Deep Learning for Vision

Instructor - Simon Lucey

RVSS - 2026

THE UNIVERSITY of ADELAIDE

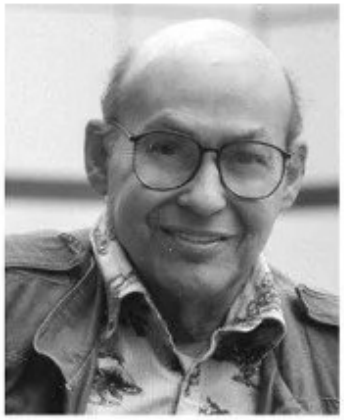
AUSTRALIAN INSTITUTE FOR MACHINE LEARNING

Today

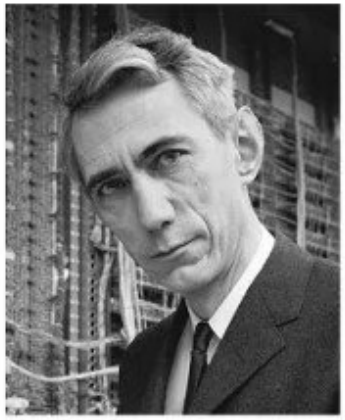
- **AI – the beginning….**
- Shallow and Deep Neural Networks
- Loss Functions and Generalization
- Optimization and Backpropagation

1956 Dartmouth Conference: The Founding Fathers of AI

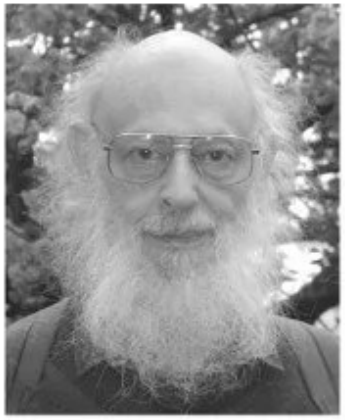
John MacCarthy



Marvin Minsky



Claude Shannon

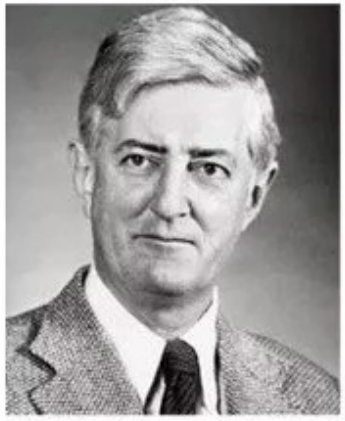


Ray Solomonoff

Alan Newell

Herbert Simon

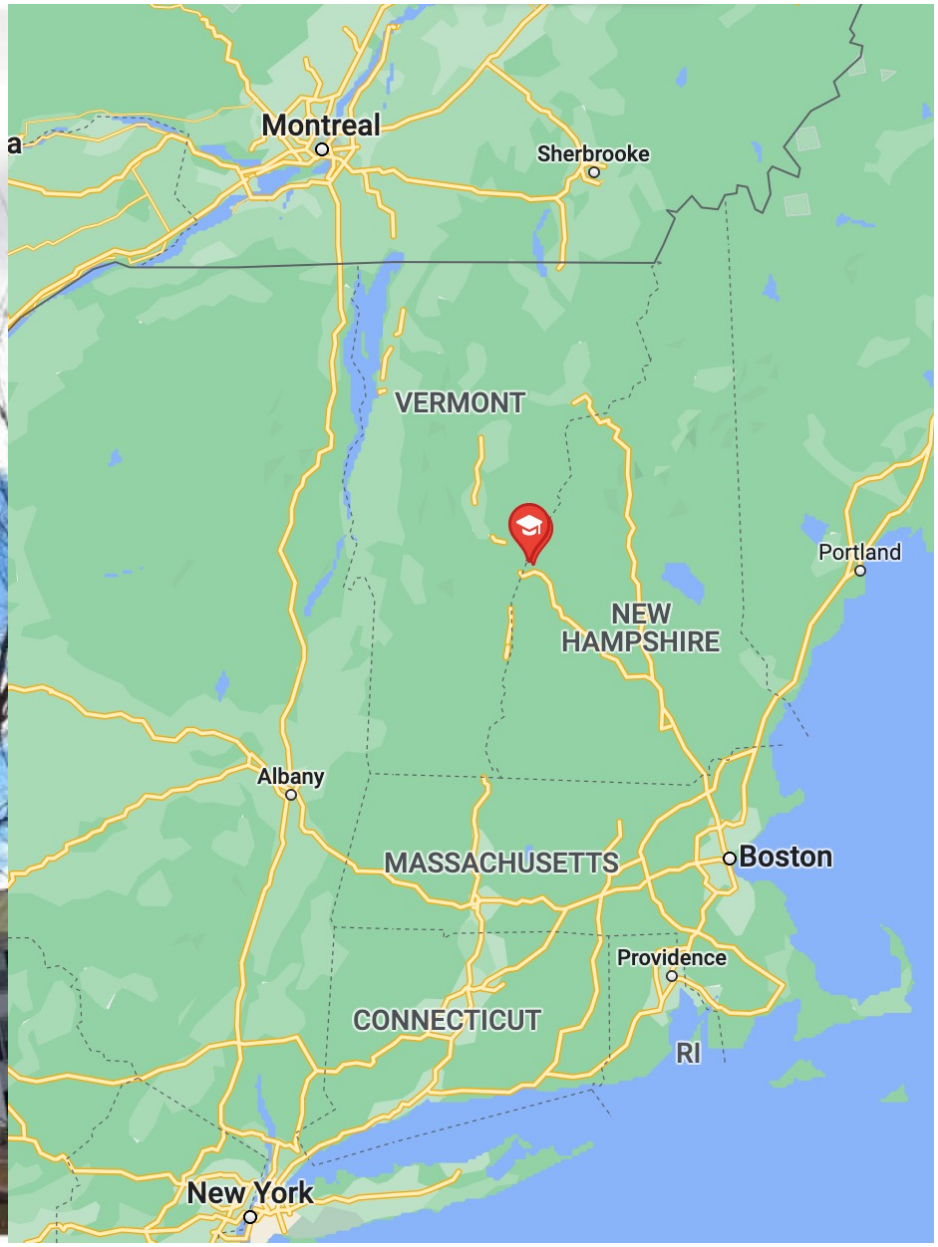
Arthur Samuel



Oliver Selfridge

Nathaniel Rochester

Trenchard More


Montreal
Sherbrooke
VERMONT
Portland
NEW
HAMPSHIRE
Albany
MASSACHUSETTS
Boston
Providence
CONNECTICUT
RI
New York

Conventional programming

```
ass Room(object):
    def __init__(self, inventory, desc, short_desc):
        self.inventory = inventory
        self.__n = None
        self.__s = None
        self.__e = None
        self.__w = None
        self.__desc = desc
        self.__short_desc = short_desc
        self.__gate_n = None
        self.__gate_s = None
        self.__gate_e = None
        self.__gate_w = None

        if not isinstance(desc, str):
            raise TypeError ("the input provided is not a string.")
        elif not isinstance(short_desc, str):
            raise ValueError ("the string provided is empty.")

    # these set the gates
    # they set the opposite gates, with checks to avoid recursion loops
    def set_n(self, other):
        if not isinstance(other, Room) or not other:
```

Machine Learning

Perceptron

- Rosenblatt simulated the perceptron on a IBM 704 computer at Cornell in 1957.
- Input scene (i.e. printed character) was illuminated by powerful lights and captured on a 20x20 cadmium sulphide photo cells.
- Weights of perceptron were applied using variable rotary resistors.
- Often times referred to as the very first neural network.

“Frank Rosenblatt”

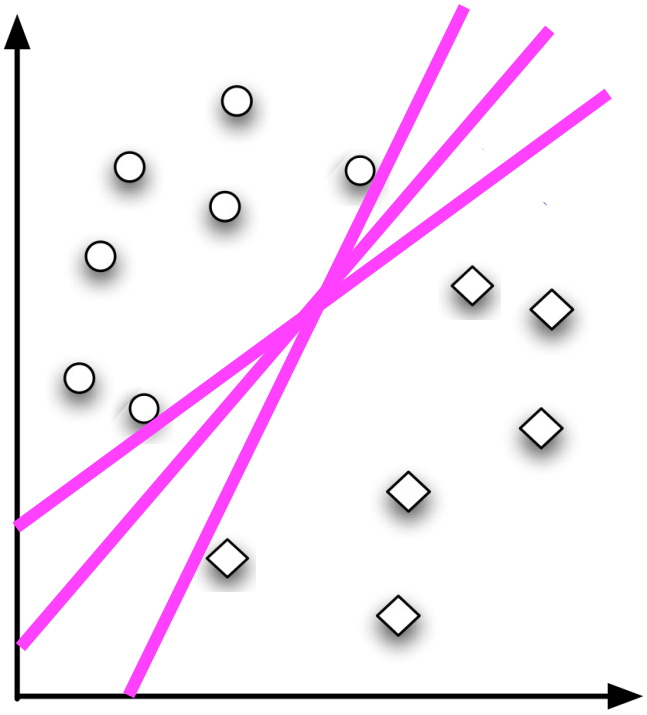
The New York Times

NEW NAVY DEVICE LEARNS BY DOING

July 8, 1958

"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence... Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the ets as mechanical space explorers"

Perceptron = Linear Discriminant



$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} \max(0, -y_i \mathbf{x}_i^T \mathbf{w})$$

$y_i = +1$

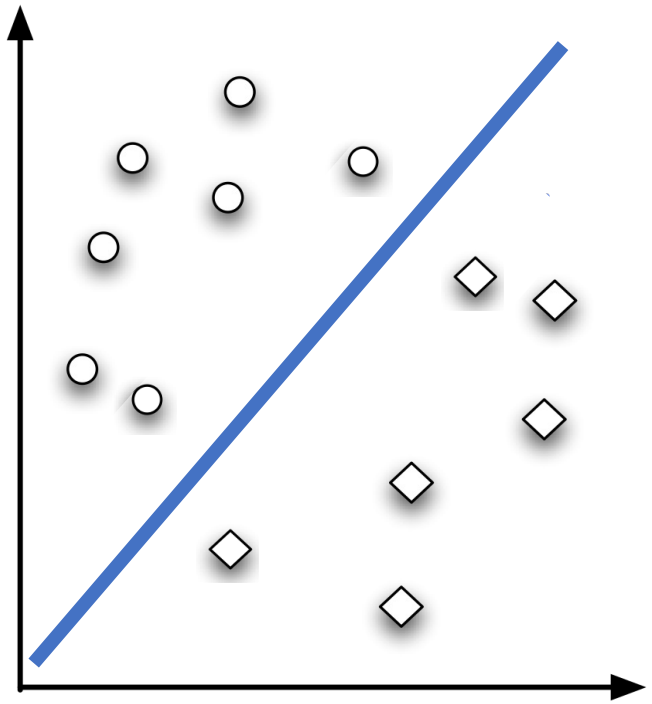
$y_i = -1$

binary labels

$\mathbf{x}_i = i$-th training example

$\mathbf{w} =$ weight vector

Perceptron = Linear Discriminant



$t_i = +1$

$t_i = -1$

binary labels

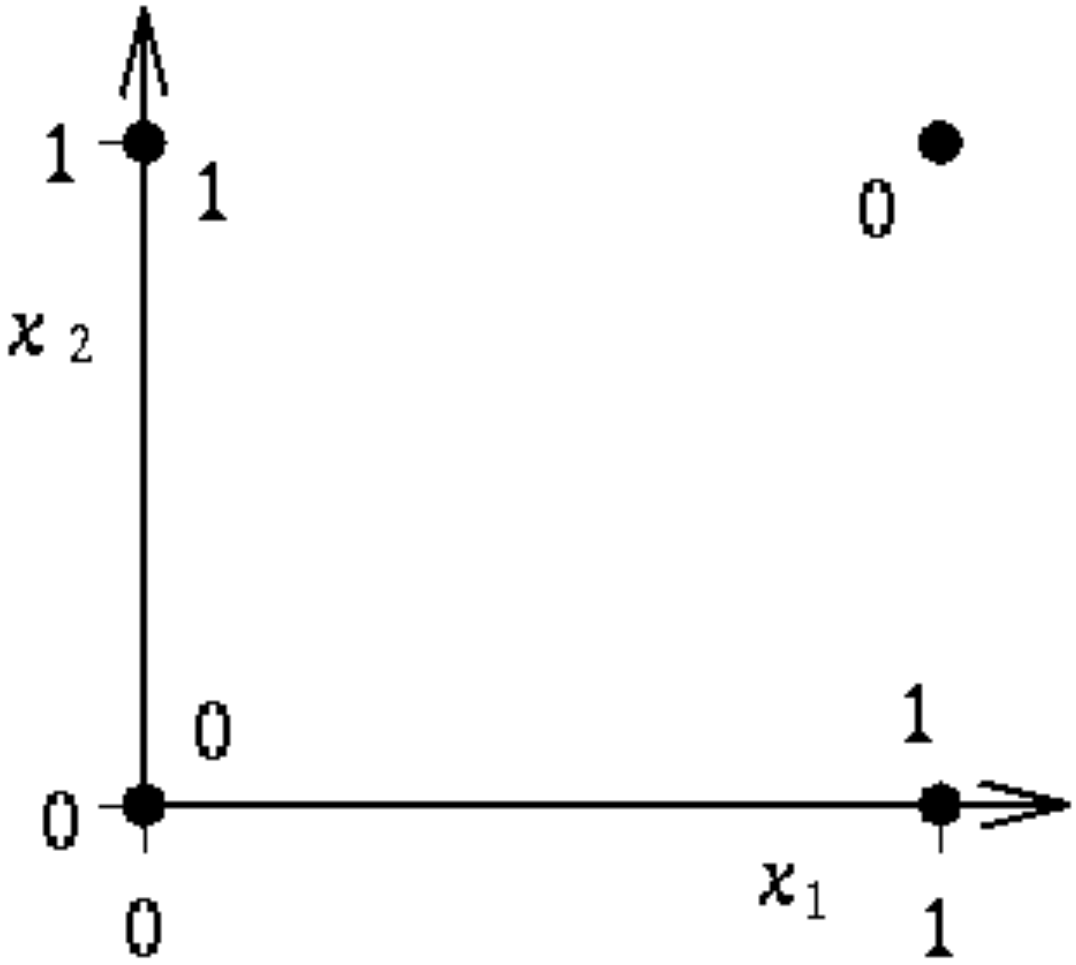
$\mathbf{x}_i = i$-th training example

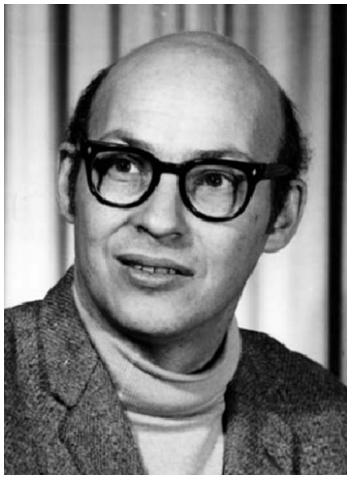
$\mathbf{w}$ = weight vector

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} \mathcal{L}(y_i \cdot \mathbf{x}_i^T \mathbf{w}) + \Omega(\mathbf{w})$$

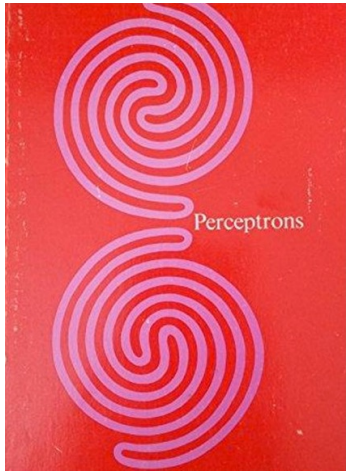
"regularizer"

XORs, Minsky and the AI Winter





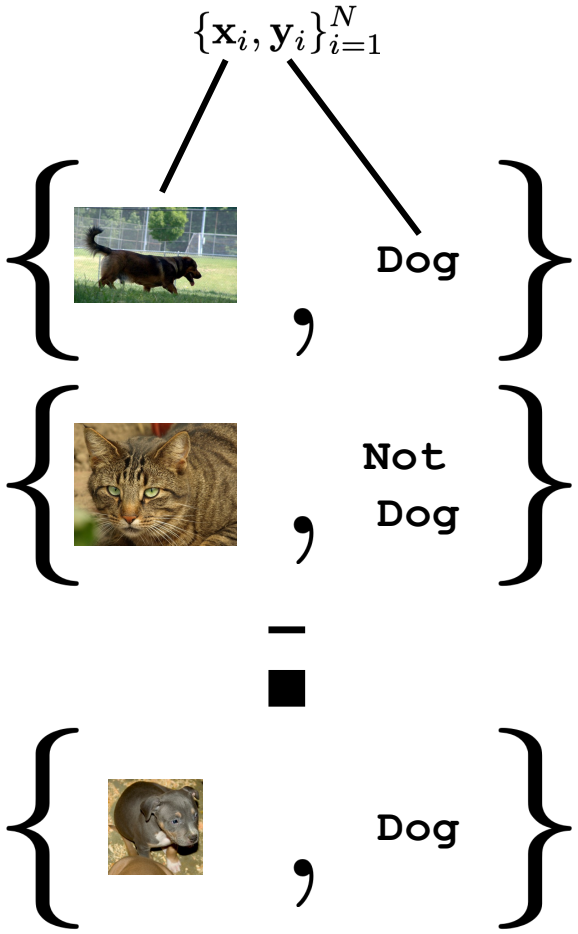
"Marvin Minsky"



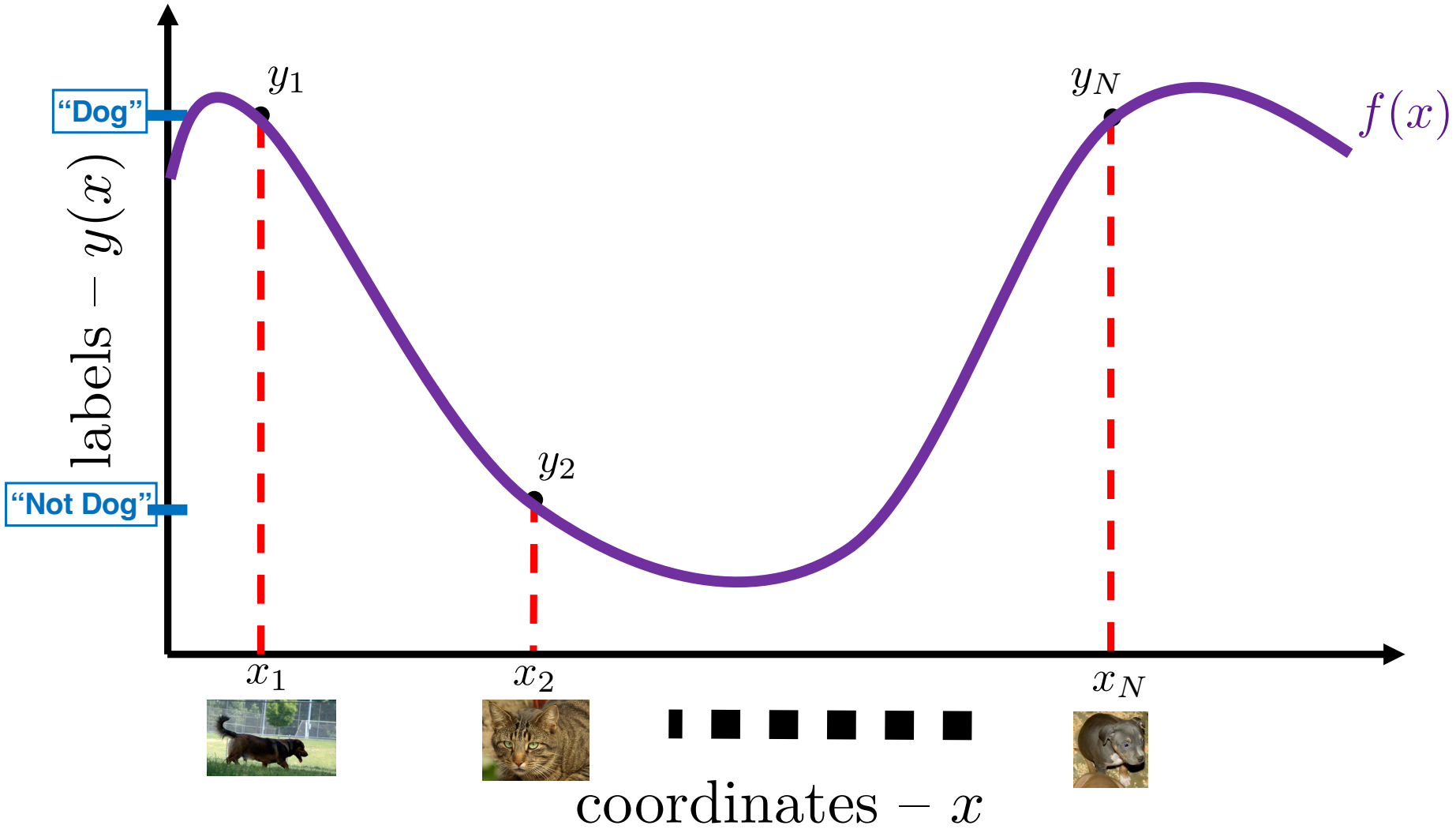
Today

- AI – the beginning….
- **Shallow and Deep Neural Networks**
- Loss Functions and Generalization
- Optimization and Backpropagation

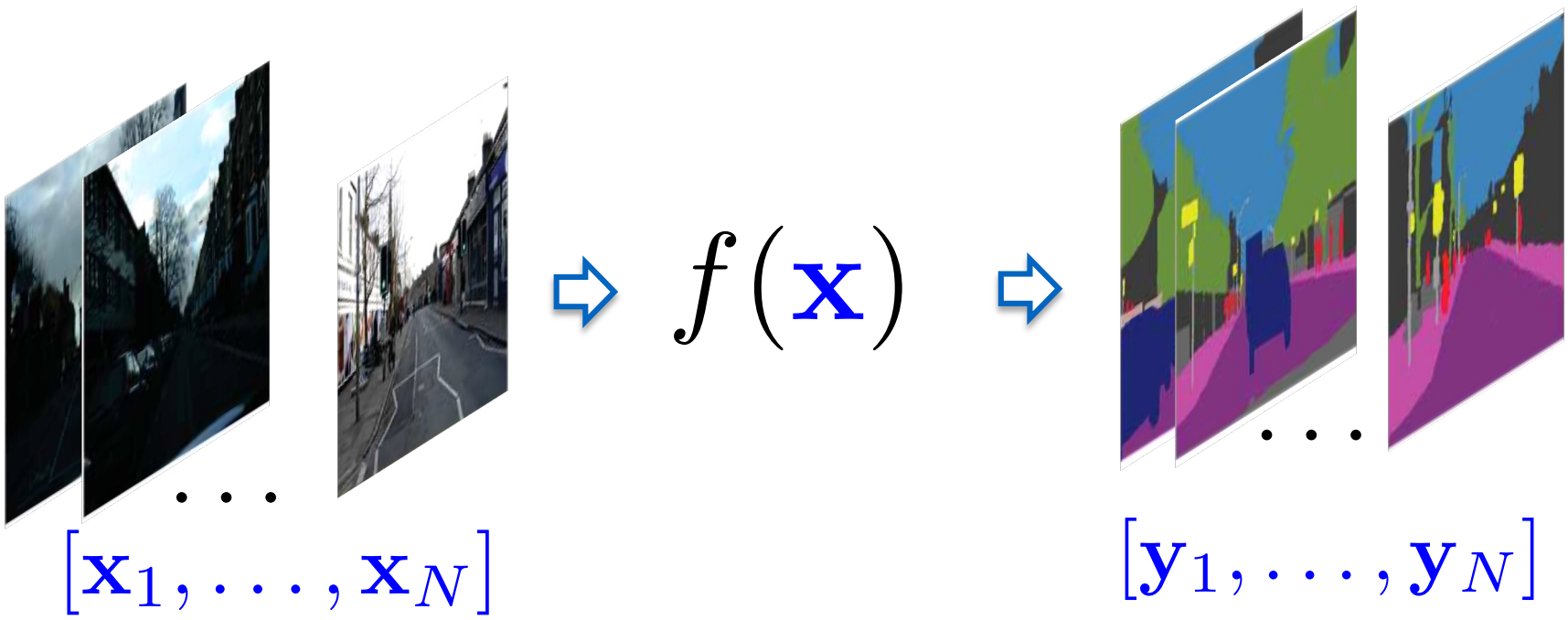
Visual Learning



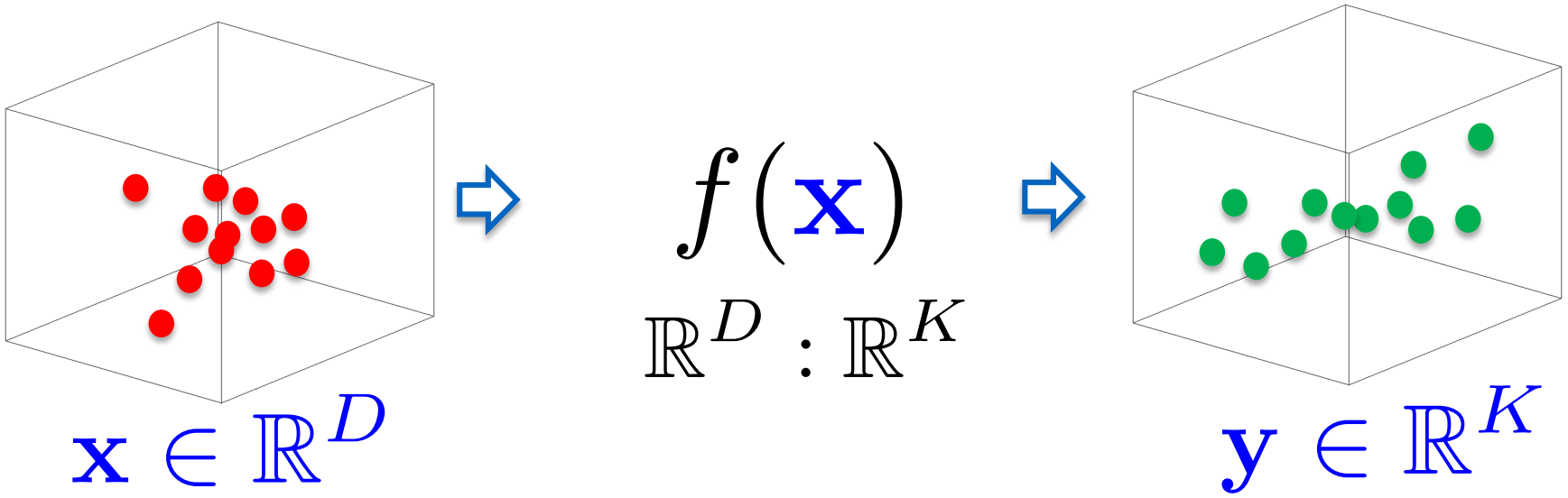
Visual Learning is Curve Fitting!!!



Multi-dimensional Sampling?



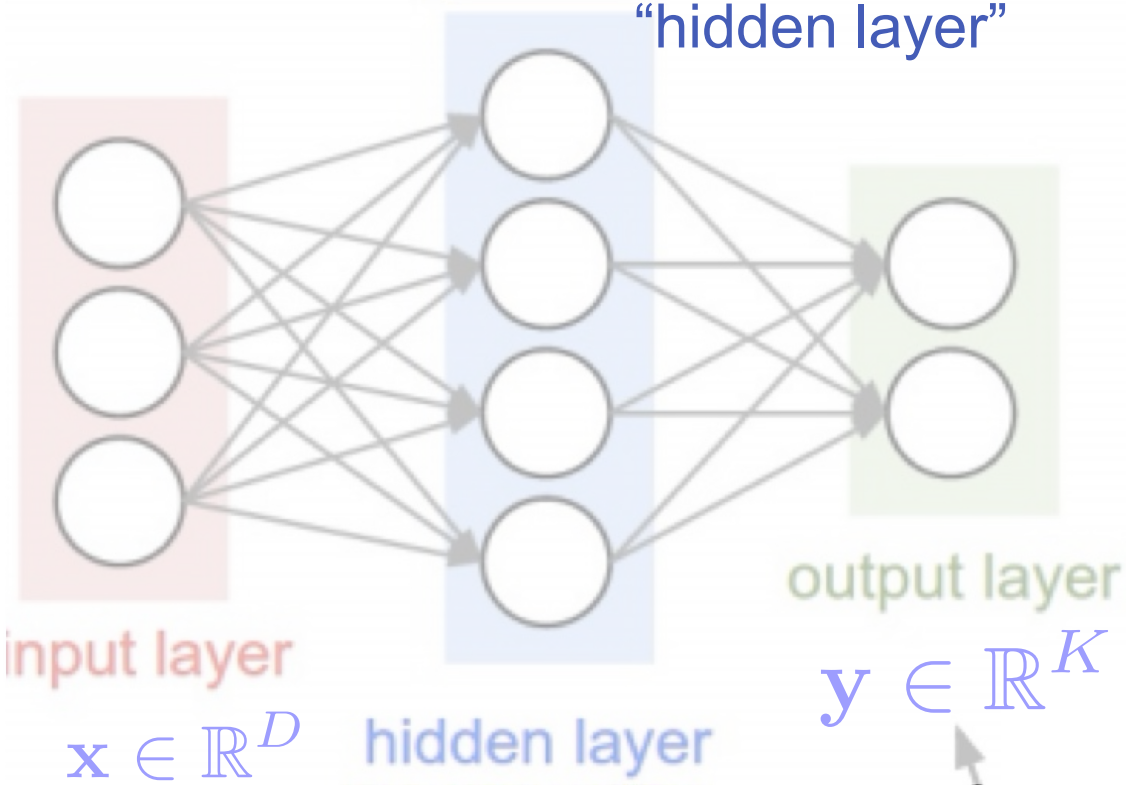
Multi-dimensional Sampling?



"learned"

$$f(\mathbf{x}) \approx \boldsymbol{\theta}\eta(\mathbf{W}\mathbf{x} + \boldsymbol{\delta})$$

"hidden layer"



$$f(\mathbf{x}) \approx \underset{(K \times M)}{\boldsymbol{\theta}} \eta\Big(\underset{(M \times D)}{\mathbf{W}}\mathbf{x} + \underset{(M \times 1)}{\boldsymbol{\delta}}\Big)$$

Shallow vs Deep Network

- A shallow learner has **only one** hidden vector unit,

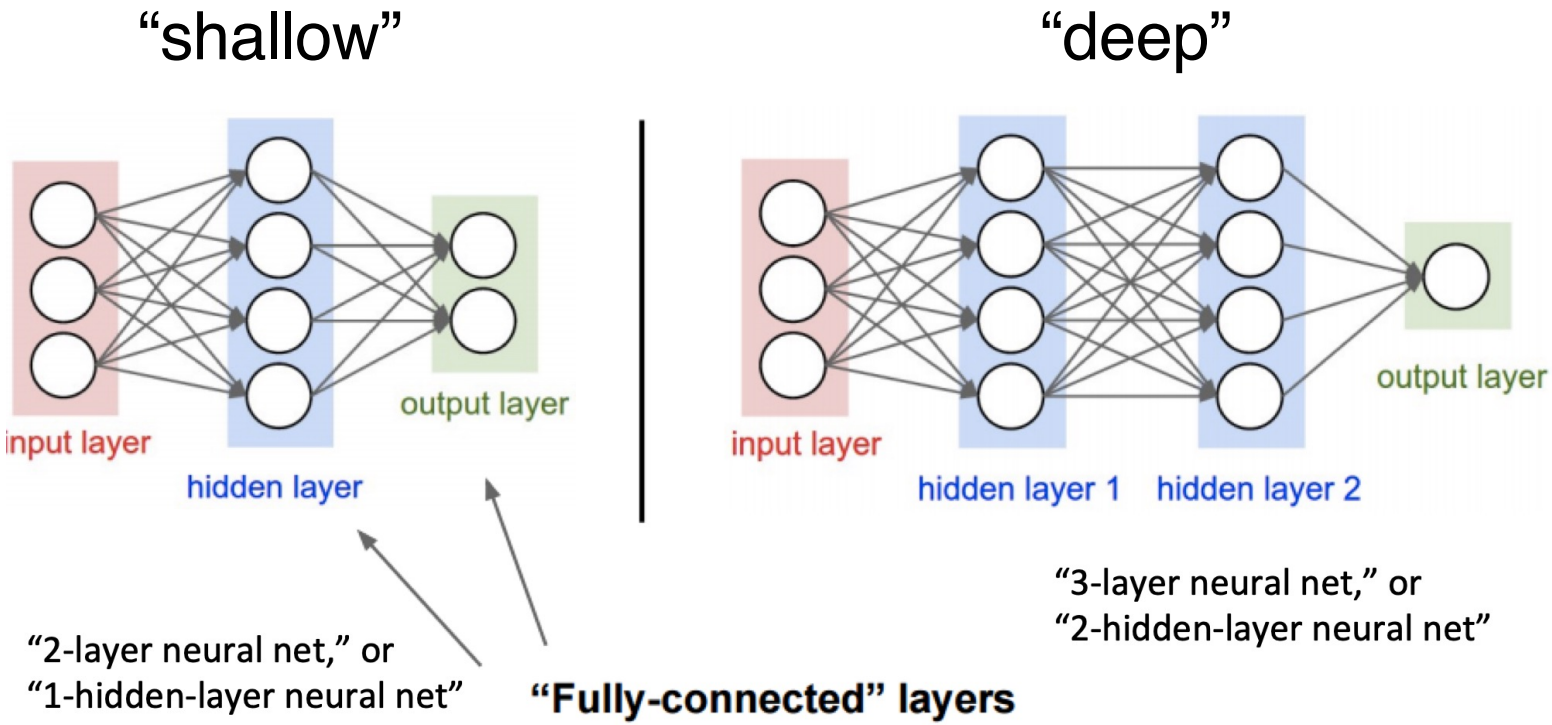
$$\boldsymbol{\theta}\eta(\mathbf{W}\mathbf{x} + \boldsymbol{\delta})$$

- A deep learner **has more than one** hidden vector unit,

$$\boldsymbol{\theta}\eta(\mathbf{W}^{(1)}\eta(\mathbf{W}^{(0)}\mathbf{x} + \boldsymbol{\delta}^{(0)}) + \boldsymbol{\delta}^{(1)})$$

"Geoffrey Hinton"

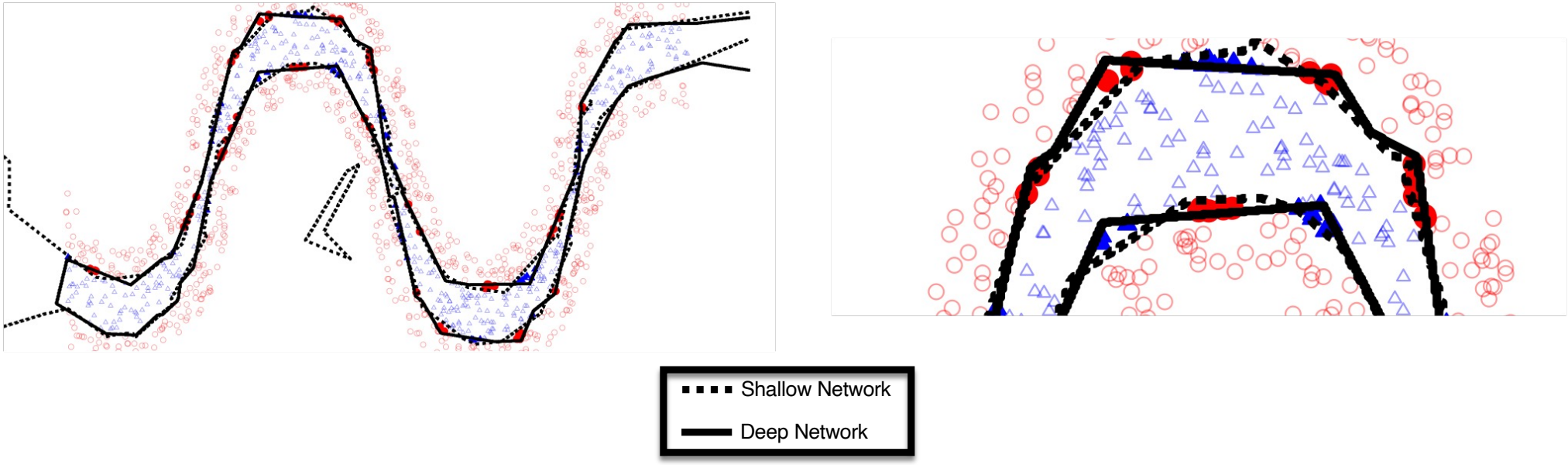
Shallow vs. Deep Layers



Why deepness?

Why Deep?

- Some theorize that deeper nets are exponentially more expressive than shallow ones.



Montufar, Guido F., et al. "On the number of linear regions of deep neural networks." NeurIPS 2014.

Today

- AI – the beginning….
- Shallow and Deep Neural Networks
- **Loss Functions and Generalization**
- Optimization

Training set, validation set, and test set

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$$

Use validation set used to evaluate learning system during development iterations

What ratio of training, validation, and test?

Small data regime (100-10,000 samples): 60%/20%/20%

Big data regime (>100,000 samples): 90%/5%/5%

Example Input/Output Pairs

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$$

$$\hat{\mathbf{y}}_i = f(\mathbf{x}_i)$$

prediction

predictor

input

Loss Functions

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$$

$$\hat{\mathbf{y}}_i = f(\mathbf{x}_i)$$

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i(f(\mathbf{x}_i), \mathbf{y}_i)$$

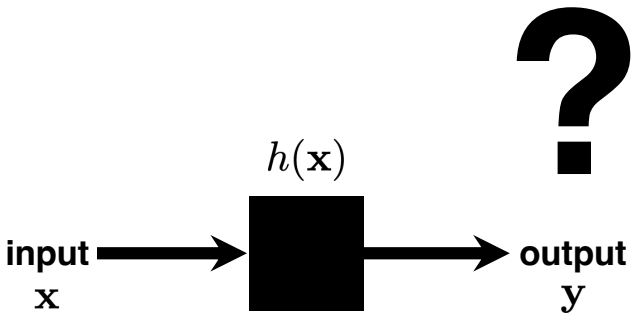
Example

$$\mathcal{L}_i = ||\mathbf{y}_i - \hat{\mathbf{y}}_i||_2^2$$ **Gaussian**

$$\mathcal{L}_i = -\log\left(\frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}\right)$$ **Softmax**

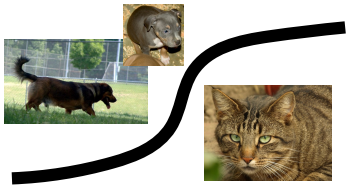
Visual Learning VS Pure Optimization

Input is specifically imagery (images, depth, or video), output is “understanding”



Visual Learning VS Pure Optimization

Optimize on Training Set but Evaluate on Test Set

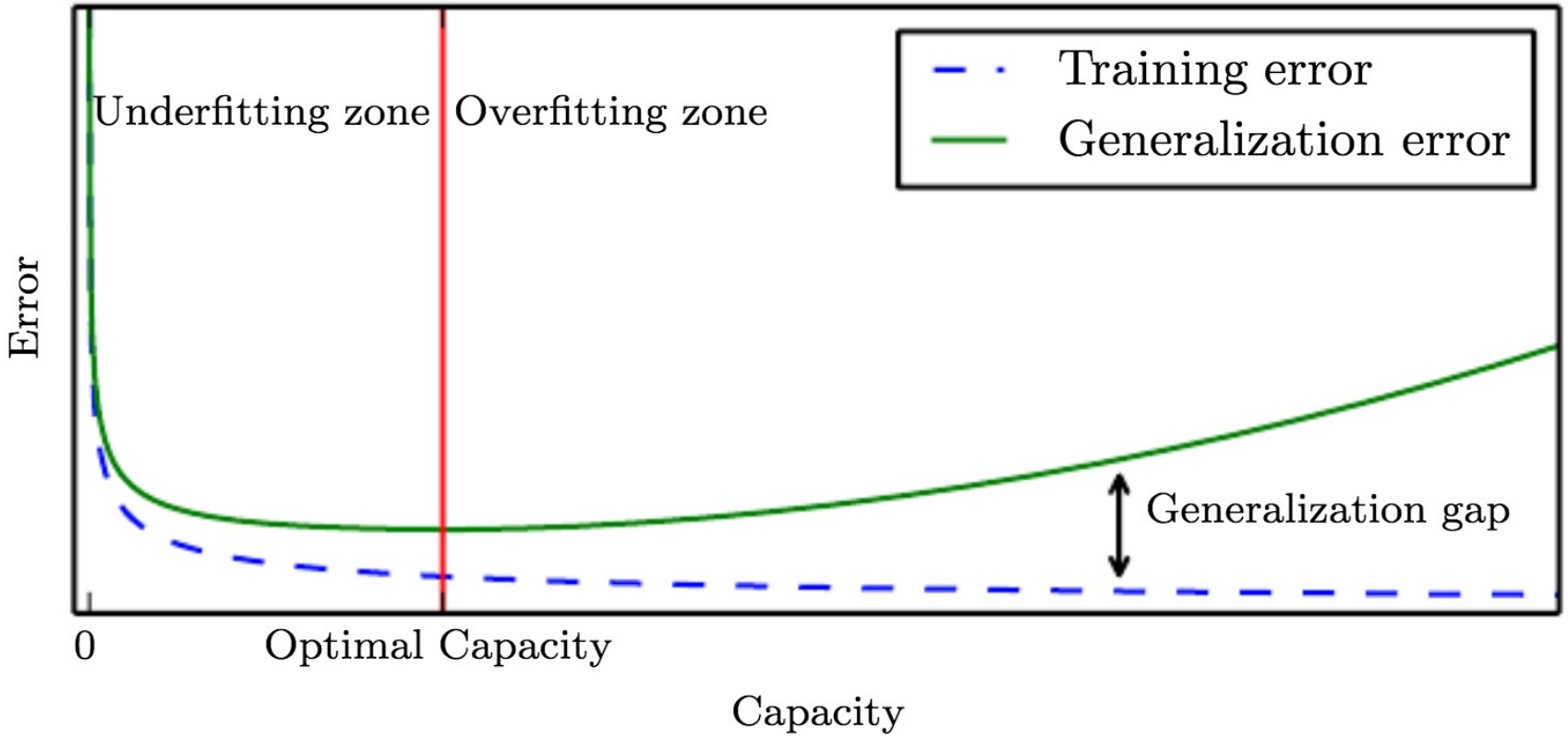


	System A	System B	System C	System D
Training Set Error:	2%	22%	22%	2%
Test Set Error:	24%	24%	34%	3%
	"overfitting" high variance	"underfitting" high bias	high bias and high variance	low bias and low variance

Key concern for high capacity learning system

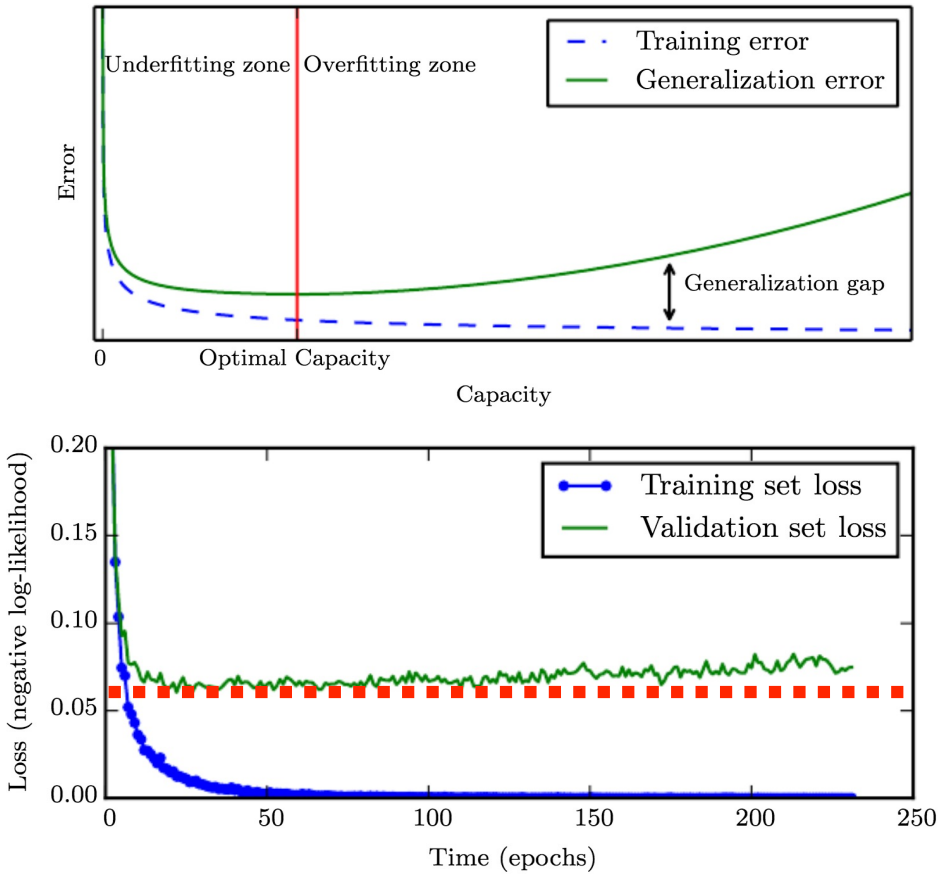
Visual Learning VS Pure Optimization

Input is specifically imagery (images, depth, or video), output is “understanding”



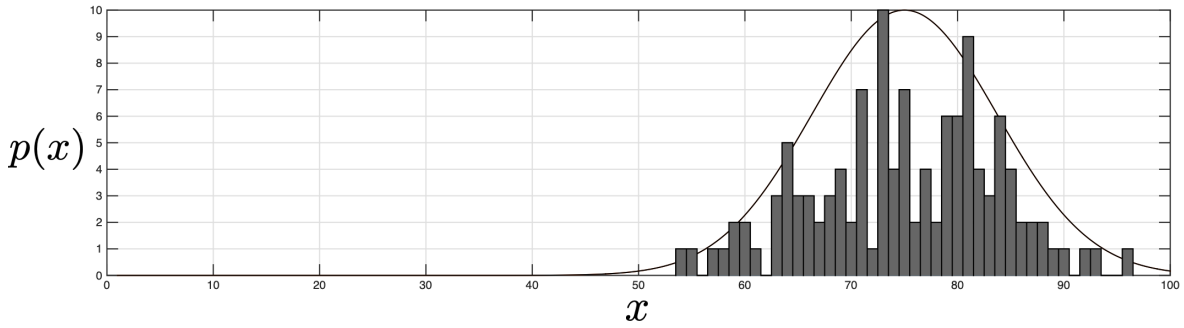
Visual Learning VS Pure Optimization

Termination Strategy: Early Stopping



Visual Learning VS Pure Optimization

Empirical Risk Minimization



$$J^*(\mathtt{W}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}} \mathcal{L}(f(\mathbf{x};\mathtt{W}),\mathbf{y}))$$

p_{data} : **Data-Generating Distribution**

usually unknown

$$J(\mathtt{W}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \hat{p}_{\text{data}}} \mathcal{L}(f(\mathbf{x};\mathtt{W}),\mathbf{y}))$$

$\hat{p}_{\text{data}}$: **Empirical Distribution**

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \hat{p}_{\text{data}}}[\mathcal{L}(f(\mathbf{x};\mathtt{W}),\mathbf{y}))] = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i(f(\mathbf{x}_i;\mathtt{W}),\mathbf{y}_i)$$

$$\mathbb{E}[h(x)] = \int_{\mathbb{R}} h(x)p(x)dx$$

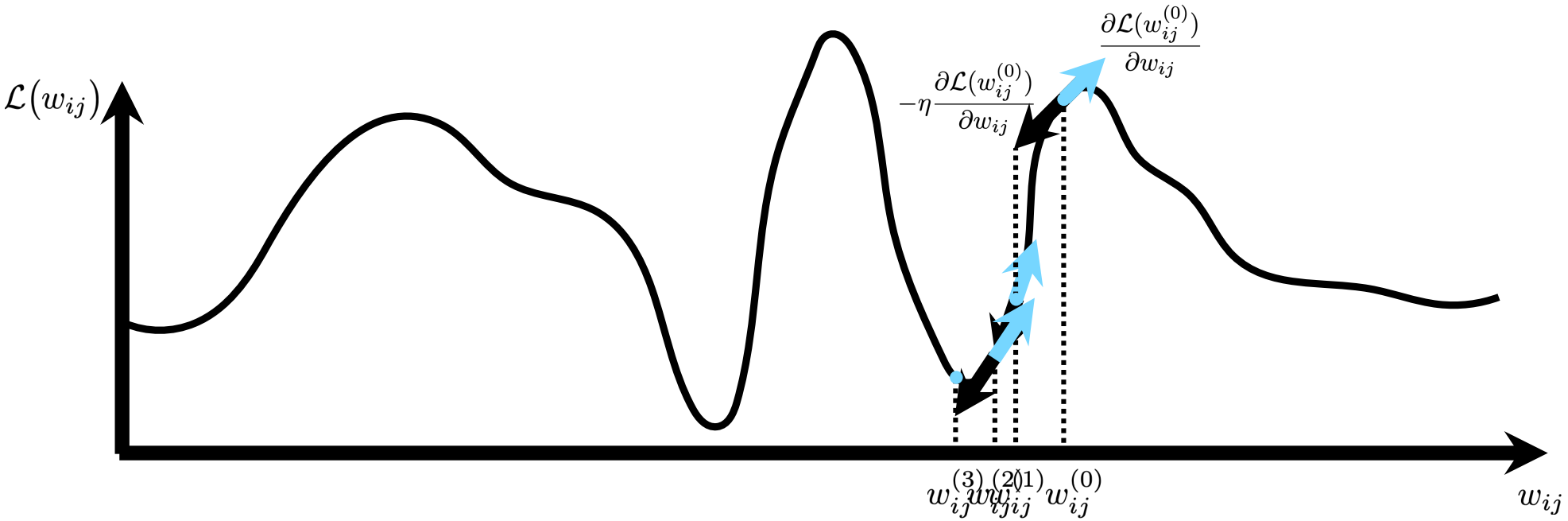
Today

- AI – the beginning….
- Shallow and Deep Neural Networks
- Loss Functions and Generalization
- **Optimization and Backpropagation**

Review: First-order Methods

Gradient Descent: Take a step proportional to the negative of the gradient

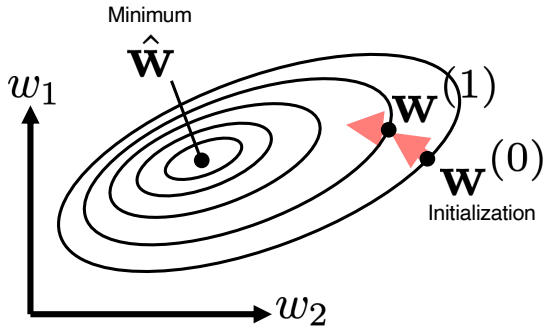
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial \mathcal{L}(w_{ij}^{(t)})}{\partial w_{ij}}$$



Review: First-order Methods

Gradient Descent: Take a step proportional to the negative of the gradient

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial \mathcal{L}(w_{ij}^{(t)})}{\partial w_{ij}}$$



Learning Rate

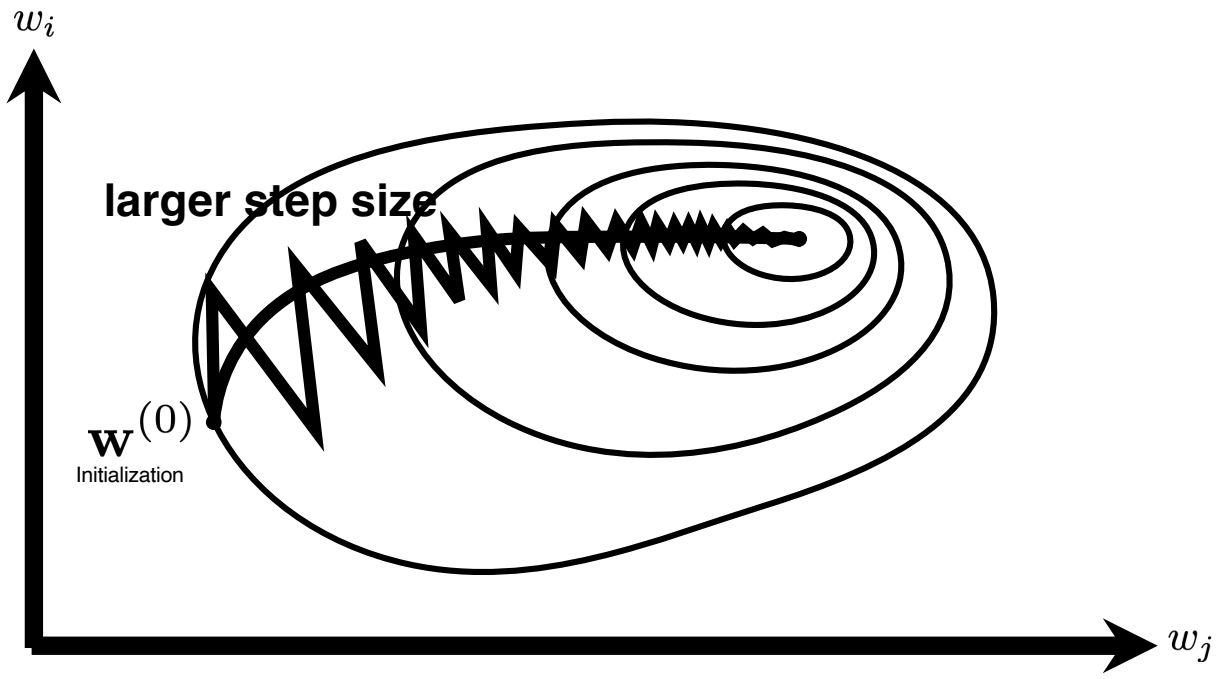
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^{(t)})$$

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{11}} & \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{12}} & \cdots & \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{UV}} \end{pmatrix}$$

Jacobian

Review: First-order Methods

Zigzagging: Gradient descent does not exploit curvature information



Review: First-order Methods

Gradient Descent: Take a step proportional to the negative of the gradient

Photo: Dhruba Jyoti Baruah

Review: First-order Methods

Gradient Descent: Take a step proportional to the negative of the gradient

Photo: Dhruba Jyoti Baruah

Review: First-order Methods

Gradient Descent: Take a step proportional to the negative of the gradient

Photo: Dhruba Jyoti Baruah

Optimization Challenges

Gradient Descent: Prone to Local Minima

Photo: Dhruba Jyoti Baruah

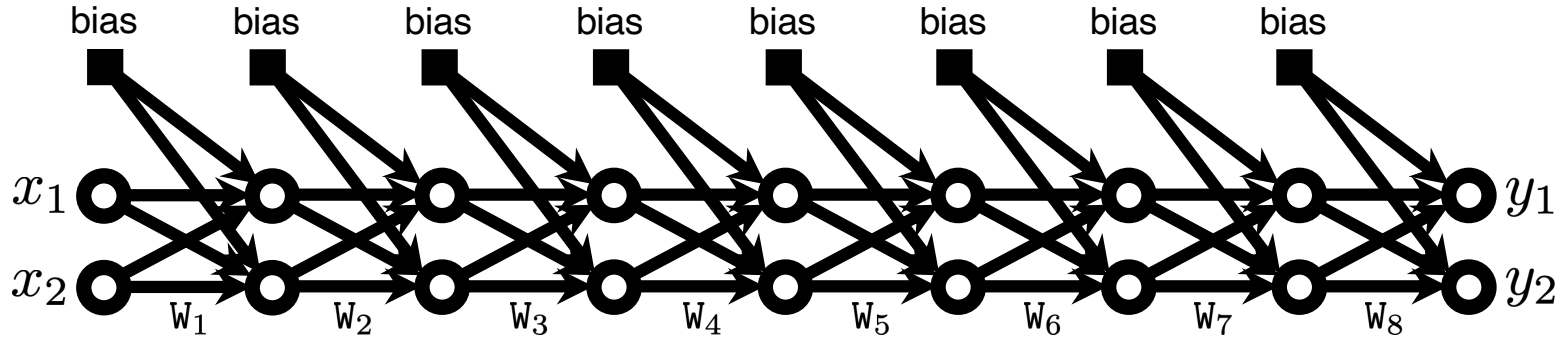
Optimization Challenges

Gradient Descent: Ill-conditioning

Photo: Bob Wick

Optimization Challenges

Vanishing/Exploding Gradients in Deep Networks



$$\hat{\mathbf{y}} = \mathtt{W}_8\ \mathtt{W}_7\ \mathtt{W}_6\ \mathtt{W}_5\ \mathtt{W}_4\ \mathtt{W}_3\ \mathtt{W}_2\ \mathtt{W}_1\ \mathbf{x}$$ **(assume no nonlinearities)**

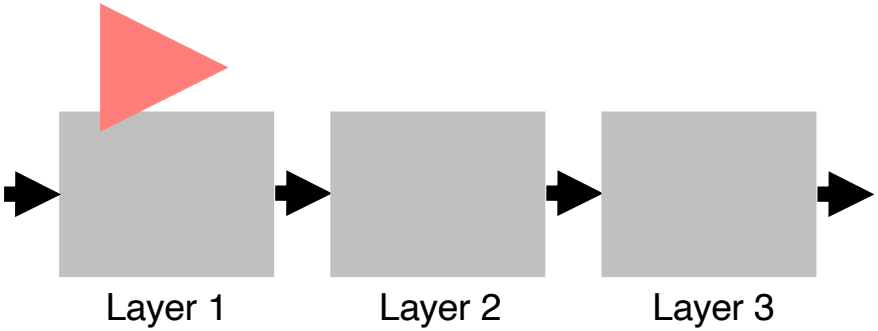
$$\hat{\mathbf{y}} = \mathtt{W}^8\mathbf{x}$$ **(assume tied weights)**

$$\mathtt{W}_i = \mathtt{W} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \qquad \mathtt{W}^8 = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^8 = \begin{bmatrix} 25.6289 & 0 \\ 0 & 25.6289 \end{bmatrix}$$ **exploding**

$$\mathtt{W}_i = \mathtt{W} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \qquad \mathtt{W}^8 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^8 = \begin{bmatrix} 0.0039 & 0 \\ 0 & 0.0039 \end{bmatrix}$$ **vanishing**

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



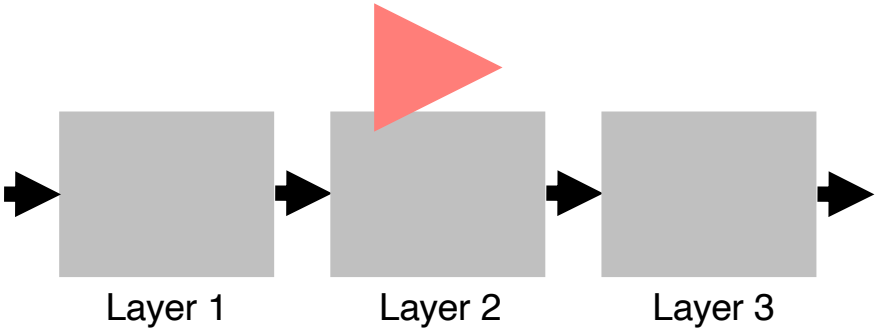
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



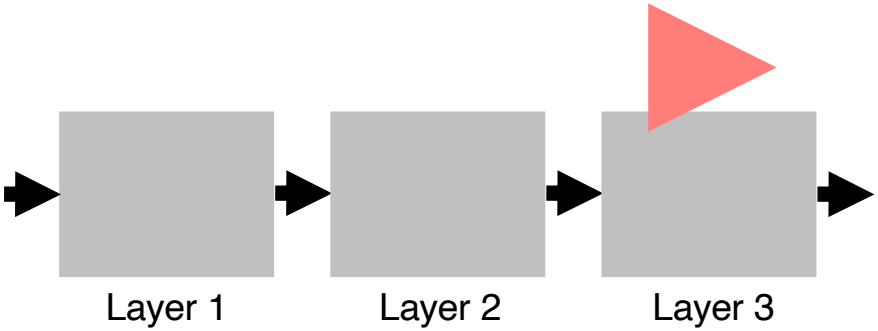
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



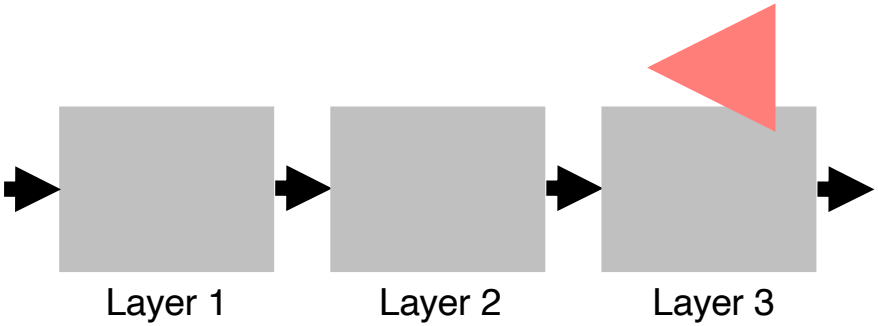
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



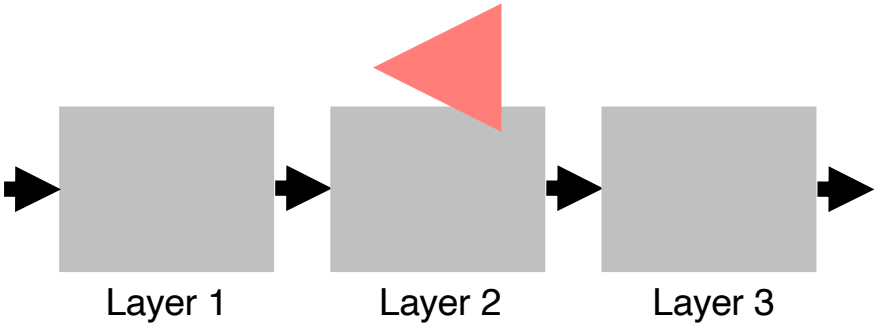
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



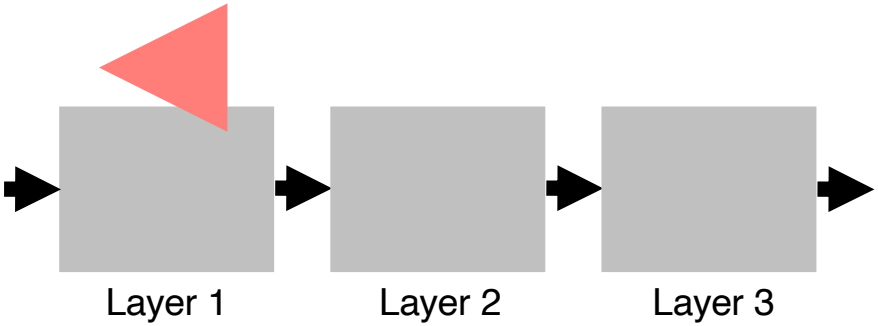
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



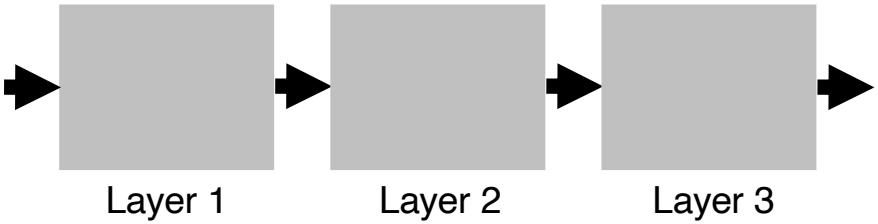
Step 1: Compute loss on training batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



Step 1: Compute loss on mini-batch (Forward Pass)

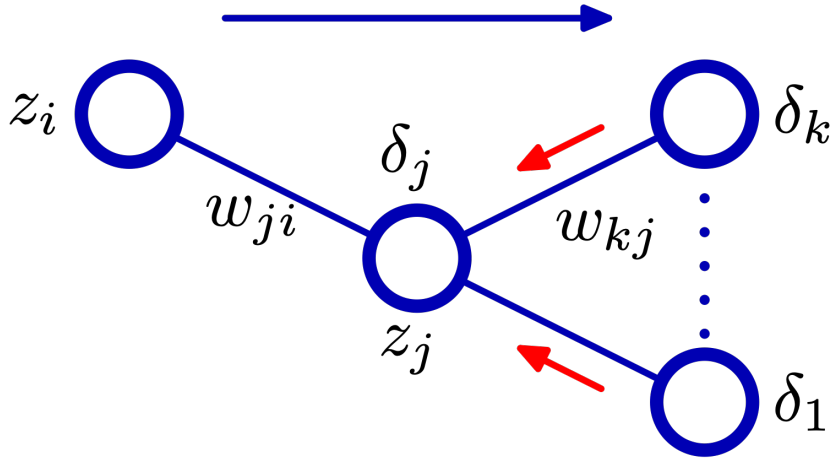
Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^{(t)})$$

Back Propagation

- Back propagation refers to the property that components of gradients found at higher layers, can be re-used at lower layers.



Optimization Algorithms

Gradient Descent for Classification

$$\mathtt{W}^* = \arg\min_{\mathtt{W}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(f(\mathbf{x}_i; \mathtt{W}), \mathbf{y}_i)$$

gradient descent update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^{(t)})$$

Standard error of the mean: $\frac{\sigma}{\sqrt{n}}$

100 samples → **100x computation** / **10x error reduction** → **10,000 samples**

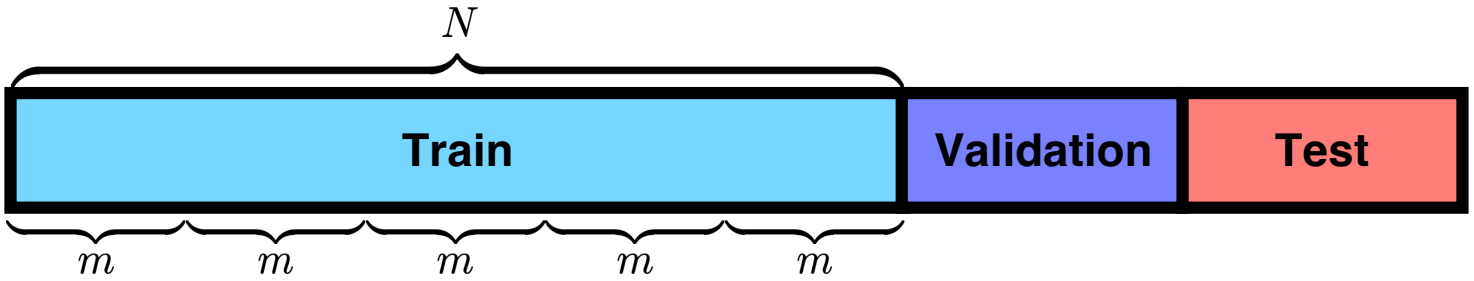
Is it better to compute approximate estimates of the gradient and iterate rapidly

OR

compute the exact gradient and iterate slowly?

Optimization Algorithms

SGD: Stochastic Gradient Descent estimator introduces noise

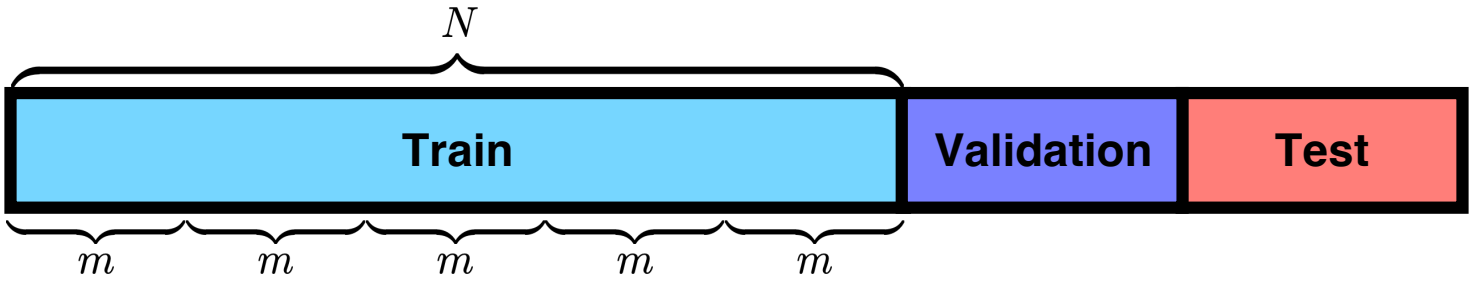


Algorithm: Stochastic Gradient Descent (SGD) update at training iteration t

Input: Learning rate ϵ_t, initial parameter $\mathbf{W}$.

Optimization Algorithms

SGD: Terminology



Epoch: One forward/backward pass over all the dataset

Batch: Number of training examples used in one forward/backward pass (typically 32-128)

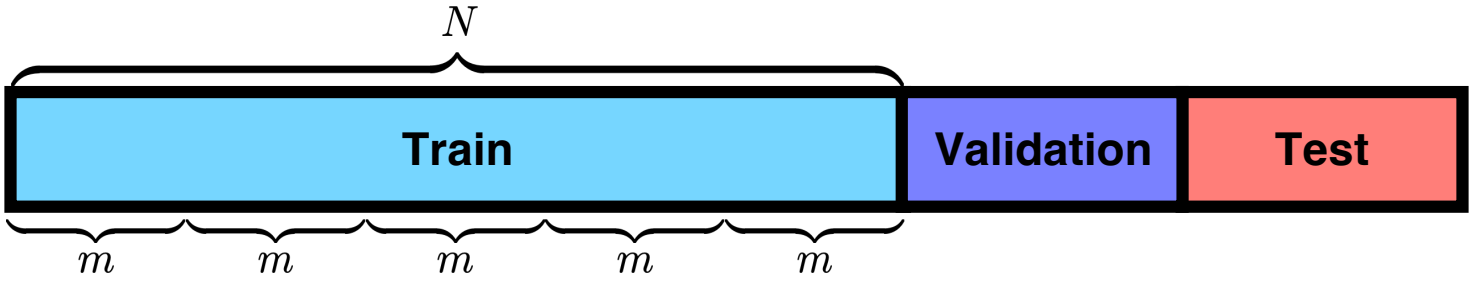
Stochastic Gradient Descent: Gradient Descent with a batch size of 1 *Why Stochastic?*

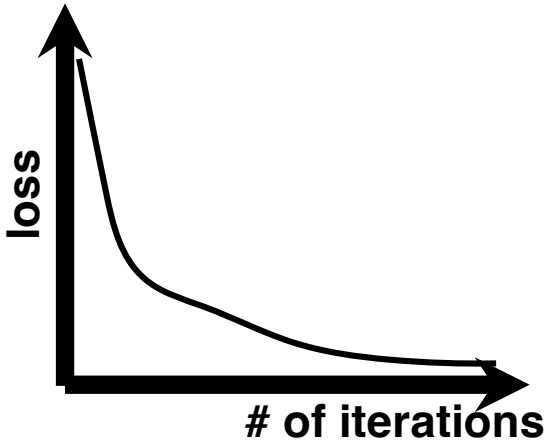
Iteration: Number of (forward/backward) passes

Example: For 30,000 training examples and a batch size of 10, it will take 3000 iterations to complete 1 epoch.

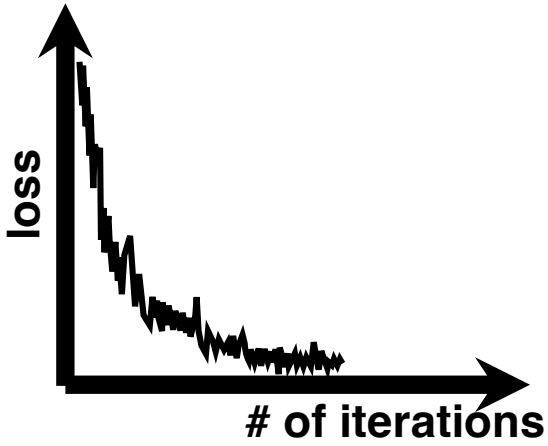
Optimzation algorithms

Batch vs Minibatch Gradient Descent





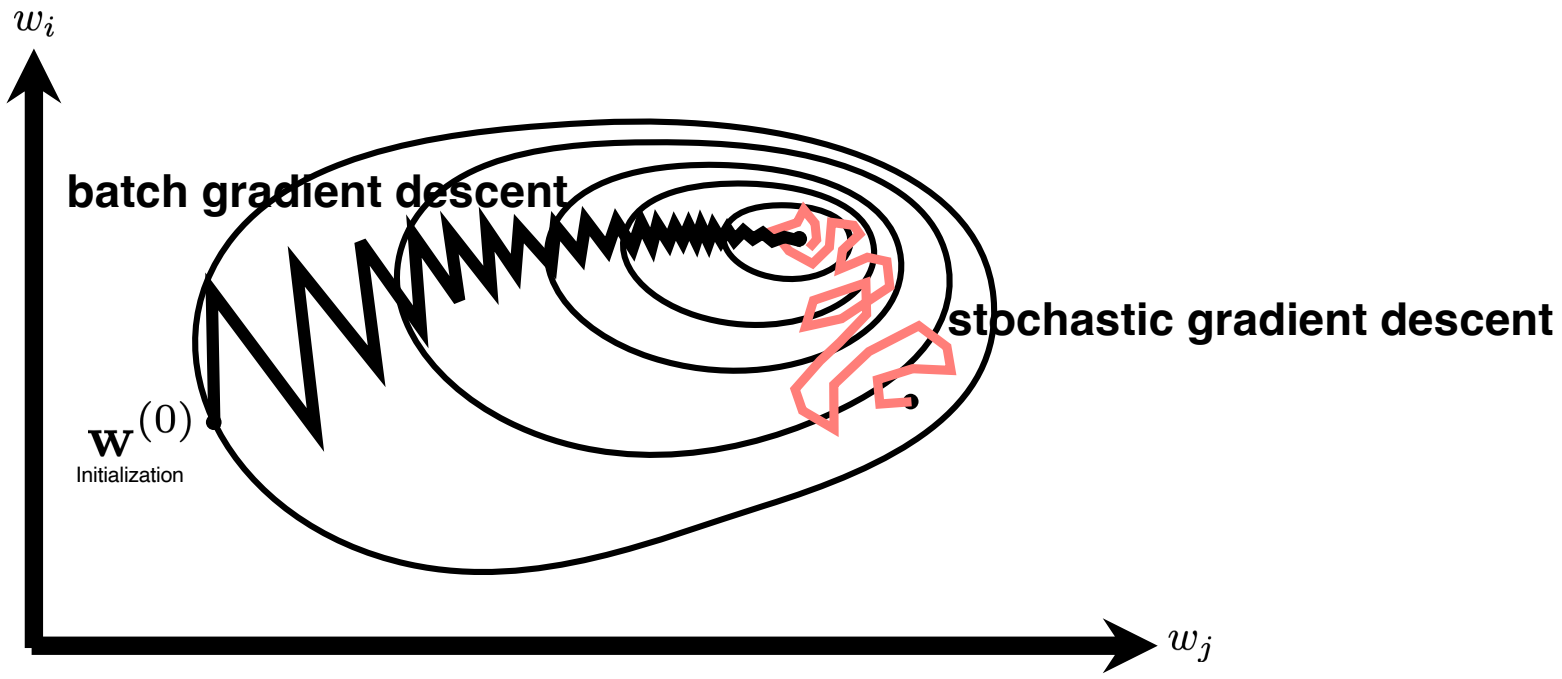
Batch Gradient Descent



Minibatch Gradient Descent

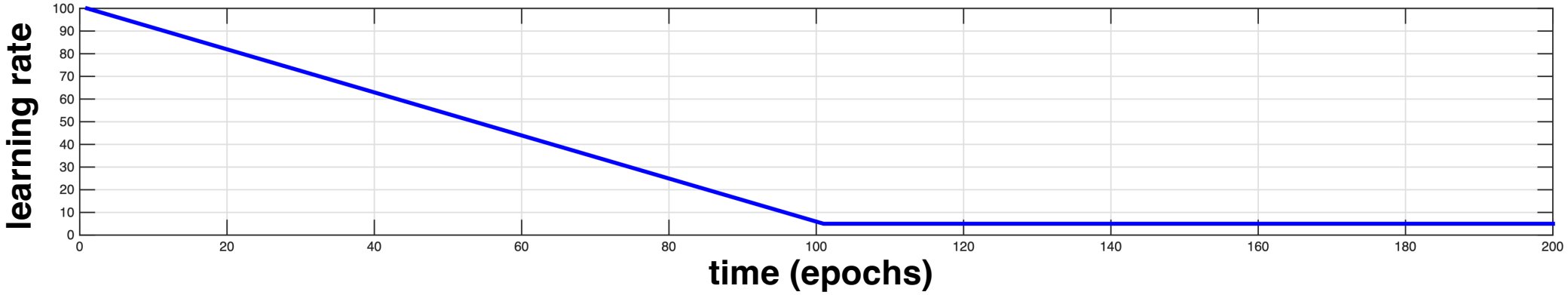
Review: First-order Methods

Zigzagging: Gradient descent does not exploit curvature information



Optimization Algorithms

SGD: Learning Rate decreases over time to encourage convergence

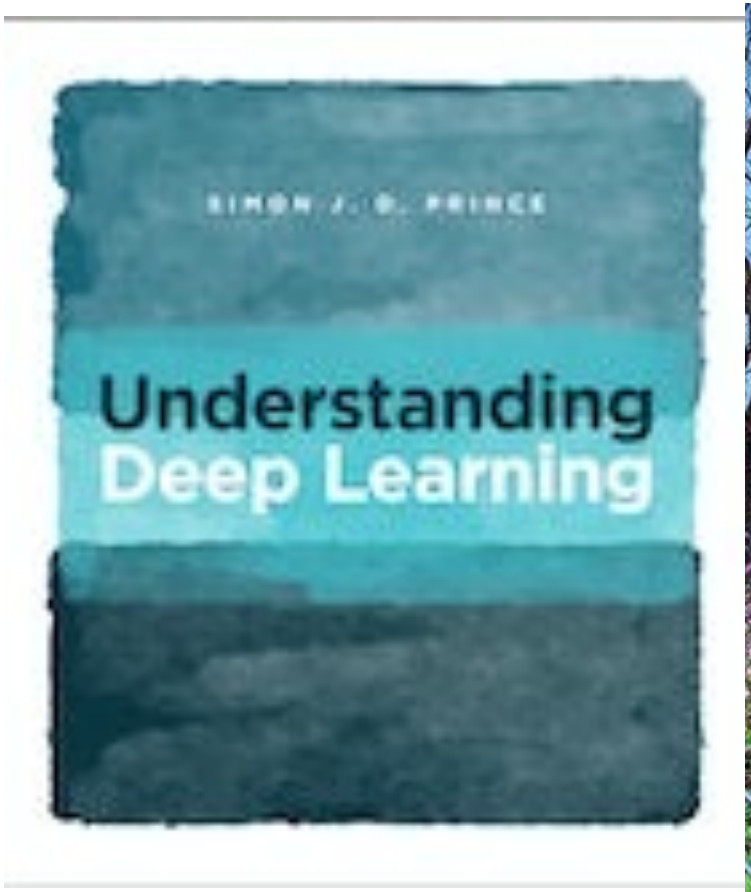


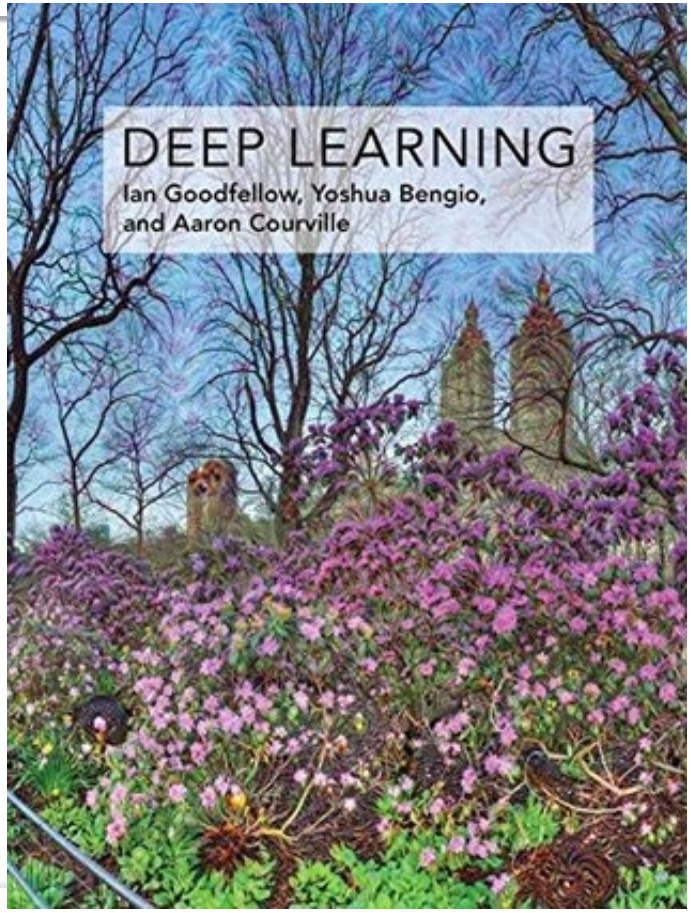
$$\epsilon_t = \begin{cases} (1 - \frac{t}{\tau})\epsilon_0 + \frac{t}{\tau}\epsilon_\tau, & t < \tau \\ \epsilon_\tau, & \text{otherwise} \end{cases}$$

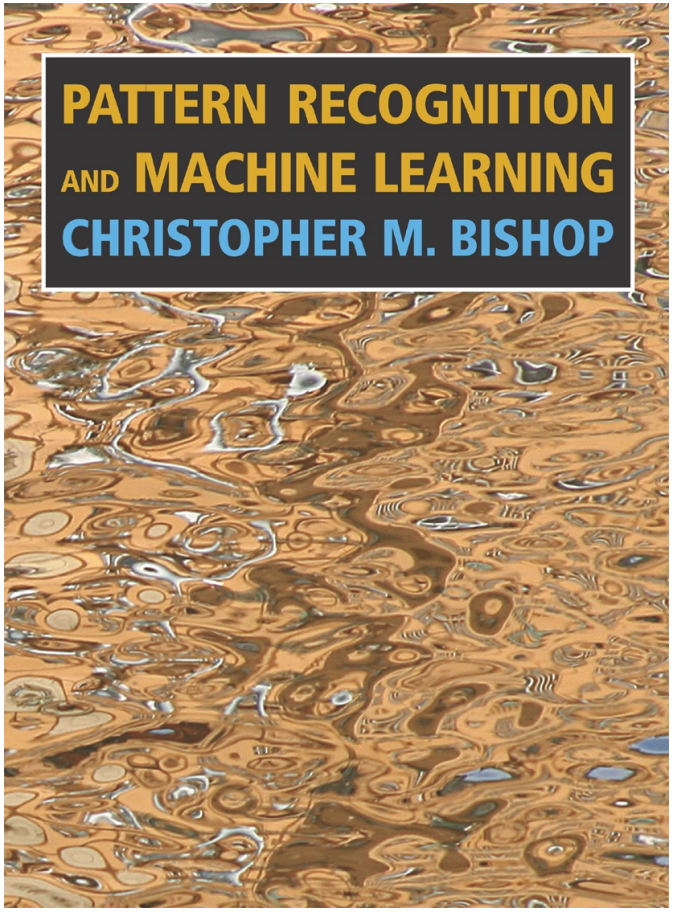
Various heuristics to choose $(\epsilon_0, \epsilon_\tau, \tau)$. Proceed with caution!

Stochastic Gradient Descent: Computation time per update does not grow with the number of training examples.

More to read…


SIMON J. D. PRINCE
Understanding
Deep Learning


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville


PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP

Let’s have a play!!!

https://github.com/slucey-cs-cmu-edu/RVSS26/blob/main/Classification_MLP_simple.ipynb

\