

**Class #20:**  
Training Neural Networks

---

Machine Learning (COMP 135): M. Allen, 01 April 20

1

## Review: Basic Neuron Model

Source: Russel & Norvig,  
*AI: A Modern Approach*  
(Prentice Hal, 2010)

- ▶ Each neuron has a set of **weights** on input connections coming into it
  - ▶ Inputs are numbers coming from other neurons, or from original input
- ▶ Neuron computes linear weighted sum of the inputs and passes that value through its **activation function, g**
- ▶ Output  $a$  is either passed along to other neurons, or is used as final output for the entire network

▶ Wednesday, 01 Apr. 2020
Machine Learning (COMP 135) 2

2

## Activation Functions Everywhere!

---

▶ Logistic	$f(x) = \frac{1}{1 + e^{-x}}$	$\frac{\delta f}{\delta x}(x) = f(x)(1 - f(x))$
▶ ReLU	$f(x) = \max(0, x)$	$\frac{\delta f}{\delta x}(x) = \{0, \text{undef}, 1\}$
▶ Softplus	$f(x) = \ln(1 + e^x)$	$\frac{\delta f}{\delta x}(x) = \frac{1}{1 + e^{-x}}$
▶ Hyperbolic Tangent	$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$	$\frac{\delta f}{\delta x}(x) = 1 - f(x)^2$
▶ Gaussian	$f(x) = e^{-\frac{x^2}{2}}$	$\frac{\delta f}{\delta x}(x) = -x f(x)$

▶ Wednesday, 01 Apr. 2020
Machine Learning (COMP 135) 3

3

## Single-Layer Perceptron Networks

- ▶ In such a network, input features (boxes) are connected **directly** to a single layer of output neurons, each of which represents one possible data classification
- ▶ Each output is the result of the activation function on the weighted input features, e.g.:
 
$$in_3 = w_{0,3} + w_{1,3} x_1 + w_{2,3} x_2$$

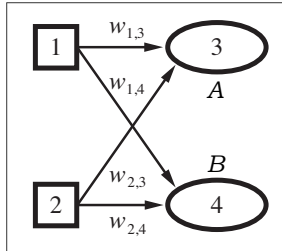
$$a_3 = g(in_3) = \frac{1}{1 + e^{-in_3}}$$

Remember: in these slides, we will use the logistic activation function as an example, but other options are possible.

▶ Wednesday, 01 Apr. 2020
Machine Learning (COMP 135) 4

4

## Gradient Descent in Single-Layer Networks



- ▶ A small network like this has two output neurons, corresponding to two classes of data, *A* and *B*
- ▶ If the input *is* of *A* type, and the network is working properly:
  - ▶ The value output by 3 should be (close to) 1
  - ▶ The value output by 4 should be (close to) 0
- ▶ If input *is not* of *A* type, and the network is working properly:
  - ▶ The value output by 3 should be (close to) 0
  - ▶ The value output by 4 should be (close to) 1

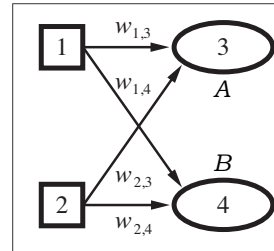
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

5

5

## Gradient Descent in Single-Layer Networks



- ▶ For each of the output neurons, we can measure its **error**, comparing the output we get to the classification, *y*, that we want to see
- ▶ E.g., if neuron 3 represents class *A*, then if the input *is* of *A* type:

$$\begin{aligned} Err_3 &= y - g(in_3) \\ &= 1 - g(in_3) \end{aligned}$$

- ▶ If input *is not* of *A* type:

$$\begin{aligned} Err_3 &= y - g(in_3) \\ &= 0 - g(in_3) \end{aligned}$$

Remember: neuron for class *B* will also calculate its own output and error.

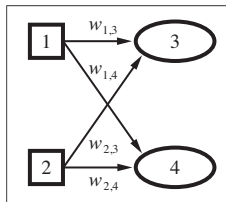
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

6

6

## Gradient Descent in Single-Layer Networks



- ▶ Once we know the error, we can do a **gradient-based** update to get new weights for each neuron:

$$w_i \leftarrow w_i + \alpha Err_j \times g'(in_j) \times x_i$$

- ▶ Where we have:

- $x_i$ : the value of the input to that connection
- $Err_j$ : the error on the output (if any)
- $g'(in_j)$ : the derivative of the activation function
- $\alpha$ : convergence control parameter

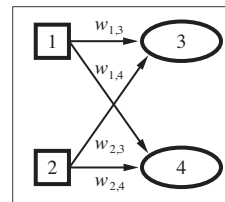
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

7

7

## Gradient Descent in Single-Layer Networks



- ▶ Once we know the error, we can do a **gradient-based** update to get new weights for each neuron:

$$w_i \leftarrow w_i + \alpha Err_j \times g'(in_j) \times x_i$$

- ▶ E.g., neuron 3 and logistic function:

$$\begin{aligned} w_{0,3} &\leftarrow w_{0,3} + \alpha (y - g(in_3)) \times g(in_3)(1 - g(in_3)) \\ w_{1,3} &\leftarrow w_{1,3} + \alpha (y - g(in_3)) \times g(in_3)(1 - g(in_3)) \times x_1 \\ w_{2,3} &\leftarrow w_{2,3} + \alpha (y - g(in_3)) \times g(in_3)(1 - g(in_3)) \times x_2 \end{aligned}$$

- ▶ As in linear regression and other methods, we continue making these updates until the weights **converge**

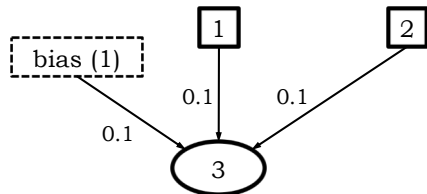
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

8

8

## Gradient Descent: Details



- Suppose we initialize neuron 3 with weight 0.1 on every connection, and we provide it with training example:

$$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$$

Inputs (1 & 2)

Outputs (3 & 4)

Wednesday, 01 Apr. 2020

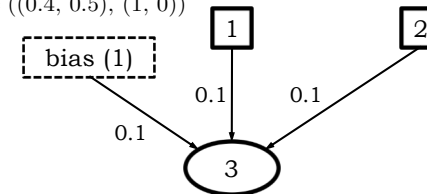
Machine Learning (COMP 135)

9

9

## Gradient Descent: Details

$$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$$



- Given these inputs, and these weights, the neuron outputs:

$$in_3 = 0.1 + (0.1 \times 0.4) + (0.1 \times 0.5) = 0.19$$

$$g(in_3) = \frac{1}{1 + e^{-0.19}} \approx 0.5474$$

Wednesday, 01 Apr. 2020

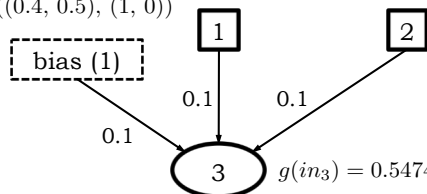
Machine Learning (COMP 135)

10

10

## Gradient Descent: Details

$$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$$



- We can now calculate error and the derivative term:

$$Err_3 = (1 - g(in_3)) = (1 - 0.5474) = 0.4526$$

$$g'(in_3) = g(in_3)(1 - g(in_3)) = (0.5474 \times 0.4526) \approx 0.2478$$

Wednesday, 01 Apr. 2020

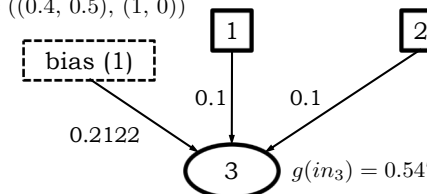
Machine Learning (COMP 135)

11

11

## Gradient Descent: Details

$$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$$



- Finally, we update each weight and get (assuming that for now we have  $\alpha = 1$ ):

$$w_0 \leftarrow w_0 + \alpha Err_3 \times g'(in_3) \times x_0$$

$$w_0 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 1) \approx 0.2122$$

Wednesday, 01 Apr. 2020

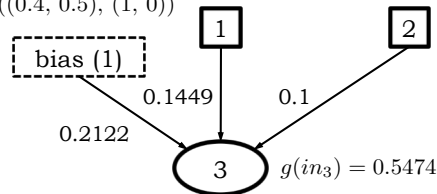
Machine Learning (COMP 135)

12

12

## Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



- ▶ Finally, we update each weight and get (assuming that for now we have  $\alpha = 1$ ):

$$w_1 \leftarrow w_1 + \alpha \text{Err}_3 \times g'(in_3) \times x_1$$

$$w_1 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 0.4) \approx 0.1449$$

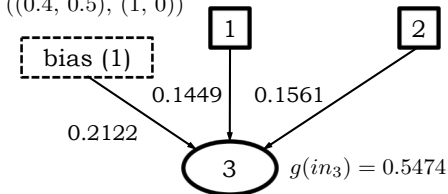
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 13

13

## Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



- ▶ Finally, we update each weight and get (assuming that for now we have  $\alpha = 1$ ):

$$w_2 \leftarrow w_2 + \alpha \text{Err}_3 \times g'(in_3) \times x_2$$

$$w_2 \leftarrow 0.1 + (0.4526 \times 0.2478 \times 0.5) \approx 0.1561$$

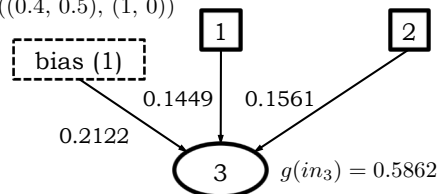
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 14

14

## Gradient Descent: Details

$((x_1, x_2), \mathbf{y}) = ((0.4, 0.5), (1, 0))$



- ▶ The result is that each connection has been **strengthened**, increasing each weight in proportion to both error made and the input to that connection
- ▶ As a result, the value computed by the neuron increases as well (reducing, but not eliminating the amount of error the neuron makes):

$$in_3 = 0.2122 + (0.1449 \times 0.4) + (0.1561 \times 0.5) = 0.34821$$

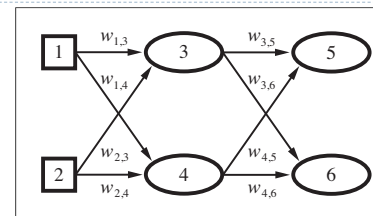
$$g(in_3) = \frac{1}{1 + e^{-0.34821}} \approx 0.5862$$

▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 15

15

## Multi-Layer Feed-Forward Neural Networks



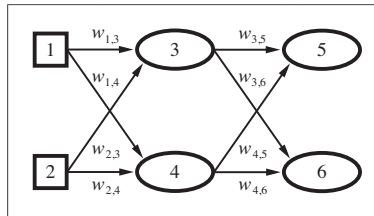
- ▶ It was known from the beginning that a single-layer network could not represent all possible functions
- ▶ For more complex functions, we will add one or more **hidden layers** of neurons between input and output layers
- ▶ In a **feed-forward** network, each layer is connected only to next layer in the order (inputs  $\rightarrow$  outputs)

▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 16

16

## Multi-Layer Feed-Forward Neural Networks



- ▶ Output neurons process inputs **indirectly**, via inputs that come from the hidden layer “above” them, e.g.:

$$\begin{aligned}
 a_5 &= g(in_5) = g(w_{0,5} + w_{3,5} a_3 + w_{4,5} a_4) \\
 &= g(w_{0,5} + w_{3,5} g(w_{0,3} + w_{1,3} x_1 + w_{2,3} x_2) \\
 &\quad + w_{4,5} g(w_{0,4} + w_{1,4} x_1 + w_{2,4} x_2))
 \end{aligned}$$

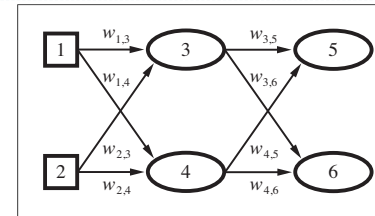
▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

17

17

## Learning in Neural Networks



- ▶ A neural network can learn a classification function by adjusting its weights to compute different responses
- ▶ This process is another version of **gradient descent**: the algorithm moves through a complex space of partial solutions, always seeking to **minimize** overall error

▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

18

18

## Back-Propagation (Hinton, et al.)

Source: Russel & Norvig,  
AI: A Modern Approach  
(Prentice Hal, 2010)

**function** BACK-PROP-LEARNING(*examples, network*) returns a neural network  
**inputs:** *examples*, a set of examples, each with input vector **x** and output vector **y**  
*network*, a multilayer network with *L* layers, weights  $w_{i,j}$ , activation function *g*  
**local variables:**  $\Delta$ , a vector of errors, indexed by network node

```

repeat
  for each weight  $w_{i,j}$  in network do
     $w_{i,j} \leftarrow$  a small random number
  for each example  $(\mathbf{x}, \mathbf{y})$  in examples do
    /* Propagate the inputs forward to compute the outputs */
    for each node  $i$  in the input layer do
       $a_i \leftarrow x_i$ 
    for  $\ell = 2$  to  $L$  do
      for each node  $j$  in layer  $\ell$  do
         $in_j \leftarrow \sum_i w_{i,j} a_i$ 
         $a_j \leftarrow g(in_j)$ 
    /* Propagate deltas backward from output layer to input layer */
    for each node  $j$  in the output layer do
       $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$ 
    for  $\ell = L - 1$  to 1 do
      for each node  $i$  in layer  $\ell$  do
         $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 
    /* Update every weight in network using deltas */
    for each weight  $w_{i,j}$  in network do
       $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ 
until some stopping criterion is satisfied
return network
    
```

Initial random weights

Loop over all training examples, generating the output, and then updating weights based on error

Stop when weights converge or error is minimized

▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

19

19

## Propagating Output Values Forward

/\* Propagate the inputs forward to compute the outputs \*/

for each node  $i$  in the input layer do

$$a_i \leftarrow x_i$$

for  $\ell = 2$  to  $L$  do

for each node  $j$  in layer  $\ell$  do

$$in_j \leftarrow \sum_i w_{i,j} a_i$$

$$a_j \leftarrow g(in_j)$$

At first (“top”) layer, each neuron input is set to the corresponding feature value

Go down layer-by-layer, calculating weighted input sums for each neuron, and computing output function  $g$

▶ Wednesday, 01 Apr. 2020

Machine Learning (COMP 135)

20

20

## Propagating Error Backward

```
/* Propagate deltas backward from output layer to input layer */
for each node  $j$  in the output layer do
   $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$ 
for  $\ell = L - 1$  to 1 do
  for each node  $i$  in layer  $\ell$  do
     $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 
/* Update every weight in network using deltas */
for each weight  $w_{i,j}$  in network do
   $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ 
```

After all the delta values are computed,  
update weights on every node in the  
network

At output ("bottom")  
layer, each delta-value is  
set to the error on that  
neuron, multiplied by the  
derivative of function  $g$

Go bottom-up and set  
delta to derivative value  
multiplied by sum of  
deltas at the next layer  
down (weighting each  
such value appropriately)

Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 21

21

## This Week & Next

- ▶ **Topics: Neural Networks**
- ▶ **Homework 04 (last one!)**
  - ▶ Out Monday, 30 March
  - ▶ Due Monday, 13 April, 5:00 PM Eastern
- ▶ **Office Hours:**
  - ▶ Virtual office hours with Zoom links can be found on the class Piazza and Canvas sites

Wednesday, 01 Apr. 2020

Machine Learning (COMP 135) 22

22