Neural Networks: Learning (Week 5)

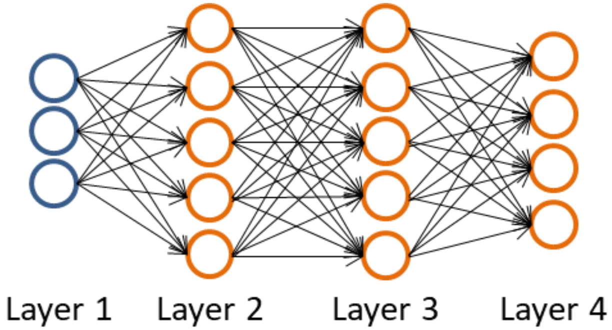
Coursera – Machine Learning

3 Blue, 1 Brown – Intuition of Backpropagation

NVIDIA – Deep Networks

Cost Function

- L is number of layers, s_l is the number of neurons in layer l (not bias)
- *m* is number of training examples: (x, y)
- *K* classes
- $(h_\Theta(x))_i = i^{th}$ output



$$J(\Theta) = -\frac{1}{m}\left[\underbrace{\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)}_{\text{Costs associated with each neuron}}\right] + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}_{\text{Regularization}}$$

Recall Logistic Regression Cost Function:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Backpropagation Algorithm

- Minimization of the cost function to find optimal parameters theta
  - like gradient descent for logistic regression
- For every node *j*, compute the error $\delta_j^{(l)}$, and recall: $a^{(4)} = h_\Theta(x)$
- For the last layer (*L*):

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

- For hidden layers:

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \qquad \delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Backpropagation continued

- Compute the change: $\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
- Then the partial derivatives terms: $\boxed{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}}$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \text{ if } j = 0$$

Algorithm

For training example t =1 to m:

- Set $a^{(1)} := x^{(t)}$
- Perform forward propagation to compute $a^{(l)}$ for l=2,3,…,L
- Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)})\ .*\ a^{(l)}\ .*\ (1 - a^{(l)})$
- $\Delta^{(l)}_{i,j} := \Delta^{(l)}_{i,j} + a^{(l)}_j \delta^{(l+1)}_i$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
- $D^{(l)}_{i,j} := \dfrac{1}{m}\left(\Delta^{(l)}_{i,j} + \lambda\Theta^{(l)}_{i,j}\right)$ If j≠0
- $D^{(l)}_{i,j} := \dfrac{1}{m}\Delta^{(l)}_{i,j}$ If j=0

Implementation Note: Random Initialization

- When initializing the parameters theta, DO NOT initialize all parameters to zero
  - All nodes will update to the same value repeatedly
    - Same weights, same deltas, same change

- Instead, initialize to (small) random values in a range centered on zero

Putting it together

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$
3. Compute cost function
4. Implement backpropagation to compute partial derivatives
5. (DEBUG ONLY) Use gradient checking to confirm backpropagation
6. Use gradient descent (or other optimization function) to minimize cost function with the weights in theta

For more details, refer to notes:
https://github.com/vkosuri/CourseraMachineLearning/blob/master/home/week-5/lectures/notes.pdf

3 Blue, 1 Brown – Intuition of Backpropagation

Backpropagation Intuition

- This video from 3Blue1Brown explains better than I could:

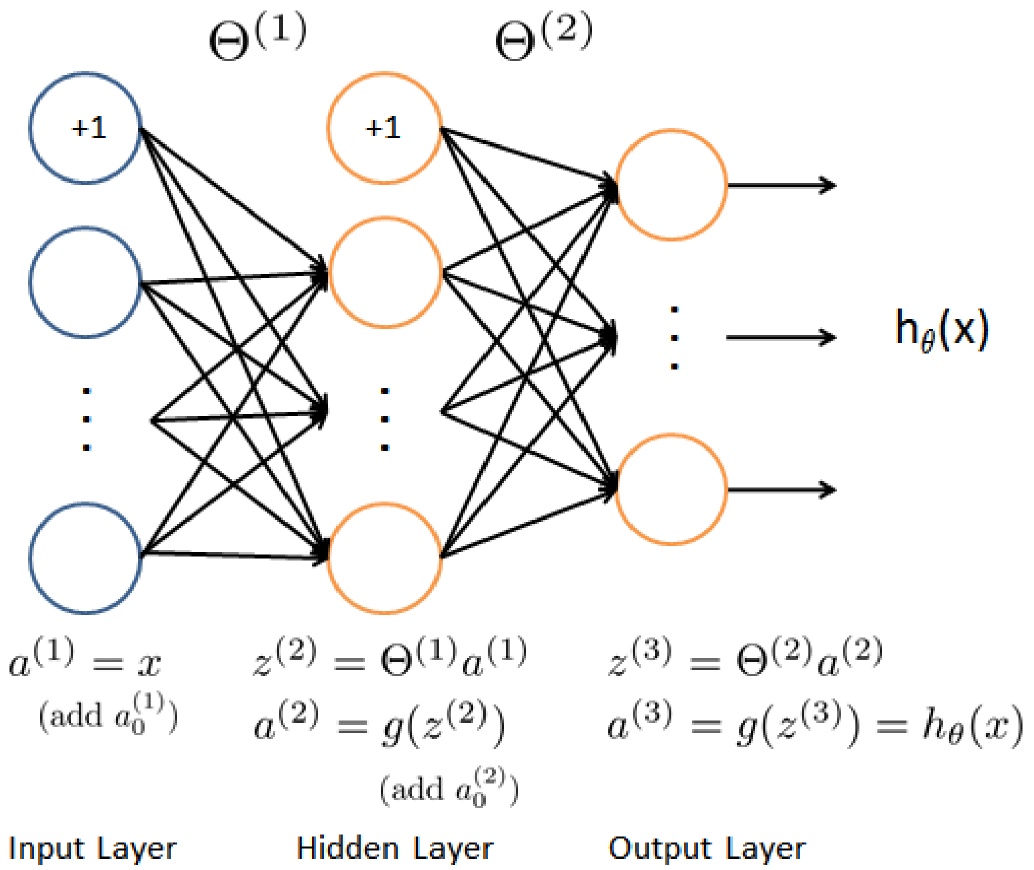
https://www.youtube.com/watch?v=Ilg3gGewQ5U

MNINTS Data Set Exercise

1. Display sample from dataset (100/5000 handwritten numerals)

2. Load provided parameter weights for NN with 400+1 input units, 10 output units and 25+1 hidden layer units

3. Implement Regularized Cost Function



MNINTS Data Set Exercise

4. Implement Backpropagation Loop
    - calculate the gradient
        - implement Sigmoid gradient function first to help
    - check gradients to confirm
    - add regularization
    - use optimizer (ex fmincg) to minimize cost

5. Initialize parameters (theta) with random weights
    - Critical – look at result without

6. Visualize the Hidden Layer

MNINTS Data Set Exercise

Backpropagation Loop:

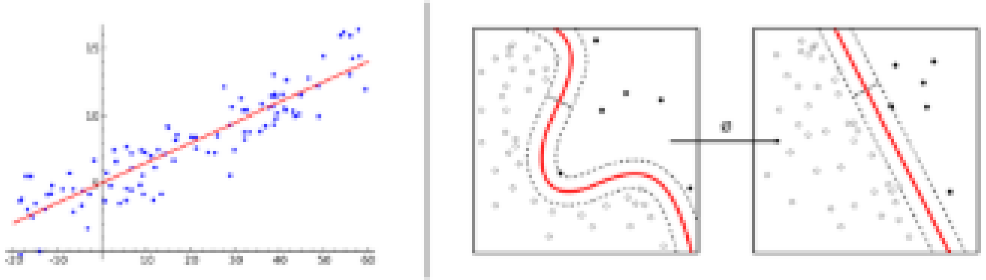
1. Set input to the current training ex and perform feedforward pass to compute activations
2. Calculate output layer's deltas using true label
3. Calculate hidden layer's deltas using gradient
4. Accumulate gradient of layers together
5. Normalize to get the NN Cost function's gradient

Difference in Workflow

Classic Machine Learning [1990 : now]

Input → Hand Designed Features → Model / Mapping → Output

Examples [Regression and SVMs]

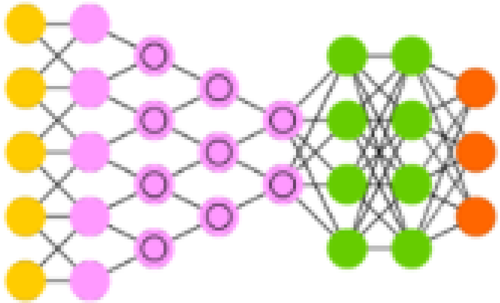


Deep/End-to-End Learning [2012 : now]

Input → Simple Features → Complex Features → Model/ Mapping → Output

Let algorithm selects features for us!

Example [Conv Net]



NVIDIA Deep Neural Networks Training

- DIGITS is a tool to train deep neural networks from NVIDIA (Deep Learning GPU Training System) that makes it easy to get started
  - Select Dataset
  - Select DNN Model
  - Train
  - Test
- Works with NVIDIA Caffe and Tensorflow frameworks
- We can use pre-trained networks for our tasks instead of starting from scratch
- Access to multi-GPU systems over the cloud with safety built in – can't break it!

More Information: https://developer.nvidia.com/digits

GPU Exercise 5: Object Detection

- Open DIGITS
  - Dogs and Cats Data Set
    - Clone Job – look at all settings available
  - AlexNet – Customize – Visualize
    - Data must flow
    - Math Matters
      - Replace fully connected layers with a convolution layers -> accept any size image
- In Jupyter
  - change directory, improvement by looking at whole image
- If interest/time, can look at Detect Net/COCO dataset

More Information: https://www.nvidia.com/en-us/deep-learning-ai/education/