CAI 4104/6108 – Machine Learning Engineering:

Neural Networks

Prof. Vincent Bindschaedler

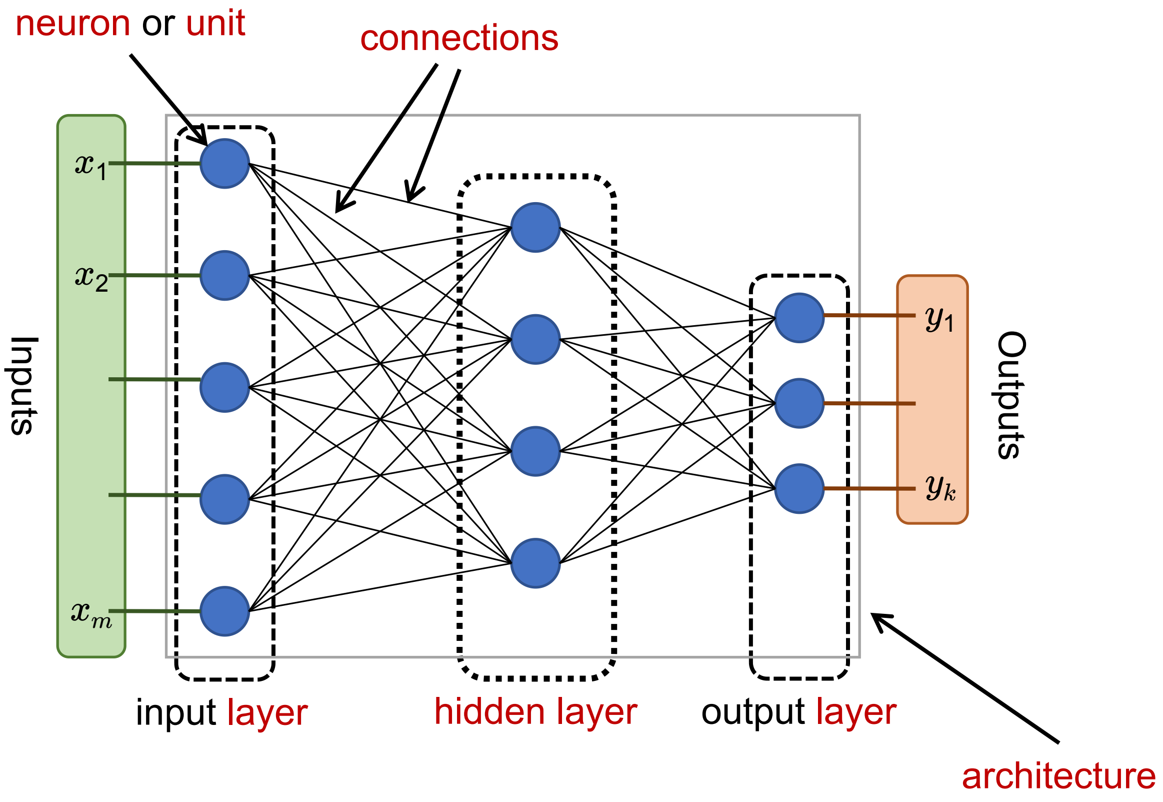
Spring 2024

History of Neural Networks

- (Artificial) Neural Networks
  - Large class of models / architecture
  - **Loosely** inspired by the biology of our brains

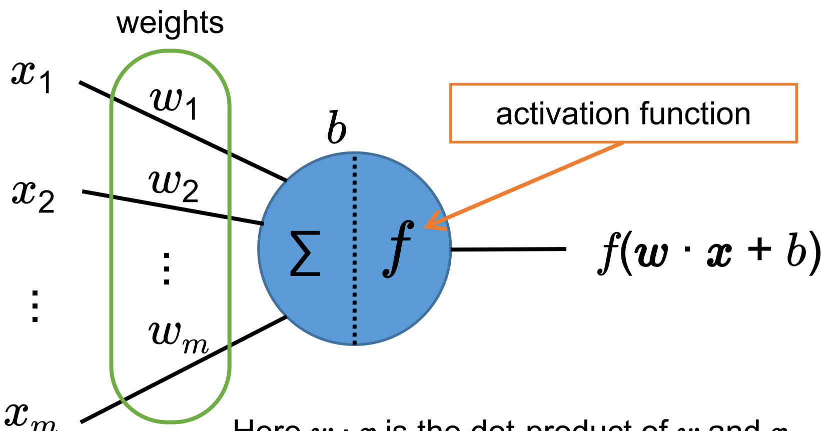
- Timeline
  - 1943: McCulloch and Pitts. "A logical calculus of the ideas immanent in nervous activity." Bulletin of Mathematical Biophysics.
  - 1958: Perceptron algorithm
  - 1960s: backpropagation derived by many researchers independently
  - 1980s: application of backpropagation to multi-layer neural networks by Rumelhart, Hinton and Williams (1986) and Yann Lecun in his PhD thesis (1987)
  - 2010s: deep learning revolution
    - greater availability of data; more computational power; techniques to overcome difficulty of training deep neural networks; fast implementations on GPUs, etc.

Neural Network Terminology



A Simple Neural Network

- Consider a single neuron / unit
  - The model is $h_{\boldsymbol{w},b}(\boldsymbol{x}) = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$
    - What if we take f to be the identity function?
      - That is: $f(z) = z$
    - What if we take f to be the sigmoid / logistic function?
      - That is: $f(z) = 1/(1+e^{-z})$

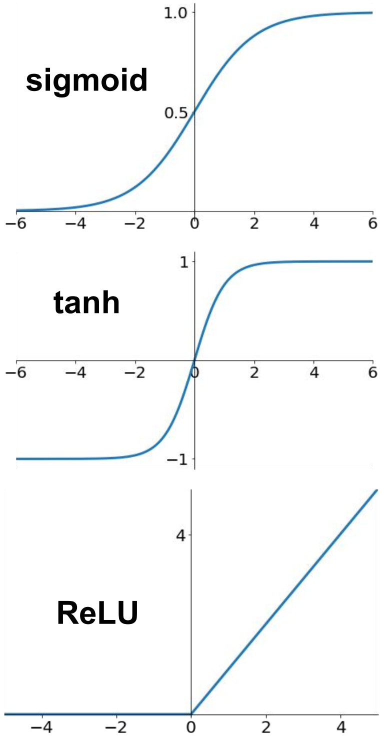


Here $\boldsymbol{w} \cdot \boldsymbol{x}$ is the dot-product of $\boldsymbol{w}$ and $\boldsymbol{x}$.
So $f(\boldsymbol{w} \cdot \boldsymbol{x} + b) = f(\sum w_i x_i + b)$

- The Perceptron
  - Invented by Frank Rosenblatt in 1957
    - "The Perceptron—a perceiving and recognizing automaton". Report 85-460-1. Cornell Aeronautical Laboratory
  - A different neuronal architecture called a threshold linear unit (TLU)
    - No bias term
    - With a step activation function. For example:
      - heaviside(z) = 0 if z ≤ 0, 1 otherwise (z ≥ 1) ; or sign(z)

Components

- Types of Layers
  - Dense (i.e., fully-connected)
  - Convolutional
  - Recurrent
- Activation Functions
  - Identity / Linear (or none): $f(z) = z$
  - Sigmoid: $f(z) = 1/(1+e^{-z})$
  - TanH: $f(z) = (e^{z} - e^{-z}) / (e^{z} + e^{-z})$
  - ReLU: $f(z) = \max(0, z)$
  - Softmax: $f(z_j) = \exp(z_j / T) / \sum_i \exp(z_i / T)$
    - Note: in that case the activation function is over an entire layer, not a single unit
- Loss
  - Whatever you like (e.g., squared error loss) as long as it's differentiable
    - Note: make sure the loss function and activation function of the output layer are consistent with each other!



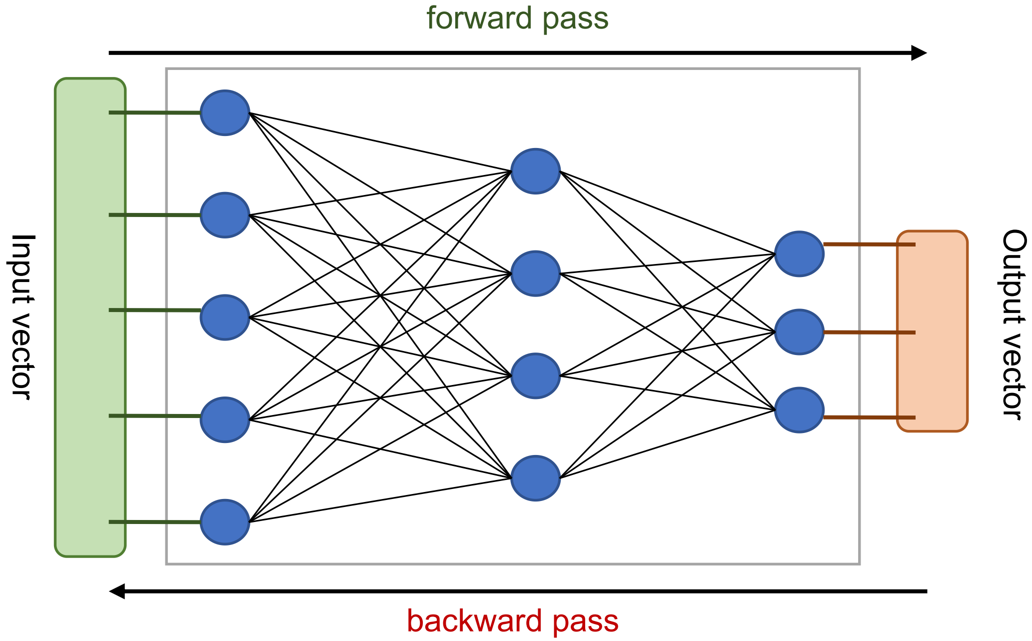
Examples & Special Cases

- Single neuron?
  - Linear regression
    - One layer neural network with a single neuron with a linear activation function and MSE as loss function
  - (Binary) Logistic regression
    - One layer neural network with a single neuron with a sigmoid activation function and binary cross-entropy as loss function

- Multi-Layer Perceptron (MLP)
  - Input layer (passthrough)
  - One or more hidden layers
  - Output layer

- Neural network with multiple layers all with linear activation
  - Q: Good idea?
    - **No. A linear function of a linear function is still a linear function!**

- Backpropagation:
  - Reverse pass to measure error and propagate error gradient backwards in the network
    - Using gradient descent to update the parameters

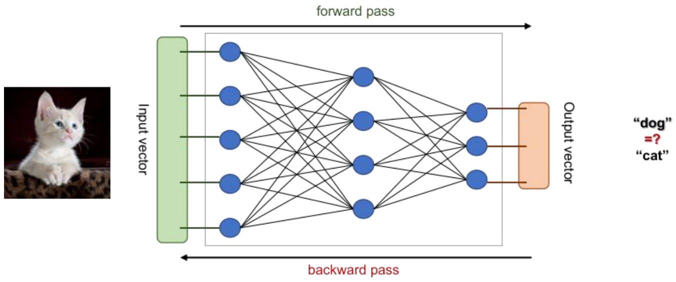


"dog"
=?
"cat"

Adjust the weights (parameters) to decrease the loss

Backpropagation

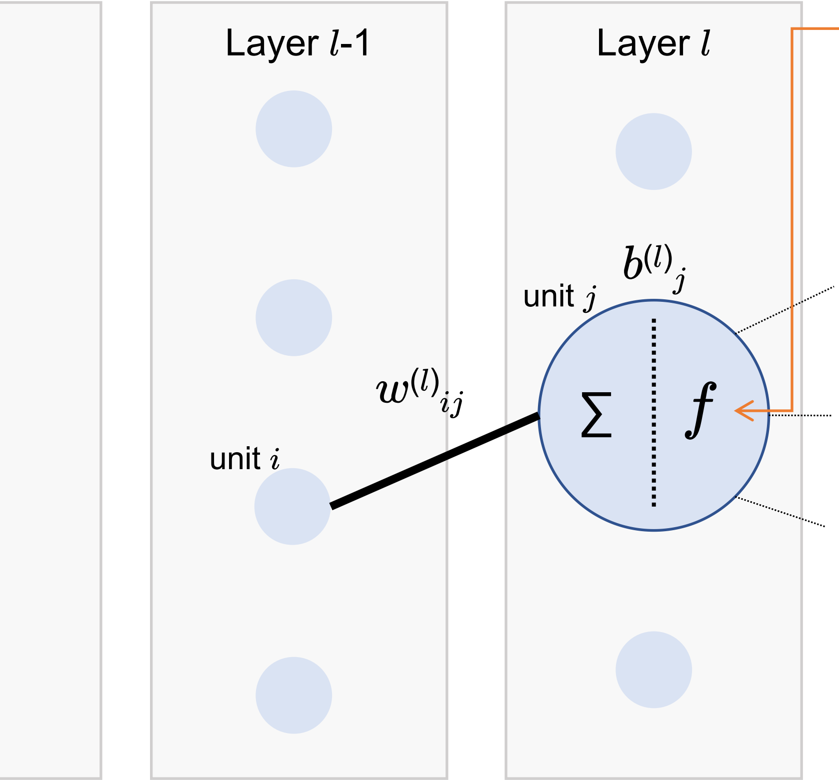
- Seminal Paper:
  - "*Learning representations by back-propagating errors.*" Rumelhart, Hinton, and Williams. Nature 1986.



- Terminology
  - Backpropagation: how to compute the gradients efficiently
  - Gradient descent: how to update the parameters to minimize the loss given the gradient

- Algorithm: given a mini-batch B
  - Compute the forward pass for the mini-batch B saving the intermediate results at each layer
  - Compute the loss on the mini-batch B (compares output of network to labels/targets → error)
  - Backwards pass: computes the per-weight gradients (error contribution) layer by layer
    - This is done using the chain rule (if z depends on y and y depends on x: $dz/dx = dz/dy \cdot dy/dx$)
  - (Stochastic) gradient descent: update the weights based on the gradients

Backpropagation: Illustration



output: $a^{(l)}_j = f^{(l)}(z^{(l)}_j)$ $\quad z^{(l)}_j = \sum_i w^{(l)}_{ij} a^{(l-1)}_i + b^{(l)}_j$

- How should we update $w^{(l)}_{ij}$?
  - Based on $\partial L/\partial w^{(l)}_{ij}$ where L is the loss
  - How to compute $\partial L/\partial w^{(l)}_{ij}$?
    - Activation output $f^{(l)}$ is a function of $z^{(l)}_j$ and $z^{(l)}_j$ is a function of $w^{(l)}_{ij}$
    - Chain rule: $\partial L/\partial w^{(l)}_{ij} = \partial L/\partial z^{(l)}_j \cdot \partial z^{(l)}_j/\partial w^{(l)}_{ij}$
    - $\partial z^{(l)}_j/\partial w^{(l)}_{ij} = a^{(l-1)}_i$
  - If we let $\delta^{(l)}_j = \partial L/\partial z^{(l)}_j$, we see that $\delta^{(l-1)}_i$ depends on $\delta^{(l)}_j$
    - So we can compute the gradients right to left (i.e., backwards)

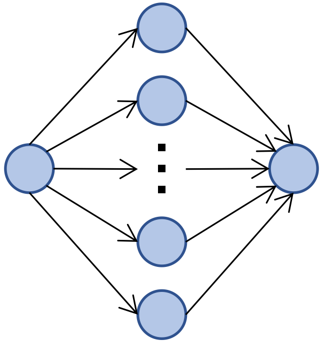
When To Use Neural Networks?

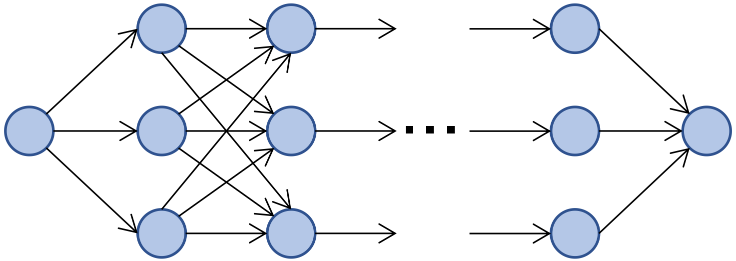
- You have a complex learning task or complex data
  - For many problems, neural networks provide the best performance (for some tasks performance is better than humans)
    - E.g.: image classification / captioning tasks
    - E.g.: speech recognition
    - E.g.: natural language modeling

- When should you NOT use a neural network?
  - If you can solve your problem with a simple model (e.g., linear model, decision tree, SVM)
  - If you do not have **a lot** of data
    - Neural networks (esp. deep neural networks) require a lot of data to achieve good performance
  - You need an explainable/interpretable model
    - Note: there exists techniques to explain decisions from neural networks

Universality of Neural Network

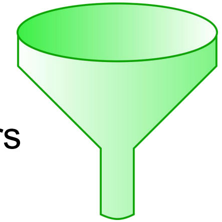
- Universal Approximation Theorems
  - *(Feed-forward) Neural networks can* approximately *represent any function*
  - Arbitrary width; bounded depth:
    - True even if we have a single hidden layer as long as it can have arbitrarily many units
  - Bounded width; arbitrary depth:
    - True even if we have layers of bounded width, as long as the network can have arbitrarily many layer





Neural Network Architecture?

- Pitfall: inconsistent activation function of output layer with the loss function
  - Examples:
    - Multiclass classification with cross-entropy loss, softmax activation for output layer => **Okay**
    - Regression with MSE as loss, tanh as activation for output layer => **Fail**
    - Regression with MSE as loss, linear activation for output layer => **Okay**
- Tip: the “funnel”
  - For supervised learning we typically have large input feature vectors and small output vectors
    - We should make the network look like a funnel



- Example: Multiclass classification with 10 classes and m=100 input features.
  - The network could look like this:
    - (Input, hidden layer 1, hidden layer 2, hidden layer 3, output layer)
      - 100, 64, 32, 16, 10
    - Activations:
      - Output: Softmax
      - Elsewhere: ReLU

Next Time

- Wednesday (2/28): Lecture

- Upcoming:
  - Homework 3 will be out soon