

Stanford CS230 Notes

Contents

1	Introduction to Deep Learning	2
1.1	What is a Neural Network?	2
1.2	Supervised Learning with Neural Networks	2
1.3	Why is Deep Learning taking off?	2
2	Basics of Neural Network Programming	3
2.1	Binary Classification	3
2.1.1	Binary Classification	3
2.1.2	Notation	3
2.2	Logistic Regression	3
2.3	Logistic Regression cost function	4
2.4	Gradient Descent	5
2.5	Derivatives	5
2.6	Computation Graph	6
2.7	Derivatives with a Computation Graph	6
2.8	Logistic Regression Gradient Descent	7
2.8.1	Logicstic regression recap	7
2.8.2	Logistic regression derivatives	7
2.9	Gradient Descent on m examples	8
2.10	Vectorization	8
2.11	More Vectorization Examples	8
2.11.1	Neural network programming guideline	8
2.11.2	Vectors and matrix valued functions	8
2.12	Vectorizing Logistic Regression	9

2.13	Vectorizing Logistic Regression's Gradient Computation . . .	9
2.14	Broadcasting in Python	9
2.15	Explanation of logistic regression cost function (Optional) . .	9
2.15.1	Logistic regression cost function	9
2.15.2	Cost on m examples	9
3	Deep Reinforcement Learning	11
3.1	Motivation	11
3.1.1	Why RL	11
3.1.2	What is RL	11
3.1.3	Examples	11
3.2	Recycling Game	11
3.2.1	Q learning	11
3.2.2	Summary	12
3.3	Deep Q-Networks	13
3.3.1	Main idea	13
3.3.2	Deep Q Learning	13
3.4	Application of Deep Q-Network: Breakout (Ataria)	13
3.5	Advanced topics	13

1 Introduction to Deep Learning

1.1 What is a Neural Network?

neuron, links.

1.2 Supervised Learning with Neural Networks

Supervised Learning

Examples: Standard NN, Convolutional NN, Recurrent NN

Structured Data: tabular data; Unstructured Data: audio/image/text

1.3 Why is Deep Learning taking off?

Amount of labeled data.

2 Basics of Neural Network Programming

2.1 Binary Classification

2.1.1 Binary Classification

image \rightarrow 1 (cat) vs 0 (non cat)

2.1.2 Notation

m : number of examples

n_x : input size

n_y : output size

x : input, column vector

y : output, 0/1

X : input matrix, shape = (n_x, m)

Y : output matrix, shape = $(1, m)$

$x^{(i)}$: superscript (i) will denote the i^{th} example.

2.2 Logistic Regression

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

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

Output:

$$z = w^T x + b \tag{1}$$

$$\hat{y} = \sigma(z) \tag{2}$$

$$\sigma(z) \approx \frac{1}{1 + e^{-z}} \tag{3}$$

$$z \approx \infty, \sigma(z) \approx \frac{1}{1 + 0} = 1 \tag{4}$$

$$z \approx -\infty, \sigma(z) \approx \frac{1}{1 + \infty} = 0 \tag{5}$$

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

$$\theta_0 = b, \theta_1 \dots \theta_{n_x} = w \quad (6)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_{n_x} \end{bmatrix} \quad (7)$$

$$\hat{y} = \sigma(\theta^T x) \quad (8)$$

2.3 Logistic Regression cost function

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

Loss (error) function (mean square error/cross entropy):

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \quad (9)$$

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \quad (10)$$

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

Cost function:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}, y) \quad (11)$$

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (12)$$

Given a random variable X with probability mass function $p_X(x)$, the self information of measuring X as outcome x is defined as:

$$I_X(x) = \log[p_X(x)] = \log\left(\frac{1}{p_X(x)}\right) \quad (13)$$

Shannon Entropy of X :

$$H(X) = \sum_x -p_X(x) \log p_X(x) \quad (14)$$

$$= \sum_x p_X(x) I_X(x) \quad (15)$$

$$= E[I_X(x)] \quad (16)$$

Cross Entropy of the the true distributions p and estimated distribution q :

$$H(p, q) = E_p[-\log q] = - \sum_{x \in \mathcal{X}} p(x) \log q(x) \quad (17)$$

2.4 Gradient Descent

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

Repeat:

$$w := w - \alpha \frac{dJ(w, b)}{dw} \quad (18)$$

$$b := b - \alpha \frac{dJ(w, b)}{db} \quad (19)$$

α : learning rate

2.5 Derivatives

$$f(a) = 3a, \frac{df(a)}{da} = 3 = \frac{d}{da} f(a) \quad (20)$$

$$f(a) = a^2, \frac{d}{da} f(a) = 2a \quad (21)$$

$$f(a) = a^3, \frac{d}{da} f(a) = 3a^2 \quad (22)$$

$$f(a) = \log_e a = \ln a, \frac{d}{da} f(a) = \frac{1}{a} \quad (23)$$

$$f(x) = \log_a x, \frac{d}{dx} f(x) = \frac{1}{x \ln a} \quad (24)$$

$$f(x) = a^x, \frac{d}{dx} f(x) = a^x \ln a \quad (25)$$

$$\log_a b = \frac{\log_c b}{\log_c a} = \frac{\ln b}{\ln a} \quad (26)$$

2.6 Computation Graph

$$J(a, b, c) = 3(a + bc) \quad (27)$$

$$= 3(a + u) \quad (28)$$

$$= 3v \quad (29)$$

$$u = bc \quad (30)$$

$$v = a + u \quad (31)$$

$$J = 3v \quad (32)$$

2.7 Derivatives with a Computation Graph

$$a = 5, b = 3, c = 2$$

$$\frac{dJ}{dv} = 3 \quad (33)$$

$$\frac{dJ}{da} = \frac{dJ}{dv} \frac{dv}{da} \quad (34)$$

$$= 3 * 1 \quad (35)$$

$$= 3 \quad (36)$$

$$\frac{dJ}{du} = \frac{dJ}{dv} \frac{dv}{du} \quad (37)$$

$$= 3 \quad (38)$$

$$\frac{dJ}{db} = \frac{dJ}{du} \frac{du}{db} \quad (39)$$

$$= 3 * c \quad (40)$$

$$= 3 * 2 \quad (41)$$

$$= 6 \quad (42)$$

$$\frac{dJ}{dc} = \frac{dJ}{du} \frac{du}{dc} \quad (43)$$

$$= 3 * b \quad (44)$$

$$= 3 * 3 \quad (45)$$

$$= 9 \quad (46)$$

2.8 Logistic Regression Gradient Descent

2.8.1 Logistic regression recap

$$z = w^T x + b \quad (47)$$

$$\hat{y} = a = \sigma(z) \quad (48)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a)) \quad (49)$$

2.8.2 Logistic regression derivatives

$$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow \mathcal{L}(a, y)$$

$$da = \frac{d\mathcal{L}(a, y)}{da} \quad (50)$$

$$= -\frac{y}{a} + \frac{1 - y}{1 - a} \quad (51)$$

$$dz = \frac{d\mathcal{L}(a, y)}{dz} \quad (52)$$

$$= \frac{d\mathcal{L}}{da} * \frac{da}{dz} \quad (53)$$

$$= \left(-\frac{y}{a} + \frac{1 - y}{1 - a}\right) * a * (1 - a) \quad (54)$$

$$= -(1 - a)y + a(1 - y) \quad (55)$$

$$= a - y \quad (56)$$

$$\frac{da}{dz} = a * (1 - a) \quad (57)$$

$$= \frac{1}{1 + e^{-z}} * \left(1 - \frac{1}{1 + e^{-z}}\right) \quad (58)$$

$$dw_1 = x_1 * dz \quad (59)$$

$$dw_2 = x_2 * dz \quad (60)$$

$$db = dz \quad (61)$$

Repeat:

$$w_1 := w_1 - \alpha dw_1 \quad (62)$$

$$w_2 := w_2 - \alpha dw_2 \quad (63)$$

$$b := b - \alpha db \quad (64)$$

2.9 Gradient Descent on m examples

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a^{(i)}, y(i)) \quad (65)$$

$$\frac{d}{dw_1} J(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{d}{dw_1} \mathcal{L}(a^{(i)}, y(i)) \quad (66)$$

$$= \frac{1}{m} \sum_{i=1}^m dw_1^{(i)} \quad (67)$$

2.10 Vectorization

Vectorized, for GPU.

```
z = np.dot(w, x) + b
```

2.11 More Vectorization Examples

2.11.1 Neural network programming guideline

Whenever possible, avoid explicit for-loops.

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

2.11.2 Vectors and matrix valued functions

```
import numpy as np
```

```
u = np.exp(v)
np.log(v)
np.abs(v)
np.maximum(v, 0)
np.pow(v, 2)
np.divide(1, v)
```


2.12 Vectorizing Logistic Regression

2.13 Vectorizing Logistic Regression's Gradient Computation

2.14 Broadcasting in Python

```
np.add((1.0, 2.0), 2.0)    # (3.0, 4.0)
```

2.15 Explanation of logistic regression cost function (Optional)

2.15.1 Logistic regression cost function

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

$$\hat{y} = p(y=1|x) \quad (68)$$

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

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

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{(1-y)} \quad (69)$$

$$y = 1 : p(y|x) = \hat{y} \quad (70)$$

$$y = 0 : p(y|x) = 1 - \hat{y} \quad (71)$$

$$\log p(y|x) = y \log \hat{y} + (1 - y) \log 1 - \hat{y} \quad (72)$$

$$= -\mathcal{L}(\hat{y}, y) \quad (73)$$

2.15.2 Cost on m examples

Maximum likelihood estimation:

$$\log p(\dots) = \sum_{i=1}^m -\mathcal{L}(\hat{y}, y) \quad (74)$$

$$= -\sum_{i=1}^m \mathcal{L}(\hat{y}, y) \quad (75)$$

Minimize cost:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}, y) \quad (76)$$

3 Deep Reinforcement Learning

3.1 Motivation

3.1.1 Why RL

Making sequences of decisions.

Delayed labels. After you take the action, you will know the result.

3.1.2 What is RL

Automatically learn to make good sequences of decision.

3.1.3 Examples

Robotics

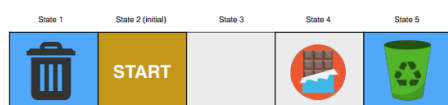
Games

Advertisement

3.2 Recycling Game

3.2.1 Q learning

Problem statement



Define reward “ r ” in every state



Best strategy to follow if $\gamma = 1$



Goal: maximize the return (rewards)

Number of states: 5

Types of states: initial normal terminal

Agent's Possible actions:

Additional rule: garbage collector coming in 3min, it takes 1min to move between states

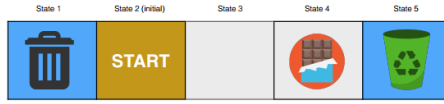
How to define the long-term return?

Discounted return $R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

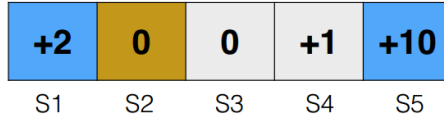
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

图 1: Game

Problem statement



Define reward “ r ” in every state



Assuming $\gamma = 0.9$

$$\text{Discounted return } R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

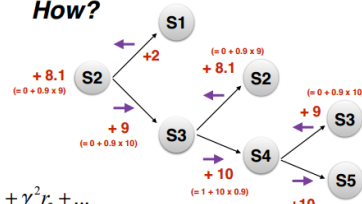
What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$Q = \begin{matrix} & \begin{matrix} \text{actions} \\ \rightarrow \end{matrix} \\ \begin{matrix} \rightarrow \\ \text{states} \end{matrix} & \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix} \end{matrix}$$

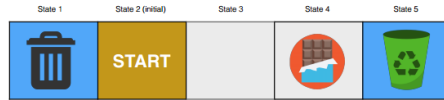
How?



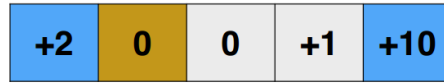
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

图 2: Q-table

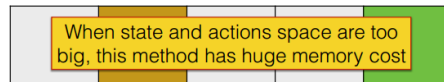
Problem statement



Define reward “ r ” in every state



Best strategy to follow if $\gamma = 0.9$



Function telling us our best strategy

What do we want to learn?

how good is it to take action 1 in state 2

Q-table

$$Q = \begin{matrix} & \begin{matrix} \text{actions} \\ \rightarrow \end{matrix} \\ \begin{matrix} \rightarrow \\ \text{states} \end{matrix} & \begin{pmatrix} 0 & 0 \\ 2 & 9 \\ 8.1 & 10 \\ 9 & 10 \\ 0 & 0 \end{pmatrix} \end{matrix}$$

Bellman equation
(optimality equation)

$$Q^*(s, a) = r + \gamma \max_{a'} (Q^*(s', a'))$$

Policy

$$\pi(s) = \arg \max_a (Q^*(s, a))$$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

图 3: Bellman equation and Policy

3.2.2 Summary

Vocabulary: environment, agent, state, action, reward, total return, discount factor/discount rate.

Q-table: matrix of entries representing “how good is it to take action a in state s”

Policy: function telling us what’s the best strategy to adopt

Bellman equation satisfied by the optimal Q-table

3.3 Deep Q-Networks

3.3.1 Main idea

Main idea: find a Q-function to replace the Q-table

Q-function: Neural Network.

3.3.2 Deep Q Learning

$$\hat{y} = Q(s, a) \tag{77}$$

$$L = (y - \hat{y})^2 \tag{78}$$

$$y = r + \gamma \max_{a'} (Q(s_a^{next}, a')) \tag{79}$$

Backpropagation: Compute $\frac{\partial L}{\partial W}$ and update W using stochastic gradient descent.

3.4 Application of Deep Q-Network: Breakout (Ataria)

- Deep Q-network architecture: CNN for image input.
- Exploration vs. Exploitation
- Implementation:
 - Preprocessing
 - Detect terminal State
 - Experience replay
 - Epsilon greedy action

3.5 Advanced topics

Policy Gradient Methods: PPO(Proximal Policy Optimization), TRPO(Trust Region Policy Optimization).