UNIVERSITY OF MINNESOTA
Driven to Discover℠

EE 5561: Image Processing and Applications

Lecture 16

Mehmet Akçakaya

Recap of Last “Module”

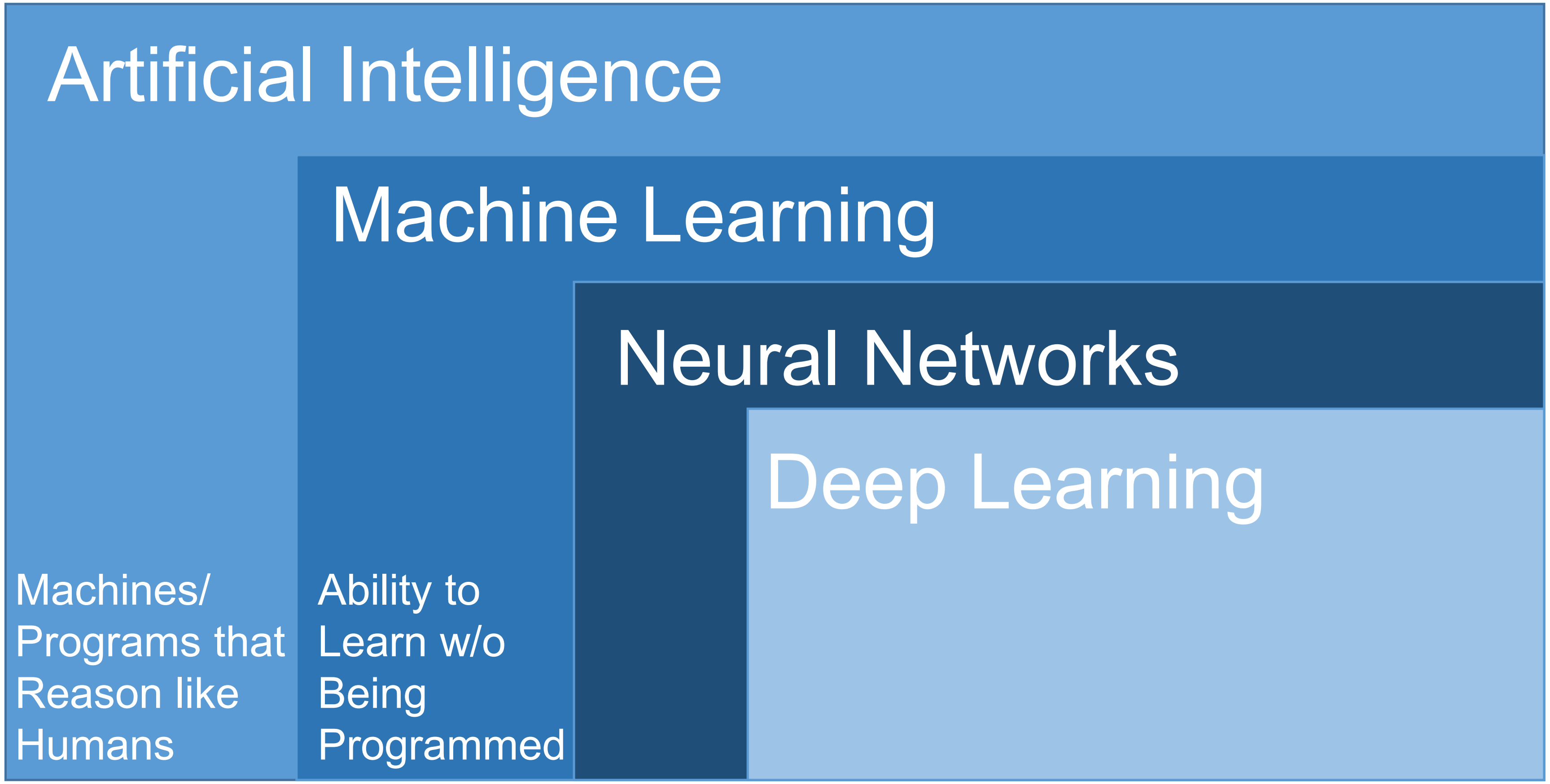
- Statistical image processing
  - Statistical models for **n** and for images **x**
  - For **x**, we relied on certain image properties, e.g.
    - Neighbors have similar signal intensity
    - Images are compressible in pre-designed transform domains
    - Image transformation vector fields are smooth
    - Self-similarity in images

- New module, new idea (machine learning/AI-based methods)
  - Learn such properties from large databases of images

Machine Learning

- One definition: A machine is said to learn from experience E, with respect to some class of <u>tasks</u> T and <u>performance measure</u> P, if
  - its performance at tasks in T measured by P improves with experience E.
- Tasks:
  - Classification
  - Segmentation
  - Anomaly detection
  - Denoising/restoration
  - Density estimation

Machine Learning



very “roughly”

Machine Learning

- Performance measures:
  - Mean squared error (between target/label and prediction)
  - L_1 error
  - Accuracy of predicted labels vs. true labels (e.g. classification)
  - Likelihood: Probability of predicting outcome for samples given model parameters
    - Consider training data (input, label) $\{(x_i, y_i)\}_{i=1}^n$
    - The likelihood is given as

model parameters

$$l(\boldsymbol{\theta}) = \log \mathbb{P}(y_1, \ldots, y_n | x_1, \ldots, x_n, \boldsymbol{\theta})$$

    - If training data samples are i.i.d., then

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \mathbb{P}(y_i | x_i, \boldsymbol{\theta})$$

Formulation

- X: input data
  - Categorial (i.e. discrete on finite set)
  - Continuous
- Y: label or output data
- $f_{\boldsymbol{\theta}}$: "task", i.e. the function we want to estimate/learn

- In ML, we consider function classes that are parametrized by some $\boldsymbol{\theta}$
  - e.g., neural networks

- $\mathcal{L}(f_{\boldsymbol{\theta}}(X), Y)$ loss function/performance metric/error

Formulation

- What are we learning?
- Given a database of input & label data

$$\mathcal{D} = \big\{(x_1, y_1), \ldots, (x_n, y_n)\big\}$$

our goal is to find the best function f_θ that minimizes some loss function (or optimizes some performance metric), i.e.

$$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

- We often think of (x_k, y_k) as samples from a distribution on (X,Y)
  - i.e. we are minimizing the sample mean loss

Formulation

- What are we learning?
- Given a database of input & label data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

our goal is to find the best function $f_{\boldsymbol{\theta}}$ that minimizes some loss function (or optimizes some performance metric), i.e.

$$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

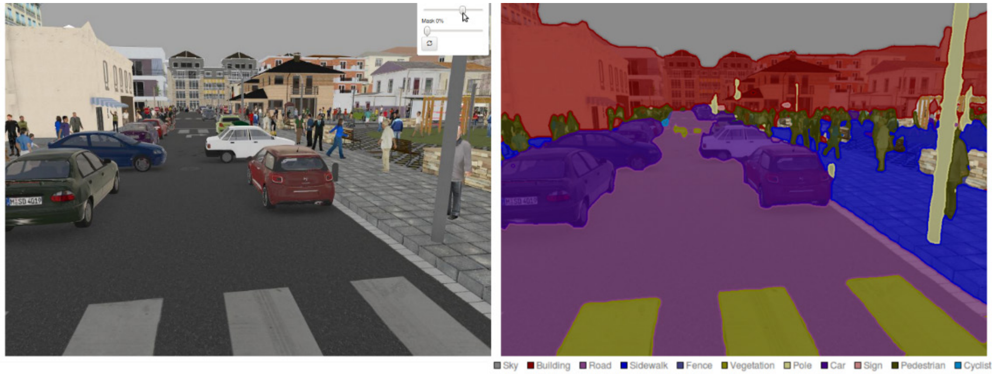
- This formulation is called *supervised* learning, i.e. we know the true labels for a given output
- *Unsupervised* learning exists (e.g. dimensionality reduction), and is a hot research topic

Formulation

- Different names for tasks based on what X and Y are
  - Classification (e.g. segmentation)

    X: categorical or continuous

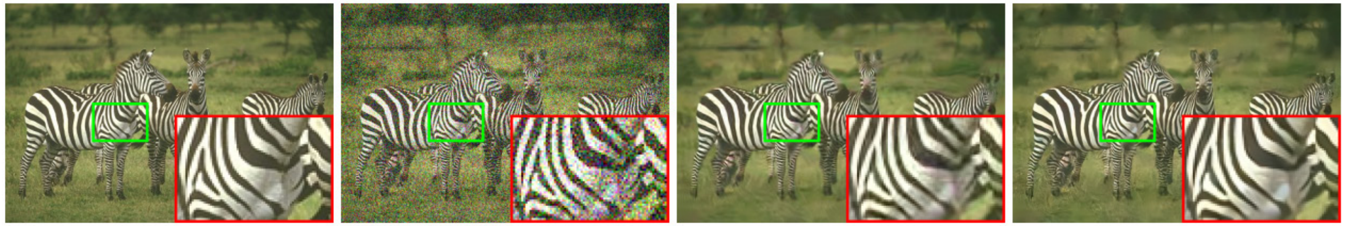
    Y: categorical



  - Regression (e.g. denoising)

    X: continuous or categorical

    Y: continuous

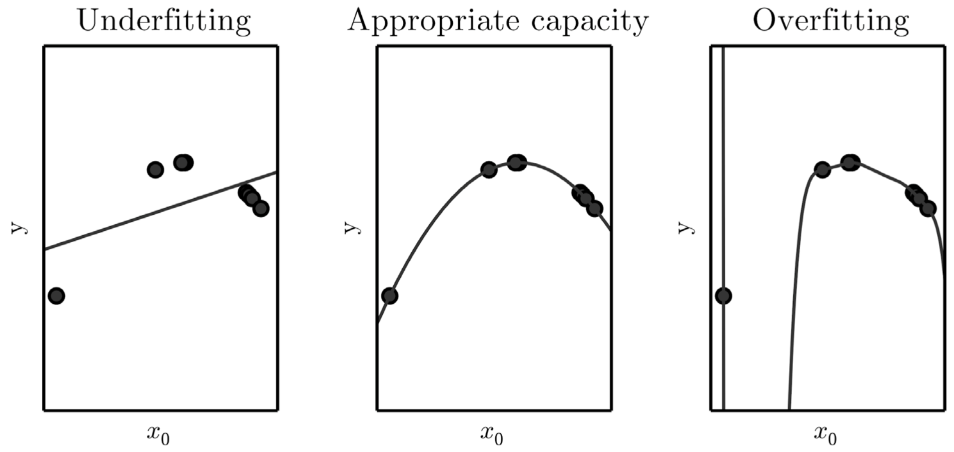


(a) Ground-truth (b) Noisy / 15.07dB (c) CBM3D / 26.97dB (d) CDnCNN-B / 27.87dB

- How to solve the cost/loss function?
  - Gradient descent (to be discussed later)

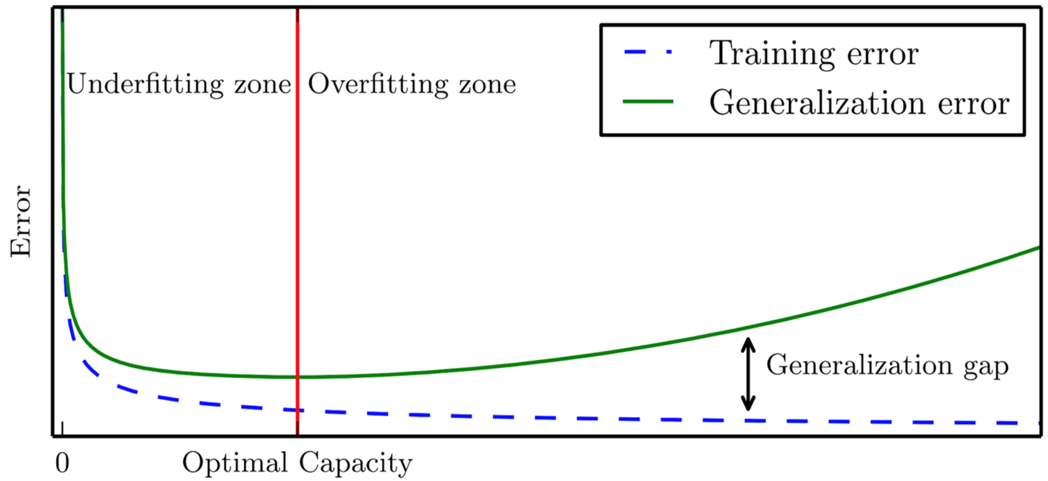
Model Capacity

- So far: task, training data, loss function, optimization
- Are we done?
  - We don't know what class of $\{f_\theta\}$ to consider
- More importantly: We need <u>generalization</u> in ML
  - i.e. The model should work for the task on *new data*
  - Otherwise it's easy to find f that minimizes the loss over given training database
  - How? e.g. Add more non-linear transformations/ parameters to your model



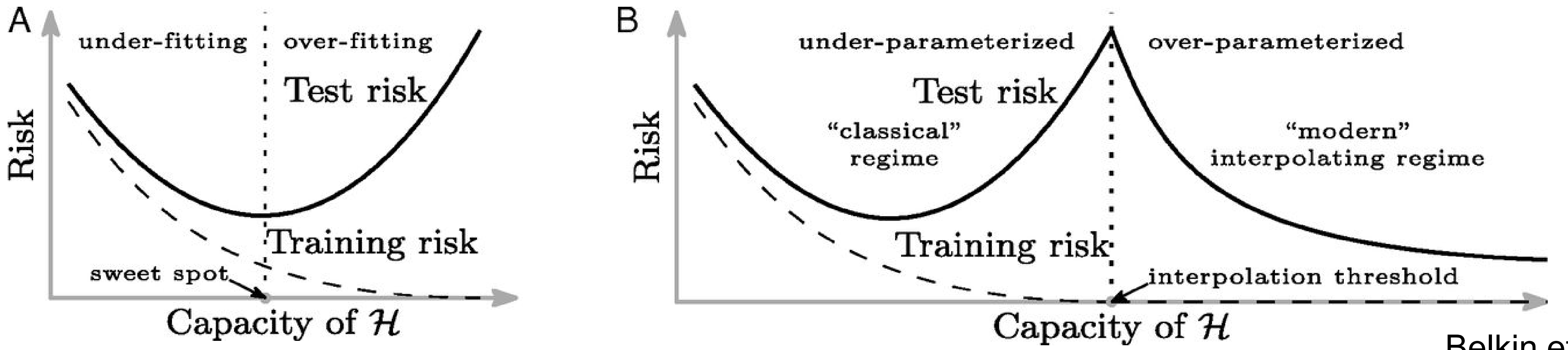
Model Capacity

- How do we do this in practice?
  - Cross-validation!
  - Randomly sample & set aside a part of the dataset (Test set)
  - Rest of the data is the training test (more on this later)
  - Optimize loss on the training set
  - Track loss on both training & test sets
  - Stop training or adding parameters to the model when the gap between losses on training & test sets starts to grow (generalization gap)



Model Capacity – In the Modern Era

- How do we do this in practice?
  - Cross-validation!
  - Randomly sample & set aside a part of the dataset (Test set)
  - Rest of the data is the training test (more on this later)
  - Optimize loss on the training set
  - Track loss on both training & test sets
  - Stop training or adding parameters to the model when the gap between losses on training & test sets starts to grow (generalization gap)



Belkin et al, PNAS, 2019

Functions to Learn

- Let's go back to the class of $\{f_\theta\}$ to consider
  - Structure of f in general is not learnable
  - This has to be decided prior to training/learning
  - Hyperparameters: Control the structure of f (and also training)
    - e.g. number of "layers" (structure)
    - e.g. step size in gradient descent (training)
  - One also needs to tune these hyperparameters
    - Now we need a third set in cross-validation:
      - Training
      - Testing (for checking optimality of learned parameters)
      - Validation (for checking optimality of hyperparameters)

Feed-Forward Networks

- We will consider multi-layer structures

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f^{[h]}_{\boldsymbol{\theta}_h}\Big(\ldots\big(f^{[2]}_{\boldsymbol{\theta}_2}(f^{[1]}_{\boldsymbol{\theta}_1}(\mathbf{x}))\big)\Big)$$

$$\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_h]$$

- Called feed-forward networks (or multi-layer perceptron)
- Layers 1 to *h*-1 will contribute to output (but indirectly) → hidden layers
- What's the simplest structure for $f^{[k]}_{\boldsymbol{\theta}_k}$?
  - Linear or affine functions, e.g.

$$f^{[k]}_{\boldsymbol{\theta}_k}(\mathbf{x}) = \underbrace{\mathbf{W}^{[k]}}_{\text{matrix}}\mathbf{x} + \underbrace{\mathbf{b}^{[k]}}_{\text{vector or scalar}}$$

  - Here: $\boldsymbol{\theta}^{[k]}_k = [\mathbf{W}^{[k]}, \mathbf{b}^{[k]}]$

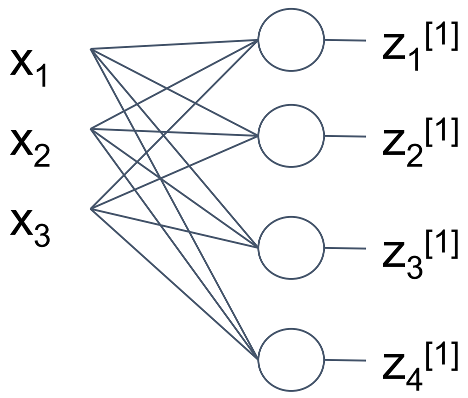
Feed-Forward Networks

- For example

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & W_1^{[1]^T} & — \\ — & W_2^{[1]^T} & — \\ & \vdots & \\ — & W_4^{[1]^T} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

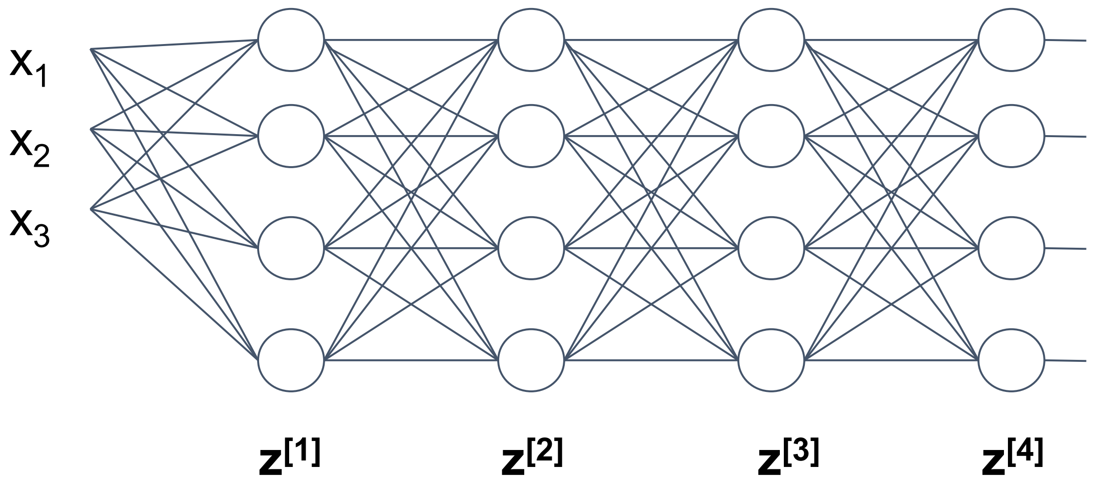
$$z^{[1]} = W^{[1]} x + b^{[1]}$$

- We can look at this as a graph



Feed-Forward Network

- What happens when we have multiple layers of these?



$$\mathbf{z}^{[4]} = \mathbf{W}^{[4]}\mathbf{z}^{[3]} + \mathbf{b}^{[4]}$$

$$\mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{z}^{[2]} + \mathbf{b}^{[3]}$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

Feed-Forward Networks

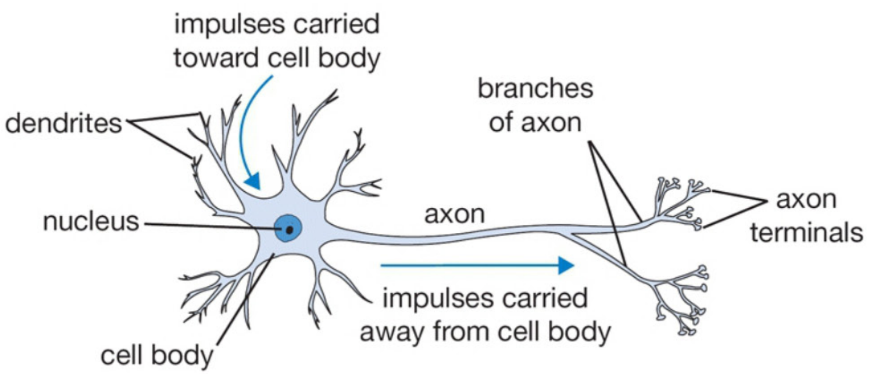
- What happens when we have multiple layers of these?

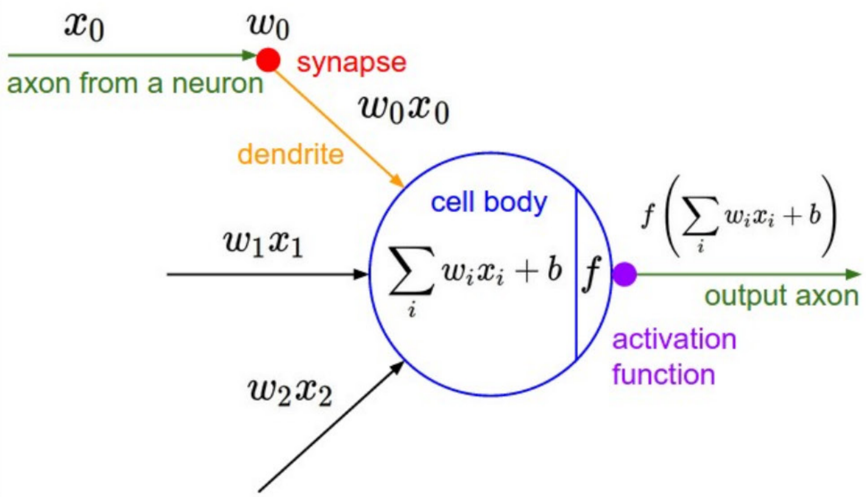
$$
\begin{aligned}
f_{\boldsymbol{\theta}}(\mathbf{x}) &= f^{[h]}_{\boldsymbol{\theta}_h}\Big(\dots \Big(f^{[2]}_{\boldsymbol{\theta}_2}\big(f^{[1]}_{\boldsymbol{\theta}_1}(\mathbf{x})\big)\Big)\Big) \\
&= \mathbf{W}^{[n]}\Big(\dots \Big(\mathbf{W}^{[2]}\big(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}\big)\big) + \mathbf{b}^{[2]}\Big) \dots + \mathbf{b}^{[n]}\Big) \\
&= \mathbf{W}^{[n]} \dots \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[n+1]}
\end{aligned}
$$

  - Still linear! (affine)
  - Not much for power for representation by adding multiple layers in this linear setup
- We need nonlinearities in our layers
- Neural networks give us this power

Neural Networks

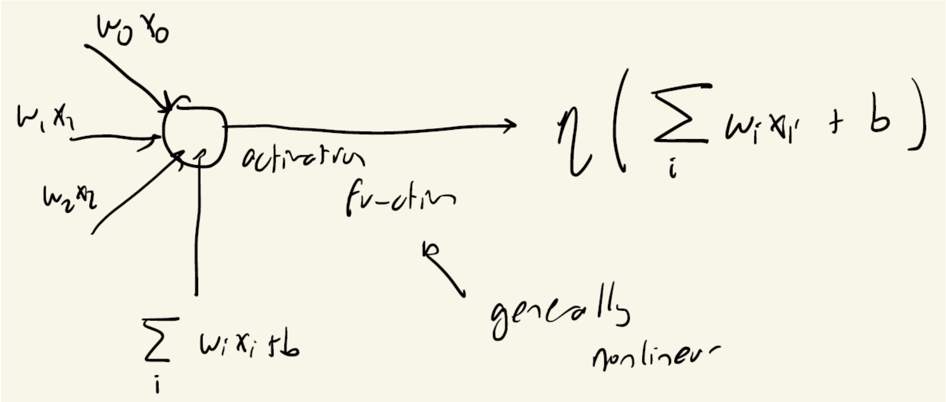
- Neural networks
  - Neuron: Basic computational unit of the brain
  - Loosely inspired by neurons





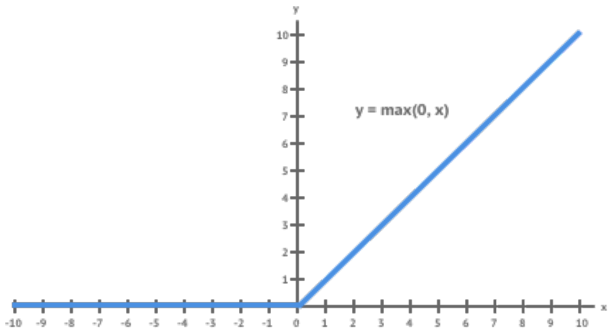
Neural Networks

- Neural networks
  - Neuron: Basic computational unit of the brain
  - Loosely inspired by neurons
  - Our mathematical model consists of combining incoming signals and applying an activation function



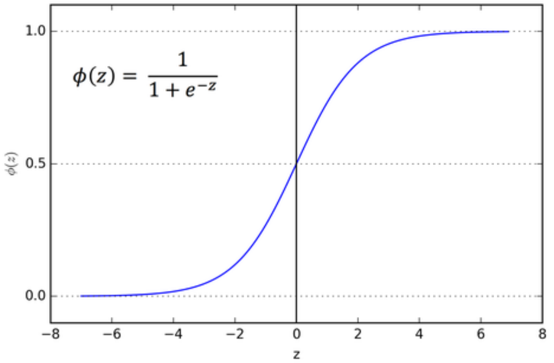
Activation Functions

- Rectified linear unit (ReLU)



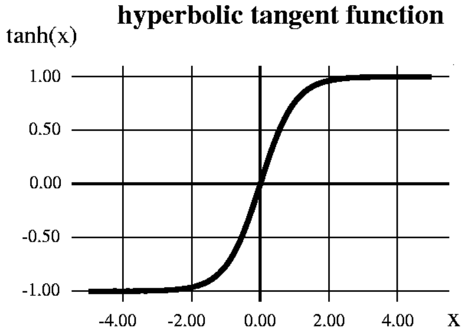
$$y = \max(x, 0)$$

- Sigmoid



$$y = \frac{1}{1 + e^{-x}}$$

- Hyperbolic tangent



$$y = \tanh(x)$$

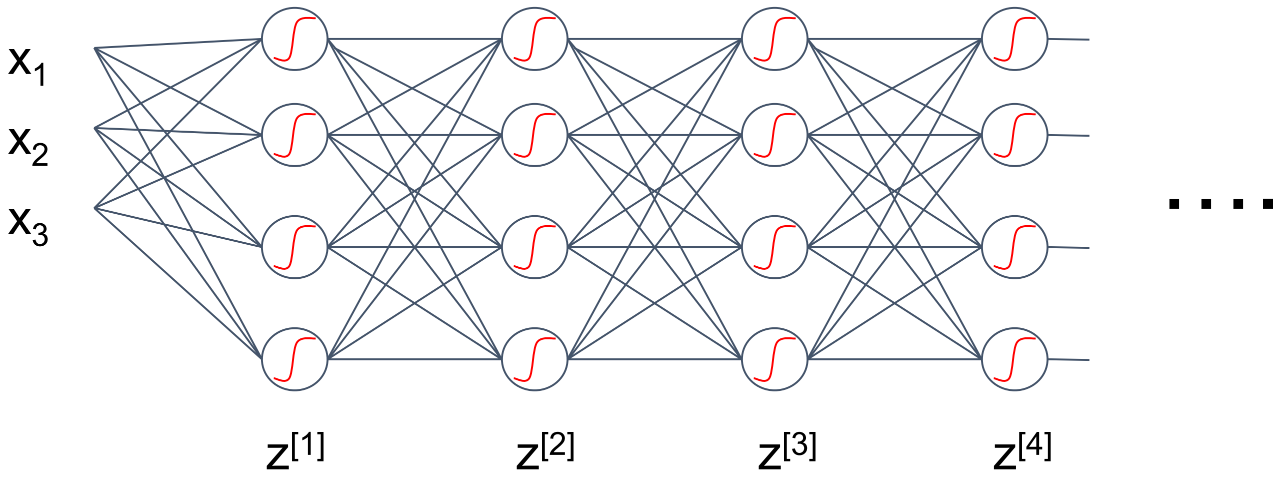
Neural Networks

- Now we can have a multi-layer neural network

$$\mathbf{z}^{[1]} = \eta\big(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}\big)$$

$$\vdots$$

$$\mathbf{z}^{[n]} = \eta\big(\mathbf{W}^{[n]}\mathbf{z}^{[n-1]} + \mathbf{b}^{[n]}\big)$$



Neural Networks

- Now we can have a multi-layer neural network

$$\mathbf{z}^{[1]} = \eta\left(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}\right)$$

$$\vdots$$

$$\mathbf{z}^{[n]} = \eta\left(\mathbf{W}^{[n]}\mathbf{z}^{[n-1]} + \mathbf{b}^{[n]}\right)$$

- Note:
  - Last (n^{th}) layer may or may not have an activation
  - Dimensionality of each layer can be different
  - Each layer can have different activation functions
  - We can do weight-sharing in layers (e.g. convolutional neural networks)
  - …

Neural Networks

- What happens when we have multiple layers of these?

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \eta\left(\mathbf{W}^{[n]}\left(\dots\eta\left(\mathbf{W}^{[2]}\eta(\mathbf{W}^{[1]}\mathbf{x}+\mathbf{b}^{[1]})\right)+\mathbf{b}^{[2]}\right)\dots+\mathbf{b}^{[n]}\right)\Bigg)$$

  - Last (n^{th}) layer may or may not have an activation

- Our parameters are

$$\boldsymbol{\theta} = \{\mathbf{W}^{[n]},\dots,\mathbf{W}^{[1]},\mathbf{b}^{[n]},\dots,\mathbf{b}^{[1]}\}$$

- Choice of η is a hyperparameter (design stage)

Recap

- Machine learning concepts
  - Tasks, performance measures
  - X: input data, Y: label
  - $f_{\boldsymbol{\theta}}$: function we want to estimate/learn, parametrized by $\boldsymbol{\theta}$
  - $\mathcal{L}(f_{\boldsymbol{\theta}}(X), Y)$ loss function
  - Sample mean of loss

$$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

  - Multi-layer networks
  - Activation functions
  - Neural networks