EECE-6822 Machine Learning

Neural Networks

Cris Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

1

PART 1
Neural Networks

2

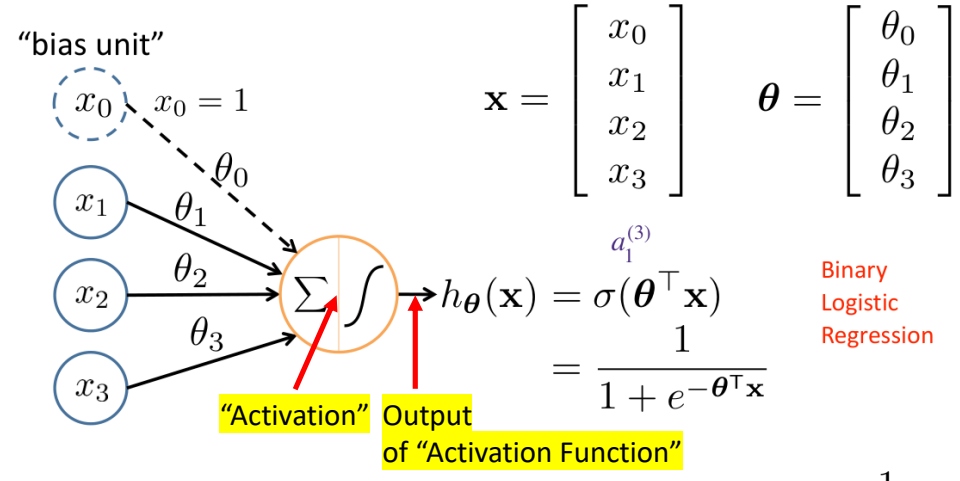
From Linear to Nonlinear

- One of the major weaknesses of linear models (perceptron and regularized linear models), is that they are linear!
- They are unable to learn arbitrary decision boundaries.
- One approach to do better is to chain together a collection of Perceptrons to build more complex **Neural Networks (NN)**.
- Neural networks
  - Takes an input vector of p variables X = (X1, X2, . . . , Xp) and builds a nonlinear function f(X) to predict the response Y.
  - Identify good basis transformations for our data.
  - Extraordinarily flexible class of models.
  - Used to solve a variety of different problem types.

3

3

Single Neuron = Linear regression + Activation function (neuron a.k.a. "unit" or "node")

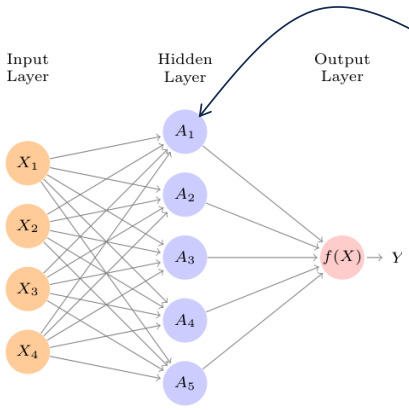


Sigmoid (logistic) activation function: $g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$

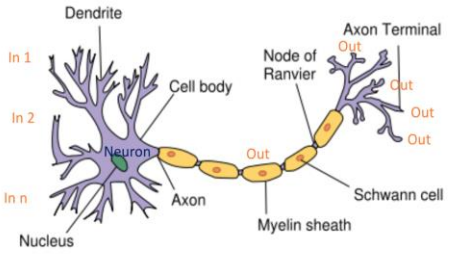
4

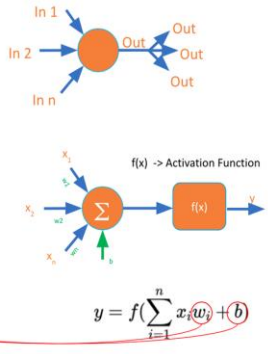
4

Neural Network



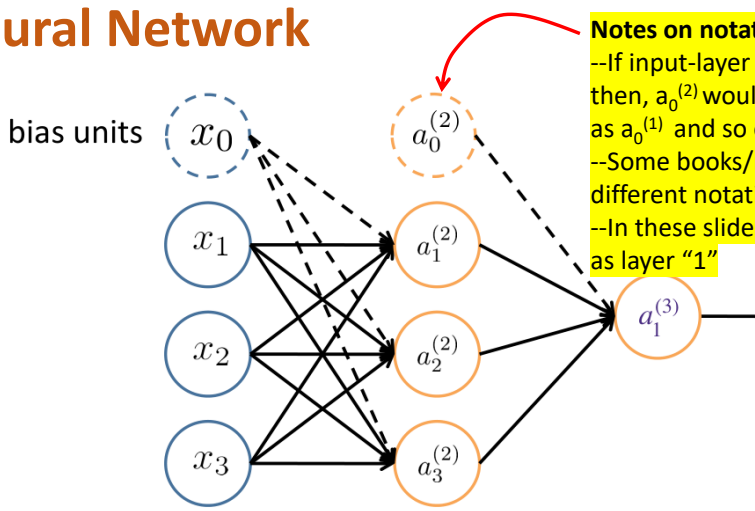
Neuron (Perceptron)





Neural Network

Neural Network

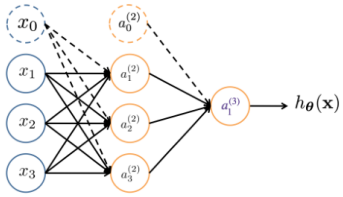


Notes on notations:
--If input-layer is not "counted", then, $a_0^{(2)}$ would be denoted as $a_0^{(1)}$ and so on…
--Some books/resources use different notations like that…
--In these slides, we "count" input-layer as layer "1"

Layer 1 (Input Layer)

Layer 2 (Hidden Layer)

Layer 3 (Output Layer)



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix stores parameters from layer j to layer $j+1$

Notes on notations:
Some books/resources use different notations:
--"activation" $a_1^{(2)}$ would be the weighted summation
--output of "activation function" g(.) would be called neuron "output" denoted as $z_1^{(2)}$ = g("activation")
--parameters θ are also commonly denoted as ω (for "weights")

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer j+1, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

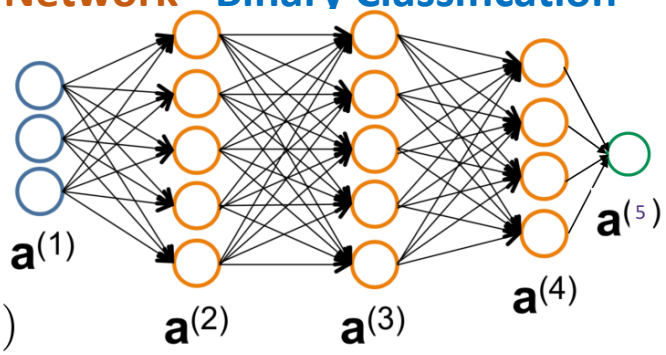
7

7

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$
$$a^{(2)} = g(\Theta^{(1)}a^{(1)})$$
$$\vdots$$
$$a^{(l+1)} = g(\Theta^{(l)}a^{(l)})$$
$$\vdots$$
$$\hat{y} = g(\Theta^{(L)}a^{(L)})$$

L here is number of Layers



L here is Loss

$$L(y, \hat{y}) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$

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

Binary Logistic Regression

8

Multi-layer Neural Network - Activation functions

Activation functions for nodes at intermediate layers include sigmoid, tanh, softplus, **ReLU**…

$$a^{(1)} = x$$

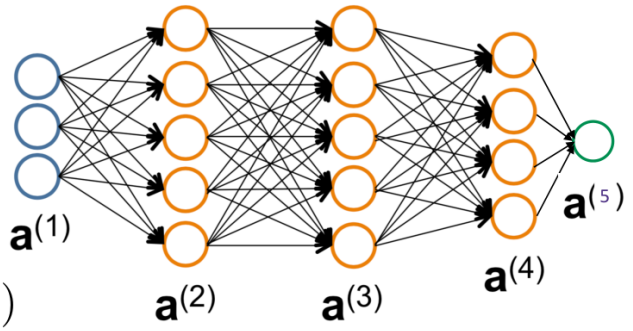
$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

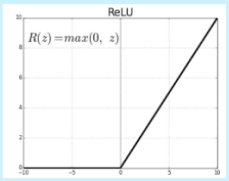
$$\vdots$$

$$\hat{y} = \sigma(\Theta^{(L)} a^{(L)})$$



ReLU (Rectified Linear Unit) is most common:

$$g(z) = \max\{0, z\}$$



9

9

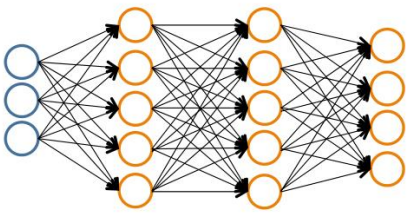
Multiple Output Units: One vs. Rest


Pedestrian


Car


Motorcycle


Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

Multi-class Logistic Regression

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

10

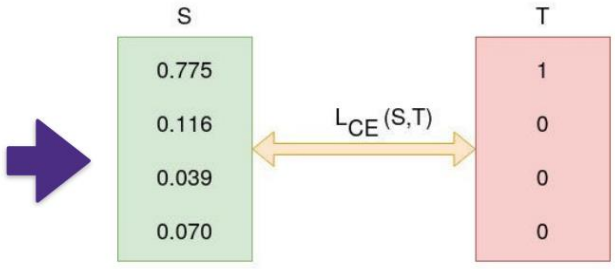
10

Multi-class classification: Softmax + Binary Cross-Entropy (BCE)

Recall: softmax function

$$P\left(y = c_j \middle| x\right) = \frac{e^{w_j^\top x}}{\sum_{j'} e^{w_{j'}^\top x}}$$

Network's final output layer is a **softmax layer**, applies softmax activation function



$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$

Ground truth label

Compare predicted values to ground truth using binary **cross-entropy**

$$H(y, \hat{y}) = -\sum_{j=1}^{K} y_j \log(\hat{y}_j)$$

11

Multi-layer Neural Network - Regression

$$a^{(1)} = x$$

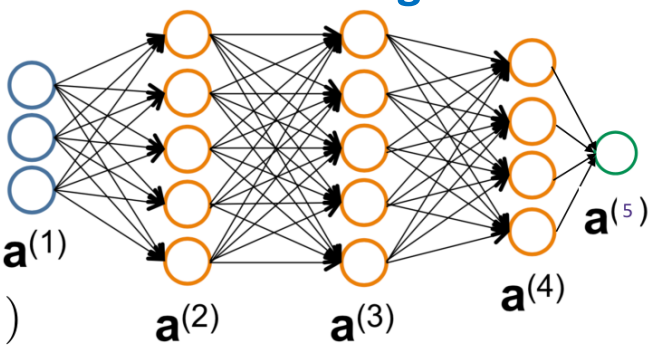
$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\hat{y} = \Theta^{(L)} a^{(L)}$$



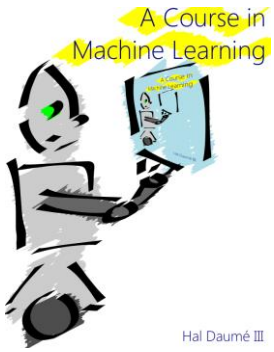
Regression: apply squared error loss to output

$$L(y, \hat{y}) = (y - \hat{y})^2$$

12

Neural Networks are arbitrary function approximators

Theorem 10 (Two-Layer Networks are Universal Function Approximators). *Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximate F arbitrarily well. Namely, for all $\boldsymbol{x}$ in the domain of F, $\left|F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x})\right| < \epsilon$.*



13

Training Neural Networks

14

14

Perceptron Learning Rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(y - h(\mathbf{x}))\mathbf{x}$$

Equivalent to the intuitive rules:

- If output is correct, don't change the weights
- If output is low ($h(\mathbf{x}) = 0$, $y = 1$), increment weights for all the inputs which are 1
- If output is high ($h(\mathbf{x}) = 1$, $y = 0$), decrement weights for all inputs which are 1

Perceptron Convergence Theorem:

- If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge [Minicksy & Papert, 1969]

15

15

Batch Perceptron

Given training data $\left\{(\boldsymbol{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
  Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
  for $i = 1 \ldots n$, do
    if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
      $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)}\boldsymbol{x}^{(i)}$
  $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}/n$ // compute average update
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- Simplest case: α = 1 and don't normalize, yields the fixed increment perceptron
- Each increment of outer loop is called an **epoch**

16

16

Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

17

17

Cost Function

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Notes on notations:
--Cost function "J" is loss function "L"

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$
$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

8

18

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right] + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_\Theta J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

- But, tends to work well in practice

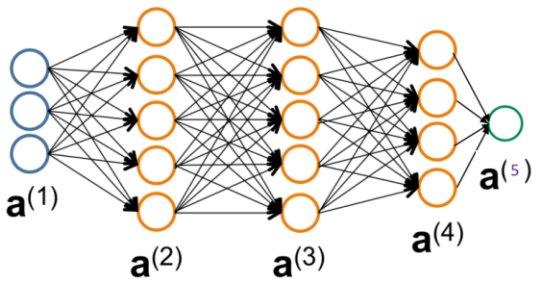
Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

19

Gradient Descent (GD)

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y,\hat{y}) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$
$$g(z) = \frac{1}{1+e^{-z}}$$

$y_i \in \{0,1\}$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y,\hat{y}) \quad \forall l$

Need to calculate

20

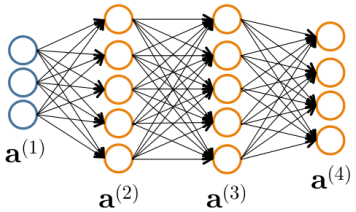
20

Forward Propagation

- Given one labeled training instance $(\mathbf{x}, y)$:

Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$



Notes on notations:
--Here $z^{(2)}$ is new notation to represent the weighted summation (i.e., linear regression)
--"Activation" $a^{(2)}$ denotes the output of "activation function"
--Differences in notations are not that important; conceptually, they describe the same thing!

21

21

Forward Propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$a^{(l)} = g\left(z^{(l)}\right)$$
$$z^{(l+1)} = \Theta^{(l)}a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

Backward Propagation:

$$L(y, \hat{y}) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$
$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}}$$

Need to compute partial derivatives (used by optimization like GD):

$$\frac{\partial L(y,\hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

For Output layer:

$$\delta^{(L+1)} = y - a^{(L+1)}$$

For Hidden layers:

$$\delta_i^{(l)} = a_i^{(l)}(1-a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

BackProp gives us derivatives needed by GD:

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y,\hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

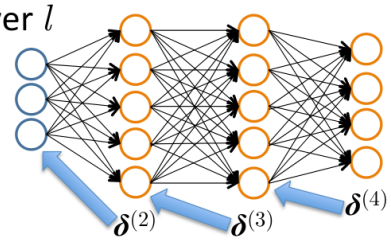
22

22

Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

(#layers L = 4)



Element-wise product .*

Backpropagation

- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^{\mathsf{T}}\boldsymbol{\delta}^{(4)}$.* $g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^{\mathsf{T}}\boldsymbol{\delta}^{(3)}$.* $g'(\mathbf{z}^{(2)})$
- (No $\boldsymbol{\delta}^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)}$.* $(1-\mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)}$.* $(1-\mathbf{a}^{(2)})$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignoring λ; if $\lambda = 0$)

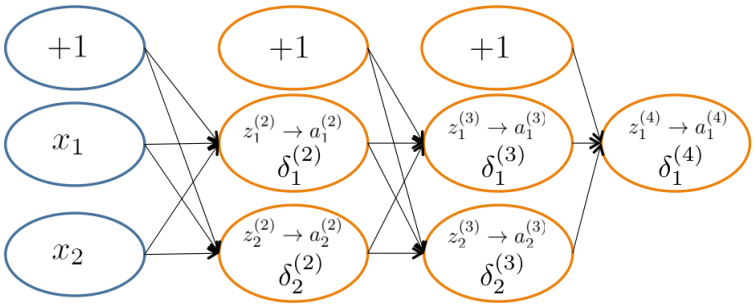
23

Backpropagation Intuition

- Each hidden node j is "responsible" for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects
- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node
- Then, the "blame" is propagated back to provide the error values for the hidden layer

24

Backpropagation Intuition



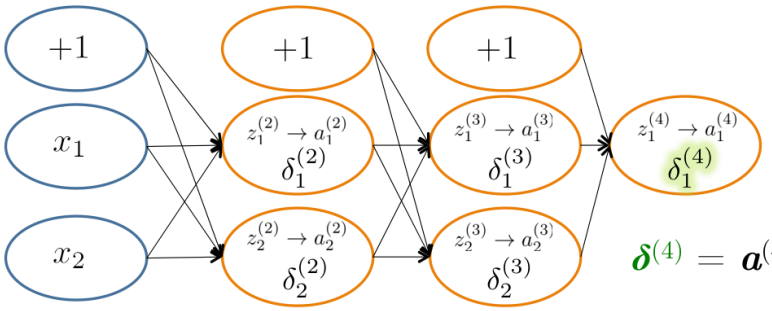
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$ 25

25

Backpropagation Intuition



$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$

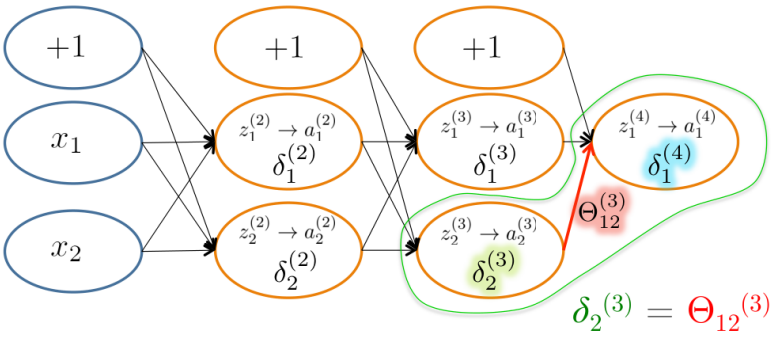
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$ 26

26

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

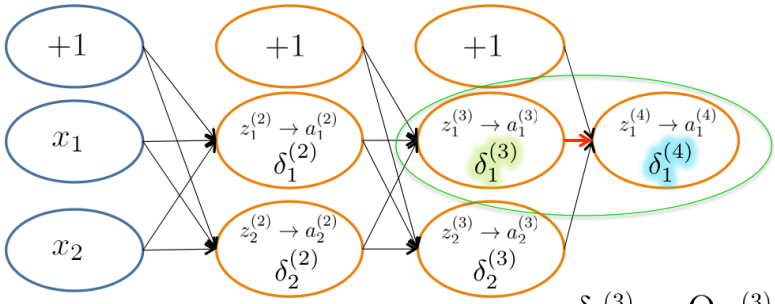
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

27

27

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

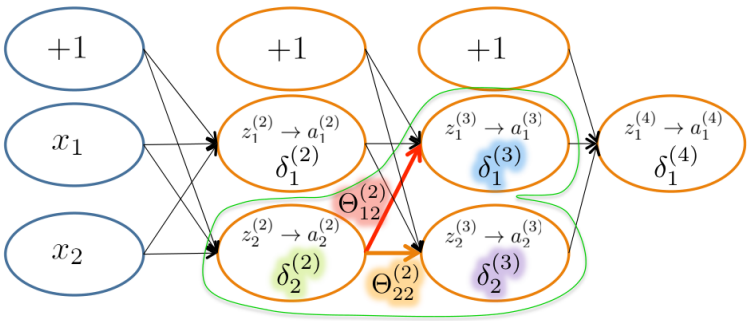
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

28

28

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

29

29

Training a Neural Network via Gradient Descent with BackProp

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

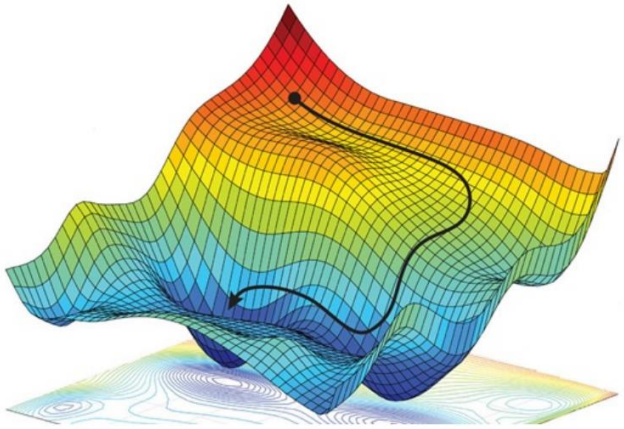
Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$

Until weights converge or max #epochs is reached

Backpropagation

30

Training issues:
Neural networks are non-convex



- Optimization is hard
- Local minima can be a problem
- Rich history of work on getting these things to train effectively

31

31

Training issues:
Neural networks are non-convex

- For large networks, **gradients** can **blow up** or **go to zero**. This can be helped by **batchnorm** or ResNet architecture

- **Hyperparameters** like **Stepsize**, **batchsize**, **momentum** all have large impact on optimizing the training error *and* generalization performance

- Fancier alternatives to SGD (Adagrad, Adam, LAMB, etc.) can significantly improve training

-Overfitting is common and not undesirable: typical to achieve 100% training accuracy even if test accuracy is just 80%

- **Overparameterization:** Making the network *bigger* may make training *faster* and it might generalize *better!*

32

32

Training issues

- **Training is too slow:**
  - Use larger step sizes, develop step size reduction schedule
  - Use GPU resources
  - Change batch size
  - Use momentum and more exotic optimizers (e.g., Adam)
  - Apply batch normalization
  - Make network larger or smaller (# layers, # filters per layer, etc.)
- **Test accuracy is low**
  - Try modifying all of the above, plus changing other hyperparameters
- https://playground.tensorflow.org/
- http://yann.lecun.com/exdb/lenet/

33

33

PART 2

Code Time!

34

34

Code Time

- See demonstration and discussion in class.
- See also links in the “Code Examples” for this Lecture.

35

35

Conclusion

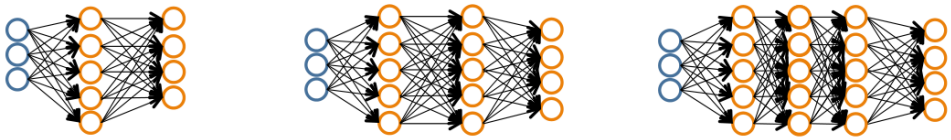
Takeaways

36

36

Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

37

37

Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(\mathbf{x}_i)$ for any instance $\mathbf{x}_i$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ computed using backpropagation vs. the numerical gradient estimate.
   - Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

38

38

Gradient Descent

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \hat{y}) \quad \forall l$

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation
2. Convenient libraries
3. GPU support

39

39

References and Credits

Many of the teaching materials for this course have been adapted from various sources. We are very grateful and thank the following professors, researchers, and practitioners for sharing their teaching materials (in no particular order):

- Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. https://amlbook.com/slides.html
- Ethem Alpaydin. https://www.cmpe.boun.edu.tr/~ethem/i2ml3e/
- Natasha Jaques. https://courses.cs.washington.edu/courses/cse446/25sp/
- Lyle Ungar. https://alliance.seas.upenn.edu/~cis520/dynamic/2022/wiki/index.php?n=Lectures.Lectures
- Aurelien Geron. https://github.com/ageron/handson-ml3
- Sebastian Raschka. https://github.com/rasbt/machine-learning-book
- Trevor Hastie. https://www.statlearning.com/resources-python
- Andrew Ng. https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU
- Richard Povineli. https://www.richard.povinelli.org/teaching
- … and many others.

40

40