

content of courses :

Courses in this Specialization

1. Neural Networks and Deep Learning ←
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project
4. Convolutional Neural Networks
5. Natural Language Processing: Building sequence models

hi Andrew Ng, I'm back~

Introduction:

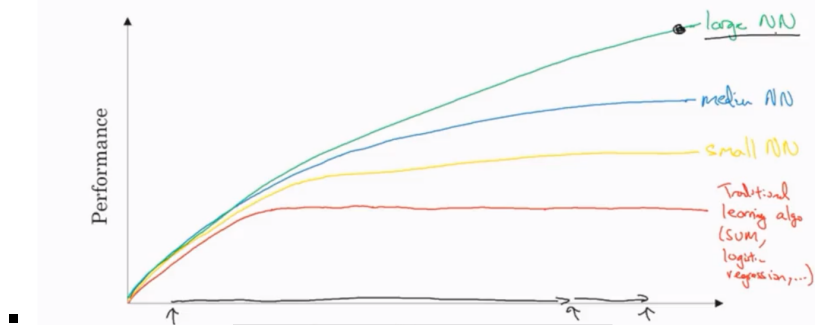
- applications e.g.,

Supervised Learning

Input(x) ←	Output (y) ←	Application
Home features	Price	Real Estate
Ad, user info ←	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging } CNN
Audio	Text transcript	Speech recognition } RNN
English	Chinese	Machine translation }
Image, Radar info	Position of other cars	Autonomous driving } Custom/ Hybrid

- implements in both structured data/ unstructured data
- why NN go bananas nowadays
 - boosting data!
 - hardware improvement!
 - algorithm help reduce computation. (e.g., sigmoid function → Relu function)
 - see the performance compared with traditional machine learning in Big Data

Scale drives deep learning progress



Logistic Regression in NN

- as usual, notation first:

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)
- $n_l^{[l]}$: number of hidden units of the l^{th} layer
- In a few hops, it is possible to denote $n_x = n_0^{[0]}$ and $n_y = n_L^{[L]}$ (number of layers +1).
- L : number of layers in the network.

Objects:

- $X \in \mathbb{R}^{m \times n_x}$ is the input matrix
- $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector

- $Y \in \mathbb{R}^{m \times n_y}$ 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 $d^{[L]}$ 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

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

- 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:

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

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

1

2

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

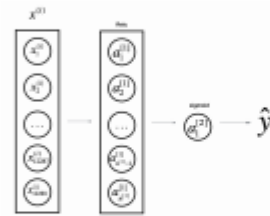


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.

o caution! In Python, our data shape is **different** from Matlab. (the vector is transposed.)

$$X = \begin{bmatrix} | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & \dots & | \end{bmatrix}$$

$X \in \mathbb{R}^{n_x \times m}$ $X.shape = (n_x, m)$

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$

$Y \in \mathbb{R}^{1 \times m}$
 $Y.shape = (1, m)$

- Review Logistic Regression and see things new in neural network.
 - hypothesis:

Logistic Regression

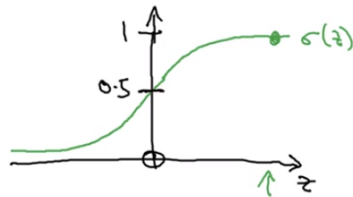
Given x , want $\hat{y} = \frac{P(y=1|x)}{P(y=1|x) + P(y=0|x)}$

$x \in \mathbb{R}^{n_x}$ $0 \leq \hat{y} \leq 1$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

different from traditional ML, we separate real number b from vector w into two variable, which means we don't use X_0 to denote b

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_z)$



$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \leftarrow \text{sigmoid function}$$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1 + e^{-z}} \approx \frac{1}{1 + \text{big number}} \approx 0$$

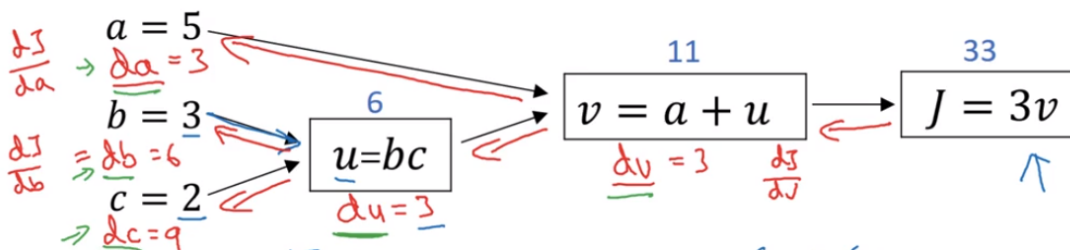
- loss function & cost function(same)

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

Cost function: $J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$

- Intuition of backward propagation
 - task: to compute parameters' derivatives
 - how? using **chain rule!**

Computing derivatives



- coding habit: use "dvar" as the derivatives of those output variables. e.g., use "da, db, dc" to denote the derivatives of a, b, c above.

- Whole **algorithm**(forward propagation+backward propagation)

#forward propagation(Pseudo code)

$Z = W.T * X + b$

$A = \text{sigmoid}(Z)$ #y-hat, our hypothesis $\text{sigmoid}(Z) = 1/(1+e^{(-Z)})$

$\text{Loss}(A, Y) = -1/m * (Y * \log(A) + (1-Y) * \log(1-Y))$

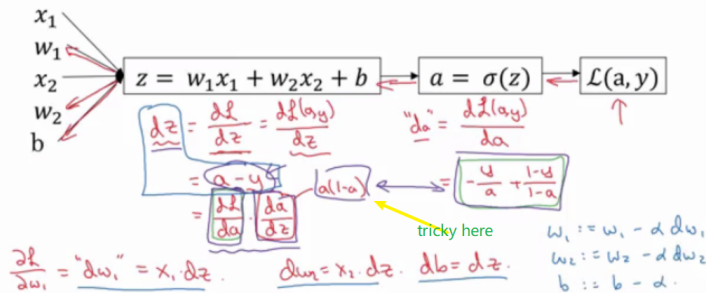
#backward propagation(Pseudo code)

$dZ = A - Y$

$dW = (X.T * dZ) / m$

$db = \text{sum}(dZ) / m$

Logistic regression derivatives



- o details:

$$\frac{dL}{dz} = \frac{dL}{da} \times \frac{da}{dz}$$

$$\therefore \frac{dz}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{da}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}$$

$$\therefore \frac{da}{dz} = a \cdot (1-a)$$

$$\therefore \frac{dL}{dz} = a(1-a) \times \left(-\frac{y}{a} + \frac{(1-y)}{1-a} \right)$$

$$= a - y$$

• concept of Numpy

- o **broadcasting** (expand the matrix into same size automatically)

Broadcasting example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$$

$(m, n) \quad (2, 3) \quad (1, n) \rightsquigarrow (m, n) \quad (2, 3)$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$$

$(m, n) \quad (m, 1) \quad (m, n)$

- flexible enough but sometime cause subtle bugs TAT.
- **Coding habits in Numpy**
 - specify the shape of a vector, don't use "rank 1 array" which looks like this: (m,)
 - don't be shy to use:
 - `assert(a.shape == (5,1))` to check if your matrix shape is the required one.
 - reshape matrix to (m, n) #the size you want.
- Axis
 - `obj.sum(axis = 0)` sums the columns while `obj.sum(axis = 1)` sums the rows.
- get help from document
 - write `np.exp?` (for example) to get quick access to the documentation.
- normalize

1.4 - Normalizing rows

Another common technique we use in Machine Learning and Deep Learning is to normalize our data. It often leads to a better performance because gradient descent converges faster after normalization. Here, by normalization we mean changing x to $\frac{x}{\|x\|}$ (dividing each row vector of x by its norm).

For example, if

$$x = \begin{bmatrix} 0 & 3 & 4 \\ 2 & 6 & 4 \end{bmatrix} \quad (3)$$

then

$$\|x\| = np.linalg.norm(x, axis = 1, keepdims = True) = \begin{bmatrix} 5 \\ \sqrt{56} \end{bmatrix} \quad (4)$$

and

$$x_{normalized} = \frac{x}{\|x\|} = \begin{bmatrix} 0 & \frac{3}{5} & \frac{4}{5} \\ \frac{2}{\sqrt{56}} & \frac{6}{\sqrt{56}} & \frac{4}{\sqrt{56}} \end{bmatrix} \quad (5)$$

the self product of row

-
- **L1 & L2 Loss**

$$L_1(\hat{y}, y) = \sum_{i=0}^m |y^{(i)} - \hat{y}^{(i)}|$$

▪

```
loss_1 = sum(abs(y-yhat))
```

$$L_2(\hat{y}, y) = \sum_{i=0}^m (y^{(i)} - \hat{y}^{(i)})^2$$

▪

```
loss_2 = np.dot(yhat-y,yhat-y)
```

- reshape & flatten images

A trick when you want to flatten a matrix X of shape (a,b,c,d) to a matrix $X_{flatten}$ of shape (b*c*d, a) is to use:

```
X_flatten = X.reshape(X.shape[0], -1).T # X.T is the transpose of X
```

▪

Assignment: Classify Cat or Non-Cat!

- Step by step

- **data preview and preprocess**
 - get intuition of your data, i.e. know what their size are.
 - **regularization:**
 - instead of $\text{std}=(x-\mu)/\text{sqrt}(\sigma)$, here we just divide every row of the dataset by 255 (the maximum value of a pixel channel).
- **function prepare:**

```
def sigmoid(z):  
    s = 1/(1+np.exp(-z))  
    return s
```

- **Initialization**
 - use `np.zeros((n,m))`
- **Forward and Backward Propagation to optimize parameters**
- **Make Prediction**