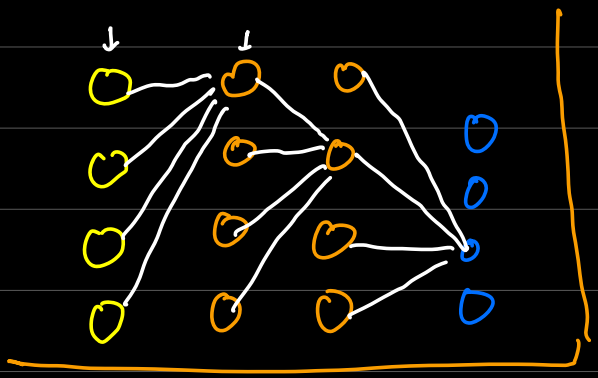
1) Summarize backprop algorithm for MLP

2) Variants of gradient descent

RECAP BACKPROP EXPRESSIONS FOR AN MLP



FINAL LAYER

$e = y - \hat{y}$ or $\frac{dJ}{d\hat{y}}$

$\delta = \frac{dJ}{dz}$, $e = \frac{dJ}{da}$

$$\delta^{(k)} = g'(z^{(k)}) \odot e^{(k)}$$

$$e^{(k)} = W^T \delta^{(k+1)}$$

↳ Transpose of wt used in forward pass

FULL ALGORITHM

1) Initialize weights randomly → Another Class!

2) Do forward propagation or 'forward pass'.

3) $z^{k+1} = W^{(k)} a^{(k)}$

$a^{k+1} = g(z^{k+1})$ → pointwise

Calculate e - at final layer $\rightarrow \frac{\partial J}{\partial \hat{y}} : y - \hat{y}$

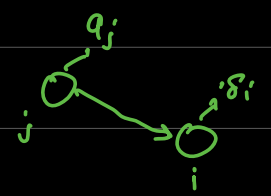
3) Backprop

$$\delta^{(k)} = g'(z^{(k)}) \odot e^{(k)} \qquad R \text{ to } L$$

$$e^{(k)} = (w^{(k)})^T \delta^{(k+1)}$$

Calculate this till the first hidden layer

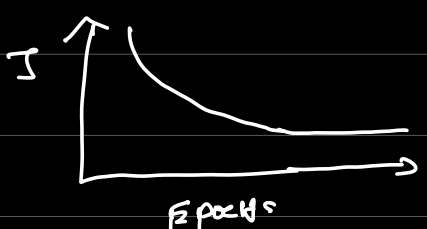
4) $\Delta w_{ij} = +\delta_i a_j \underbrace{\alpha}_{LR}$

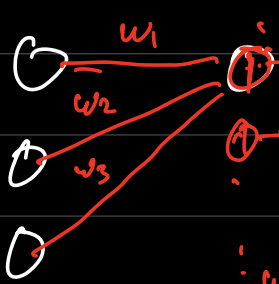


$w_{ij} = w_{ij} + \Delta w_{ij}$ } stochastic gradient descent

5) Repeat 2-4 for all datapts (SGD)

6) Repeat 2-5 till Convergence





$$a \quad w_1 x_1^{(1)} + w_2 x_2^{(1)} + w_3 x_3^{(1)} \rightarrow z^{(1)}$$

$$\vdots \quad w_1 x_1^{(m)} + w_2 x_2^{(m)} + w_3 x_3^{(m)} \rightarrow z^{(m)}$$

$$Z = g\left(\begin{bmatrix} \rightarrow & \bar{z}^{(1)} & \bar{z}^{(2)} & \cdots & \bar{z}^{(m)} \\ & \downarrow & \vdots & & \vdots \\ & \downarrow & 1 & & 1 \end{bmatrix}\right)$$

$$\begin{bmatrix} a^{(1)} & \cdots & a^{(2)} & \cdots & a^{(m)} \\ \vdots & & \vdots & & \vdots \end{bmatrix}$$

Variants of gradient descent

Vector $\in$ $w_{t+1} \leftarrow w_t - \alpha \nabla_w J_i^{(t)}$ ∇_w = gradient wrt w

t - time or iteration number

{one data point at a time}
Is SGD

1) what is α? How to choose α

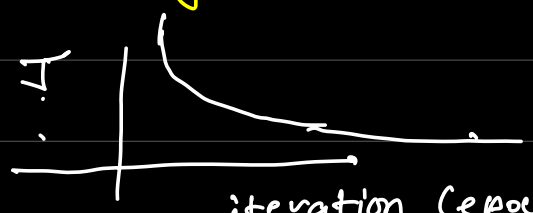
2) Should we change alpha with time?
Meaning - Can we change with each iteration

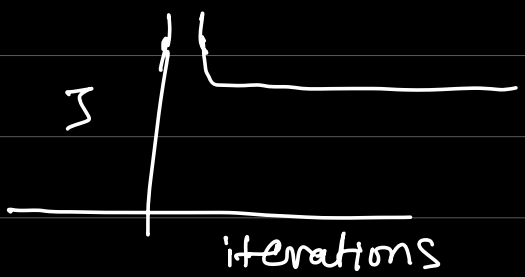
Scheduling techniques -> Are imposed

eg Reduce α by a factor of 10 every 100 epochs

3) Why is α same for all parameters?

4) How do we get out of local minima, saddle points and regions of small gradients?





Solutions: Various adaptive approaches

Momentum, NAG: speeds up Convergence even when gradient is low

Adagrad, RMS prop: Adapt → over time i.e iterations, over parameters

Adam: Hybrid → Goto method

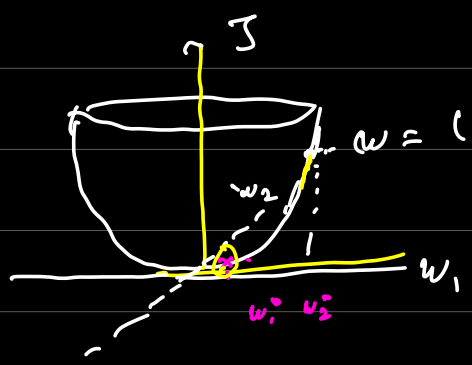
Vanilla GD

- $w \leftarrow w + \Delta w$
- $\Delta w = -\alpha g$
- $g = \nabla_w J$

Momentum

$$\left.\begin{aligned} w &\leftarrow w + m \\ m &= -\alpha g + \beta m \end{aligned}\right\}$$

$$\left\{\begin{aligned} m_t &= -\alpha g_t + \beta m_{t-1} \\ w_t &\leftarrow w_{t-1} + m_t \end{aligned}\right\}$$



$w = (w_1, w_2)$, $\nabla_w J(w_1, w_2)$

$\nabla_w J(w_1^*, w_2^*) \sim 0$

$\Sigma \Delta w \sim 0$

Momentum ← Earlier steps should also effect where we will move in the current step

$$\left\{\begin{aligned} (\Delta w_t) + \\ + m_t &= -\alpha g_t + \beta m_{t-1} \\ w_t &\leftarrow w_{t-1} + m_t \end{aligned}\right\}$$

Initially $m_0 = 0$ $\quad \beta < 1$

$m_1 = -\alpha g_1 + \beta m_0$ → gradient descent

Δw_1

$$m_2 = -\alpha g_2 + \beta m_1$$

$$\Delta w_2 + \beta(-\alpha g_1)$$

$$= \Delta w_2 + \beta \Delta w_1 \rightarrow \text{first gradient update}$$

$\nabla_w J(w)$ → vector

$$\begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{pmatrix}$$

$$m_3 = \Delta w_3 + \beta m_2$$

$$= \Delta w_3 + \beta \Delta w_2 + \beta^2 \Delta w_1$$

Exponential smoothing

$$\left[m_t = \Delta w_t + \beta \Delta w_{t-1} + \beta^2 \Delta w_{t-2} + \cdots + \beta^{t-1} \Delta w_1 \right]$$

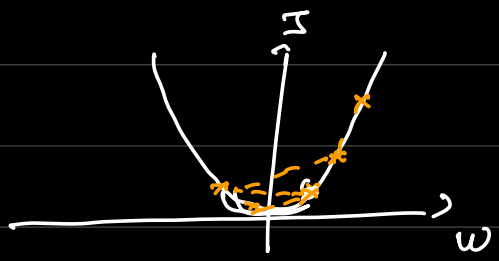
$\beta = 0$, gradient descent

$\beta = 0.1 \rightarrow (10^{-1})^{10} = 10^{-10}$

Nesterov Accelerated Gradient

(NAG)

Reduces the oscillations in momentum



Idea: Look-ahead

Use information about the update you are going to make

Momentum

$w \leftarrow w + m$

$m \leftarrow \Delta w + (\beta m)$

Δw: current; m: previous steps

NAG

$w \leftarrow w + m$

$m \leftarrow \Delta w^* + (\beta m)$

$\Delta w^* = -\nabla J(w^*)$

$w^* = w + \beta m_{t-1}$

$w \leftarrow w_{t-1}$

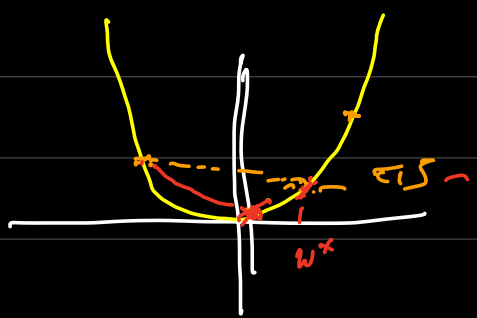
$w^* \leftarrow w + \beta m_{t-1}$ (previous momentum)

↳ look ahead

$\Delta w^* = -\alpha \nabla J(w^*)$

$$\begin{cases} m_t = \Delta w^* + \beta m_{t-1} \\ w = w + m_t \end{cases}$$

↳ Same as momentum



$\alpha \nabla J(w^*)$

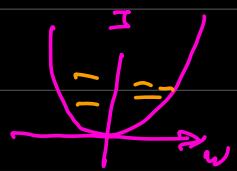
"Over fits"

Adagrad

α – ideally depend on the parameter

→ take into account the gradient wrt every parameter

→ for a particular $w_{ij}^{(l)}$ if it's gradient ↑ then it's alpha_α should ↓



$$w_{t+1} \leftarrow w_t - \frac{\alpha \nabla J_t}{\sqrt{V_i + \varepsilon}}$$

Adaptive update

$$V_i = \sum_{i=0}^{t} G_i^2 \quad \text{(component wise)}, \qquad G_i = (\nabla J)_i$$

↳ sum of all gradients

$t=1$ $\quad$ $t=2$ $\quad \dots$

$$\sim \begin{pmatrix} \left(\frac{\partial J}{\partial w_1}\right)^2 \\ \left(\frac{\partial J}{\partial w_2}\right)^2 \\ \left(\frac{\partial J}{\partial w_3}\right)^2 \end{pmatrix} + \begin{pmatrix} \left(\frac{\partial J}{\partial w_1}\right)^2 \\ \left(\frac{\partial J}{\partial w_2}\right)^2 \\ \left(\frac{\partial J}{\partial w_3}\right)^2 \end{pmatrix} + \dots + \begin{pmatrix} \quad \\ \quad \\ \quad \end{pmatrix}$$

$$+ \qquad + \qquad \dots$$

$$\begin{pmatrix} \left(\frac{\partial J}{\partial w_1}\right)^2 + \dots \\ \vdots \\ \end{pmatrix}$$

$$w_{t+1} \leftarrow w_t - \frac{\alpha \nabla J_t}{\sqrt{V_i + \varepsilon}}$$

$$\rightarrow V_i = \sum_{i=0}^{t} G_i^2 \quad . \quad G_i = (\nabla J)_i$$

↳ Sum of all gradients

α → for each Parameter

→ Scales differently — based on it's gradient history

$w_0 \rightarrow [0, 0.1]$ $[0. \quad 0.01]$

$w_1 \rightarrow [0.5 \quad 0.95]$ ← 10

RMS PROP : ADAGRAD :: NAG : Momentum

↑ smoothed version of Adagrad

→ ADAM → ADAGRAD + MOMENTUM

Adapt α for Parameters

Time adaptation of α

ADAM first → RMSPROP – Adagrad

SGD

ADAM updates → STANDARD

$m_t = \beta_1 m_{t-1} - \alpha \nabla_w J(w_t)$ → momentum update

$v_t = \beta_2 v_{t-1} + (\nabla_w J(w_t))^2$ → Adaptive gradient

$w_{t+1} : w_t - \frac{\alpha m_t}{\sqrt{v_t} + \varepsilon}$

{ ADAGRAD + MOMENTUM }

INPUT DATA PRE-PROCESSING

1) NORMALIZATION
2) STANDARDIZATION → Z-SCORE
3) PCA → Discuss this briefly

1) NORMALISATION

no assumptions

INPUT- x

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \rightarrow [0, 1]$$

↳ What does this transformation accomplish?

x_{min} and x_{max} how are they estimated?

what data is used to estimate x_{min}, x_{max}?

↳ 1) Use training data? → yes!

2) Use training for normalization during training

3) In test data - normalize using x_{min} and x_{max} estimated from test data

4) clip test data

Standardization

1) Assume that features are from a Gaussian distribution

2) Make the distribution 0-mean and 1-standard deviation

8) Features are independent

$$\hat{x}_i = x_i - \bar{x}_i \qquad \bar{x}_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)}$$

↳ training set

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} (\hat{x}_i^{(j)})^2$$

$$x_i \leftarrow \left[\frac{x_i - \bar{x}_i}{\sigma_i}\right] \rightarrow \text{Z-score}$$

$\sim N(0, I)$

BW (LBS) x_1	Ht (feet) x_2	y
100	5.	
200	6	
150	7	
300	4s	
	6f	

$\frac{|x_1|}{|x_2|} \sim 20$

$y \sim BMI$

$$\hat{y} = w_1 x_1 + w_2 x_2$$

Δw_1 and Δw_2 cannot change the "same way"

$w_1 \leftarrow w_1 + \Delta w_1$ ~ 10

$w_2 \leftarrow w_2 + \Delta w_2$ ~ 10

$$\hat{y} = w_1 \cdot 100 + w_2 \cdot 5.$$

↳ This term dominates

$w_1, w_2 \sim 0.6$

can be ignored

Faster - Convergence

x_1^{train} → [-25 .... 125]

$x_{min} = -25$ $x_{max} = 125$

x_1^{test} [-28 (-25), -27 (-25), -45 (-25) ... 120, 150 ...]

$x_{min} = -45$, $x_{max} = 150$

To normalize

$$x \leftarrow \frac{x^{(test)} - x_{min}^{(test)}}{x_{max}^{(test)} - x_{min}^{(test)}}$$

Standardization

Features ~ with different σ^2

σ^2 - 100

σ^2 - 10

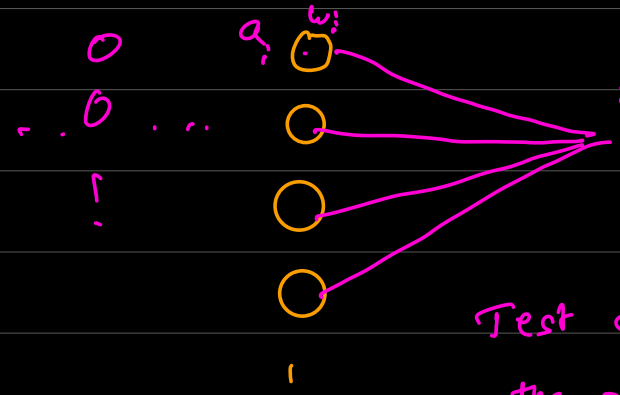
:

X → Y → Regression → Dist. range of Y

classification → one hot vector

If there are outliers in Y (x) → remove them from training

Final layer Fully connected neural network



$\hat{y}$ → Housing price
$[100,000 \ldots 10^6]$

$= \sum w_i q_i$ - linear
- relu

Test data is drawn from the same distribution

Trained $y \in [10^4 \ 10^6]$ → might fail
testing $y \in [10^6 \ 10^9]$

Problems in generalization
lack of generalization

Principal Component analysis

In many real world cases, features might not be independent

⇒ Take an image → pixels are correlated!

$X \rightarrow n$ features, m - dat points, you first construct the Co-variance matrix

Size of the co-variance matrix $(n \times n)$

$x_1 \ldots x_n$

$$\Sigma = \begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{pmatrix}$$

Zero-out non diagonal elements

diagonal → variance of individual features

Represents Co-variance between individual features

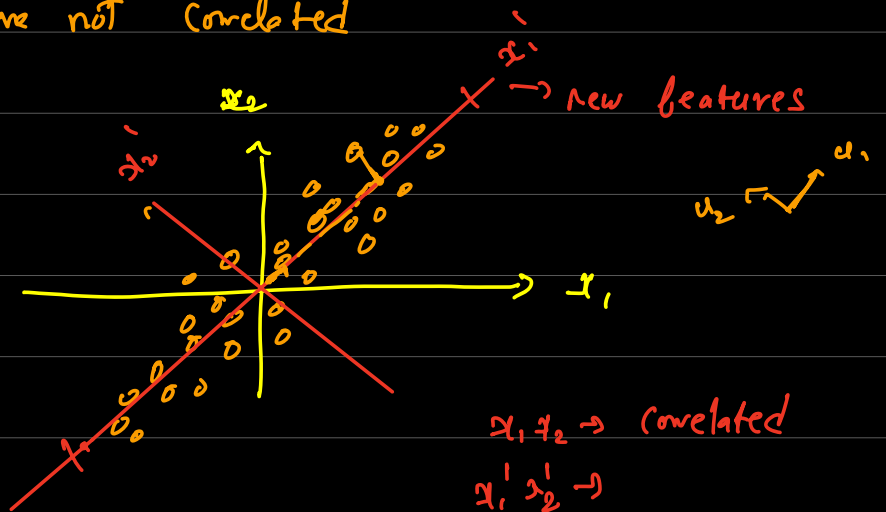
Diagonalization of the co-variance matrix:

Solve the eigen value problem

$$\Sigma \bar{U} = \lambda \bar{U}$$

$\bar{U}$ → eigen vectors, λ → eigen values

U → Can be used to project the features so that they are not correlated



x_1, x_2 → correlated

x_1', x_2' →

x_1' and x_2' are not correlated

You can just use x_1' to represent your data

Variance is captured by $\lambda \rightarrow n$

Throwing out features (transformed) corresponding to smallest $\lambda \rightarrow$ Dimensionality reduction

---

$$\underset{n}{\Sigma} = \underset{n \times m}{?_1}\,\underset{m \times n}{(x-\bar{x})^T (x-\bar{x})} = \underset{n \times n}{\Sigma}$$

$$X' = XU$$

Weight initialization, Batch normalization } Dropouts - L1 and L2 regularization

Convex optimization problems → $w = 0$.

Neural network ← what happens?

look at weight updates equations

$W \sim N(0, I)$ → own set of problems

$W \sim N\left(0, \frac{2}{n}\right)$ - Kaiming He initialization

$\frac{1}{n} \cdot I$

n is the number of input features in a layer