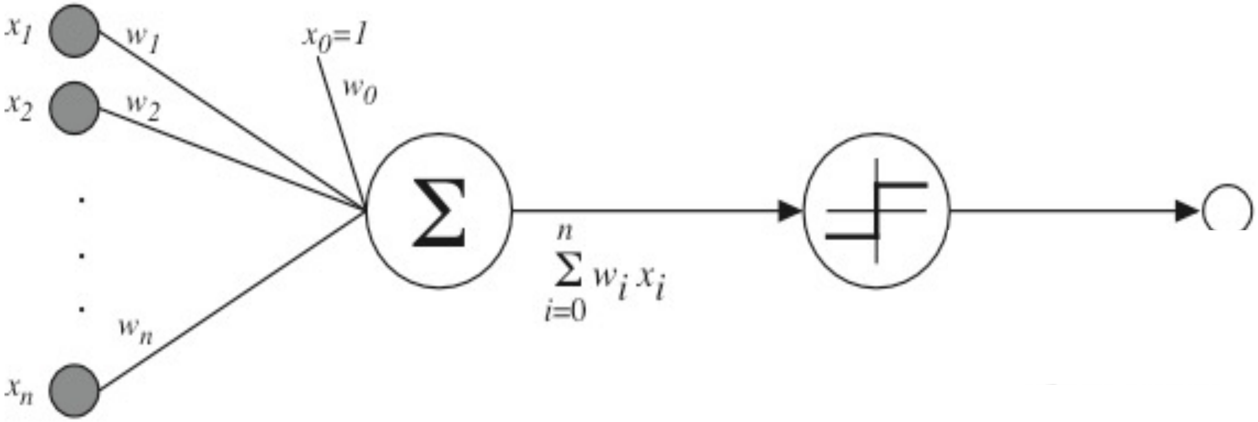
COMP 451 – Fundamentals of Machine Learning Lecture 21 – Neural Networks

William L. Hamilton

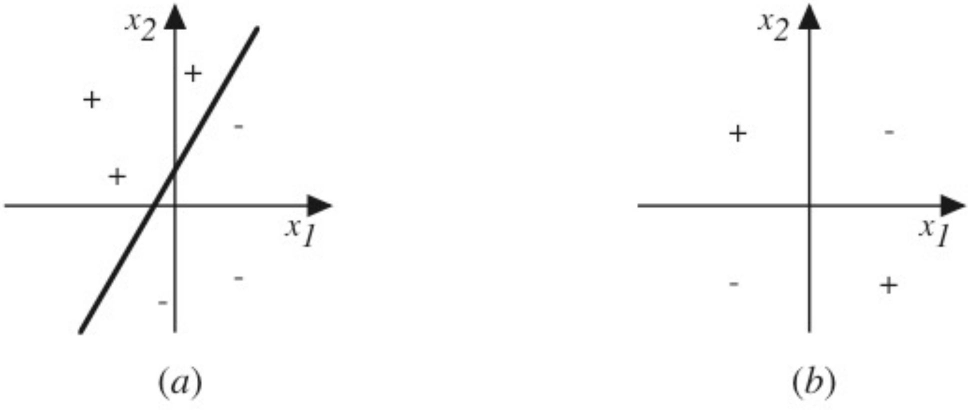
Recall the perceptron



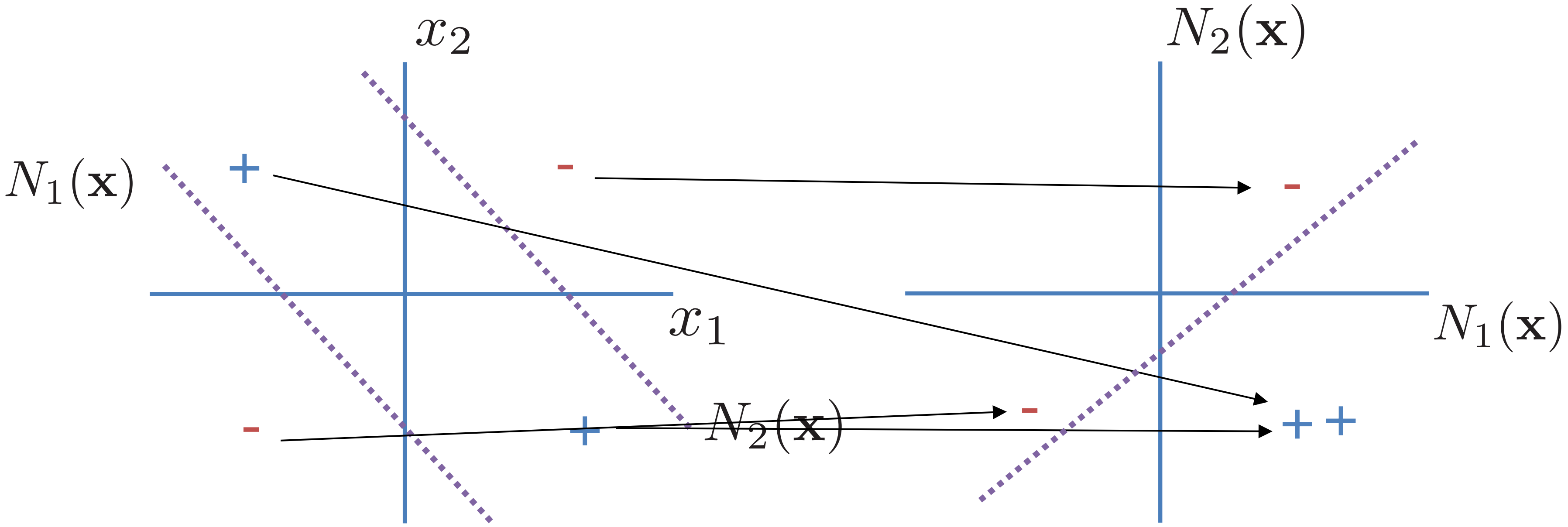
$$h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}(\mathbf{x} \cdot \mathbf{w}) = \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Decision surface of a perceptron

- Single perceptron can represent linear boundaries.
- To represent non-linearly separate functions (e.g. XOR), we could use a network of stacked perceptron-like elements.
- If we connect perceptrons into networks, the error surface for the network is not differentiable (because of the hard threshold).



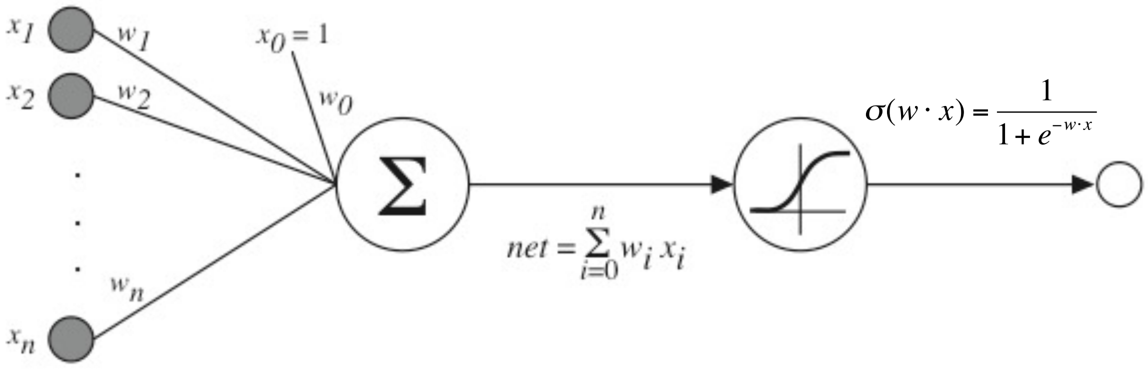
Example: A network representing XOR



1) Run two perceptrons (N_1 and N_2) on the original dataset and get the decision boundaries above

2) New dataset defined by the output of N_1 and N_2 is linearly separable!

Recall the sigmoid function



Sigmoid provide “soft threshold”, whereas perceptron provides “hard threshold”

- It has the following nice property: $\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$

We can derive a **gradient descent rule** to train:

- One sigmoid unit -> multi-layer networks of sigmoid units.

Feed-forward neural networks

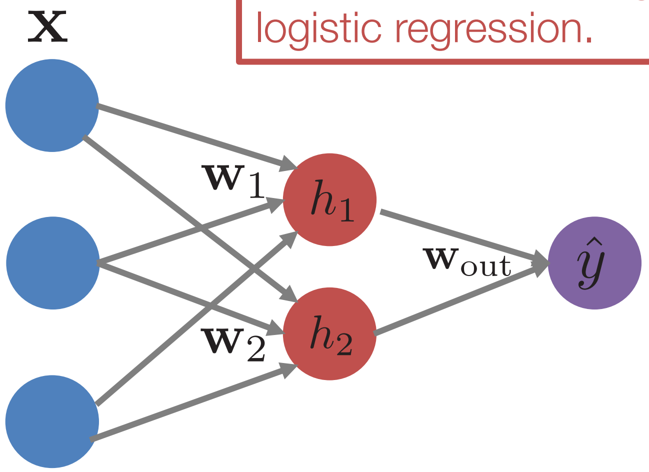
- We are stacking simple models with sigmoid output functions.
  - (I.e., basically stacking logistic regression models)

- "Hidden" units are the output of the sigmoid/logistic models in the stack.

- Note that unlike a Boltzmann machine, the connections are directed and information only flows in one direction!

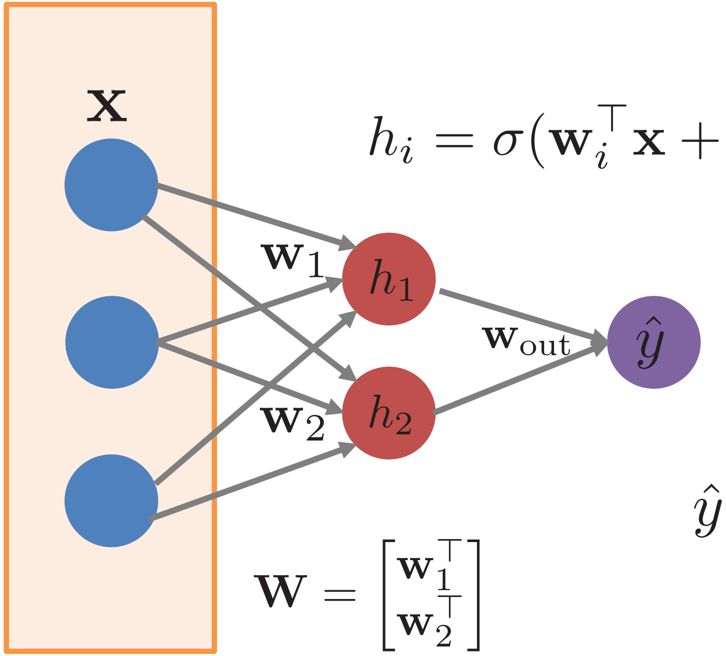
$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$

Hidden units are linear + sigmoid activation, i.e., analogous to logistic regression.



Feed-forward neural networks

Input data (or input units)



$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$

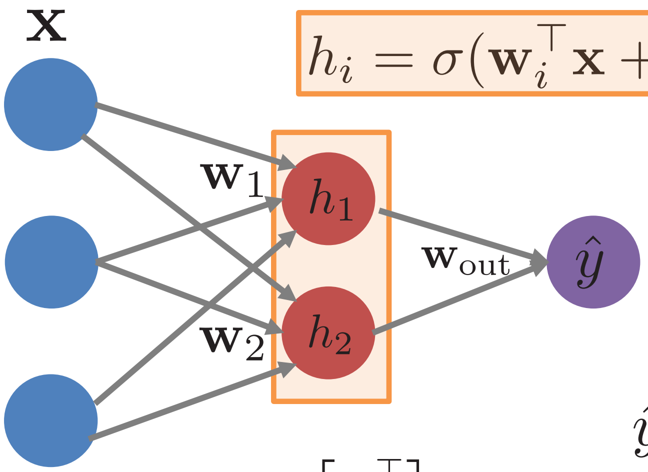
$$\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}} \mathbf{h} + b_{\text{out}})$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Feed-forward neural networks

Hidden units are linear function + sigmoid applied to input.



$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$

$$\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}}\mathbf{h} + b_{\text{out}})$$

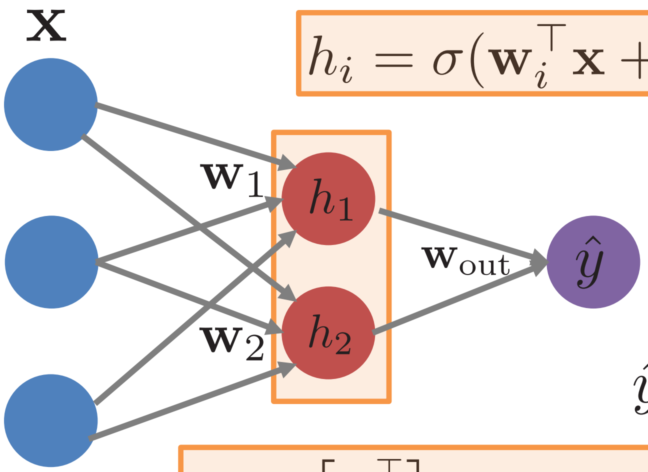
$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Feed-forward neural networks

Matrix notation: We can combine the hidden units together into a vector and their weights into a matrix

$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$



$$\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}}\mathbf{h} + b_{\text{out}})$$

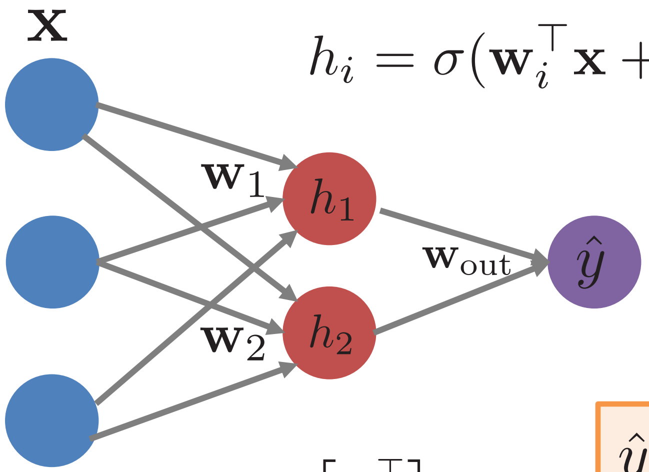
$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Feed-forward neural networks

Output unit: Linear function of the hidden units followed by an "activation function", ϕ_{out}.

$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

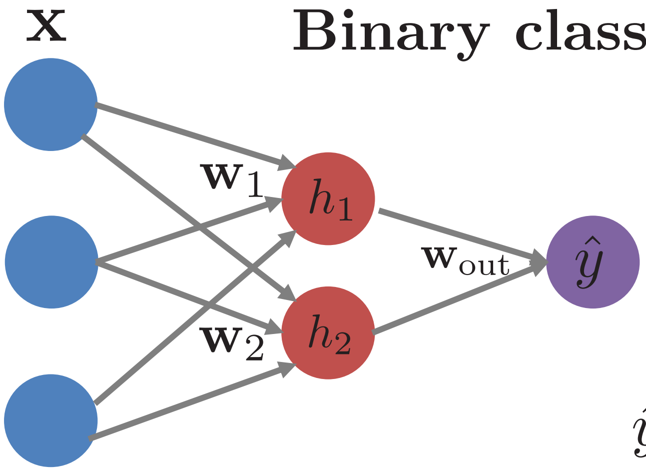
$$\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}}\mathbf{h} + b_{\text{out}})$$

Feed-forward neural networks

$$\textbf{Regression}: \phi_{\text{out}}(z) = z$$

$$\textbf{Binary classification}: \phi_{\text{out}}(z) = \sigma(z)$$

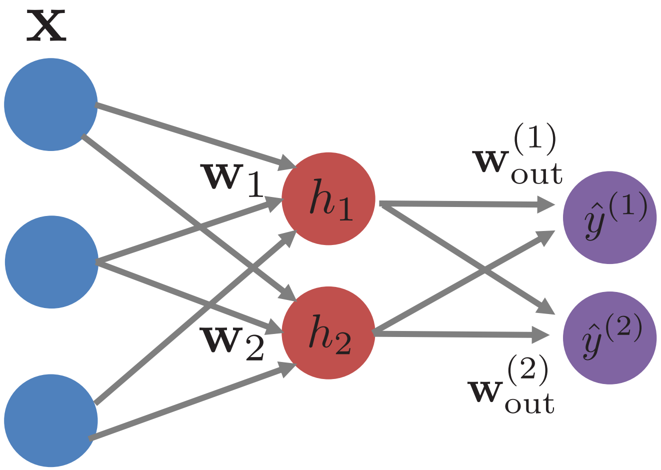
The activation function on the output depends on the task (e.g., regression or classification)



$$\hat{y} = \phi_{\text{out}}(\mathbf{w}_{\text{out}}\mathbf{h} + b_{\text{out}})$$

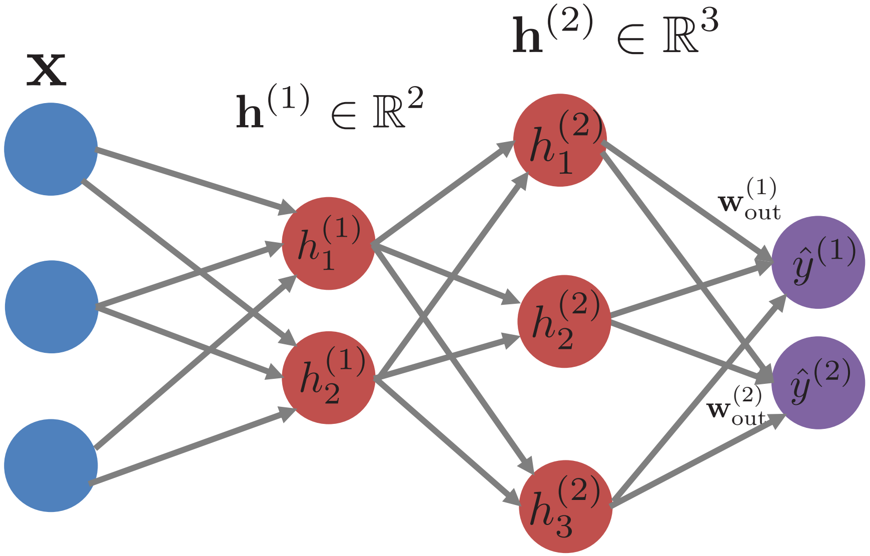
Feed-forward neural networks

- It is possible to have multiple output units.
- E.g., for multi-label classification.



Feed-forward neural networks

- It is possible to stack more than one hidden layer.
- This is known as the "depth" of the network.

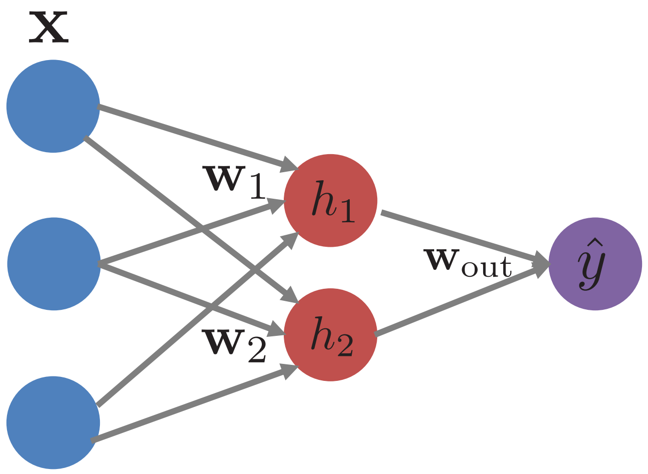


$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \rightarrow \mathbf{h}^{(2)} = \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \rightarrow \hat{\mathbf{y}} = \phi_{out}(\mathbf{W}_{out}\mathbf{h}^{(2)} + \mathbf{b}_{out})$$

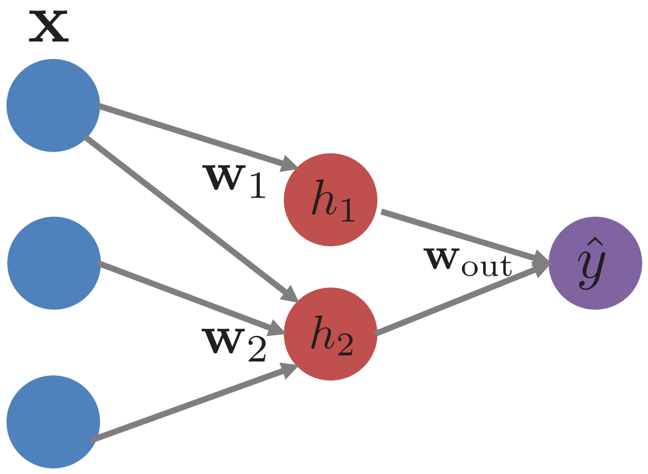
Why this name?

- In <u>feed-forward networks</u> the output of units in layer j become input to the units in layers $j+1$.
- No cross-connection between units in the same layer.
- No backward connections from layers downstream
- In <u>fully-connected networks</u>, all units in layer j provide input to all units in layer $j+1$.

Fully-connected networks



Fully-connected network



Network with missing connections

$$\mathbf{w}_1 = [w_{1,1}, 0, 0]$$

Fully connected networks are far more common!

Feed-forward neural networks

- In general, we have an **input layer, H hidden layers, and an output layer.**
- Computing the output is called running the "forward pass":

$$\mathbf{h}^0 = \mathbf{x}$$

Initialize

for i=1...H:

$$\mathbf{h}^{(i)} = \sigma(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)})$$

Compute each hidden layer sequentially

$$\hat{\mathbf{y}} = \phi_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{h}^{(H)} + \mathbf{b}_{\text{out}})$$

Compute the output

Learning in feed-forward neural networks

- Assume the network structure (units + connections) is given.

- The learning problem is finding a good set of weights to minimize the error at the output of the network.

- Approach: gradient descent, because the form of the hypothesis formed by the network is:
  - Differentiable! Because of the choice of sigmoid units.
  - Very complex! Hence direct computation of the optimal weights is not possible.

Gradient-descent preliminaries for NN

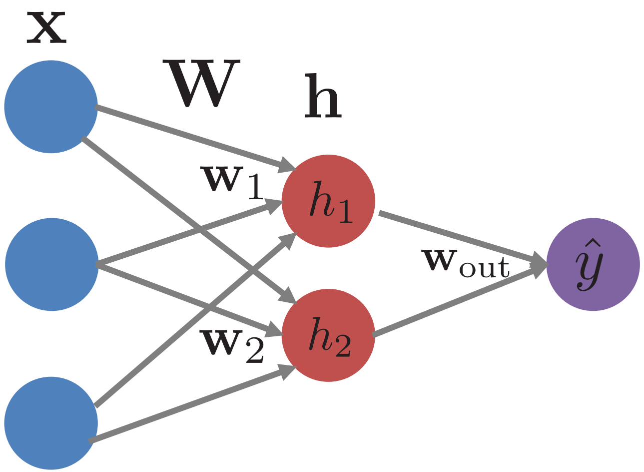
- Take regression as a simple case (i.e., the y values are one-dimensional and real-valued).
- Assume we have a fully-connected network with one hidden layer.
- We want to compute the weight update after seeing **a single training example** $<\mathbf{x}, y>$.
- We are using the squared loss: $J(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2$

Gradient-descent update for the output node

$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

Apply the chain rule

<u>Basic Neural Net</u>



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the output node

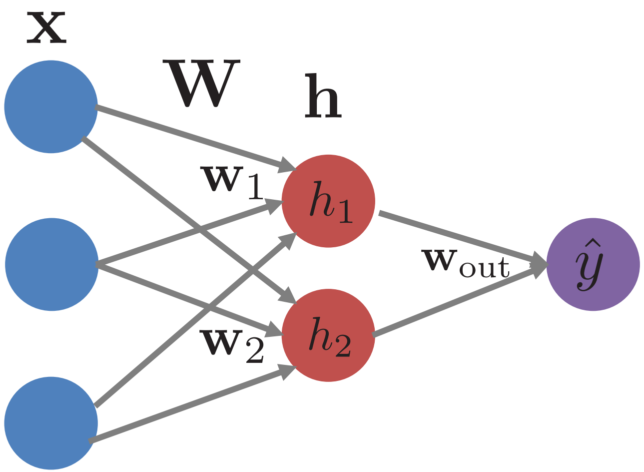
$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

$$= (\hat{y} - y) \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

Recall that:

$$J(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2$$

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the output node

$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

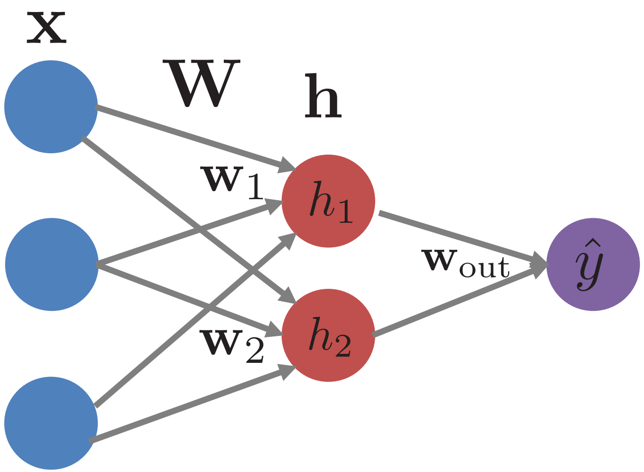
$$= (\hat{y} - y) \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

$$= (\hat{y} - y) \frac{\partial \left(\mathbf{w}_{\text{out}} \mathbf{h} + b_{\text{out}} \right)}{\partial \mathbf{w}_{\text{out}}}$$

Recall that:

$$\hat{y} = \mathbf{w}_{\text{out}}^{\top} \mathbf{h} + b_{\text{out}}$$

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^{\top} \\ \mathbf{w}_2^{\top} \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^{\top} \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the output node

$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

$$= (\hat{y} - y) \frac{\partial \hat{y}}{\partial \mathbf{w}_{\text{out}}}$$

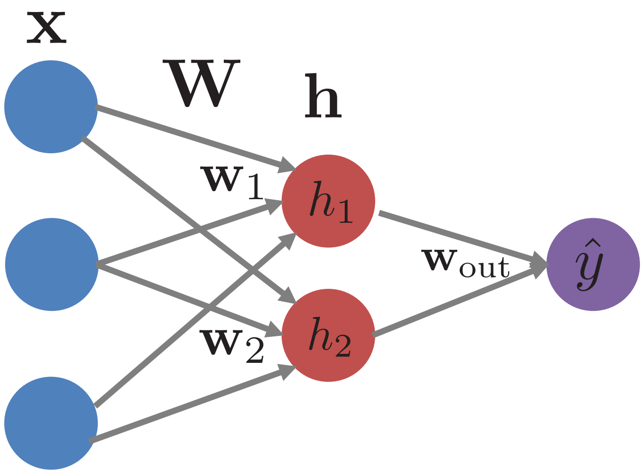
$$= (\hat{y} - y) \frac{\partial \left(\mathbf{w}_{\text{out}} \mathbf{h} + b_{\text{out}}\right)}{\partial \mathbf{w}_{\text{out}}}$$

$$= (\hat{y} - y)\mathbf{h}$$

$$= \delta_{out} \mathbf{h}$$

We can think of this of this as the "error signal" at the output node.

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

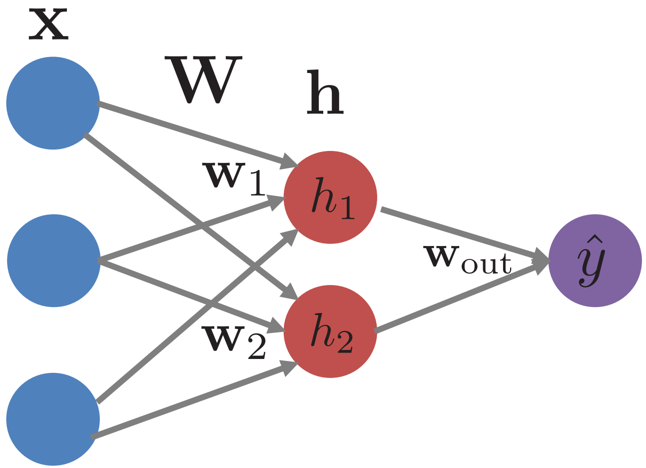
$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i}$$

We want to determine the derivative of the error w.r.t. to the weights of the hidden node.

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

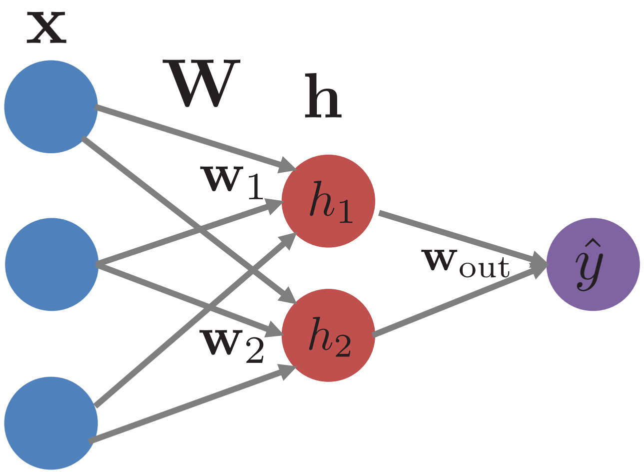
$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

Again, apply the chain rule

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

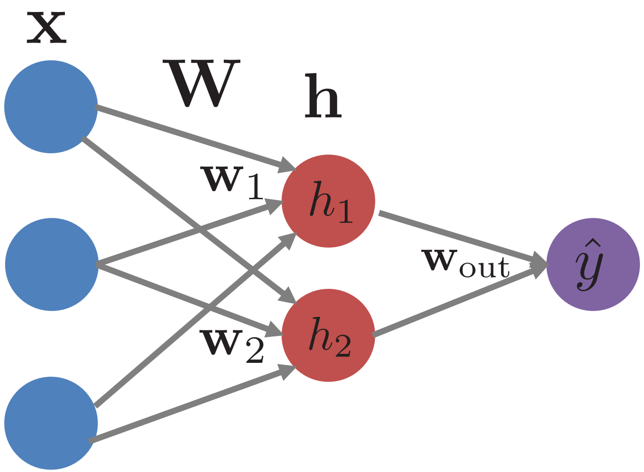
$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

We already compute the error at the output node, so we can just substitute this in.

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

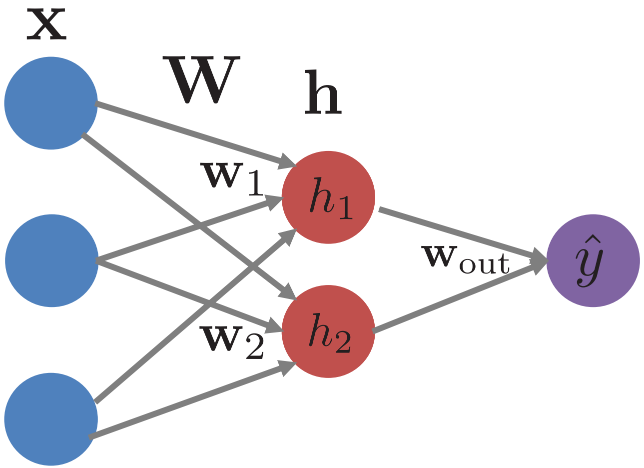
$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial h_j} \frac{\partial h_j}{\partial \mathbf{w}_j}$$

Recall that:

$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$

And again, apply the chain rule….

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

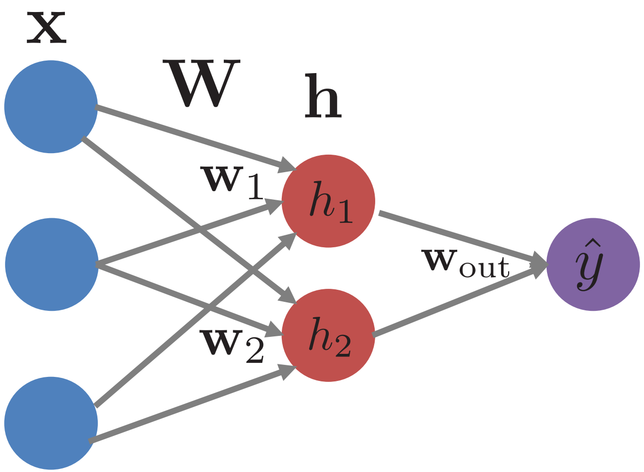
$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial h_j} \frac{\partial h_j}{\partial \mathbf{w}_j}$$
$$= \delta_{out} w_{\text{out},j} \frac{\partial h_j}{\partial \mathbf{w}_j}$$

Recall that

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

and note that the j'th hidden node only interacts with the j'th value in $\mathbf{w}_{\text{out}}$

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$

$$= \delta_{\text{out}} \frac{\partial \hat{y}}{\partial h_j} \frac{\partial h_j}{\partial \mathbf{w}_j}$$

$$= \delta_{out} w_{\text{out},j} \frac{\partial h_j}{\partial \mathbf{w}_j}$$

$$= \delta_{out} w_{\text{out},j} \sigma(\mathbf{w}_j^\top \mathbf{x} + b)(1 - \sigma(\mathbf{w}_j^\top \mathbf{x} + b))\mathbf{x}$$

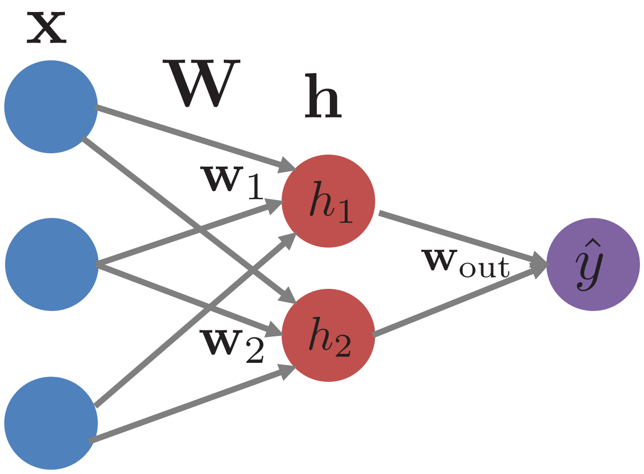
Recall that

$$h_i = \sigma(\mathbf{w}_i^\top \mathbf{x} + b_i), \forall i$$

and the identity

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

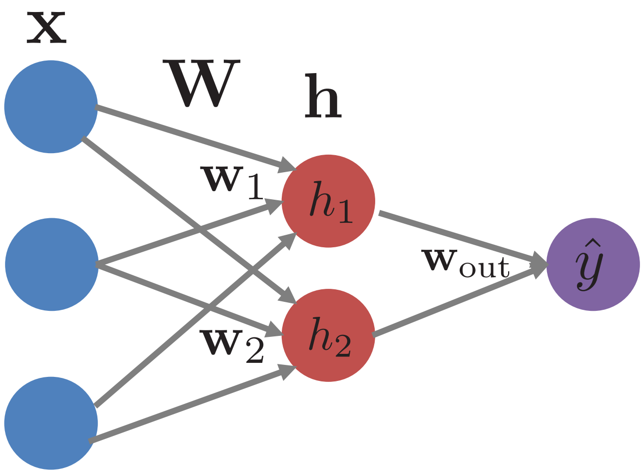
$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}}\frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}}\frac{\partial \hat{y}}{\partial h_j}\frac{\partial h_j}{\partial \mathbf{w}_j}$$
$$= \delta_{out} w_{\text{out},j}\frac{\partial h_j}{\partial \mathbf{w}_j}$$
$$= \delta_{out} w_{\text{out},j}\sigma(\mathbf{w}_j^\top \mathbf{x} + b)(1 - \sigma(\mathbf{w}_j^\top \mathbf{x} + b))\mathbf{x}$$
$$= \delta_{h_j}\mathbf{x}$$

We can think of this of this a the "error signal" at the hidden node.

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

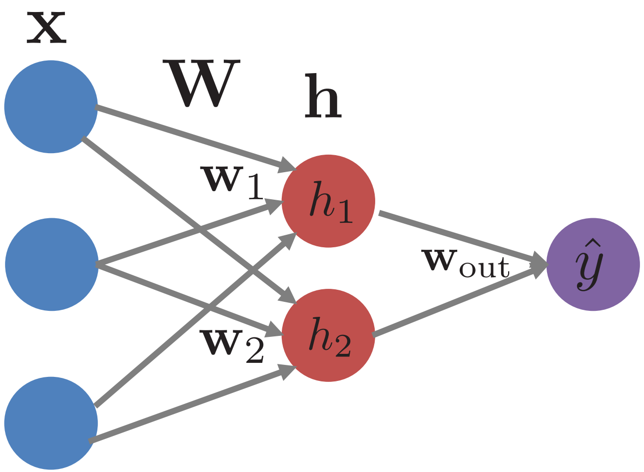
Gradient-descent update for the hidden node

$$\frac{\partial J}{\partial \mathbf{w}_i} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}}\frac{\partial \hat{y}}{\partial \mathbf{w}_j}$$
$$= \delta_{\text{out}}\frac{\partial \hat{y}}{\partial h_j}\frac{\partial h_j}{\partial \mathbf{w}_j}$$
$$= \delta_{out} w_{\text{out},j}\frac{\partial h_j}{\partial \mathbf{w}_j}$$
$$= \delta_{out} w_{\text{out},j}\sigma(\mathbf{w}_j^\top \mathbf{x} + b)(1 - \sigma(\mathbf{w}_j^\top \mathbf{x} + b))\mathbf{x}$$
$$= \delta_{h_j}\mathbf{x}$$

The error at the hidden node is a function of the error at the output, and we are "propagating" this error backwards through the network.

We can think of this of this a the "error signal" at the hidden node.

Basic Neural Net



$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \end{bmatrix}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\hat{y} = \mathbf{w}_{\text{out}}^\top \mathbf{h} + b_{\text{out}}$$

Stochastic gradient descent

- Initialize all weights to small random numbers.
- Repeat until convergence:
  - Pick a training example, $\mathbf{x}$.
  - Feed example through network to compute output y.
  - For the output unit, compute the correction:

$$\frac{\partial J}{\partial \mathbf{w}_{\text{out}}} = \delta_{\text{out}} \mathbf{x}$$

  - For each hidden unit *j*, compute its share of the correction:

$$\frac{\partial J}{\partial \mathbf{w}_j} = \delta_{\text{out}} w_{out,j} \sigma(\mathbf{w}_j^\top \mathbf{x} + b)(1 - \sigma(\mathbf{w}_j^\top \mathbf{x} + b))\mathbf{x}$$

  - Update each network weight:

$$\mathbf{w}_j = \mathbf{w}_j - \alpha \frac{\partial J}{\partial \mathbf{w}_j} \ \forall j, \qquad \mathbf{w}_{out} = \mathbf{w}_{out} - \alpha \frac{\partial J}{\partial \mathbf{w}_{out}}$$

Initialization

Forward pass

Backpropagation

Gradient descent

Organizing the training data

- **Stochastic gradient descent**: Compute error on a single example at a time (as in previous slide).

- **Batch gradient descent**: Compute error on all examples.
  - Loop through the training data, accumulating weight changes.
  - Update all weights and repeat.

- **Mini-batch gradient descent**: Compute error on small subset.
  - Randomly select a "mini-batch" (i.e. subset of training examples).
  - Calculate error on mini-batch, apply to update weights, and repeat.

Expressiveness of feed-forward NN

A neural network with no hidden layers?

- Same representational power as logistic/linear regression or a perceptron; Boolean AND, OR, NOT, but not XOR.

Expressiveness of feed-forward NN

A neural network with no hidden layers?

- Same representational power as logistic/linear regression or a perceptron; Boolean AND, OR, NOT, but not XOR.

A neural network with a single hidden layer?

- Can represent every boolean function, but might require a number of hidden units that is exponential in the number of inputs.
- Every bounded continuous function can be approximated with arbitrary precision by a boolean function.

Expressiveness of feed-forward NN

A neural network with no hidden layers?

- Same representational power as logistic/linear regression or a perceptron; Boolean AND, OR, NOT, but not XOR.

A neural network with a single hidden layer?

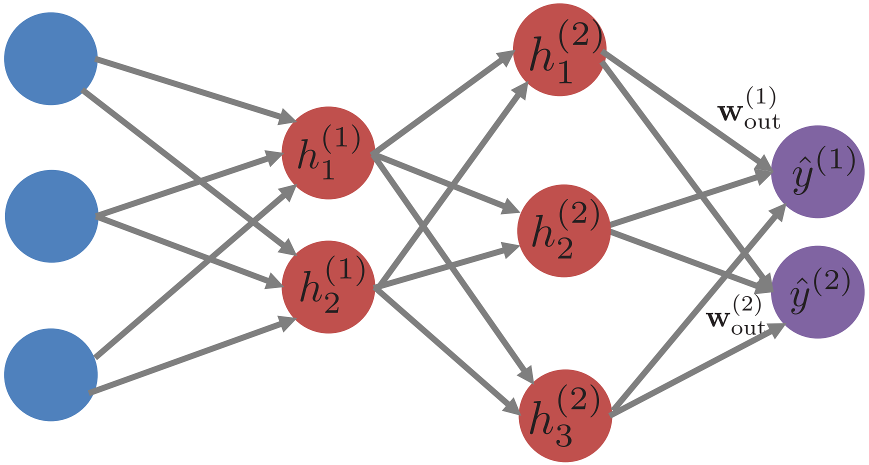
- Can represent every boolean function, but might require a number of hidden units that is exponential in the number of inputs.
- Every bounded continuous function can be approximated with arbitrary precision by a boolean function.

A neural network with two hidden layers?

- Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Generalizing the feed-forward NN

- Can use arbitrary output activation functions.
- In practice, we do not necessarily need to use a sigmoid activation in the hidden layer.
- We can make networks as deep as we want.
- We can add regularization.
- But how to compute these nasty derivatives..? (Next lecture!)



$$\mathbf{h}^{(i)} = \phi_i(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)})$$

Can be an arbitrary **non-linear** activation function