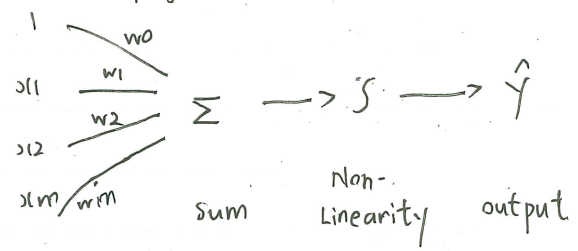
Stanford ML Note f.t. Andrew Ng

Deep Learning.

AI - machine Learning: learn without being explicitly being programmed

- Deep Learning: Extract patterns from data using neural networks

Forward Propagation.

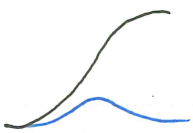


Activation Functions.

$$\hat{y} = g(w_0 + x^T w) = g(w_0 + \sum_{i=1}^{m} x_i w_i)$$

Common Activation Functions.

Sigmoid

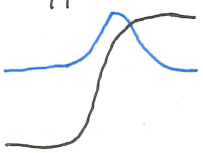


$$g(z) = \frac{1}{1+e^{-z}}$$

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

tf.nn.sigmoid(z)

Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

tf.nn.tanh(z)

Rectified Linear Unit

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

$$g'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

tf.nn.relu(z)

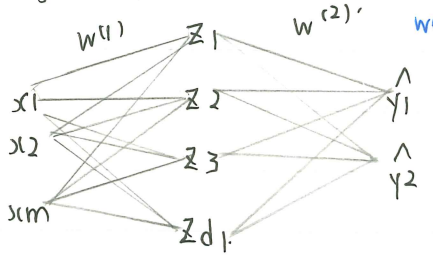
No.

Date

Three steps to compute the output of perceptron.

1. Dot product
2. Add a biases
3. Take non-linearity

Single Layer Neural Network.



Inputs Hidden. Output.

$$z_i = w_{0i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d1} z_j w_{ji}^{(2)}\right)$$

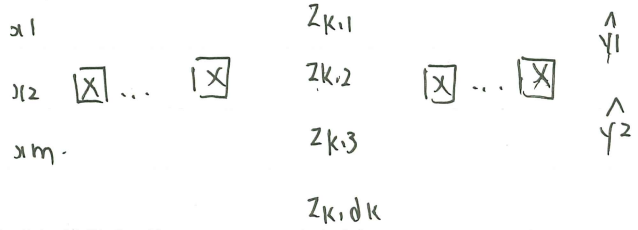
Multi Output Peceptron.

```
from tf.keras.layers import
inputs = Inputs(m)
hidden = Dense(d1)(inputs)
outputs = Dense(2)(hidden)
model = Model(inputs, outputs)
```

No.

Date

Deep Neural Network.



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

The model must be trained!

Quantifying loss

The loss of our network measures the cost incurred from incorrect predictions

$L(\underbrace{f(x^{(i)}; w)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$

Empirical Loss: measures the total loss over our entire data set.

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)}; w), y^{(i)})$$

Binary Cross Entropy Loss: used when output is probability between [0, 1]

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log(f(x^{(i)}; w)) + (1-y^{(i)}) \log(1 - f(x^{(i)}; w))$$

Mean Squared Error Loss:

Used with regression models that output continuous real numbers.

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}; w))^2$$

Define the loss is an art.

No.

Date

Loss optimization.:

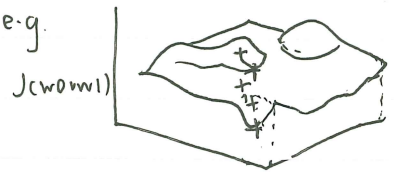
To achieve the lowest loss.

$$w^* = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} L(f(x^{(i)}; w), y^{(i)})$$

$$= \arg\min_w J(w).$$

Loss is a function of the network weights!

e.g.



1. Randomly pick an (w_0, w_1)
2. Compute gradient $\frac{\partial J(w)}{\partial w}$
3. Take small step in opposite direction
4. Repeat until convergence

No guarantee it's a global mininum.
But we only need to optimize local mininum.
otherwise too expensive to compute.

Gradient Descent.

Algorithm

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient $\frac{\partial J(w)}{\partial w}$
4. Update weights $w \leftarrow w - \eta \frac{\partial J(w)}{\partial w}$.
5. Return weights

Backpropagation $x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \longrightarrow J(w)$

$$\frac{\partial J(w)}{\partial w_2} = \frac{\partial J(w)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} \qquad \frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

Campus

No.

Date

Setting learning rate.

Too small: — converge slowly, get stuck in false local minima

Too large: — overshoot, become unstable and diverge.

Adaptive Learning Rates: not fixed.

e.g. Momentum / Adagrad / Adadelta / Adam / RMS Prop

Stochastic Gradient Descent.

$\frac{\partial J_i(W)}{\partial W}$ is easy to compute but very noisy and stochastic

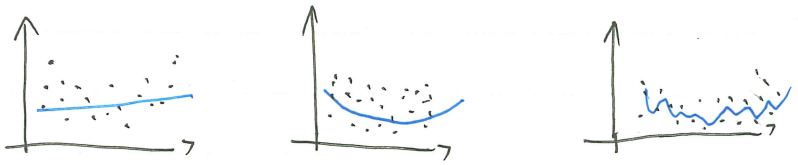
Batch data into mini batches $\frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(W)}{\partial W}$

- faster and more accurate.

parallelized computation. send batches across GPU

compute simultaneously

aggregate them back.



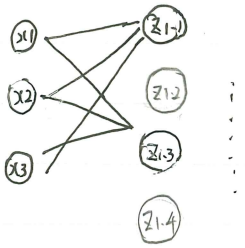
Regularization.

Technique that constrains our optimization problem to discourage complex models.

- Improve generalization of model on unseen data.

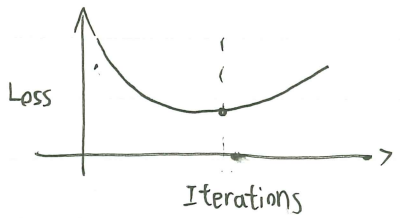
Regularization I : Dropout.

- During training, radomly set some activations to 0.
  - Typically 'drop' 50% of activation in layer
  - Force network to not rely on any I node.



II: Early Stopping

- Stop training before we have chance to overfit.



No.

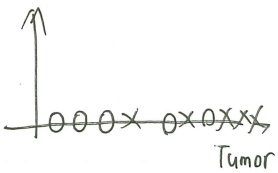
Date

Stanford Machine Learning

Supervised Learning – input→output mappings

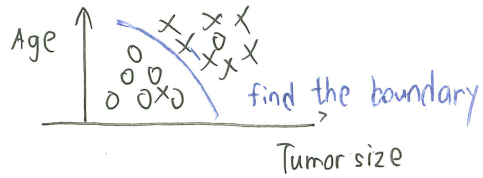
1. Regression (回归) – predict infinite possible numbers
2. Classification algorithom – predict categories

e.g



Breast cancer

malignant or benign?



Learn from the right numbers.

Unsupervised learning - Data only comes with inputs x but not output labels y.

Find some structure/ pattern in unlabeled data

1. Clustering e.g Google News / DNA micro array / Grouping Customers.

— Group similar data points together

Anomaly detection

Dimensionality reduction

No.

Date

Linear Regression.

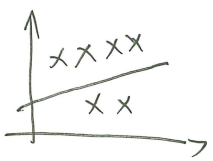
Terminology

x = "Input" variable : feature.

y : "output" variable : "target" variable.

m : number of training examples.

(x, y) = single training example



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

$f(x) = wx + b$

Univariate linear regression

one

training set

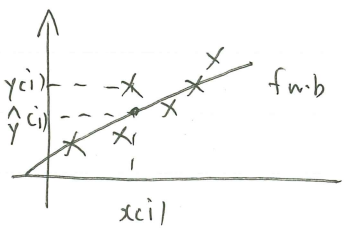
↓

learning algorithm.

↓

$x \longrightarrow [f] \longrightarrow \hat{y}$

feature — model — prediction (estimated target)



Cost Function

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

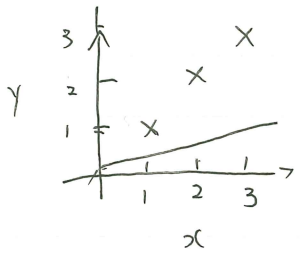
coefficients

Squared error cost function

For a simplified version.....

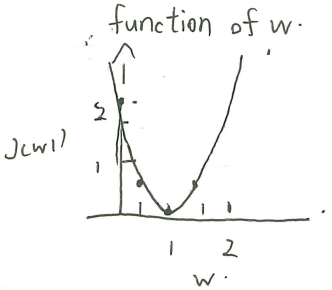
$f_w(x)$

function of x



$J(w)$

function of w.



$$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right] = \frac{7}{12}$$

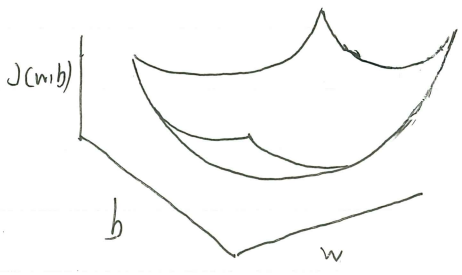
$$J(0) = \frac{14}{6}$$

Goal of linear regression.

- choose w to minimize $J(w)$

General case.

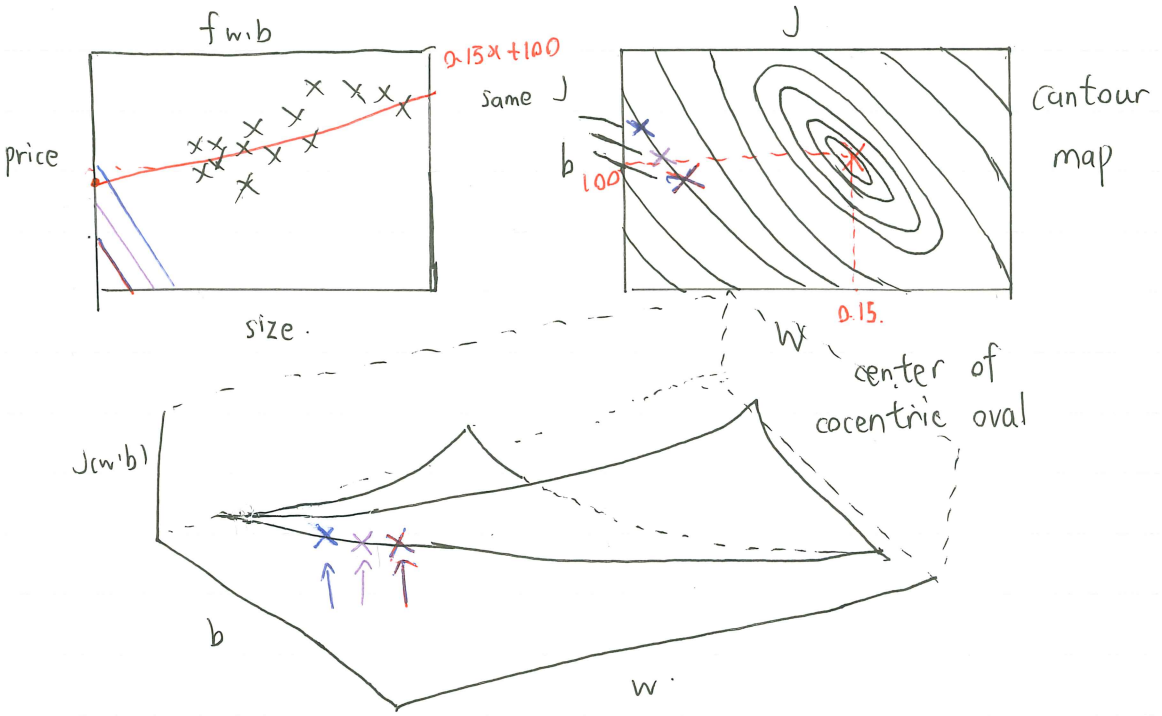
minimize $J(w,b)$
w b



Use contour map.

Square error cost always leads to a bowl-shaped graph.

convex function


f w,b
0.13x+100
price
size
same J
J
b
100
0.15
w
contour
map
center of
cocentric oval
J(w,b)
b
w

No.

Date

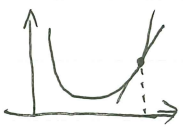
Gradient descent (one feature)

Algorithm: repeat until convergence

$w = w - a\frac{\partial}{\partial w}J(w,b)$ a is Learning rate.

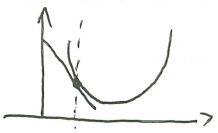
$b = b - a\frac{\partial}{\partial b}J(w,b)$

Simultaneously update w and b.



$w = w - a\frac{d}{dw}J(w)$

$w = w - a \cdot (\text{positive number})$ $\therefore$ w moves left



$w = w - a\frac{d}{dw}J(w)$

$w = w - a(\text{negative number})$ $\therefore$ w moves right

Learning rate

If a is too small.... Gradient descent too slow.

If a is to large Gradient descent may diverge.

- Can reach local minimum with fixed learning rate a.

  Near a l.m., - Derivative becomes smaller

  - Update step becomes smaller.

No.

Date

Gradient descent algorithm.

repeat until convergence:

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

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

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

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

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

"Batch" Each step of gradient descent uses all the training examples

Multiple features (variables)

x_j = j^{th} feature

n = number of features

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

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b = \vec{w} \cdot \vec{x} + b$$

multiple linear regression.

parameters: $\vec{w} = [w_1 \; w_2 \; \dots \; w_n]$, b

No.

Date

Vectorization

* linear algebra count from 1 but code count from 0.

$$f_{\vec{w},b}(\vec{x}) = \sum_{i=1}^{n} w_j x_j + b$$

Without Vectorization

```
f = 0
for j in range(n):
    f = f + x[j] * w[j]
f = f + b
```

Vectorization.

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

☺

1. shorter
2. Faster

Numpy dot function can use Parellel hardware. — more efficient.

to $f + w[0] * x[0]$

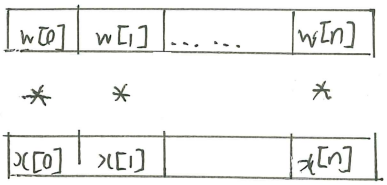
t1 $f + w[1] * x[1]$

. . . . . .

tn $f + w[n] * x[1]$

— Step by step. one step at a time.

to



in parallel

t1

$w[0] * x[0] + \ldots\ldots + w[n] * x[n]$

Parellel hardware is efficient when scale to large data set.

No.

Date

Gradient descent.

$\vec{w} = (w_1\ w_2\ \dots\ w_n)$ $\quad \vec{d} = (d_1\ d_2\ \dots\ d_n)$

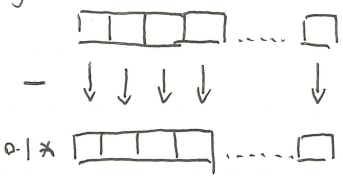
$w = np.array([w_1, w_2, \dots w_n])$

$d = np.array([d_1, d_2, \dots d_n])$.

compute $w_j = w_j - 0.1\, d_j$

a = learning rate

Vectorization: $\vec{w} = \vec{w} - 0.1\vec{d}$



code: $w = w - 0.1 * d$

When we have n features $(n \geq 2)$

repeat {

$j=1$ $\quad w_1 = w_1 - a\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\, x_1^{(i)}$

$\vdots$

$j=n$ $\quad w_n = w_n - a\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\, x_n^{(i)}$

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

> simultaneously update w_j (for $j = 1 \dots n$) and b.

* An alternative to gradient descent — Normal equation.

sometimes used in machine learning libraries.

- Only for linear regression.
- Solve for w, b without iterations.

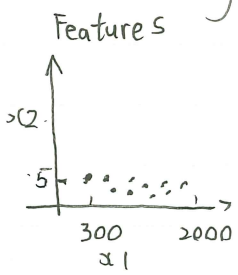
Disadvantage: • Doesn't generalize to other learning algorithms

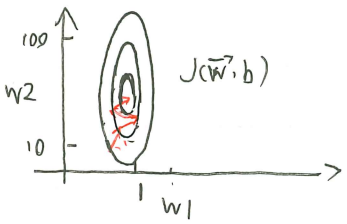
• Slow when number of features is large ($> 10{,}000$)

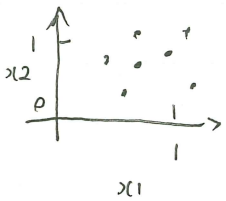
No.

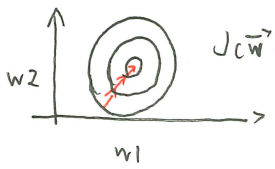
Date

Feature Scaling — can help converge faster



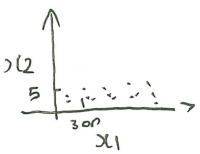




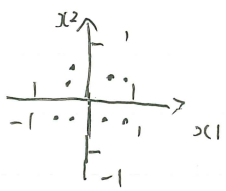


more direct path to minimum.

Mean normalization



$300 \leq x_1 \leq 2000$ $\quad 0 \leq x_2 \leq 5$



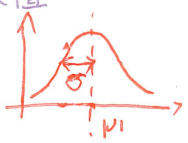
$$x_1 = \frac{x_1 - \mu_1}{2000 - 300} \qquad x_2 = \frac{x_2 - \mu_2}{5 - 0}$$

$-0.18 \leq x_1 \leq 0.82$ $\quad -0.46 \leq x_2 \leq 0.54$

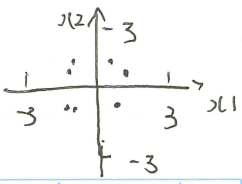
消除量极给分析带来的不便. 但数据的真实意义需还原值

with a similar scale

Z-score deviation — feature center at 0.



standard deviation σ



$$x_1 = \frac{x_1 - \mu_1}{\sigma_1 = 450} \qquad x_2 = \frac{x_2 - \mu_2}{\sigma_2 = 1.4}$$

$-0.67 \leq x_1 \leq 3.1$ $\quad -1.6 \leq x_2 \leq 1.9$.

$$\mu_j = \frac{1}{m} \sum_{i=0}^{m-1} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m} \sum_{i=0}^{m-1} (x_j^{(i)} - \mu_j)^2$$

KOKUYO

No.

Date

(divide maximum)

Feature scaling aim for about $-1 \leq x_j \leq 1$ for each feature x_i

$(-3, 3)$ / $(-0.3, 0.3)$ acceptable

$[0, 3]$ / $[-2, 0.5]$ ok

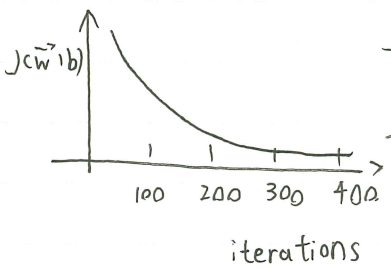
$[-100, 100]$ too large

$[-0.001, 0.001]$ too small → rescale

$98.6 \leq x \leq 105$ too large

Make sure gradient descent is working correctly

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



$J(\vec{w}, b)$ should decrease after every iteration

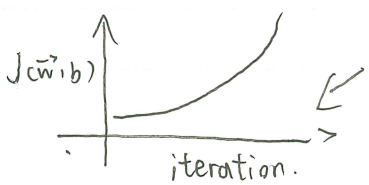
likely converged by 400 iterations

Automatic convergence test

If $J(\vec{w}, b)$ decreases by $\leq \varepsilon = 10^{-3}$ in one iteration.

declare convergence.

(check the graph is always the best way!)



There's a bug / α is too large.

* Use a very small learning rate to check whether there's a bug.

Values of α to try

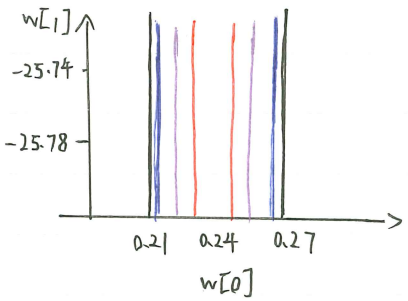
0.001 0.003 0.01 0.03 0.1 0.3 1 .... 3 times bigger every time

pick a suitable α according to graph.

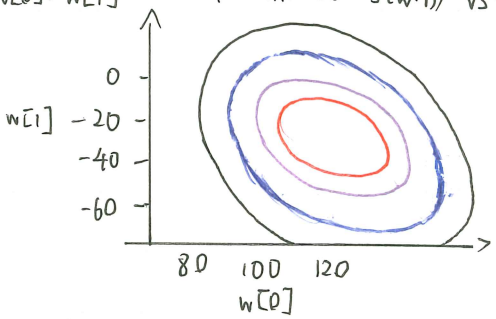
* Any predictions using the parameters learned from a normalized training set must also be normalized

Cost contour with equal scale.

Unnormalized $J(w,b)$ vs $w[0]$, $w[1]$



Normalized $J(w,b)$ vs. $w[0]$, $w[1]$



Feature Engineering:

Using intuition to design new features, by transforming or combining original features.

e.g. x_1 (length) * x_2 (width) = x_3 (area)

Polynomial regression. x = size

$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$

$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + b$

It's important to combine it with feature scaling!

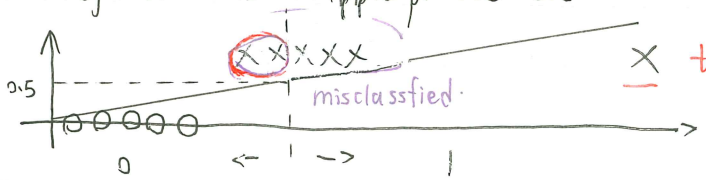
$f_{\vec{w},b}(x) = w_1 x + w_2 \sqrt{x} + b$

No.

Date

Classification.

Linear regression is inappropriate here.

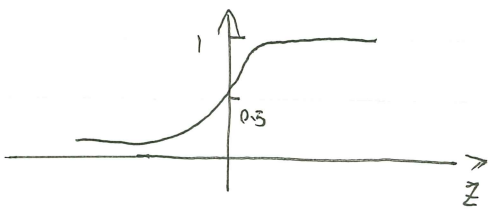


× this cause the line to shift.

Logistic Regression.

We want outputs between 0 and 1

Sigmoid function (logistic function)



$$g(z) = \frac{1}{1+e^{-z}} \qquad (0 < g(z) < 1)$$

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

"Probability"

$f_{\vec{w},b}(\vec{x}) = 0.7 \longrightarrow$ 70% chance that y is 1.

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

Probability that y is 1, given input $\vec{x}$, parameters $\vec{w}, b$

Is $f_{\vec{w},b}(\vec{x}) \geq 0.5$? (threshold)

Yes: $\hat{y} = 1$ No: $\hat{y} = 0$

$g(z) \geq 0.5$?

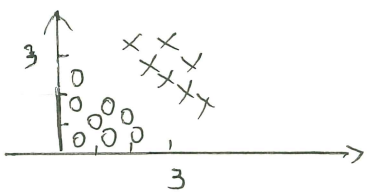
$z \geq 0$?

$\vec{w}\vec{x}+b \geq 0$?

No.

Date

Decision boundary



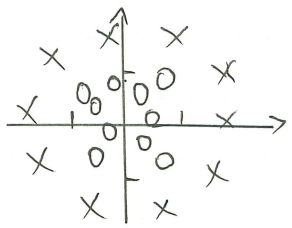
$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1 + w_2x_2 + b)$

if $w_1 = w_2 = 1$, $b = -3$

Decision boundary is $z = 0$

$x_1 + x_2 = 3$

Non-Linear decision boundaries



$f_{\vec{w},b}(\vec{x}) = g(w_1x_1^2 + w_2x_2^2 + b)$

Let $w_1 = w_2 = 1$, $b = -1$

$x_1^2 + x_2^2 = 1$

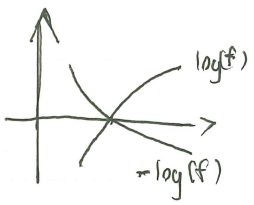
logistic regression $f_{\vec{w},b}(\vec{x}) = \dfrac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

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

non-convex

Logic loss function.

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

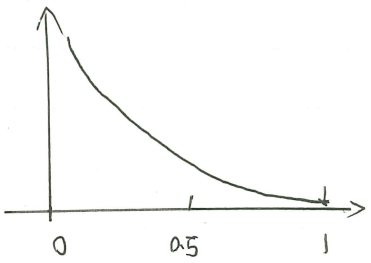


No.

Date

$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$:

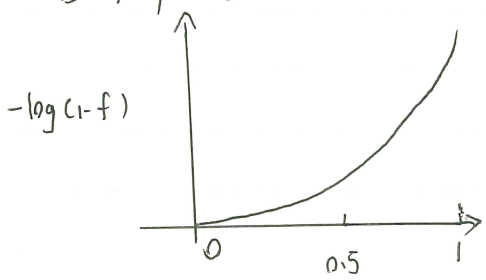
① If $y^{(i)}=1$



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$, then loss $\to 0$ ☺

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$ ☹

② If $y^{(i)}=0$



As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to 0$ ☺

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ loss $\to \infty$ ☹

∴ If model get answer wrong, we penalize model with high loss

Cost $\therefore J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L(f_{\vec{w},b}(x^{(i)}), y^{(i)})}_{\text{loss}}$

it's convex now! $\text{loss} = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=1 \\ -\log(1-f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)}=0 \end{cases}$

find the w.b that minimize the cost

$\therefore L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \boxed{-y^{(i)}\log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1-y^{(i)})\log(1-f_{\vec{w},b}(\vec{x}^{(i)}))}$

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

Campus The function is picked out by maximum likelihood. to reach single global maximum.

No.

Date

Gradient descent for logistic regression

repeat {

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

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

} simultaneous updates.

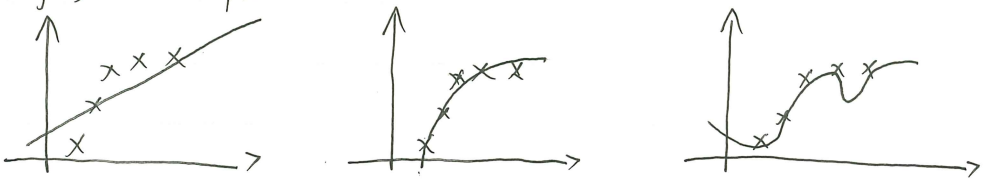
Seemingly same with linear regression

but $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$ here.

- Monitor gradient descent (learning curve) for convergence
- Using vectorization to make gradient descent faster
- Feature scaling also help converge faster.

(Bias and Variances - take minutes to learn
take life to master.

Regression example.



underfit (high biases)
- size: $w_1 x + b$

generalization
size: $w_1 x + w_2 x^2 + b$

overfit (high variance)
size: $w_1 x + \dots + w_4 x^4 + b$

If you have too many features, it might fit too well

Our goal is to predict outcomes correctly for new examples.

A model which does this is said to generalize well.

No.

Date

Address overfitting

1: collect new data

2. select features

throw away unnecessary feature

disadvantage: useful information might be lost

3. regularization

keep all the features, shrink the parameters

$f(x) = 28x - 385x^2 + 39x^3 - 174x^4 + 100$

eliminate

$f(x) = 13x - 0.23x^2 + 0.0000014x^3 - 0.0001x^4 + 100$

small values for w_j

normally no need to regularize b (just $w_1 \cdots w_j$!)

makes a difference!

*extreme examples can increase overfitting

Regularization.

for $x_1 \cdots\cdots x_{100}$

Penalize all the features;

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

"lamda" λ: regularization parameter

mean square error regularization term

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

fit data keep w_j small

λ balance both goals.

$\lambda = 0$ → overfit

$\lambda = 10^{10}$ → underfit

Gradient descent.

repeat {

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

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

$$\frac{\partial}{\partial w_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)})-y^{(i)})x_j^{(i)} + \boxed{\frac{\lambda}{m}w_j}$$

$$\frac{\partial}{\partial b}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)})-y^{(i)}) \quad \text{(don't have to regularize } b\text{)}$$

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

slightly smaller than 1

No.

Date

Regularized logistic regression.

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-z}}$$

$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1x_2^2 + w_5x_1^2x_2^3 + \dots + b.$$

Cost function

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

$$\min J(\vec{w},b) \rightarrow w_j \downarrow$$

$$\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

looks same as the algorithm for linear regression.

except the $f_{\vec{w},b}(\vec{x})$ is different here.

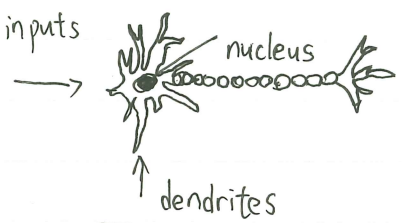
The skill of reduce overfitting is very valuable in real world!

No.

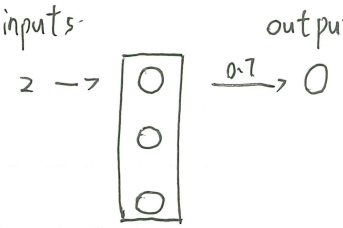
Date

Neural networks

Biological neuron.

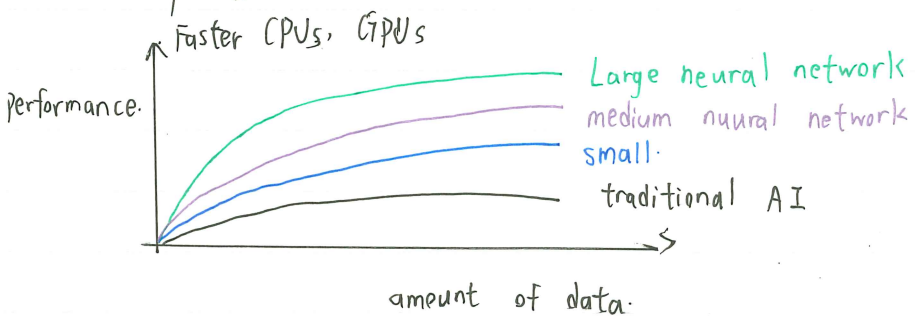


Simplified mathematical model

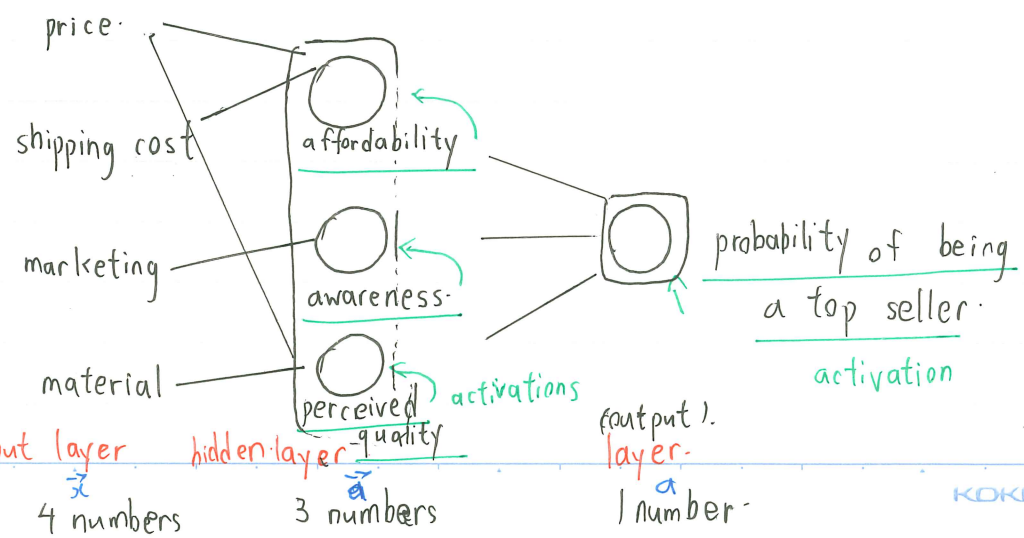


(How brain exactly works is yet to be discovered)

Why Now?



Demand Prediction.



KOKUYO

No.

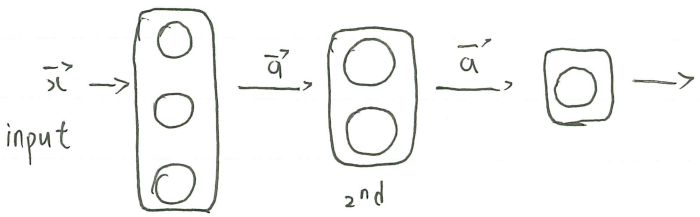
Date

Why neural network is powerful?

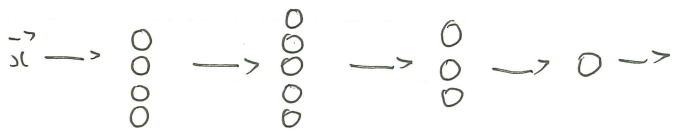
It can Learn its own features that makes it easier to accurately predict

human doesn't need to explicitly decide the features

Multiple hidden layers:



or.



(neural network architecture)

e.g. Face recognition