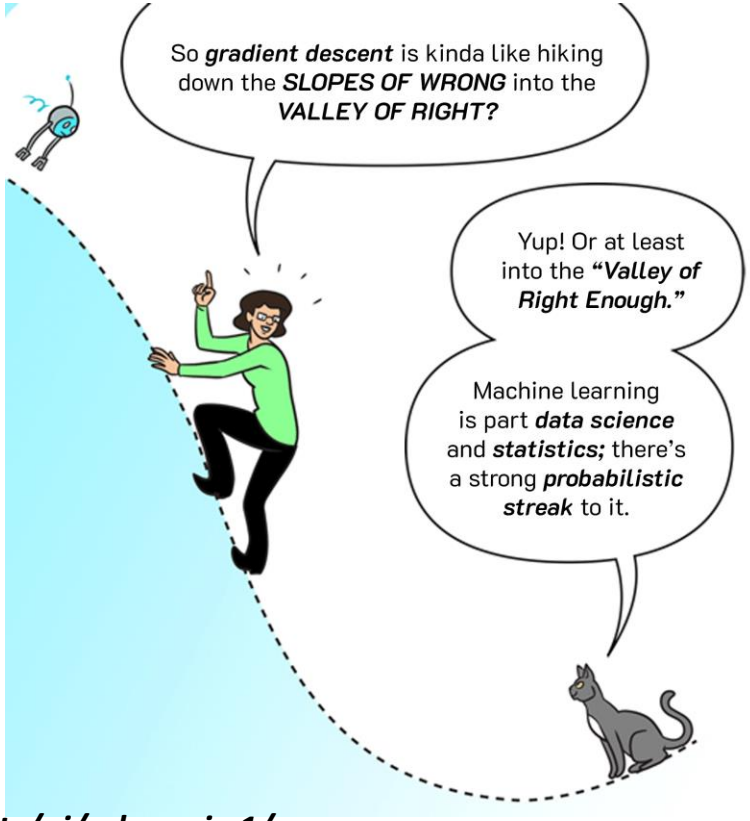
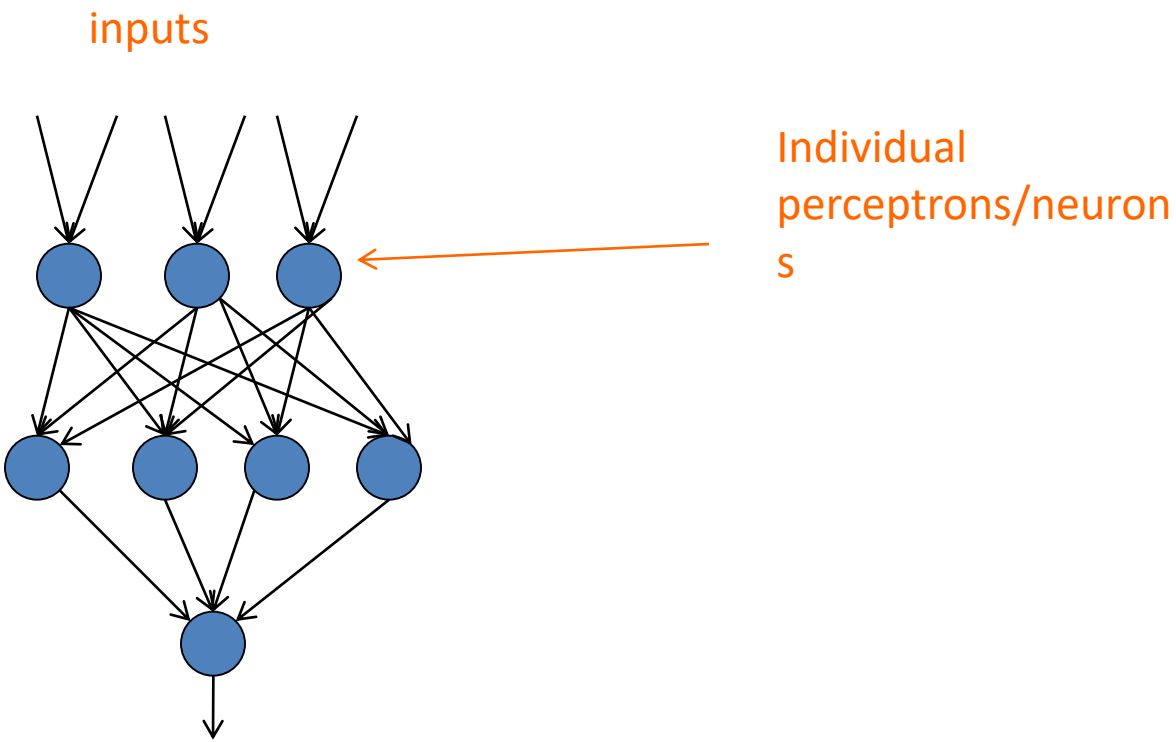
Backpropagation

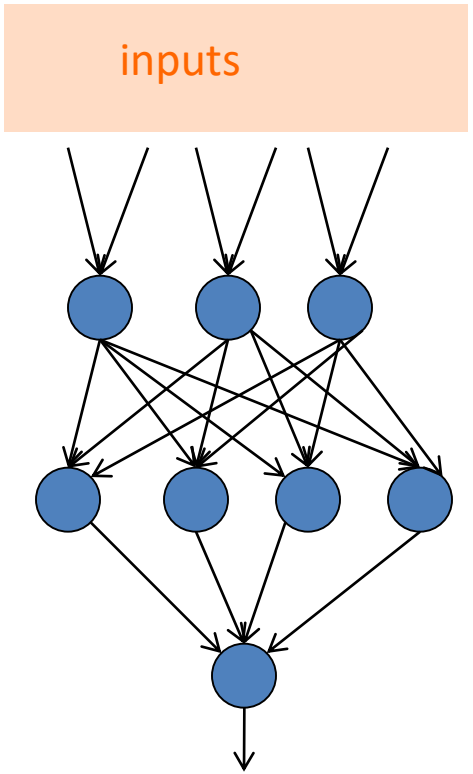


https://cloud.google.com/products/ai/ml-comic-1/

Neural network



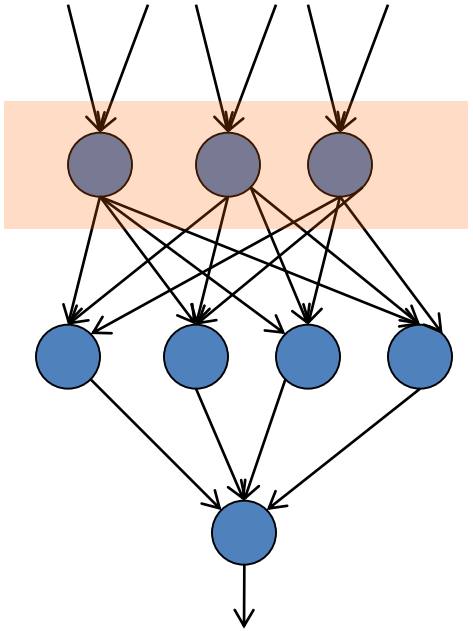
Neural network



some inputs are provided/entered

Neural network

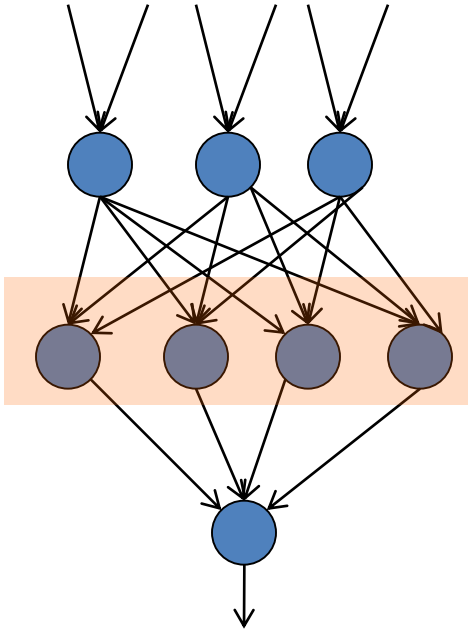
inputs



each perceptron computes
and calculates an answer

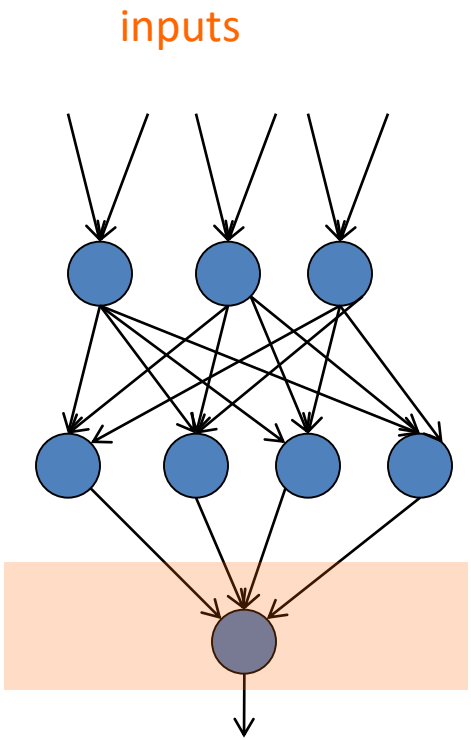
Neural network

inputs



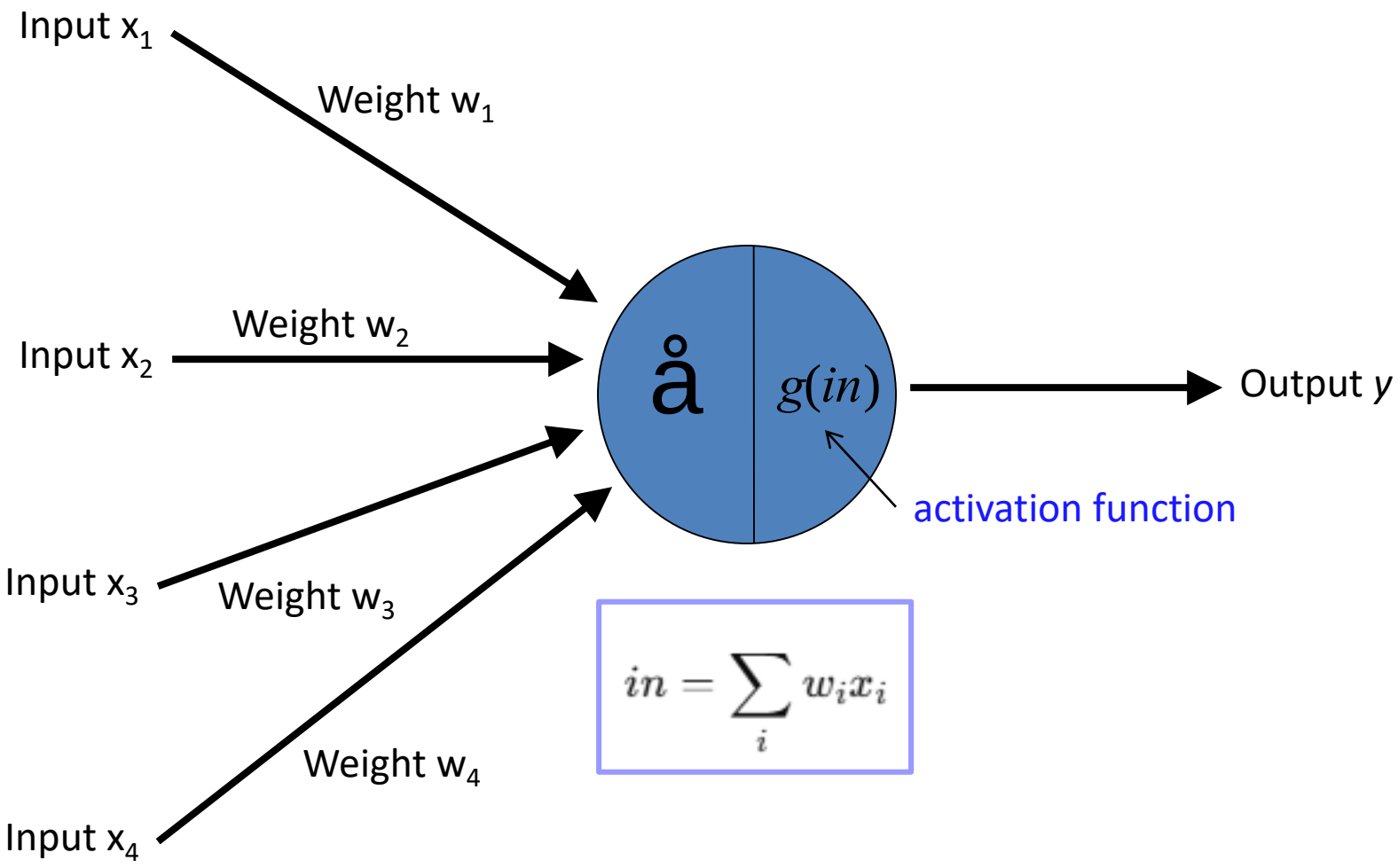
those answers become inputs
for the next level

Neural network



finally get the answer after all levels compute

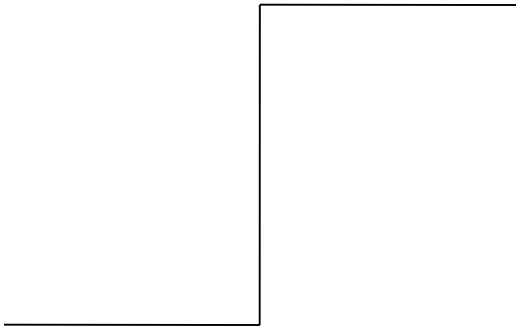
A neuron/perceptron



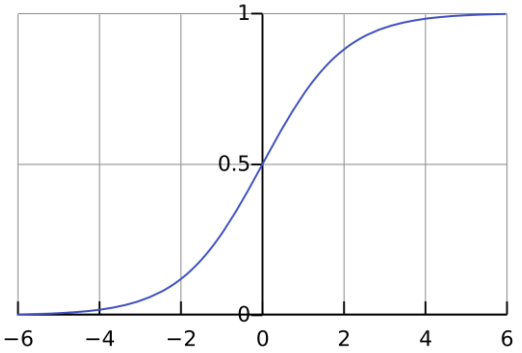
Activation functions

hard threshold:

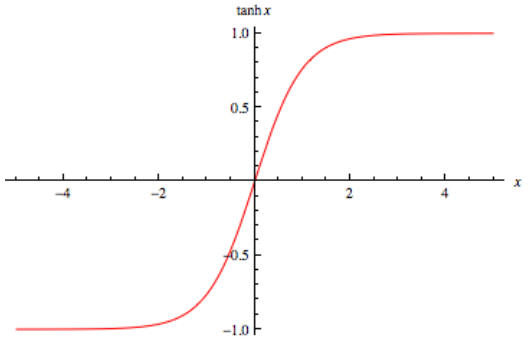
$$g(in) = \begin{cases} 1 & \text{if } in > -b \\ 0 & \text{otherwise} \end{cases}$$



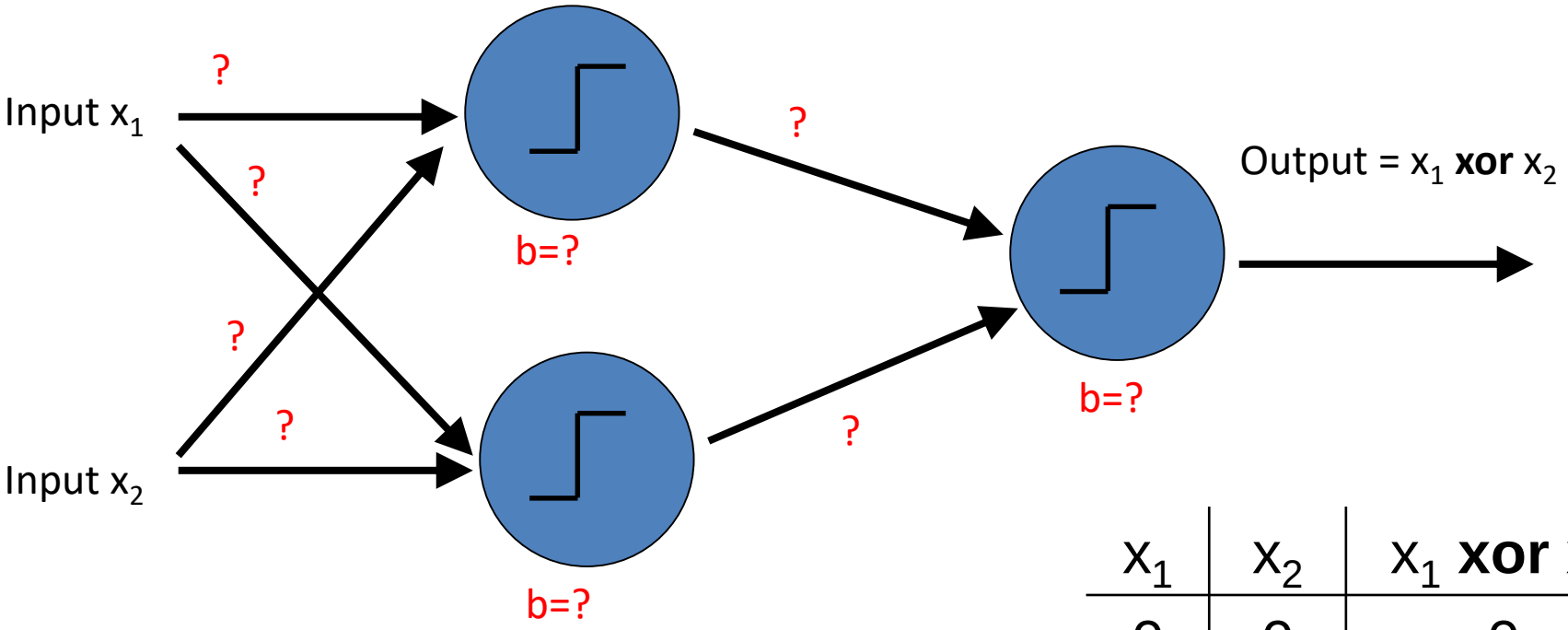
sigmoid $g(x) = \frac{1}{1+e^{-x}}$



tanh x



Training

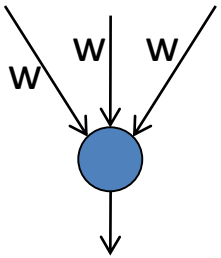


How do we learn the weights?

x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

Learning in multilayer networks

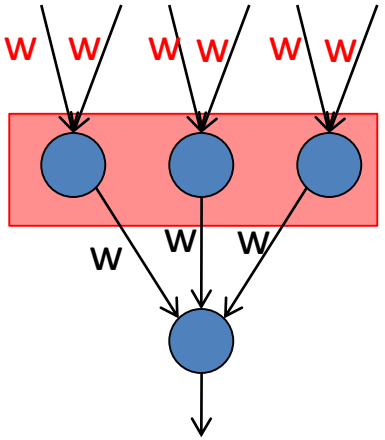
Challenge: for multilayer networks, we don't know what the expected output/error is for the internal nodes!



perceptron/
linear model

how do we learn these weights?

expected output?



neural network

Backpropagation: intuition

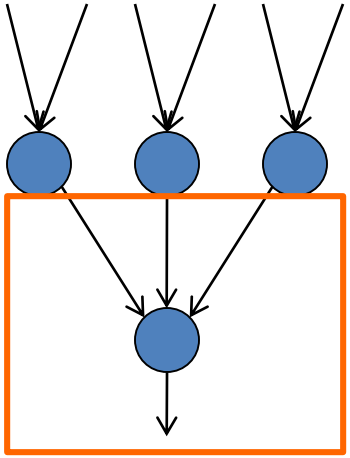
Gradient descent method for learning weights by optimizing a loss function

1. calculate output of all nodes

2. calculate the weights for the output layer based on the error

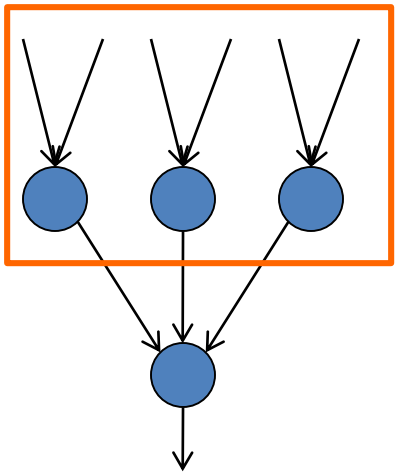
3. “backpropagate” errors through hidden layers

Backpropagation: intuition



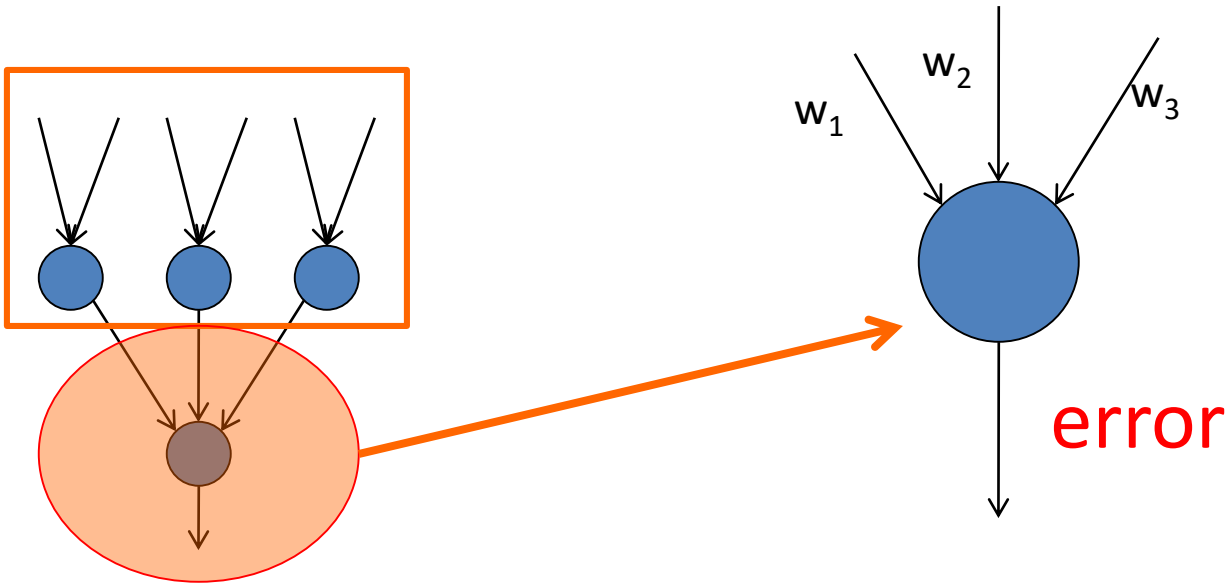
We can calculate the actual error here

Backpropagation: intuition



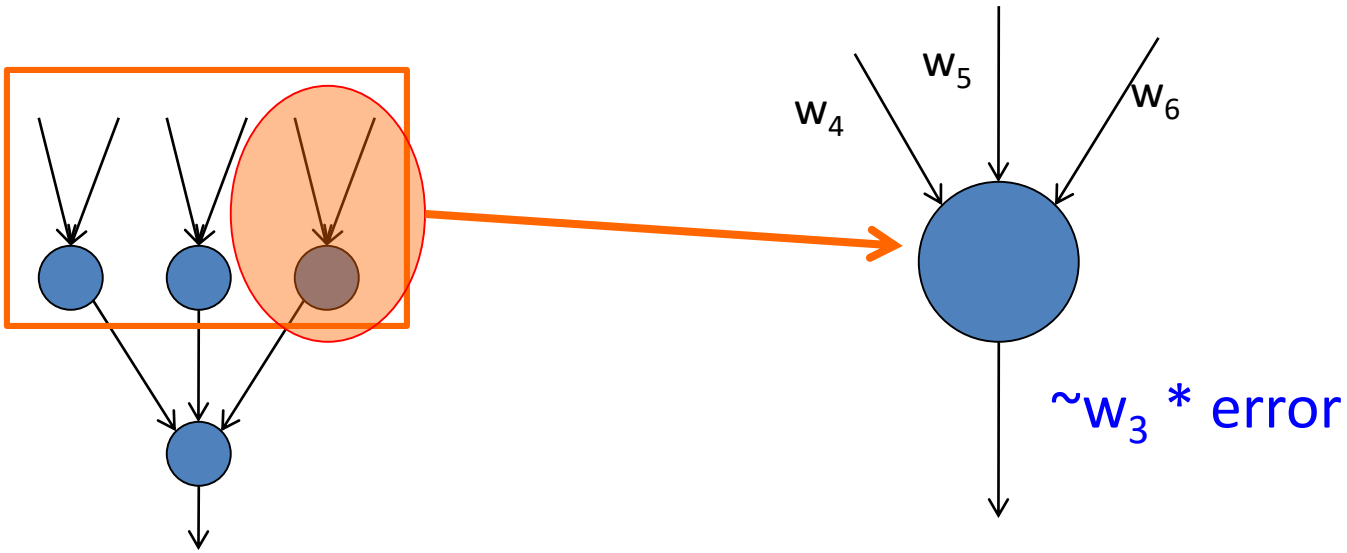
Key idea: propagate the error back to this layer

Backpropagation: intuition



error for node is ~ w_i * error

Backpropagation: intuition



Calculate as normal, but weight the error

Backpropagation: the details

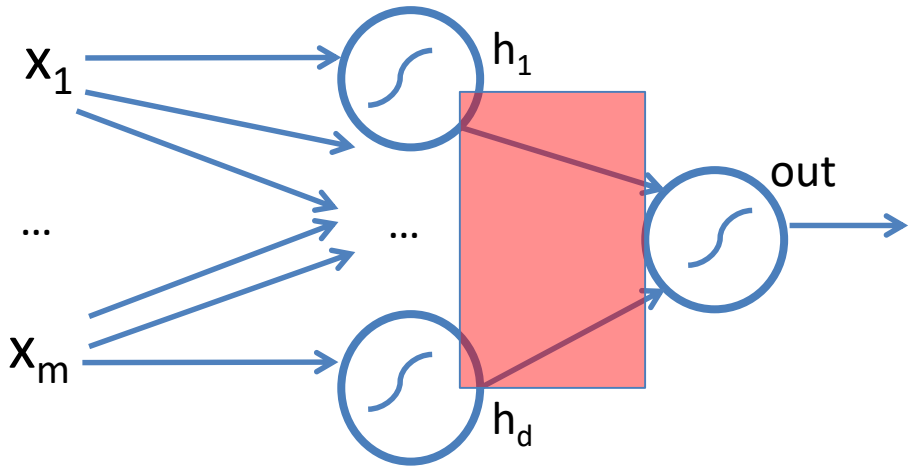
Gradient descent method for learning weights by optimizing a loss function

1. calculate output of all nodes
2. calculate the updates directly for the output layer
3. "backpropagate" errors through hidden layers

$$\text{Loss} = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$ squared error

Backpropagation: the details

Notation:



m: features/inputs

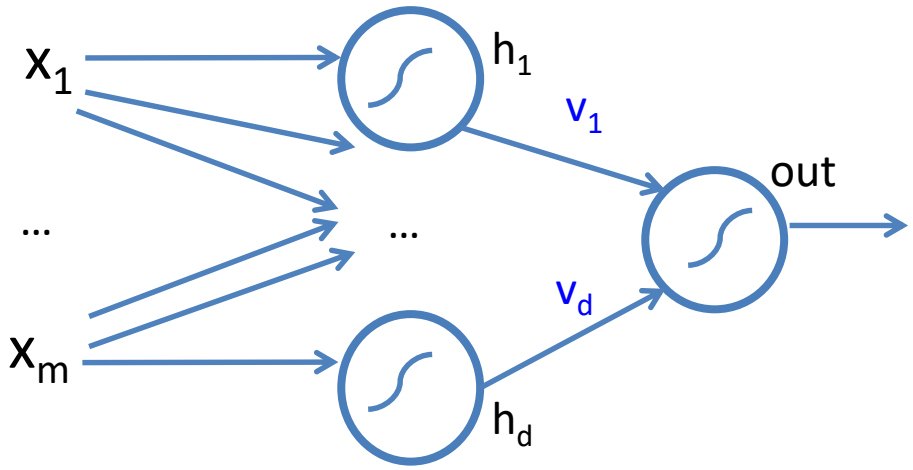
d: hidden nodes

h_k: output from hidden nodes

How many weights (ignore bias for now)?

Backpropagation: the details

Notation:



d weights: denote v_k

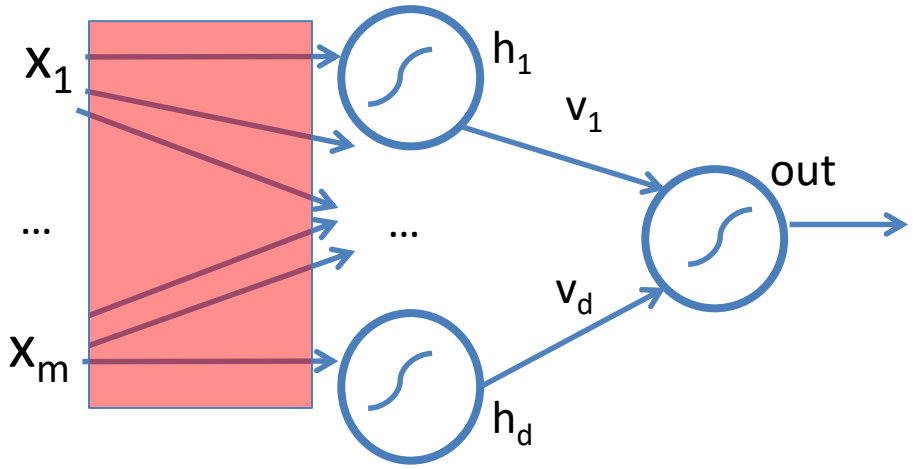
m: features/inputs

d: hidden nodes

h_k: output from hidden nodes

Backpropagation: the details

Notation:



m: features/inputs

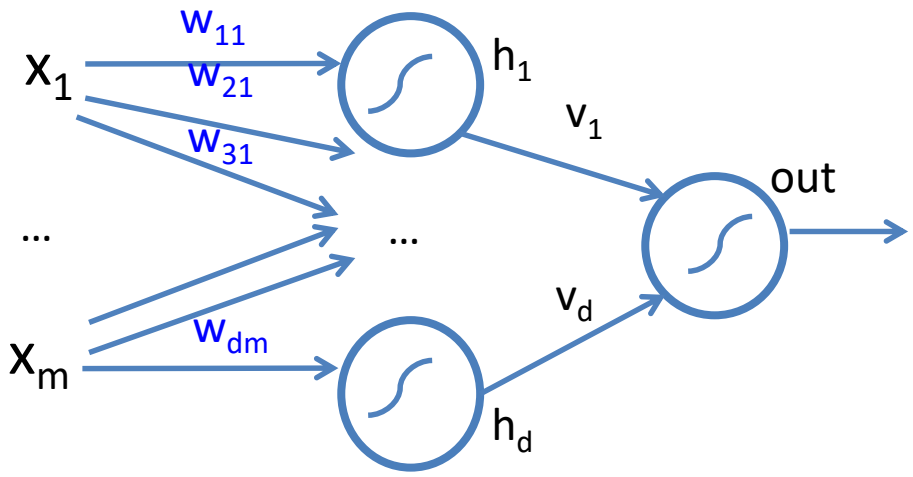
d: hidden nodes

h_k: output from hidden nodes

How many weights?

Backpropagation: the details

Notation:



m: features/inputs

d: hidden nodes

h_k: output from hidden nodes

d * m: denote w_{kj}

first index = hidden node
second index = feature

- w_{23}: weight from input 3 to hidden node 2
- w_4: all the *m* weights associated with hidden node 4

Backpropagation: the details

Gradient descent method for learning weights by optimizing a loss function

$$\arg\min_{w,v} \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

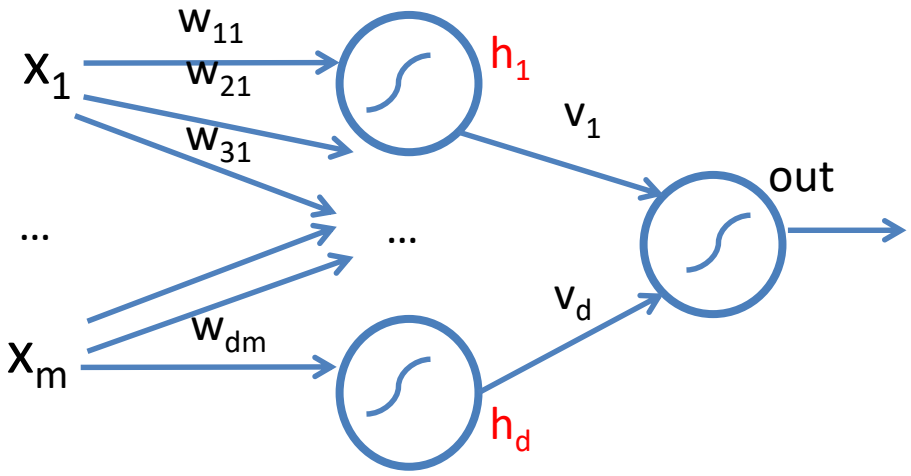
1. calculate output of all nodes

2. calculate the updates directly for the output layer

3. “backpropagate” errors through hidden layers

Backpropagation: the details

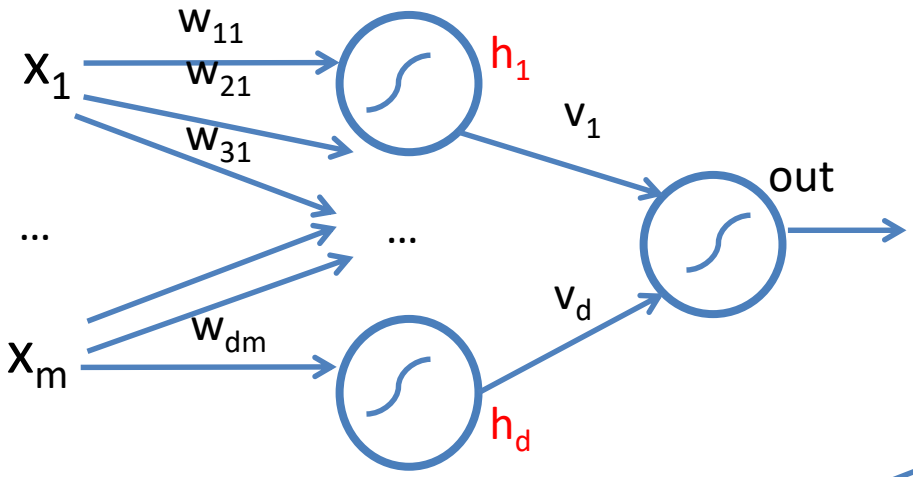
1. Calculate outputs of all nodes



What are h_k in terms of x and w?

Backpropagation: the details

1. Calculate outputs of all nodes



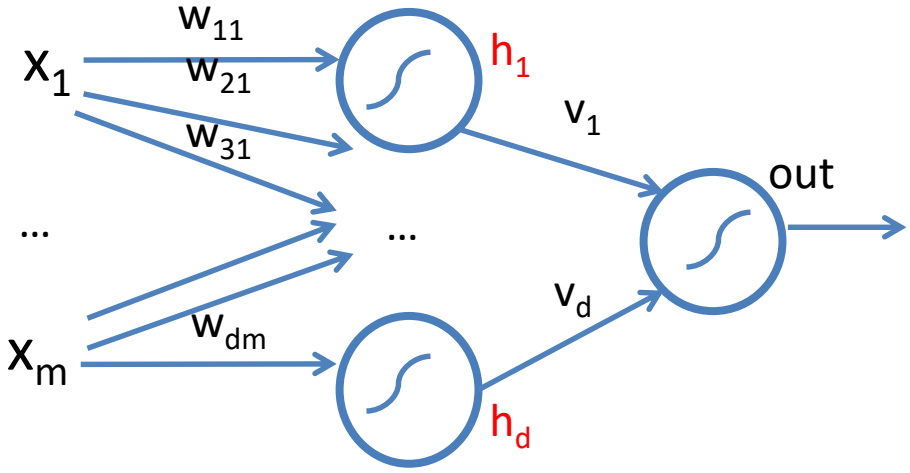
$$w_k \cdot x = \sum_{j=1}^{m} w_{kj} x_j$$

$$h_k = f(w_k \times x)$$

f is the activation function

Backpropagation: the details

1. Calculate outputs of all nodes

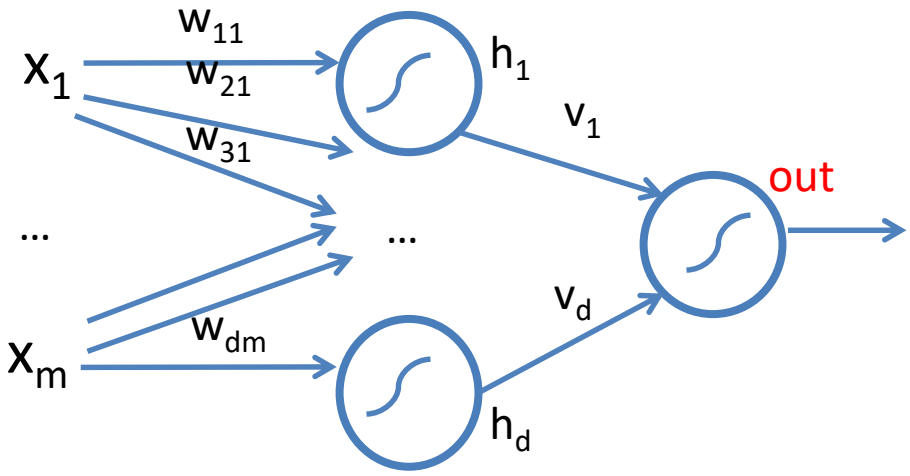


$$h_k = f(w_k^\top x) = \frac{1}{1 + e^{-w_k^\top x}}$$

f is the activation function

Backpropagation: the details

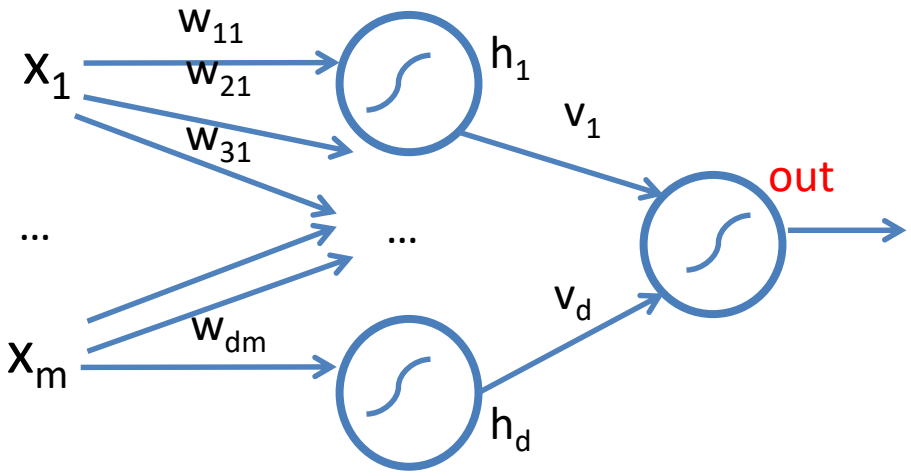
1. Calculate outputs of all nodes



What is *out* in terms of *h* and *v*?

Backpropagation: the details

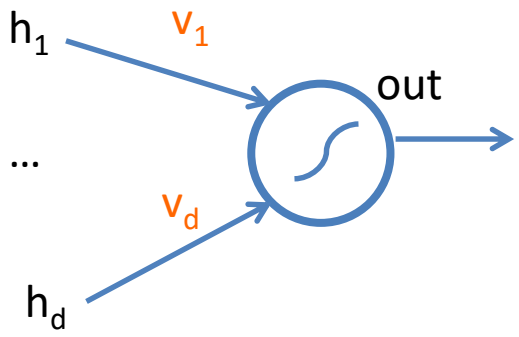
1. Calculate outputs of all nodes



$$\text{out} = f(v^\top h) = \frac{1}{1 + e^{-v^\top h}}$$

Backpropagation: the details

2. Calculate new weights for output layer

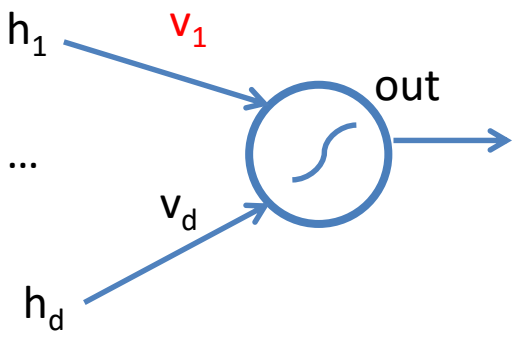


$$\arg\min_{w,v} \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Want to take a small step towards decreasing loss

Output layer weights

$$\arg\min_{w,v} \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



$$\frac{dL}{dv_k} = \sum_{i=1}^{n} \frac{d}{dv_k}\left(\frac{1}{2}(y_i - \hat{y}_i)^2\right)$$

$$= \frac{d}{dv_k}\left(\frac{1}{2}(y - f(v \cdot h))^2\right)$$

• • • • •

$$\frac{dL}{dv_k} = -(y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

where:

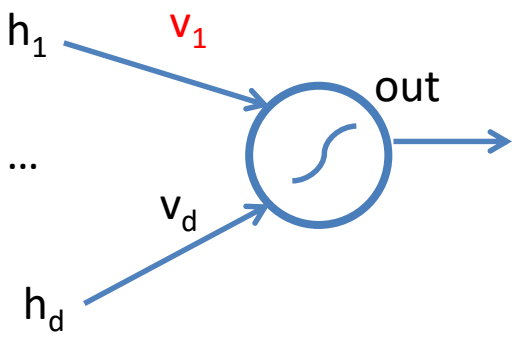
- $\hat{y}_i = f(v^\top h^{(i)}) = \frac{1}{1+e^{-v^\top h^{(i)}}}$
- $h^{(i)}$: hidden layer output for example i

$$v^\top h = \sum_k v_k h_k$$

Output layer weights

$$\frac{dL}{dv_k} = -(y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

$$v^\top h = \sum_k v_k h_k$$



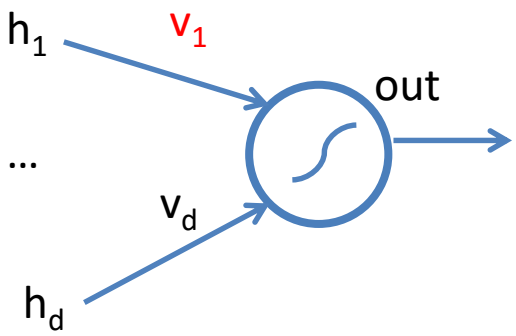
The actual update is a step towards *decreasing* loss:

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

- y is the true label.
- $\hat{y} = f(v^\top h) = \frac{1}{1+e^{-v^\top h}}$ is the predicted output, passed through the sigmoid function.
- $f'(v^\top h) = \hat{y}(1 - \hat{y})$ is the derivative of the sigmoid function.
- h_k is the activation from the hidden layer.

Output layer weights

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

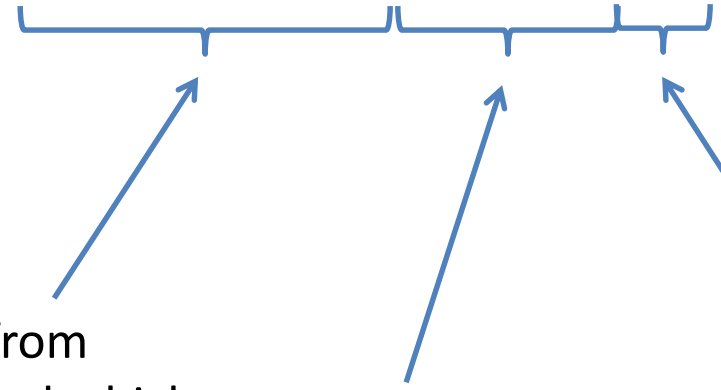


What are each of these?

Do they make sense individually?

Output layer weights

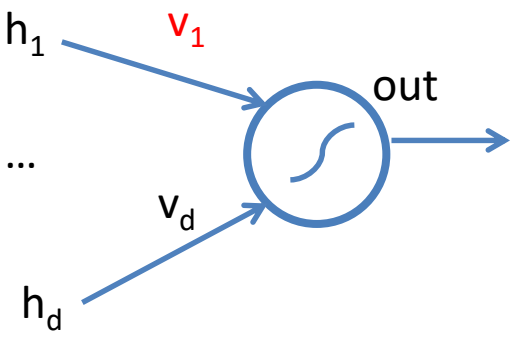
$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$



how far from correct and which direction

slope of the activation function where input is at

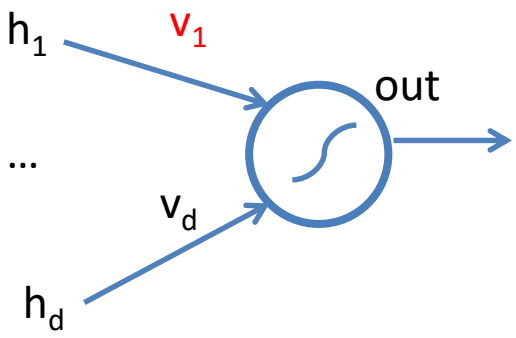
size and direction of the feature associated with this weight



Output layer weights

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

how far from correct and which direction



Case 1: $(y - \hat{y}) > 0$

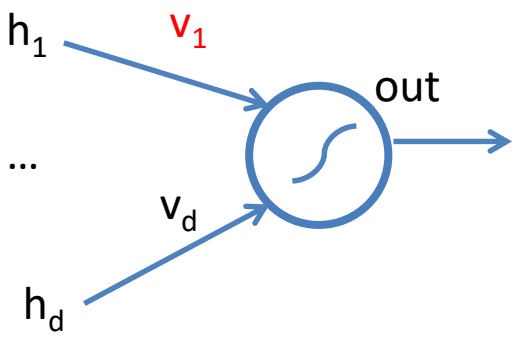
Case 2: $(y - \hat{y}) < 0$

?

Output layer weights

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$

how far from correct and which direction



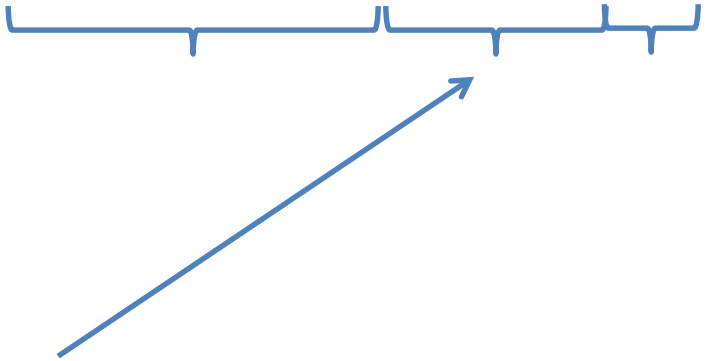
Case 1: $(y - \hat{y}) > 0$ prediction < label: increase the weight

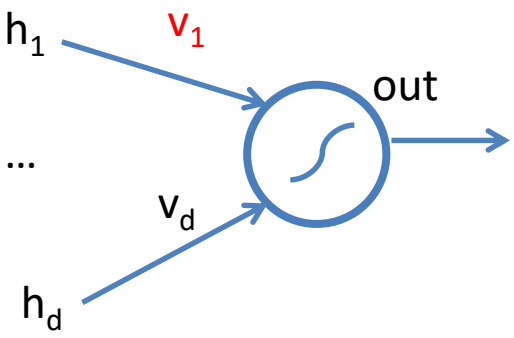
Case 2: $(y - \hat{y}) < 0$ prediction > label: decrease the weight

bigger difference = bigger change

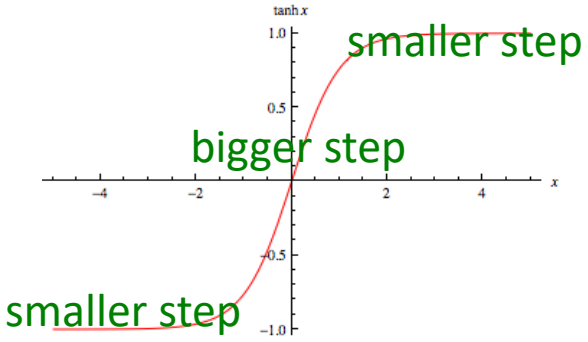
Output layer weights

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$



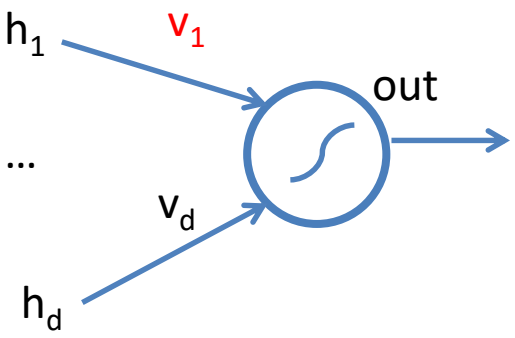


slope of the activation function where input is at



Output layer weights

$$v_k \leftarrow v_k + \eta \cdot (y - \hat{y}) \cdot f'(v^\top h) \cdot h_k$$



size and direction of the feature associated with this weight

perceptron update:

$$w_j = w_j + x_{ij} y_i$$

gradient descent update:

$$w_j = w_j + x_{ij} y_i c$$

Backpropagation: the details

Gradient descent method for learning weights by optimizing a loss function

$$\arg\min_{w,v} \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

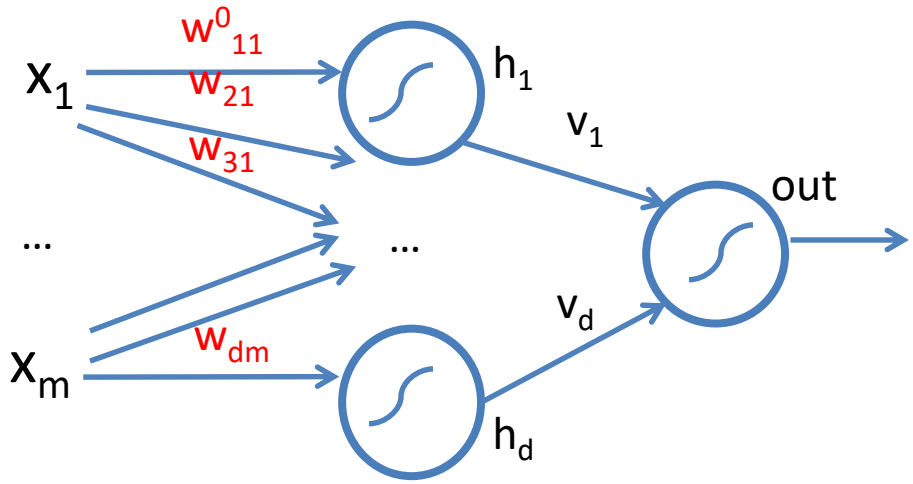
1. calculate output of all nodes

2. calculate the updates directly for the output layer

3. “backpropagate” errors through hidden layers

Backpropagation

3. “backpropagate” errors through hidden layers



$$\arg\min_{w,v} \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

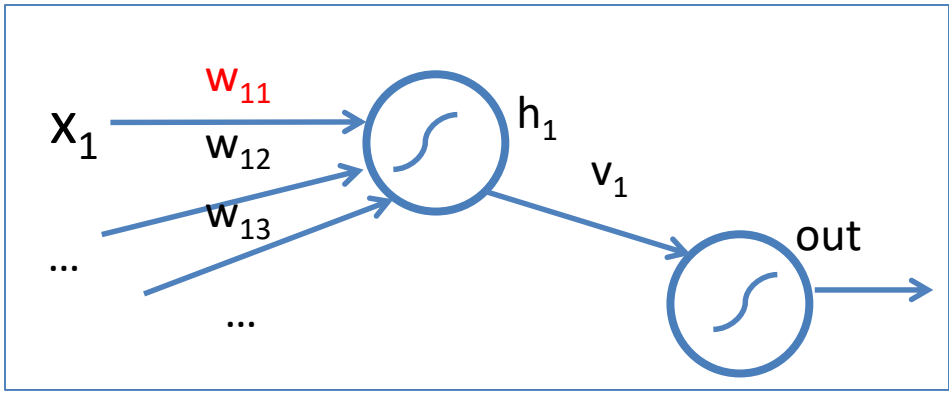
Want to take a small step towards decreasing loss

Hidden layer weights

$$L = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

Where:

- $y^{(i)}$ is the true label for example i
- $\hat{y}^{(i)} = f(v \cdot h^{(i)})$, the predicted output
- $h_k^{(i)} = \sigma(w_k \cdot x^{(i)})$, activation of hidden neuron k
- w_{kj} is the weight from input neuron j to hidden neuron k



$$\frac{dL}{dw_{kj}} = \sum_i \left[-(y_i - \hat{y}_i) \cdot f'(v \cdot h^{(i)}) \cdot v_k \cdot f'(w_k \cdot x^{(i)}) \cdot x_j^{(i)}\right]$$

Where for each example i:

- $\hat{y}_i = f(v \cdot h^{(i)})$
- $h^{(i)}$ is the hidden layer activation for input $x^{(i)}$
- f' is the sigmoid derivative: $f'(z) = f(z)(1 - f(z))$
- $x_j^{(i)}$ is the j^{th} input for example i

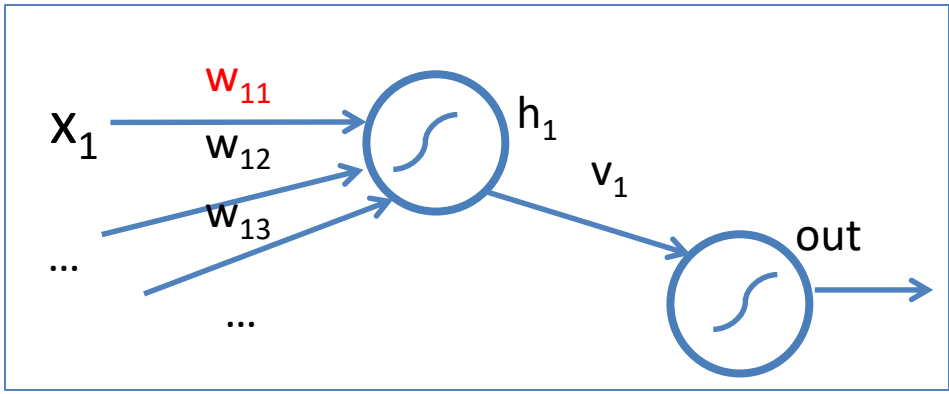
$$w_k \cdot x = \sum_j w_{kj} x_j$$

Hidden layer weights

$$\frac{dloss}{dw_{kj}} = \frac{d}{dw_{kj}}\left(\frac{1}{2}(y-\hat{y})^2\right)$$

$$= \frac{d}{dw_{kj}}\left(\frac{1}{2}(y - f(v \cdot h))^2\right)$$

$$= \ldots.$$



$$\hat{y} = f(v \cdot h)$$

$$\frac{dL}{dw_{kj}} = \sum_i \left[-(y_i - \hat{y}_i) \cdot f'(v \cdot h^{(i)}) \cdot v_k \cdot f'(w_k \cdot x^{(i)}) \cdot x_j^{(i)}\right] \quad w_k \cdot x = \sum_j w_{kj} x_j$$

weight from hidden layer to output layer

slope of wx

input feature

Backpropagation

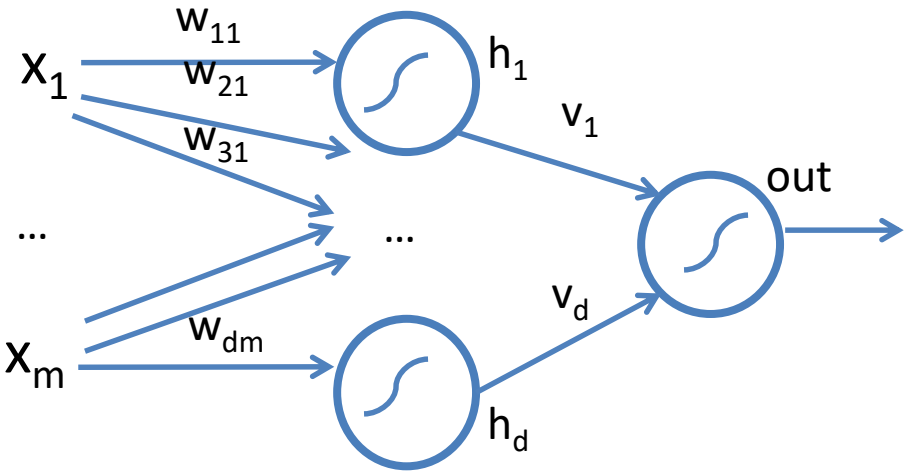
output layer

$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

hidden layer

$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot v_k \cdot f'(w_k \cdot x) \cdot x_j$$

What's different?



Backpropagation

output layer

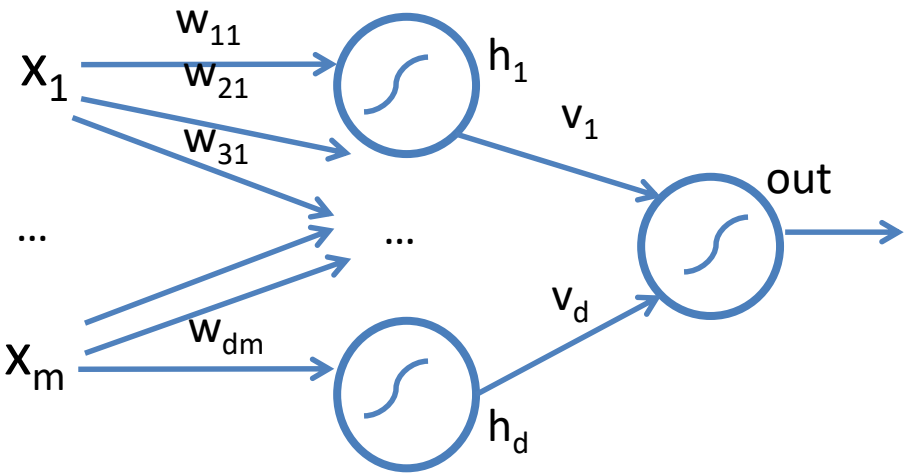
$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

error — output activation slope — input

hidden layer

$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot v_k \cdot f'(w_k \cdot x) \cdot x_j$$

error — output activation slope — input



weight from hidden layer to output layer

slope of wx

Backpropagation

output layer

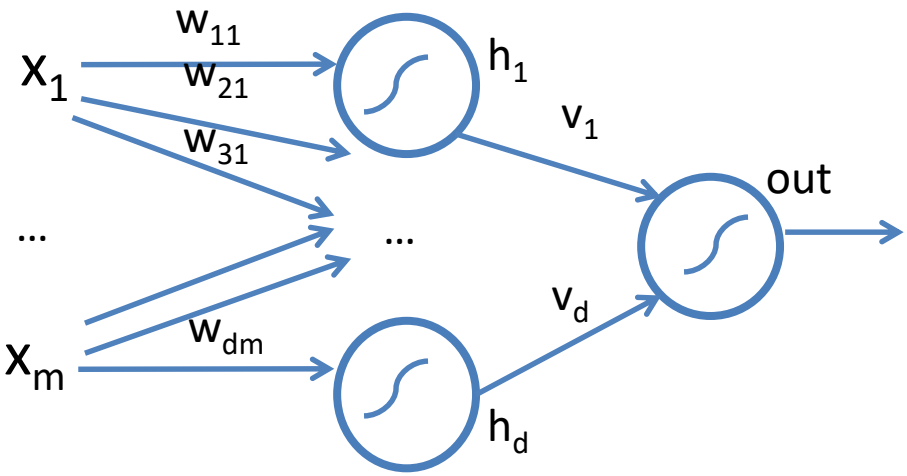
$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

error — output activation slope — input

hidden layer

$$-(y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot v_k \cdot f'(w_k \cdot x) \cdot x_j$$

error — output activation slope — input



how much of the error came from this hidden node

how much do we need to change

Backpropgation generalization

output layer

$$v_k = v_k + (y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

$$v_k = v_k + h_k \cdot (y - f(v \cdot h)) \cdot f'(v \cdot h)$$

$$v_k = v_k + h_k \mathrm{D}_{out}$$

$$\mathrm{D}_{out} = f'(v \cdot h)(y - f(v \cdot h))$$ modified error

derivative of input at node

error

Backpropgation generalization

output layer

$$v_k = v_k + (y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

$$v_k = v_k + h_k \cdot (y - f(v \cdot h)) \cdot f'(v \cdot h)$$

$$v_k = v_k + h_k \mathrm{D}_{out}$$

$$\mathrm{D}_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

$$w_{kj} = w_{kj} + (y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot v_k \cdot f'(w_k \cdot x) \cdot x_j$$

$$w_{kj} = w_{kj} + x_j \cdot f'(w_k \cdot x) \cdot v_k \cdot f'(v \cdot h) \cdot (y - f(v \cdot h))$$

$$w_{kj} = w_{kj} + x_j \mathrm{D}_k$$

$$\mathrm{D}_k = f'(w_k \cdot x) v_k f'(v \cdot h)(y - f(v \cdot h))$$

Can we write this more succinctly?

Backpropgation generalization

output layer

$$v_k = v_k + (y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot h_k$$

$$v_k = v_k + h_k \cdot (y - f(v \cdot h)) \cdot f'(v \cdot h)$$

$$v_k = v_k + h_k \mathrm{D}_{out}$$

$$\mathrm{D}_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

$$w_{kj} = w_{kj} + (y - f(v \cdot h)) \cdot f'(v \cdot h) \cdot v_k \cdot f'(w_k \cdot x) \cdot x_j$$

$$w_{kj} = w_{kj} + x_j \cdot f'(w_k \cdot x) \cdot v_k \cdot f'(v \cdot h) \cdot (y - f(v \cdot h))$$

$$w_{kj} = w_{kj} + x_j \mathrm{D}_k$$

$$\mathrm{D}_k = f'(w_k \cdot x) v_k f'(v \cdot h)(y - f(v \cdot h))$$

$$= f'(w_k \cdot x) v_k \mathrm{D}_{out}$$

Backpropgation generalization

output layer

$$v_k = v_k + h_k \mathrm{D}_{out}$$

$$\mathrm{D}_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

$$w_{kj} = w_{kj} + x_j \mathrm{D}_k$$

$$\mathrm{D}_k = f'(w_k \cdot x) v_k f'(v \cdot h)(y - f(v \cdot h))$$

$$= f'(w_k \cdot x) v_k \mathrm{D}_{out}$$

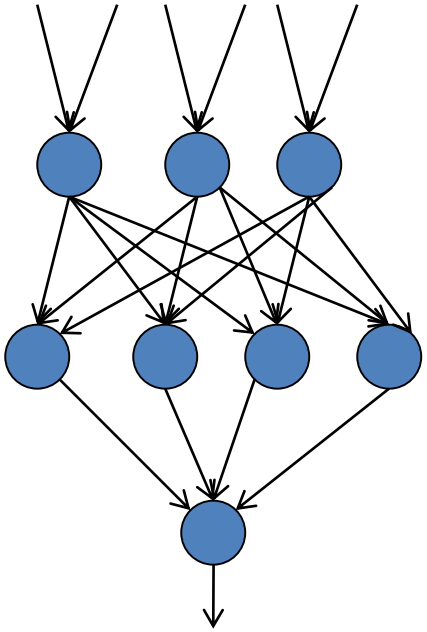
weight to output layer

modified error of output layer

$$w = w + input * \mathrm{D}_{current}$$

$$\mathrm{D}_{current} = f'(current\_input) w_{output} \mathrm{D}_{output}$$

Backprop on multilayer networks

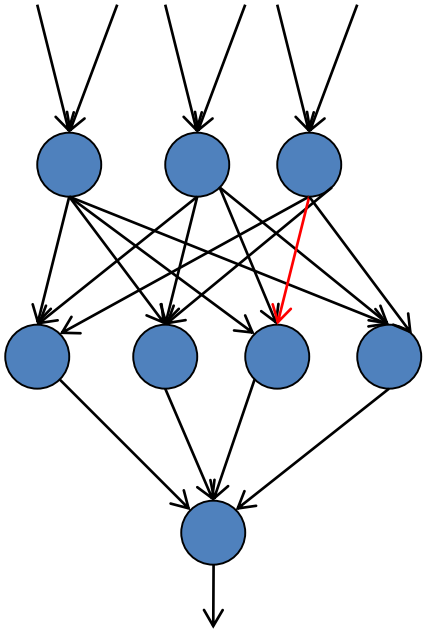


Anything different here?

$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

$$w = w + input * \Delta_{output}$$

Backprop on multilayer networks

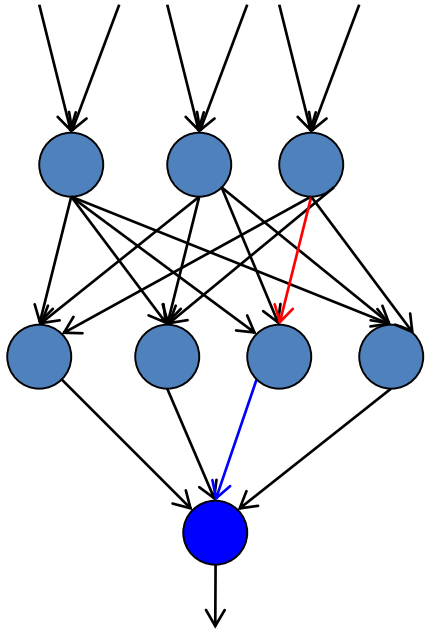


$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

$$w = w + input * \Delta_{output}$$

What "errors" at the next layer does the highlighted edge affect?

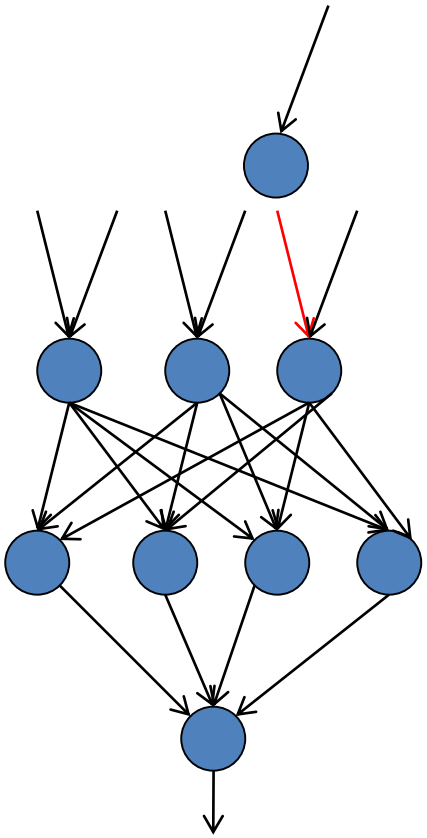
Backprop on multilayer networks



$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

$$w = w + input * \Delta_{output}$$

Backprop on multilayer networks

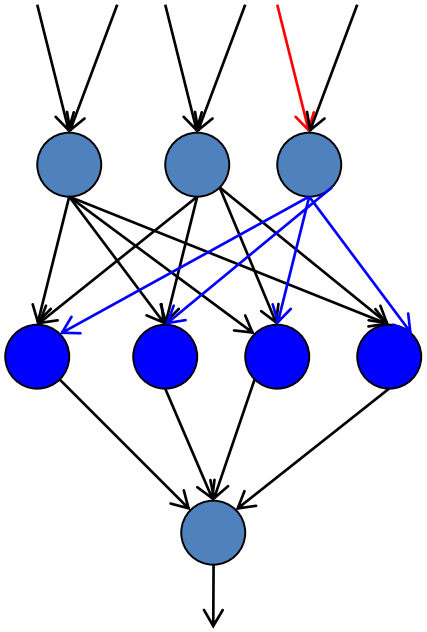


$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

$$w = w + input * \Delta_{output}$$

What "errors" at the next layer does the highlighted edge affect?

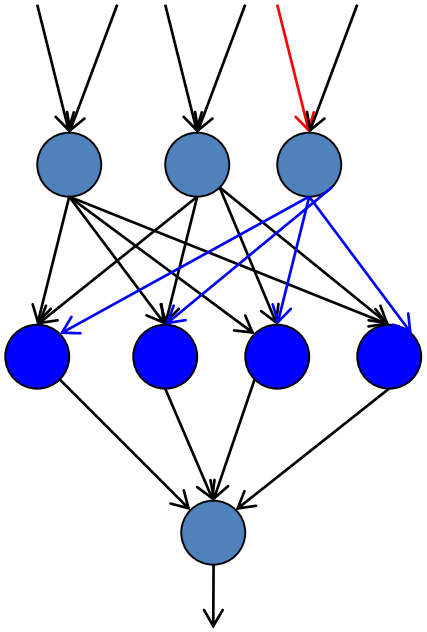
Backprop on multilayer networks



$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input) w_{output} \mathrm{D}_{output}$$

$$w = w + input * \mathrm{D}_{output}$$

Backprop on multilayer networks

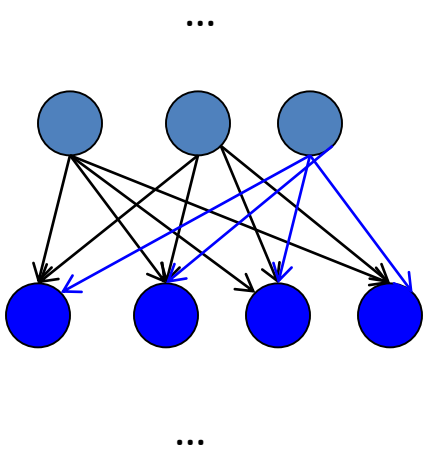


$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input)\sum w_{output}\mathrm{D}_{output}$$

$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input) w_{output}\mathrm{D}_{output}$$

$$w = w + input * \mathrm{D}_{output}$$

Backprop on multilayer networks

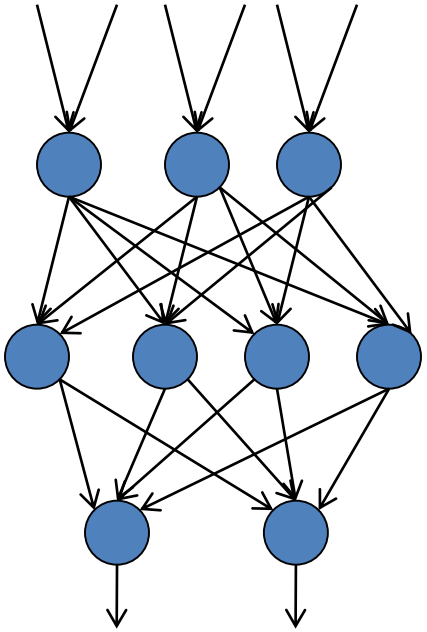


$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input)\sum w_{output}\mathrm{D}_{output}$$

Backpropogation:

- Calculate new weights and modified errors at output layer
- Recursively calculate new weights and modified errors on hidden layers based on recursive relationship
- Update model with new weights

Multiple output nodes



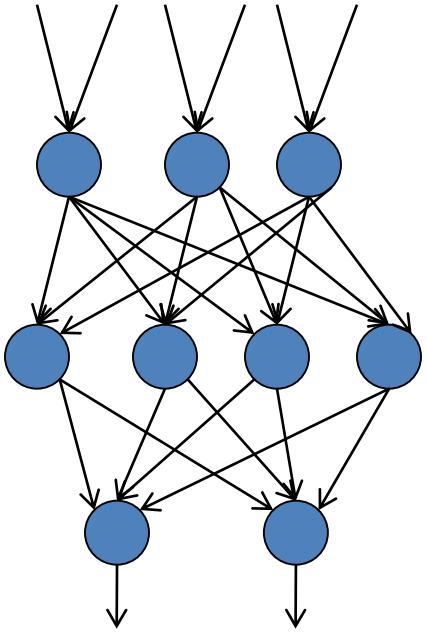
$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input)\sum w_{output}\Delta_{output}$$

$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output}\Delta_{output}$$

$$w = w + input * \Delta_{output}$$

How does multiple outputs change things?

Multiple output nodes



$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input)\sum w_{output}\mathrm{D}_{output}$$

$$w = w + input * \mathrm{D}_{current}$$
$$\mathrm{D}_{current} = f'(current\_input)\sum w_{output}\mathrm{D}_{output}$$

$$w = w + input * \mathrm{D}_{output}$$

How does multiple outputs change things?

Backpropagation implementation

Output layer update:

$$v_k = v_k + h_k \cdot (y - f(v \cdot h)) \cdot f'(v \cdot h)$$

Hidden layer update:

$$w_{kj} = w_{kj} + x_j \cdot f'(w_k \cdot x) \cdot v_k \cdot f'(v \cdot h) \cdot (y - f(v \cdot h))$$

Any missing information for implementation?

Backpropagation implementation

Output layer update:

$$v_k = v_k + h_k \cdot (y - f(v \cdot h)) \cdot f'(v \cdot h)$$

Hidden layer update:

$$w_{kj} = w_{kj} + x_j \cdot f'(w_k \cdot x) \cdot v_k \cdot f'(v \cdot h) \cdot (y - f(v \cdot h))$$

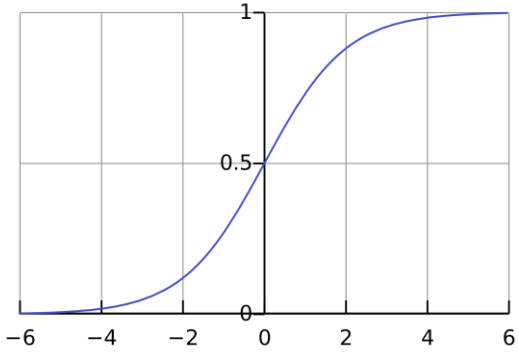
1. What activation function are we using
2. What is the derivative of that activation function

Activation function derivatives

sigmoid

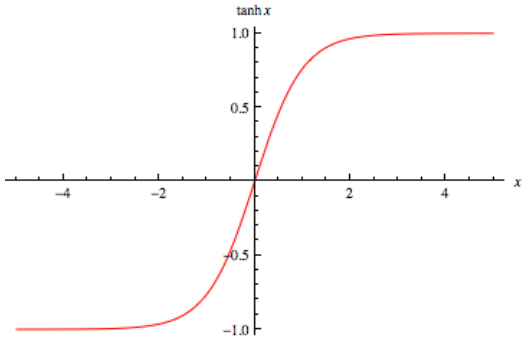
$$s(x) = \frac{1}{1 + e^{-x}}$$

$$s'(x) = s(x)(1 - s(x))$$



tanh

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2 x$$



Learning rate

Layer	Weight Update Rule
Output Layer	$v_k = v_k + \eta \cdot h_k \cdot (y - \hat{y}) \cdot f'(v \cdot h)$
Hidden Layer	$w_{kj} = w_{kj} + \eta \cdot x_j \cdot f'(w_k \cdot x) \cdot v_k \cdot f'(v \cdot h) \cdot (y - \hat{y})$

Where:

- η: learning rate
- $\hat{y} = f(v \cdot h)$: model prediction
- f: sigmoid activation
- $h_k = f(w_k \cdot x)$: hidden unit activations

Backpropagation implementation

Just like gradient descent!

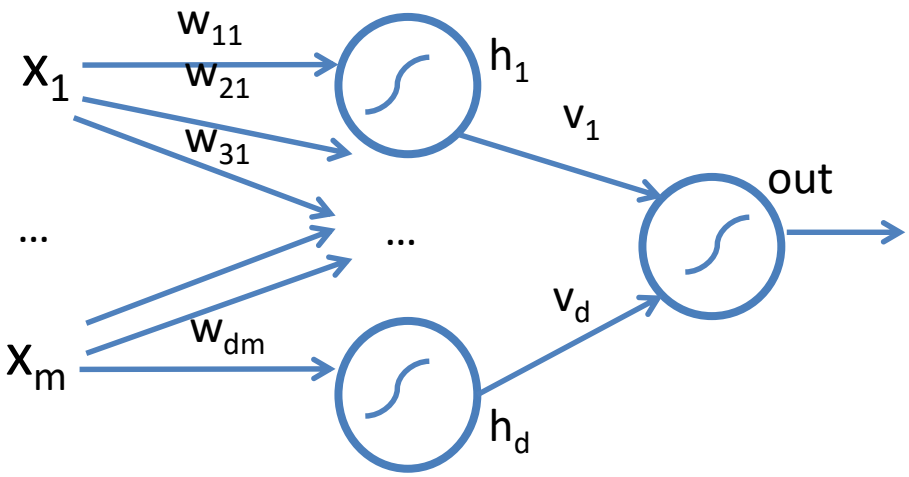
for some number of iterations:

randomly shuffle training data

for each example:

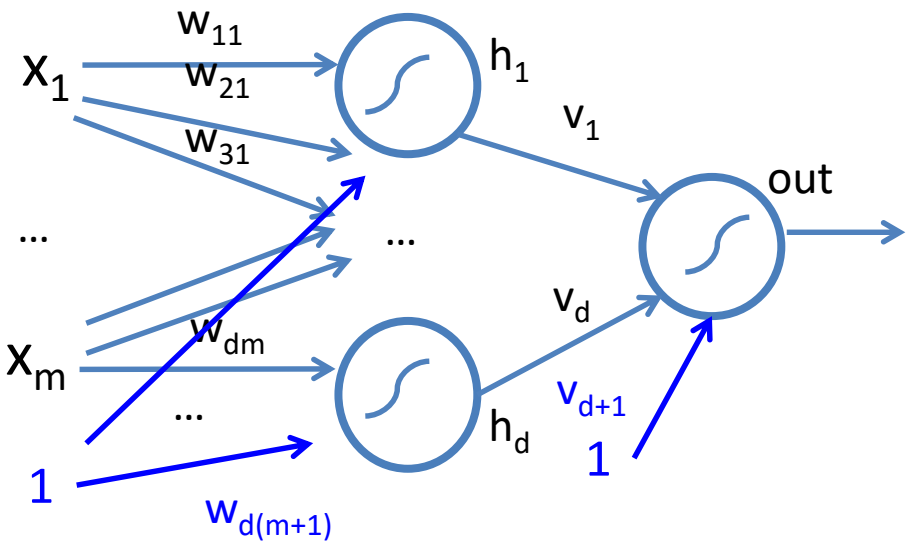
- Compute all outputs going forward
- Calculate new weights and modified errors at output layer
- Recursively calculate new weights and modified errors on hidden layers based on recursive relationship
- Update model with new weights

Handling bias



How should we learn the bias?

Handling bias



1. Add an extra feature hard-wired to 1 to all the examples
2. For other layers, add an extra parameter whose input is always 1

Online vs. batch learning

for some number of iterations:

randomly shuffle training data

for each example:

- Compute all outputs going forward
- Calculate new weights and modified errors at output layer
- Recursively calculate new weights and modified errors on hidden layers based on recursive relationship
- Update model with new weights

Online learning: update weights after each example

Batch learning?

Batch learning

for some number of iterations:

randomly shuffle training data

initialize weight accumulators to 0 (one for each weight)

for each example:

- Compute all outputs going forward
- Calculate new weights and modified errors at output layer
- Recursively calculate new weights and modified errors on hidden layers based on recursive relationship
- Add new weights to weight accumulators

Divide weight accumulators by number of examples

Update model weights by weight accumulators

Process *all* of the examples before updating the weights

Challenges of neural networks?

Picking network configuration

Can be slow to train for large networks and large amounts of data

Loss functions (including squared error) are generally not convex *with respect to the parameter space*

History of Neural Networks

McCulloch and Pitts (1943) – introduced model of artificial neurons and suggested they could learn

Hebb (1949) – Simple updating rule for learning

Rosenblatt (1962) - the *perceptron* model

Minsky and Papert (1969) – wrote *Perceptrons*

Bryson and Ho (1969, but largely ignored until 1980s--Rosenblatt) – invented backpropagation learning for multilayer networks