Imperial College
London

Convolutional Neural Networks

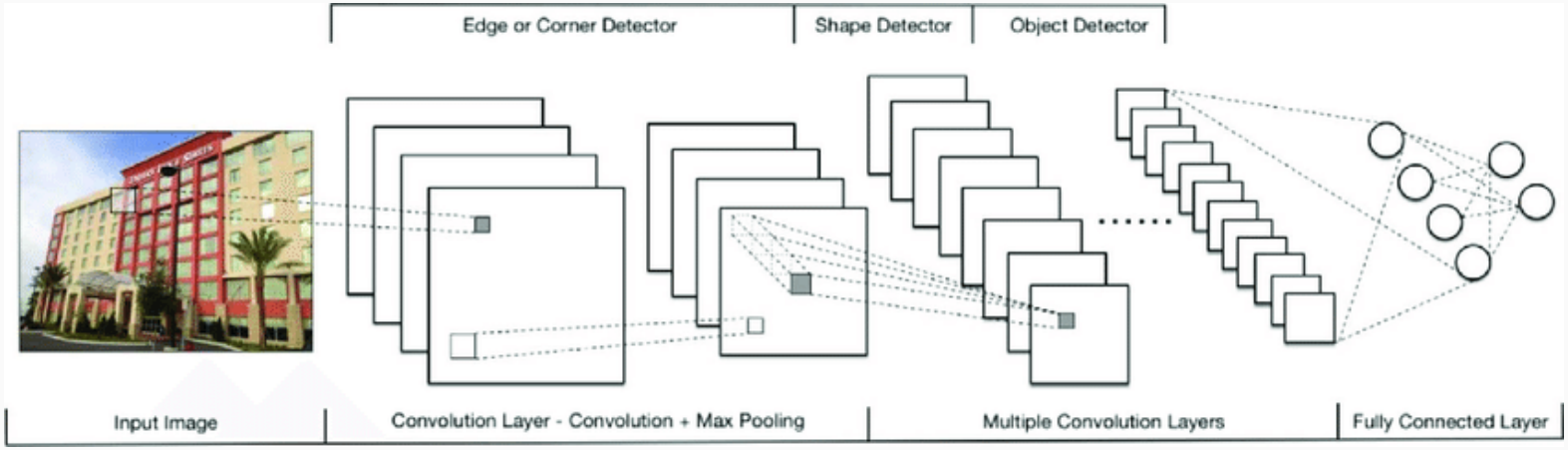
Deep Learning - MLSS 2019

Pierre Harvey Richemond, Kevin Webster

Outline

Outline

Convolutional neural networks



- **Convolutional neural network**, **CNN** or **ConvNet** is a class of neural network with a special structure
- Breakthrough improvements in image processing
- Also used in NLP, audio waveform analysis and generation, and RL models for games

Convolution operation

The convolution operator

$$(f * w)(t) := \int_{-\infty}^{\infty} f(\tau) w(t - \tau) d\tau$$

can be described as a weighted average of the **input** f according to the weighting (or **kernel**) w at each point in time t.

The discrete convolution is given by

$$(f * w)(t) := \sum_{\tau=-\infty}^{\infty} f(\tau) w(t - \tau).$$

Note that when w has a finite support, a finite summation may be used.

Convolution operation

In convolutional neural networks with image inputs, a two dimensional convolution is used:

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

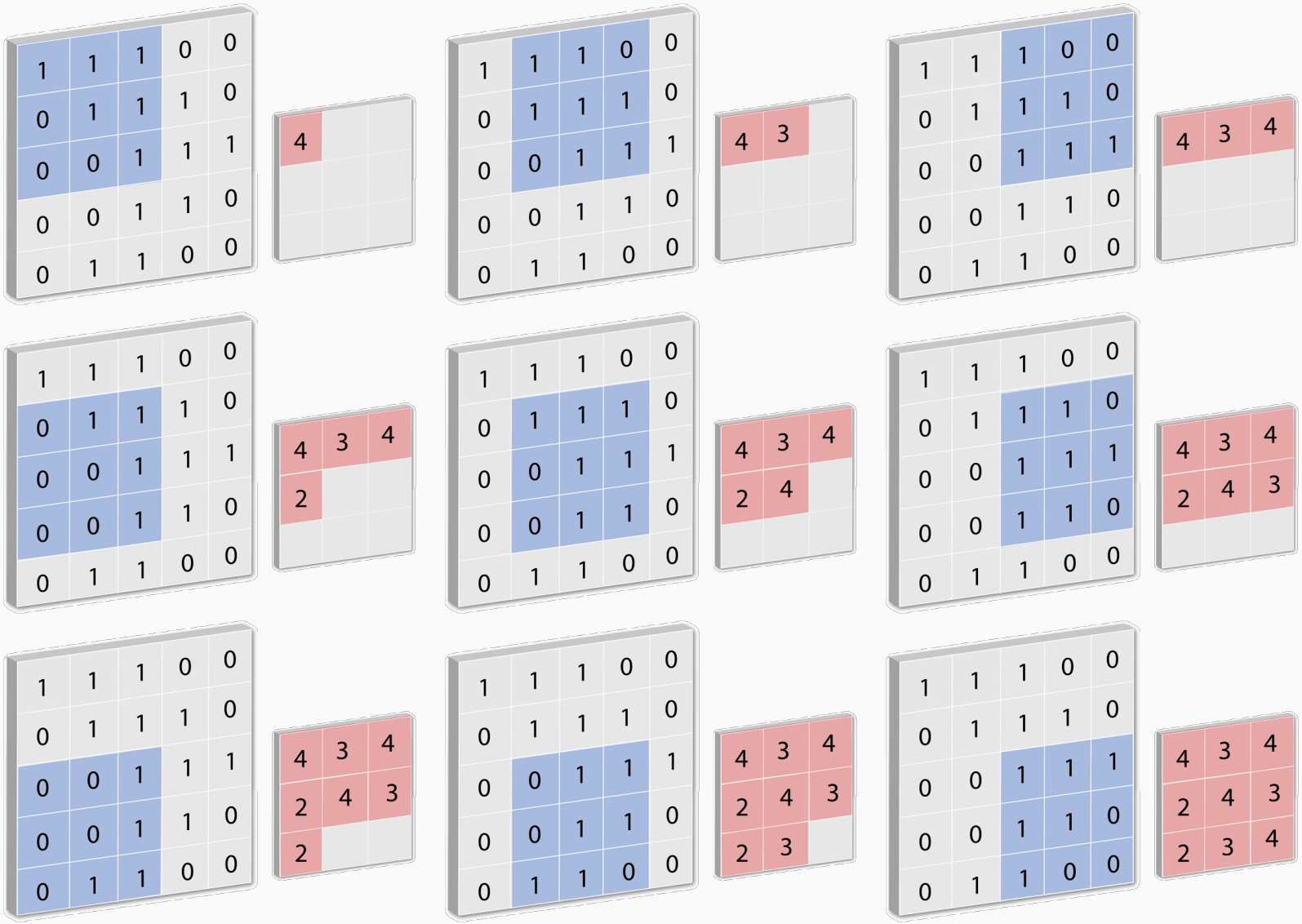
The kernel K will have a fixed size, outside which is can be assumed to be zero.

In practice, many libraries implement the cross-correlation operation, which is the same as above but changing the orientation of the arguments:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

We will consider the above operation in CNNs and refer to it as a **convolution**.

Convolution operation



Convolution operation

- The **kernel** is also referred to as a **filter**
- The support of the kernel defines its **receptive field**
- Output of a convolution is often referred to as a **feature map**
- There are usually many feature maps defined for each layer
- Within each layer, there are several **channels** (e.g. there are 3 channels in the input layer for an RGB image)
- In practice, the kernels for each feature map are usually three dimensional tensors (esp. for images) and the convolution operation is performed across all channels in the input layer:

$$S^l(i,j) = \sum_m \sum_n \sum_c I(i+m, j+n, c) K^l(m,n,c)$$

- The convolution is usually combined with a shared bias (one per channel) and passed through an activation function

Convolution properties

- Design inspired by visual processing systems
- **Sparse interactions**: a feedforward network connects every input to every output. The sparse connectivity of CNNs reduces number of parameters significantly and improves efficiency
- **Parameter sharing**, or tied weights, means that the CNN does not need to learn a separate set of parameters for each location, but reuses one set in every location
- **Equivariant features**: due to the parameter sharing used in CNNs, the feature maps are equivariant with respect to translations. (However note that convolutions are not equivariant with respect to other transformations such as rotation.)
- Also enables some flexibility in input size

Padding

In general, we can also define the convolution operation for N-dimensional inputs. The kernel/filter is then a tensor of shape $(n, m, k_1, \ldots, k_N)$, where

$$\begin{aligned} n &= \text{number of output maps} \\ m &= \text{number of input maps} \\ k_j &= \text{kernel size in dimension } j \end{aligned}$$

The output size o of a convolution along a dimension with input size i and kernel size k is given by

$$o = (i - k) + 1.$$

Padding

The output size is frequently controlled by padding the input with zeros, effectively increasing the input size.

- **'SAME'** (or half) padding adds $p = k - 1$ zeros to the input to constrain the output size to be the same for unit strides, $o = i$. For odd-sized kernels, add $p = \lfloor k/2 \rfloor$ zeros both sides of the input
- **'VALID'** padding is the standard terminology for when no padding is used
- **'FULL'** padding adds $p = 2(k - 1)$ zeros to the input ($k - 1$ zeros on both sides), resulting in an output size of $i + (k - 1)$

Strides

Convolutions may also use a **stride** s, which is the distance between consecutive positions of the kernel (the convolutions considered so far use $s = 1$).

In this case, the output size is given by

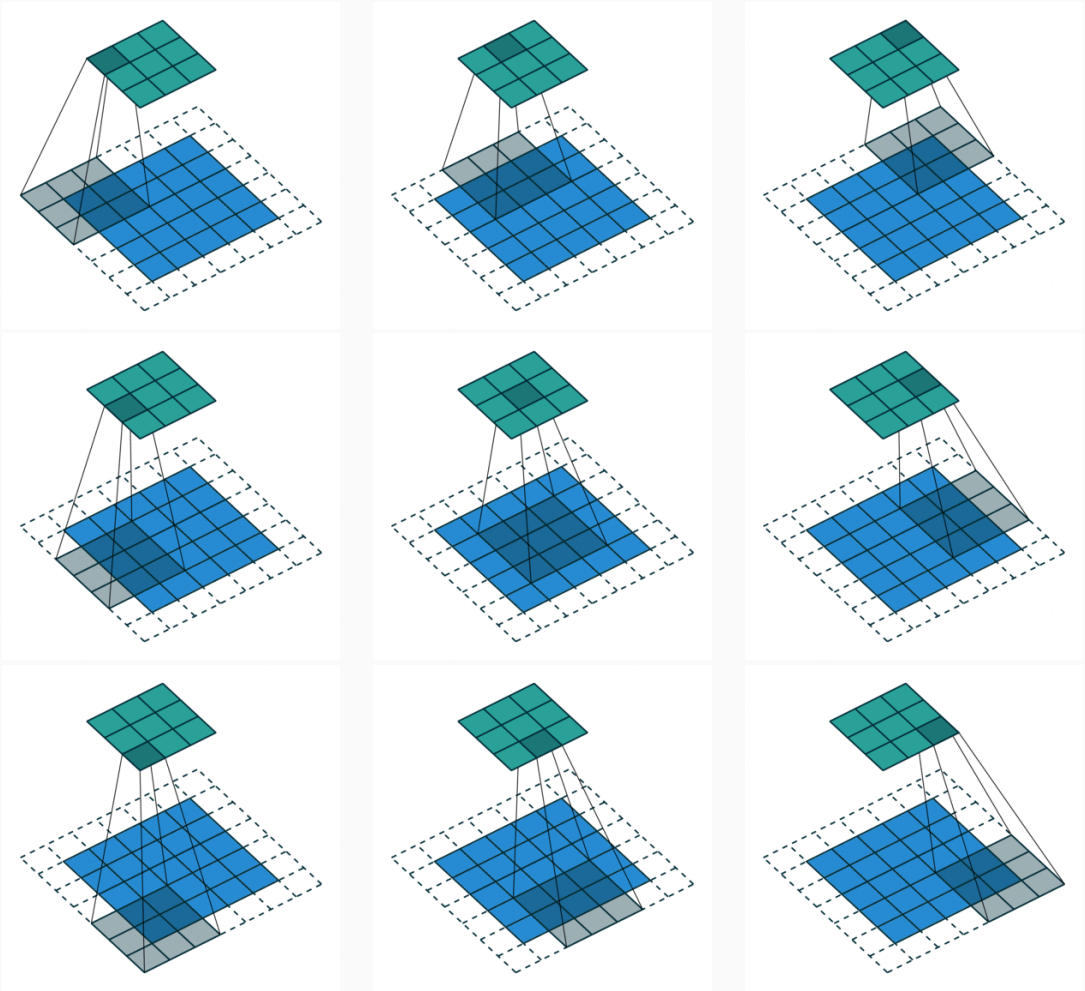
$$o = \left\lfloor \frac{i + p - k}{s} \right\rfloor + 1$$

Thus, strides are one way of *downsampling* within a convolutional neural network.

Note that when using strides $s > 1$, different input sizes can still lead to the same output size.

Strides

Example: 3×3 convolution with half ('SAME') padding $p = 2$, stride $s = 2$, $i = 5 \quad \Rightarrow \quad$ output size $o = 3$.



Pooling

In typical CNNs, convolutional layers are interleaved with **pooling** layers. These layers compute a summary statistic over regions of the input space identified by a sliding window.

It can be thought of as being similar to the convolution operation, where the linear transformation is replaced with a pooling operation.

- **Max pooling** computes the maximum activation per channel in the window
- **Average pooling** computes the average activation per channel in the window
- **L^2 norm** computes the 2-norm of activations per channel in the window

Pooling

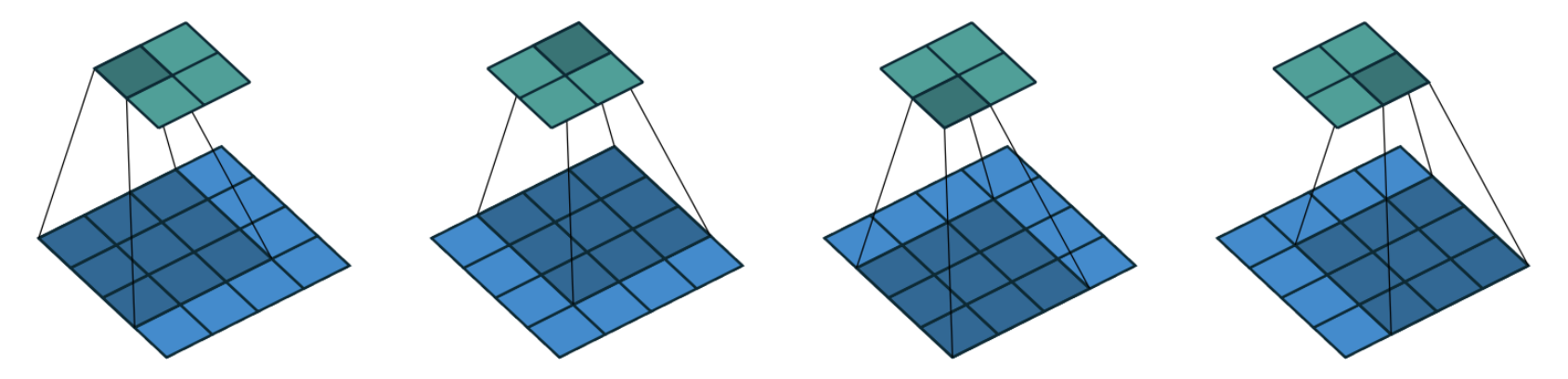
- The pooling operation introduces a degree of *translation invariance* to the input
- This is appropriate when we do not need to know the exact location of a feature, just that it is there
- It can be thought of as a prior assumption that the learned function must have translation invariance
- Pooling is often used with strided windows, leading to downsampling of the input
- We can also pool over the outputs of separate feature maps, allowing the network to learn which transformations to become invariant to
- Pooling output can also be computed as $o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1$
- Pooling is useful for handling inputs of varying size

Transposed convolutions

- Also called *fractionally strided convolutions* or *deconvolutions*
- The typical convolution $\rightarrow$ pooling combination leads to spatial downsampling.
- Sometimes we need to do the opposite: transposed convolutions provide one way to do this
- For example, we might want to construct a convolutional decoder as part of an autoencoder
- Transposed convolutions are intended to be the analogue to transposing the weight matrix in the fully connected case

Transposed convolutions

Consider the following as a simple example case:



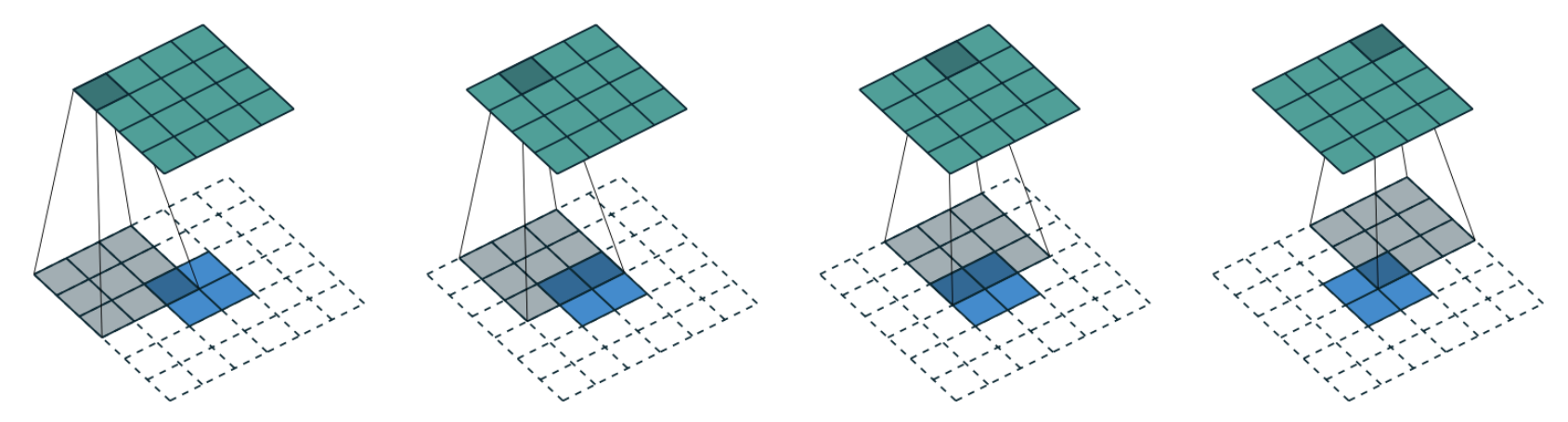
We can 'unroll' this operation to obtain the sparse matrix

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Transposed convolutions

- This 4×16 matrix flattens the input layer and the output is then reshaped to a 2×2 matrix
- In the backward pass, the error is backpropagated by multiplying the loss by the transpose of this matrix
- The connectivity pattern is compatible with the original convolution by construction
- Transposed convolutions essentially swap the forward and backward passes of a convolution

Transposed convolutions as convolutions



- Note that it is possible to emulate a transposed convolution with a direct convolution by adding zeros to the input
- The above convolution is equivalent to the transpose of the previous convolution example
- Note that the kernel size and stride are the same, but the input is padded with zeros
- The connectivity pattern is the same

Transposed convolutions as convolutions: unit strides

- In the case of unit strides ($s = 1$), consider a convolution with padding p, kernel size k and input size i
- It has an associated *transposed convolution* with $k' = k$, $s' = s$ and $p' = 2(k-1) - p$. Its output size is given by

$$o' = i' + (k-1) - p$$

- No padding of the convolution input leads to full padding for the transposed convolution and vice versa
- Half ('SAME') padding of the convolution input leads to half padding for the transposed convolution

Batch Norm in CNNs

- Recall BN normalises each feature map separately
- For CNNs we want to respect the structure and normalise over all spatial locations
- In particular, an input in a 2D ConvNet will have shape $[N, H, W, C]$, where
  - N is the number of examples in the minibatch
  - H and W are image height and width respectively
  - C is the number of channels
- Standard BN would compute $H \times W \times C$ means and stddevs to normalise each feature separately at each spatial location
- BN in ConvNets instead computes C means and stddevs and normalises jointly for every spatial location

Outline

CNN architectures

- Certain CNN architectures have contributed to significant progress in image recognition
- ImageNet is a standard dataset to compare networks against each other
- ImageNet has been running an annual competition since 2010: the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
- 1.2 million images with 1000 classes for recognition

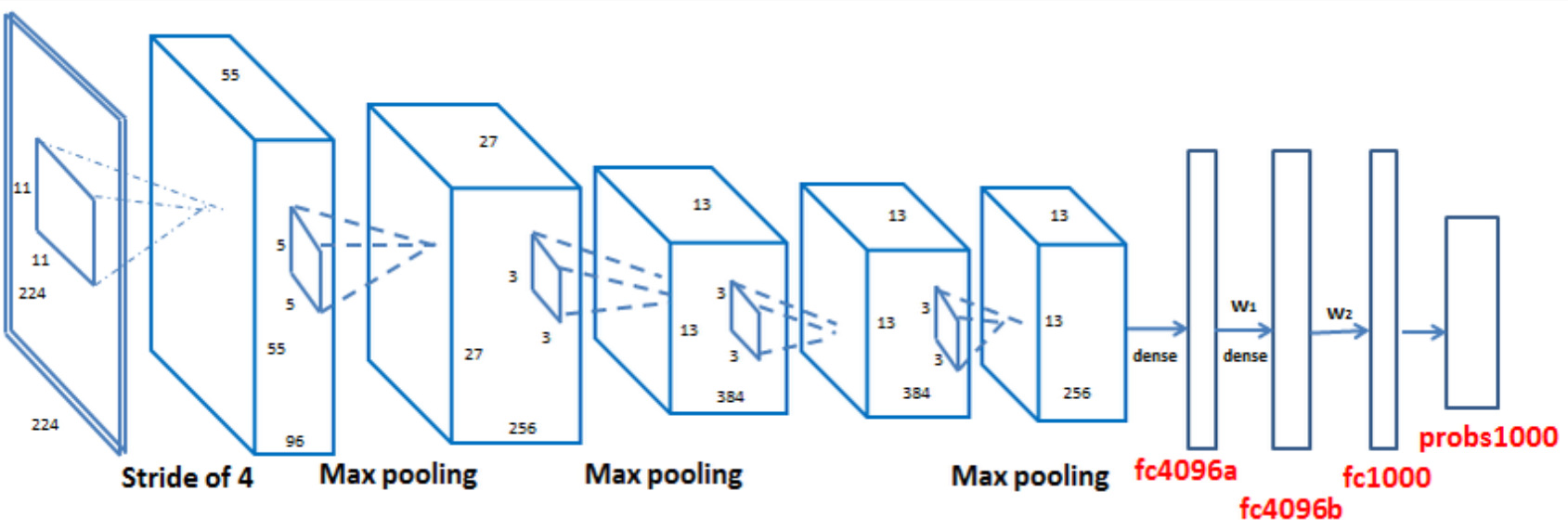
CNN architectures and ILSVRC

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

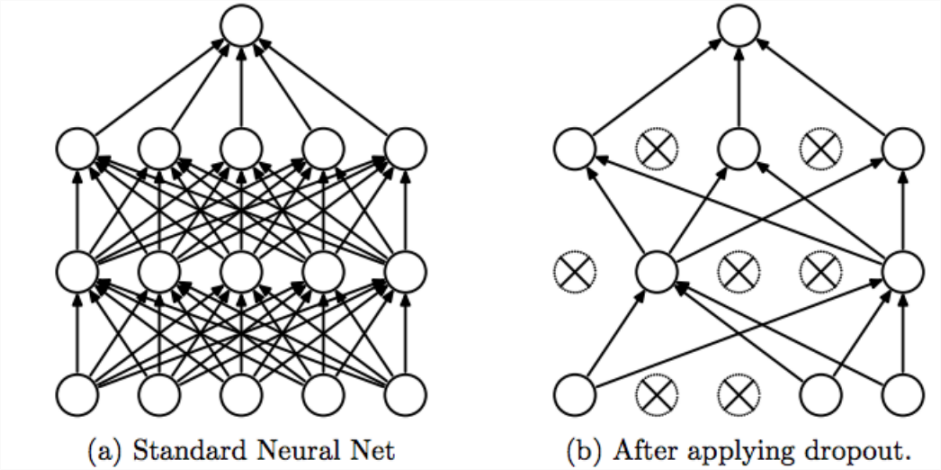
AlexNet

- AlexNet was published in 2012, and competed in the ImageNet competition in the same year
- Achieved top-5 error rate of 15.3%, more than 10.8% lower than that of the runner up
- One of the first deep networks to significantly outperform traditional methodologies in image classification
- Composed of 5 convolutional layers followed by 3 fully connected layers
- Used ReLU for the non-linear activations, instead of a tanh or sigmoid
- Reduced over-fitting by using a dropout layer after each fully connected layer

AlexNet



AlexNet architecture

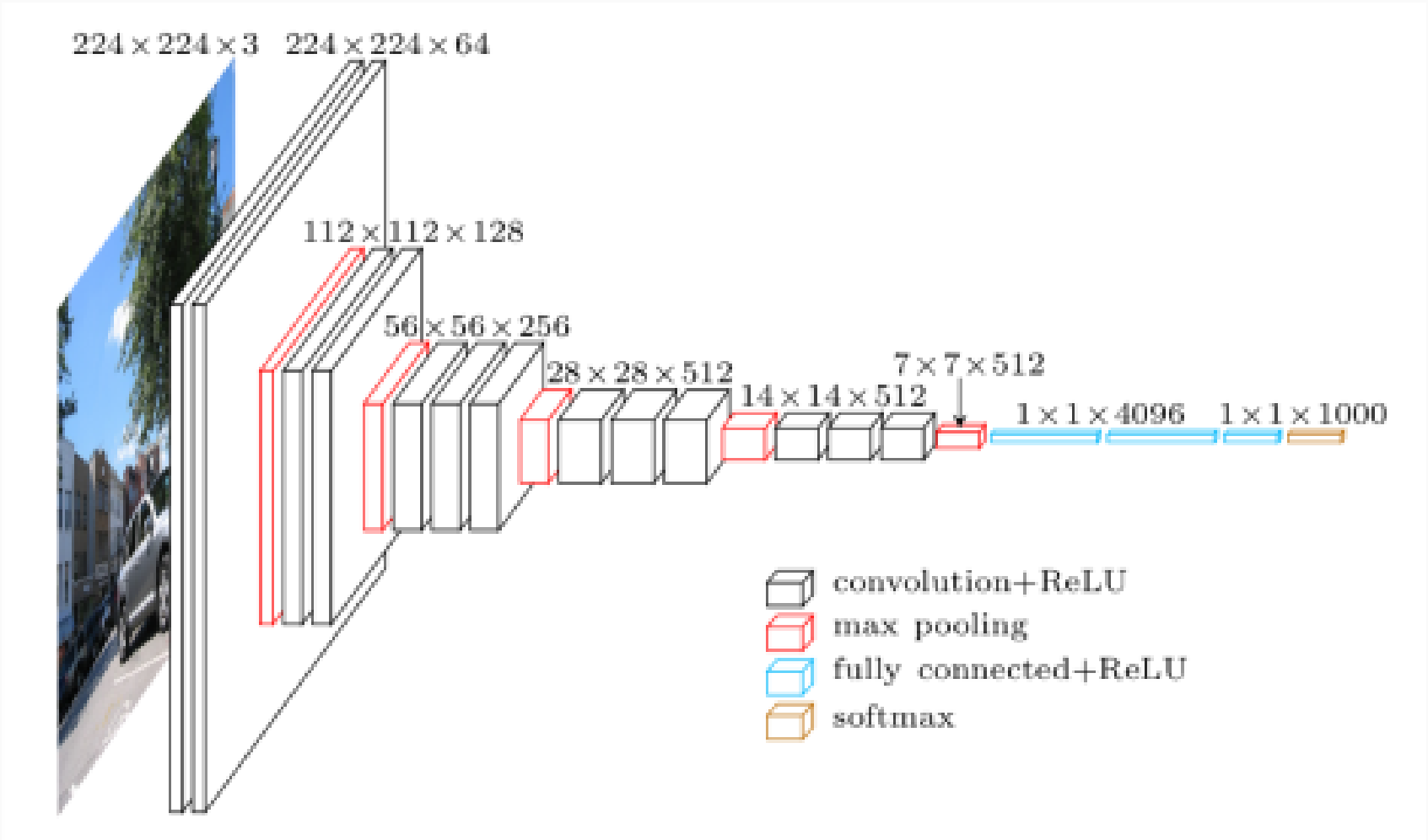


(a) Standard Neural Net (b) After applying dropout.

VGG

- Developed by the VGG (Visual Geometry Group) in Oxford
- Improves AlexNet by replacing the large filters in the first two layers by multiple 3x3 filters
- Multiple stacked smaller size kernels allows the network to learn more complex features
- Also reduces number of parameters
- VGG convolutional layers are followed by 3 fully connected layers
- Width of the network (number of channels) starts small at 64 and increases by a factor of 2 after every pooling layer
- Achieved the top-5 error rate of 7.3% on ImageNet in 2014

VGG-16


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Residual networks

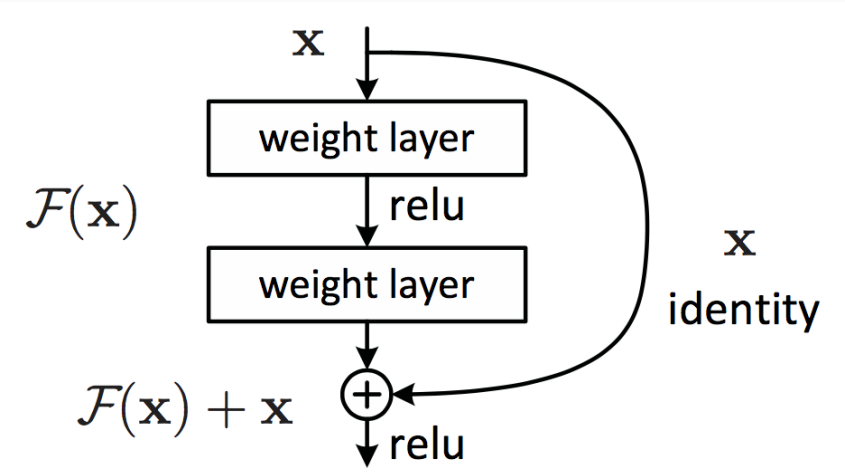
- Previous developments show increased depth increases capacity
- Backpropagated signal becomes vanishingly small with large depth
- Increased parameters also present optimisation problems
- Proposed solution is to introduce **residual blocks** into the architecture, that use shortcut connections to propagate the signal
- Assumption is that the optimal learned function is closer to identity than to zero
- Similar to GoogLeNet, uses a global average pooling followed by the classification layer
- ResNets were learned with network depth of as large as 152
- Similar to the VGGNet, consisting mostly of 3X3 filters
- ResNet-152 achieved 3.6% top-5 error rate on ImageNet in 2015, surpassing human performance

Residual block

- Residual blocks add the input to the output:

$$\mathbf{y} = \sigma(\mathcal{F}(\mathbf{x}) + \mathbf{x}),$$

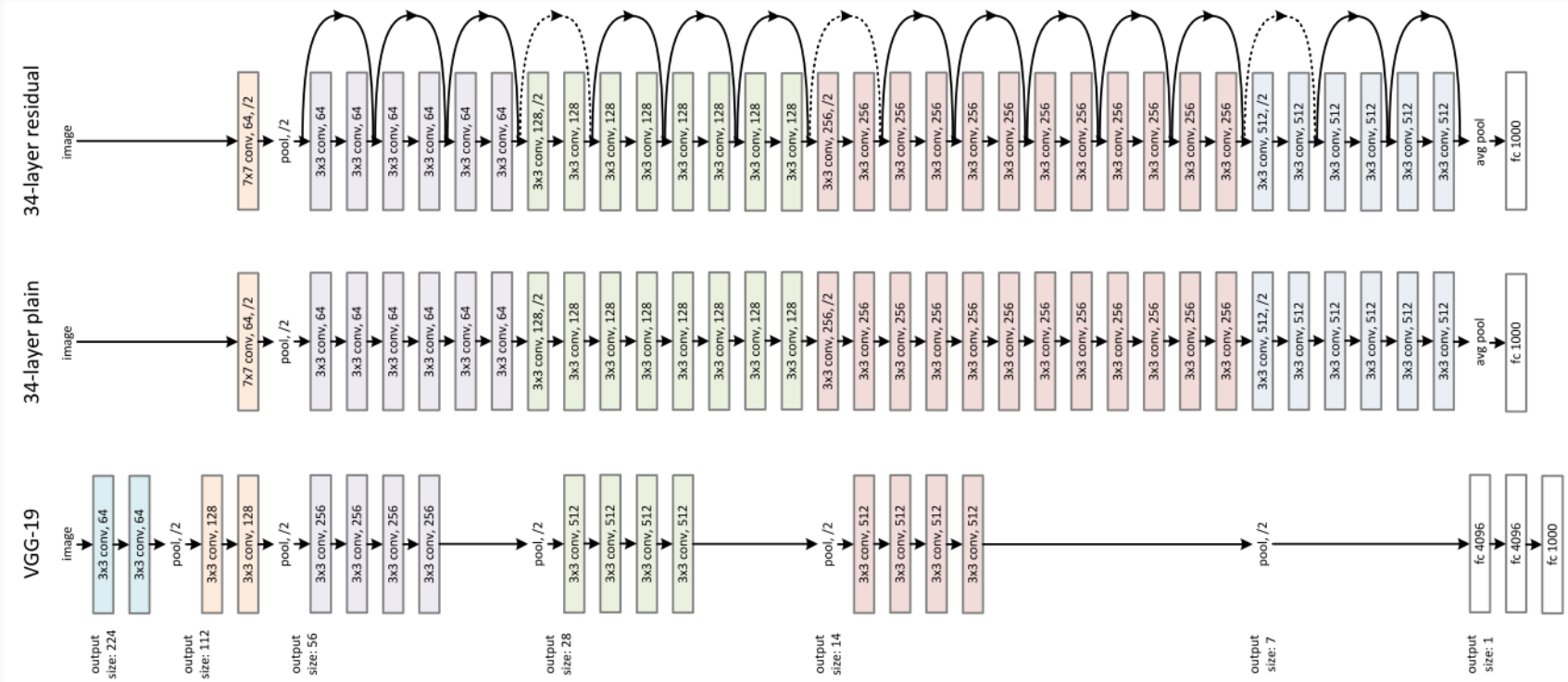
where $\mathcal{F}(\mathbf{x}) = W_2\sigma(W_1\mathbf{x} + b_1) + b_2$



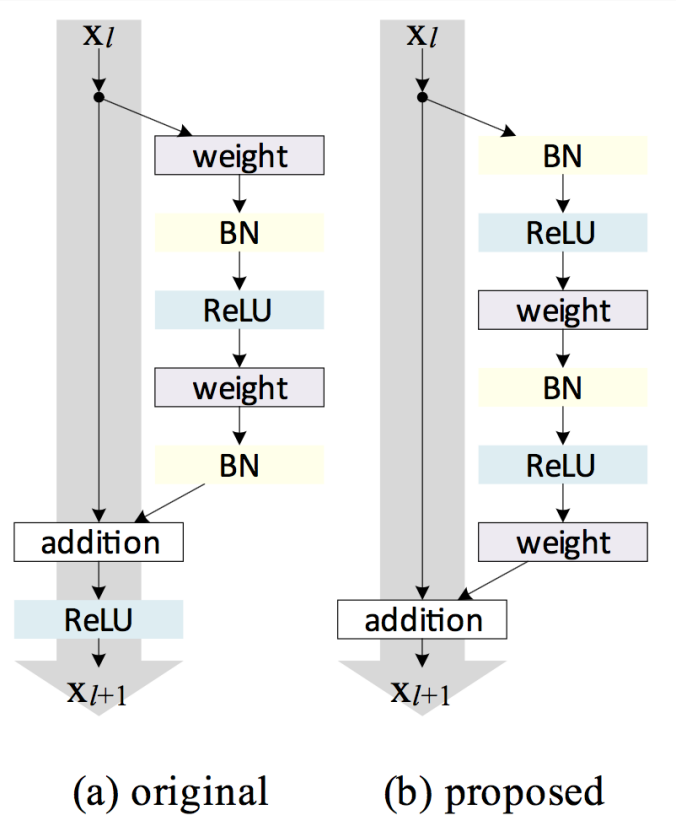
- Two layers are included inside the block, since one layer would be too close to a linear transformation:

$$\tilde{\mathbf{y}} = \sigma(W_1\mathbf{x} + b_1 + \mathbf{x})$$

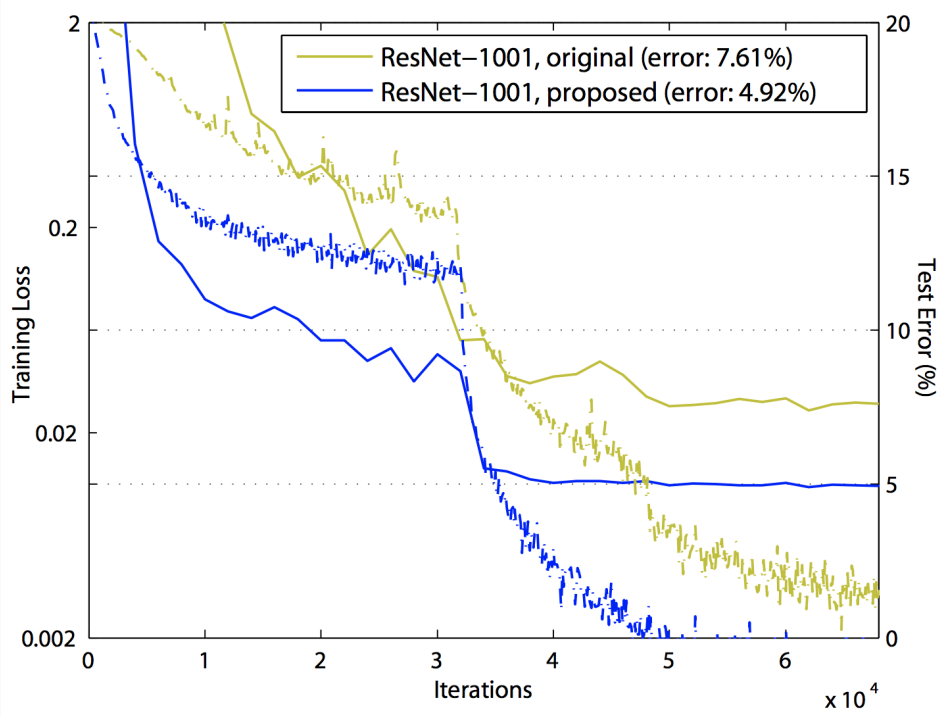
ResNet, plain, VGG-19



Revised ResNet (2016)



Original and revised ResNet blocks



Training curves on CIFAR-10 of 1001-layer ResNets. Solid lines denote test error, and dashed lines denote training loss

References i


He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Deep residual learning for image recognition.
CoRR, abs/1512.03385.


He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Identity mappings in deep residual networks.
CoRR, abs/1603.05027.


Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.
In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

References ii


LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989).
Backpropagation applied to handwritten zip code recognition.
Neural Computation, 1(4):541–551.


LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010).
Convolutional networks and applications in vision.
In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, pages 253–256.


Simonyan, K. and Zisserman, A. (2014).
Very deep convolutional networks for large-scale image recognition.
CoRR, abs/1409.1556.

References iii


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014).
Going deeper with convolutions.

Imperial College London

Sequence Modelling

Deep Learning - London MLSS 2019

Pierre Harvey Richemond, Kevin Webster

Outline

Outline

Example sequence modelling problems

- **Speech recognition**

$\Rightarrow$ "These aren't the droids you're looking for"

- **Machine translation**

"I want forty kilograms of persimmons" $\Rightarrow$ "Ich will vierzig Kilogramm Persimonen"

- **Question answering**

"What sport is this?" $\Rightarrow$ "Baseball"

Example sequence modelling problems

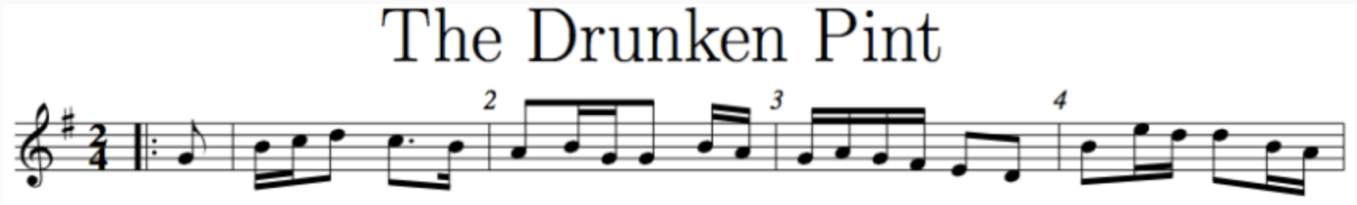
- **Sentiment analysis**

$\Rightarrow$ Tweet analyzed as positive for Comfort

- **Anomaly detection**

$\Rightarrow$ "Robbery"

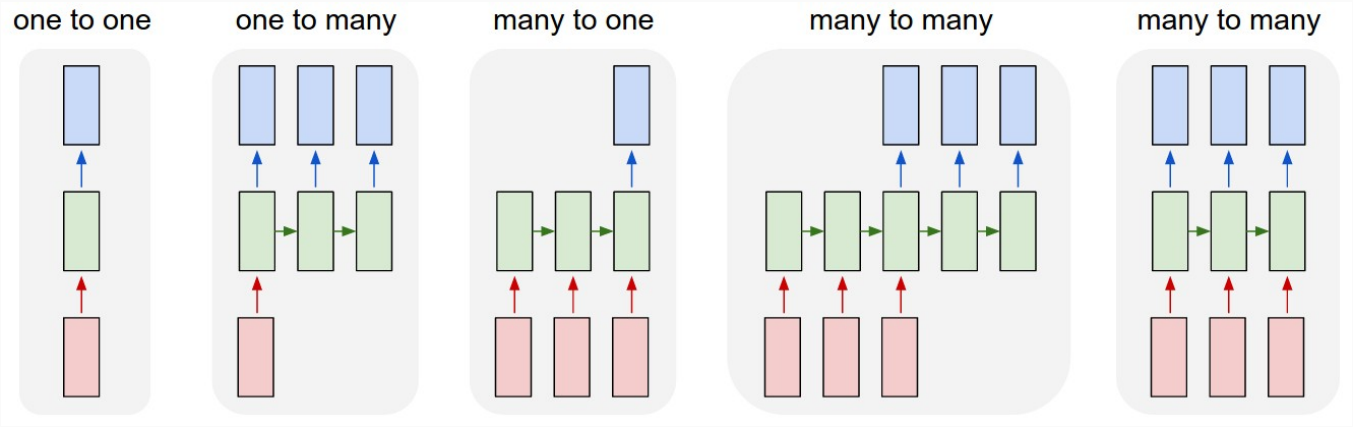
- **Music generation**



Outline

Recurrent Neural Networks

- Feedforward / MLP networks are constrained to a fixed size input and output, and fixed number of hidden layers
- Recurrent Neural Networks (RNNs) are designed to handle sequential data
- They allow flexibility in the lengths of inputs and outputs
- Similar to ConvNets, they also use weight sharing for learning features across the sequence



Flexibility of RNN architectures. Red: inputs, green: hidden states, blue: outputs.

Recurrent Neural Networks

Main idea : implement

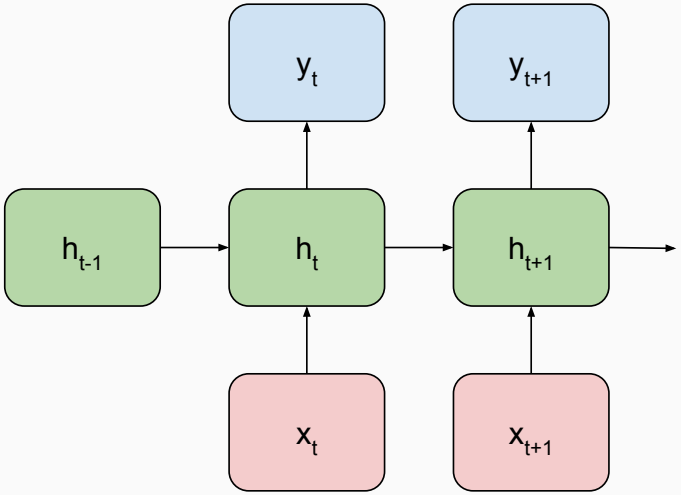
$$y_t = f(x_t, h_t)$$

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

with f and g neural networks whose weights will be learned.

Recurrent Neural Networks

Basic RNN computation for inputs $x_t \in \mathbb{R}^{n_{in}}$ and outputs $y_t \in \mathbb{R}^{n_{out}}$:



$$
\begin{aligned}
h_t &= \sigma(W^{(hh)} h_{t-1} + W^{(xh)} x_t + b_h), \\
y_t &= W^{(hy)} h_t + b_y,
\end{aligned}
$$

where σ is an activation function, $h_t \in \mathbb{R}^{n_h}$ is the hidden state, $W^{(hh)} \in \mathbb{R}^{n_h \times n_h}$, $W^{(xh)} \in \mathbb{R}^{n_h \times n_{in}}$, $W^{(hy)} \in \mathbb{R}^{n_{out} \times n_h}$, $b_h \in \mathbb{R}^{n_h}$ and $b_y \in \mathbb{R}^{n_{out}}$.

The output could also be passed through e.g. a softmax layer.

RNN properties

- In theory, RNNs model sequences using all information from the past (cf. Markov models)
- RNNs use a distributed hidden state that allows them to store a lot of information (cf. HMMs)
- Nonlinear transformations allow them to update the hidden state in complicated ways
- The internal dynamics of the RNN is deterministic (although the output can be made to be stochastic)
- Models can be made more powerful by stacking extra hidden layers
- Note that there is the issue of the initial hidden state: in practice the initial state can either be learned in the same way as the weights, or simply set to the zero vector

Backpropagation through time

- We can think of the RNN as a layered, feedforward network with shared weights
- Training RNNs also uses the backpropagation algorithm
- In the case of RNNs, we can think of the forward and backward passes stepping through time
- After the backward pass we add the derivatives at all the different times for each weight to preserve weight sharing:

$$
\begin{aligned}
\text{To constrain:} \quad & w_1 = w_2 \\
\text{We need:} \quad & \Delta w_1 = \Delta w_2 \\
\text{Compute:} \quad & \frac{\partial L}{\partial w_1} \text{ and } \frac{\partial L}{\partial w_2} \\
\text{Use:} \quad & \frac{\partial L}{\partial w_1} + \frac{\partial L}{\partial w_2} \text{ for both } w_1 \text{ and } w_2
\end{aligned}
$$

Backpropagation through time

- In practice, training sequences may be very long and/or be of different lengths
- It may be necessary to truncate the sequence lengths used for training (truncated backpropagation through time)
  - Longer sequences are split into shorter subsequences for training
  - The internal RNN state can be carried over between subsequences of a full sequence
  - It is also possible to separate the number of steps in the forward and backward pass
- Shorter sequences could be zero padded to fit into a minibatch tensor for training

Truncated Backpropagation through time

1. Present a sequence of k_1 timesteps of input and output pairs to the network.
2. Unroll the network (every copy of the network shares the same parameters), then calculate and accumulate errors across k_2 timesteps.
3. Roll-up the network and update weights.
4. Repeat.

Vanishing and exploding gradients

- Recall from the backpropagation calculation that gradients can vanish or explode backwards through the layers[1]:

$$\delta_i = \left(\prod_{k=i}^{N-1} \Sigma'(\hat{h}_k)(W^{(k)})^T \right) \Sigma'(\hat{h}_N) \nabla_{h_N = \mathbf{y}} L.$$

- This problem is especially bad in recurrent networks that are trained on long sequences (e.g. 100 time steps)
- Good weight initialisation can mitigate this to an extent
- Learning unitary weight matrices should also help
- Still in general, RNNs struggle with long-range dependencies

[1] [Hochreiter, 1991]

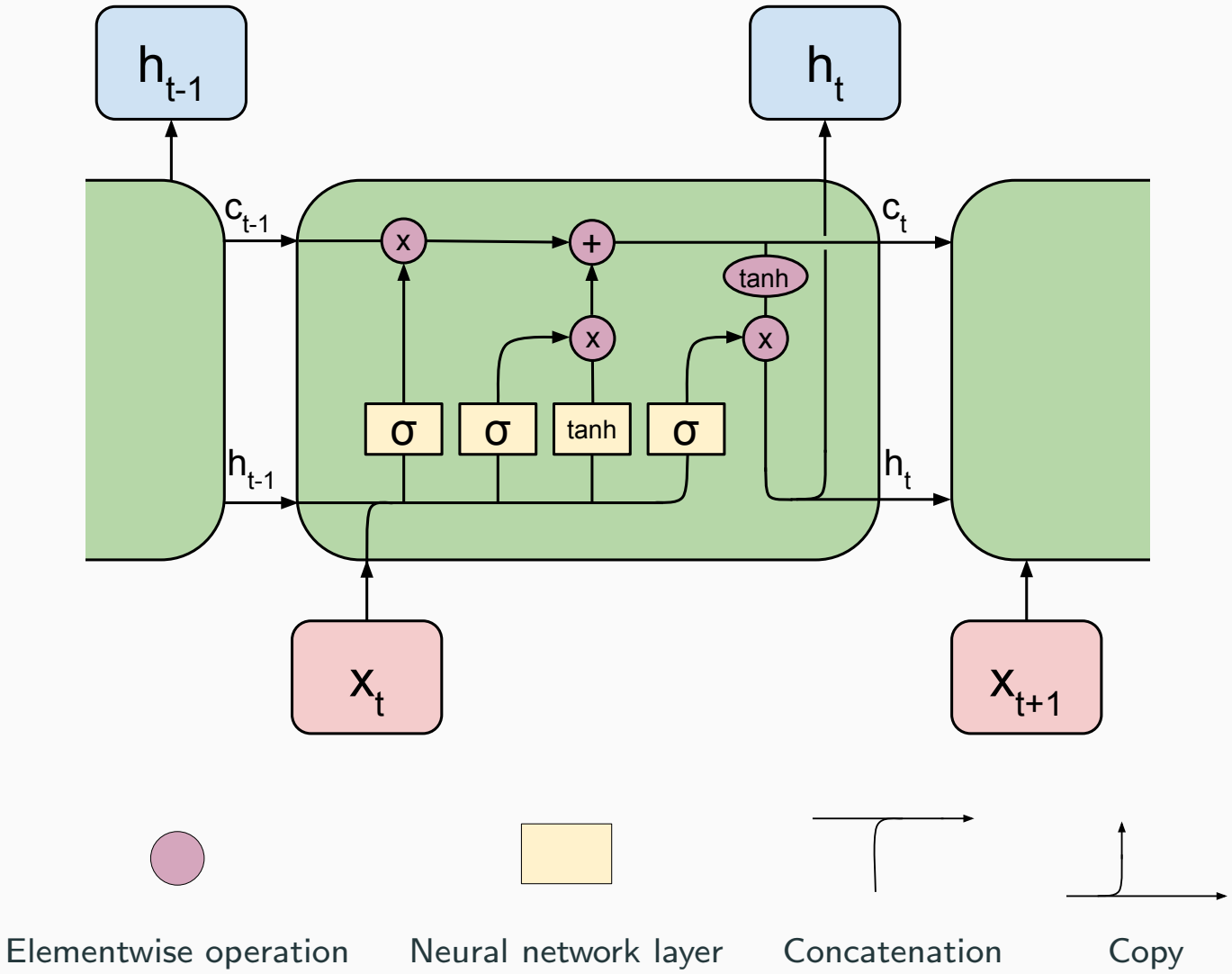
Long Short Term Memory

- The Long Short Term Memory (LSTM) network was introduced[2] to treat the problem of vanishing gradients and enable the network to remember things for a long time
- The LSTM cell has inputs x_t and h_{t-1} and calculates h_t as before
- However, it also includes an internal cell state c_t that allows the unit to store and retain information
- It uses a gating mechanism consisting of logistic and linear units with multiplicative interactions:
  - Information is allowed into the cell state when the 'write' gate is on
  - Information stays in the cell state when the 'keep' gate is on
  - Information can be read from the cell state when the 'read' gate is on

[2] [Hochreiter and Schmidhuber, 1997]

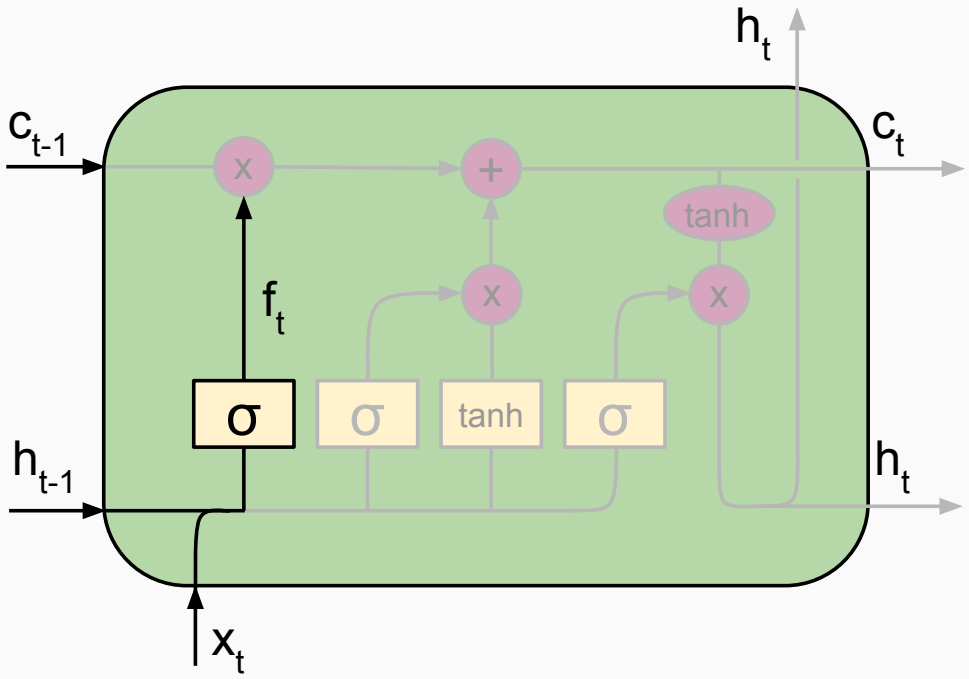
Long Short Term Memory

Schematic diagram for the LSTM unit:



Long Short Term Memory

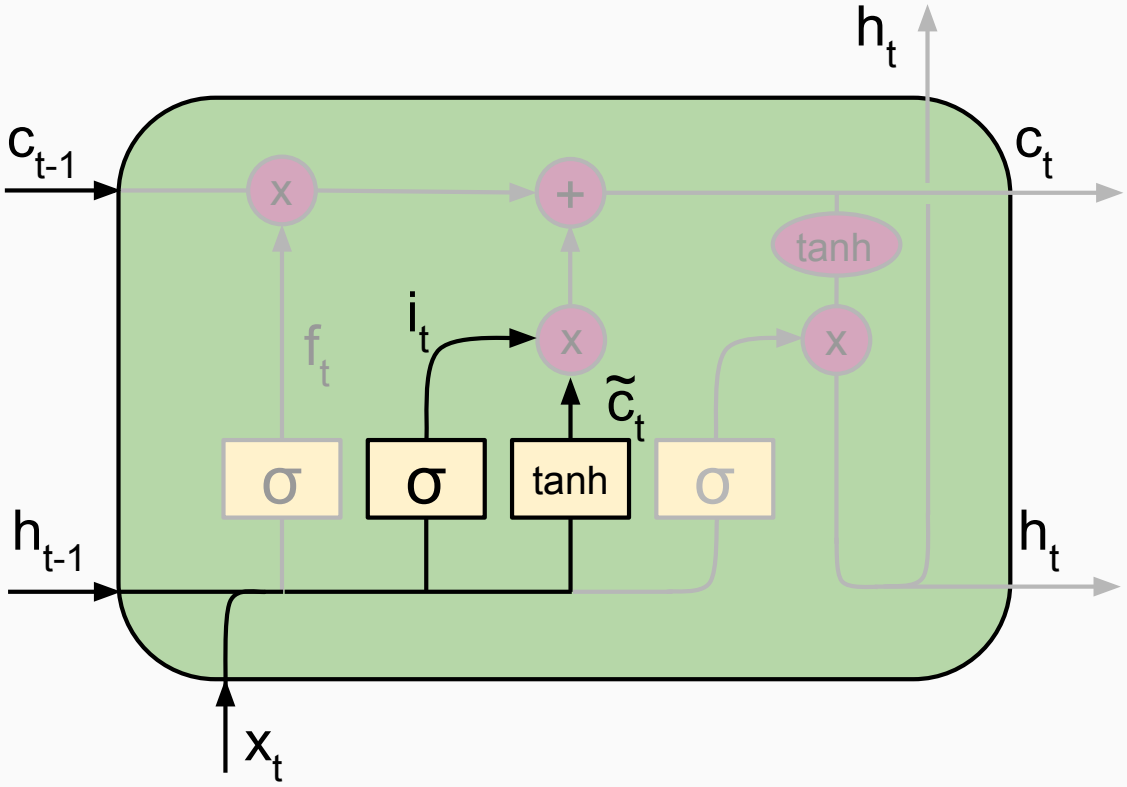
- The LSTM unit can be understood as a combination of gating mechanisms



- The *forget* gate determines what should be erased from the cell state:

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

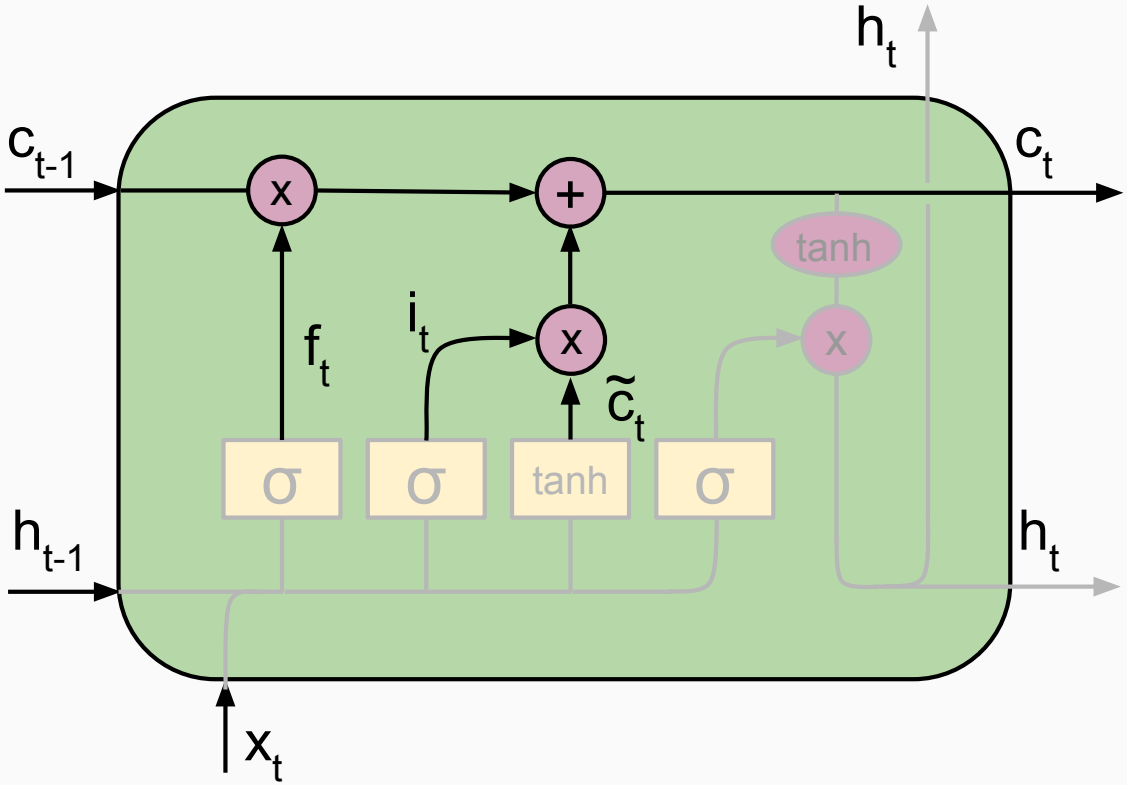
Long Short Term Memory



- The *input* gate decides which values to update in the cell state, with the candidate content given by $\tilde{c}_t$:

$$
\begin{aligned}
i_t &= \sigma(W^{(i)}.[x_t, h_{t-1}] + b_i) \\
\tilde{c}_t &= \tanh(W^{(c)}.[x_t, h_{t-1}] + b_c)
\end{aligned}
$$

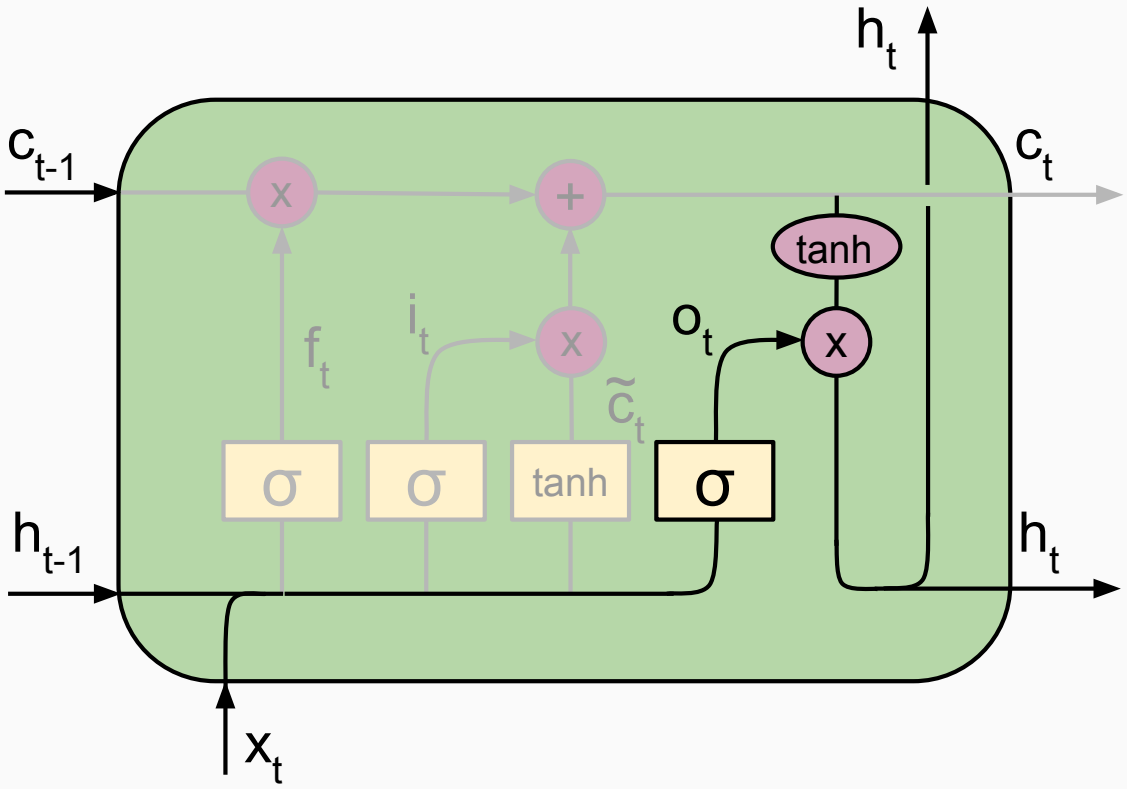
Long Short Term Memory



- The cell state is then updated using the output of the forget, input and content gates:

$$c_t = c_{t-1} \odot f_t + i_t \odot \tilde{c}_t$$

Long Short Term Memory



- Finally, the *output* gate decides which cell state values should be output in the hidden state:

$$o_t = \sigma(W^{(o)}.[x_t, h_{t-1}] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day When little srain would be attain'd into being never fed, And who is but a chain and subjects of his death, I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul, Breaking and strongly should be buried, when I perish The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and my fair nues begun out of the fact, to be conveyed, Whose noble souls I'll have the heart of the wars.

LSTM algebraic geometry

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciely in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M^\bullet} = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

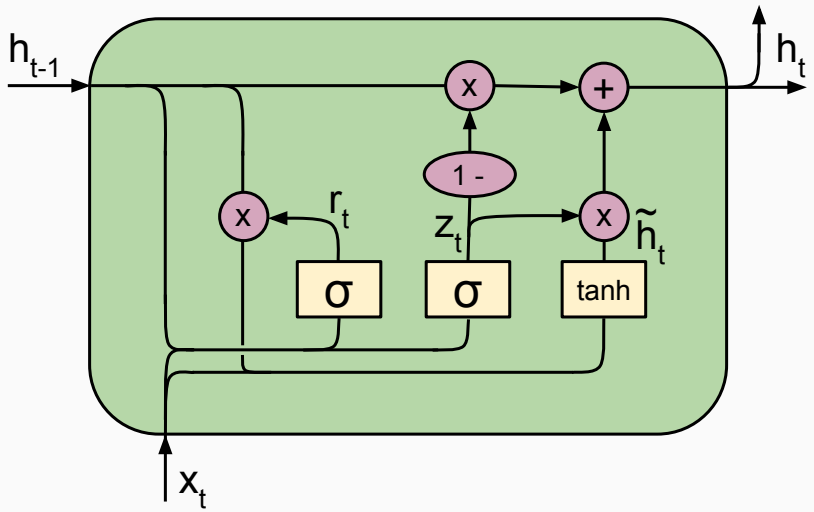
Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Gated Recurrent Unit

The Gated Recurrent Unit (GRU)[3] is a variation on the same idea. It combines the forget and input gates into a single 'update gate'. It also merges the cell state and hidden state.



$$
\begin{aligned}
z_t &= \sigma(W^{(z)}.[x_t, h_{t-1}] + b_z) \\
r_t &= \sigma(W^{(r)}.[x_t, h_{t-1}] + b_r) \\
\tilde{h}_t &= \tanh(W.[x_t, r_t \odot h_{t-1}] + b) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

[3] [Cho et al., 2014]

References i

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014).
Learning phrase representations using rnn encoder–decoder for statistical machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015).
DRAW: A recurrent neural network for image generation.
In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1462–1471.

References ii


Hochreiter, S. (1991).
Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.


Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural Comput., 9(8):1735–1780.


Karpathy, A. (2015).
The unreasonable effectiveness of recurrent neural networks.
http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

References iii


Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014).
A clockwork rnn.
In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1863–1871, Bejing, China. PMLR.


Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2017).
Samplernn: An unconditional end-to-end neural audio generation model.


Olah, C. (2015).
Understanding lstm networks.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

References iv


Schuster, M. and Paliwal, K. (1997).
Bidirectional recurrent neural networks.
Trans. Sig. Proc., 45(11):2673–2681.


Sutskever, I., Vinyals, O., and Le, Q. V. (2014).
Sequence to sequence learning with neural networks.
In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.


van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016).
Conditional image generation with pixelcnn decoders.
CoRR, abs/1606.05328.

References v


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017).
Attention is all you need.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Imperial College London

Generative Adversarial Networks

Deep Learning - MLSS 2019

Pierre Harvey Richemond, Kevin Webster

GANs: The original formulation

Quick reminder on probabilistic modelling :

- We want to learn a data from its distribution x based on samples (x_i).
- We are using function approximation where model parameters are θ.
- We will maximize the log-likelihood of the data,

$$\log p(x, \theta)$$

- In practice we maximize the expected log-likelihood

$$\mathbb{E}[\log p(x, \theta)]$$

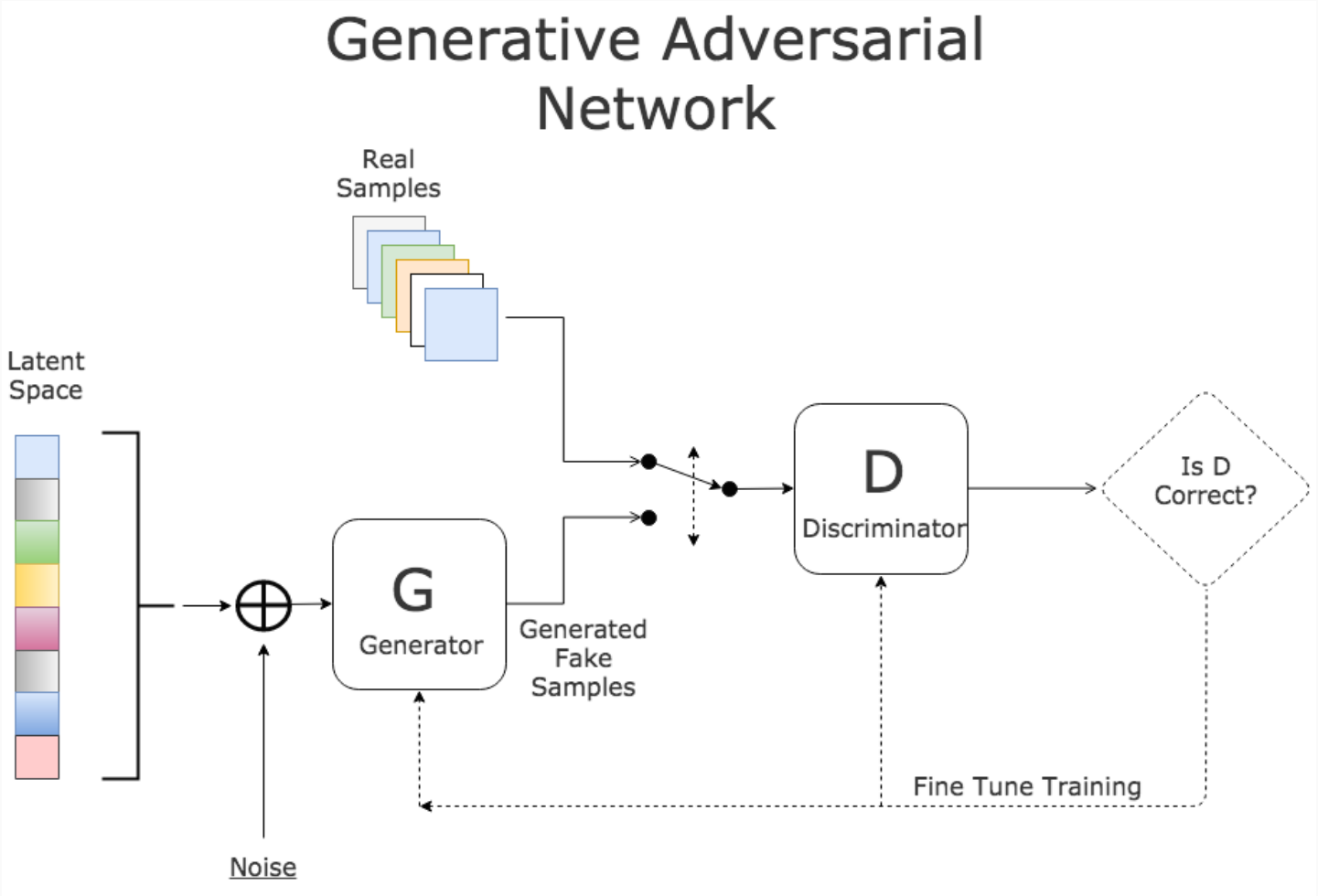
that gets approximated via sampling:

$$\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log p_\theta(x^{(i)})$$

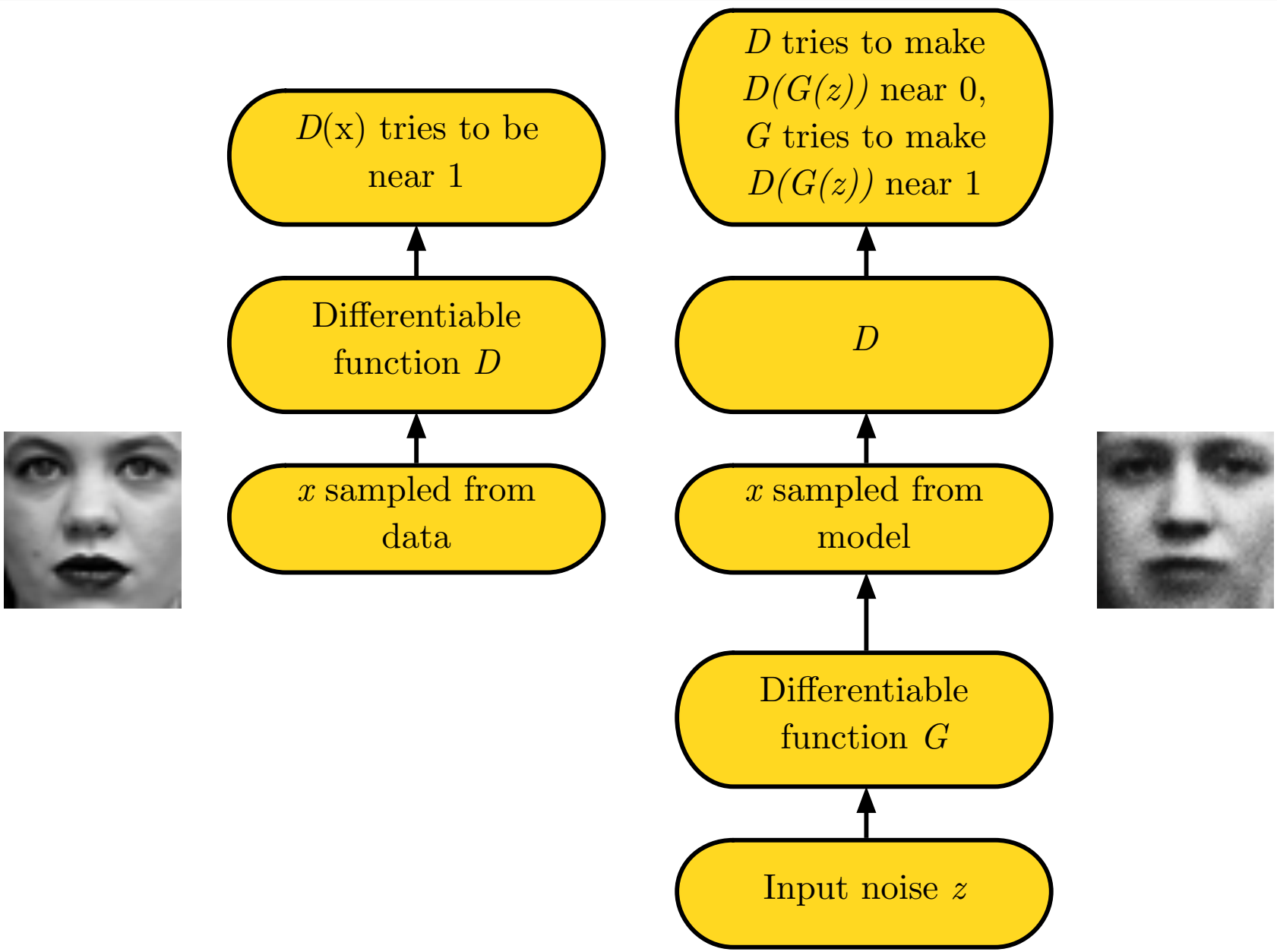
GANs: The original formulation - Splitting things in two

- GANs [Goodfellow et al., 2014] split our probabilistic model in two neural networks : the *generator G* and the *discriminator D*.
- G, with params θ_G, takes a noisy random variable z as an input (with a noise prior distribution $p_z(z)$) and generates candidate samples matching the data distribution x. So the mapping to data space is $G(z, \theta_G)$. $G(z)$ is the generated image.
- D, with params θ_D, takes a sample data point as an input and outputs a single scalar $D(x, \theta_d)$. $D(x)$ represents the probability that x came from the data, rather than the generator-implied distribution p_G.

GANs: The original formulation



GANs: The original formulation

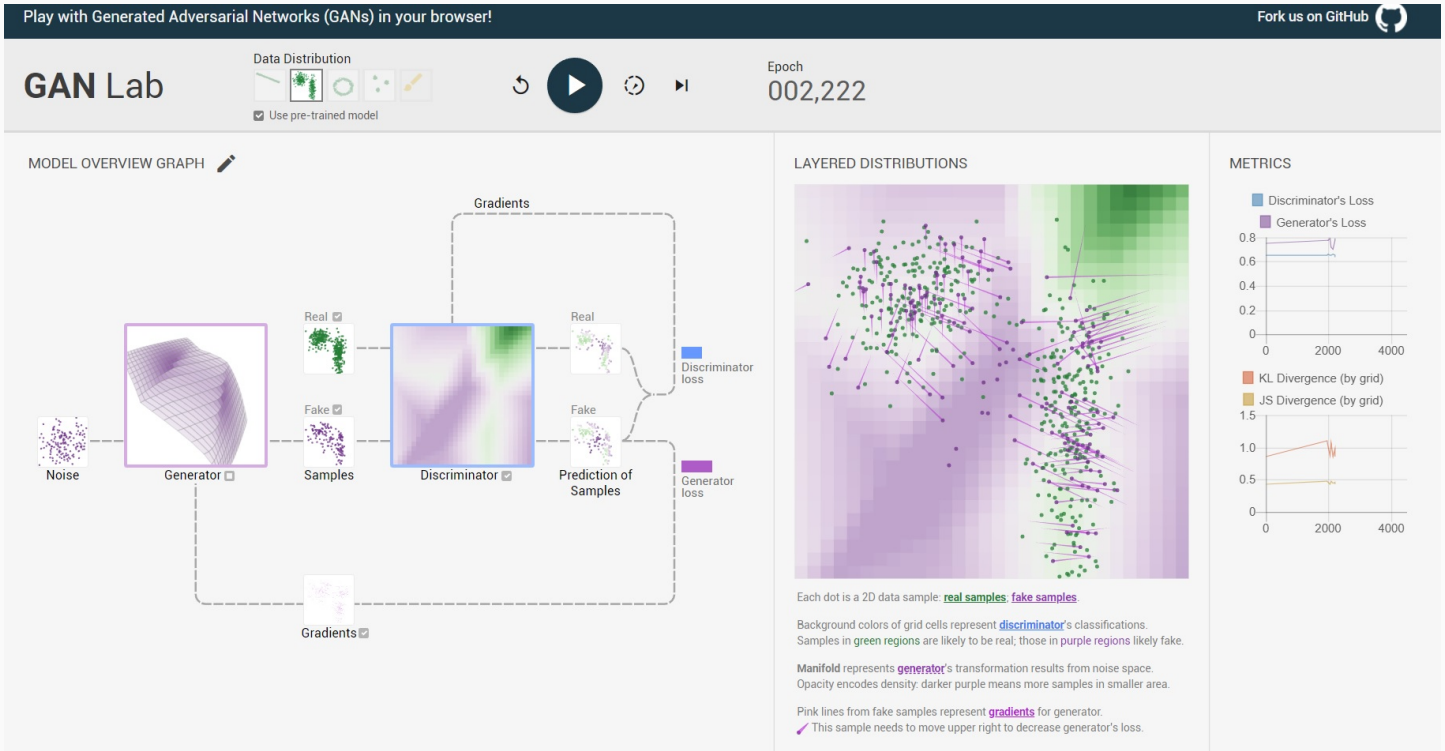


Interpretation

- The discriminator learns using traditional supervised learning techniques, dividing inputs into two classes (real or fake) - it is 'the police'.
- The generator (counterfeinter) is trained to fool the discriminator.
- These two neural networks improve by playing an 'adversarial' game.

Interactive browser visualization

GANLab, made in TensorFlow.js



Adversarial training (1)

- We train D to maximize the probability of assigning the correct (binary) label both to training examples and samples from G ;
- We train G to maximize $\log[D(G(z))]$, or, equivalently, minimize $\log[1 - D(G(z))]$.
- Like so, training becomes a min-max game :

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x \sim p_x}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

$$= \min_{\theta_G} \max_{\theta_D} J_D(\theta_G, \theta_D) + J_G(\theta_G, \theta_D)$$

$$= \min_{\theta_G} \max_{\theta_D} V(G, D)$$

Adversarial training (2)

- This formulation lends itself to theoretical mathematical analysis (min-max : either optimization or game theory).
- The discriminator wishes to minimize $J_D(\theta_G, \theta_D)$ and must do so while controlling only θ_D. The generator wishes to minimize $J_G(\theta_G, \theta_D)$ and must do so while controlling only θ_G.
- In practice, we must implement the game using an iterative, numerical approach. Training D to completion in the inner loop of training is computationally prohibitive.
- Instead, we alternate between k steps of training D and one step of training G. This results in D being maintained near its optimal solution, so long as G changes slowly enough.

Potential issues in practice

- Stability : alternate descent/ascent on min/max (saddle point)
- How to choose params like k and/or control the quality of learning of both networks ?
- Gradient saturation : The output of D will be a probability (most likely coming from a sigmoid unit) and can therefore saturate yielding gradient zero. Two main issues could arise.
- Poor initialization would pin D near zero and provide little training gradient (signal) to the discriminator
- Once the generator works well for some part of the distribution, we could stay stuck in that $D = 1$ region for a while (mode collapse)

All these issues are real and very practical concerns that have been iteratively addressed in an onslaught of papers (1500+) in the last three years. We will present some of those practical improvements.

Naive minibatch-SGD GAN algorithm

Algorithm 1 Standard GAN algorithm.

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.

  • Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.

  • Update discriminator by ascending its stochastic gradient:

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

  Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.

  Update generator by descending its stochastic gradient:

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

Theoretical analysis - Likelihood ratio

The cost used for the discriminator is just the standard cross-entropy loss that is minimized when training a standard binary classifier with a sigmoid output. The only difference is that the classifier is trained on two minibatches of data; one coming from the dataset, where the label is 1 for all examples, and one coming from the generator, where the label is 0 for all examples.

We will see that by training the discriminator, we are able to obtain an estimate of the ratio

$$\frac{p_{data}(x)}{p_{model}(x)}$$

at every point x.

Elementary proof sketch

Just expanding V yields

$$\min_G \max_D V(D,G) = \int_x \Big[p_r(x) \log D(x) + p_G(x) \log(1 - D(x)) \Big] dx$$

We now solve for the optimal discriminator D^*. If

$$Y = a \log(y) + b \log(1 - y)$$

then

$$y^* = \frac{a}{a+b} = \frac{1}{1 + \frac{b}{a}}$$

Therefore

$$D^*(x) = \frac{1}{1 + \frac{p_G(x)}{p_r(x)}}$$

.

Likelihood ratio - 2

The discriminator trained till optimality is

$$D_G^*(x) = \frac{q_{\text{data}}(x)}{q_{\text{data}}(x) + p_G(x)} = \text{sigmoid}(f^*(x)), \tag{1}$$

$$\text{where } f^*(x) = \log q_{\text{data}}(x) - \log p_G(x), \tag{2}$$

and its derivative

$$\nabla_x f^*(x) = \frac{1}{q_{\text{data}}(x)} \nabla_x q_{\text{data}}(x) - \frac{1}{p_G(x)} \nabla_x p_G(x) \tag{3}$$

can be unbounded or even incomputable.

Likelihood ratio training, illustrated

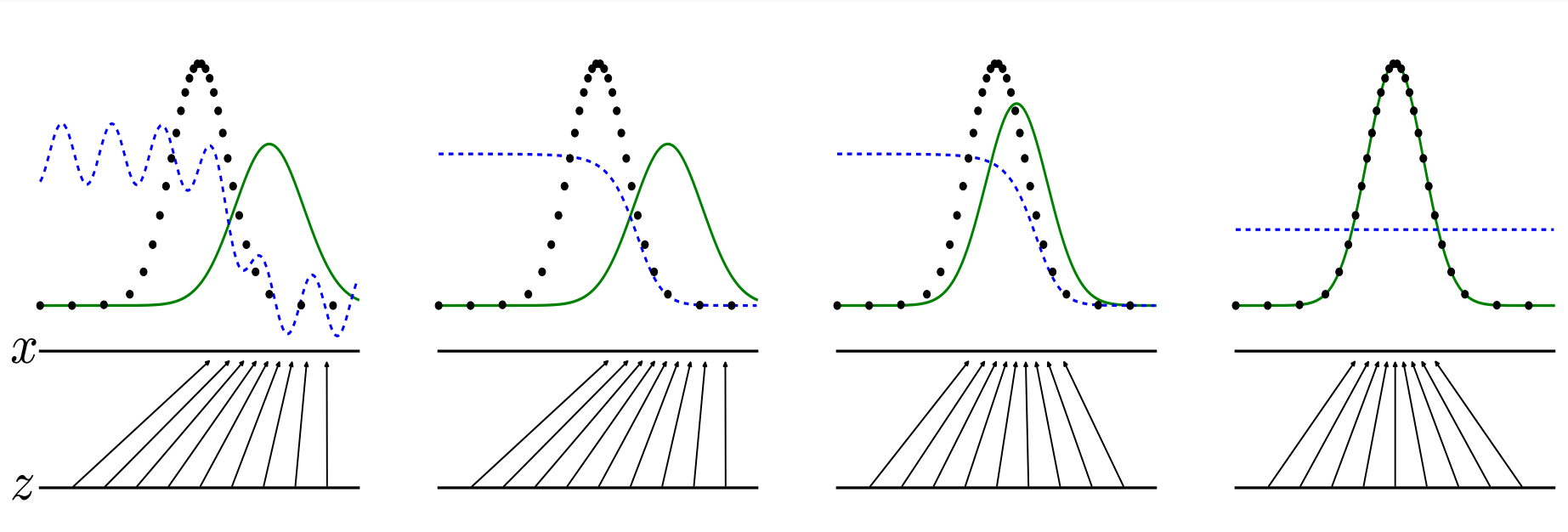


Figure 1: Several steps of adversarial training. The data distribution is black dots, the generator-induced distribution is the green line, and the discriminative distribution is the blue dotted line.

GAN Convergence theorem (Goodfellow, 2014)

Theorem

The global minimum of the virtual training criterion $C(G) = V(D_G^*, G)$ *is achieved if and only if* $p_g = p_{data}$. *At that point,* $C(G)$ *achieves the value* $-\log 4$.

At any point in training (for any p_G), we have:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \,\middle\|\, \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \,\middle\|\, \frac{p_{\text{data}} + p_g}{2}\right)$$

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

where KL is the Kullback-Leibler divergence between two distributions, and JSD is its symmetrized version, the *Jensen-Shannon* divergence.

On balancing G and D, by Ian Goodfellow

'Many people have an intuition that it is necessary to somehow balance the two players to prevent one from overpowering the other. If such balance is desirable and feasible, it has not yet been demonstrated in any compelling fashion. The author's present belief is that GANs work by estimating the ratio of the data density and model density. This ratio is estimated correctly only when the discriminator is optimal, so it is fine for the discriminator to overpower the generator. Sometimes the gradient for the generator can vanish when the discriminator becomes too accurate. The right way to solve this problem is not to limit the power of the discriminator, but to use a parameterization of the game where the gradient does not vanish.' [Goodfellow, 2017]

TL;DR: As the discriminator saturates we get vanishing gradients (the density ratio in the KL blows up).

Mode collapse, illustrated

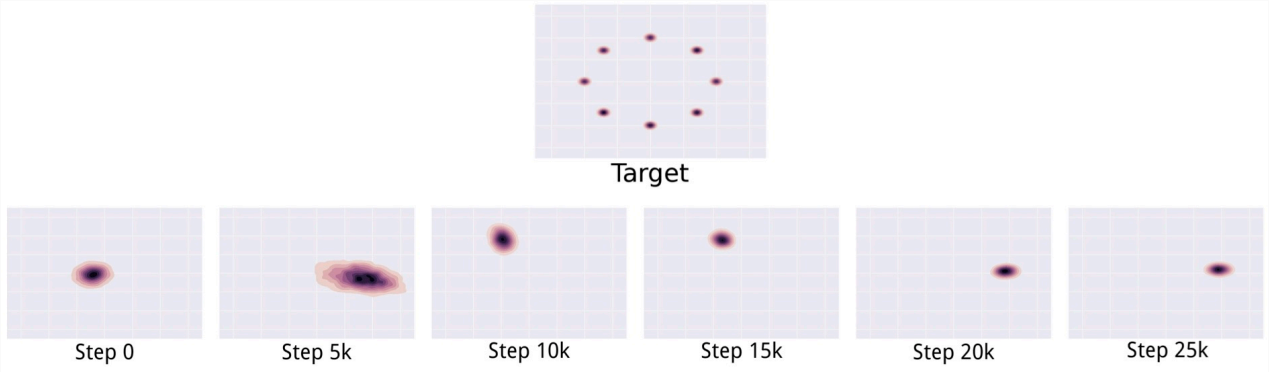


Figure 2: On a Gaussian mixture toy dataset - rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. On a real dataset, this would translate into low-diversity generative samples.

First attempts at robust implementation (2015)

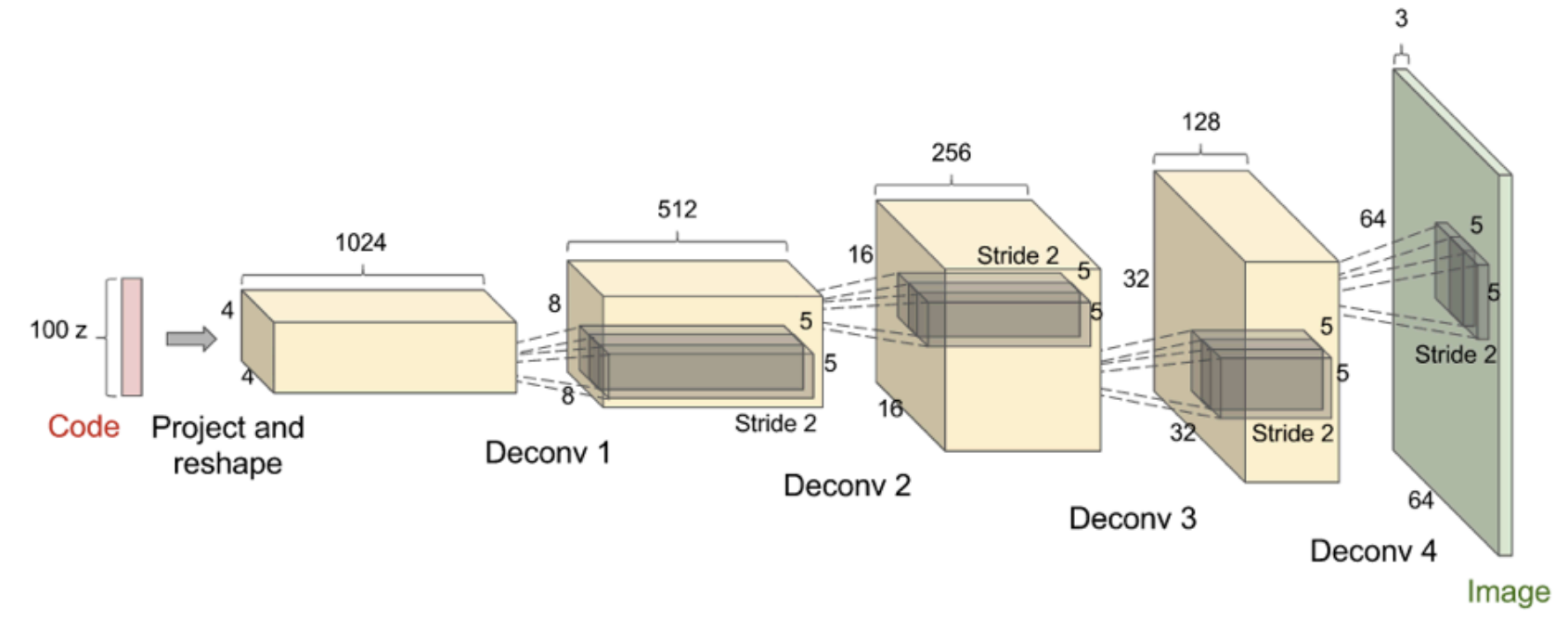


Figure 3: Generator network in the DCGAN architecture [Radford et al., 2015].

First attempts at robust implementation

- To generate images, we use convolutional networks : a standard convnet outputting a scalar from a sigmoid unit as a discriminator, and a reverse convolutional network with *transposed* convolutions (or *deconvolutions*).
- Early model design makes heavy use of batch normalization (in most layers of both the discriminator and the generator, with the two minibatches for the discriminator normalized separately) and of an all-convolutional architecture (no pooling nor “unpooling” layers - when the generator needs to increase the spatial dimension of the representation, it uses transposed convolution with a stride greater than 1).
- However, further stabilization was still required to scale up to more than 32 or 64 square pixels.

Wasserstein GAN (1) - 2017

- Motivation : recall the empirical log-likelihood maximization

$$\max_{\theta} \quad \frac{1}{m}\sum_{i=1}^{m} \log p_\theta(x^{(i)}) \underset{m\to\infty}{\sim} \quad \min_{\theta} \quad KL(p_x||p_\theta)$$

- [Arjovsky et al., 2017] argue that this only makes sense if the model density p_θ exists (!), hence not in situations where we are dealing with data distributions supported by low dimensional manifolds.
- 'It is then unlikely that the model manifold and the true distribution's support have a non-negligible intersection, and this means that the KL distance is not defined (or simply infinite).'
- Another simpler way to put it is a thought experiment where $p_x = N(0,1)$. Imagine either case of $p_g = N(3,1)$ or $p_g = N(6,1)$, then $\nabla D(p_g) = 0$. Said otherwise, GANs cannot recover from a poor initialization (where there is no overlap between the supports of the real data distribution and the generator's).

Wasserstein GAN (2)

- This disjoint support problem is largely due to the use of the KL or JSD metric (explicitly involving the likelihood ratio). It can be alleviated by using optimal transport instead, and its metric the *Wasserstein* distance.
- The Wasserstein-1 distance between $\mathbb{P}_\alpha$ and $\mathbb{P}_\beta$ is given by

$$\inf_{\pi \in \Pi(\mathbb{P}_\alpha, \mathbb{P}_\beta)} \mathbb{E}_{(x,y)\sim\pi}[|x - y|]$$

  where $\Pi(\mathbb{P}_\alpha, \mathbb{P}_\beta)$ is the set of all coupled random variables π whose marginals are $\mathbb{P}_\alpha$ and $\mathbb{P}_\beta$.
- Intuitively the distance is how much 'mass' must be transported in order to transform the distribution.
- This generalizes to a W_p analogue of the L^p norm.
- In one dimension only we have the formula that for two r.v.s X and Y whose inverse cumulative density functions are known,

$$W_p(X, Y) = \left(\int_0^1 |F_X^{-1}(u) - F_Y^{-1}(u)|^p du \right)^p$$

Some simple examples

W_p distance between two Diracs :

$$W_p(\delta_{a_1}, \delta_{a_2}) = |a_1 - a_2|$$

W_2 distance between two Gaussians :

$$W_2^2(\mathcal{N}(m_1, C_1), \mathcal{N}(m_2, C_2)) = \|m_1 - m_2\|_2^2 + \mathrm{tr}\left(C_1 + C_2 - 2\left(C_2^{1/2} C_1 C_2^{1/2}\right)^{1/2}\right)$$

(also known as *Frechet* distance).

The Wasserstein distance, illustrated

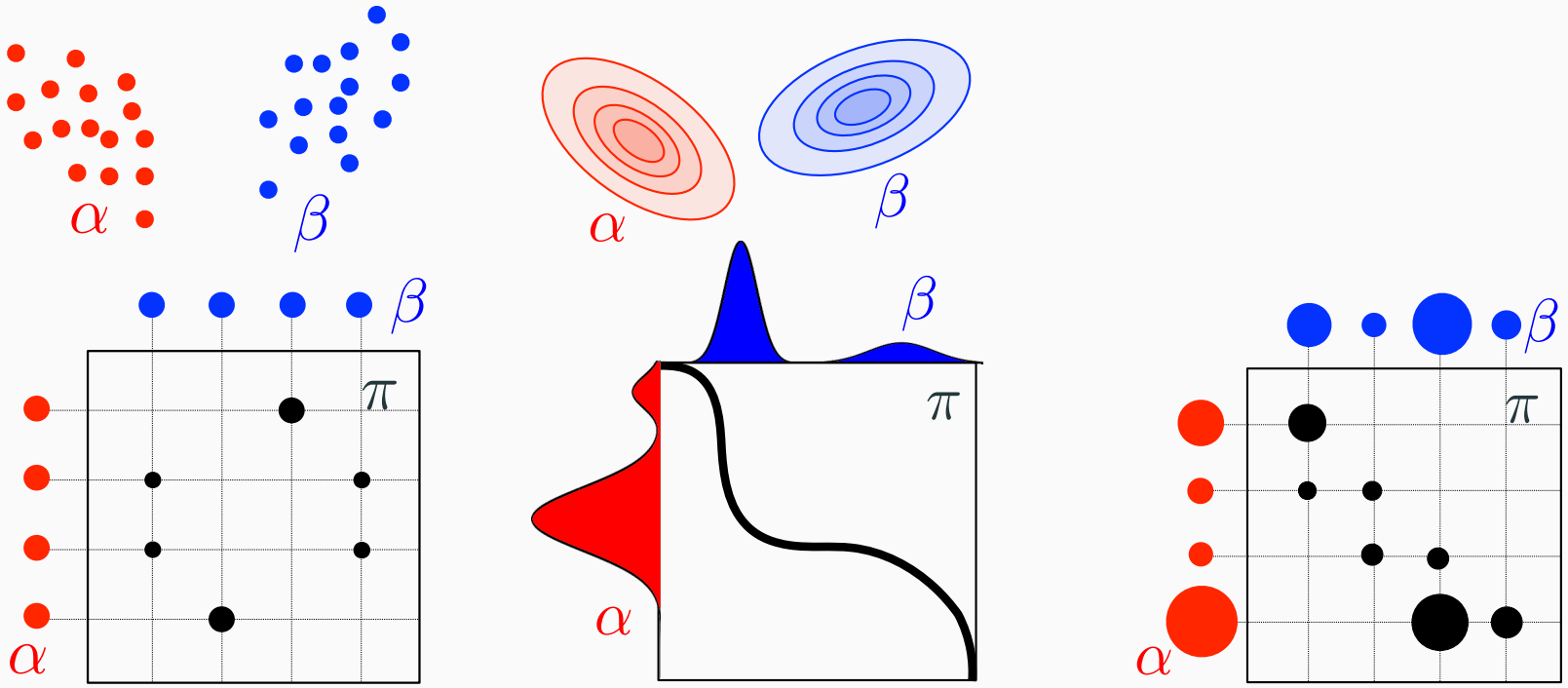


Figure 4: Schematic view of input measures (α, β) and couplings π coming in the optimal transport.

Optimal transport - Interpolation of Gaussians

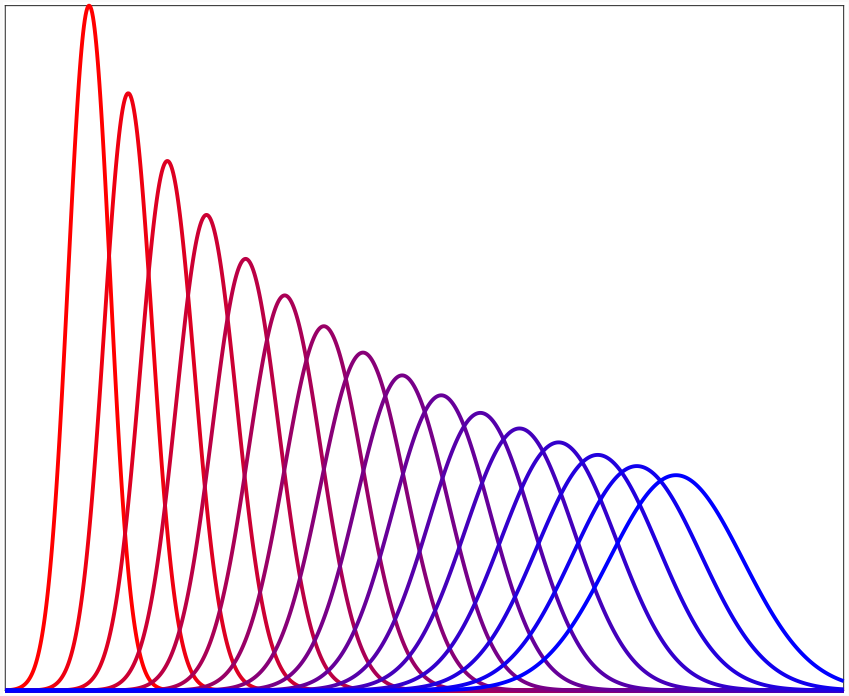


Figure 5: Computation of the *displacement interpolation* between two 1-d Gaussian distributions. The Wasserstein barycenter is unimodal, unlike the L^2 one. W_1 is also called *the Earth Mover distance*.

Earth moving, illustrated

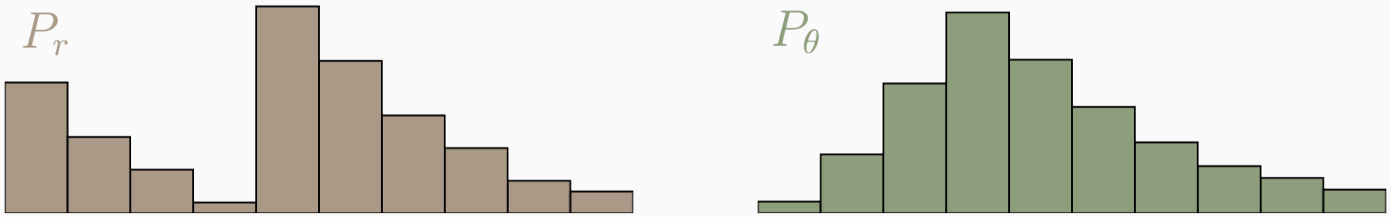


Figure 6: Two discrete distributions P_r and P_θ.

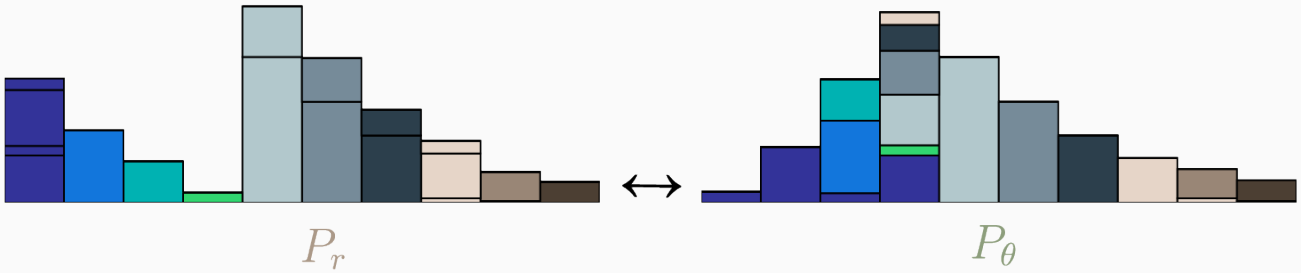


Figure 7: *Earth-moving* between P_r and P_θ.

Wasserstein GAN (3)

For more on the topic, see Cedric Villani's opus magnum Optimal transport: old and new, or Gabriel Peyre's Computational Optimal Transport, whose figures you saw above.
Unlike the parameterized KL we have:

Theorem

For G continuous and locally Lipschitz in its parameters θ, the map $\theta \to W(\mathbb{P}_X, \mathbb{P}_\theta)$ is continuous and diffentiable, a.e.

The Wasserstein distance metric is about as weak as convergence in distribution (so weaker than the KL), but still sensible and better behaved.
Our programme is therefore to replace *KL* with *W*, and compute its gradient ∇_θ.

Kantorovich-Rubinstein duality

In the case of W_1 only, we can make the infimum in the definition tractable using the duality formula

$$W_1(\mathbb{P}_x, \mathbb{P}_\theta) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_x}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

where the supremum (integral probability metric) is over the class of 1-Lipschitz test functions. Therefore we can differentiate and get the representation formula:

Theorem

If $\mathbb{P}_\theta$ is the distribution of $G_\theta(Z)$ with Z a r.v. of density p, then we have for all $\mathbb{P}_x$

$$\nabla_\theta W_1(\mathbb{P}_x, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f^*(G_\theta(z))]$$

where f^* realizes the sup *in the Kantorovich-Rubinstein duality above.*

Loss function in practice

How do we find f^* in practice ?

Well we can approximate it via a neural network with weights w and backpropagate through $\mathbb{E}_{z\sim p(z)}[\nabla_\theta f_w(G_\theta(z))]$.

This is because the WGAN value function is straight out of Kantorovich duality:

$$\min_G \quad \max_D \quad \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{y\sim\mathbb{P}_g}[D(y)], \qquad y = G(z) \quad , (z \sim p(z))$$

The Lipschitz constraint is then enforced (very) naively via weight clipping.

This way, the discriminator D has now turned into a *critic* f. The fact that the Wasserstein distance is continuous and differentiable a.e. means that we can (and should) train the critic till optimality (correlation between discriminator loss, and sample quality).

The WGAN algorithm

Algorithm 2 The WGAN algorithm. All experiments in the paper used the values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Input: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Input: w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**

for $t = 0, \ldots, n_{\text{critic}}$ **do**

- Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
- Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
- $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
- Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
- $w \leftarrow \text{clip}(w, -c, c)$

Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.

$g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$

$\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$

Differences in practice between WGAN and GAN

- No log in the loss - the output of D is no longer a probability, hence we do not apply any sigmoid units at the output of D
- Weight clipping for D
- Train D more than G
- Use RMSProp instead of ADAM
- Use a low learning rate, [Arjovsky et al., 2017] uses $\alpha = 0.00005$
- Monitor discriminator loss (TensorBoard) to check for convergence

WGAN Gradients, illustrated

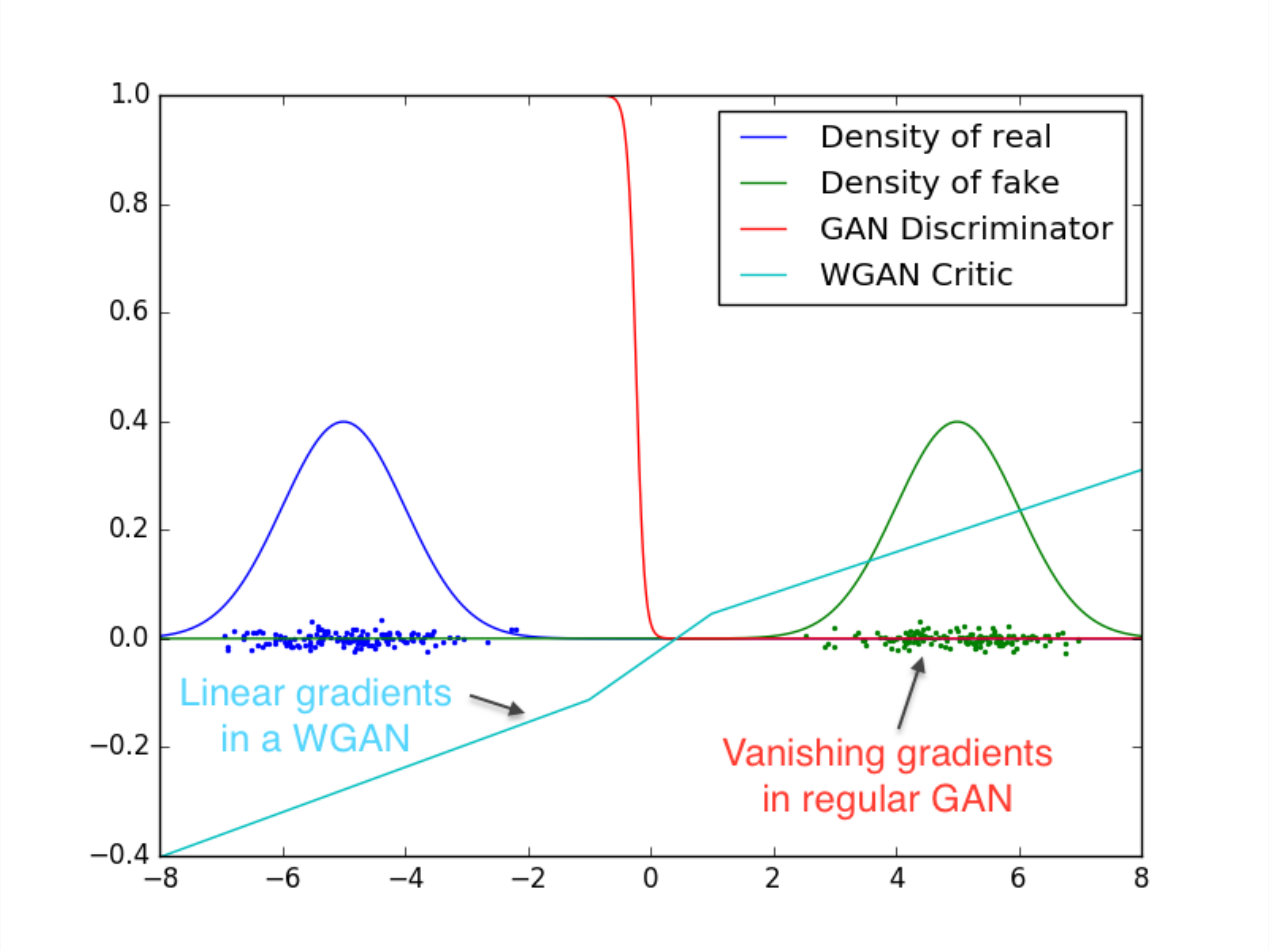


Figure 8: Optimal discriminator and critic when learning to differentiate two Gaussians. The discriminator of a standard GAN saturates and results in vanishing gradients. The WGAN critic provides clean gradients.

More on WGAN

- We have replaced $f(x) = \mathsf{log}(x)$ by $f(x) = x$ by duality in the loss

$$\min_{\theta_g} \max_{\theta_d} \quad \mathbb{E}_{x \sim \mathcal{D}_{real}} f(\mathsf{D}(\mathsf{x}; \theta_d)) + \mathbb{E}_{z \sim \mathcal{G}} f(1 - \mathsf{D}(\mathsf{G}(\mathsf{z}; \theta_g); \theta_d))$$

- This buys us some numerical stability, but in practice, the Wasserstein-GAN is not robust to large learning rates and complex optimizers *a la* ADAM (works with RMSProp).
- WGAN-GP (Gradient Penalty) [Gulrajani et al., 2017] is a notable extension. W_1 weight clipping only really cares about first moment matching - so, combat critic underfitting with adding a gradient regularization term

$$\lambda \cdot \left(||\nabla_x D_w(x)||_2 - 1 \right)^2$$

- Re-metrization alternatives : Least Squares GAN, FisherGAN, SobolevGAN, CramerGAN, Frechet distance... even Wasserstein auto-encoders using similar principles. Endless list on ArXiv, see the excellent review paper [Kurach et al., 2018]...

Spectral Normalization (2018) - 1

- The performance control of the discriminator is still challenging in WGAN. This is because of the basic idea of enforcing the Lipschitz bound via weight clipping.
- Can we normalize discriminator weights in a more principled way ?
- Assuming neural network nonlinearities are Lipschitz themselves, we can control the Lipschitz constant of a multilayer NN by constraining the spectral norm, or largest singular value, $\sigma(W_l)$ of each layer:

$$||f||_{Lip} \leq \prod_{l=1}^{L+1} \sigma(W_l) = \prod_{l=1}^{L+1} \max_{||h||_2 \leq 1} ||W_l h||_2$$

- Spectral normalization [Miyato et al., 2018] simply normalizes the spectral norm of each weight matrix W so that it satisfies Lipschitz constraint 1:

$$\hat{W}_{SN}(W) := \frac{W}{\sigma(W)}$$

Spectral Normalization - 2

- Can we compute or approximate $\sigma(W_l)$ in a tractable way other than singular value decomposition ?
- Yes, with the power iteration method : for square layers, compute $b_{k+1} \leftarrow W_l b_k / ||W_l b_k||$ iteratively from random b_0.
- In general, we begin with vectors $\tilde{\boldsymbol{u}}$ randomly initialized for each weight. If there is no multiplicity in the dominant singular values and if $\tilde{\boldsymbol{u}}$ is not orthogonal to the first left singular vectors, we can produce the first left and right singular vectors through the following update rule:

$$\tilde{\boldsymbol{v}} \leftarrow W^{\mathrm{T}} \tilde{\boldsymbol{u}} / \|W^{\mathrm{T}} \tilde{\boldsymbol{u}}\|_2, \ \tilde{\boldsymbol{u}} \leftarrow W \tilde{\boldsymbol{v}} / \|W \tilde{\boldsymbol{v}}\|_2.$$

We can then approximate the spectral norm of W with the pair of approximative singular vectors:

$$\sigma(W) \approx \tilde{\boldsymbol{u}}^{\mathrm{T}} W \tilde{\boldsymbol{v}}.$$

Spectral Normalization - 3 - Algorithm

Algorithm 3 SGD with spectral normalization

Initialize $\tilde{u}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).

For each update and each layer l:

Apply power iteration method to a unnormalized weight W^l:

$$\tilde{v}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{u}_l / \|(W^l)^{\mathrm{T}} \tilde{u}_l\|_2 \tag{4}$$

$$\tilde{u}_l \leftarrow W^l \tilde{v}_l / \|W^l \tilde{v}_l\|_2 \tag{5}$$

Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{u}_l^{\mathrm{T}} W^l \tilde{v}_l$$

Update W^l with SGD on minibatch data $\mathcal{D}_M$ with learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$$

Stabilizing saddle SGD : Prediction methods (2017)

Saddle-point optimization problems have the general form

$$\min_u \max_v \mathcal{L}(u, v)$$

for a loss function $\mathcal{L}$ and variables u and v. The standard stochastic gradient method to solve saddle-point problems involving neural networks alternates between updating u with a gradient *descent* step, and then updating v with a gradient *ascent* step:

$$\begin{aligned} u^{k+1} &= u^k - \alpha_k \cdot \mathcal{L}'_u(u^k, v^k) \quad | \quad \text{gradient descent in } u, \text{ starting at } (u^k, v^k) \\ v^{k+1} &= v^k + \beta_k \cdot \mathcal{L}'_v(u^{k+1}, v^k) \quad | \quad \text{gradient ascent in } v, \text{ starting at } (u^{k+1}, v^k) \ . \end{aligned}$$

Problem: one proves, assuming exact gradients, that this scheme oscillates around solutions of a simple bilinear saddle $\mathcal{L}(u, v) = v^T K u$ (exercise with ODEs !)

Prediction methods

Yadav and Goldstein [Yadav et al., 2017] propose to stabilize the training of adversarial networks by adding a *prediction* step. Rather than calculating v^{k+1} using u^{k+1}, we first make a prediction, $\bar{u}^{k+1}$, about where the u iterates will be in the future, and use this predicted value to obtain v^{k+1}.

Prediction Method:

$$
\begin{aligned}
u^{k+1} &= u^k - \alpha_k \cdot \mathcal{L}'_u(u^k, v^k) && \mid \quad \text{gradient descent in } u, \text{ starting at } (u^k, v^k) \\
\bar{u}^{k+1} &= u^{k+1} + (u^{k+1} - u^k) && \mid \quad \textit{predict} \text{ future value of } u \\
v^{k+1} &= v^k + \beta_k \cdot \mathcal{L}'_v(\bar{u}^{k+1}, v^k) && \mid \quad \text{gradient ascent in } v, \text{ starting at } (\bar{u}^{k+1}, v^k)\ .
\end{aligned}
$$

The prediction step tries to estimate where u is going to be in the future, by assuming its trajectory remains the same as in the current iteration.

Prediction methods - Intuition

The standard alternating SGD switches between minimization and maximization steps. There is a risk that the minimization step can overpower the maximization step (or vice versa), in which case the iterates will “slide off” the edge of saddle, leading to instability.

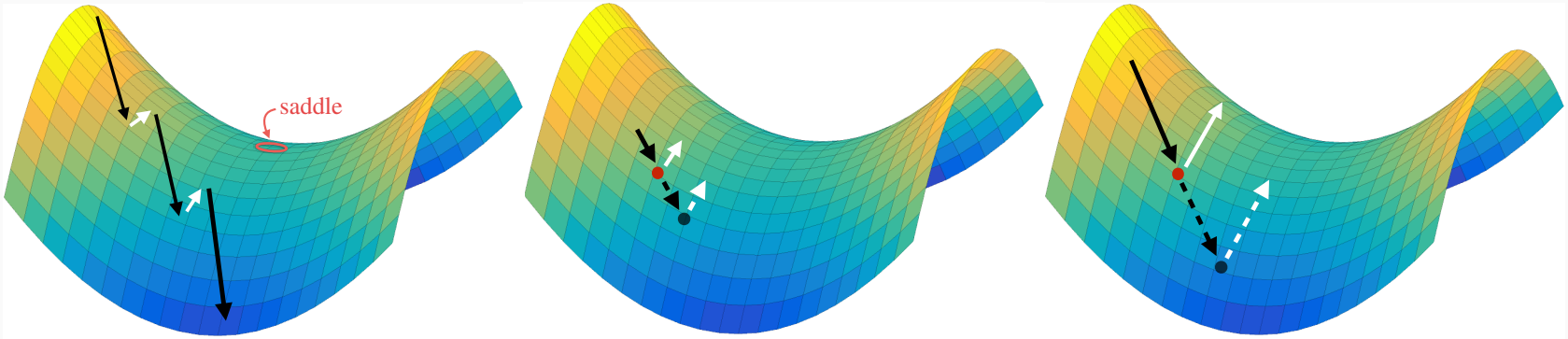


Figure 9: *Left*: If minimization (or, conversely, maximization) is more powerful, the solution path ”slides off” the saddle loss surface and the algorithm becomes unstable (left). *Middle & Right*: The prediction method, enforcing ”momentum in one direction only”. For more on this line of work, see [Gidel et al., 2018].

Prediction methods - Theoretical guarantees

Theorem

Suppose the function $\mathcal{L}(u, v)$ is convex in u, concave in v, and that the partial gradient $\mathcal{L}'_v$ is uniformly Lipschitz smooth in u ($\|\mathcal{L}'_v(u_1, v) - \mathcal{L}'_v(u_2, v)\| \leq L_v\|u_1 - u_2\|$). Suppose further that the stochastic gradients satisfy $\mathbb{E}\|\mathcal{L}'_u(u, v)\|^2 \leq G_u^2$, $\mathbb{E}\|\mathcal{L}'_v(u, v)\|^2 \leq G_v^2$ for scalars G_u and G_v, and that $\mathbb{E}\|u^k - u^\star\|^2 \leq D_u^2$, and $\mathbb{E}\|v^k - v^\star\|^2 \leq D_v^2$ for scalars D_u and D_v. If we choose decreasing learning rates of the form $\alpha_k = \frac{C_\alpha}{\sqrt{k}}$ and $\beta_k = \frac{C_\beta}{\sqrt{k}}$, then the SGD method with prediction converges in expectation, and we have the error bound

$$\mathbb{E}[P(\hat{u}^l, \hat{v}^l)] \leq \frac{1}{2\sqrt{l}}\left(\frac{D_u^2}{C_\alpha} + \frac{D_v^2}{C_\beta}\right) + \frac{\sqrt{l+1}}{l}\left(\frac{C_\alpha G_u^2}{2} + C_\alpha L_v G_u^2 + C_\alpha L_v D_v^2 + \frac{C_\beta G_v^2}{2}\right)$$

where $\hat{u}^l := \frac{1}{l}\sum_{k=1}^{l} u^k$, $\hat{v}^l := \frac{1}{l}\sum_{k=1}^{l} v^k$.

Progressive Growing of GANs (2017)

- Another solution to the disjoint support problem : learn the distribution pinned in place !
- Start with small networks generating 4x4 images and progressively grow them by adding more layers, each doubling the spatial resolution, in a coarse-to-fine fashion.
- According to [Karras et al., 2017], 'This allows the training to first discover large-scale structure of the image distribution and then shift attention to increasingly finer scale detail, instead of having to learn all scales simultaneously. Early on, the generation of smaller images is substantially more stable because there is less class information and fewer modes.'
- This proves so robust that, while best results are obtained with the WGAN-GP loss, one can even use simple least squares.

Progressive Growing of GANs - Architecture

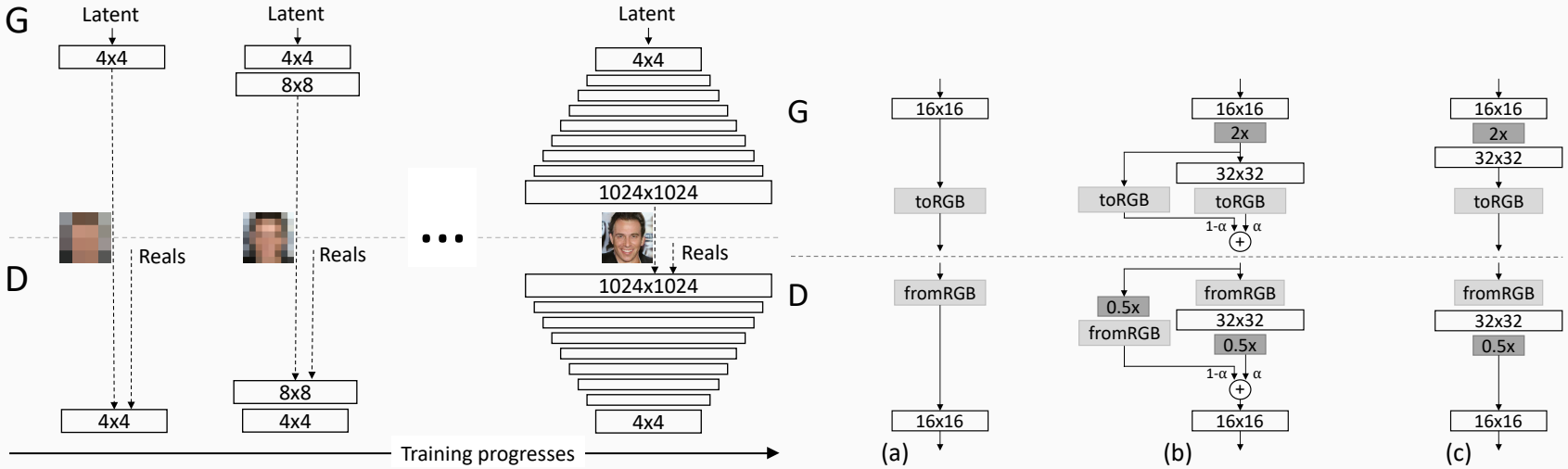


Figure 10: *Left*: Training starts with both generator and discriminator at a spatial resolution of 4x4 pixels - then as training goes further, more layers are added to G and D, doubling the spatial resolution each step. No freezing of layer weights occurs. *Right*: fading in new, double-resolution layers smoothly. α is annealed from 0 to 1.

Progressive Growing of GANs - Results

Figure 11: 1024x1024 generated samples on *CelebA-HQ*. Note some smearing artifacts are present around hair.

Progressive Growing of GANs - Results, late 2018

Figure 12: High-resolution generations from [Karras et al., 2018].

Conclusion

- From MNIST to megapixels : extraordinary progress in only 4 years
- The current frontiers : class-conditional ImageNet [Brock et al., 2018], semi-supervised learning
- Generated samples, in the hard class-conditional case, still suffer from geometry problems - 'global coherence is the primary challenge at high resolution–a model may understand that a spider has "a number" of legs, and that number is between "many" and "lots" but nothing in the networks' inductive biases really forces it to learn "eight"' (Andrew Brock)
- All methods haven't yet been combined together (progressive growing + spectral norm + prediction)
- Once stabilized, GANs are a valuable, generic distribution inference method
- Hence they also find applications connected to *variational inference* (Shakir) and *reinforcement learning* (Katja)

GitHub code repositories

TensorFlow

DCGAN WGAN WGAN-GP Progressive-Growing
Generative-Models-Collection Spectral-Norm BigGAN-Colab

Keras

Keras-GAN-repo DCGAN InfoGAN WGAN

PyTorch

PyTorch-GAN-repo WGAN WGAN-GP SN-GAN

References i

Arjovsky, M., Chintala, S., and Bottou, L. (2017).
Wasserstein GAN.
ArXiv e-prints.

Brock, A., Donahue, J., and Simonyan, K. (2018).
Large Scale GAN Training for High Fidelity Natural Image Synthesis.
ArXiv e-prints.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016).
InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.
ArXiv e-prints.

References ii


Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S. (2018).
A Variational Inequality Perspective on Generative Adversarial Networks.
arXiv e-prints.


Goodfellow, I. (2017).
NIPS 2016 Tutorial: Generative Adversarial Networks.
ArXiv e-prints.


Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative Adversarial Networks.
ArXiv e-prints.

References iii


Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017).
Improved Training of Wasserstein GANs.
ArXiv e-prints.


Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017).
Progressive Growing of GANs for Improved Quality, Stability, and Variation.
ArXiv e-prints.


Karras, T., Laine, S., and Aila, T. (2018).
A Style-Based Generator Architecture for Generative Adversarial Networks.
arXiv e-prints.

References iv


Kurach, K., Lucic, M., Zhai, X., Michalski, M., and Gelly, S. (2018).
The GAN Landscape: Losses, Architectures, Regularization, and Normalization.
ArXiv e-prints.


Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018).
Spectral Normalization for Generative Adversarial Networks.
ArXiv e-prints.


Radford, A., Metz, L., and Chintala, S. (2015).
Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.
ArXiv e-prints.


Yadav, A., Shah, S., Xu, Z., Jacobs, D., and Goldstein, T. (2017).
Stabilizing Adversarial Nets With Prediction Methods.
ArXiv e-prints.

References v