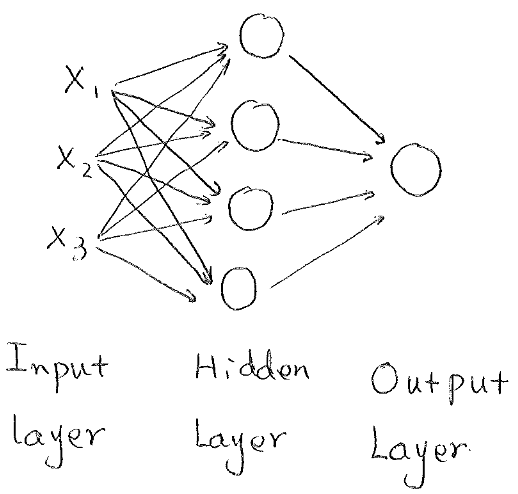
# Feed forward Neural Network



## 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.

$$A_1 = \sigma(Z_1)$$

$\sigma$ : activation function.

## Hidden layer

Matrix notation:

$$\underset{\substack{\uparrow\\ 4\times1\ \text{vector}}}{Z^{[1]}} = \underset{\substack{\uparrow\\ 4\times3\ \text{matrix}}}{W^{[1]}}\ \underset{\substack{\uparrow\\ 3\times1\ \text{vector}}}{x} + \underset{\substack{\uparrow\\ 4\times1\ \text{vector}}}{b^{[1]}}$$

Superscript [1] : number of layer.

$$\underset{\substack{\uparrow\\ 4\times1\ \text{vector}}}{A^{[1]}} = \sigma(Z^{[1]})$$ component wise.

## Output layer.

$$\underset{\substack{\uparrow\\ \text{real number}}}{Z^{[2]}} = \underset{\substack{\uparrow\\ 1\times4\ \text{matrix}}}{W^{[2]}}\ \underset{\substack{\uparrow\\ 4\times1\ \text{vector}}}{A^{[1]}} + \underset{\substack{\uparrow\\ \text{real number}}}{b^{[2]}}$$

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

As a result.

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

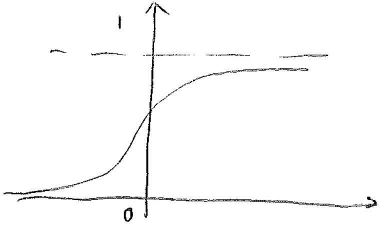
composition of Linear functions and nonlinear activation functions

# Activation function

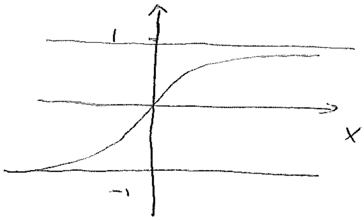
$\text{sigmoid}(x) = \dfrac{1}{1+e^{-x}}$ $\qquad \dfrac{d}{dx}\,\text{sigmoid}(x) = \dfrac{1}{1-e^{-x}}\left(1 - \dfrac{1}{1-e^{-x}}\right)$

( soft max )

sigmoid



tanh

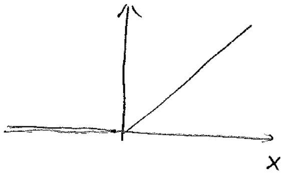


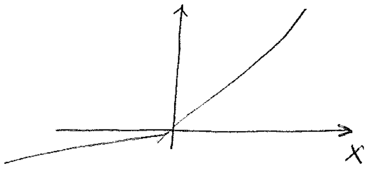
$\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ $\qquad \dfrac{d}{dx}\tanh(x) = 1 - (\tanh(x))^{2}$

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

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

Relu





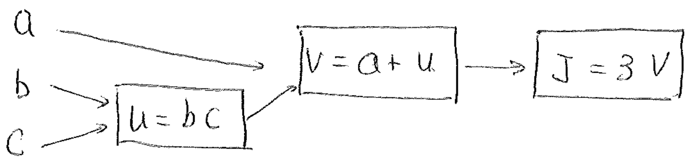
Leaky Relu

- default choice.

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

# Training NN

- Backward propagation.



sensitivity of J with respect a:

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial V}\,\frac{\partial v}{\partial a}.$$

( 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

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

- 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

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

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

(1) $Z = np.dot(w, x) + b$ response variable.

$$A = \underset{\substack{\uparrow \\ \text{sigmoid.}}}{\sigma(Z)} \qquad A = \begin{bmatrix} a^{(1)} \\ \vdots \\ a^{(n)} \end{bmatrix} \qquad \overset{\swarrow}{Y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

(2) denote $dZ = A - Y$

we have seen from note before that

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

$\| \boxed{\text{denote}}$ $\quad\quad$ $\|$ denote

$dW$ $\quad\quad$ $db$

$dW = 0.$ $\quad\quad$ $db = 0$

$dW += x^{(1)} dZ^{(1)}$ $\quad\quad$ $db += dZ^{(1)}$

$dW += x^{(2)} dZ^{(2)}$ $\quad\quad$ $db += dZ^{(2)}$

$\vdots$ $\quad\quad$ $\vdots$

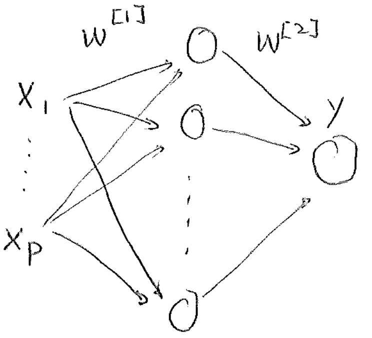
$dW /= m$ $\quad\quad$ $db /= n.$

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

(4) $w$ [illegible] $= -\partial dw$ $\quad\quad$ $b- =$ [illegible] $- \partial db.$

# Gradient descent for NN

$$J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$ : 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 $X$.

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

element product

$$\left( \text{Chain rule} \quad \frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \underset{\substack{\| \\ W^{[2]}}}{\frac{\partial Z^{[2]}}{\partial A^{[1]}}} \underset{\substack{\| \\ (g^{[1]})'(Z^{[1]})}}{\frac{\partial A^{[1]}}{\partial Z^{[1]}}} \right)$$

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

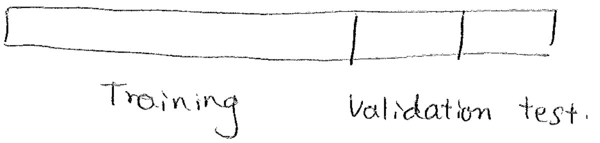
$$db^{[1]} = \frac{1}{n} np.sum(dZ^{[1]})$$

## Training for N N.

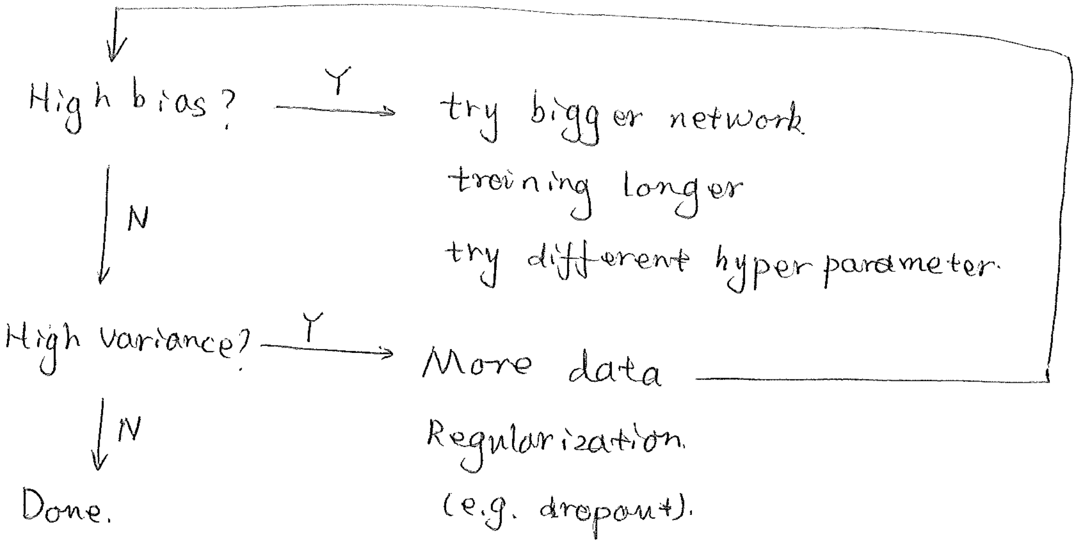
rule of thumb : 60 - 20 - 20.

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

Data.



Validation is used to test different models.



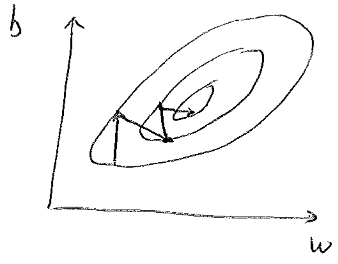
## Optimization problem in N N

Setting sup optimization problem:

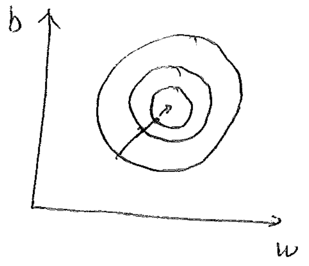
- Normalizing inputs. $x \to \frac{x-\mu}{\sigma}$

why?

unnormalized



normalized



- Exploding or vanishing gradients

deep net work: $\hat{y} = W^{[L]} W^{[L-1]} \dots W^{[1]} x.$ $b^{[i]} \equiv 0.\ \forall i.$

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

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

exploding                vanishing

## random weight initialization

$$Z = W_1 X_1 + W_2 X_2 + \dots + W_n X_n.$$

$\{W_i\}$ i.i.d.

$\mathrm{Var}(W_i) = \frac{1}{n}$    larger $n$ → smaller $W_i$

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

## optimization

- mini-batch gradient descent

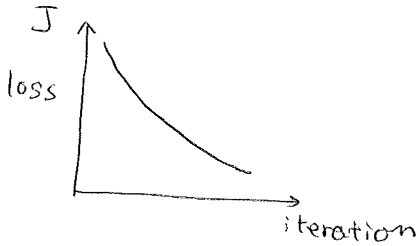
$$x = [\underbrace{x^{(1)}, x^{(2)}, \dots, x^{(n)}}_{X^{[1]}}, \underbrace{x^{(n+1)}, \dots, x^{(2n)}}_{X^{[2]}}, \underbrace{\dots, x^{(\overset{5,000,000}{m})}}_{x^{[5000]}}]$$

$X^{[1]}$ mini batch.
1000

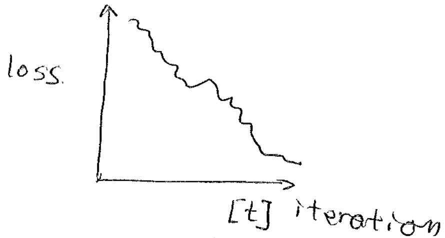
Calculate gradient using mini batch.

```
repeat{ going through data multiple times.
    for $t=1, \cdots, 5000$
        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 mini batch.)                                  (1 epoch.)
}
```

Batch gradient descent



Full data



mini-batch

Size of mini-batch.

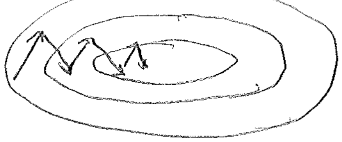
mini batch size = Full data size : Gradient descent

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.

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

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

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

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.

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

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

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

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.

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

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

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

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

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