PGE 383 Subsurface Machine Learning

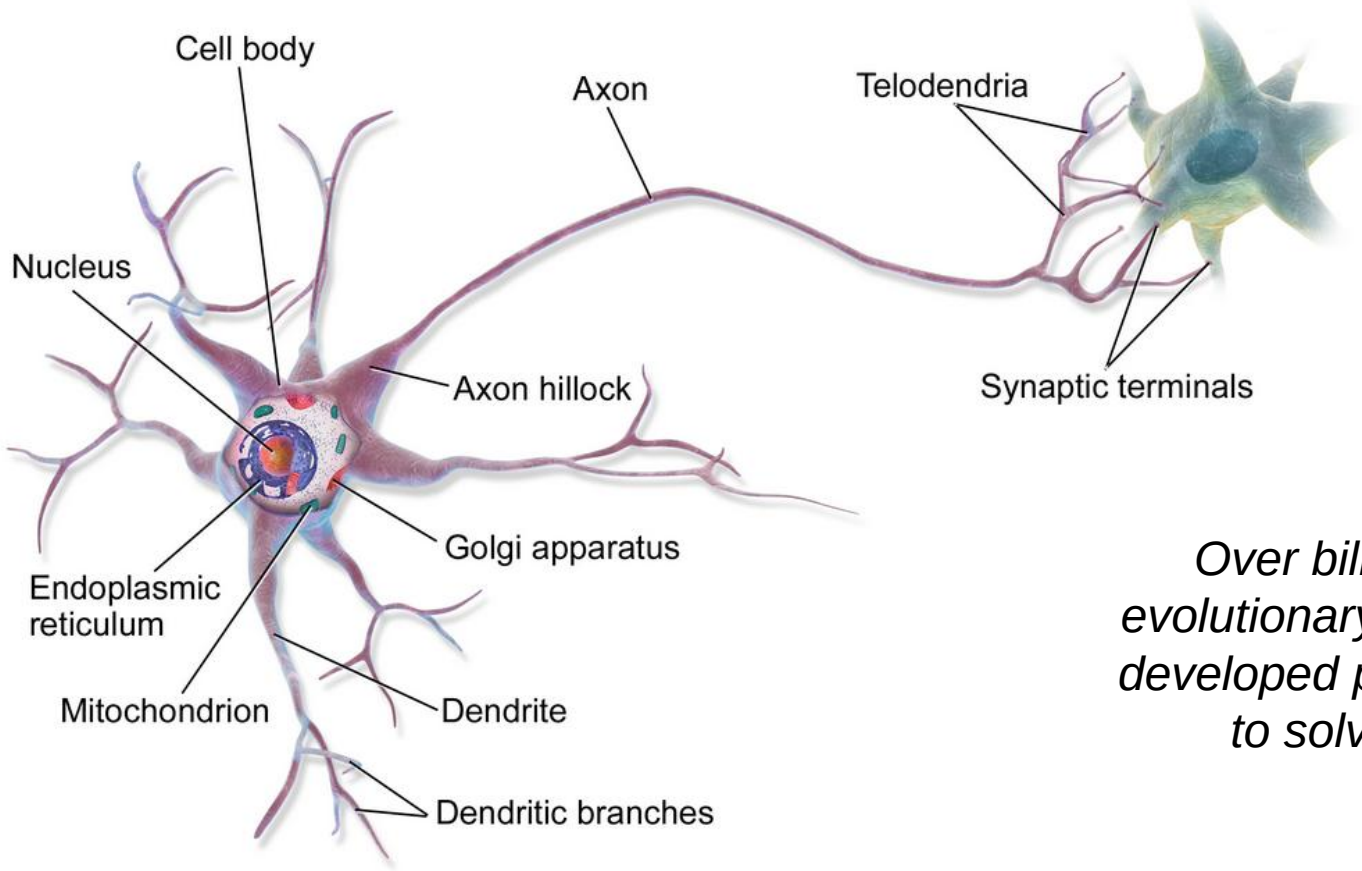
Lecture 17: Neural Networks

Lecture outline:

- **Neural Networks**
- **Neural Networks Example**
- **Neural Networks Hands-on**

Motivation

Very powerful, nature inspired computing building block of convolutional neural networks and other very powerful machine learning methods



Over billions of years, evolutionary processes have developed powerful methods to solve problems.

Anatomy of multipolar neuron, image from https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png

PGE 383 Subsurface Machine Learning

Lecture 17: Neural Networks

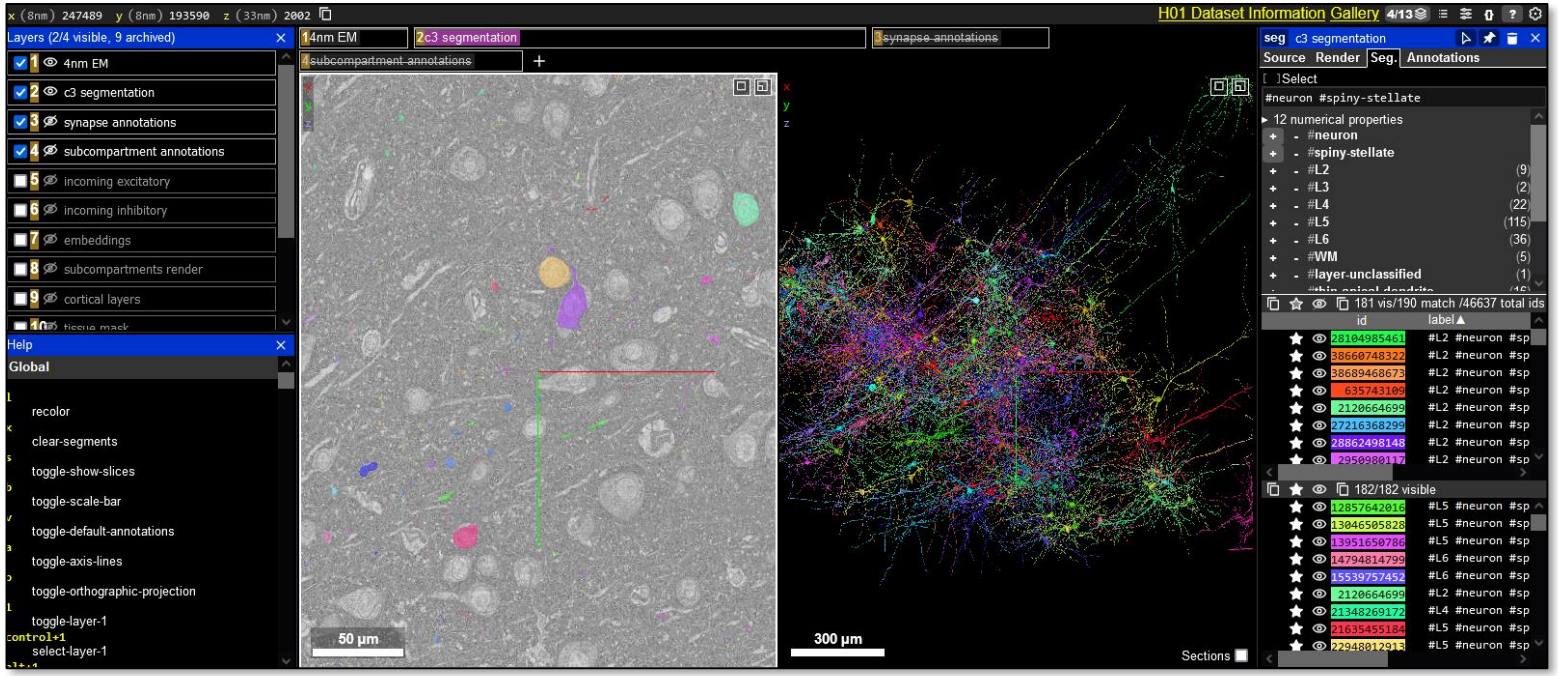
Lecture outline:

- **Neural Networks**

Nature-Inspired Computing

Looking to nature for inspiration to develop novel problem-solving methods.

- artificial neural networks are inspired by biological neural networks
- the nodes in our model are *artificial neurons*
- the connections between nodes are *artificial synapses*
- intelligence emerges from many connected processors



Google's NeuroGlancer based on a human brain sample available at https://h01-dot-neuroglancer-demo.appspot.com/. 50,000 cells, connected by 130 million synapses. The 3D model is 1.4 petabytes.

Artificial Neural Networks

We want to build a machine learning prediction, recall:

$$Y = f(X) + \epsilon$$

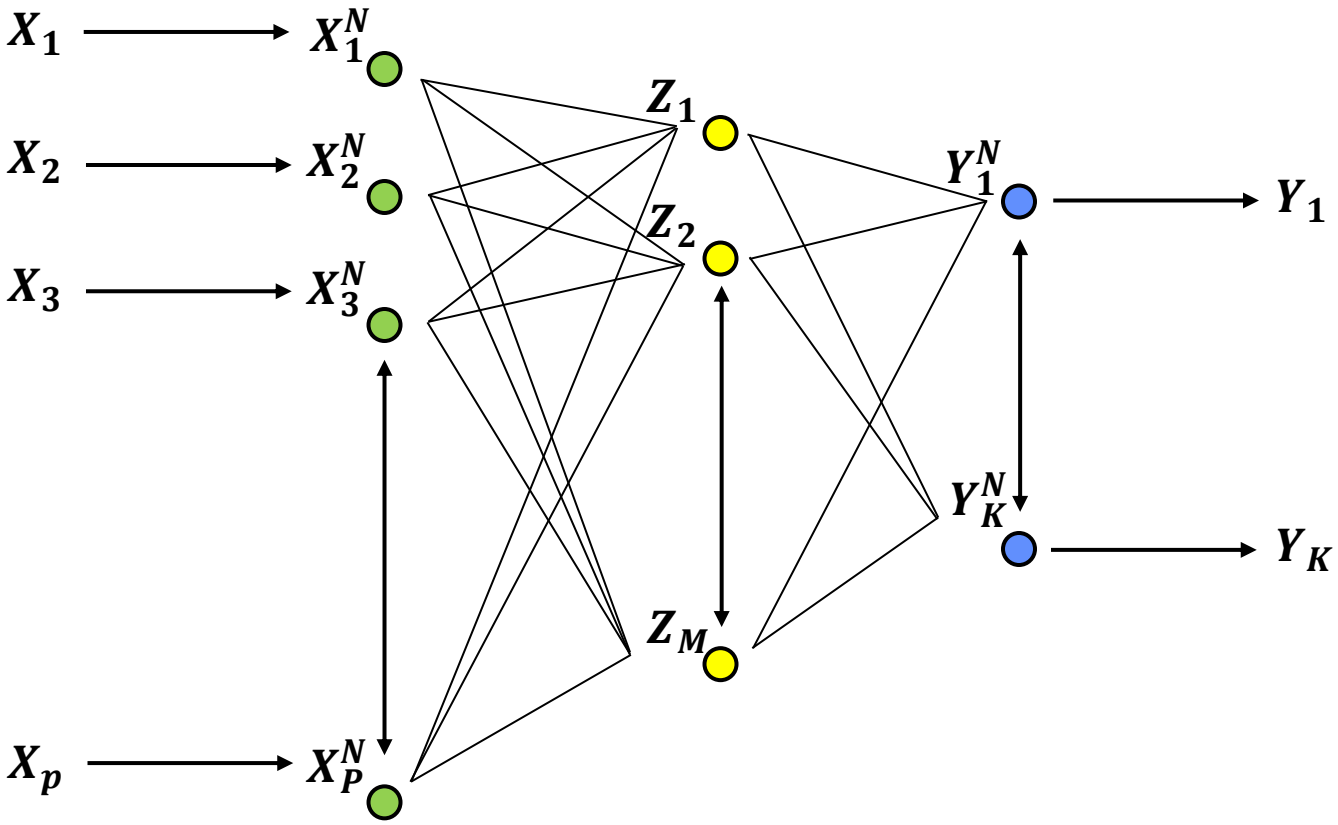
- **Supervised learning** – we will provide training data with predictor features, $X_1, \ldots, X_m$ and response features $Y_1, \ldots, Y_K$, expecting some model prediction error, ϵ

- **Nonlinear** - that can capture / predict with complicated nonlinear, univariate and multivariate relationships

- **Universal Function Approximator (Universal Approximation Theorem)**
  - ability to learn any possible function shape of f over an interval
  - the theorem does not address the process of fitting the weights, just indicates the model is possible
  - developed for an arbitrary wide (single hidden layer) by Cybenko (1989), and arbitrary depth by Lu and others (2017).

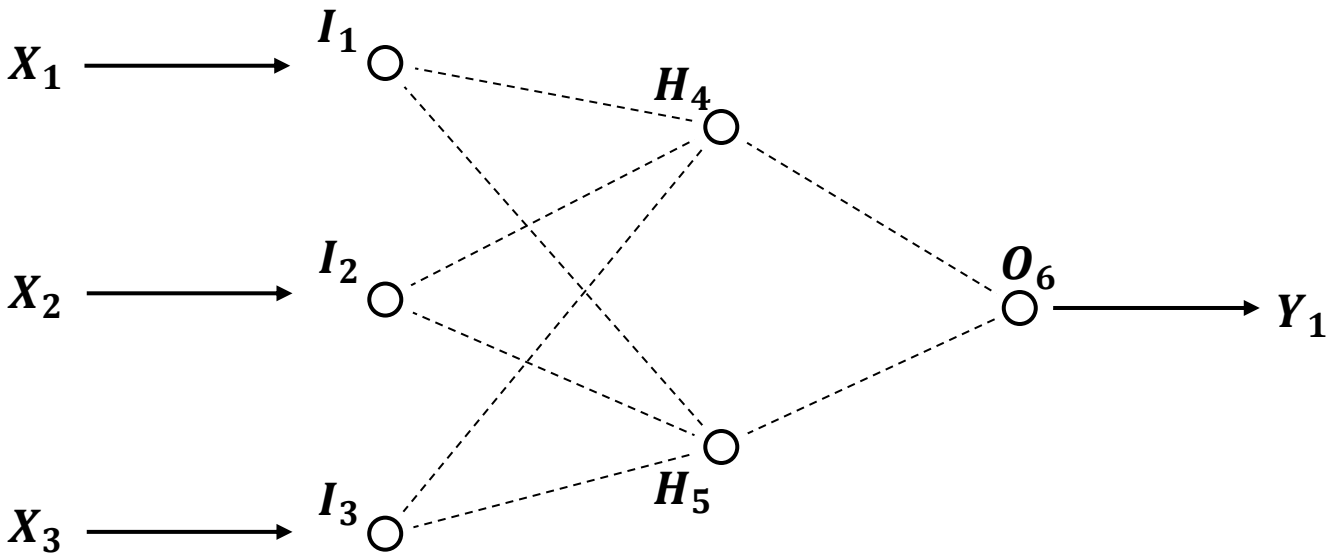
Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



Dissecting a Simple Artificial Neural Networks

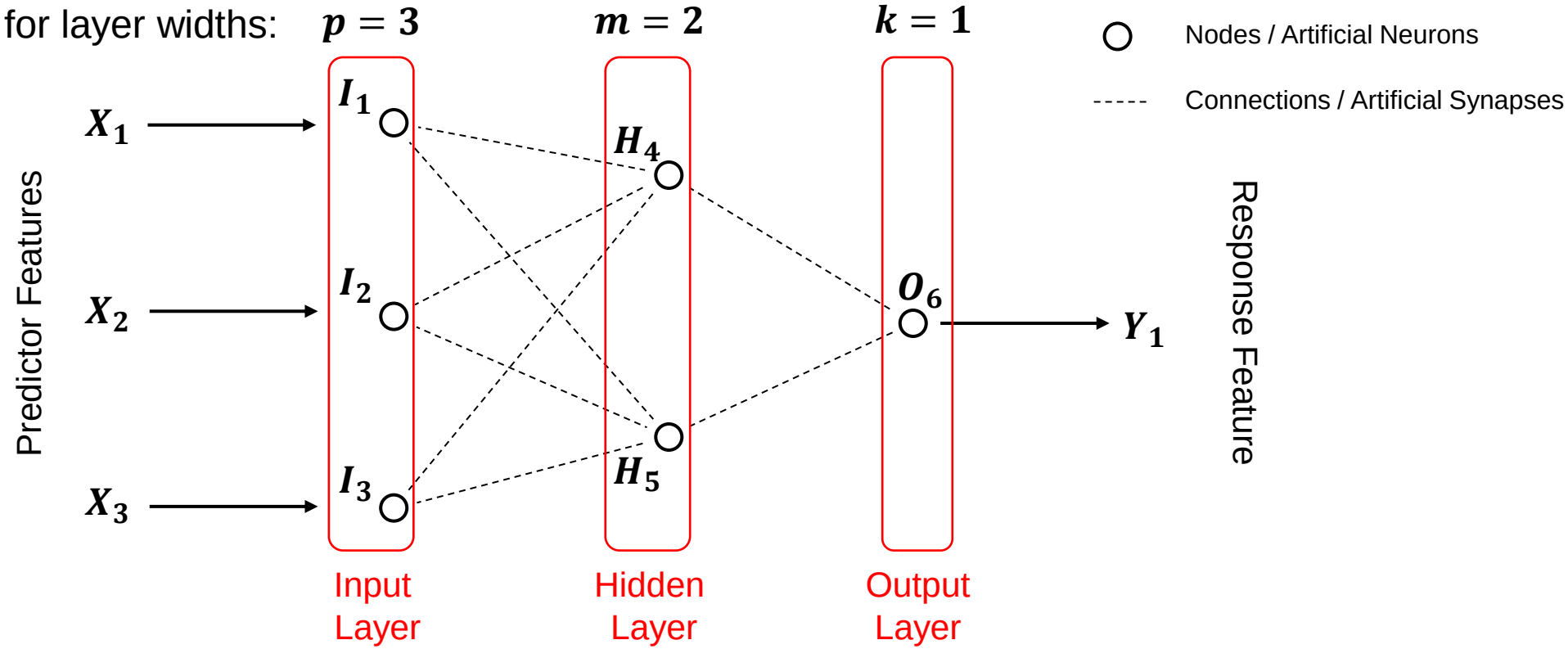
Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.

Typical index for layer widths: $p = 3$ $m = 2$ $k = 1$

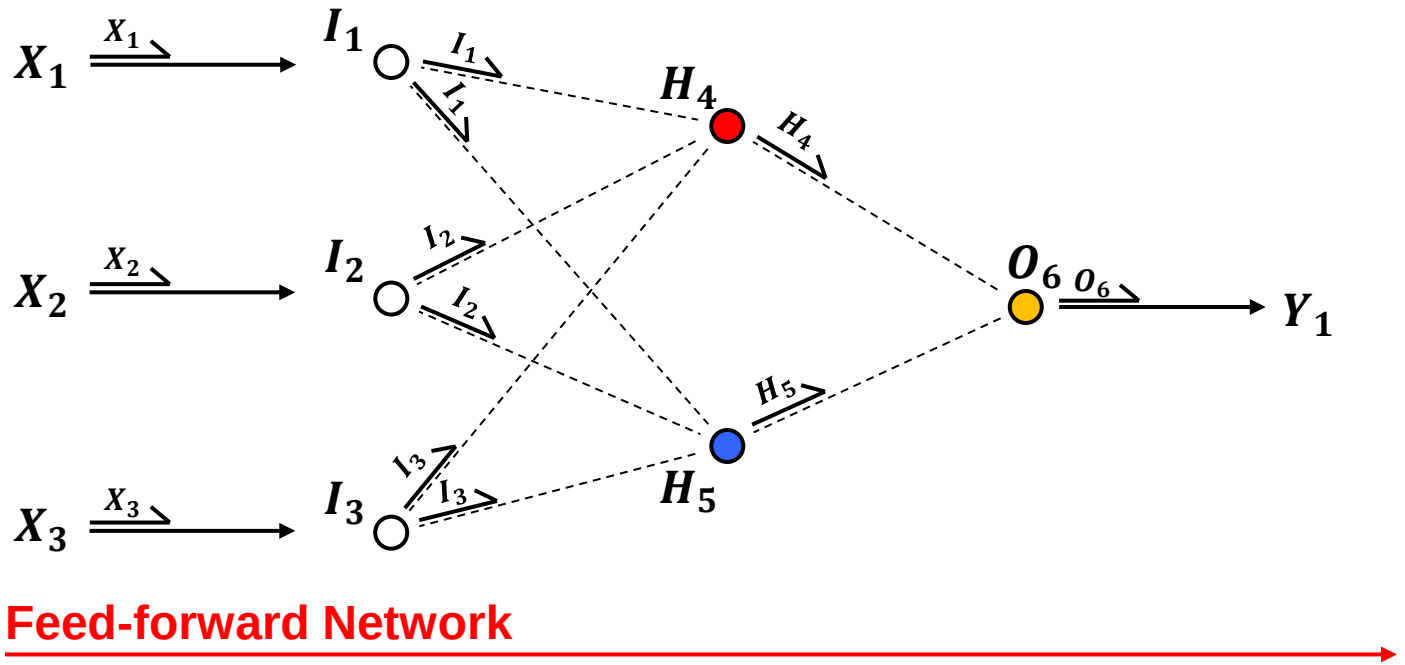


Let's start by defining the parts of this system.

Note, **Deep Learning** is the use of more than one hidden layer

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.

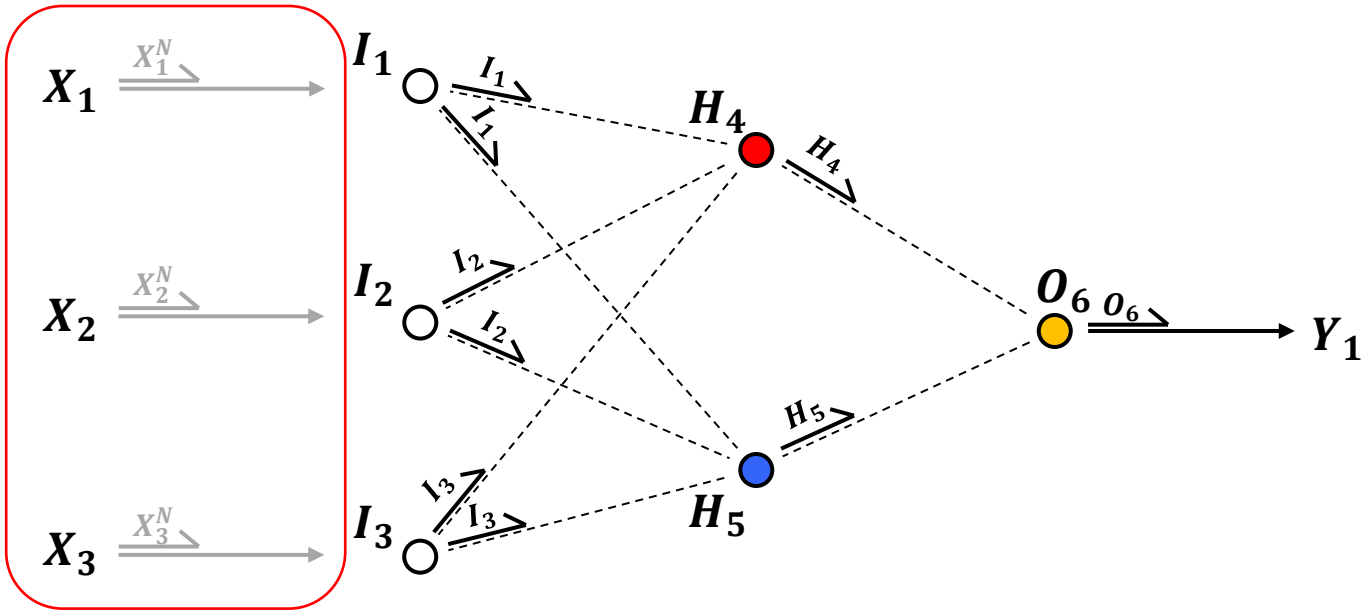


Feed-forward – all information flows from left to right, input to output. Each node sends the same signal to the connected nodes in the next layer.

Fully connected – all nodes connected to all nodes in the next layer.

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



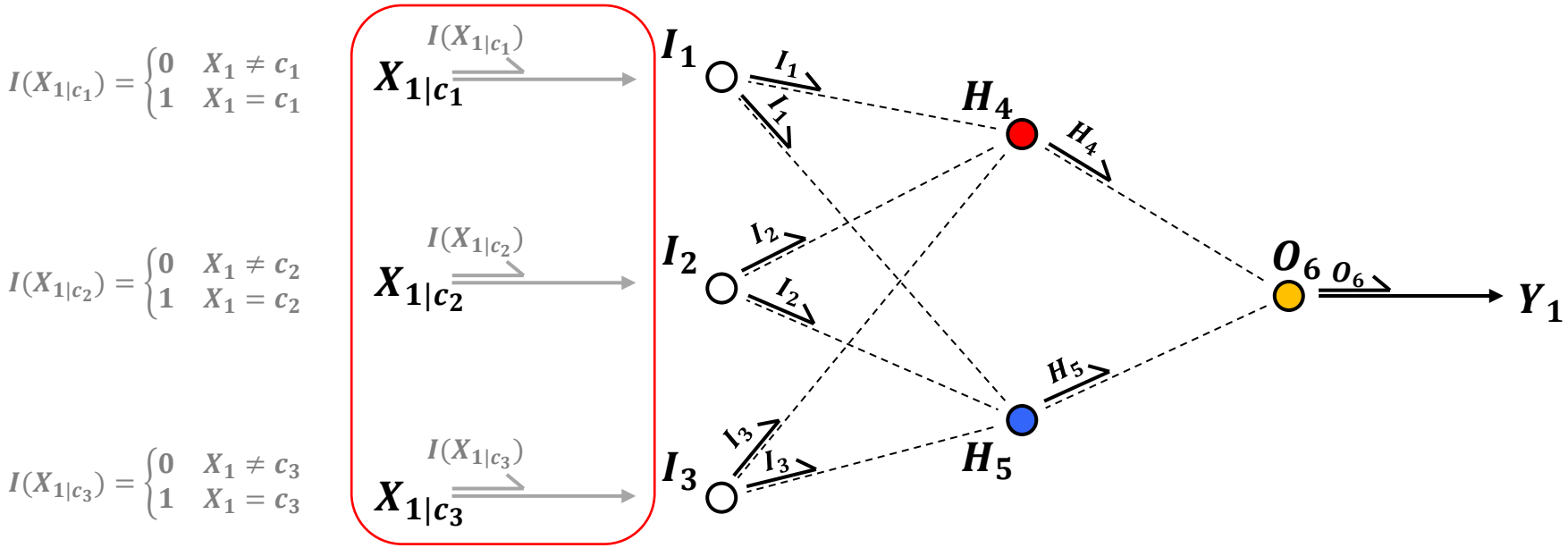
Inputs for Continuous Predictor Features X_i X_i^N I_i

The input features are normalized and pass directly to the input nodes, 1 input node per feature.

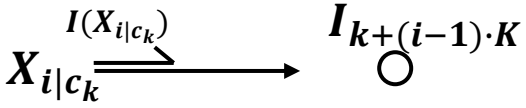
- Min / max normalization to a range $[-1,1]$ or $[0,1]$ to improve activation function sensitivity
- Also, removes the influence of scale differences in predictor features, and avoid "zig-zag" solutions

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



Inputs for Categorical Predictor Features



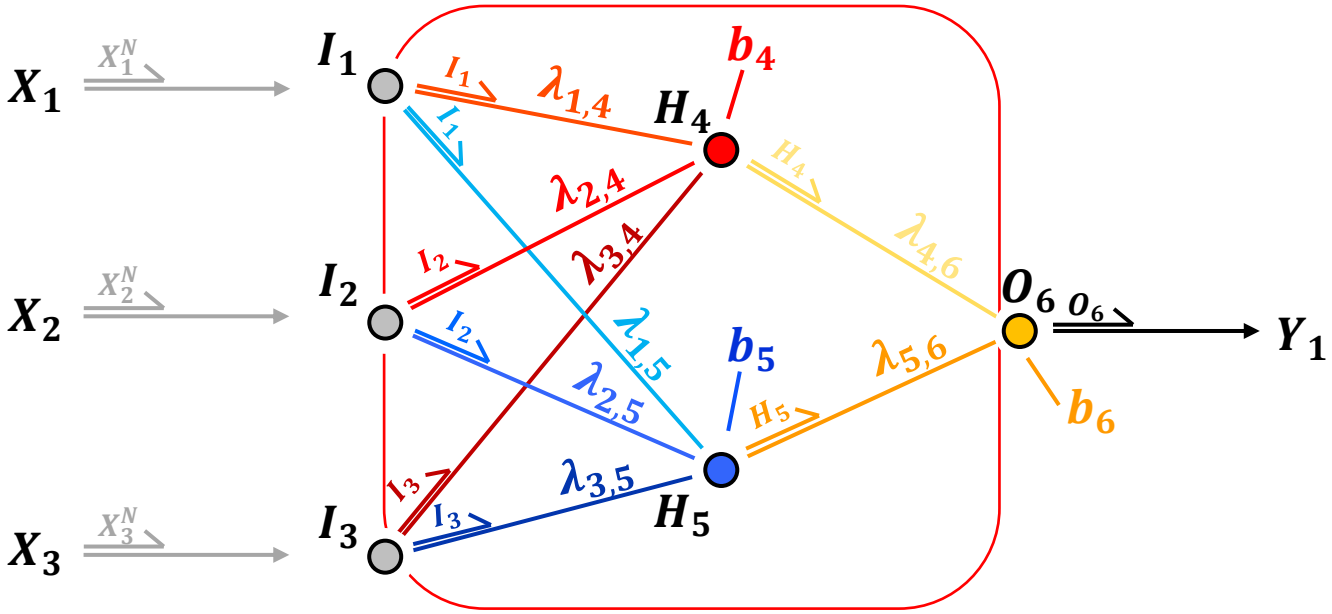
$k + (i-1) \cdot K$ is the input node index given sort on category first and then feature.

One input node per each category for each predictor feature, after one-hot-encoding of the feature.

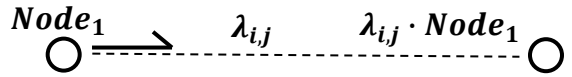
- Indicator operator, 1 if the specific category, 0 otherwise, avoids implicit assumption of ordering if categorical index applied with a single input node for each feature.

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



Artificial Synapses / Connections

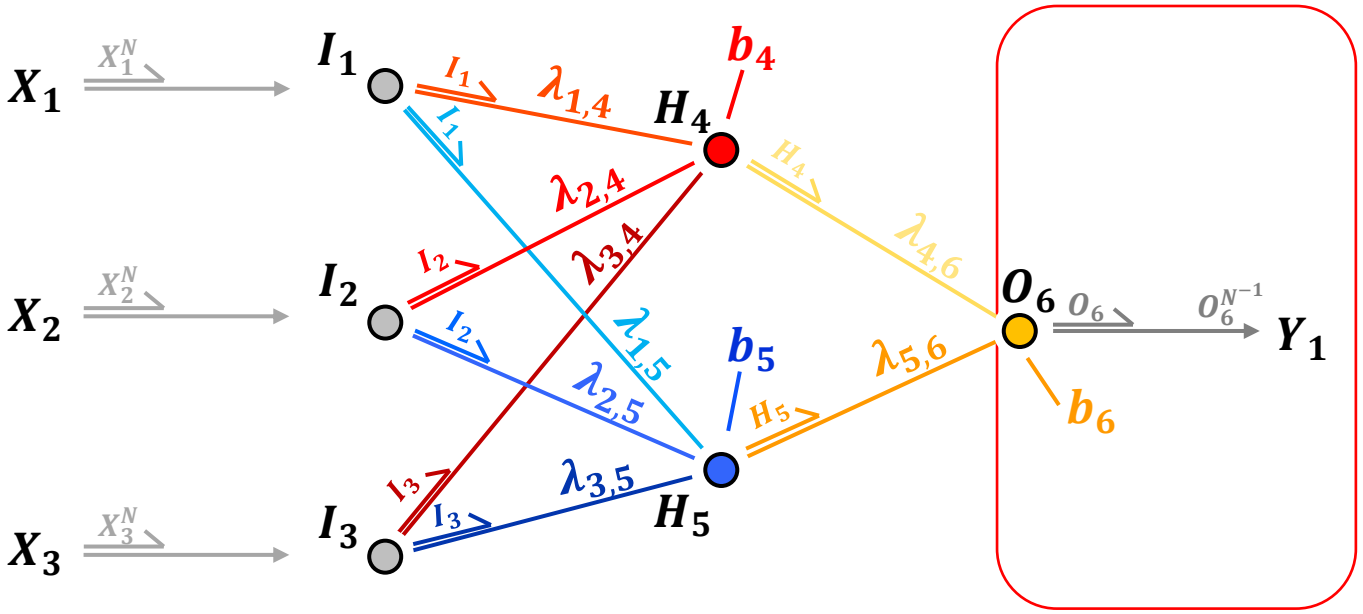


Connections between the nodes of each layer have trainable weights, $\lambda_{i,j}, i$ is previous node, j is next node

- The signal passed from i node to j node, into the next node is weighted, $\lambda_{i,j} \cdot Node_i$ for example, $\lambda_{1,4} \cdot I_1$

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



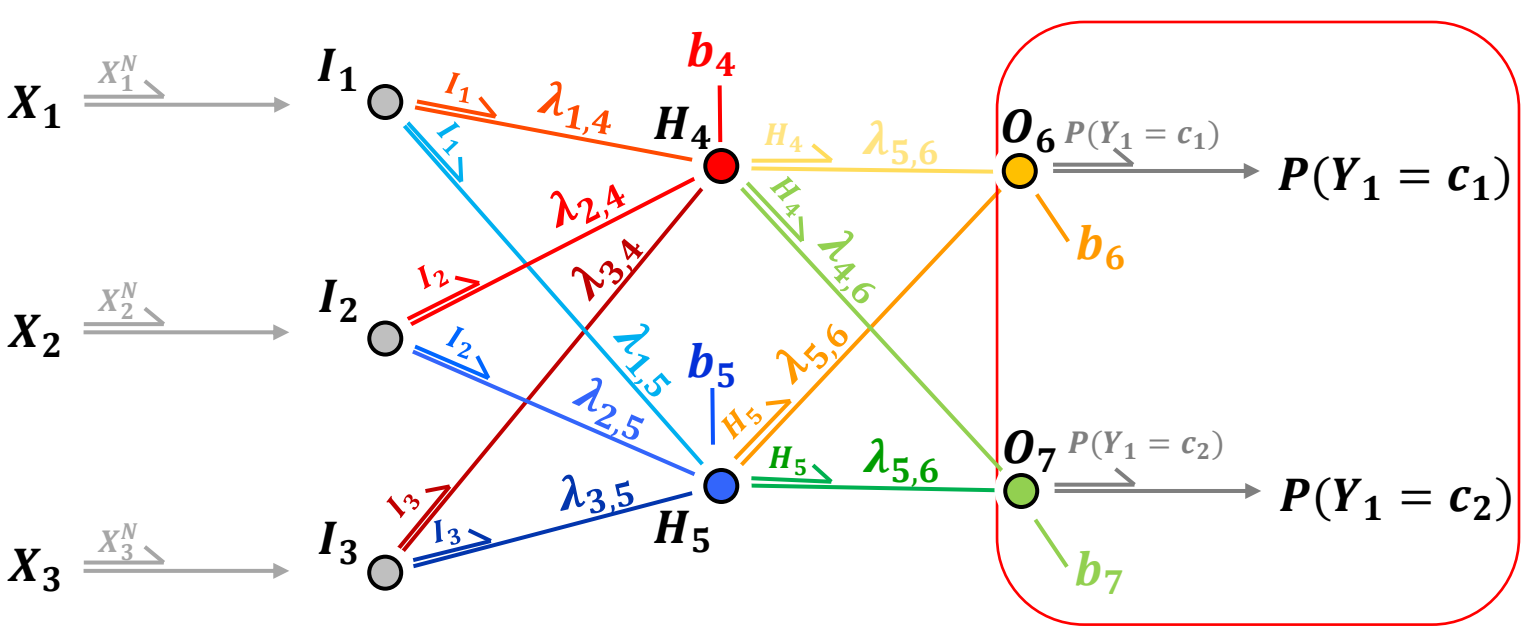
For Prediction, Outputs are Continuous Response Feature(s)

One node per response feature, back transformation from normalized to original response feature(s)

- Min / max normalization is applied to response features also to a range $[-1,1]$ or $[0,1]$ to improve activation function sensitivity

Dissecting a Simple Artificial Neural Networks

Let's build a neural net, **single hidden layer, fully connected, feed-forward neural network**.



Training Data Response
One-hot encoded

$$I(Y_{1|c_1}) = \begin{cases} 0 & Y_1 \neq c_1 \\ 1 & Y_1 = c_1 \end{cases}$$

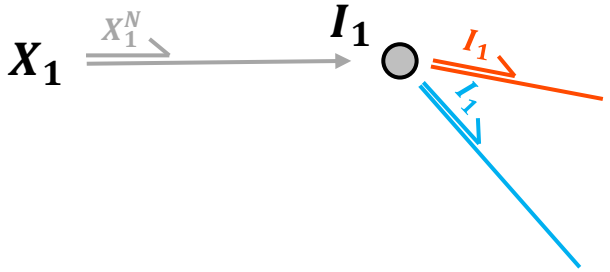
$$I(Y_{1|c_2}) = \begin{cases} 0 & Y_1 \neq c_2 \\ 1 & Y_1 = c_2 \end{cases}$$

For Classification, Outputs are Categorical Response Feature(s)

One output node per each category for each response feature

- The output nodes report the probability of each category, honors non-negative and sum to one probability constraints, training data response feature is one-hot encoded (indicator transform) (like categorical input)

Dissecting a Simple Artificial Neural Networks



Inside Input Layer Nodes

Input nodes just pass the input from the features, $I_j = X_j^N$,

- normalized continuous predictor feature value

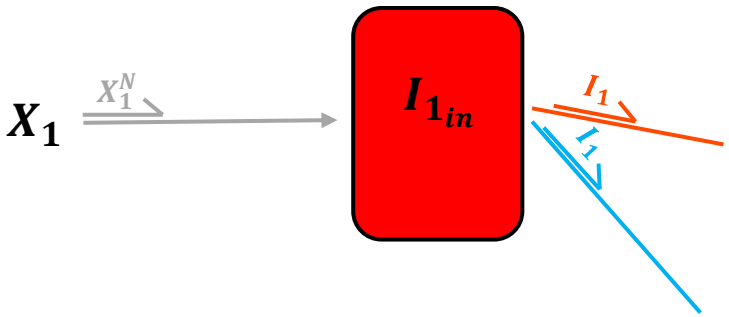
Recall

$$I_j = X_j^N = N(X_j)$$

- or one-hot-encoding single value [0 or 1] for categorical prediction features

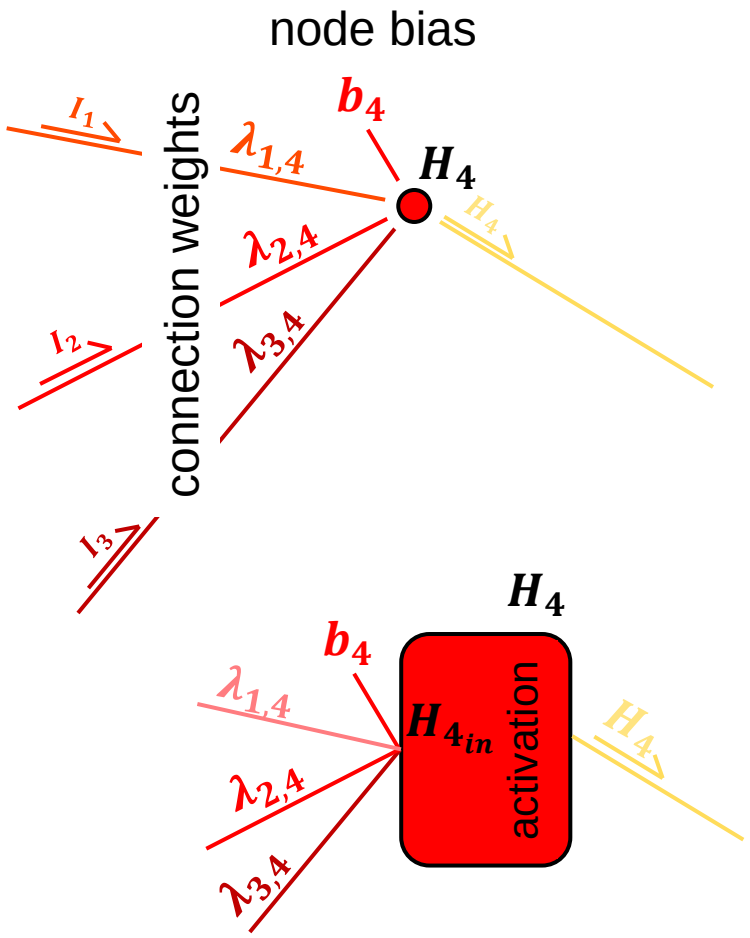
Recall

$$I_j = X_j^N = \begin{cases} 1, & if\ X_j = c_j \\ 0, & otherwise \end{cases}$$



into the hidden layer.

Dissecting a Simple Artificial Neural Networks



$H_{4_{in}}$ is preactivation and H_4 is after action, node output.

Inside Hidden Layer Nodes

take linearly weighted combinations of inputs and then ***nonlinearly transform*** the result, this transform is call the **activation function,** α.

- A very simple processor!
- Through many interconnected nodes we gain a very flexible predictor, emergent ability to characterize complicated, nonlinear patterns.

Prior to Activation

$$H_{j_{in}} = \sum_{i=1}^{p} \lambda_{i,j} \cdot I_i + b_j$$

After Activation

$$H_j = \alpha\left(H_{j_{in}}\right)$$

nonlinearly transformed linear combination of the inputs $I_1,\ldots, I_p$ plus bias term, b_j

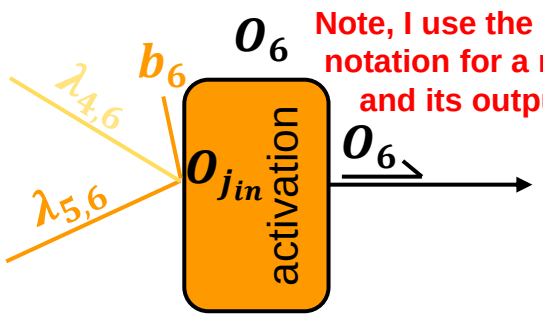
$$H_j = \alpha\left(b_j + \lambda_j^T X\right)$$ for vector notation

For example, $H_4 = \alpha\left(\lambda_{1,4} \cdot I_1 + \lambda_{2,4} \cdot I_2 + \lambda_{3,4} \cdot I_3 + b_4\right)$

Dissecting a Simple Artificial Neural Networks

Inside Output Layer Nodes

take linearly weighted combinations of inputs and apply an **activation function,** α.



Note, I use the same notation for a node and its output.

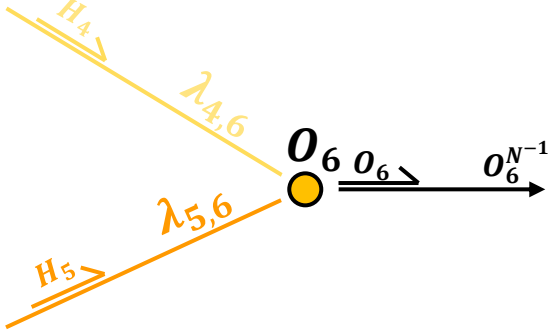
Prior to Activation or Preactivation

$$O_{j_{in}} = \sum_{j=1}^{m} \lambda_{i,j} \cdot H_i + b_j$$

After Activation or Postactivation or Node Output

$$O_j = \alpha(O_{j_{in}})$$

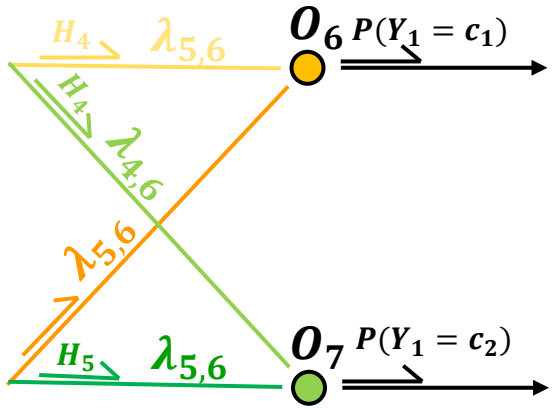
For regression:



linear or identity activation function

$$O_j = \alpha(O_{j_{in}}) = O_{j_{in}}$$

For classification:



Softmax activation function

Non-negative

$$O_j = g_k(O_{j_{in}}) = \frac{e^{O_{j_{in}}}}{\sum_{l=1}^{K} e^{O_{l_{in}}}}$$

Sum to one

Examples for Node, O_6:

$$O_6 = O_{6_{in}}$$

$$O_6 = \frac{e^{O_{6_{in}}}}{e^{O_{6_{in}}} + e^{O_{7_{in}}}} \qquad O_7 = \frac{e^{O_{7_{in}}}}{e^{O_{6_{in}}} + e^{O_{7_{in}}}}$$

Artificial Neural Networks Parameters

The Model Parameters, ϑ, in the Neural Network

For every connection there is a weight:

$$\lambda_{I_{1,\ldots,p},H_{1,\ldots,m}} \text{ and } \lambda_{H_{1,\ldots,m},O_{1,\ldots,k}}$$

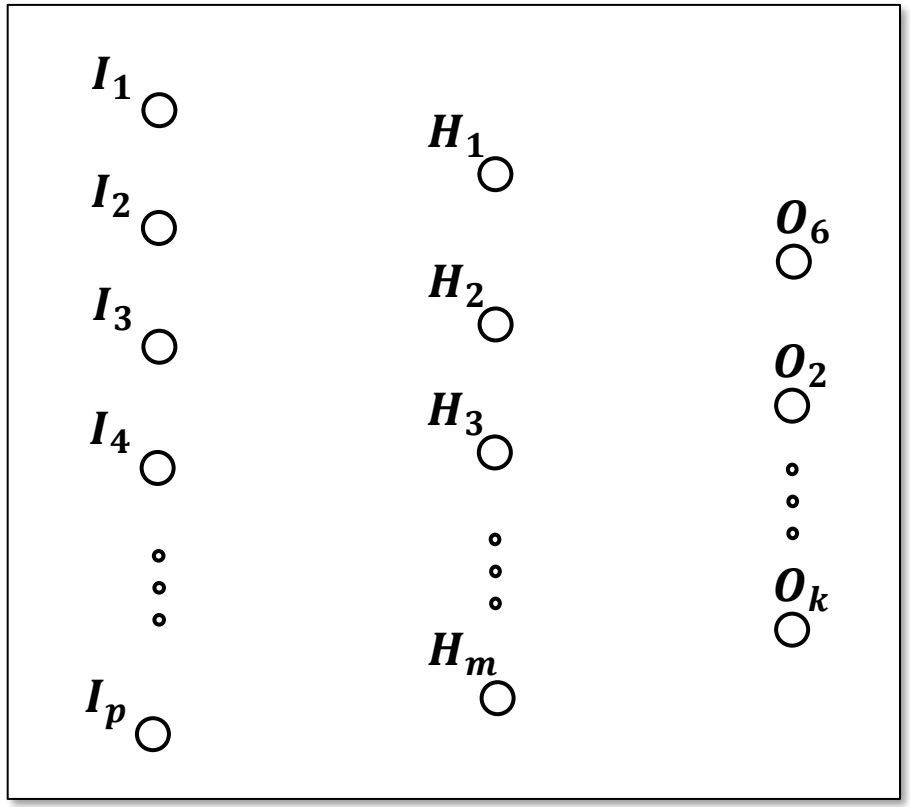
with full connectivity the number of weights is $p \times m$ and $m \times k$

At each node there is a bias term (the constant)

$$b_{\boldsymbol{H_{1,\ldots,m}}} \text{ and } b_{\boldsymbol{O_{1,\ldots,k}}}$$

number of model parameters, $|\vartheta| = p \times m + m \times k + m + k$

- assuming a bias term at each hidden layer node and output layer node, but may use the same bias term in all nodes for each layer.



Input, hidden and output layer nodes with convenient indices.

Our network, $p = 3, m = 2, k = 1.$

$|\vartheta| = p \times m + m \times k + m + k$

$|\vartheta| = 3 \times 2 + 2 \times 1 + 2 + 1 = 11$

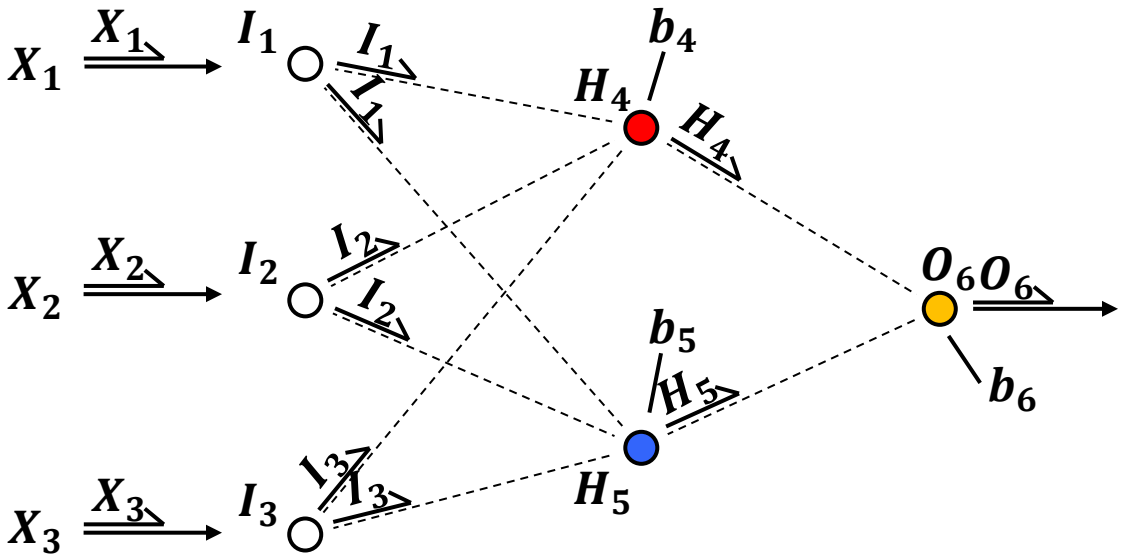
Artificial Neural Networks
Activation Functions

The Activation Function is a nonlinear transform of the linear combination of the inputs to a node.

- introduce non-linear properties to the network
- without the activation function we would have linear regression, the entire system collapses
- increases the power to learn complicate patterns
- also known as a ***transfer function***

$$Y_1 = \lambda_{4,6} \cdot \left(\lambda_{1,4} \cdot X_1 + \lambda_{2,4} \cdot X_2 + \lambda_{3,4} \cdot X_3 + b_{H_4}\right) + \lambda_{5,6} \cdot \left(\lambda_{1,5} \cdot X_1 + \lambda_{2,5} \cdot X_2 + \lambda_{3,5} \cdot X_3 + b_{H_5}\right)$$

Without Activation



$$Y_1 = \left(\lambda_{4,6} \cdot \lambda_{1,4} + \lambda_{5,6} \cdot \lambda_{1,5}\right) \cdot X_1 + \left(\lambda_{4,6} \cdot \lambda_{2,4} + \lambda_{5,6} \cdot \lambda_{2,5}\right) \cdot X_2 + \left(\lambda_{4,6} \cdot \lambda_{3,4} + \lambda_{5,6} \cdot \lambda_{3,5}\right) \cdot X_3 + \lambda_{4,6} \cdot b_{H_4} + \lambda_{5,6} \cdot b_{H_5}$$

After expanding and grouping like terms,

$$Y_1 = a \cdot X_1 + b \cdot X_2 + c \cdot X_3 + d$$

where a, b, c, d are constants.

Neural networks without activation collapses to a linear regression model!

Artificial Neural Networks
Activation Functions

Here's some common activation functions:

preactivation x_{in} $O_{6_{in}}$ O_6 O_6 x_{out} postactivation

- Sigmoid or Logistic

$$x_{out} = \alpha(x_{in}) = \frac{1}{1 + e^{-x_{in}}}$$

$$\alpha'(x) = x_{out} \cdot (1 - x_{out})$$

- Tanh – hyperbolic tangent

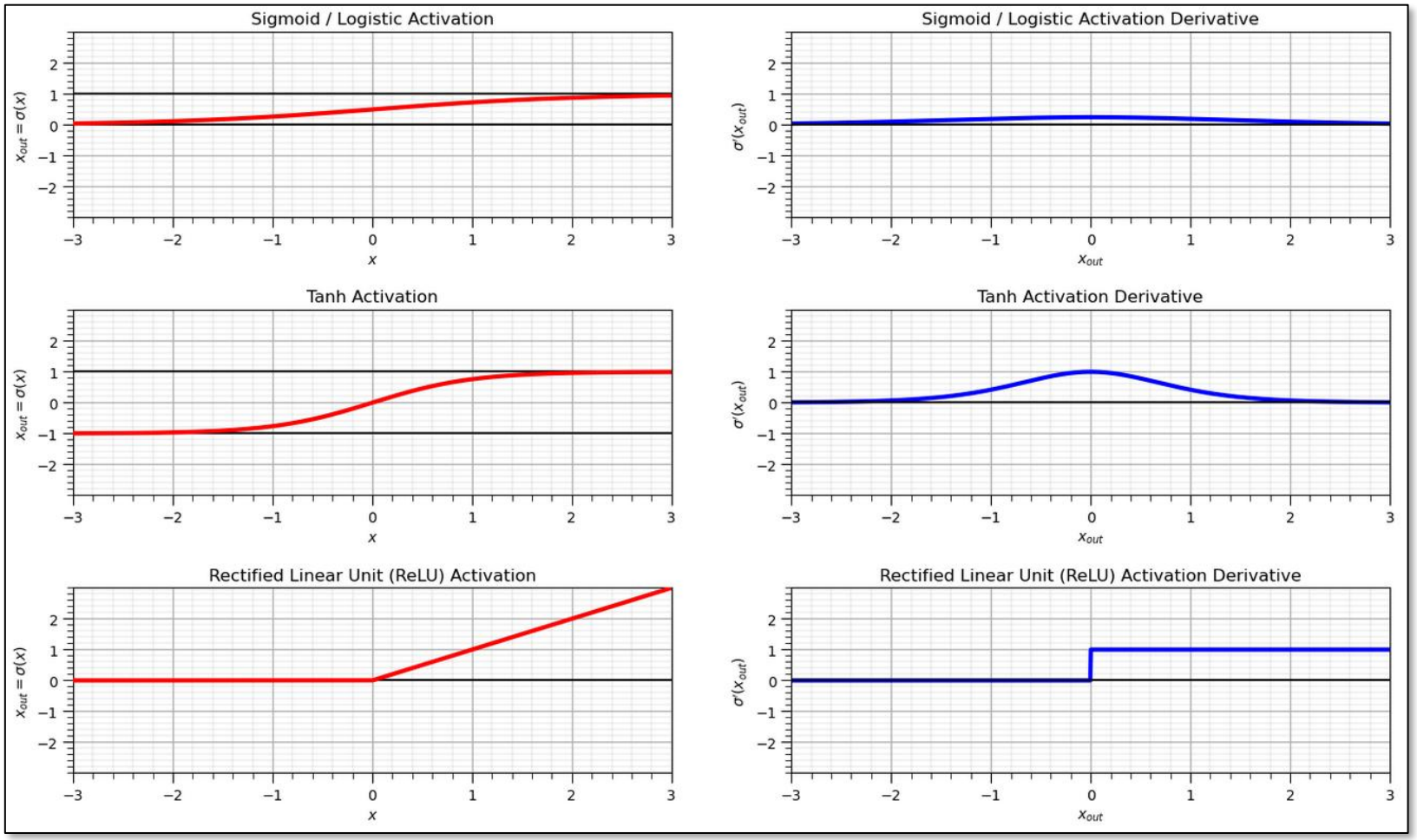
$$x_{out} = \alpha\,(x_{in}) = \frac{e^{x_{in}} - e^{-x_{in}}}{e^{x_{in}} + e^{-x_{in}}}$$

$$\alpha'(x) = 1 - {x_{out}}^2$$

- ReLU – rectified linear units

$$x_{out} = \alpha(x_{in}) = \max(0, x_{in})$$

$$\alpha'(x) = \begin{cases} 0 & x_{out} \leq 0 \\ 1 & x_{out} > 0 \end{cases}$$

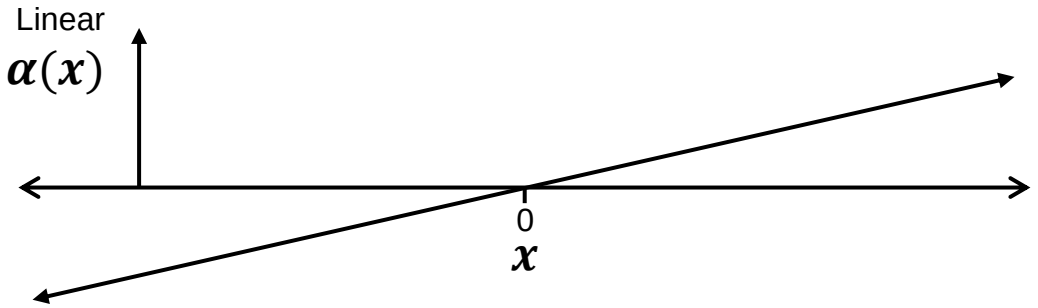


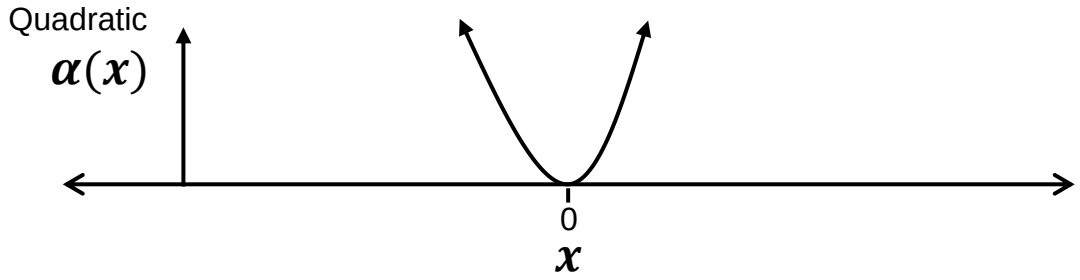
Common activation functions and associated derivatives, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

Artificial Neural Networks
Activation Functions

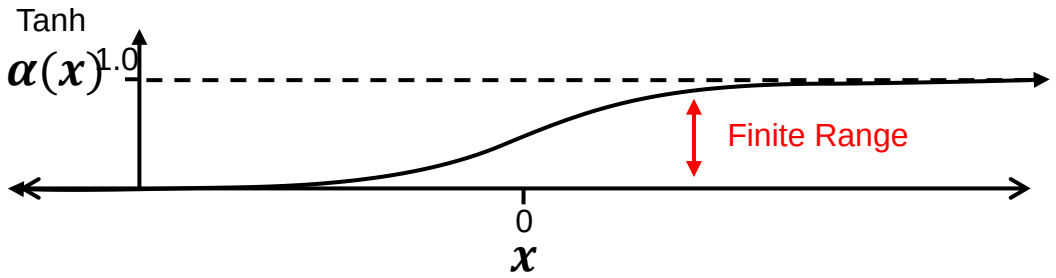
Some Considerations for selecting activation functions:

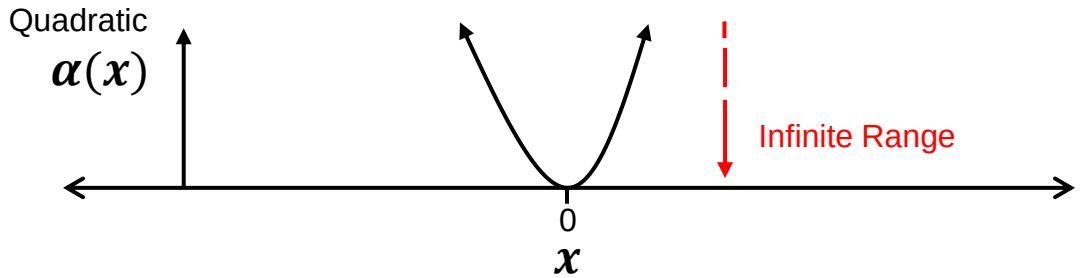
- **Nonlinear** – required to impose nonlinearity into the predictor. Proved to be a universal function approximator if at least 1 hidden layer (Cybenko, 1989).





- **Range** – finite for more stability gradient-based learning (prevents exploding activations and gradients), infinite for more efficient training (avoids vanishing gradient)(but requires a slower learning rate)

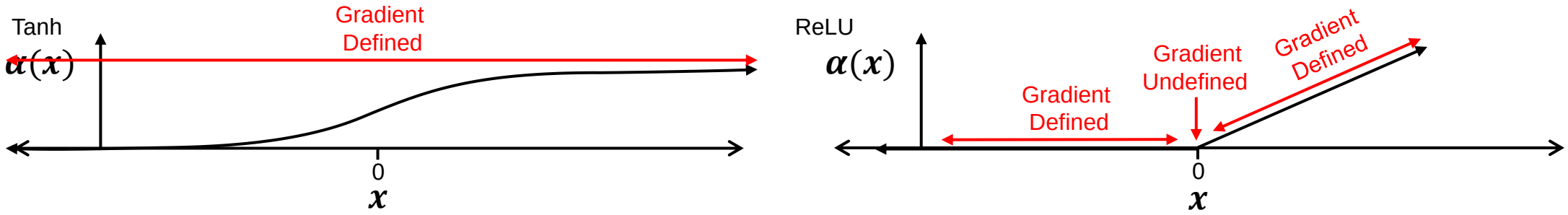




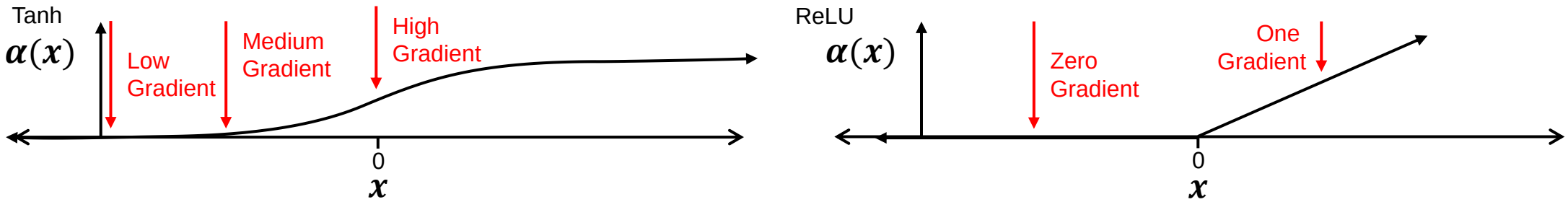
Artificial Neural Networks
Activation Functions

Some Considerations for selecting activation functions:

- **Continuously Differentiable** – required for stable gradient-based optimization



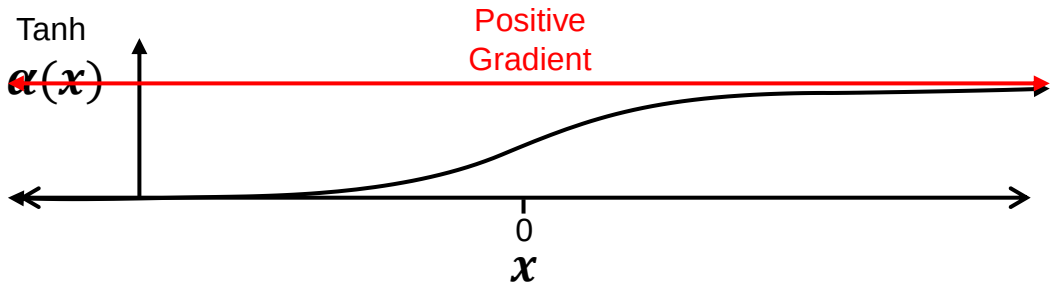
- **Smooth Functions** – gradual change, small continuous changes in gradients reduces risk of unstable and oscillatory training, non-smooth may cause early gradient imbalances.

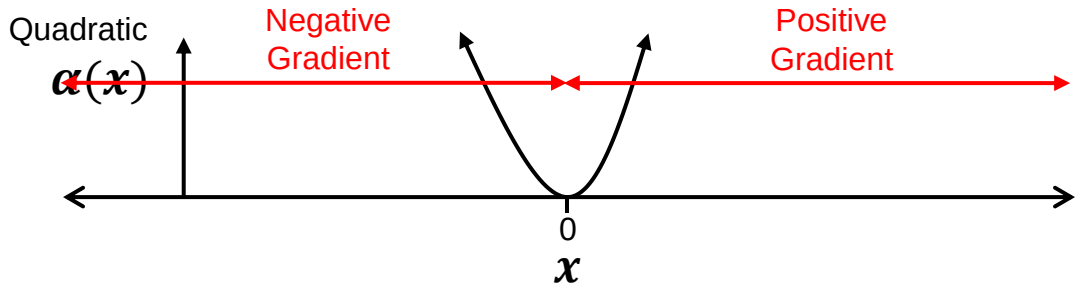


Artificial Neural Networks
Activation Functions

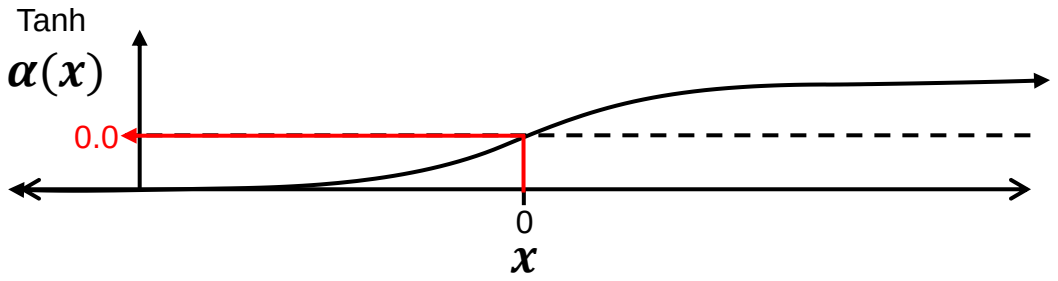
Some Considerations for selecting activation functions :

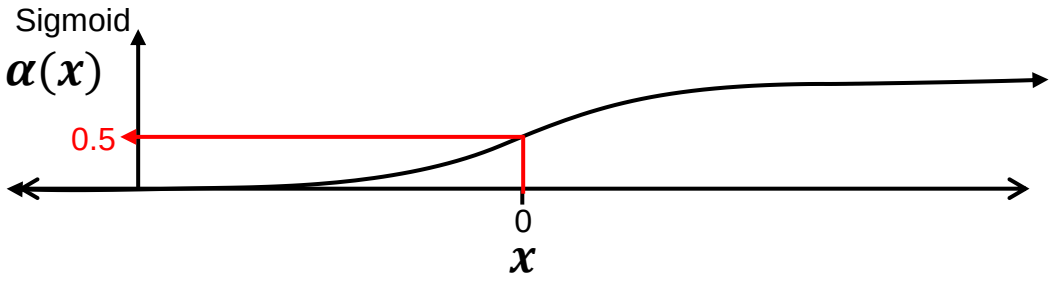
- **Monotonic Derivative** – derivative does not change sign that may cause oscillation in training





- **Approximates Identity at the Origin** $(\alpha(0) = 0)$ – learns efficiently with the weights initialized as small random values, (e.g., ReLU and Tanh)





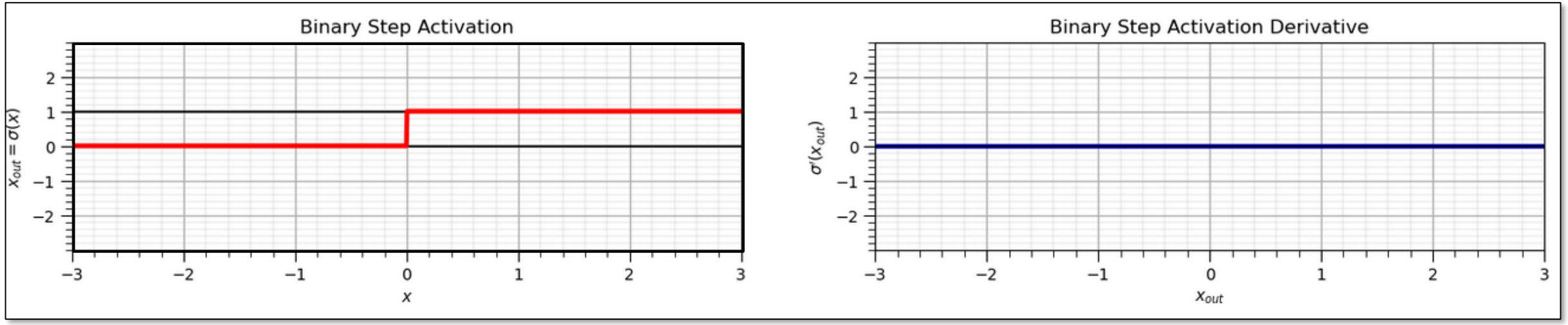
Artificial Neural Networks
Activation Functions

Binary Step Activation:

- Only two possible signals, 0 or 1
- Gradient of zero if $x \neq 0$, and no gradient if $x = 0$; therefore, cannot be applied with gradient-based optimization, can not used standard backpropogation, instead surrogate gradients borrowed from sigmoid!
- Single layer perceptron neural network is only capable of linear decision boundaries
- The bias term moves the boundary and doesn't change the orientation

$$\alpha(x) = \begin{cases} 0 & if\ x \leq 0 \\ 1 & else \end{cases}$$

$$\alpha'(x) = 0.0, \text{ if } x \neq 0.0$$



Binary step activation and associated derivative, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

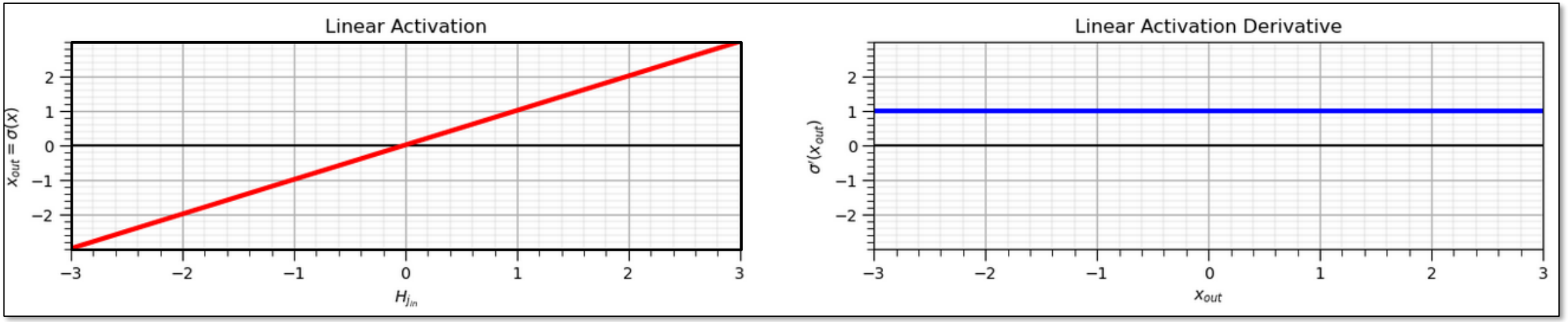
Artificial Neural Networks Activation Functions

Linear / Identity Activation:

- Constant derivative, no relation to x, cannot use for gradient-based optimization as we can't calculate the impact of weights on loss function
- If all activation functions are linear, artificial neural network collapses to linear regression
- Generally used for the output node of a regression model only

$$\alpha(x) = x_{out}$$

$$\alpha'(x) = 1.0$$



Linear activation and associated derivative, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

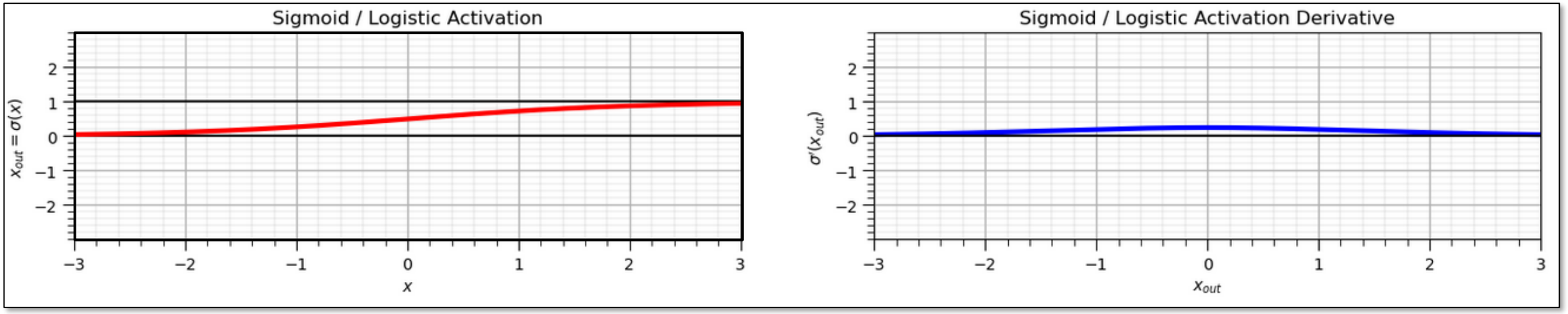
Artificial Neural Networks
Activation Functions

Logistic/Sigmoid/Soft Step Function:

- Smooth gradient for gradient-base optimization
- Computationally cheap derivative calculation,
- Vanishing gradient, for large $|x|$ gradient $\rightarrow$0, slow learning
- Outputs are not centered on 0, but has a convenient probability interpretation

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\alpha'(x) = x_{out} \cdot (1 - x_{out})$$



Sigmoid activation and associated derivative, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

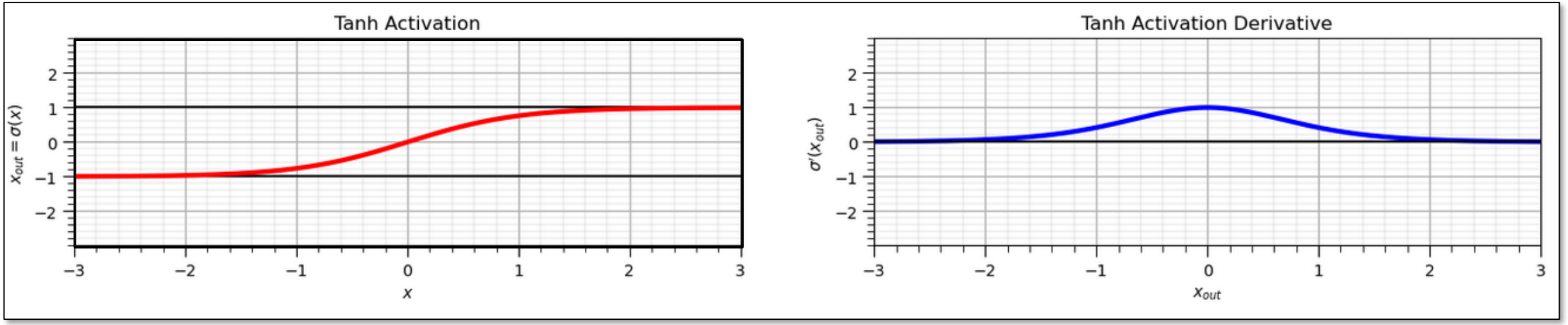
Artificial Neural Networks
Activation Functions

Tanh Function:

- Smooth gradient for gradient-base optimization
- Computationally expensive derivative calculation
- Vanishing gradient, for large $|x|$ gradient $\rightarrow$0, slow learning
- Outputs are centered on 0.0 for ease of weight initialization

$$\alpha(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\alpha'(x) = 1 - x_{out}^{2}$$



Tanh activation and associated derivative, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

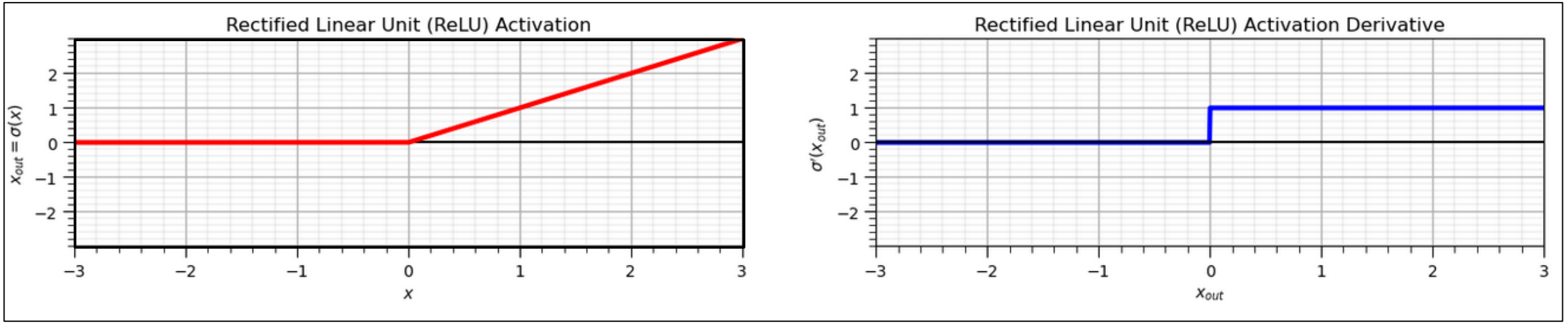
Artificial Neural Networks
Activation Functions

Rectified Linear Unit (ReLU) Function:

- Computationally efficient, results in sparse networks that converge quickly
- Piecewise differentiable allowing for back propagation, 0 is a speedbump is a wide road
- Dying ReLU problem, at $x \leq 0$ no learning, consider leaky ReLU

$$\alpha(x) = \max(0, x)$$

$$\alpha'(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$



ReLU activation and associated derivative, file is SubsurfaceDataAnalytics_ANN_ActivationFunctions.

Artificial Neural Networks Model Parameters Initialization

How Do We Initialize the Model Parameters

- In general, random weights specified based on the ANN design and activation function (stochastic gradient descent random start)

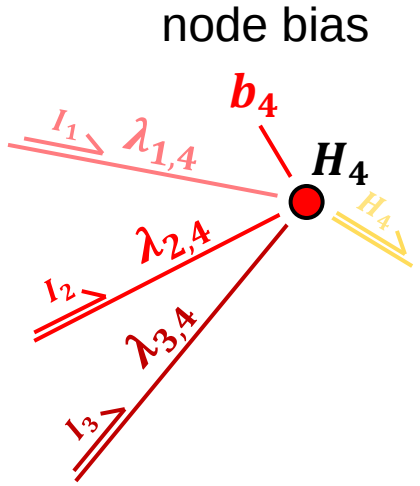
$$\lambda_{i,j} = F^{-1}_{U\left[-\frac{1}{\sqrt{p}},\frac{1}{\sqrt{p}}\right]}\left(p^{\ell}\right)$$ Xavier Weight Initialization

where p is the number of inputs and p^{ℓ} is the ℓ realization cumulative probability and $F^{-1}_{U[min\,,max]}$ is the inverse of the uniform CDF.

$$\lambda_{i,j} = F^{-1}_{U\left[-\frac{1}{\sqrt{p}+k},\frac{1}{\sqrt{p+k}}\right]}\left(p^{\ell}\right)$$ Normalized Xavier Weight Initialization

where we add k, the number of outputs.

- Hidden layer bias terms may be initialized as 0.0.
- Output layer bias terms may be initialized as the mean of the response for regression models.



Example for Node H_4

Xavier $\lambda_{i,4} = F^{-1}_{U\left[-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right]}\left(p^{\ell}\right)$

Normalized Xavier $\lambda_{i,4} = F^{-1}_{U\left[-\frac{1}{\sqrt{3+1}},\frac{1}{\sqrt{3+1}}\right]}\left(p^{\ell}\right)$

Artificial Neural Networks Model Parameter Training

How Do We Train the Neural Net?

Iterate over all data and then epochs (epoch is 1 iteration over all data).

Forward Pass

$$Y = f(X_1, X_2, X_3)$$

with all the current weights, $\lambda_{i,j}$, and biases, b_j

Start of iteration

with random weights

Calculate the Loss Derivative

in the Prediction, $\frac{\partial \mathrm{L}}{\partial \hat{y}} = \hat{Y} - Y$

note we assume an error loss of $L = \frac{\mathbf{1}}{\mathbf{2}}\left(\hat{Y} - Y\right)^{\mathbf{2}}$

Backpropagate Loss Derivative

$$\frac{\partial \mathrm{L}}{\partial \lambda_{i,j}} = \frac{\partial O_{j_{in}}}{\partial \lambda_{i,j}} \cdot \frac{\partial O_j}{\partial O_{j_{in}}} \cdot \frac{\partial \mathrm{L}}{\partial O_j}$$

identify the impact model parameters on the loss derivative.

Update the Model Parameters

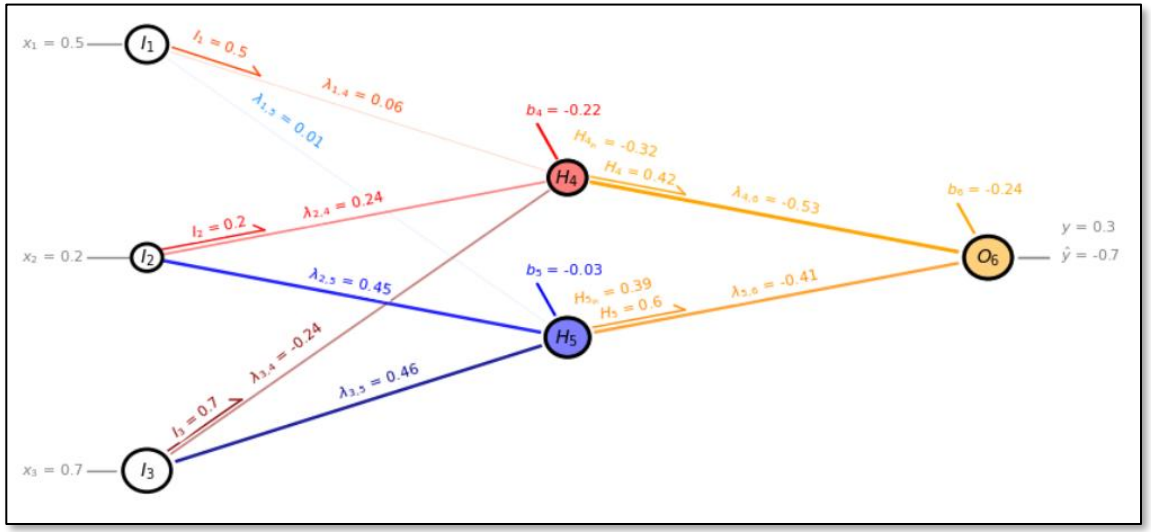
$$\lambda_{i,j} = \eta \cdot \frac{\partial \mathrm{L}}{\partial \lambda_{i,j}} \qquad b_i = \eta \cdot \frac{\partial \mathrm{L}}{\partial b_i}$$

update the model parameters at a defined learning rate.

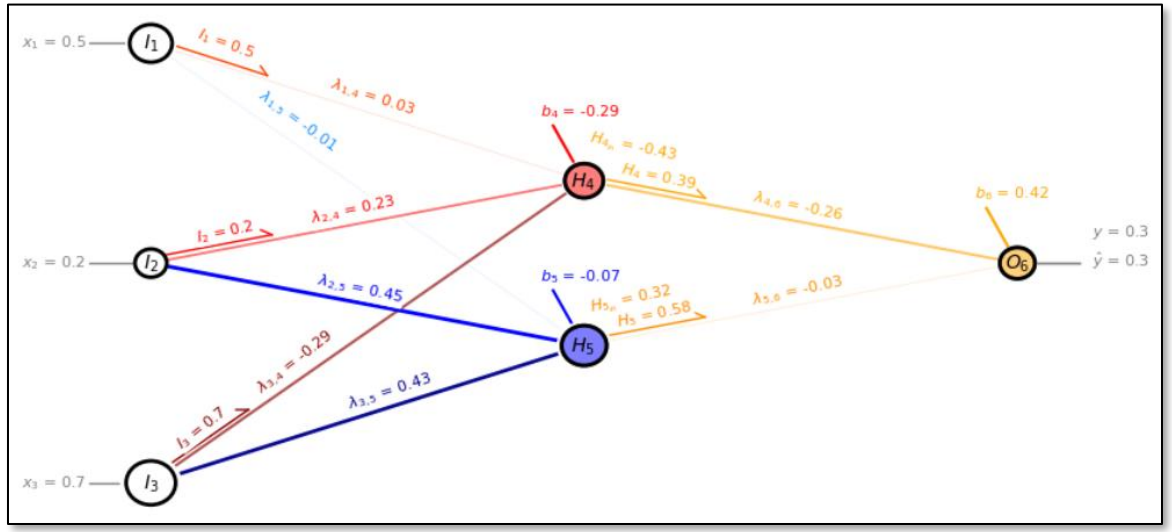
Artificial Neural Networks Forward Pass

Example Forward Passes of Our Artificial Neural Network

Initial Model



Trained Model



Initial random weights and biases (left) and trained weights and biases (right) for an ANN, from Interactive_ANN.ipynb, random seed = 3.

$$Y_1 = \lambda_{4,6} \cdot \alpha\left[\lambda_{1,4} \cdot X_1 + \lambda_{2,4} \cdot X_2 + \lambda_{3,4} \cdot X_3 + b_{H_4}\right] + \lambda_{5,6} \cdot \alpha\left[\lambda_{1,5} \cdot X_1 + \lambda_{2,5} \cdot X_2 + \lambda_{3,5} \cdot X_3 + b_{H_5}\right]$$

Example untrained initial model:

$$-0.7 = -0.53 \cdot \alpha[0.06\ (0.5) + 0.24(0.2) - 0.24(0.7) - 0.22] + -0.41 \cdot \alpha[0.01\ (0.5) + 0.45\ (0.2) + 0.46\ (0.7) - 0.03]$$

$$-0.7 = \qquad -0.53 \cdot \mathbf{0.42} \qquad\qquad -0.41 \cdot \mathbf{0.60}$$

Artificial Neural Networks
Model Parameter Training

The Network Loss function, e.g., L_2 Loss

Given the truth, y, and our prediction, $\hat{y}$, we calculate our **L_2 Loss** as**,**

$$L = \frac{\mathbf{1}}{\mathbf{2}}(\hat{y} - y)^{\mathbf{2}}$$

- we calculate the partial derivative, $\frac{\partial L}{\partial \hat{y}}$, rate of change in loss, L, with respect to change in model estimate, $\hat{y}$ as,

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

- our choice of loss function allows us to use the prediction error as the loss derivative, $\frac{\partial L}{\partial \hat{y}} = \Delta y$.

For now, we demonstrate back propagation of this loss derivative for a single training data sample, y.

- we address multiple samples latter, $y_i, i = 1, \ldots, n$

Artificial Neural Networks Backpropagate Loss Derivative

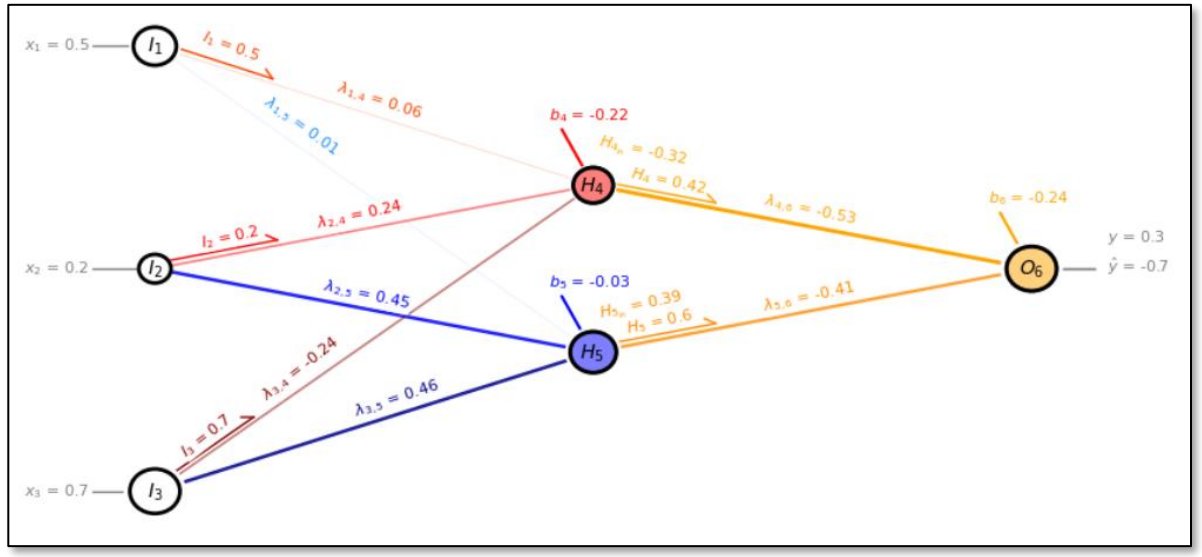
How Do Each Model Parameter Impact this Loss Derivative?

Given $\frac{\partial L}{\partial \hat{y}}$ for our model prediction and our model, to perform gradient descent optimization we need to know the direction and gradient for the model parameters:

- For this prediction case we need:

$$\frac{\partial L}{\partial \lambda_{4,6}}, \frac{\partial L}{\partial \lambda_{5,6}}, \frac{\partial L}{\partial \lambda_{1,4}}, \frac{\partial L}{\partial \lambda_{2,4}}, \frac{\partial L}{\partial \lambda_{3,4}}, \frac{\partial L}{\partial \lambda_{1,5}}, \frac{\partial L}{\partial \lambda_{2,5}}, \frac{\partial L}{\partial \lambda_{3,5}}$$ for all the weights

$$\frac{\partial L}{\partial b_4}, \frac{\partial L}{\partial b_5}, \frac{\partial L}{\partial b_6}$$ for all the biases



Initial random weights and biases for our simple artificial neural network.

For this prediction we need to ‘allocate’ the partial derivative of the loss function backwards through our neural network.

This is back propagation!

and it uses a lot of **partial derivatives** and a lot of **book keeping**.

Training Artificial Neural Networks Prerequisite

Recall the Chain Rule:

Given:

$$\frac{\partial}{\partial x} f\big(g(x)\big)$$

Then:

$$\frac{\partial}{\partial x} f\big(g(x)\big) = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \qquad \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = f'(g(x))g'(x)$$

We can continue:

$$\frac{\partial}{\partial x} f\big(g(h(x))\big) = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial x}$$

We apply the chain rule to perform back propagation,

to step our loss derivative backwards through our network

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative back to the Output Node?

Let's start with the loss derivative,

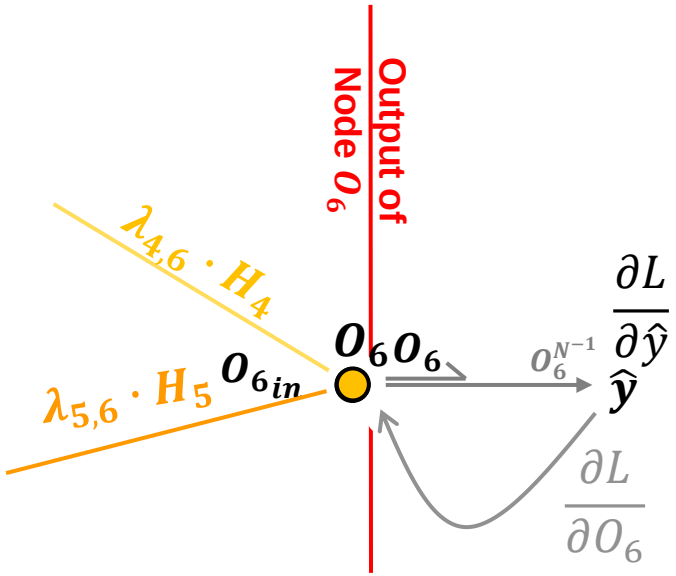
$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

Note our model prediction, $\hat{y}$, is the output from node $\boldsymbol{O_6}$ denoted as $\boldsymbol{O_6}$.

$$\frac{\partial L}{\partial O_6} = \frac{\partial L}{\partial \hat{y}} = \hat{y} - y = O_6 - y$$

So this is our loss derivative backpropagated to the output our output node.

Back propagate at output of output node: $\frac{\partial L}{\partial O_6} = (O_6 - y)$



Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Through a Node?

Let's backpropagate through a node, with this activation,

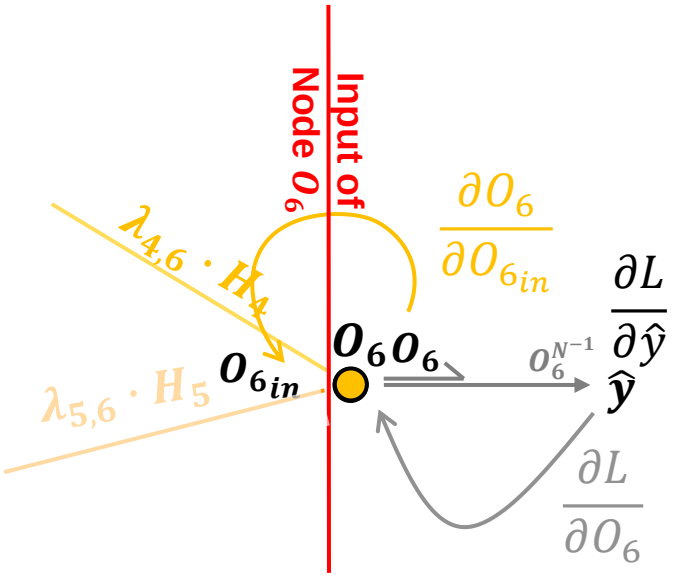
$$O_6 = \sigma(O_{6_{in}}) = O_{6_{in}}$$

identity / linear activation

Take the derivative with respect to $O_{6_{in}}$,

$$\frac{\partial O_6}{\partial O_{6_{in}}} = \frac{\partial (O_{6_{in}})}{\partial O_{6_{in}}} = 1.0$$

Substituting in chain rule,



Back propagate through a node: $\frac{\partial O_6}{\partial O_{6_{in}}} = 1.0$

From Chain Rule: $\frac{\partial L}{\partial O_{6_{in}}} = \frac{\partial O_6}{\partial O_{6_{in}}} \cdot \frac{\partial L}{\partial O_6} = 1.0 \cdot (O_6 - y)$

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Along a Connection to Output of Hidden Layer Node, H_4?

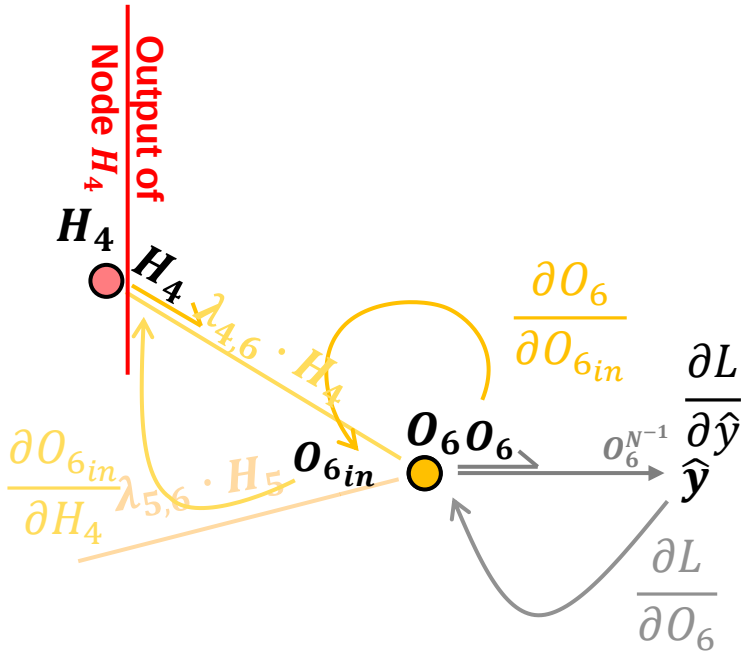
Pre-activation, input to node O_6 we have,

$$O_{6_{in}} = \lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + c$$

We calculate the derivative along the connection as,

$$\frac{\partial O_{6_{in}}}{\partial H_4} = \frac{\partial}{\partial H_4}\left(\lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + b_6\right) = \lambda_{4,6}$$

we backpropagate along a connection by applying the connection weight.



Back propagate along a connection: $\frac{\partial O_{6_{in}}}{\partial H_4} = \lambda_{4,6}$

From Chain Rule: $\frac{\partial L}{\partial H_4} = \frac{\partial O_{6_{in}}}{\partial H_4} \cdot \frac{\partial O_6}{\partial O_{6_{in}}} \cdot \frac{\partial L}{\partial O_6} = \lambda_{4,6} \cdot 1.0 \cdot (O_6 - y)$

Training Artificial Neural Back Propagation

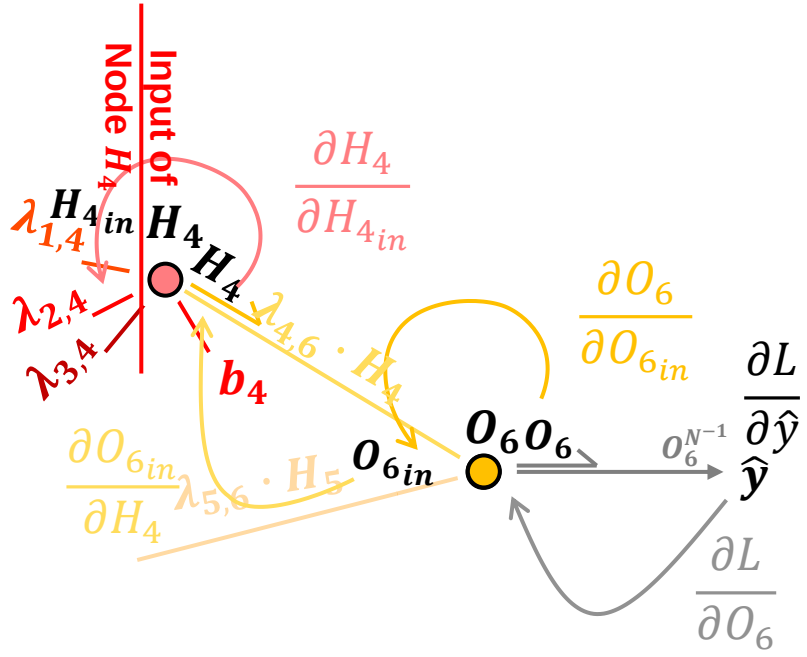
How Do We Backpropagate the Loss Derivative Through a Node?

Let's backpropagate through a node, and for clean logic, let's resolve it as a function of the output rather than the input,

$$H_4 = \sigma\left(H_{4_{in}}\right) = \frac{1}{1 + e^{-H_{4_{in}}}}$$

sigmoid activation

Let's derive the derivative to accomplish this.



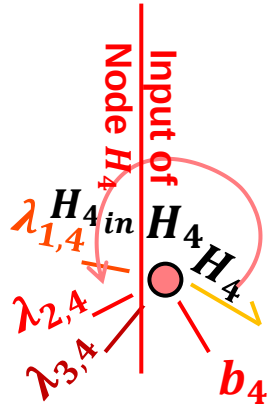
Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Through a Node?

Let's backpropagate through a node, and for clean logic, let's resolve it as a function of the output rather than the input,

$$H_4 = \sigma(H_{4_{in}}) = \frac{1}{1 + e^{-H_{4_{in}}}}$$

sigmoid activation



Take the derivative with respect to $H_{4_{in}}$,

chain rule

$$\frac{\partial H_4}{\partial H_{4_{in}}} = \frac{\partial}{\partial H_{4_{in}}}\left(\frac{1}{1 + e^{-H_{4_{in}}}}\right)$$

set $e^{-H_{4_{in}}} = u$

$$\frac{\partial H_4}{\partial H_{4_{in}}} = \frac{\partial}{\partial H_{4_{in}}}\left(\frac{1}{1+u}\right) = -\frac{u}{(1+u)^2}\frac{\partial u}{\partial H_{4_{in}}}$$

differentiate $u = e^{-O_{6_{in}}}$,

$$\frac{\partial u}{\partial H_{4_{in}}} = -e^{-H_{4_{in}}} = -u$$

Substituting in chain rule,

$$\frac{\partial H_4}{\partial H_{4_{in}}} = -\frac{1}{(1+u)^2} \cdot (-u) = \frac{u}{(1+u)^2}$$

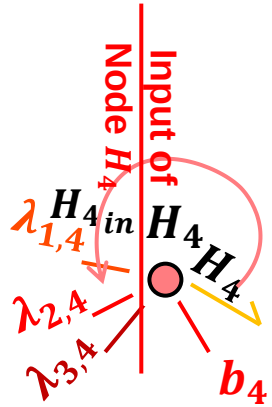
Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Through a Node?

Let's backpropagate through a node, and for clean logic, let's resolve it as a function of the output rather than the input,

$$H_4 = \sigma(H_{4_{in}}) = \frac{1}{1 + e^{-H_{4_{in}}}}$$

sigmoid activation



Take the derivative with respect to $H_{4_{in}}$,

chain rule

$$\frac{\partial H_4}{\partial H_{4_{in}}} = \frac{\partial}{\partial H_{4_{in}}}\left(\frac{1}{1 + e^{-H_{4_{in}}}}\right)$$

set $e^{-H_{4_{in}}} = u$

$$\frac{\partial H_4}{\partial H_{4_{in}}} = \frac{\partial}{\partial H_{4_{in}}}\left(\frac{1}{1+u}\right) = -\frac{u}{(1+u)^2}\frac{\partial u}{\partial H_{4_{in}}}$$

differentiate $u = e^{-O_{6_{in}}}$,

$$\frac{\partial u}{\partial H_{4_{in}}} = -e^{-H_{4_{in}}} = -u$$

Substituting in chain rule,

$$\frac{\partial H_4}{\partial H_{4_{in}}} = -\frac{1}{(1+u)^2}\cdot(-u) = \frac{u}{(1+u)^2}$$

Express in terms of $H_4 = 1/1 + u$,

$$u = \frac{1 - H_4}{H_4}$$

$$\frac{\partial H_4}{\partial H_{4_{in}}} = \frac{(1 - H_4)/H_4}{\left(1/H_4\right)^2} = \frac{1 - H_4}{H_4}\cdot H_4^2 = (1 - H_4)\cdot H_4$$

Back propagate through a node: $\frac{\partial H_4}{\partial H_{4_{in}}} = (1 - H_4)\cdot H_4$

From Chain Rule: $\frac{\partial L}{\partial H_{4_{in}}} = \frac{\partial H_4}{\partial H_{4_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_4}\cdot\frac{\partial O_6}{\partial O_{6_{in}}}\cdot\frac{\partial L}{\partial O_6} = (1 - H_4)\cdot H_4\cdot\lambda_{4,6}\cdot 1.0\cdot(O_6 - y)$

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Along a Connection to Output of Input Layer Node, I_1?

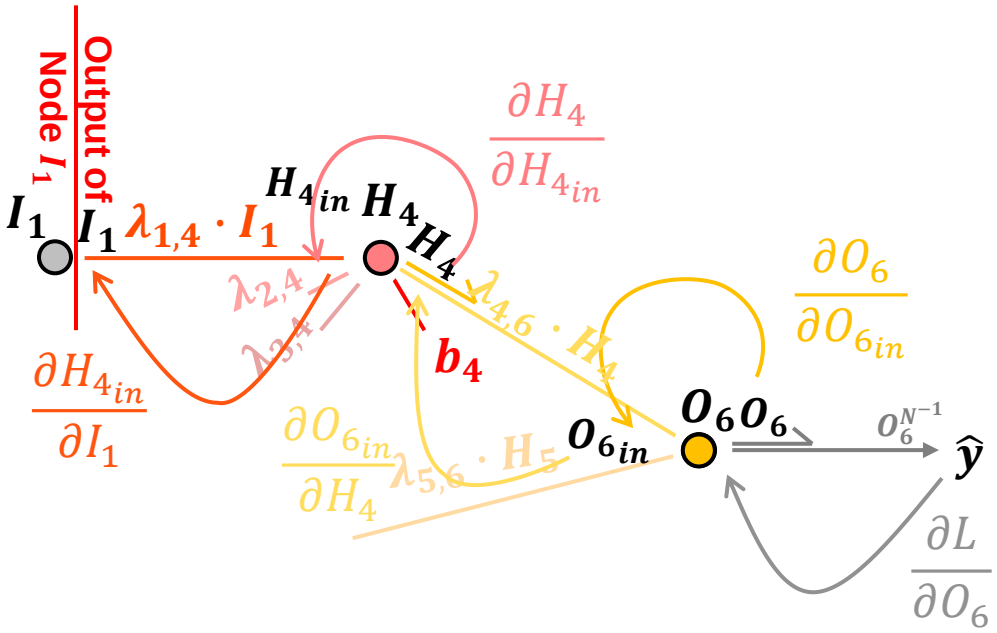
Pre-activation, input to node H_4 we have,

$$H_{4_{in}} = \lambda_{1,4} \cdot I_1 + \lambda_{2,4} \cdot I_2 + \lambda_{3,4} \cdot I_3 + b_4$$

We calculate the derivative along the connection as,

$$\frac{\partial H_{4_{in}}}{\partial I_1} = \frac{\partial}{\partial I_1}\left(\lambda_{1,4} \cdot I_1 + \lambda_{2,4} \cdot I_2 + \lambda_{3,4} \cdot I_3 + b_4\right) = \lambda_{1,4}$$

we backpropagate along a connection by applying the connection weight.



Back propagate along a connection: $\frac{\partial H_{4_{in}}}{\partial I_1} = \lambda_{1,4}$

From Chain Rule: $\frac{\partial L}{\partial I_1} = \frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial O_{6_{in}}}{\partial H_4} \cdot \frac{\partial O_6}{\partial O_{6_{in}}} \cdot \frac{\partial L}{\partial O_6} = \lambda_{1,4} \cdot [(1 - H_4) \cdot H_4] \cdot \lambda_{4,6} \cdot 1.0 \cdot (O_6 - y)$

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Along a Connection to Output of Input Layer Node, I_1?

Our $\frac{\partial L}{\partial I_1}$ did not account for the other path from nodes I_1 and O_6.

$$\frac{\partial L}{\partial I_1} = \frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial L}{\partial H_{4_{in}}} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial L}{\partial H_{5_{in}}}$$

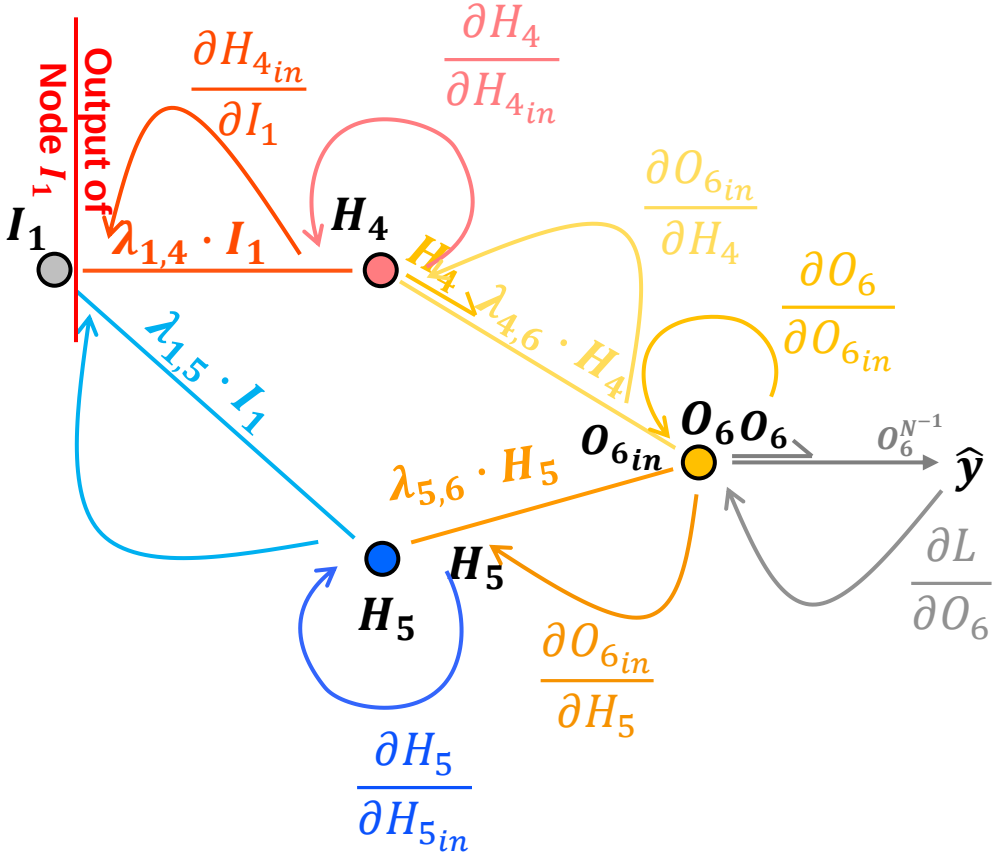
$$\frac{\partial L}{\partial I_1} = \frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial H_4}{\partial H_{4_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_4} \cdot \frac{\partial O_6}{\partial O_{6_{in}}} \cdot \frac{\partial L}{\partial O_6} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial H_5}{\partial H_{5_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_5} \cdot \frac{\partial O_6}{\partial O_{6_{in}}} \cdot \frac{\partial L}{\partial O_6}$$

1.0 1.0

factor

$$\frac{\partial L}{\partial I_1} = \left[\frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial H_4}{\partial H_{4_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_4} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial H_5}{\partial H_{5_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_5}\right] \cdot \frac{\partial L}{\partial O_6}$$

$$\frac{\partial L}{\partial I_1} = \left[\lambda_{1,4} \cdot [(1 - H_4) \cdot H_4] \cdot \lambda_{4,6} + \lambda_{1,5} \cdot [(1 - H_5) \cdot H_5] \cdot \lambda_{5,6}\right] \cdot (O_6 - y)$$



For backpropagation we sum all the paths.

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative Through the Input Node Node, I_1?

$$\frac{\partial I_1}{\partial I_{1_{in}}} = \frac{\partial (I_{1_{in}})}{\partial I_{1_{in}}} = 1.0$$

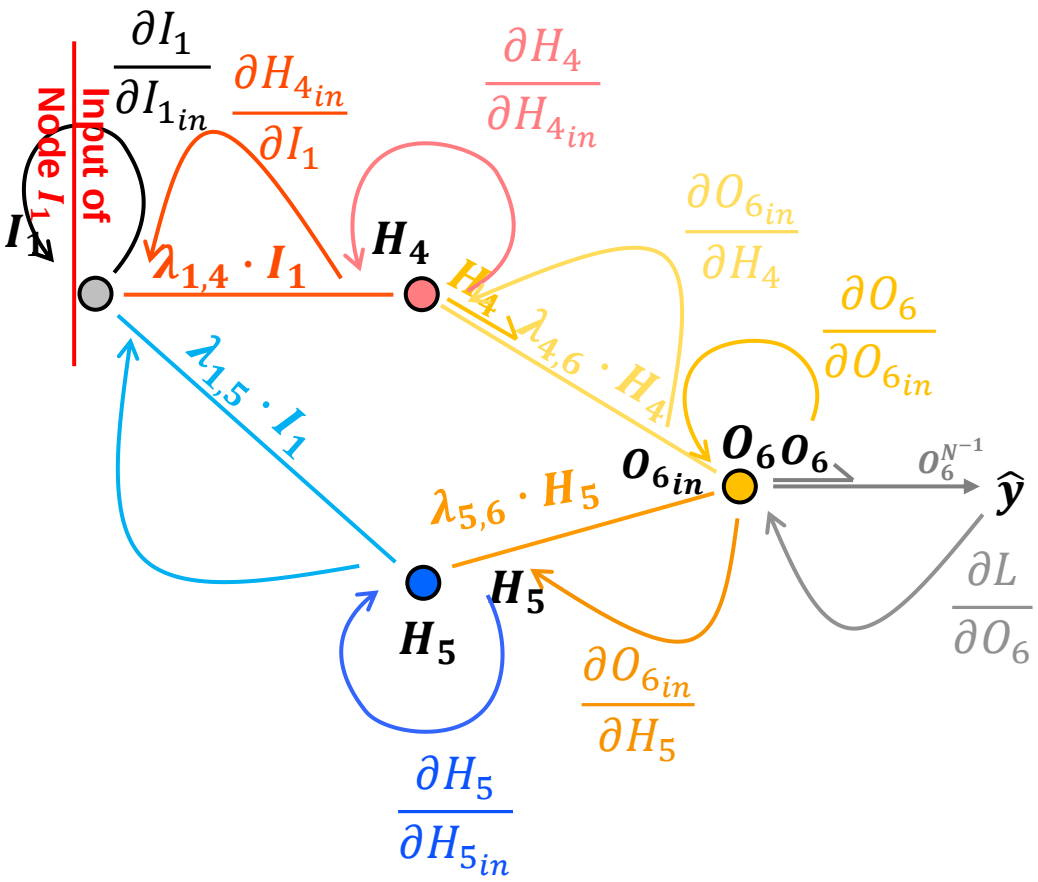
identity activation for input nodes

$$\frac{\partial L}{\partial I_{1_{in}}} = \frac{\partial I_1}{\partial I_{1_{in}}} \cdot \left[\frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial L}{\partial H_{4_{in}}} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial L}{\partial H_{5_{in}}} \right]$$

$$\frac{\partial L}{\partial I_{1_{in}}} = \cancel{\frac{\partial I_1}{\partial I_{1_{in}}}}^{1.0} \cdot \left[\frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial H_4}{\partial H_{4_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_4} \cdot \cancel{\frac{\partial O_6}{\partial O_{6_{in}}}}^{1.0} \cdot \frac{\partial L}{\partial O_6} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial H_5}{\partial H_{5_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_5} \cdot \cancel{\frac{\partial O_6}{\partial O_{6_{in}}}}^{1.0} \cdot \frac{\partial L}{\partial O_6} \right]$$

factor

$$\frac{\partial L}{\partial I_{1_{in}}} = \left[\frac{\partial H_{4_{in}}}{\partial I_1} \cdot \frac{\partial H_4}{\partial H_{4_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_4} + \frac{\partial H_{5_{in}}}{\partial I_1} \cdot \frac{\partial H_5}{\partial H_{5_{in}}} \cdot \frac{\partial O_{6_{in}}}{\partial H_5} \right] \cdot \frac{\partial L}{\partial O_6}$$



$$\frac{\partial L}{\partial I_{1_{in}}} = \left[\lambda_{1,4} \cdot [(1 - H_4) \cdot H_4] \cdot \lambda_{4,6} + \lambda_{1,5} \cdot [(1 - H_5) \cdot H_5] \cdot \lambda_{5,6} \right] \cdot (O_6 - y)$$

For backpropagation we sum all the paths.

Training Artificial Neural Back Propagation

How Do We Account for Multiple Paths?

We sum the loss derivatives over all the possible paths.

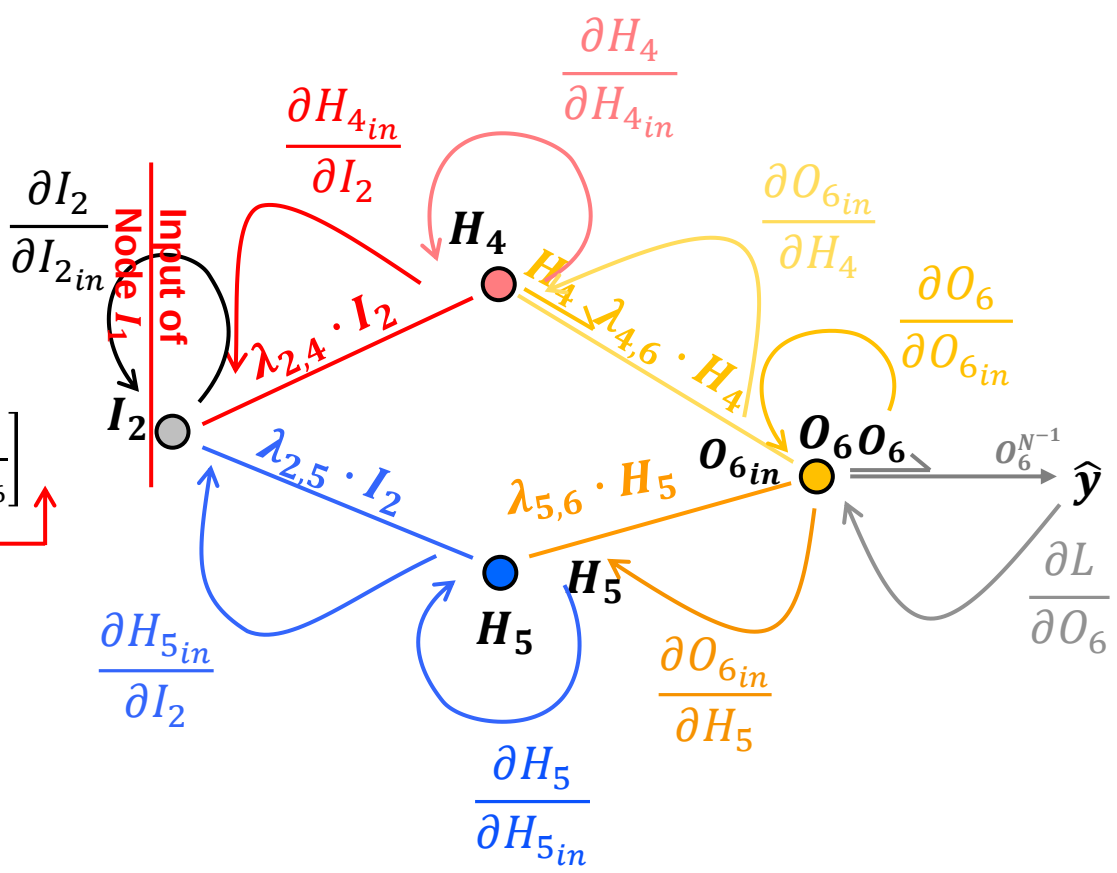
$$\frac{\partial L}{\partial I_{2_{in}}} = \frac{\partial I_2}{\partial I_{2_{in}}}\left[\frac{\partial H_{4_{in}}}{\partial I_2}\cdot\frac{\partial L}{\partial H_{4_{in}}} + \frac{\partial H_{5_{in}}}{\partial I_2}\cdot\frac{\partial L}{\partial H_{5_{in}}}\right]$$

$$\frac{\partial L}{\partial I_{2_{in}}} = \frac{\partial I_2}{\partial I_{2_{in}}}\left[\frac{\partial H_{4_{in}}}{\partial I_2}\cdot\frac{\partial H_4}{\partial H_{4_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_4}\cdot\frac{\partial O_6}{\partial O_{6_{in}}}\cdot\frac{\partial L}{\partial O_6} + \frac{\partial H_{5_{in}}}{\partial I_2}\cdot\frac{\partial H_5}{\partial H_{5_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_5}\cdot\frac{\partial O_6}{\partial O_{6_{in}}}\cdot\frac{\partial L}{\partial O_6}\right]$$

1.0 1.0 Identity activation 1.0

factor

$$\frac{\partial L}{\partial I_{2_{in}}} = \left[\frac{\partial H_{4_{in}}}{\partial I_2}\cdot\frac{\partial H_4}{\partial H_{4_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_4} + \frac{\partial H_{5_{in}}}{\partial I_2}\cdot\frac{\partial H_5}{\partial H_{5_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_5}\right]\cdot\frac{\partial L}{\partial O_6}$$



$$\frac{\partial L}{\partial I_{2_{in}}} = \left[\lambda_{2,4}\cdot[(1-H_4)\cdot H_4]\cdot\lambda_{4,6} + \lambda_{2,5}\cdot[(1-H_5)\cdot H_5]\cdot\lambda_{5,6}\right]\cdot(O_6 - y)$$

Training Artificial Neural Back Propagation

How Do We Account for Multiple Paths? Other examples.

We sum the loss derivatives over all the possible paths.

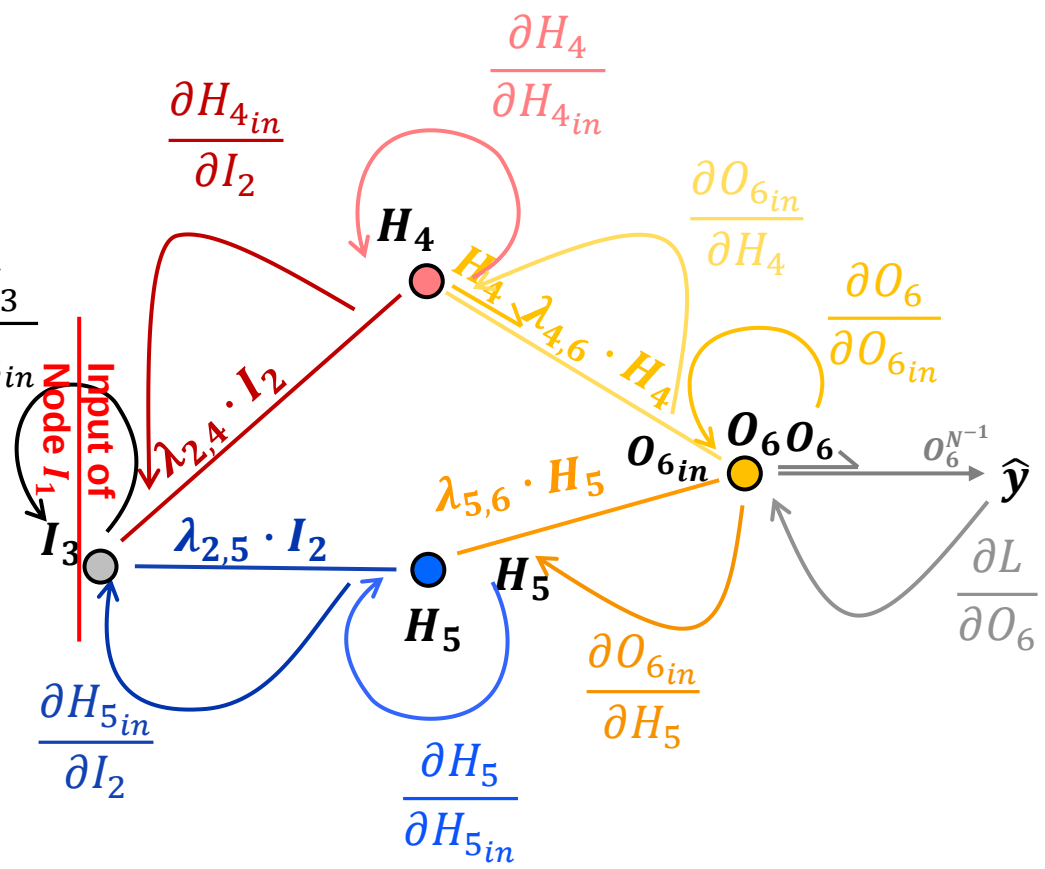
$$\frac{\partial L}{\partial I_{3_{in}}} = \frac{\partial I_3}{\partial I_{3_{in}}}\left[\frac{\partial H_{4_{in}}}{\partial I_3}\cdot\frac{\partial L}{\partial H_{4_{in}}} + \frac{\partial H_{5_{in}}}{\partial I_3}\cdot\frac{\partial L}{\partial H_{5_{in}}}\right]$$

1.0 1.0 Identity activation 1.0

$$\frac{\partial L}{\partial I_{3_{in}}} = \frac{\partial I_3}{\partial I_{3_{in}}}\left[\frac{\partial H_{4_{in}}}{\partial I_3}\cdot\frac{\partial H_4}{\partial H_{4_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_4}\cdot\frac{\partial O_6}{\partial O_{6_{in}}}\cdot\frac{\partial L}{\partial O_6} + \frac{\partial H_{5_{in}}}{\partial I_3}\cdot\frac{\partial H_5}{\partial H_{5_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_5}\cdot\frac{\partial O_6}{\partial O_{6_{in}}}\cdot\frac{\partial L}{\partial O_6}\right]$$

factor

$$\frac{\partial L}{\partial I_{3_{in}}} = \left[\frac{\partial H_{4_{in}}}{\partial I_3}\cdot\frac{\partial H_4}{\partial H_{4_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_4} + \frac{\partial H_{5_{in}}}{\partial I_3}\cdot\frac{\partial H_5}{\partial H_{5_{in}}}\cdot\frac{\partial O_{6_{in}}}{\partial H_5}\right]\cdot\frac{\partial L}{\partial O_6}$$



$$\frac{\partial L}{\partial I_{3_{in}}} = \left[\lambda_{3,4}\cdot[(1-H_4)\cdot H_4]\cdot\lambda_{4,6} + \lambda_{3,5}\cdot[(1-H_5)\cdot H_5]\cdot\lambda_{5,6}\right]\cdot(O_6 - y)$$

Training Artificial Neural Back Propagation

Now we have backpropagated the loss derivative - we have the loss derivative with respect to the input and output of each node in our network.

$$\frac{\partial L}{\partial I_{1_{in}}}, \frac{\partial L}{\partial I_1}, \frac{\partial L}{\partial I_{2_{in}}}, \frac{\partial L}{\partial I_2}, \frac{\partial L}{\partial I_{3_{in}}}, \frac{\partial L}{\partial I_3}, \frac{\partial L}{\partial H_{4_{in}}}, \frac{\partial L}{\partial H_4},$$

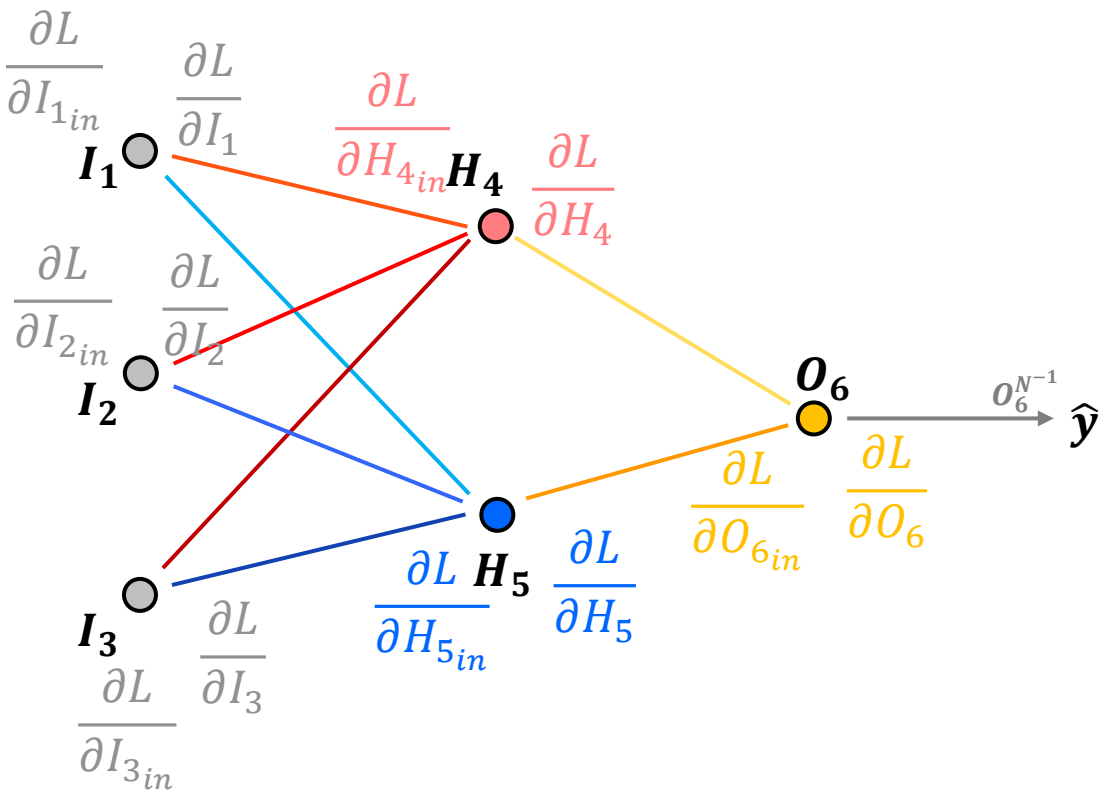
$$\frac{\partial L}{\partial H_5}, \frac{\partial L}{\partial H_5}, \frac{\partial L}{\partial O_{6_{in}}}, \frac{\partial L}{\partial O_6}$$

But what we actually need is the **loss derivative with respect to each connection weight and node bias**.

Weights: $\frac{\partial L}{\partial \lambda_{1,4}}, \frac{\partial L}{\partial \lambda_{2,4}}, \frac{\partial L}{\partial \lambda_{3,4}}, \frac{\partial L}{\partial \lambda_{1,5}}, \frac{\partial L}{\partial \lambda_{2,5}}, \frac{\partial L}{\partial \lambda_{3,5}},$

$$\frac{\partial L}{\partial \lambda_{4,6}}, \frac{\partial L}{\partial \lambda_{5,6}}$$

Biases: $\frac{\partial L}{\partial b_4}, \frac{\partial L}{\partial b_5}, \frac{\partial L}{\partial b_6}$



Training Artificial Neural Backpropagation

How Do We Backpropagate the Loss Derivative to a Connection Weight?

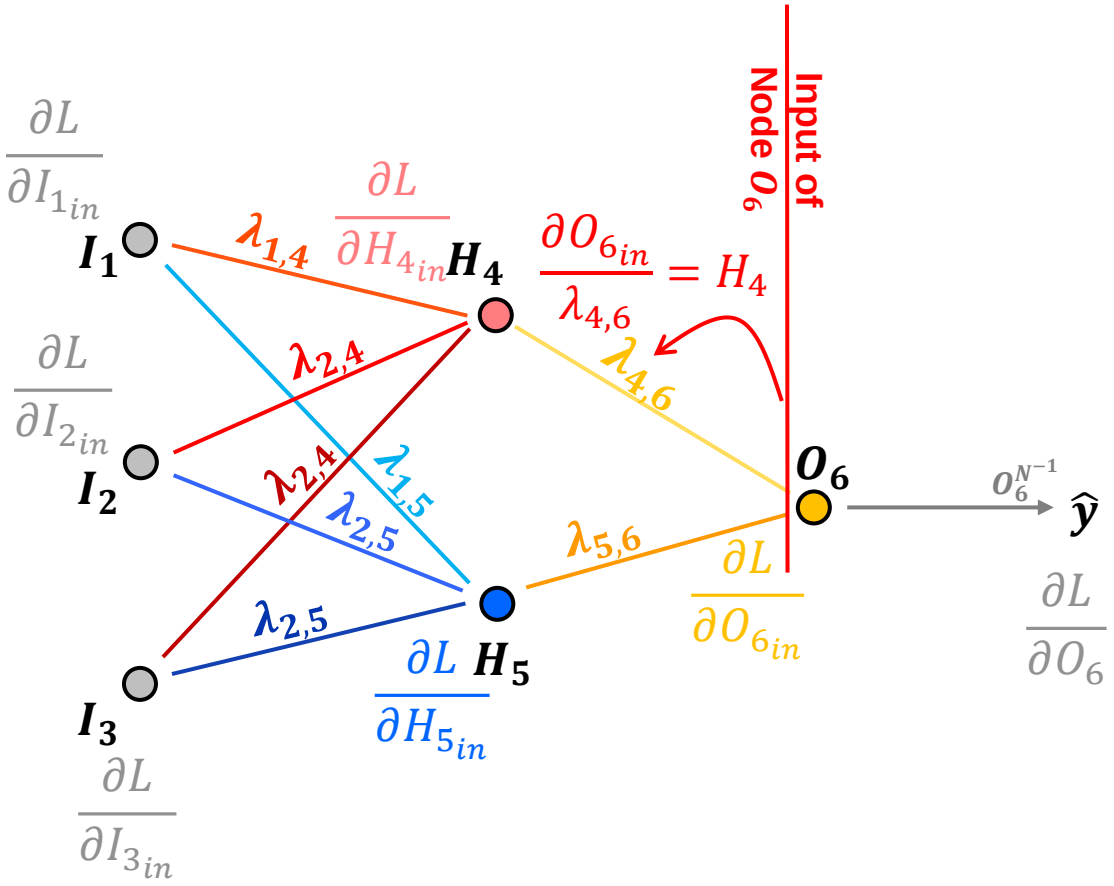
Pre-activation, input to node O_6 we have,

$$O_{6_{in}} = \lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + b_6$$

We calculate the derivative with respect to the connection weight as,

$$\frac{\partial O_{6_{in}}}{\partial \lambda_{4,6}} = \frac{\partial \left(\lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + b_6 \right)}{\partial \lambda_{4,6}} = H_4$$

we need the output of the node in the previous layer passed along the connection.



Recall:

Back propagate to a connection weight: $\frac{\partial O_{6_{in}}}{\lambda_{4,6}} = H_4$ $\frac{\partial L}{\partial O_{6_{in}}} = 1.0 \cdot (O_6 - y)$

From Chain Rule: $\frac{\partial L}{\lambda_{4,6}} = \frac{\partial O_{6_{in}}}{\lambda_{4,6}} \cdot \frac{\partial L}{\partial O_{6_{in}}} = H_4 \cdot 1.0 \cdot (O_6 - y)$

Training Artificial Neural Backpropagation

How Do We Backpropagate the Loss Derivative to a Connection Weight?

To calculate all the error derivatives for all connection weights, $\frac{\partial L}{\partial I_{1_{in}}}$

$$\frac{\partial L}{\lambda_{1,4}} = \frac{\partial H_{4_{in}}}{\lambda_{1,4}} \cdot \frac{\partial L}{\partial H_{4_{in}}} = I_1 \cdot \frac{\partial L}{\partial H_{4_{in}}}$$

$$\frac{\partial L}{\lambda_{2,4}} = \frac{\partial H_{4_{in}}}{\lambda_{2,4}} \cdot \frac{\partial L}{\partial H_{4_{in}}} = I_2 \cdot \frac{\partial L}{\partial H_{4_{in}}}$$

$$\frac{\partial L}{\lambda_{3,4}} = \frac{\partial H_{4_{in}}}{\lambda_{3,4}} \cdot \frac{\partial L}{\partial H_{4_{in}}} = I_3 \cdot \frac{\partial L}{\partial H_{4_{in}}}$$

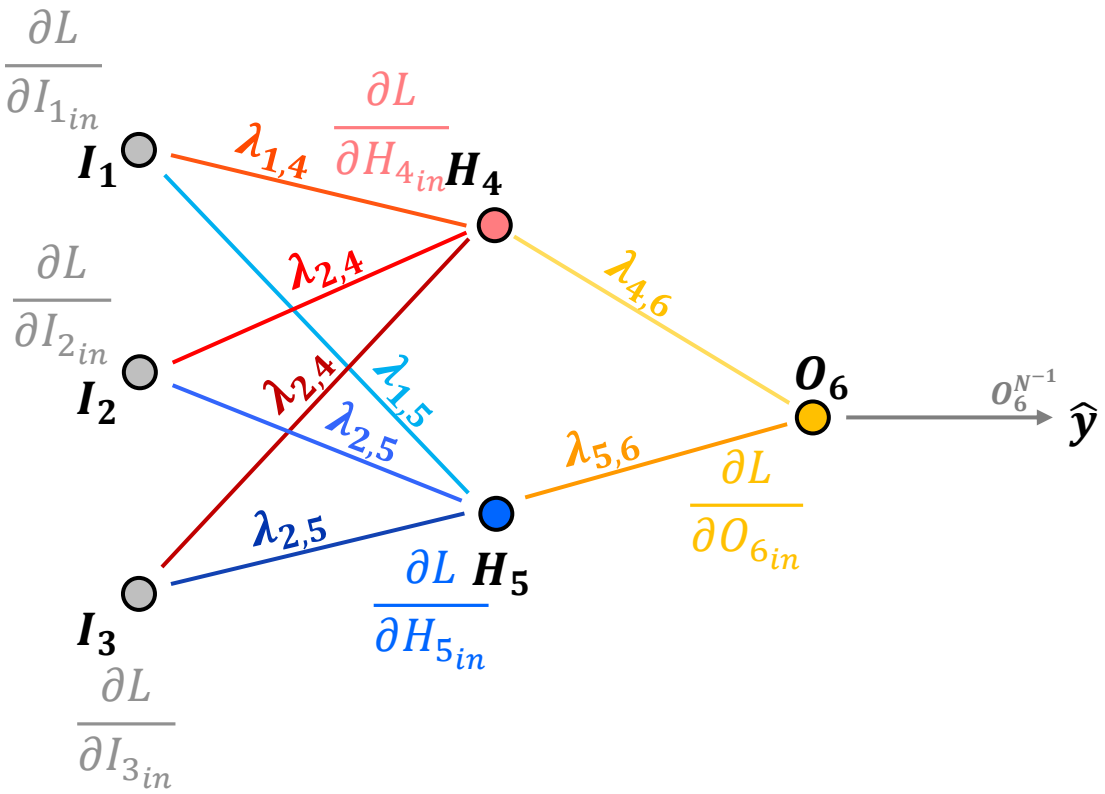
$$\frac{\partial L}{\lambda_{1,5}} = \frac{\partial H_{5_{in}}}{\lambda_{1,5}} \cdot \frac{\partial L}{\partial H_{5_{in}}} = I_1 \cdot \frac{\partial L}{\partial H_{5_{in}}}$$

$$\frac{\partial L}{\lambda_{2,5}} = \frac{\partial H_{5_{in}}}{\lambda_{2,5}} \cdot \frac{\partial L}{\partial H_{5_{in}}} = I_2 \cdot \frac{\partial L}{\partial H_{5_{in}}}$$

$$\frac{\partial L}{\lambda_{3,5}} = \frac{\partial H_{5_{in}}}{\lambda_{3,5}} \cdot \frac{\partial L}{\partial H_{5_{in}}} = I_3 \cdot \frac{\partial L}{\partial H_{5_{in}}}$$

$$\frac{\partial L}{\lambda_{4,6}} = \frac{\partial O_{6_{in}}}{\lambda_{4,6}} \cdot \frac{\partial L}{\partial O_{6_{in}}} = H_4 \cdot \frac{\partial L}{\partial O_{6_{in}}}$$

$$\frac{\partial L}{\lambda_{5,6}} = \frac{\partial O_{6_{in}}}{\lambda_{5,6}} \cdot \frac{\partial L}{\partial O_{6_{in}}} = H_5 \cdot \frac{\partial L}{\partial O_{6_{in}}}$$



Loss derivative for a connection weights is:

Connection Signal x Loss Derivative of Next Node Input

Training Artificial Neural Back Propagation

How Do We Backpropagate the Loss Derivative to a Node Bias?

Pre-activation, input to node O_6 we have,

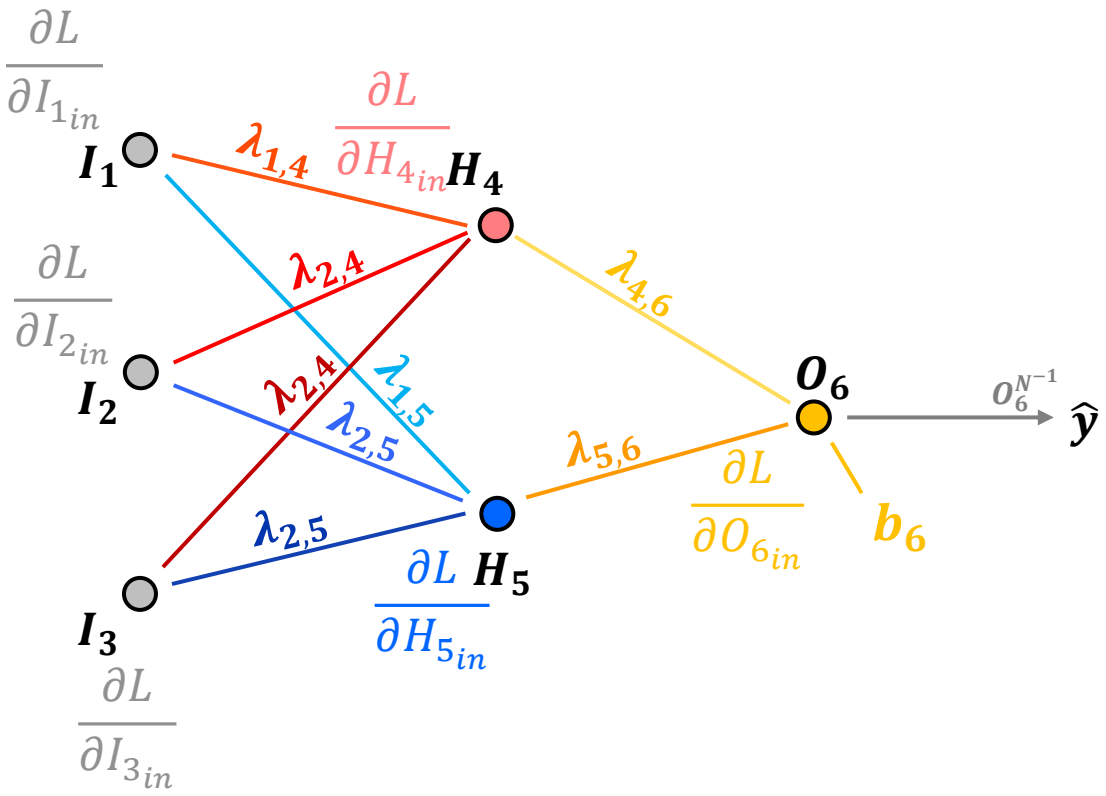
$$O_{6_{in}} = \lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + b_6$$

We calculate the derivative of a connection weight as,

$$\frac{\partial O_{6_{in}}}{\partial b_6} = \frac{\partial\left(\lambda_{4,6} \cdot H_4 + \lambda_{5,6} \cdot H_5 + b_6\right)}{\partial b_6} = 1.0$$

so our bias loss derivative is equal to the node input loss derivative,

$$\frac{\partial L}{\partial b_6} = \frac{\partial O_{6_{in}}}{\partial b_6} \cdot \frac{\partial L}{\partial O_{6_{in}}} = 1.0 \cdot \frac{\partial L}{\partial O_{6_{in}}}$$



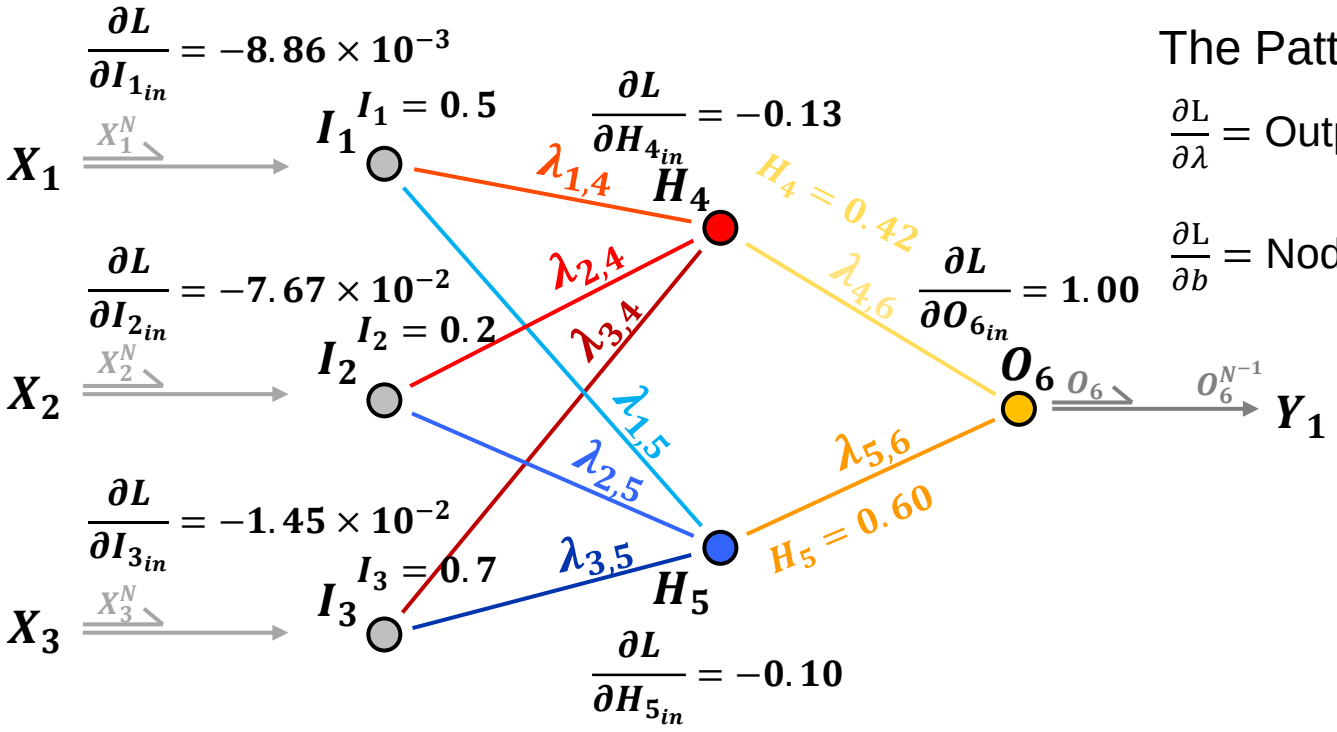
Loss derivative for a node bias is:

Loss Derivative of the Node Input

Training Artificial Neural Back Propagation

Example of Backpropagation with our Artificial Neural Network

We have solved for the loss derivatives with respect to the input for all our networks nodes.



The Pattern for Weights and Biases Derivatives

$\frac{\partial L}{\partial \lambda}$ = Output From Previous Node x Next Node Input Derivative

$\frac{\partial L}{\partial b}$ = Node Input Derivative

Example Weights and Biases Derivatives:

$$\frac{\partial L}{\partial \lambda_{4,6}} = H_4 \cdot \frac{\partial L}{\partial O_{6_{in}}} = 0.42 \cdot 1.00 = 0.42$$

$$\frac{\partial L}{\partial \lambda_{1,4}} = I_1 \cdot \frac{\partial L}{\partial H_{4_{in}}} = 0.5 \cdot (-0.13) = 0.065$$

$$\frac{\partial L}{\partial b_4} = \frac{\partial L}{\partial H_{4_{in}}} = -0.32$$

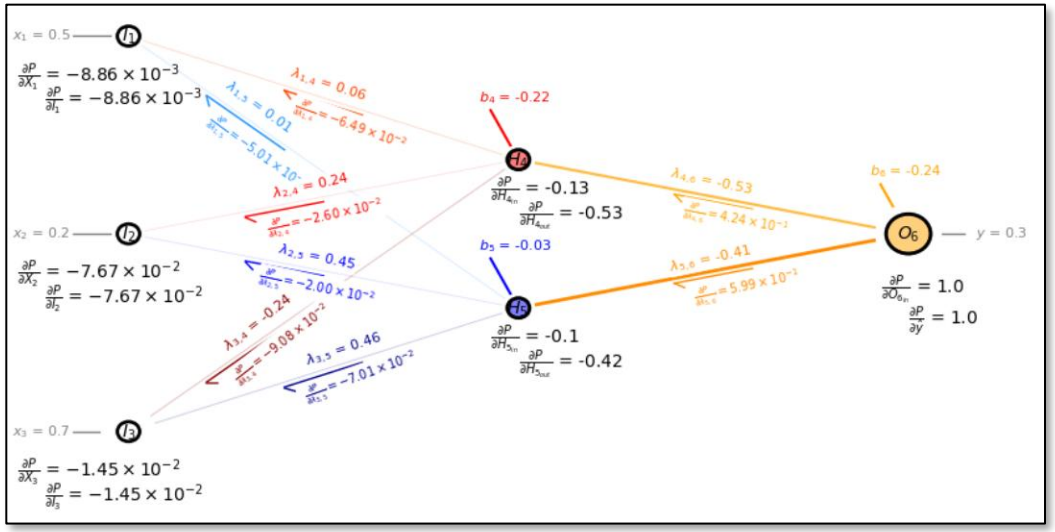
Now we can calculate the gradients at the weights and biases.

For weights, $\frac{\partial L}{\partial \lambda_{i,j}} = H_i \cdot \frac{\partial L}{\partial H_{j_{in}}}$ For biases, $\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial H_{j_{in}}}$

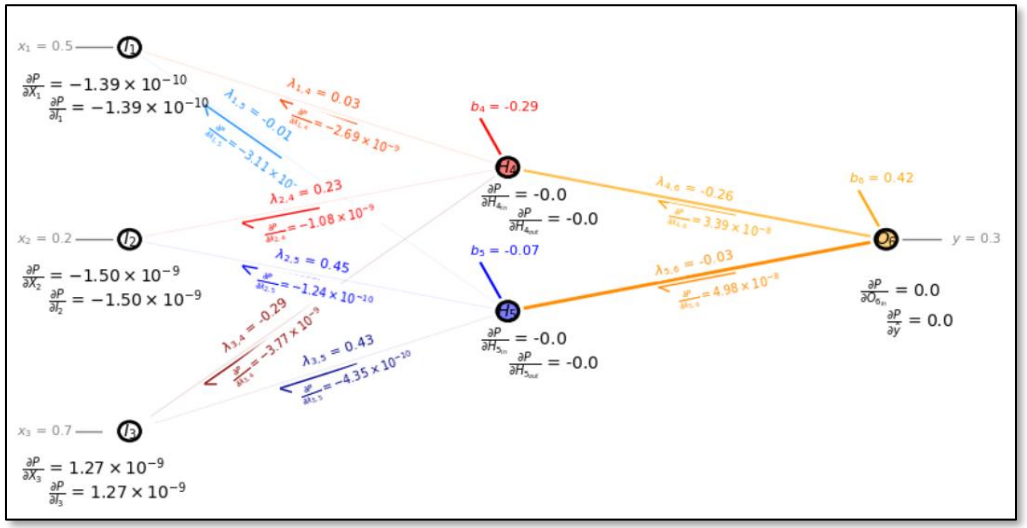
Training Artificial Neural Back Propagation

Example Forward Passes of Our Artificial Neural Network

Initial Model



Trained Model



Initial random weights and biases (left) and trained weights and biases (right) for an ANN, from Interactive_ANN_Simple.ipynb, random seed = 3.

Example untrained initial model:

$$\frac{\partial L}{\partial \lambda_{4,6}} = H_4 \cdot \frac{\partial L}{\partial O_{6_{in}}} = 0.42 \cdot 1.00 = 0.42 \qquad \frac{\partial L}{\partial \lambda_{1,4}} = I_1 \cdot \frac{\partial L}{\partial H_{4_{in}}} = 0.5 \cdot (-0.13) = 0.065 \qquad \frac{\partial L}{\partial b_4} = \frac{\partial L}{\partial H_{4_{in}}} = -0.32$$

Training Artificial Updating Model Parameters

We now have a gradient, the change to the individual weights is scaled by a learning rate, η.

$$\frac{\partial L}{\partial \lambda_{4,6}} = H_4 \cdot \frac{\partial L}{\partial O_{6_{in}}} = 0.42 \cdot 1.00 = 0.42 \qquad \frac{\partial L}{\partial \lambda_{1,4}} = I_1 \cdot \frac{\partial L}{\partial H_{4_{in}}} = 0.5 \cdot (-0.13) = 0.065 \qquad \frac{\partial L}{\partial b_4} = \frac{\partial L}{\partial H_{4_{in}}} = -0.32$$

- This process of gradient calculation and parameter/weights updating is iterated.
- **Stochastic Gradient Descent Optimization** - stochasticity is introduced with neural networks through the use of subsets of the data, batches, and dropout regularization (see Lecture 10b, Lasso Regression).

Parameter Updating Examples, assuming $\eta = 0.1$ and random number seed, $S = 3$, updating to epoch $\ell = 1$:

$$\lambda_{4,6}^{\ell} = \lambda_{4,6}^{\ell-1} + \eta \cdot \frac{\partial L}{\partial \lambda_{4,6}} \qquad \lambda_{1,4}^{\ell} = \lambda_{1,4}^{\ell-1} + \eta \cdot \frac{\partial L}{\partial \lambda_{1,4}} \qquad b_4^{\ell} = b_4^{\ell-1} + \eta \cdot \frac{\partial L}{\partial b_4}$$

$$\lambda_{4,6}^{\ell} = -0.53 + (0.1) \cdot 0.42 = -0.49 \qquad \lambda_{1,4}^{\ell} = 0.06 + (0.1) \cdot (-0.065) = 0.05 \qquad b_4^{\ell} = -0.22 + (0.1) \cdot (-0.13) = -0.24$$

Training Artificial Updating Model Parameters

Gradient Descent Optimization - We Need to Avoid Local Minimums by Setting these Hyperparameters.

Learning Rate/Step Size ($0.0 - 1.0, 0.01$ is the default in Tensorflow, Keras Module):

The η hyperparameter that controls the rate of weight and bias updating, in response to the loss derivative with respect to connection weight, e.g., $\frac{\partial L}{\partial \lambda_{1,4}}$, or network bias, $\frac{\partial L}{\partial b_4}$.

$$\lambda_{1,4}^{\ell} = \lambda_{1,4}^{\ell-1} + \eta \cdot \frac{\partial L}{\partial \lambda_{1,4}} \qquad b_4^{\ell} = b_4^{\ell-1} + \eta \cdot \frac{\partial L}{\partial b_4}$$ where ℓ is the epoch

- Low Learning Rate – more stable, but a slower solution, may get stuck

- High Learning Rate – may be unstable, but perhaps a faster solution, may diverge out of the global minimum

Learning Rate Decay (>0) to avoid oscillation $\eta^{\ell} = \eta^{\ell-1} \cdot \left(\frac{1}{1+decay \cdot \ell}\right)$, where ℓ is the epoch

Training Artificial Updating Model Parameters

Parameter Updating Examples, assuming $\eta = 0.1$ and random number seed, $S = 3$, updating to epoch $\ell = 1$:

$$\frac{\partial L}{\partial \lambda_{4,6}} = H_4 \cdot \frac{\partial L}{\partial O_{6_{in}}} = 0.42 \cdot 1.00 = 0.42 \qquad \frac{\partial L}{\partial \lambda_{1,4}} = I_1 \cdot \frac{\partial L}{\partial H_{4_{in}}} = 0.5 \cdot (-0.13) = 0.065 \qquad \frac{\partial L}{\partial b_4} = \frac{\partial L}{\partial H_{4_{in}}} = -0.32$$

$$\lambda_{4,6}^{\ell} = \lambda_{4,6}^{\ell-1} + \eta \cdot \frac{\partial L}{\partial \lambda_{4,6}} \qquad \lambda_{1,4}^{\ell} = \lambda_{1,4}^{\ell-1} + \eta \cdot \frac{\partial L}{\partial \lambda_{1,4}} \qquad b_4^{\ell} = b_4^{\ell-1} + \eta \cdot \frac{\partial L}{\partial b_4}$$

$$\lambda_{4,6}^{\ell} = -0.53 + (0.1) \cdot 0.42 = -0.49 \qquad \lambda_{1,4}^{\ell} = 0.06 + (0.1) \cdot (-0.065) = 0.05 \qquad b_4^{\ell} = -0.22 + (0.1) \cdot (-0.13) = -0.24$$

Use my Interactive_ANN.ipynb dashboard to reproduce this result.

Notice that the model parameter updates are for a single training data case?

H_4, I_1, above and P are for a specific sample, $x_1, \ldots, x_m$ and y!

- we calculate the update over all samples in the batch and apply the average of the updates.

Training Artificial Updating Model Parameters

The Training is Iterative, Mini-Batches Introduce Stochasticity

Epochs: one pass through all of the training data – forward calculation of error, loss function, loss derivative, backpropagation to adjust the weights and biases. We usually use multiple mini-batches.

Mini-Batch: the number of training data considered in each forward pass – back-propagation iteration. We denote as t in the following slides.

e.g., 1000 training data, mini-batch of 100, requires 10 iterations to complete an Epoch

Comments:

- Larger mini-batches result in better estimates of the error to loss derivatives, but slower epochs (more computational time)
- Smaller batches result in nosier estimates of the error to loss derivatives, but faster epochs often results in faster learning and even possibly more robust models
- Smaller batches often result in models that are better able to generalize, ability to new cases

Training Artificial Updating Model Parameters

Gradient Descent Optimization - We Need to Avoid Local Minimums by Setting these Hyperparameters.

Momentum:

The integration of memory from the previous update.

$$\lambda_{1,4}^{\ell} = \eta \cdot \left[\boldsymbol{\alpha} \cdot \left(\frac{\partial L}{\partial \lambda_{1,4}} \right)^{\ell-1} + \cdot (1 - \boldsymbol{\alpha}) \cdot \left(\frac{\partial L}{\partial \lambda_{1,4}} \right)^{\ell} \right] \qquad \text{where } \ell \text{ is the epoch}$$

where $\left(\frac{\partial L}{\partial \lambda_{1,4}} \right)^{\ell-1}$ is the previous update vector, α, is the momentum parameter, and $\left(\frac{\partial L}{\partial \lambda_{1,4}} \right)^{\ell}$ is the new update vector.

- The gradients calculated for the partial derivatives of the loss function for each weight have noise. Momentum smooths out, reduces this noise.

- Momentum helps the solution proceed down the general slope of the loss function, rather than oscillating in local ravines or dimples.

Artificial Neural Networks Design

Hyperparameters:

Learning Rate/Step Size, r , rate of weight adjustment in back propagation

Momentum, α, memory in weight adjustment in back propagation

Performance metric, P – measure of goodness of the predictions

Width of the ANN – the number of nodes in each layer

Depth of the ANN – the number of hidden layers

Activation Functions – hidden layers

Output Function – output layer

Artificial Neural Networks Design

Limitations of Neural Nets

No Free Lunch Theorem – cannot guarantee the model is always optimum for predicting new, unobserved cases.

Parameter Rich – requires a large number of training data.

Low Interpretability – generally difficult to interrogate the model, not compact.

High Complexity Model – resulting in high model variance and lower model bias.

Artificial Neural Networks Design

There are a variety of designs based on the single hidden layer, feed-forward design that we reviewed.

Deep Learning – use of multiple hidden layers, we will try this

Convolutional Neural Networks – accounts for 2D / 3D information, next lecture

Recurrent Neural Networks – information can flow backwards

Auto Encoder – dimensionality reduction

Adam (Adaptive Moment Estimation) Optimization

A popular variant of stochastic gradient descent used for training artificial neural networks. Combining these methods:

1. **Stochastic gradient descent** with momentum (moving average of the gradient instead of instantaneous gradient)
2. **Adagrad** - Adaptive learning rates for each neural network weight
3. **RMSProp** - normalization balances step size, avoid exploding and vanishing

Given the gradient $\frac{\partial \mathrm{L}}{\partial \varphi}$ is a random variable (due to estimation with a limited batch); therefore, we can estimate the 1st and 2nd uncentered moments ($n = 1$ and 2 respectively).

$$m^n = E\left\{\frac{\partial L}{\partial \varphi}^n\right\}$$

- Calculate the moving window 1st and 2nd moment of the gradient over time steps with exponential decay, powers B_1 and B_2 respectively.
- The learning rate is scaled for each model parameter as the inverse of the second moment, a moving window variance in the gradient

Adam (Adaptive Moment Estimation) Optimization

Calculate the moving window 1st and 2nd moment of the gradient, $\frac{\partial \mathrm{L}}{\partial \varphi}$, over time steps with exponential decay, powers B_1 and B_2 respectively.

Local average gradient for current batch, t.

$$m^1_{t,\varphi_\alpha} = \beta_1 m^1_{t-1,\varphi_\alpha} + (1-\beta_1)\left(\frac{\partial L}{\partial \varphi_\alpha}\right)_t$$

1st moment of the gradient, $\frac{\partial \mathrm{L}}{\partial \varphi}$, for current batch, t.

Local gradient dispersion for current batch, t.

$$m^2_{t,\varphi_\alpha} = \beta_2 m^2_{t-1,\varphi_\alpha} + (1-\beta_2)\left(\frac{\partial L}{\partial \varphi_\alpha}\right)^2_t$$

2nd moment of the gradient, $\frac{\partial \mathrm{L}}{\partial \varphi}$, for current batch, t.

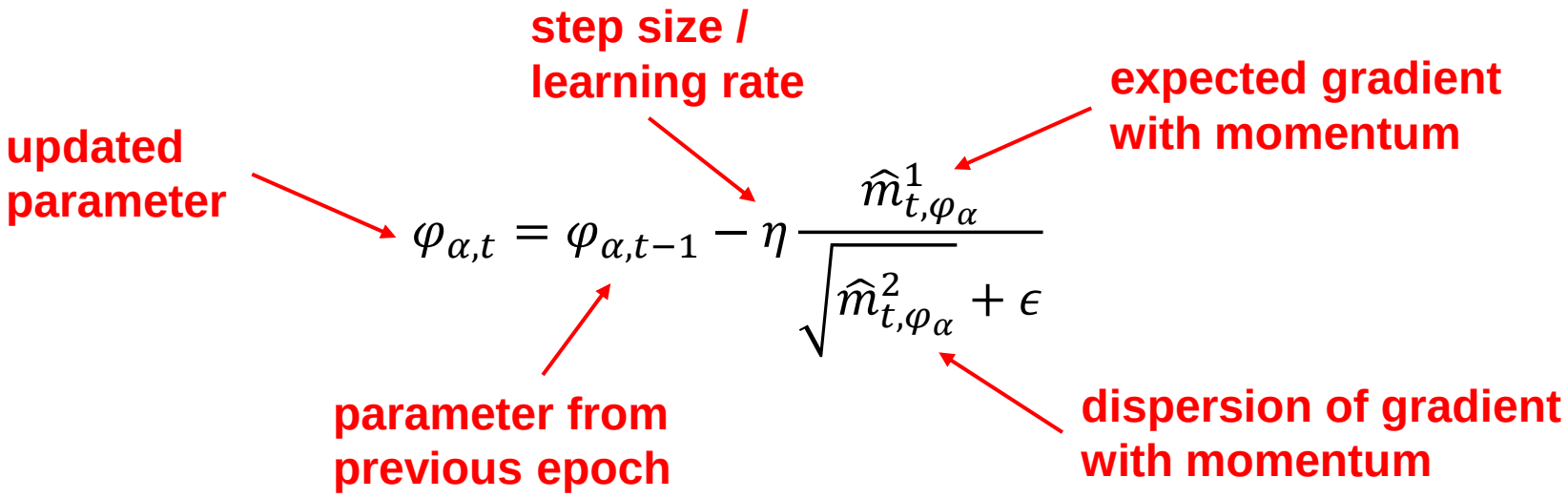
Where $t = 1, \ldots, n_{batches}$, and φ_α are neural network parameters, weight or bias term, α. Then we debias the estimators of the first and second moments.

$$\hat{m}^1_{t,\varphi_\alpha} = \frac{m^1_{t,\varphi_\alpha}}{1-\beta_1^t} \text{ and } \hat{m}^2_{t,\varphi_\alpha} = \frac{m^2_{t,\varphi_\alpha}}{1-\beta_2^t}$$

Adam (Adaptive Moment Estimation) Optimization

Now we apply the moving averages. The moving average of the 1st moment is our local estimate of gradient with moment, m^1_{t,φ_α}.

We reduce the rate of learning by the inverse of the square root of the non-centered second moment of the gradient (the non-centered measure of dispersion).



where r is the learning rate, and ϵ is a constant to avoid divide by zero.

PGE 383 Subsurface Machine Learning

Lecture 17: Neural Networks

Lecture outline:

- **Neural Networks Example**

Artificial Neural Networks Example

We will use a well and seismic data to build a neural net-based predictive model.

Train with 80% of well data (144 wells)

Test with 20% of well data.

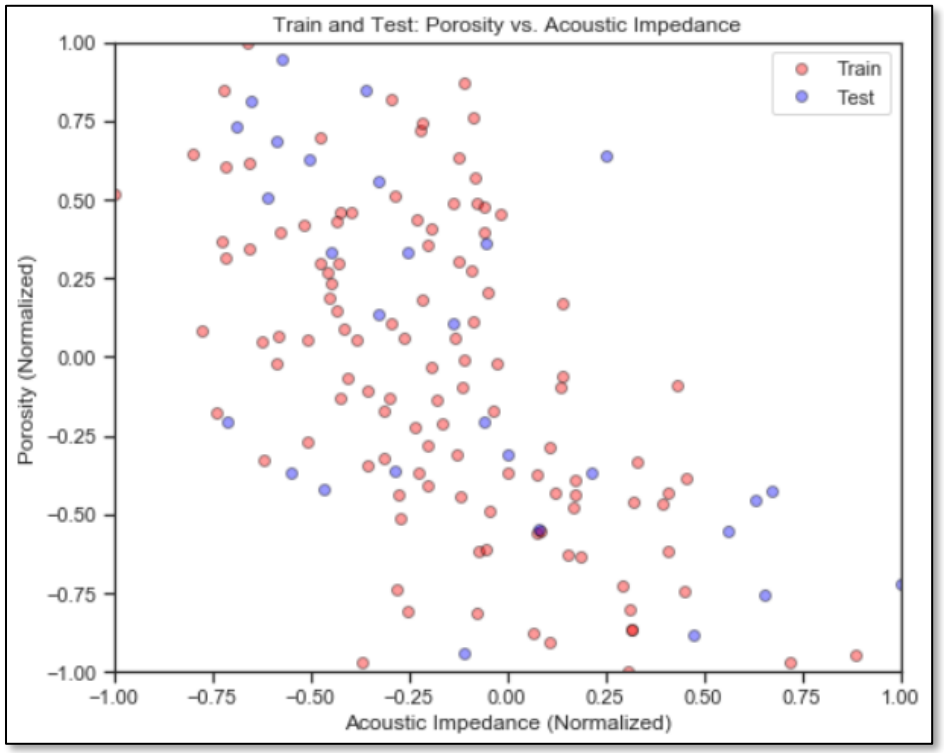
Application: Apply the model to predict porosity from a acoustic impedance map!

between well estimation of porosity in a single unit

Artificial Neural Networks Example

Normalize the variables to range from -1 to 1.

- the relationship between acoustic impedance and porosity.

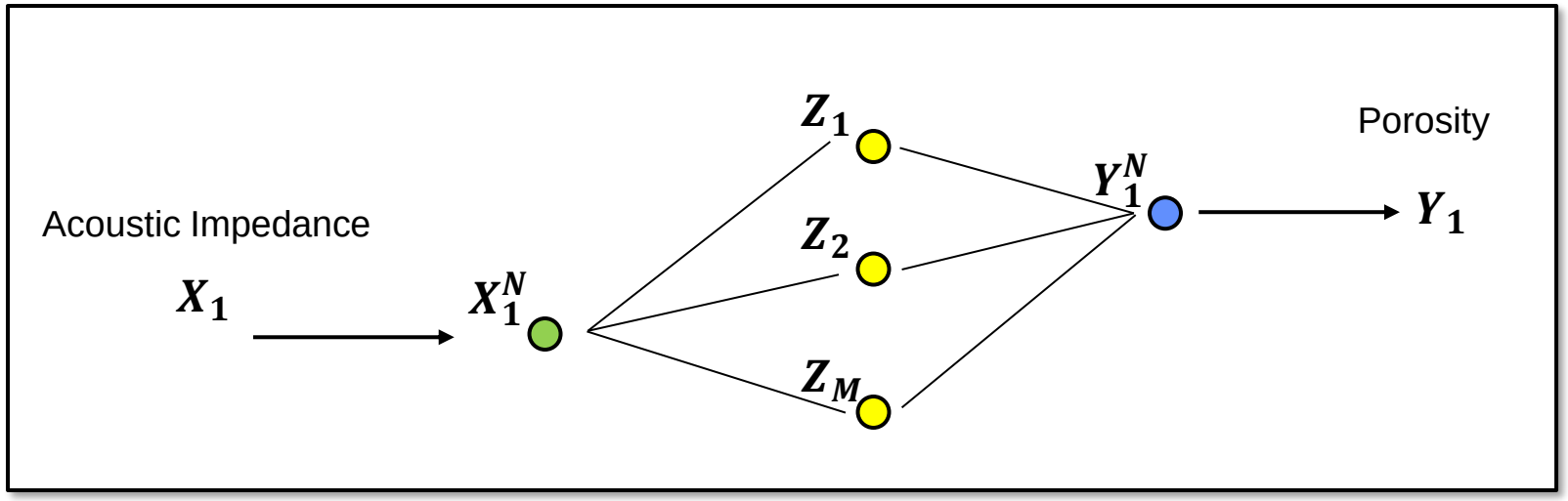


Training and testing data from SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.

Artificial Neural Networks Example

Design a Simple Neural Network

- 1 hidden layer, 3 nodes in hidden layer
- 1 node in input and output layer
- ReLu activation functions on hidden layer nodes

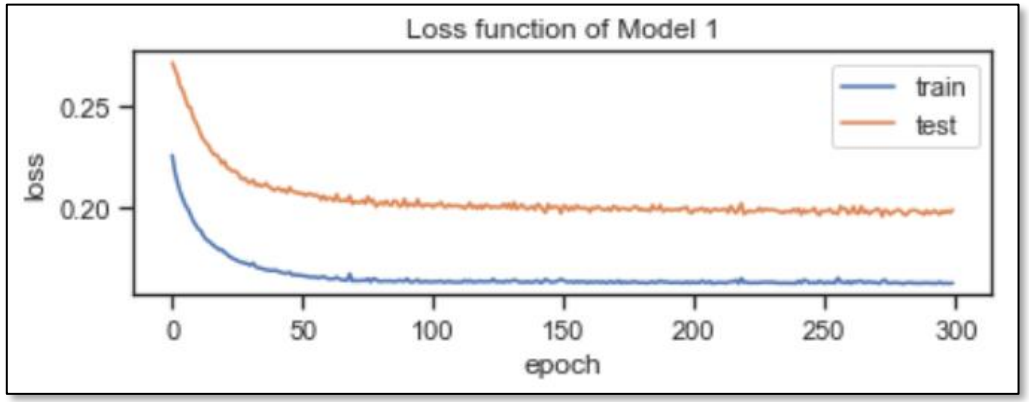


Artificial neural network design from SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.

Artificial Neural Networks Example

Training a Simple Neural Network

- 300 epochs with batches of 5
- train and test loss

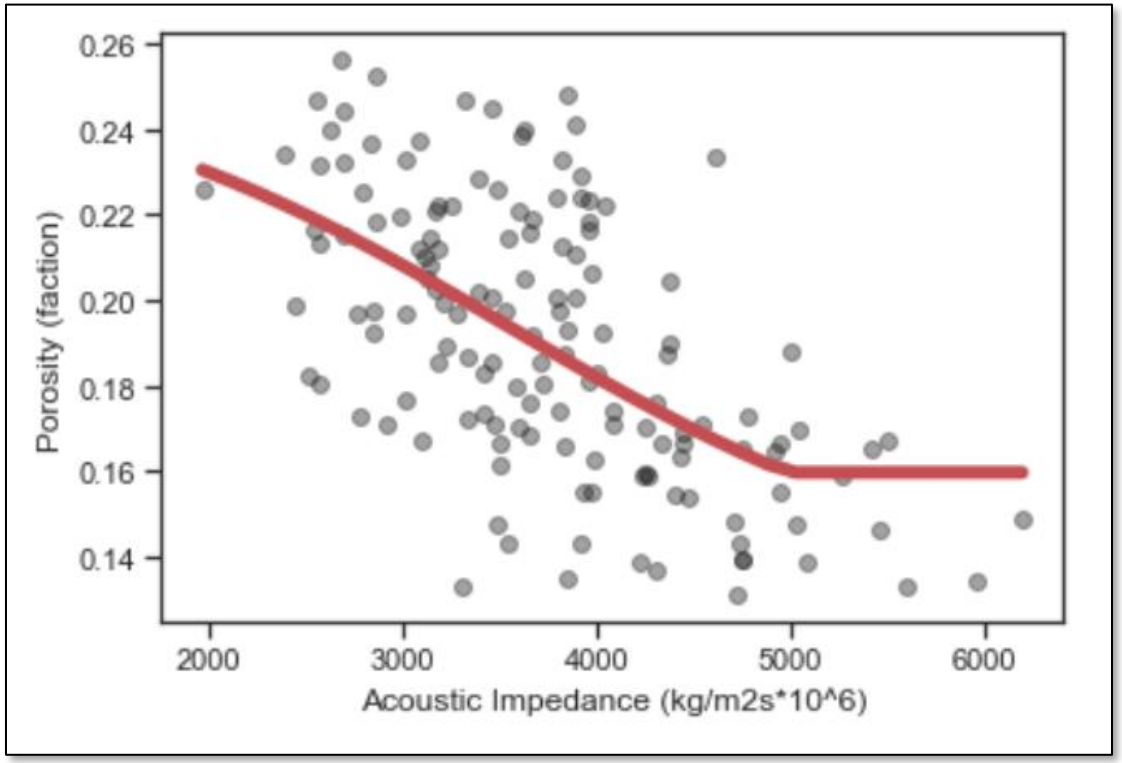


Artificial neural network training and testing error over model training epochs from SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.

Artificial Neural Networks Example

Model Predictions

- the model predictions with all training and testing data
- demonstrates the ability to fit non-linear data

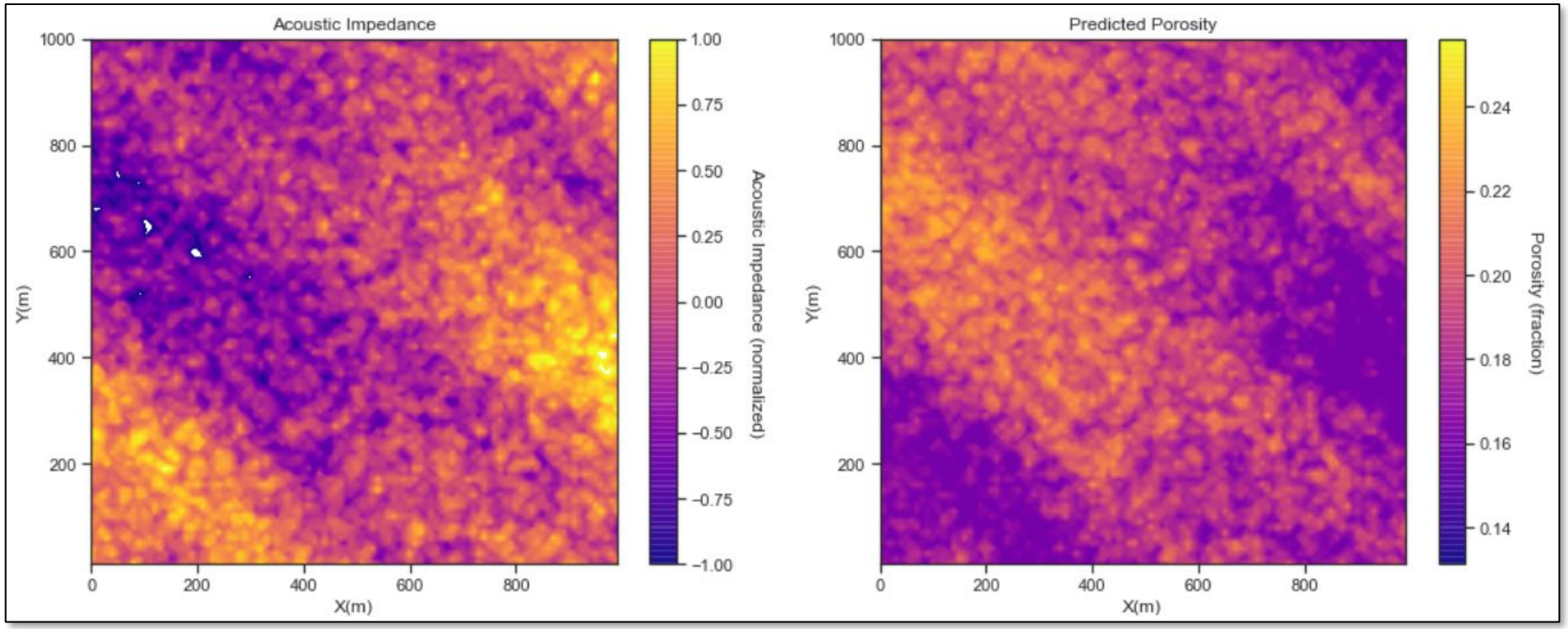


Artificial neural network prediction from SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.

Artificial Neural Networks Example

Model Predictions

- prediction of porosity from acoustic impedance between well locations.



Artificial neural network-based porosity prediction from acoustic impedance from SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.

PGE 383 Subsurface Machine Learning

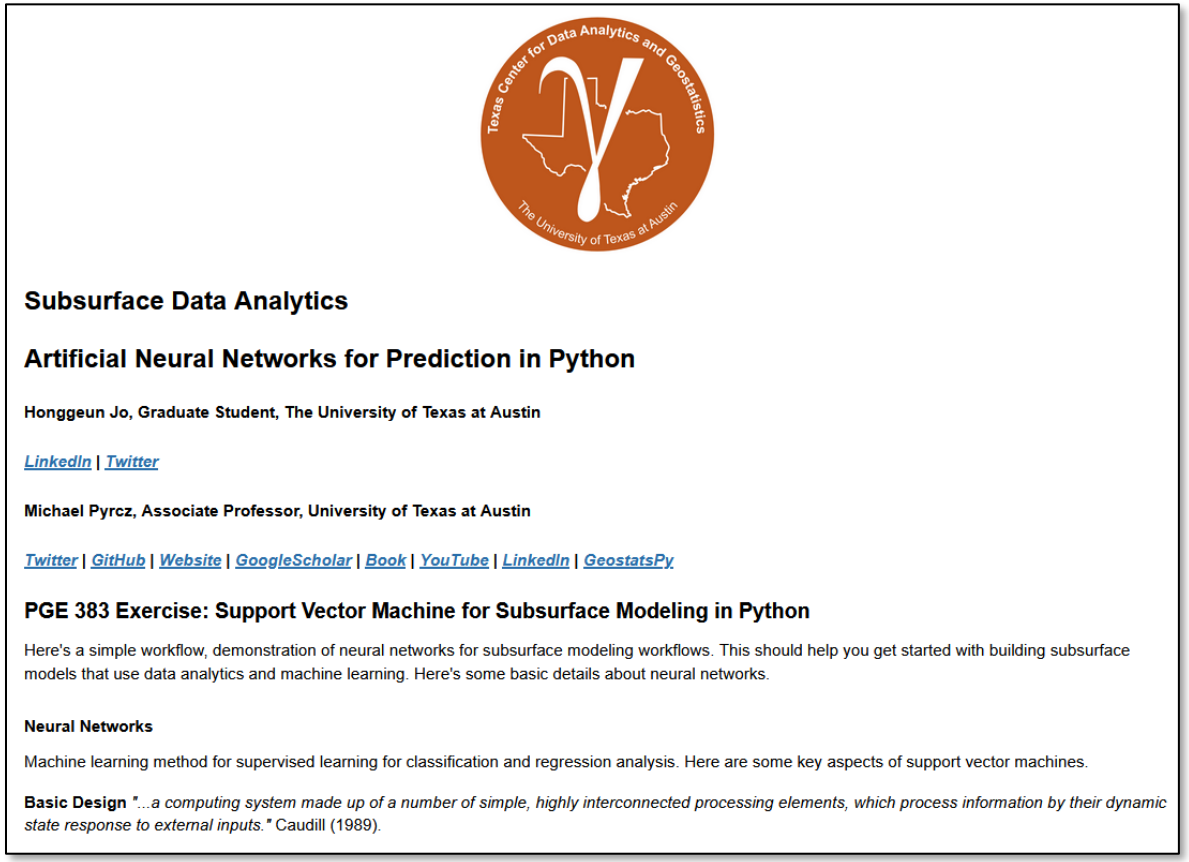
Lecture 17: Neural Networks

Lecture outline:

- **Neural Networks Hands-on**

Artificial Neural Network Demonstration

Demonstration workflow with artificial neural network for supervised learning from training data.



Subsurface Data Analytics

Artificial Neural Networks for Prediction in Python

Honggeun Jo, Graduate Student, The University of Texas at Austin

LinkedIn | *Twitter*

Michael Pyrcz, Associate Professor, University of Texas at Austin

Twitter | *GitHub* | *Website* | *GoogleScholar* | *Book* | *YouTube* | *LinkedIn* | *GeostatsPy*

PGE 383 Exercise: Support Vector Machine for Subsurface Modeling in Python

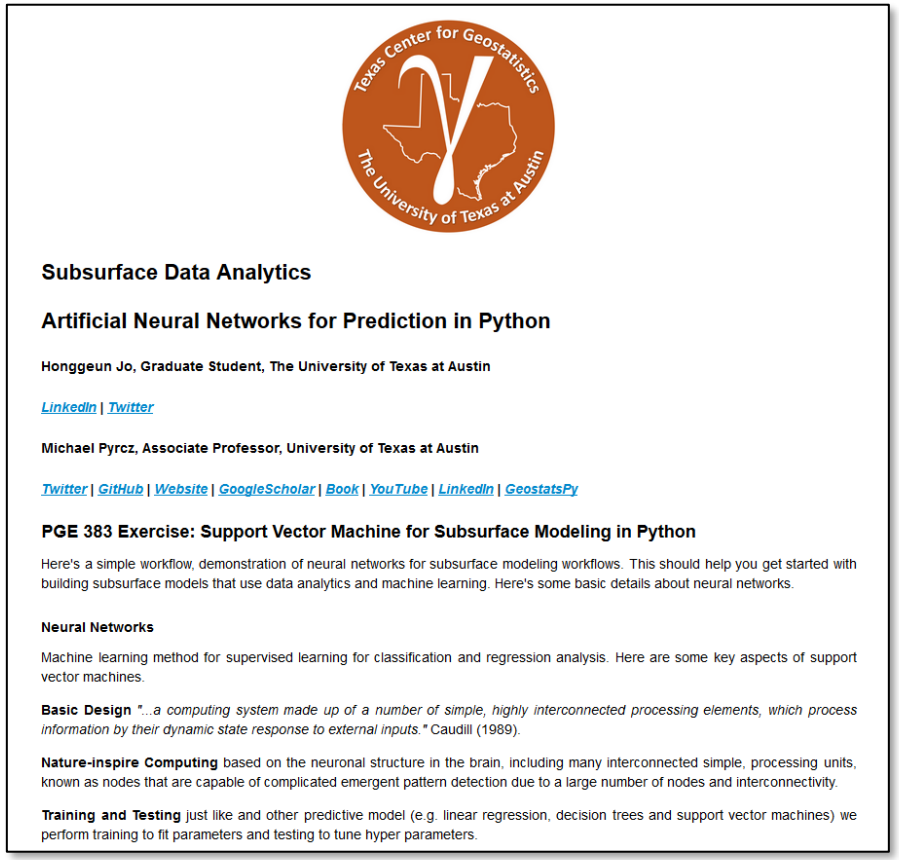
Here's a simple workflow, demonstration of neural networks for subsurface modeling workflows. This should help you get started with building subsurface models that use data analytics and machine learning. Here's some basic details about neural networks.

Neural Networks

Machine learning method for supervised learning for classification and regression analysis. Here are some key aspects of support vector machines.

Basic Design *"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."* Caudill (1989).

Above porosity map from acoustic impedance, file is SubsurfaceDataAnalytics_NeuralNet_Map.ipynb.



Subsurface Data Analytics

Artificial Neural Networks for Prediction in Python

Honggeun Jo, Graduate Student, The University of Texas at Austin

LinkedIn | *Twitter*

Michael Pyrcz, Associate Professor, University of Texas at Austin

Twitter | *GitHub* | *Website* | *GoogleScholar* | *Book* | *YouTube* | *LinkedIn* | *GeostatsPy*

PGE 383 Exercise: Support Vector Machine for Subsurface Modeling in Python

Here's a simple workflow, demonstration of neural networks for subsurface modeling workflows. This should help you get started with building subsurface models that use data analytics and machine learning. Here's some basic details about neural networks.

Neural Networks

Machine learning method for supervised learning for classification and regression analysis. Here are some key aspects of support vector machines.

Basic Design *"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."* Caudill (1989).

Nature-inspire Computing based on the neuronal structure in the brain, including many interconnected simple, processing units, known as nodes that are capable of complicated emergent pattern detection due to a large number of nodes and interconnectivity.

Training and Testing just like and other predictive model (e.g. linear regression, decision trees and support vector machines) we perform training to fit parameters and testing to tune hyper parameters.

Porosity 1D from depth with demonstration of overfit, file is SubsurfaceDataAnalytics_NeuralNet.ipynb.

PGE 383 Subsurface Machine Learning

Lecture 17: Neural Networks

Lecture outline:

- **Neural Networks**
- **Neural Networks Example**
- **Neural Networks Hands-on**