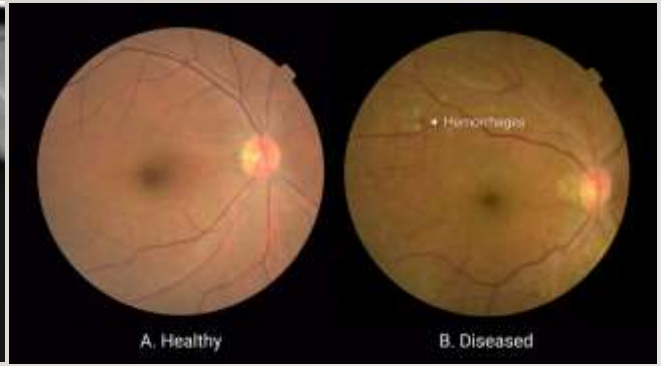
INTRODUCTION TO DEEP LEARNING

Vijay Dwivedi

DL IS POWERING MANY RECENT TECHNOLOGIES



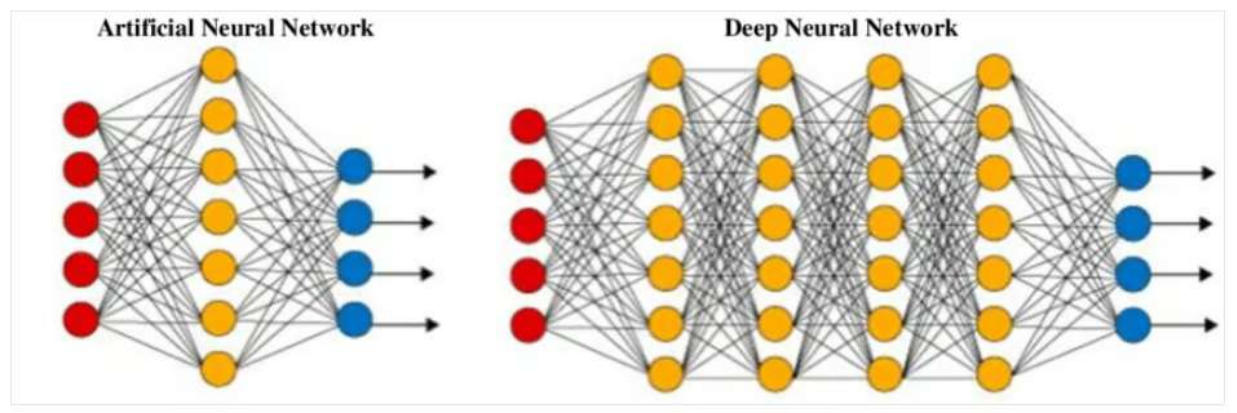
WHAT ARE DEEP NEURAL NETWORKS?

- An **Artificial Neural Network** (ANN) or a simple traditional neural network aims to solve trivial tasks with a straightforward network outline. An artificial neural network is loosely inspired from biological neural networks. It is tough for these networks to solve complicated image processing, computer vision, and natural language processing tasks***. (why?)***

- A **Deep Neural Networks** is a type of artificial neural network (**ANN**) with multiple layers between its input and output layers. Each layer consists of multiple nodes that perform computations on input data. It often have a complex hidden layer structure with a wide variety of different layers, such as a **convolutional layer, max-pooling layer, dense layer**, and other unique layers. These additional layers help the model to understand problems better and provide optimal solutions to complex projects. A deep neural network has more layers (more depth) than ANN and each layer adds complexity to the model while enabling the model to process the inputs concisely for outputting the ideal solution.

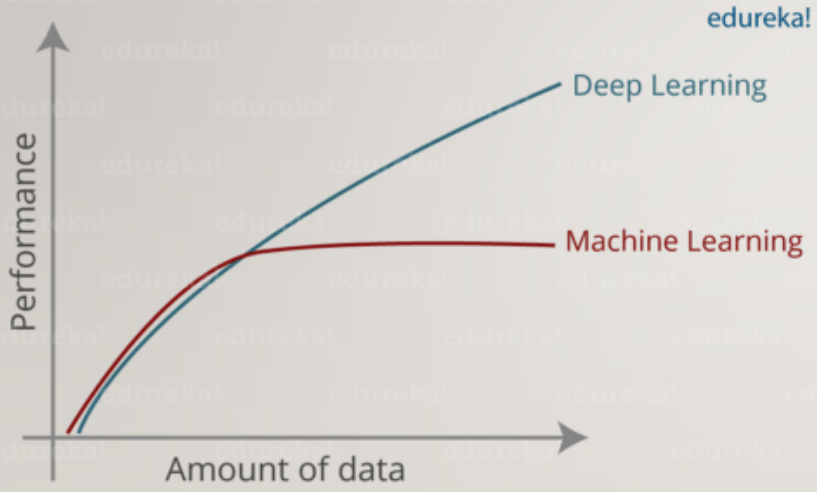
WHAT ARE DEEP NEURAL NETWORKS?

- After training a well-built deep neural network, they can achieve the desired results with high accuracy scores. They are popular in all aspects of deep learning, including computer vision, natural language processing, and transfer learning.
- The premier examples of deep neural networks are their utility in object detection with models such as **YOLO** (You Only Look Once), language translation tasks with **BERT** (Bidirectional Encoder Representations from Transformers) models, transfer learning models, such as **VGG-19, RESNET-50, efficient net,** and other similar networks for image processing projects.

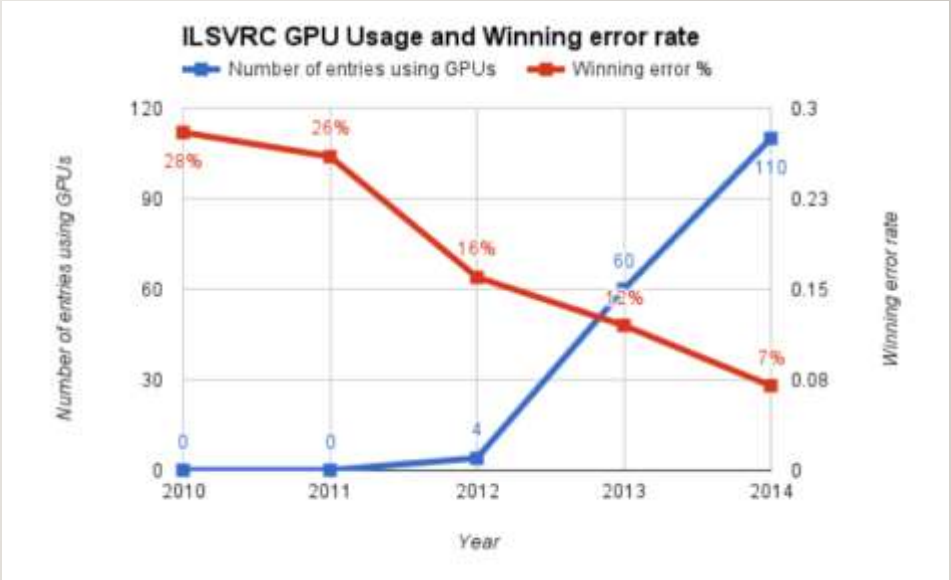


WHY DO NEURAL NETWORKS FINALLY WORK NOW?

1) Data: large curated datasets



2) GPUs: linear algebra accelerators



3) Algorithmic advances: optimizers, regularization, normalization … etc.

DEEP FORWARD NEURAL NETWORKS (DNNS)

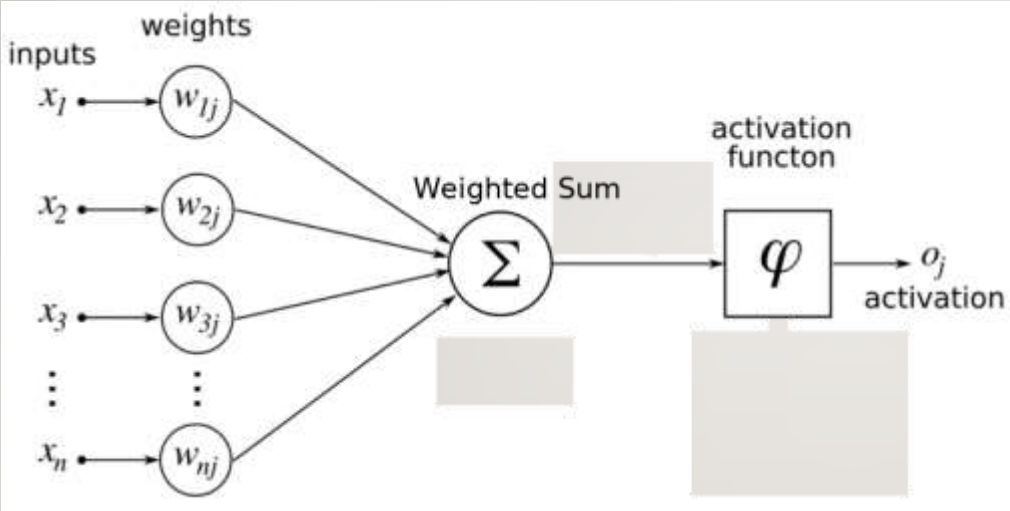
The objective of NNs is to approximate a function:

$$y = f^*(x)$$

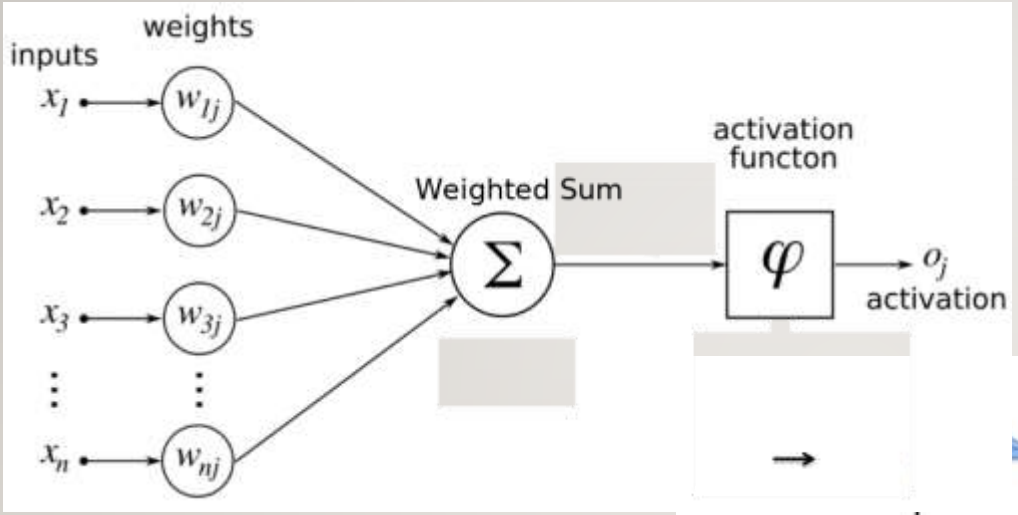
The NN learns an approximate function $y = f(x; W)$ with parameters W. This approximator is hierarchically composed of simpler functions

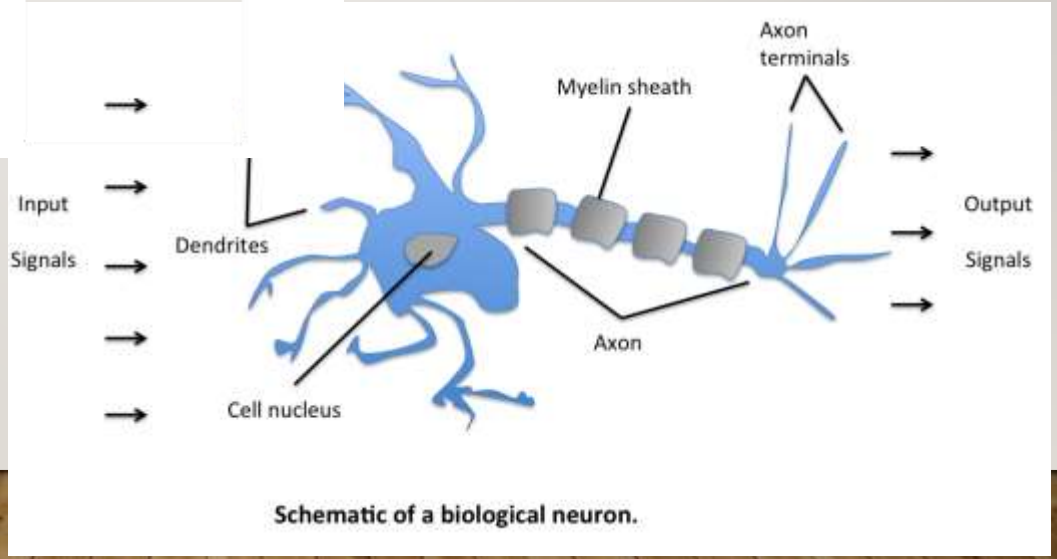
$$y = f^n(f^{n-1}(\cdots f^2(f^1(x))\cdots))$$

ACTIVATION FUNCTIONS



ACTIVATION FUNCTIONS



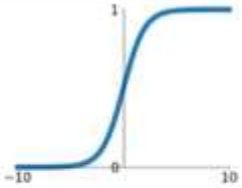


Schematic of a biological neuron.

ACTIVATION FUNCTIONS

Sigmoid

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

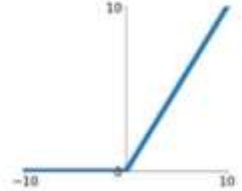


tanh

$$\tanh(x)$$

ReLU

$$\max(0, x)$$



Leaky ReLU

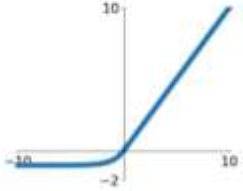
$$\max(0.1x, x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

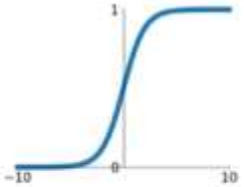
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ACTIVATION FUNCTIONS

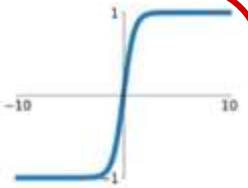
Sigmoid

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



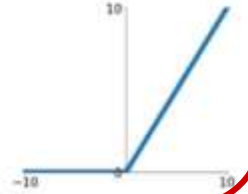
tanh

$$\tanh(x)$$



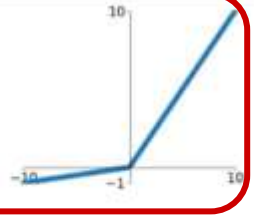
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

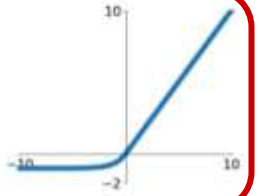


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

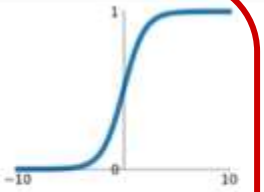


Most commonly used in modern networks

ACTIVATION FUNCTIONS

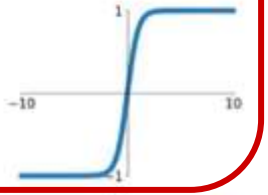
Sigmoid

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



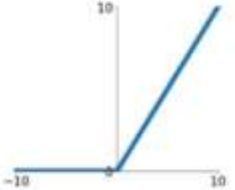
tanh

$$\tanh(x)$$



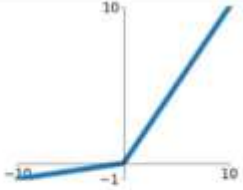
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

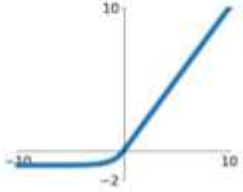


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

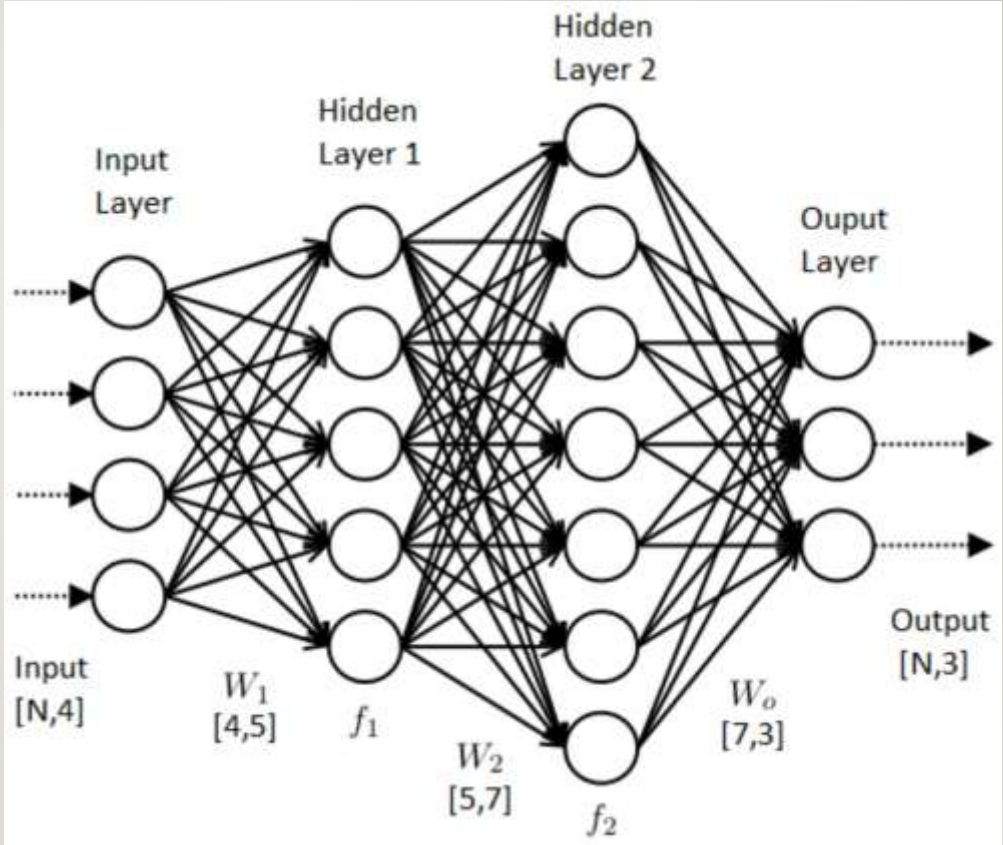
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Often used for output layers

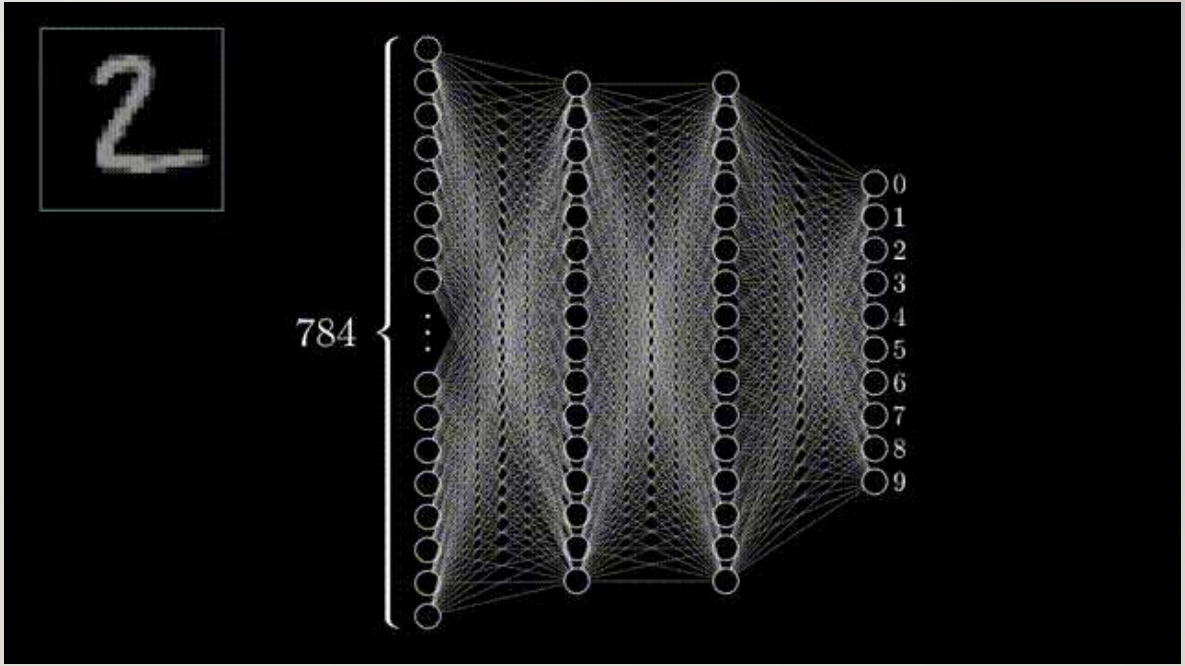
DEEP FORWARD NEURAL NETWORKS (DNNS)

$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$



DEEP FORWARD NEURAL NETWORKS (DNNS)

$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$



Animation adapted from: youtube.com/watch?v=aircAruvnKk&feature

COST FUNCTION & LOSS

To optimize the network parameters for the task at hand we build a cost function on the training dataset:

$$J(W) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(f(x; W), y)$$

COST FUNCTION & LOSS

To optimize the network parameters for the task at hand we build a cost function on the training dataset:

$$J(W) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(f(x; W), y)$$

Most NNs are trained using a maximum likelihood (i.e. find the parameters that maximize the probability of the training dataset):

$$J(W) = -\mathbb{E}_{x,y \sim \hat{p}_{data}} \log p_{model}(y|x)$$

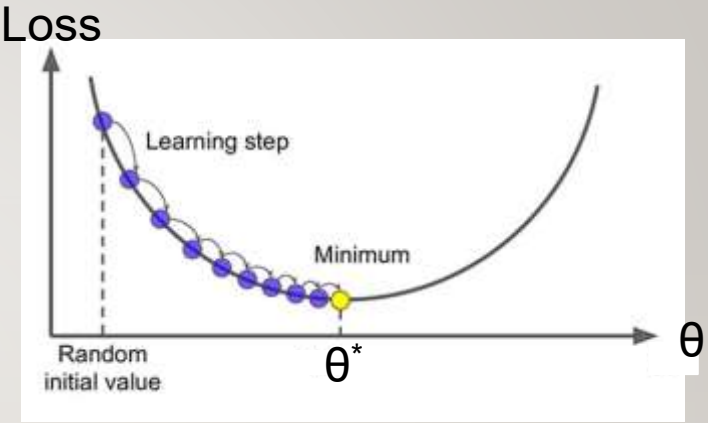
GRADIENT DESCENT

Gradient descent is the dominant method to optimize networks parameters θ to minimize loss function L(θ).

The update rule is (**α** is the “learning rate”):

$$W_{i+1} \leftarrow W_i - \alpha \nabla L(W)$$

The gradient is typically averaged over a minibatch of examples in minibatch **stochastic gradient descent.**



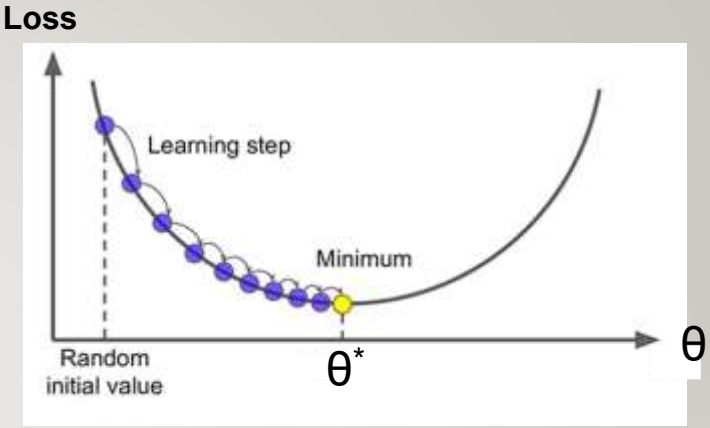
GRADIENT DESCENT

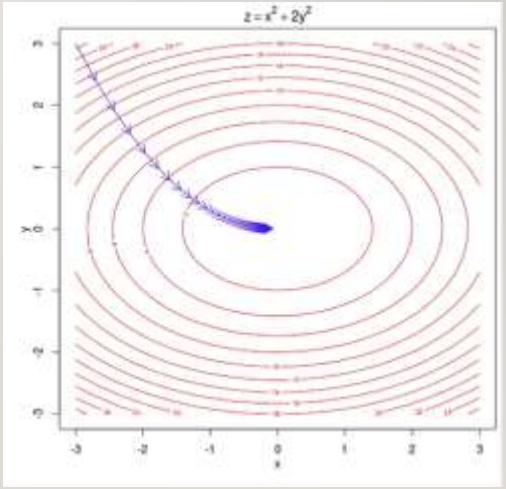
Gradient descent is the dominant method to optimize networks parameters θ to minimize loss function L(θ).

The update rule is (**α** is the “learning rate”):

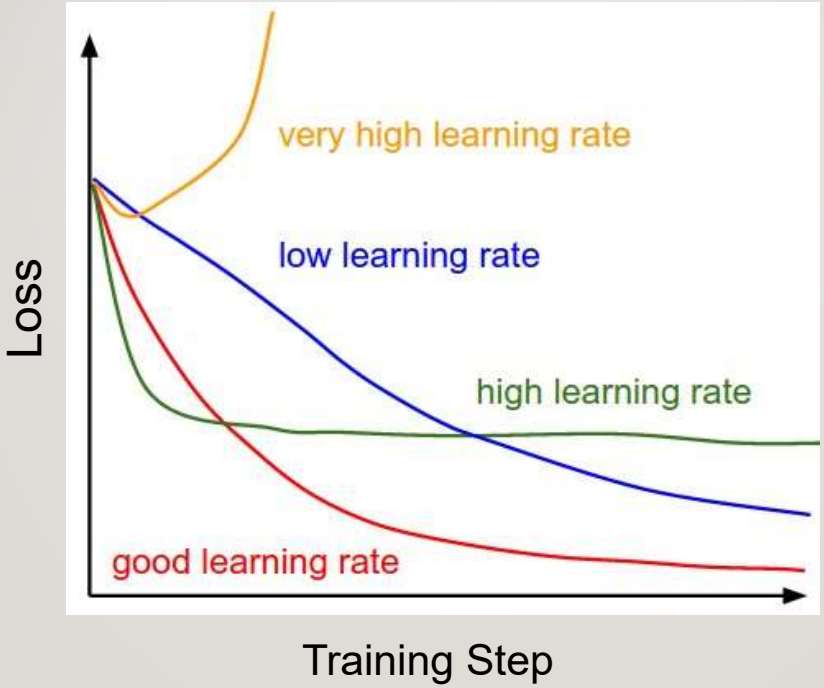
$$W_{i+1} \leftarrow W_i - \alpha \nabla L(W)$$

The gradient is typically averaged over a minibatch of examples in minibatch **stochastic gradient descent.**



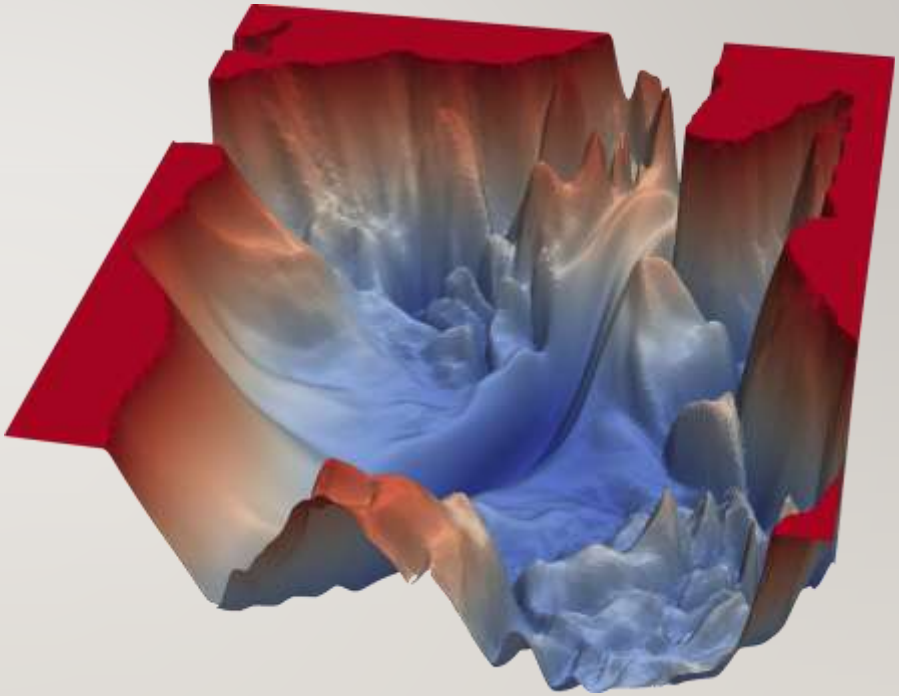


COST FUNCTION & LOSS



STOCHASTIC GRADIENT DESCENT VARIANTS

Gradient descent can get trapped in the abundant saddle points, ravines and local minimas of neural networks loss functions.

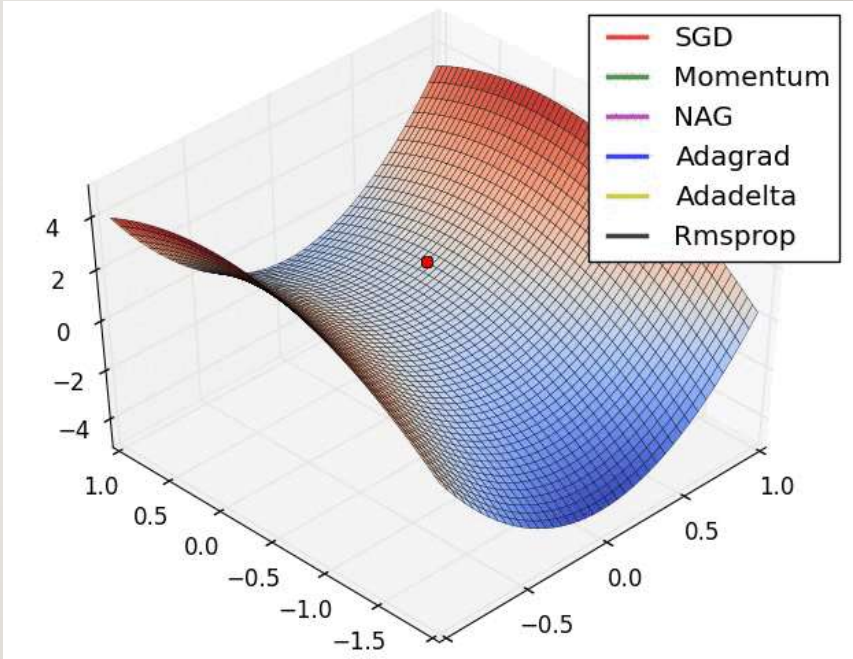


STOCHASTIC GRADIENT DESCENT VARIANTS

Gradient descent can get trapped in the abundant saddle points, ravines and local minimas of neural networks loss functions.

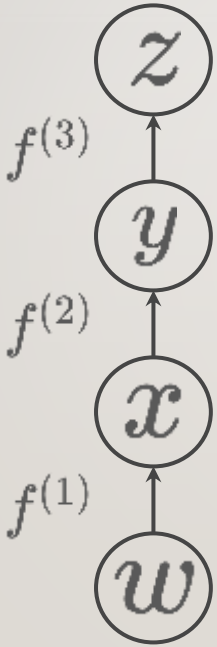
To accelerate the optimization on such functions we use a variety of methods:

- SGD + Momentum
- Nestrov
- AdaGrad
- RMSProp
- …
- Adam



BACKPROPAGATION

Updates to individual network parameters are propagated from the cost function through the network using the chain-rule of calculus. This is known is “backpropagation”.

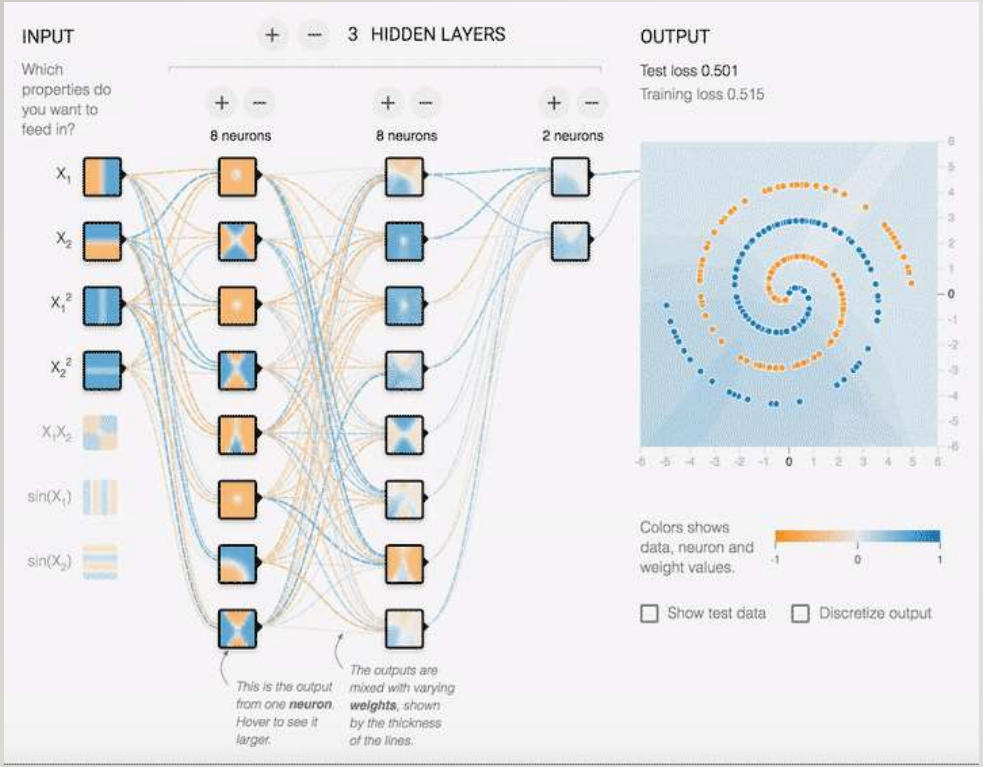


$$\frac{\partial z}{\partial w}$$

$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= f^{(3)\prime}(y)f^{(2)\prime}(x)f^{(1)\prime}(w)$$

NEURAL NETWORK TRAINING IN ACTION

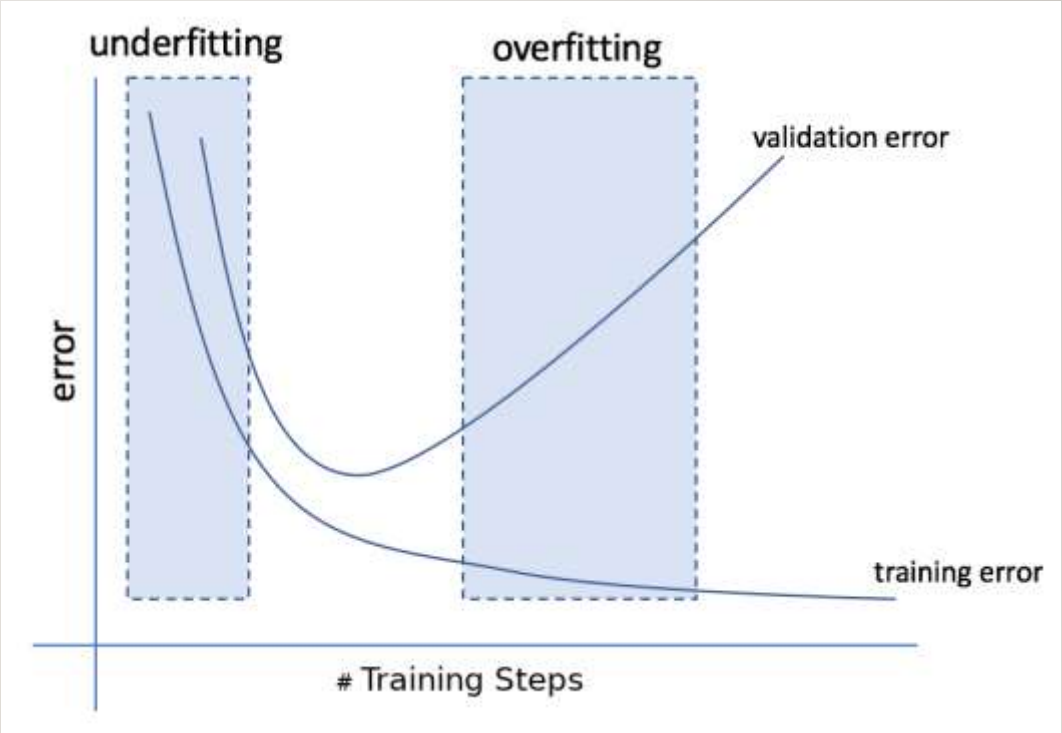


https://playground.tensorflow.org

REGULARIZATION: HOW TO TRAIN OVER-PARAMETERIZED NETWORKS?

Underfitting: training loss is high

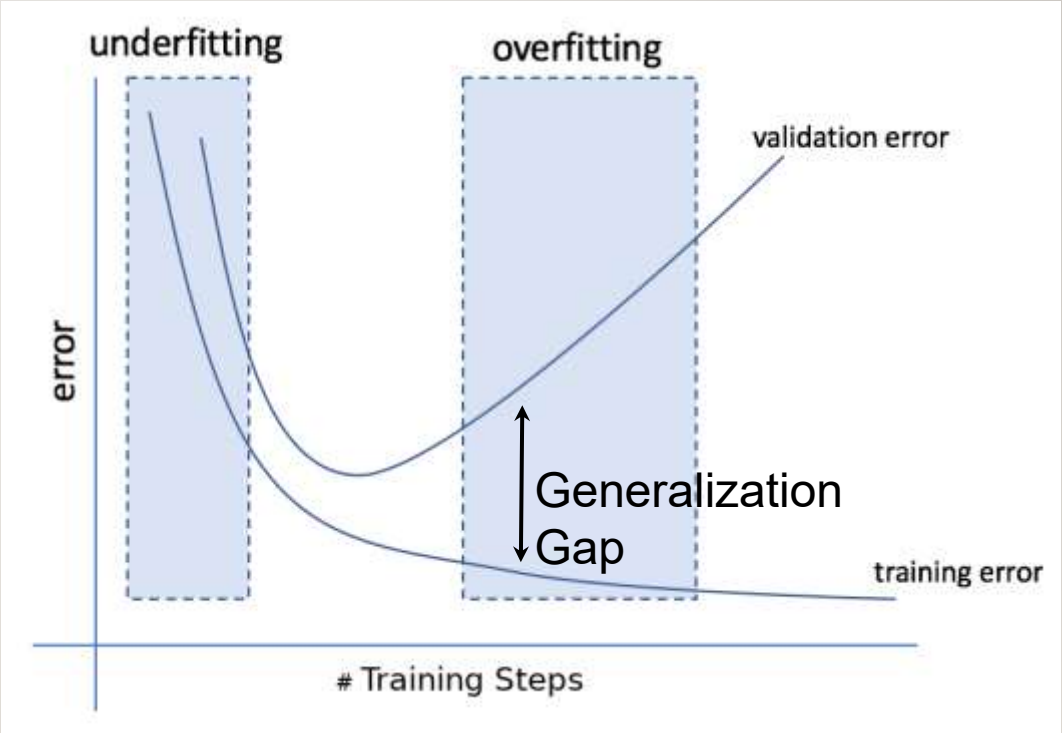
- check model architecture.
- check Learning Rate.
- train longer.
- check other hyper-parameters.



REGULARIZATION: HOW TO TRAIN OVER-PARAMETERIZED NETWORKS?

Overfitting: training loss is low, validation loss is high

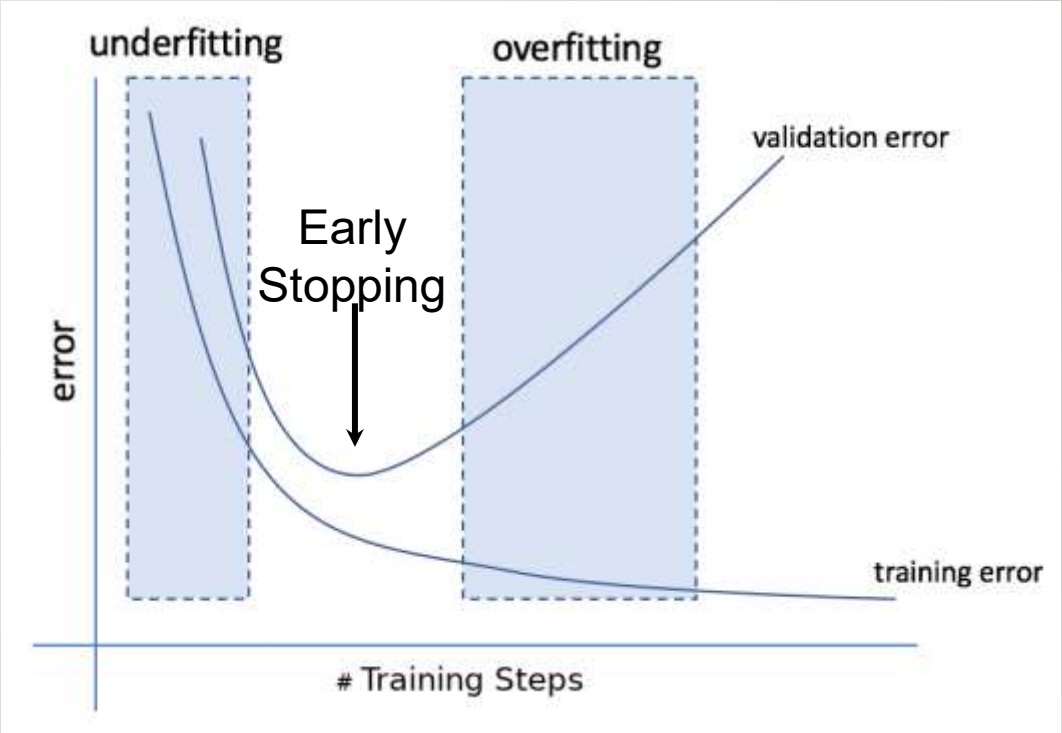
- Do you have enough data?
- Can you employ data augmentation?
- Learning-Rate tuning. Other hyper-parameters
- Regularization techniques …
- Reduce model complexity



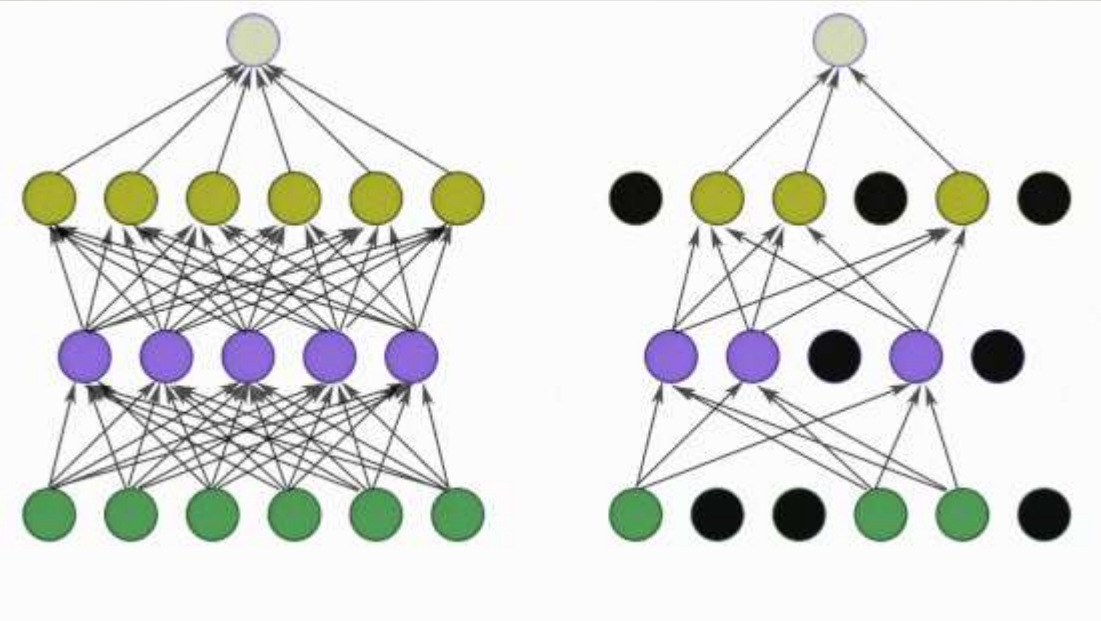
REGULARIZATION: HOW TO TRAIN OVER-PARAMETERIZED NETWORKS?

Overfitting: training loss is low, validation loss is high

- Do you have enough data?
- Can you employ data augmentation?
- Learning-Rate tuning. Other hyper-parameters
- Regularization techniques …
- Reduce model complexity

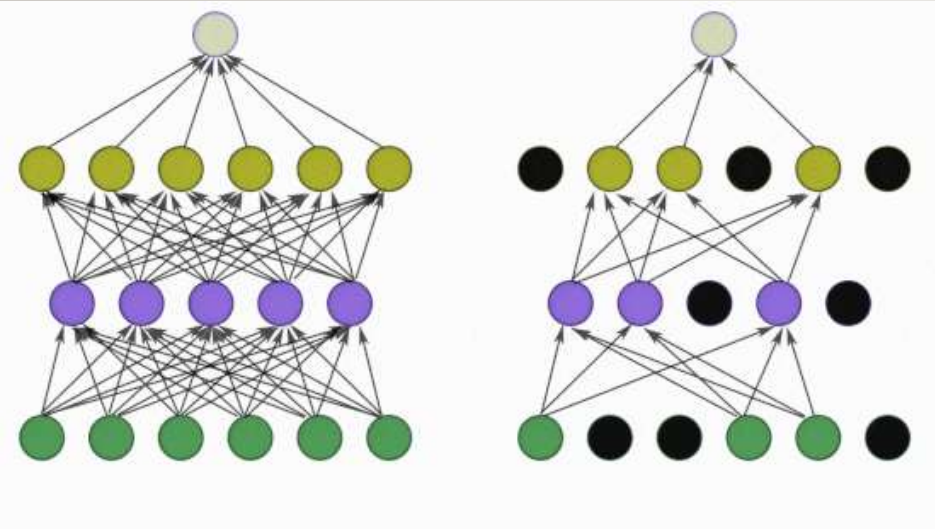


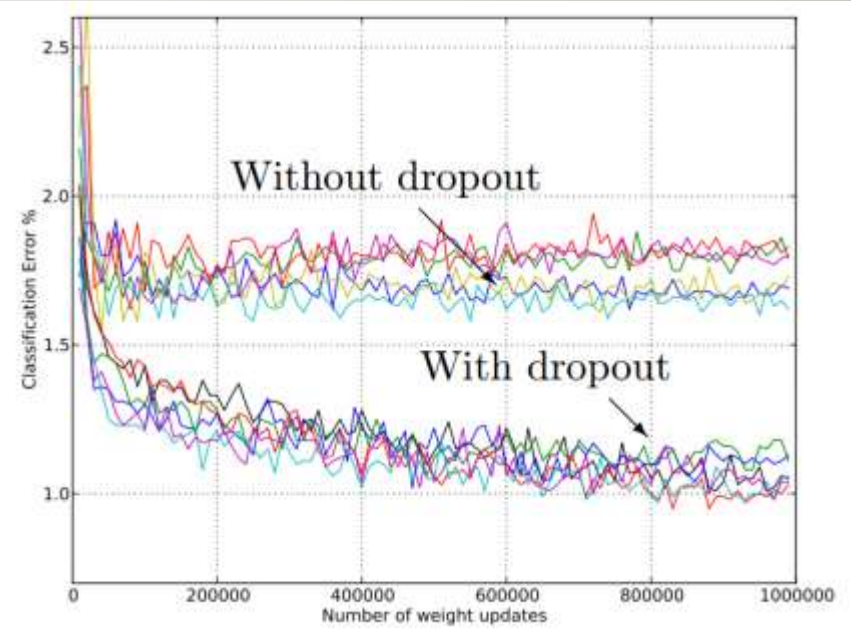
REGULARIZATION: HOW TO TRAIN OVER-PARAMETERIZED NETWORKS?



Dropout: randomly dropping out network connections with a fixed probability during training.

REGULARIZATION: HOW TO TRAIN OVER-PARAMETERIZED NETWORKS?





Dropout: randomly dropping out network connections with a fixed probability during training.

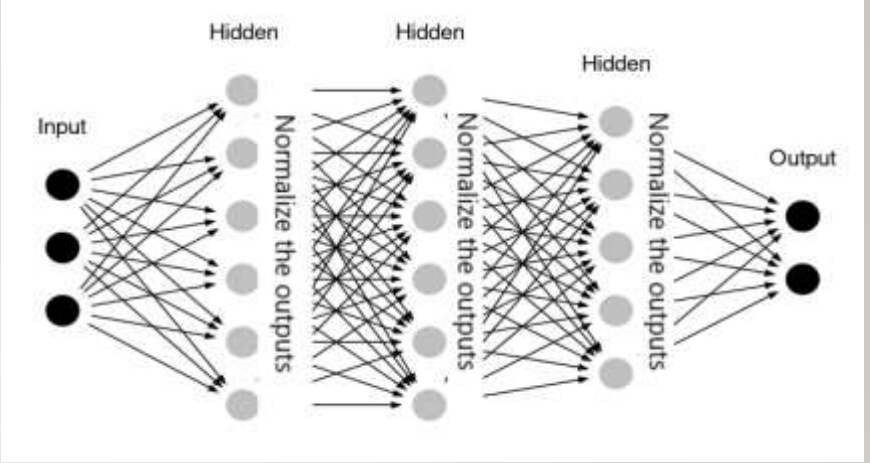
BATCH NORMALIZATION

BN normalizes mean and variance statistics of layers activations, batch-by-batch.

BN accelerates learning by making the optimization landscape much smoother.

BN makes NNs less sensitive to some hyperparameters.

BN is an implicit regularizes, it affects network capacity and generalization.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

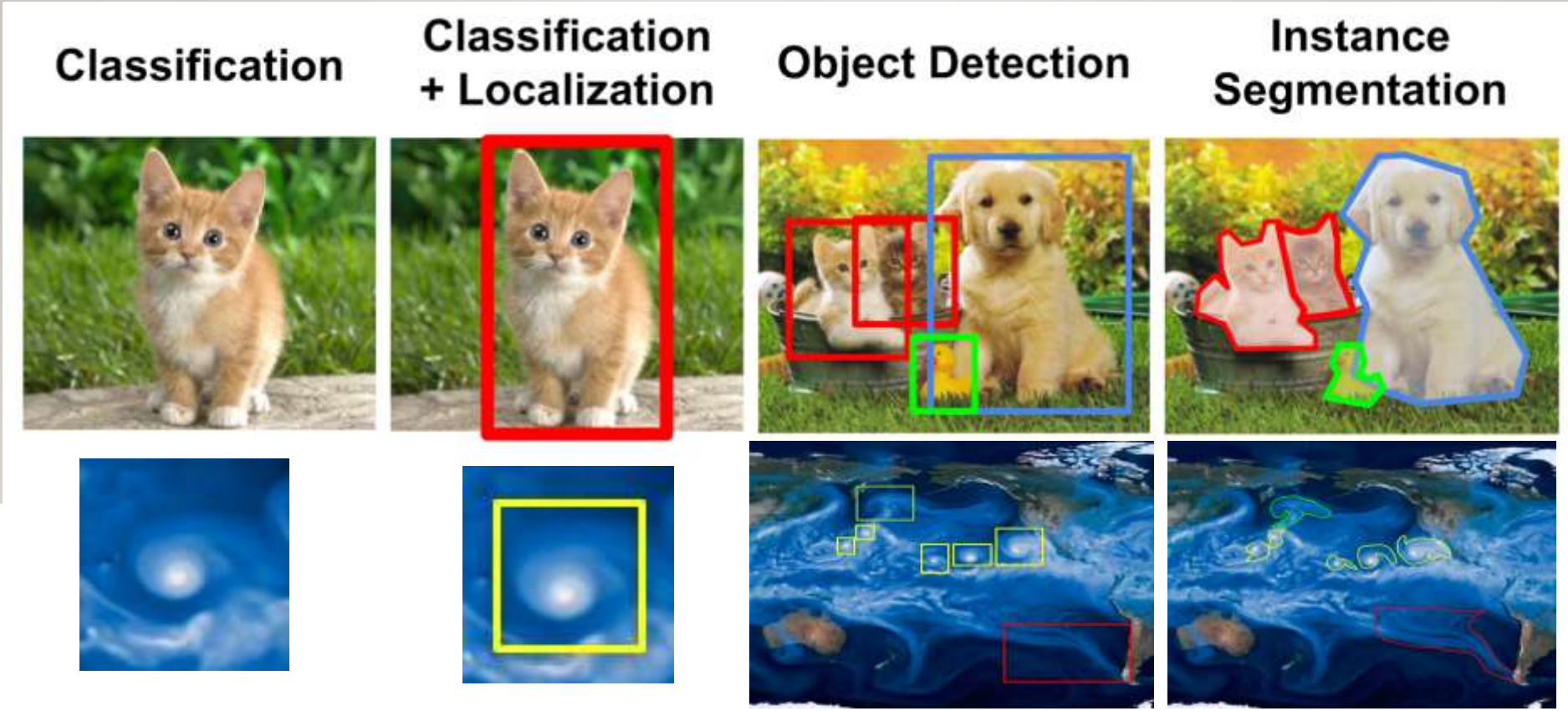
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

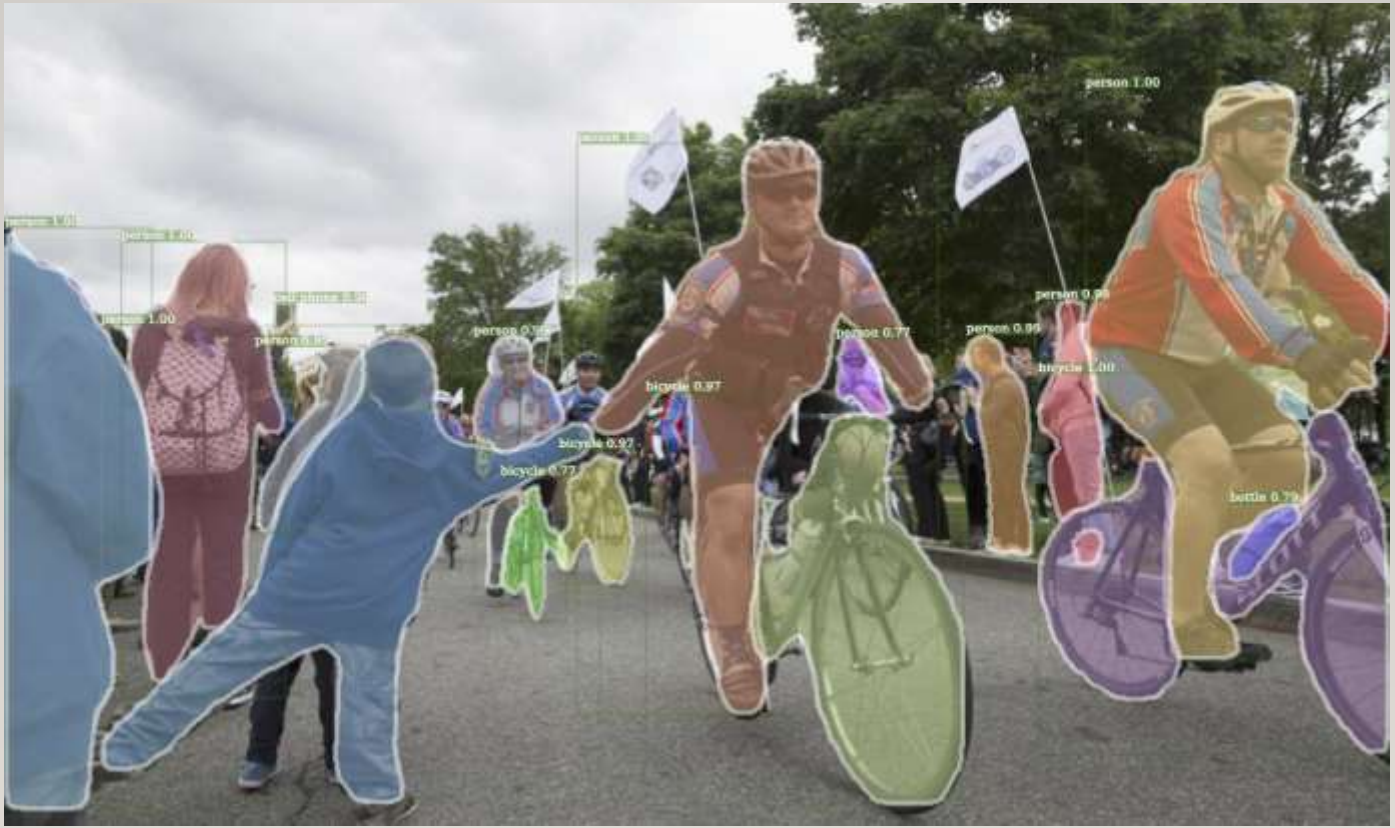
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

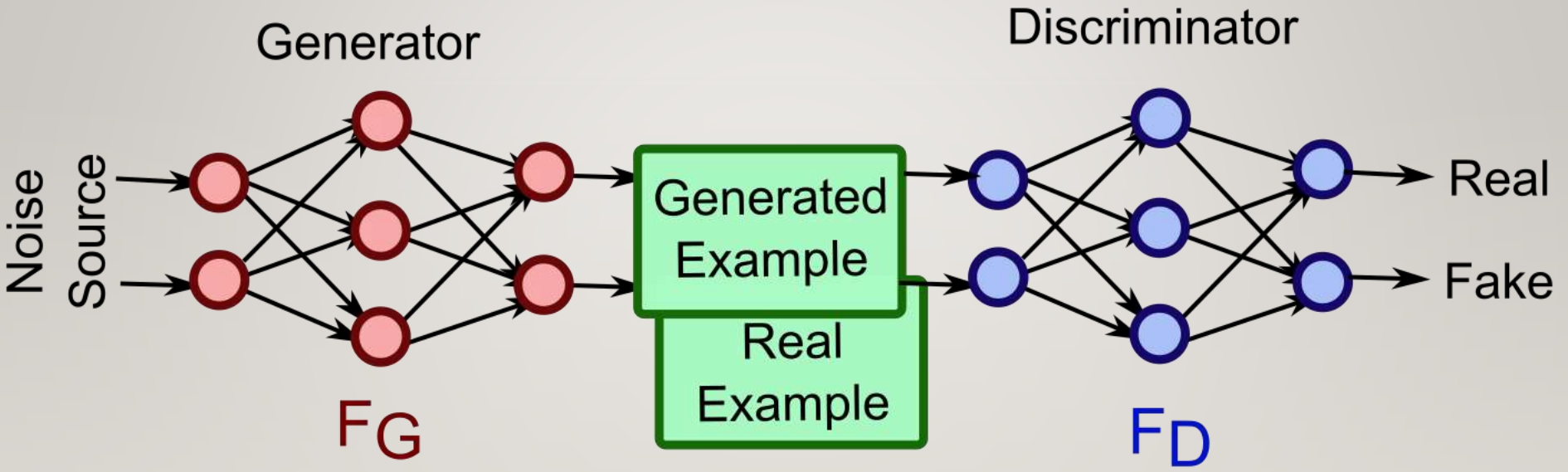
APPLICATIONS OF DNNS: CLASSIFICATION AND SEGMENTATION



APPLICATIONS OF DNNS: CLASSIFICATION AND SEGMENTATION



APPLICATIONS OF DNNS: GENERATIVE ADVERSARIAL NETWORKS



APPLICATIONS OF DNNS: GENERATIVE ADVERSARIAL NETWORKS

DNN (DEEPER) NETWORKS: CHALLENGES & SOLUTIONS

What is a DNN?

- A neural network with multiple hidden layers between input and output.
- Enables learning of complex patterns and hierarchies in data.

Why go deeper?

- Shallow networks struggle with high-level abstractions.
- Deeper networks can approximate highly non-linear functions.

Real-World Applications

- Image & speech recognition (CNNs, RNNs)
- Language understanding (Transformers)
- Game playing (AlphaGo, RL agents)

Depth enables abstraction — but brings its own set of challenges

DNN (DEEPER) NETWORKS: CHALLENGES & SOLUTIONS

Vanishing/Exploding Gradients

- Gradients become too small or large during backpropagation
- Training slows or becomes unstable

Overfitting

- More parameters = risk of memorizing instead of generalizing
- Poor performance on unseen data

Computational Bottlenecks

- Training deep networks requires **more time**, **memory**, and **hardware acceleration**

Difficulty in Optimization

- Complex loss surfaces
- More susceptible to **local minima**, **plateaus**

Degradation Problem

- Adding more layers can **reduce accuracy**, not improve it
- Not always “the deeper, the better”

Depth enables abstraction — but brings its own set of challenges

VANISHING/EXPLODING GRADIENTS IN DNN

Vanishing Gradients

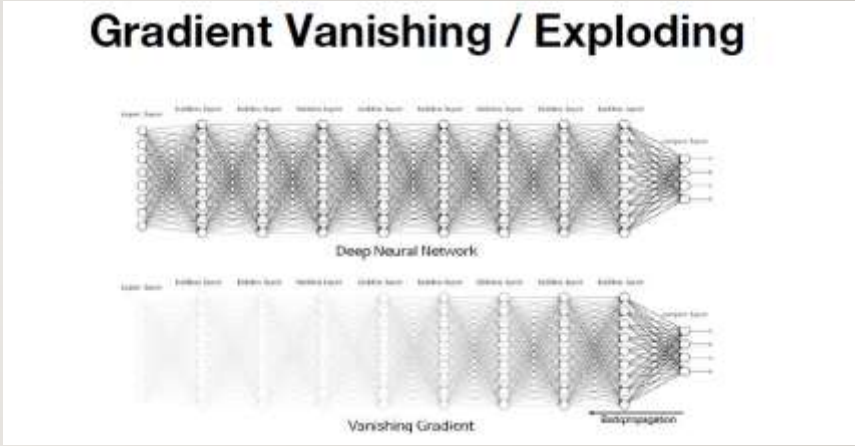
- During backpropagation, gradients become **too small**.
- Early layers update **very slowly** or not at all.
- Network “forgets” how to learn from earlier layers.

Exploding Gradients

- Gradients become **too large**, leading to **unstable weight updates**.
- Model may **diverge** during training.

Root Cause:

- Multiplying many small/large derivatives across layers
- Especially severe with **sigmoid/tanh** activations and **deep networks**

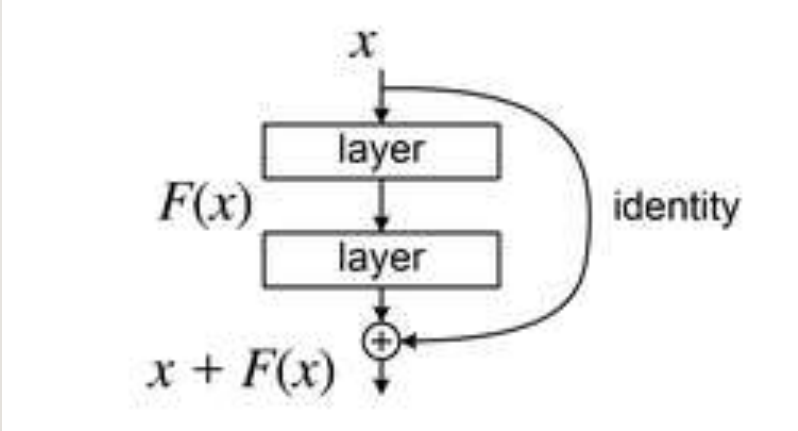


RESIDUAL CONNECTIONS : DEFINITION

What Is a Residual Block?

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called **Residual Blocks**. In this network, we use a technique called ***skip connections***. The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together.

The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping.



DEEP LEARNING APPROACH

Supervised learning

- Convolutional Networks
- Recurrent Networks (LSTM, GRU)

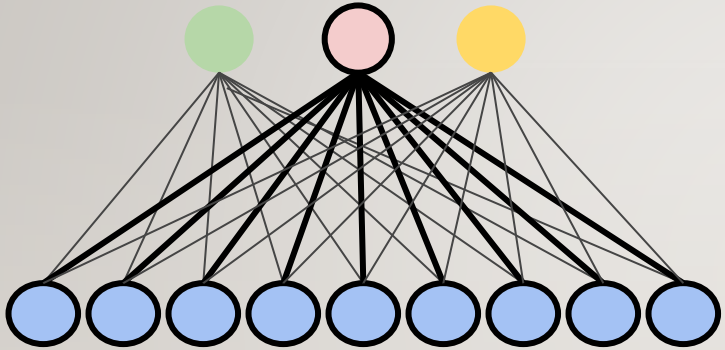
Unsupervised learning

- Autoencoders & Decoders
- GANs
- Restricted Boltzmann Machines (RBMs)
- Transformers

CONVOLUTIONAL NEURAL NETWORKS (CNNS) AND THE STORY OF DEPTH

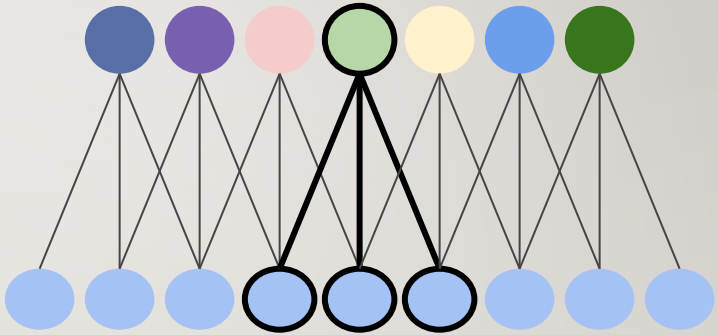
PARAMETER SHARING

Fully Connected (Dense) Layer



Every neuron is connected to all components of input vector.

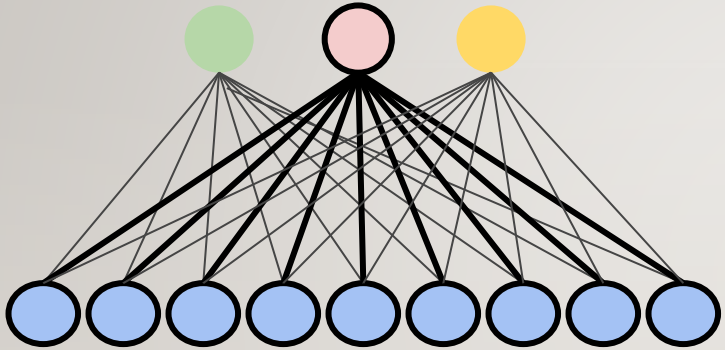
Sparse connectivity



In sparse connectivity, the output of every neuron is only affected by a limited input “receptive field”; 3 in this example.

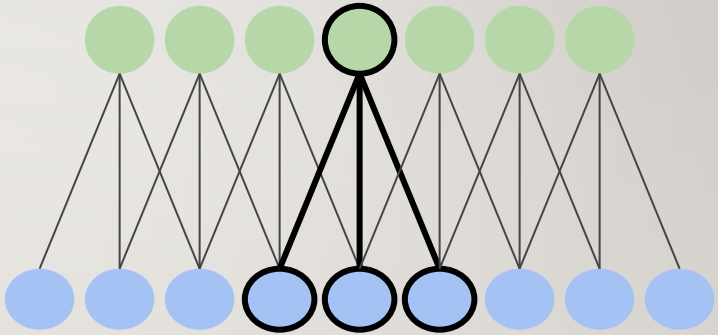
PARAMETER SHARING

Fully Connected (Dense) Layer



Every neuron is connected to all components of input vector.

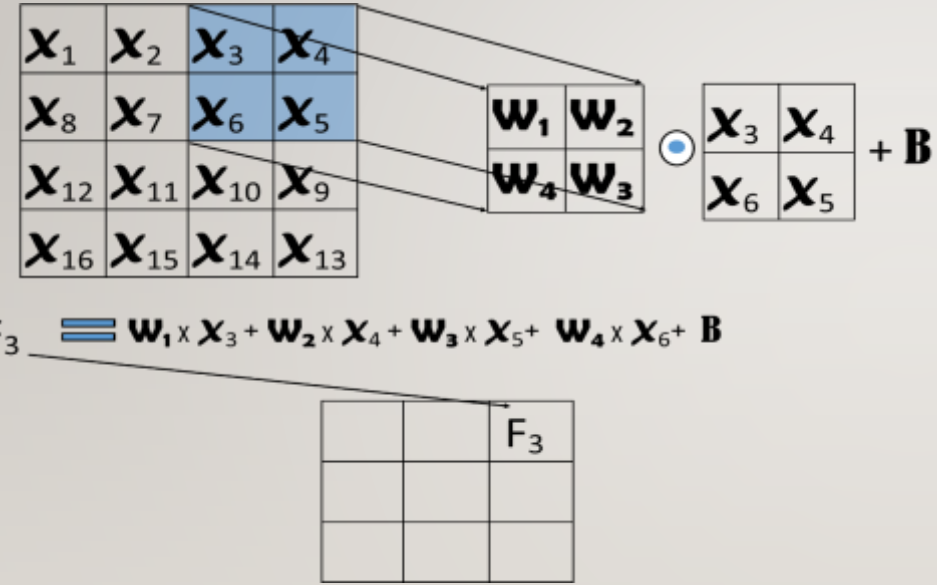
Sparse connectivity

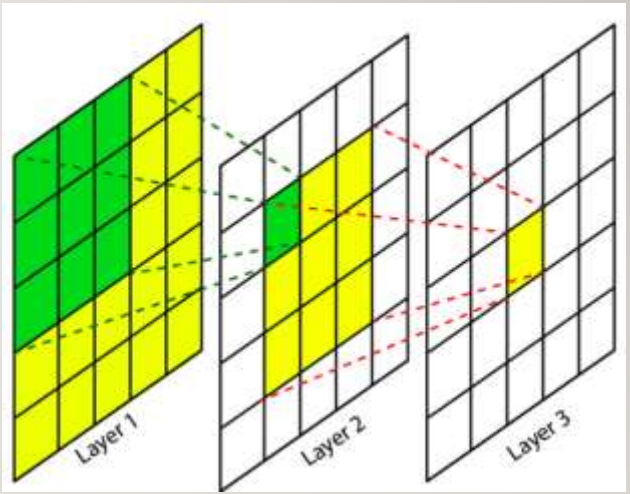


In sparse connectivity, the output of every neuron is only affected by a limited input “receptive field”; 3 in this example. Furthermore, the parameters of the function can be shared.

CONVOLUTIONAL NEURAL NETWORKS (CNNS)

CNNs implement the convolution operation over input.





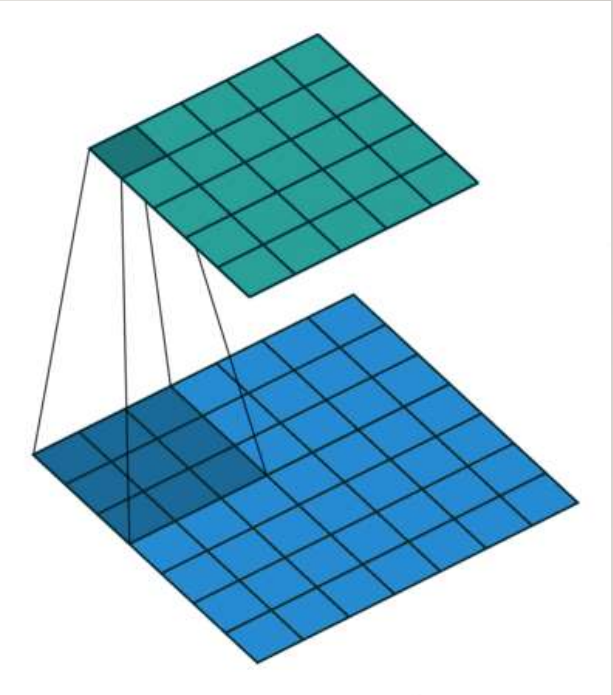
Receptive Field Concept

CONVOLUTIONAL NEURAL NETWORKS (CNNS)

CNNs implement the convolution operation over input.

CNNs are translation equivariant by construction.

CNNs achieve: sparse connectivity, parameter sharing and translation equivariance.



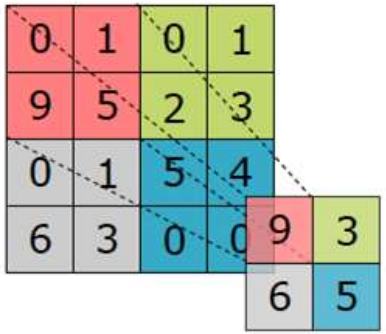
Sliding convolution kernel with size 3x3 over an input of 7x7.

POOLING

Pooling layers replace their input by a summary statistic of the nearby pixels.

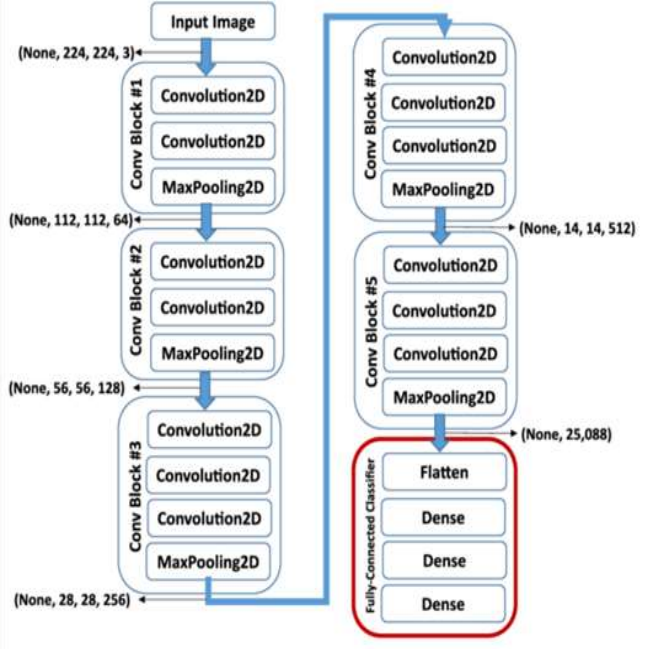
Max-pool and Avg-pool are the most common pooling layers.

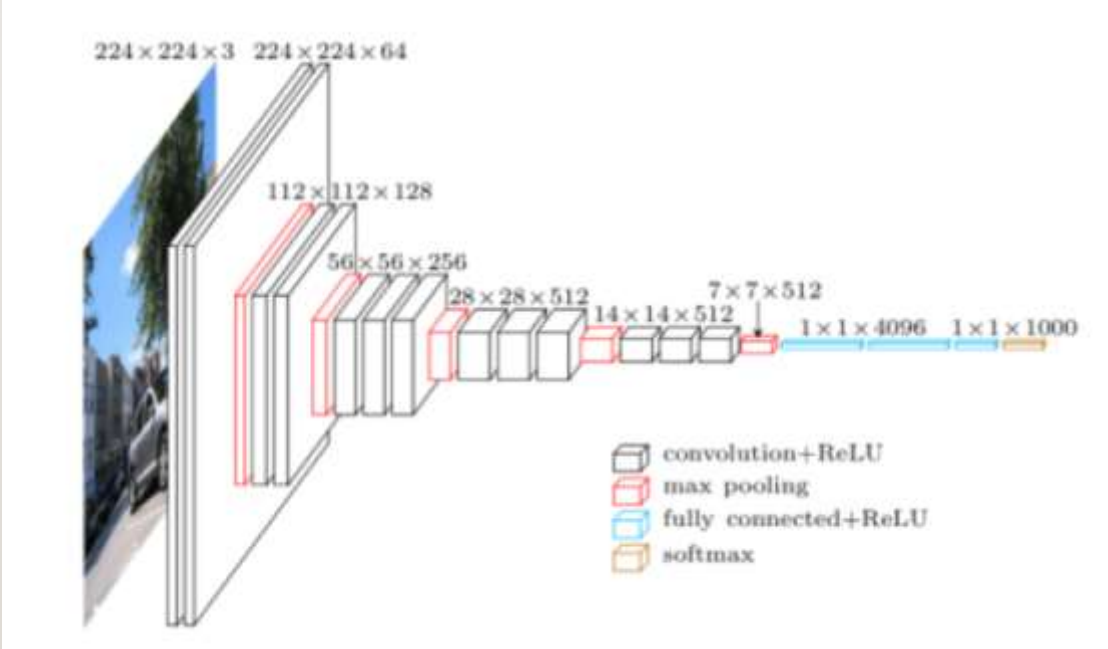
Max-Pool with kernel size 2



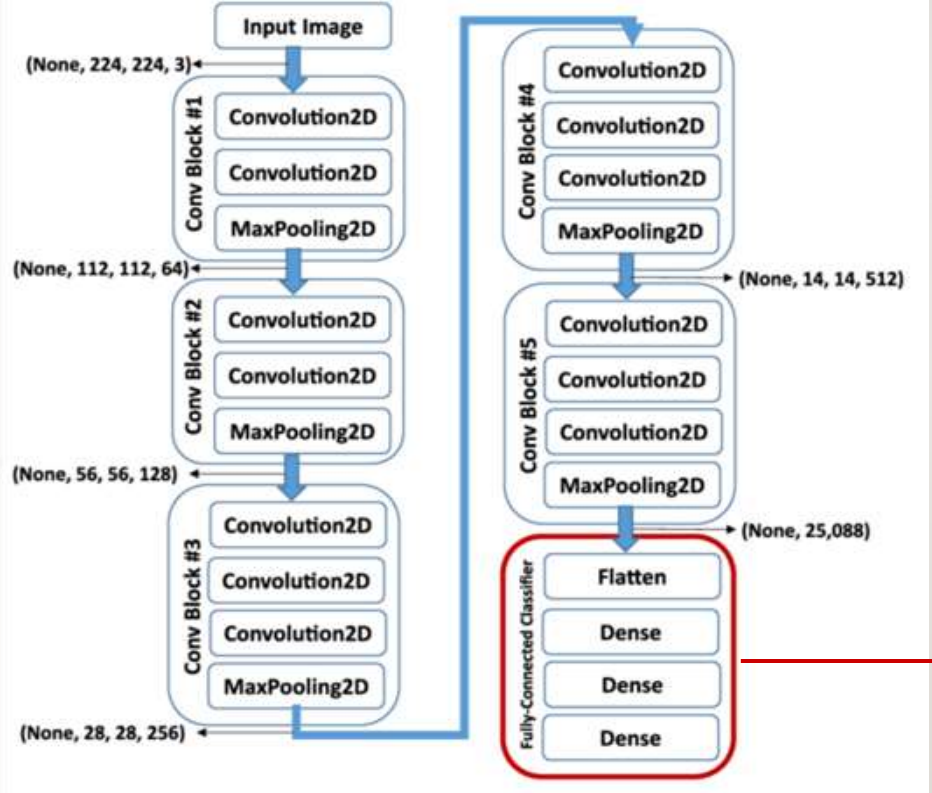
LET US PUT IT ALL TOGETHER: A TYPICAL CNN NETWORK ARCHITECTURE

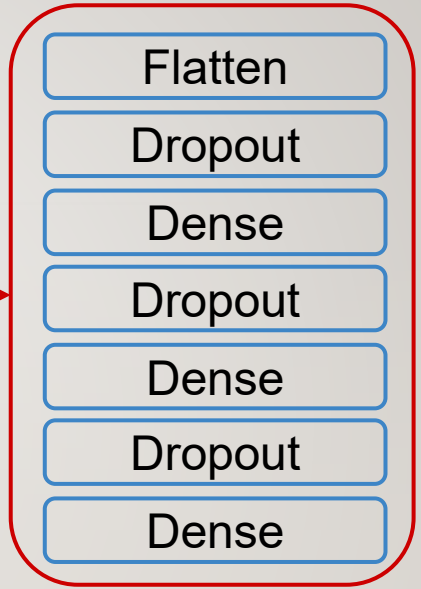
A schematic of VGG-16 Deep Convolutional Neural Network (DCNN) architecture trained on ImageNet (2014 ILSVRC winner)





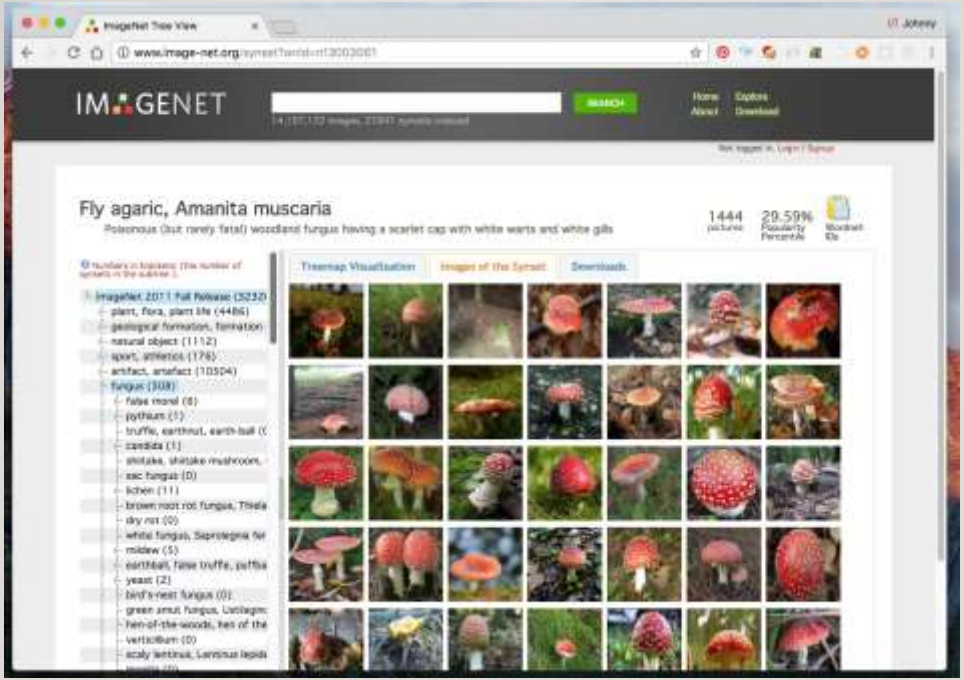
LET US PUT IT ALL TOGETHER: A TYPICAL CNN NETWORK ARCHITECTURE





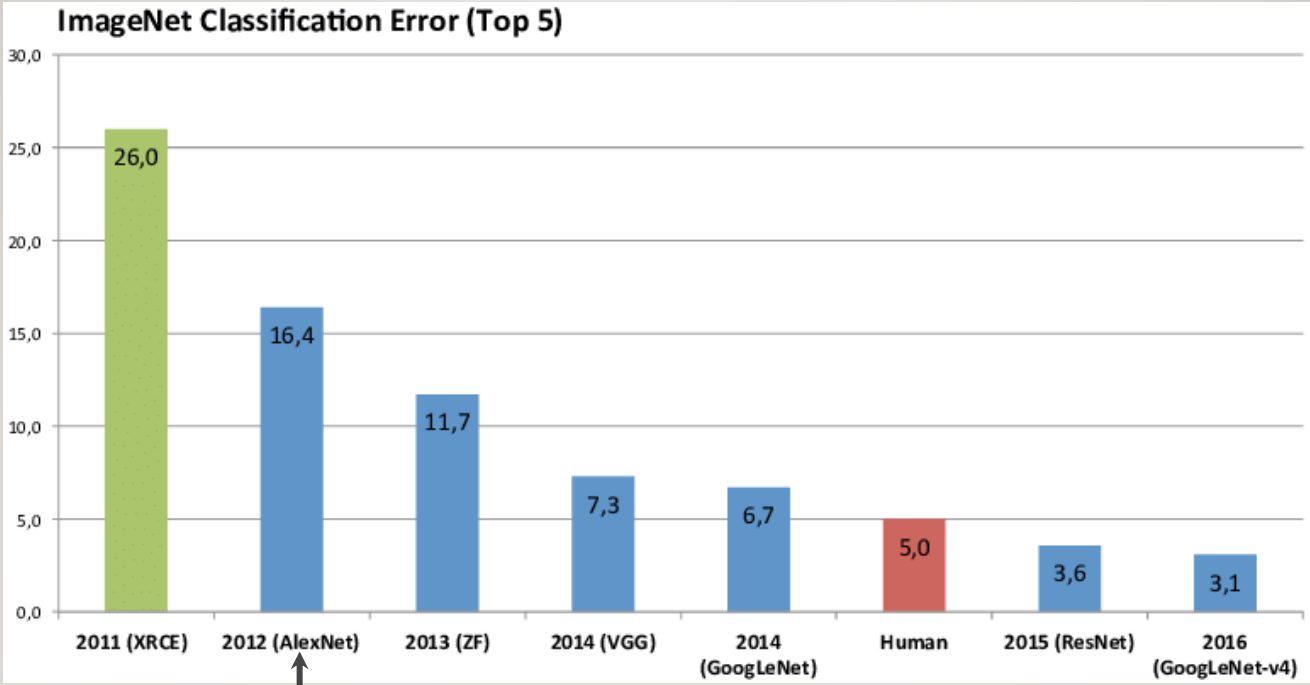
One possible configuration of dropout layers for regularization

CURATED DATASETS: (E.G. IMAGENET)



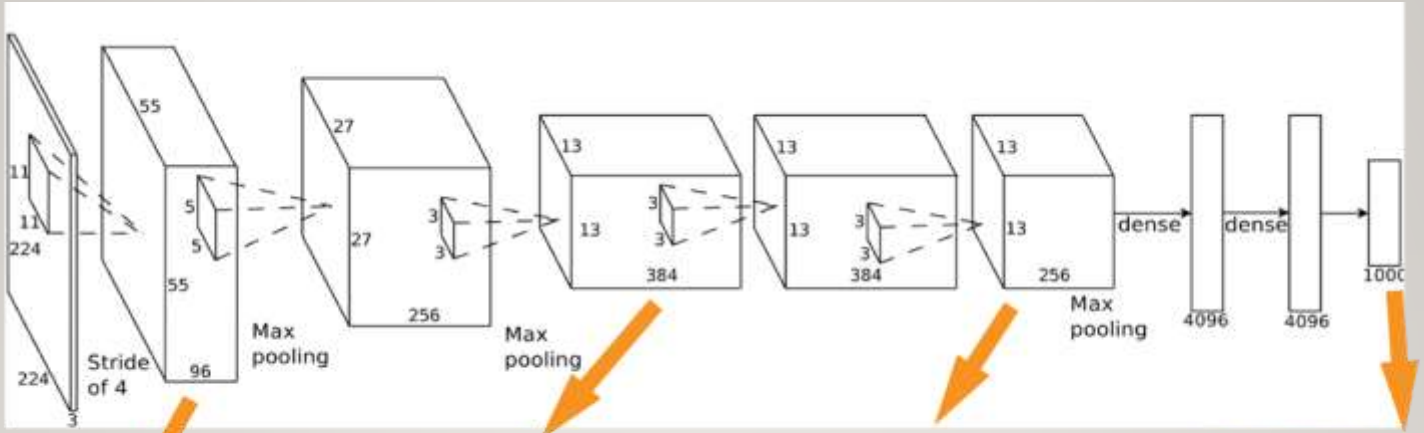
https://image-net.org/index.php

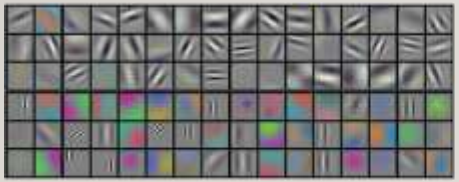
IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE (ILSVRC)



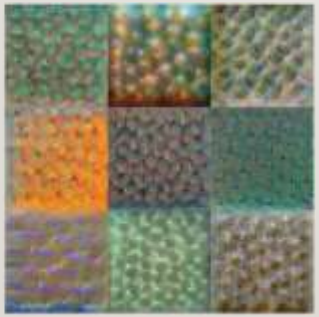
First Deep Learning winner

ALEXNET: THE ONSET OF DEEP LEARNING WINNING STREAK





Conv 1: Edge+Blob



Conv 3: Texture

Numerical

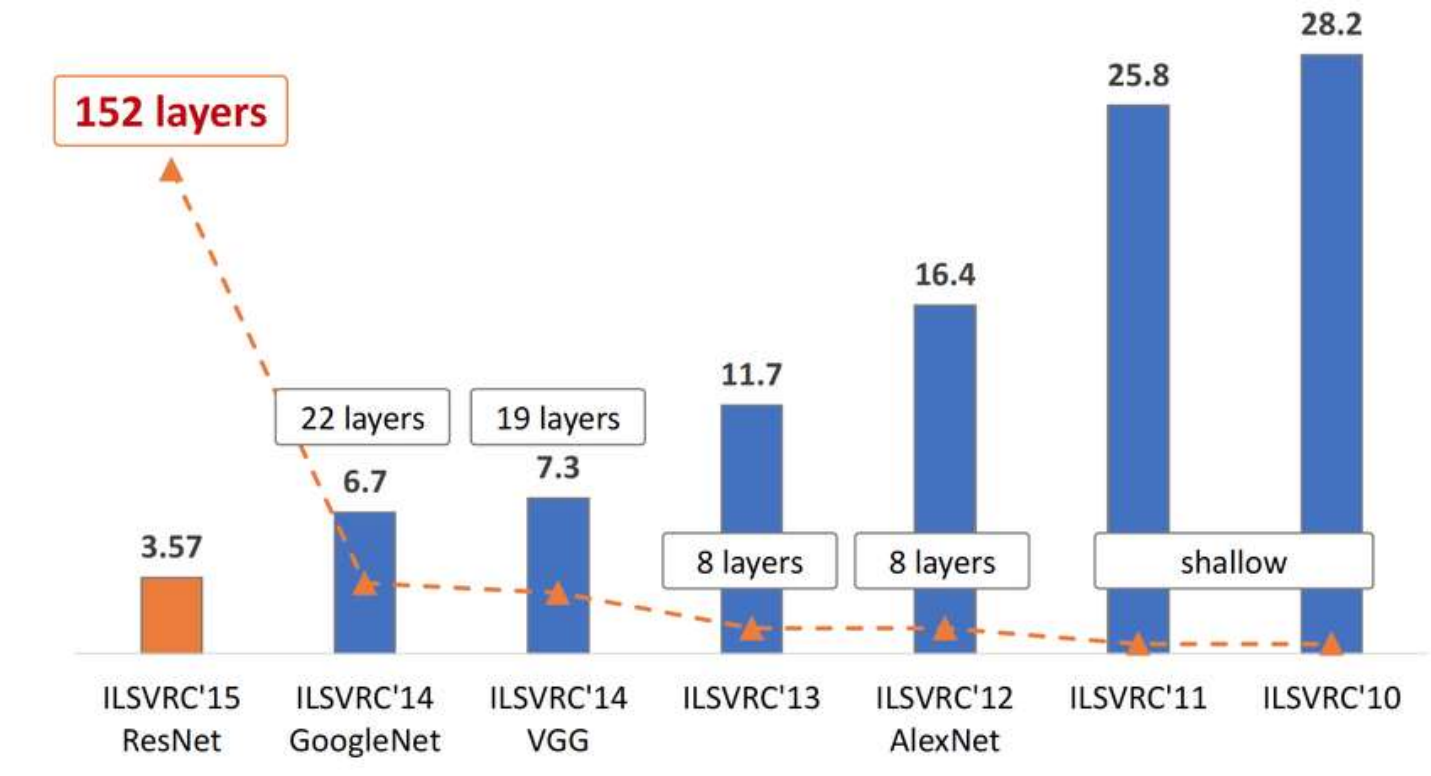
Data-driven

Conv 5: Object Parts

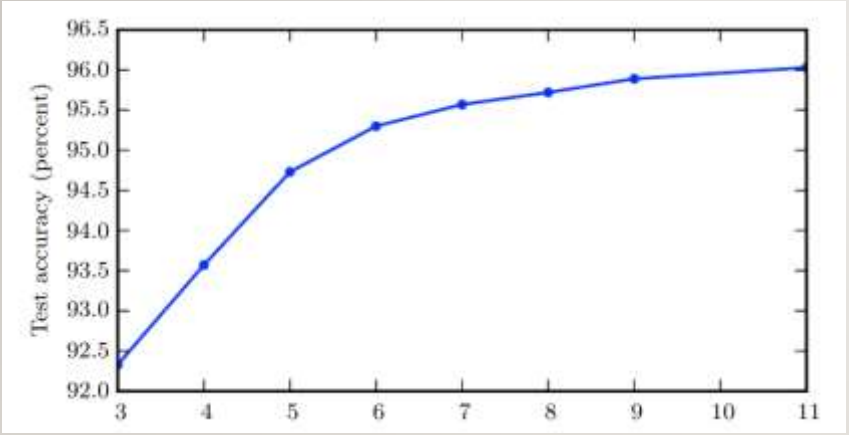
Fc8: Object Classes

mNeuron: A Matlab Plugin to Visualize Neurons from Deep Models vision03.csail.mit.edu/cnn_art/index.html

THE REVOLUTION (OR REVELATION) OF DEPTH

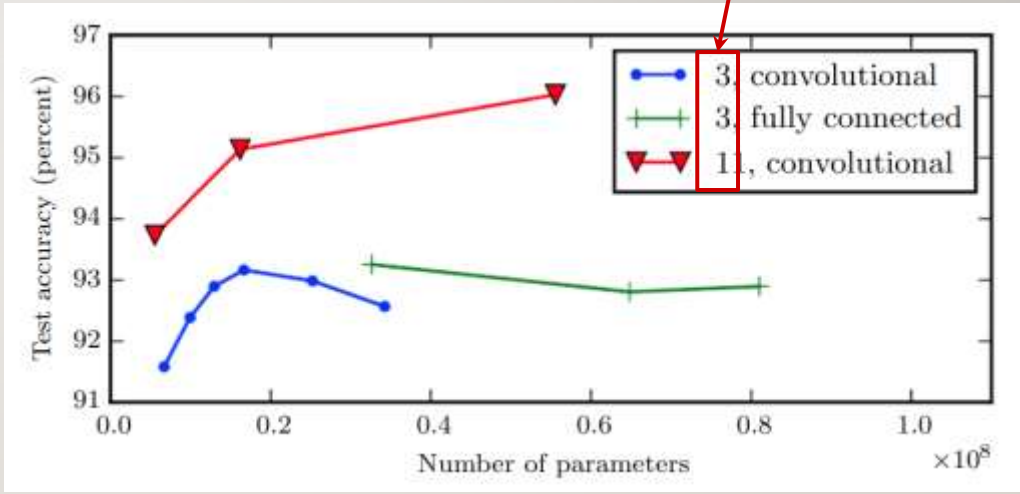


THE IMPORTANCE OF DEPTH



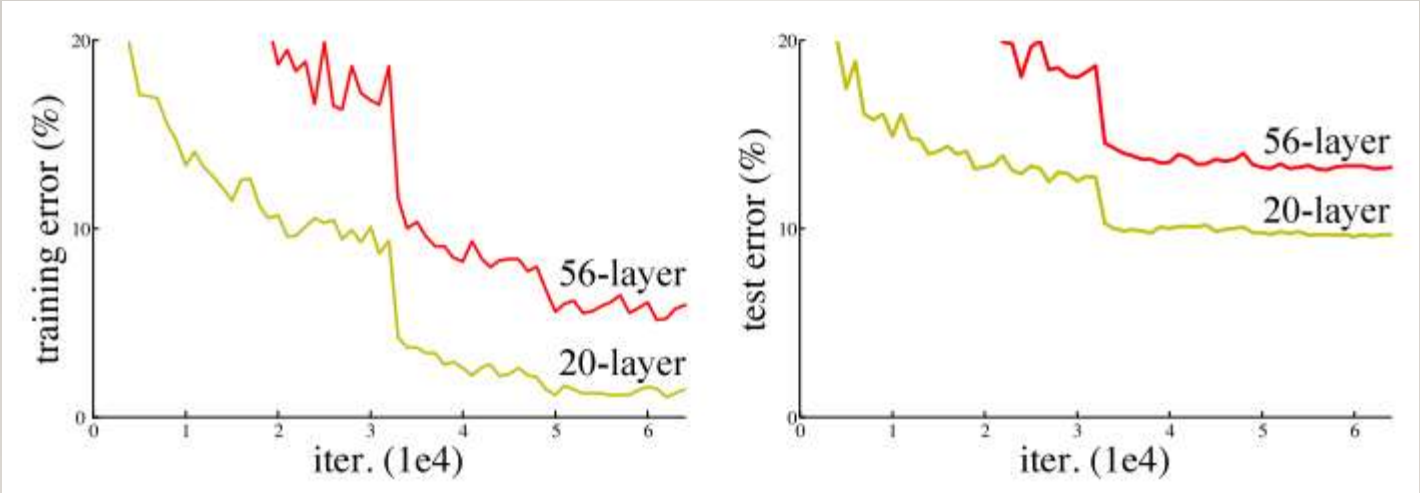
Accuracy of model increases as depth increases.

Number of layers



Deeper models outperform wider modes with the same total number of parameters.

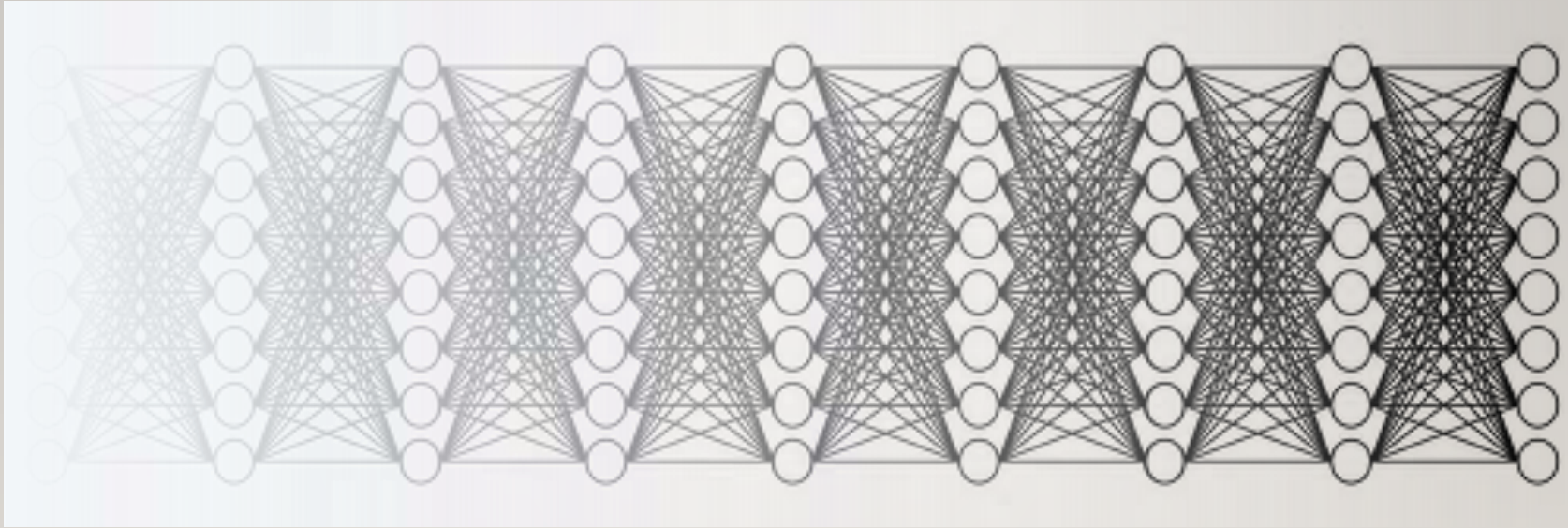
HOWEVER, TRAINING DEEP MODELS IS NOT EASY



Deeper models are more difficult to optimize.

Fig. credit: arXiv:1512.03385

VANISHING GRADIENTS PROBLEM



Gradients get worse with increasing number of layers.

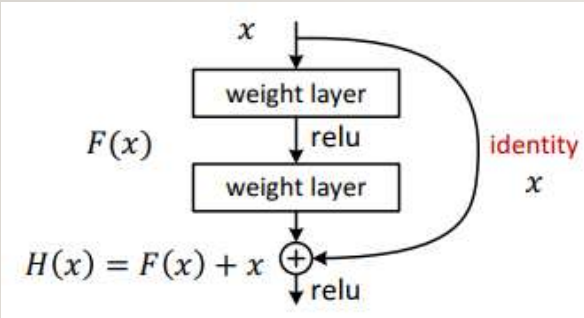
Networks can also suffer from the opposite effect, i.e. exploding gradients.

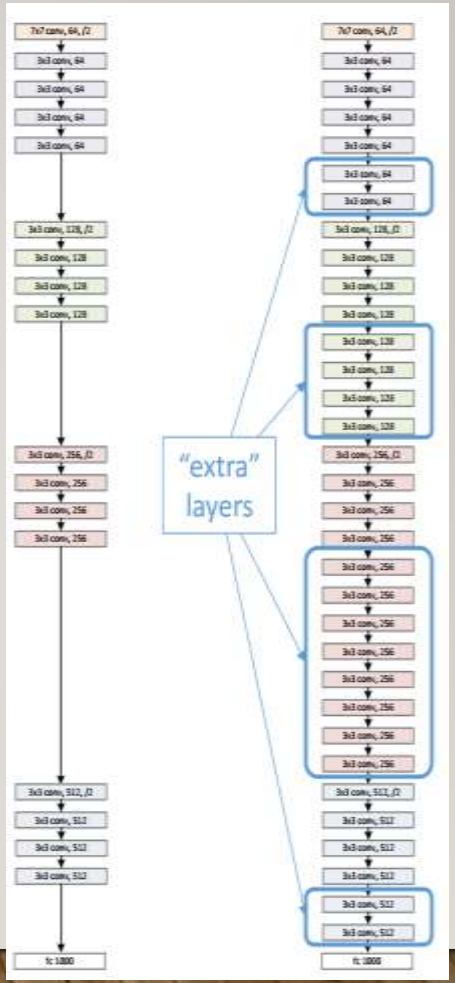
GRADIENT HIGHWAYS; SKIP CONNECTIONS

If the network to the left performs well, why doesn't the network to the right with the extra layers just learn identity mapping?!

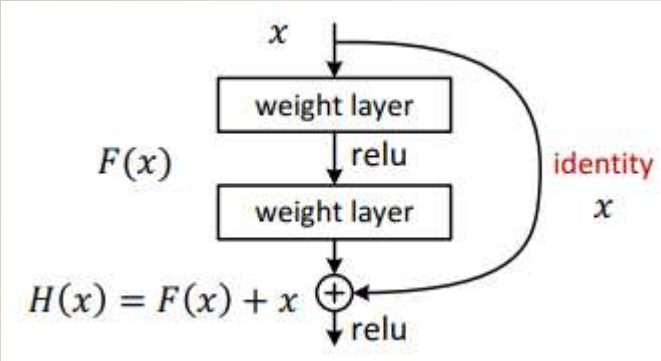
What if we have the identity mapping by construction?

Come the "Residual block":

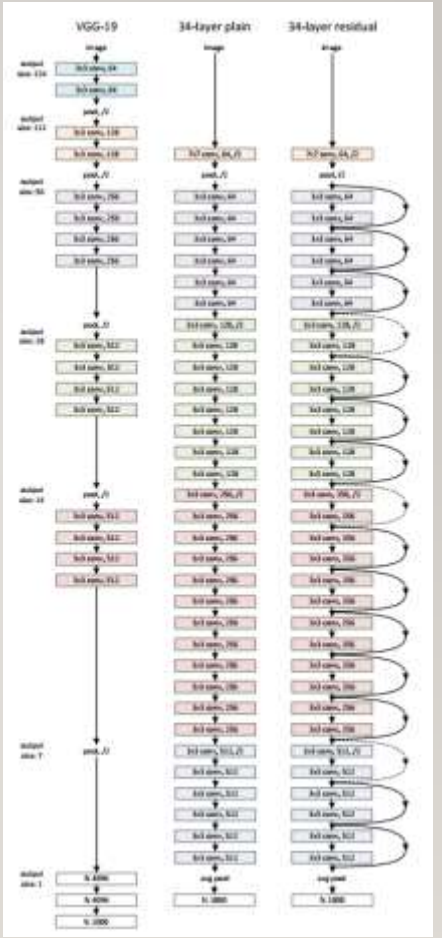




RESIDUAL NETWORKS (RESNET)



Skip connections allow training very deep networks (100s-1000s of layers), side-stepping the vanishing gradient problem.




AI