Fundamentals of machine learning

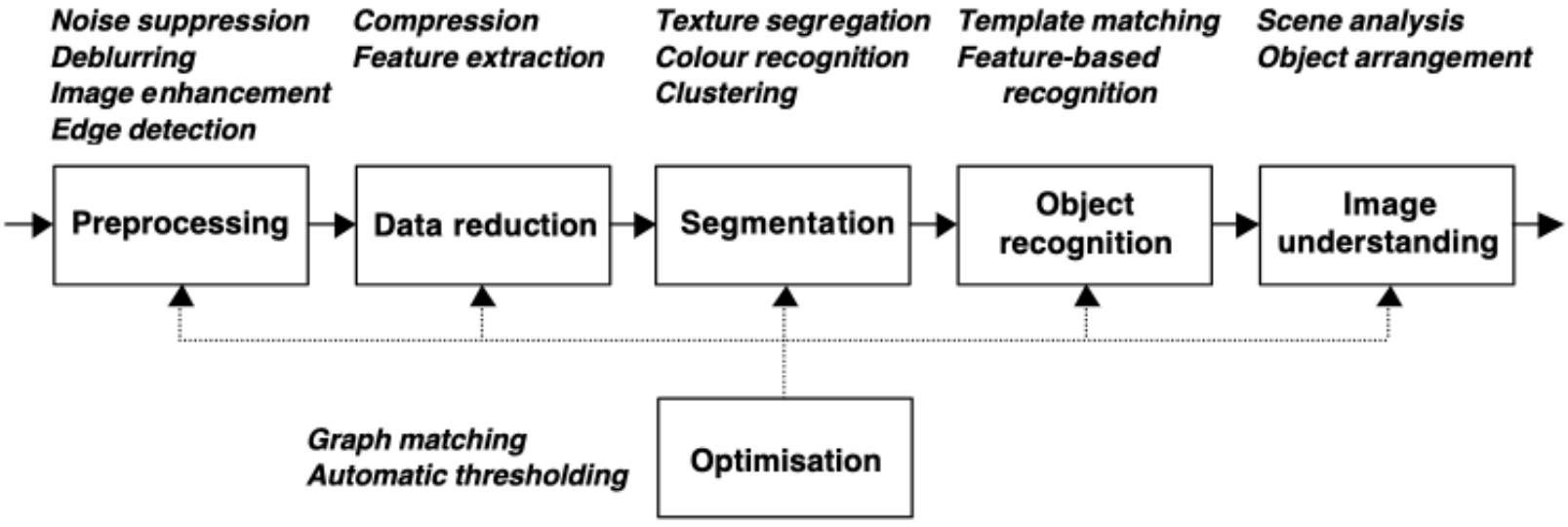
Course 10 : advanced neural networks

Mathurin Massias & Titouan Vayer
email: mathurin.massias@inria.fr, titouan.vayer@inria.fr

March 31, 2025

ENS DE LYON

Neural networks in image processing

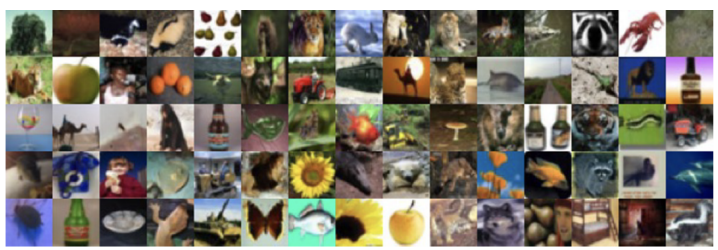


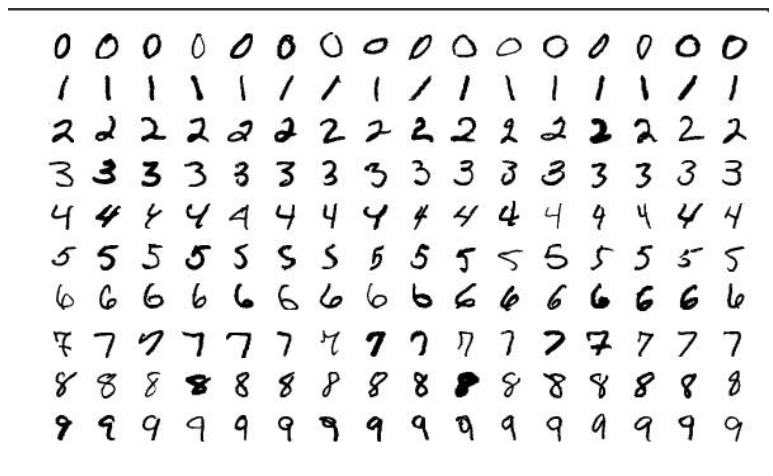
https://www.egmont-petersen.nl/science/Journal-papers/Egmont-PR-Review2002.pdf

- Applications: pattern recognition, face recognition, optical character recognition, video surveillance
- Other applications: driverless cars, robots design, virtual personal assistant

Neural networks in image classification

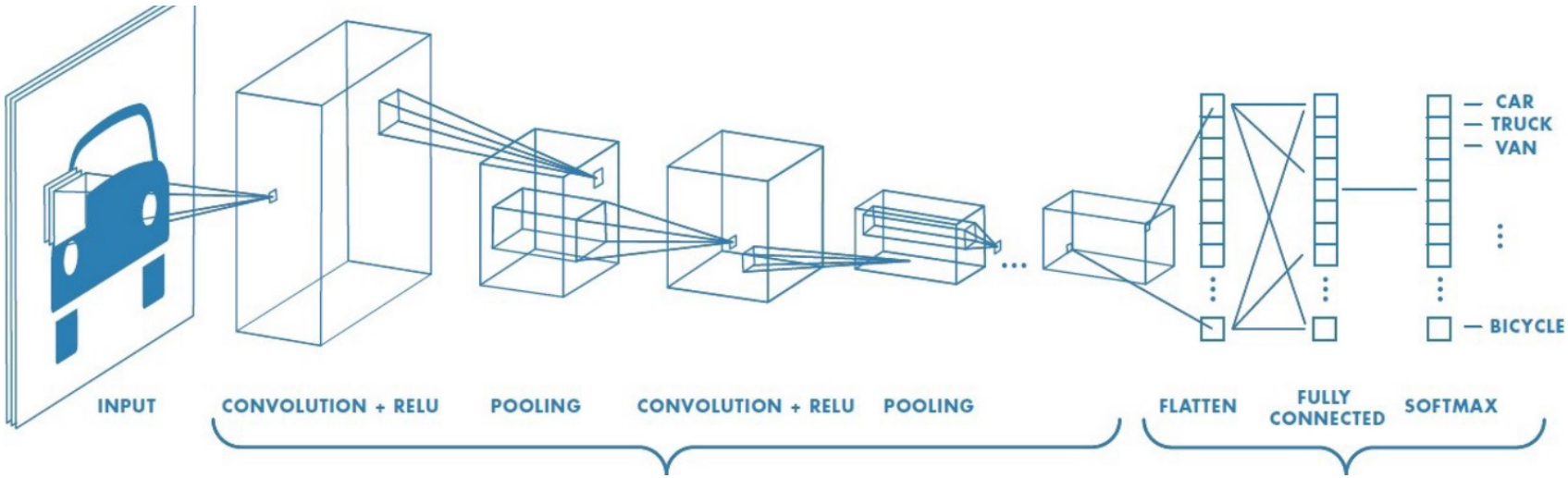
- Around 2010: big successes of NN in image classification.
- Massive databases of labeled images were being accumulated
- MNIST: collection of 70000 handwritten 28×28 grayscale digits
- CIFAR-10 : 60000 32x32 colour images in 10 classes, with 6000 images per class.
- CIFAR-100 : 60000 32x32 colour images in 100 classes (20 superclasses, with five classes each)
  `https://www.cs.toronto.edu/~kriz/cifar.html`
- ImageNet: $>$ 14 millions of images manually annotated, objects classified in $>$ 20.000 classes `https://www.image-net.org/`





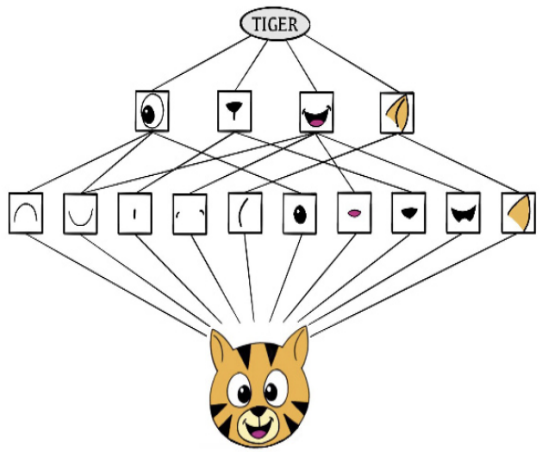
Convolutional neural networks (CNN)

- A special family of networks for image classification
- CNNs mimic how humans classify images: recognize specific features or patterns
- They are composed of convolution and pooling layers + a standard feedforward classification network
- AlexNet: 2012 Imagenet winner, Neurips 2022 test of time award, 140 k citations



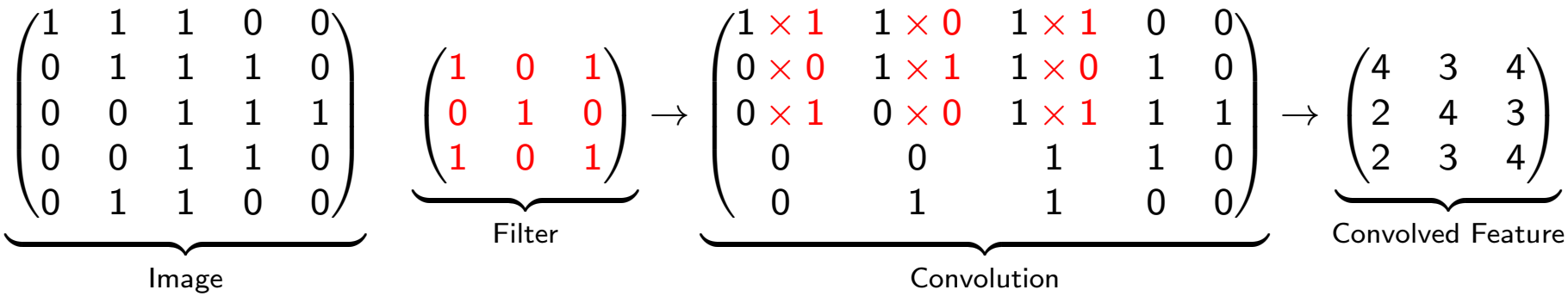
How do they work?

- CNN first identifies low-level features (small edges, patches of color)
- Low-level features are combined to form higher-level features (ears, eyes)
- Presence/absence of higher-level features contributes to the probability of any output class



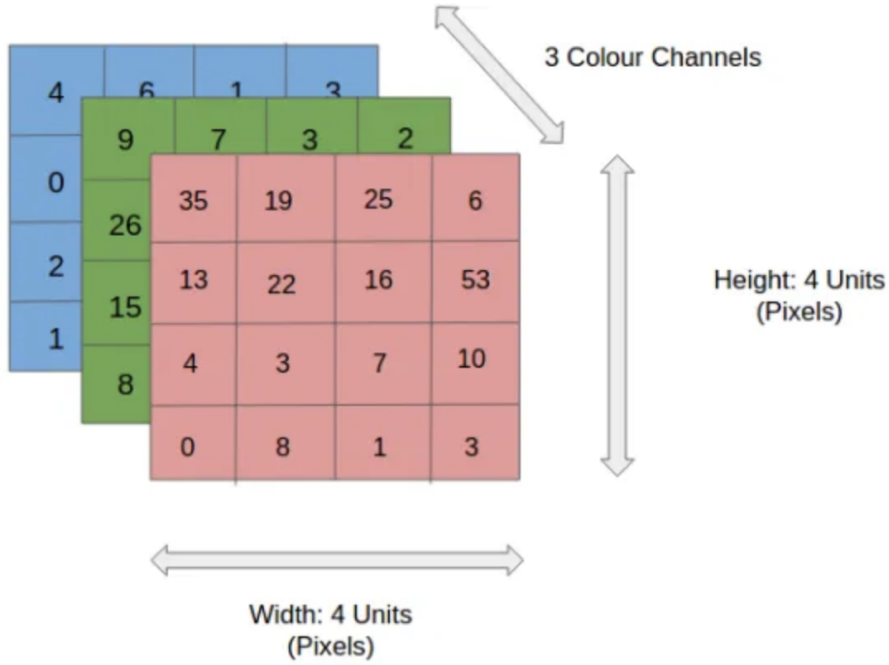
Convolution layers

- A convolution layer is made up of a large number of convolution filters
- Each filter determines whether a particular local feature is present in an image.
- A convolution filter relies on a very simple operation, called convolution, which basically amounts to repeatedly multiplying matrix elements and then adding the results
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53



Is it fundamentlly different from a fully connected layer?

Color input image



Repeat on all the channels and sum up.

Color input image

- If a submatrix of the original image resembles the convolution filter, then it will have a large value in the convolved image
- Thus, the convolved image highlights regions of the original image that resemble the convolution filter

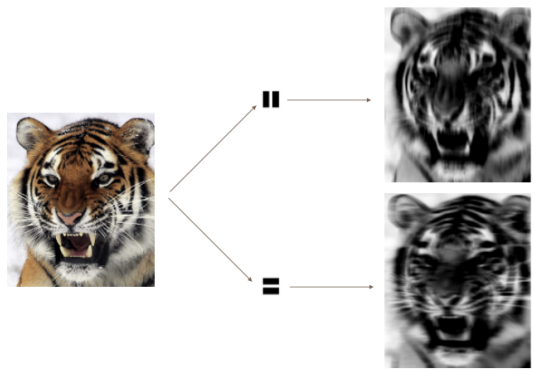


Figure: Convolution filter contains mostly zeros (black), with a narrow strip of ones (white) oriented either vertically or horizontally within the image

Classical image processing vs CNNs

- In a convolution layer, we use a whole bank of filters to pick out a variety of differently-oriented edges and shapes in the image.
- Using predefined filters is standard practice in classical image processing.
- By contrast, with CNNs the filters are learned for the specific classification task
- The first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc.
- With added layers, the architecture adapts to the High-Level features as well

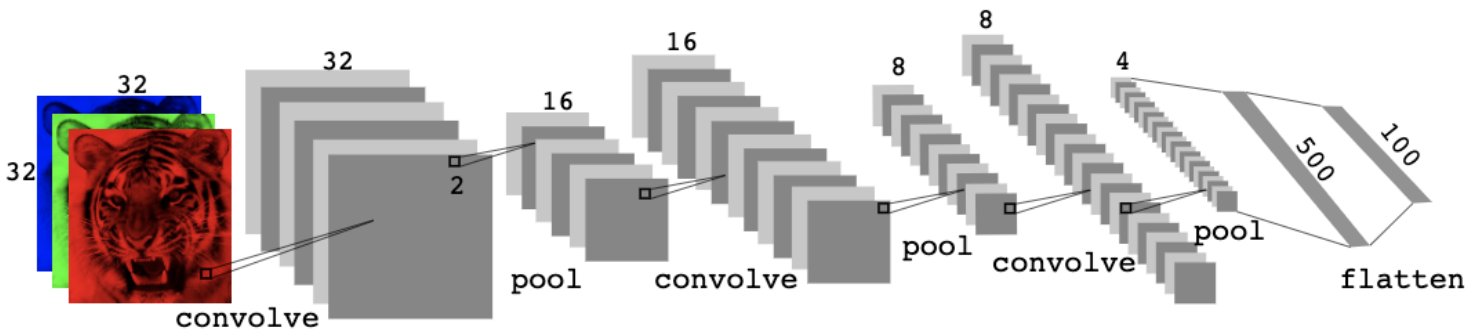
Pooling layers

- A pooling layer provides a way to condense a large image into a smaller image
- Max pooling: summarizes each non-overlapping 2×2 block of pixels using the maximum value in the block.
- This reduces the size of the image by a factor of two in each direction

$$\begin{pmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 3 & 5 \\ 2 & 4 \end{pmatrix}$$

CNN architecture

- Each filter produces a new two-dimensional feature map.
- The number of convolution filters is akin to the number of units at a particular hidden layer in a fully-connected neural network
- Example with 6 filters (2 for each channel)



Data augmentation

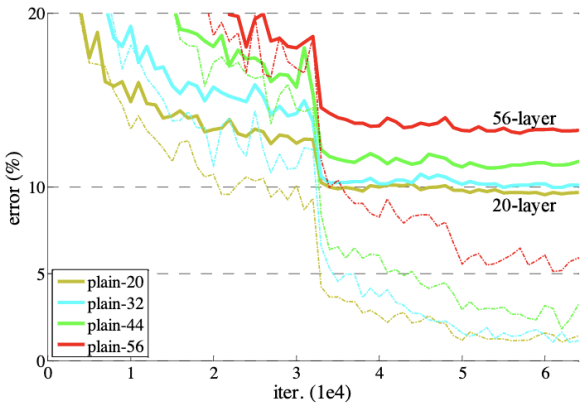
- Trick to improve the training for image analysis
- Each training image is replicated many times, with each replicate randomly distorted
- Examples: zoom, horizontal and vertical shift, shear, small rotations, horizontal flips.
- Effects:
  - increasing the training set
  - is a form of regularization and protects against overfitting.

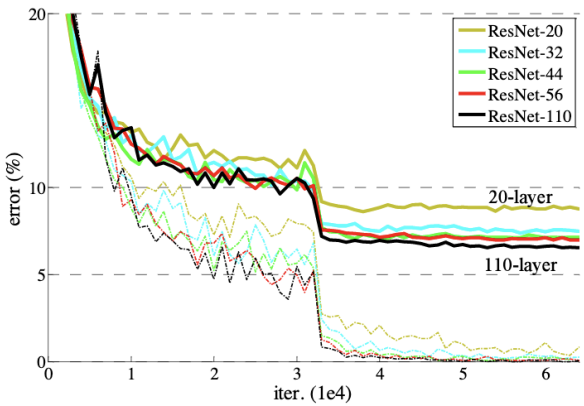
Baseline results and pretrained networks

- CNN: `https://www.cs.toronto.edu/~kriz/cifar.html`
- More advanced networks: `https://paperswithcode.com/sota/image-classification-on-cifar-10`

ImageNet Large Scale Visual Recognition Challenge (LSVRC)
`https://www.image-net.org/challenges/LSVRC/`

- AlexNet `https://pytorch.org/hub/pytorch_vision_alexnet/`
- VGG `https://www.robots.ox.ac.uk/~vgg/research/very_deep/`
- ResNet `https://arxiv.org/abs/1512.03385`





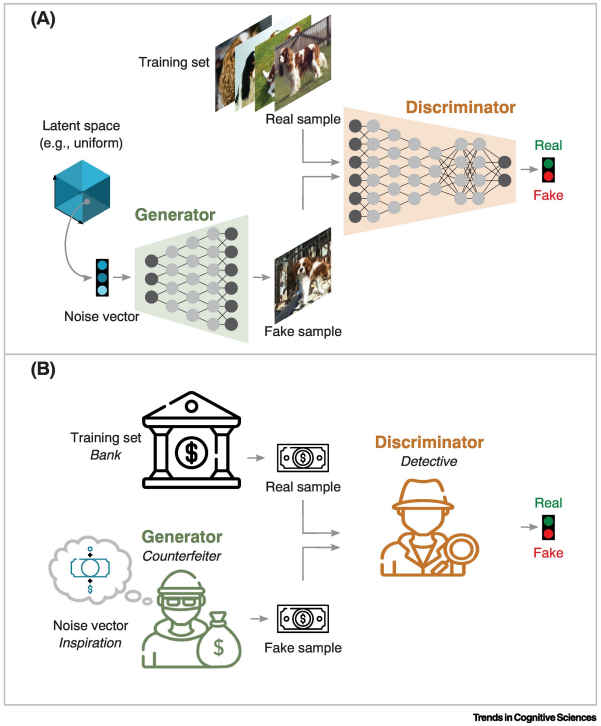
Generative adversarial networks (GANs)

- GANs are generative models: they create new data instances that resemble your training data.
- Example: create images that look like photographs of human faces, even though the faces don't belong to any real person.
- `https://developers.google.com/machine-learning/gan`

Figure 1: Images generated by a GAN created by NVIDIA.

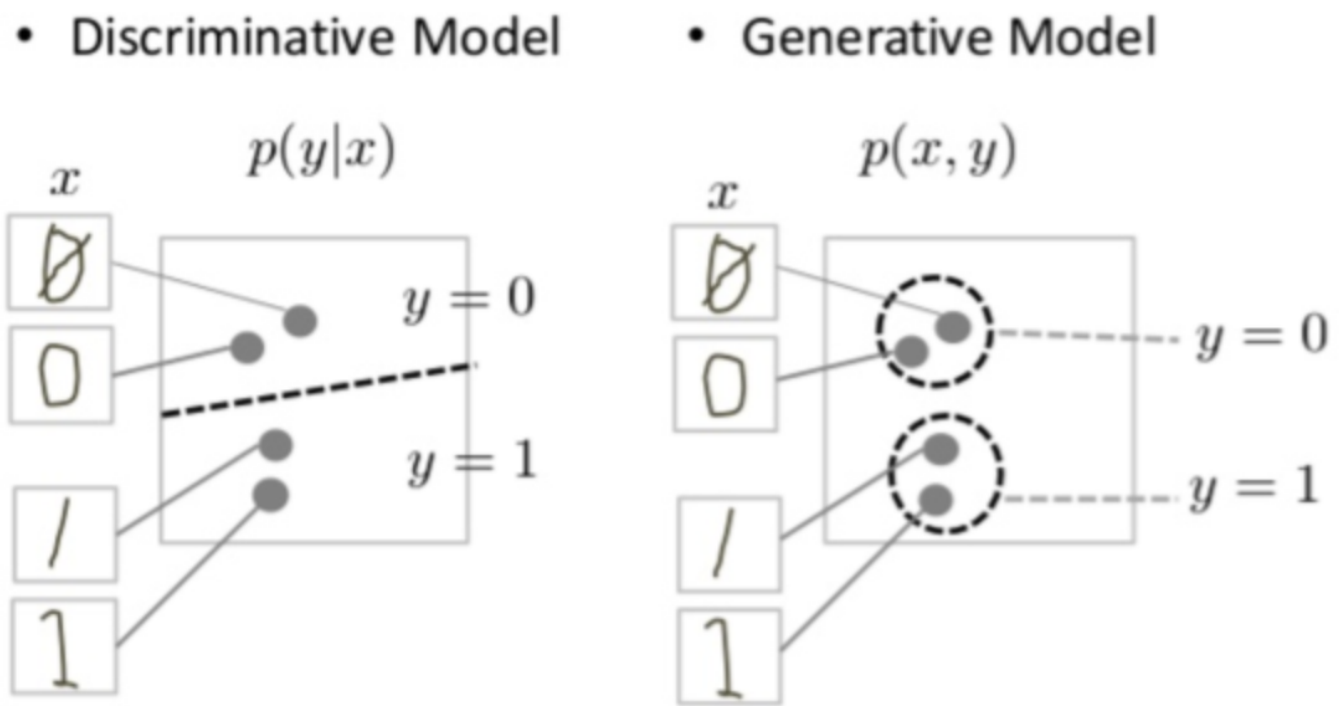
Components of a GAN

- GANs= generator (which learns to produce the target output) + discriminator (learns to distinguish true data from the output of the generator)
- The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled.

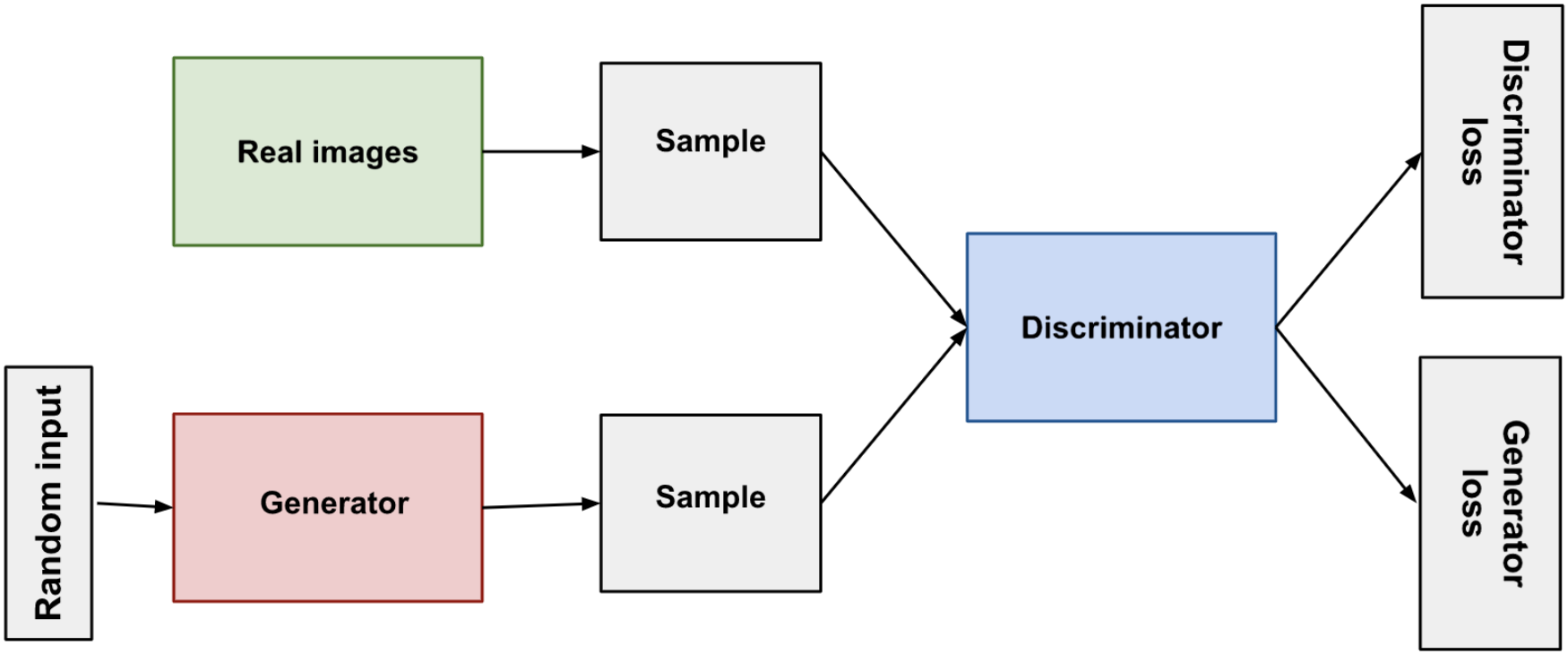


Generator vs discriminator

- Generative models can generate new data instances (generate new photos of animals that look like real animals)
- Discriminative models discriminate between different kinds of data instances (tell a dog from a cat)
- Formally, given a set of data instances X and a set of labels Y:
  - Generative models capture the joint probability $p(X, Y)$
  - Discriminative models capture the conditional probability $p(Y|X)$.

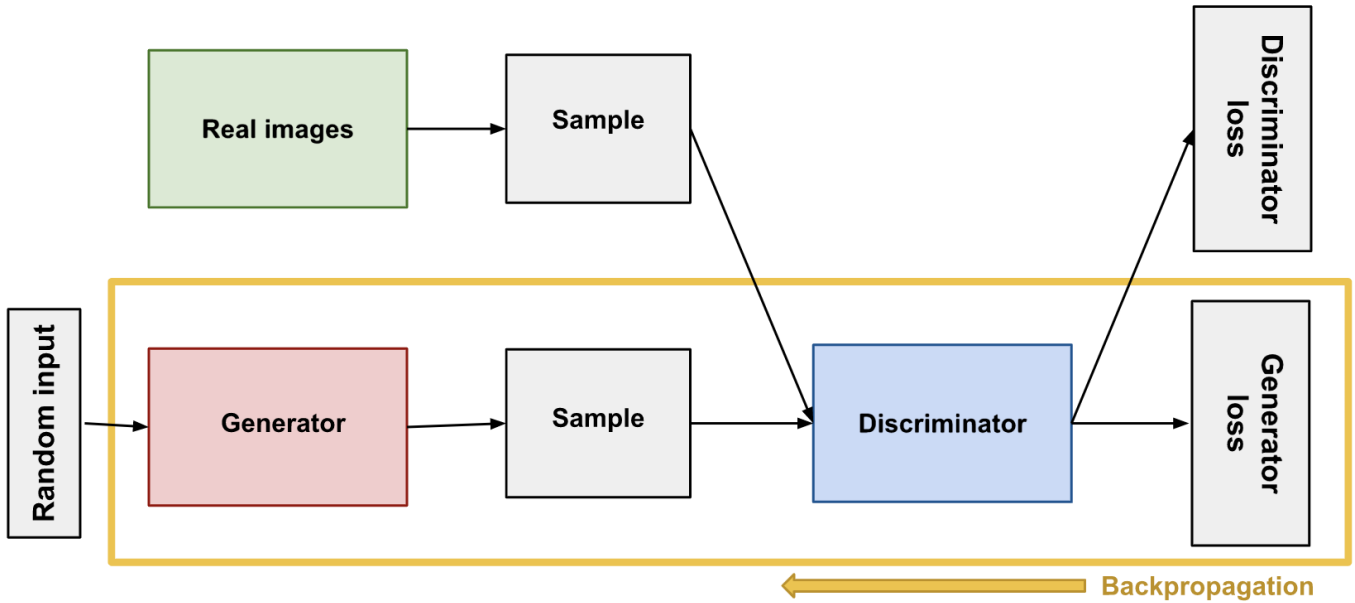


Training of the GAN



Training of the GAN

- The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.



Training of the GAN

- The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

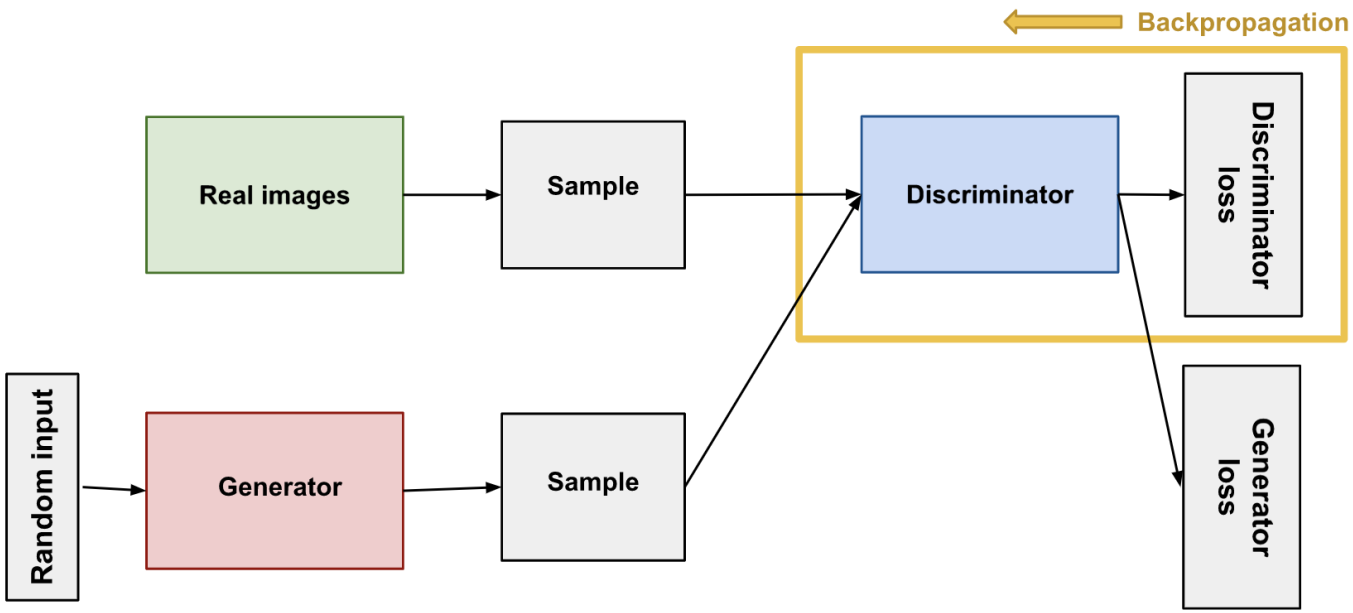
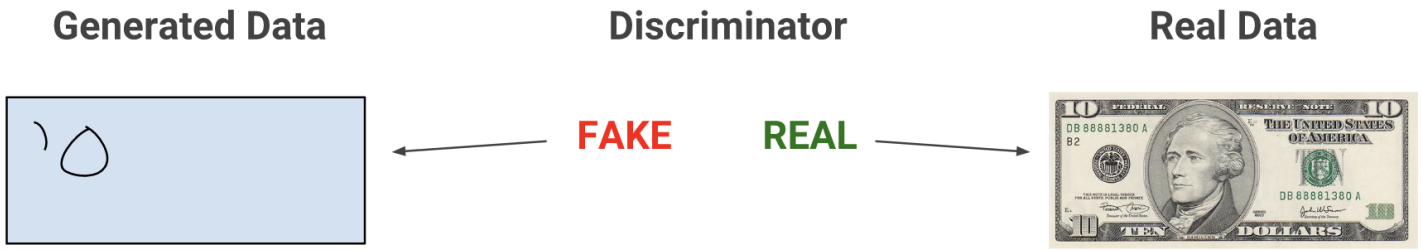


Figure 1: Backpropagation in discriminator training.

Training of the GAN



As training progresses, the generator gets closer to producing output that can fool the discriminator:

Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

Training of the GAN

Alternating Training

1. The discriminator trains for one or more epochs.
2. The generator trains for one or more epochs.
3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

Convergence

The training of G depends on the feedback of D: important that one is not overtrained over the other, and to stop at the right time. $\rightarrow$ For a GAN, convergence is often a fleeting, rather than stable, state.

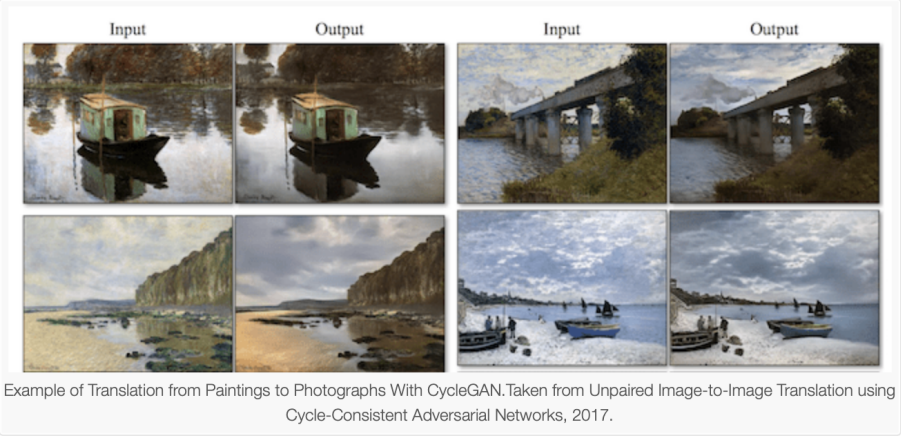
Loss function: the Minimax Loss

G minimizes while D maximizes the following function:

$$E_x(\log(D(x))) + E_z(\log(1 - D(G(z))))$$

- $D(x)$ is the discriminator's estimate of the probability x is real.
- E_x is the expected value over all real data instances.
- $G(z)$ is the generator's output when given noise z.
- $D(G(z))$ is the discriminator's estimate of the probability that a fake instance is real.
- E_z is the expected value over all generated fake instances $G(z)$
- The formula derives from the cross-entropy between the real and generated distributions.
- The generator can't directly affect the $\log(D(x))$ term in the function, so, for the generator, minimizing the loss is equivalent to minimizing $\log(1 - D(G(z)))$.
- More advanced GANs use other loss functions (e.g., Wasserstein loss for TF-GAN)

Extensions of GANs: Image-to-Image Translation



Example of Translation from Paintings to Photographs With CycleGAN.Taken from Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

Example of Four Image-to-Image Translations Performed With CycleGANTaken from Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

Example of Sketches to Color Photographs With pix2pix.Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

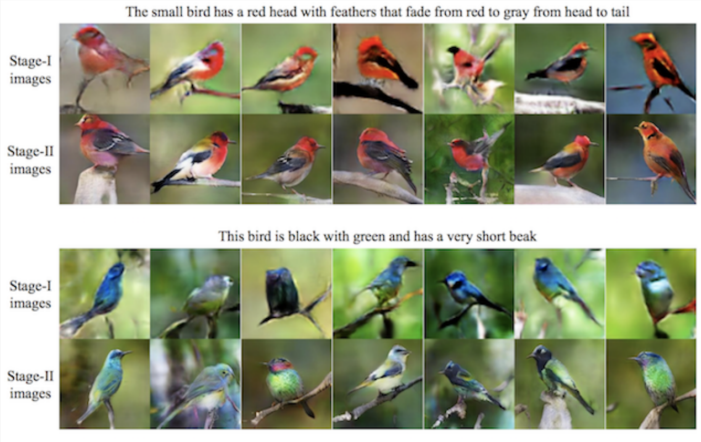
Example of Photographs of Daytime Cityscapes to Nighttime With pix2pix.Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

Satellite to Google Maps, day to night, sketches to color photo, black and white to color..

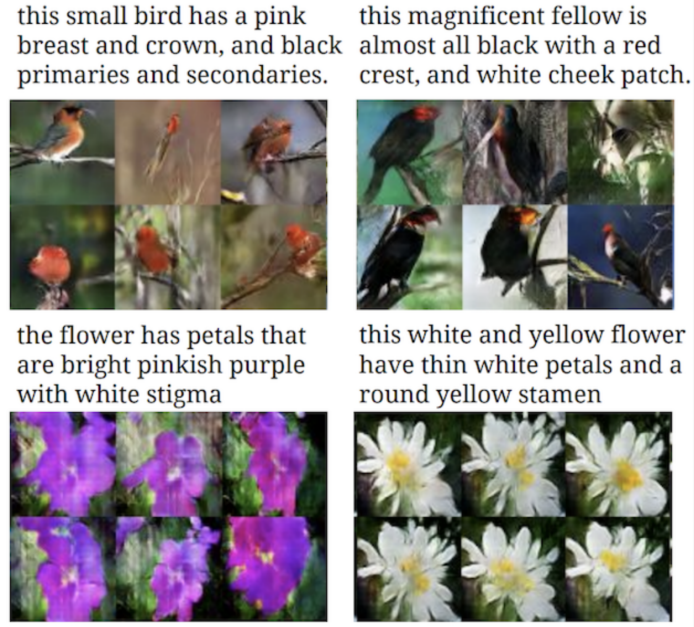
`https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/`

Text-to-Image Translation (text2image)

Generate realistic looking photographs from textual descriptions



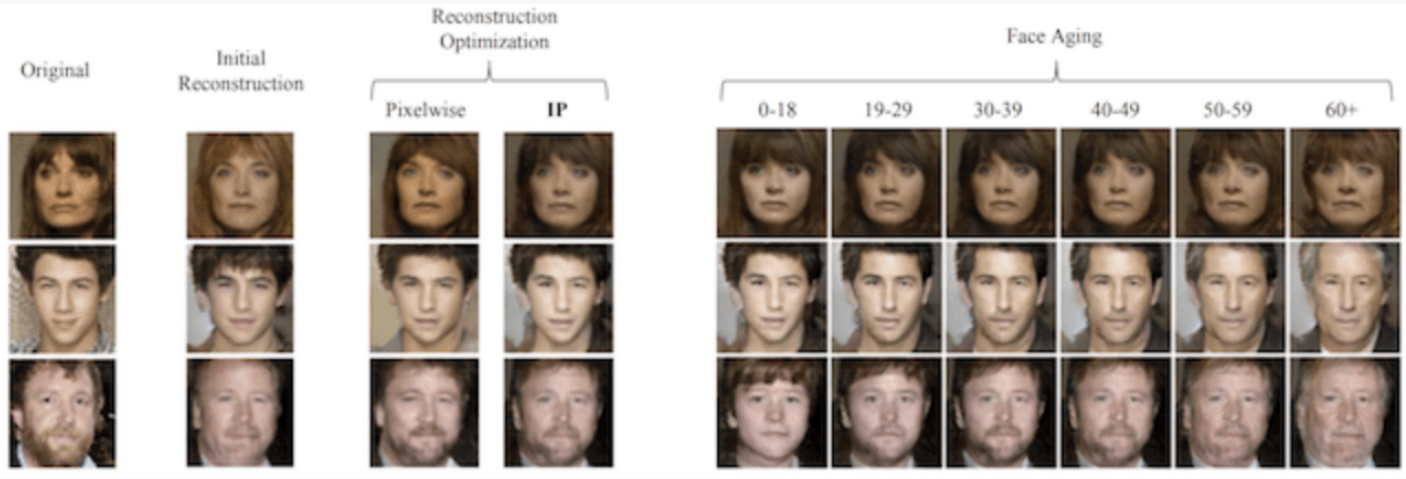
Example of Textual Descriptions and GAN-Generated Photographs of BirdsTaken from StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2016.



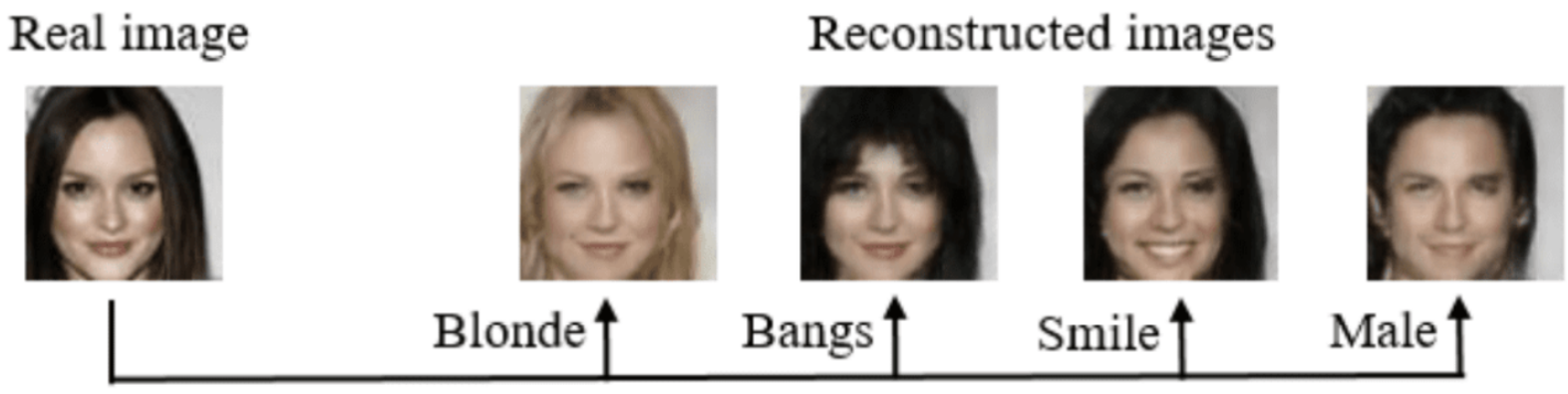
Example of Textual Descriptions and GAN-Generated Photographs of Birds and Flowers.Taken from Generative Adversarial Text to Image Synthesis.

Photograph Editing

Photograph Editing (reconstruct photographs with specific specified features, such as changes in hair color, style, facial expression, age)



Example of Photographs of Faces Generated With a GAN With Different Apparent Ages.Taken from Face Aging With Conditional Generative Adversarial Networks, 2017.



Example of Face Photo Editing with IcGAN.Taken from Invertible Conditional GANs For Image Editing, 2016.

Classical solution of partial differential equations (PDEs)

A PDE problem

Given a domain $\Omega \subset \mathbb{R}^d$, we consider the following differential system:

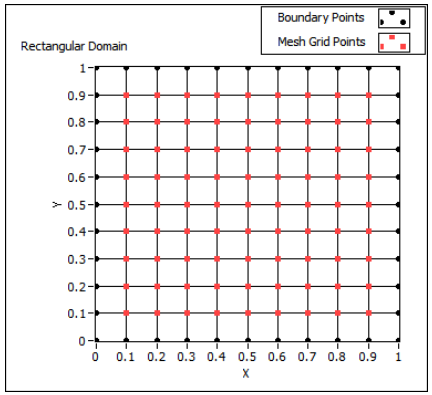
$$\begin{cases} \mathcal{L}(u(z)) = g(z) \text{ for } z \in \Omega \\ \mathcal{B}(u(z)) = b(z) \text{ for } z \in \partial\Omega \end{cases}$$

where $\mathcal{L}$ and $\mathcal{B}$ are two (possibly nonlinear) differential operators and g and b are two given functions.

Linear case

$\mathcal{L}$ is a linear operator. The PDE is discretized on a mesh, yielding a linear system:

$$Lu = g$$

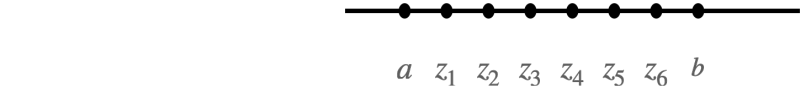


Example: Poisson's equation $\Delta u(z) = r(z)$

1D case

Equation: $\Omega = [a, b]$, $z \in \mathbb{R}$, $\quad u''(z) = r(z)$

Choose a mesh with m interior points



Discretize second order derivative

$$u''(z_i) \sim \frac{u(z_{i+1} - 2u(z_i) + u(z_{i-1})}{h^2}, \; h = \frac{1}{m+1},$$

Solve the system

$Lu = r$

$$L = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & \dots \\ 1 & -2 & 1 & \dots \\ & & \ddots & \\ \dots & & 1 & -2 \end{pmatrix}, \quad g = \begin{pmatrix} g(a) \\ g(z_1) \\ \vdots \\ g(z_m) \\ g(b) \end{pmatrix} \quad u = \begin{pmatrix} u(a) \\ u(z_1) \\ \vdots \\ u(z_m) \\ u(b) \end{pmatrix}$$

Example: Poisson's equation $\Delta u(z) = r(z)$

2D case

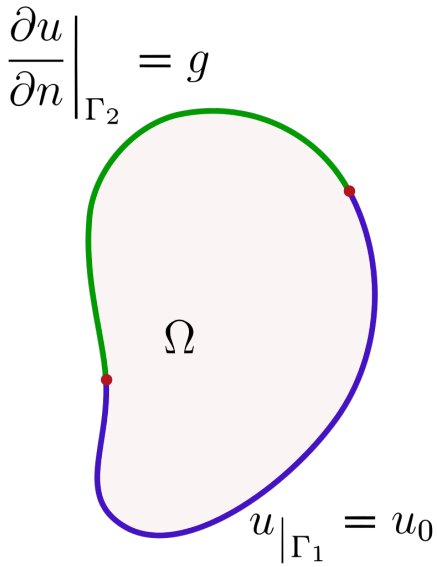
$$z = (x, y) \in \mathbb{R}^2, \ \Delta u(z) = r(z)$$

$$\Delta u(z) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

$$A = \left[\begin{array}{ccc|ccc|ccc}
4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\
\hline
-1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\
0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\
\hline
0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4
\end{array}\right]$$

Examples of boundary conditions

- Dirichlet: $u|_{\partial\Omega} = b$
- Neumann: $\frac{\partial u}{\partial z}|_{\partial\Omega} = b$
- Robin: $\alpha u|_{\partial\Omega} + \beta \frac{\partial u}{\partial z}|_{\partial\Omega} = b$



Limitations

Curse of the dimensionality

- 1D case: m points
- 2D case: m^2 points
- 3D case: m^3 pionts

The dimensionality of the problem increases exponentially with the dimension of the domain.

Nonlinearity?

- Cannot simply solve a linear system
- Need for an iterative method (sequentially linearize the problem)
- Requires to solve a lot of linear systems

A new perspective: neural networks for PDEs

- Idea: Approximate the solution of the problem by a neural network $NN(z) : \mathbb{R}^d \to \mathbb{R}$ such that

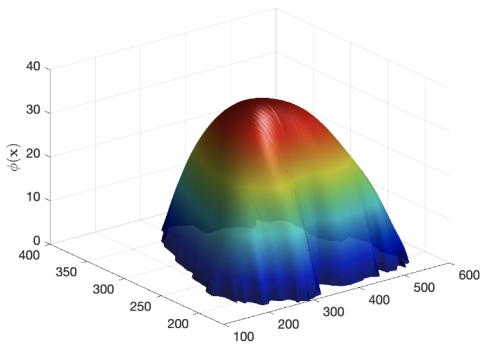
$$NN_\theta(z) \sim u(z), \qquad \text{for all } z \in \Omega$$

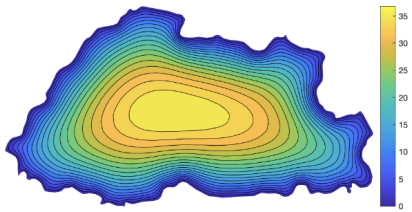
- The neural network is trained by minimization of a loss that takes into account the physical information contained in the PDE.
- A recent development: Physics Informed Neural Networks (PINNs)

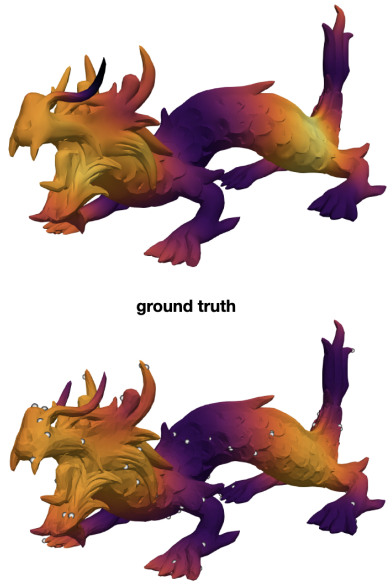
M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, 2019.

Why this approach ?

- Natural approach for nonlinear equations
- Provides analytic and continuously differentiable expression of the approximate solution
- The solution is meshless, well suited for problems with complex geometries
- The training is highly parallelizable on GPU
- Allows to alleviate the effect of the curse of dimensionality
- These techniques are easy to implement and to adapt to different kind of differential equations and boundary conditions.



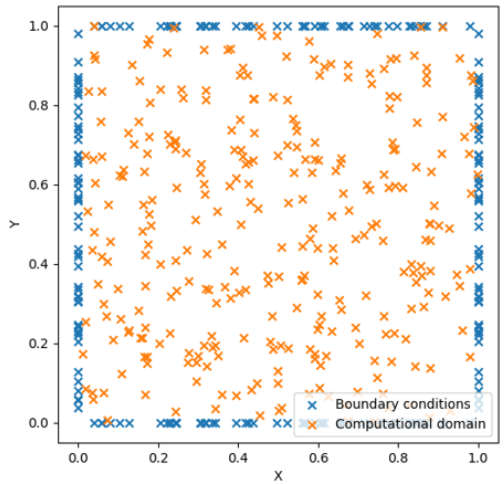




How to train a PINN

Step 1: build a sampling set

$z = (x, y) \in \mathbb{R}^2 \rightarrow z_\Omega, z_{\partial\Omega}, |z_\Omega| = N_\Omega, |z_{\partial\Omega}| = N_{\partial\Omega}.$



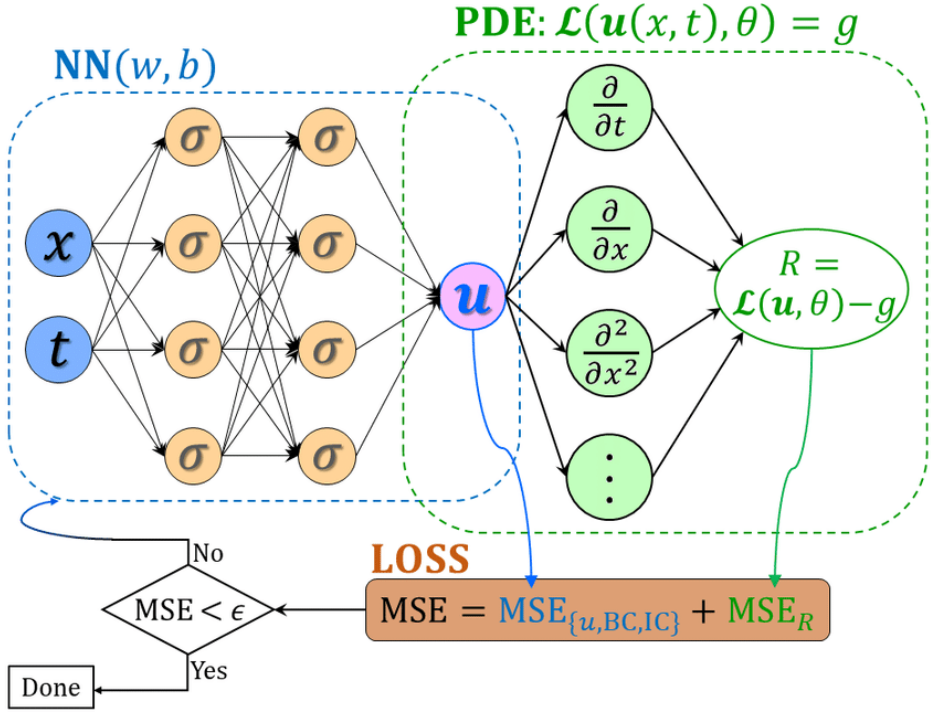
How to train a PINN

Step 2: define the loss function

$$\ell(\theta) = \underbrace{\frac{\lambda_\Omega}{N_\Omega}\sum_{j=1}^{N_\Omega}(\mathcal{L}(NN_\theta(z_\Omega^j)) - g(z_\Omega^j))^2}_{MSE_R} + \underbrace{\frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{j=1}^{N_{\partial\Omega}}(\mathcal{B}(NN_\theta(z_{\partial\Omega}^j)) - b(z_{\partial\Omega}^j))^2}_{MSE_B}$$

where λ_Ω, $\lambda_{\partial\Omega}$ are some positive weights which balance the contribution of the residual of the PDE and the residual of the boundary conditions.

Physics Informed Neural Networks (PINNs)



Step 3: minimize the loss function wrt θ

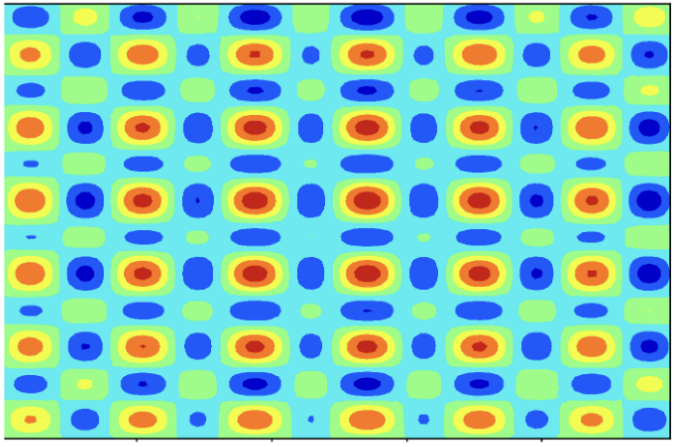
SGD, SGD + momentum, LBFGS

Current research on PINNs

1) Approximation properties

Estimates on the generalization error of Physics Informed NeuralNetworks (PINNs) for approximating PDEs. S. Mishra and R. Molinaro, 2021

2) On the spectral bias of neural networks



On the Spectral Bias of Neural Networks

Nasim Rahaman [*1 2] Aristide Baratin [*1] Devansh Arpit [1] Felix Draxler [2] Min Lin [1] Fred A. Hamprecht [2]
Yoshua Bengio [1] Aaron Courville [1]

WHEN AND WHY PINNS FAIL TO TRAIN: A NEURAL TANGENT KERNEL PERSPECTIVE

A PREPRINT

Sifan Wang
Graduate Group in Applied Mathematics
and Computational Science
University of Pennsylvania
Philadelphia, PA 19104
sifanw@sas.upenn.edu

Xinling Yu
Graduate Group in Applied Mathematics
and Computational Science
University of Pennsylvania
Philadelphia, PA 19104
xlyu@sas.upenn.edu

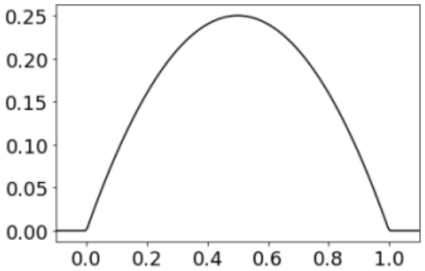
Paris Perdikaris
Department of Mechanichal Engineering
and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104
pgp@seas.upenn.edu

$\Rightarrow$ PINNs are not effective in approximating highly oscillatory solutions

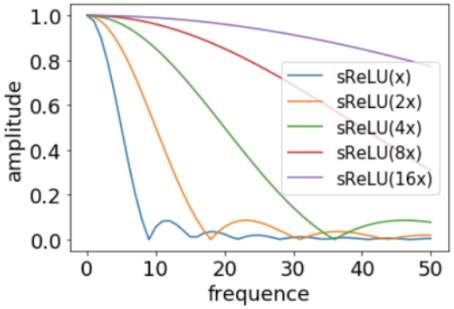
Mscale networks

Z.Q. Liu, W. Cai, and Z.Q. John Xu, Multi-scale Deep Neural Network (MscaleDNN) forSolving Poisson-Boltzmann Equation in Complex Domains, 2020

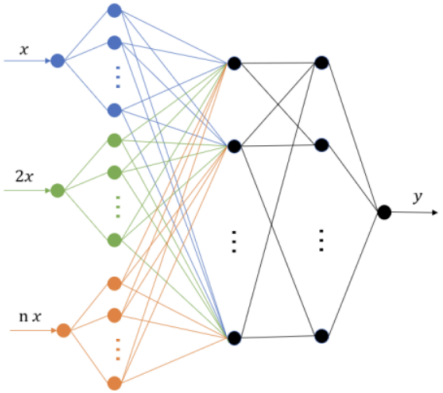
Idea: simultaneous training of frequency-selective subnetworks



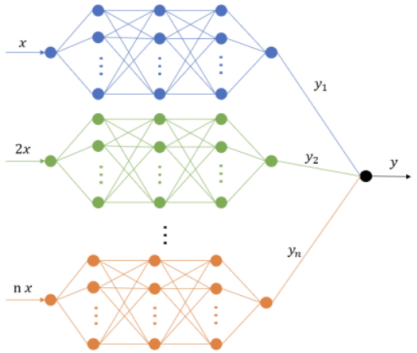
(b) sReLU



(a) sReLU



(a) MscaleDNN-1



(b) MscaleDNN-2

Recurrent neural networks (RNNs)

- Designed to handle data sources that are sequential in nature
- Examples: Translation of text, speech recognition (transforming spoken words into text), classification of events at every point in a movie
- Humans don't start their thinking from scratch every second.
- Example: while reading you understand each word based on your understanding of previous words.
- RNNs: networks with loops in them, allowing information to persist, connecting previous information to the present task

Structure of a RNN

- Input object X is a sequence
- Example: a document $X = \{X_1, X_2, \ldots, X_L\}$, where each X_l represents a word
- The output Y can be a sequence (language translation), or a scalar (binary sentiment label of a movie review document)

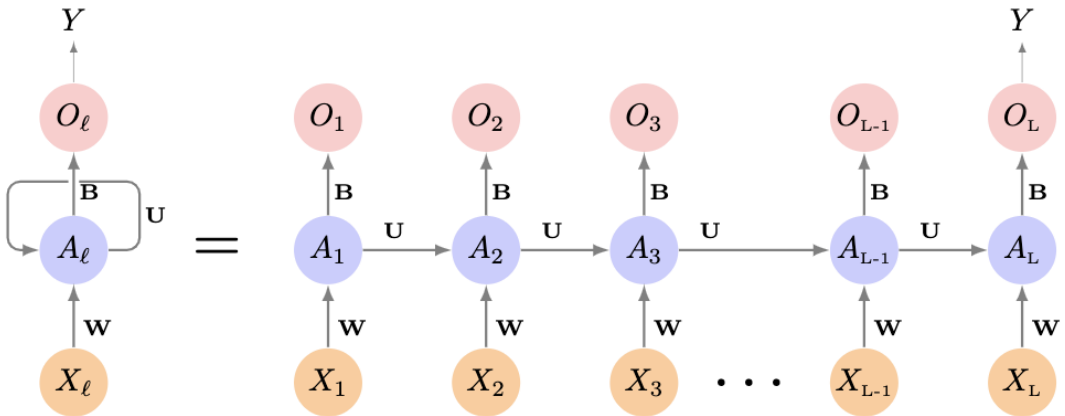
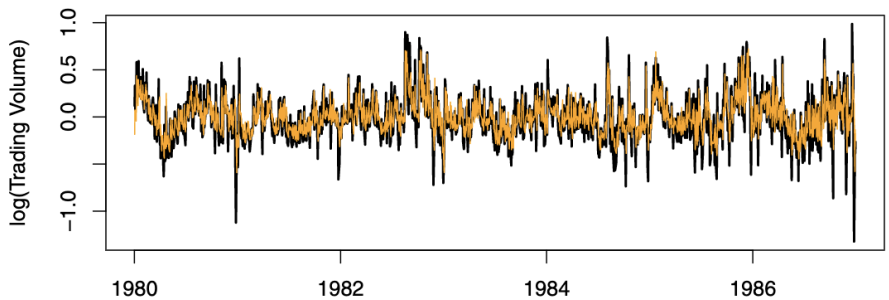


FIGURE 10.12. *Schematic of a simple recurrent neural network. The input is a sequence of vectors* $\{X_\ell\}_1^L$*, and here the target is a single response. The network processes the input sequence* X *sequentially; each* X_ℓ *feeds into the hidden layer, which also has as input the activation vector* $A_{\ell-1}$ *from the previous element in the sequence, and produces the current activation vector* A_ℓ*. The same collections of weights* $\mathbf{W}$*,* $\mathbf{U}$ *and* $\mathbf{B}$ *are used as each element of the sequence is processed. The output layer produces a sequence of predictions* O_ℓ *from the current activation* A_ℓ*, but typically only the last of these,* O_L*, is of relevance. To the left of the equal sign is a concise representation of the network, which is* unrolled *into a more explicit version on the right.*

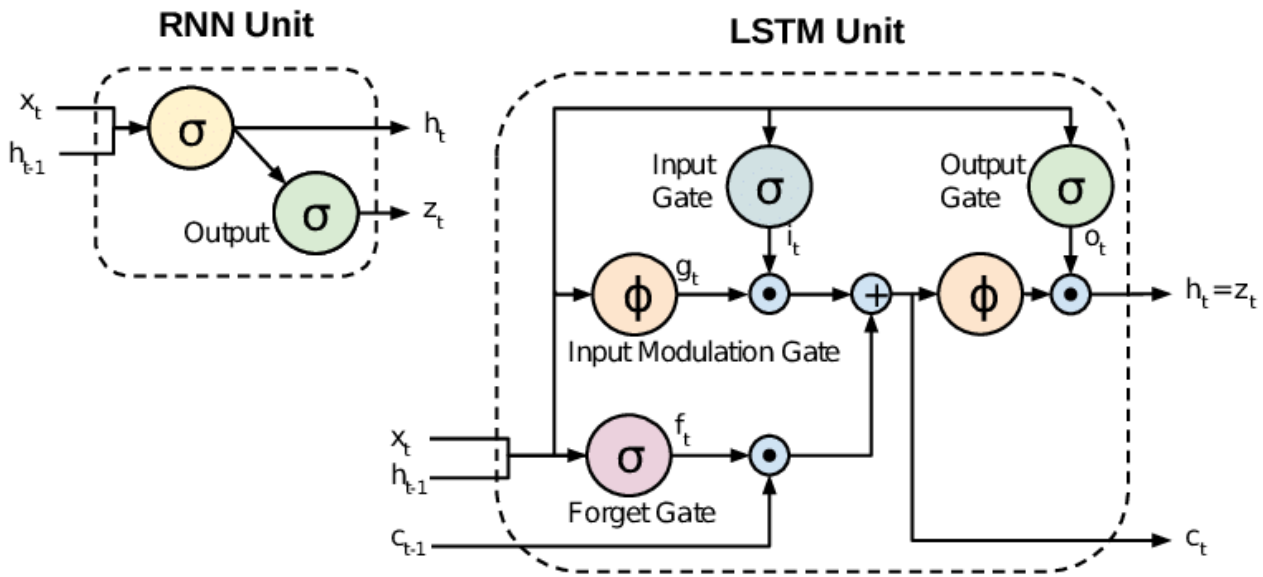
Structure of a RNN

- The same weights W, U, B are used for each element in the sequence
- This is a form of weight sharing
- The activations A_l accumulate a history of what has been seen before: the learned context can be used for prediction
- Example: Historical trading statistics from the New York Stock Exchange. We wish to predict trading volume on any day, given the history on all earlier days (Train: 1962-1980)



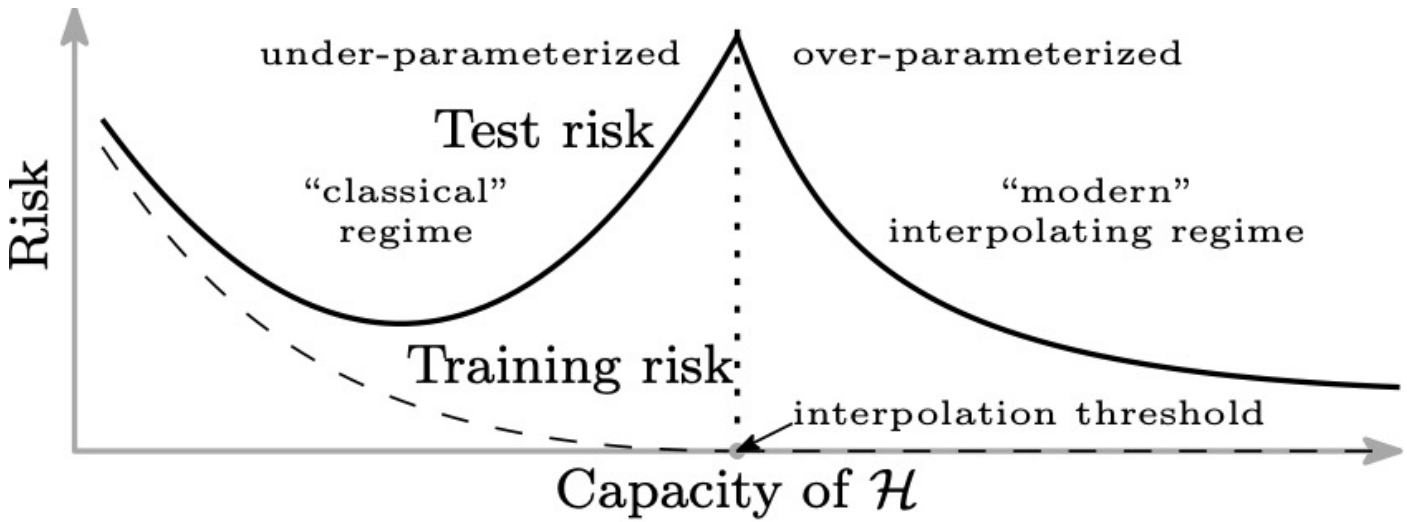
LSTM-RNN

- More elaborate versions use long term and short term memory (LSTM).
- Two tracks of hidden-layer activations are maintained
- A_l gets input from hidden units both further back in time and closer in time
- A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten.

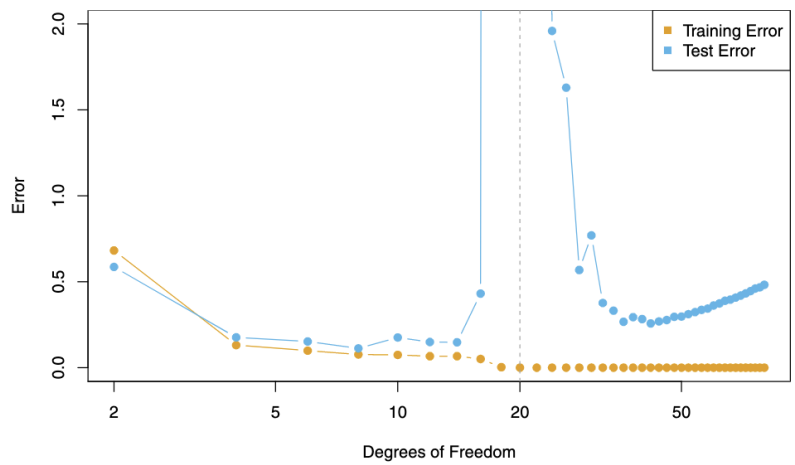


Interpolation and Double Descent, overparametrized networks

- Bias-variance trade-off: statistical learning methods tend to perform the best (test error) for an intermediate level of model complexity.
- Test error has a U-shape, training error decreases monotonically
- It is generally not a good idea to interpolate the training data
- Double descent phenomenon: in certain specific settings it is possible for a statistical learning method that interpolates the training data to perform well



- This is particularly true in problems with high signal-to-noise ratio, such as natural image recognition and language translation
- SGD has a natural bias: tends to select a "smooth" interpolating model that has good test-set performance on these kinds of problems
- The double-descent phenomenon does not contradict the bias-variance trade-off, it's a different regime
- Most of the statistical learning methods do not exhibit double descent (regularized models)
- Very good test-set performance can be obtained with good regularization, usually much better than with interpolation
- Huge number of parameters: we typically do not want to rely on this behavior.



NN: an always evolving topic

Some open research directions

- Introduce sparsity in neural networks
- NN training and inference in low/mixed precision
- Characterize approximation properties of NN
- New architectures: transformers (NLP), MIT's "Liquid" Machine-Learning System (adapts to changing conditions)
- New hardware: GPUs, FPGAs, TPUs