Neural Networks

for Regression, Classification,
Dimensionality Reduction, Time Series, etc.

Assoc. Prof. Karl Ezra Pilario, Ph.D.

Process Systems Engineering Laboratory
Department of Chemical Engineering
University of the Philippines Diliman

Outline

- Artificial Neural Networks
  - Architecture
  - Activation Functions
  - Forward Propagation
  - Backpropagation
  - Regularization

- ANNs for Other Tasks
  - Introduction to Deep Learning
  - Convolutional Neural Nets (Images)
  - Autoencoders (Dimensionality Reduction)
  - RNNs, GRUs, LSTMs (Time Series)

Artificial Neural Networks

- The model consists of neurons that pass information from one layer to the next.

- Historically, ANNs were widely used since the 80s, but diminished towards the late 90s. In the 2010s, interest in ANNs have revived!

- **Why neural networks?**

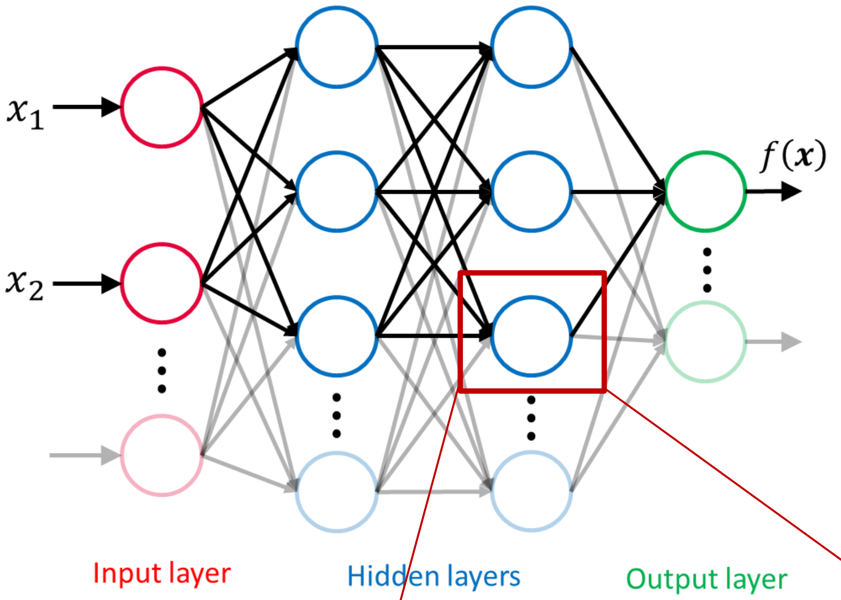
  - Neural networks were inspired from the structure of the human brain.

  - It was theoretically proven that neural networks have a universal approximation property:

> The output $f(x)$ can theoretically approximate any function to an arbitrary degree of accuracy. (*Hornik et al., 1989*)

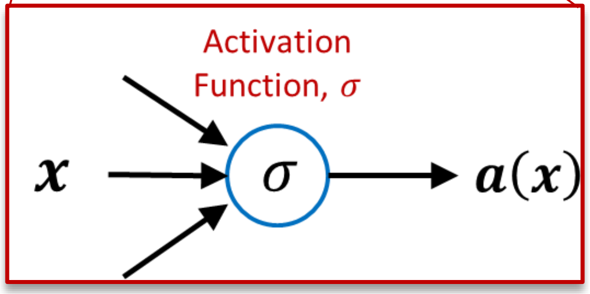
Multi-layer Perceptron (MLP)

Other names: Feedforward Neural Network (FFNN), Backpropagation Neural Network (BPNN), Fully-connected NN (FCNN)

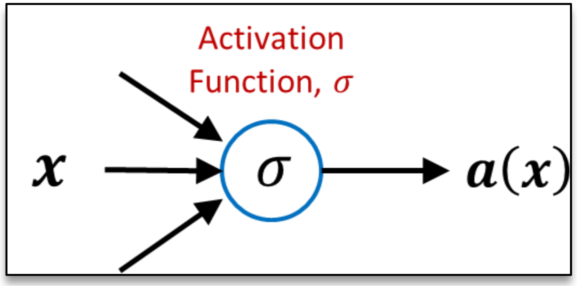


Each neuron in the *hidden* and *output* layers is a function,

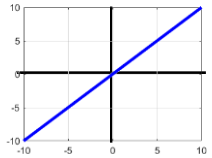
$$\boldsymbol{a}(\boldsymbol{x}) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$



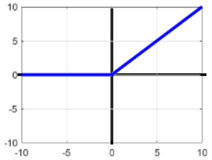
Artificial Neural Networks



Typical choices of activation function:



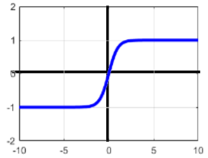
Linear $\sigma(z) = z$



ReLU (Rectified Linear Unit)

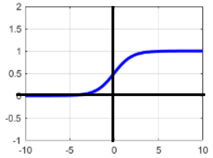
$$\sigma(z) = \begin{cases} z, & z > 0 \\ 0, & z \le 0 \end{cases} \text{ or}$$

$$\sigma(z) = \max(0, z)$$



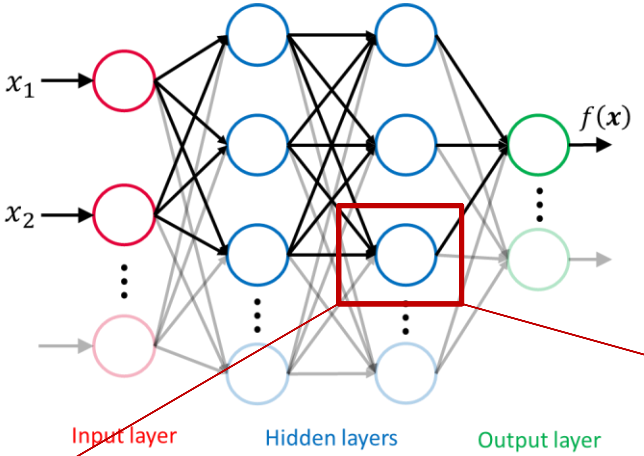
Tanh

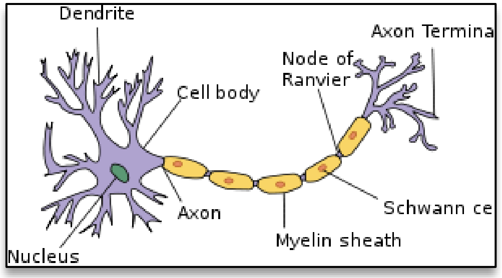
$$\sigma(z) = \frac{2}{1 + \exp(-2z)} - 1$$



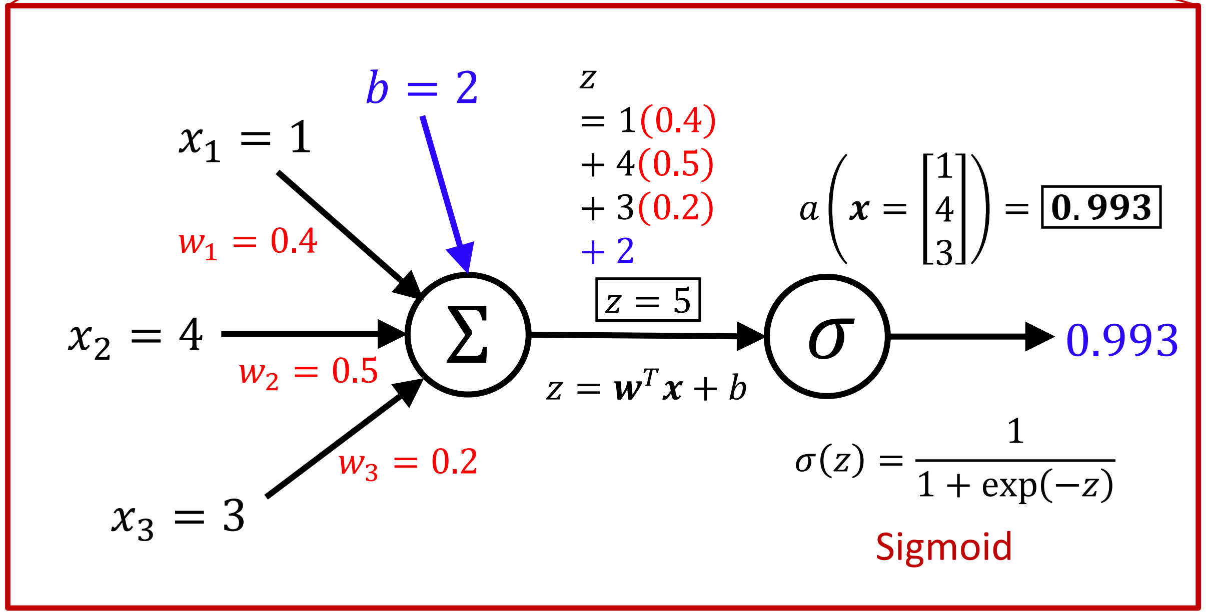
Sigmoid (Logistic)

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$





(McCulloch-Pitts, 1943)

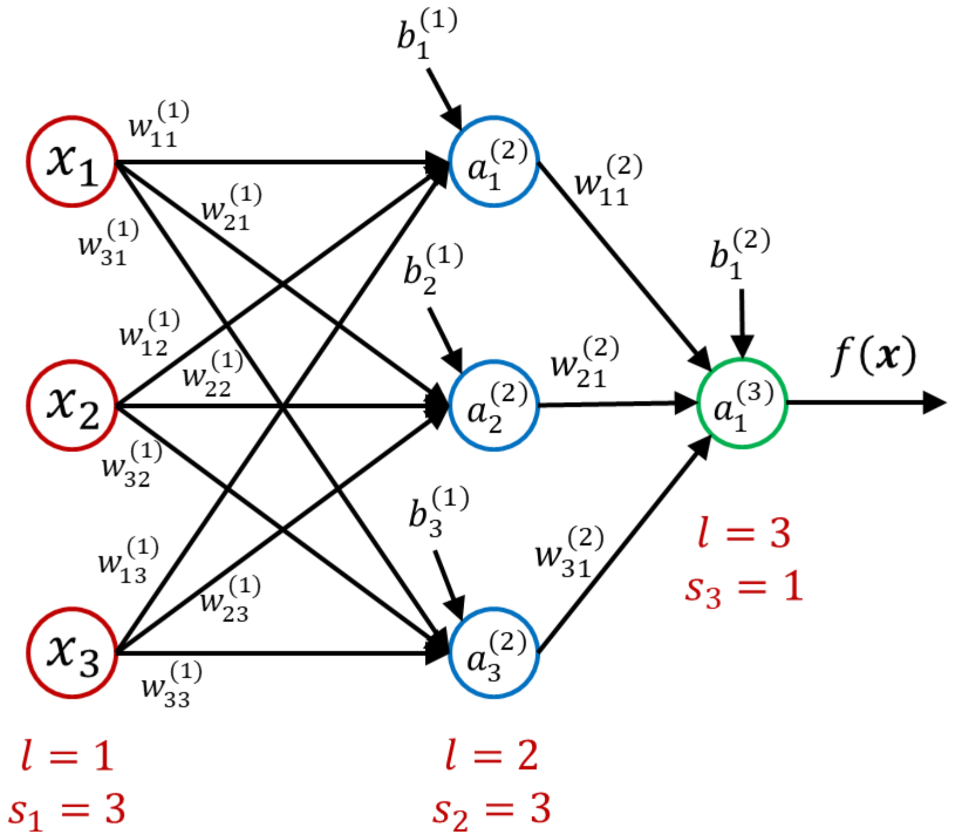


Source: https://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic

Artificial Neural Networks

How do neural networks make predictions?

Consider a simple MLP:



Let: $w_{ij}^{(l)}$ = Weights at layer l, connecting unit j from layer l to unit i at the next layer $(l+1)$.

$b_i^{(l)}$ = Bias i applied to elements at layer l.

$\sigma(\cdot)$ = Activation function

$a_i^{(l)}$ = Activation output of unit i at layer l.

$z_i^{(l)}$ = Weighted sum of inputs from layer l, entering unit i.

x_i = Input i from layer $l = 1$ only.

s_l = Number of units at any layer l.

Forward propagation: (Values of $w_{ij}^{(l)}$ and $b_i^{(l)}$ are fixed)

$$a_1^{(2)} = \sigma\left(z_1^{(1)}\right) = \sigma\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

$$a_2^{(2)} = \sigma\left(z_2^{(1)}\right) = \sigma\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

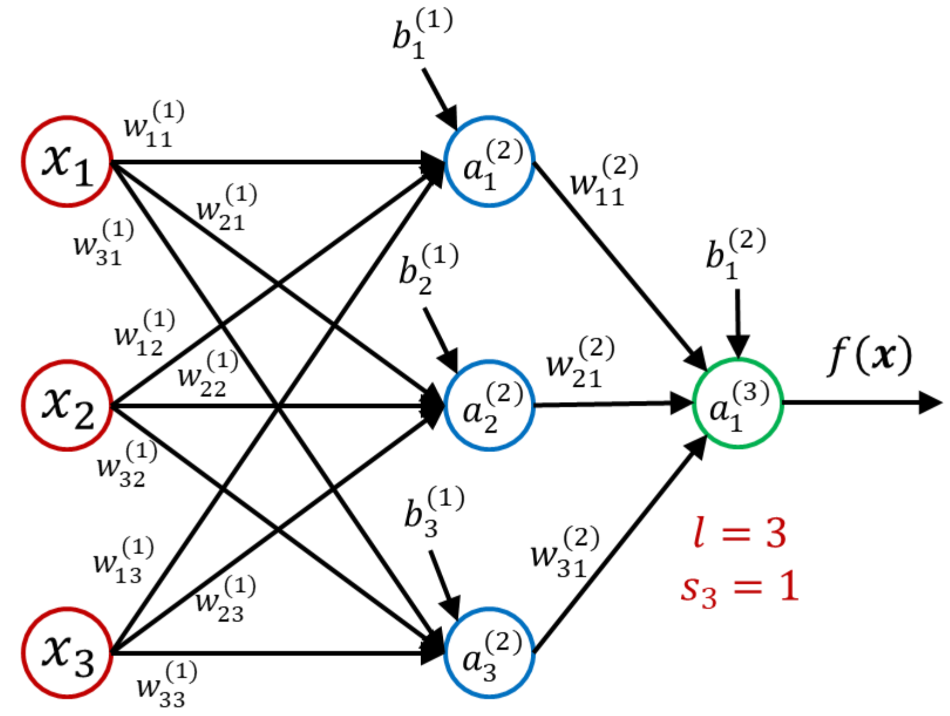
$$a_3^{(2)} = \sigma\left(z_3^{(1)}\right) = \sigma\left(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}\right)$$

$$f(\boldsymbol{x}) = a_1^{(3)} = \sigma\left(z_1^{(2)}\right) = \sigma\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)$$

Artificial Neural Networks

How do neural networks make predictions?

Consider a simple MLP:



Forward propagation: (matrix notation)

$$\begin{cases} \boldsymbol{a}^{(2)} = \sigma\left(\boldsymbol{z}^{(1)}\right) = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{x} + \boldsymbol{b}^{(l)}\right) & \in \mathbb{R}^{s_2 \times 1} \\ \boldsymbol{a}^{(l+1)} = \sigma\left(\boldsymbol{z}^{(l)}\right) = \sigma\left(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}\right) & \in \mathbb{R}^{s_{l+1} \times 1} \end{cases}$$

$\boldsymbol{W}^{(l)} =$ matrix of weights that control the mapping from layer l to layer $(l+1)$

$$\boldsymbol{a}^{(l)} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ a_3^{(l)} \\ \vdots \end{bmatrix} \qquad \boldsymbol{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & w_{12}^{(l)} & w_{13}^{(l)} & \cdots \\ w_{21}^{(l)} & w_{22}^{(l)} & w_{23}^{(l)} & \cdots \\ w_{31}^{(l)} & w_{32}^{(l)} & w_{33}^{(l)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \qquad \boldsymbol{b}^{(l)} = \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ b_3^{(l)} \\ \vdots \end{bmatrix}$$

$\boldsymbol{a}^{(l)} \in \mathbb{R}^{s_l \times 1}$ (column vector)

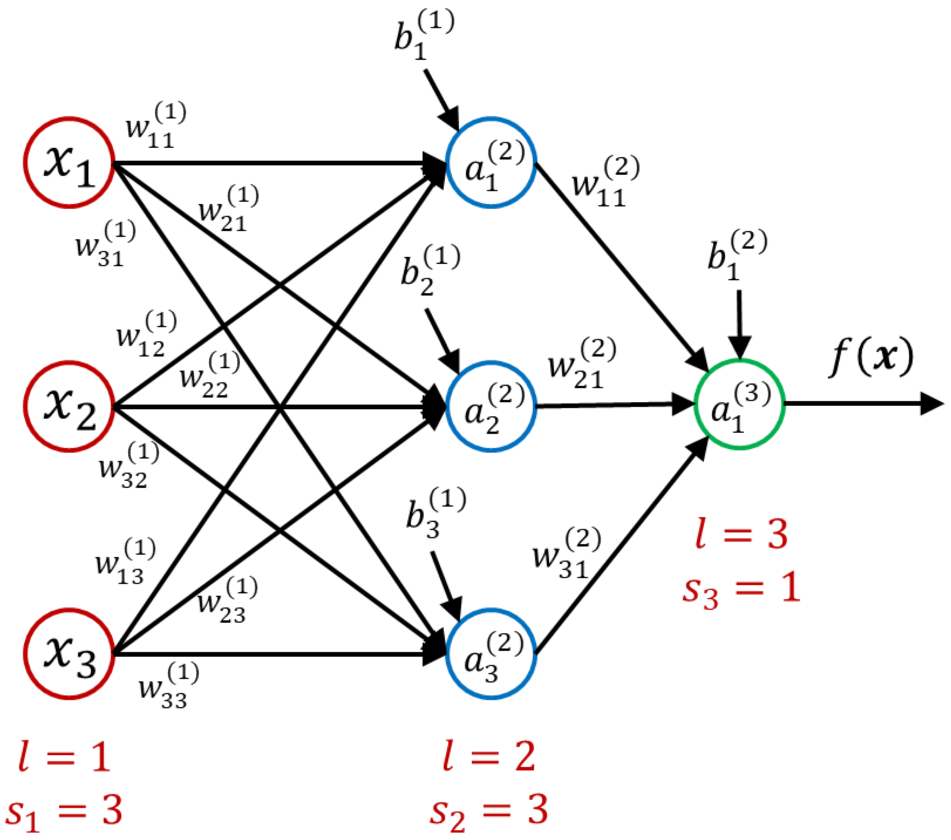
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{s_{l+1} \times s_l}$ (matrix)

$\boldsymbol{b}^{(l)} \in \mathbb{R}^{s_l \times 1}$ (column vector)

Artificial Neural Networks

How to train a neural network?

Find the values of $\boldsymbol{W}$ and $\boldsymbol{b}$ in all layers that minimize a loss function upon exposing it to a given data set for training.



Cost / Loss function for **Regression**

Prediction $f(\boldsymbol{x})$: Continuous

Mean Squared Error

The cost function for ANN regression models is the same as that in linear regression:

$$\min_{\boldsymbol{W},\boldsymbol{b}} \quad C(\boldsymbol{W},\boldsymbol{b}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(\boldsymbol{x}_i)\right)^2$$

Cost / Loss function for **Binary Classification**

Prediction $f(\boldsymbol{x})$: Binary: 0 / 1

Binary Cross-entropy Loss

The cost function for ANN classifiers is the same as that in logistic regression:

$$\min_{\boldsymbol{W},\boldsymbol{b}} \quad C(\boldsymbol{W},\boldsymbol{b}) = \frac{1}{N}\sum_{i=1}^{N} -y_{\mathrm{i}}\ \log(f(\boldsymbol{x}_i)) - (1 - y_{\mathrm{i}})\log(1 - f(\boldsymbol{x}_i))$$

Cost / Loss function for **Multi-class Classification**

Prediction $f(\boldsymbol{x})$: Classes: 0, 1, …, K

Categorical Cross-entropy Loss

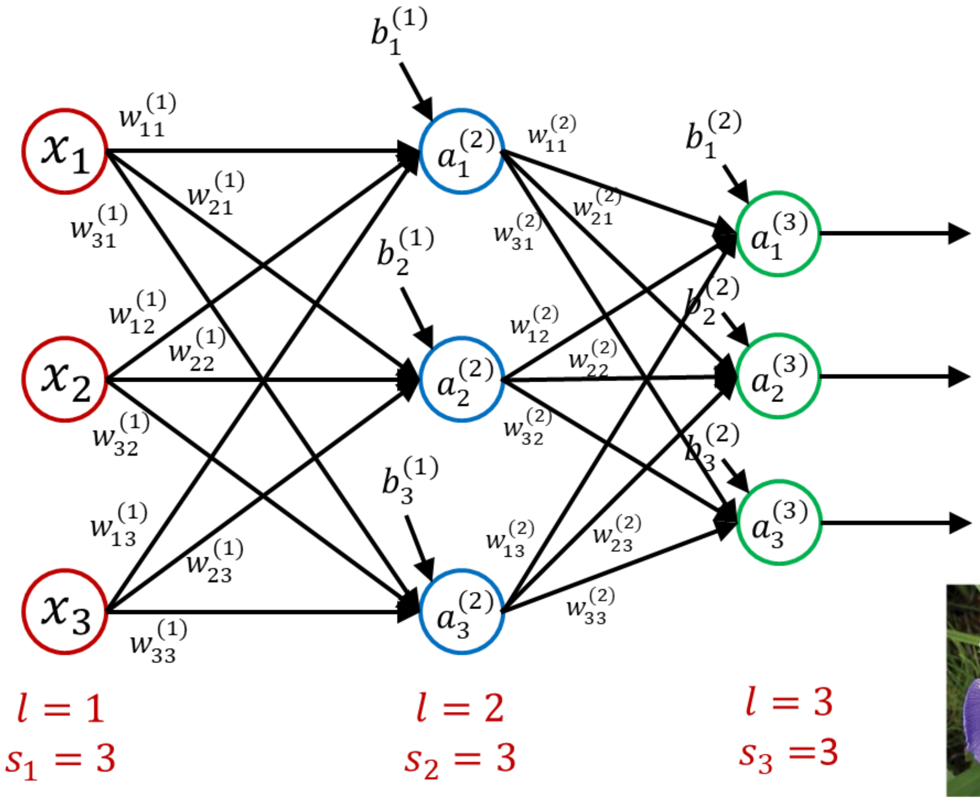
The cost function for ANN multi-class classifiers is also related to cross-entropy:

$$\min_{\boldsymbol{W},\boldsymbol{b}} \quad C(\boldsymbol{W},\boldsymbol{b}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} -y_{i,j}\log\left(f_j(\boldsymbol{x}_i)\right)$$

Artificial Neural Networks

Neural Network for **Multi-class Classification**

The architecture of the ANN for multi-class classification is different from the one for regression or binary classification:



Softmax activation:

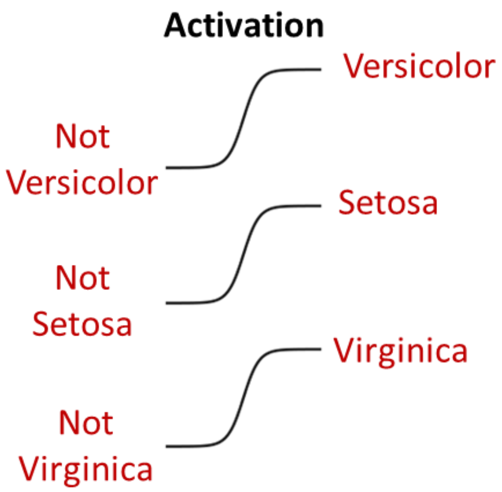
K = no. of classes

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \qquad i = 1, 2, \dots, K$$

One-hot Encoding:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Versicolor Setosa Virginica



The number of output neurons is set to the number of classes in the problem.

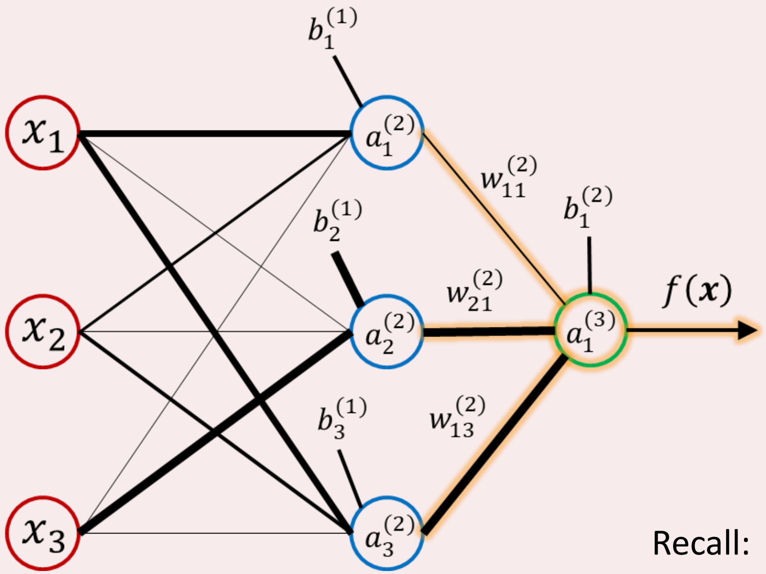
Artificial Neural Networks

Loss Function: $$\min_{\boldsymbol{W},\boldsymbol{b}} \; C(\boldsymbol{W},\boldsymbol{b}) = \sum_{i=1}^{N} \left(y_i - f(\boldsymbol{x}_i)\right)^2$$

- In order to minimize $C(\boldsymbol{W},\boldsymbol{b})$, we need to compute gradients $\frac{\partial C}{\partial \boldsymbol{W}}$ and $\frac{\partial C}{\partial \boldsymbol{b}}$ using the *chain rule of differentiation*.
- Due to the chain rule, we now need to pass information *backwards*, from the output layer to the input layer.

Backpropagation

- What we are actually propagating backwards is the prediction error. The amount of error propagated on a layer l is used to decide how much the parameters ($\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}$) should change to decrease the loss function most effectively.
- Backprop is *reminiscent of* **Hebbian Learning** in living organisms: "Neurons that fire together, wire together." (Donald Hebb, 1949)
- The discovery of backpropagation is attributed to **Paul Werbos** in his 1974 PhD Dissertation.



Suppose that the magnitude of values are depicted as the **line widths**.

All elements of Layer 3 are highlighted in yellow.

MSE cost function:
$$C = \left(y_i - f(\boldsymbol{x}_i)\right)^2 \text{ or } = \left(y_i - a_1^{(3)}\right)^2$$

Recall:
$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$
$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{a}^{(2)} + \boldsymbol{b}^{(2)}$$
$$\sigma(\boldsymbol{z}) = 1/(1+e^{-\boldsymbol{z}})$$

Let's compute the derivatives:

Layer 3

$$\frac{\partial C}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{W}^{(2)}} \times \frac{\partial a_1^{(3)}}{\partial \boldsymbol{z}^{(2)}} \times \frac{\partial C}{\partial a_1^{(3)}} = \boldsymbol{a}^{(2)} \times \sigma'(\boldsymbol{z}^{(2)}) \times \underbrace{-2\left(y_i - a_1^{(3)}\right)}_{\text{Error}}$$

$$\frac{\partial C}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{b}^{(2)}} \times \frac{\partial a_1^{(3)}}{\partial \boldsymbol{z}^{(2)}} \times \frac{\partial C}{\partial a_1^{(3)}} = 1 \times \sigma'(\boldsymbol{z}^{(2)}) \times \underbrace{-2\left(y_i - a_1^{(3)}\right)}_{\text{Error}}$$

$$\boxed{\frac{\partial C}{\partial \boldsymbol{a}^{(2)}}} = \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{a}^{(2)}} \times \frac{\partial a_1^{(3)}}{\partial \boldsymbol{z}^{(2)}} \times \frac{\partial C}{\partial a_1^{(3)}} = \boldsymbol{W}^{(2)} \times \sigma'(\boldsymbol{z}^{(2)}) \times \underbrace{-2\left(y_i - a_1^{(3)}\right)}_{\text{Error}}$$

Layer 2

$$\frac{\partial C}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial \boldsymbol{z}^{(1)}}{\partial \boldsymbol{W}^{(1)}} \times \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(1)}} \times \boxed{\frac{\partial C}{\partial \boldsymbol{a}^{(2)}}} = \boldsymbol{x} \times \sigma'(\boldsymbol{z}^{(1)}) \times \frac{\partial C}{\partial \boldsymbol{a}^{(2)}}$$

$$\frac{\partial C}{\partial \boldsymbol{b}^{(1)}} = \frac{\partial \boldsymbol{z}^{(1)}}{\partial \boldsymbol{b}^{(1)}} \times \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(1)}} \times \boxed{\frac{\partial C}{\partial \boldsymbol{a}^{(2)}}} = 1 \times \sigma'(\boldsymbol{z}^{(1)}) \times \frac{\partial C}{\partial \boldsymbol{a}^{(2)}}$$

Artificial Neural Networks

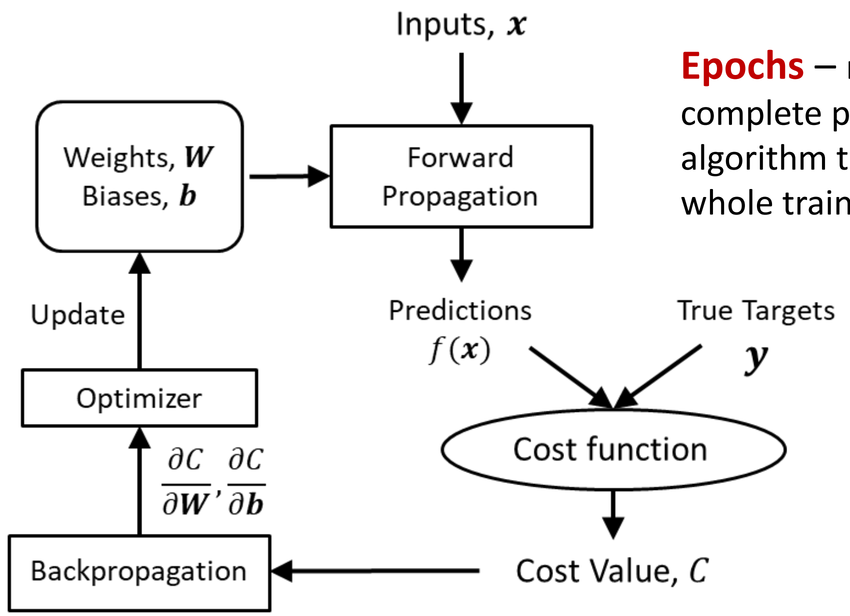
Algorithm: Gradient Descent

Given: Training Data $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

Initialization:
Setup the NN architecture (no. of layers, etc.)
Choose activation functions, $\sigma(\cdot)$
Choose an optimizer and its parameters (e.g. learning rate)
Choose a loss function (e.g. MSE, Cross-entropy)
Set a *random* initial $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \dots$

Do Forward Propagation then compute the initial cost, C.
While the Cost C is not minimum,
    For each training sample k,
        Do Forward Propagation for sample (x_k, y_k)
        Do Backpropagation for sample (x_k, y_k)
    End For
    Average the gradients $\frac{\partial C}{\partial \boldsymbol{W}}$ and $\frac{\partial C}{\partial \boldsymbol{b}}$ from all k samples.
    Update all $\boldsymbol{W}$ and $\boldsymbol{b}$ using the *optimizer*.
End While

1 epoch



Epochs – no. of complete passes of the algorithm through the whole training data set.

Optimizers:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \gamma \frac{\partial C}{\partial \boldsymbol{\Theta}}$$

$$\boldsymbol{\Theta}^{(l)} = \begin{bmatrix} \boldsymbol{W}^{(l)}(:) \\ \boldsymbol{b}^{(l)} \end{bmatrix}$$

1. **AdaGrad** (Adaptive Gradient)

   The learning rate γ decays in each epoch.

2. **RMSprop** (Root Mean Square Propagation)

   The decay for γ is less aggressive near convergence.

3. **Adam** (Adaptive Moment Estimation) (*Kingma et al. 2014*)

   Decay the γ and also use the cumulative history of the gradients to better guide the parameter updates.

Artificial Neural Networks

Gradient Descent Variants

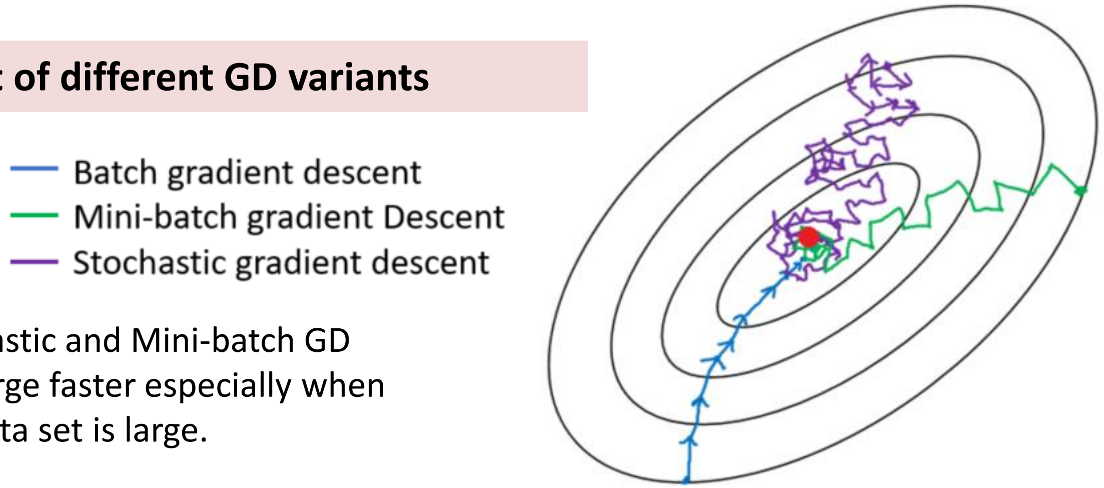
These variants differ in how much training data is used to compute the gradient of the cost function.

1. **Batch Gradient Descent**
   - Compute the gradient $\frac{\partial C}{\partial \boldsymbol{\Theta}}$ using the entire training data in every epoch.

2. **Stochastic Gradient Descent**
   - Compute the gradient $\frac{\partial C}{\partial \boldsymbol{\Theta}}$ and update the parameters $\boldsymbol{\Theta}$ right away after every sample.
   - In each pass, the sample is chosen randomly.
   - It's faster than Batch GD, but the loss function fluctuates more wildly.

3. **Mini-Batch Gradient Descent**
   - Compute the gradient $\frac{\partial C}{\partial \boldsymbol{\Theta}}$ and update the parameters $\boldsymbol{\Theta}$ for every mini-batch of samples.
   - Less fluctuations than SGD, but still fast.

Effect of different GD variants

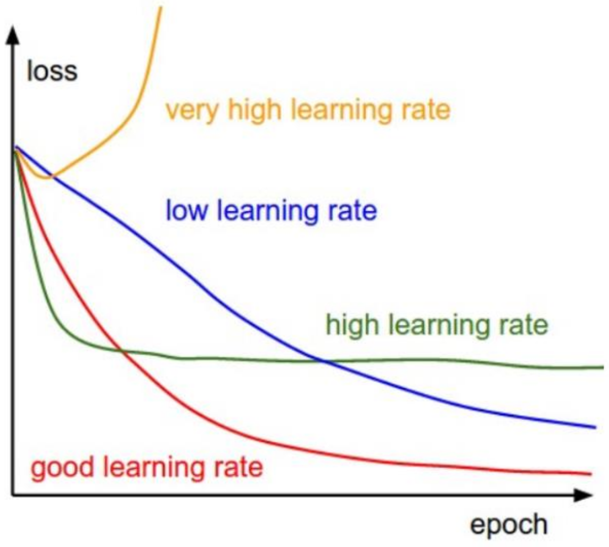


Stochastic and Mini-batch GD converge faster especially when the data set is large.

Effect of different learning rates, γ

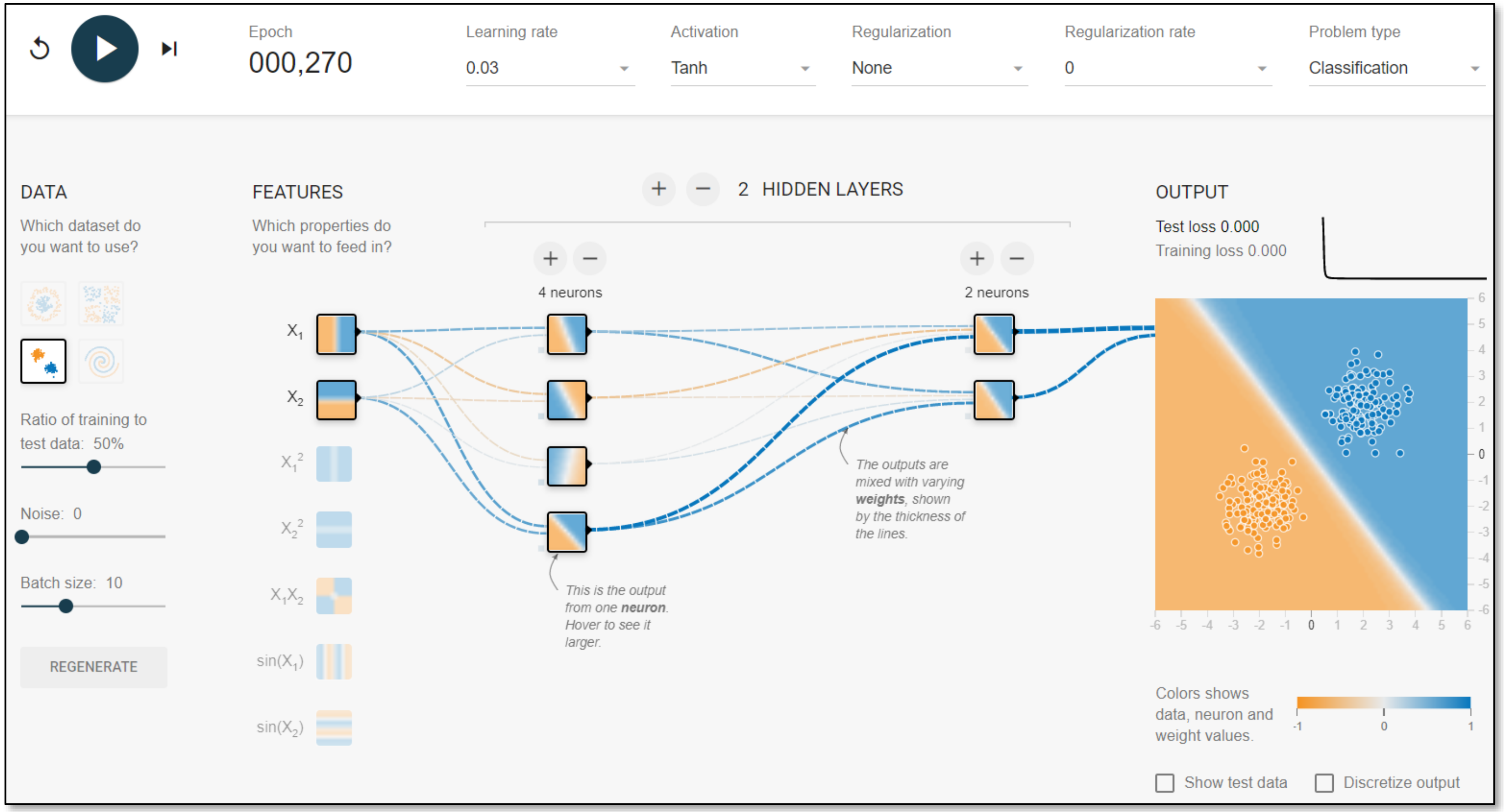
The γ dictates the **step size** applied to update the parameters $\boldsymbol{\Theta}$.

- A high γ is likely to miss the minimum of the loss function.
- A low γ takes too much time to converge.



Artificial Neural Networks

https://playground.tensorflow.org/



Artificial Neural Networks

	Model Parameters	Hyper-Parameters
Linear Regression	Weights, w	Regularization parameter, λ Type of regularization
Logistic Regression	Weights, w	Regularization parameter, λ Type of regularization Solver
Locally Weighted Regression	Weights, w	Weighting function, ω Bandwidth, τ
Support Vector Classifier	Dual variables, α Bias, b	Kernel type Kernel scale Box constraint Multi-class strategy
Naïve Bayes	None	Distribution: Gaussian or KDE Class Priors Laplace smoothing?
k-Nearest Neighbors	None	No. of neighbors, k Distance metric
Decision Trees	Splits	Max Depth Splitting Criterion min_samples_leaf min_samples_split max_features

Neural Network

- **Model Parameters** — Weights, w; Bias, b
- **Hyper-parameters** — Regularization parameter, λ; Type of regularization; Architecture; Activation Functions; Optimizer; Learning Rate

Recall: Hyper-parameter Tuning can be performed using

- Manual Search,
- Grid Search,
- Random Search, or
- Bayesian Optimization (Optuna)

Artificial Neural Networks

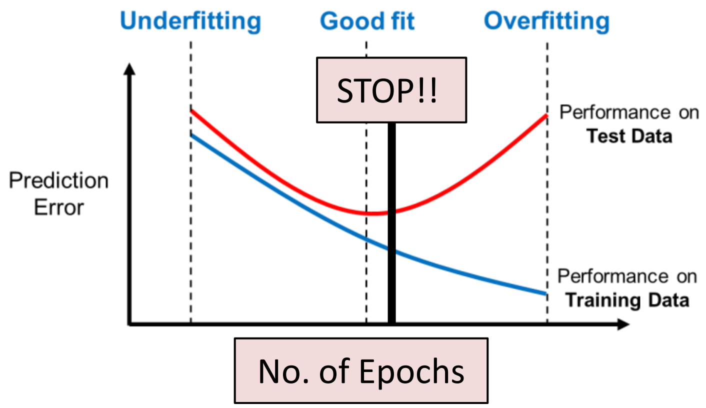
How to prevent overfitting in neural networks?

- Regularization penalizes the values of model parameters in the cost function.
  - L1 (Lasso) and L2 (Ridge) regularization
  - Elastic Net regularization

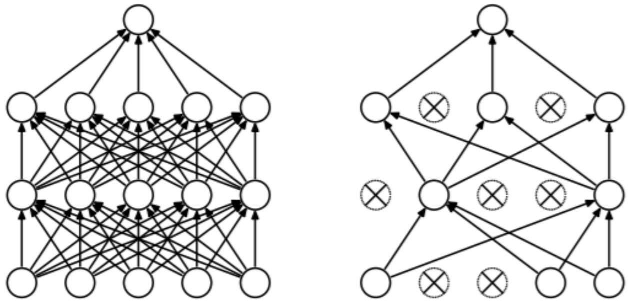
$$
L^2 \begin{cases} \min\limits_{\boldsymbol{W},\boldsymbol{b}} \; C(\boldsymbol{W},\boldsymbol{b}) = \sum\limits_{i=1}^{N} -y_i \log(f(\boldsymbol{x}_i)) - (1-y_i)\log\big(1-f(\boldsymbol{x}_i)\big) + \frac{\lambda}{2}\sum\limits_{l=1}^{L} {\boldsymbol{\Theta}^{(l)}}^2 \\ \min\limits_{\boldsymbol{W},\boldsymbol{b}} \; C(\boldsymbol{W},\boldsymbol{b}) = \sum\limits_{i=1}^{N} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \frac{\lambda}{2}\sum\limits_{l=1}^{L} {\boldsymbol{\Theta}^{(l)}}^2 \end{cases}
$$

λ = regularization rate/parameter

- Early Stopping – interrupt the training process when the validation error no longer improves.



- Dropout - randomly zero-out certain units of a layer in order to break spurious correlations in the training data that the layer is exposed to.



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

- Reduce the network size – If a model overfits, it may have too many learnable parameters (too much *capacity*). You must find the right capacity for the complexity of the data.

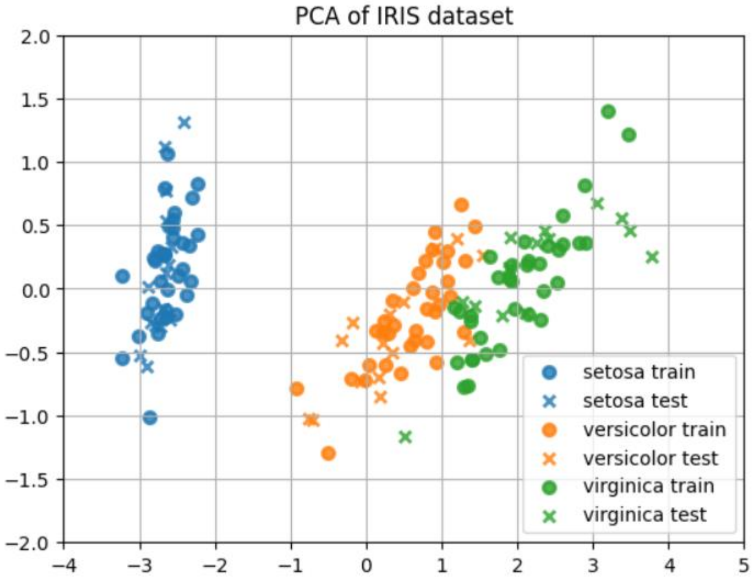
In general, overfitting can be prevented by collecting more training data!

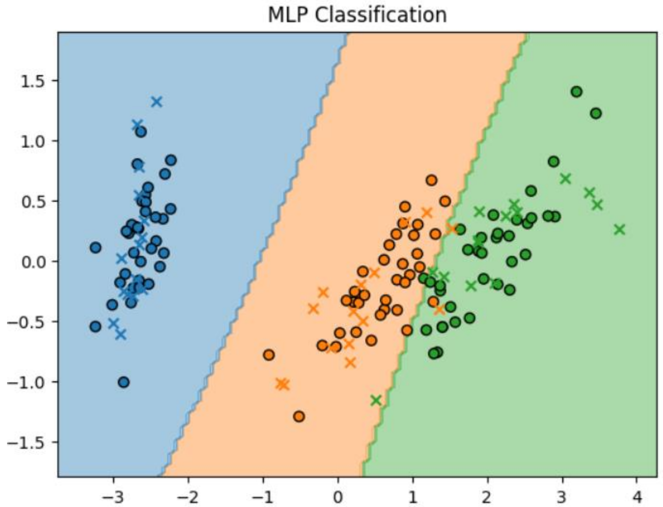
Artificial Neural Networks

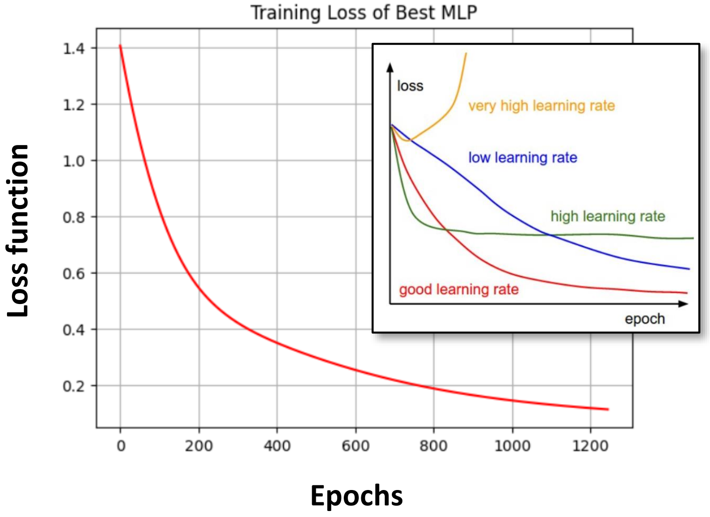
Example 1

Fit an **MLP Classifier** that predicts the species of Iris flower given the 2-D PCA features of their measurement data. Tune the architecture within the following ranges via *Random Search*:

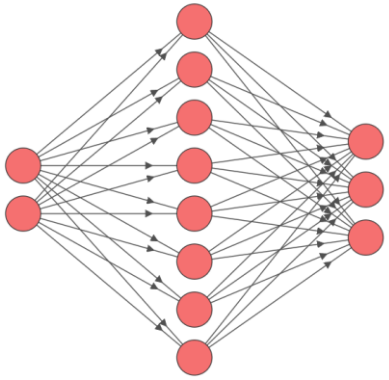
- Regularization (logspace) : $\alpha = [10^{-4}, 10^{1}]$
- Solver: 'adam', 'sgd'
- No. of hidden neurons: $[4, 15]$
- Activation in hidden layer: 'identity', 'logistic', 'tanh', 'relu'





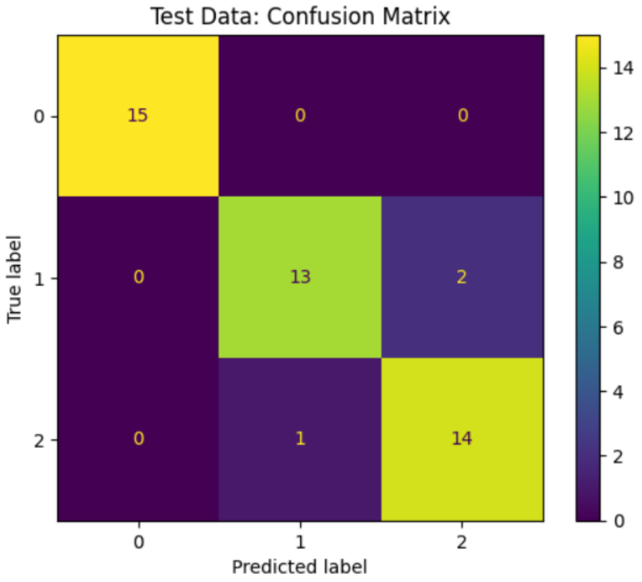


Best MLP:



Regularization: α = 0.0246 ($10^{-1.6}$)
Solver: 'adam'
No. of hidden neurons: 8
Activation: 'tanh'
Initial Learning Rate: 0.001 (default)

https://alexlenail.me/NN-SVG/index.html

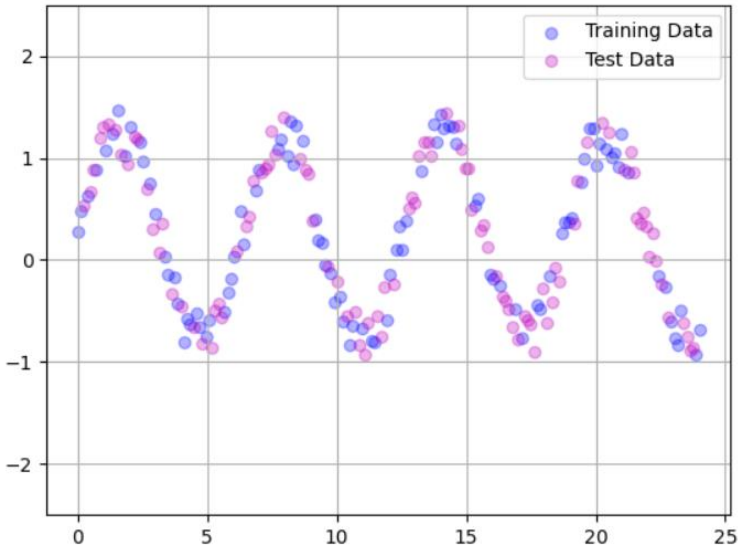


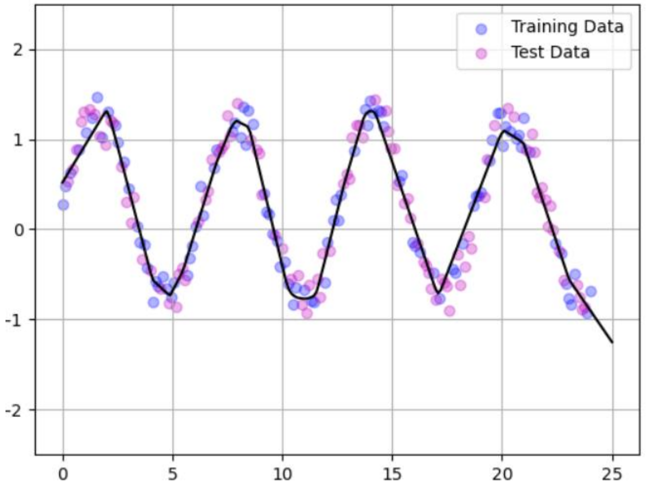
Artificial Neural Networks

Example 2

Fit an **MLP Regressor** that learns the sine function based on a few training samples with noise. Tune the architecture within the following ranges via *Random Search*:

- Regularization (logspace) : $\alpha = [10^{-4}, 10^{1}]$
- Solver: 'adam', 'sgd'
- No. of hidden neurons: $[10, 50]$ (decreasing each layer)
- No. of hidden layers: 3
- Activation in hidden layer: 'identity', 'logistic', 'tanh', 'relu'





Best MLP:

Regularization:	α = 0.0482
Solver:	'adam'
No. of hidden neurons:	[45, 45, 11]
Activation:	'relu'
Initial Learning Rate:	0.001 (default)

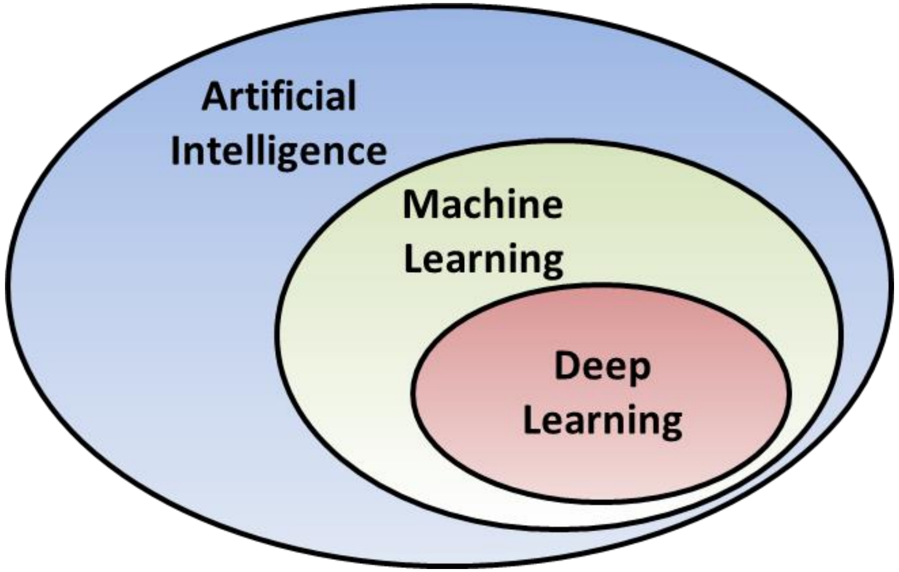
This result demonstrates how *data-hungry* ANNs are. Even though they are universal approximators, ANNs need many training samples to approximate a function to a desired accuracy.

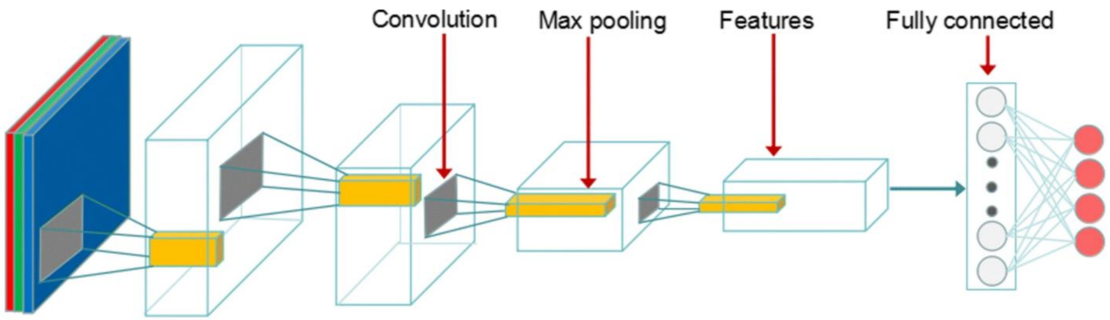
Outline

- Artificial Neural Networks
  - Architecture
  - Activation Functions
  - Forward Propagation
  - Backpropagation
  - Regularization

- ANNs for Other Tasks
  - Introduction to Deep Learning
  - Convolutional Neural Nets (Images)
  - Autoencoders (Dimensionality Reduction)
  - RNNs, GRUs, LSTMs (Time Series)

Deep Learning





Why Deep Learning?

- **Deep Learning vs. Shallow Learning**
  - Traditional machine learning is only shallow learning, as it involves only 1-2 layers (e.g. PCA + SVM).
  - Deep learning offers a new take on learning representations from data---by adding more successive layers!
- **Automatic Feature Engineering**
  - Deep learners can be trained to do automatic feature engineering.
  - Each layer tries to learn more features.
  - There is no need for the user to hand-craft features that are domain-specific.
  - You can replace a multi-stage ML workflow with a single, end-to-end, general-purpose deep learning model.
- **Are stacking ensembles equivalent to deep learning?**
  - No. Deep learning allows a model to learn all layers of representation jointly or at the same time. This is more powerful than greedily training each layer in succession.

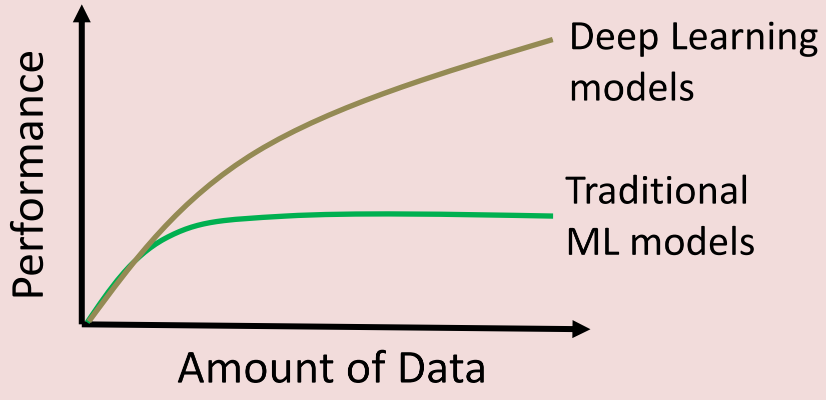
Source: Chollet. Deep Learning with Python (2018). Manning Publications Co.

Deep Learning

Deep Learning models are very complex.
Does that mean they will always overfit?

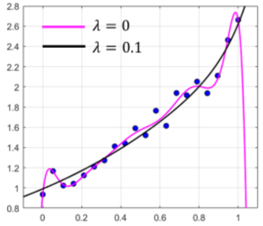
- A very complex model (having too many parameters) is prone to overfit, but deep learning (DL) models tend to violate this notion. Why??
- It turns out that DL models benefit from **over-parametrization**.

However, DL models need lots and lots of data and lots of computation time.

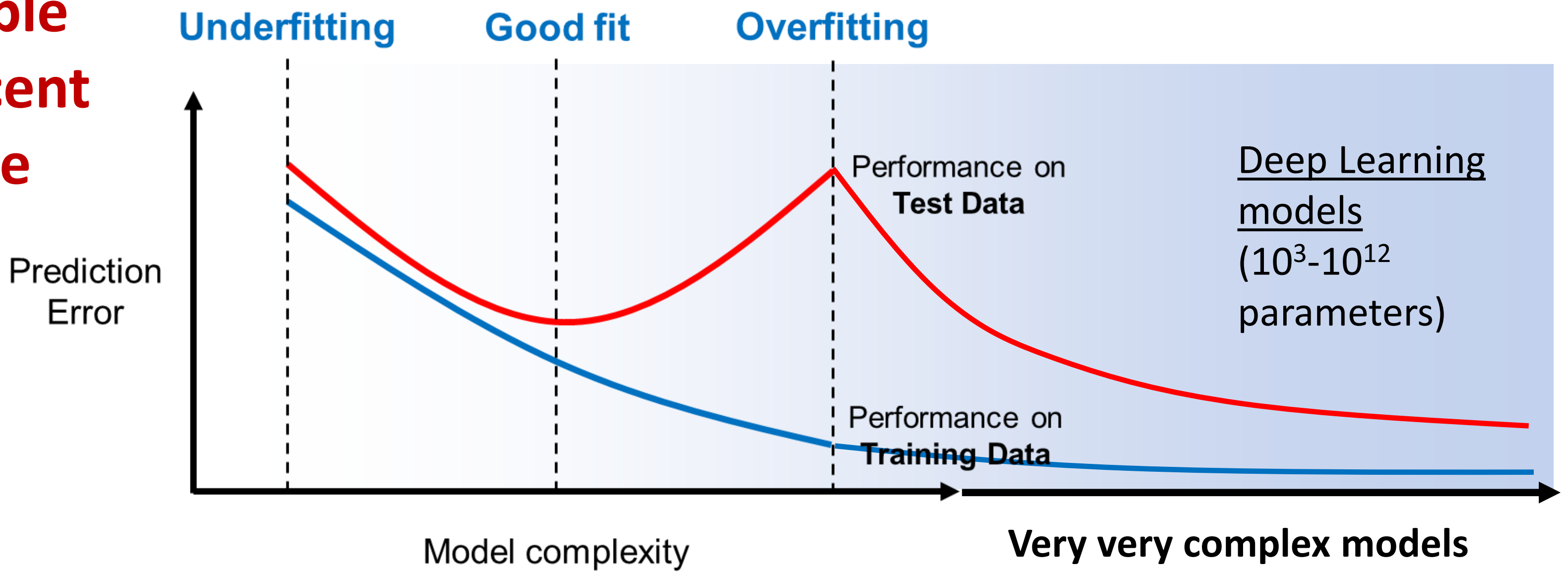


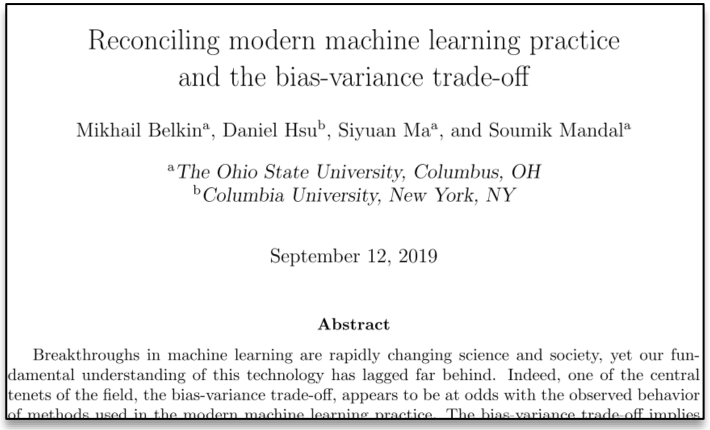
https://www.youtube.com/watch?v=xflCLdJh0n0

The model is a perfect interpolator



Double Descent Curve





Reconciling modern machine learning practice and the bias-variance trade-off

Mikhail Belkin[a], Daniel Hsu[b], Siyuan Ma[a], and Soumik Mandal[a]

[a] The Ohio State University, Columbus, OH
[b] Columbia University, New York, NY

September 12, 2019

Abstract

Breakthroughs in machine learning are rapidly changing science and society, yet our fundamental understanding of this technology has lagged far behind. Indeed, one of the central tenets of the field, the bias-variance trade-off, appears to be at odds with the observed behavior of methods used in the modern machine learning practice. The bias-variance trade-off implies

Deep Learning

How did humanity get to deep learning?

Wang, H. and Raj, B. (2017). On the Origin of Deep Learning. https://doi.org/10.48550/arXiv.1702.07800

Neuron

Perceptron

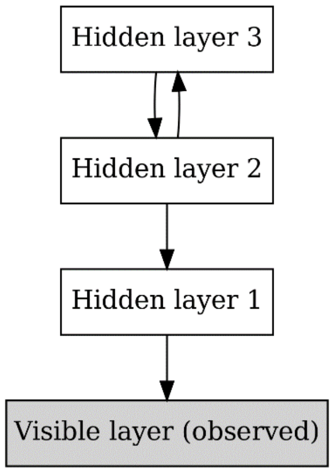
Recurrent NNs

ConvNets

LSTMs

Deep Belief Networks

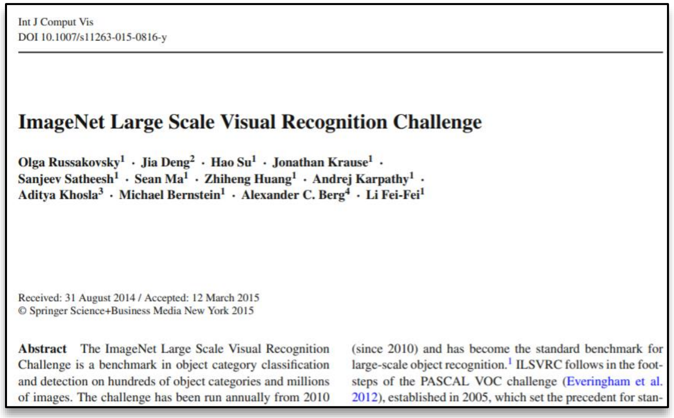
Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks



Deep Learning: Computer Vision

ImageNet Large-Scale Visual Recognition Challenge

- a.k.a. **ILSVRC**
- A benchmark in object category classification and detection on hundreds of object categories and millions of images.
- The competition runs annually since 2010.
- Maintains a publicly available data set accessible at: http://image-net.org/challenges/LSVRC/.
- It's a massive classification problem: **1000** object classes, **1.2 million** training images, **50,000** validation images, **100,000** test images. (ImageNet-1K)



Int J Comput Vis
DOI 10.1007/s11263-015-0816-y

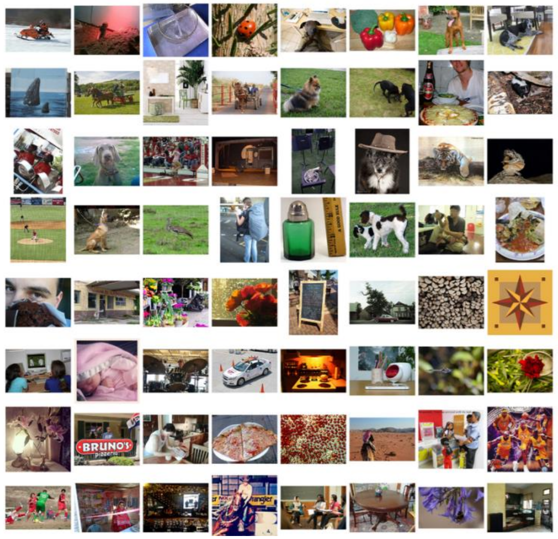
ImageNet Large Scale Visual Recognition Challenge

Olga Russakovsky[1] · Jia Deng[2] · Hao Su[1] · Jonathan Krause[1] · Sanjeev Satheesh[1] · Sean Ma[1] · Zhiheng Huang[1] · Andrej Karpathy[1] · Aditya Khosla[3] · Michael Bernstein[1] · Alexander C. Berg[4] · Li Fei-Fei[1]

Received: 31 August 2014 / Accepted: 12 March 2015

Abstract The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images. The challenge has been run annually from 2010 (since 2010) and has become the standard benchmark for large-scale object recognition.[1] ILSVRC follows in the footsteps of the PASCAL VOC challenge (Everingham et al. 2012), established in 2005, which set the precedent for stan-

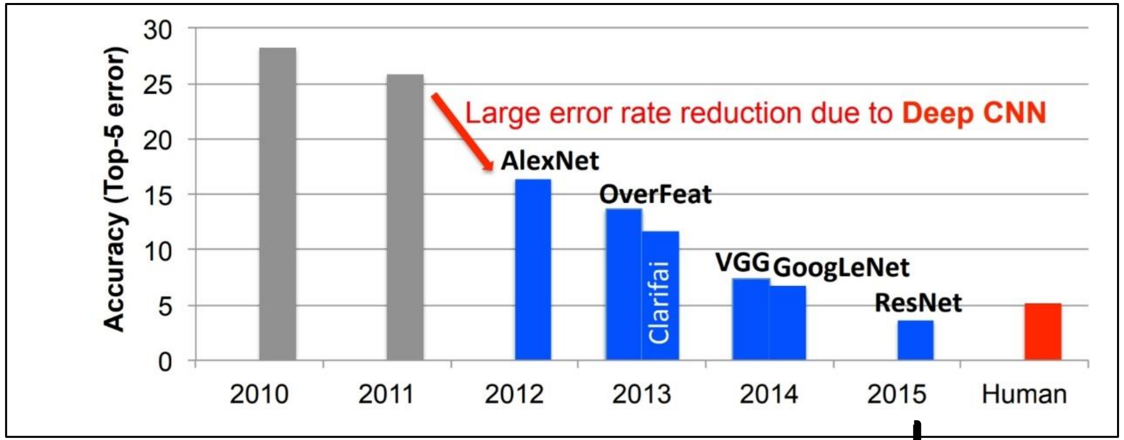
Russakovsky, O., Deng, J., Su, H. *et al.* ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vis* **115,** 211–252 (2015).



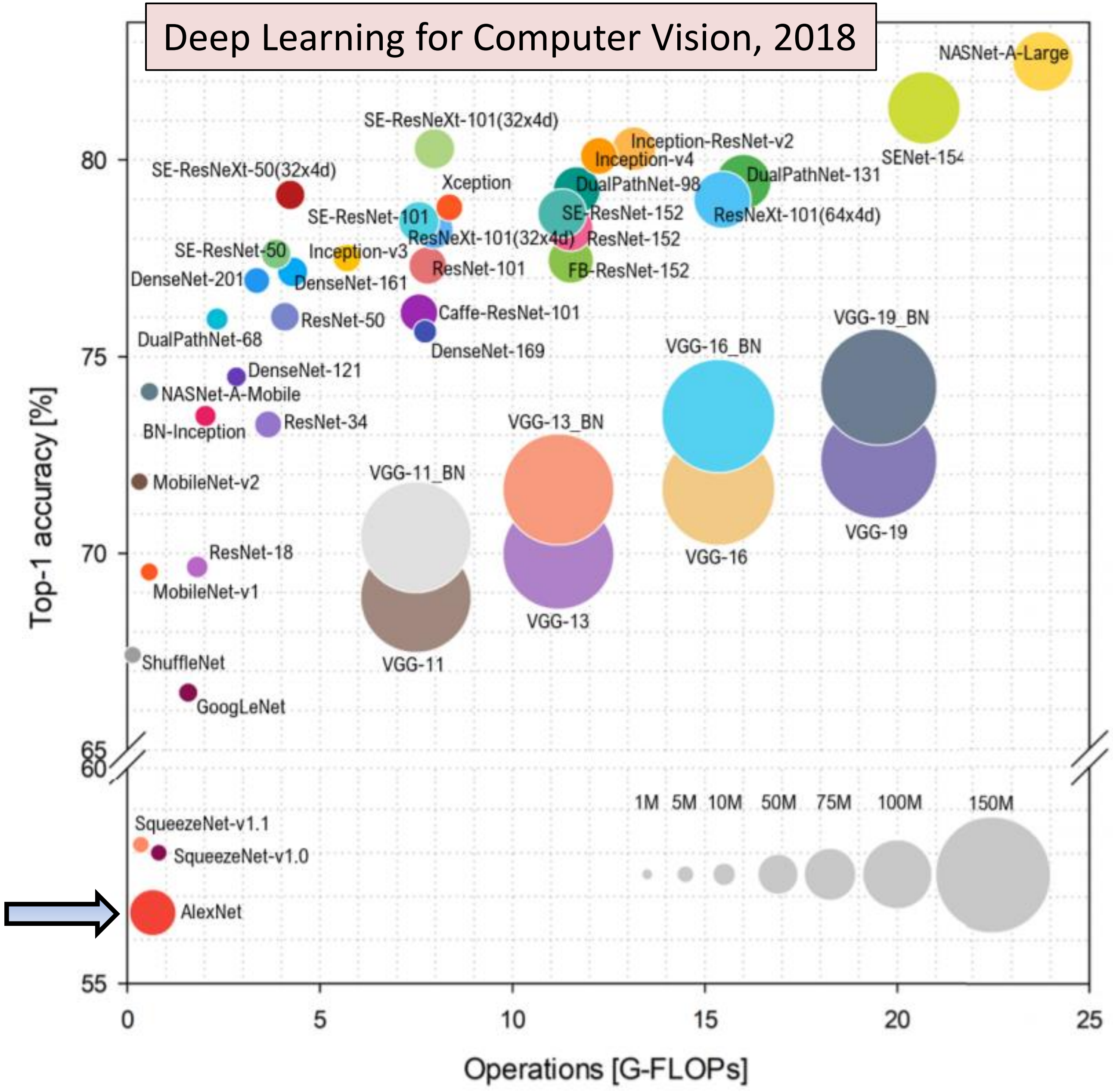
Randomly selected images from the data set (some images were taken from Flickr)

Deep Learning: Computer Vision

AlexNet (2012) was the first *convolutional neural network* (a deep neural network) to win the ImageNet Challenge.



- Meanwhile, **ResNet (2015)** was the first winner to exceed human-level vision accuracy.

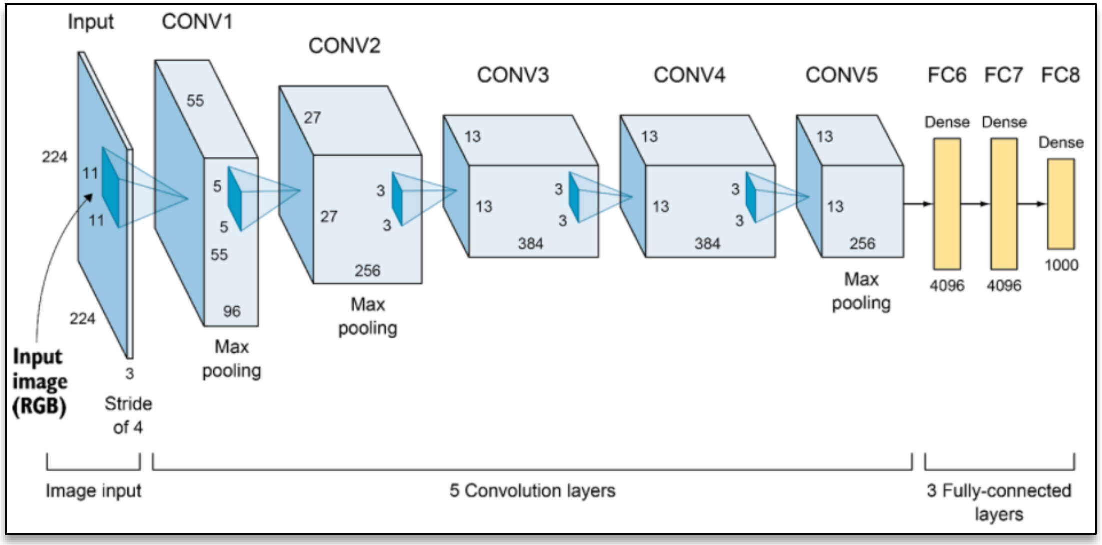


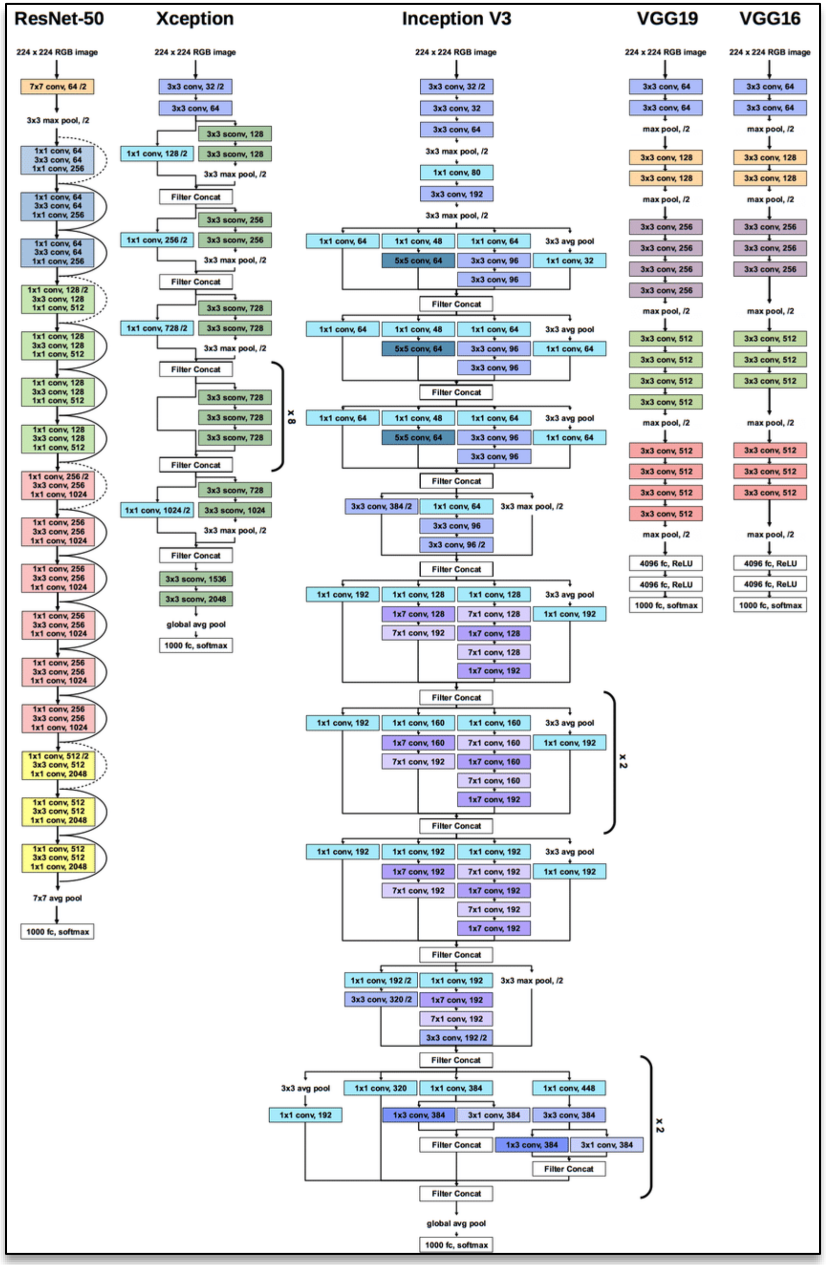
AlexNet
(2012)

Deep Learning: Computer Vision

AlexNet (2012) was the first *convolutional neural network* (a deep neural network) to win the ImageNet Challenge.

- Has **8 layers**: 5 are convolutional and 3 are fully-connected layers.
- Convolutional layers are used to detect certain features from the image regardless of where they are (translation-invariant).
- The Fully-connected layers are used for the actual classification.
- For training, Stochastic Gradient Descent + momentum was used.





ResNet:
50 layers

Xception:
71 layers

Inception V3:
48 layers

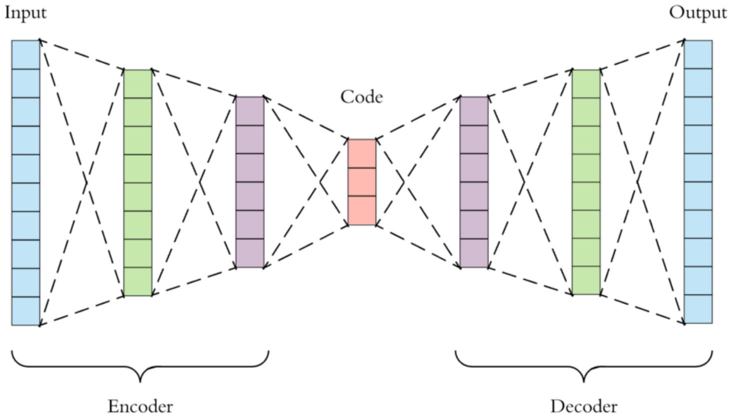
VGG-16:
41 layers

VGG-19:
47 layers

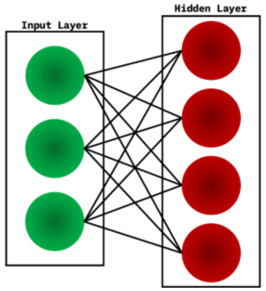
VGG = Visual Geometry Group, Dept. of Engineering Sciences, Oxford U.

Deep Learning Architectures

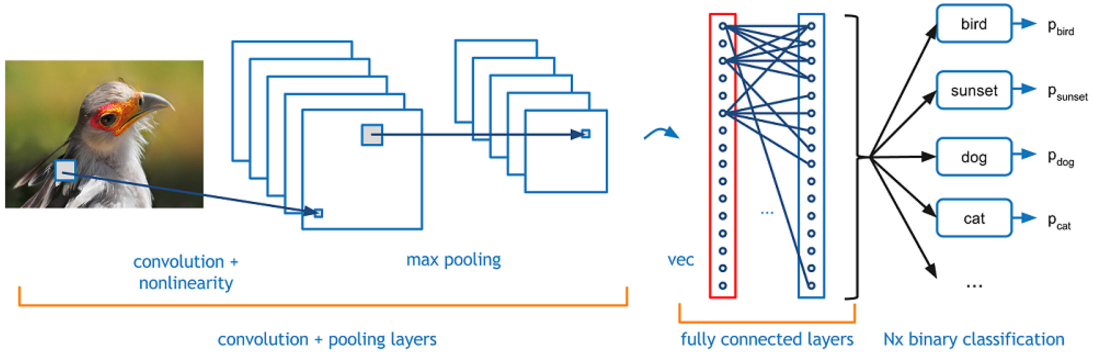
Autoencoders
(Dimensionality Reduction)



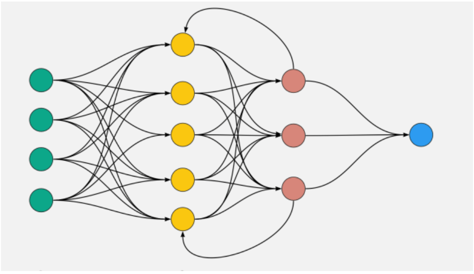
Restricted Boltzmann Machines
(Dimensionality Reduction, Density Estimation)

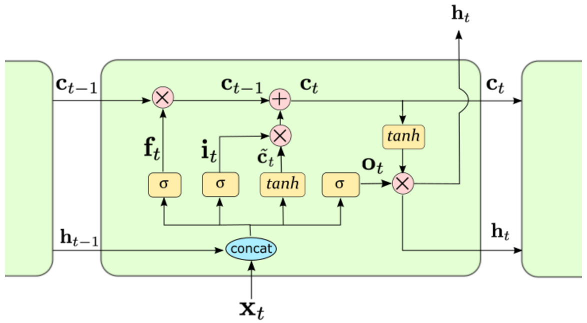


Convolutional Neural Networks
(Classification or Regression)



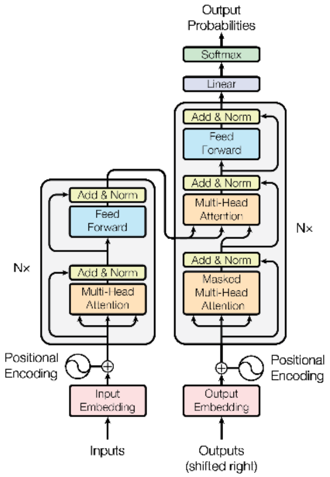
Recurrent Neural Nets
(Classification or Regression, Time Series)

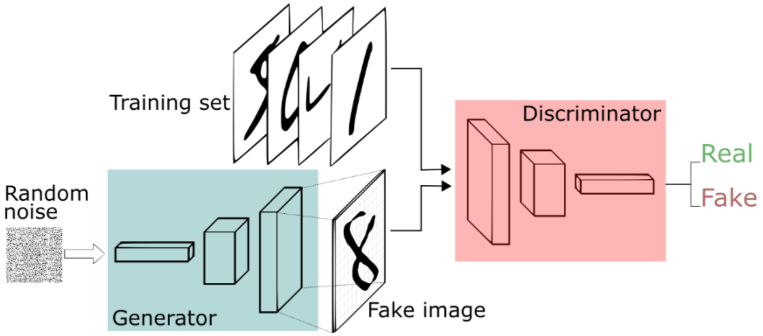




LSTM (Long Short-term Memory)
(Classification or Regression, Time Series)

Transformers
(Semi-supervised learning)





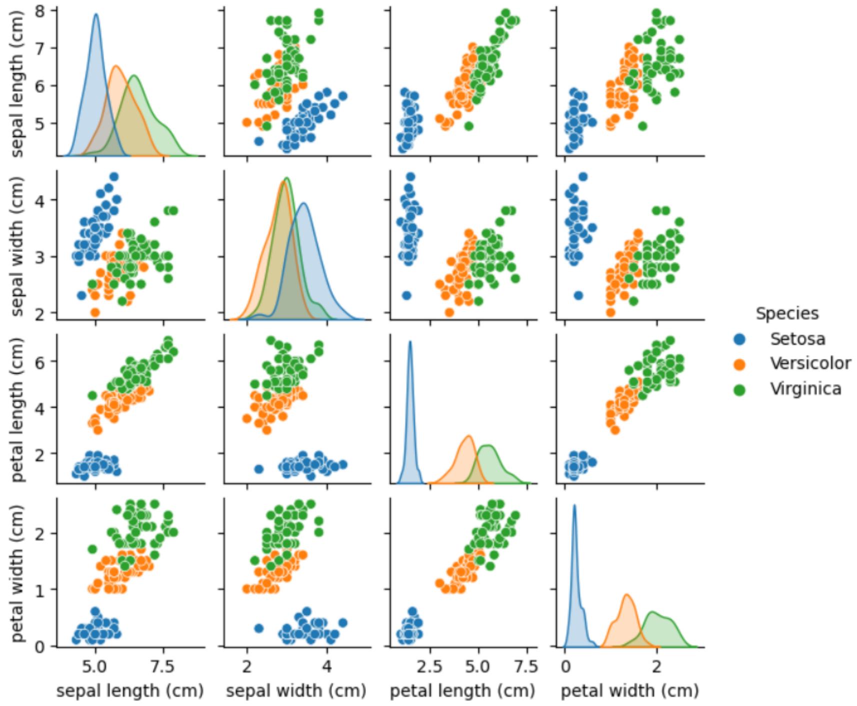
Generative Adversarial Networks
(Generative learning)

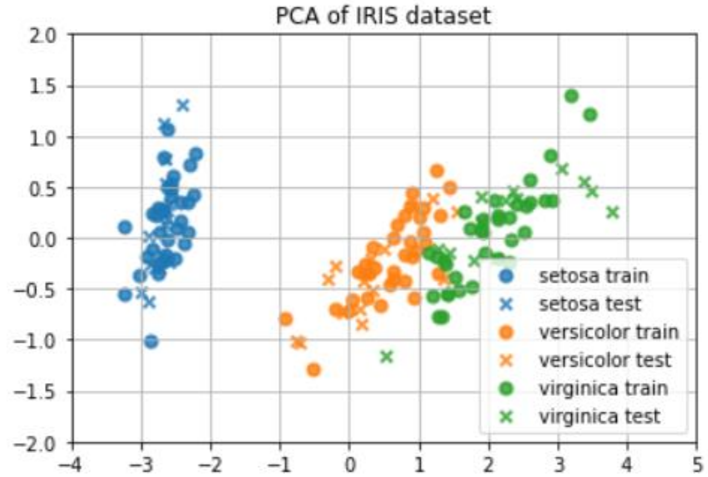
Deep Learning: Stacked Autoencoder

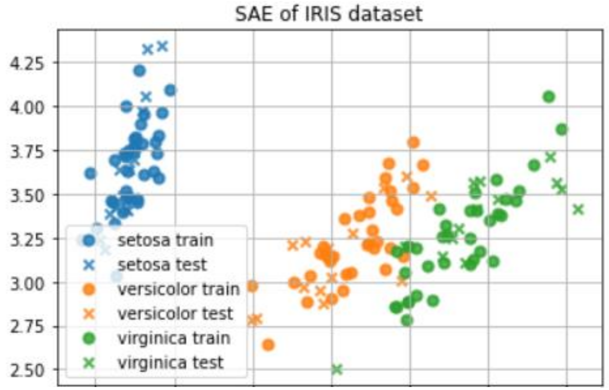
Example 3

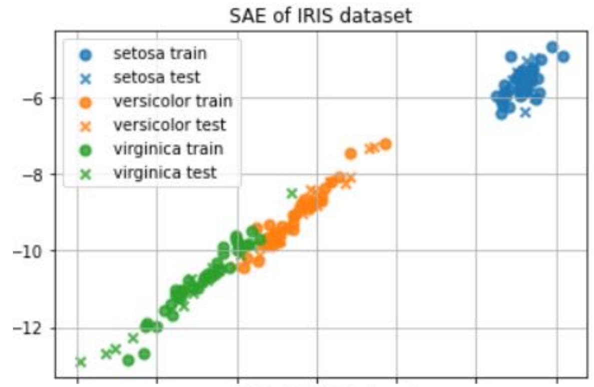
Compare the reduction of the Fisher Iris Data based on PCA from that based on the Stacked Autoencoder with the following settings:

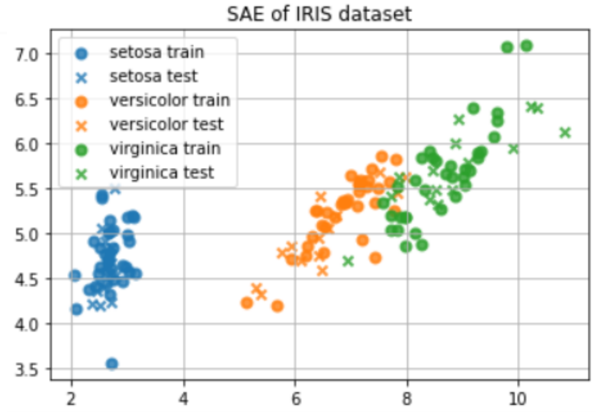
Architecture:	4-4-2-4
Learn Rate:	0.001
Epochs:	1000

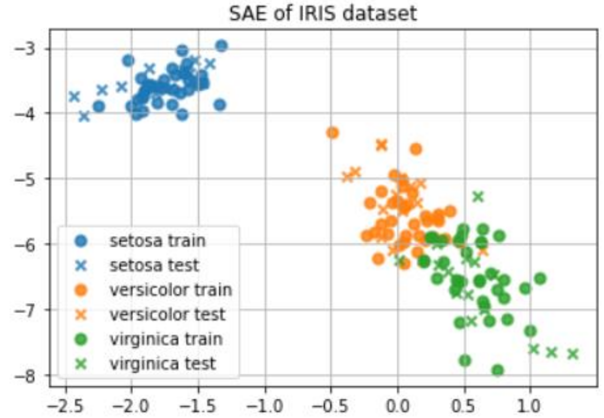






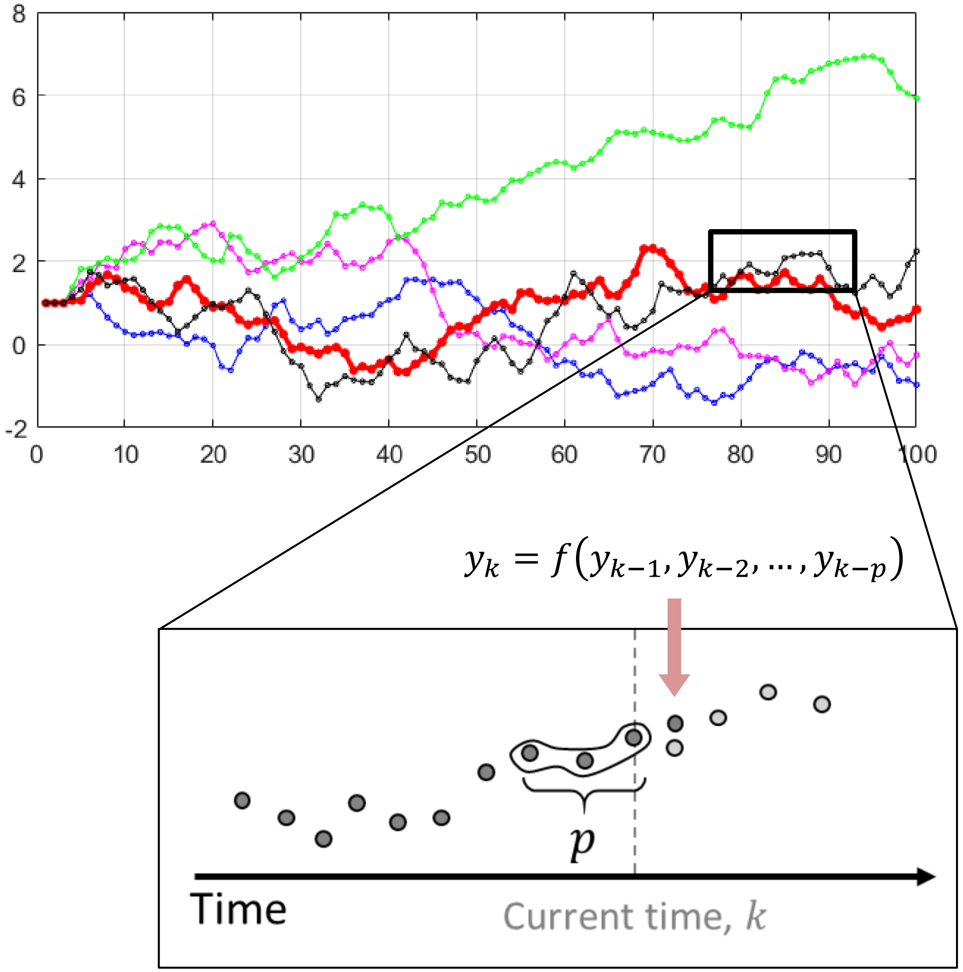






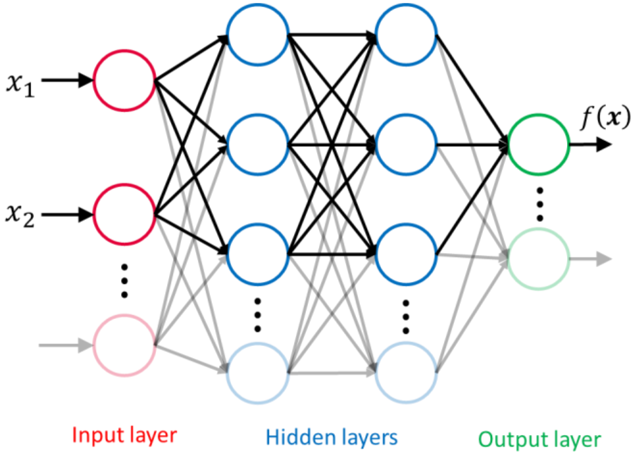
Deep Learning: Time Series

- For time series, models can be equipped with an *autoregressive* component.



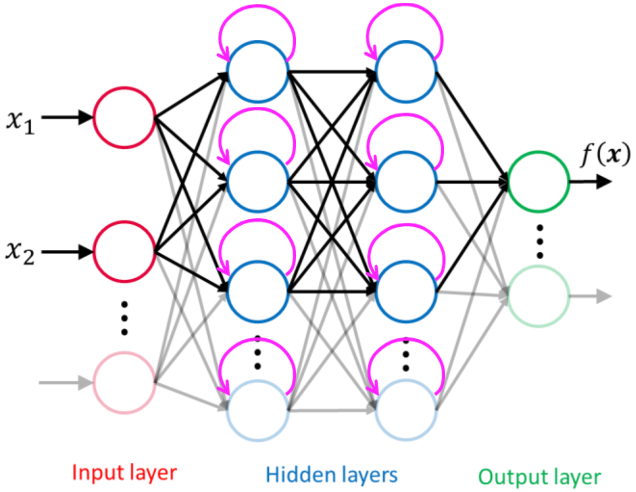
Artificial Neural Network (ANN)

All flow of information is from inputs to outputs only.



Recurrent Neural Network (RNN)

The hidden neurons are recurrent cells, where the information is internally fed back from output to input.

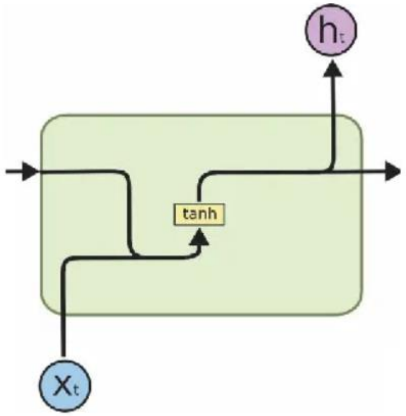


Deep Learning: Time Series

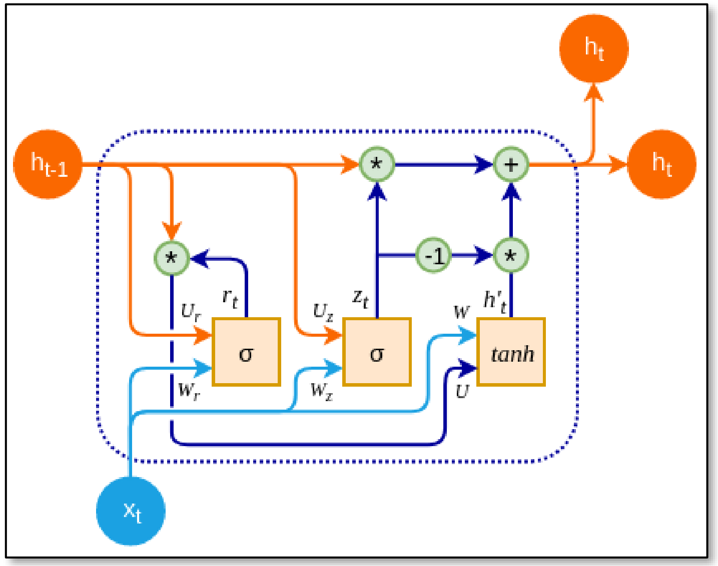
The typical DL models for time series are:

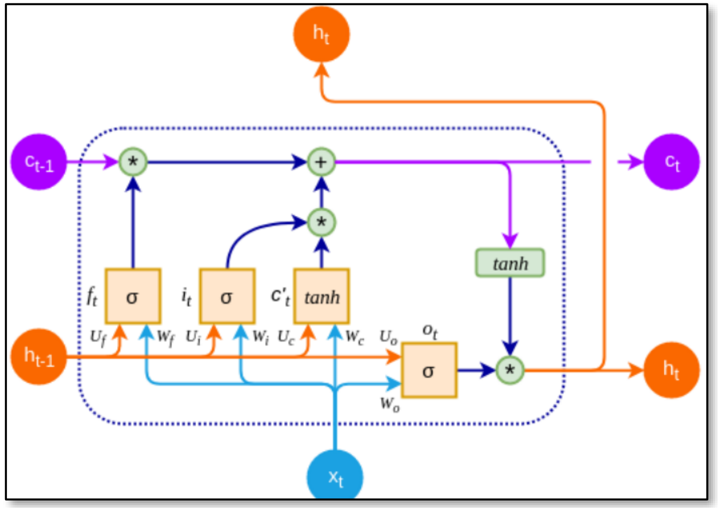
- Recurrent Neural Nets (RNNs)
- Gated Recurrent Units (GRUs)
- Long Short-term Memory (LSTMs)

Recurrent Neural Nets (RNNs)



- Contains recurrent units.
- Trained by Backpropagation Through Time (BPTT)
- Suffers from the *Vanishing Gradient* problem.
  - The partial derivative of the loss function approaches zero when there are many layers.





GRU

- Solves the *Vanishing Gradient* problem using gates, which are neural networks, σ, themselves.

- **Update Gate, z_t:** decides whether to update the state h_{t-1} with the new h'_t.
- **Reset Gate, r_t:** decides how much memory from the past h_{t-1} influences the present.

LSTM

- Similar to GRU but with added features.

- **Forget Gate, f_t:** decides how much the previous state h_{t-1} is remembered now.
- **Input Gate, i_t:** decides how much new information is retained.
- **Output Gate, o_t:** decides how the next hidden state is computed.
- **Cell state, c_t:** the key variable in LSTM that learns long-term dependencies.

Source: Vasilev (2019). Advanced Deep Learning with Python. Packt Publishing.

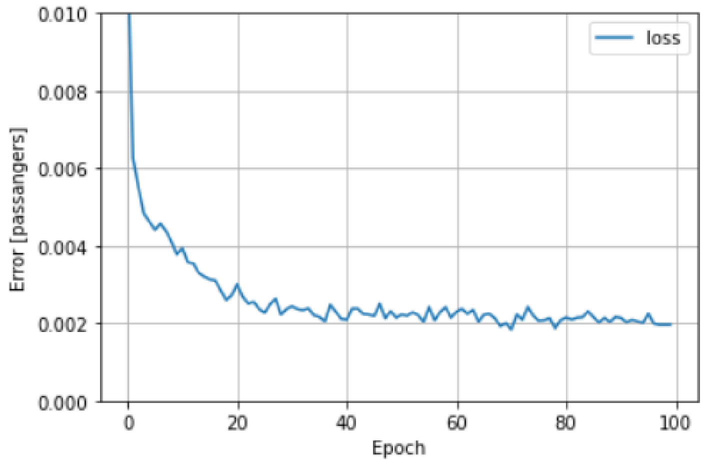
Deep Learning: Time Series

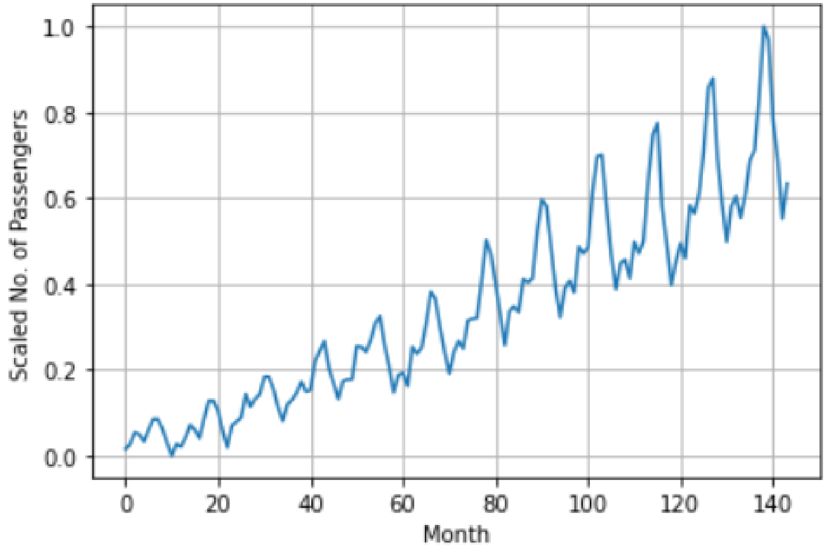
Example 4

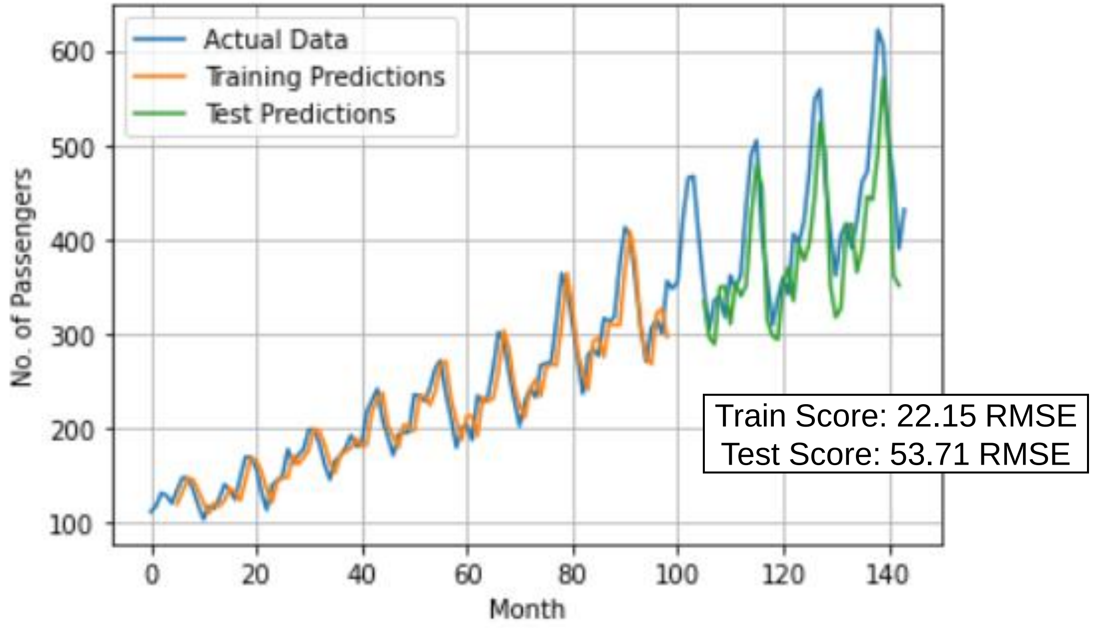
Train an LSTM-ANN for forecasting the last 30% of the data in the Airline Passengers Data Set. Use a single LSTM layer followed by a fully connected layer with 8 hidden neurons.

```
In [4]:  model = Sequential()
         model.add(LSTM(8, input_shape=(1, look_back)))
         model.add(Dense(8))
         model.add(Dense(1))
         model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 8)                 448
_________________________________________________________________
dense (Dense)                (None, 8)                 72
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 529
Trainable params: 529
Non-trainable params: 0
_________________________________________________________________
```







Source: Vasilev (2019). Advanced Deep Learning with Python. Packt Publishing.

Outline

- Artificial Neural Networks
  - Architecture
  - Activation Functions
  - Forward Propagation
  - Backpropagation
  - Regularization

- ANNs for Other Tasks
  - Introduction to Deep Learning
  - Convolutional Neural Nets (Images)
  - Autoencoders (Dimensionality Reduction)
  - RNNs, GRUs, LSTMs (Time Series)

Further Reading

- https://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic
- Werbos, Paul J. (1994). The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting. New York: John Wiley & Sons.
- F. Chollet. Deep Learning with Python (2018). Manning Publications Co.
- Hinton, Geoffrey E. “Connectionist learning procedures.” Artificial intelligence 40.1 (1989): 185-234.
- He, Kaiming, et al (2015). “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification.”
- Kingma, Diederik, and Jimmy Ba (2014) “Adam: A method for stochastic optimization.” https://doi.org/10.48550/arXiv.1412.6980
- Li et al. (2019). Gradient Descent with Early Stopping is Provably Robust to Label Noise for Overparameterized Neural Networks. arXiV. https://doi.org/10.48550/arXiv.1903.11680
- Srivastava et al. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research.
- LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015). https://doi.org/10.1038/nature14539
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256.
- https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- https://blog.ml.cmu.edu/2020/08/31/4-overfitting/
- Belkin et al. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. PNAS.
- Wang, H. and Raj, B. (2017). On the Origin of Deep Learning. https://doi.org/10.48550/arXiv.1702.07800
- Russakovsky, O., Deng, J., Su, H. *et al.* ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vis* **115,** 211–252 (2015).

Further Reading

- Bianco et al. (2018). *Benchmark Analysis of Representative Deep Neural Network Architectures*. IEEE Access. https://ieeexplore.ieee.org/document/8506339

- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems 25 (NIPS 2012)

- K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, "LSTM: A Search Space Odyssey," in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2222-2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924. https://ieeexplore.ieee.org/document/7508408

- Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Nets (PDF). Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.