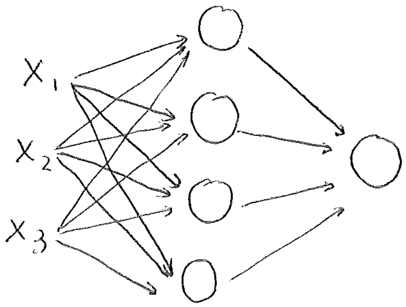
Feed forward Neural Network



Input layer | Hidden Layer | Output Layer

First neuron:

$$Z_1 = W_{11} x_1 + W_{12} x_2 + W_{13} x_3 + b_1$$
$$= W_1 x + b_1$$

W_1 : weight, b_1 : bias.

$$A_1 = \sigma(Z_1)$$

σ : activation function.

Hidden layer

Matrix notation:

$$\underset{4\times 1 \text{ vector}}{Z^{[1]}} = \underset{4\times 3 \text{ matrix}}{W^{[1]}} \underset{3\times 1 \text{ vector}}{x} + \underset{4\times 1 \text{ vector}}{b^{[1]}}$$

Superscript [1] : number of layer.

$$\underset{4\times 1 \text{ vector}}{A^{[1]}} = \sigma(Z^{[1]}) \quad \text{component wise.}$$

Output layer.

$$\underset{\text{real number}}{Z^{[2]}} = \underset{1\times 4 \text{ matrix}}{W^{[2]}} \underset{4\times 1 \text{ vector}}{A^{[1]}} + \underset{\text{real number}}{b^{[2]}}$$

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

As a result.

$$y = A^{[2]} = f(x_1, x_2, x_3)$$

f: composition of Linear functions and nonlinear activation functions

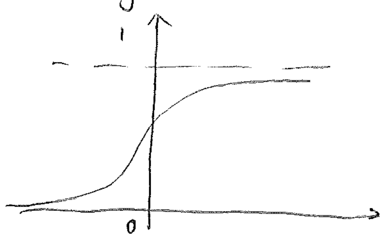
Activation function

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

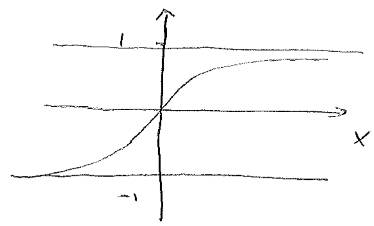
(soft max)

$$\frac{d}{dx}\,\text{sigmoid}(x) = \frac{1}{1-e^{-x}}\left(1-\frac{1}{1-e^{-x}}\right)$$

sigmoid



tanh



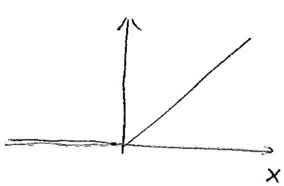
$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$\frac{d}{dx}\tanh(x) = 1-(\tanh(x))^2$$

- tanh (x) is a scaled version of sigmoid (x)
- good for centering data.
- better than sigmoid
- has "vanishing gradient problem"

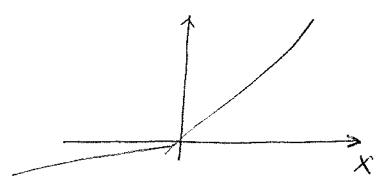
$$\text{Relu}(x) = \max(0, x).$$

Relu



$$\frac{d}{dx}\,\text{Relu}(x) = \begin{cases} 1 & \text{if } x>0 \\ 0. & \text{if } x<0 \end{cases}$$

- default choice.



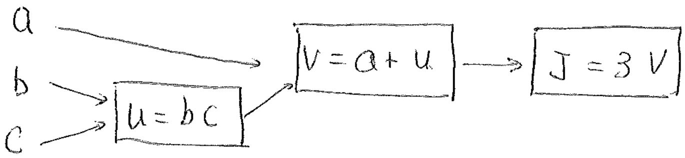
Leaky Relu

Leaky Relu

$$\text{derivative} = \begin{cases} 1 & \text{if } x>0 \\ 0.01 & \text{if } x<0. \end{cases}$$

Training NN

- Backward propagation.



sensitivity of J with respect a :

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial V}\,\frac{\partial V}{\partial a}. \quad \text{(chain rule)}$$

with respect to b.

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial V}\,\frac{\partial V}{\partial u}\,\frac{\partial u}{\partial b}$$

if J is the objective function, in gradient descend, we need

$$a_{n+1} = a_n - \underset{\substack{\uparrow \\ \text{learning rate.}}}{\alpha}\frac{\partial J}{\partial a}$$

- Vectorization.

$$Z = \underset{\substack{\uparrow\\ \text{matrix}}}{W}\ \underset{\substack{\uparrow\\ \text{vector}}}{X} + \underset{\substack{\nwarrow\\ \text{vector}}}{b}$$

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

$$V = \begin{bmatrix} V_1 \\ \vdots \\ V_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{V_1} \\ \vdots \\ e^{V_n} \end{bmatrix}$$

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

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

much faster !

Logistic regression

$$X = \begin{bmatrix} \cdots & x^{(1)} & \cdots \\ & \vdots & \\ \cdots & x^{(n)} & \cdots \end{bmatrix}$$ (n, p) matrix

$$\begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(n)} \end{bmatrix} = W X + \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$$

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

$A = \sigma(Z)$ ← sigmoid.

$$A = \begin{bmatrix} a^{(1)} \\ \vdots \\ a^{(n)} \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

response variable.

(2) denote $dZ = A - Y$

we have seen from note before that

$$\frac{\partial J}{\partial w} = \frac{1}{m} X^T dZ \qquad \frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{n} dz^{(i)}$$

$\parallel$ $\boxed{\text{denote}}$ $\qquad$ $\parallel$ denote

$dW \qquad db$

$dW = 0. \qquad db = 0$

$dW += x^{(1)} dz^{(1)} \qquad db += dz^{(1)}$

$dW += x^{(2)} dz^{(2)} \qquad db += dz^{(2)}$

$\vdots \qquad \vdots$

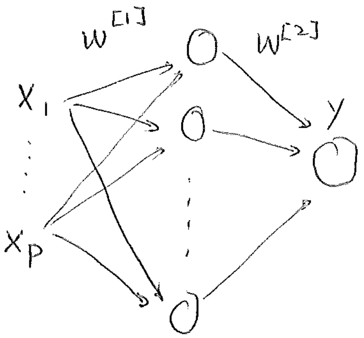
$dW /= m \qquad db /= n.$

(3) $dW = \frac{1}{m} np.dot(X.T, dZ) \qquad db = \frac{1}{n} np.sum(dZ)$

(4) $w -= \partial dw \qquad b -= \partial db.$

Gradient descent for NN

$$J = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \quad : \quad \text{Loss function}$$



sigmoid is chosen as activation function for the output layer.

Forward:

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

Backward propregation:

$$\left.\begin{aligned} dZ^{[2]} &= A^{[2]} - Y \\ dW^{[2]} &= \frac{1}{n} (A^{[1]})^T dZ^{[2]} \\ db^{[2]} &= \frac{1}{n} np.sum(dZ^{[2]}) \end{aligned}\right\}$$

Same as Logistic regression with $A^{[1]}$ as X.

$$dZ^{[1]} = W^{[2]} dZ^{[2]} \underset{\substack{\uparrow \\ \text{element product}}}{*} (g^{[1]})'(Z^{[1]})$$

$$\left(\text{chain rule} \quad \frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \underbrace{\frac{\partial Z^{[2]}}{\partial A^{[1]}}}_{= W^{[2]}} \underbrace{\frac{\partial A^{[1]}}{\partial Z^{[1]}}}_{= (g^{[1]})'(Z^{[1]})} \right)$$

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

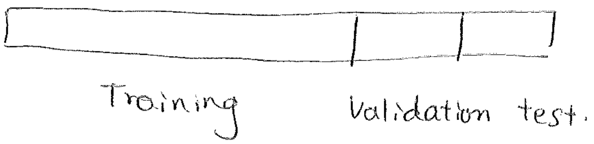
$$db^{[1]} = \frac{1}{n} np.sum(dZ^{[1]})$$

Training for N N.

rule of thumb : 60 - 20 - 20.

Bigdata 1,000,000, - 10,000 - 10,000.

Data.



Validation is used to test different models.

High bias? $\xrightarrow{Y}$ try bigger network
training longer
try different hyper parameter.

↓ N

High variance? $\xrightarrow{Y}$ More data
Regularization.
(e.g. dropout).

↓ N

Done.

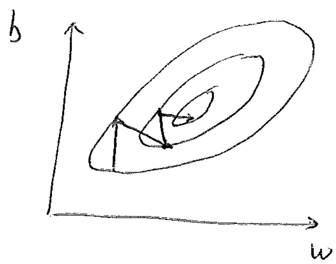
Optimization problem in N N

Setting sup optimization problem.

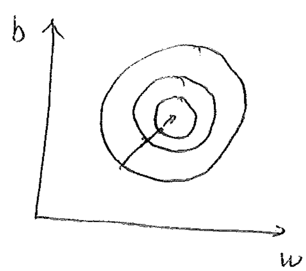
- Normalizing inputs. $x \to \dfrac{x-\mu}{\sigma}$

why?

unnormalized



normalized



- Exploding or vanishing gradients

deep network: $\hat{y} = w^{[L]} w^{[L-1]} \cdots w^{[1]} x$. $b^{[i]} \equiv 0 \ \forall i$.

$$w^{[i]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} . \quad w^{[i]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 1.5^{L} & 0 \\ 0 & 1.5^{L} \end{bmatrix} x \quad \text{or} \quad \hat{y} = \begin{bmatrix} 0.5^{L} & 0 \\ 0 & 0.5^{L} \end{bmatrix} x$$

exploding   vanishing

random weight initialization

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n .$$

$\{w_i\}$ i.i.d,

$\mathrm{Var}(w_i) = \frac{1}{n}$ larger $n \longrightarrow$ smaller w_i

$w^{[i]}$ = np.random.randn(shape) * np.sqrt($\frac{1}{n^{[i]}}$)

optimization

- mini-batch gradient descent

$$x = [\underbrace{x^{(1)}, x^{(2)}, \ldots, x^{(n)}}_{\substack{x^{[1]} \text{ mini batch.} \\ 1000}}, \underbrace{x^{(n+1)}, \ldots, x^{(2n)}}_{x^{[2]}}, \underbrace{\ldots, x^{(m)}}_{x^{[5000]}}]$$

m = 5,000,000

Calculate gradient using mini batch.

repeat { going through data multiple times.

for $t = 1, \cdots, 5000$

1 mini batch. {

Forword prop on $X^{[t]}$

For i in each layer

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

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

compute cost

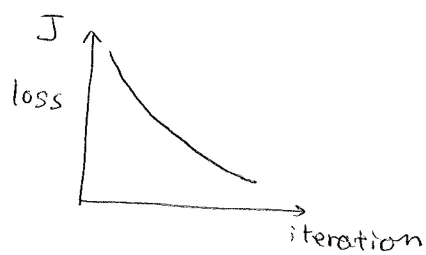
$$J^{[t]} = \frac{1}{1000} \sum_{j=1}^{1000} \mathcal{L}(\hat{y}^{(j)}, y^{(j)}) + \frac{\lambda}{2} \frac{1}{1000} \sum \|W^{[i]}\|^2.$$

Gradient descent to upgrade all $W^{[i]}$

} 1 epoch.

}

Batch gradient descent



Full data

mini-batch

Size of mini-batch.

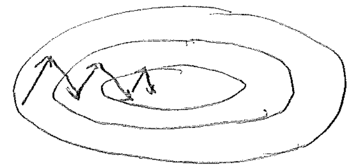
mini batch size = Full data size : Gradient descent

mini batch size = 1 : Stochastic gradient descent

In practice : between 1 and Full data size.

typically 64, 128, 256, 512

Gradient descent with momentum



On iteration t,

compute dw, db on mini-batch.

$$V_{dw} = \beta V_{dw} + (1-\beta) dW$$

$$V_{db} = \beta V_{db} + (1-\beta) db. \qquad \beta = 0.9$$

$$W = W - \alpha V_{dw} \qquad b = b - \alpha V_{db}$$

we want faster descent on horizon direction and slower descent on vectical direction

Oscillation in vectical direction almost cancel each other.

RMS prop

On iteration t,

compute dw, db on mini-batch.

$$S_{dw} = \beta S_{dw} + (1-\beta)(dw)^2$$

$$S_{db} = \beta S_{db} + (1-\beta)(db)^2$$

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw} + \varepsilon}} \qquad b = b - \alpha \frac{db}{\sqrt{S_{db} + \varepsilon}}$$

if db is large, S_{db} is large, the ratio dampen updating in b.

Adam (adaptive momentum estimation)

$V_{dw} = 0$, $S_{dw} = 0$, $V_{db} = 0$, $S_{db} = 0$.

On iteration t,

compute dw, db using mini-batch.

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW \qquad V_{db} = \beta_1 V_{db} + (1-\beta_1) db$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)(dW)^2 \qquad S_{db} = \beta_2 S_{db} + (1-\beta_2)(db)^2$$

$$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t) \qquad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t) \qquad S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$$

$$W = W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S^{corrected} + \varepsilon}} \qquad b = b - \alpha\, V_{db}^{corrected} \Big/ \sqrt{S_{db}^{corrected} + \varepsilon}$$