P9120 - Statistical Learning and Data Mining

Lecture 6 - Deep Learning

Min Qian

Department of Biostatistics, Columbia University

October 10th, 2024

Outline

NEWS | 08 October 2024

Physics Nobel scooped by machine-learning pioneers

John Hopfield and Geoffrey Hinton pioneered computational methods that enabled the development of neural networks.

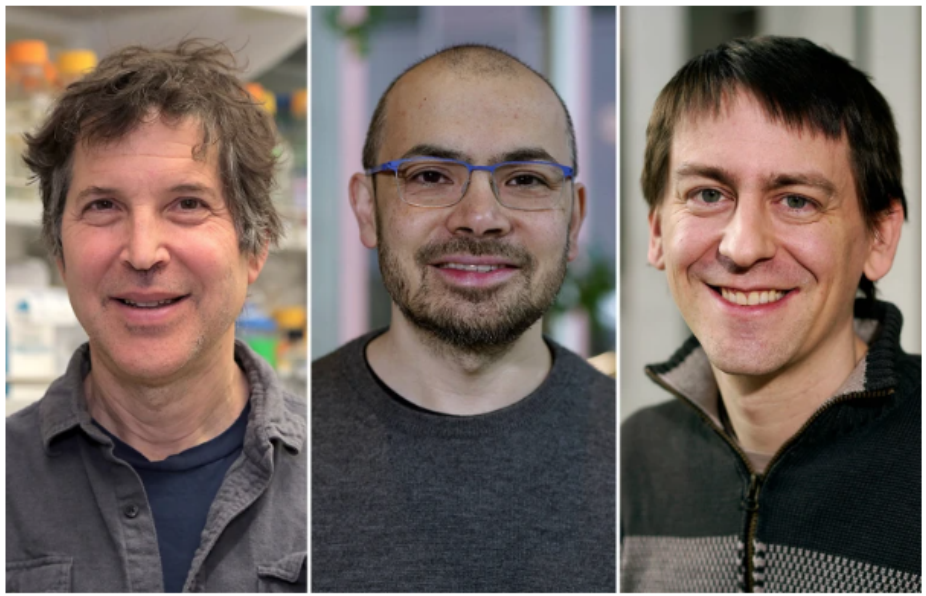
The winners were announced by the Royal Swedish Academy of Sciences in Stockholm. Credit: Jonathan Nackstrand/AFP via Getty

Two researchers who developed tools for understanding the neural networks that underpin today's boom in artificial intelligence (AI) have won the 2024 Nobel Prize in Physics.

NEWS | 09 October 2024

Chemistry Nobel goes to developers of AlphaFold AI that predicts protein structures

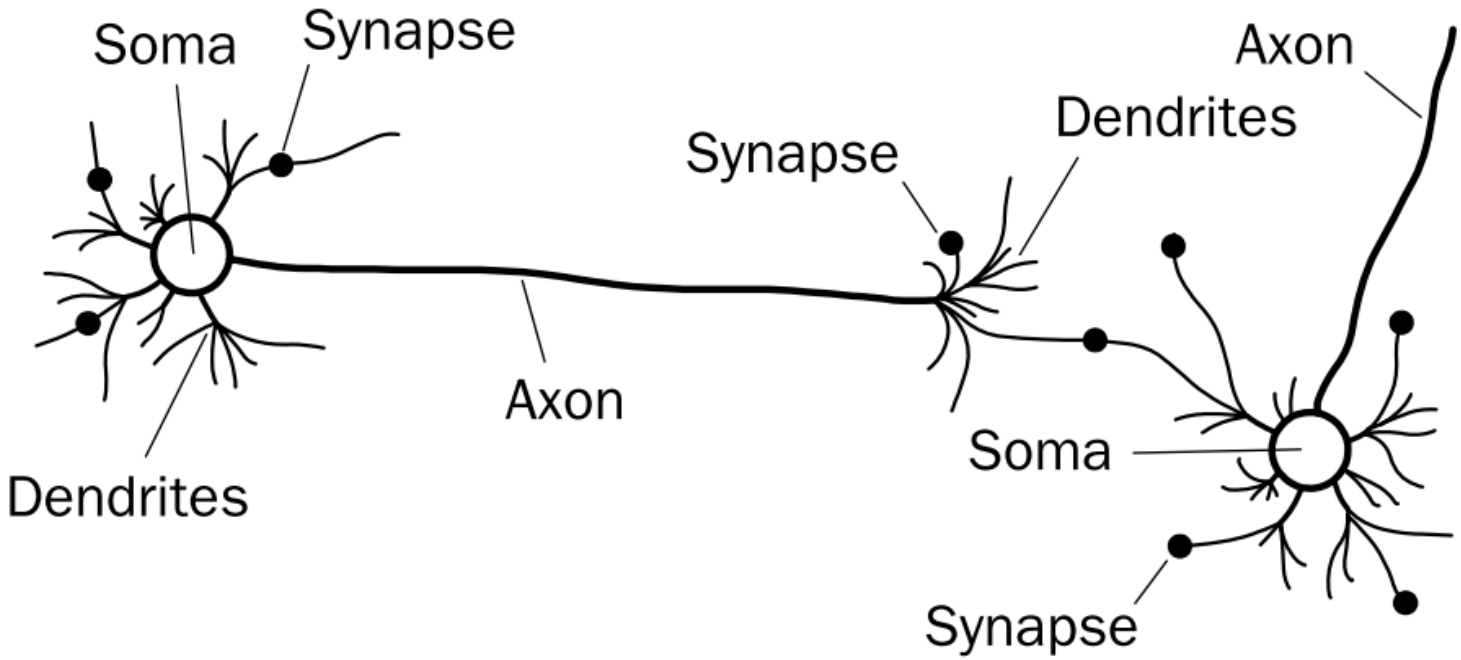
This year's prize celebrates computational tools that have transformed biology and have the potential to revolutionize drug discovery.



David Baker, Demis Hassabis and John Jumper (left to right) won the chemistry Nobel for developing computational tools that can predict and design protein structures. Credit: BBVA Foundation

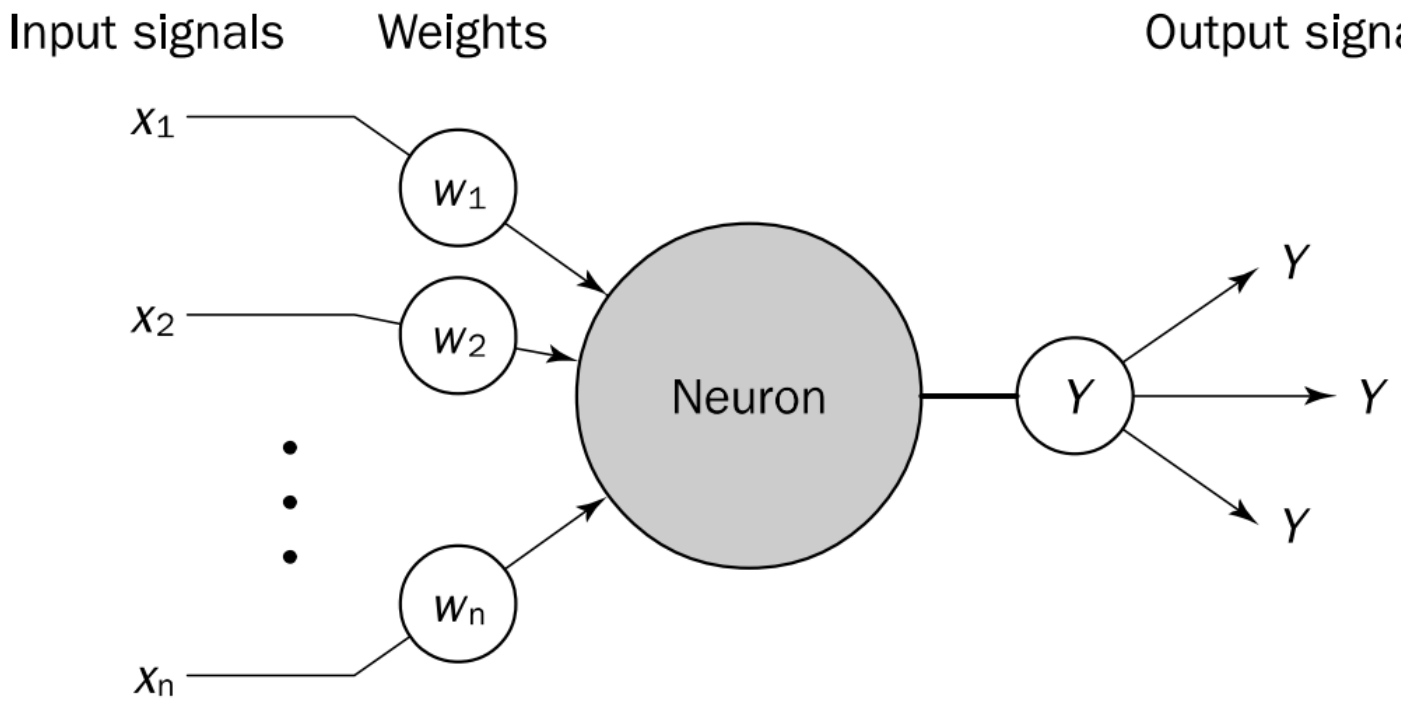
For the first time – and probably not the last – a scientific breakthrough enabled by artificial intelligence (AI) has been recognized with a Nobel prize. The 2024 chemistry Nobel was awarded to John Jumper and Demis Hassabis at Google DeepMind in London, for developing a game-changing AI tool for predicting protein structures called AlphaFold, and David Baker, at the University of Washington in Seattle, for his work on computational protein design, which

Biological Neural Network



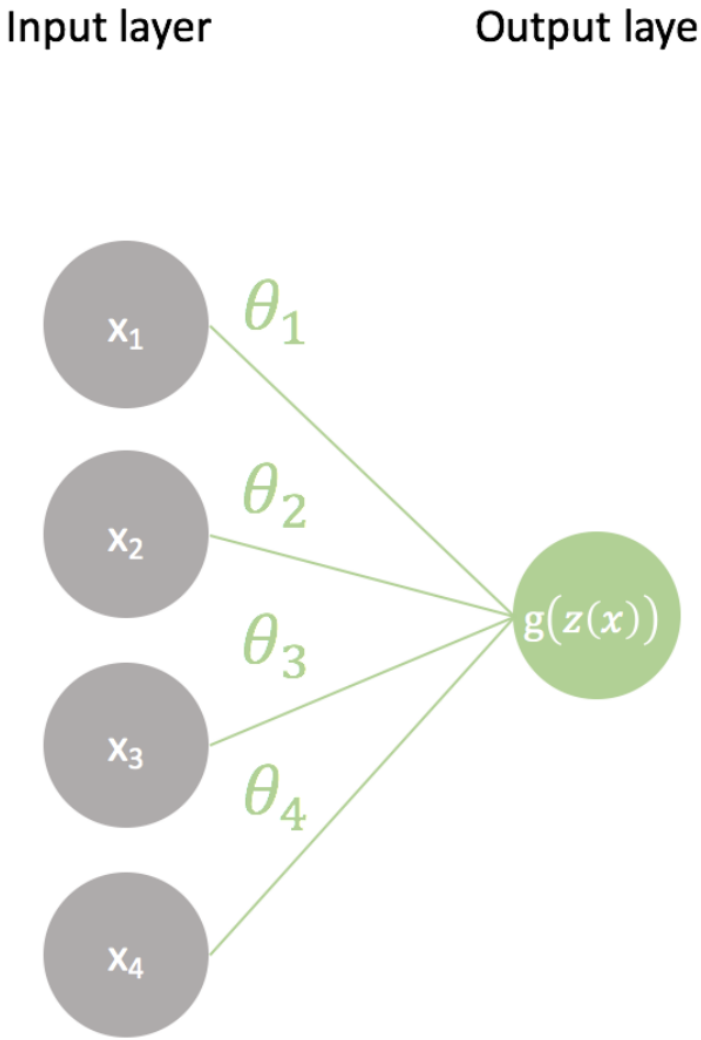
- Dendrites: receive signals
- Cell body (soma): controls the cell's functioning.
- Axon: transmits messages from the neuron to Dendrites of adjacent neurons through synapses (connection).

Artificial Neural Network



Biological Neural Network	Artificial Neural Network
Cell body (soma)	neuron (activation function)
Dendrite	Input
Axon	Output
synapses	Weight

Linear Model as a Neural Network



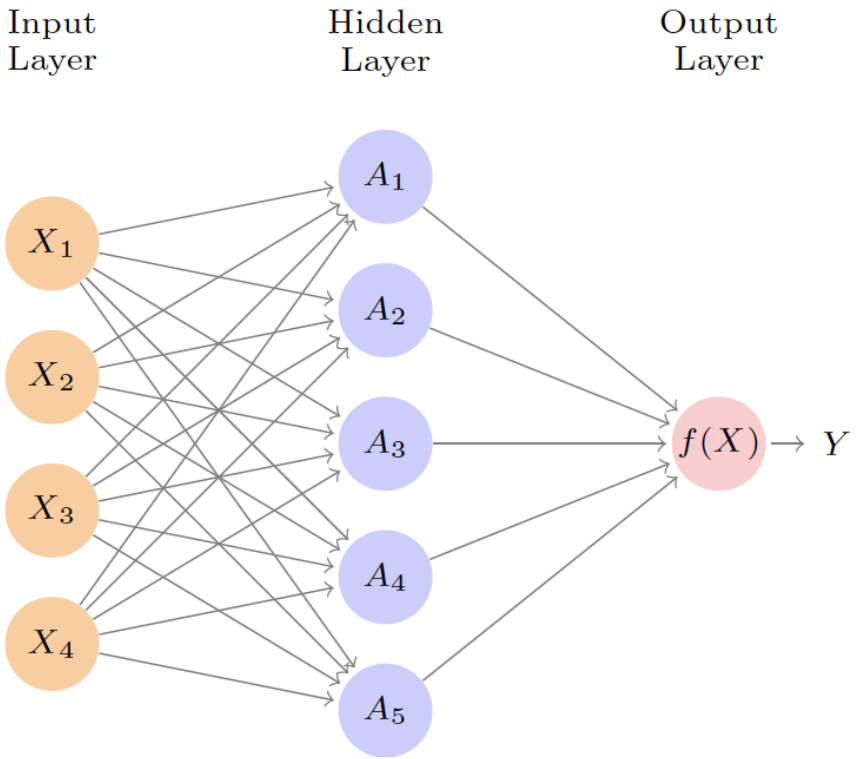
- $z(\mathbf{X}) = \theta_0 + \sum_{j=1}^{p} \theta_j X_j$
- Linear regression:

$$g(z) = z$$
$$E(Y|\mathbf{X}) = g(z(\mathbf{X}))$$

- Logistic regression:

$$g(z) = \frac{\exp(z)}{1 + \exp(z)}$$
$$P(Y = 1|\mathbf{X}) = g(z(\mathbf{X})).$$

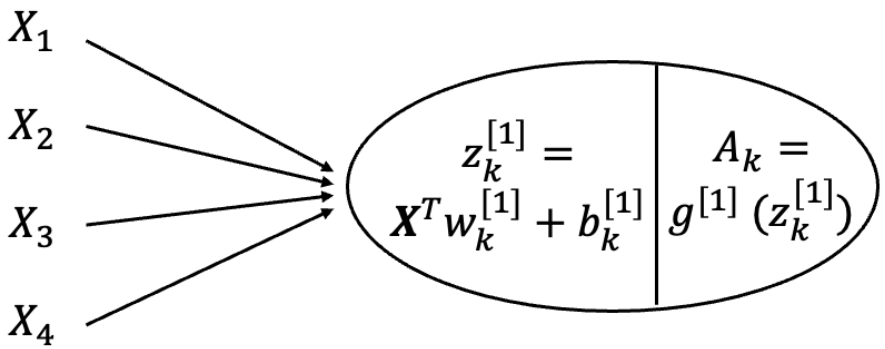
2-Layer Neural Network (Single Hidden Layer)



Input Layer $\mathbf{X} \in \mathbb{R}^p$

$\rightarrow$ Hidden Layer $\mathbf{A} \in \mathbb{R}^K$

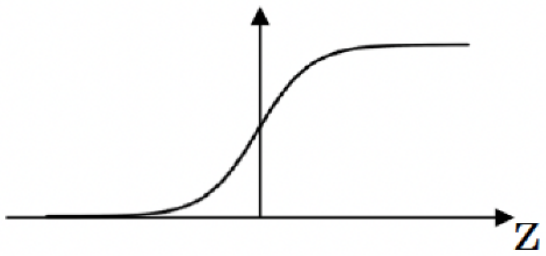
For each A_k,



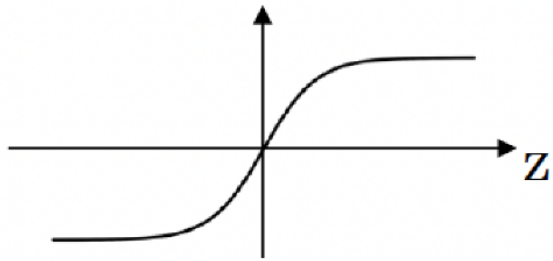
- $g^{[1]}(\cdot)$ is the hidden layer (layer-1) activation function.
- $\{(w_k^{[1]}, b_k^{[1]}) : k = 1, \ldots, K\}$ are hidden layer parameters (weights).

Commonly Used Hidden Layer Activation Functions

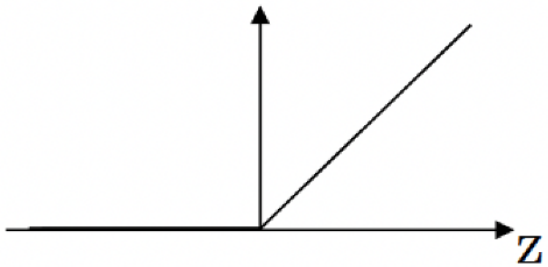
Sigmoid: $g(z) = \frac{1}{1+\exp(-z)}$



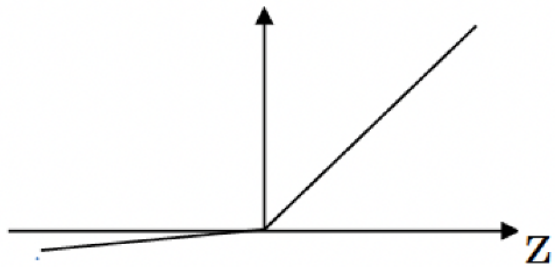
tanh: $g(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$



ReLU: $g(z) = \max(0, z)$



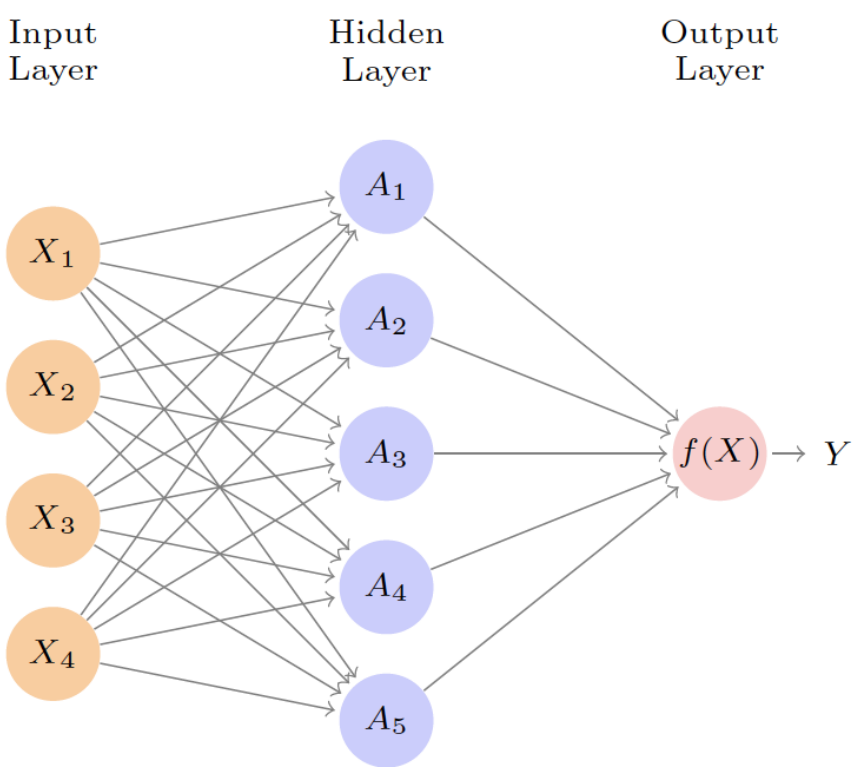
Leaky ReLU: $g(z) = \max(0.01z, z)$



Others: Threshold function $g(z) = 1_{z>0}$.

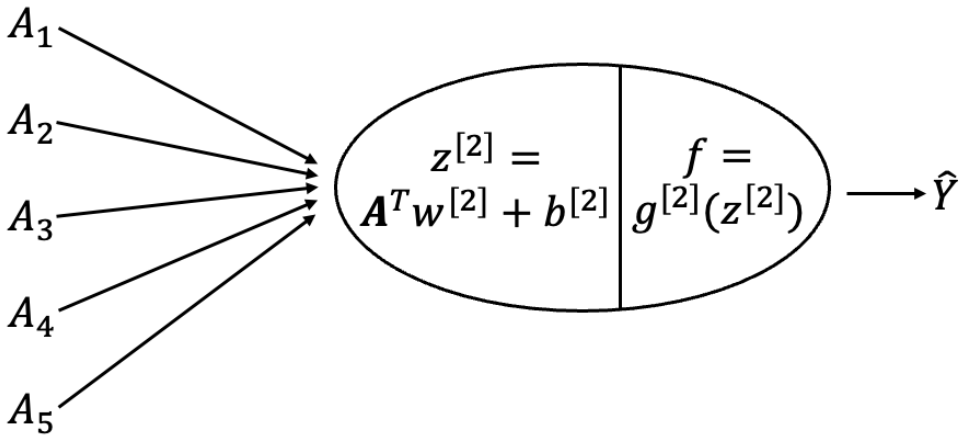
By Andrew Ng, Sigmoid $\prec$ tanh $\prec$ ReLU $\precsim$ Leaky ReLU

2-Layer Neural Network (Single Hidden Layer)



Hidden Layer $\mathbf{A} \in \mathbb{R}^K$

$\rightarrow$ Output Layer



- $w^{[2]}$ and $b^{[2]}$'s are layer-2 parameters (weights).

$g^{[2]}(\cdot)$ is the output layer activation function:

- Continuous outcome: $g^{[2]}(z^{[2]}) = z^{[2]} = \widehat{E}(Y|\mathbf{X}) = \widehat{Y}$
- Binary outcome $\{0, 1\}$: $g^{[2]}(z^{[2]}) = \frac{1}{1+\exp(-z^{[2]})} = \hat{P}(Y = 1|\mathbf{X})$.

What Can NNs Represent?

- Binary outcome $Y \in \{0, 1\}$ and binary predictors $\mathbf{X} \in \{0, 1\}^p$.

Claim: Any Boolean function $\{0, 1\}^p \to \{0, 1\}$ can be represented by a 2-layer (i.e., one hidden layer) NN with threshold activation function.

- There are 2^p possible input patterns.
- One hidden unit for every input pattern that has output $Y = 1$
- Arrange weights from input variables to the hidden unit so that the hidden unit "responds" just to that pattern. e.g., set weight= 1 If the input variable takes value 1 , else weight= -1; set the intercept = 1− # of variables taking value 1 in the input pattern.
- All hidden units to output weights = 1 and intercept = -0.5.

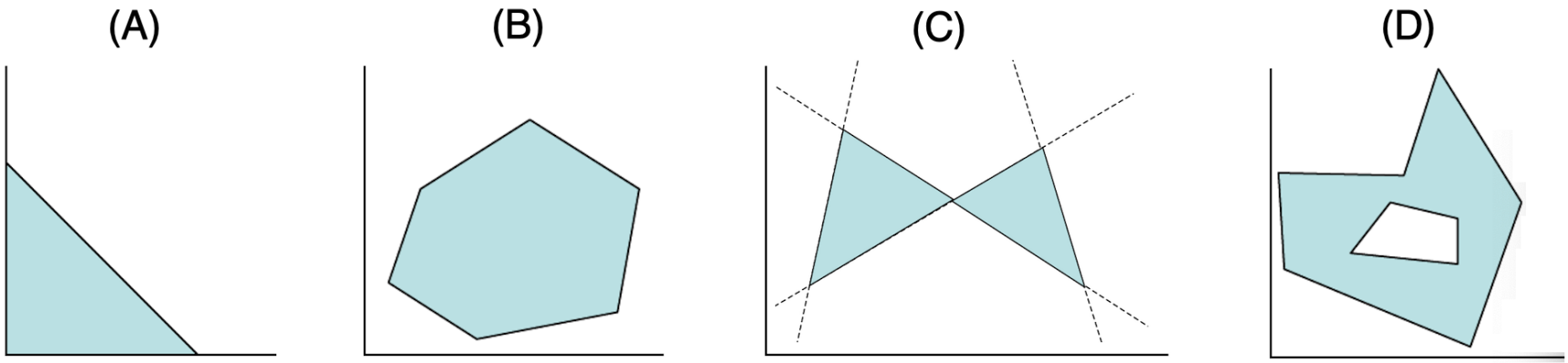
Example

X_1	X_2	X_3	Y
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

The number of hidden units grows exponentially with p.

What Can NNs Represent?

- Binary outcome, continuous predictors, threshold activation.



- (A): linear model (no hidden layer)
- (B): 2-layer NN (one hidden layer) with 6 hidden units
- (C) and (D): 3-layer NN (two hidden layers).

Universal Approximation Theorems

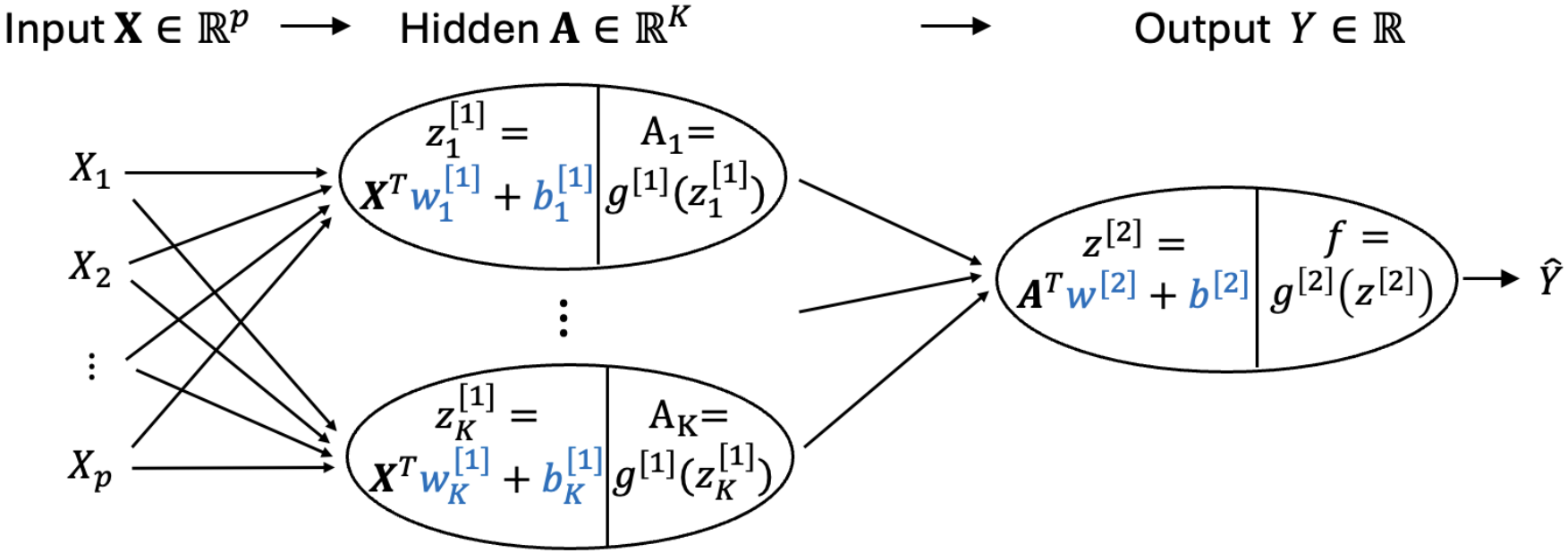
Width: number of units in a hidden layer; Depth: number of layers

- In 1980-1990s, 2-layer with arbitrary width
  With an appropriate choice of the activation function ("squashing" functions, ReLU, etc.), the NN model can approximate any continuous function $f(X)$ arbitrarily well.

- Recently, bounded width with arbitrary depth
  - Most activation functions can approximate continuous function $f(X)$ arbitrarily well.
  - ReLU and Leaky ReLU can approximate any p-integrable function.

- Bounded width and depth (Guliyev and Ismailov, 2018):
  For certain activation functions depth-2 width-2 NN are universal approximators for univariate functions and depth-3 width-$(2d+2)$ NN are universal approximators for d-variable functions.

Fitting 2-Layer NN for Continuous Outcome Y

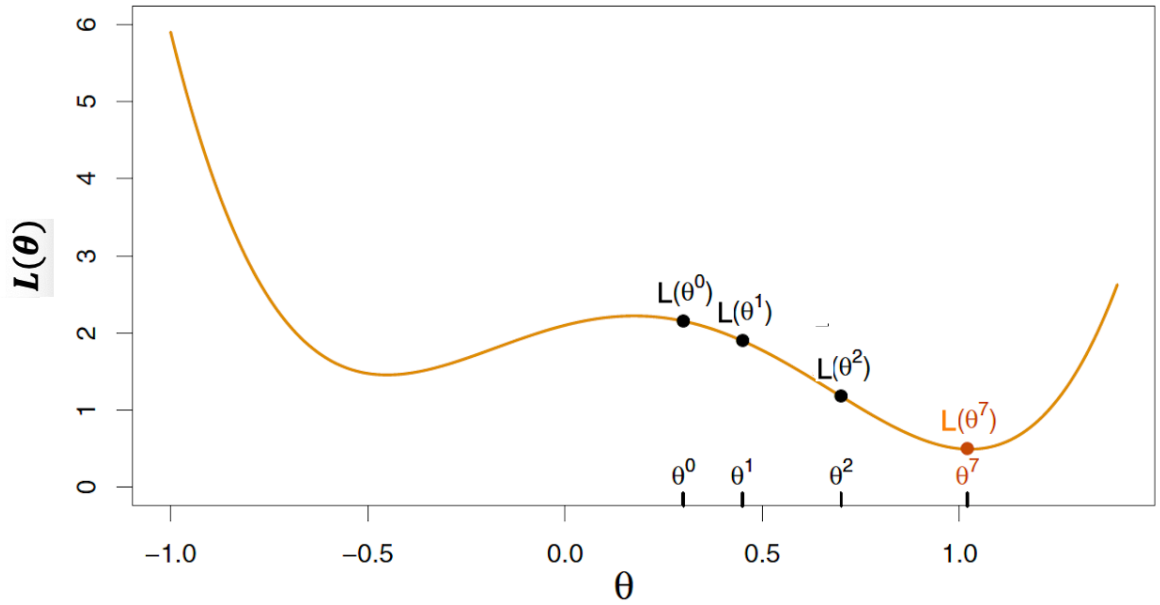


- For continuous outcome, $\widehat{Y} = g^{[2]}(z^{[2]}) = z^{[2]}$.
- Estimate $\theta = \{(b^{[2]}, w^{[2]}, b_k^{[1]}, w_k^{[1]}) : k = 1, \ldots, K\}$ by minimizing

$$L(\theta) = \sum_{i=1}^{n} l_i(\theta) = \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2$$

Gradient Descent for Neural Networks

Gradient Descent: $\theta^{t+1} \leftarrow \theta^t - \gamma \nabla L(\theta^t)$, where γ is the learning rate

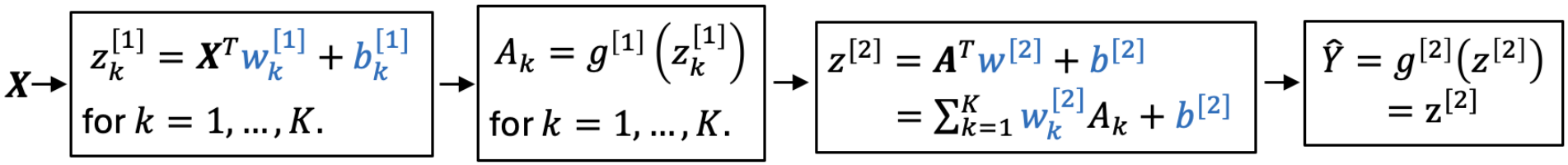


$$\nabla L(\theta) = \left(\frac{\partial L(\theta)}{\partial b^{[2]}}, \frac{\partial L(\theta)}{\partial w^{[2]}}, \frac{\partial L(\theta)}{\partial b_k^{[1]}}, \frac{\partial L(\theta)}{\partial w_k^{[1]}}, k = 1, \ldots, K \right)$$

$$= \sum_{i=1}^{n} \left(\frac{\partial l_i(\theta)}{\partial b^{[2]}}, \frac{\partial l_i(\theta)}{\partial w^{[2]}}, \frac{\partial l_i(\theta)}{\partial b_k^{[1]}}, \frac{\partial l_i(\theta)}{\partial w_k^{[1]}}, k = 1, \ldots, K \right)$$

Fitting Neural Networks: Gradient of $l(\theta) = (Y - \hat{Y})^2$

Gradient Descent: $\theta^{t+1} \leftarrow \theta^t - \gamma \nabla L(\theta^t)$



$$\frac{\partial l(\theta)}{\partial b^{[2]}} = \frac{\partial l}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial b^{[2]}} = -2(Y - \hat{Y}),$$

$$\frac{\partial l(\theta)}{\partial w^{[2]}} = \frac{\partial l}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}} = -2(Y - \hat{Y})\boldsymbol{A}$$

$$\frac{\partial l(\theta)}{\partial b_k^{[1]}} = \frac{\partial l}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A_k} \cdot \frac{\partial A_k}{\partial z_k^{[1]}} \cdot \frac{\partial z_k^{[1]}}{\partial b_k^{[1]}} = -2(Y - \hat{Y}) w_k^{[2]} g^{[1]\prime}\left(z_k^{[1]}\right)$$

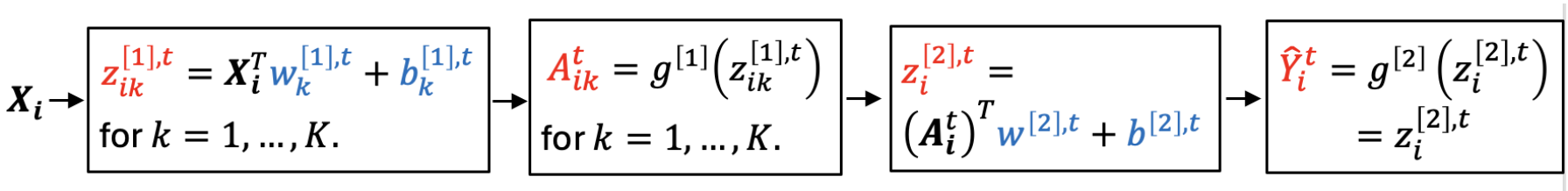
$$\frac{\partial l(\theta)}{\partial w_k^{[1]}} = \frac{\partial l}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A_k} \cdot \frac{\partial A_k}{\partial z_k^{[1]}} \cdot \frac{\partial z_k^{[1]}}{\partial w_k^{[1]}} = -2(Y - \hat{Y}) w_k^{[2]} g^{[1]\prime}\left(z_k^{[1]}\right)\boldsymbol{X}$$

Batch Mode Backpropagation

Initialize $\theta^0 = \{(b^{[2],0}, w^{[2],0}, b_k^{[1],0}, w_k^{[1],0}) : k = 1, \ldots, K\}$

For $t = 0, 1, 2 \ldots$, do the following until convergence

1. Forward step: for each data point $(\boldsymbol{X}_i, Y_i), i = 1, \ldots, n$, compute $z_{ik}^{[1],t}, A_{ik}^t, z_i^{[2],t}$ and $\widehat{Y}_i^t$:



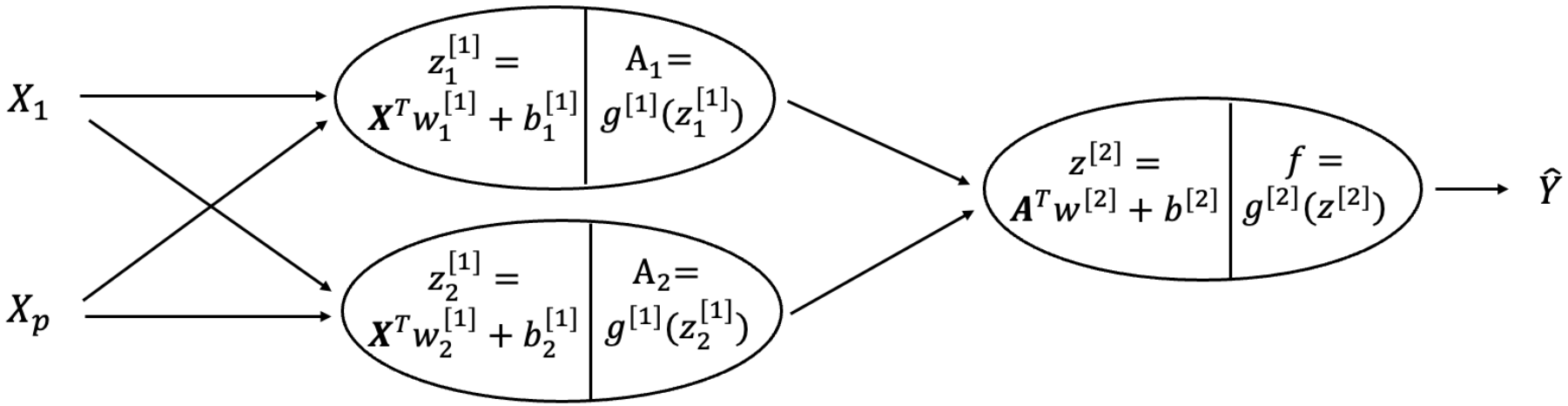
2. Do a backward pass through the network computing the gradient:

$$\nabla L(\theta^t) = \sum_{i=1}^{n} \Big(-2(Y_i - \widehat{Y}_i^t), \quad -2(Y_i - \widehat{Y}_i^t)\boldsymbol{A}_i^t, \\ -2(Y_i - \widehat{Y}_i^t) w_k^{[2],t} g^{[1]\prime}(z_{ik}^{[1],t}), -2(Y_i - \widehat{Y}_i^t) w_k^{[2],t} g^{[1]\prime}(z_{ik}^{[1],t}) \boldsymbol{X}_i \Big).$$

3. Update the parameters $\theta^{t+1} \leftarrow \theta^t - \gamma \nabla L(\theta^t)$.

Random Initialization of Weights with Small Values

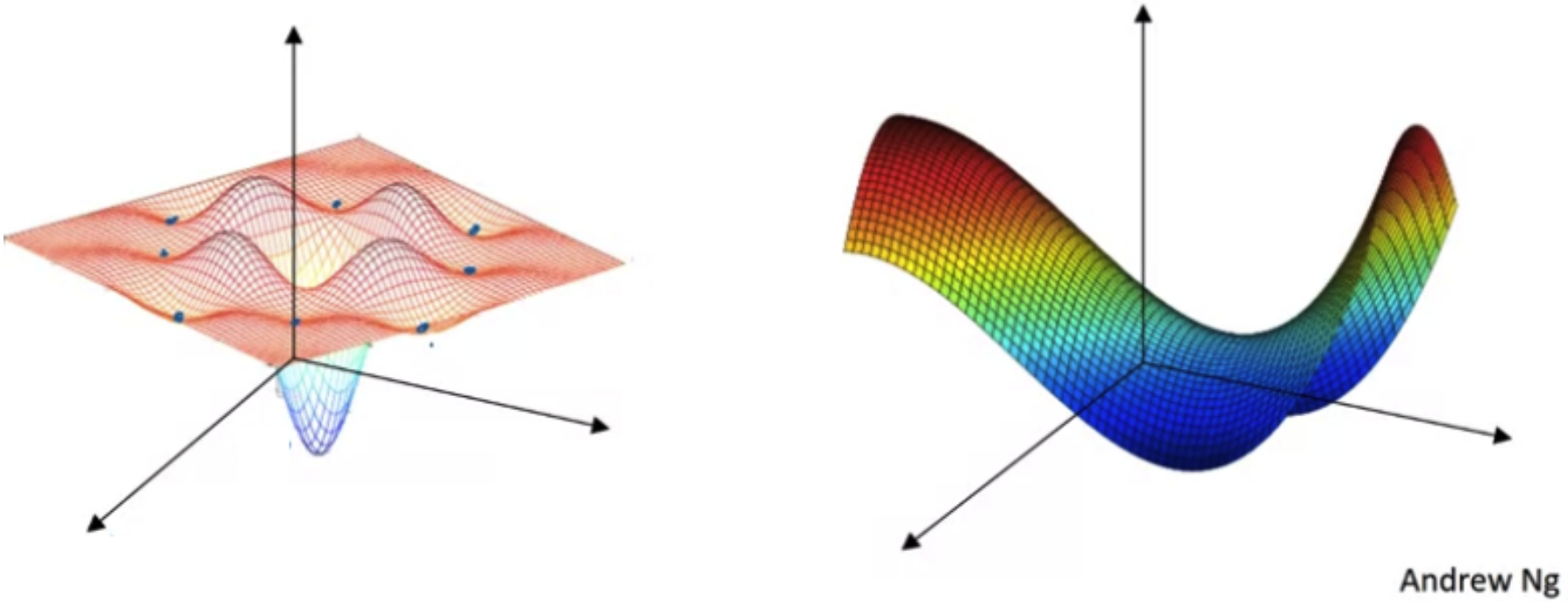
- Cannot initialize all the weights to a constant value
- Vanishing and exploding gradients (especially for very deep NN!!!)



$$\text{Gradient: } \frac{\partial l(\theta)}{\partial w_1^{[1]}} = \frac{\partial l(\theta)}{\partial f} \cdot \frac{\partial f}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial A_1} \cdot \frac{\partial A_1}{\partial z_1^{[1]}} \cdot \frac{\partial z_1^{[1]}}{\partial w_1^{[1]}}$$

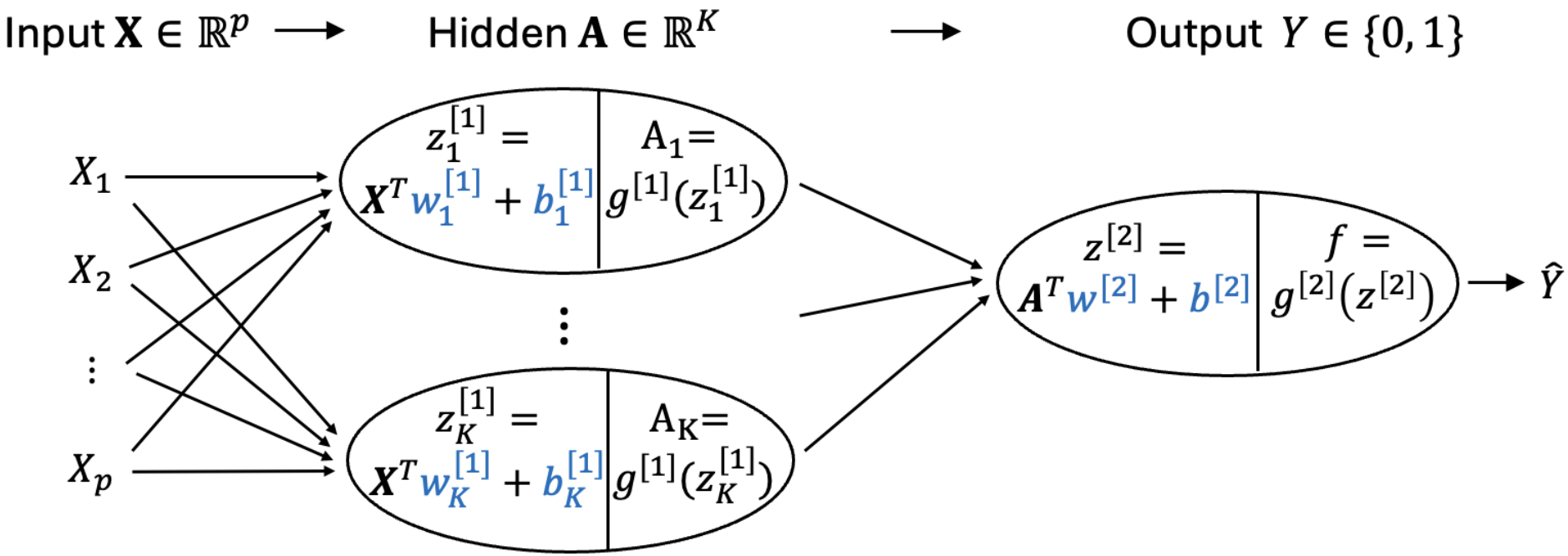
Random initialize e.g., randn(0, 0.1)/100 (often $\text{var}(w) \propto \#\text{ input}^{-1}$).

Local Optima in NN



- Complex NN is unlikely to get stuck in a bad local optima.
- Plateaus can make learning slow.

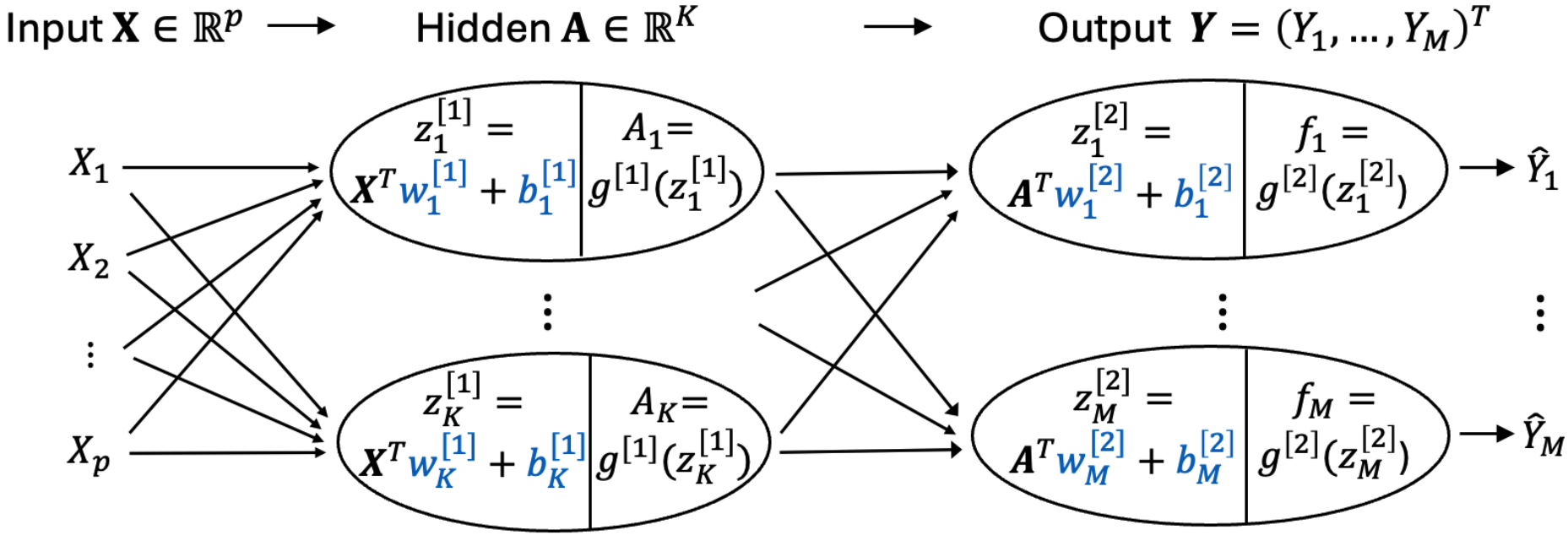
2-Layer NN for Binary Outcome



- $\widehat{P}(Y = 1|X) = f(X) = g^{[2]}(z^{[2]}) = \frac{1}{1+\exp(-z^{[2]})}$
- Loss function:

$$l(\theta) = -\Big[Y \log(f(X)) + (1 - Y) \log(1 - f(X))\Big]$$

NN for Multiple Outcomes



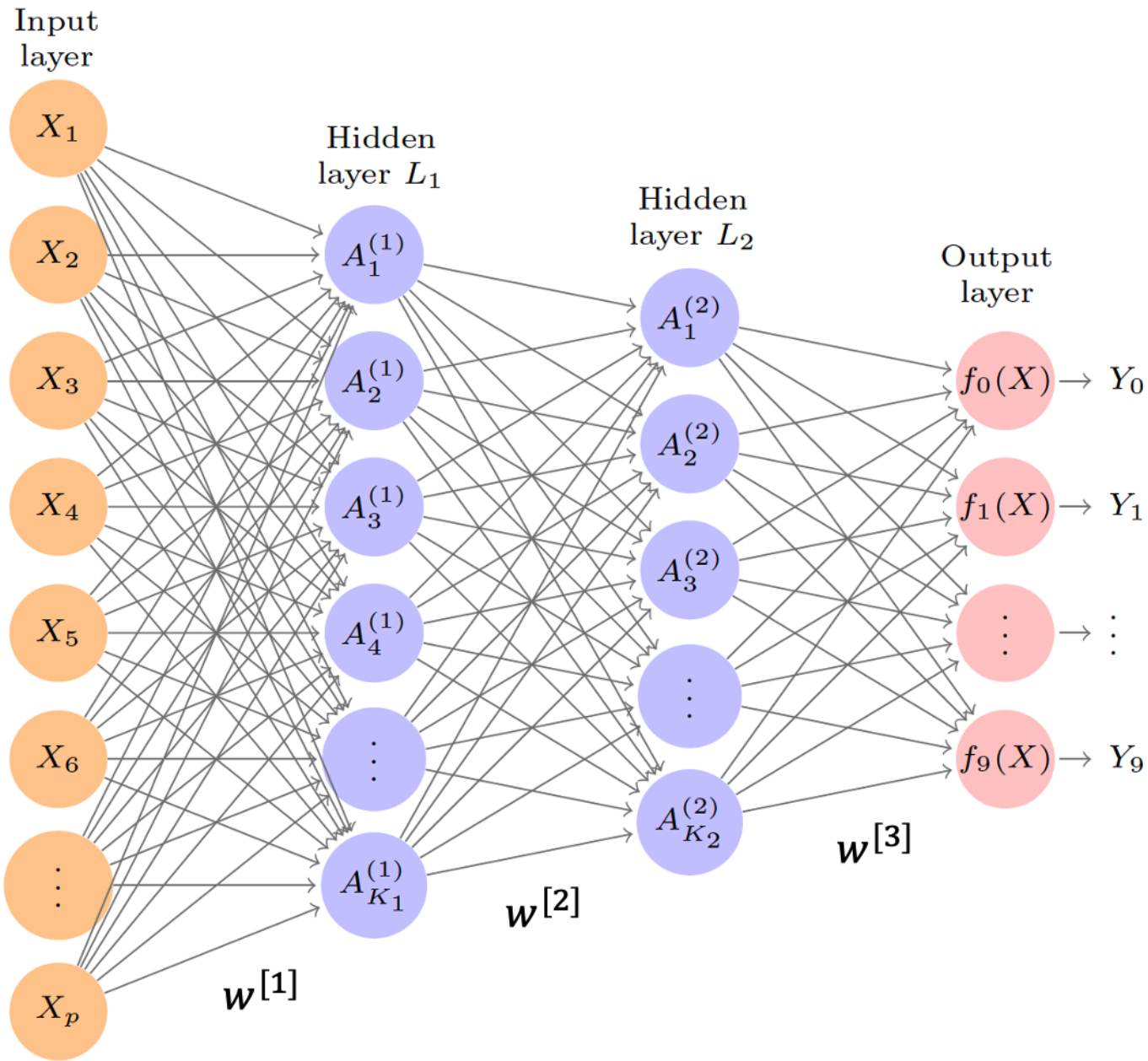
- Continuous outcomes: $\mathbf{Y} = (Y_1, \ldots, Y_M) \in \mathbb{R}^M$

$$\widehat{Y}_m = f_m(\mathbf{X}) = z_m^{[2]}, \text{ Loss: } \sum_{m=1}^{M} (Y_m - \widehat{Y}_m)^2$$

- Multi-class classification: $\mathbf{Y} = (Y_1, \ldots, Y_M), Y_m \in \{0, 1\}$

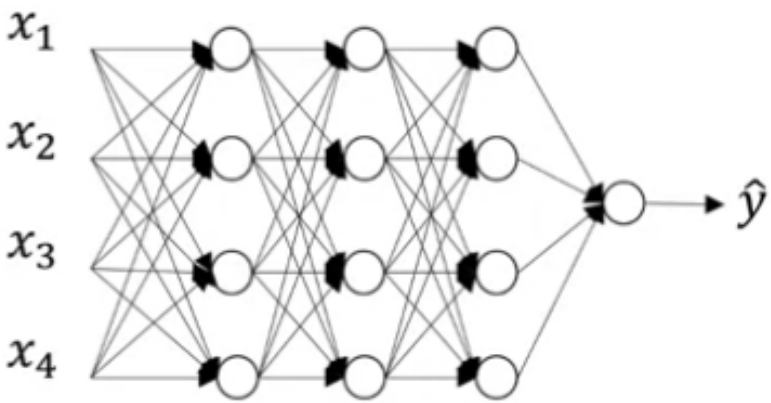
$$\widehat{P}(Y_m = 1|\mathbf{X}) = f_m(\mathbf{X}) = \frac{\exp(z_m^{[2]})}{\sum_{j=1}^{M} \exp(z_j^{[2]})}, \text{ Loss: } -\sum_{m=1}^{M} Y_m \log[f_m(\mathbf{X})]$$

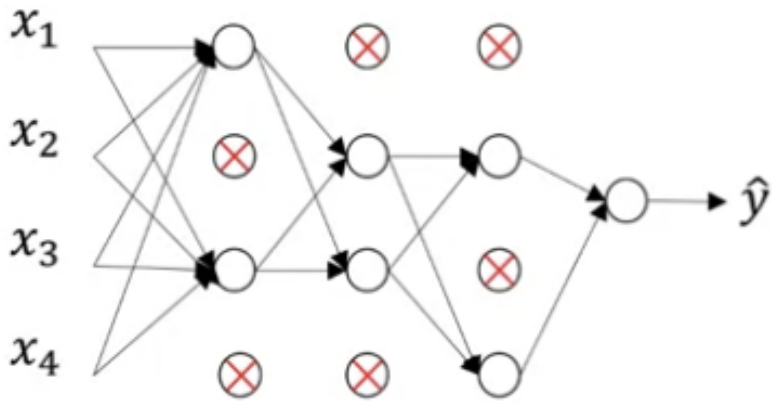
Deeper Neural Networks



Improving Deep NN: Regularization

- l_2-penalty (aka, weight decay, most common method), l_1-penalty
- Early stopping of gradient descent iterations.
- Dropout: randomly drop out some units at each iteration

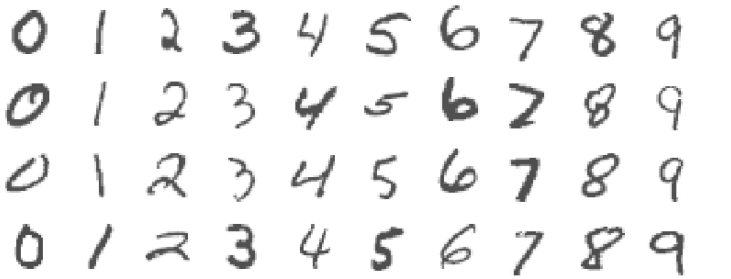


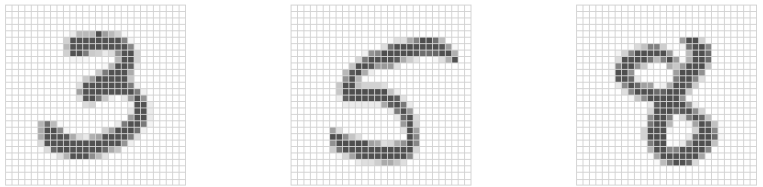


re-scale the values of units in each layer by the keep probability.

Multilayer NN: Handwritten Digits Example

- Input: $p = 28 * 28 = 784$
- Output: $Y \in \{0, 1, \ldots, 9\}$.
- Sample size $n = 60,000$.
- Two hidden layers:
  $$K_1 = 256, K_2 = 128.$$
- Number of parameters:
  $$(p+1) \times K_1 + (K_1+1) \times K_2 + (K_2+1) \times 10 = 235,146.$$





- Results:

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

Variations of Gradient Descent (GD)

- Batch GD: one iteration: $\theta^{t+1} \leftarrow \theta^t - \gamma \sum_{i=1}^{n} \nabla l_i(\theta)$.
- Mini-batch GD:
  Divide data into m mini-batches, each of size n_m (usually $n_m = 2^8$ to 2^9). For $j = 1, \ldots, m$,

$$\text{one iteration:} \quad \theta^{t+1} \leftarrow \theta^t - \gamma \sum_{i \text{ in batch j}} \nabla l_i(\theta).$$

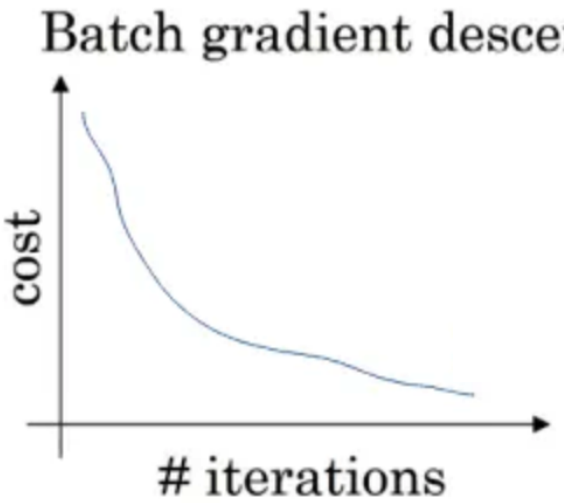
- Stochastic GD:
  Randomly draw (with/without replacement) one data point i,

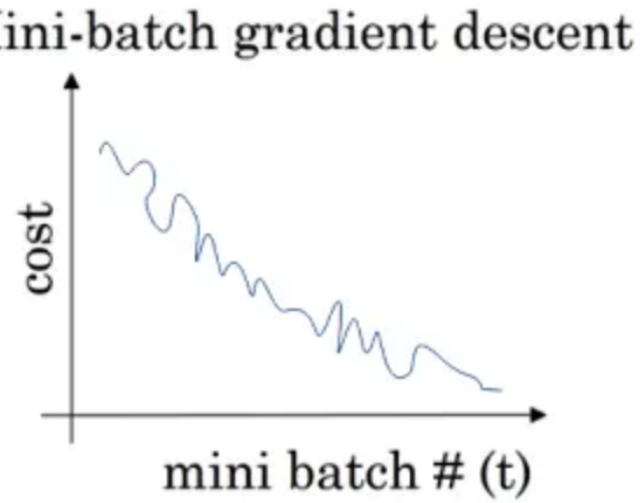
$$\text{one iteration:} \quad \theta^{t+1} \leftarrow \theta^t - \nabla l_i(\theta).$$

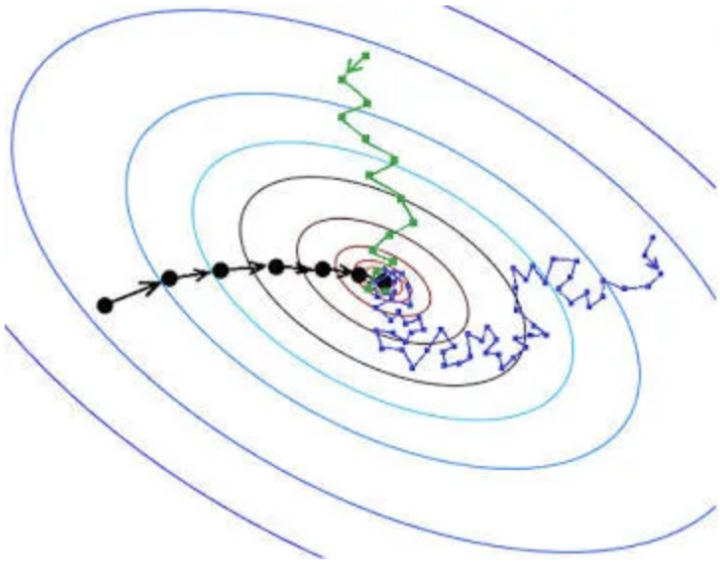
- GD with momentum: replace gradient by some moving average

One epoch is one cycle of GD on all data points in the training set.

Batch vs. Mini-batch vs Stochastic GD







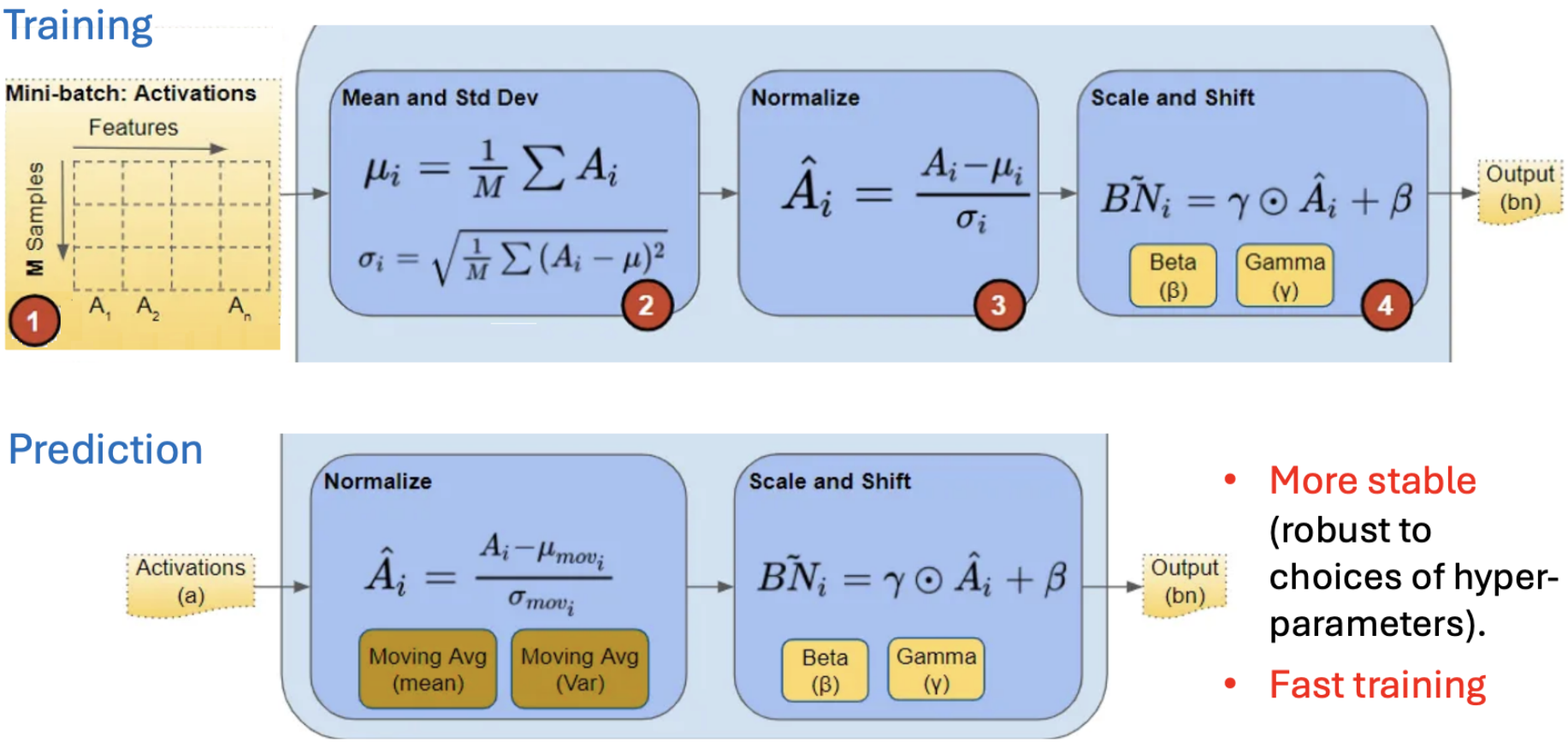
Batch GD
- Slowest
- Perfect gradient

Stochastic GD
- Fastest
- Rough-estimate grad

Mini-batch GD
- Compromise

In deep learning, usually use mini-batch stochastic (without replacement) GD.

Batch Normalization



- More stable (robust to choices of hyper-parameters).
- Fast training

- Batch Norm could be placed before or after activation.
- In practice, replace σ_i by $\sqrt{\sigma_i^2 + \epsilon}$.
- Works with momentum (i.e. use moving avg of mean and SD).

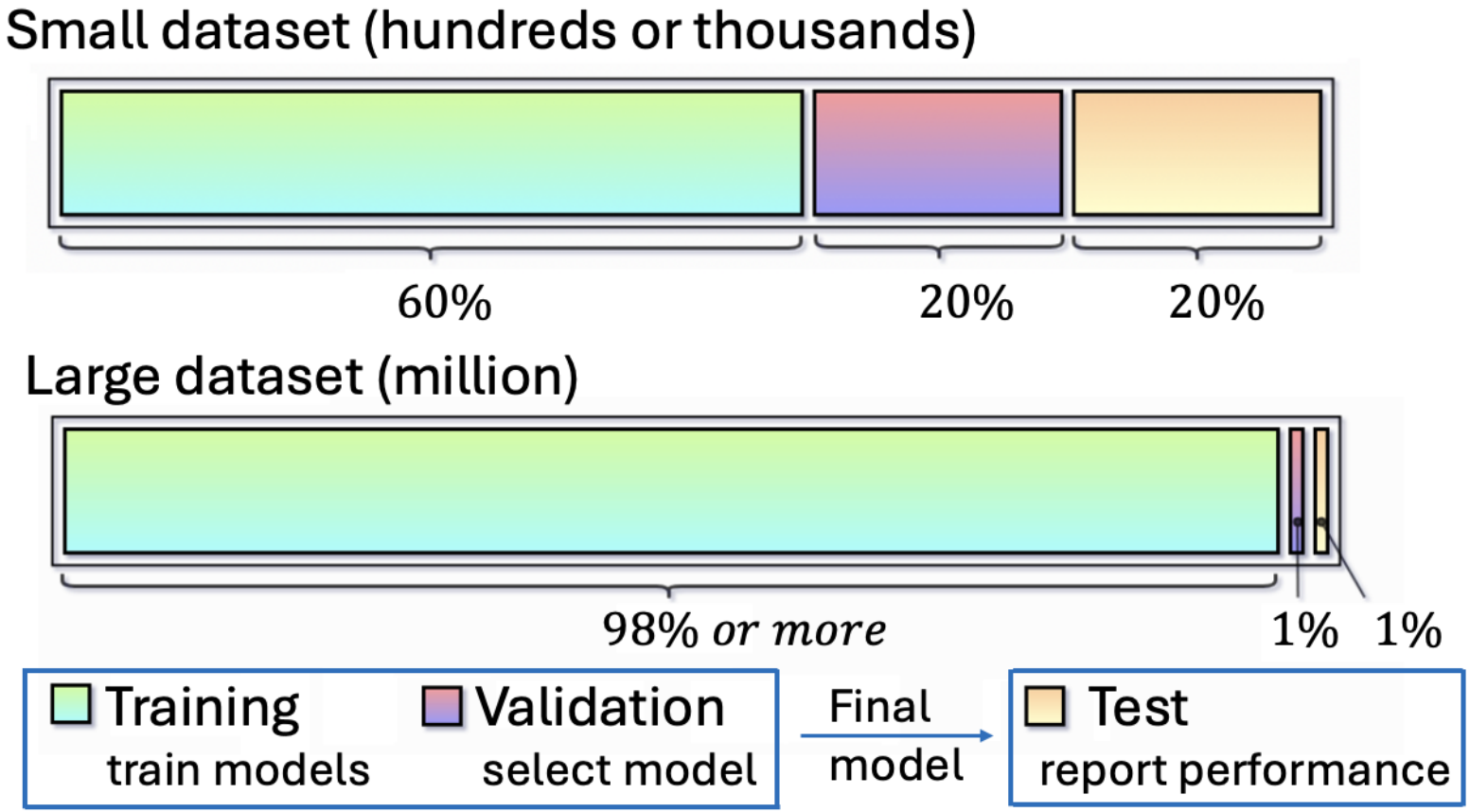
Hyper-parameters

Applied deep learning is a very empirical process.

Andrew Ng's list by importance:

- Learning rate
- momentum
- # of hidden units
- mini-batch size
- learning rate decay
- # of hidden layers
- activation functions: most popular ReLU,
- # iterations of GD
- dropout rate (if needed): 20%-52% of each layer
- tuning parameters in l_2-penalty
- ...

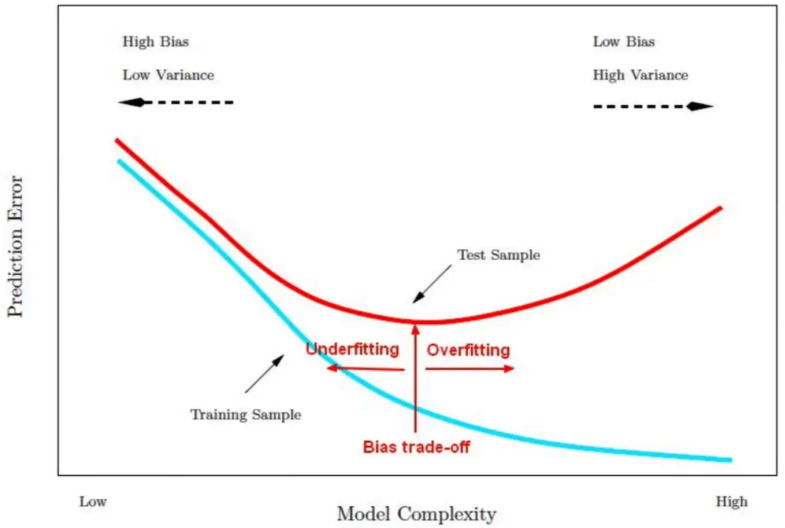
Training/Validation/Test Sets



In case of different training/test distribution, make sure validation and test come from the same distribution.

Diagnosis of Bias/Variance Problem

- large training error (compared to Bayes classifier) → high bias
  - try more complex model
  - try longer training time
  - appropriate NN architecture.
- small training error, large validation error → high variance
  - need more data
  - need regularization
  - appropriate NN architecture.
- large training error, even larger validation error → high bias, high variance



Key elements in deep learning to drive down bias & variance:

- more data
- bigger NN
- regularization
- other training tools

Why Deep not Wide - Circuit Theory

"Deep": more hidden layers

"Wide": more hidden units in a layer.

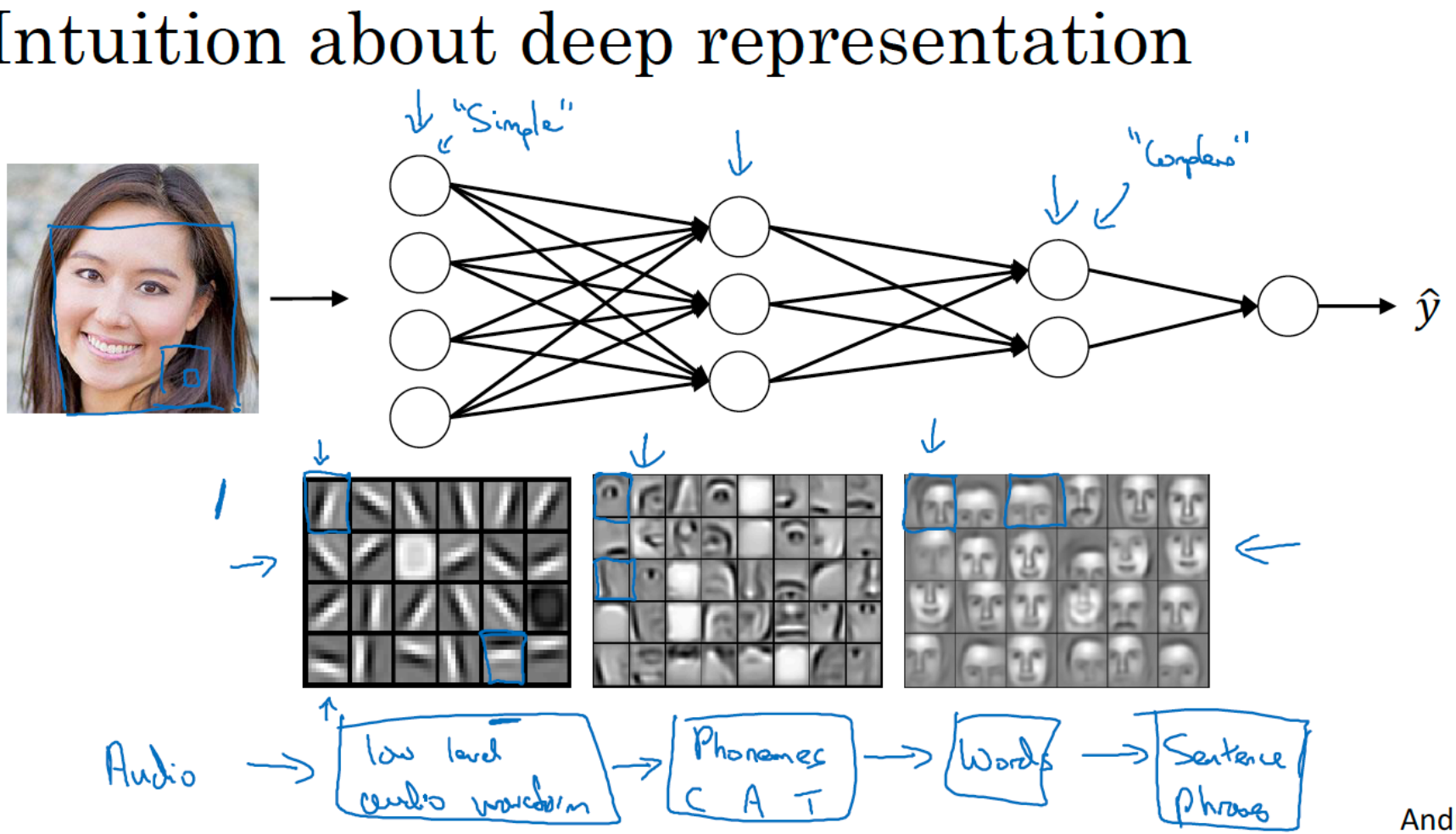
There are functions you can compute with a "small" L-layer deep NN that shallower networks requires exponentially more hidden unites to compute.

Consider function $Y = f(X) = \text{XOR}(X_1, \ldots, X_p)$,
where $X_1, \ldots, X_p, Y$ are all binary $\{0, 1\}$.

$f(x)$ can be represented by

- a shallow 2-layer NN with 2^{p-1} hidden units.
- a $O(\log_2 p)$-layer deep NN with a total of $O(p)$ hidden units.

Why Deep not Wide - Intuition



Why Deep not Wide - Summary

- Hierarchical structure: raw data is transformed into progressively more abstract and meaningful representations.
- Transferability/generalization
- Feature learning, interpretability, Invariance, etc.

Discussion here:
`https://stats.stackexchange.com/questions/222883/why-are-neural-networks-becoming-deeper-but-not-wider`

`https://medium.com/@Coursesteach/deep-learning-part-32-why-deep-representations-3779ac41595c`

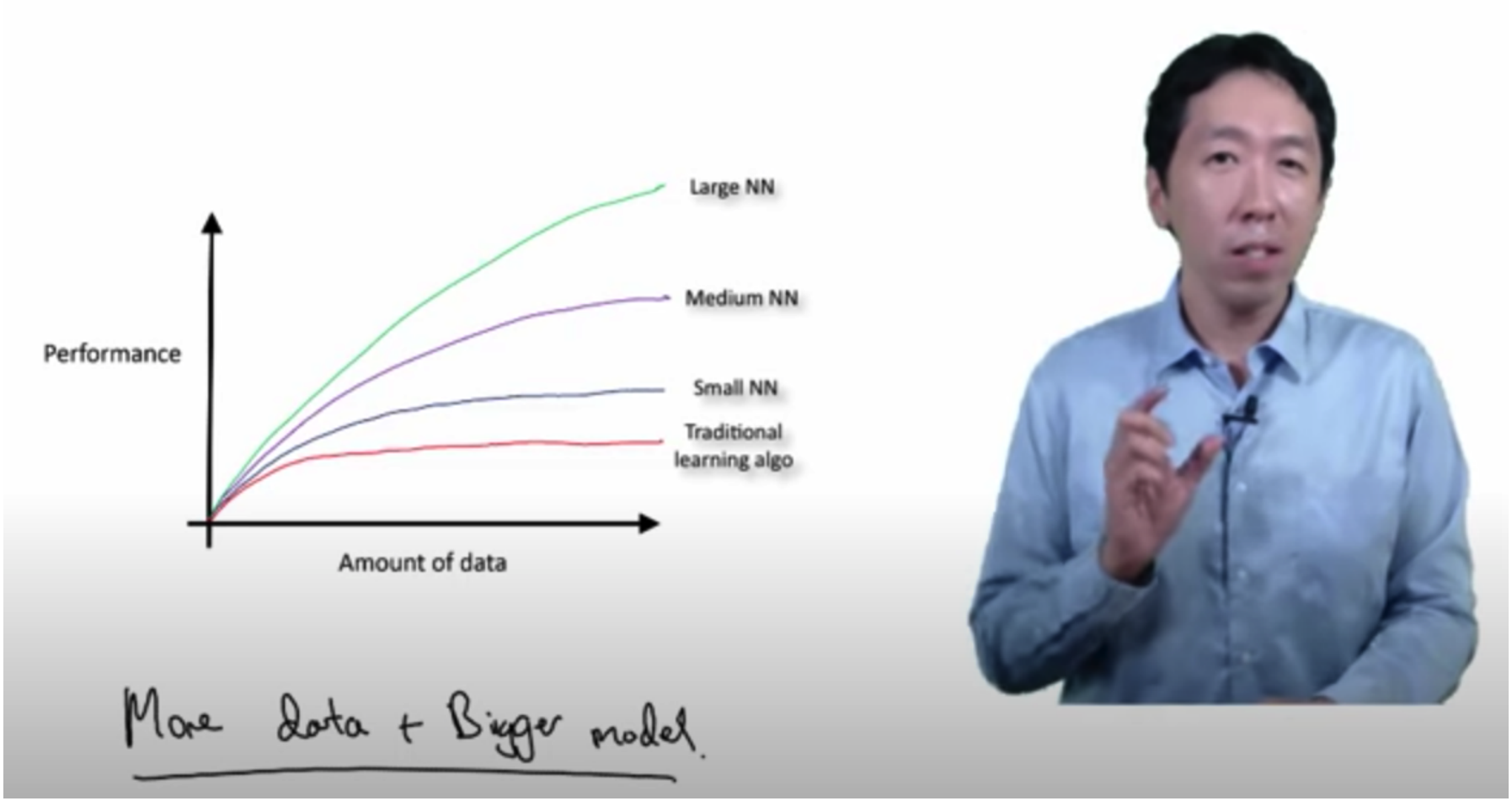
When to use Deep Learning

Hitters dataset example

- Y: Salary of a baseball player in 1987
- $X \in \mathbb{R}^{19}$: the player's performance statistics from 1986
- A total of 263 players divided 176 for training and 87 for testing.

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network (one hidden layer, 64 ReLU unit)	1409	257.4	0.54

When to use Deep Learning



DL is appropriate when the sample size is large and the interpretability is not a high priority.