BACKPROPAGATION

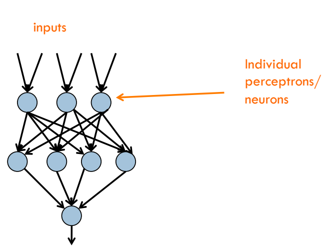
David Kauchak
CS158 – Fall 2016

Admin

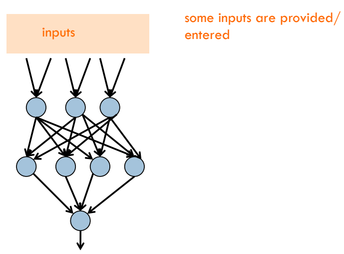
Assignment 7

Assignment 8
- Goals today

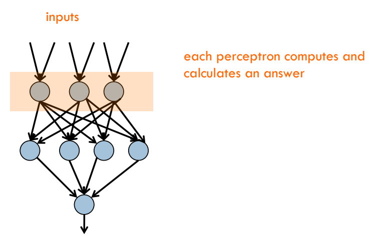
Neural network



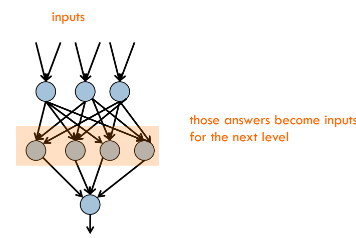
Neural network



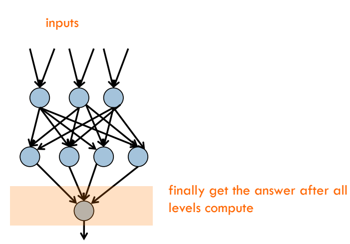
Neural network



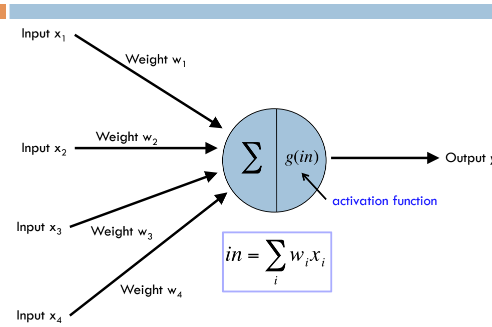
Neural network



Neural network



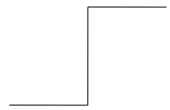
A neuron/perceptron



Activation functions

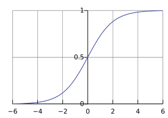
hard threshold:

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

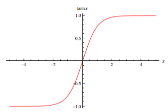


sigmoid

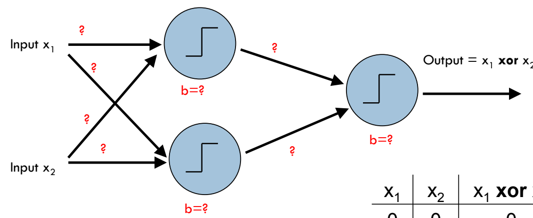
$$g(x) = \frac{1}{1+e^{-x}}$$



tanh x



Training

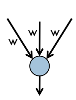


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

How do we learn the weights?

Learning in multilayer networks

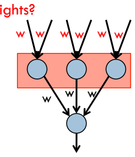
Challenge: for multilayer networks, we don't know what the expected output/error is for the internal nodes!



perceptron/
linear model

how do we learn these weights?

expected output?



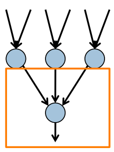
neural network

Backpropagation: intuition

Gradient descent method for learning weights by optimizing a loss function

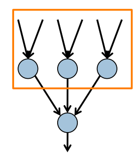
1. calculate output of all nodes
2. calculate the weights for the output layer based on the error
3. "backpropagate" errors through hidden layers

Backpropagation: intuition



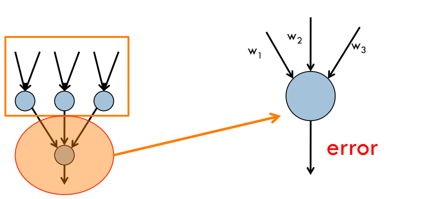
We can calculate the actual error here

Backpropagation: intuition



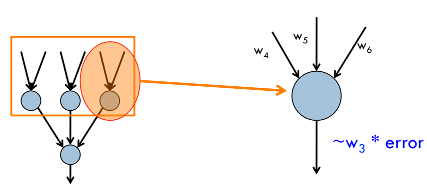
Key idea: propagate the error back to this layer

Backpropagation: intuition



error for node is ~ w_i * **error**

Backpropagation: intuition



Calculate as normal, but weight the error

Backpropagation: the details

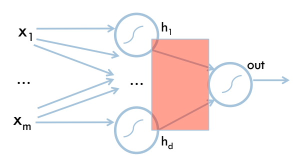
Gradient descent method for learning weights by optimizing a loss function

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

$$loss = \sum_x \frac{1}{2}(y - \hat{y})^2$$ squared error

Backpropagation: the details

Notation:



m: features/inputs

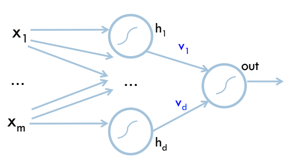
d: hidden nodes

hj: output from hidden nodes

How many weights (ignore bias for now)?

Backpropagation: the details

Notation:



m: features/inputs

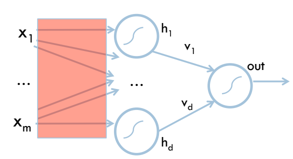
d: hidden nodes

hj: output from hidden nodes

d weights: denote v_k

Backpropagation: the details

Notation:



m: features/inputs

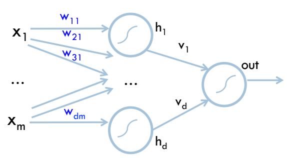
d: hidden nodes

hj: 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

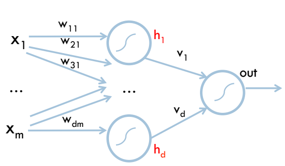
Gradient descent method for learning weights by optimizing a loss function

$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y - \hat{y})^2$$

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

Backpropagation: the details

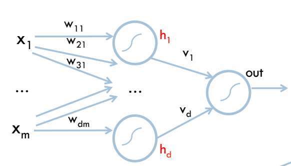
1. Calculate outputs of all nodes



What are h_k in terms of *x* and *w*?

Backpropagation: the details

1. Calculate outputs of all nodes



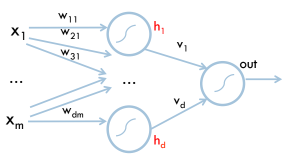
$$w_k \cdot x = \sum_j w_{kj} x$$

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

f is the activation function

Backpropagation: the details

1. Calculate outputs of all nodes

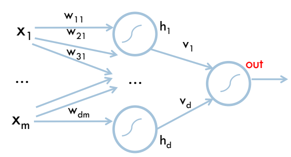


$$h_k = f(w_k \cdot x) = \frac{1}{1+e^{-w_k \cdot x}}$$

f is the activation function

Backpropagation: the details

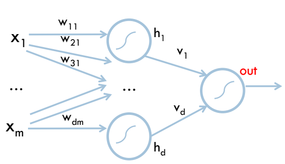
1. Calculate outputs of all nodes



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

Backpropagation: the details

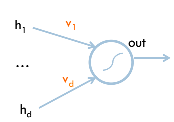
1. Calculate outputs of all nodes



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

Backpropagation: the details

2. Calculate new weights for output layer

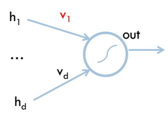


$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y-\hat{y})^2$$

Want to take a small step towards decreasing loss

Output layer weights

$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y-\hat{y})^2$$



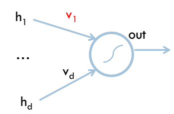
$$\frac{dloss}{dv_k} = \frac{d}{dv_k}\left(\frac{1}{2}(y-\hat{y})^2\right)$$

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

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

Output layer weights

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



$$= -(y-f(v\cdot h))\frac{d}{dv_k}f(v\cdot h)$$

$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dv_k}v\cdot h$$

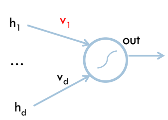
$$= -(y-f(v\cdot h))f'(v\cdot h)h_k \qquad v\cdot h = \sum_k v_k h_k$$

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

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

Output layer weights

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

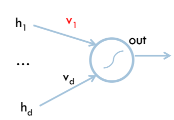


What are each of these?

Do they make sense individually?

Output layer weights

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



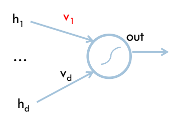
how far from correct and which direction

slope of the activation function where input is at

size and direction of the feature associated with this weight

Output layer weights

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



how far from correct
and which direction

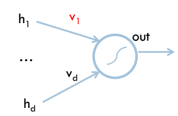
$$(y - f(v \cdot h)) > 0$$

$$(y - f(v \cdot h)) < 0$$

?

Output layer weights

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



how far from correct
and which direction

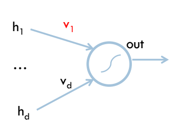
$(y - f(v \cdot h)) > 0$ prediction < label: increase the weight

$(y - f(v \cdot h)) < 0$ prediction > label: decrease the weight

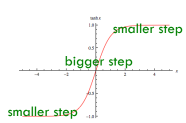
bigger difference = bigger change

Output layer weights

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

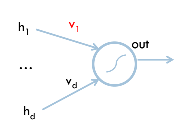


slope of the activation
function where input is at



Output layer weights

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



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

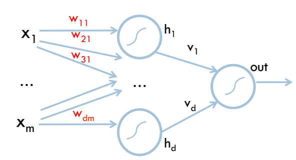
Gradient descent method for learning weights by optimizing a loss function

$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y-\hat{y})^2$$

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

Backpropagation

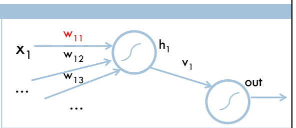
3. "backpropagate" errors through hidden layers



$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y-\hat{y})^2$$

Want to take a small step towards decreasing loss

Hidden layer weights



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

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

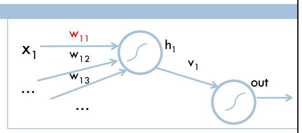
$$\hat{y} = f(v\cdot h)$$

$$= (y-f(v\cdot h))\frac{d}{dw_{kj}}(y-f(v\cdot h))$$

$$= -(y-f(v\cdot h))\frac{d}{dw_{kj}}f(v\cdot h)$$

$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dw_{kj}}v\cdot h$$

Hidden layer weights



$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dw_{kj}}v\cdot h$$

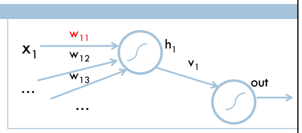
$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dw_{kj}}v_k h_k$$

derivative of other vh components are not affected by w_{kj}

$$= -(y-f(v\cdot h))f'(v\cdot h)v_k\frac{d}{dw_{kj}}h_k$$

$$= -(y-f(v\cdot h))f'(v\cdot h)v_k\frac{d}{dw_{kj}}f(w_k\cdot x)$$

Hidden layer weights



$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} f(w_k \cdot x)$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k f'(w_k \cdot x) \frac{d}{dw_{kj}} w_k \cdot x$$

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

Why all the math?

"I also wouldn't mind more math!"

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

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

$$= (y - f(v \cdot h)) \frac{d}{dv_k}(y - f(v \cdot h))$$

$$= -(y - f(v \cdot h)) \frac{d}{dv_k} f(v \cdot h)$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) \frac{d}{dv_k} v \cdot h$$

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

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

$$= (y - f(v \cdot h)) \frac{d}{dw_{kj}}(y - f(v \cdot h))$$

$$= -(y - f(v \cdot h)) \frac{d}{dw_{kj}} f(v \cdot h)$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) \frac{d}{dw_{kj}} v \cdot h$$

What happened here?

$$= -(y - f(v \cdot h)) f'(v \cdot h) \frac{d}{dw_{kj}} v_k h_k$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} h_k$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} f(w_k \cdot x)$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k f'(w_k \cdot x) \frac{d}{dw_{kj}} w_k \cdot x$$

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

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

$$= -(y - f(v \cdot h)) f'(v \cdot h) \frac{d}{dw_{kj}} v \cdot h$$

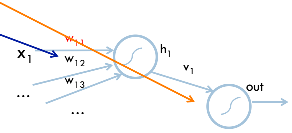
$$= -(y - f(v \cdot h)) f'(v \cdot h) \frac{d}{dw_{kj}} v_k h_k$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} h_k$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} f(w_k \cdot x)$$

$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k f'(w_k \cdot x) \frac{d}{dw_{kj}} w_k \cdot x$$

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



What is the slope vh with respect to w_{kj}

$= -(y - f(v\cdot h)) f'(v\cdot h) \frac{d}{dw_{kj}} v\cdot h$

What is the slope vh with respect to w_{kj}

$= -(y - f(v\cdot h)) f'(v\cdot h) \frac{d}{dw_{kj}} v_k h_k$

$= -(y - f(v\cdot h)) f'(v\cdot h) v_k \frac{d}{dw_{kj}} h_k$

$= -(y - f(v\cdot h)) f'(v\cdot h) v_k \frac{d}{dw_{kj}} f(w_k\cdot x)$

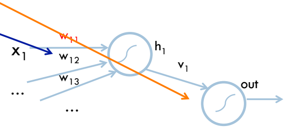
$= -(y - f(v\cdot h)) f'(v\cdot h) v_k f'(w_k\cdot x) \frac{d}{dw_{kj}} w_k\cdot x$

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

weight from hidden layer to output layer

slope of wx

input feature



Backpropagation

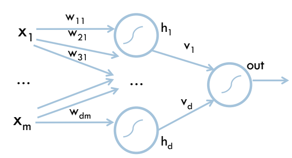
output layer

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

hidden layer

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

What's different?



Backpropagation

output layer

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

error

output activation slope

input

hidden layer

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

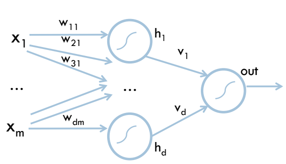
error

output activation slope

input

weight from hidden layer to output layer

slope of wx



Backpropagation

output layer

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

error

output activation slope

input

hidden layer

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

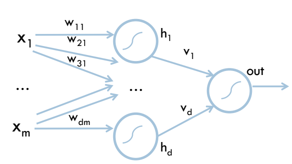
error

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

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

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

$$v_k = v_k + h_k \Delta_{out}$$

$$\Delta_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

modified error

derivative of input at node — error

Backpropgation generalization

output layer

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

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

$$v_k = v_k + h_k \Delta_{out}$$

$$\Delta_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

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

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

$$w_{kj} = w_{kj} + x_j \Delta_k$$

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

Can we write this more succinctly?

Backpropgation generalization

output layer

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

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

$$v_k = v_k + h_k \Delta_{out}$$

$$\Delta_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

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

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

$$w_{kj} = w_{kj} + x_j \Delta_k$$

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

$$= f'(w_k \cdot x) v_k \Delta_{out}$$

Backpropgation generalization

output layer

$$v_k = v_k + h_k \Delta_{out}$$

$$\Delta_{out} = f'(v \cdot h)(y - f(v \cdot h))$$

hidden layer

$$w_{kj} = w_{kj} + x_j \Delta_k$$

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

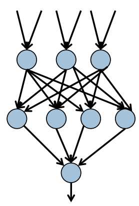
$$= f'(w_k \cdot x) v_k \Delta_{out}$$

weight to output layer — modified error of output layer

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

$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

Backprop on multilayer networks

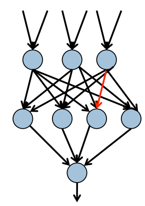


Anything different here?

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

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

Backprop on multilayer networks

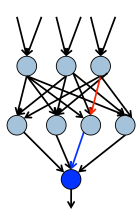


$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$

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

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

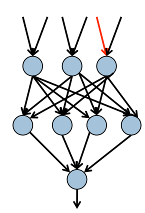
Backprop on multilayer networks



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

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

Backprop on multilayer networks

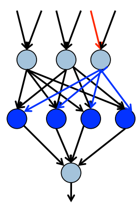


$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$

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

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

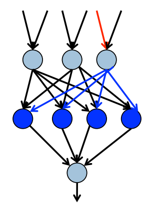
Backprop on multilayer networks



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

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

Backprop on multilayer networks

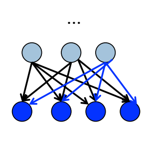


$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) \sum w_{output} \Delta_{output}$$

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

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

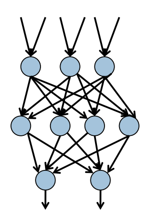
Backprop on multilayer networks



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

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



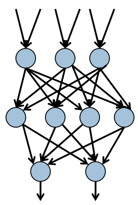
$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) \sum w_{output} \Delta_{output}$$

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

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

How does multiple outputs change things?

Multiple output nodes



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

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

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

How does multiple outputs change things?

Backpropagation implementation

Output layer update:

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

Hidden layer update:

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

Any missing information for implementation?

Backpropagation implementation

Output layer update:

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

Hidden layer update:

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

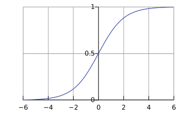
1. What activation function are we using
2. What is the derivative of that activation function

Activation function derivatives

sigmoid

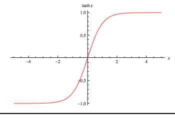
$$s(x) = \frac{1}{1 + e^{-x}}$$

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



tanh

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



Learning rate

Output layer update:

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

Hidden layer update:

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

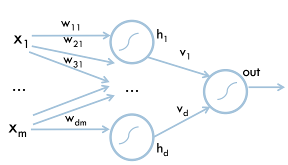
- Like gradient descent for linear classifiers, use a learning rate
- Often will start larger and then get smaller

Backpropagation implementation

Just like gradient descent!

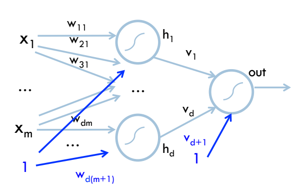
for some number of iterations:
  randomly shuffle training data
  for each example:
   - Compute all outputs going forward
   - Calculate new weights and modified errors at output layer
   - Recursively calculate new weights and modified errors on hidden layers based on recursive relationship
   - Update 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

Many variations

Momentum: include a factor in the weight update to keep moving in the direction of the previous update

Mini-batch:
- Compromise between online and batch
- Avoids noisiness of updates from online while making more educated weight updates

Simulated annealing:
- With some probability make a random weight update
- Reduce this probability over time

…

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

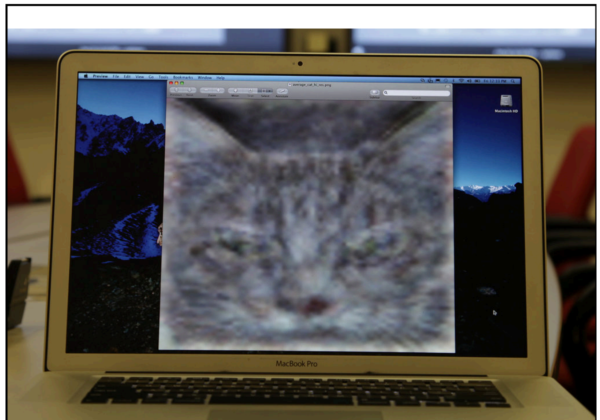
McCulloch and Pitts (1943) – introduced model of artificial neurons and suggested they could learn

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



http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?_r=0