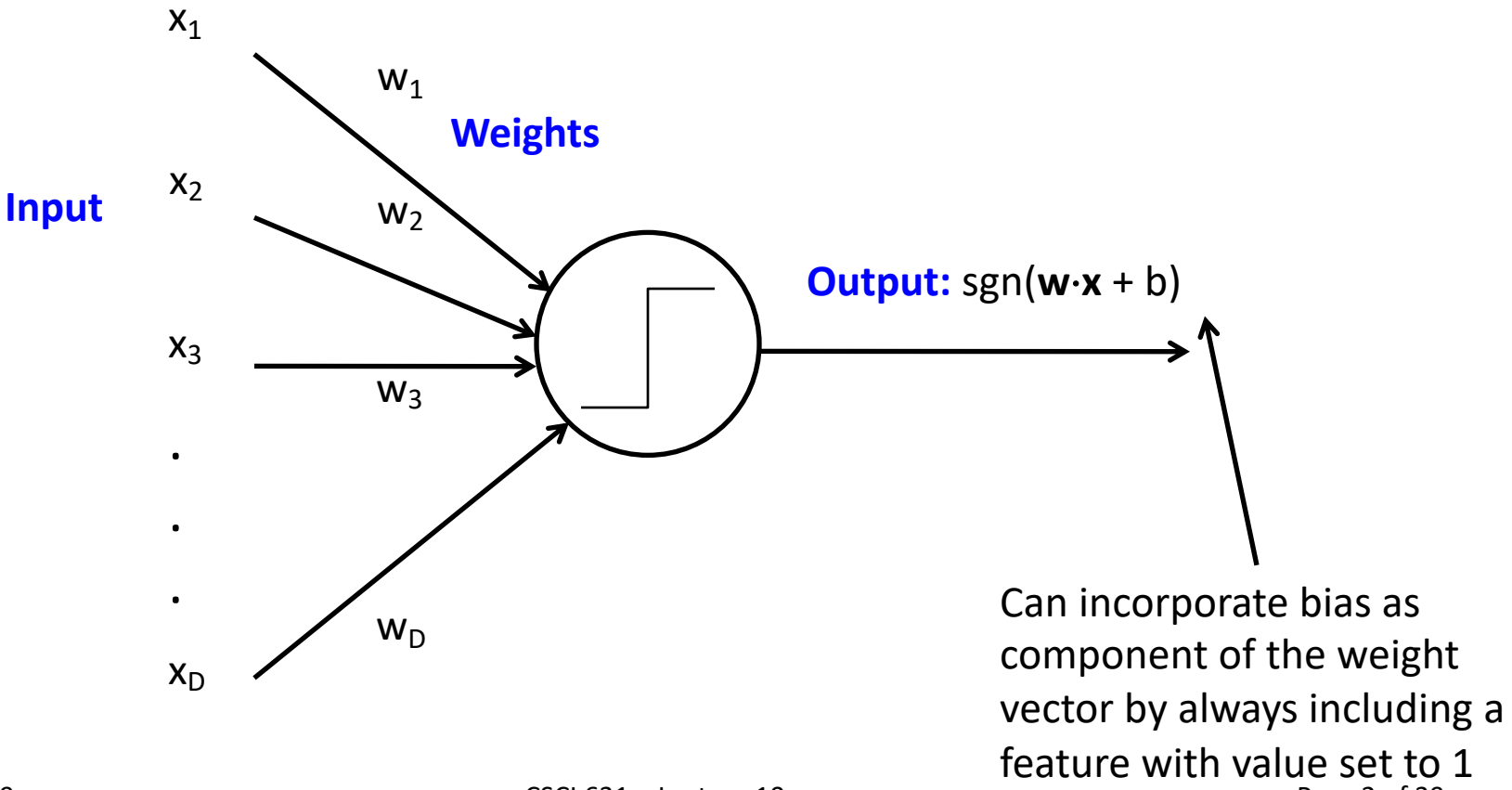
CSCI 631
Foundations of Computer Vision

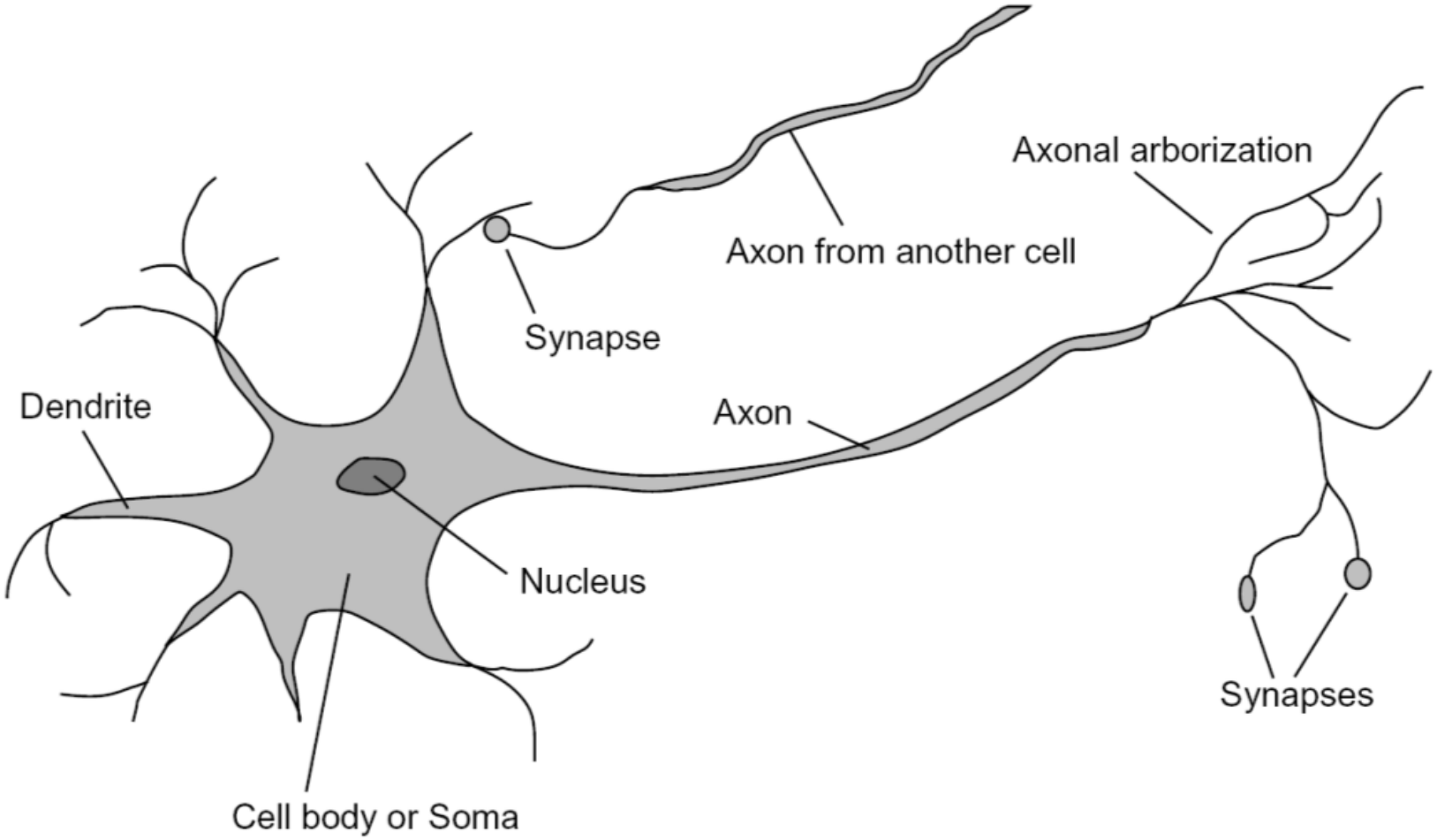
Ifeoma Nwogu
ion@cs.rit.edu

Artificial Neural Networks

Logistic regression unit (or perceptron or single neuron)

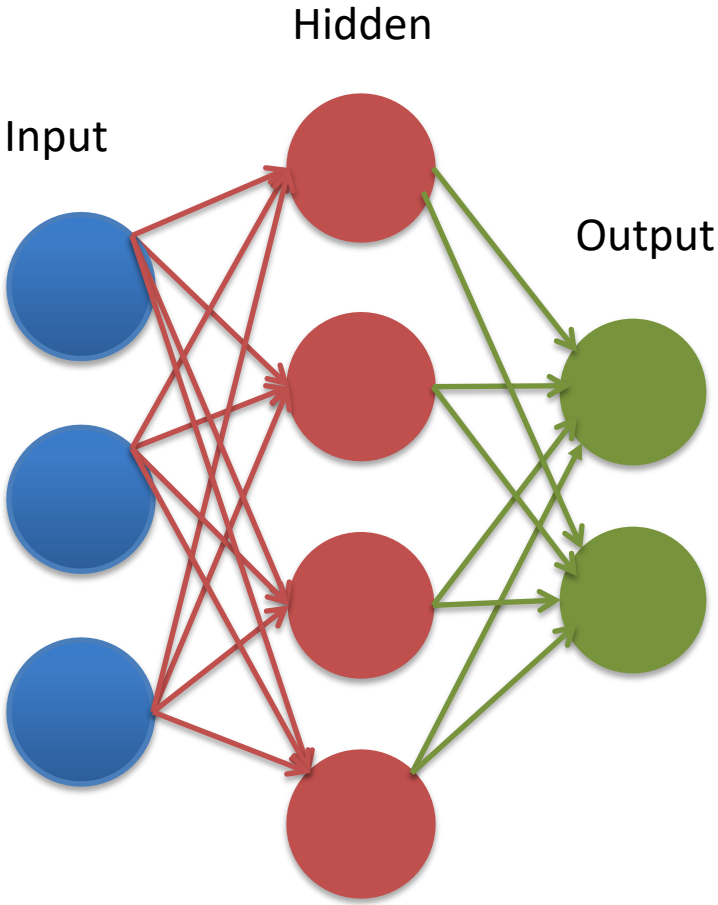


Loose inspiration: Human neurons



From LR to neural networks

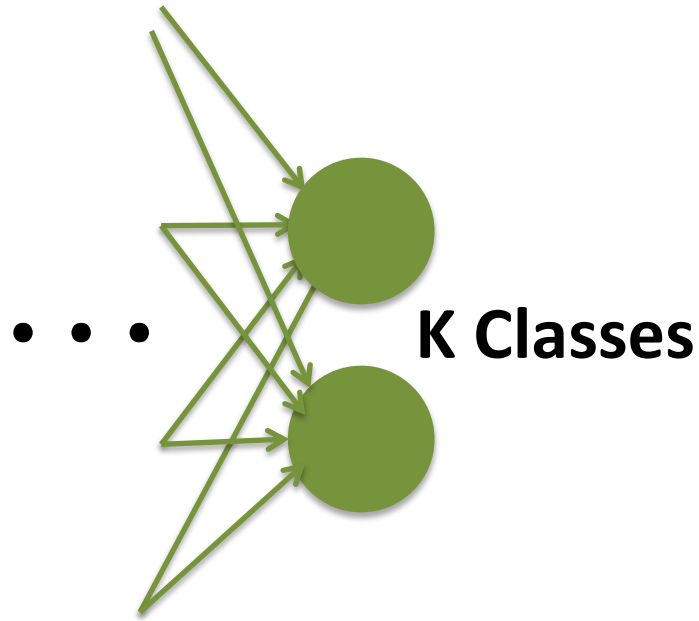
- More complex architecture
  - a network of neurons
- NN have 1+ hidden layers
- Every node in one layer is connected to every node in the next layer
- There can be one or more output nodes



Nonlinear activation functions

- Sigmoid(x) = $\frac{1}{1+e^{-x}}$
- Tanh(x) = $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- ReLU(x) = $\max(0,x)$
  - ReLU work extremely well for problems in vision

Multi-class classification



- K classes => k output nodes
  - K-output vector
- Targets are represented as an **indicator matrix**
- Unlike binary classification, no rounding with softmax
  - Labels are 0,1,2,3…
  - Softmax outputs probabilities between 0 and 1

Categorization of softmax outputs

$$\begin{bmatrix} 0 \\ 4 \\ 1 \\ 3 \\ 3 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Ind(n,k) = \begin{cases} 1 & \text{if } Y(n) == K \\ 0 & otherwise \end{cases}$$

Softmax outputs continued

- Predictions look like
  - [0.1, 0.2, 0.4, 0.1, 0.2]
  - assume probability of target is highest
- Target labels look like:
  - [0, 0, 1, 0, 0]; target class label is **2**

$$\begin{bmatrix} 0.2 & 0.2 & 0.5 & 0.1 \\ 0.7 & 0.1 & 0.1 & 0.1 \\ 0.3 & 0.5 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.3 \\ 0.1 & 0.5 & 0.2 & 0.2 \end{bmatrix}$$

PREDICTIONS

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TARGETS

From binary to multiclass classification

From:

- Input = array of 48*48 pixels; Output = +ve/-ve face

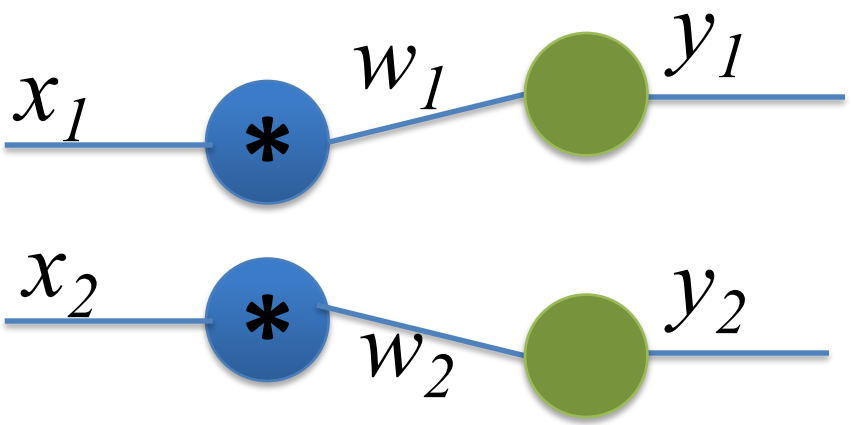
To:

- Input = array of 48*48 pixels; Output = happy, sad, angry, surprised, disgusted, afraid, neutral

- Other examples…? MNIST?

Beyond logistic regression (multiclass)

- For a binary class
  - We only need one output node to represent the probability of the class we are interested in
  - If P(Y=1|X); then P(Y=0|X) = 1 - P(Y=1|X)
  - Alternatively, we could introduce **2 output nodes** so that:



Beyond logistic regression (multiclass)

- P(Y=1|X) = $\frac{e^{a_1}}{e^{a_1} + e^{a_2}}$
- P(Y=2|X) = $\frac{e^{a_2}}{e^{a_1} + e^{a_2}}$
- These sum to 1

Beyond logistic regression (multiclass)

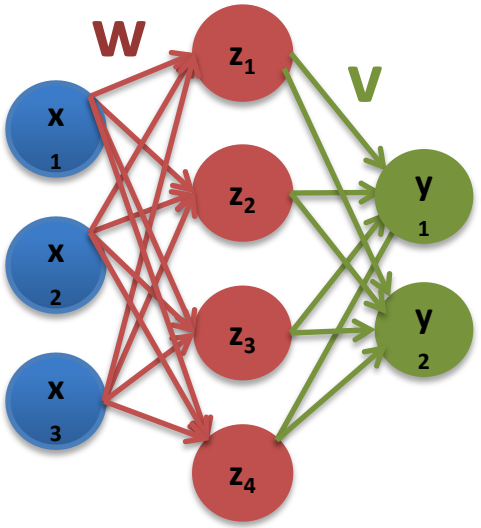
- Extending this notion to k classes
- $P(Y=m|X) = \frac{e^{a_m}}{e^{a_1} + e^{a_2} + \cdots + e^{a_k}}$
- This classifier is known as **Softmax** classifier

Softmax cross-entropy function

$$J = -\sum_{i}^{nclasses} t_i \log y_i$$

Can we derive the previous function from this general form?

Feedforward



How predictions are made:

- Inputs are fed into the blue nodes and multiplied by the weights connecting blue to red nodes giving $w^T x$
- In the red nodes, a nonlinear activation function is applied to $w^T x$, yielding new values z.
- These results z act as the input to the next layer and are multiplied by the green weights giving $v^T z$
- This process is continued for as many layers as there are
- Finally, softmax is applied at the last layer (for classification)

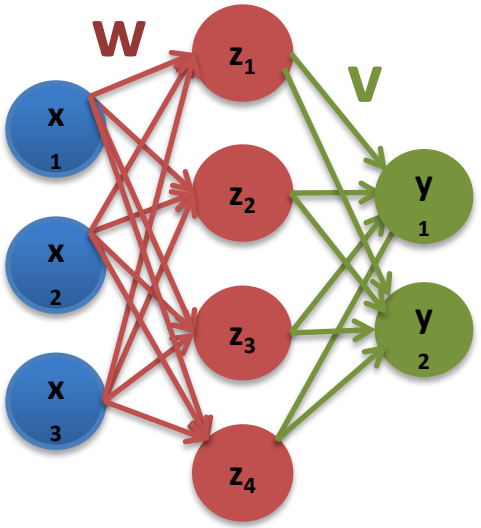
Symbols

- X is the input data matrix having N rows and D cols
  - N = number of data samples available
  - D = number of dimensions or features for each sample
- Y is the predicted output matrix having N rows and k cols (for a k-classification problem)
  - For binary classification, Y is Nx1
- T is a target value, also N x k (or Nx1)

Symbols cont'd

- J is the cost/loss/error function
  - We want to minimize the cost function
- L is the likelihood function P(data|params)
  - We want to maximize the likelihood (-ve J)
  - ℓ is the log likelihood (log L)
    - We often maximize the log likelihood instead

Backpropagation



How neural network weights are learned:

- Errors are propagated backwards starting from the last layer
- Green weights **V** depend on the errors at the output (between the targets and predictions) - softmax
- Red weights **W** depend on the errors at the red hidden nodes – gradient descent
- The pattern continues backwards for as many layers as there are…
- Weights are updated based on propagated errors

Pseudocode for NN (feedforward)

Assign all network inputs and output
Initialize all weights with small random numbers between -1 and 1

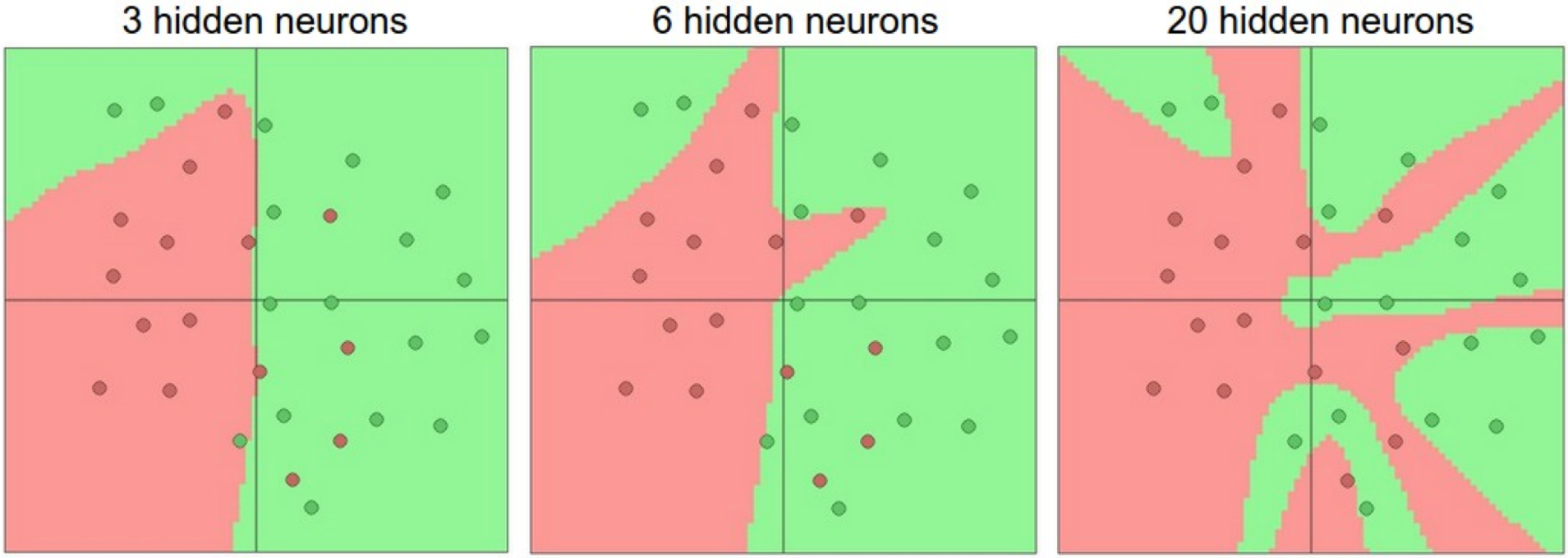
```
while ((max_num_iter < spec_iters) AND  (curr_err is > spec_err))
do  //one epoch
   for every data point in the training set
      Present the point as input to the network
      //Propagate input forward through the network:
         for each layer in the network
            for every node in the layer
               1. Calculate weighted sum of the inputs to the node
               2. Add the bias to the sum
               3. Calculate the activation for the node
         end
     end
```

Pseudocode for NN (backprop)

```
//Propagate the errors backward through the network
for every node in the output layer
      calculate the error signal
end
for all hidden layers
   for every node in the layer
         1. Calculate the derivatives at the node (wts and biases)
         2. Update each node's weight in the network
   end
end

//Calculate Global Error
Calculate the Error Function
end // for every data point
end /while…do
```

Neural Networks



Can represent nonlinear functions (when built from composition of logistic regression units or neurons)

Source: http://cs231n.github.io/neural-networks-1/

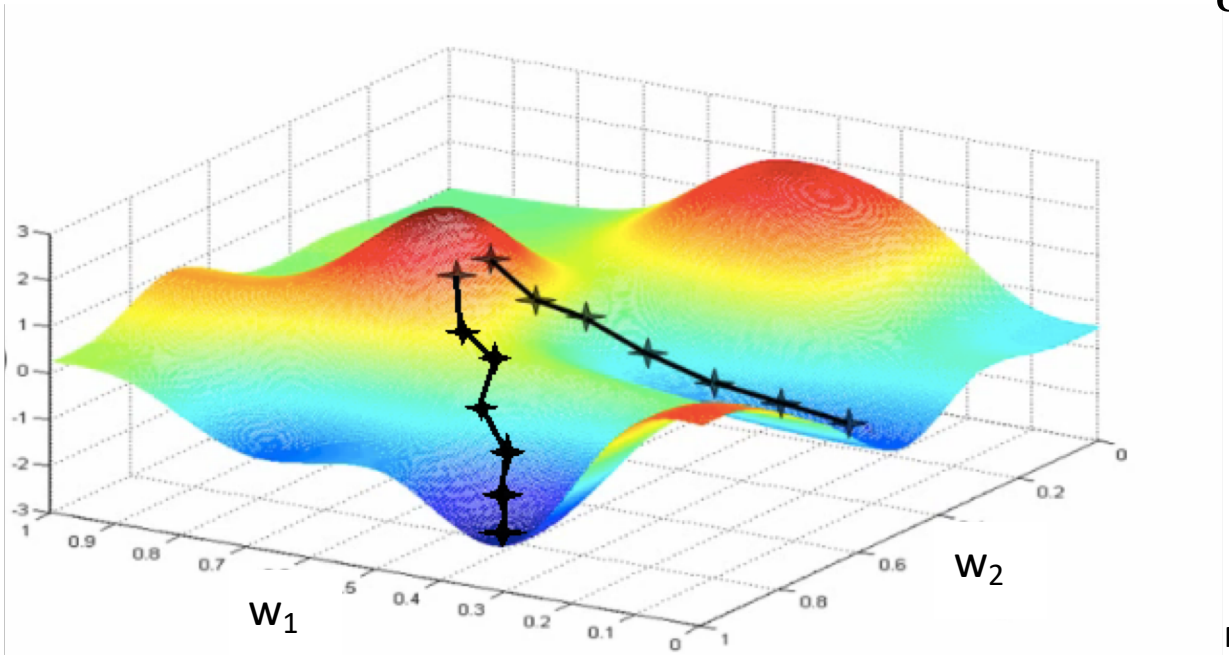
Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$



Multi-Layer Neural Networks

- Beyond a single hidden layer:

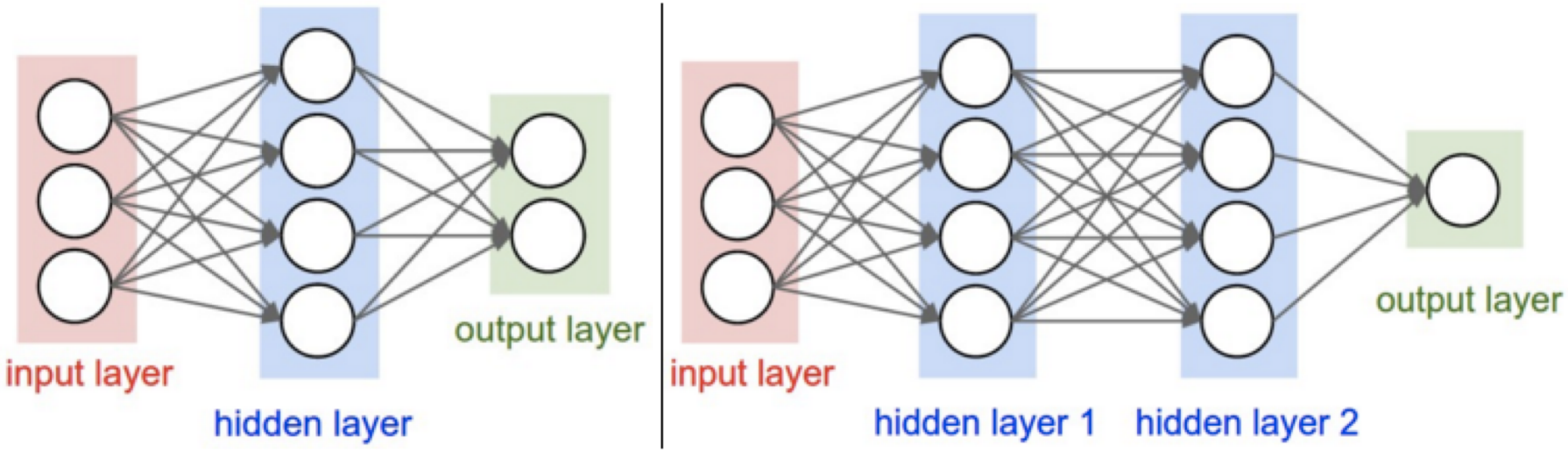


Figure source: http://cs231n.github.io/neural-networks-1/

Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Choosing activation functions

- The last layer activation function depends on your problem – regression/classification
  - Softmax
- The activation function of hidden layers is more interesting:
  - Sigmoid not good for v deep networks; vanishing gradients; get very small with recursion
  - ReLu nice for vision problems, but in other cases can cause a “dead neuron”

Choosing activation functions

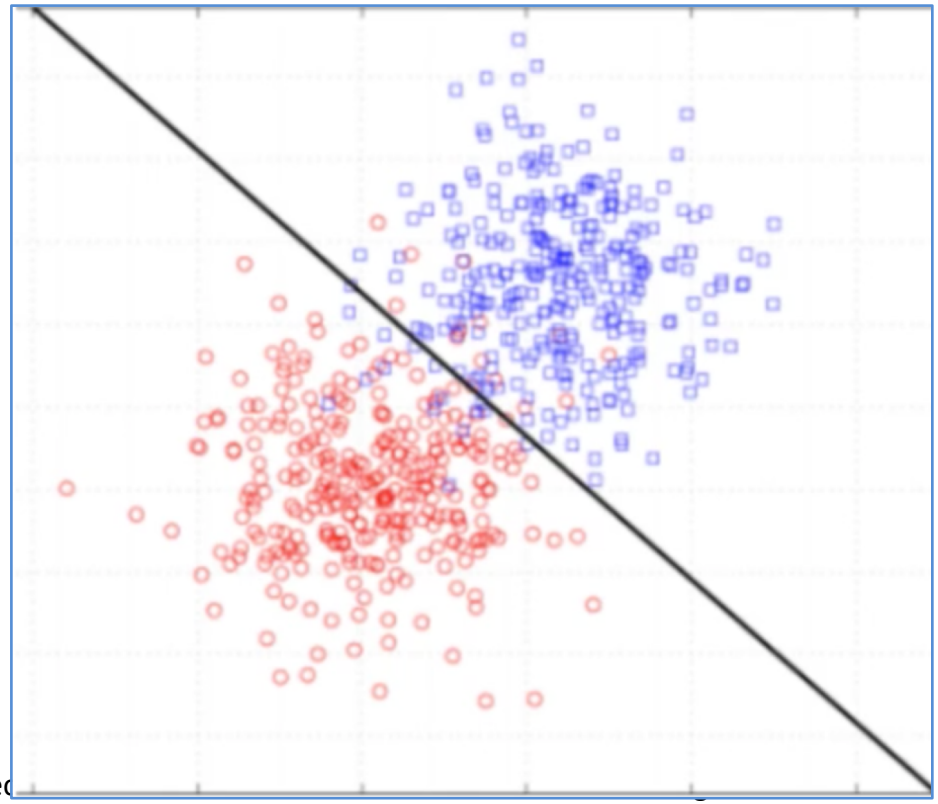
- Which has stronger gradient values?
  - Between –1 and +1, the derivative of the tanh exists between 0.42 and 1
  - Between 0 and 1, the derivative of the sigmoid exisits between 0.20 and 0.25

Neural networks: Pros and cons

- Pros
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- Cons
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power may be required to get good performance
  - The space of implementation choices is huge (network architectures, parameters)

Regularization and overfitting

- Major problem with our cost function
  - Ideal weights are infinite!
- Given the data distribution below:
- Solved weights are:
  - [0, 4, 4]
  - $[w_0, w_1, w_2]$
  - Equation of this line?



Ideal solution = infinite weights!

$$J = -\sum_{n=1}^{N} t_n \log(y_n) + (1 - t_n) \log(1 - y_n)$$

- Taking a test point x_1=1, x_2 = 1 with label t_1 = 1;
- If weights = [0,1,1] then,

  $\boldsymbol{\sigma}(w_0 + w_1x_1 + w_2x_2) = \boldsymbol{\sigma}(2) = 0.88080$

  y = 0.88080 => J = 0.126928
- If weights = [0,4,4] then,

  $\boldsymbol{\sigma}(w_0 + w_1x_1 + w_2x_2) = \boldsymbol{\sigma}(8) = 0.99966$

  y = 0.99966 => J = 0.00034

Ideal solution = infinite weights!

- If weights = [0,10,10] then,

  $\sigma(w_0 + w_1x_1 + w_2x_2) = \sigma(20) = 0.9999$

  $y = 0.9999 \quad \Rightarrow \quad J = 2.0611 \times 10^{-9}$

- Ideal weights = [0, ∞, ∞]

- When we have disproportionately large weights, the network starts to overfit.

Regularization – L2

- Regularization penalizes very large weight!
  - We accomplish this by adding a penalty function to our initial cost function

$$J_{reg} = J + \frac{\lambda}{2}\|W\|_2^2$$

Penalty function

  - λ is a smoothing parameter usually between 0.1 and 1, but is data dependent

Solving for w from J_{reg}

- Gradient descent is now $\frac{\partial J_{reg}}{\partial w}$ not $\frac{\partial J}{\partial w}$

- The original derivative is not affected by the addition, so we just differentiate the penalty and add

$$J_{penalty} = \frac{\lambda}{2}\left(w_0^2 + w_1^2 + w_2^2 ....\right)$$

$$\frac{\partial J_{penalty}}{\partial w_i} = \lambda w_i$$

L2 regularized gradient decent update

- In matrix form,

$$\frac{\partial J_{reg}}{\partial w_i} = X^T(Y - T) + \lambda W$$

L1 regularization

Goal:

- to select a small number of features that predict the trend
- remove features that are noise
- Induce “sparsity” where many weights = 0

L1 regularization continued

- Similar to L2, just add a penalty term

$$J_{RIDGE} = J + \frac{\lambda}{2}\|W\|_2^2$$

$$J_{LASSO} = J + \frac{\lambda}{2}\|W\|$$

L1 regularized gradient decent update

- In matrix form,

$$\frac{\partial J_{REG}}{\partial w_i} = X^T(Y - T) + \lambda * sign(W)$$

$$sign(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$$

λ here is less restrictive in the values it takes

Summary

- Both L1 and L2 prevent overfitting to noise
- L1 selects the most important features and induces sparsity (setting other wts to zero)
- L2 ensures that none of the weights are disproportionately large
- Both L1 and L2 improve generalization on never-been-seen-before data

Elastic Net Regularization

- Elastic net is when both L1 and L2 regularizations are combined simultaneously

$$J_{ELASTICNET} = J + \lambda_1 \|W\| + \lambda_2 \|W\|_2^2$$

Adam Optimizer

- Adam is a popular algorithm in the field of deep learning because it achieves good results fast
- A more efficient alternative to SGD
- Maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems)

Questions