Advanced Deep Learning

Deep Feedforward Networks

U Kang
Seoul National University

In This Lecture

- Overview of deep feedforward networks
  - Cost function
  - Output units
  - Hidden units
  - Architecture design

Deep FeedForwad Networks

- Deep feedforward networks are the key deep learning models
  - Also called feedforward neural networks or multi-layer perceptrons (MLP)
  - Goal: approximate some function f*
    - E.g., a classifier y = f*(x) maps an input x to a category y
  - A feedforward network defines a mapping $y = f(x;\theta)$ and learns the value of θ

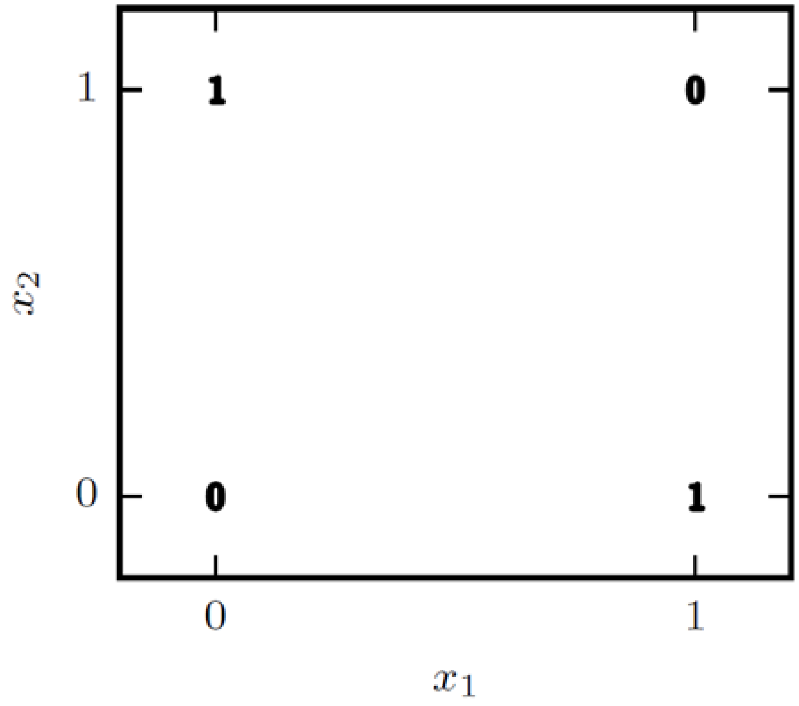
Deep FeedForwad Networks

- Deep feedforward networks are the key deep learning models
  - These models are called feedforward because information flows through the function from x to f to output y
  - These models are called networks because they are typically represented by composing together many different functions
    - E.g., three functions $f^{(1)}, f^{(2)}, f^{(3)}$ connected in a chain to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$

$$f(x) = \underset{\substack{\nearrow \\ \text{output} \\ \text{layer}}}{f^{(3)}}(\underset{\substack{\uparrow \\ \text{second} \\ \text{layer}}}{f^{(2)}}(\underset{\substack{\uparrow \\ \text{first} \\ \text{layer}}}{f^{(1)}}(\underset{\substack{\nwarrow \\ \text{input} \\ \text{layer}}}{x})))$$

Learning XOR

- XOR function: an operation on two binary values
  - XOR outputs 1 only when exactly one of the two values is 1



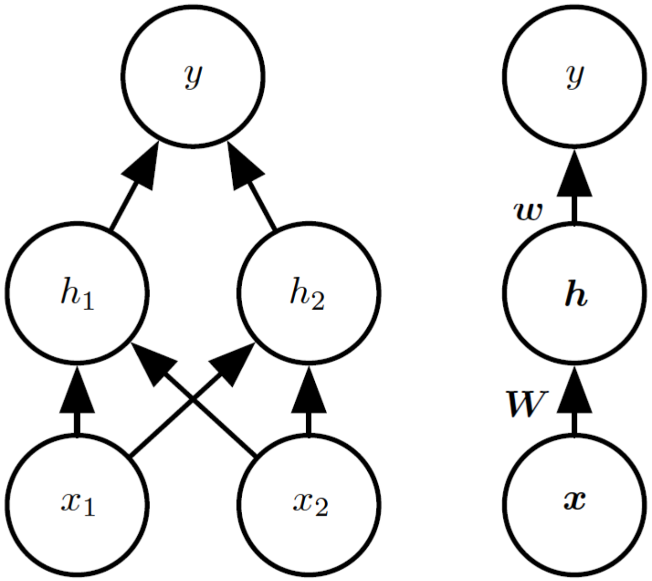
Learning XOR

- Our model provides $y = f(x;\theta)$, and our learning algorithm learns θ such that f outputs the same value as the target XOR function f^*
  - Evaluation will be performed on four points: X={(0,0), (0,1), (1,0), (1,1)}
  - MSE loss function: $J(\theta) = \frac{1}{4}\sum_{x\in X}(f^*(x) - f(x;\theta))^2$

- First model: $f(x;w,b) = x^T w + b$
  - Solving the normal equation, we obtain w = 0 and b = ½
  - That is, it outputs 0.5 everywhere
  - Linear models always fail for XOR!

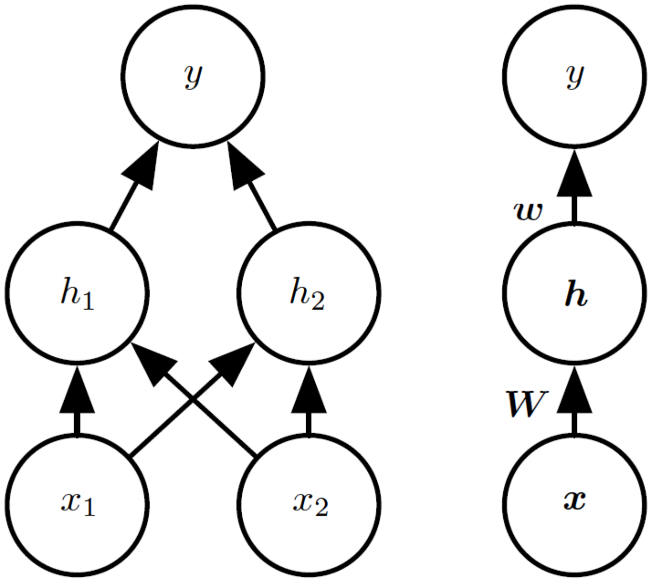
Feedforward Network for XOR

- Feedforward network with one hidden layer with two hidden units
  - The vector of hidden units are computed by $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$
  - The output unit is computed by $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, \boldsymbol{b})$
  - The complete model is $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, \boldsymbol{b}) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$



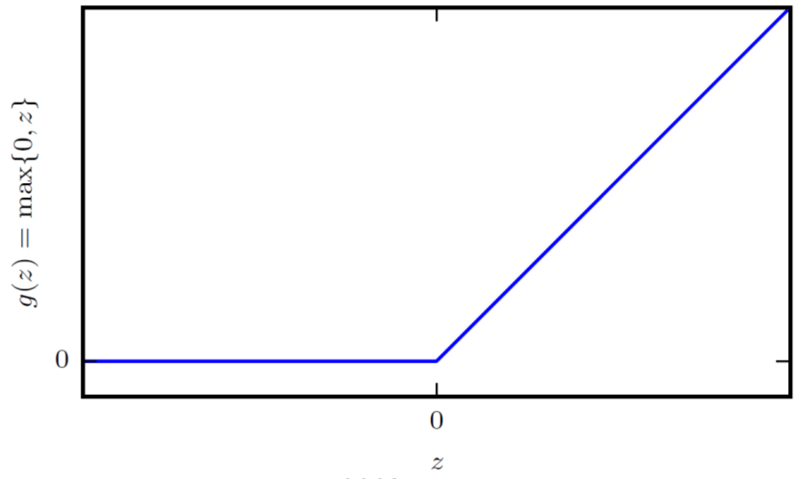
Feedforward Network for XOR

- Assume we use linear regression model for $f^{(2)}$
  - I.e., $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^T\boldsymbol{w}$
- What function should $f^{(1)}$ compute?
  - What if $f^{(1)}$ is linear?



Feedforward Network for XOR

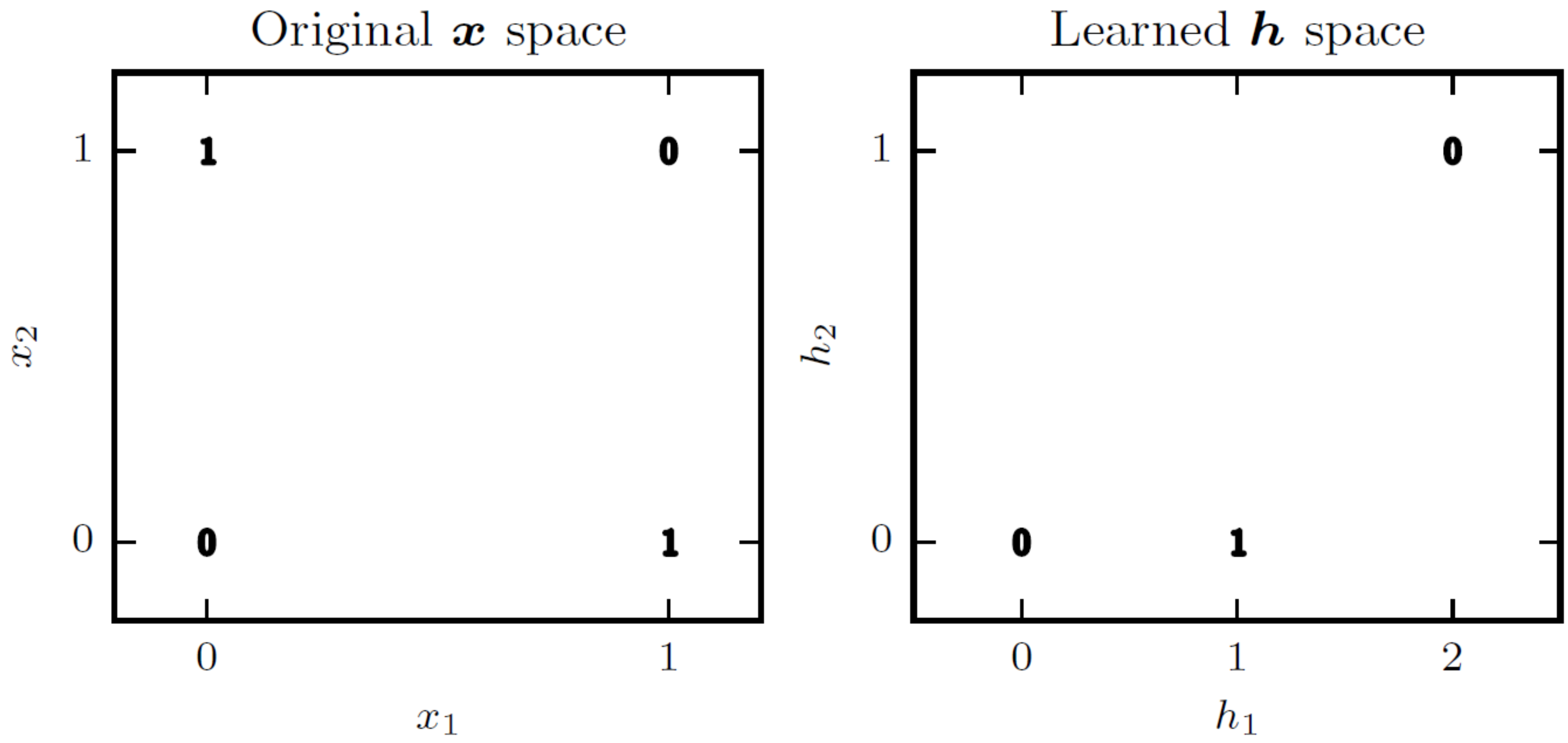
- We need a non-linear function to describe features
- Most neural networks do so using an affine transformation by a fixed, nonlinear function called an activation function
  - $\boldsymbol{h} = g(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c})$
  - g is typically chosen to be a function applied elementwise with $h_i = g(\boldsymbol{x}^T \boldsymbol{W}_{:,i} + c_i)$
  - The default activation function is rectified linear unit or ReLU: g(z) = max{0,z}



Feedforward Network for XOR

- Feedforward network with ReLU
  - $f(\boldsymbol{x};\boldsymbol{W},\boldsymbol{c},\boldsymbol{w},\boldsymbol{b}) = \boldsymbol{w}^T \max\{0,\boldsymbol{W}^T\boldsymbol{x}+\boldsymbol{c}\} + b$
- Solution to the XOR problem
  - $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, c = [0 \ -1]^T, w = [1 \ -2]^T, b = 0$
- From input to output
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, adding $\boldsymbol{c} \rightarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$
  - Applying ReLU $\rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, multiplying by the weight vector $\boldsymbol{w} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

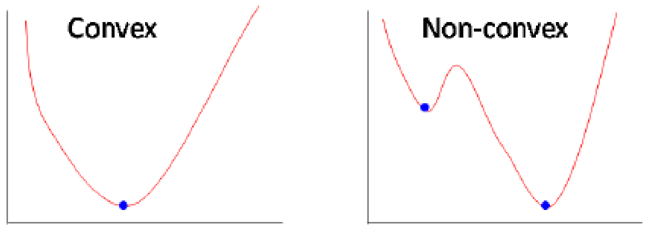
Solving XOR



Original $\boldsymbol{x}$ space
Learned $\boldsymbol{h}$ space
x_1
x_2
h_1
h_2

Gradient-Based Learning

- Neural networks are trained by using iterative, gradient-based optimizers
  - The objective function is non-convex
  - These optimizers find a sufficiently low value, rather than global minimum



- Two important components in gradient-based learning
  - Cost functions
  - Output units

Cost Functions

- In most cases, our model defines $p(y|x;\theta)$ and we use the principle of maximum likelihood
  - I.e., minimize cross-entropy between the training data and the model's prediction
  - $J(\theta) = -E_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$
  - If $p_{model}(y|x) = N(y; f(x;\theta), I)$, then we recover the mean squared error cost: $J(\theta) = \frac{1}{2} E_{x,y\sim\hat{p}_{data}} ||y - f(x;\theta)||^2$
  - The total cost function often is combined with a regularization term
    - E.g., weight decay parameter for linear regression

Maximum Likelihood Estimation

- Consider a set of m examples $X = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn from the true but unknown data generating distribution $p_{data}(x)$
- Let $p_{model}(x; \theta)$ be a parametric family of our model distribution
- The maximum likelihood estimator for θ is defined as
  - $$\begin{aligned}\theta_{ML} &= argmax_\theta\, p_{model}(X; \theta) \\ &= argmax_\theta \prod_{i=1}^{m} p_{model}(x^{(i)}; \theta) \\ &= argmax_\theta \sum_{i=1}^{m} \log p_{model}(x^{(i)}; \theta) \\ &= argmax_\theta\, E_{x\sim\hat{p}_{data}} \log p_{model}(x; \theta)\end{aligned}$$

    where $\hat{p}_{data}$ is the empirical distribution defined by the training data

  - E.g., estimating mean of a Gaussian

Maximum Likelihood Estimation

- The maximum likelihood estimator (MLE) for θ is defined as
  - $\theta_{ML} = argmax_{\theta}\, E_{x\sim\hat{p}_{data}} \log p_{model}(x;\theta)$
- MLE is equivalent to minimizing the dissimilarity between the empirical distribution $\hat{p}_{data}$ and the model distribution in terms of KL divergence
  - $D_{KL}(\hat{p}_{data}||p_{model}) = E_{x\sim\hat{p}_{data}}[\log \hat{p}_{data}(x) - \log p_{model}(x)]$
  - Minimizing the above KL divergence by training the model (finding the best parameters) is equivalent to minimizing $-E_{x\sim\hat{p}_{data}}[\log p_{model}(x)]$
- Note that minimizing D_{KL}(P||Q) with regard to Q is equivalent to minimizing the cross entropy H(P,Q) with regard to Q
  - Thus, minimizing cross entropy is equal to finding MLE
- Maximum likelihood is an attempt to make the model distribution match the empirical distribution $\hat{p}_{data}$
  - Ideally, we would like to match p_{data}, but we do not really know it

Output Units

- In most cases, the cost function is the cross-entropy between the data distribution and the model distribution: $J(\theta) = -E_{x\sim\hat{p}_{data}} \log p_{model}(y|x)$
- The choice of how to represent the output then determines the form of the cross-entropy function
- We assume the feedforward network provides a set of hidden features defined by $h = f(x;\theta)$
- Our loss function is interpreted as $-\log p(y;h)$
  - h provides the parameters for distribution of y
  - I.e., our learning algorithm learns θ so that $p(y;f(x;\theta))$ is maximized

Linear Units for Gaussian Output Distributions

- The output vector $\boldsymbol{y}$ contains real numbers of any range
- Given features $\boldsymbol{h}$, a layer of linear output units produces a vector $\hat{\boldsymbol{y}} = \boldsymbol{W}^T \boldsymbol{h} + \boldsymbol{b}$
- Linear output layers are often used to produce the mean of a conditional Gaussian distribution
  - $p(\boldsymbol{y}|\boldsymbol{x}) = N(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$
  - Maximizing the log-likelihood is equivalent to minimizing the mean squared error

Sigmoid Units for Bernoulli Output Distributions

- The output value y contains 1 or 0
- Use sigmoid function to output the probability in [0,1]
- Given features $\boldsymbol{h}$, a layer of sigmoid output units produces a number $\hat{y} = \sigma(\boldsymbol{w}^T\boldsymbol{h} + b)$
- The loss function for maximum likelihood learning of a Bernoulli parameterized by a sigmoid is
  - $J(\theta) = -\log P(y|x) = -\log \sigma\big((2y-1)z\big) = \zeta((1-2y)z)$

    where $z = \boldsymbol{w}^T\boldsymbol{h} + b$ and $\zeta(x) = \log(1 + \exp(x))$
  - I.e. our learning algorithm learns parameters to maximize

$$p\big(y; f(x;\theta)\big) = \begin{cases} \sigma\big(\boldsymbol{w}^T\boldsymbol{h} + b\big) & if\ y = 1 \\ 1 - \sigma\big(\boldsymbol{w}^T\boldsymbol{h} + b\big) & if\ y = 0 \end{cases}$$

Fact:

$1 - \sigma(x) = \sigma(-x)$

$\log \sigma(x) = -\zeta(-x)$

Softmax Units for Multinoulli Output Distributions

- Useful to represent a categorical distribution (= a probability distribution over a discrete variable with n possible values)
- The output vector $\boldsymbol{y}$ contains n probabilities
- Softmax is a generalization of sigmoid for n possible values:
  $softmax(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, where $\boldsymbol{z} \in R^n$ and $i \in Z^n \; in \; [0, n-1]$
- Given features $\boldsymbol{h}$, a layer of softmax output units produces a vector $\hat{\boldsymbol{y}} = softmax(\boldsymbol{W}^T\boldsymbol{h} + b)$
- The loss function is $J(\theta) = -\log P(y|x) = -\log softmax(\boldsymbol{z})_y$ where $z = \boldsymbol{W}^T\boldsymbol{h} + b$

Hidden Units

- How to choose the type of hidden unit to use in the hidden layers of the model?
- Active area of research; not many theoretical results
- It is usually impossible to predict in advance which types will work best: the design process consists of trial and error

Non-differentiable Hidden Units

- Some hidden units are not actually differentiable at all input points
  - E.g., ReLU function g(z) = max{0, z} is not differentiable at z = 0
- However, gradient descent still performs well enough
  - Hidden units that are not differentiable are usually non-differentiable at only a small number of points
  - Neural network training algorithms do not usually arrive at a local minimum of the cost function, but merely reduce its errors significantly
  - We don’t expect training to reach a point where gradient is 0; thus it is acceptable for the minima of the cost function to correspond to points with undefined gradients
  - Software implementations of neural network training usually return one of the one-sided derivatives
    - Justification: the argument of g(0) of ReLU may not be true 0 but a very small number rounded to 0

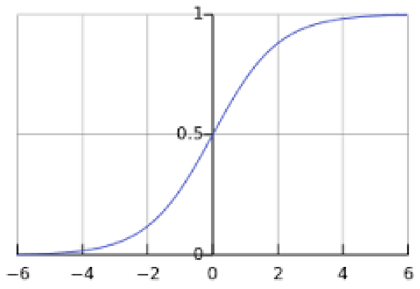
Hidden Units

- Rectified linear units (ReLU)
  - g(z) = max{0,z}
  - Advantage: simple and effective (no vanishing gradient problem)
  - Disadvantage: cannot learn via gradient-based methods on examples for which their activation is 0
  - ReLU are typically used on top of affine transformation: $\boldsymbol{h} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$
- Generalizations of ReLU
  - Generalizations using non-zero slope α_i when $z_i < 0$: $h_i = \max(0, z_i) + \alpha_i \min(0, z_i)$
    - Absolute value rectification: use $\alpha_i = -1$ to obtain g(z)=|z|
    - Leaky ReLU: fixes α_i to a small value like 0.01
    - PReLU (parametric ReLU) treats α_i as a learnable parameter

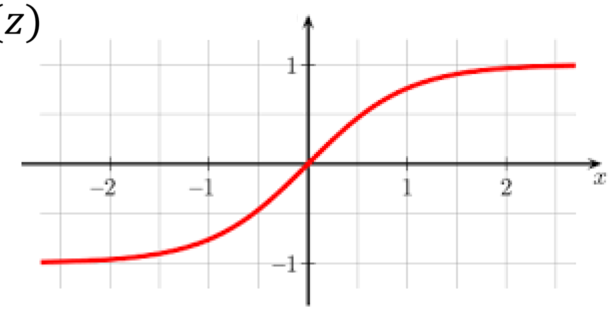
Hidden Units

- Logistic sigmoid and hyperbolic tangent
  - Famous hidden units before the introduction of ReLU
  - Sigmoid function $\sigma(z)$
  - Hyperbolic tangent function $\tanh(z) = 2\sigma(2z) - 1$

$\sigma(z)$



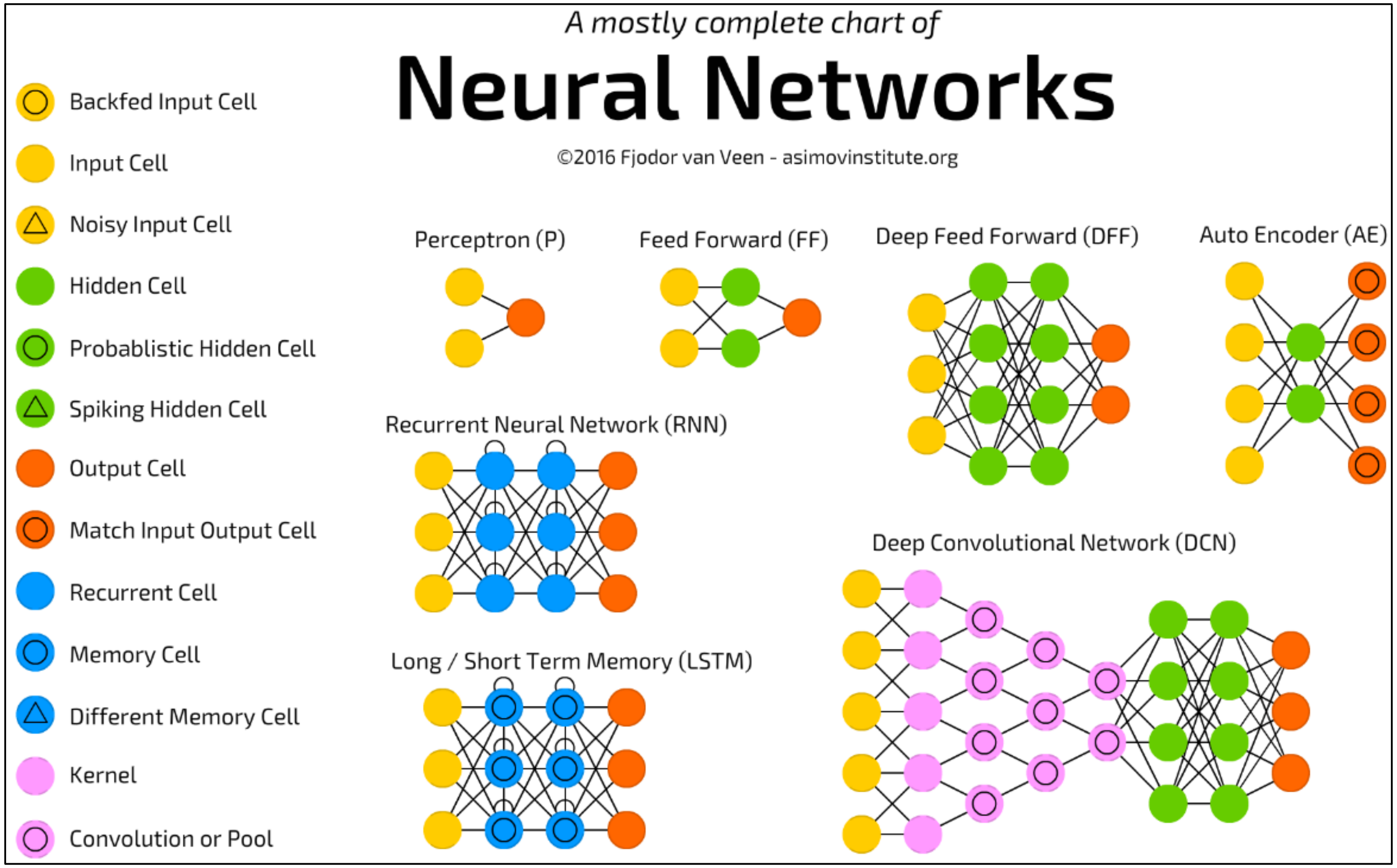
$tanh(z)$



  - Problems: saturate to a high value when z is very positive, and to a low value when z is very negative
    - Gradient is close to 0 when they saturate: only strongly sensitive to their input when z is near 0

Architecture Design

- Architecture refers to the overall structure of the network: how many units it should have and how these units should be connected to each other



Architecture Design

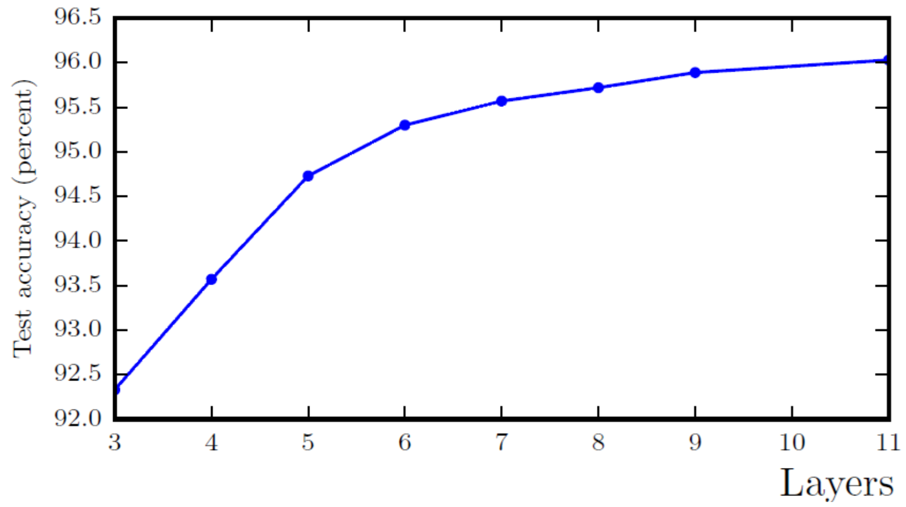
- Most neural networks are organized into groups of units called layers; these layers are typically arranged in a chain structure
  - The first layer is given by $\boldsymbol{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)T}\boldsymbol{x} + \boldsymbol{b}^{(1)})$
  - The second layer is given by $\boldsymbol{h}^{(2)} = g^{(2)}(\boldsymbol{W}^{(2)T}\boldsymbol{x} + \boldsymbol{b}^{(2)})$
  - In these chain-based architectures, the main architectural considerations are to choose the depth of the network and the width of each layer
  - A network with even one hidden layer is sufficient to fit the training set
  - Deeper networks often use far fewer units per layer and far fewer parameters, and often generalize well, but also often harder to optimize
  - The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error

Architecture Design

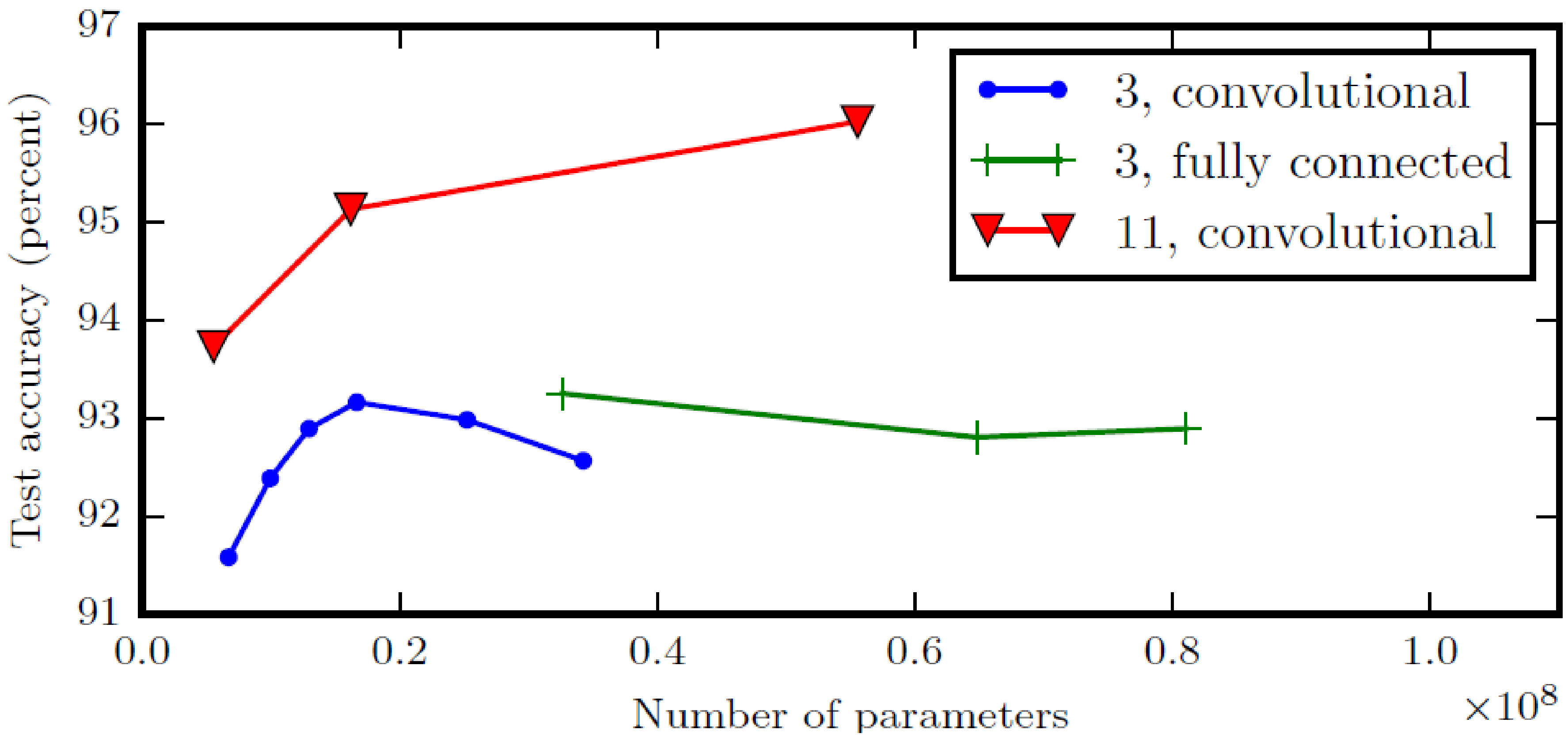
- Universal approximation theorem (Hornik et al. 1989, Cybenko 1989)
  - A feedforward network with a linear output layer and at least one hidden layer with any “squashing” activation function (e.g. logistic sigmoid) can approximate any continuous function with any desired non-zero amount of error
  - This means that regardless of what function we are trying to learn, a large MLP will be able to **represent** this function
  - However, we are not guaranteed that the training algorithm will be able to **learn** that function
    - The optimization algorithm may not be able to find the parameters that correspond to the desired function
    - The training algorithm might choose the wrong function due to overfitting

Exponential Advantage of Depth

- Piecewise linear networks (which can be obtained from rectifier nonlinearities) can represent functions with a number of regions that is exponential in the depth of the network
- Using deeper models can reduce the number of units required to represent the desired function, and can reduce the generalization error
- Empirical results for transcribing multi-digit numbers from photographs of addresses



Shallow Models Overfit More



What You Need to Know

- Deep feedforward networks: enable non-linear mapping inputs to outputs
  - Cost function: cross-entropy
  - Output units: linear, sigmoid, softmax
  - Hidden units: ReLU and its variants
  - Architecture design: deep architecture is preferred despite the universal approximation theorem

Questions?