

CSDS 440: Machine Learning

Soumya Ray (he/him, sray@case.edu)

Olin 516

Office hours T, Th 11:15-11:45 or by appointment

[Zoom link](#)

Recap

- A convex set is a set where for any pair of points, the l_____ joining them _____.
- A convex function is a function whose e_____ is convex.
- Convex functions satisfy J_____ i_____.
- For convex functions, every l_____ o_____ is a _____.
- In constrained optimization, the constraints define a f_____ r_____ where _____.
- In a linear program, the feasible region is a p_____ and the solution lies at a v_____.
- From a primal linear program, we can create a d_____ LP.
- The conditions for optimality of a primal-dual convex system are given by the _____ conditions.
- What is “integrate and fire” in an actual neuron?
- In a perceptron, there are i_____ units and an o_____ unit. Each e_____ has a w_____ associated with it. If the d_____ p_____ between i_____ and w_____ is g_____ than the a_____ t_____, the perceptron will fire.
- The perceptron is a p_____ model. The p_____ are the w_____ and the b_____.

Today

- Artificial Neural Networks (Ch 4, Mitchell)

Training Phase

- Given Data:

x_1	x_2	y
0	0	-1
0	1	1

- Find (\mathbf{w}, σ) so that the resulting perceptron matches y

Loss Function

- We will define a function called a “loss function”

$$L(\mathbf{w}, \sigma)$$

- This function will measure the difference between our *current estimates* of y (\hat{y}) and the *true* y (which is known), over all training examples, with respect to (\mathbf{w}, σ)
- Then our goal will be to *minimize* the loss function with respect to (\mathbf{w}, σ)

Training Phase

- Given a training sample and their class labels

$$D = \begin{pmatrix} x_{11} & \cdots & x_{1n} & -1 & y_1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & -1 & y_m \end{pmatrix}$$

- Find parameters

$$\mathbf{W} = (w_1, w_2, w_3, \dots, w_n, \sigma)$$

- To minimize “loss” $L(\mathbf{w})$

\hat{y}

Loss function

- A common loss function is “squared loss”

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^m (y_i - \hat{y}_i)^2 = \frac{1}{2} \sum_{i=1}^m (y_i - \text{sign}(\mathbf{w} \cdot \mathbf{x}_i))^2$$

- Notice that $\text{sign}(1)=1$ and $\text{sign}(-1)=-1$, so if we can get $\mathbf{w} \cdot \mathbf{x}$ to equal y we can drop the sign function
 - This is easier to deal with (differentiable)

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

Iterative parameter update

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$$

- Calculate gradient with respect to parameters \mathbf{w} :

$$\nabla_{\mathbf{w}} L = \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i)(-\mathbf{x}_i)$$

- Parameter Update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L$$

Learning rate
or
step size

Convergence

- This is gradient descent
- Notice that this loss function is everywhere differentiable and bounded below
 - A well defined (unique!) minimum exists for any \mathbf{D}
 - This iterative procedure will find it

Stochastic G.D. vs Regular G.D.

Instead of summing the gradients across all examples, could perform update separately for each example (or a few examples)

Regular

$$\nabla_{\mathbf{w}} L = \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i) (-\mathbf{x}_i)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L$$

Stochastic

$$\nabla_{\mathbf{w}} \tilde{L} = (y_i - \mathbf{w} \cdot \mathbf{x}_i) (-\mathbf{x}_i)$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \tilde{L}$$

$i=1 \dots m$

Stochastic Gradient Descent

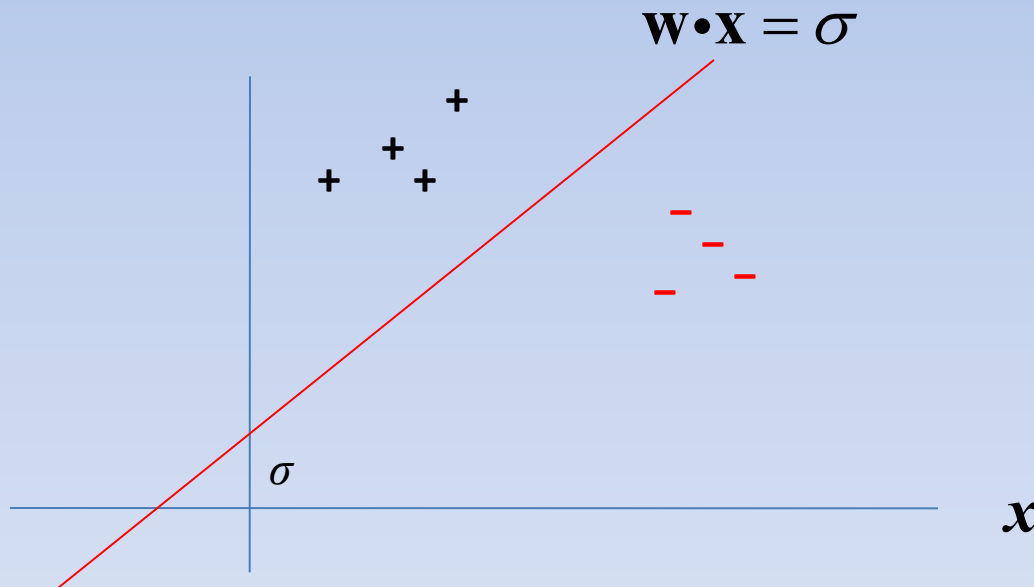
- Since the total gradient is a vector sum, this isn't the same as summing the gradient first
- But it is incremental, useful for online updates and can help with local minima (when it exists)

Stochastic G. D. Convergence

- If doing stochastic gradient descent, technically need η to go to zero and $\sum \eta^2$ to be bounded
- Since we'll usually perform finitely many parameter updates, often ignore this in practice

Geometry of the Perceptron

- A perceptron's separating surface defines a hyperplane in feature space



Linear Separability

- Since the perceptron defines a linear decision boundary, any dataset that it can separate with no error is said to be “linearly separable”
 - Such a dataset will have zero loss wrt our loss function

Logic with a perceptron

- Conjunctions

$$x_1 \wedge x_2 \wedge x_3 \Leftrightarrow y$$

$$1 \bullet x_1 + 1 \bullet x_2 + 1 \bullet x_3 \geq 3$$

- At least *m-of-n*

$$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3) \Leftrightarrow y$$

$$1 \bullet x_1 + 1 \bullet x_2 + 1 \bullet x_3 \geq 2$$

Things that **can't** be learned

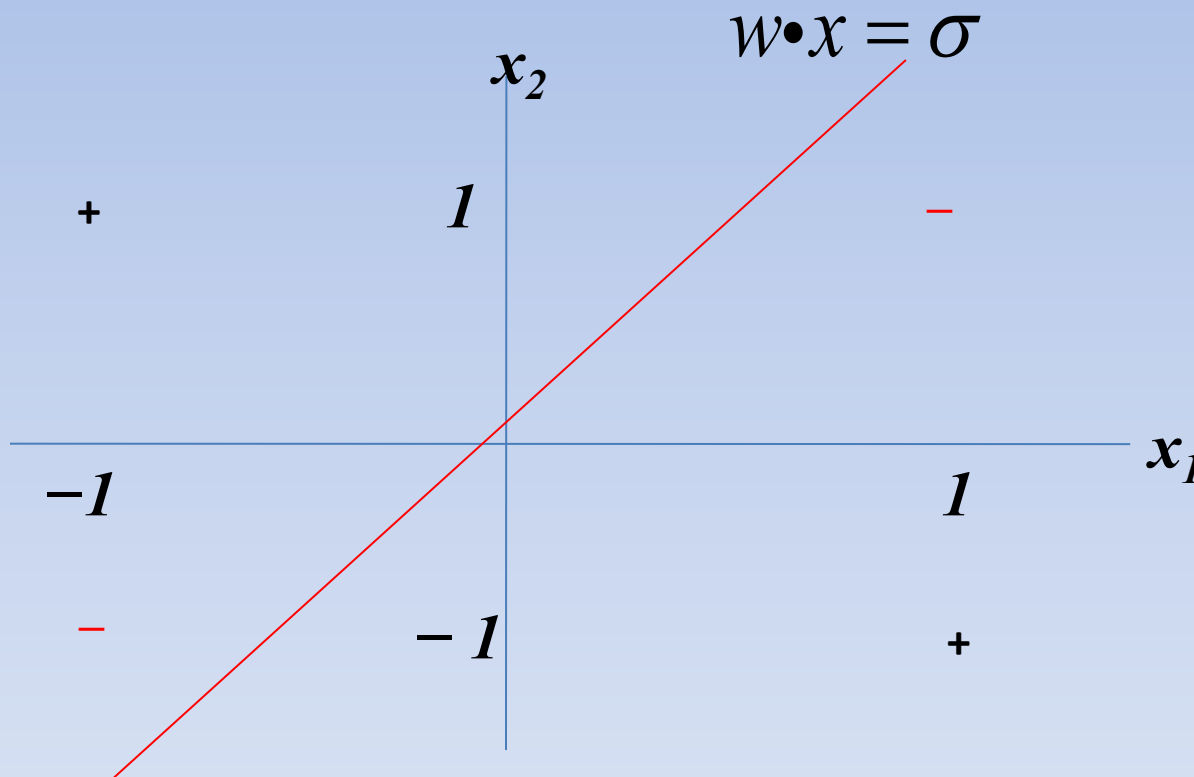
- Complex disjunctions

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow y$$

- Exclusive-OR

$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \Leftrightarrow y$$

XOR and the Perceptron

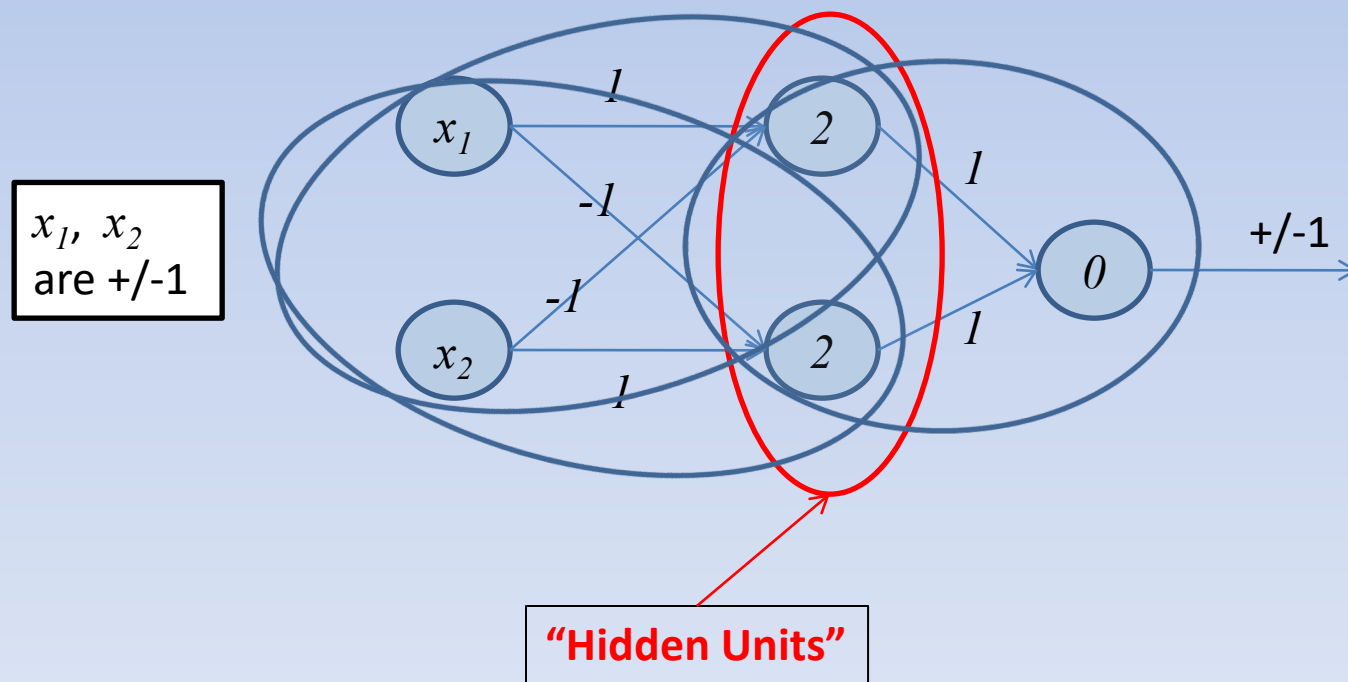


x_1	x_2	f
F	F	F
T	F	T
F	T	T
T	T	F

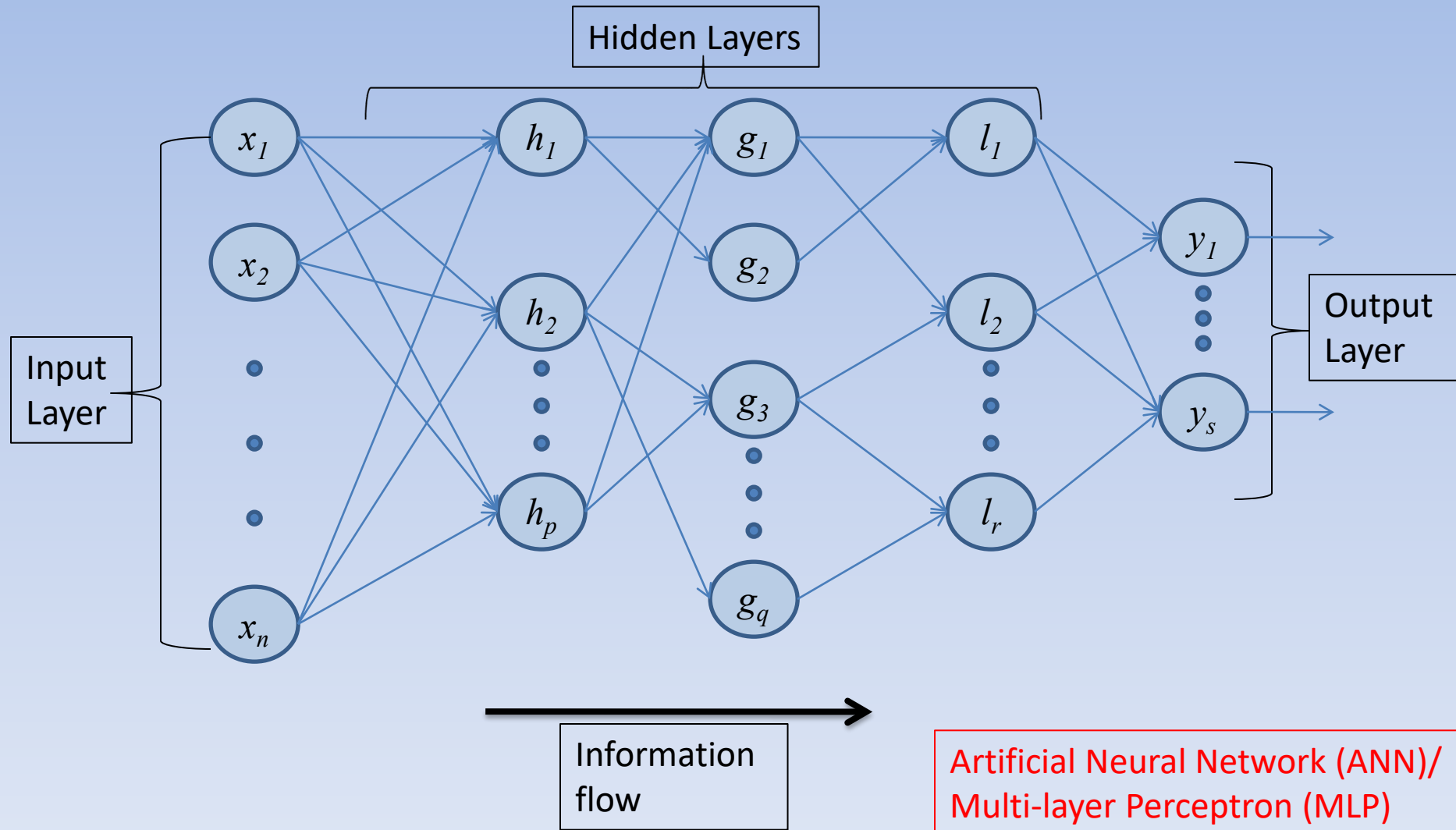
How to fix this?

Idea

- So far, single “neuron”
- Let’s connect them into a network



“Feedforward” Network Topology



Representation Ability

- Every Boolean function can be represented by a network with one hidden layer of units (Minsky and Papert)
- Every bounded continuous function can be represented by a network with one hidden layer (Cybenko et al)*
- Every function on \mathbf{R}^n can be represented by a network with two hidden layers! (Cybenko et al)*
 - *(with arbitrary numbers of hidden units)

Tradeoffs

- Very large number of degrees of freedom
 - Network topology
 - Number of layers, number of hidden units in each layer, edge configuration, even different activation functions
 - Network parameter values
- Power comes at a price
 - Very very easy to overfit if structure is complex
 - Very long time and large samples required to train
 - Structure and function relationship is opaque to people (and often uninterpretable even when not)

Training Phase (ANN)

- As before, given a training sample and their class labels

$$D = \begin{pmatrix} x_{11} & \cdots & x_{1n} & -1 & y_1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & -1 & y_m \end{pmatrix}$$

- Find parameters \mathbf{W}
- To minimize “loss” $L(\mathbf{w})$

Hidden Layer Activation Function

- For a perceptron, we used a *sign()* function as an activation function
- This function is problematic for optimization because it isn't differentiable
 - For perceptron this was OK, we were able to remove the *sign()* and handle this
 - But for a general ANN we need a smooth activation function

Activation Functions

- Sigmoid

$$h(\mathbf{x}; \mathbf{w}) = (1 + e^{-\mathbf{w} \cdot \mathbf{x}})^{-1}$$

- Radial Basis Function

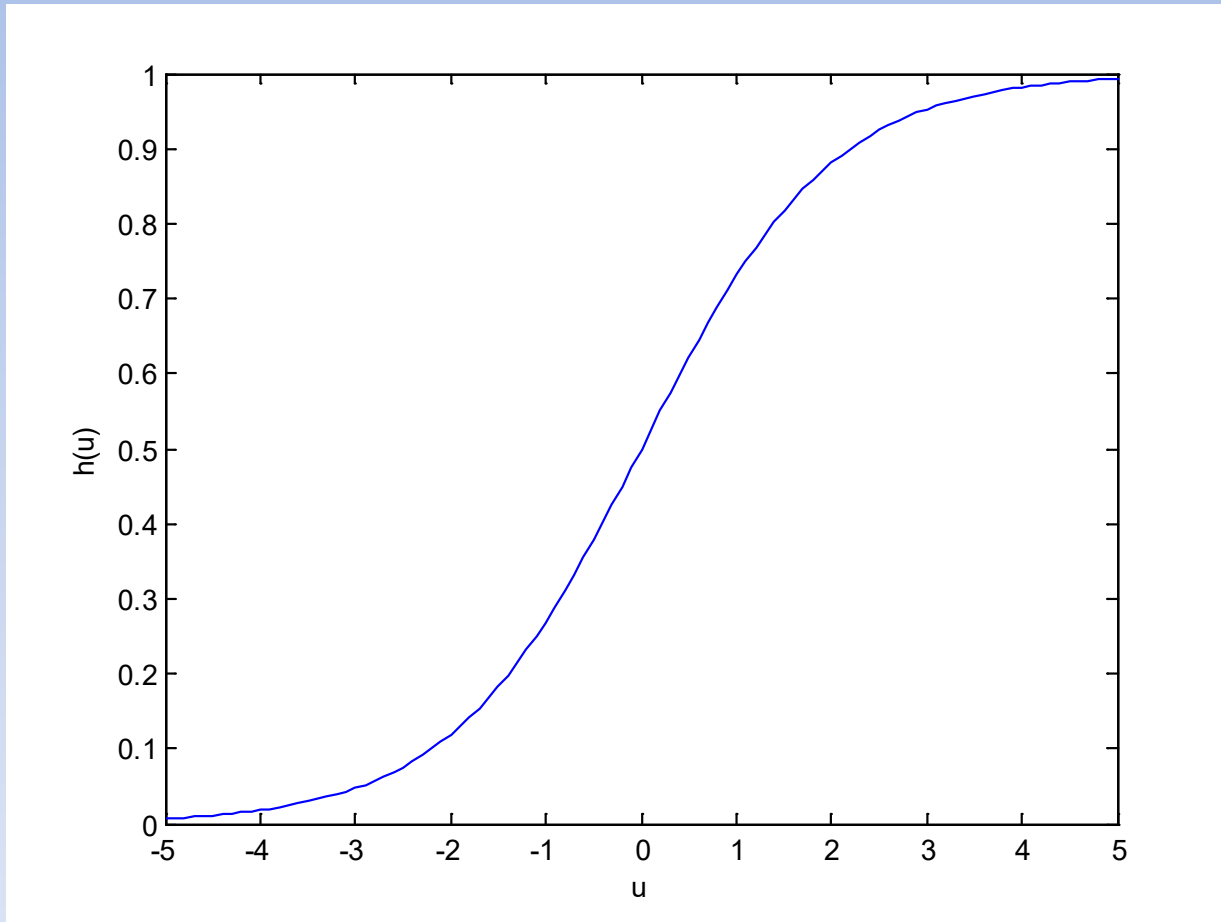
$$h(\mathbf{x}; \mathbf{w}, \mathbf{c}, \beta) = e^{\beta \|\mathbf{w} \cdot \mathbf{x} - \mathbf{c}\|^2}$$

- Hyperbolic Tangent

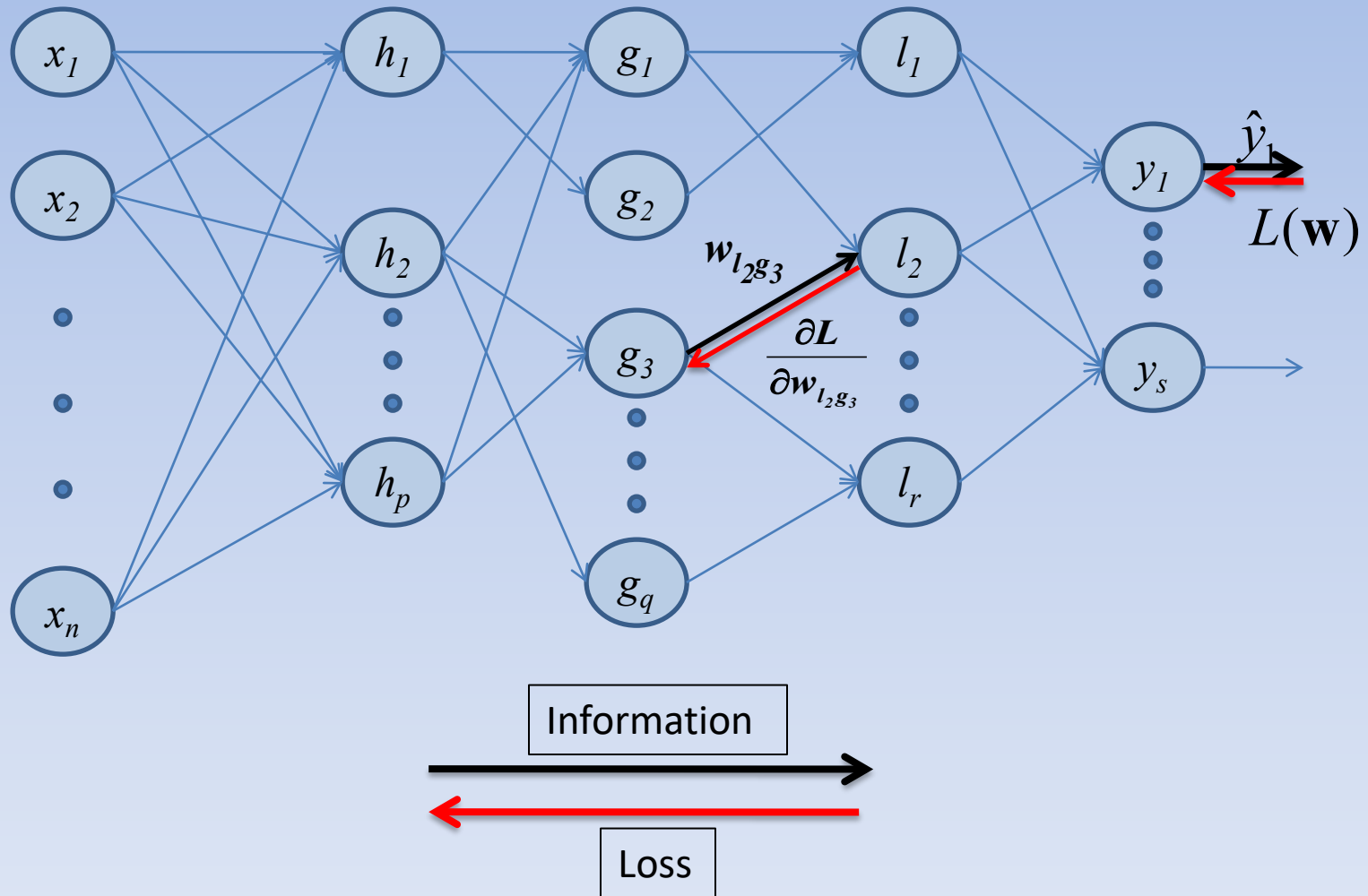
$$h(\mathbf{x}; \mathbf{w}, a, b) = a \frac{(e^{b\mathbf{w} \cdot \mathbf{x}} - e^{-b\mathbf{w} \cdot \mathbf{x}})}{(e^{b\mathbf{w} \cdot \mathbf{x}} + e^{-b\mathbf{w} \cdot \mathbf{x}})}$$

Activation Function: Sigmoid

$$h(u) = (1 + e^{-u})^{-1}$$



Backpropagation

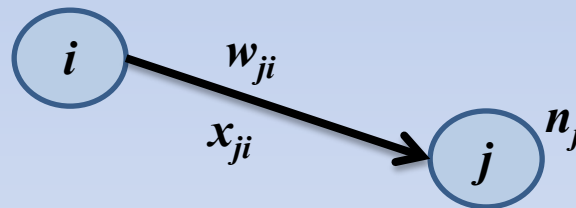


Backpropagation

- Feed the examples forward through the network, observe the output and calculate the loss
- Perform “layer-wise” gradient descent on the loss function with respect to each weight, starting with output layer
 - For each weight in each layer, calculate its contribution to the overall loss using the chain rule
 - Update the weight in the negative gradient direction

Backpropagation (SGD)

- Let x_{ji} be the i^{th} input to unit j
- Let w_{ji} be the parameter associated with x_{ji}
- Let $n_j = \sum_i w_{ji} x_{ji}$ be the “net input” to unit j



- Observe that
$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial n_j} \frac{\partial n_j}{\partial w_{ji}} = \frac{\partial L}{\partial n_j} x_{ji}$$