Neural Networks and Deep Learning

A quick overview of Coursera's 1st specialization course

George Zoto
gezotos@gmail.com

The 1st course of the Deep Learning Specialization

Neural Networks and Deep Learning

deeplearning.ai

About this Course
If you want to break into cutting-edge AI, this course will help you do so. Deep learning engineers are highly sought after, and mastering deep learning will give you numerous new career opportunities. Deep learning is also a new "superpower" that will let you build AI systems that just weren't possible a few years ago.

In this course, you will learn the foundations of deep learning. When you finish this class, you will:
- Understand the major technology trends driving Deep Learning
- Be able to build, train and apply fully connected deep neural networks
- Know how to implement efficient (vectorized) neural networks
- Understand the key parameters in a neural network's architecture

This course also teaches you how Deep Learning actually works, rather than presenting only a cursory or surface-level description. So after completing it, you will be able to apply deep learning to a your own applications. If you are looking for a job in AI, after this course you will also be able to answer basic interview questions.

This is the first course of the Deep Learning Specialization.

Setup

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. You can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

https://colab.research.google.com

Week 1: Introduction to Deep Learning

Week 2: Python and Vectorization
Logistic Regression as a Neural Network

Python and Vectorization

Python/Numpy vectors

https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

https://docs.scipy.org/doc/numpy/user/theory.broadcasting.html#array-broadcasting-in-numpy

[ ]

The Broadcasting Rule:
In order to broadcast, the size of the trailing axes for both arrays in an operation must either be the same size or one of them must be one.

[ ]

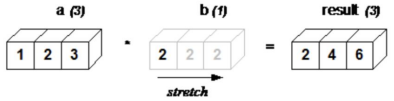


Figure 1

In the simplest example of broadcasting, the scalar ``b`` is stretched to become an array of with the same shape as ``a`` so the shapes are compatible for element-by-element multiplication.

[ ]

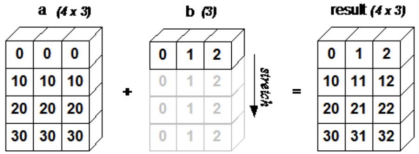


Figure 2

A two dimensional array multiplied by a one dimensional array results in broadcasting if number of 1-d array elements matches the number of 2-d array columns.

In the following example, both the `A` and `B` arrays have axes with length one that are expanded to a larger size during the broadcast operation:

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5
```

Here are some more examples:

```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):      3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):      3 x 1
Result (3d array):  15 x 3 x 5
```

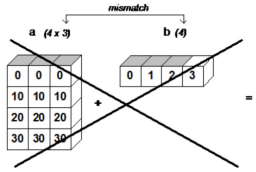


Figure 3

When the trailing dimensions of the arrays are unequal, broadcasting fails because it is impossible to align the values in the rows of the 1st array with the elements of the 2nd arrays for element-by-element addition.

Here are examples of shapes that do not broadcast:

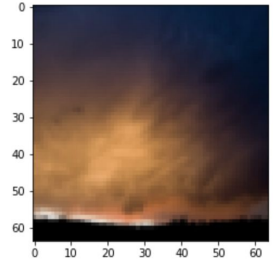
```
A      (1d array):  3
B      (1d array):  4 # trailing dimensions do not match

A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

Logistic Regression as a Neural Network

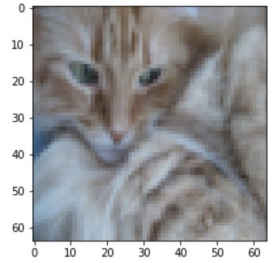
2 - Overview of the Problem set

Problem Statement: You are given a dataset ("data.h5") containing: - a training set of m_train images labeled as cat (y=1) or non-cat (y=0) - a test set of m_test images labeled as cat or non-cat - each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels (RGB). Thus, each image is square (height = num_px) and (width = num_px).

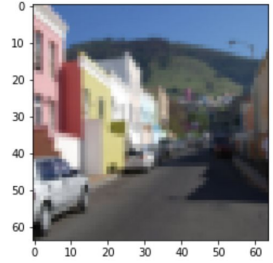
You will build a simple image-recognition algorithm that can correctly classify pictures as cat or non-cat.



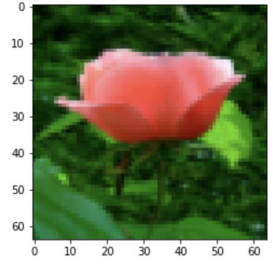
index:0, y = [0], it's a 'non-cat' picture.



index:2, y = [1], it's a 'cat' picture.



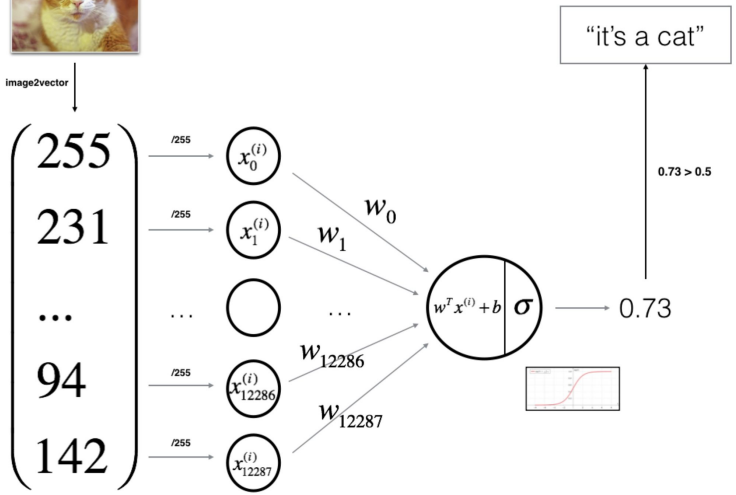
index:1, y = [0], it's a 'non-cat' picture.



index:3, y = [0], it's a 'non-cat' picture.

$$z^{(i)} = w^T x^{(i)} + b$$
$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$$
$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$
$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

$$\frac{\partial J}{\partial w} = \frac{1}{m} X(A - Y)^T$$
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)})$$



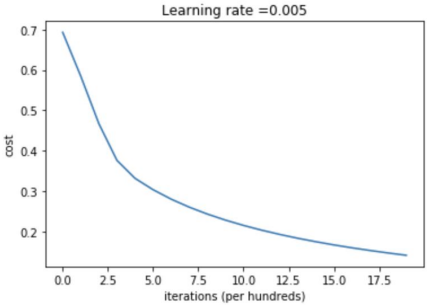
Logistic Regression as a Neural Network

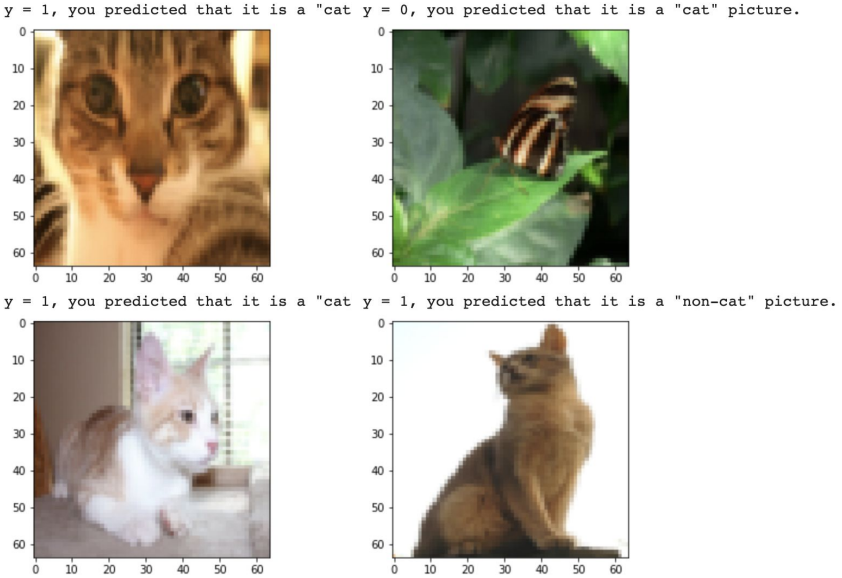
2 - Overview of the Problem set

Problem Statement: You are given a dataset ("data.h5") containing: - a training set of m_train images labeled as cat (y=1) or non-cat (y=0) - a test set of m_test images labeled as cat or non-cat - each image is of shape (num_px, num_px, 3) where 3 is for the 3 channels (RGB). Thus, each image is square (height = num_px) and (width = num_px).

You will build a simple image-recognition algorithm that can correctly classify pictures as cat or non-cat.

```
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.584508
Cost after iteration 200: 0.466949
Cost after iteration 300: 0.376007
Cost after iteration 400: 0.331463
Cost after iteration 500: 0.303273
Cost after iteration 600: 0.279880
Cost after iteration 700: 0.260042
Cost after iteration 800: 0.242941
Cost after iteration 900: 0.228004
Cost after iteration 1000: 0.214820
Cost after iteration 1100: 0.203078
Cost after iteration 1200: 0.192544
Cost after iteration 1300: 0.183033
Cost after iteration 1400: 0.174399
Cost after iteration 1500: 0.166521
Cost after iteration 1600: 0.159305
Cost after iteration 1700: 0.152667
Cost after iteration 1800: 0.146542
Cost after iteration 1900: 0.140872
train accuracy: 99.04306220095694 %
test accuracy: 70.0 %
```



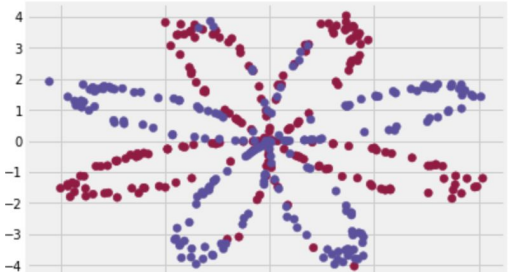


Week 3: Shallow Neural Network

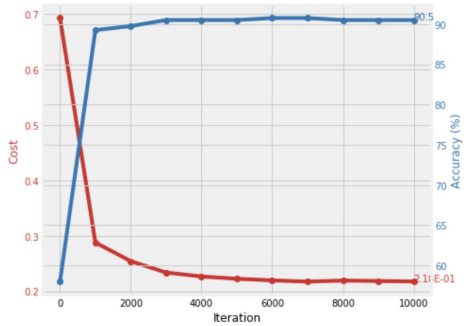
Planar data classification with one hidden layer

Planar data classification with one hidden layer

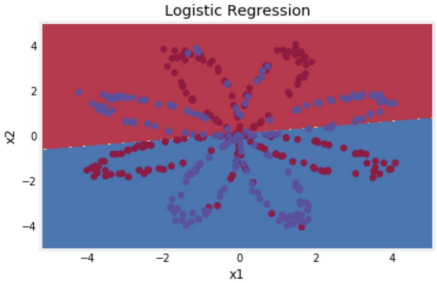
- Implement a 2-class classification neural network with a single hidden layer
- Use units with a non-linear activation function, such as tanh
- Compute the cross entropy loss
- Implement forward and backward propagation
- The data looks like a "flower" with some red (label y=0) and some blue (y=1) points.
- Your goal is to build a model to fit this data.



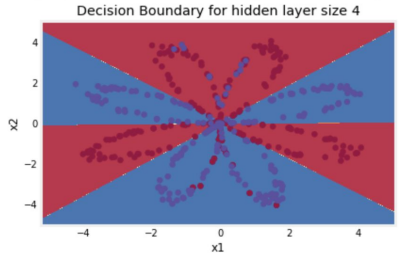
```
Cost after iteration 10000: 0.217821
Accuracy: 90%
```

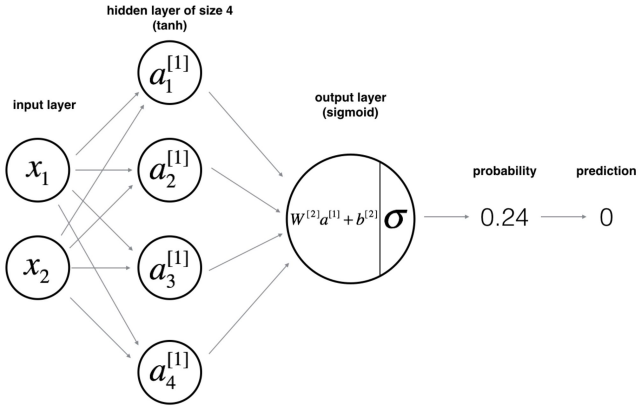



```
Accuracy of logistic regression: 47 % (percentage of correctly labelled
```




```
Text(0.5, 1.0, 'Decision Boundary for hidden layer size 4')
```





$$dz^{[2]} = a^{[2]} - y \qquad dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = dz^{[2]}a^{[1]^T} \qquad dW^{[2]} = \frac{1}{m} dZ^{[2]}A^{[1]^T}$$

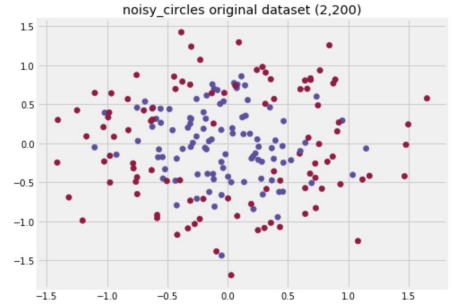
$$db^{[2]} = dz^{[2]} \qquad db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

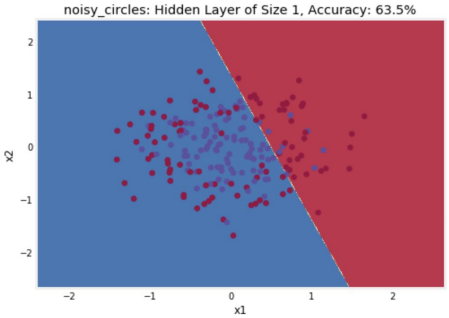
$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]}) \qquad dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

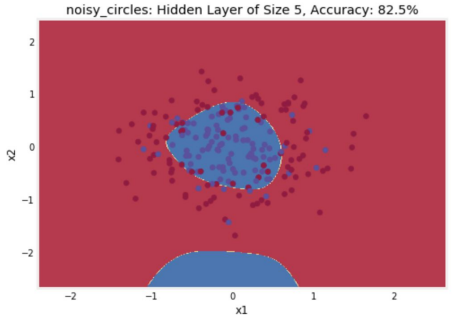
$$dW^{[1]} = dz^{[1]}x^T \qquad dW^{[1]} = \frac{1}{m} dZ^{[1]}X^T$$

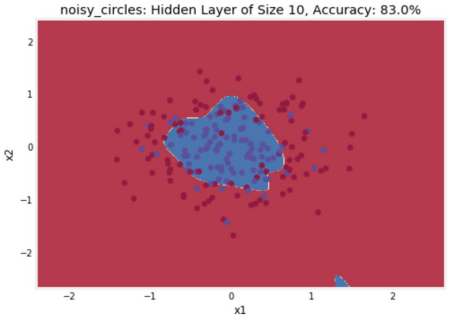
$$db^{[1]} = dz^{[1]} \qquad db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

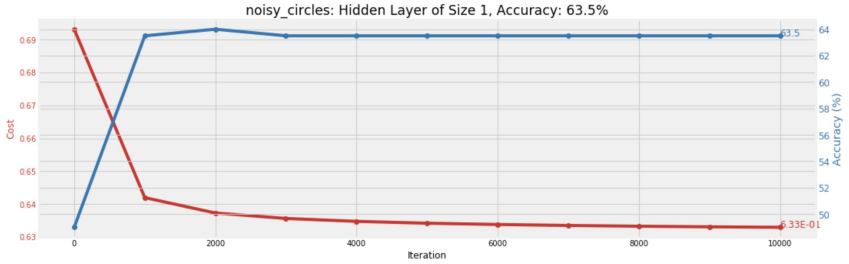
Performance on other datasets - noisy circles

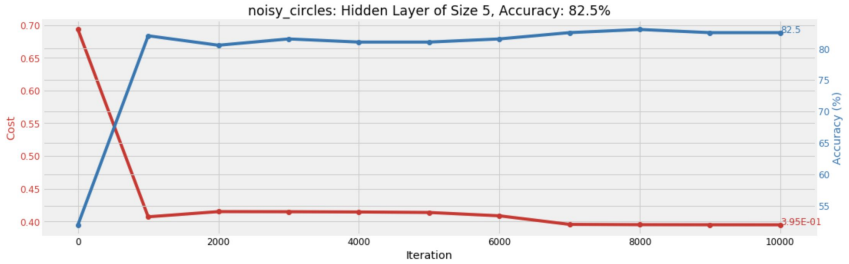


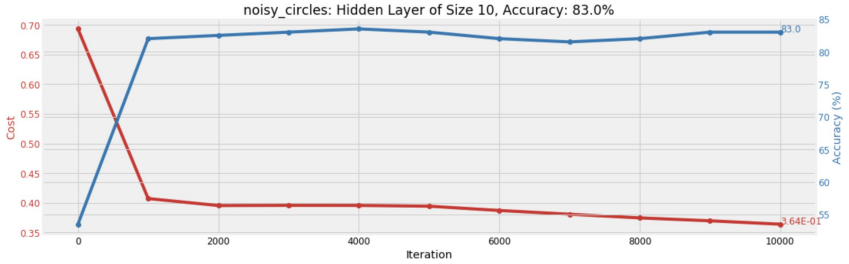




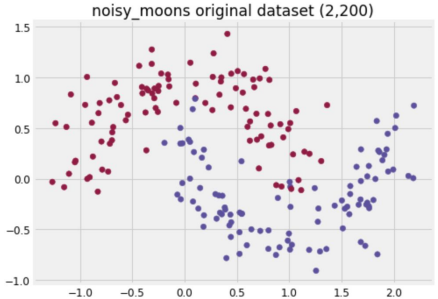


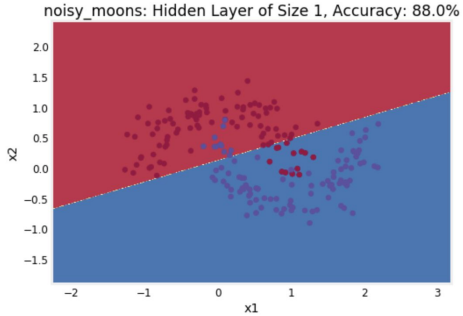


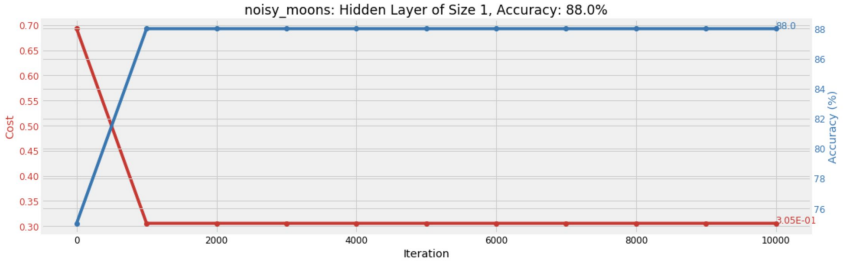


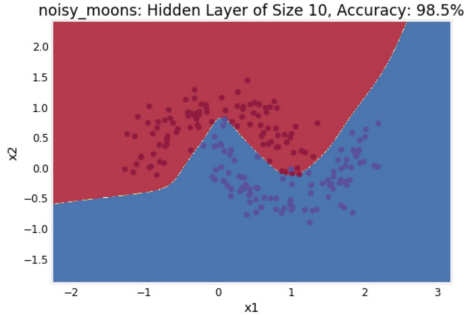


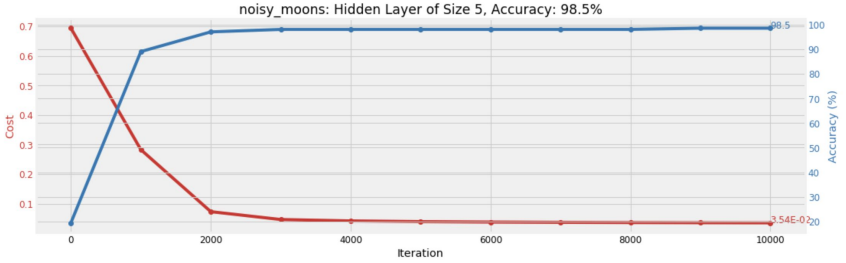
Performance on other datasets - noisy moons

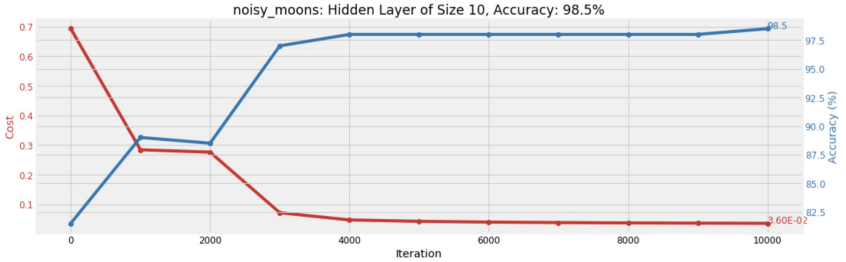




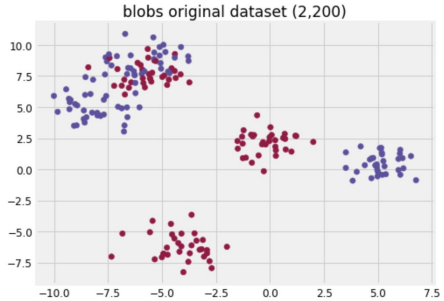


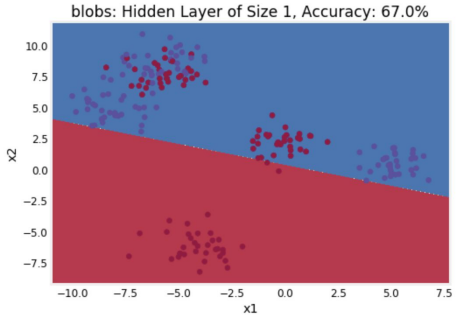


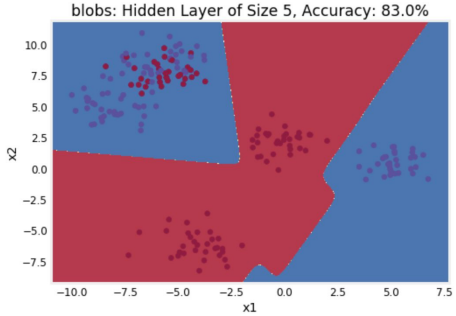


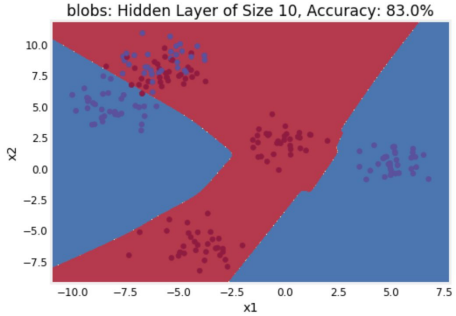


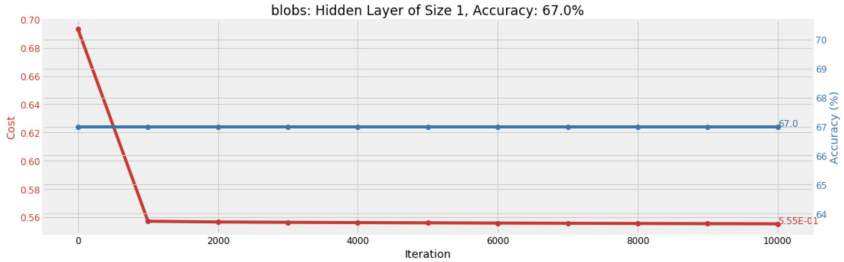
Performance on other datasets - clusters

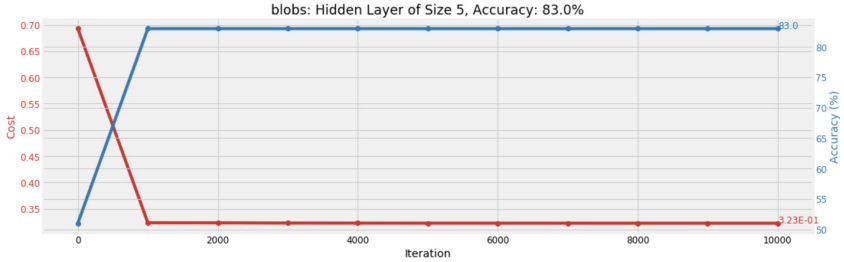


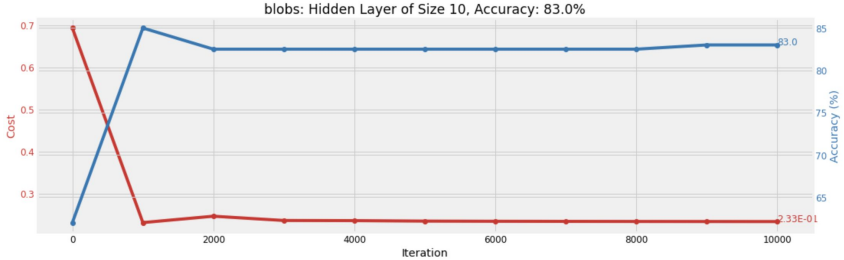




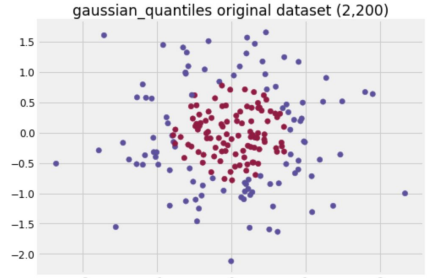


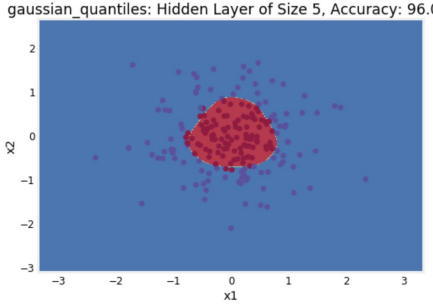


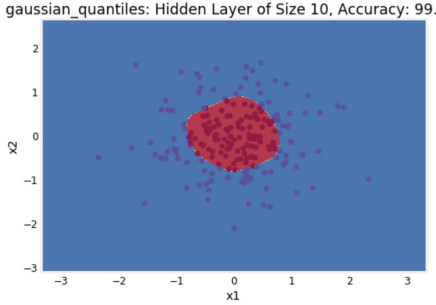


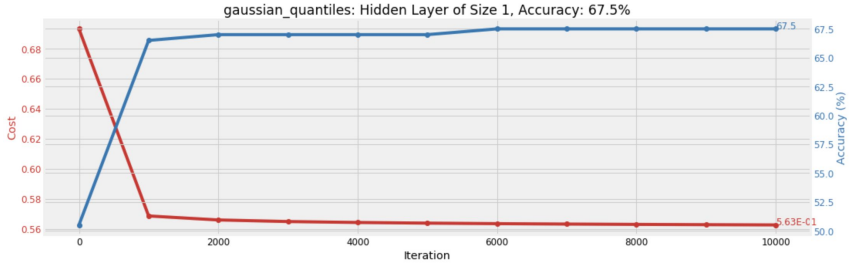


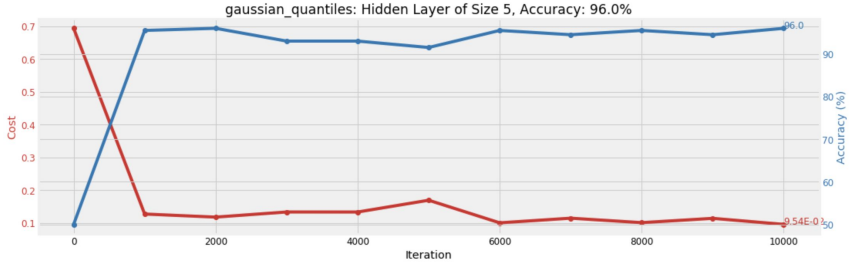
Performance on other datasets - gaussian quantiles

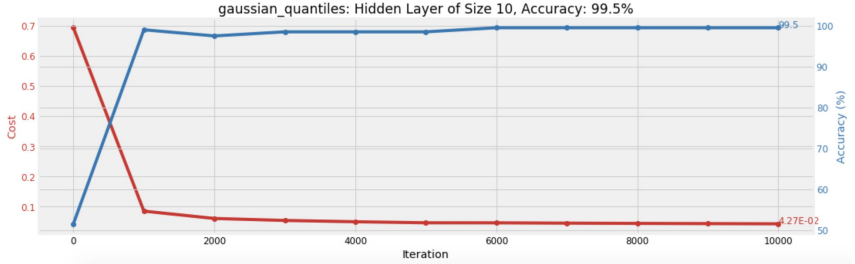








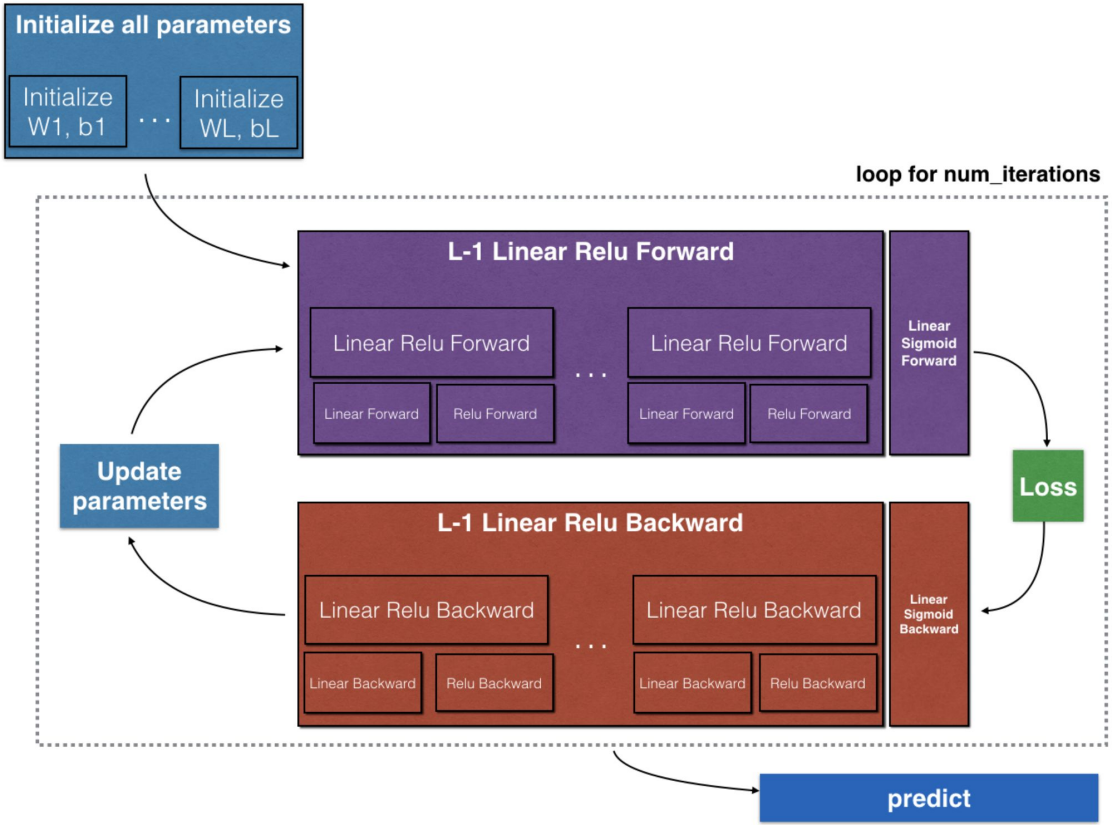




Week 4: Building your Deep Neural Network

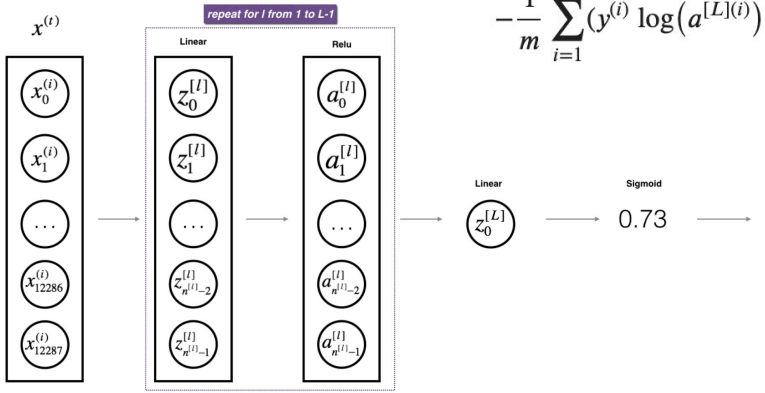
Deep Neural Network for Image Classification: Application

Building your Deep Neural Network

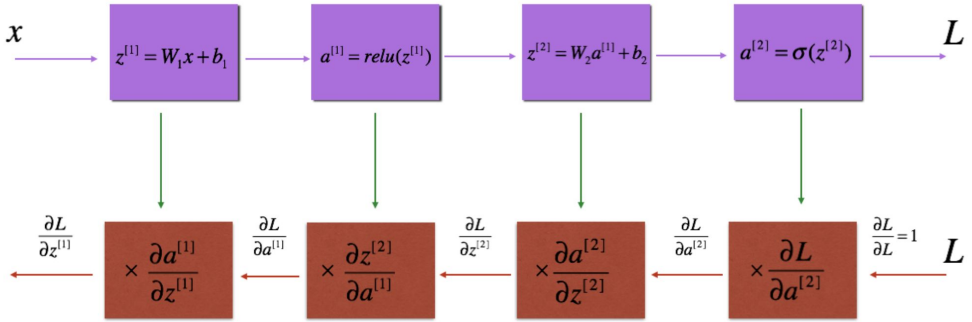


Building your Deep Neural Network

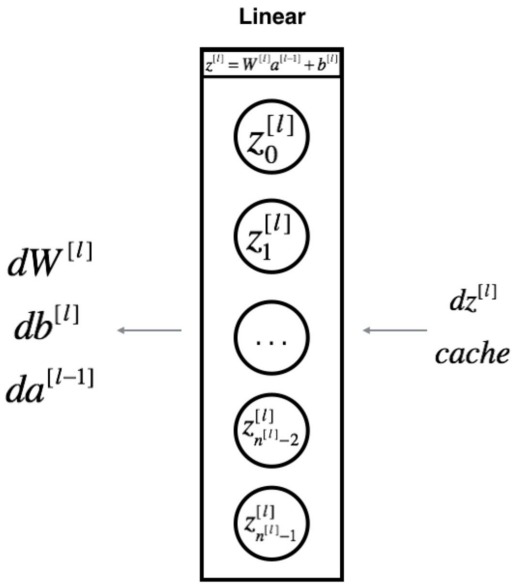
	Shape of W	Shape of b	Activation	Shape of Activation
Layer 1	$(n^{[1]}, 12288)$	$(n^{[1]}, 1)$	$Z^{[1]} = W^{[1]}X + b^{[1]}$	$(n^{[1]}, 209)$
Layer 2	$(n^{[2]}, n^{[1]})$	$(n^{[2]}, 1)$	$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$(n^{[2]}, 209)$
⋮	⋮	⋮	⋮	⋮
Layer L-1	$(n^{[L-1]}, n^{[L-2]})$	$(n^{[L-1]}, 1)$	$Z^{[L-1]} = W^{[L-1]}A^{[L-2]} + b^{[L-1]}$	$(n^{[L-1]}, 209)$
Layer L	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 209)$



$$-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right)+\left(1-y^{(i)}\right)\log\left(1-a^{[L](i)}\right)\right)$$



Building your Deep Neural Network



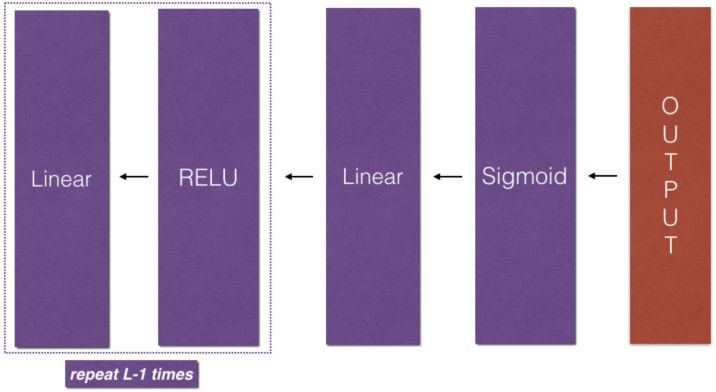
$$dW^{[l]} = \frac{\partial \mathcal{L}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

$$db^{[l]} = \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$$

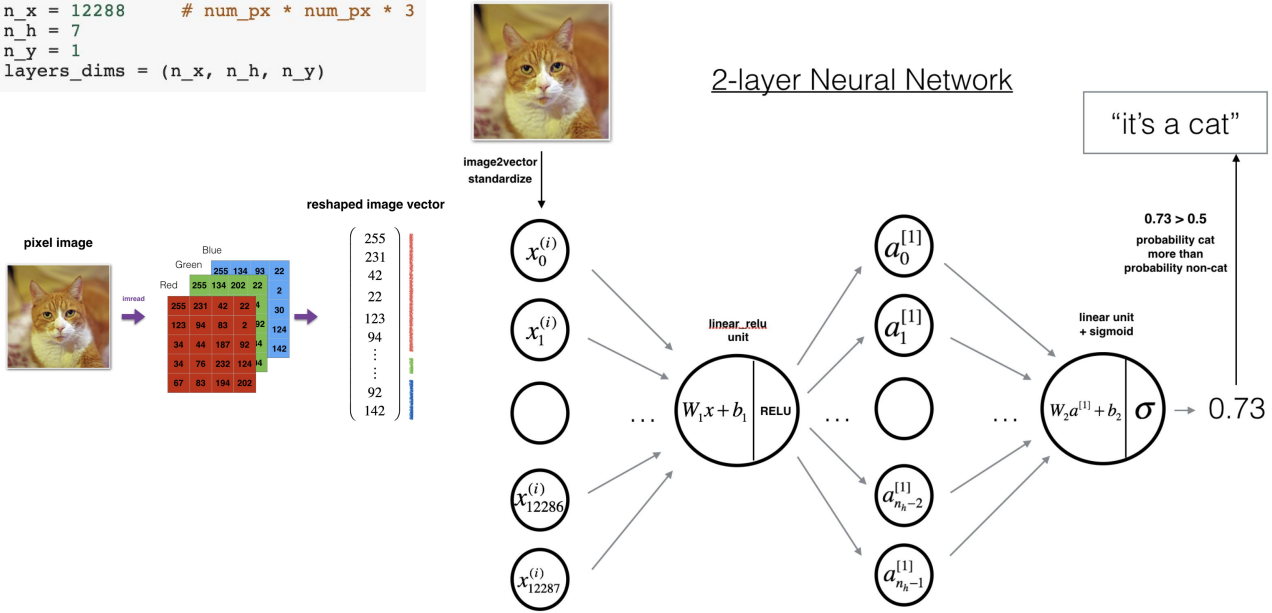
$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

$$W^{[l]} = W^{[l]} - \alpha \, dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha \, db^{[l]}$$

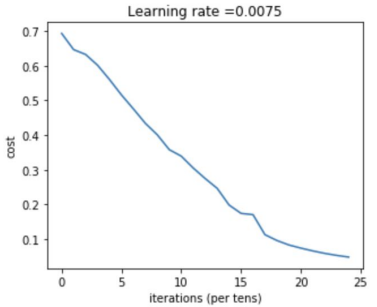


Deep Neural Network for Image Classification

```
n_x = 12288     # num_px * num_px * 3
n_h = 7
n_y = 1
layers_dims = (n_x, n_h, n_y)
```




```
Cost after iteration 0: 0.693049735659989
Cost after iteration 100: 0.6464320953428849
Cost after iteration 200: 0.6325140647912678
Cost after iteration 300: 0.6015024920354665
Cost after iteration 400: 0.5601966311605748
Cost after iteration 500: 0.515830477276473
Cost after iteration 600: 0.47549013139433255
Cost after iteration 700: 0.4339163151225749
Cost after iteration 800: 0.4007977536203889
Cost after iteration 900: 0.3580705011323798
Cost after iteration 1000: 0.3394281538366412
Cost after iteration 1100: 0.30527536361962637
Cost after iteration 1200: 0.27491377282130164
Cost after iteration 1300: 0.24681768210614818
Cost after iteration 1400: 0.1985073503746612
Cost after iteration 1500: 0.17448318112556635
Cost after iteration 1600: 0.1708076297809699
Cost after iteration 1700: 0.11306524562164708
Cost after iteration 1800: 0.09629426845937154
Cost after iteration 1900: 0.08342617959726871
Cost after iteration 2000: 0.07439078704319087
Cost after iteration 2100: 0.06630748132267933
Cost after iteration 2200: 0.05919329501038172
Cost after iteration 2300: 0.053361403485605585
Cost after iteration 2400: 0.048554785628770226
```




```
predictions_train = predict(train_x, train_y, parameters)
```

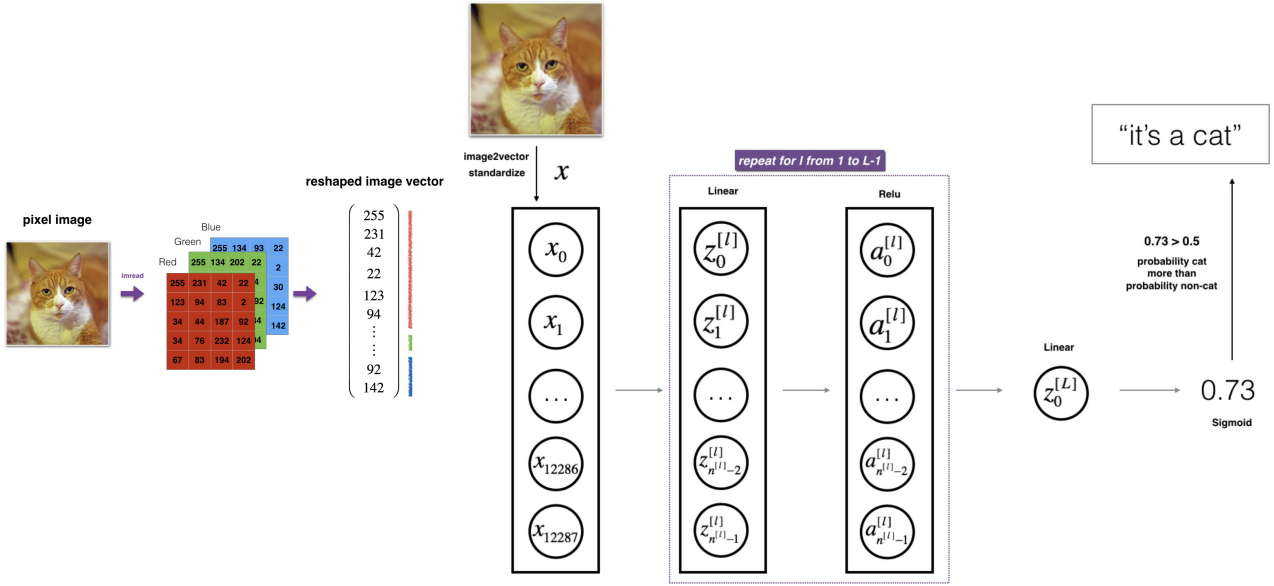
```
Accuracy: 0.9999999999999998
```

```
predictions_test = predict(test_x, test_y, parameters)
```

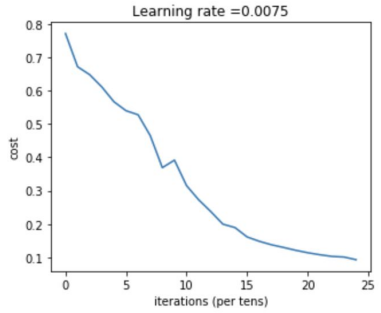
```
Accuracy: 0.72
```

Deep Neural Network for Image Classification

```
layers_dims = [12288, 20, 7, 5, 1] #  4-layer model
```




```
Cost after iteration 0: 0.771749
Cost after iteration 100: 0.672053
Cost after iteration 200: 0.648263
Cost after iteration 300: 0.611507
Cost after iteration 400: 0.567047
Cost after iteration 500: 0.540138
Cost after iteration 600: 0.527930
Cost after iteration 700: 0.465477
Cost after iteration 800: 0.369126
Cost after iteration 900: 0.391747
Cost after iteration 1000: 0.315187
Cost after iteration 1100: 0.272700
Cost after iteration 1200: 0.237419
Cost after iteration 1300: 0.199601
Cost after iteration 1400: 0.189263
Cost after iteration 1500: 0.161189
Cost after iteration 1600: 0.148214
Cost after iteration 1700: 0.137775
Cost after iteration 1800: 0.129740
Cost after iteration 1900: 0.121225
Cost after iteration 2000: 0.113821
Cost after iteration 2100: 0.107839
Cost after iteration 2200: 0.102855
Cost after iteration 2300: 0.100897
Cost after iteration 2400: 0.092878
```




```
pred_train = predict(train_x, train_y, parameters)
```

```
Accuracy: 0.9856459330143539
```

```
pred_test = predict(test_x, test_y, parameters)
```

```
Accuracy: 0.8
```

Deep Neural Network for Image Classification

Prediction: cat
Class: non-cat

Prediction: non-cat
Class: cat

Prediction: cat
Class: non-cat

Prediction: non-cat
Class: cat

Prediction: non-cat
Class: cat

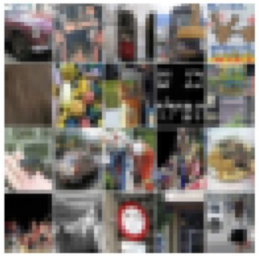
Prediction: cat
Class: non-cat

A few types of images the model tends to do poorly on in

- Cat body in an unusual position
- Cat appears against a background of a similar color
- Unusual cat color and species
- Camera Angle
- Brightness of the picture
- Scale variation (cat is very large or small in image)

Prediction: cat
Class: non-cat

Prediction: cat
Class: non-cat



Prediction: cat
Class: non-cat

Prediction: cat
Class: non-cat

Questions

Discussion

Standard notations for Deep Learning

This document has the purpose of discussing a new standard for deep learning mathematical notations.

1 Neural Networks Notations.

General comments:

· superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

·m : number of examples in the dataset

·n_x : input size

·n_y : output size (or number of classes)

·$n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h{}^{[\text{number of layers } +1]}$.

·L : number of layers in the network.

Objects:

·$X \in \mathbb{R}^{n_x \times m}$ is the input matrix

·$x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th}example represented as a column vector

·$Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

·$y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

·$W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

·$b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

·$\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

Common forward propagation equation examples:

$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1)$ where $g^{[l]}$ denotes the l^{th} layer activation function

$\hat{y}^{(i)} = softmax(W_h h + b_2)$

· General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

· $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Examples of cost function:

· $J_{CE}(\hat{y}, y) = -\sum_{i=0}^{m} y^{(i)} \log \hat{y}^{(i)}$

· $J_1(\hat{y}, y) = \sum_{i=0}^{m} \mid y^{(i)} - \hat{y}^{(i)} \mid$

2 Deep Learning representations

For representations:

· nodes represent inputs, activations or outputs

· edges represent weights or biases

Here are several examples of Standard deep learning representations

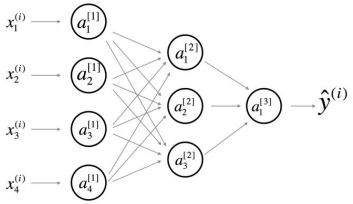


Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

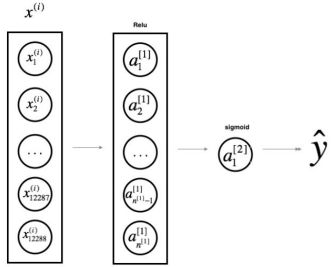


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.