Supervised ML: Classification and Regression

@deepankar sharma

Saturday, February 4, 2023 12:04 PM

Applications of ML
1. Retail
2. Travel
3. Transportation
4. Automotive
5. Material Manufacturing

Machine Learning Algorithms
- → Supervised Learning
- → Unsupervised Learning
- → Recommender System
- → Reinforcement Learning

Supervised ML → "learn from labelled data"
1. Regression
2. Classification

$x \longrightarrow y$
Input Output

Unsupervised ML → "find something interesting in unlabelled data"
1. Clustering ; Google News , DNA microarray
2. Anomaly Detection
3. Dimensionality Reduction

Regression (Notation)
- x → input variable
- y → output variable
- m = # training examples

$(x^{(i)}, y^{(i)})$ → i th training example

Cost Function

$$\boxed{f_{w,b}(x) = wx + b}$$

w, b → parameters {adjust these}

cost function $J(w,b)$ $= \frac{1}{(2m)} \sum_{i=1}^{m} \underbrace{\left(\hat{y}^{(i)} - y^{(i)}\right)^2}_{\text{error}}$

↑ minimize $J(w,b)$ w,b

Gradient Descent

$J(w,b)$

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

Repeat until convergence:

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

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

size of the step α = learning rate

simultaneously update $\boxed{w \,\&\, b}$

Correct : simultaneous update

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

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

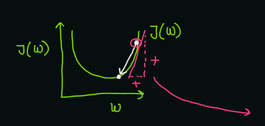
$$w = temp_w$$

$$b = temp_b$$

Incorrect :

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

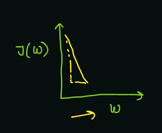
$$b = b - \alpha \frac{\partial}{\partial w} J(w,b)$$



$\min_w J(w)$

$$w = w - \alpha \underbrace{\frac{\partial}{\partial w} J(w)}_{+\text{ve term}}$$

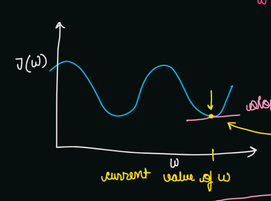
$w = w - \alpha$ (+ve number)



$\min_w J(w)$

$$w = w - \alpha \underbrace{\frac{\partial}{\partial w} J(w)}_{-\text{ve number}}$$

$w = w - \alpha$ (−ve number)



$$w = w - \alpha \boxed{\frac{\partial}{\partial w} J(w)}$$
$= 0$

$w = w - \alpha \cdot (0)$

$w = w$

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \cancel{2}\left(wx^{(i)} + b - y^{(i)}\right) * x^{(i)}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) * x^{(i)}$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \cancel{2}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

Multiple Features

x_j = j th feature

n = # features

$\vec{x}^{(i)}$ = i th training example

Vectorization

$$\boxed{np.dot(w, x) + b}$$

4 features

$f_{w,b}(x) = wx + b$

$f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

n features

$\vec{x} = [x_1, x_2, x_3, \ldots, x_n]$

$\vec{w} = [w_1, w_2, w_3, \ldots, w_n]$

$$\boxed{f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b}$$

dot product of vectors

"Multiple Linear Regression"

Not Multivariate Linear Regression

Model ⇒ $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

cost ⇒ $J(\vec{w}, b)$

gradient Descent ⇒ repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

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

}

Normal Equation
- → only for linear equation
- → solve w, b without iterations

Disadvantages
- → doesn't generalize well to other learning algorithms
- → slow when the # features is large (> 10,000)

n features

$\frac{\partial}{\partial w_1} J(\vec{w}, b)$

repeat {

$$w_1 = w_1 - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_1^{(i)}}$$

⋮

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)$$

simultaneously update w_j (for $j = 1, \ldots, n$) and b

}

Feature scaling

$300 \le x_1 \le 2000$

x_1 scaled $= \frac{x_1}{2000}$

↓

$0.15 \le x_1$ scaled ≤ 1

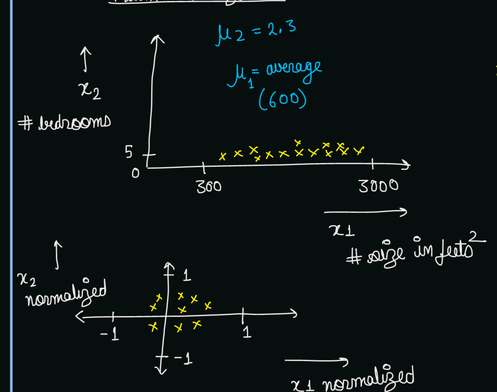
$0 \le x_2 \le 5$

x_2 scaled $= \frac{x_2}{5}$ max

↓

$0 \le x_2$ scaled ≤ 1

Mean Normalization



$300 \le x_1 \le 3000$

$$x_1 = \frac{x_1 - \mu_1}{3000 - 300} \quad (\max - \min)$$

↓

$-0.18 \le x_1 \le 0.82$

$0 \le x_2 \le 5$

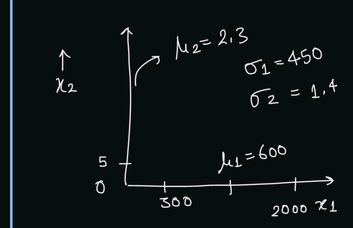
$$x_2 = \frac{x_2 - \mu_2}{5 - 0} \quad (\max - \min)$$

↓

$-0.46 \le x_2 \le 0.54$

σ → standard deviation

Z-score normalization



$300 \le x_1 \le 2000$

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

↓

$-0.67 \le x_1 \le 3.1$

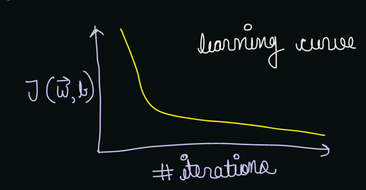
$0 \le x_2 \le 5$

↓

$x_2 = x_2 - \mu_2$

$-1.6 \le x_1 \le 1.9$

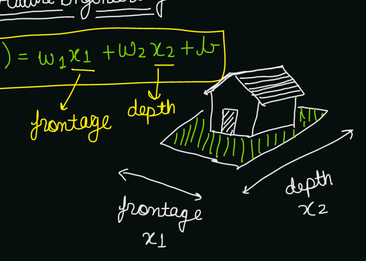
Checking gradient descent for convergence



Feature Engineering

$$\boxed{f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + b}$$

frontage, depth



∵ area = frontage * depth

$\boxed{x_3 = x_1x_3}$ new feature

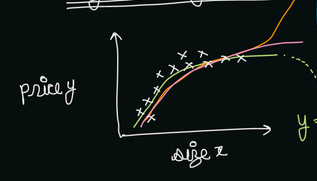
$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$

Polynomial Regression

$$\boxed{y = w_1x + w_2x^2 + w_3x^3 + b}$$ cubic function

size $1 - 10^3$, size^2 $1 - 10^6$, size^3 $1 - 10^9$

feature scaling



$y = w_1x + w_2x^2 + b$ → takin quadratic function fir gol ghoom jayega

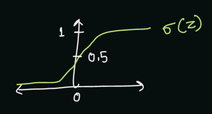
size, size^2

$y = w_1x + w_2\sqrt{x} + b$

Logistic Regression

sigmoid function / logistic function $\left(\sigma(z) = \frac{1}{1+e^{-z}}\right)$

$0 < \sigma(z) < 1$



$z = \vec{w} \cdot \vec{x} + b$

$\sigma(z)$ # probability of class is 1

Decision Boundary

$$\hat{y} = f_{\vec{w},b}(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

$= P(y = 1 \mid x; \vec{w}, b)$

if $\hat{y} \ge 0.5$; $\hat{y} = 1$	else; $\hat{y} = 0$

Cost Function

m → training examples

n → training features

squared error cost

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \boxed{\left(\hat{y}^{(i)} - y^{(i)}\right)^2}$$

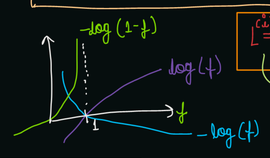
loss 'L'

Logistic Loss

$$L(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}), & \text{if } \hat{y}^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}), & \text{if } \hat{y}^{(i)} = 0 \end{cases}$$

Loss Function ⇒ 1 training example

Cost Function ⇒ entire training set



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

convex (single global minima)

cost

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

maximum likelihood estimation

$\boxed{\text{Linear regression} \Rightarrow \hat{y} = \vec{w} \cdot \vec{x} + b}$

$\boxed{\text{Logistic regression} \Rightarrow \hat{y} = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}}$

Gradient Descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}\right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)\right]$$

} simultaneous updates

Overfitting

underfitting ⇒ high bias
- ↳ doesn't fit the training set well

overfitting ⇒ high variance
- ↳ fit training set extremely well

Addressing overfitting
1. get more training data
2. feature selection
3. Regularization ⇒ regularize $\vec{w}$ & b → optional

"Reduce the size of parameters"

n features → penalize all of them

λ = regularization parameter > 0

$$\boxed{J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y} - y)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2}$$

$\lambda = 0$, overfitting

$\lambda = 10^{10}$, underfitting

Linear Regression Gradient Descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left[\left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}\right] + \frac{\lambda}{m} w_j\right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)$$

}

$$w_j = \underbrace{w_j - \frac{\alpha\lambda}{m} w_j}_{w_j\left(1 - \frac{\alpha\lambda}{m}\right)} - \alpha \frac{1}{m} \sum_{1}^{m} (\hat{y} - y) x$$

Logistic Regression

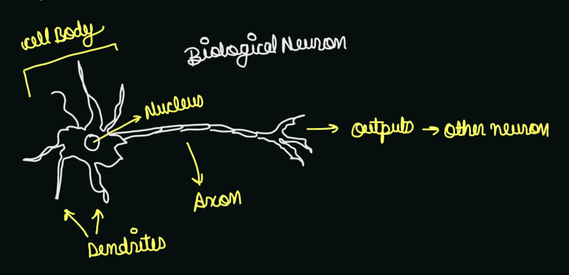
$$J = \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right] + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

$$\frac{\partial}{\partial w_j}(J) = \frac{1}{m} \sum_{1}^{m} \left[(\hat{y} - y)(x_j)\right] + \frac{\lambda}{m} w_j$$

Advanced Learning Algorithms-I

Wednesday, March 1, 2023 7:19 AM

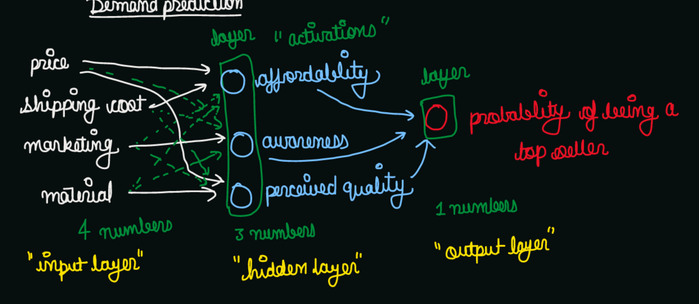
Neural Networks → speech → images → text (NLP) → ...
inference (prediction)
training
Decision trees



AI — The "one learning algorithm" hypothesis

- ANI (artificial narrow intelligence)
  - → smart speaker
  - → self driving car
  - → web search
  - → AI in farming & factories
- AGI (artificial general intelligence)
  - Do anything a human can do

Demand prediction



"multi-layer perceptron"
"neural-network architecture"

Vectorization

```
x= np.array([200, 17]) # vector
W= np.array([[ 1, -3, 5],
             [-2, 4, -6]]) # 2D matrix
b= np.array([-1, 1, 2]) # vector

def dense(a_in, W, b):
    units= W.shape[1]
    a_out= np.zeros(units)
    for j in range(units):
        w= W[:, j]
        z= np.dot(w, a_in)+ b[j]
        a_out[j]= g(z)
    return a_out
```

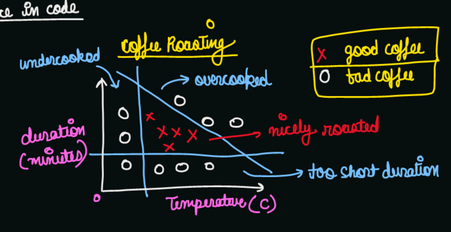
→ vector (1x3)

```
X= np.array([[200, 17]]) # 2D array → 1x2
W= np.array([[ 1, -3, 5],
             [-2, 4, -6]]) # 2D array → 2x3
B= np.array([[-1, 1, 2]]) # 2D array → 1x3

def dense(A_in, W, B):
    Z= np.matmul(A_in, W)+ B
    A_out= g(Z)
    return A_out
```

→ matrix multiplication
Z= A_in @ W
→ 2D array (1x3)

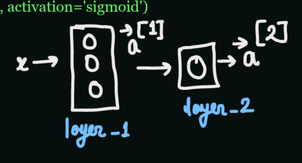
Inference in code




```
x= np.array([[200.0, 17.0]])
layer_1= Dense(units=3, activation='sigmoid')
a1= layer_1(x)

layer_2= Dense(units=1, activation='sigmoid')
a2= layer_2(a1)

if a2>=0.5:
    yhat=1
else:
    yhat= 0
```



Convolutional Layer

Each neuron only looks at part of previous layer's inputs

why?
- → speeds up the computation
- → Need less training data (Less prone to overfitting)

Matrix Multiplication

$z = \vec{a} \cdot \vec{w} \longleftrightarrow z = \vec{a}^T \vec{w}$ equivalent

matrix = # vectors stacked together

epochs = # steps in gradient descent

model.compile(loss = BinaryCrossentropy())

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

→ logistic loss
also called BinaryCrossEntropy } for classification
MeanSquaredError() } for regression

updates

$w = w - \alpha \, dJ/dw$
$b = b - \alpha \, dJ/db$
→ simultaneous updates

```
x= np.array([[1, 2, 3],
             [4, 5, 6]])
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} (2 \times 3)$$

```
import sympy
J,w= sympy.symbols('J, w')

J = w**3 # J = w^3
dJ_dw = sympy.diff(J, w) # dJ/dw = 3w^2
dJ_dw.subs([(w,4)])
```

↳ $3w^2 = 3(4)^2 = 12$

Backward Propagation
- → uses computation graphs
- → chain Rule
- → efficient

$$\boxed{\frac{\partial J}{\partial a} = \frac{\partial z}{\partial a} \times \frac{\partial J}{\partial z}}$$

Backprop is efficient
N → nodes
P → parameters
compute derivatives in $\boxed{N+P}$ not $N \times P$

Feature Vectors

temperature (Celcius)	duration (minutes)	Good coffee?
200.0	17.0	1
425.0	18.5	0
...	...	...

```
x= np.array( [[200.0, 17.0]] )
```

Logistic Regression

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

$loss = -y \log(a) - (1-y) \log(1-a)$

```
model.compile(loss= BinaryCrossEntropy()) # less accurate
```

$loss = -y \log\left(\frac{1}{1+e^{-z}}\right) - (1-y) \log\left(\frac{1}{1+e^{-z}}\right)$ logit: z

```
model.compile(loss= BinaryCrossEntropy(from_logits= True))
```

More accurate {numerical Round off errors}

Optimization Algorithms

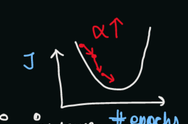
gradient descent
$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w,b)$
↳ learning rate

"Adam" algorithm
Adam: Adaptive Moment Estimation — not just one α

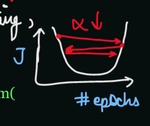
$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(w,b)$
⋮
$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(w,b)$
$b = b - \alpha_{11} \frac{\partial}{\partial b} J(w,b)$

11 α - learning rates

intuition of Adam



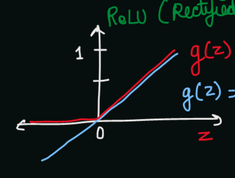
if w_j (or b) keeps moving in same direction, increase α_j
if w_j (or b) keeps oscillating, decrease α_j




```
model.compile(
    optimizers= tf.keras.optimizers.Adam(
        learning_rate=1e-3
    ),
    loss= loss # like previous one
)
```

Alternatives to sigmoid activation

ReLU (Rectified Linear Unit) → most common choice for hidden layers



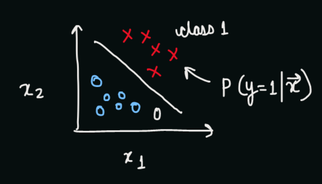
$g(z) = \max(0, z)$
$g(z) = z$ (linear activation function)
↓
"no activation function"

"linear function of a linear function is a linear function"

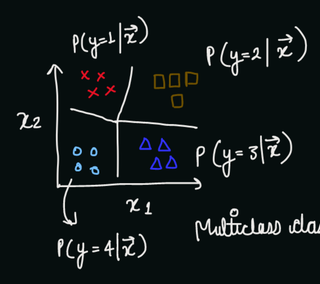
```
x= np.array( [[200.0, 17.0]] ) → [200 , 17] (1x2) Row Matrix

x= np.array( [[200.0], [17.0]] ) → [200; 17] (2x1) Column matrix

x= np.array( [200.0, 17.0] ) → "1D" "vector" linear array
```



Binary classification



Multiclass classification

Logistic Regression (2 possible output values)

$z = \vec{w} \cdot \vec{x} + b$

$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$

$a_2 = 1 - a_1 = P(y=0|\vec{x})$

Softmax Regression (4 possible output values)

$z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \quad a_1 = \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}} = 0.30 \Rightarrow P(y=1|\vec{x})$

$z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \quad a_2 = e^{z_2}/(e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}) = 0.20$

$z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \quad a_3 = e^{z_3}/(e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}) = 0.15$

$z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad a_4 = e^{z_4}/(e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}) = ?$

$1 - (0.30 + 0.20 + 0.15) \Rightarrow 1 - 0.65 = 0.35$

Building a Neural Network Architecture

```
X= np.array([[200.0, 17.0]
             [120.0, 5.0],
             [425.0, 20.0],
             [212.0, 18.0]])
y= np.array([1, 0, 0, 1]) #targets
```

(4x2)

X		y
200	17	1
120	5	0
425	20	0
212	18	1

```
layer_1= Dense(units=3, activation='sigmoid')
layer_2= Dense(units=1, activation='sigmoid')
model= Sequential( [layer_1, layer_2] )

model.compile(...)
model.fit(X, y)

model.predict( z_new )
```

```
model= Sequential( [
    Dense(units=3, activation='sigmoid'),
    Dense(units=1, activation='sigmoid') ] )
```

can do any!!

Softmax Regression
(N possible outputs) $y = 1, 2, 3, \ldots N$

$z_j = \vec{w}_j \cdot \vec{x} + b_j, \quad j = 1, 2, 3, \ldots, N$

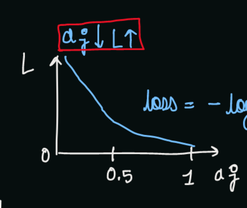
$$\boxed{a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j|\vec{x})}$$

$a_1 + a_2 + a_3 + \ldots + a_N = 1$

Loss (cross entropy loss)

$$loss(a_1, a_2, a_3 \ldots a_N, y) = \begin{cases} -\log a_1, & \text{if } y=1 \\ -\log a_2, & \text{if } y=2 \\ \vdots \\ -\log a_N, & \text{if } y=N \end{cases}$$

$\boxed{a_j \downarrow L \uparrow}$



$loss = -\log a_j \quad \text{if } y = j$

#Works Okay but...
Not Recommended

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model= Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax'),
])
```

↳ linear

specify the model

```
from tensorflow.keras.losses import SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy())
```

loss & cost

```
model.fit(X, y, epochs=100)
```

train

use this!!

```
model.compile(loss= SparseCategoricalCrossentropy(from_logits= True))

logits = model(X)
f_x = tf.nn.softmax(logits) # more numerically accurate
```

↳ sigmoid (logistic regression)

Forward prop in a single layer

$w_1^{[2]} \Rightarrow$ w2_1

$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$

conventions
matrix → Uppercase (W)
vectors/scalars → lowercase (w)

$$W = \begin{bmatrix} 1 & -3 & 5 \\ 2 & 4 & -6 \end{bmatrix} (2 \times 3)$$

$b_1^{[1]} = -1, \quad b_2^{[1]} = 1, \quad b_3^{[1]} = 2$

$b = [-1, 1, 2]$

```
def dense(a_in, W, b):
    units= W.shape[1]
    a_out= np.zeros(units)
    for j in range(units):
        w= W[:, j] # jth column of the matrix
        z= np.dot(w, a_in)+ b[j]
        a_out[j]= g(z)
    return a_out
```

Advanced Learning Algorithms-II

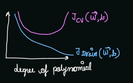
Friday, March 10, 2023 4:20 PM

large error in prediction
- → get more training examples
- → try smaller set of features
- → try adding additional features
- → try adding polynomial features
- → try decreasing λ
- → try increasing λ

cross validation set (dev set, validation set)

high Bias → underfit
- J_{train} is high
- J_{cv} is also high

high Variance → overfit
- J_{train} is low
- J_{cv} is high

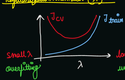


(high Bias (underfit))
- J_{train} will be high.
- $(J_{train} \approx J_{cv})$

High Variance (overfit)
- J_{train} is low
- $J_{cv} >> J_{train}$

High Bias & High Variance
- J_{train} is high
- $J_{cv} >> J_{train}$

Regularization Parameter (λ)



Speech Recognition

human level performance → 10.6% ↑ 0.2%

training error J_{train} → 10.8% ↑ 4% high variance

Cross validation error J_{cv} → 14.8%

High Bias → If a learning algorithm has high bias, getting more data will not help much by itself.

High Variance → If an algorithm suffers from high variance then getting more data is likely to help.

Debugging a learning algorithm

Regularized Linear Regression

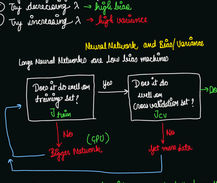
$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

unacceptably large error in prediction

1. get more training examples → high variance
2. try smaller set of features → high variance
3. try getting additional features → high bias
4. try adding polynomial features (x_1^2, x_2^2, x_1x_2 etc.) → high bias
5. try decreasing λ → high bias
6. try increasing λ → high variance

Neural Network and Bias/Variance

Large neural Networks are low bias machines



A large neural network will usually do better/as well than a smaller one so long as regularization is chosen appropriately.

Machine Learning Development Process

Iterative loop of ML Development

Adding Data

→ Add more data of everything / e.g. "Honeypot" project

→ Add more data of the type where error analysis indicated it might help

Data augmentation

Augmentation: modifying an existing training example to create new training example

Transfer Learning

Option
1. Supervised Pretraining
2. Fine Tuning

(less data)
1. only train output layer parameters
2. train all parameters (more data)

full cycle of Machine learning project

Deployment

Skewed Datasets

		1	0
Predicted	1	TP	FP
class	0	FN	TN

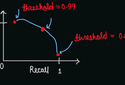
Actual class

Precision = $\frac{\#\text{ True positives (TP)}}{\#\text{ predicted positives (TP+FP)}}$

Recall = $\frac{\#\text{ True positives (TP)}}{\#\text{ actual positives (TP+FN)}}$

"high precision & high recall"

Trade off between Precision & recall



F_1 score

$$F_1 \text{ score} = \frac{1}{\frac{1}{2}(1/P + 1/R)} = \frac{2PR}{P+R}$$

Harmonic mean

Decision Trees

Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?
→ Maximize purity (or minimize impurity)

Decision 2: When do you stop splitting?
- → when a node is 100% one class
- → when splitting a node will result in a tree exceeding a maximum depth
- → when improvements in purity score are below a threshold
- → when # examples in a node is below a threshold

Measuring Purity

{C, C, C, D, D, D}

Entropy → measurement of impurity of data

P_1 = fraction of examples that are cat (C)

$P_2 = 3/6$

$H(P_1) = 1$

$\Rightarrow P_0 = 1 - P_1$

Entropy

$$H(h) = -P_1\log_2(P_1) - P_0\log_2(P_0)$$
$$\Rightarrow -P_1\log_2(P_1) - (1-P_1)\log_2(1-P_1)$$

{ note: "0 log(0)" = 0 }

choosing a split

$P_1^{root} = 5/10 = 0.5$ {C, C, C, D, D, D, C, D, C, D}

Ear shape: $P_1^{left} = 4/6$, $w^{left} = 6/10$; $P_1^{right} = 1/4$, $w^{right} = 4/10$

$P_1 = 5/10 = 0.5$, $H(0.5) = 1$

{C, C, C, C, D} $P_1 = 4/6 = 0.8$, $H(0.8) = 0.72$

{D, D, D, D} $P_1 = 1/4 = 0.2$, $H(0.2) = 0.72$

$$H(0.5) - \left(\frac{6}{10}H(0.8) + \frac{4}{10}H(0.2)\right)$$

$\Rightarrow 0.28$ {Information gain}

$$\text{Information gain} = H(P_1^{root}) - \left(w^{left}H(P_1^{left}) + w^{right}H(P_1^{right})\right)$$

Decision Tree Learning

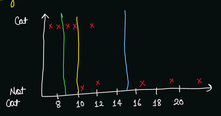
1. Start with all examples at the root node
2. Calculate information gain for all possible features, and pick the one with the highest information gain
3. Split dataset according to selected feature & create left and right branches of the tree.
4. Keep repeating splitting process until stopping criteria is met:
   - → when a node is 100% one class
   - → when splitting a node will result in the tree exceeding a maximum depth
   - → Information gain from additional splits is less than threshold
   - → when # examples in a node is below a threshold.

Recursive splitting

One hot encoding

If a categorical feature can take on k values, create k features (0 or 1 valued).

Splitting on a continuous variable



$$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$$

$$H(0.5) - \left(\frac{4}{10}H\left(\frac{4}{4}\right) + \frac{6}{10}H\left(\frac{1}{6}\right)\right) = 0.61 \text{ (max information gain)}$$

$$H(0.5) - \left(\frac{7}{10}H\left(\frac{5}{7}\right) + \frac{3}{10}H\left(\frac{0}{3}\right)\right) = 0.40$$

Regression with Decision Trees

"predicting weight"

choosing a split?

classification → minimize average weighted entropy

Regression → minimize average weighted variance

Tree Ensembles

using multiple decision trees → & get them to vote

1. Trees are highly sensitive to small changes of the data

Sampling with Replacement → to generate new training set for tree ensembles.

Tokens:

Sampling with replacement:

Random Forest Algorithm

(generating a tree sample)

"Bagged Decision Tree"

Given a training set of size m

for b = 1 to B:
- use sampling with replacement to create new training set of size m
- Train a decision tree on the new dataset

Randomizing the feature choice (Random Forest Algorithm)

At each node, when choosing a feature to use to split, if n features are available pick a random subset of $k < n$ features & allow the algorithm to only choose from that subset of features.

large n, $k = \sqrt{n}$

XGBoost

Boosted trees intuition

Given training set of size m

for b=1 to B:
- use sampling with replacement to create a new training set of size m
- But instead of picking from all the examples with equal (1/m) probability, make it more likely to pick misclassified examples from the previously trained trees.
- Train a decision tree on the new dataset

XGBoost (eXtreme Gradient Boosting)
- → open source implementation of Boosted Trees
- → fast, efficient implementation
- → good choice of default splitting criteria & criteria for when to stop splitting
- → built in regularization to prevent overfitting
- → highly competitive algorithm for ML competitions

```
from xgboost import XGBClassifier/ XGBRegressor

model= XGBClassifier()/ XGBRegressor()

model.fit(X_train, y_train)
y_pred= model.predict(X_test)
```

Decision Trees v/s Neural Networks

Decision Trees & Tree Ensembles
- → work well on tabular (structured) data
- → Not recommended for unstructured data (images, audio, text)
- → fast
- → small decision trees may be human interpretable

Neural Networks
- → Works well on all types of data, including tabular (structured) and unstructured data
- → may be slower than a decision tree
- → works with transfer learning
- → when building a system of multiple models working together, it might be easier to string together multiple neural networks

Unsupervised Learning , Recommenders, Reinforcement learning-I

Monday, April 24, 2023 10:11 AM

Unsupervised learning

- clustering
- anomaly detection

Recommender System

Reinforcement learning

Clustering

Training set: $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$

K-means Algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K$

Repeat {

→ Assign points to cluster centroids

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

→ move the cluster centroids

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}

K-means optimization objective

$c^{(1)}, c^{(2)} \dots c^{(m)}$ = cluster index of cluster $(1,2,\dots,K)$ to which example $(x^{(i)})$ is currently assigned

μ_k = cluster centroid k

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $(x^{(i)})$ has been assigned

Cost function

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

(distortion function)

Random initialization

choose $K < m$

Randomly pick K training examples

Set $\mu_1, \mu_2, \dots, \mu_K$ equal to these K examples

for i = 1 to 50 { # 50 random initializations

Randomly initialize K means

Run K-means. Get $c^{(1)}, c^{(2)} \dots c^{(m)}, \mu_1, \dots, \mu_K$

Compute cost function (distortion)

$J(c^{(1)}, c^{(2)} \dots c^{(m)}, \mu_1, \dots, \mu_K)$

} # Pick set of clusters that gives lowest cost J

Choosing the # clusters

Choosing the total cluster of K

Elbow method

see plot K vs J to optimize cost

Note: choose K just by looking this ?

often you want to get clusters for some later (downstream) purpose

Evaluate K-means based on how well it performs on that later purpose

Image compression

Anomaly Detection

Anomaly: finding patterns

x_1 = heat generated

x_2 = vibration intensity

⋮

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

New engine : x_{test}

(Anomaly)

Density estimation

Training dataset:

model(x_1) = probability of x being one in dataset

→ high probability

→ low probability

$p(x) < \varepsilon$

→ Is x our anomalous ?

x_{test} : anomalous !

Housekeeping:

Real-number evaluation: Develop numerical review

$y^{(i)}$ = 1 if anomaly and 0 if normal

→ Hard: $p(x)$ from data.

→ have few big $y=1$

→ use large normal

→ 10 features

→ 20 features

→ 50 features

→ Gaussian

Anomaly detection compared to data centric

$x^{(i)}$ = features of machine i

x_1 = memory use

x_2 = # disk accesses/sec

x_3 = CPU load

x_4 = CPU load / network traffic

Gaussian (normal) distribution

Say x is a number.

Probability of x is determined by a Gaussian with mean μ, std deviation σ

$x \sim N(\mu, \sigma^2)$

→ standard deviation σ

→ variance

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Parameter fitting/learning estimate for μ, σ^2

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

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

Anomaly detection Algorithm

Training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

Each example $x^{(i)}$ have n features

$$\bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\bar{x}) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \dots \times p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

(Anomaly)

$p(x_1; \text{high temperature}) = 1/10$

$p(x_2; \text{high vibration}) = 1/20$

$p(\bar{x}; x_1, x_2) = p(x_1) \times p(x_2)$

$= 1/10 \times 1/20 = 1/200$

Anomaly detection Algorithm

① choose n features x_j that you think might be indicative of anomaly.

② fit parameters $\mu_1, \mu_2, \dots, \mu_n, \sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

③ Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi}\,\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

④ Anomaly if $p(x) < \varepsilon$

Real-number evaluation

When developing a learning algorithm (choosing features, etc)

making decisions is much easier if you have a way of evaluating your learning algorithm

assume:

non-anomaly → y = 0

anomaly → y = 1

Training set : $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ (assume normal examples/not anomalous) y = 0

Cross validation set / Test set: } include a few anomalous examples (y = 1) ; mostly examples (y = 0)

Anomaly detection vs Supervised learning

Very small # positive examples (0-20)

→ 2-50 is common

large # negative examples (y=0)

Many different types of anomalous ideas for any algorithm to learn from positive examples are like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

"unknown unknowns"

Many positive examples (y=1)

large # positive examples for the algorithm to get a sense of what positive examples are like; future positive examples are likely to be similar to the ones in the training set. "known known"

Many different types of anomalous ideas; algorithm's positive examples may look pretty similar.

"future examples"

Recommender System

Function Movie Ratings

users rate the movies 0-5 stars

Movie	u_1	u_2	u_3	u_4	u_5
1	5	5	0	0	0
2	5	5	0	0	0
3	0	0	5	5	5
4	0	0	5	5	5
5	5	0	0	0	0

$n_u = 5$ → # users

$n_m = 5$ → # movies

$r(i,j)$ = 1 if user j rated movie i

$y^{(i,j)}$ = rating given by user j to movie i

$\bar{x}^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$ movie 1

$\bar{x}^{(2)} = \begin{bmatrix} 1.0 \\ 0.01 \end{bmatrix}$ movie 2

$\bar{x}^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$ movie 3

$\bar{x}^{(4)} = \begin{bmatrix} 0.1 \\ 1.0 \end{bmatrix}$ movie 4

$\bar{x}^{(5)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ movie 5

n = # features

$x_0 = 1$

x_1 = romance

x_2 = action

Learn parameters $\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}$ for each user

$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ user 1

$\theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ user 2

$\theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ user 3

$\theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ user 4

$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ user 5

For user 1: Predict rating of movie 3 as: $(\theta^{(1)})^T x^{(3)} = 4.95$

$\theta^{(j)}$ = parameters for user j

$x^{(i)}$ = features for movie i

For user j: predict rating for movie i as: $(\theta^{(j)})^T x^{(i)}$

$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}$ → n_u users

$x^{(1)}, x^{(2)} \dots x^{(n_m)}$ → n_m movies

n = # features

Learn $\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}$ given $x^{(1)}, x^{(2)} \dots x^{(n_m)}$

$$J(\theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Gradient descent

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left[\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda\theta_k^{(j)}\right] \quad (\text{for } k \neq 0)$$

Collaborative filtering

Cost function to learn $x^{(1)}, x^{(2)} \dots x^{(n_m)}$ collaborative

$$J(x^{(1)}, \dots, x^{(n_m)}) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$$

Cost function to learn $\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}$

$$J(\theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Minimizing x and θ:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

Gradient descent

parameters : w, b, x

Repeat {

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

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

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

}

Binary labels

Example application

① Did user j purchase an item after being shown? (0,1)

② Did user j find item interesting? (0,1)

③ Did user j spend at least 30 sec with an item? (0,1)

④ Did user j click on an item (0,1)

Matrix of ratings

① All user assign the same rating to an item

→ engaged after being shown item

→ item not get shown

Binary label: favorite, purchased, liked

Collaborative filtering with binary labels

Classification:

Predict $y^{(i,j)}$ as $\sigma((\theta^{(j)})^T x^{(i)})$

For Recommender Systems:

Predict the probability of $y^{(i,j)} = 1$

where:

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

$$y^{(i,j)} = \sigma\left((\theta^{(j)})^T x^{(i)}\right)$$

Loss → for binary example

$$L(f_{(w,b)}(x), y) = -y^{(i,j)}\log\left(f_{(w,b)}(x)\right) - (1-y^{(i,j)})\log\left(1 - f_{(w,b)}(x)\right)$$

→ binary

→ cost of an example

$$J(w,b) = \sum_{(i,j):r(i,j)=1} L\left(f_{(w,b)}(x^{(i)}), y^{(i,j)}\right)$$

Mean Normalization

users who haven't rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$\begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & ? \end{bmatrix} - \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \end{bmatrix} = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 \\ 2.5 & ? & ? & -2.5 \\ ? & -1 & 1 & ? \\ -2.25 & -2.25 & 2.75 & 1.75 \\ -2.25 & -2.25 & 2.75 & -2.25 \end{bmatrix}$$

(average)

for user j, on movie i, predict:

$$(\theta^{(j)})^T x^{(i)} + \mu_i$$

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad \mu_3 = 1.25$$

learn $\theta^{(j)}, b^{(j)}$

$$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}, b^{(1)}, b^{(2)} \dots b^{(n_u)} = 0$$

$$\theta^{(j)} \in \mathbb{R}^n, \quad b^{(j)} \in \mathbb{R}$$

Predict for user j, on movie i: $(\theta^{(j)})^T x^{(i)} + b^{(j)}$

learn $\theta^{(j)}, b^{(j)}$

$$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}, b^{(1)}, b^{(2)} \dots b^{(n_u)} = 0$$

learning

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad b^{(5)} = 0$$

learn $\theta^{(j)}, b^{(j)}$

$$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}, b^{(1)}, b^{(2)} \dots b^{(n_u)} = 0$$

learning

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad b^{(5)} = 0$$

Predict for user j, on movie i: $(\theta^{(j)})^T x^{(i)} + b^{(j)} + \mu_i$

learn $\theta^{(j)}, b^{(j)}$

$$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}, b^{(1)}, b^{(2)} \dots b^{(n_u)} = 0$$

learning

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad b^{(5)} = 0$$

learn $\theta^{(j)}, b^{(j)}$

$$\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}, b^{(1)}, b^{(2)} \dots b^{(n_u)} = 0$$

learning

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad b^{(5)} = 0$$

Predict for user j, on movie i: $(\theta^{(j)})^T x^{(i)} + b^{(j)} + \mu_i

Unsupervised Learning , Recommenders, Reinforcement learning-II

Saturday, April 29, 2023 1:06 PM

Reinforcement Learning

position of helicopter ⟶ how to move control sticks

State s ⟶ action a

reward function

positive reward : helicopter flying well +1

negative reward : helicopter flying poorly −1000

Applications

1. controlling robots
2. Factory optimization
3. Financial (stock) trading
4. Playing games (including video games)

Mars Rover Example

terminal state ↓ terminal state

100	0	0		0	40

state : 1 2 3 4 5 6

← left → right

state	4	3	2	1
reward	0	0	0	100

	4	5	6
	0	0	40

$(s, a, R(s), s')$

↓

$(4, \leftarrow, 0, 3)$

Return = Σ all rewards (with discount)

$$\boxed{\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots\ldots + (\text{until Terminal state})}$$

γ → discount factor

Let $\gamma = 0.9$

Return $= 0 + 0.9(0) + (0.9)^2(0) + (0.9)^3(100) = 72.9$

(state 4 to 1)

Example of Return

100	50	25	12.5	6.25	40
100	0	0	0	0	40
1	2	3	4	5	6

← return ← reward $\gamma = 0.5$

Return depends on the action you take

100	2.5	5	10	20	40
100	0	0	0	0	40
1	2	3	4	5	6

← return ← reward $\gamma = 0.5$

100	50	25	12.5	20	40
100	0	0	0	0	40
1	2	3	4	5	6

← return ← reward $\gamma = 0.5$

100	50	25	12.5	6.25	40
100	0	0	0	0	40
1	2	3	4	5	6

← return ← reward $\gamma = 0.5$

Policy (π)

state s $\xrightarrow[\pi]{\text{policy}}$ action a

→ mapping from states to action, telling what actions to take in a given state s.

$\pi(2) = \leftarrow$
$\pi(3) = \leftarrow$
$\pi(4) = \leftarrow$
$\pi(5) = \rightarrow$
$\pi(s) = a$

The goal of RL is to find a policy π, $(a = \pi(s))$ that tells you what action to take in every step to maximize the return.

	Mars Rover	Helicopter	Chess
states	6 states	position of hc.	pieces on board
action	←, →	move control	set of legal moves
rewards	100, 0, 40	+1, −1000	+1, 0, −1
discount factor γ	0.5	0.99	0.995
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots$	$R_1 + R_2(1) + R_3\gamma^2 + \ldots$	$R_1 + R_2\gamma + R_3\gamma^2 + \ldots$
policy π	100 ← ← ← → 40	$\pi(s) = a$	$\pi(s) = a$

Markov Decision Process (MDP)

State Action Value function (Q function)

$Q(s,a)$ = Return if you (Best Return)

→ start in state s

→ take action a (once)

→ then behave optimally after that

100	50	25	12.5	20	40
100	0	0	0	0	40
1	2	3	4	5	6

← return ← action ← reward

$Q(2, \rightarrow) = (0) + (0.5)(0) + (0.5^2)(0) + (0.5^3)(100) = 12.5$

State : 2 3 2 1

$Q(2,\leftarrow)$ $Q(2,\rightarrow)$

(terminal state)

100	50	12.5	25	6.25	12.5	10	6.25	20	4.9	40
100	0		0		0		0		40	
1	2		3		4		5		6	

(terminal state)

$\max_a Q(s,a)$ → best possible return from state s

Best possible action in state s which give the maximum Q-value

$\max_a Q(s,a)$

$\pi(s) = a$

Q^* / Optimal Q-function

Bellman Equation

$Q(s,a)$ = Return if you

- start in state s
- take action a (once)
- then behave optimally after that

s : current state

a : current action

s' : state you get to after taking action a

a' : action you take in state s'

$R(s)$ = reward of current state

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')$$

(terminal state)

100	100	50	12.5	25	6.25	12.5	10	6.25	20	4.9	40
100		0		0		0		0		40	
1		2		3		4		5		6	

(terminal state)

↳ $Q(s,a) = R(s)$

$Q(s,a) = R(s)$

$Q(2, \rightarrow) \Rightarrow R(2) + 0.5 \max_{a'}(3, a')$

$\Rightarrow 0 + 0.5(25) \Rightarrow 12.5$

$s = 2$, $a = \rightarrow$, $s' = 3$

$Q(4, \leftarrow) = R(4) + 0.5 \max_{a'}(3, a')$

$= 0 + 0.5(25) = 12.5$

$s = 4$, $a = \leftarrow$, $s' = 3$

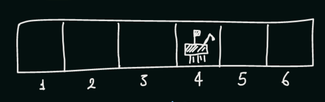
$$Q(s,a) = \underbrace{R(s)}_{\text{Reward you get right away (Immediate Reward)}} + \underbrace{\gamma \max_{a'} Q(s', a')}_{\text{Return from behaving optimally starting from state } s'}$$

$$Q(s,a) = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots$$

$$= R_1 + \gamma\underbrace{[R_2 + \gamma R_3 + \gamma^2 R_4 + \ldots]}_{\text{optimal return if you start from state } s'}$$

R_1 → immediate reward of state s.

Stochastic Environment



1 2 3 4 5 6

← 0.9 0.1

→ 0.1 0.9

Expected Return = Average$(R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots)$

$= E[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \ldots]$

Discrete V/s Continuous State

$s = \{1, 2, 3, 4, 5, 6\}$

Discrete State : 6-state Lunar lander

1	2	3	4	5	6

Continuous State :

Truck/car

$$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

Helicopter

$$s = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \omega \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}$$

position (3D)

ϕ → roll, θ → pitch, ω → yaw

continuous space MDP

Lunar Lander

actions :

→ do nothing

→ left thruster

→ main thruster

→ right thruster

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

l, r : 0 or 1

Learn a policy π for given state s, pick action, $a = \pi(s)$ so as to maximize the return $\gamma = 0.985$

Reward function

→ Getting to landing pad : 100-140

→ Additional reward for moving toward/away from pad

→ Crash : −100

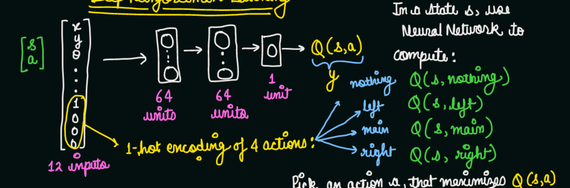
→ Soft Landing : +100

→ Leg grounded : +10

→ Fire main engine : −0.3

→ Fire side thruster : −0.03

Deep Reinforcement Learning



In a state s, use Neural Network to compute :

$Q(s, \text{nothing})$

$Q(s, \text{left})$

$Q(s, \text{main})$

$Q(s, \text{right})$

Pick an action a, that maximizes $Q(s,a)$

Bellman Equation

$$\underbrace{Q(s,a)}_{X} = \underbrace{R(s) + \gamma \max_{a'} Q(s', a')}_{y}$$

Random guess

$$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$$

$$\boxed{f_{w,b}(X) \approx y}$$

$(s, a, R(s), s')$

$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)})$

$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)})$

$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)})$

$(s^{(m)}, a^{(m)}, R(s^{(m)}), s'^{(m)})$

x	y
$x^{(1)} = (s^{(1)}, a^{(1)})$ (8, 4)	$y^{(1)}$
$x^{(2)} = (s^{(2)}, a^{(2)})$	$y^{(2)}$
⋮	⋮
$x^{(10,000)}$	$y^{(10,000)}$

Learning Algorithm (Deep Q-Network)

Initialize neural network randomly as guess $Q(s,a)$

Repeat {

Take actions in the Lunar Lander. Get $(s, a, R(s), s')$.

Store 10,000 most recent $(s, a, R(s), s')$ tuples. ← Replay Buffer

Train neural Network :

create training set of 10,000 examples using (X, y) ← use minibatch of 1,000 examples (noisy, but speed up performance)

$x = (s, a)$ & $y = R(s) + \gamma \max_{a'} Q(s', a')$

Train Q_{new} such that $Q_{new}(s,a) \approx y$

$f_{w,b}(x) \approx y$

Set $Q = Q_{new}$ ← Soft update

}

Refinements

$$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

8 inputs → 64 units → 64 units → 4 units

$Q(s, \text{nothing})$

$Q(s, \text{left})$

$Q(s, \text{main})$

$Q(s, \text{right})$

$Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')$

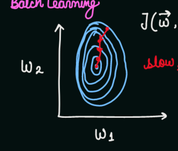
In a state s, input s to neural network, Pick the action a, that maximizes $Q(s,a)$

Mini Batch

$m = 100{,}000{,}000$ (very large, slow GD ☹)

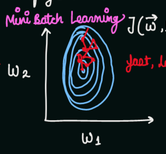
$m' = 5{,}000$ {subset} → new minibatch on each iteration of gradient Descent

Batch Learning



slow, more accurate

Mini Batch Learning



fast, less accurate

E-greedy policy

How to choose actions while still learning?

In some state s

Option 1

Pick the action that maximizes $Q(s,a)$

Option 2 ($\varepsilon = 0.05$)

With probability 0.95, pick the action that maximizes $Q(s,a)$ "Exploitation" greedy

With probability 0.05, pick an action randomly → ε "Exploration" with ε

Start ε high # More exploration

$1.0 \rightsquigarrow 0.01$

gradually decrease # More greedy

Soft update

Set $Q = Q_{new}$

w, b ← w_{new}, b_{new} → converge more reliably

$$\boxed{\begin{aligned} w &= 0.01 w_{new} + 0.99 w \\ b &= 0.01 b_{new} + 0.99 b \end{aligned}}$$

Limitations of Reinforcement Learning

- Much easier to get to work in a simulation than a real robot!
- Far fewer applications than Supervised & unsupervised learning.
- But... exciting research direction with potential for future applications.