BACKPROPAGATION

David Kauchak
CS158 – Fall 2023

1

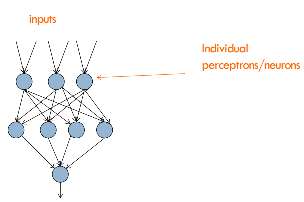
Admin

Assignment 7

Assignment 8 released on Monday. Start ASAP!

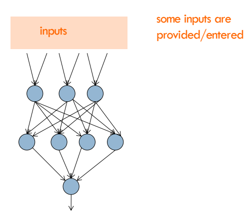
2

Neural network



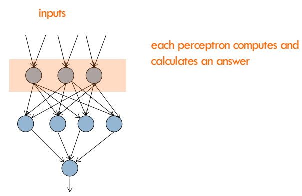
3

Neural network



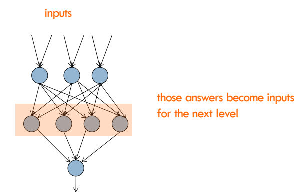
4

Neural network



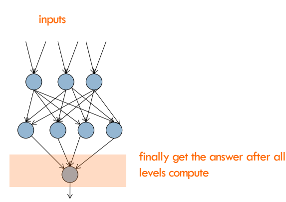
5

Neural network



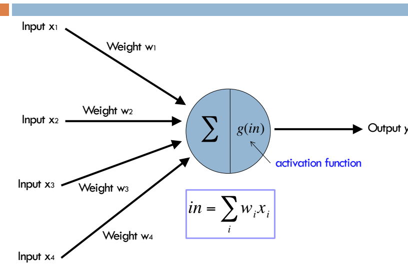
6

Neural network



7

A neuron/perceptron

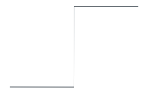


8

Activation functions

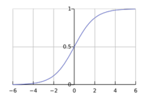
hard threshold:

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

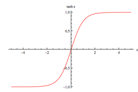


sigmoid

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

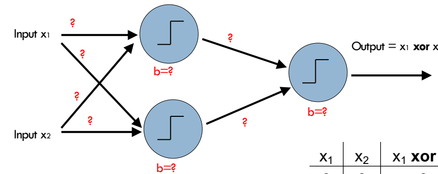


tanh x



9

Training



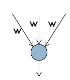
How do we learn the weights?

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

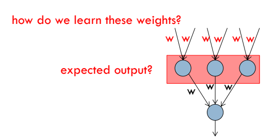
10

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



neural network

11

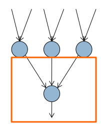
Backpropagation: intuition

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

12

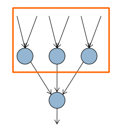
Backpropagation: intuition



We can calculate the actual error here

13

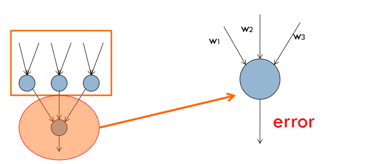
Backpropagation: intuition



Key idea: propagate the error back to this layer

14

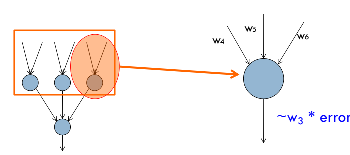
Backpropagation: intuition



error for node is ~ w_i * error

15

Backpropagation: intuition



Calculate as normal, but weight the error

16

Backpropagation: the details

Gradient descent method for learning weights by optimizing a loss function

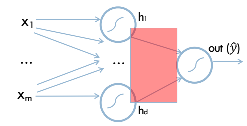
1. calculate output of all nodes
2. calculate the updates directly for the output layer
3. “backpropagate” errors through hidden layers

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

17

Backpropagation: the details

Notation:



m: features/inputs

d: hidden nodes

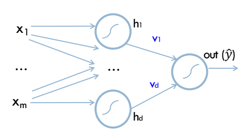
h_k: output from hidden node k

How many weights (ignore bias for now)?

18

Backpropagation: the details

Notation:



m: features/inputs

d: hidden nodes

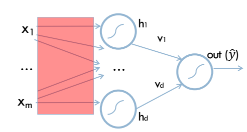
h_k: output from hidden nodes

d weights: denote v_k

19

Backpropagation: the details

Notation:



m: features/inputs

d: hidden nodes

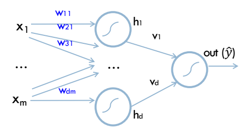
h_k: output from hidden nodes

How many weights?

20

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

21

Backpropagation: the details

Gradient descent method for learning weights by optimizing a loss function

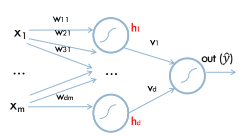
$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y - \hat{y})^2$$

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

22

Backpropagation: the details

1. Calculate outputs of all nodes

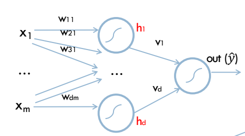


What are h_k in terms of *x* and *w*?

23

Backpropagation: the details

1. Calculate outputs of all nodes



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

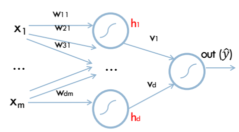
$$h_k = f(w_k \cdot x)$$

f is the activation function

24

Backpropagation: the details

1. Calculate outputs of all nodes



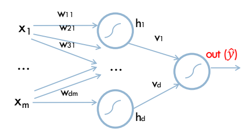
$$h_k = f(w_k \cdot x) = \frac{1}{1+e^{-w_k \cdot x}}$$

f is the activation function

25

Backpropagation: the details

1. Calculate outputs of all nodes

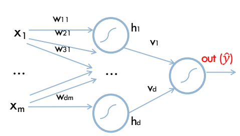


What is *out* in terms of *h* and *v*?

26

Backpropagation: the details

1. Calculate outputs of all nodes

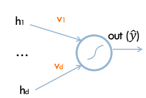


$$out = f(v \cdot h) = \frac{1}{1+e^{-v \cdot h}}$$

27

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. How?

28

Recall: derivative chain rule

$$\frac{d}{dx}(f(g(x)) = ?$$

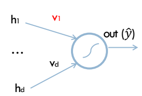
29

Recall: derivative chain rule

$$\frac{d}{dx}(f(g(x)) = f'\big(g(x)\big)\frac{d}{dx}g(x)$$

30

Output layer weights



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

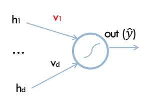
$$\frac{dloss}{dv_k} = \frac{d}{dv_k}\left(\frac{1}{2}(y-\hat{y})^2\right)$$

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

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

31

Output layer weights



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

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

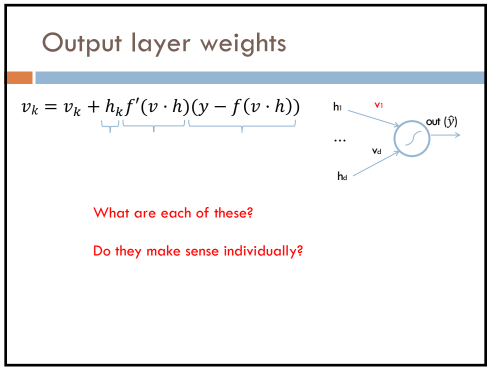
$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dv_k}v\cdot h$$

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

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

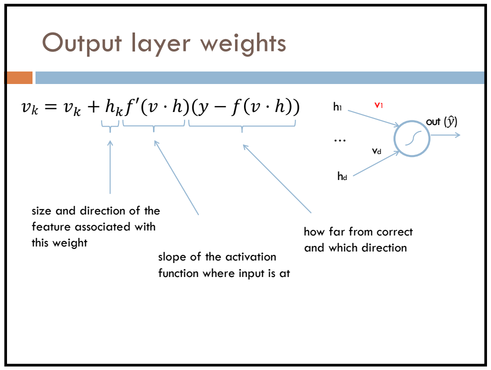
$$v_k = v_k + h_k f'(v\cdot h)(y-f(v\cdot h))$$

32



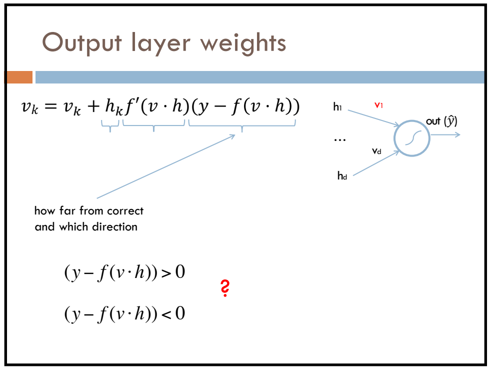
Output layer weights
$v_k = v_k + h_k f'(v \cdot h)(y - f(v \cdot h))$
h1
v1
...
vd
hd
out ($\hat{y}$)
What are each of these?
Do they make sense individually?

33



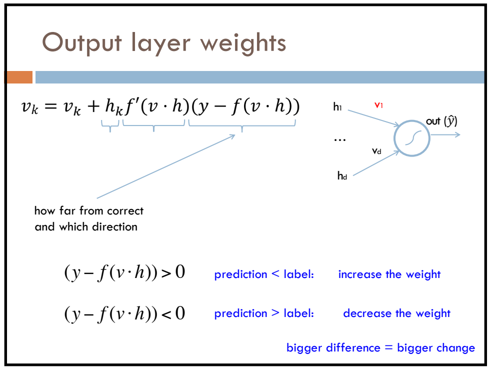
Output layer weights
$v_k = v_k + h_k f'(v \cdot h)(y - f(v \cdot h))$
h1
v1
...
vd
hd
out ($\hat{y}$)
size and direction of the feature associated with this weight
slope of the activation function where input is at
how far from correct and which direction

34



Output layer weights
$v_k = v_k + h_k f'(v \cdot h)(y - f(v \cdot h))$
h1
v1
...
vd
hd
out ($\hat{y}$)
how far from correct and which direction
$(y - f(v \cdot h)) > 0$
$(y - f(v \cdot h)) < 0$
?

35

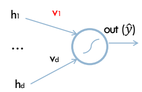


Output layer weights
$v_k = v_k + h_k f'(v \cdot h)(y - f(v \cdot h))$
h1
v1
...
vd
hd
out ($\hat{y}$)
how far from correct and which direction
$(y - f(v \cdot h)) > 0$ prediction < label: increase the weight
$(y - f(v \cdot h)) < 0$ prediction > label: decrease the weight
bigger difference = bigger change

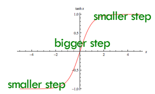
36

Output layer weights

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



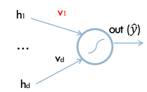
slope of the activation function where input is at



37

Output layer weights

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



size and direction of the feature associated with this weight

perceptron update:

$w_j = w_j + x_{ij} y_i$

gradient descent update:

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

38

Backpropagation: the details

Gradient descent method for learning weights by optimizing a loss function

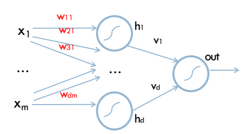
$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y - \hat{y})^2$$

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

39

Backpropagation

3. "backpropagate" errors through hidden layers

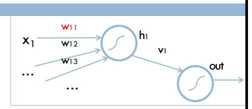


$$\text{argmin}_{w,v} \sum_x \frac{1}{2}(y - \hat{y})^2$$

Want to take a small step towards decreasing loss. How?

40

Hidden layer weights



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

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

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

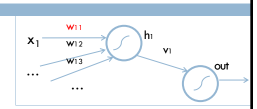
$$= -(y - f(v \cdot h))\frac{d}{dw_{kj}} f(v \cdot h)$$

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

chain rule

41

Hidden layer weights

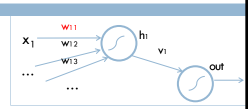


$$\frac{dloss}{dw_{kj}} = \frac{d}{dw_{kj}}\left(\frac{1}{2}(y-\hat{y})^2\right)$$

Remember: w_{kj} is the weight for hidden node k from input j

42

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 the other v_k components are not affected by w_{kj}

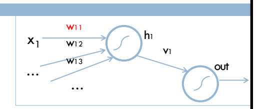
$$= -(y - f(v \cdot h)) f'(v \cdot h) v_k \frac{d}{dw_{kj}} h_k$$

v_k is a constant

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

43

Hidden layer weights



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

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

chain rule

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

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

44

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

What happened here?

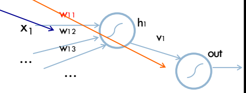
$$= -h_k f'(v\cdot h)(y-f(v\cdot h))$$

$$\frac{dloss}{dw_{kj}} = \frac{d}{dw_{kj}}\left(\frac{1}{2}(y-\hat{y})^2\right)$$
$$= \frac{d}{dw_{kj}}\left(\frac{1}{2}\left(y-f(v\cdot h)^2\right)\right)$$
$$= (y-f(v\cdot h))\frac{d}{dw_{kj}}(y-f(v\cdot h))$$
$$= -(y-f(v\cdot h))\frac{d}{dw_{kj}}f(v\cdot h)$$
$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dw_{kj}}v\cdot h$$
$$= -(y-f(v\cdot h))f'(v\cdot h)\frac{d}{dw_{kj}}v_k h_k$$
$$= -(y-f(v\cdot h))f'(v\cdot h)v_k\frac{d}{dw_{kj}}h_k$$
$$= -(y-f(v\cdot h))f'(v\cdot h)v_k\frac{d}{dw_{kj}}f(w_k\cdot x)$$
$$= -(y-f(v\cdot h))f'(v\cdot h)v_k f'(w_k\cdot x)\frac{d}{dw_{kj}}w_k\cdot x$$

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

45

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



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

46

Backpropagation

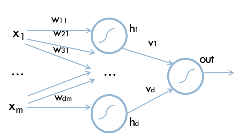
output layer

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

hidden layer

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

What's different?



48

Backpropagation

output layer

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

input

output activation slope

error

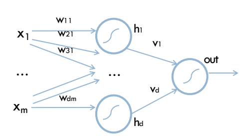
hidden layer

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

input

output activation slope

error



slope of wx

weight from hidden layer to output layer

49

Backpropagation

output layer

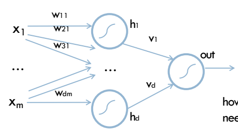
$= -h_k f'(v \cdot h)(y - f(v \cdot h))$

input — output activation slope — error

hidden layer

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

input — output activation slope — error



how much do we need to change

how much of the error came from this hidden node

50

Backpropgation generalization

output layer

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

⬇

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

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

derivative of input at node — error

51

Backpropgation generalization

output layer

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

⬇

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

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

hidden layer

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

⬇

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

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

Can we write this more succinctly?

52

Backpropgation generalization

output layer

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

⬇

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

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

hidden layer

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

⬇

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

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

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

53

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

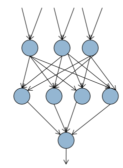
modified error of output layer

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

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

54

Backprop on multilayer networks



Anything different at this layer?

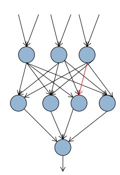
$w = w + input * \Delta_{current}$

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

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

55

Backprop on multilayer networks



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

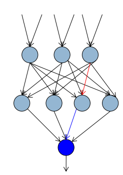
$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$

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

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

56

Backprop on multilayer networks



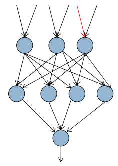
$w = w + input * \Delta_{current}$

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

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

57

Backprop on multilayer networks



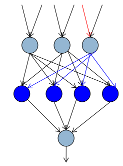
$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$

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

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

58

Backprop on multilayer networks

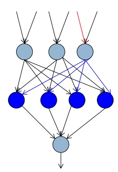


$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$

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

59

Backprop on multilayer networks



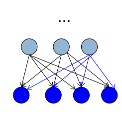
$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) \sum w_{output} \Delta_{output}$

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

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

60

Backprop on multilayer networks



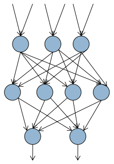
$w = w + input * \Delta_{current}$
$\Delta_{current} = f'(current\_input) \sum w_{output} \Delta_{output}$

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

61

Multiple output nodes



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

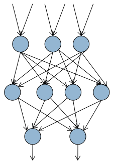
$$w = w + input * \Delta_{current}$$
$$\Delta_{current} = f'(current\_input) w_{output} \Delta_{output}$$

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

How does multiple outputs change things?

62

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?

63

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?

64

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

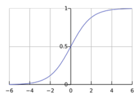
65

Activation function derivatives

sigmoid

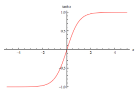
$$s(x) = \frac{1}{1+e^{-x}}$$

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



tanh

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



66

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

67

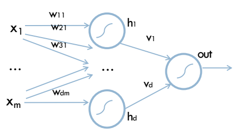
Backpropagation implementation

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

68

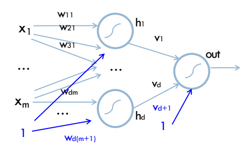
Handling bias



How should we learn the bias?

69

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

70

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?

71

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

72

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

…

73

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*

74

History of Neural Networks

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

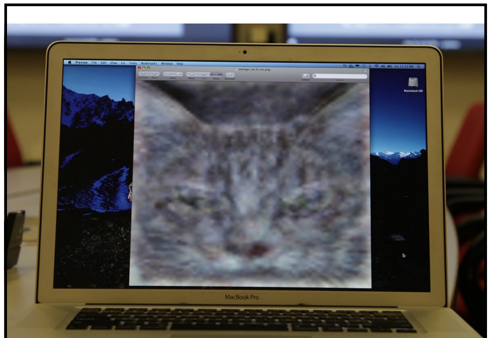
Hebb (1949) – Simple updating rule for learning

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

75



76

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

77