PCA continued...

We wish to project our feature vectors onto a lower dimensional plane

That plane will not be parallel to existing basis planes

PCA helps us to find the vectors that span the new plane

If we collect our feature vectors into one giant matrix:

$$X = \begin{bmatrix} - \ \vec{x}^{(1)T} \ - \\ \vdots \\ - \ \vec{x}^{(n)T} \ - \end{bmatrix}_{N \times d}$$

We want a new feature matrix with reduced dimension $N \times k$, $k < d$

We can follow the same steps as on slides, but subtract the ML estimate of the mean from each $\vec{x}^{(j)}$ ($\hat{\mu}_{ML}$ = sample average)

$$X_{norm} = \begin{bmatrix} \text{---} & (\vec{x}^{(1)} - \hat{\mu}_{ML})^T & \text{---} \\ & \vdots & \\ \text{---} & (\vec{x}^{(N)} - \hat{\mu}_{ML})^T & \text{---} \end{bmatrix}$$

On slides we saw

$$\hat{\Sigma} = \frac{1}{N} X^T X$$

If we divide $X_{norm}^T X_{norm}$ by $N-1$ instead of N, then each term would be

$$\hat{\sigma}_{a,b} = \frac{1}{N-1} \sum_{j=1}^{N} (x_a^{(j)} - \hat{\mu}_{a,ML})(x_b^{(j)} - \hat{\mu}_{b,ML}) \quad \text{for } a,b = 1,\dots,d$$

$$= \hat{\sigma}_{b,a}^2$$

along diagonal: $\hat{\sigma}_{a,ML}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_a^{(i)} - \hat{\mu}_{a,ML})^2$, for $a = 1,\dots,d$

⇒ This product is the unbiased ML estimate of the [illegible] covariance matrix:

$$\hat{\Sigma}_{ML} = \frac{1}{N} X_{norm} X_{norm}^T$$

$d \times d$ $\quad d \times N \quad N \times d$

→ possible semi-definite ($\forall$ vectors $f^T \hat{\Sigma} f \geq 0$ ⇒ all e-values will be $\mathbb{R}$)

You can obtain the eigenvalues and eigenvectors by brute force, or employ existing numerical methods:

in Matlab: $[V, D] = eig(C)$ → some $d \times d$ matrix

- V: matrix of all e-vectors of C
- D: diagonal matrix with e-values of C

In python: import numpy as np

eigenvalues, eigenvectors = np.linalg.eig(C) → some $d \times d$ matrix

vector of e-values vector of e-vectors

you could use SVD for a square matrix, the value "σ", will be the square root of the e-values, "λ":

$$\sigma = \sqrt{|\lambda|}$$

so if you run $eig(\Sigma_{mL})$ and the first λ_1 is returned with the first

$$\underbrace{\vec{V}_1}_{\text{e-vector}} \Rightarrow \Sigma_{mL} \cdot \vec{V}_1 = \lambda_1 \cdot \vec{V}_1$$

what steps do we follow for PCA?

Step1: collect all features into large matrix and mean normalize them:

$$X_{norm} = \begin{bmatrix} -(\vec{x}^{(1)} - \vec{\mu}_{mL})^T - \\ \\ -(\vec{x}^{(n)} - \vec{\mu}_{mL})^T - \end{bmatrix}$$

and use this to obtain the unbiased ML estimate of the covariance matrix

$$\hat{\Sigma}_{ML} = \frac{1}{N} X_{norm}$$

Step 2: obtain eigenvalues / eigenvectors pairs

Identify the largest eigenvalues of $\hat{\Sigma}_{ML}$

⇒ these flag the directions (eigen-vectors with most variance

⇒ these flag the basis for the sub-dimensions for our new feature

If matlab returned

$$V = \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_d \\ | & | & & | \end{bmatrix}$$

$$D = \begin{bmatrix} \lambda_1 & 0 & & 0 \\ 0 & \lambda_2 & \cdots & \\ 0 & & & \lambda_d \end{bmatrix}$$

say you want to reduce your dimensions to 2: select the two largest λ_i

Say two biggest eigenvalues λ_1, λ_2

$\Rightarrow$ basis for your new sub-plane will be

be $\vec{v}_1$ and $\vec{v}_2$

we must remember to normalize these

e-vectors to be unit length

basis: $\vec{u}_1 = \frac{1}{\|\vec{v}_1\|}\vec{v}_1$ $\vec{u}_2 = \frac{1}{\|\vec{v}_2\|}\vec{v}_2$

Step 3: Project each training instance feature vector onto your new subplane

Ex] in our slide we want to project our 3-D feature

vector $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})^T$

for $i=1, \ldots N$ to 2-D vectors

$\vec{z}^{(i)} = (z_1^{(i)}, z_2^{(i)})^T$

for $i=1, \ldots N$

From linear algebra, we know that projection is just the (dot) product

Ex) $\vec{x}^{(i)} \cdot \vec{u}_1 = \langle \vec{x}^{(i)}, \vec{u}_1 \rangle = z_1^{(i)}$

$\vec{x}^{(i)} \cdot \vec{u}_2 = \langle \vec{x}^{(i)}, \vec{u}_2 \rangle = z_2^{(i)}$

$\vec{z}^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$

Apply this projection to all training feature vectors:

$$Z = \begin{bmatrix} -\vec{u}_1^T- \\ \vdots \\ -\vec{u}_k^T- \end{bmatrix}_{k \times d} \cdot \begin{bmatrix} | & | & & | \\ \vec{x}^{(1)} & \vec{x}^{(2)} & \cdots & \vec{x}^{(n)} \\ | & | & & | \end{bmatrix}_{d \times N}$$

$$= \begin{bmatrix} z_1^{(1)} & \cdots & \cdots & z_1^{(n)} \\ \vdots & & & \vdots \\ z_k^{(1)} & & & z_k^{(n)} \end{bmatrix}_{k \times N} = \begin{bmatrix} | & | & & | \\ \vec{z}^{(1)} & \vec{z}^{(2)} & \cdots & \vec{z}^{(n)} \\ | & | & & | \end{bmatrix}$$

Def | the covariance matrix's largest eigenvalues is termed "ipal component" or "its s ndius"

NOTE

Do not forget that all new instances must also be projected onto the k-dimensional space before prediction or classification.

NOTE

The information of the d-k dimensions is lost with PCA

Choice of k <d

Variance is our metric of information.

total variance $\underbrace{\text{Original}}_{\text{before PCA}} \triangleq \sum_{j=1}^{d} \hat{\sigma}^2_{ML,j}$

The contribution (%) of each of the original features

is proportion of feature $j = \frac{\hat{\sigma}^2_{w.j}}{\text{total variance original}}$

We have to select k:

I) you have a memory/ storage limitation,

or you want to visualize data

$\rightarrow$ k is chosen for you

II) Else: select k such that the total variance of "new" feature does not lose more than,

say 5% or 10% of total original variance

$$\frac{\text{total variance new feature, } \vec{z}}{\text{total variance original, } \vec{x}} \geq 0.9 \text{ or } 0.95 \text{ or whatever}$$

III) $D = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \lambda_3 & \\ & & & \ddots \\ 0 & & & \lambda_d \end{bmatrix}$ } much bigger

} much smaller and to represent replaced information

select the bigger one

before [illegible]

Neural

work well with problems like computer vision with pixelated images (hundreds or thousands of features) Assistants are highly trained. Also good speech recognition.

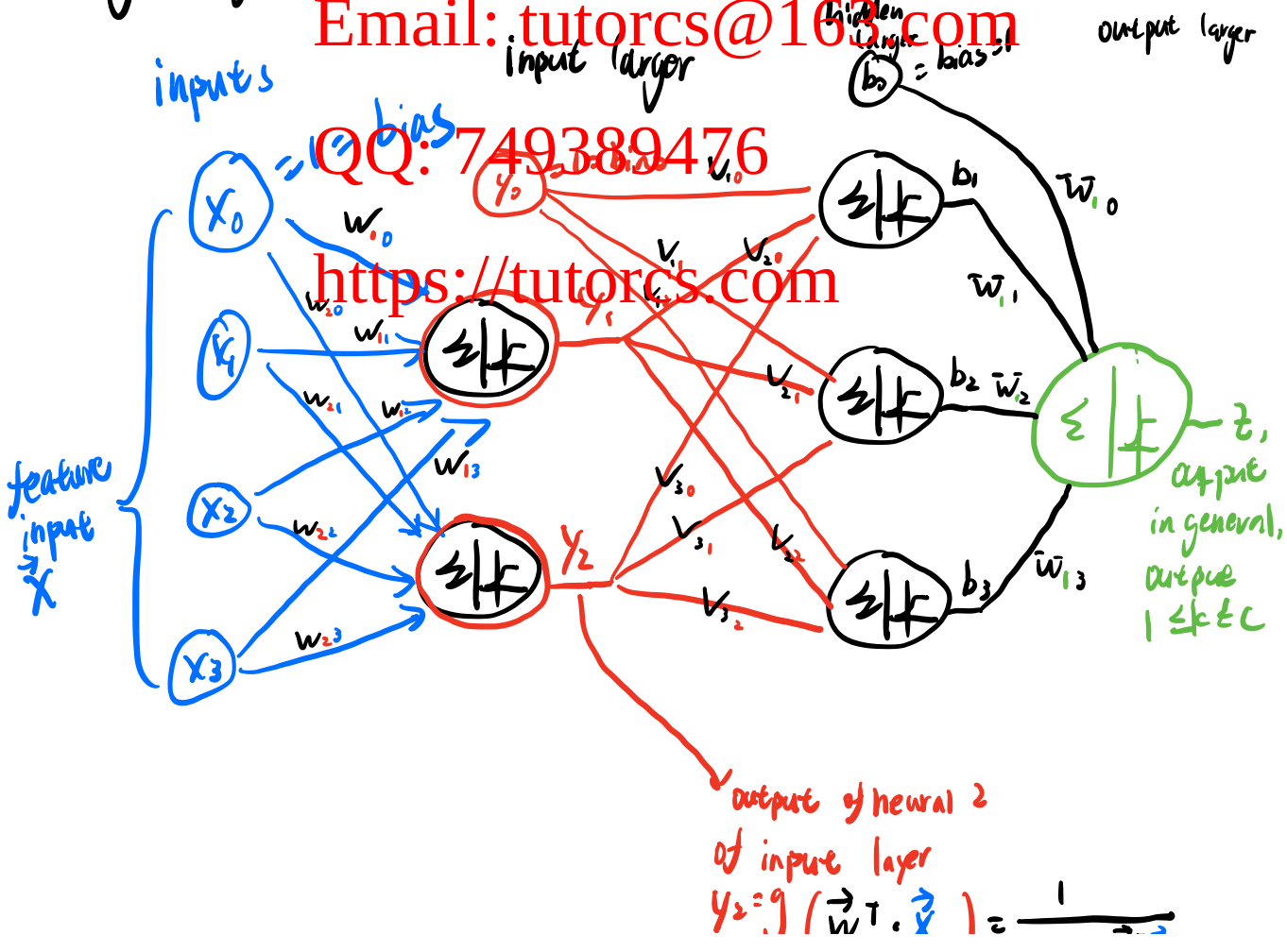
A well-trained neural network is designed (NN) to approximate most arbitrary relationships between features (it's a function approximator)

Setback: NN optimization is plagued with local minima

The network is a type of feed-forward signal along layers of nodes

Each node is a projection (dot product) of the weight previous vector (Ex for input larger, the previous vector is feature $\vec{x}$ and bias 1)

... and then decide if that projection is strong enough (big enough) to be transmitted forward.



$\vec{w}$ = vector of weights, $\vec{x}$ = feature includes bias

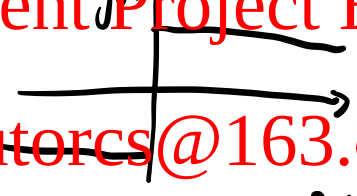
$f(\vec{x}) = 1$ if $e^{\vec{w}^T \vec{x}}$

Note) Activat

can be of other functions:

sigmoid function $g(net) = \dfrac{1}{1+e^{-net}}$

$$g(net) = \begin{cases} -1 & \text{if } net < 0 \\ 0 & \text{if } net = 0 \\ 1 & \text{if } net > 0 \end{cases}$$



delayed step function $g(net) = \begin{cases} 1 & \text{if } net \geq T \\ 0 & \text{if } net < T \end{cases}$

Neural Networks (NN)

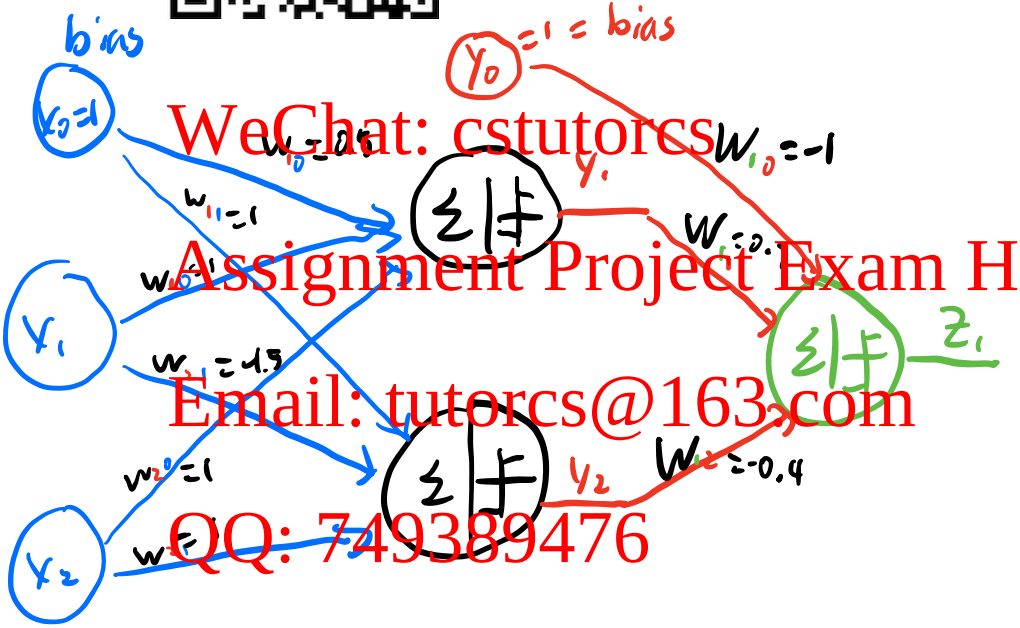
ex) we design a NN that inputs 2 bits and outputs the XOR of those two bits. We allow inputs to be "noisy"

Here, bit = 0 has feature = -1

bit = 1 has feature = +1

We use function

to be in function



We perform "FORWARD PROPAGATION"

x_1 x_2	$t_1 = x_1 \text{ XOR } x_2$	$y_1 = \text{Sign}(\vec{w}_1^T \vec{x})$	$y_2 = \text{Sign}(\vec{w}_2^T \vec{x})$	$z_1 = \text{Sign}(\vec{w}_1^T \vec{y})$
-1 -1	-1	$g([0.5\ 1\ 1]\begin{bmatrix}-1\\-1\\-1\end{bmatrix}) = g(-2.5) = -1$	$g([-1.5\ 1\ 1]\begin{bmatrix}-1\\-1\\-1\end{bmatrix}) = g(-0.5) = -1$	$g([-1\ 0.7\ -0.4]\begin{bmatrix}-1\\-1\\-1\end{bmatrix}) = g(-0.3) = -1$
-1 +1	+1	$g([0.5\ 1\ 1]\begin{bmatrix}-1\\+1\\-1\end{bmatrix}) = g(0.5) = +1$	$g([-1.5\ 1\ 1]\begin{bmatrix}-1\\+1\\-1\end{bmatrix}) = g(-1.5) = -1$	$g([-1\ 0.7\ -0.4]\begin{bmatrix}-1\\+1\\-1\end{bmatrix}) = g(0.3) = +1$
+1 -1	+1	$g([0.5\ 1\ 1]\begin{bmatrix}+1\\-1\\-1\end{bmatrix}) = g(0.5) = +1$	$g([-1.5\ 1\ 1]\begin{bmatrix}+1\\-1\\-1\end{bmatrix}) = g(-0.5) = -1$	$g([-1\ 0.7\ -0.4]\begin{bmatrix}+1\\-1\\-1\end{bmatrix}) = g(0.3) = +1$
+1 +1	-1	$g([0.5\ 1\ 1]\begin{bmatrix}+1\\+1\\+1\end{bmatrix}) = g(2.5) = +1$	$g([-1.5\ 1\ 1]\begin{bmatrix}+1\\+1\\+1\end{bmatrix}) = g(0.5) = +1$	$g([-1\ 0.7\ -0.4]\begin{bmatrix}+1\\+1\\+1\end{bmatrix}) = g(-0.7) = -1$

Even with perturbations, this NN produces a linear boundary (2 lines) splitting an XOR between x_1 & x_2

※

A given NN, you will have served layers, and each layer will have served nods so:

$$\vec{z} = f(\vec{x}, W_{10} \dots W_{2m}, W_{10} \dots W_{2m}, ?)$$