
Deep Learning

Part 2: Backpropagation

Nagiza F. Samatova, samatova@csc.ncsu.edu

Professor, Department of Computer Science
North Carolina State University

Senior Scientist, Computer Science & Mathematics Division
Oak Ridge National Laboratory

Outline

- **How to train a NN: High-Level Idea**
- **What is Backpropagation:**
 - **Forward & Backward Passes**
- **Understanding Gradient**
- **Calculating Node Deltas**
 - **Output Node: Quadratic & cross-entropy error functions**
 - **Interior Node**
- **Derivatives of the Activation Function**
- **Tuning NN**
 - **Convergence**
 - **Backpropagation weight update**
 - **Learning rate & momentum**
- **Applying Backpropagation**
 - **Batch and Online training**
 - **Gradient and Stochastic Gradient Descent**
 - **Nesterov Momentum**

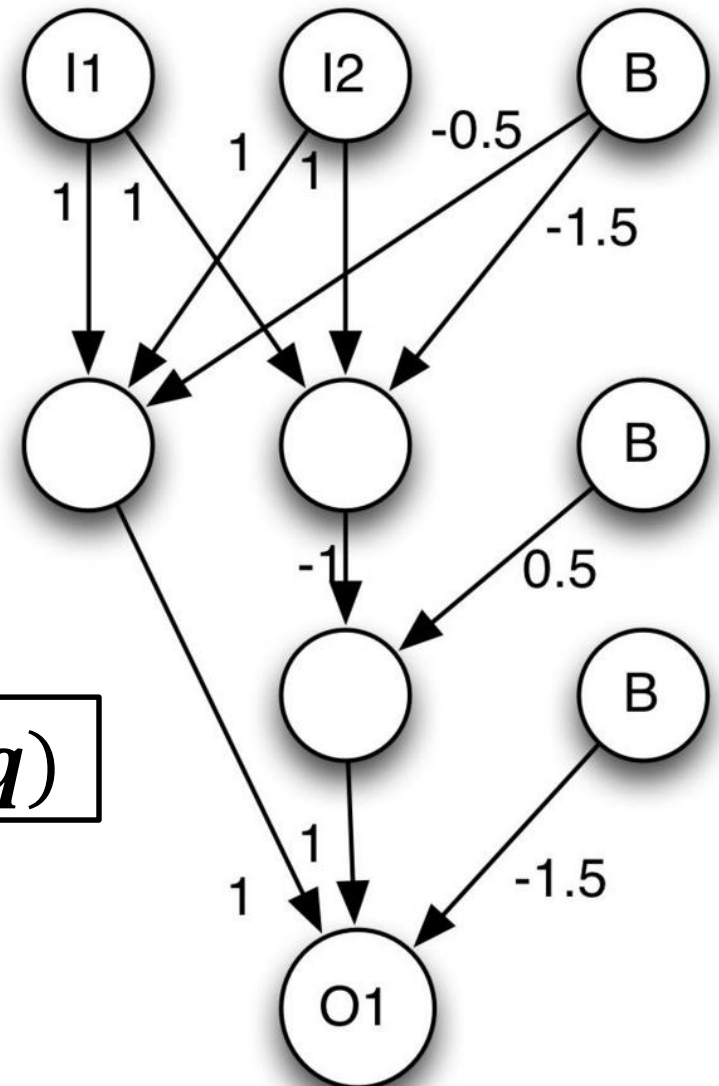
Notation and Definition

- **Fully connected neural networks:**
 - Neurons will connect to all neurons in the previous and next layer
- **Activation Function :**
 - A transformation function that signals whether a neuron is triggered
- **Input Layer :**
 - Layer of neurons that take input from data
- **$W^{L1, L2}$:**
 - Weight matrix between layer 1 and layer 2
- **Output Layer:**
 - Layer of neurons that creates output from the given training data
- **Hidden Layers:**
 - One or multiple layers that lies between input and output layer that does linear or non-linear transform of the input data

XOR's NN with the Ground Truth Weights

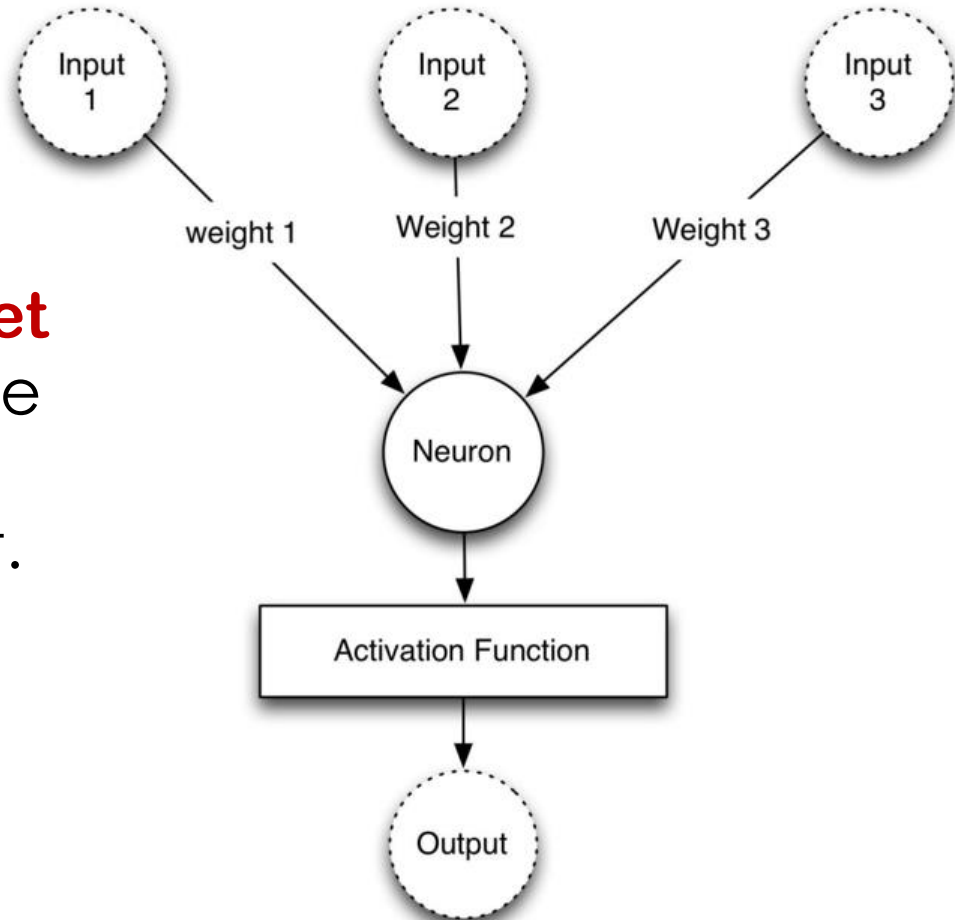
- False XOR False = False
- True XOR False = True
- False XOR True = True
- True XOR True = False

$$p \otimes q = (p \vee q) \wedge \neg (p \wedge q)$$



Training a Neural Network

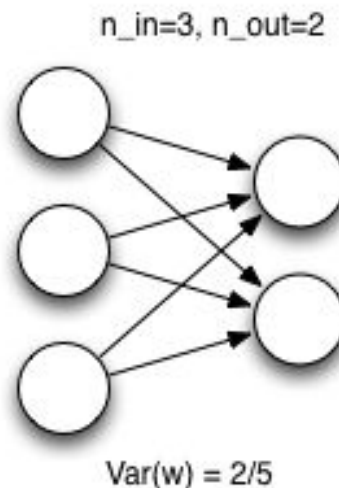
Training is a **search for the set of weights** that will cause the neural network to have the lowest error for a training set.



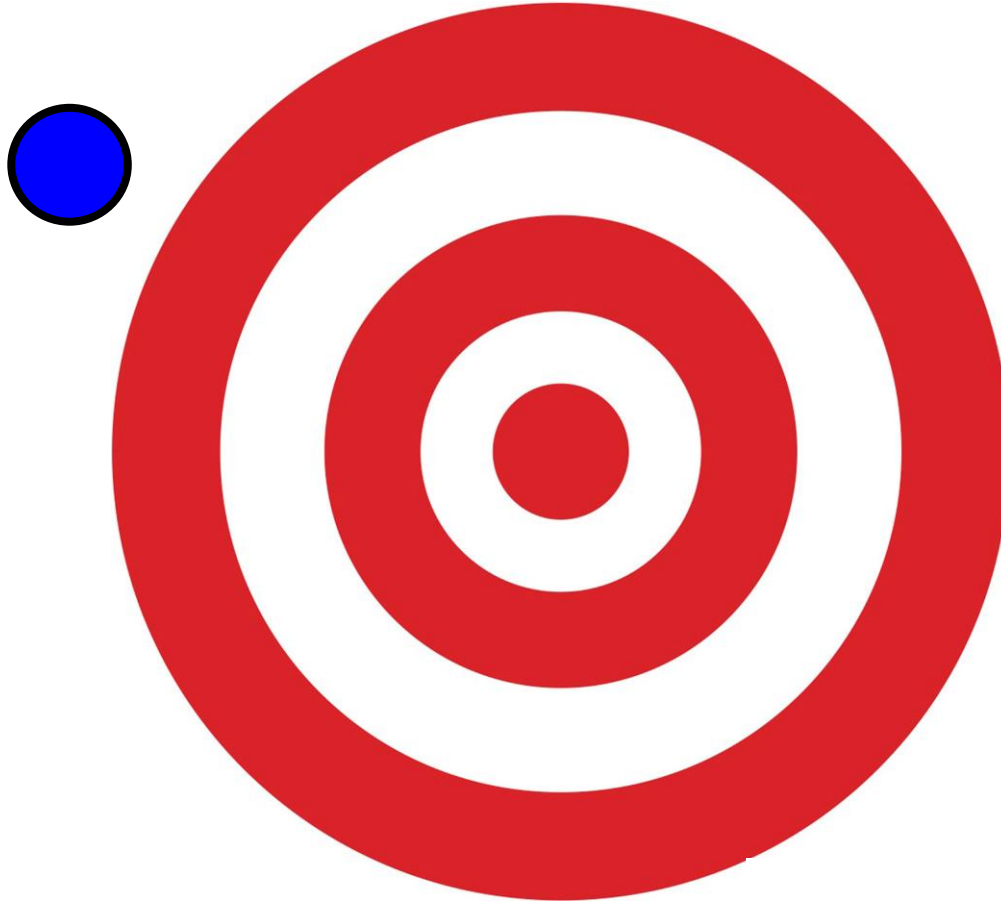
Recall: Xavier Weight Initialization

- Sets all of the weights to normally distributed random numbers
- Weights within the same layer have the same init value
- The weights are always centered at 0
- And the standard deviation (or sqrt of the variance) defined as a function of the number of neurons in the in-layer and the out-layer provided the edge connects nodes between in-to-out layers:

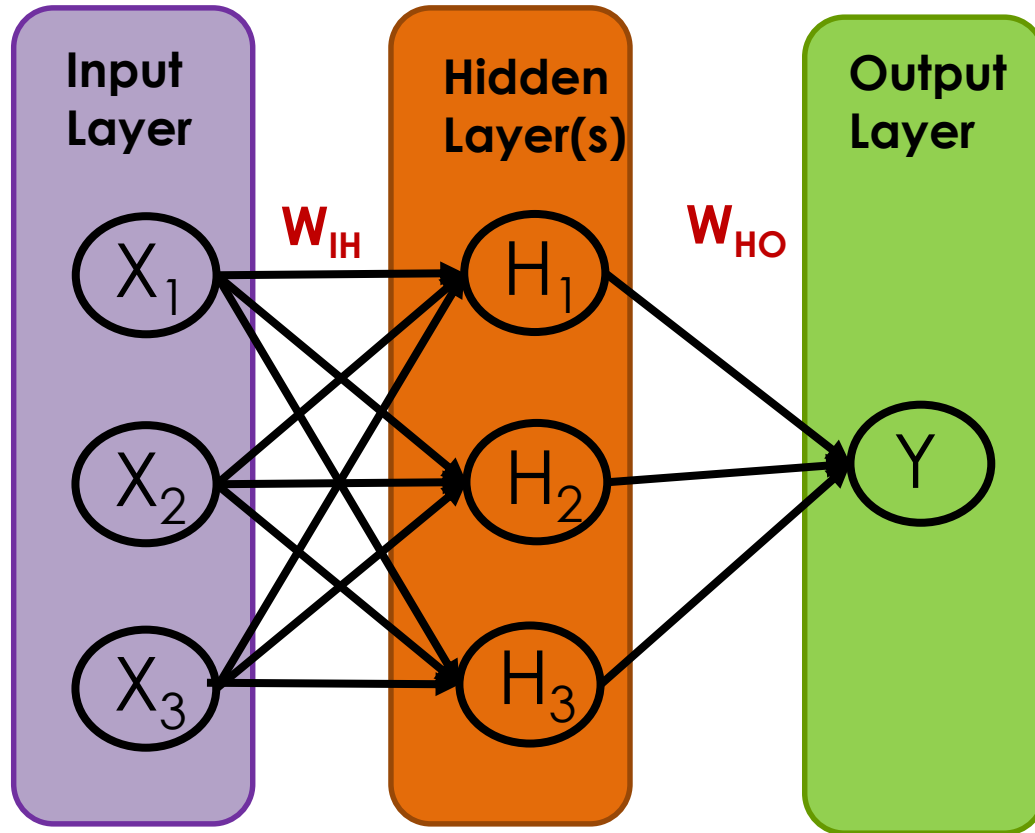
$$\text{Var}(\text{weight}) = \frac{2}{n_{in} + n_{out}}$$



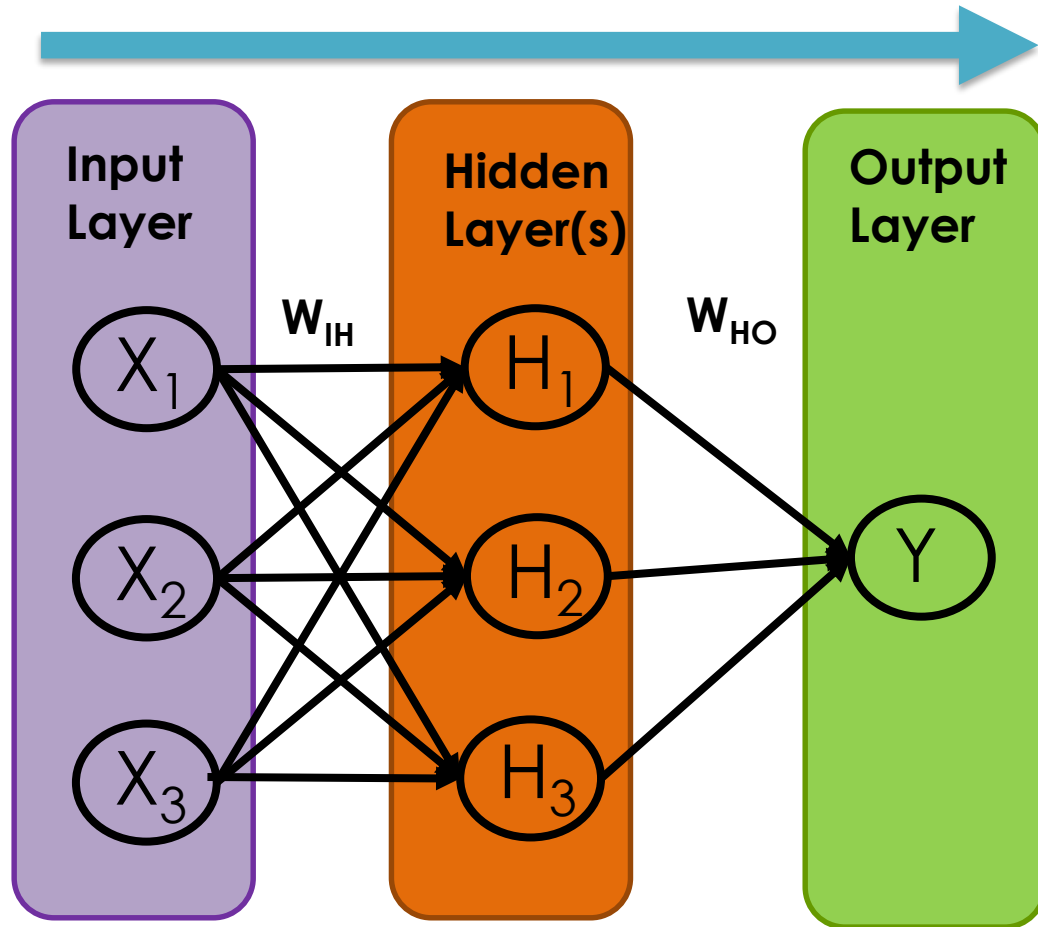
Beginning: Just Shoot Randomly



Random Shoot: Initiate Weight Matrix Randomly



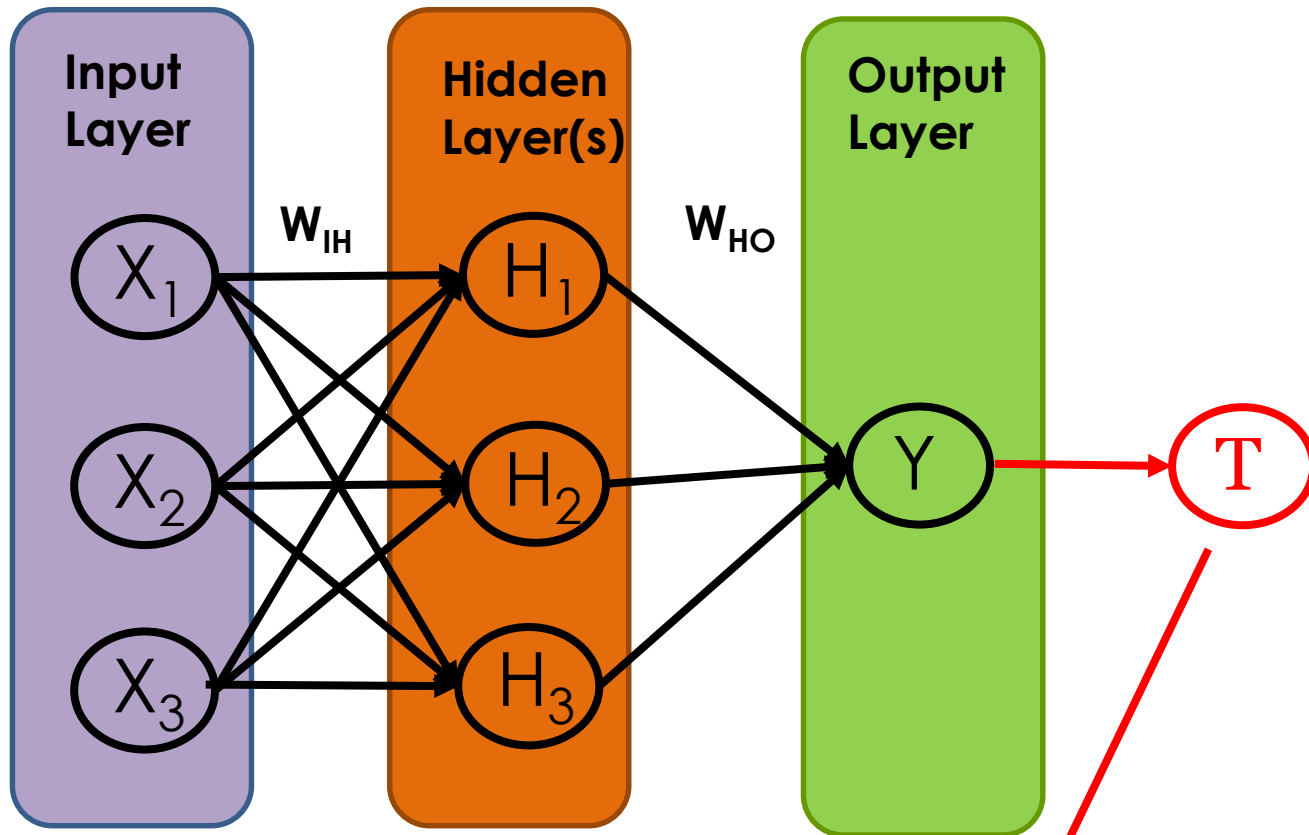
Forward Pass: Compute the Output



A Feed Forward Neural Network

$$Y = \sum_{j=1}^J \varphi \left(\sum_{i=1}^I W_{ij}^{IH} * X_i + b_j \right) * W_j^{HO} =$$
$$\begin{pmatrix} \varphi \left(\begin{matrix} X_1 * W_{11}^{IH} + \\ + X_2 * W_{21}^{IH} + b1 \\ + X_3 * W_{31}^{IH} \end{matrix} \right) * W_{11}^{HO} + \\ \varphi \left(\begin{matrix} X_1 * W_{12}^{IH} \\ + X_2 * W_{22}^{IH} + b2 \\ + X_3 * W_{32}^{IH} \end{matrix} \right) * W_{21}^{HO} + \\ \varphi \left(\begin{matrix} X_1 * W_{13}^{IH} + \\ + X_2 * W_{23}^{IH} + b3 \\ + X_3 * W_{33}^{IH} \end{matrix} \right) * W_{31}^{HO} \end{pmatrix}$$

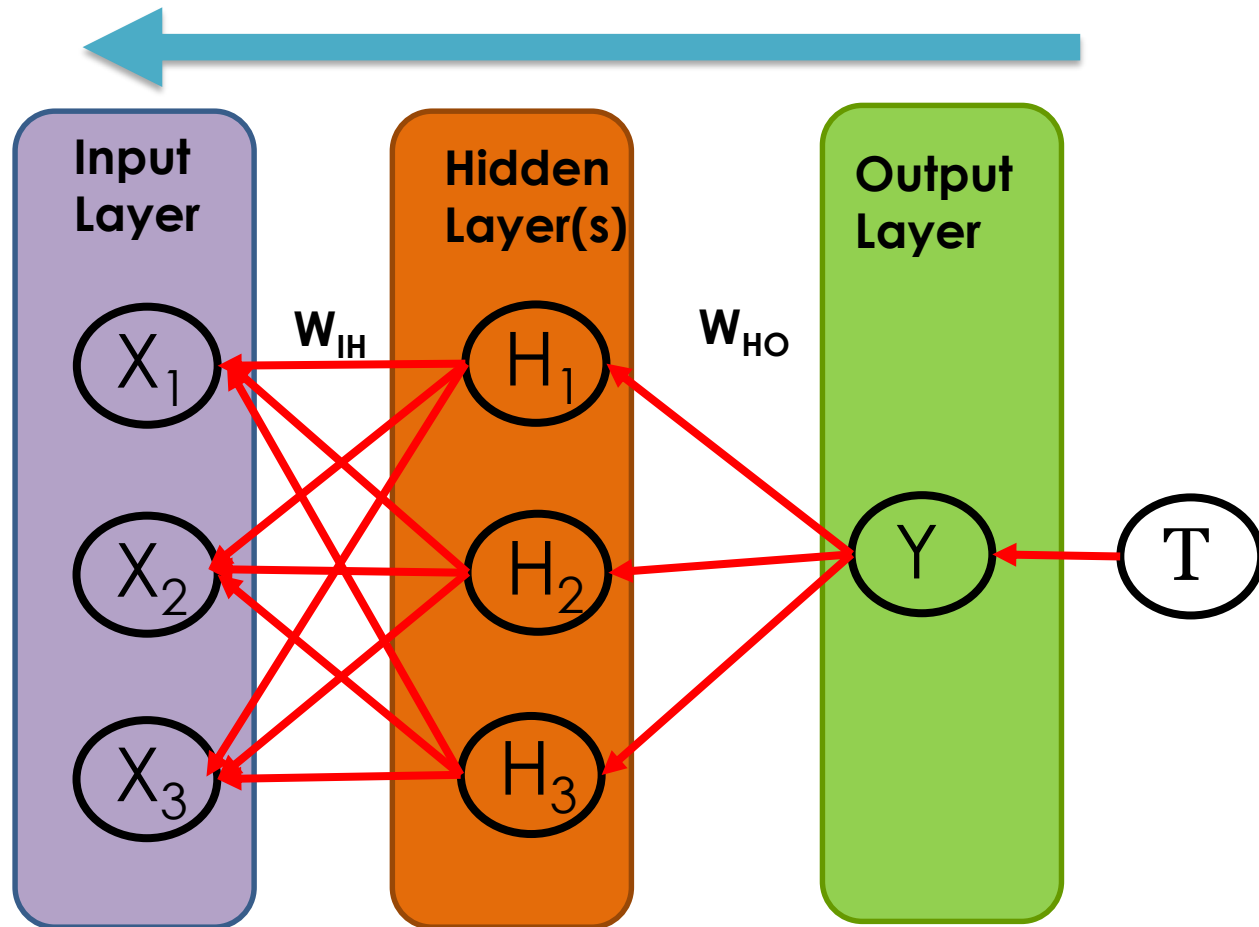
Weight Matrix: How You Aim



$$\text{Error function} = E = \frac{1}{2} \sum (y - t)^2$$

How far off you are

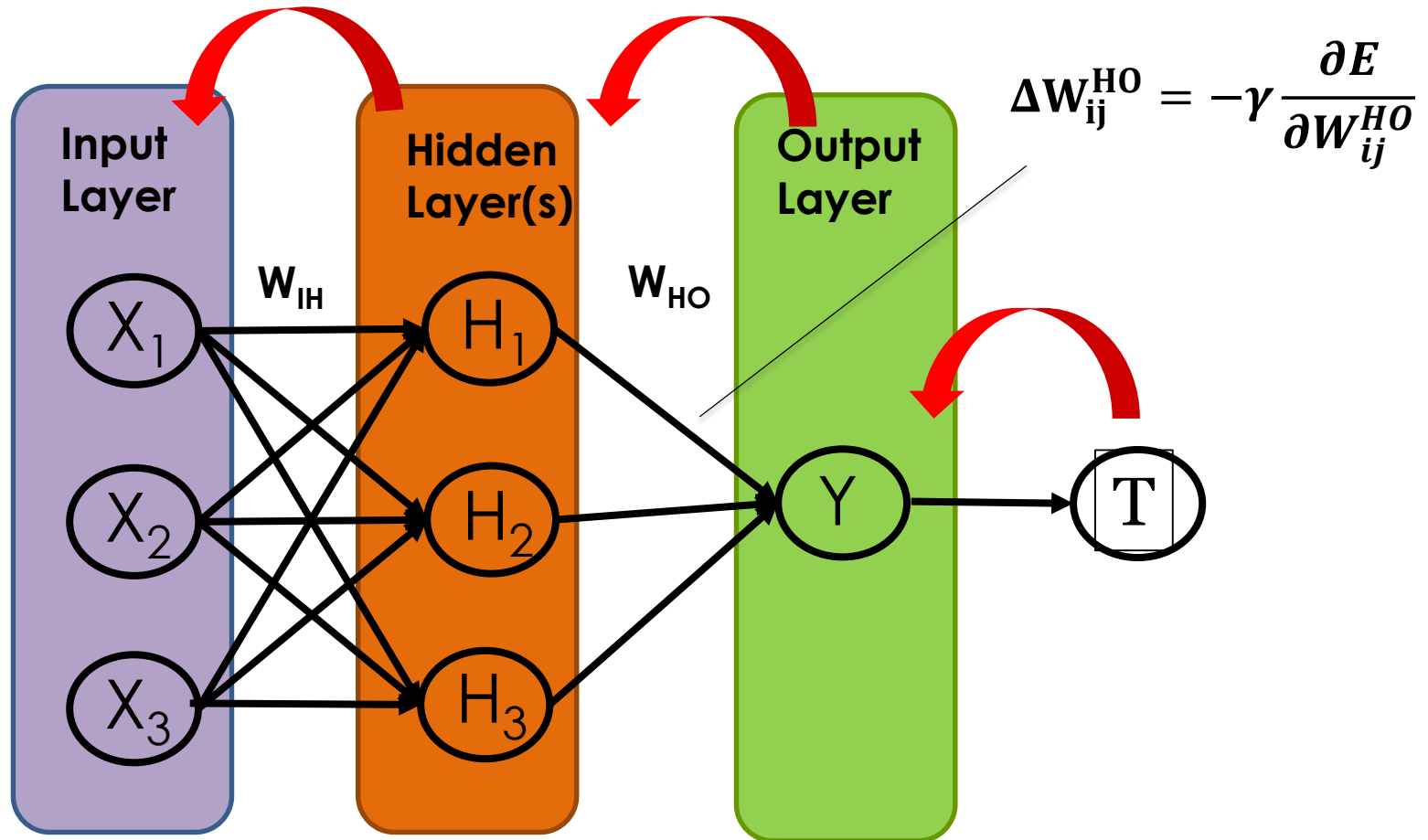
Backpropagation: Pass Error Back



$$\text{Error function} = E = \frac{1}{2} \sum (y - t)^2$$

Check each parameters contribution to the error

By Updating the Weights w/ Gradient Descent



Back-propagation ends when:

1. The gradient becomes less significant
2. Certain number of iterations is reached

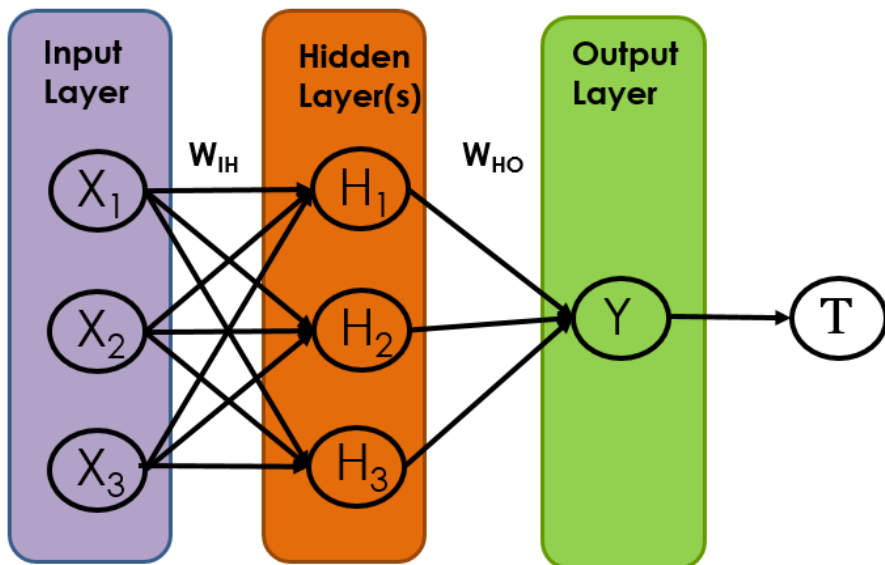
Varied based on different implementation

Based on Weights' Error Contribution

Contribution to the error = $\frac{\partial E}{\partial \text{Variable}}$

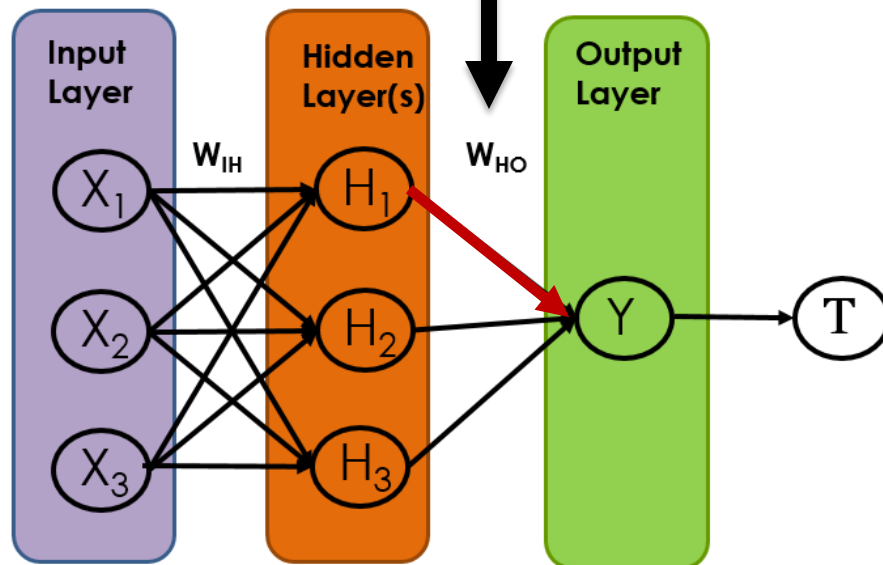
Example:

$$\frac{\partial E}{\partial W} = \partial \left(T - \left(\varphi \left(\begin{array}{c} X_1 * W_{11}^{IH} \\ + X_2 * W_{21}^{IH} + b1 \\ + X_3 * W_{31}^{IH} \end{array} \right) * W_{11}^{HO} + \right. \right. \\ \left. \left(\varphi \left(\begin{array}{c} X_1 * W_{12}^{IH} \\ + X_2 * W_{22}^{IH} + b2 \\ + X_3 * W_{32}^{IH} \end{array} \right) * W_{21}^{HO} + \right. \right. \\ \left. \left(\varphi \left(\begin{array}{c} X_1 * W_{13}^{IH} \\ + X_2 * W_{23}^{IH} + b3 \\ + X_3 * W_{33}^{IH} \end{array} \right) * W_{31}^{HO} \right) \right) / \partial W$$

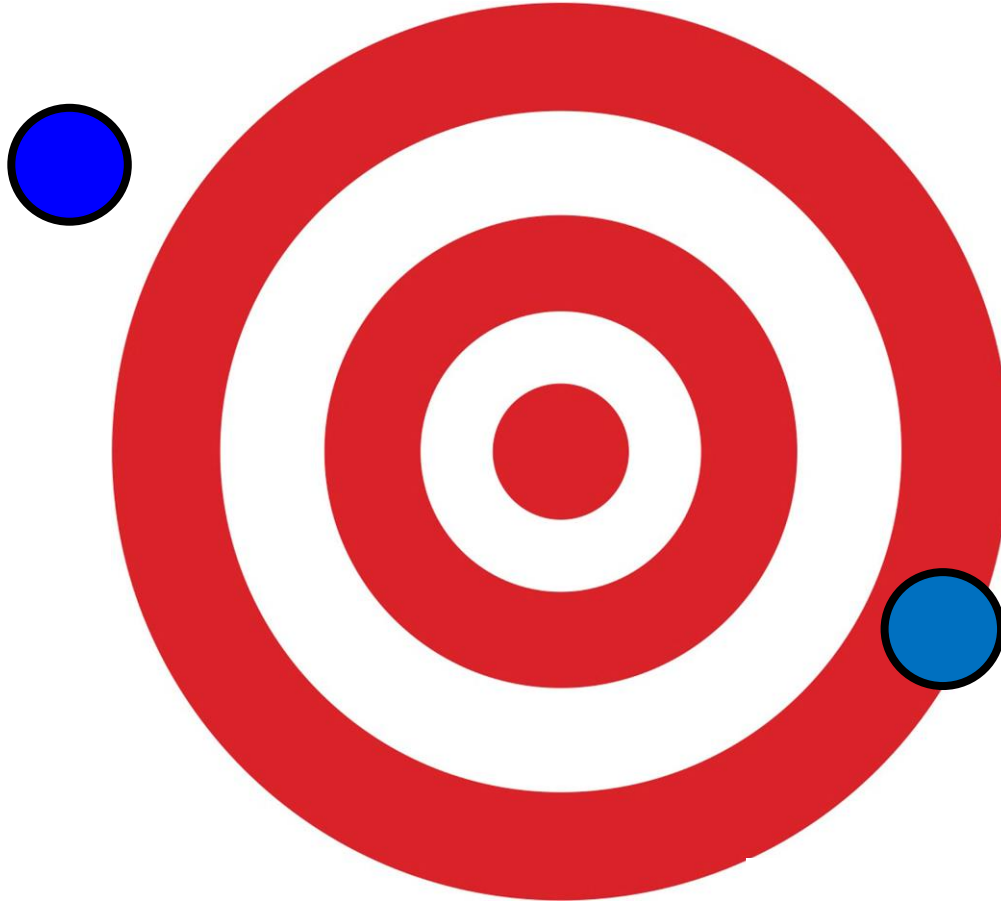


Updated One Edge/Weight At a Time

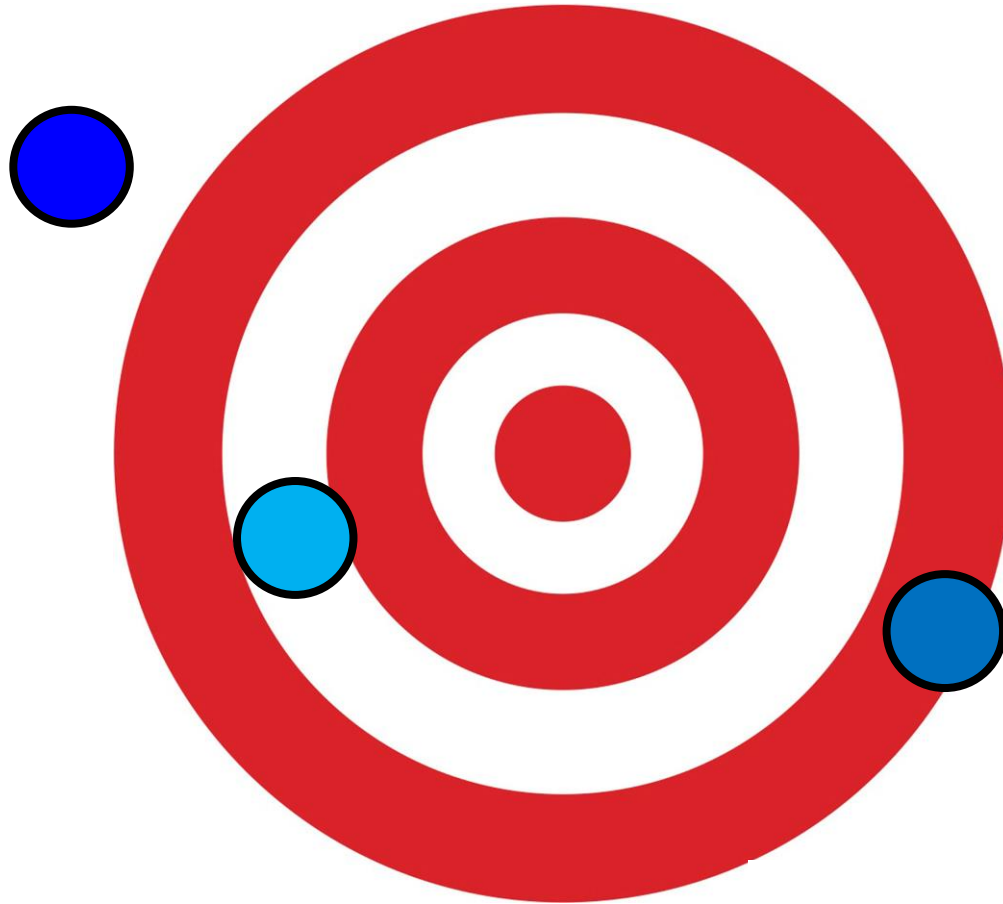
$$\frac{\partial E}{\partial W_{11}^{HO}} = \partial \left(T - \left(\varphi \left(\begin{array}{c} X_1 * W_{11}^{IH} \\ + X_2 * W_{21}^{IH} + b1 \\ + X_3 * W_{31}^{IH} \end{array} \right) \right) * W_{11}^{HO} + \right. \\ \left. \left(\varphi \left(\begin{array}{c} \cancel{X_1 * W_{12}^{IH}} \\ + \cancel{X_2 * W_{22}^{IH}} + b2 \\ + \cancel{X_3 * W_{32}^{IH}} \end{array} \right) \right) * \cancel{W_{21}^{HO}} + \right. \\ \left. \left(\varphi \left(\begin{array}{c} \cancel{X_1 * W_{13}^{IH}} \\ + \cancel{X_2 * W_{23}^{IH}} + b3 \\ + \cancel{X_3 * W_{33}^{IH}} \end{array} \right) \right) * \cancel{W_{31}^{HO}} \right) / \partial W$$



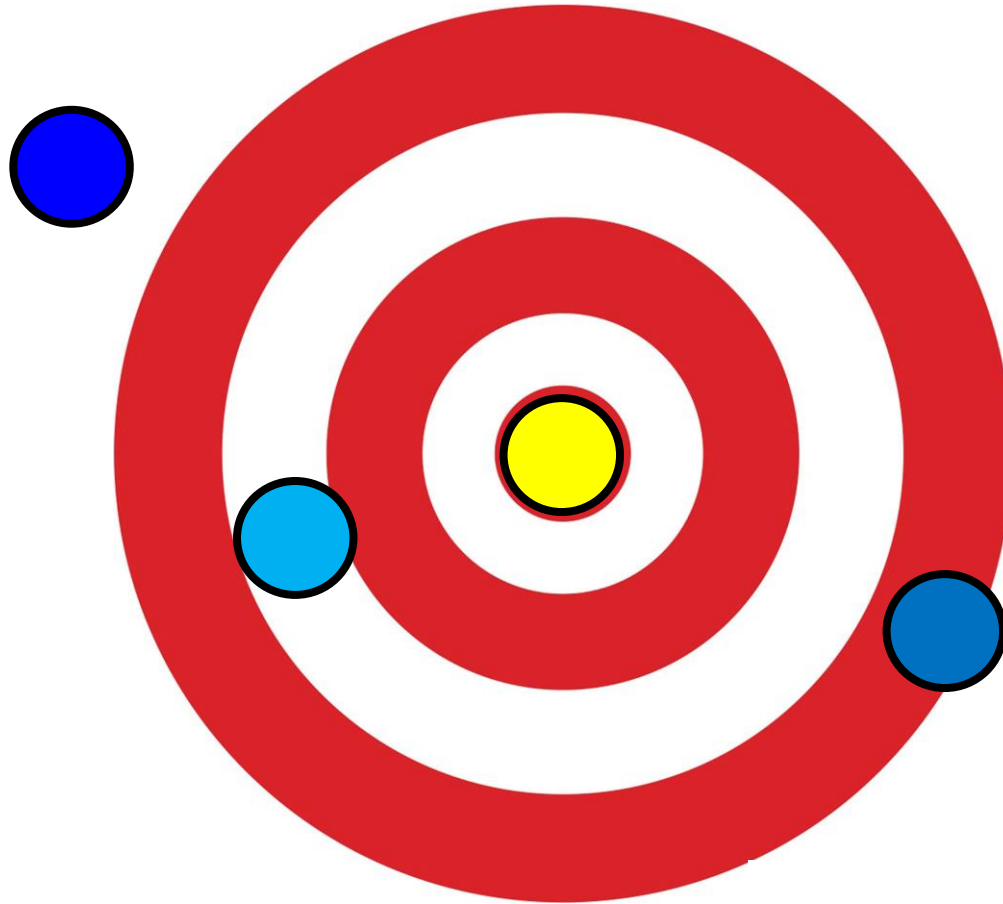
Thus, Modify the Aim & Repeat the Steps



And Will Get Closer



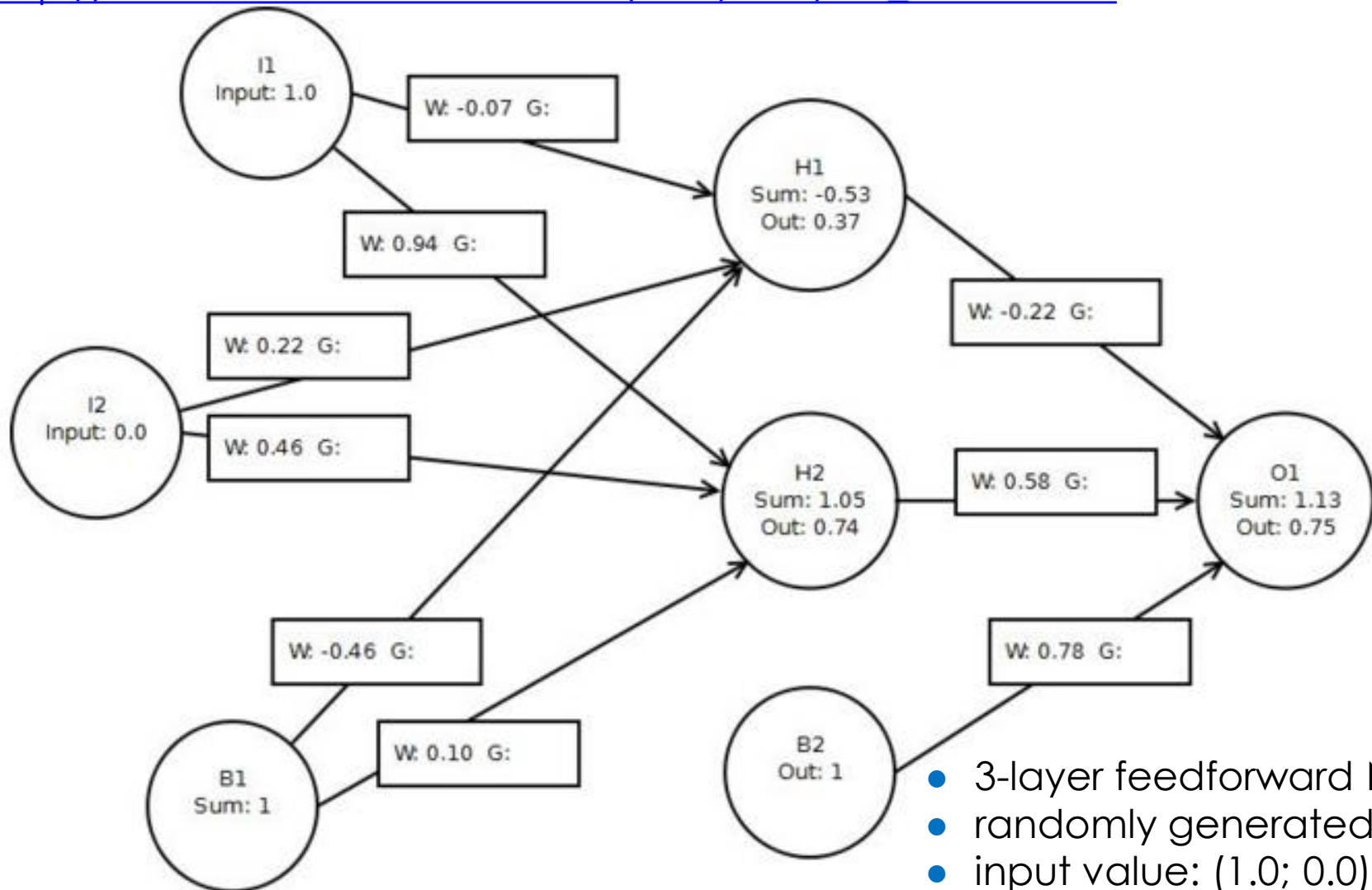
Finally, Get to the Bull's Eye



Ex: Training XOR Neural Network w/ Backpropagation

http://www.heatonresearch.com/aifh/vol3/xor_online.html

http://www.heatonresearch.com/aifh/vol3/xor_batch.html



- 3-layer feedforward NN
- randomly generated weights
- input value: (1.0; 0.0)

Outline

- How to train a NN: High-Level Idea
- **What is Backpropagation:**
 - **Forward & Backward Passes**
- Understanding Gradient
- Calculating Node Deltas
 - Output Node: Quadratic & cross-entropy error functions
 - Interior Node
- Derivatives of the Activation Function
- Applying Backpropagation
 - Batch and Online training
 - Gradient and Stochastic Gradient Descent
 - Backpropagation weight update
- Tuning NN
 - Convergence
 - Learning rate
 - Nesterov Momentum

What is Backpropagation in the Context of NN?

- **Backpropagation:**
 - Method for training a neural network (NN)
 - Introduced by Hinton and Williams (1986)
 - It is a type of Gradient Descent (GD)
 - Lots of extensions based on Stochastic Gradient Descent (SGD)
- **Why to use backpropagation for deep learning?**
 - It scales really well when run on GPUs

Two Passes of Backpropagation

- **Forward Pass**

- The forward pass **computes the output** of the NN

- **Backward Pass**

- Calculate the gradient **ONLY** for the current item in the training
- Update the relevant weights based on the gradient sign

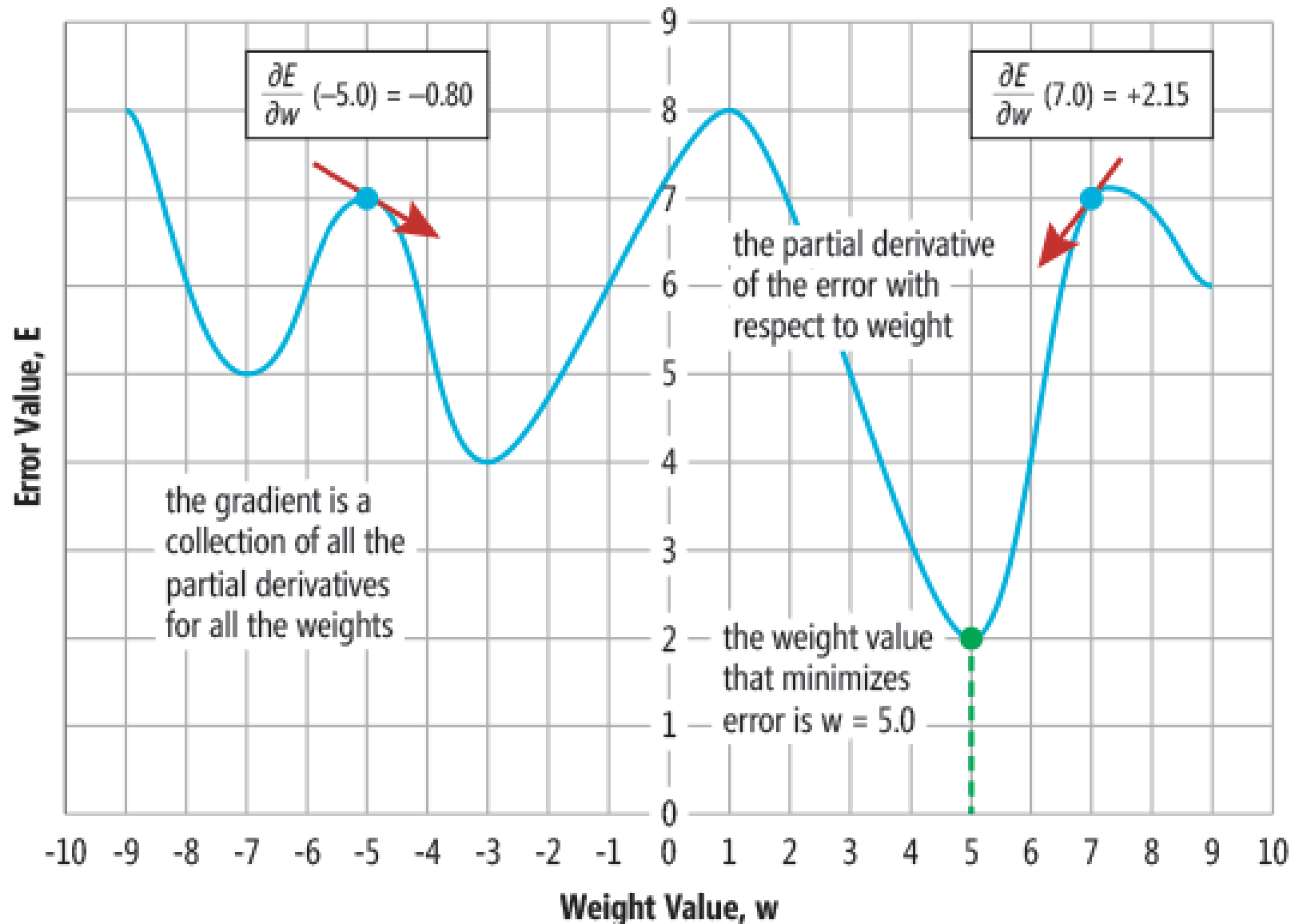
Outline

- How to train a NN: High-Level Idea
- What is Backpropagation:
 - Forward & Backward Passes
- **Understanding Gradient**
- Calculating Node Deltas
 - Output Node: Quadratic & cross-entropy error functions
 - Interior Node
- Derivatives of the Activation Function
- Applying Backpropagation
 - Batch and Online training
 - Gradient and Stochastic Gradient Descent
 - Backpropagation weight update
- Tuning NN
 - Convergence
 - Learning rate
 - Nesterov Momentum

Reminder: Gradient Descent

- **Gradient**
 - The vector of **partial derivatives of the error** function over each weight computed at the weight's current value
 - The error function (loss, cost) measures the distance of the NN's output from the expected output
 - Gradient is the **instantaneous slope** of the error function at the specified weight
 - **Each weight's gradient is the slope of the error function:**
 - The weight is a connection between two neurons / nodes in NN
- The **sign of the gradient** tells the NN the following:
 - **Zero gradient:** weight is not contributing to the error of the NN
 - **Negative gradient:** **increase** the weight to lower the error
 - **Positive gradient:** **decrease** the weight to lower the error

Error Depends on the Weight Value



Outline

- How to train a NN: High-Level Idea
- What is Backpropagation:
 - Forward & Backward Passes
- Understanding Gradient
- **Calculating Node Deltas**
 - **Output Node: Quadratic & cross-entropy error functions**
 - **Interior Node**
- Derivatives of the Activation Function
- Applying Backpropagation
 - Batch and Online training
 - Gradient and Stochastic Gradient Descent
 - Backpropagation weight update
- Tuning NN
 - Convergence
 - Learning rate
 - Nesterov Momentum

Calculating the Gradient for Each Weight

- Step 1: Calculate the error based on the ideal of the training set
- Step 2: Calculate the node delta for the output neurons
- Step 3: Calculate the node delta for the interior neurons
- Step 4: Calculate individual gradients

Step 2: Calculate Output Node Deltas

- **Backward Pass**
 - **Start** with the **output nodes** and work our way **backward** through the neural network
 - Calculate the errors for the output neurons
 - **Propagate** these **errors backwards** through the neural network
- **Node Delta**
 - The node delta is calculated for each node
 - Calculate node deltas one layer at a time
- **Output Node Deltas**
 - Calculated first
 - Take into account the error function for the NN:
 - E.g., quadratic error function or the cross-entropy error function
- **Interior Node Deltas**
 - ???

Output Node Delta for the Error Function

Quadratic Error Function

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

The **output node delta** for the quadratic error function:

$$\delta_i = (\hat{y}_i - y_i) \times \phi_i'$$

phi-prime: the derivative of the activation function

Cross Entropy Error Function

$$CE = \frac{1}{n} \sum_{i=1, \dots, n} (y_i \times \ln \hat{y}_i + (1 - y_i) \times \ln(1 - \hat{y}_i))$$

cross-entropy (CE): the log loss

The **output node delta** for the cross entropy error function:

$$\delta_i = (\hat{y}_i - y_i)$$

Step 3: Calculate Interior Node Delta

The **interior node delta** (all hidden and bias neurons):

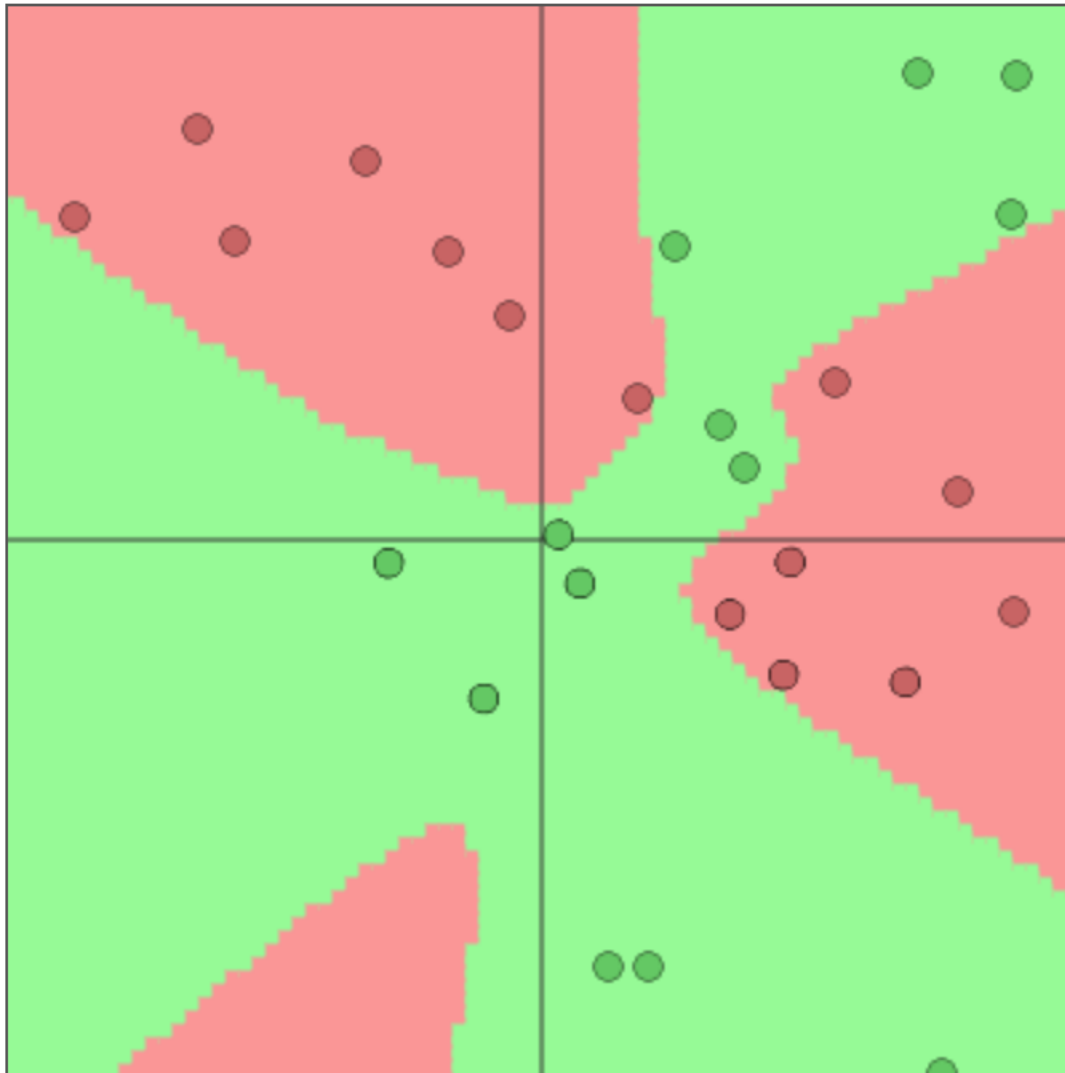
$$\delta_i = \phi'_i \times \sum_k w_{ki} \delta_k$$

phi-prime: the derivative of
the activation function

Outline

- How to train a NN: High-Level Idea
- What is Backpropagation:
 - Forward & Backward Passes
- Understanding Gradient
- Calculating Node Deltas
 - Output Node: Quadratic & cross-entropy error functions
 - Interior Node
- **Derivatives of the Activation Function**
- Applying Backpropagation
 - Backpropagation weight update
 - Batch and Online training
 - Gradient and Stochastic Gradient Descent
- Tuning NN
 - Convergence
 - Learning rate
 - Nesterov Momentum

Example: Learn the Boundaries by NN-2



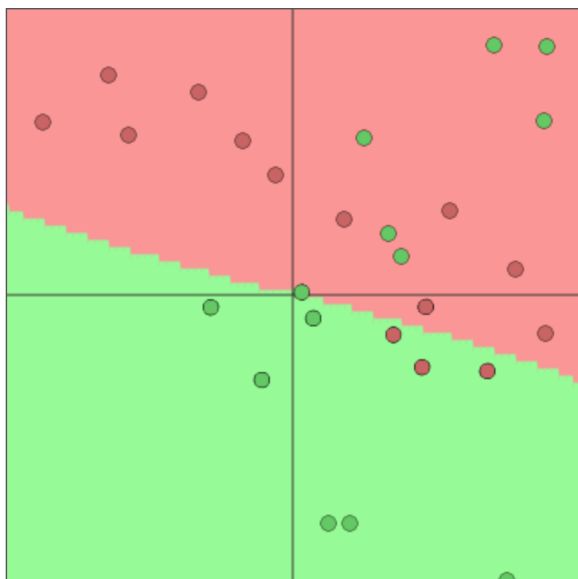
This is better!

Questions

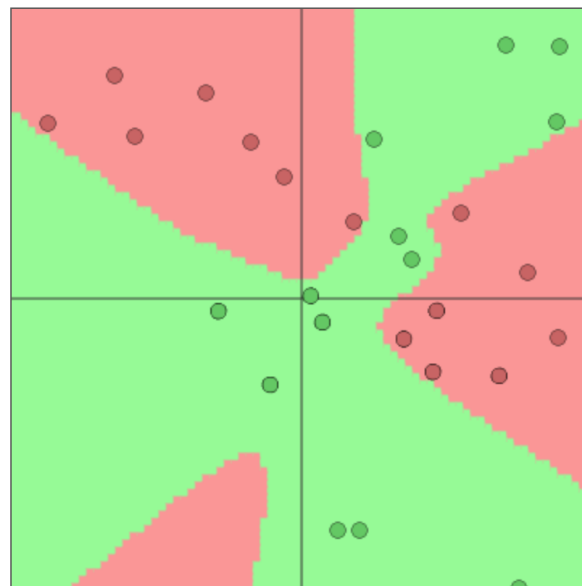
- Why we need the activation function, ϕ ?
- What are the proper functions for ϕ in neural networks ?

Why Non-linearity?

*Net = $X_1W_1 + X_2W_2 + X_3W_3 + b$ Is just linear transformation



?



Linear

Non-linear transformation will allow the model to capture complex patterns, we call it activation function

Non-Linear

Derivatives of the Activation Function

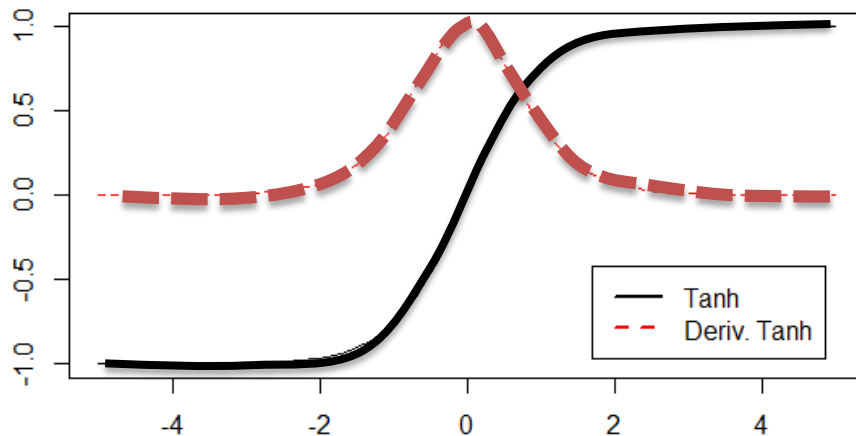
Activation Function	AF Formula, $\phi(z)$	AF Derivative, $\phi'(z)$
Linear AF	$\phi(z)$	$\phi'(z) = 1$
Softmax AF	$\phi(z_i) = \frac{e^{z_i}}{\sum_{j \in \text{group}} e^{z_j}}$	$\phi'(z_i) = \phi(z_i)(1 - \phi(z_i))$
Sigmoid	$\phi(z) = \frac{1}{1 + e^{-z}}$	$\phi'(z) = \phi(z)(1 - \phi(z))$
Hyperbolic Tangent AF	$\phi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$\phi'(z) = 1 - \phi^2(z)$
Rectified Linear Units (ReLU) AF	$\phi(z) = \max(0, z)$	$\phi'(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$

- **ReLU does NOT have a derivative at zero.** BUT, because of convention, the gradient of zero is substituted when $z = 0$
- Deep neural networks are difficult to train with sigmoid and hyperbolic tangent AFs using backpropagation:
 - **Vanishing gradient problem**

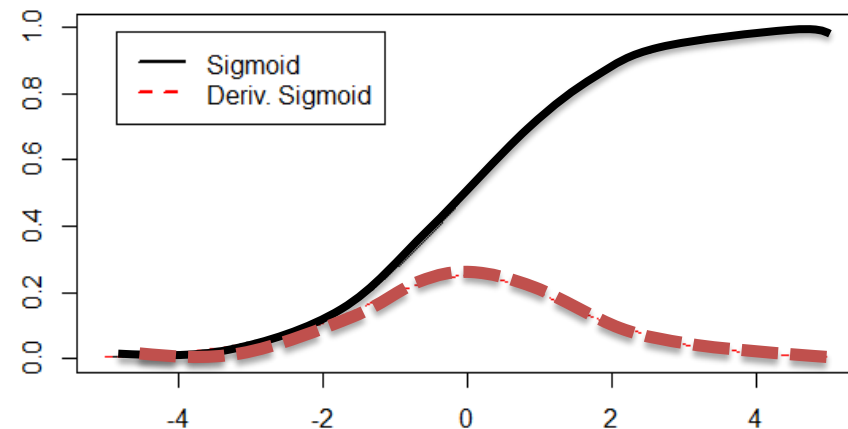
AF Saturation and its Effect on Derivative

Saturation Problem: Because both **hyperbolic tangent** and **sigmoid** AFs **saturate** to $-1/1$ and $0/1$, respectively, their **derivatives vanish** to zero but **ReLU** does NOT have this problem.

Hyperbolic Tangent & its Derivative



Sigmoid & its Derivative



Linear, Softmax, and ReLU AF's

- Do not use the derivative of the linear or softmax AF for the cross-entropy error function (as it is = 1)
- Use **linear** and **softmax** AF ONLY at the **output layer** of the NN
- Use **ReLU** at the **hidden layers** of the NN, i.e., inferior nodes of the NN

Error Function	Linear AF Derivative, $\phi'(z)$	Softmax Derivative
Output nodes with cross entropy error function	$\phi'(z) = 1$	$\phi'(z) = 1$

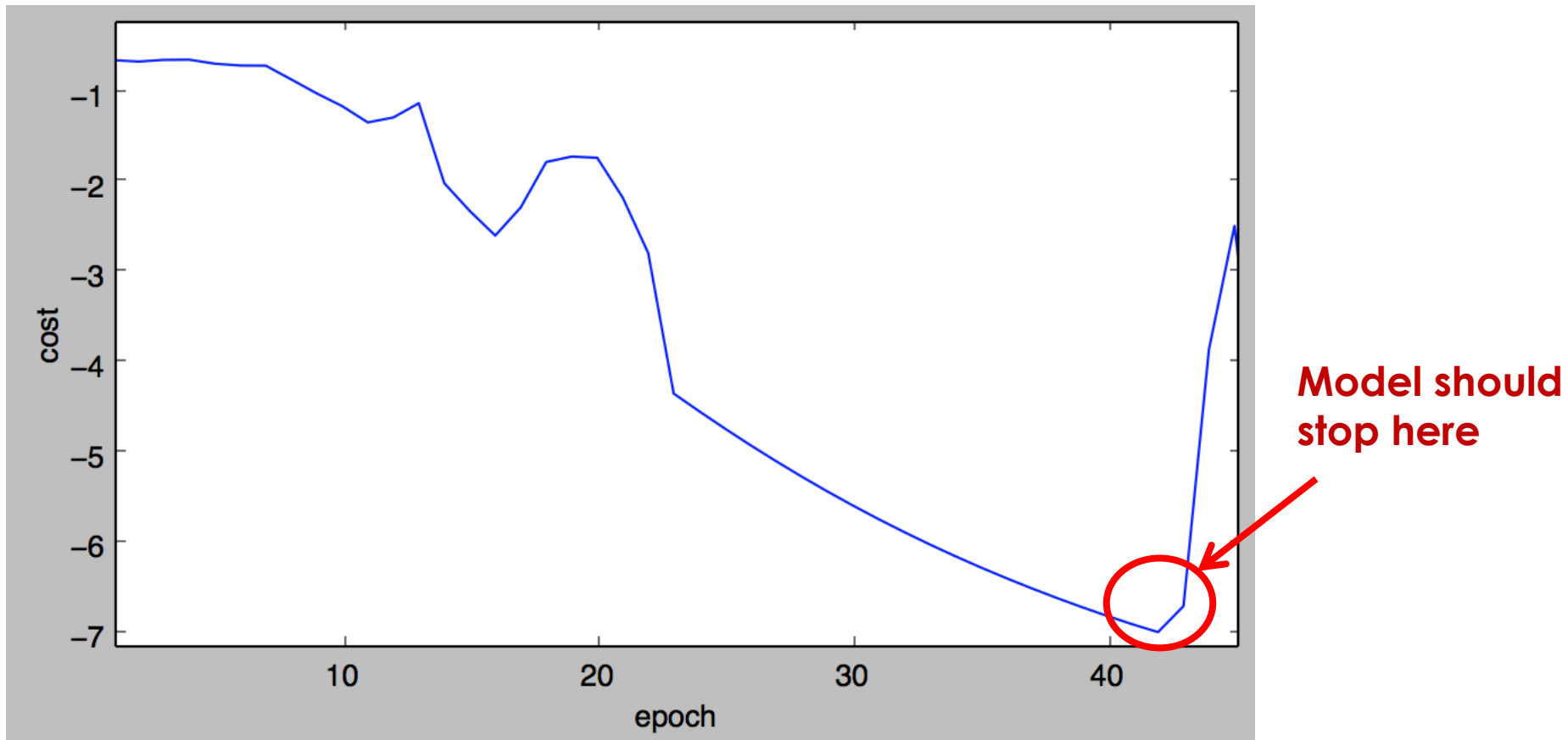
Other ways to solve vanishing gradient problem

1. Use domain knowledge to create very good initial weights for the network ← Which is very difficult
2. Pre-Train the model through unsupervised neural network models ← (Stacked) Auto Encoder

Outline

- How to train a NN: High-Level Idea
- What is Backpropagation:
 - Forward & Backward Passes
- Understanding Gradient
- Calculating Node Deltas
 - Output Node: Quadratic & cross-entropy error functions
 - Interior Node
- Derivatives of the Activation Function
- **Tuning NN**
 - **Convergence**
 - **Backpropagation weight update**
 - **Learning rate & momentum**
- Applying Backpropagation
 - Batch and Online training
 - Gradient and Stochastic Gradient Descent
 - Nesterov Momentum

Evolution of the Cost / Error



1. How do we make sure the model can reach local minimum?
2. How do we make sure the model can reach global minimum?
3. How can we train the model fast?

Backpropagation Weight Update

$$\Delta w_t = -\gamma \frac{\partial Error}{\partial w_t} + \alpha \Delta w_{t-1}$$

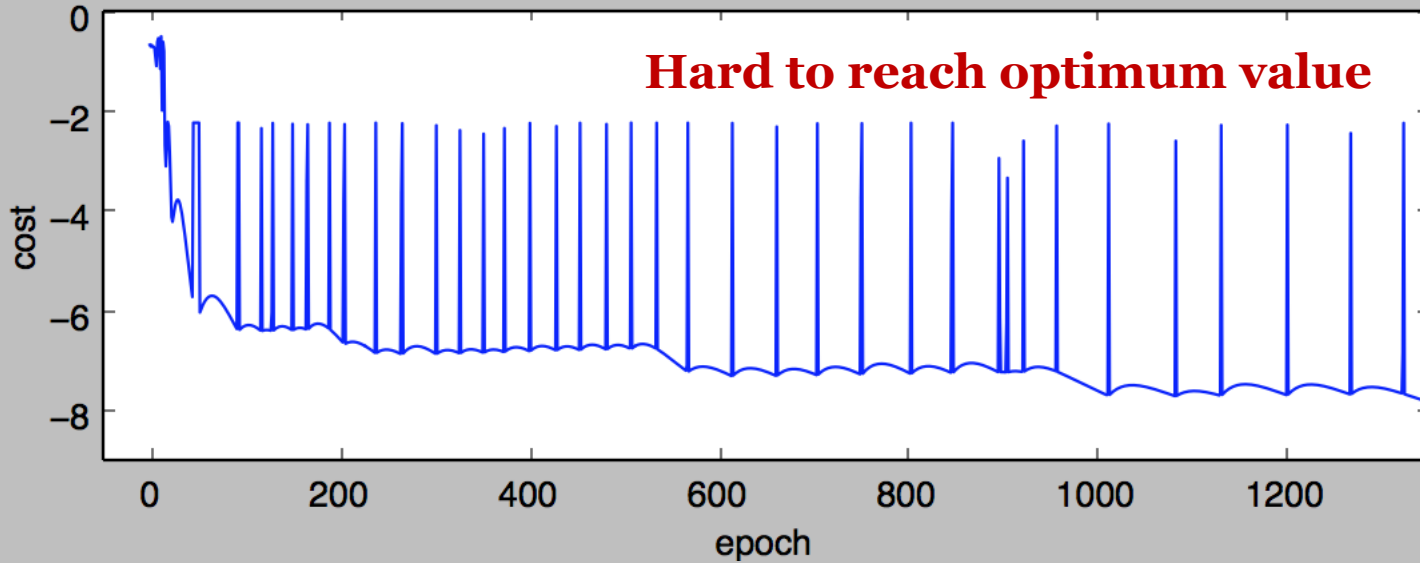
- t : the current iteration step
- γ : the **learning rate**
- α : the **momentum** value (how much the previous weight change should be counted for)

Choosing Learning Rate & Momentum

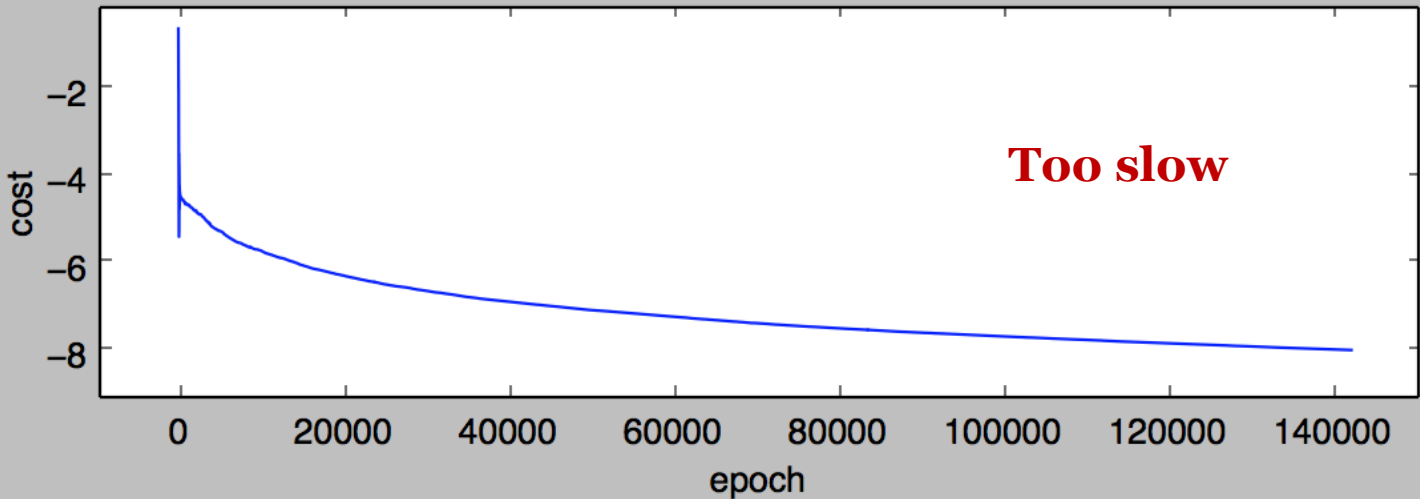
- Note: Choosing the learning rate and the momentum is the trial and error
- **Learning Rate, γ ($\gamma \sim 0.1$):**
 - $\gamma = 0.5$ will decrease every gradient by 50%
 - $\gamma < 1$ slows down learning (most likely choice)
 - $\gamma > 1$ accelerated learning
 - γ is too high \rightarrow causes NN to fail to converge and have a high global error bounce around instead of converging to a lower value
 - γ is too low \rightarrow causes NN to take lots of time to converge
- **Momentum, α ($\alpha \sim 0.9$):**
 - Helps the training to escape local minima that are not true global minimum
 - It gives a NN some force to break through a local minimum

Comparison of Learning Rates

High Learning Rate



Low Learning Rate



Learning Rate Strategies

- **Fixed**
 - Pros: Simple
 - Cons: Its hard to find one learning rate for all
- **Steadily Decreasing**
 - Pros: Also Simple
 - Cons: Still very universal
- **Fixed at the beginning, Steadily Decreasing after a certain point**
 - - *It assumes the weight should train faster at beginning*
 - Pros: Also Simple
 - Cons: Still very universal
- **Adaptive Subgradient**
 - Pros: Subjectively penalized weights based on need
 - Cons: Harder to implement, need to keep track of gradients

Adaptive Subgradient (Adagrad)

When the model is sampling data, we might want to treat data subjectively. i.e.:

*Google News **is a** free news aggregator provided **and** operated **by** Google, selecting up-to-date news from thousands **of** publications. **A** beta version **was** launched in September 2002, **and** released officially **in** January 2006.*



We might want to penalized these samples

AdaGrad alters learning rate based on historical information, so events with high frequency will get small learning rate.

$\gamma = \text{Fixed Learning Rate}$

$$\gamma_i = \text{Learning rate for feature } i = \frac{\gamma}{\sqrt{\sum (\text{Gradient of } i)^2}}$$

Outline

- How to train a NN: High-Level Idea
- What is Backpropagation:
 - Forward & Backward Passes
- Understanding Gradient
- Calculating Node Deltas
 - Output Node: Quadratic & cross-entropy error functions
 - Interior Node
- Derivatives of the Activation Function
- Tuning NN
 - Convergence
 - Backpropagation weight update
 - Learning rate & momentum
- **Applying Backpropagation**
 - **Batch and Online training**
 - **Gradient and Stochastic Gradient Descent**
 - **Nesterov Momentum**

Speed Up, and Jump Out Of Local Minimum

When input data is large, or have many hidden layers in the model. The speed for processing gradient descent for each position in the weight matrices will become an issue for efficient model training. A proposed method is Stochastic Gradient Descent that randomly picks one training sample for each iteration.

General Gradient Descent:

- For each iteration

 - For each sample in test data

 - Gradient Descent on the parameters

Stochastic Gradient Descent:

- For each iteration

 - For **one** sample in test data

 - Gradient Descent on the parameters

Mini Batch Update

While Stochastic Gradient is speeding up the training speed, we can still improve the efficiency by utilizing matrix multiplications.

Mini-Batch update samples B gradients to update the model.

When $B = 1$, its normal Stochastic Gradient Descent

When $B = \text{training data size}$, its normal Gradient Descent

Can use priority sampling:

The input data will be asked to be pooled to high/low priority.

During each batch, it will sample a portion from high and low pool

Problem w/ Mini-Batching & Nesterov Momentum

- Problem with Mini-Batching:
 - SGD can produce erratic results b/s of randomness introduced by mini-batching:
 - Weights might get beneficial updates in one iteration but
 - A poor choice of training samples can undo it in the next batch
- Nesterov Momentum
 - Designed to mitigate this erratic training result
 - Referred as Accelerated Gradient Descent

$$\mathbf{n}_0 = \mathbf{0}$$

$$\mathbf{n}_t = \alpha \times \mathbf{n}_{t-1} + \gamma \times \frac{\partial \text{Error}}{\partial \mathbf{w}_t}$$

$$\Delta \mathbf{w}_t = \alpha \times \mathbf{n}_{t-1} - (1 + \alpha) \times \mathbf{n}_t$$

References:

1. Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1), 926.
2. Socher, R., Lin, C. C., Manning, C., & Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 129-136).
3. Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade* (pp. 437-478). Springer Berlin Heidelberg.
4. Course materials from Richard Socher 2015 Spring course: CS224d: Deep Learning for Natural Language Processing
5. Dyer, C. Notes on AdaGrad.
6. Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *The Journal of Machine Learning Research*, 11, 625-660.

References:

7. Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507.
8. Why Google is Investing in Deep Learning
<http://video.mit.edu/watch/why-google-is-investing-in-deep-learning-14382/> (12/8/2015)
9. Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.
10. Karpathy, A., & Fei-Fei, L. (2014). Deep visual-semantic alignments for generating image descriptions. arXiv preprint arXiv:1412.2306.
11. Zero to expert in 8 hours: These robots harness deep learning,
<http://www.thespec.com/news-story/6158522-zero-to-expert-in-8-hours-these-robots-harness-deep-learning/> (12/8/2015)

Extra Slides

Differentiating Sigmoid Function

$$\text{sigmoid function} = S(t) = \frac{1}{1 + e^{-t}} = (1 + e^{-t})^{-1}$$

$$\begin{aligned}\frac{dS(t)}{d(t)} &= (1 + e^{-t})^{-2} * \frac{d(1 + e^{-t})}{d(t)} \\&= (1 + e^{-t})^{-2} * -e^{-t} \\&= (1 + e^{-t})^{-2} - e^{-t} * + (1 + e^{-t})^{-2} - (1 + e^{-t})^{-2} \\&= (1 + e^{-t})^{-2} * (1 - 1 + e^{-t}) \\&= (1 + e^{-t})^{-2} - (1 + e^{-t}) * (1 + e^{-t})^{-2} \\&= S(t)^2 - S(t) \\&= S(t) * (S(t) - 1)\end{aligned}$$

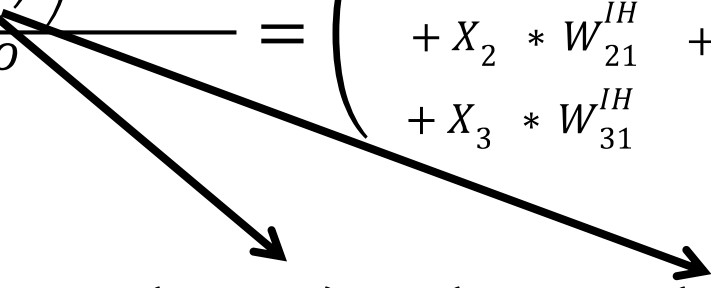
Chain Rule

$$\frac{\partial Z}{\partial X} = \frac{\partial Z}{\partial Y} * \frac{\partial Y}{\partial X} , \text{ if } Z \text{ is a function of } Y$$

$$\begin{aligned} \frac{\partial E_1}{\partial W_{11}^{HO}} &= \frac{\partial E_1}{\partial O_1} * \frac{\partial O_1}{\partial W_{11}^{HO}} \\ &= \frac{\partial E_1}{\partial O_1} * \frac{\partial O_1}{\partial Net_1} * \frac{\partial Net_1}{\partial W_{11}^{HO}} \end{aligned}$$

Chain Rule – Cont.

$$\frac{\partial E_1}{\partial W_{11}^{HO}} = \frac{\partial E_1}{\partial O_1} * \frac{\partial O_1}{\partial Net_1} * \frac{\partial Net_1}{\partial W_{11}^{HO}}$$

$$\frac{\partial Net_1}{\partial W_{11}^{HO}} = \frac{\partial \left(\left(\begin{array}{c} X_1 * W_{11}^{IH} \\ + X_2 * W_{21}^{IH} \\ + X_3 * W_{31}^{IH} \end{array} \right) + b1 \right) * W_{11}^{HO}}{\partial W_{11}^{HO}} = \left(\begin{array}{c} X_1 * W_{11}^{IH} \\ + X_2 * W_{21}^{IH} \\ + X_3 * W_{31}^{IH} \end{array} + b1 \right)$$


$$\frac{\partial O_1}{\partial Net_1} = \frac{\partial \varphi(Net_1)}{\partial (Net_1)} = \varphi(Net_1) * (1 - \varphi(Net_1))$$

$$\frac{\partial E_1}{\partial O_1} = \frac{\partial E_1}{\partial y_1} = \frac{\partial \frac{1}{2}(t_1 - y_1)^2}{\partial y_1} = y_1 - t_1$$

Full Back Propagation

$$\begin{aligned}\frac{\partial E_1}{\partial W_{11}^{HO}} &= \frac{\partial E_1}{\partial O_1} * \frac{\partial O_1}{\partial Net_1} * \frac{\partial Net_1}{\partial W_{11}^{HO}} = \\ &= (y_1 - t_1) * \varphi(Net_1) * \left(1 - \varphi(Net_1)\right) * \text{Input to Layer 0}\end{aligned}$$

$$\begin{aligned}\frac{\partial E_1}{\partial W_{11}^{IH}} &= \frac{\partial E_1}{\partial \varphi(Net_1)} * \frac{\partial \varphi(Net_1)}{\partial Net'_1} * \frac{\partial Net'_1}{\partial W_{11}^{IH}} \\ &= \text{error of next layer} * \varphi(Net'_1) * \left(1 - \varphi(Net'_1)\right) \\ &\quad * \text{Original Input}\end{aligned}$$