Course 1: Neural Networks and Deep Learning

Content:

Week 1: Introduction to deep learning

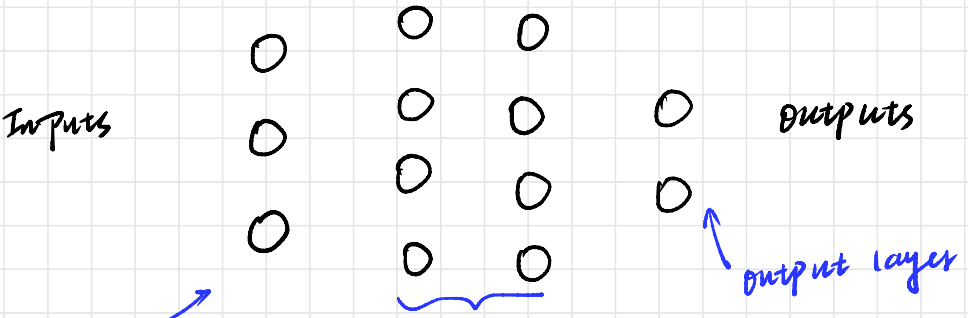
Week 2: Neural Networks Basics

Week 3: Shallow Neural Networks

Week 4: Deep Neural Networks

Coursera

Deep Learning Specialization



we can't see these layers in training set

Structured vs. unstructured data

things that has a defined meaning. e.g Price, age

e.g. Pixel, raw audio, text.

ReLU: rectified linear unit

CNN: convolutional neural network → useful in computer vision

RNN: Recurrent neural network → useful in speech recognition

Standard NN: useful in structured data

Sigmoid: $\hat{y} = P(y=1|x)$ $y \in (0,1)$ Prob.

$$\hat{y} = \text{sigmoid}(w^T x + b) = \sigma(w^T x + b)$$

$$\text{sigmoid}(z) = 1/(1+e^{-z}), \quad \sigma'(z) = \sigma(z)(1-\sigma(z))$$

loss function:

square root error: $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

↓ lead us to optimization prob. which is non convex

we use: $L(\hat{y}^{(i)}, y^{(i)}) = -[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

① if $y^{(i)} = 1$, $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$

$\log(\hat{y}^{(i)})$ & $\hat{y}^{(i)}$ close to 1.

② if $y^{(i)} = 0$, $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$

$\log(1 - \hat{y}^{(i)})$ & $\hat{y}^{(i)}$ close to 0.

cost function:

↓

Average of loss function of entire training set

Goal: find w & b s.t minimize overall cost

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y})^{(i)})]$$

* loss fnc: measures how well the model is doing on single training example

* cost fnc: measures how well parameters w & b are doing on entire training set

Gradient Descent

Predict w & b that minimizes cost fnc → convex

initialize w & b to 0, 0 OR to random value

↓ logistic regression

$$w := w - \alpha \underbrace{\partial J(w,b)/\partial w}_{dw}, \quad b := b - \alpha \underbrace{\partial J(w,b)/\partial b}_{db}$$

In logistic regression:

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b) \quad \text{where } \sigma(z^{(i)}) = 1/(1+e^{-z^{(i)}})$$

Single: $z = w^T x + b$, $\hat{y} = a = \sigma(z)$

$$L(a, y) = -(y \log(a) + (1-y) \log(1-a))$$

m: $z^{(i)} = w^T x^{(i)} + b$, $\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$

$$J(w, b) = 1/m \sum_i^m (\hat{y}^{(i)}, y^{(i)})$$

$$= -1/m \sum_i^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

$$dw_1 = 1/m \sum_i^m x_1^{(i)} (a^{(i)} - y^{(i)})$$

$$db = 1/m \sum_i^m (a^{(i)} - y^{(i)})$$

Vectorization

in coding → avoid for loops.

↓ in python

X : $[N_x, m]$

matrix Y : $[N_y, m]$

compute: $[z_1, z_2, \dots z_m] = w^T x + [b, \dots b, b]$ ← some constants

Python:

z → $(1, m)$ $= np.dot(w.T, x) + b$

$(1, m)$ $A = sigmoid(z)$

$(1, m)$ $dz = A - Y$

$(N_x, 1)$ $dw = 1/m * np.dot(X, dz.T)$ } vec. → boost computation

$db = 1/m * np.sum(dz)$

$w = w - alpha * dw$ } update parameters

$b = b - alpha * db$

↓

coding part Refer to lab assignments.

Single examples

↓ shape

X	$(n_x, 1)$	input with n_x features.
$W^{[1]}$	(n_1, n_x)	matrix of 1st hidden layer
$b^{[1]}$	$(n_1, 1)$	bias term of hidden layer
$z^{[1]}$	$(n_1, 1)$	result of $z^{[1]} = W^{[1]}x + b$
$a^{[1]}$	$(n_1, 1)$	result of $a^{[1]} = sigmoid(z^{[1]})$, act. of hid. layer.

$W^{[2]}$ $(1, n_1)$, $b^{[2]} = (1, 1)$, $z^{[2]}$ $(1, 1)$, $a^{[2]}$ $(1, 1)$

Multiple examples.

$X \rightarrow (n_x, m)$, $Z^{[1]} \rightarrow (n_1, m)$, $A^{[1]} \rightarrow (n_1, m)$

Activation function

* sigmoid $a = 1/(1+e^{-z})$ → $a(1-a)$ the derivative

output value bit 0,1

tanh $a = (e^z - e^{-z})/(e^z + e^{-z})$ works better than sigmoid

$\begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z>0 \end{cases}$ ↑ $1-a^2$

for hidden units use the mean of its output closer to 0.

* ReLU $a = \max(0, z)$ most widely used

Leaky ReLU $a = \max(0.01z, z)$ improved ReLU

$\begin{cases} 0.01 & \text{if } z<0 \\ 1 & \text{if } z>0 \end{cases}$ ↓ this can be a parameter

Gradient descent

X	(n_x, m)			variables
$Z^{[1]}$	(n_1, m)	$Z^{[2]}$	$(n_2, 1)$	
$A^{[1]}$	(n_1, m)	$A^{[2]}$	$(n_2, 1)$	
$W^{[1]}$	(n_1, n_x)	$W^{[2]}$	(n_2, n_1)	parameters
$b^{[1]}$	$(n_1, 1)$	$b^{[2]}$	$(n_2, 1)$	

Deep N.N.

- $n^{[0]}$ # of neurons in input layer
- $n^{[l]}$ .... in l th layer
- $W^{[l]}$ weights of lth layer shape $(n^{[l]}, n^{[l-1]})$
- $b^{[l]}$ bias term of lth layer shape $(n^{[l]}, 1)$
- $Z^{[l]}$ affine result of the layer shape $(n^{[l]}, m)$, $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
- $g^{[l]}$ activation fnc of lth layer
- $A^{[l]}$ activation output of lth layer shape $(n^{[l]}, m)$

  $A^{[l]} = g^{[l]}(Z^{[l]})$

Summary of Dimension:

$w^{[l]}$	$(n^{[l]}, n^{[l-1]})$	$dw^{[l]}$	$(n^{[l]}, n^{[l-1]})$
$b^{[l]}$	$(n^{[l]}, 1)$	$db^{[l]}$	$(n^{[l]}, 1)$
$z^{[l]}$	$(n^{[l]}, m)$	$dz^{[l]}$	$(n^{[l]}, m)$
$A^{[l]}$	$(n^{[l]}, m)$	$dA^{[l]}$	$(n^{[l]}, m)$

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

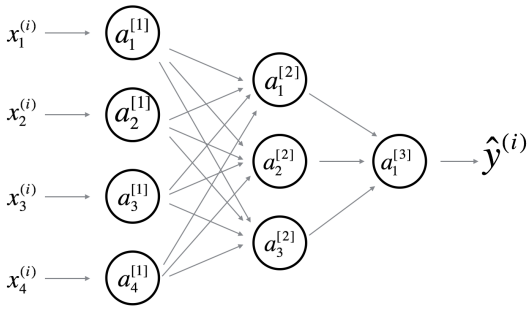


Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

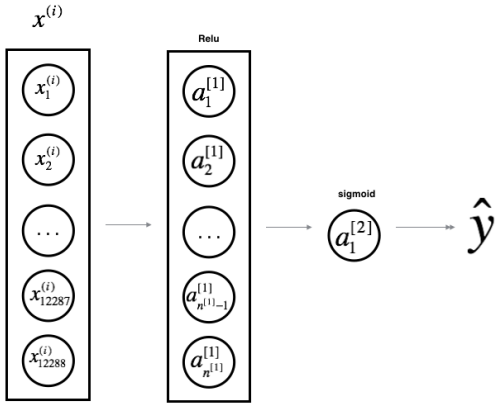


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.