = 10

Exercise 5: Count the parameters



How many parameters would this network need to learn?

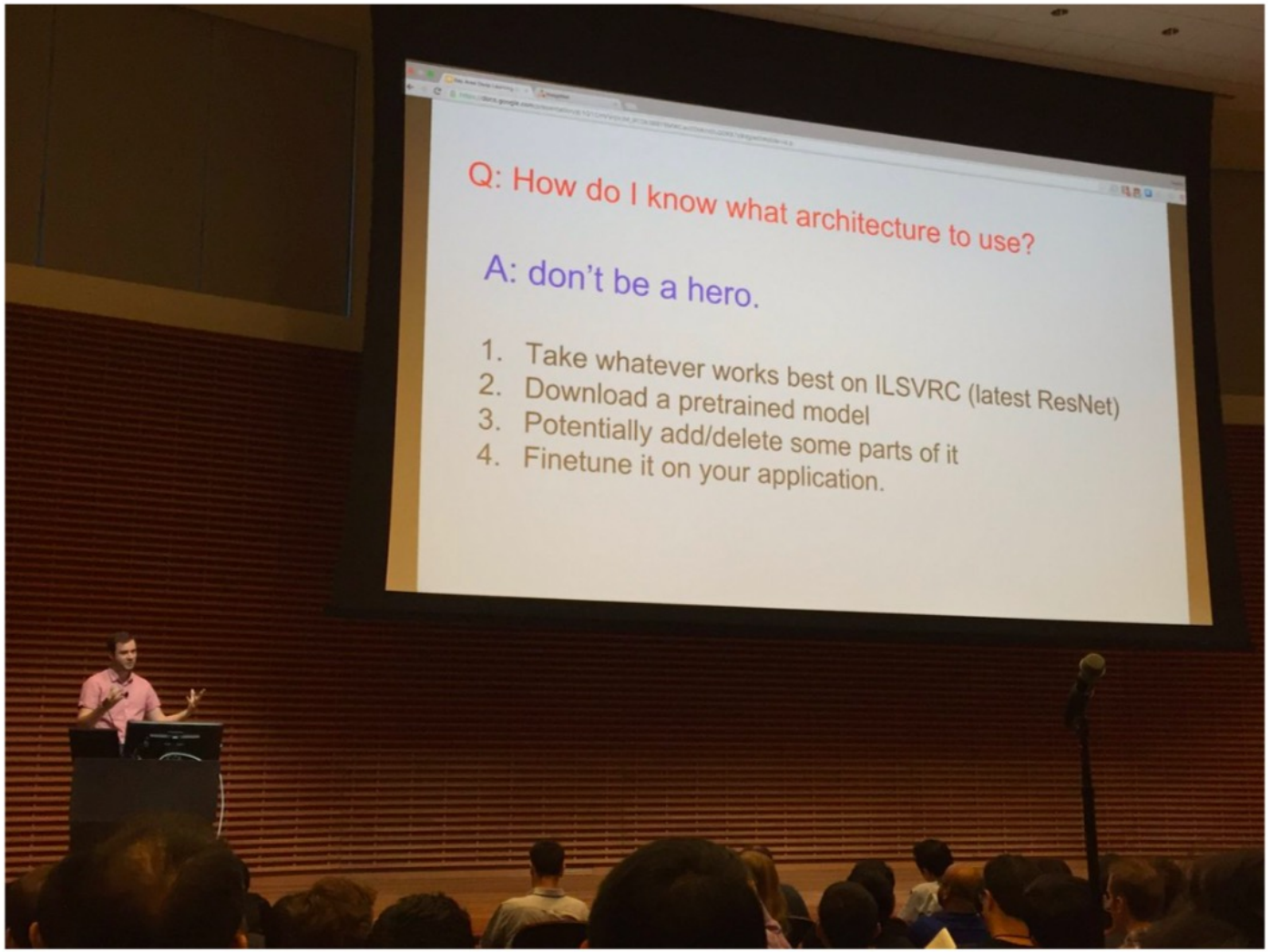
- a 2D convolution kernel applied to multiple feature maps becomes 3D
- Each feature map is produced with a different kernel
- there is one bias per (3D) convolution kernel (and then one bias per neurone in the FC layers as usual)

Exercise 6: Adapt a CNN architecture

- Suppose that your input image is not of size 32*32 anymore but 64*64.

- What would you change in the previous architecture to be able to predict your 10 different labels?
  - If you let only let convolution kernel C5 to change
  - (discuss the other possibilities: should every neuron at the entrance of the MLP have a field of view corresponding to the entire image?)

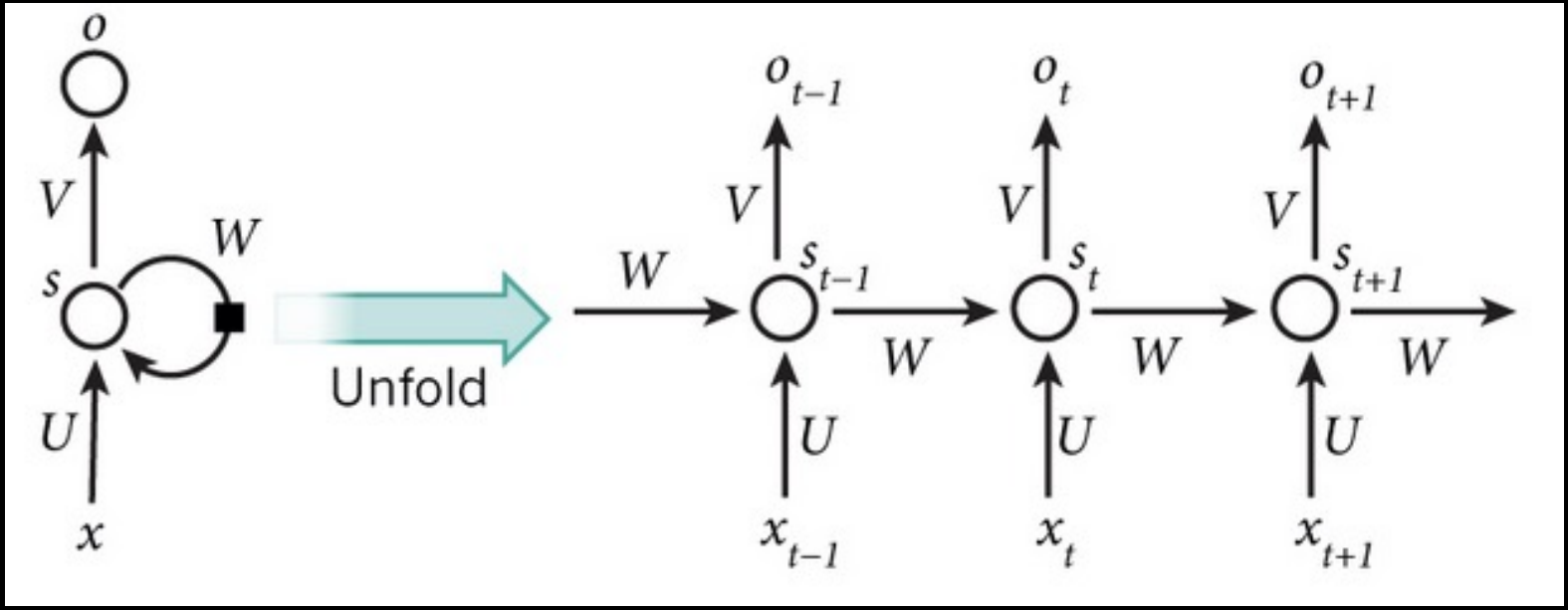
So, how to choose?



cf. Nathan Mundhenk's this morning talk
Andrej Karpathy, Deep Learning Summer School 2016

RECURRENT NEURAL NETWORKS (RNN)

A particular « brick » in a network architecture: the recurrent neurone



$$s_t = f(Ux_t + Ws_{t-1})$$

A recurrent neural network and the unfolding in time of the computation involved in its forward computation (S_t = « memory of the network »). SOURCE: Nature 2015

Recurrent?

« Recurrent Neural Networks are called recurrent because they perform the **same task** for every element of a sequence, with the output being depended on the previous computations… they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps »

Pb: vanishing gradient in RNN

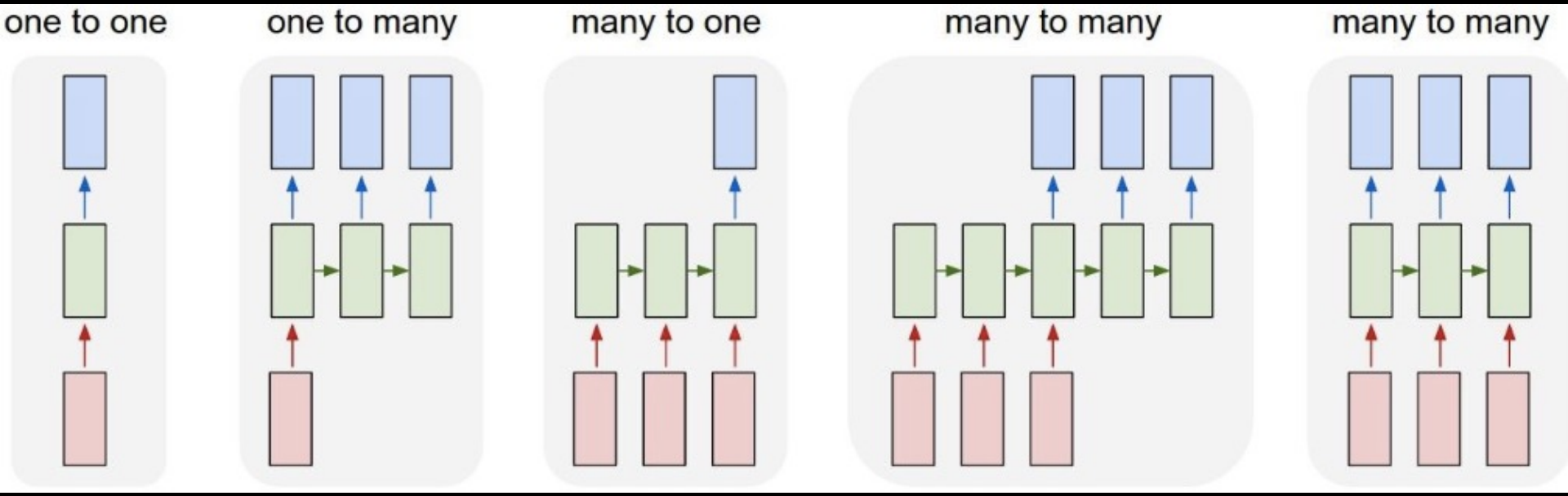
Solution: 2 very popular RNN:

1. LSTM (Long Short Term Memory)
2. GRU (Gated Recurrent Unit)

(https://jhui.github.io/2017/03/15/RNN-LSTM-GRU/)

(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

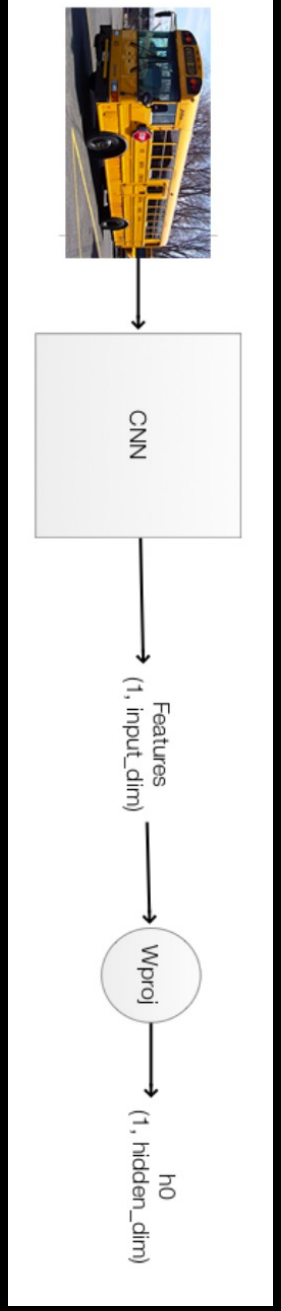
What can RNNs do?

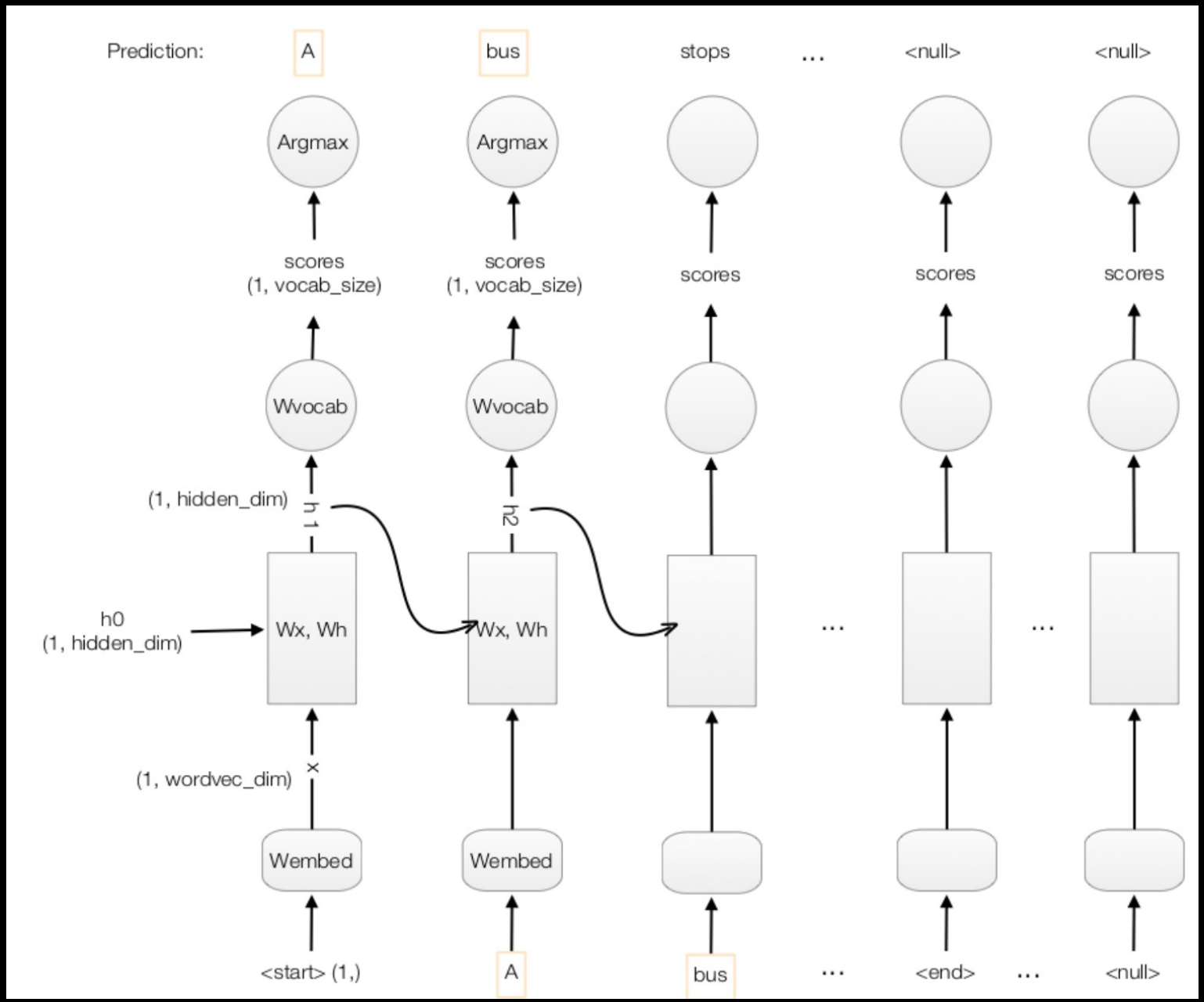


e.g. Ir
image

e.g. **Video classification on frame level**

Ex: predict the caption of an image



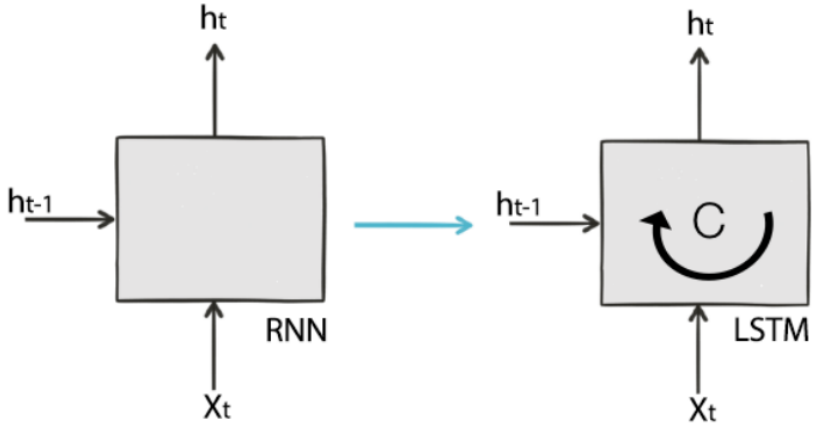


LSTM (Hochreiter & Schmidhuber (1997))

h_t in RNN serves 2 purpose:

- Make an output prediction, and
- A hidden state representing the data sequence processed so far.

LSTM splits these 2 roles into 2 separate variables h_t and C. The hidden state of the LSTM cell is now C.



Long & Short Term Memory

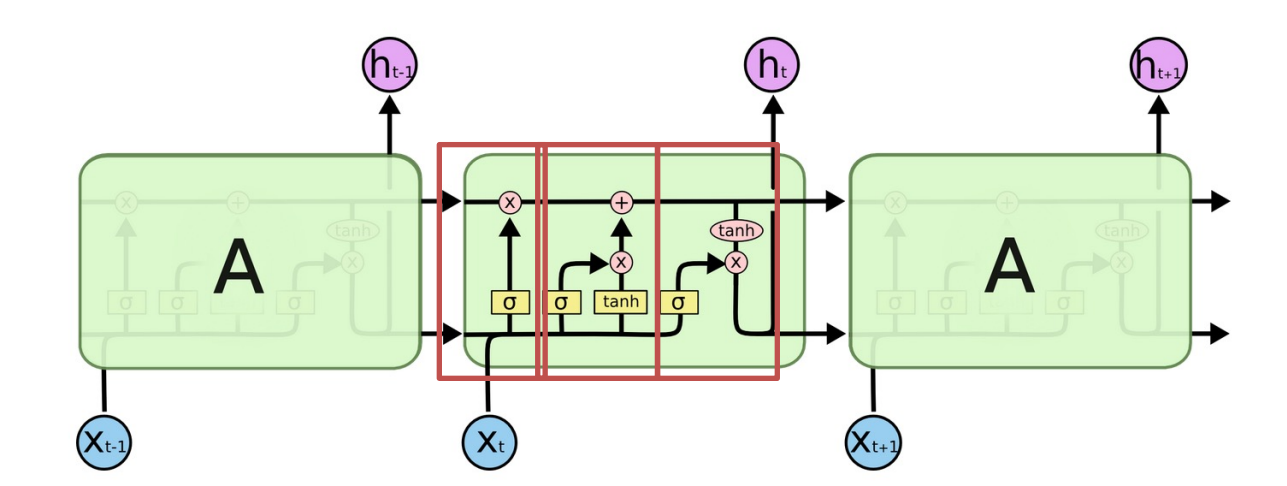
LSTM: 3 gates

There are 3 gates in LSTM. All gates are function of x_t and h_{t-1}

$$gate = \sigma(W_x X_t + W_h h_{t-1} + b)$$

- $gate_{forget}$ controls what part of the previous cell state will be kept.
- $gate_{input}$ controls what part of the new computed information will be added to the cell state C.
- $gate_{out}$ controls what part of the cell state will exposed as the hidden state.

https://jhui.github.io/2017/03/15/RNN-LSTM-GRU/



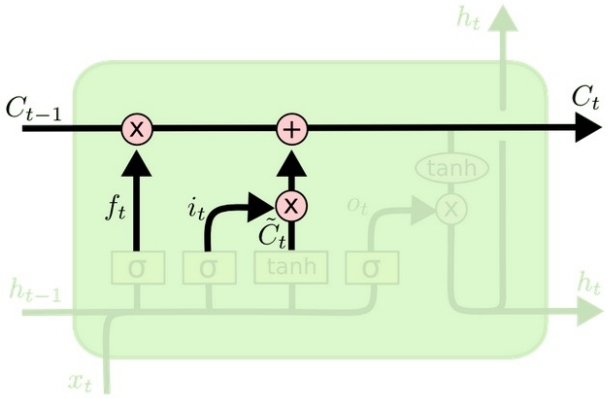
LSTM equations

There are 3 gates controlling what information will pass through:

$$gate_{forget} = \sigma(W_{fx}X_t + W_{fh}h_{t-1} + b_f)$$
$$gate_{input} = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i)$$
$$gate_{out} = \sigma(W_{ox}X_t + W_{oh}h_{t-1} + b_o)$$

3 equations to update the cell state and the hidden state:

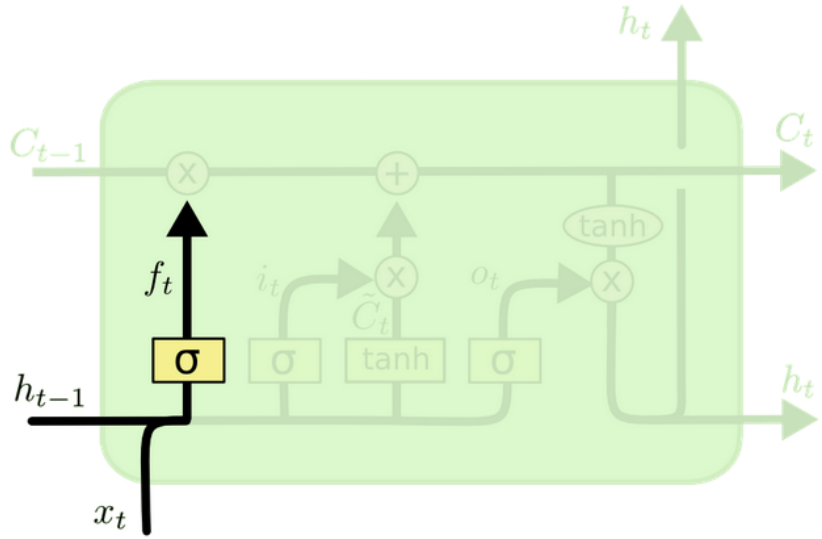
$$\tilde{C} = \tanh(W_{cx}X_t + W_{ch}h_{t-1} + b_c)$$
$$C_t = gate_{forget} \cdot C_{t-1} + gate_{input} \cdot \tilde{C}$$
$$h_t = gate_{out} \cdot \tanh(C_t)$$



- W_{ix}: the weight matrix associated between the input X_t and the Input gate I
- $\tilde{C}$: is the new proposal for the state
- σ :in the original LSTM, the sigmoid activation function

LSTM: forget gate

It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents “completely keep this” while a 0 represents “completely get rid of this. »

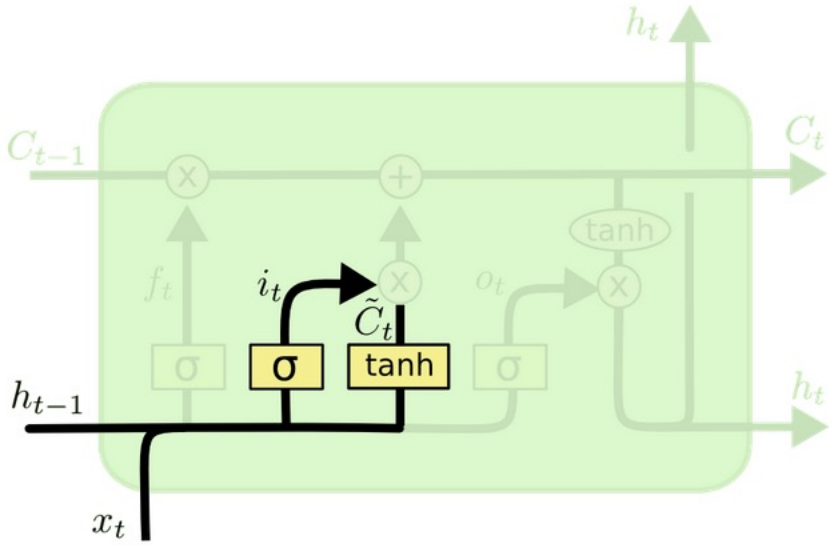


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Ex: when we see a new subject, we want to forget the gender of the old subject.

LSTM: input gate

Decide what new information we're going to store in the cell state. 1) a sigmoid layer called the "input gate layer" decides which values we'll update. 2) a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state



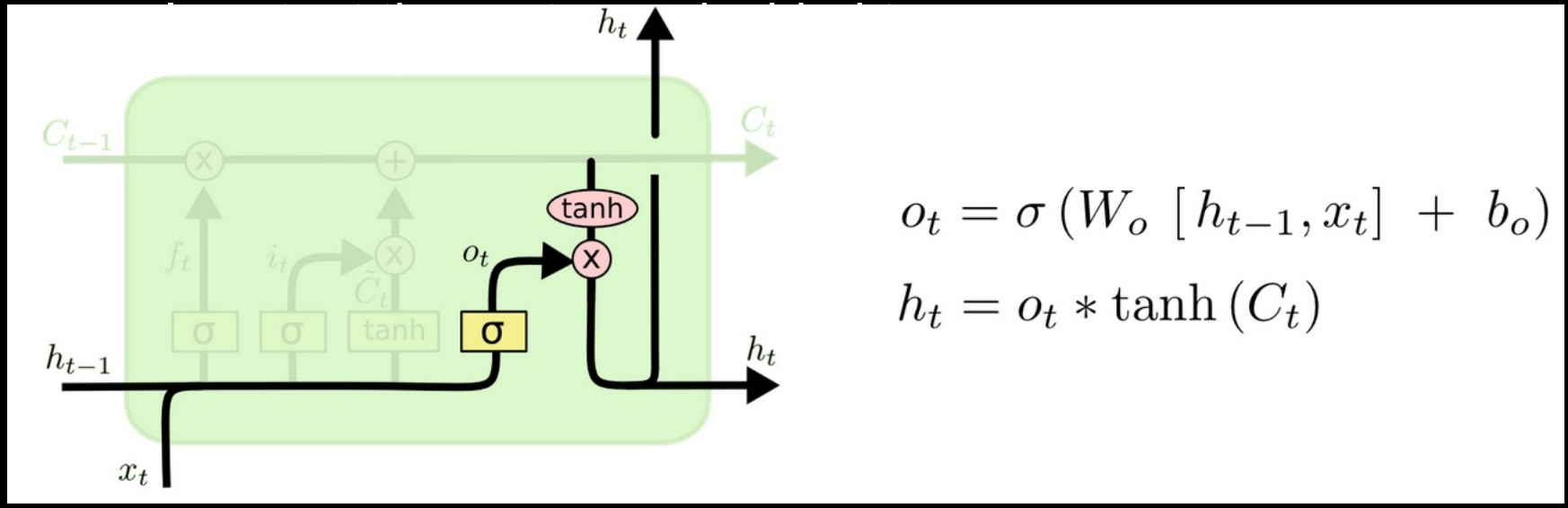
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Ex: we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting

LSTM: out gate

Decide what we're going to output. 1) run a sigmoid layer which decides what parts of the cell state we're going to output. 2) put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that



Ex: output verb well conjugated

GRU Unit

Compare with LSTM, GRU does not maintain a cell state C and use 2 gates instead of 3.

$$gate_r = \sigma(W_{rx}X_t + W_{rh}h_{t-1} + b)$$
$$gate_{update} = \sigma(W_{ux}X_t + W_{uh}h_{t-1} + b)$$

The new hidden state is compute as:

$$h_t = (1 - gate_{update}) \cdot h_{t-1} + gate_{update} \cdot \tilde{h}_t$$

As seen, we use the compliment of $gate_{update}$ instead of creating a new gate to control what we want to keep from the h_{t-1}.

The new proposed $\tilde{h}_t$ is calculated as:

$$\tilde{h}_t = \tanh(W_{hx}X_t + W_{hh} \cdot (gate_r \cdot h_{t-1}) + b)$$

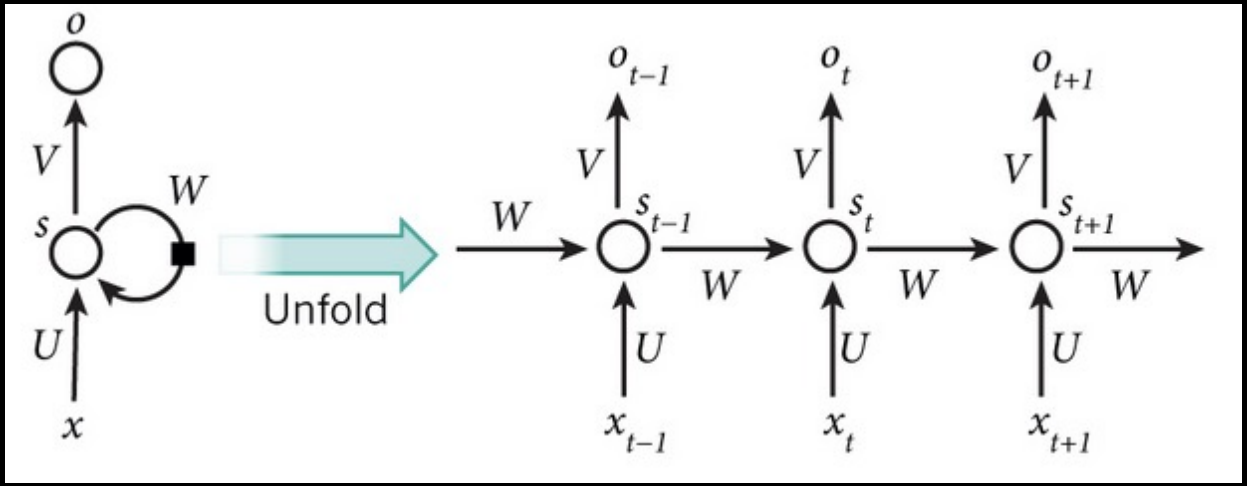
We use $gate_r$ to control what part of h_{t-1} we need to compute a new proposal.

GRU vs LSTM?

- GRU has two gates (*reset* and *update* gates)
- LSTM has three gates (*input*, *output* and *forget* gates).

- The GRU unit controls the flow of information like the LSTM unit, but without having to use a ***memory unit***. It just exposes the full hidden content without any control.
- **LSTMs** should in theory **remember longer sequences** than GRUs and outperform them in tasks requiring modeling long-distance relations.
- Otherwise, GRU performance is on par with LSTM, but computationally ***more efficient*** (*less complex structure*).

Exercise 8: count parameters in RNN



$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$o_t = \text{softmax}(Vs_t)$$

The input (e.g. a word) is of size m
There are n LSTM units (neurons) in the hidden layer
The output (e.g. a word) is of size k
What's the # parameters for this RNN model?

Towards Generative Models

AE, VAE, GAN, DM

Generative vs discriminative models

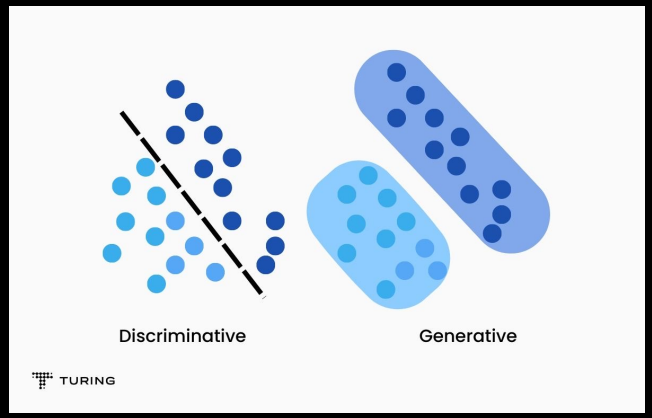
A generative model is a statistical model of the joint probability distribution P (X,Y) on given observable variable X and target variable Y. E.g *Naive Bayes, HMM, VAE, GAN, auto-regressive models (e.g. LLMs).*

A discriminative model is a model of the conditional probability P (Y|X = x) of the target Y, given an observation x *E.g logistic regression,…*

Classifiers computed without using a probability model are also referred to loosely as "discriminative". *E.g. Decision trees, SVM, MLP,…*

https://en.wikipedia.org/wiki/Generative_model

Classification in both cases



- Generative classifiers assume a functional form for P(Y) and P(X|Y), then generate *estimated parameters from the data* and use the Bayes' theorem to calculate P(Y|X) (posterior probability).

$$posterior = \frac{prior \times likelihood}{evidence} \Rightarrow P(Y|X) = \frac{P(Y) \cdot P(X|Y)}{P(X)}$$

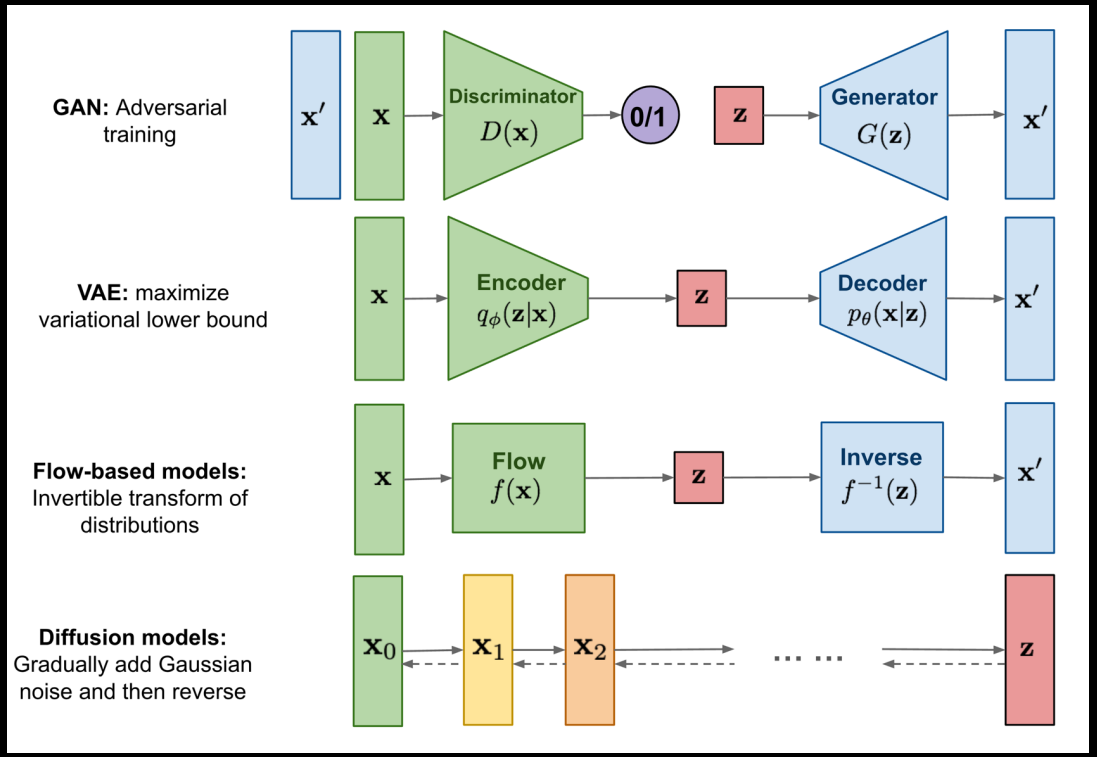
- Discriminative (conditional) classifiers assume a functional form of P(Y|X) and estimate the parameters directly from the provided data.

https://www.turing.com/kb/generative-models-vs-discriminative-models-for-deep-learning

Goal of (deep) generative models

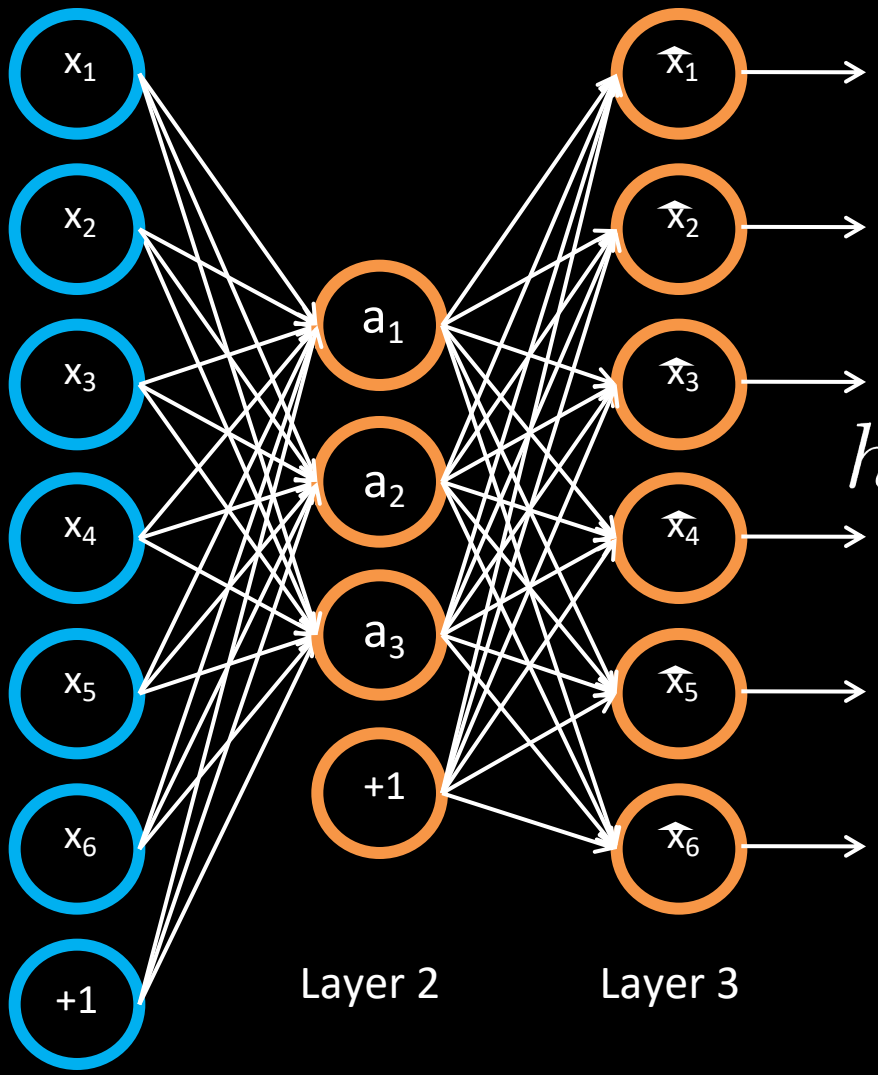
Usually
trained « unsupervised »
(not for classification)

Estimate the unknown distribution p(X) of the data, so that by sampling from this estimated distribution, we can generate new samples that look very much like the samples from the original distribution.



https://emkademy.medium.com/1-first-step-to-generative-deep-learning-with-autoencoders-22bd41e56d18

Autoencoders (AE)



Network is trained to output the input (learn identify function).

$$h_\theta(x) \approx x$$

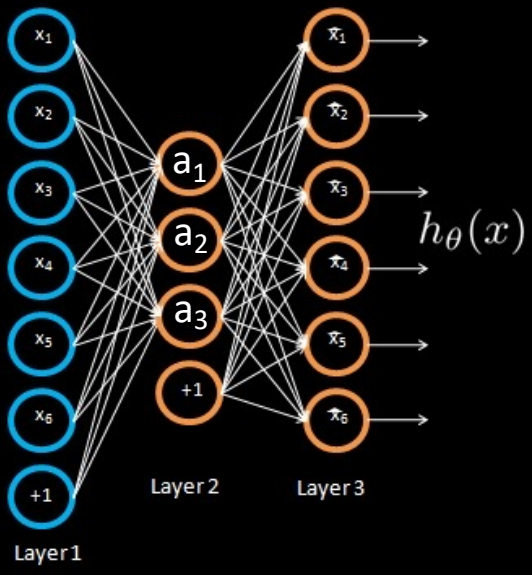
Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be **sparse**.

Train Autoencoders

Training a sparse autoencoder.

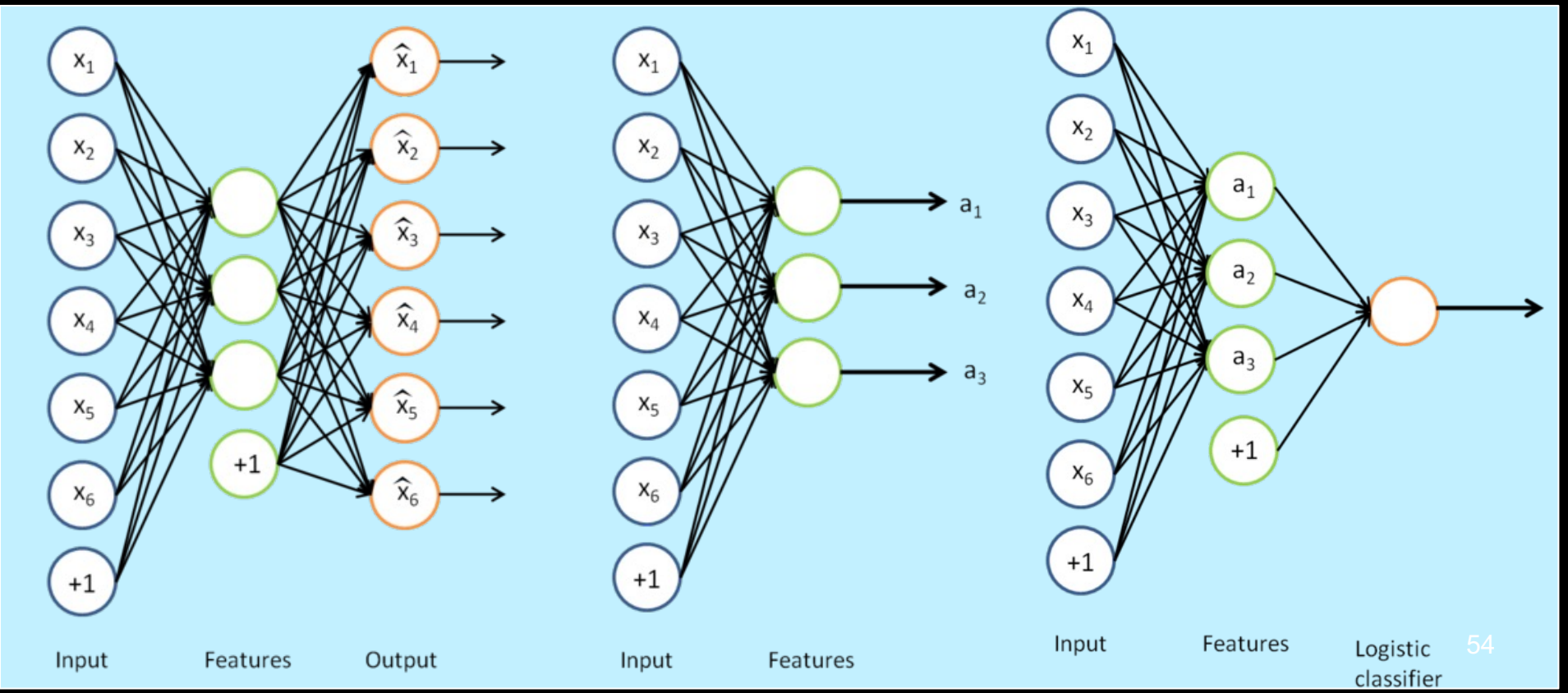
Given unlabeled training set x_1, x_2, …

$$\min_{\theta} \underbrace{\|h_\theta(x) - x\|^2}_{\text{Reconstruction error term}} + \lambda \underbrace{\sum_i |a_i|}_{L_1 \text{ sparsity term}}$$



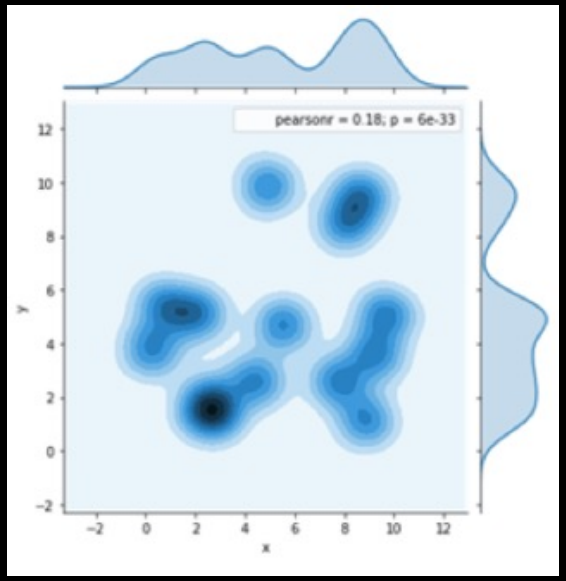
Auto-Encoders as feature generators

Can use just new features in the new training set or concatenate both



Autoencoders (AE)

- Autoencoder can do dimensionality reduction
- Autoencoders are not directly made for generative modelling ➔ reconstruction does not explicitly help to model the data distribution.
- Autoencoder mostly learns a sparse latent space => « distinct clusters in the latent space. The decoder has never learned to reconstruct vectors in between the clusters, so it will produce very abstract things - mostly garbage. »



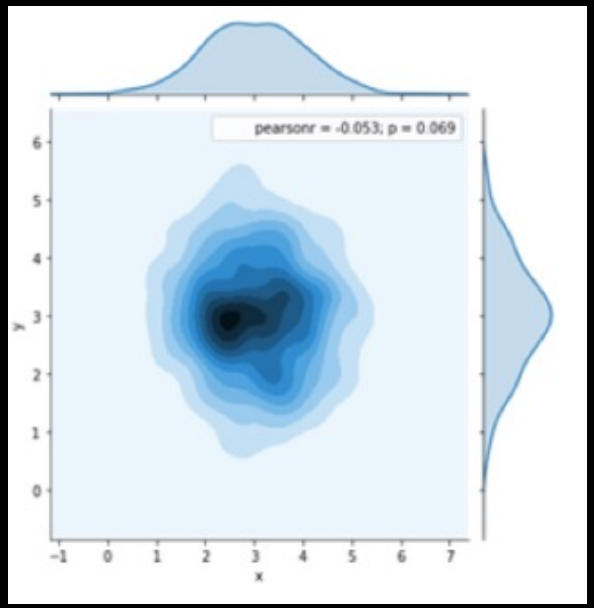
- Here are a number of tasks where they can be used:
  - classification,
  - clustering,
  - anomaly detection,
  - recommendation systems,
  - dimensionality reduction,
  - cleaning noisy images…

Variational Autoencoders (VAE)

VAE is an autoencoder whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to generate some new data.

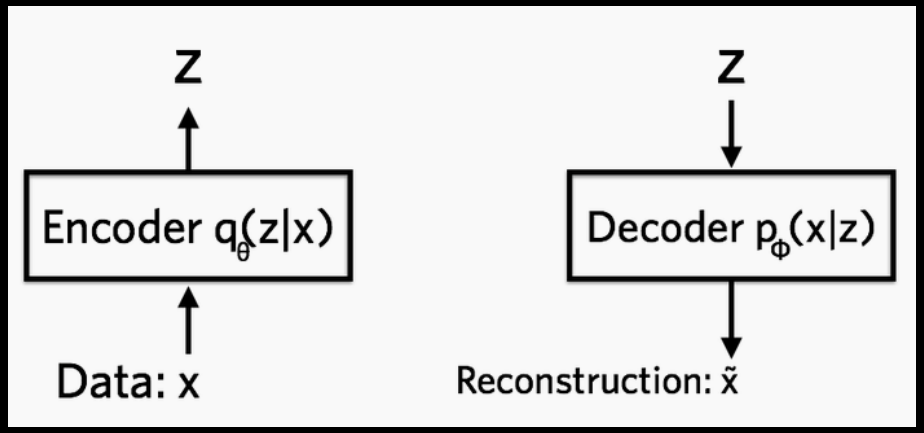
Moreover, the term “variational” comes from the close relation there is between the regularisation and the variational inference method in statistics.

https://en.wikipedia.org/wiki/Variational_Bayesian_methods



https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE in practice



Negative log likelihood

Kullback-Leibler divergence

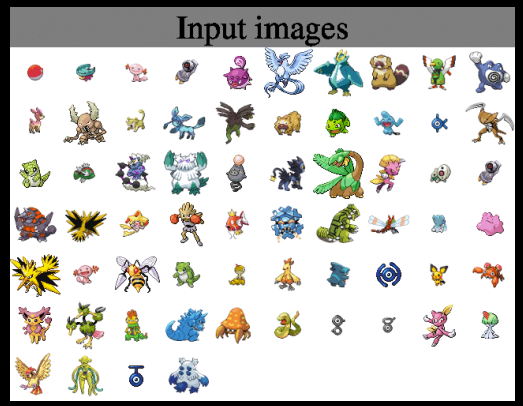
$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \,||\, p(z))$$

- i: index of the data point x_i
- **z** = bottleneck = latent space
- Neg log likelihood = reconstruction error
- KL : comparison between two distributions (Wassertein or Bhattacharyya distances are other examples)

Normal distribution with mean zero and variance one

$\mathcal{N}(0, 1)$

Generative Adversarial Networks
([Goodfellow NIPS 2014])



Input: noise

Output: generated images

Generator

Discriminator

{Real, Generated}

Real images

- The discriminator is trained to discriminate real from generated images
- The generator is trained to fool the discriminator
- During the learning phase, neither the Generator nor the Discriminator become stronger than the other

GAN in practice

- GAN Loss : min max(D, G) $E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$

- Discriminator Loss: **maximize** the average of the log probability for real object and the log of the inverted probabilities of fake objects (from the generator)

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

- Generator Loss: **minimize**

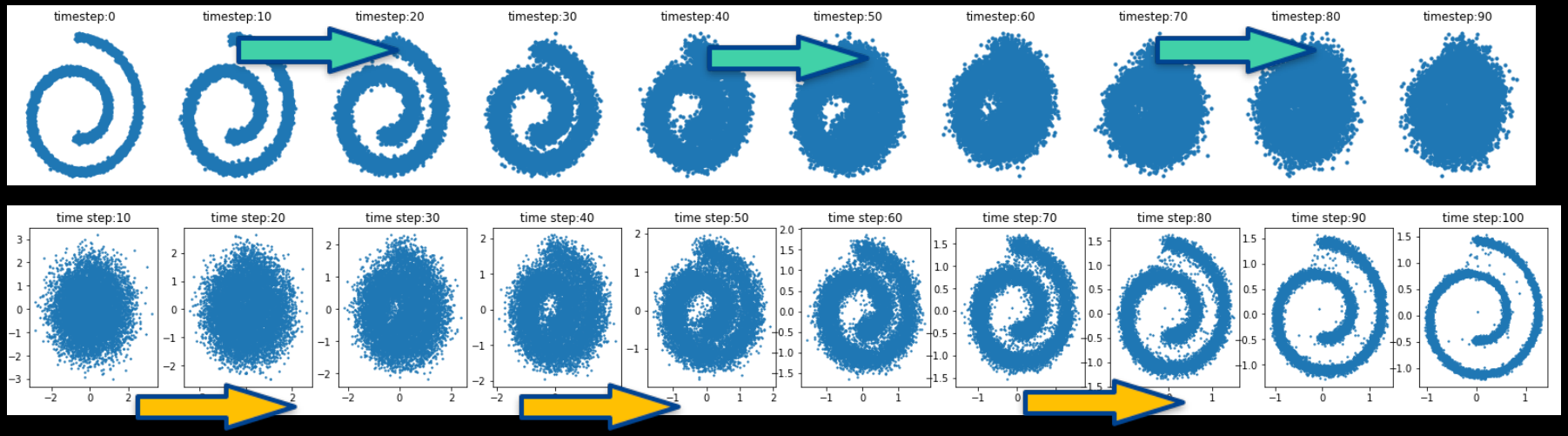
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

G gets rewarded if it successfully fools the discriminator, and gets penalized otherwise. Too avoid saturation (generator stops too early) : – log(D(G(z))) instead.

https://machinelearningmastery.com/generative-adversarial-network-loss-functions/
https://arxiv.org/pdf/1711.10337.pdf (overview of all the different GAN)
https://neptune.ai/blog/gan-loss-functions

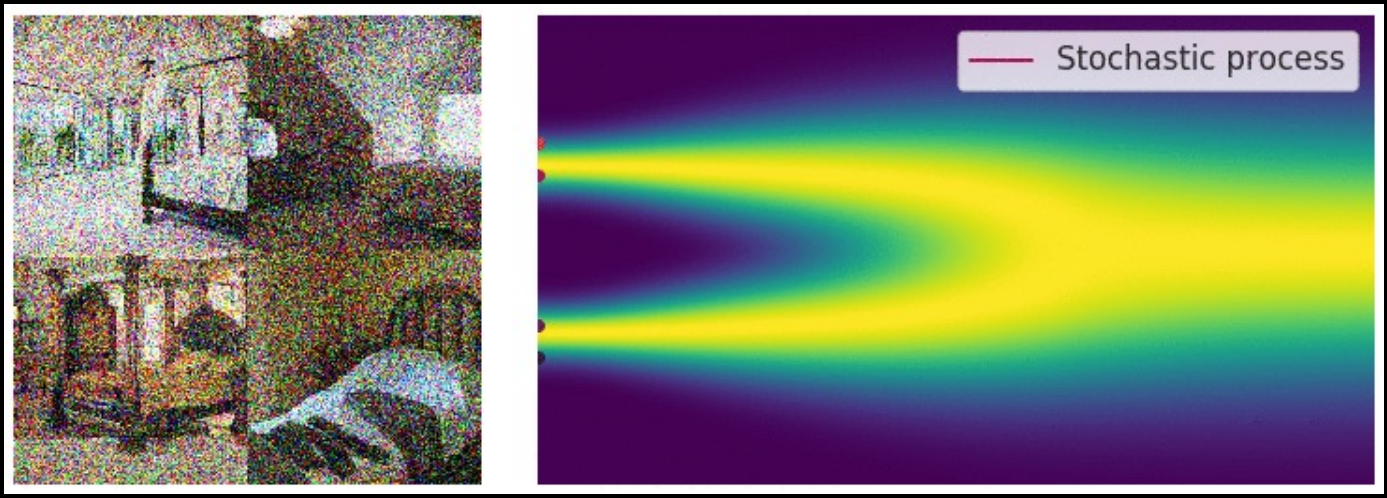
Denoising Diffusion Probabilistic Models (DDPM)

- DDPM = generative models
- Objective: generate data according to a given distribution from random noise iteratively.
- DDPMs rely on two inverse processes: Forward and Backward.
- The DDPM is trained so the backward process matches the forward.



Ho, J., Jain, A., Abbeel, P., 2020. Denoising Diffusion Probabilistic Models, in: NeurIPS.

Forward: perturb data with a SDE



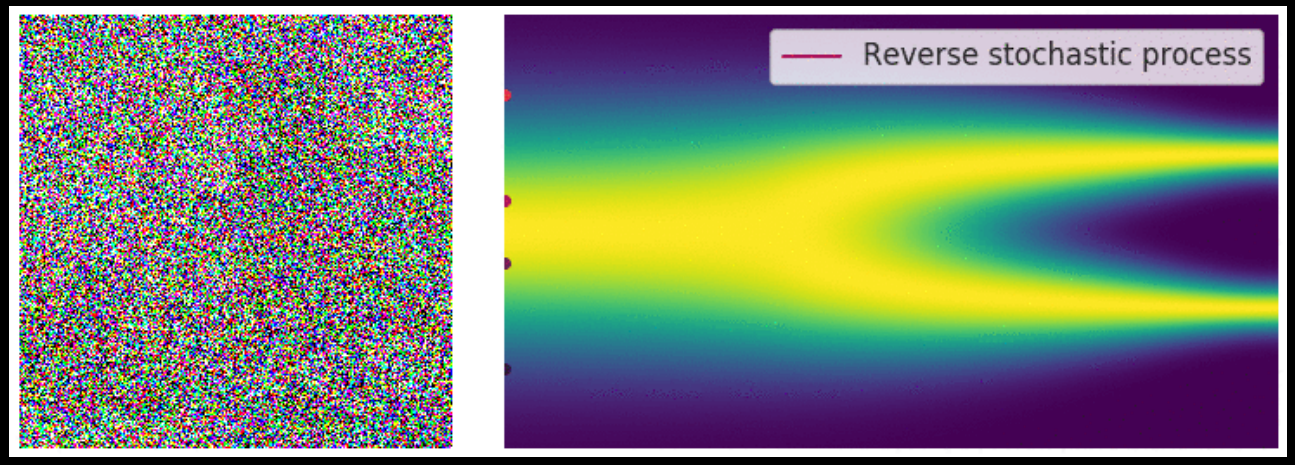
$$\mathbf{dx} = \mathbf{f}(\mathbf{x}, t)\mathbf{d}t + g(t)\mathbf{dw},$$

Handpicked SDE

where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector-valued function called the drift coefficient, $g(t) \in \mathbb{R}$ is a real-valued function called the diffusion coefficient, w denotes a standard Brownian motion, and dw can be viewed as infinitesimal white noise.

https://yang-song.net/blog/2021/score/

Backward: reversing the SDE for sample generation



Reversing the perturbation process with **annealed Langevin dynamics**

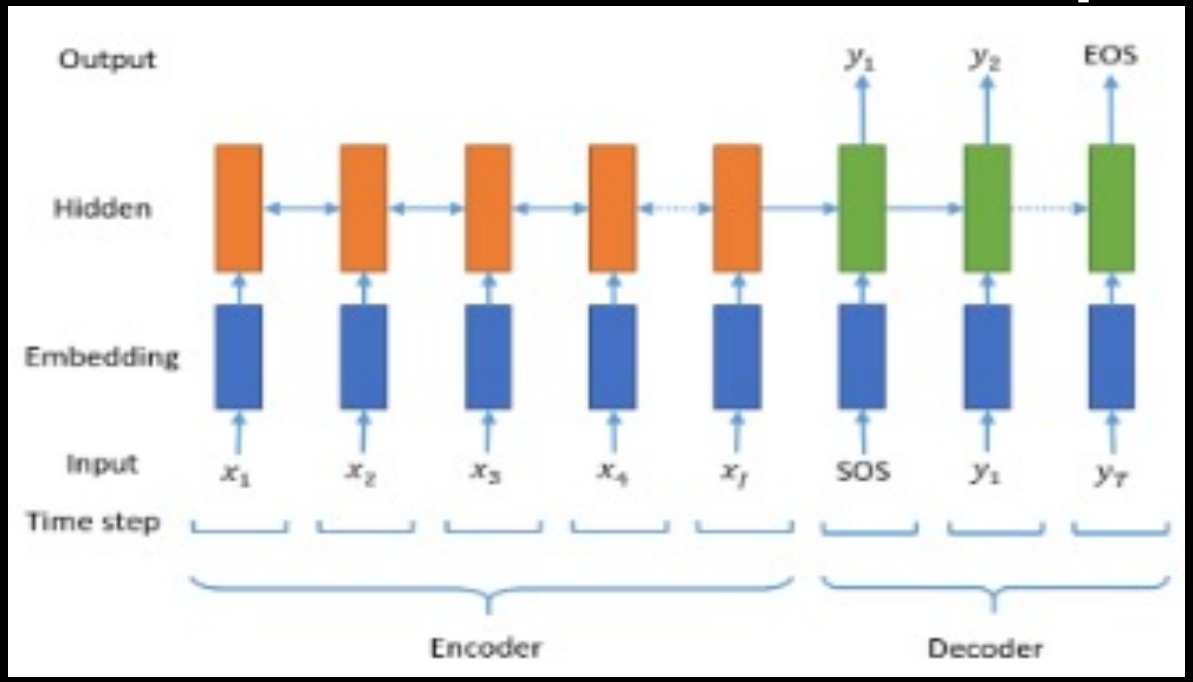
$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}.$$

Closed form for the reverse SDE

SEQ 2 SEQ : THE SPECIAL CASE OF TEXT GENERATION (WITHOUT ATTENTION)

(THANKS TO GUILLAUME GRAVIER)

RNN-based Seq2Seq



The basic seq2seq model. SOS and EOS represent the start and end of a sequence, respectively.

Sequence to sequence encoder/decoder systems combine

- a RNN to encode a message from a prompt/text, i.e., $h_0 = RNN_e(x1,...,xn)$
  ➔ h_0 is the context given to the decoder
- a RNN to generate a message conditioned on h0, i.e., $w1,...,wn = RNNd(h_0)$

from Tian Shi et al., 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models

On practical aspects and use of encoder/decoder

- Often convenient to also consider input sequence backward
  - process from xn to x1
  - use a bidirectional encoder
- Can layer RNNs, both in the encoder and decoder
- Better use ground truth in decoder at training time (or alternate) – aka teacher forcing

But…

- the input message needs be fully summarized in a single embedding h_0 (hence only rather simple inputs work in practice)
- (almost) independent choice of words might lead to poor language
  - might not respect syntax
  - short or truncated outputs
  - repeats

==> need for attention mechanisms !

https://cs.stanford.edu/~zxie/textgen.pdf

Try it out

IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

- TensorFlow (Google): https://www.tensorflow.org/
  - Python, deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- PyTorch (Facebook/ Twitter, Deepmind): http://torch.ch/
  - Python