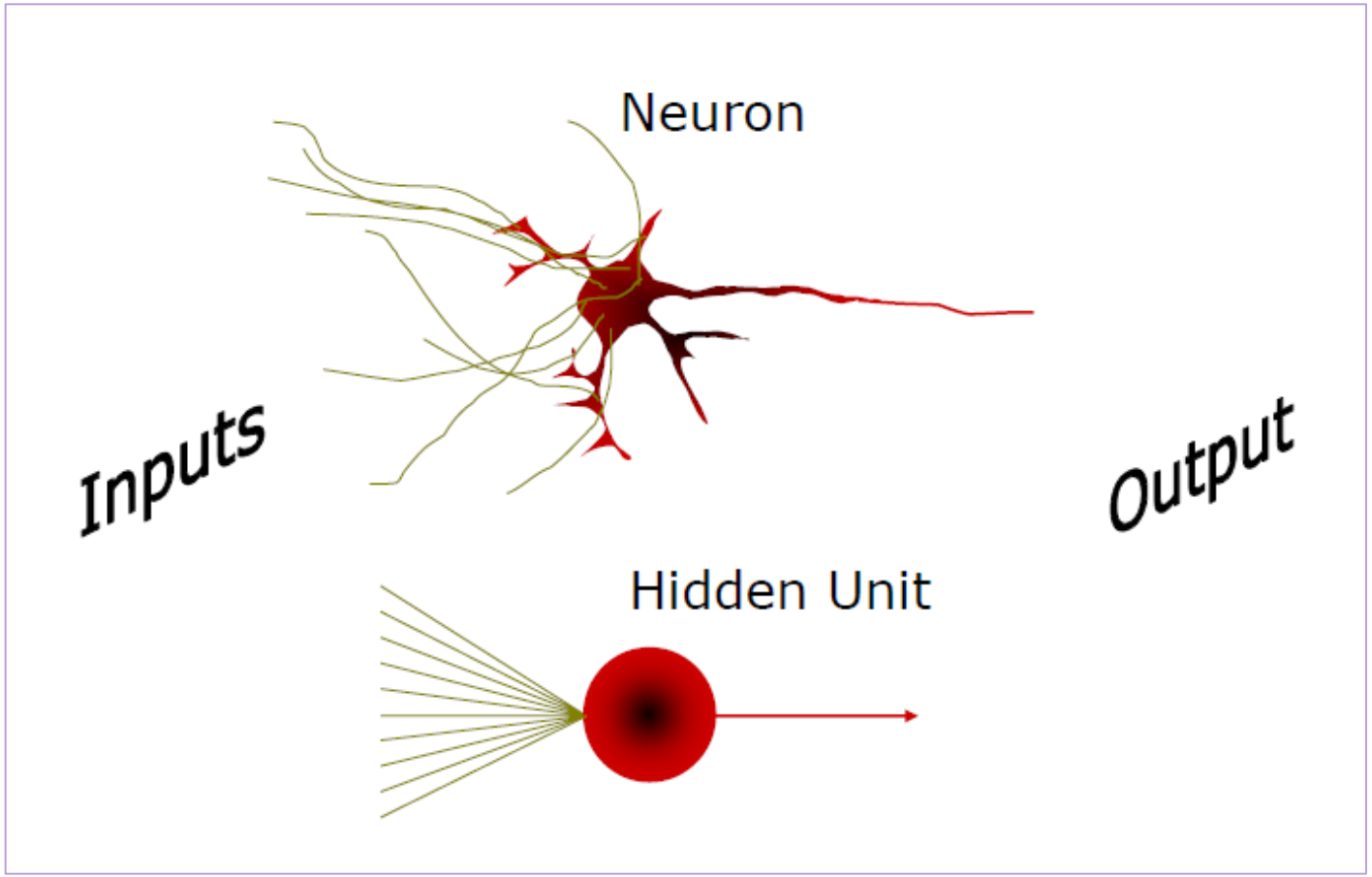
Machine Learning II

Week 3: Artificial Neural Networks

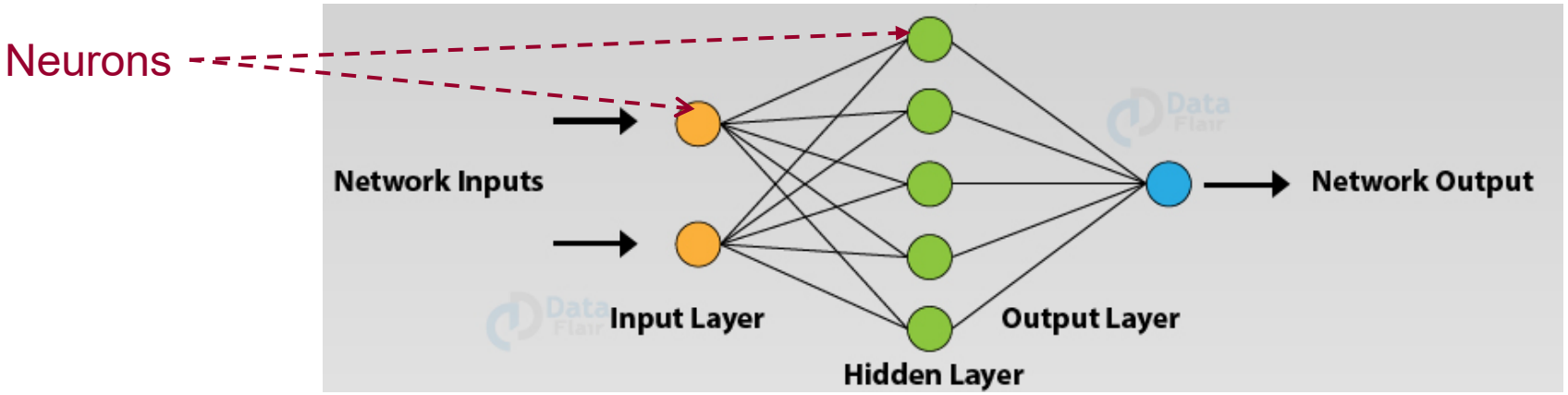
Georgetown University McDonough School of Business

Artificial Neural Networks (ANN)



Artificial Neural Networks in Machine Learning

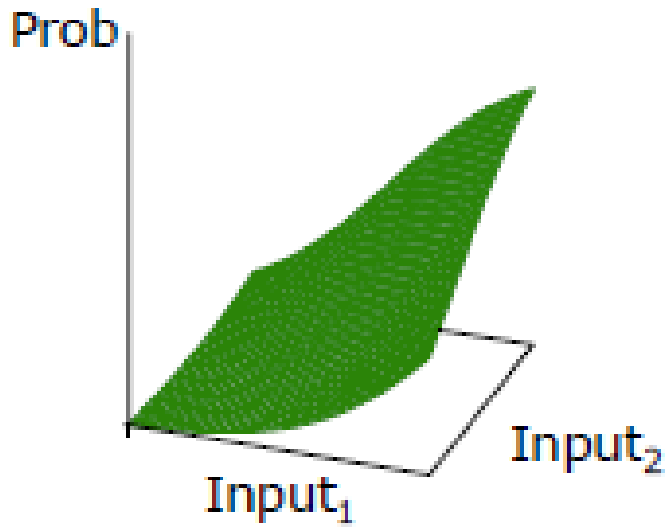
- Supervised ML Models (nonlinear model to process information in parallel)
- Organized in a network of **neurons** through **layers**
- Basic Building Blocks: Hidden units modeled after neurons
  - Each hidden unit receives a combination of input variables
  - Hidden layers transform the input variables through non-linear methods to try and best model the output variable



Limitation of Traditional Models

- In logit models, effect of each input on the logit is assumed to be linear which limits its flexibility
- Traditional nonlinear models are limited with respect to the number of inputs that they can consider
- Nonlinear and polynomial models can fail because it is difficult to specify their functional form (equation)
- Non-parametric regression models can fail because of the relative sparseness of data in higher dimensions
- Neural networks, require no specified form, often work very well in sparse, high dimensional spaces

logit(probability of target event) = linear combination of the inputs



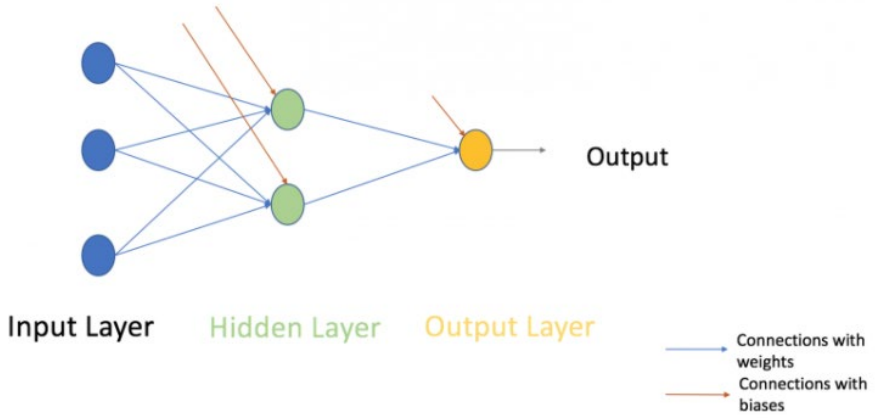
Applications of Neural Network Applications

- **Pattern Recognition**: facial recognition, object detection, fingerprint recognition, transport routing systems, etc.
- **Anomaly Detection**: fraud detection and prevention, aircraft fault detection, target tracking by military, vehicle brake system diagnosis, etc.
- **Process modeling and control** (e.g., energy and material costing, chip failure analysis in electronics)
- **Time Series Prediction**: stock price forecasting, weather forecasting, etc.
- **Natural Language Processing**: text classification, named entity recognition, speech recognition, spell checking etc.
- …

FEED FORWARD MECHANISM AND ACTIVATION FUNCTION

Methods for Creating Input-Output Relationships

- **Linear Combination of Inputs (e.g., 3 inputs):** Weights give importance to an input
  - Example: Assign w1=2, w2=3, and w3=4 to x1, x2, and x3, respectively
  - Output: w1*x1 + w2*x2 + w3*x3 > threshold
- **Add bias:** Each perceptron also has a bias, *b or* w_0, which can be thought of as how much flexible the perceptron is (*without b the line will always go through the origin (0, 0) and you may get a poorer fit)*
  - A perceptron may have two inputs, in that case, it requires three weights- one for each input and one for the bias
  - Linear representation of input in our example: w1*x1 + w2*x2 + w3*x3 + 1*b

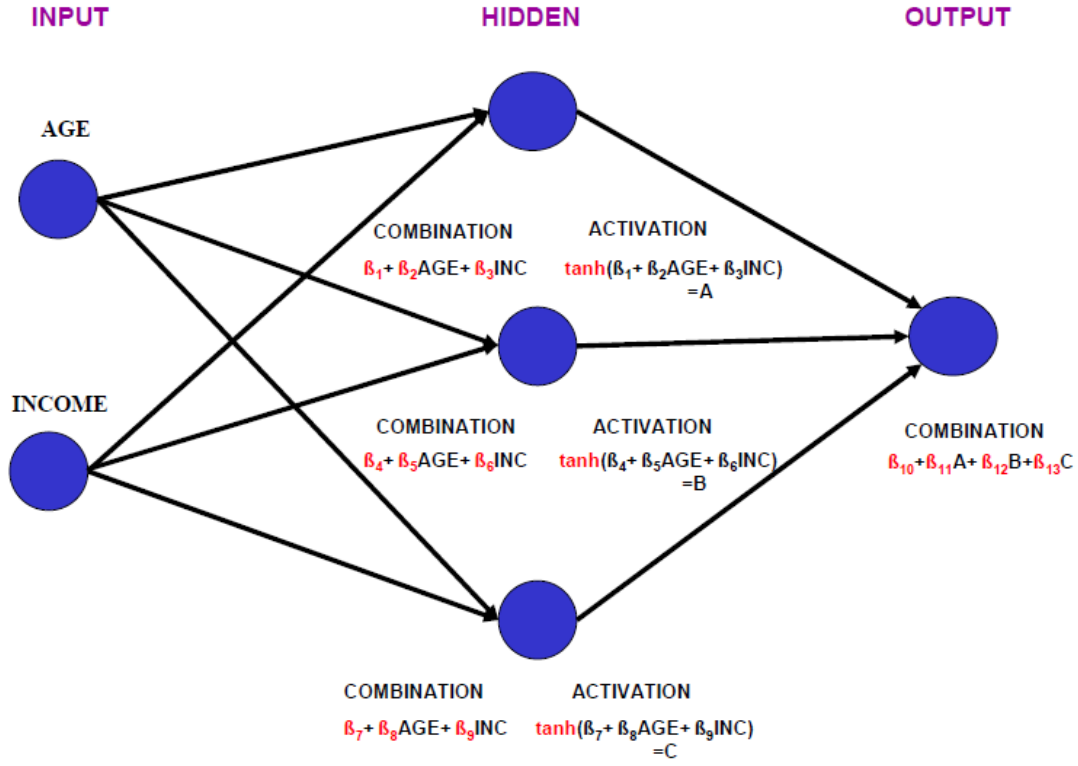


Combination Function (Example of income and age)

Linear sums the weighted inputs $\sum_{i=1}^{p} \omega_i x_i$

The inputs are combined using some weighted linear function and that is input into the hidden layer.

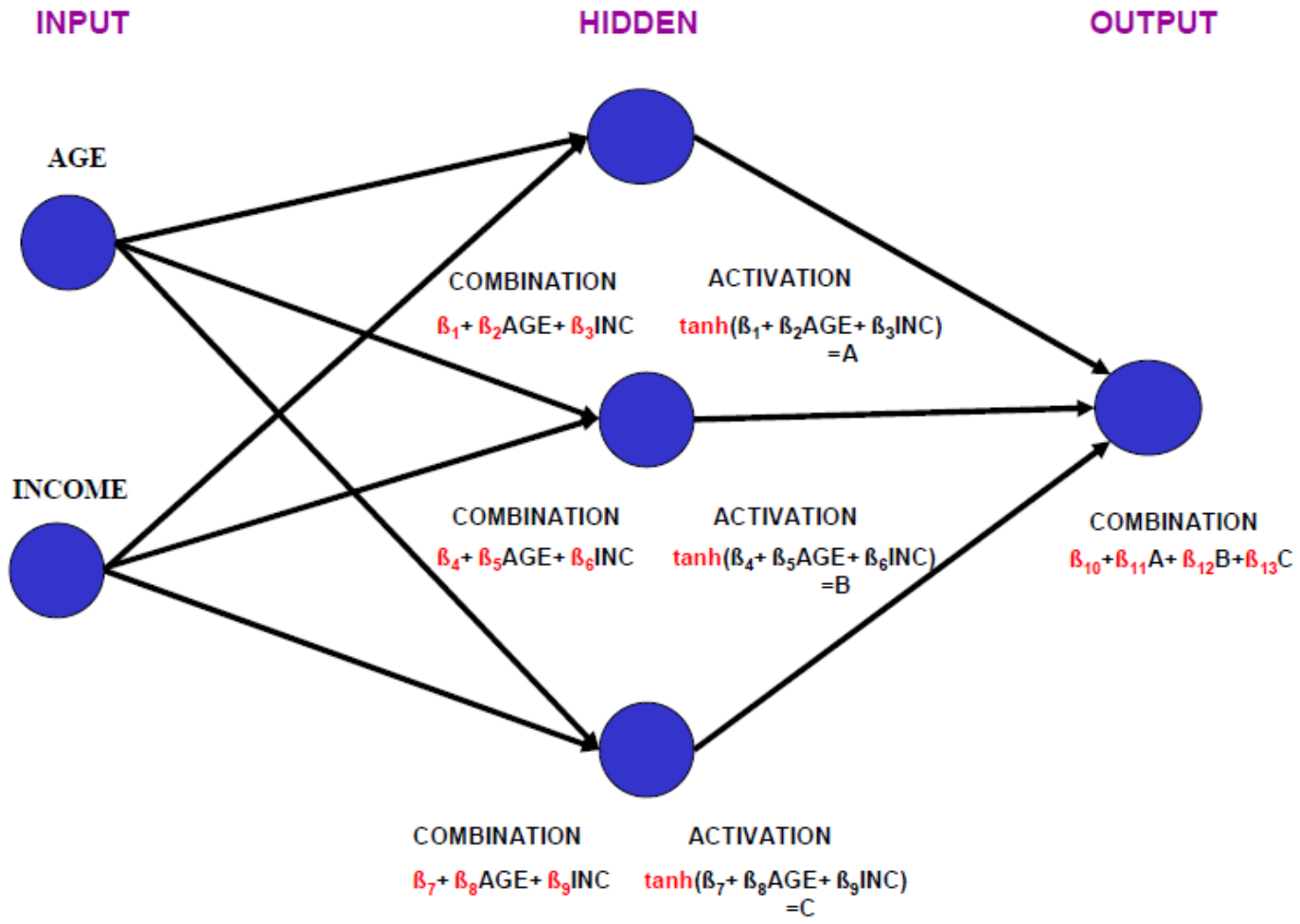
For example, age and income input variables are combined linearly here as shown in the first set of connections



Neural Network: Example

Inputs: Age and Income

- Each hidden unit outputs a nonlinear transformation called activation function of a linear combination of their inputs ((sigmoidal activation function such as tanh shown here is used)
- The output from the hidden units is linearly combined to form the input of the next layer which in this case is the output
- Feed-forward mechanism



Activation Function

- Activation Function takes the sum of weighted input (w1*x1 + w2*x2 + 1*b) and returns the output of the neuron

$$a = f(\sum_{i=1}^{N} w_i x_i)$$

In the equation, we have represented 1 as x0 and b as w0

- The activation function is mostly used to make a non-linear transformation that allows us to fit nonlinear hypotheses or to estimate the complex functions
- Multiple activation functions exist: “Sigmoid”, “Tanh”, ReLu and many others

Types of Activation Functions

- After the weighted inputs and the bias have been combined, the neuron's net input is passed through an activation function
- For the perceptron, the defining hidden unit activation functions are all members of sigmoid family
- The most famous are:
  - Sigmoid or logistic function

$$logit(\hat{p}) = \log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \log(odds)$$

  - Hyperbolic tangent (tanh) function

$$\tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

Types of Activation Functions

- Exponential function

$$\text{exponential}(net) = e^{net}$$

- SoftMax function: Multiclass target

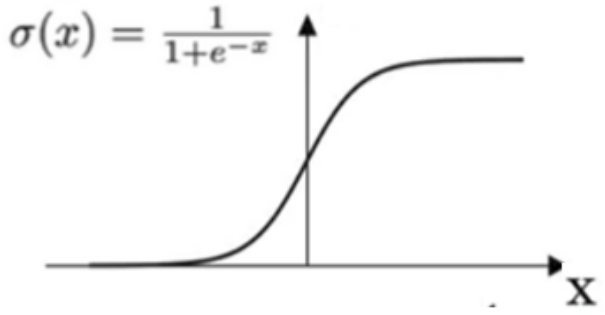
$$softmax(net_i) = \frac{e^{net_i}}{\sum_j e^{net_j}}$$

- Rectified Linear Unit (ReLu) function

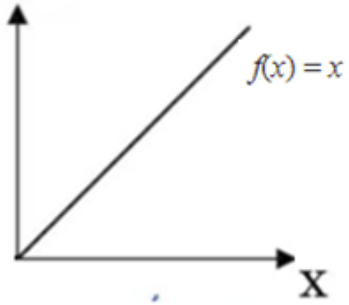
$$\text{ReLu(net)= max(0, x)}$$

Common Activation Functions

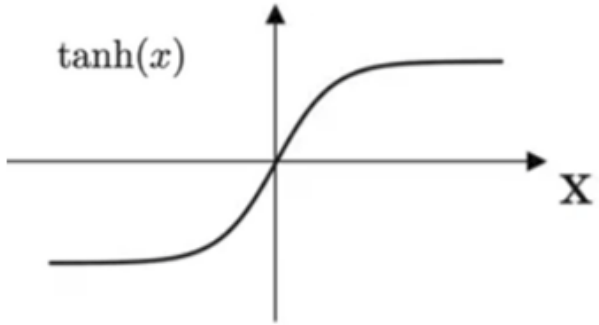
Sigmoid Function



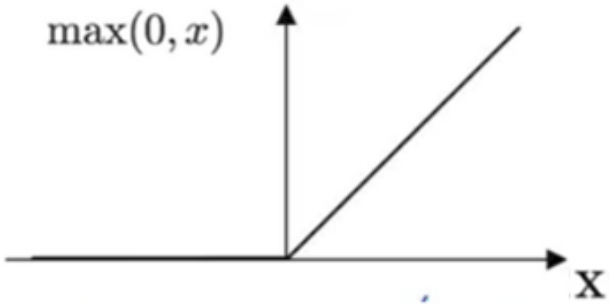
Identity Function



Hyper Tangent Function



ReLU Function



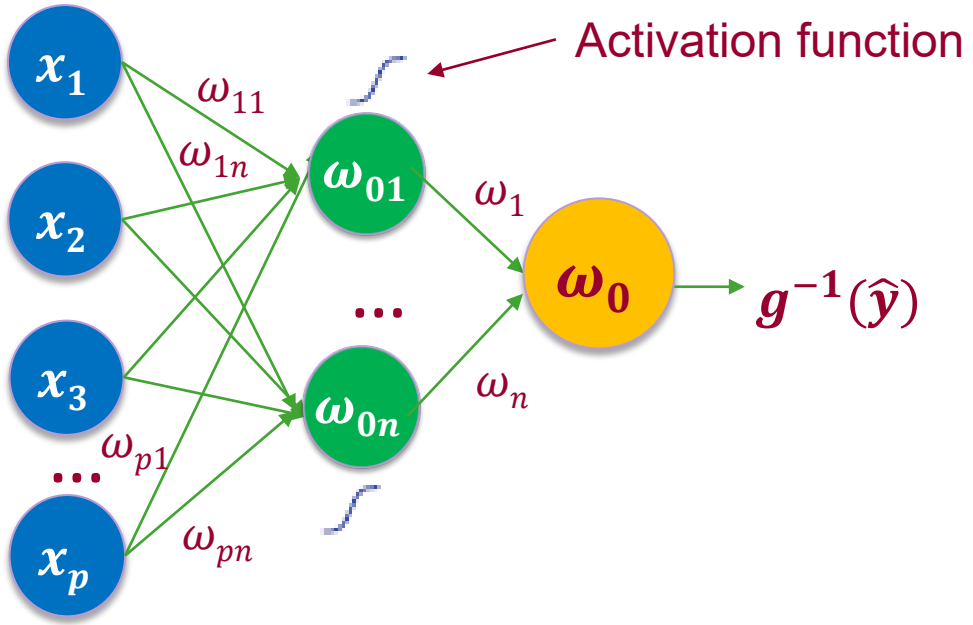
Activation Functions

Function	Output Details
Exponential	Constrains the output range of a neuron to nonnegative values (0:∞)
Identity	No transformation when you do not want to constrain the neurons activation range (-∞: ∞)
Logistic/Sigmoidal	Logit link function returns the probability (0:1) of each target class
SoftMax	Inverse logit-link function generates probability values (0:1) for each target class (also ensures that probabilities across target classes sum to 1)
tanh	Hyperbolic tangent function is sigmoidal that ranges between -1 and 1
ReLu	Rectified linear unit function maps negative inputs to 0 and positive inputs to the same output

MULTILAYER PERCEPTRONS AND HIDDEN LAYERS

Multilayer Perceptron (MLP) with One Hidden Layer

- Hidden and output layers must be connected by a nonlinear function in order to act as separate layers. Otherwise, the multilayer perceptron collapses into a linear perceptron
- Number of parameters in an MLP with *k* interval inputs grows quickly with the number of hidden units, *h*, considered
- Number of parameters to be estimated is given by the equation $h(k+1)+h+1=h(k+2)+1$



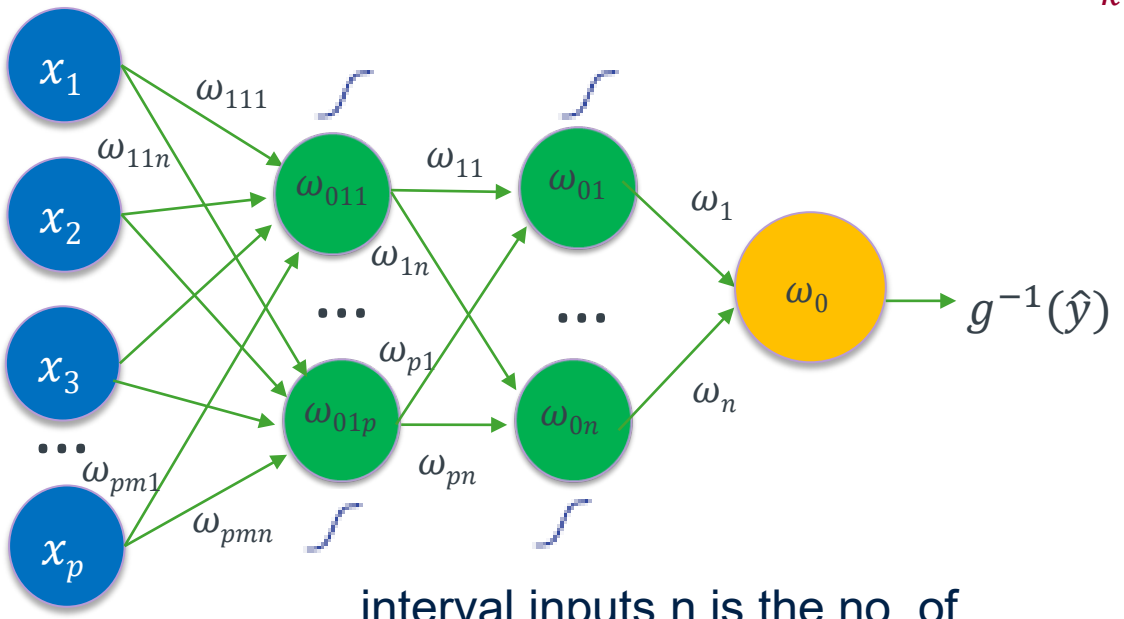
$$g^{-1}(\hat{y}) = \omega_0 + \sum_{i=1}^{h} \omega_i g_i \left(\omega_{0i} + \sum_{j=1}^{p} \omega_{ij} x_j \right)$$

hidden layer

Multilayer Perceptron with Two Hidden Layers

$$g^{-1}(\hat{y}) = \omega_0 + \sum_{k=1}^{m} \omega_k g_k \left(\omega_{0k} + \sum_{j=1}^{n} \omega_{jk} g_j \left(\omega_{0jk} + \sum_{i=1}^{p} \omega_{ijk} x_i \right) \right)$$

nested hidden layer



interval inputs n is the no. of observations

The number of parameters is given by the equation $h_1(k+1)+h_2(h_1+1)+h_2+1$

- h_1 and h_2 are number of neurons in first and second hidden layer
- p is the number of input variables (including interval inputs)
- k is the number of interval inputs
- n is the no. of observations

How Many Hidden Units (Neurons) Do You Need?

- The optimal number of units required in each hidden layer is problem specific
- Guidelines:
  - The number of units in the first hidden layer should be about twice the number of input dimensions
  - The number of units in the second hidden layer reflects the number of distinct regions needed
- The optimal number can be determined empirically

ERROR /LOSS FUNCTION

Error in Estimating Output

- What if the estimated output from Neural Network is far away from the actual output (high error)?
  - Update the biases and weights based on an error criterion
  - This weight and bias updating process is known as "**Back Propagation**" (we will look at that in the next section)

Simple Example: Neural Network

Input	Actual Output of Model
0	0
1	3
2	6
3	9
4	12

$$y = W * x$$

*W is called the **weights** of the network*

Random Initialization $\hat{y} = 3 * x$

Forward Propagation
Input>Network>Output

Simple Example: Neural Network

Input	Actual Output of Model	Desired Output
0	0	0
1	3	2
2	6	4
3	9	6
4	12	8

Loss or Error Function: Performance metric on how the NN manages to reach its goal of generating outputs close to desired values

Error/Loss = Desired output – Actual output

Simple Example: Neural Network

$$\hat{y} = 3 * x$$

Input	Actual Output of Model ($\hat{y}$)	Desired Output	Absolute Error	Squared Error
0	0	0	0	0
1	3	2	1	1
2	6	4	2	4
3	9	6	3	9
4	12	8	4	16
Total:	..	..	10	30

Loss = Absolute value (Desired output – Actual output)

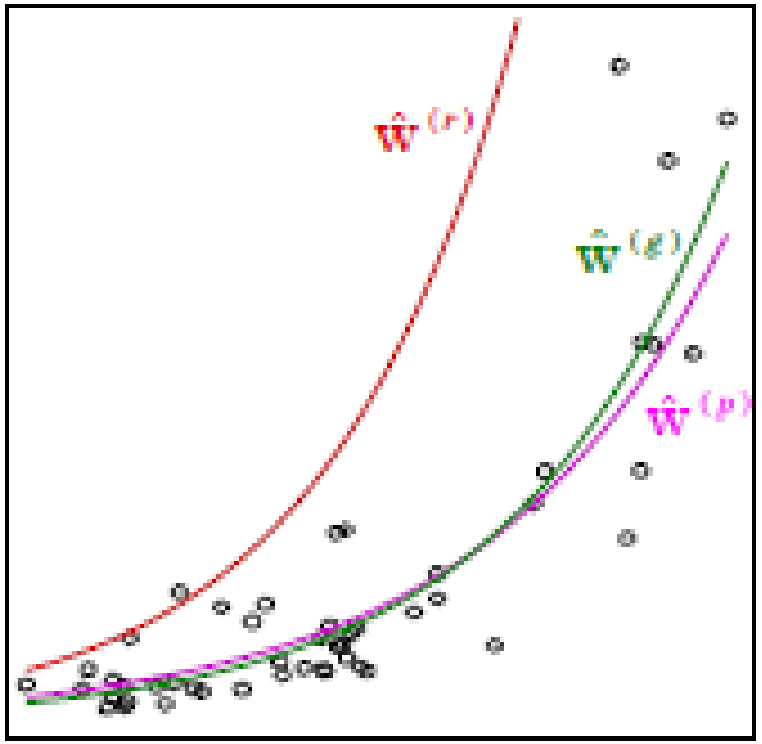
Loss = Sum of squares of absolute errors

Loss = How much precision do we lose (minimize this)

Choosing Error functions

$$Q(w) = \sum_{i=1}^{p} (yi - \mu_i(w))^2$$

- Goal is to find the parameter set that minimizes the specified error function
- The most common error estimate is the root mean squared error (RMSE) as used in the Ordinary Least Squares (shown in figure)
  - Q_w is the error estimate, y_i is the i^{th} target value, and $\mu_i(w)$ is model's estimate of the i^{th} target, given weight vector ***w***



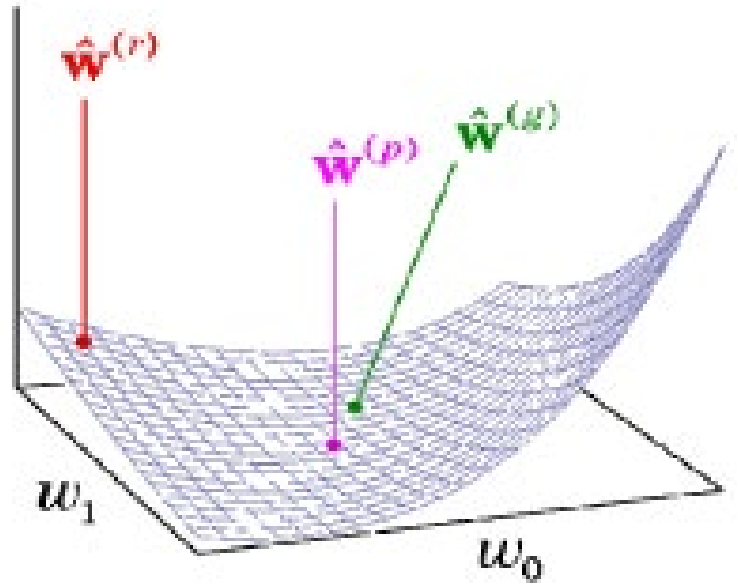
$$Error, Q(\hat{w}^r) = 1128$$
$$Error, Q(\hat{w}^r) = 259$$
$$Error, Q(\hat{w}^r) = 236$$

Other Error Functions

- Maximum Likelihood (-2 LL- Minimize)

$$Q(w) = -\ln\left(\prod_{i=1}^{n} pdf(y_i, \mu(w))\right)$$

y is the target value and $\mu(w)$ is its expected target value



- Deviance: If the probability distribution is a type of exponential family, minimizing deviance is equivalent to maximizing likelihood
- Entropy measure: minimize

Activation and Error Function Combinations

Target	Link Function	Activation Function	Error Function
Continuous	Identity	Identity	Deviance, RMSE/AIC/SBC
	Log	Exponential/Relu	Deviance/AIC/SBC
Class (nominal or ordinal)	Logit (2 categories)	Sigmoid/Tanh/Relu	Deviance/Entropy/ Mis-classification
	Generalized Logit (more than 2 categories)	SoftMax/Relu	Entropy/Misclassification

Simple Example: Neural Network

Optimization: We can use any optimization technique that modifies the internal weights of neural networks to minimize loss function

Differentiation: Derivative of the loss function (gives the rate at which this function changes its values at a point)

Effect of Derivative: How much the total error will change if we change the internal weight of the neural network with a certain small value (δ_W).

Simple Example: Neural Network

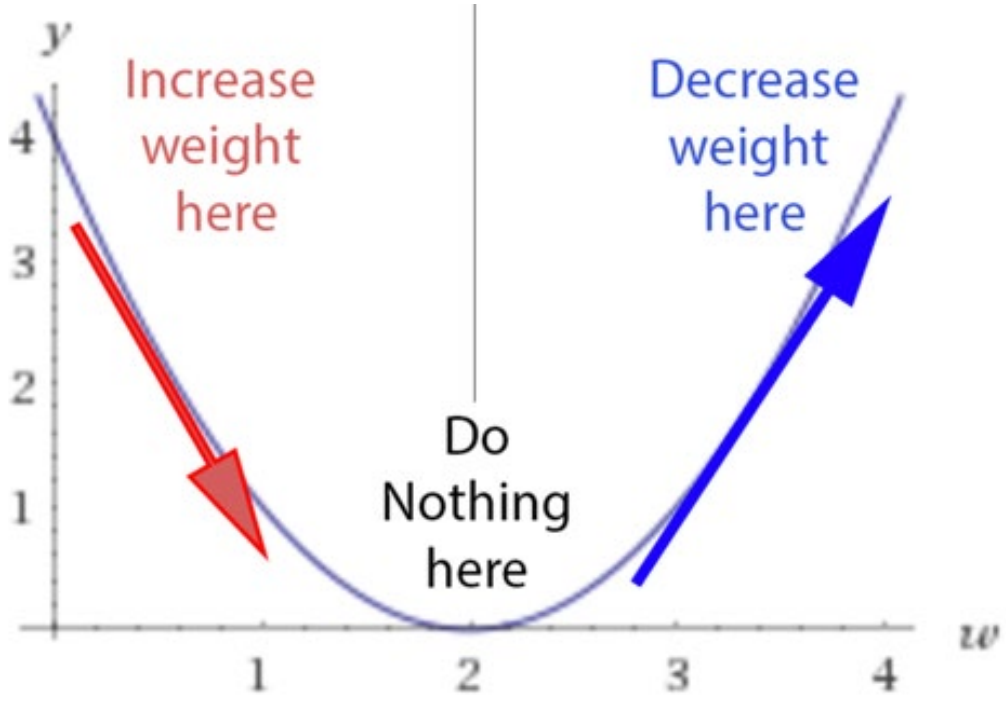
Input	Desired Output	W=3	rmse(3)	W_1=3.0001	rmse(W_1)
0	0	0	0	0	0
1	2	3	1	3.0001	1.0002
2	4	6	4	6.0002	4.0008
3	6	9	9	9.0003	9.0018
4	8	12	16	12.0004	16.0032
Total:	..	..	30	..	30.006

$$\delta_W = 0.0001$$

Derivative = rate of change in total error due change in weight
= 0.006/0.0001 = 60 times increase with increase in W by 0.0001

Loss Function in this Example

- Let's check the derivative
- If it is positive, meaning the error increases if we increase the weights, then we should decrease the weight
- If it's negative, meaning the error decreases if we increase the weights, then we should increase the weight
- If it's 0, then we do nothing, and then we reach our stable point



Gradient Descent

BACK PROPAGATION AND GRADIENT DESCENT

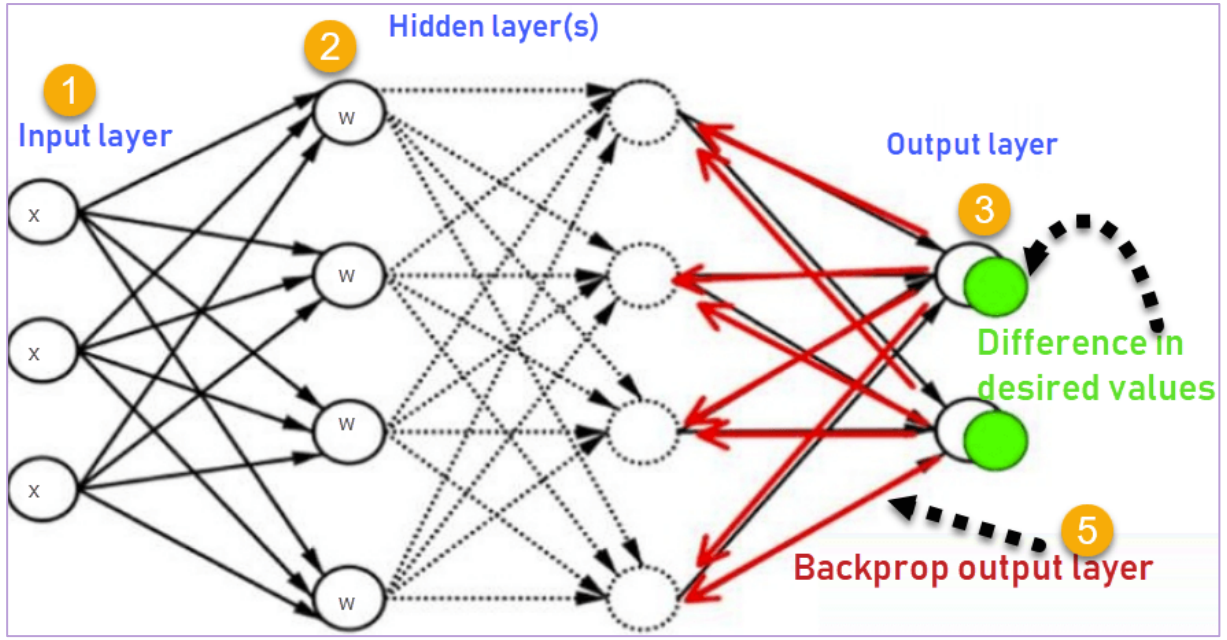
Back Propagation

- What if the estimated output is far away from the actual output (high error)?
  - Update the biases and weights based on an error criterion
  - This weight and bias updating process is known as "**Back Propagation**"
- Back-propagation (BP) algorithms work by determining the loss (or error) at the output and then propagating it back into the network
  - The weights are updated to minimize the error resulting from each neuron
  - Subsequently, the first step in minimizing the error is to determine the gradient (Derivatives) of each node w.r.t. the final output
- One iteration of forwarding and backpropagation iteration is known as "**Epoch**"

Methodology for Back Propagation of Errors

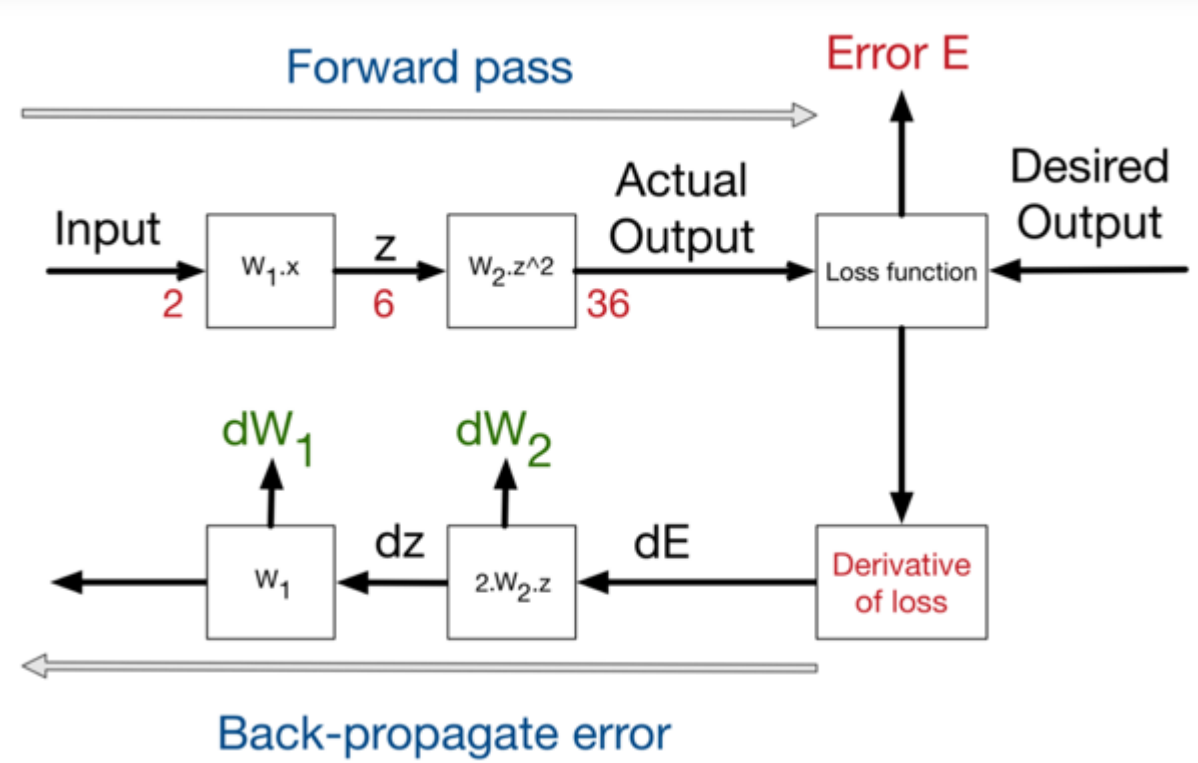
It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration)

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights w: The weights are randomly selected
3. Calculate the output for every neuron from input layer, to the hidden layers, to the output layer
4. Calculate the error in the outputs
   Error = actual output-desired output
5. Go back from the output layer to the hidden layer to adjust the weights such that error reduces
6. Keep repeating the process until desired output is achieved



Back Propagation

Input > Forward calls > Loss function > Derivative > Backpropagation



Weight Update

Weight update: The derivative is the rate of which the error changes relative to the weight changes.

New weight = old weight – Derivative Rate * Learning Rate

Learning rate: constant (usually very small) in order to force the weight to get updated very smoothly and slowly

Derivative rate =+ ve (an increase in weight will increase the error) > new weight = smaller

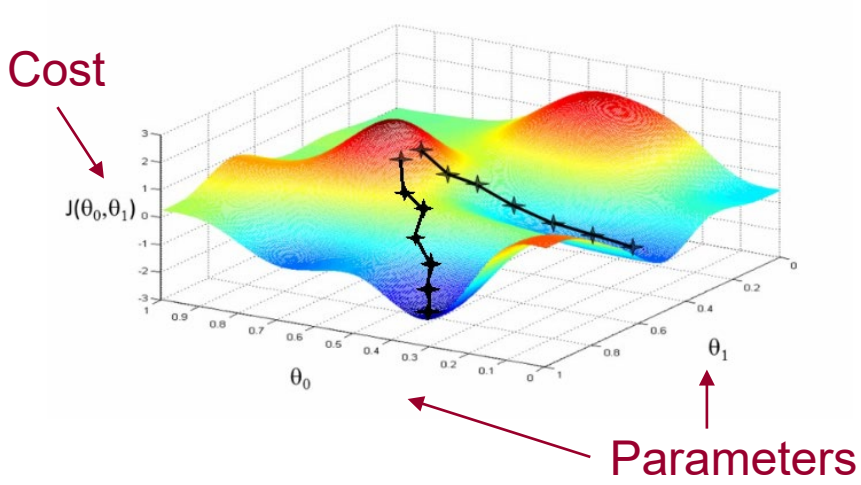
Derivative rate = – ve (an increase in weight will decrease the error) > new weight = larger

Derivative = 0 (we are in a stable minimum) >No update on weights

Iterate until Convergence

- Since weights are updated with small delta at a time, it takes several iterations in order to learn
- How many iterations are needed to converge (depends on)
  - How strong is the learning rate? High learning rate = faster learning, but with higher chance of instability
  - Meta-parameters of the network (how many layers, how complex the non-linear functions are). The more variables = the more it takes time to converge (but higher precision)
  - Optimization methods used: some weight updates rule are proven to be faster than others
  - Random initialization of network
  - Quality of the training set. If input and output has no correlation, the NN will not learn a random correlation

Optimizing Training Algorithm: Gradient Descent



Hills: high cost
Valleys: Low cost

Two types of gradient descent:

- Full batch gradient descent (whole data)
- Stochastic gradient descent (sample)

We want to find the best parameters (θ1) and (θ2) for the learning algorithm
Cost function: How would the algorithm perform when we choose particular value for a parameter?