

Deep Learning for Vision

Instructor - Simon Lucey



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



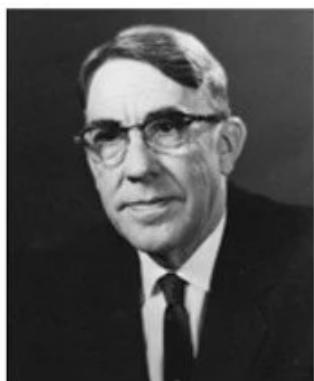
Ray Solomonoff



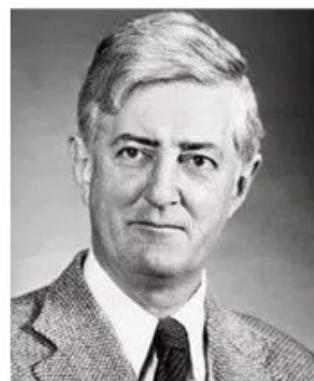
Alan Newell



Herbert Simon



Arthur Samuel



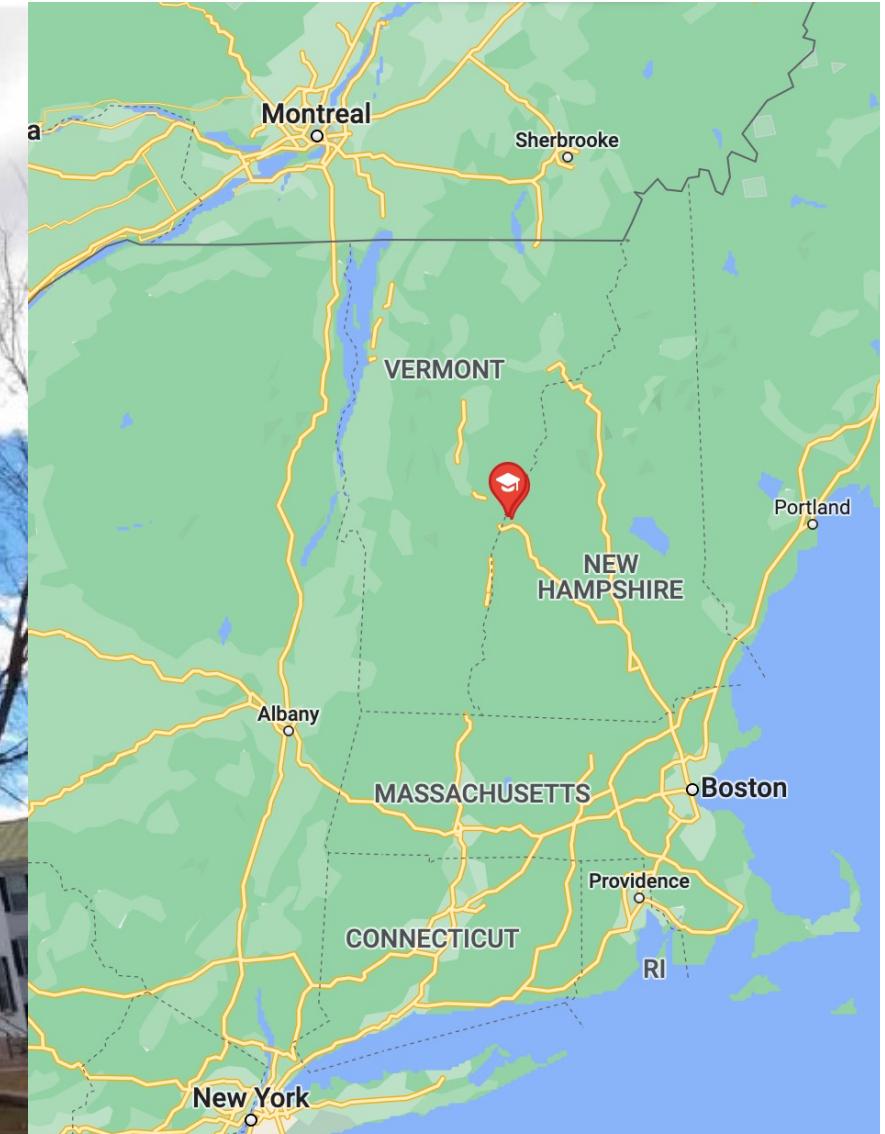
Oliver Selfridge



Nathaniel Rochester



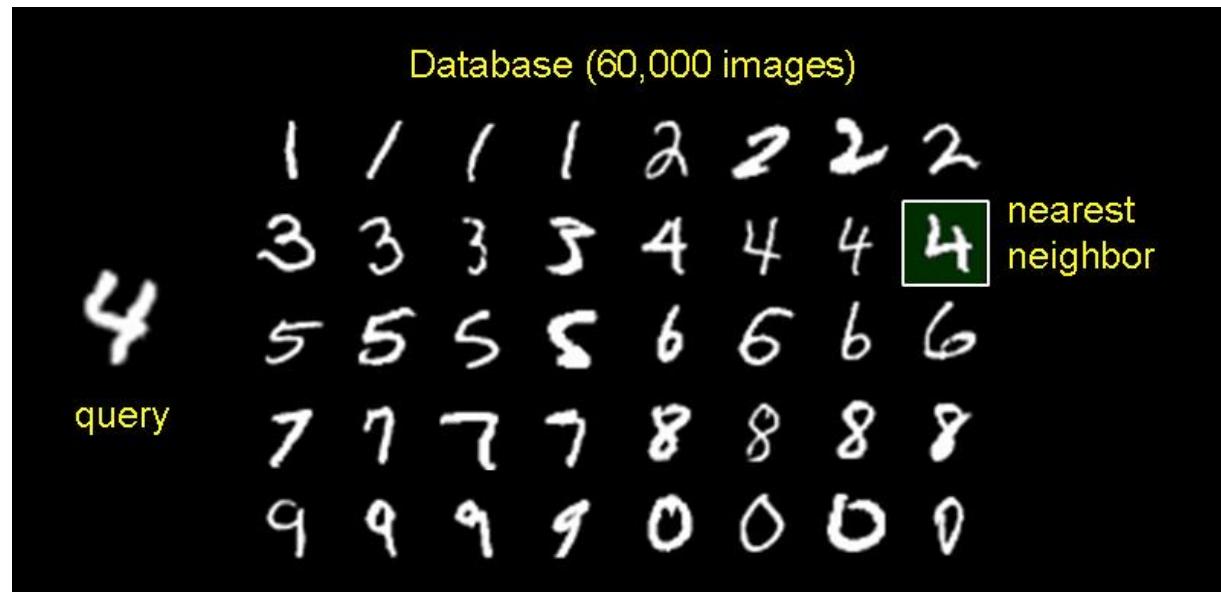
Trenchard More



Conventional programming

```
2 class Room(object):
3     def __init__(self, inventory, desc, short_desc):
4         self.inventory = inventory
5         self._n = None
6         self._s = None
7         self._e = None
8         self._w = None
9         self._desc = desc
10        self._short_desc = short_desc
11        self._gate_n = None
12        self._gate_s = None
13        self._gate_e = None
14        self._gate_w = None
15
16
17        if not isinstance(desc, str):
18            raise TypeError ("the input provided is not a string.")
19        elif not isinstance(short_desc, str):
20            raise ValueError ("the string provided is empty.")
21
22        # these set the gates
23        # they set the opposite gates, with checks to avoid recursion loops
24        def set_n(self, other):
25            if not isinstance(other, Room) or not other:
26                raise TypeError ("Room is not None or an instance of Room")
```

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.



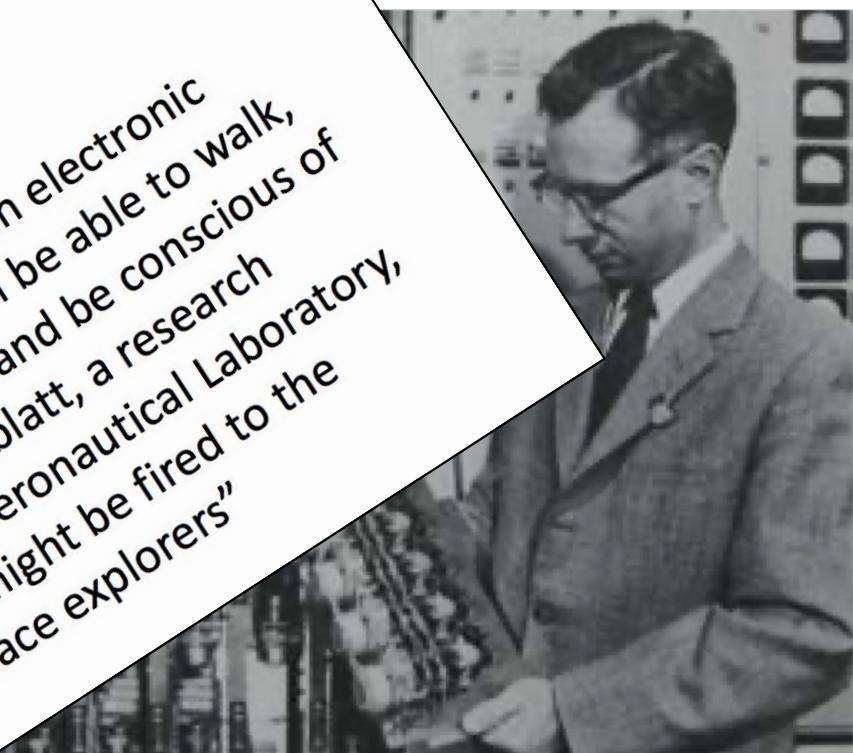
“Frank Rosenblatt”

The New York Times

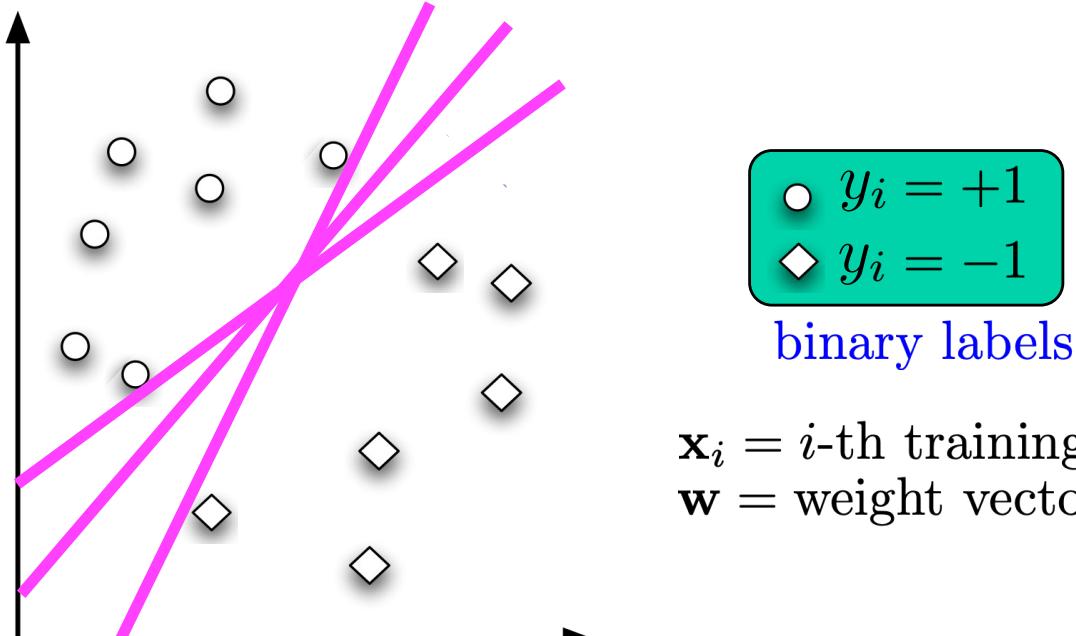
NEW NAVY DEVICE LEARNS BY DOING

July 8, 1958

"The Navy revealed the embryo of an electronic computer today that it expects will be able to 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 stars as mechanical space explorers"



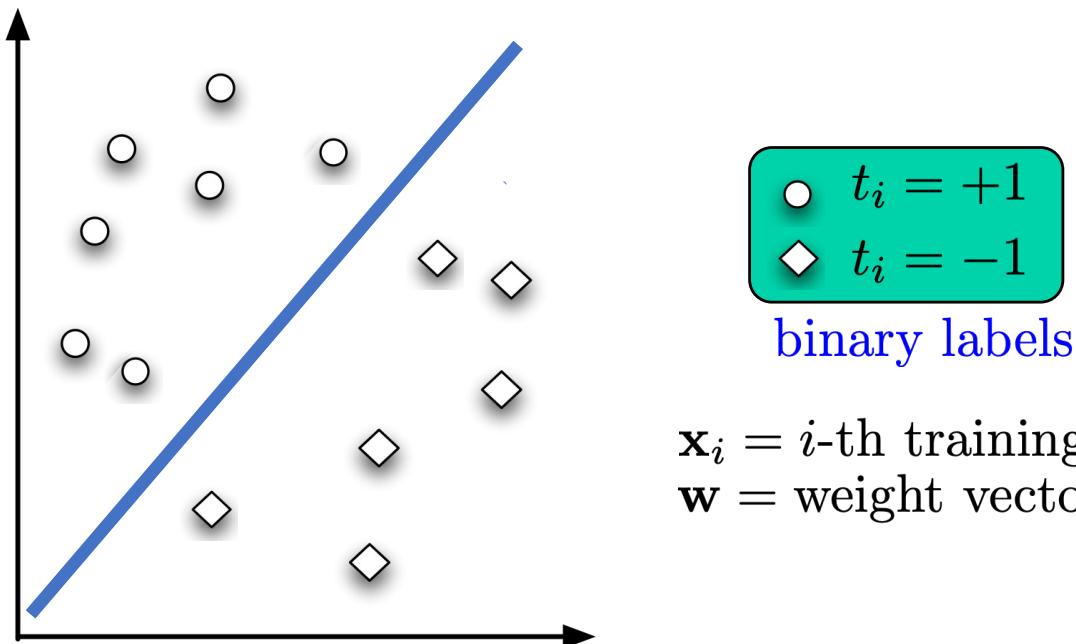
Perceptron = Linear Discriminant



\mathbf{x}_i = i -th training example
 \mathbf{w} = weight vector

$$\underset{\mathbf{w}}{\operatorname{arg\,gmin}} \sum_{i=1}^N \max_i(0, \mathbf{x}_i^T \mathbf{w})$$

Perceptron = Linear Discriminant

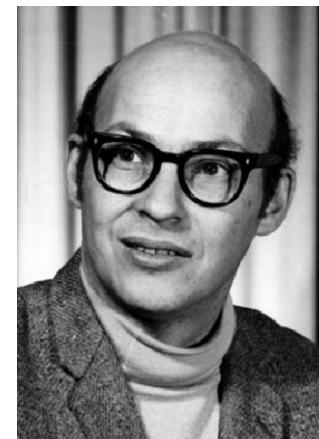
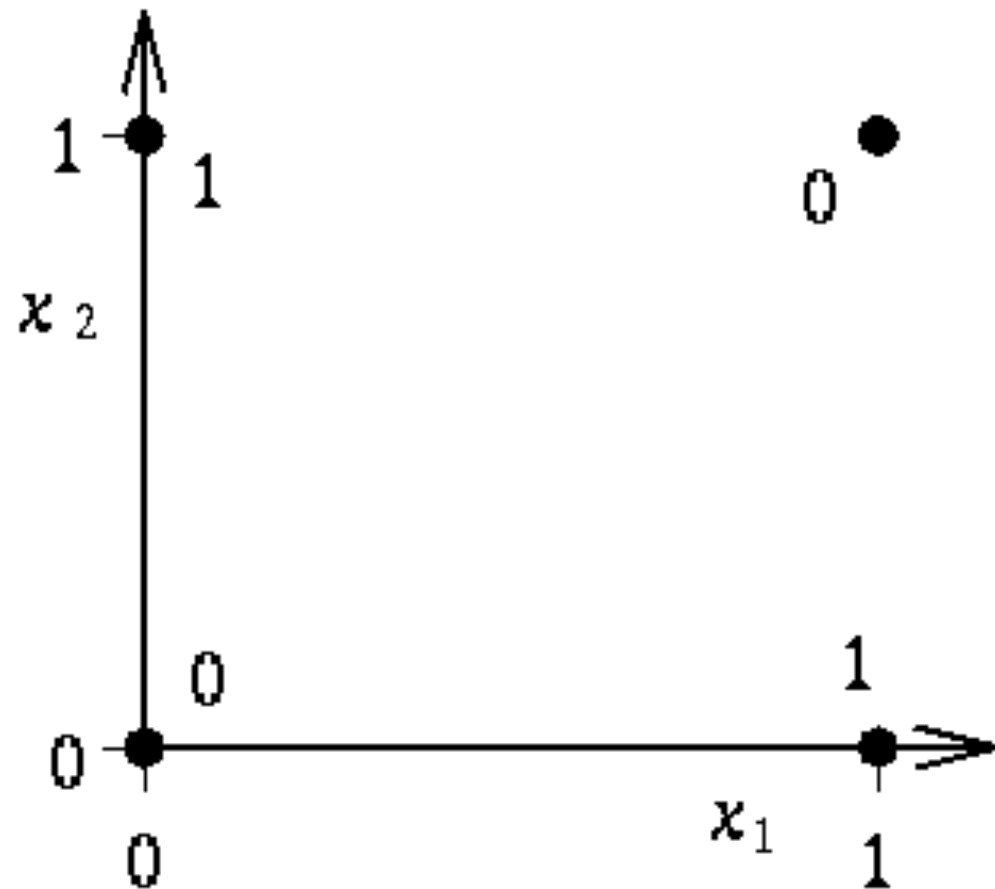


$\mathbf{x}_i = i\text{-th training example}$
 $\mathbf{w} = \text{weight vector}$

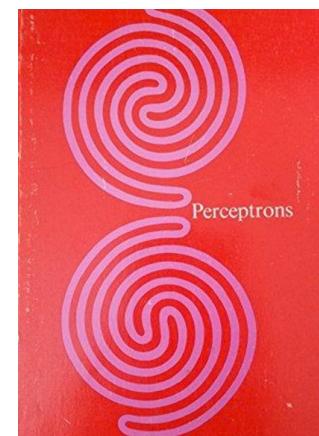
$$\arg \min_{\mathbf{w}} \sum_{i=1}^N \mathcal{L}(y_i \cdot \mathbf{x}_i^T \mathbf{w}) + \Omega(\mathbf{w})$$

“regularizer”

XORs, Minsky and the AI Winter



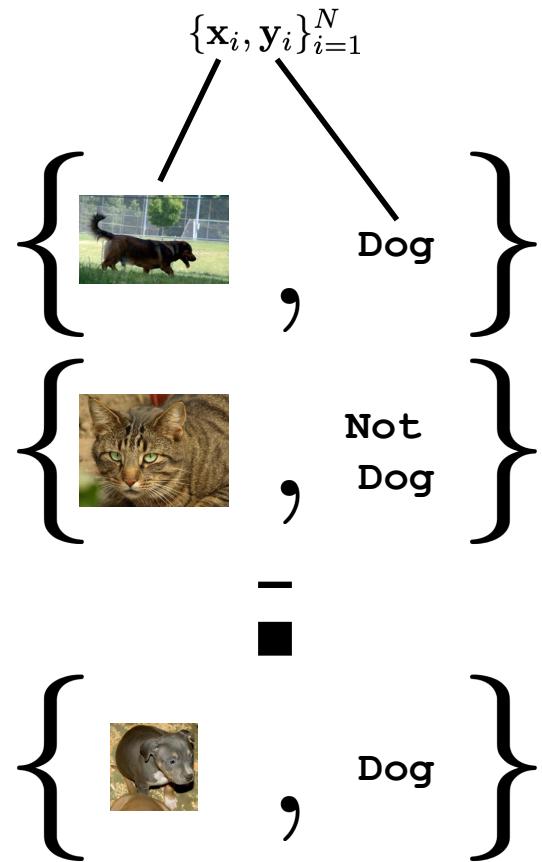
“Marvin Minsky”



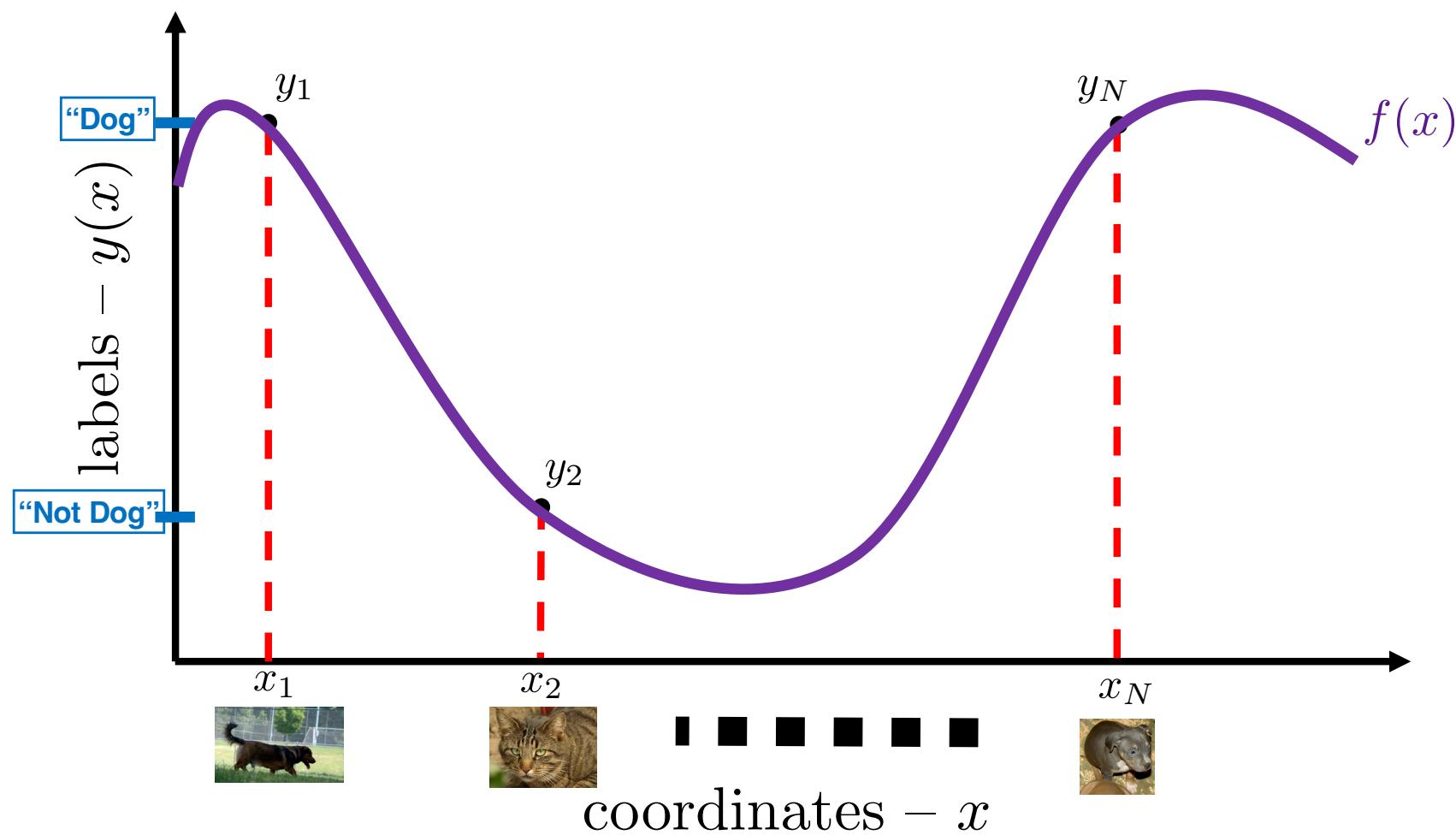
Today

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

Visual Learning

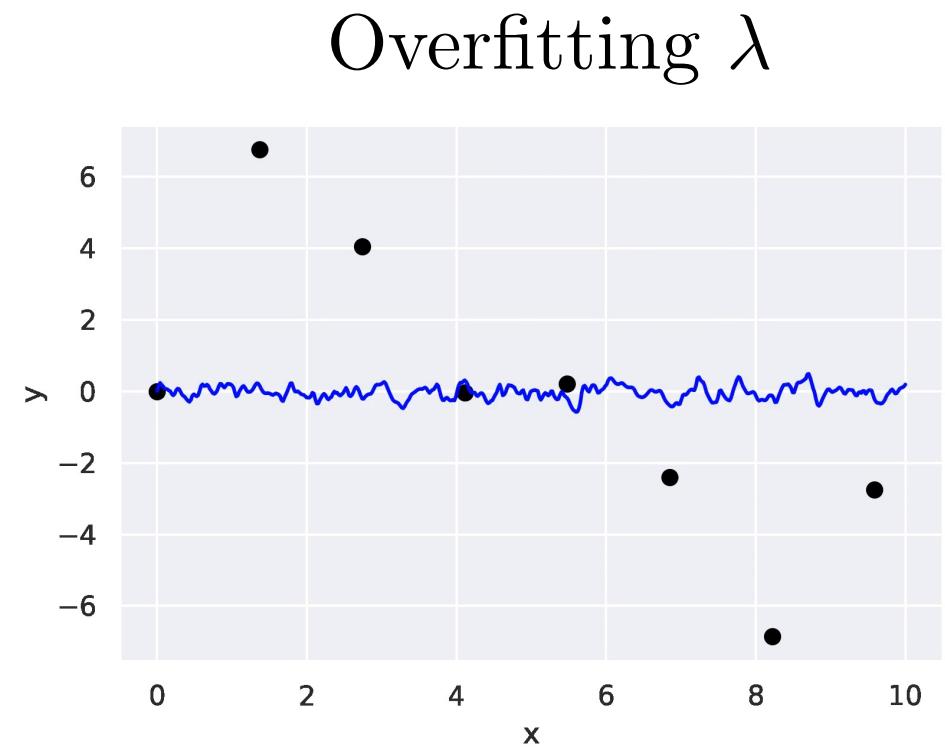
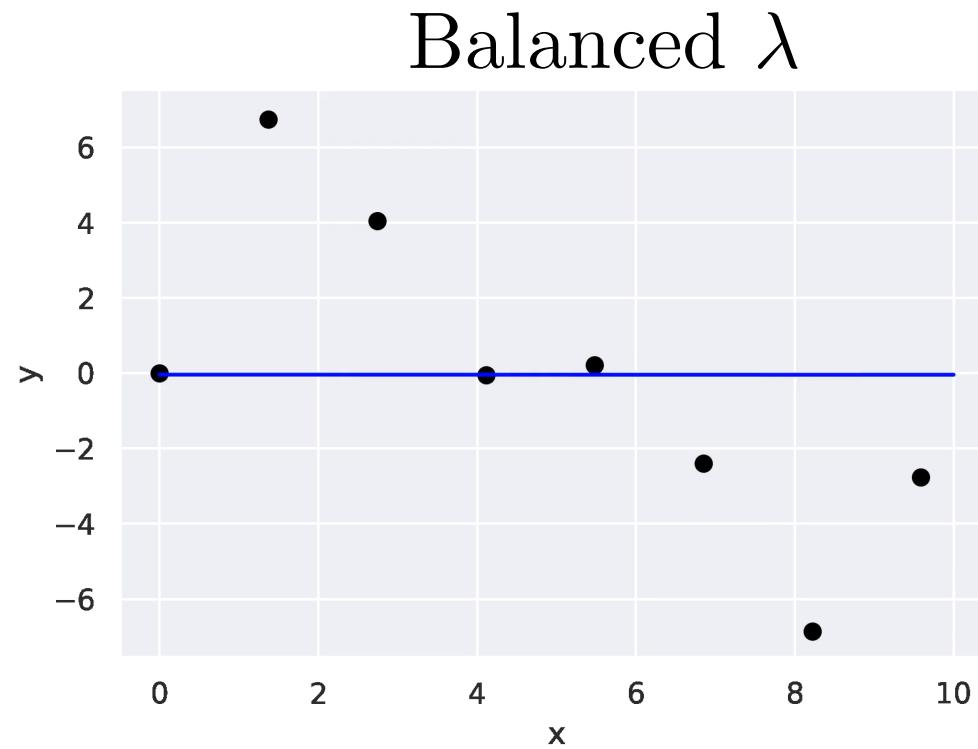


Visual Learning is Curve Fitting!!!

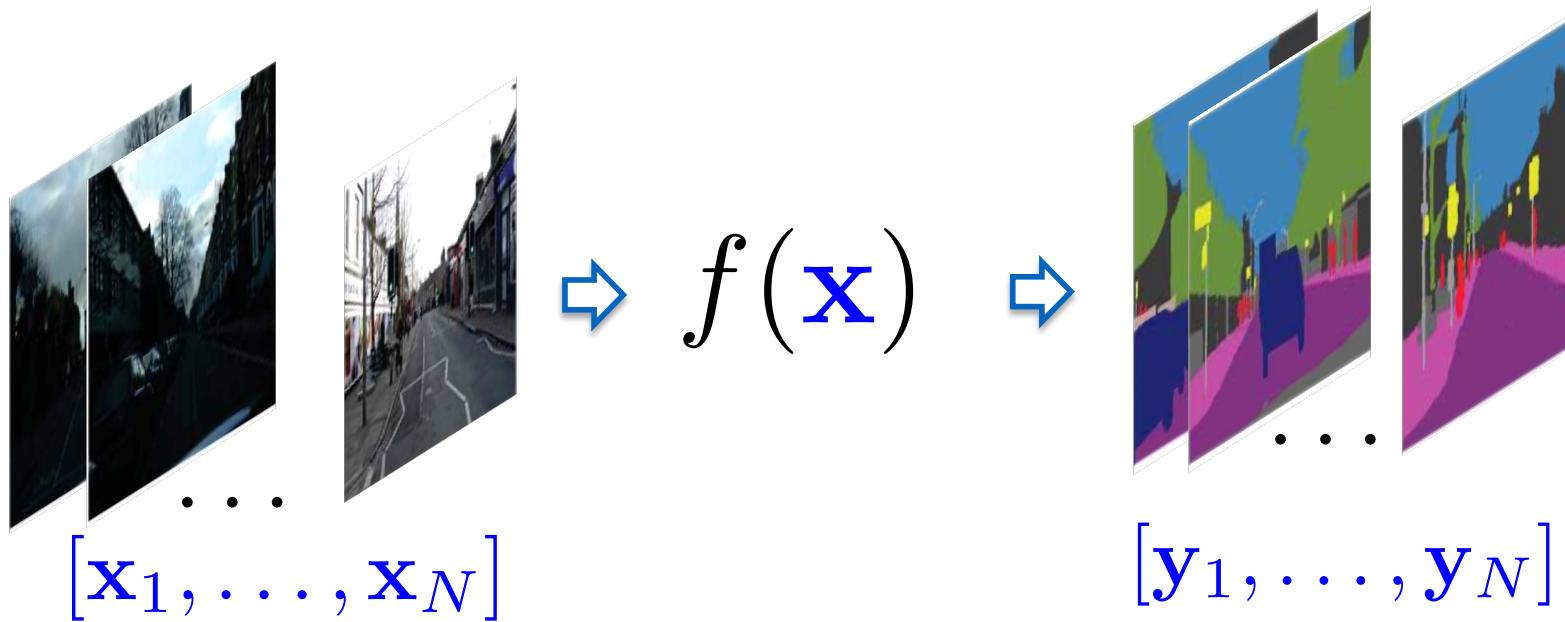


Spline Smoothing/Regression

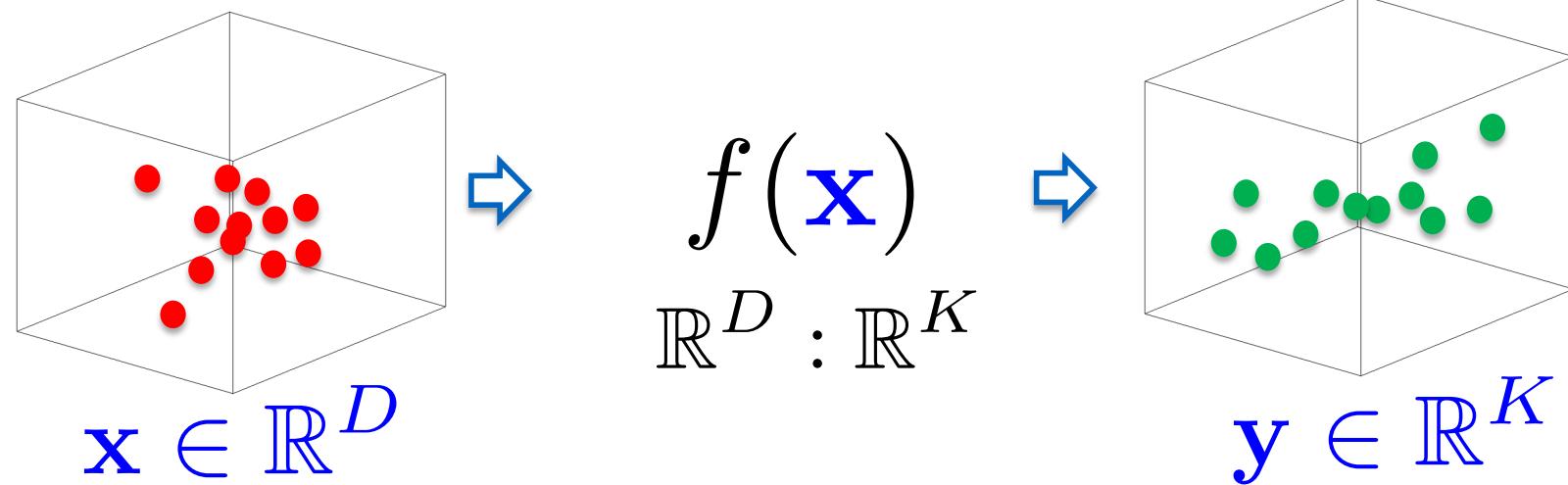
Memorization vs. Generalization



Multi-dimensional Sampling?

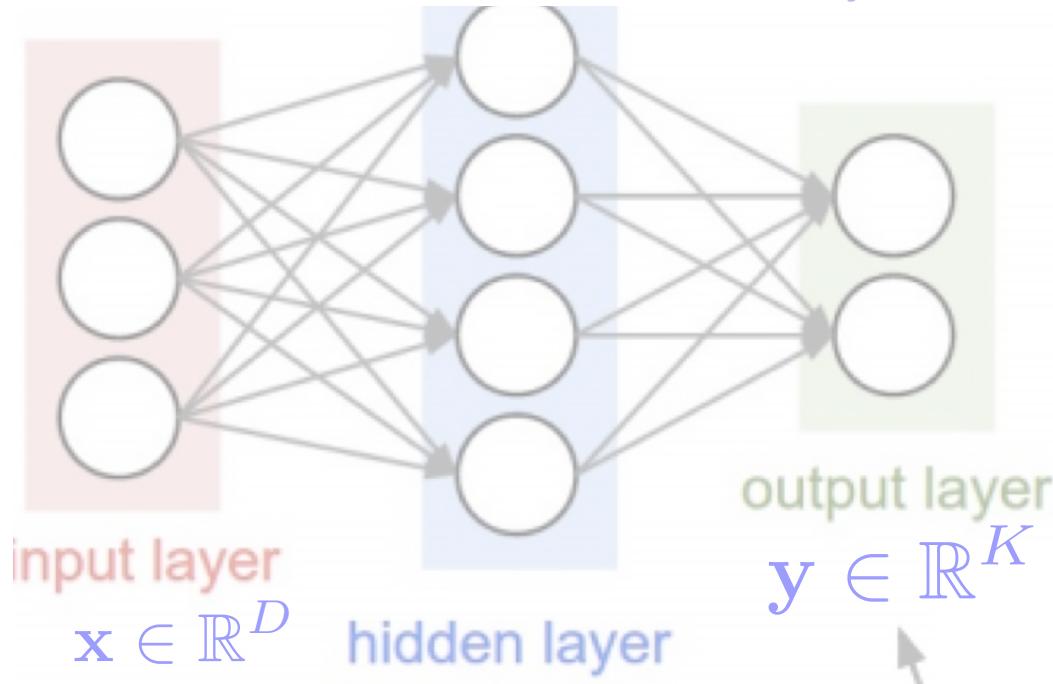


Multi-dimensional Sampling?



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

“hidden layer”



$$f(\mathbf{x}) \approx \theta_{(K\times M)}\eta(\textcolor{red}{W}_{(M\times D)}\mathbf{x}+\textcolor{red}{\delta}_{(M\times 1)})$$

Shallow vs Deep Network

- A shallow learner has only one hidden unit,

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

- A deep learner has more than one hidden unit,

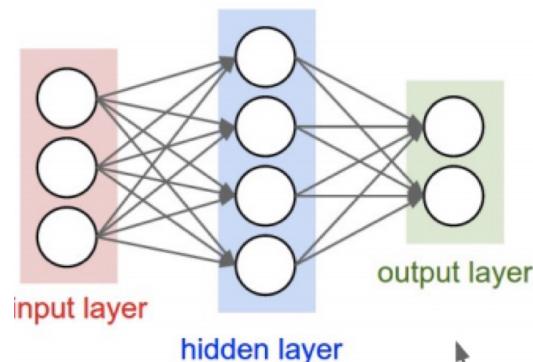
$$\theta \eta(\mathbf{W}^{(1)} \eta(\mathbf{W}^{(0)} \mathbf{x} + \boldsymbol{\delta}^{(0)}) + \boldsymbol{\delta}^{(1)})$$



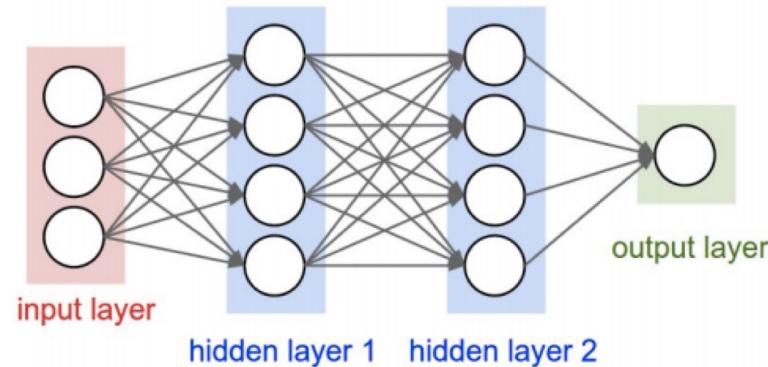
“Geoffrey Hinton”

Shallow vs. Deep Layers

“shallow”



“deep”



“2-layer neural net,” or
“1-hidden-layer neural net”

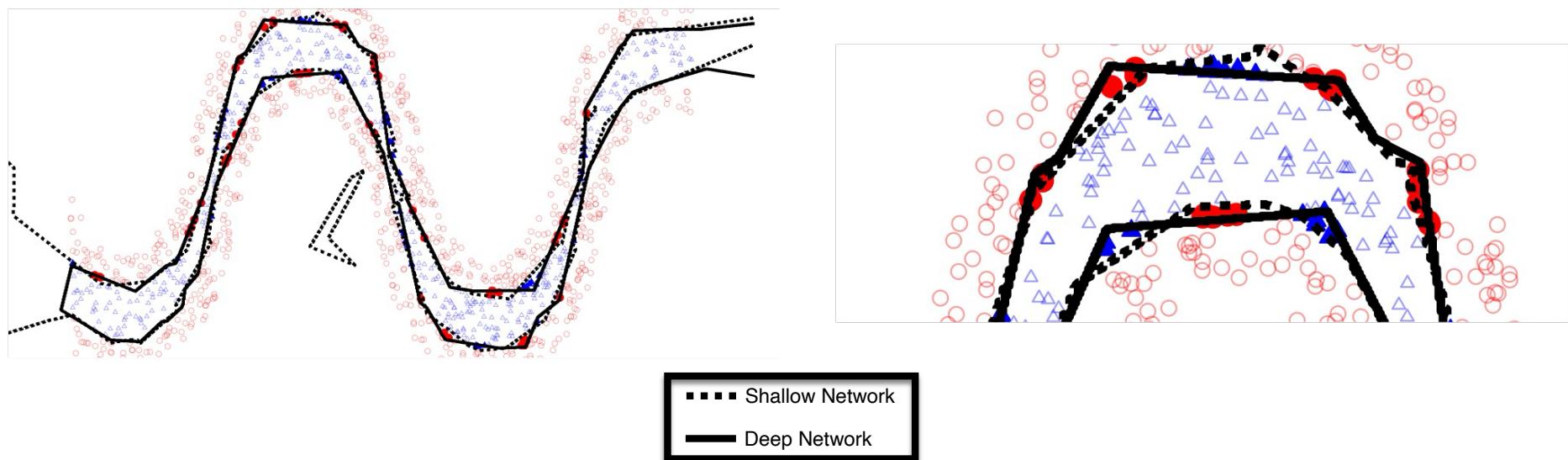
“Fully-connected” layers

“3-layer neural net,” or
“2-hidden-layer neural net”

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

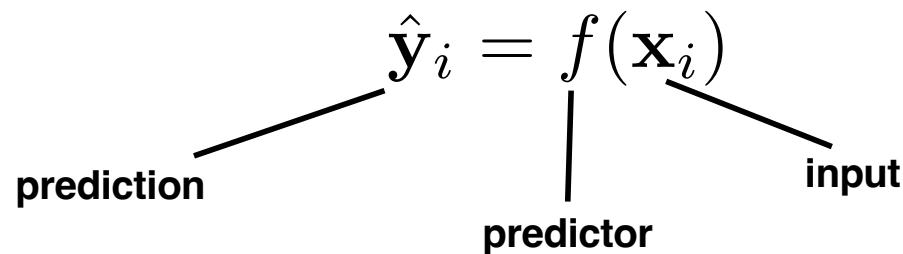
What ratio of training, validation, and test?

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

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

Example Input/Output Pairs

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



Loss Functions

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

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

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

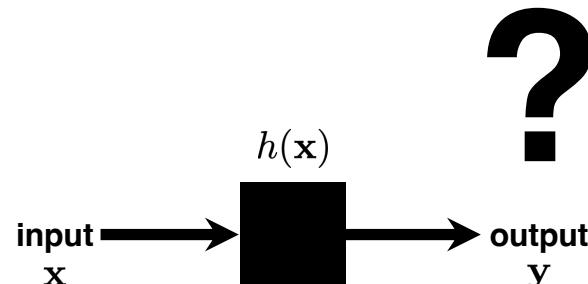
Example

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

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

Visual Learning VS Pure Optimization

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



Visual Learning VS Pure Optimization

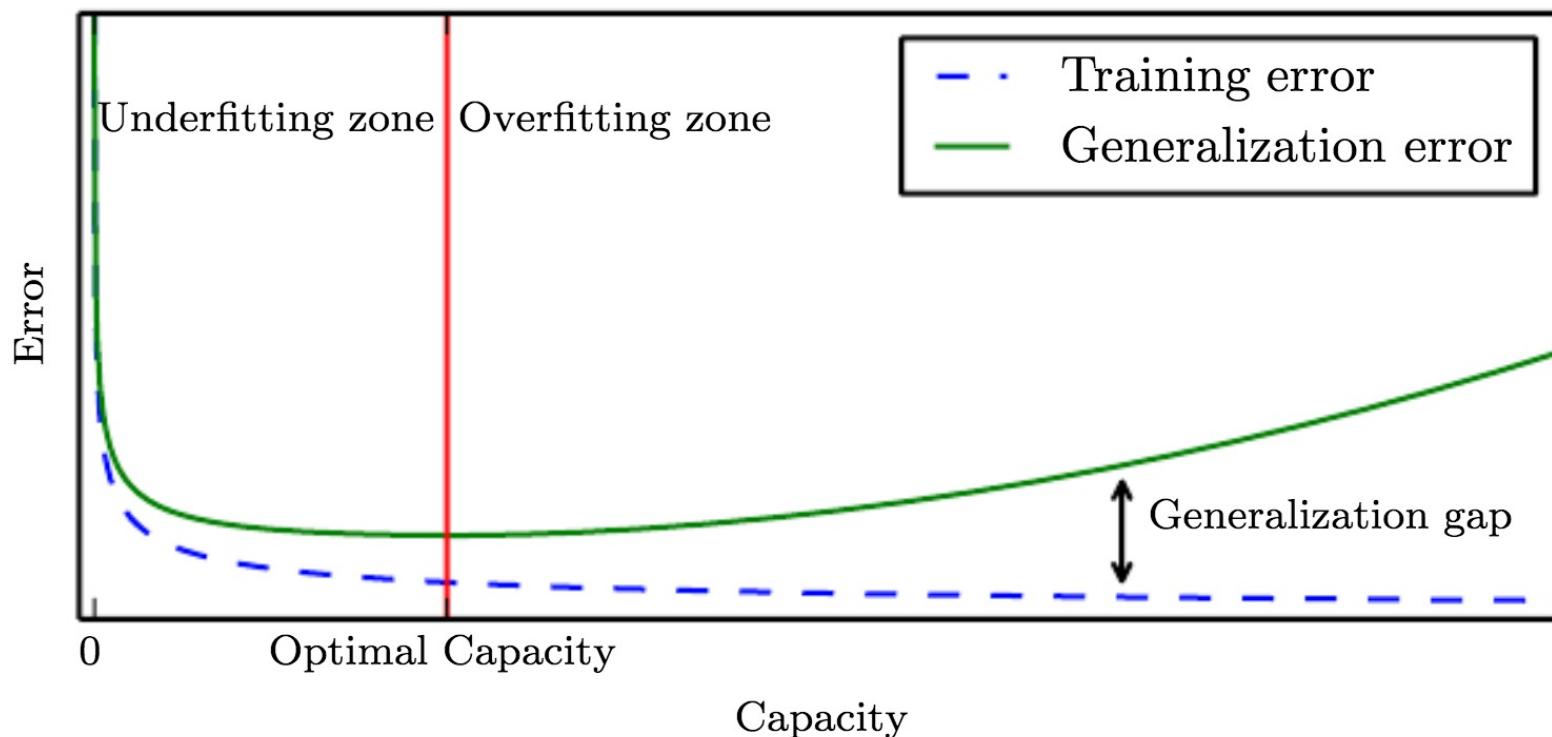
Optimize on Training Set but Evaluate on Test Set

	Train	Validation	Test	
System A				
Training Set Error:	2%	22%	22%	
Test Set Error:	24%	24%	34%	
	“overfitting” high variance	“underfitting” high bias	high bias and high variance	low bias and low variance

Key concern for high capacity learning system

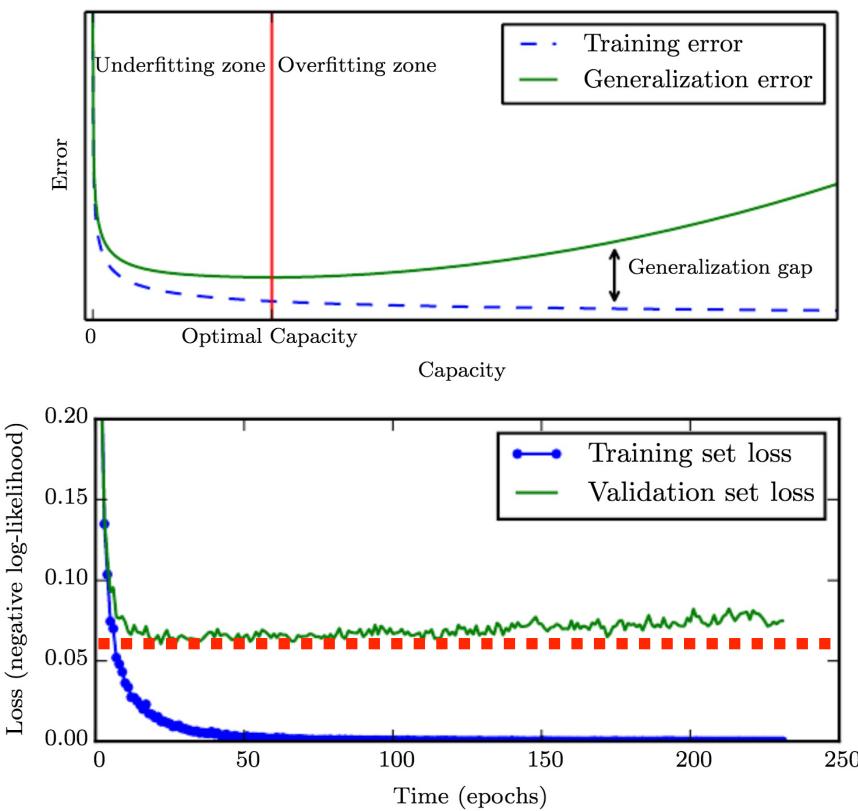
Visual Learning VS Pure Optimization

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



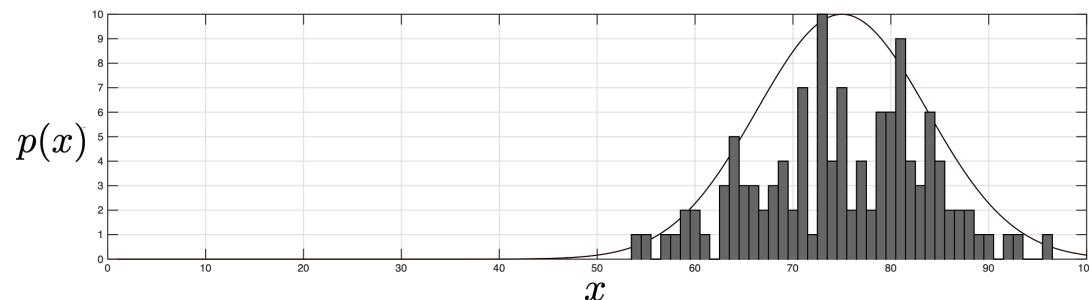
Visual Learning VS Pure Optimization

Termination Strategy: Early Stopping



Visual Learning VS Pure Optimization

Empirical Risk Minimization



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

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

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

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

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

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

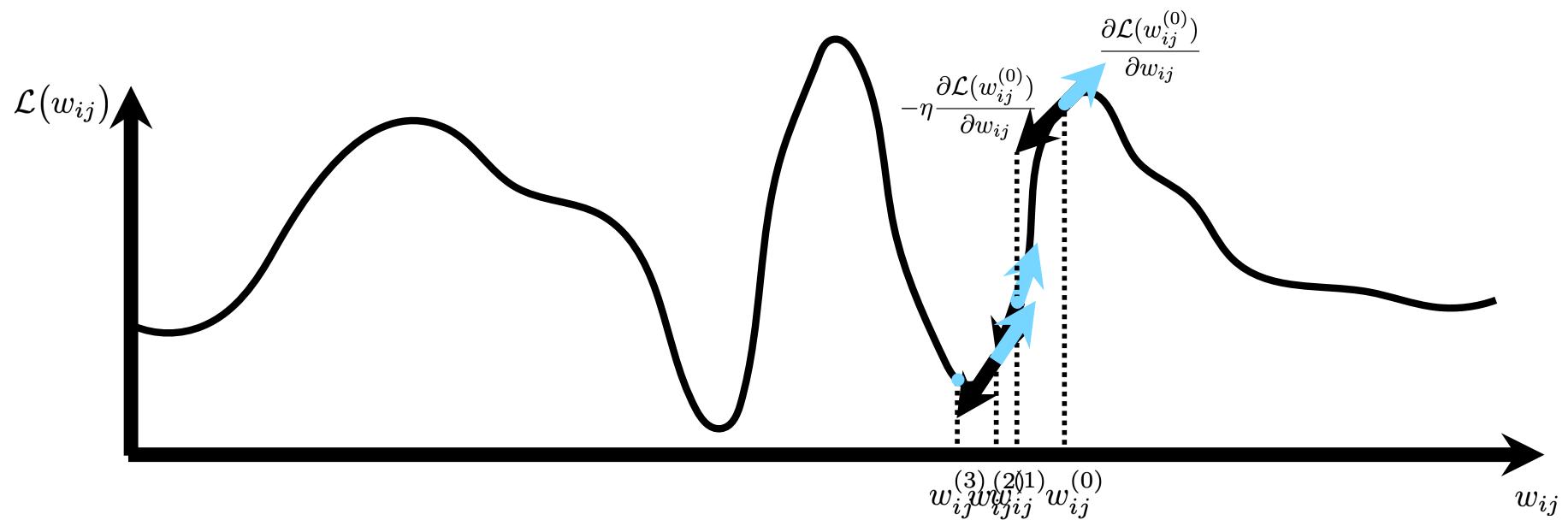
Today

- 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

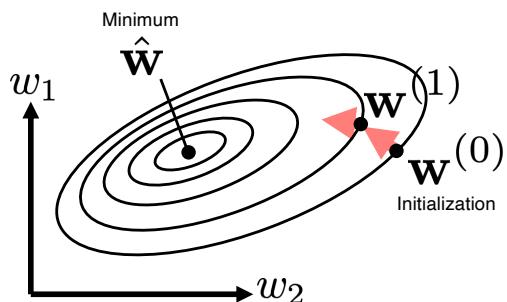
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial \mathcal{L}(w_{ij}^{(t)})}{\partial w_{ij}}$$



Review: First-order Methods

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

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

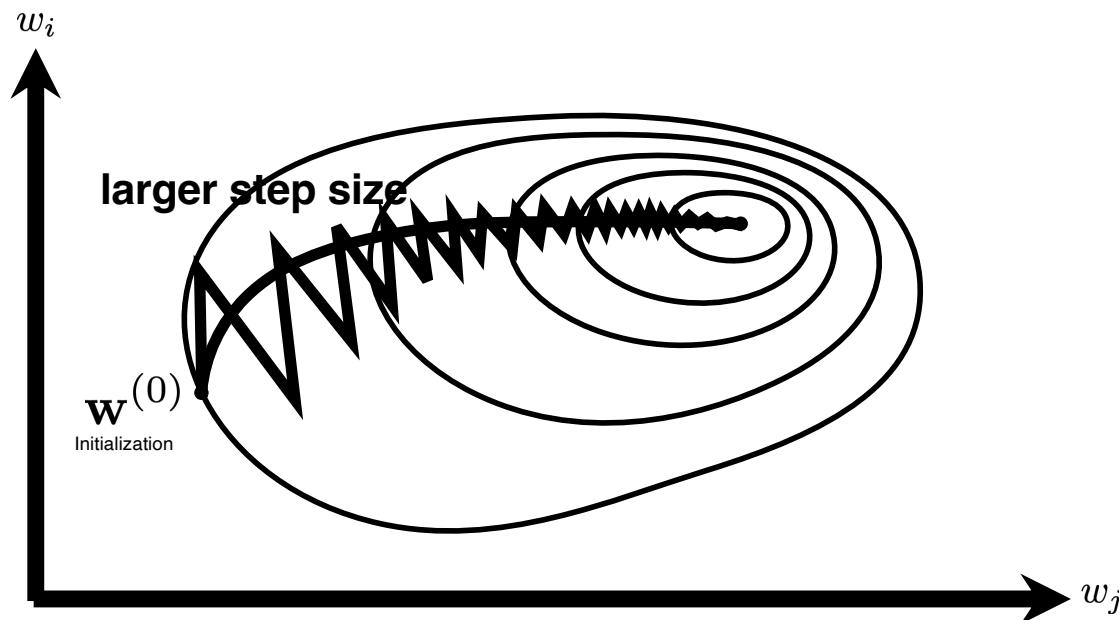


Learning Rate

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^{(t)})$$
$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{11}} & \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{12}} & \dots & \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{UV}} \end{pmatrix} \quad \text{Jacobian}$$

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

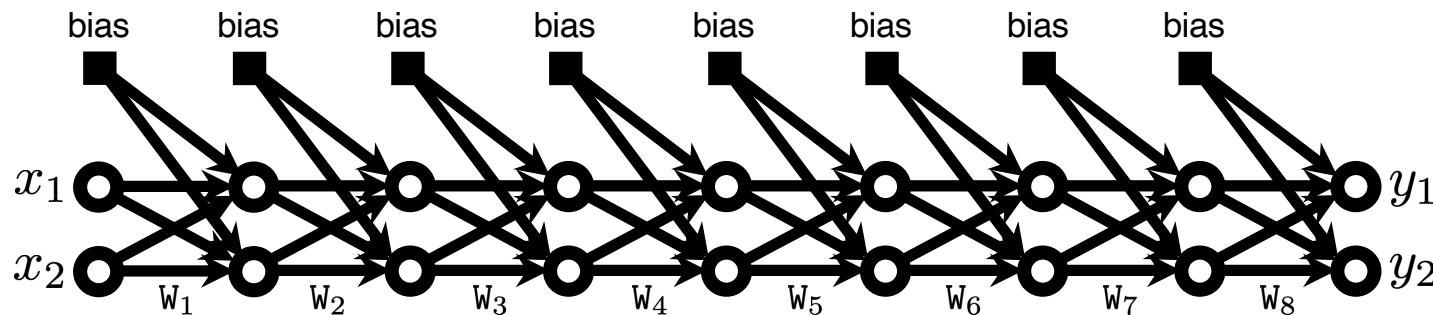
Gradient Descent: Ill-conditioning



Photo: Bob Wick

Optimization Challenges

Vanishing/Exploding Gradients in Deep Networks



$$\hat{\mathbf{y}} = W_8 \ W_7 \ W_6 \ W_5 \ W_4 \ W_3 \ W_2 \ W_1 \ \mathbf{x} \quad (\text{assume no nonlinearities})$$

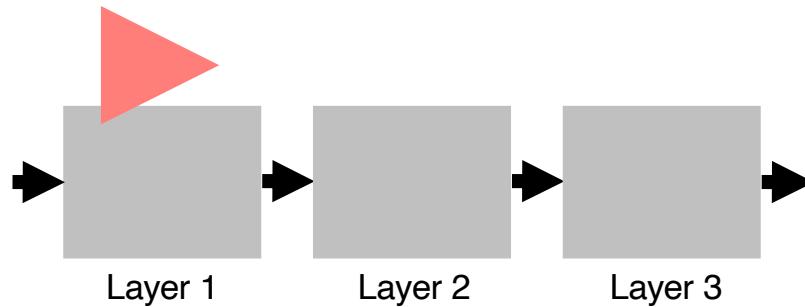
$$\hat{\mathbf{y}} = W^8 \mathbf{x} \quad (\text{assume tied weights})$$

$$W_i = W = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \quad W^8 = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^8 = \begin{bmatrix} 25.6289 & 0 \\ 0 & 25.6289 \end{bmatrix} \quad \text{exploding}$$

$$W_i = W = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad W^8 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^8 = \begin{bmatrix} 0.0039 & 0 \\ 0 & 0.0039 \end{bmatrix} \quad \text{vanishing}$$

Neural Network Training

Forward Propagation, Back Propagation, Gradient Descent



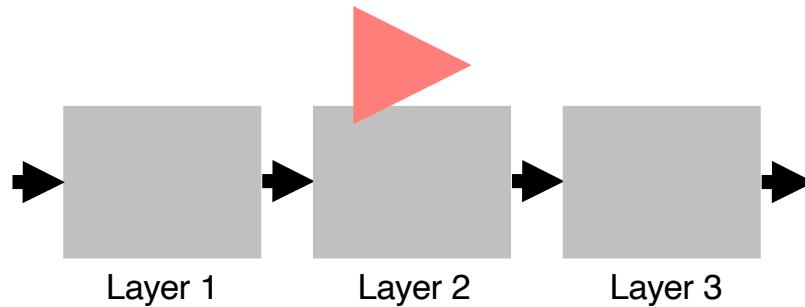
Step 1: Compute loss on training batch (Forward Pass)

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



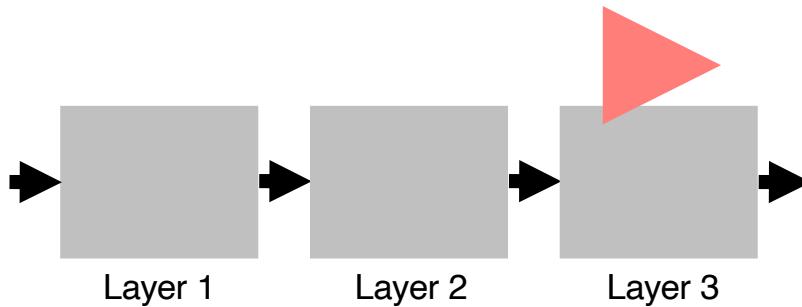
Step 1: Compute loss on training batch (Forward Pass)

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



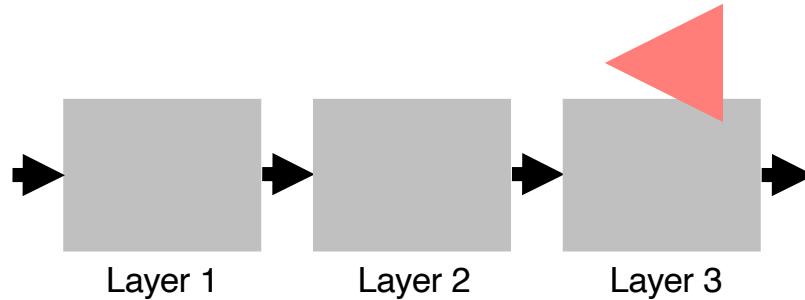
Step 1: Compute loss on training batch (Forward Pass)

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



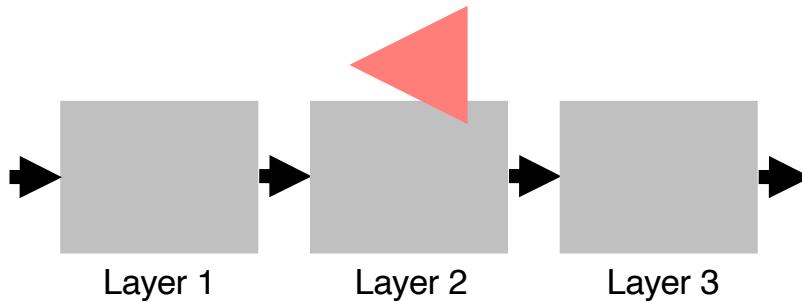
Step 1: Compute loss on training batch (Forward Pass)

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



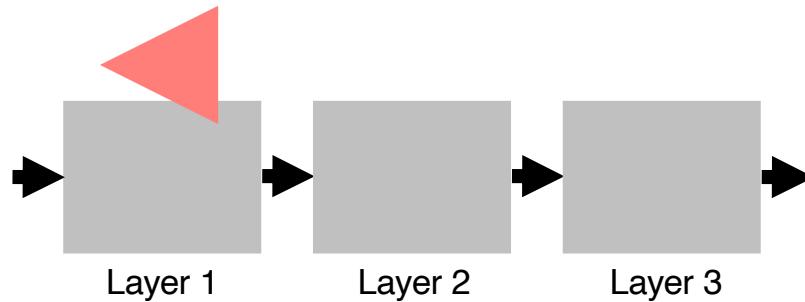
Step 1: Compute loss on training batch (Forward Pass)

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



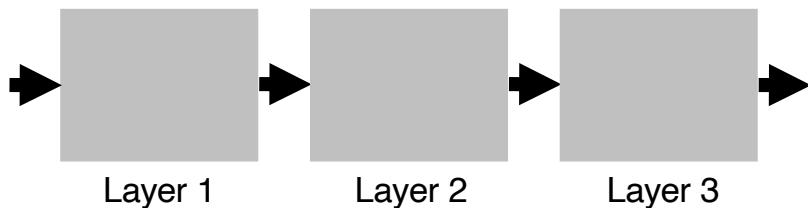
Step 1: Compute loss on training batch (Forward Pass)

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)

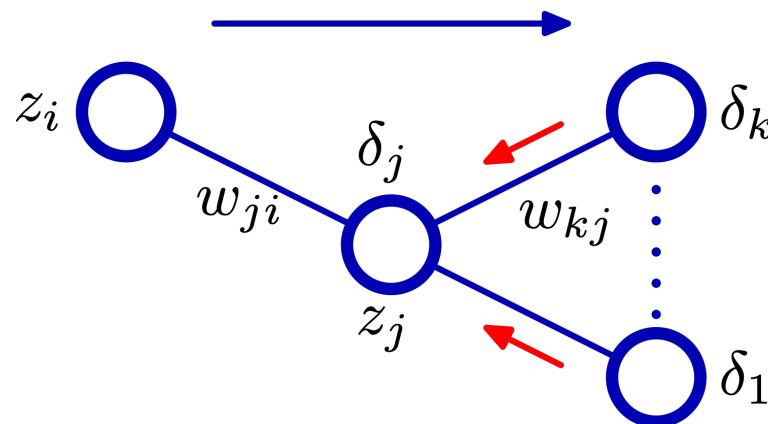
Step 2: Compute gradients with respect to weights (Backward Pass)

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

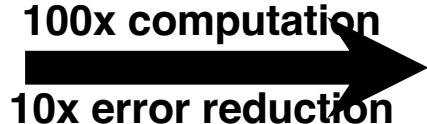
Gradient Descent for Classification

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

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: $\frac{\sigma}{\sqrt{n}}$

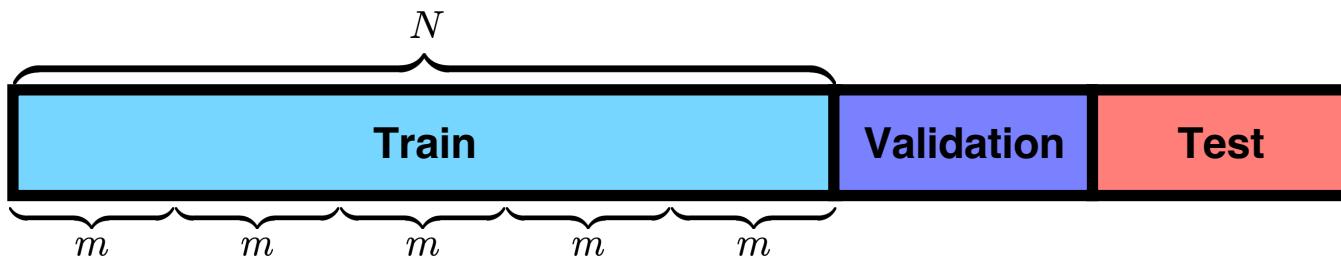
100 samples  10,000 samples

100x computation
10x error reduction

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

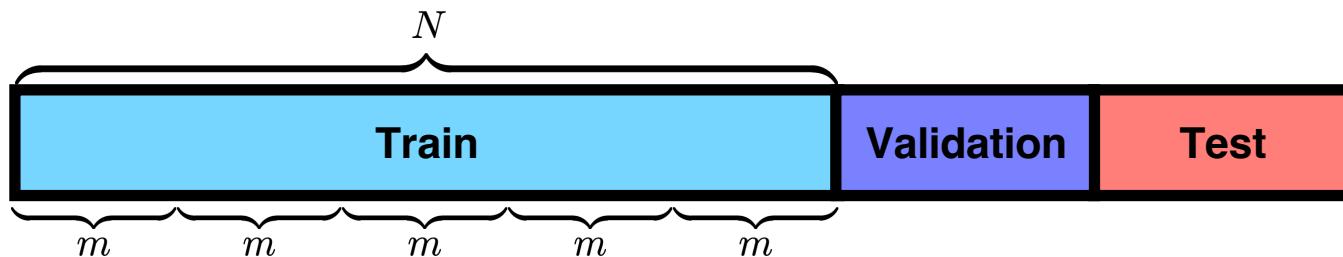


Algorithm: Stochastic Gradient Descent (SGD) update at training iteration t

Input: Learning rate ϵ_t , initial parameter 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)

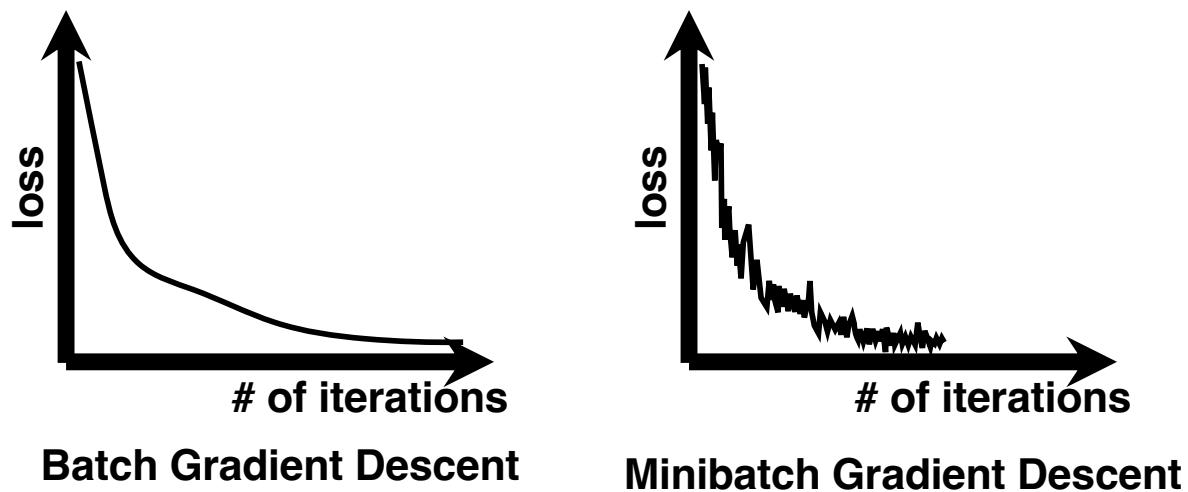
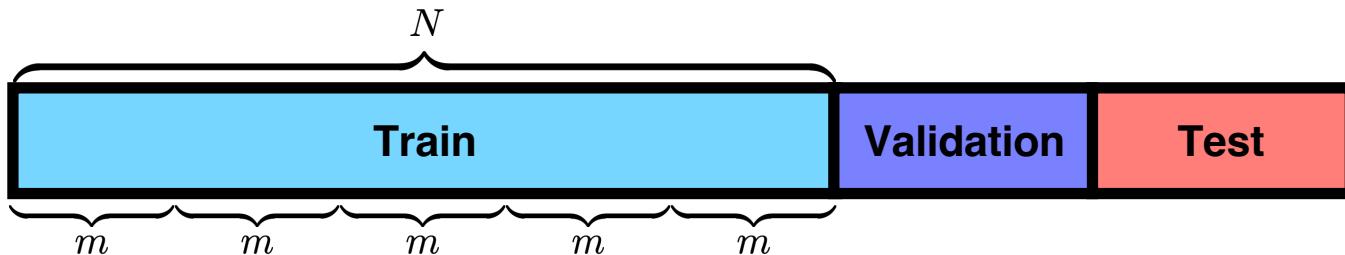
Stochastic Gradient Descent: Gradient Descent with a batch size of 1 *Why Stochastic?*

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.

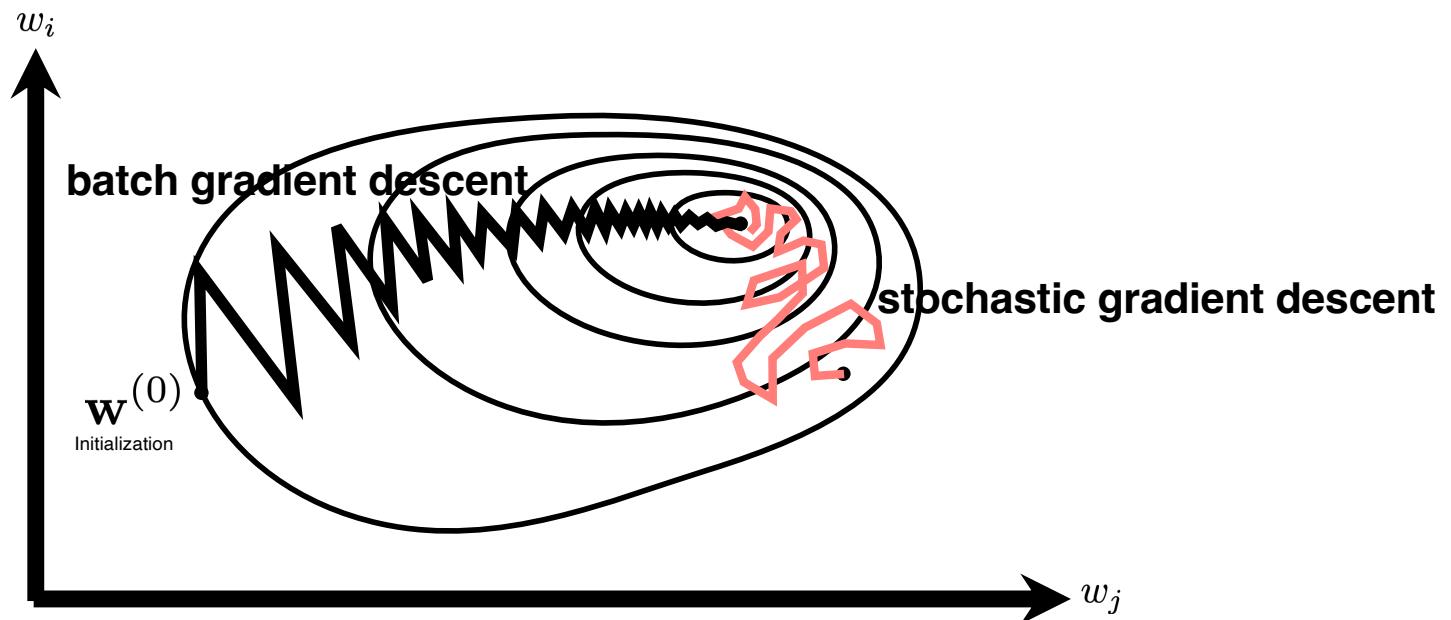
Optimization algorithms

Batch vs Minibatch Gradient Descent



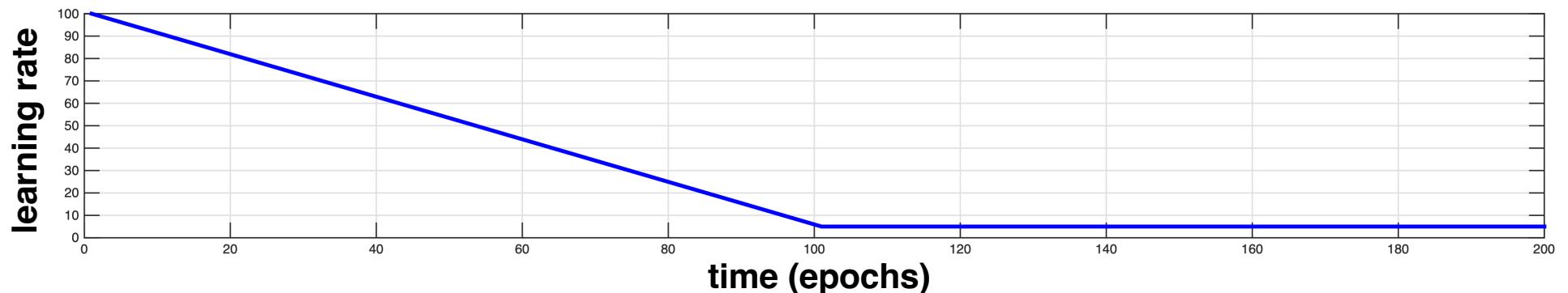
Review: First-order Methods

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...

