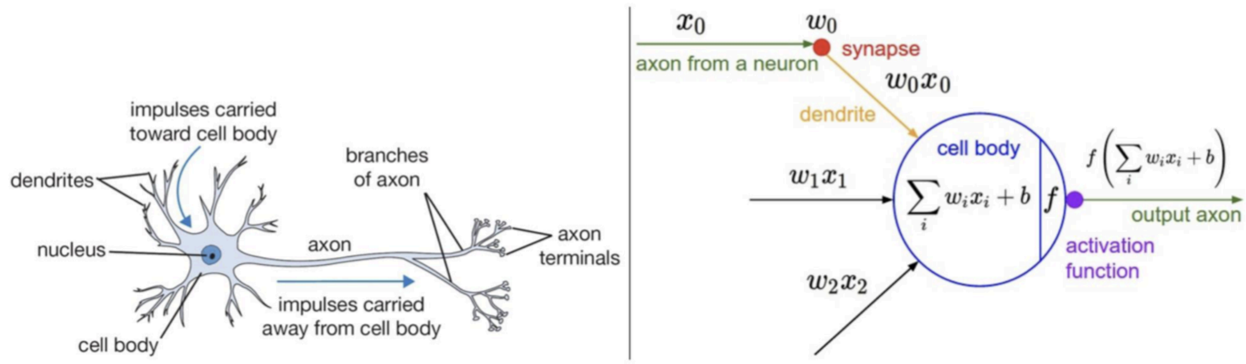
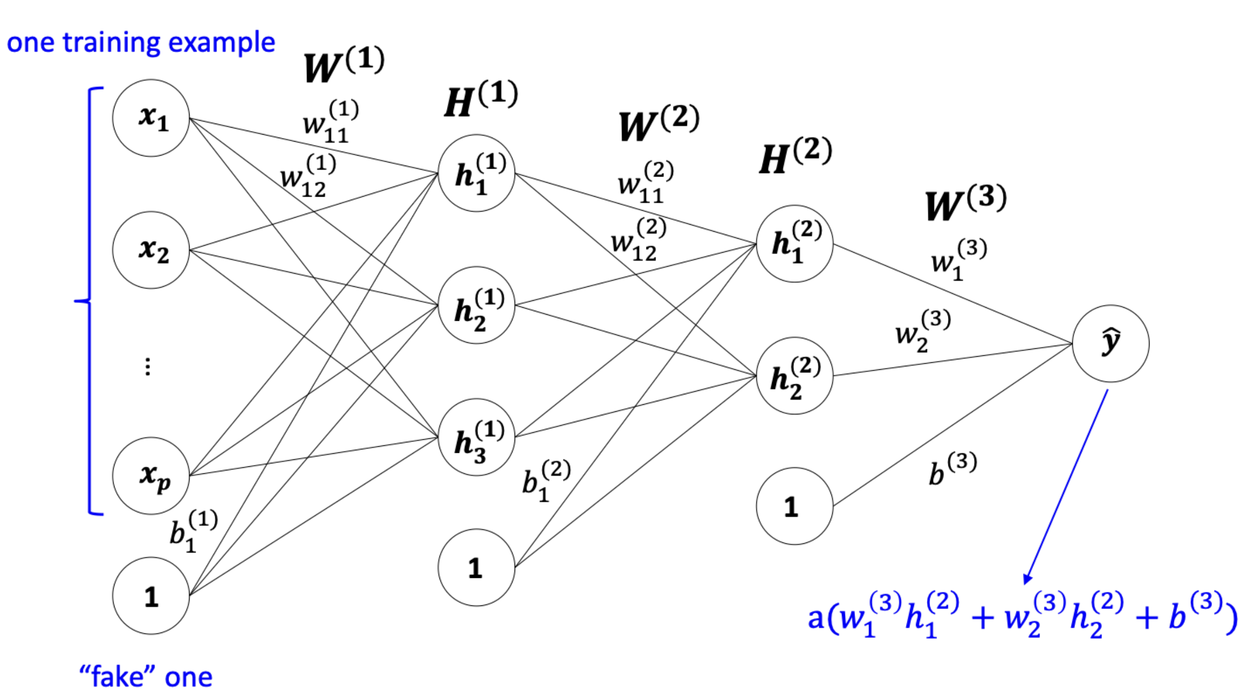
Neural Networks

- Biological Inspiration for Neural Networks



- Learn from complicated inputs→transform data into lower dimension→Multi-layer networks = "deep learning"(comes from # of hidden layers)
- History of Neural Networks
  - Perception can be interpreted as a simple neural network
  - Misconceptions about the weaknesses of perceptrons contributed to declining funding for NN research
  - Difficulty of training multi-layer NNs contributed to second setback
  - Mid 200's: breakthroughs inNN training contribute to rise of "deep learning"



- $H^{(1)} = a\left(W^{(1)}X + \vec{b}^{(1)}\right)$

p_1 = # of nodes in layer 1

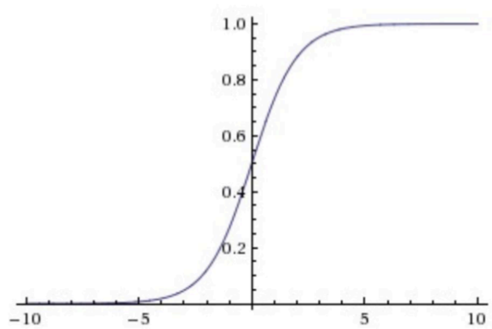
activation function → a; $W^{(1)}$: $p_1 \times p$; X: $p \times n$; $\vec{b}^{(1)}$: $p_1 \times 1$; $W^{(1)}X$: $p_1 \times n$

- $H^{(2)} = a\left(W^{(2)}H^{(1)} + \vec{b}^{(2)}\right)$

- $\hat{y} = a\left(W^{(3)}H^{(2)} + b^{(3)}\right)$

- **Activation Functions**
  - ***Sigmoid function***
    - Input; all real numbers, output:[0, 1]

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

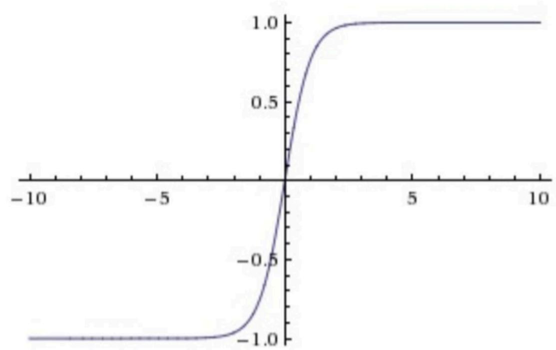


Pros and Cons:

- - (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
- - (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
- - (+) Derivative is easy to compute given function value!

- ***Hyperbolic tangent***
    - Input: all real numbers, output: [-1, 1]

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

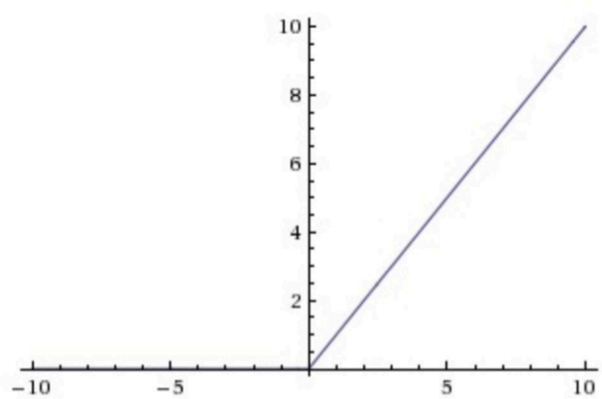


Pros and Cons:

- - (-) Still has a tendency to prematurely kill the gradient
- - (+) Zero-centered so we get a range of gradients
- - (+) Rescaling of sigmoid function so derivative is also not too difficult

- ***Rectified Linear Unit(ReLU)***
    - Return x if x is positive (i.e. threshold at 0)

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



Pros and Cons:

- - (+) Works well in practice (accelerates convergence)
- - (+) Function values are very easy to compute! (no exponentials)
- - (-) Units can have no signal if input becomes too negative throughout gradient descent

Takeaways:

- As the number of parameters grows, anon-convex function often has more and more local minima
- Starting at a "good" point is crucial
- Unsupervised pre-training uses latent structure in the data as a starting point for weight initialization
- After this process, the network is "fine-tuned"
- In practice this has been found to increase accuracy on specific tasks (which could be specifies after feature learning)

Weight initialization

- Initialize the pre-training
- All 0's initialization is bad! Causes nodes to compute the same outputs, so then the weights go through the same updates during gradient descent
- Need asymmetry! => usually use small random values

Mini-Batches

- SGD's flipside is BGD(beach gradient descent) where we compute the gradient with respect to all the data, and then update the weights
- A middle ground uses mini-batches of examples before updating the weights

Scores and softmax

- Output of final fully connected layer is a vector of length K(# of classes)
- Raw scores are transformed into probabilities using the softmax function: (let S be the score for class k)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

- Apply cross-entropy loss to these probabilities

Motivation for moving away from FC architectures

- For a 32x32x3 image (very small!) we have p = 3072 features in the input layer
- For a 200x200x3 image, we would have p = 120,000! *doesn't scale*
- Fully connected networks do not explicitly account for the structure of an image and the correlations/relationships between nearby pixels

Idea: 3D volumes of neurons

- Do not "flatten" image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3
- Each layer is also a 3 dimensional volume
- The output layer is 1x1xC, where C is the number of classes (10 for CIFAR-10)