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 ith training example while superscript [l] will denote the lth 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 lth 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 ith 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 lth 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 lth 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.

Binary Classification

In a binary classification problem, the result is a discrete value output.

For example - account hacked (1) or compromised (0)
- a tumor malign (1) or benign (0)

Example: Cat vs Non-Cat

The goal is to train a classifier that the input is an image represented by a feature vector, x, and predicts whether the corresponding label y is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).

An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n-dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, x, the pixel intensity values will be "unroll" or "reshape" for each color. The dimension of the input feature vector x is $n_x = 64\ x\ 64\ x\ 3 = 12\ 288$.

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{red} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{green} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{blue} \end{matrix}$$

Logistic Regression

Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.

Example: Cat vs No - cat

Given an image represented by a feature vector x, the algorithm will evaluate the probability of a cat being in that image.

$$Given\ x\ ,\ \ \hat{y} = P(y = 1|x), \text{where } 0\ \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The threshold: $b\ \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+\ e^{-z}}$

$(w^T x + b)$ is a linear function $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

- If z is a large positive number, then $\sigma(z)$ = 1
- If z is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5

Logistic Regression: Cost Function

To train the parameters w and b, we need to define a cost function.

Recap:

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

$x^{(i)}$ the i-th training example

$Given \left\{\left(x^{(1)}, y^{(1)}\right), \cdots, \left(x^{(m)}, y^{(m)}\right)\right\},\ we\ want\ \hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

$$L\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

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

- If $y^{(i)} = 1$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$ where $\log\left(\hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$ where $\log\left(1 - \hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 0

Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters $w\ and\ b$ that minimize the overall cost function.

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

Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode

deeplearning.ai

Introduction to Deep Learning

Welcome

- AI is the new Electricity
- Electricity had once transformed countless industries: transportation, manufacturing, healthcare, communications, and more

- AI will now bring about an equally big transformation.

Andrew Ng

What you'll learn

Courses in this sequence (Specialization):

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project train/dev/test
4. Convolutional Neural Networks CNN end-to-end
5. Natural Language Processing: Building sequence models

RNN, LSTM

Andrew Ng

deeplearning.ai

Introduction to Deep Learning

What is a Neural Network?

Housing Price Prediction

ReLU
Rectified
Linear
Unit

Housing Price Prediction

Housing Price Prediction

Drawing of previous Image

deeplearning.ai

Introduction to Deep Learning

Supervised Learning with Neural Networks

Supervised Learning

Input(x)	Output (y)	Application	
Home features	Price	Real Estate	Standard NN
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

Neural Network examples

Standard NN

Convolutional NN

Recurrent NN

Supervised Learning

Structured Data

Size	#bedrooms	…	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
…	…		…
3000	4		540

User Age	Ad Id	…	Click
41	93242		1
80	93287		0
18	87312		1
…	…		…
27	71244		1

Unstructured Data

Audio

Image

Four scores and seven years ago…

Text

deeplearning.ai

Introduction to Neural Networks

Why is Deep Learning taking off?

Scale drives deep learning progress

Andrew Ng

Scale drives deep learning progress

- Data
- Computation
- Algorithms

Idea

Experiment

Code

10 min
1 day
1 month

Andrew Ng

deeplearning.ai

Introduction to Neural Networks

About this Course

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

Outline of this Course

Week 1: Introduction

Week 2: Basics of Neural Network programming

Week 3: One hidden layer Neural Networks

Week 4: Deep Neural Networks

Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode

Basics of Neural Network Programming

Binary Classification

deeplearning.ai

Binary Classification

64

64

$\longrightarrow$ 1 (cat) vs 0 (non cat)

y

$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \longrightarrow y$

Andrew Ng

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}, \; y \in \{0, 1\}$

m training examples : $\{(\underline{x}^{(1)}, \underline{y}^{(1)}), (\underline{x}^{(2)}, \underline{y}^{(2)}), \ldots, (\underline{x}^{(m)}, \underline{y}^{(m)})\}$

$m = m_{train}$ $\qquad m_{test}$ = # test examples.

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

$$\cancel{\begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}}$$

$\longleftarrow m \longrightarrow$

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

$$Y = [y^{(1)} \; y^{(2)} \; \cdots, y^{(m)}]$$

$Y \in \mathbb{R}^{1 \times m}$

$Y.\text{shape} = (1, m)$

Andrew Ng

Basics of Neural Network Programming

Logistic Regression

deeplearning.ai

Logistic Regression

Given x, want $\hat{y} = P(y=1 \mid x)$

$x \in \mathbb{R}^{n_x}$

$0 \le \hat{y} \le 1$

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

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

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \leftarrow \\ \\ \\ \\ \\ \end{matrix} \quad \} w \leftarrow$$

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

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

If z large negative number

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

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

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

$z^{(i)} = w^T x^{(i)} + b$

$x^{(i)}$, $y^{(i)}$, $z^{(i)}$ — i-th example.

Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

$\boxed{\text{Loss}}$ (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

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

If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y = 0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log 1-\hat{y}$ large .... want $\hat{y}$ small

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

Andrew Ng

Basics of Neural Network Programming

Gradient Descent

deeplearning.ai

Gradient Descent

Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

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

Want to find w, b that minimize $J(w,b)$

Andrew Ng

Gradient Descent

Repeat {

$$w := w - \alpha \boxed{\frac{dJ(w)}{dw}}$$

learning rate

"dw"

}

$$w := w - \alpha \, dw$$

$$\frac{dJ(w)}{dw} = ?$$

$J(w,b)$

$$w := w - \alpha \boxed{\frac{\partial J(w,b)}{\partial w}}$$

$$b := b - \alpha \boxed{\frac{\partial J(w,b)}{\partial b}}$$

∂ "partial derivative"

d J

dw

db

Andrew Ng

Basics of Neural Network Programming

Derivatives

deeplearning.ai

Intuition about derivatives

$f(a) = 3a$

$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

$a = 2$ $f(a) = 6$

$a = 2.001$ $f(a) = 6.003$

slope (derivative) of $f(a)$ at $a=2$ is 3

$a = 5$ $f(a) = 15$

$a = 5.001$ $f(a) = 15.003$

slope at $a=5$ is also 3

$$\frac{d\,f(a)}{da} = 3 = \frac{d}{da} f(a)$$

0.001

0.00000001

0.000000001

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

More derivatives examples

Intuition about derivatives

$f(a) = a^2$

$$\frac{d}{da} a^2 = 2a$$

0.001

$(2a) \times 0.001$

0.001

$0.00000\ldots01$

$a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) \approx 4.004$

$(4.004\boxed{004})$

slope (derivative) of $f(a)$ at $a=2$ is 4.

$\boxed{\frac{d}{da} f(a) = 4}$ when $\boxed{a=2}$.

$a = 5$ $\quad f(a) = 25$

$a = 5.001$ $\quad f(a) \approx 25.010$

$\boxed{\frac{d}{da} f(a) = 10}$ when $\boxed{a=5}$

$$\frac{d}{da} f(a) = \underbrace{\frac{d}{da} a^2} = \boxed{2a}$$

Andrew Ng

More derivative examples

$f(a) = a^2$

$\frac{d}{da} f(a) = \underbrace{2a}_{4}$

$a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) \approx 4.004$

$f(a) = a^3$

$\frac{d}{da} f(a) = \underbrace{3a^2}_{3 \times 2^2 = 12}$

$a = 2$ $\quad f(a) = 8$

$a = \underline{2.001}$ $\quad f(a) \approx 8.\underline{012}$

$f(a) = \log_e(a)$

$\ln(a)$

$\frac{d}{da} f(a) = \underline{\frac{1}{a}}$

$\frac{d}{da} f(a) = \boxed{\frac{1}{2}}$

$a = 2$ $\quad f(a) \approx 0.69315$

$a = 2.\underline{001}$ $\quad \underline{f(a)} \approx 0.69365$

0.0005 $\quad \underline{0.0005}$

Andrew Ng

Basics of Neural Network Programming

Computation Graph

deeplearning.ai

Computation Graph

$$J(a,b,c) = \underbrace{3(\underbrace{a + \underbrace{bc}_{u}}_{v})}_{J} = 3(5 + 3\times 2) = 33$$

$u = bc$

$v = a + u$

$J = 3v$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Derivatives with a Computation Graph

Computing derivatives

$a = 5$ $\quad \frac{dJ}{da}$ "da" = 3

$b = 3$

$c = 2$

$u = bc$ (6)

$v = a + u$ (11)

$J = 3v$ (33)

$\frac{dJ}{dv}$ "dv" = 3

$\frac{dJ}{dv} = ? = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da}$

3×1

$\frac{dv}{da} = 1$

$a \to v \to J$

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$\to v = 11 \to 11.001$

$J = 33 \to 33.003$

$\frac{d\,\text{FinalOutputVar}}{d\,\text{var}}$

$dJ\,dvar$

"dvar"

$f(a) = 3a$

$\frac{df(a)}{da} = \frac{df}{da} = 3$

$J = 3v$

$\frac{dJ}{dv} = 3$

Andrew Ng

Computing derivatives

$a = 5$ $\quad \frac{dJ}{da} \to da = 3$

$b = 3$ $\quad \frac{dJ}{db} = db = 6$

$c = 2$ $\quad dc = 9$

$\boxed{u = bc}$ 6 $\quad du = 3$

$\boxed{v = a + u}$ 11 $\quad dv = 3 \quad \frac{dJ}{dv}$

$\boxed{J = 3v}$ 33

$$\frac{dJ}{du} = 3 = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$$

$$\frac{dJ}{db} = \underbrace{\boxed{\frac{dJ}{du}}}_{\to 3} \cdot \underbrace{\frac{du}{db}}_{=2} = 6$$

$$\frac{dJ}{da} = \underbrace{\boxed{\frac{dJ}{du}}}_{3 \times 3} \cdot \frac{du}{da} = 9$$

$u = 6 \to 6.001$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$b = 3 \to 3.001$

$u = b \cdot c = 6 \to 6.002$ $\quad c = 2$

$J = 33.006$ $\quad .006$

$v = 11.002$

$J = 3v$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression Gradient descent

Logistic regression recap

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

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

Andrew Ng

Logistic regression derivatives

$$dz = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$$

$$= a - y$$

$$= \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$$

$$a(1-a)$$

$$\text{"}da\text{"} = \frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz.$$

$$dw_2 = x_2 \cdot dz. \quad db = dz.$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db.$$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

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

$$(x^{(i)}, y^{(i)})$$

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

$$dw_1^{(i)}, \quad dw_2^{(i)}, \quad db^{(i)}$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)} \,-\, (x^{(i)}, y^{(i)})}$$

Andrew Ng

Logistic regression on m examples

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$

For $i = 1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$n = 2$

dw_3
$\vdots$
dw_n

$J /= m$

$dw_1 /= m;\ dw_2 /= m;\ db /= m.$

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db.$

Vectorization

Andrew Ng

Basics of Neural Network Programming

Vectorization

deeplearning.ai

What is vectorization?

$w \in \mathbb{R}^{n_x}$

$x \in \mathbb{R}^{n_x}$

$$z = \underbrace{w^T x}\ + b \qquad w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \qquad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]

z += b
```

Vectorized

$$z = \underbrace{np.\underline{dot}(w, x)}_{w^T x} + b$$

→ GPU
→ CPU

} SIMD - single instruction multiple data.

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

More vectorization examples

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$u = Av$

$u_i = \sum_i \sum_j A_{ij} v_j$

```
u = np.zeros((n,1))
for i ...
    for j ...
        u[i] += A[i][j] * v[j]
```

```
u = np.dot(A,v)
```

Andrew Ng

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

```
import numpy as np
u = np.exp(v)

np.log(v)
np.abs(v)
np.maximum(v, 0)
v**2
1/v
```

Andrew Ng

Logistic regression derivatives

$J = 0$, $\cancel{dw1 = 0, dw2 = 0}$, $db = 0$

$dw = np.zeros((n_x, 1))$

for i = 1 to n:

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

$dz^{(i)} = a^{(i)}(1 - a^{(i)})$

for $j = 1 \ldots n_x$

$dw_j \mathrel{+}= \ldots$

$\cancel{dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}}$

$\cancel{dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}}$

$n_x = 2$

$dw \mathrel{+}= x^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m$, $\cancel{dw_1 = dw_1/m, dw_2 = dw_2/m}$, $db = db/m$

$dw /= m.$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

$$z^{(1)} = \boxed{w^T x^{(1)} + b}$$
$$a^{(1)} = \sigma(z^{(1)})$$

$$z^{(2)} = \boxed{w^T x^{(2)} + b}$$
$$a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = w^T x^{(3)} + b$$
$$a^{(3)} = \sigma(z^{(3)})$$

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

(n_x, m)

$\mathbb{R}^{n_x \times m}$

$$w^T \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$Z = \boxed{\begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix}} = w^T X + \underbrace{\begin{bmatrix} b & b & \cdots & b \end{bmatrix}}_{1 \times m} = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots & w^T x^{(m)} + b \end{bmatrix}$$

$1 \times m$

$$Z = np.dot(w.T, X) + b$$

$(1,1)$ $\mathbb{R}$

"Broadcasting"

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \cdots & a^{(m)} \end{bmatrix} = \sigma(Z)$$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ $\quad \ldots$

$$\boxed{dZ = [\underbrace{dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}}_{1\times m}]}$$

$A = [a^{(1)} \cdots a^{(m)}]$. $\quad Y = [y^{(1)} \cdots y^{(m)}]$

$\rightarrow dZ = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \quad \ldots]$

$\rightarrow dw = 0$

$dw \mathrel{+}= x^{(1)} dz^{(1)}$

$dw \mathrel{+}= x^{(2)} dz^{(2)}$

$\vdots$

$dw /= m$

$db = 0$

$db \mathrel{+}= dz^{(1)}$

$db \mathrel{+}= dz^{(2)}$

$\vdots$

$db \mathrel{+}= dz^{(m)}$

$db /= m.$

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$= \frac{1}{m}\ \text{np.sum}(dZ)$$

$$dw = \frac{1}{m} X dZ^T$$

$$= \frac{1}{m} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \underbrace{\left[x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)} \right]}_{n\times 1}$$

Andrew Ng

Implementing Logistic Regression

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

$\boxed{\text{for } i = 1 \text{ to } m:}$

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$dw \mathrel{+}= x^{(i)} * dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$

$db = db/m$

for iter in range(1000):

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m} np.sum(dZ)$

$w := w - \alpha dw$

$b := b - \alpha db$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Broadcasting in Python

Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

$$
\begin{array}{r c}
 & \begin{array}{cccc} \text{Apples} & \text{Beef} & \text{Eggs} & \text{Potatoes} \end{array} \\
\begin{array}{r} \text{Carb} \\ \text{Protein} \\ \text{Fat} \end{array} & \begin{bmatrix} 56.0 & 0.0 & 4.4 & 68.0 \\ 1.2 & 104.0 & 52.0 & 8.0 \\ 1.8 & 135.0 & 99.0 & 0.9 \end{bmatrix}
\end{array}
= A
$$

(3,4)

0

1

59 cal

$\frac{56}{59} \approx 94.9\%$

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

```
cal = A.sum(axis = 0)
percentage = 100*A/(cal.reshape(1,4))
```

(3,4) / (1,4)

Broadcasting example

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} \cancel{100}$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}$$

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

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix} =$$

(m,n) $(m,1)$ (m,n)

General Principle

(m, n) matrix $\quad + \; - \; * \; /$

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

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

$(m, 1) \quad + \quad \mathbb{R}$

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

$[1 \;\; 2 \;\; 3] \quad + \quad 100 \quad = \quad [101 \;\; 102 \;\; 103]$

Matlab/Octave: bsxfun

Basics of Neural Network Programming

deeplearning.ai

Explanation of logistic regression cost function (Optional)

Logistic regression cost function

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

Interpret $\hat{y} = P(y=1|x)$

If $y=1$: $P(y|x) = \hat{y}$

If $y=0$: $P(y|x) = 1 - \hat{y}$

Andrew Ng

Logistic regression cost function

$$\boxed{\text{If} \quad y = 1: \qquad p(y|x) = \hat{y}}$$

$$\boxed{\text{If} \quad y = 0: \qquad p(y|x) = 1 - \hat{y}}$$

$p(y|x)$

$$\boxed{p(y|x) = \hat{y}^{y} \, (1-\hat{y})^{(1-y)}}$$

If $y=1$: $p(y|x) = \hat{y} \; \underbrace{(1-\hat{y})^{0}}_{=1}$

If $y=0$: $p(y|x) = \underbrace{\hat{y}^{0}}_{=1} \; (1-\hat{y})^{(1-y)} = 1 \times (1-\hat{y}) = 1-\hat{y}$

$$\uparrow \log p(y|x) = \log \hat{y}^{y} (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log(1-\hat{y})$$

$$= -\mathcal{L}(\hat{y}, y) \downarrow$$

Andrew Ng

Cost on m examples

$$\log p(\text{labels in training set}) = \log \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)})$$

$$\log p(\ldots\ldots) = \sum_{i=1}^{m} \underbrace{\log p(y^{(i)} \mid x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)},\, y^{(i)})}$$

$$= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Maximum likelihood estimation

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

Andrew Ng

Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode

deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?

deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation

$a^{[0]} = X$

$a^{[1]}$ — $W^{[1]}, b^{[1]}$

$(4,3)$ $(4,1)$

"2 layer NN"

$W^{[2]}, b^{[2]}$

$(1,4)$ $(1,1)$

$\hat{y} = a^{[2]}$

"$\hat{y} = a$"

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

→ Input layer

→ Hidden layer

→ Output layer

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Computing a Neural Network's Output

Neural Network Representation

$$z = w^T x + b$$

$$a = \sigma(z)$$

Andrew Ng

Neural Network Representation

$$z = w^T x + b$$

$$a = \sigma(z)$$

Andrew Ng

Neural Network Representation

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$(w_1^{[1]})^T x$ $\quad a^{[1]}$

$$\rightarrow z^{[1]} = \underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}} = \begin{bmatrix} \rightarrow w_1^{[1]T} x + b_1^{[1]} \\ \rightarrow w_2^{[1]T} x + b_2^{[1]} \\ \rightarrow w_3^{[1]T} x + b_3^{[1]} \\ \rightarrow w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$(4,3)$ $\quad (4,1)$

$$\rightarrow a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Andrew Ng

Neural Network Representation learning

$w^T = W^{[2]}$
$b = b^{[2]}$

$\hat{y} = a^{[2]}$

$x = a^{[0]}$

$W^{[2]}, b^{[2]}$
(1,4) (1,1)

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

Given input x:

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$
(4,1) (4,1)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples

$$
\begin{aligned}
z^{[1]} &= W^{[1]}x + b^{[1]} \\
a^{[1]} &= \sigma(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]})
\end{aligned}
$$

$$
\begin{aligned}
x &\longrightarrow a^{[2]} = \hat{y} \\
x^{(1)} &\longrightarrow a^{[2](1)} = \hat{y}^{(1)} \\
x^{(2)} &\longrightarrow a^{[2](2)} \quad \hat{y}^{(2)} \\
&\vdots \\
x^{(m)} &\longrightarrow a^{[2](m)} \quad \hat{y}^{(m)}
\end{aligned}
$$

$a^{[2](i)}$ ← example i

layer 2

for $i = 1$ to m,

$$
\begin{aligned}
z^{[1](i)} &= W^{[1]}x^{(i)} + b^{[1]} \\
a^{[1](i)} &= \sigma(z^{[1](i)}) \\
z^{[2](i)} &= W^{[2]}a^{[1](i)} + b^{[2]} \\
a^{[2](i)} &= \sigma(z^{[2](i)})
\end{aligned}
$$

Andrew Ng

Vectorizing across multiple examples

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

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

(n_x, m)

training examples

hidden units.

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

hidden units

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}}, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}}, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}}$$

$$\to 0 \qquad \to 0 \qquad \to 0$$

$$W^{[1]} = \begin{bmatrix} \text{—} \\ \text{—} \\ \text{—} \\ \text{—} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{X} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

$$+ b^{[1]} \qquad + b^{[1]} \qquad + b^{[1]}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$W^{[1]}x^{(i)} = z^{[1](i)}$$

Andrew Ng

Recap of vectorizing across multiple examples

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

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

```
for i = 1 to m
```

$$\boxed{z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$A^{[0]}$ $\quad X = a^{[0]} \quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \leftarrow W^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

One hidden layer Neural Network

Activation functions

deeplearning.ai

Activation functions

$g^{[1]}(z^{[1]}) = \tanh(z^{[1]})$

Sigmoid

$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

$\rightarrow y \in \{0, 1\}$

$0 \le \hat{y} \le 1$

$-1 \quad 1$

$a = \frac{1}{1+e^{-z}}$

$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\rightarrow \quad a^{[1]} = \cancel{\sigma(z^{[1]})} \quad g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$\rightarrow \quad a^{[2]} = \cancel{\sigma(z^{[2]})} \quad g^{[2]}(z^{[2]})$$

$z > 0$

$a = \max(0, z)$

ReLU

Rectified Linear Unit

Leaky ReLU

Andrew Ng

Pros and cons of activation functions

sigmoid: $a = \dfrac{1}{1+e^{-z}}$

tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{z}}$

ReLU $a = \max(0, z)$

$\max(0.01z, z)$

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

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function

$y \in \mathbb{R}$

$\hat{y} \geq 0$.

\$0 ..... \$1,000,000s

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \cancel{g^{[1]}(z^{[1]})} \quad z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \cancel{g^{[2]}(z^{[2]})} \quad z^{[2]}$$

$g(z) = z$

"linear activation function"

$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}\underbrace{\left(W^{[1]}x + b^{[1]}\right)}_{a^{[1]}} + b^{[2]}$$

$$= \underbrace{\left(W^{[2]}W^{[1]}\right)}_{W'}x + \underbrace{\left(W^{[2]}b^{[1]} + b^{[2]}\right)}_{b'}$$

$$= W'x + b'$$

$g(z) = z$

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function

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

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

$z = 10. \quad g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10 \quad g(z) \approx 0$

$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0 \quad g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1-\frac{1}{2}\right) = \frac{1}{4}$

$$\boxed{g'(z)} = \boxed{\frac{d}{dz} g(z)} = \text{slope of } g(x) \text{ at } z$$

$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)\,(1-g(z)) \leftarrow$$

$$= \boxed{a\,(1-a)}$$

$g'(z) = a(1-a)$

Andrew Ng

Tanh activation function

$$g(z) = \tanh(z)$$

$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= 1 - (\tanh(z))^2$$

$z = 10 \quad \tanh(z) \approx 1$

$g'(z) \approx 0$

$z = -10 \quad \tanh(z) \approx -1$

$g'(z) \approx 0$

$z = 0 \quad \tanh(z) = 0$

$g'(z) = 1$

$a = g(z), \quad g'(z) = 1 - a^2$

Andrew Ng

ReLU and Leaky ReLU

ReLU

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

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \cancel{\text{undef}} & \cancel{\text{if } z = 0} \end{cases}$$

$z = 0.0000\ldots0$

Leaky ReLU

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

Parameters: $\underset{(n^{[1]}, n^{[0]})}{W^{[1]}}, \underset{(n^{[1]}, 1)}{b^{[1]}}, \underset{(n^{[2]}, n^{[1]})}{W^{[2]}}, \underset{(n^{[2]}, 1)}{b^{[2]}}$ $\qquad n_x = n^{[0]}, \; n^{[1]}, \; \underline{n^{[2]} = 1}$

Cost function: $J(W^{[1]}, b^{[1]}, \underline{W^{[2]}}, \underline{b^{[2]}}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\underset{a^{[2]}}{\hat{y}}, y)$

Gradient descent:

→ Repeat {

→ Compute predicts $(\hat{y}^{(i)}, i = 1, \ldots, m)$

$\underline{dW^{[1]}} = \frac{\partial J}{\partial W^{[1]}}, \quad \underline{db^{[1]}} = \frac{\partial J}{\partial b^{[1]}}, \ldots$

$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$

$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$

$W^{[2]} := \ldots \qquad b^{[2]} := \ldots$

}

Andrew Ng

Formulas for computing derivatives

Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

Back propagation:

$$dZ^{[2]} = A^{[2]} - Y$$

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

$$(n^{[2]},)$$

$$(n^{[2]}, 1)$$

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

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

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

element-wise product

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

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

$$\underline{(n^{[1]}, 1)} \qquad (n^{[1]},)$$

reshape

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

Logistic regression

$$x, \; w, \; b \longrightarrow \boxed{z = w^T x + b} \longrightarrow \boxed{a = \sigma(z)} \longrightarrow \boxed{\mathcal{L}(a, y)}$$

$dz = a - y$

$dw = dz \cdot x$

$db = dz$

$$da = \frac{d}{da}\mathcal{L}(a,y) = -y\log a - (1-y)\log(1-a)$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\boxed{dz = da \cdot g'(z)}$$

$g(z) = \sigma(z)$

$$\boxed{\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \boxed{\frac{da}{dz}}}$$

"dz" = "da"

$$\frac{d}{dz} g(z) = g'(z)$$

Andrew Ng

Neural network gradients

$n_x = n^{[0]}$ $\quad n^{[1]}$ $\quad n^{[2]} = 1$

x, $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

$dW^{[1]}$, $db^{[1]}$, $dW^{[2]}$, $db^{[2]}$

$$z^{[1]} = W^{[1]}x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow z^{[2]} = W^{[2]}x + b^{[2]} \rightarrow a^{[2]} = \sigma(z^{[2]}) \rightarrow \mathcal{L}(a^{[2]}, y)$$

$\rightarrow dz^{[1]} = W^{[2]T} dz^{[2]}$ $\quad da^{[1]} = \ldots$ $\quad \rightarrow dz^{[2]} = a^{[2]} - y$ $\quad da^{[2]}$

$* \, g^{[1]\prime}(z^{[1]})$

element wise product

$\rightarrow dW^{[2]} = dz^{[2]} a^{[1]T}$ $\quad \rightarrow$ "$dw = dz \cdot x$"

$\rightarrow db^{[2]} = dz^{[2]}$

$W^{[1]} = [_____]$

$W^{[2]}$ $\quad (n^{[2]}, n^{[1]})$

$\rightarrow z^{[2]}, dz^{[2]}$ $\quad (n^{[2]}, 1)$ $\quad -(1,1)$

$\rightarrow z^{[1]}, dz^{[1]}$ $\quad (n^{[1]}, 1)$

foo $\quad W^{[1]}$ $\quad dW^{[1]} = dz^{[1]} \cdot x^T$

dfoo $\quad dW^{[2]}$ $\quad db^{[1]} = dz^{[1]}$ $\quad a^{[0]T}$

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

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]}) \, (n^{[2]}, 1) * (n^{[1]}, 1)$

Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

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

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

Vectorized Implementation:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$\underset{(n^{[1]},1)}{dz^{[1]}} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$J(\ldots) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

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

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

elementwise product

$$\underset{(n^{[1]},m)}{dZ^{[1]}} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

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

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

Andrew Ng

One hidden layer Neural Network

Random Initialization

deeplearning.ai

What happens if you initialize weights to zero?

$W^{[1]} = [0 \ 0]$

Symmetric

$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$W^{[1]} = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}$

$a_1^{[1]} = a_2^{[1]}$ $\quad dz_1^{[1]} = dz_2^{[1]}$

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\quad W^{[1]} = W^{[1]} - \alpha dW$

Andrew Ng

Random initialization

$W^{[1]} = np.random.randn((2,2)) * 0.01$

$b^{[1]} = np.zeros((2,1))$

$W^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$

100?

$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

Andrew Ng

Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode

deeplearning.ai

Deep Neural Networks

Deep L-layer Neural network

What is a deep neural network?

"1 layer NN"

"shallow"

logistic regression

"2 layer" NN

1 hidden layer

2 hidden layers

"deep"

5 hidden layers

Andrew

Deep neural network notation

4 layer NN

$X = a^{[0]}$

$L = 4$ (#layers)

$n^{[l]}$ = #units in layer l

$n^{[1]} = 5$, $n^{[2]} = 5$, $n^{[3]} = 3$, $n^{[4]} = n^{[L]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activations in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$, $W^{[l]}$, $b^{[l]}$ = weights for $z^{[l]}$

Andrew N

deeplearning.ai

Deep Neural Networks

Forward Propagation in a Deep Network

Forward propagation in a deep network

$$\boxed{\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned}}$$

$A^{[0]} = X$

X : $z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$

$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$, $a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$

$\begin{bmatrix} | & | & & | \\ z^{[2](1)} & z^{[2](2)} & \cdots & z^{[2](m)} \\ | & | & & | \end{bmatrix}$

Vectorized:

$X = A^{[0]}$

$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]})$

$\hat{Y} = g(Z^{[4]}) = A^{[4]}$

for $l = 1 \ldots 4$

Andrew N

deeplearning.ai

Deep Neural Networks

Getting your matrix dimensions right

Parameters $W^{[l]}$ and $b^{[l]}$

$L = 5$

$z^{[l]} = g^{[l]}(a^{[l]})$

$a^{[l]}$

$$\begin{cases} W^{[l]} : (n^{[l]}, n^{[l-1]}) \\ b^{[l]} : (n^{[l]}, 1) \\ dW^{[l]} : (n^{[l]}, n^{[l-1]}) \\ db^{[l]} : (n^{[l]}, 1) \end{cases}$$

$$z^{[1]} = \boxed{W^{[1]} \cdot x} + \boxed{b^{[1]}}$$

$(3,1) \leftarrow (3,2) \quad (2,1) \quad (3,1)$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, 1) \quad (n^{[1]}, 1)$

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

$W^{[1]} : (n^{[1]}, n^{[0]})$

$W^{[2]} : (5, 3) \quad (n^{[2]}, n^{[1]})$

$$z^{[2]} = \boxed{W^{[2]} \cdot a^{[1]}} + \boxed{b^{[2]}}$$

$\rightarrow (5,1) \quad (5,3) \quad (3,1) \quad (5,1)$

$(n^{[2]}, 1)$

$W^{[3]} : (4, 5)$

$W^{[4]} : (2, 4) \quad , \quad W^{[5]} : (1, 2)$

Andrew Ng

Vectorized implementation

$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$(n^{[1]}, 1)$ $(n^{[1]}, n^{[0]})$ $(n^{[0]}, 1)$ $(n^{[1]}, 1)$

$$\begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$(n^{[1]}, m)$ $(n^{[1]}, n^{[0]})$ $(n^{[0]}, m)$ $(n^{[1]}, 1)$

$(n^{[1]}, m)$

$z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l = 0 \quad A^{[0]} = X = (n^{[0]}, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

Andrew Ng

deeplearning.ai

Deep Neural Networks

Why deep representations?

Intuition about deep representation

Andrew Ng

Circuit theory and deep learning

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

Andrew Ng

deeplearning.ai

Deep Neural Networks

Building blocks of deep neural networks

Forward and backward functions

Layer l: $W^{[l]}, b^{[l]}$

→ Forward: Input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

→ Backward: Input $da^{[l]}$, output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

cache ($z^{[l]}$)

Andrew N

Forward and backward functions

Andrew N

deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Backward propagation for layer l

- Input $da^{[l]}$
- Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

$$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$$

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

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

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

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Summary

X
ReLU
ReLU
Sigmoid
$\hat{y}$
$z^{[1]}$
$z^{[2]}$
$z^{[3]}$
$\mathcal{L}(\hat{y}, y)$
$da^{[0]}$
$da^{[1]}$
$da^{[2]}$
$da^{[3]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$
$dW^{[1]}$
$db^{[1]}$
$dW^{[2]}$
$db^{[2]}$
$dW^{[3]}$
$db^{[3]}$
$dA^{[3]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \cdots - \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right.$

deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

What are hyperparameters?

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyperparameters: learning rate α

#iterations

hidden layers L

hidden units $n^{[1]}, n^{[2]}, \ldots$

choice of activation function

later: Momentum, mini-batch size, regularizations, ...

Andrew Ng

Applied deep learning is a very empirical process

Idea

$\alpha = \cancel{0.01}$
0.05

Experiment

Code

Vision, Speech, NLP, Ad, Search, Recommendation.

Andrew Ng

deeplearning.ai

Deep Neural Networks

What does this have to do with the brain?

Forward and backward propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}\left(Z^{[L]}\right) = \hat{Y}
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

"It's like the brain."

$X \longrightarrow y$

Andrew Ng

Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode

deeplearning.ai

Setting up your ML application

Train/dev/test sets

Applied ML is a highly iterative process

- # layers
- # hidden units
- learning rates
- activation functions
- …

NLP, Vision, Speech, Structural data

Ads Search Security logistic ....

Andrew Ng

Train/dev/test sets

Data | | dev | test |

training set

- Hold-out cross validation
- Development set "dev"

test

train test

Previous era: 70/30 %

60/20/20 %

100 - 1000 - 10000

Big data: 1,000,000 10,000 10,000

98/1/1 %

99.5 / ~~.25~~ .4 / ~~.25~~ .1 %

Andrew Ng

Mismatched train/test distribution

Cats

Training set:
Cat pictures from webpages

Dev/test sets:
Cat pictures from users using your app

→ Make sure dev and test come from same distribution.

"test"

train / dev

train / test

→ train / dev

Not having a test set might be okay. (Only dev set.)

Andrew Ng

deeplearning.ai

Setting up your ML application

Bias/Variance

Bias and Variance

high bias

underfitting

"just right"

high variance

overfitting

Andrew Ng

Bias and Variance

Cat classification

y=1

y=0

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance

Human: ≈0%

Optimal (Bayes) error: ~~≈0%~~ 15%

Blurry images

Andrew Ng

High bias and high variance

Andrew Ng

deeplearning.ai

Setting up your ML application

Basic “recipe” for machine learning

Basic recipe for machine learning

High bias?
(training data performance)

Y → Bigger network
→ Train longer.
(NN architecture search)

N ↓

High variance?
(dev set performance)

Y → More data
→ Regularization
(NN architecture search)

N ↓

Done

Bias ↓ ↑ ↓

variance ↓ ↓ ↑

tradeoff"

Andrew Ng

deeplearning.ai

Regularizing your neural network

Regularization

Logistic regression

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

λ = regularization parameter

lambda lambd

$$\min_{w,b} J(w,b)$$

$$J(w,b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \frac{\lambda}{2m}\underbrace{\|w\|_2^2}$$

$$\cancel{+ \frac{\lambda}{2m} b^2}$$

omit

L_2 regularization $\quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L_1 regularization $\quad \frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m}\|w\|_1$

w will be sparse

Andrew Ng

Neural network

$$\rightarrow J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2}$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

$$w^{[l]} : (n^{[l]}, n^{[l-1]})$$

"Frobenius norm" $\|\cdot\|_2^2$ $\|\cdot\|_F^2$

$$dw^{[l]} = \boxed{(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}}$$

$$\underbrace{\frac{\partial J}{\partial w^{[l]}}} = dw^{[l]}$$

$$\rightarrow w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

"Weight decay"

$$w^{[l]} := w^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]}\right]$$

$$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha\,(\text{from backprop})$$

$$= \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right)}_{<1} w^{[l]} - \alpha\,(\text{from backprop})$$

Andrew Ng

deeplearning.ai

Regularizing your neural network

Why regularization reduces overfitting

How does regularization prevent overfitting?

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

$w^{[l]} \approx 0$

Andrew Ng

How does regularization prevent overfitting?

tanh

$g(z) = \tanh(z)$

$\lambda \uparrow$ $\quad W^{[l]} \downarrow$ $\quad z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

Every layer $\approx$ linear.

$$J(\ldots) = \boxed{\sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \frac{\lambda}{2m} \sum_l \|W^{[l]}\|_F^2$$

Andrew Ng

deeplearning.ai

Regularizing your neural network

Dropout regularization

Dropout regularization

Andrew Ng

Implementing dropout ("Inverted dropout")

Illustrate with layer $l=3$. keep-prob $= 0.8$ $\quad 0.2$

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\text{-}prob$

$a3 = np.multiply(a3, d3)$ $\quad$ # $a3 \mathrel{*}= d3$.

$a3 \mathrel{/}= \cancel{0.8}\ keep\text{-}prob$

50 units. ⇝ 10 units shut off

$z^{[4]} = W^{[4]} \cdot a^{[3]} + b^{[4]}$

reduced by 20%.

$\mathrel{/}= 0.8$

Test

Andrew Ng

Making predictions at test time

$a^{[0]} = X$

No drop out.

$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = \ldots$

$\downarrow$

$\hat{y}$

/= keep-prob

Andrew Ng

deeplearning.ai

Regularizing your neural network

Understanding dropout

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights. ⇝ Shrink weights. L_2

→ Computer Vision

Downside: J

Andrew Ng

deeplearning.ai

Regularizing your neural network

Other regularization methods

Data augmentation

Andrew Ng

Early stopping

Orthogonalization.

- Optimize cost function J
  - Gradient, ...
- Not overfit.
  - Regularization, ...

L_2

$J(w,b)$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Normalizing inputs

Normalizing training sets

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

Subtract mean:

$$\boxed{\mu} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\boxed{x := x - \mu}$$

Normalize variance

$$\boxed{\sigma^2} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2$$

← element-wise

$$\boxed{x /= \sigma^2}$$

Use same μ, σ^2 to normalize test set.

Andrew Ng

Why normalize inputs?

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

w_1 x_1: 1 ... 1000

w_2 x_2: 0 ... 1

Unnormalized:

$-1 \ldots 1$

Normalized:

x_1: 0 ... 1

x_2: $-1 \ldots 1$

x_3: 1 ... 2

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Vanishing/exploding gradients

Vanishing/exploding gradients

$L = 150$

L

$g(z) = z$. $\quad b^{[l]} = 0$.

$\hat{y} = W^{[L]} W^{[L-1]} W^{[L-2]} \cdots W^{[3]} W^{[2]} W^{[1]} x$

$a^{[3]}$

1.5^L

0.5^L

$W^{[l]} > I$

$W^{[l]} < I \quad \begin{bmatrix} 0.9 & \\ & 0.9 \end{bmatrix}$

$z^{[1]} = W^{[1]} x$

$a^{[1]} = g(z^{[1]}) = z^{[1]}$

$a^{[2]} = g(z^{[2]}) = g(W^{[2]} a^{[1]})$

$$W^{[l]} = \begin{bmatrix} \cancel{1.5}\ 0.5 & 0 \\ 0 & \cancel{1.5}\ 0.5 \end{bmatrix}$$

$$\hat{y} = W^{[L]} \begin{bmatrix} \cancel{1.5}\ 0.5 & 0 \\ 0 & \cancel{1.5}\ 0.5 \end{bmatrix}^{L-1} x$$

$1.5^{L-1} x$

$0.5^{L-1} x$

Andrew Ng

Single neuron example

$w^{[l]}$

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$ $\cancel{+b}$

large n → smaller w_i

$Var(w_i) = \cancel{\frac{1}{n}} \frac{2}{n}$

$w^{[l]} = np.random.randn(shape) * \boxed{np.sqrt\left(\frac{2}{n^{[l-1]}}\right)}$

ReLU $g^{[l]}(z) = ReLU(z)$

Other variants:

tanh $\boxed{\sqrt{\frac{1}{n^{[l-1]}}}}$

Xavier initialization

$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Numerical approximation of gradients

Checking your derivative computation

$f(\theta) = \theta^3$

$\theta \in \mathbb{R}$.

$g(\theta) = \frac{d}{d\theta} f(\theta) = f'(\theta)$

$g(\theta) = 3\theta^2$.

$g(\theta) = 3 \cdot (1)^2 = 3$ when $\theta = 1$

$\frac{dJ}{db}$

$$\boxed{\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon} \approx g(\theta)}$$

$\theta = 1$

$\theta + \varepsilon = 1.01$

$$\frac{(1.01)^3 - 1^3}{0.01} = 3.0301 \approx 3$$

0.0301

3.1

3.2

Andrew Ng

Checking your derivative computation

$f(\theta) = \theta^3$

$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\frac{(1.01)^3 - (0.99)^3}{2(0.01)} = 3.0001 \approx 3$$

$$g(\theta) = 3\theta^2 = 3$$

approx error: 0.0001

(prev slide: 3.0301. error: 0.03)

$$f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon}$$

$O(\varepsilon^2)$

0.01

0.0001

$$\frac{f(\theta+\varepsilon) - f(\theta)}{\varepsilon}$$

error: $O(\varepsilon)$

0.01

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

Take $W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

concatenate

$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = J(\theta)$

Take $dW^{[1]}, db^{[1]}, \ldots, dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

concatenate

Is $d\theta$ the gradient of $J(\theta)$?

Andrew Ng

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3, \ldots)$

for each i:

$$\rightarrow d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \ldots, \theta_i + \varepsilon, \ldots) - J(\theta_1, \theta_2, \ldots, \theta_i - \varepsilon, \ldots)}{2\varepsilon}$$

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

$$d\theta_{approx} \overset{?}{\approx} d\theta$$

Check

$$\rightarrow \frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$$

$\varepsilon = 10^{-7}$

$\approx \boxed{10^{-7} \text{ – great!}} \leftarrow$

10^{-5}

$\rightarrow 10^{-3}$ – worry. $\leftarrow$

Andrew Ng

deeplearning.ai

Setting up your optimization problem

Gradient Checking implementation notes

Gradient checking implementation notes

- Don't use in training – only to debug

$d\theta_{approx}[i] \longleftrightarrow d\theta[i]$

- If algorithm fails grad check, look at components to try to identify bug.

$db^{[l]}$ $dW^{[l]}$

$J(\theta) = \frac{1}{m}\sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_l \|W^{[l]}\|_F^2$

- Remember regularization.

$d\theta$ = gradient of J w.r.t. θ

- Doesn't work with dropout.

J keep-prob = 1.0

- Run at random initialization; perhaps again after some training.

$W, b \approx 0$

Andrew Ng