PAGE
DATE

Machine Learning with Python
From Linear models
To Deep Learning

MITx 6.86x

Nawar

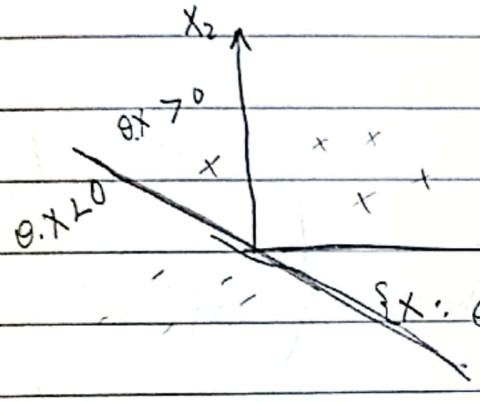
PA
DATE

Lecture 2

Linear classifier and Perceptron

3. Linear Classifiers Mathematically Revisited

* Linear classifiers through origin
  - the dividing line has to go through origin

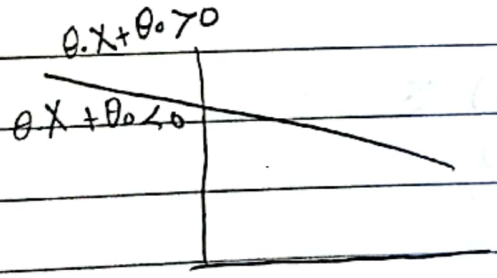


$\{X : \theta_1 X_1 + \theta_2 X_2 = 0\}$ Decision Boundary

$\{X : \theta.X = 0\}$ where $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, $X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$

$h(X;\theta) = \text{sign}(\theta.X)$

* Linear classifier
  - we can move decision Boundry Line every where



$\{X : \theta.X + \theta_0 = 0\}$ d.B

$h(X;\theta,\theta_0) = \text{sign}(\theta.X + \theta_0)$

$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} [X_1\ X_2] + \theta_0$

4. * Linear separation:

Training examples $S_n = \{(x^{(i)}, y^{(i)}), i = 1, \ldots, n\}$ are linearly separable if there exists a parameter vector $\hat{\theta}$ and offset parameter θ_0 such that $y^{(i)}(\hat{\theta}.X^{(i)} + \hat{\theta}_0) > 0$ for all $i = 0, 1, \ldots n$

PAGE
DATE

5. The Perceptron Algorithm:

* Training error for linear classifier

$$E_n(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} [\, y^{(i)} (\theta \cdot x^{(i)} + \theta_0) \leq 0 \,]$$

if $y=1$ & out $=1 \rightarrow 1 > 0$

if $y=-1$ & out $=-1 \rightarrow 1 > 0$

* Learning Algorithm: Perceptron

$\theta = 0$ (Vector)

for $t = 1 \ldots T$ do

  for $i = 1 \ldots n$ do

    if $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$ then

      $\theta := \theta + y^{(i)} x^{(i)}$

return θ

(Without The offset)

$\theta = 0$ (vector) $\quad \theta_0 = 0$ (scalar)

for $t = 1 \ldots T$ do

  for $i = 1 \ldots n$ do

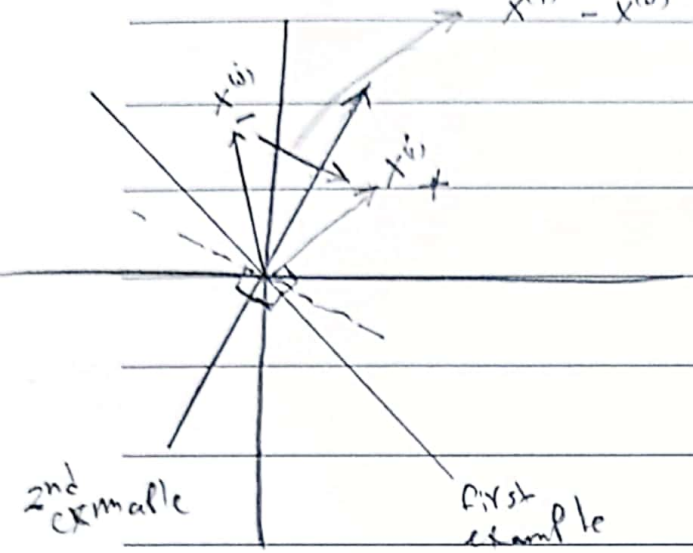
    if $y^{(i)}(\theta x^{(i)} + \theta_0) \leq 0$ then

$$\begin{cases} \theta = \theta + y^{(i)} x^{(i)} \\ \theta_0 = \theta_0 + y^{(i)} \end{cases}$$

return θ, θ_0

$$\begin{bmatrix} \theta \\ \theta_0 \end{bmatrix} = \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix} + y^{(i)} \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}$$

example: The Perceptron Algorithm



let $\theta = 0$

$\theta = \underset{0}{\theta} + \underset{+1}{y^{(i)}} x^{(i)}$ } for $x^{(i)}$

$\theta = x^{(i)}$

$\theta = \underset{\downarrow}{\theta} + \underset{-1}{y^{(j)}} x^{(j)}$ } for $x^{(j)}$

$= x^{(i)} - x^{(j)}$

if $y^{(i)}(\theta x^{(i)}) \leq 0$ → error

PAGE
DATE

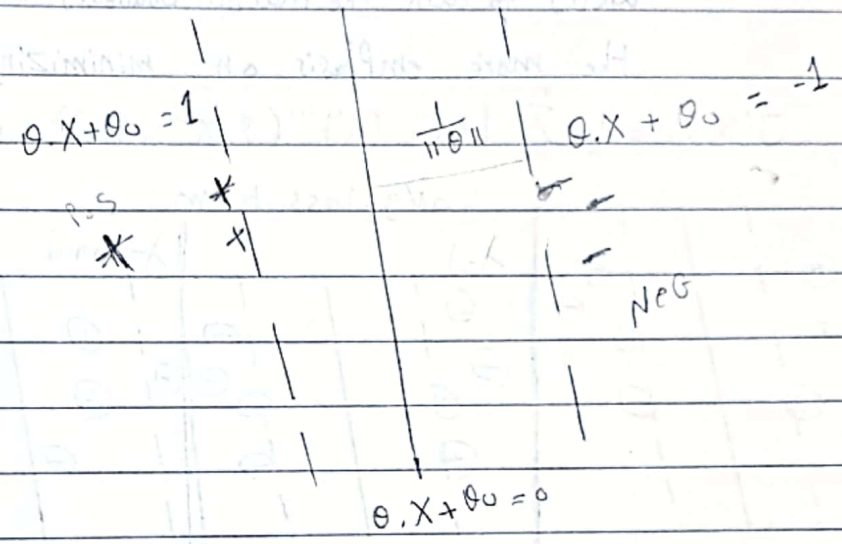
Lectur 3

Hinge Loss, Margin boundaries and Regularization

2. Introduction:

★ Large margin classifier: leaving lots of space on both sides before hitting the training examples.

3. Margin Boundary



4. Hinge Loss and objective function

★ Hinge Loss:

$$\text{Loss}_h(\underbrace{y^{(i)}(\theta \cdot x^{(i)} + \theta_0)}_{z}) = \begin{cases} 0 & \text{if } z \geq 1 \\ 1 - z & \text{if } z < 1 \end{cases}$$

agreement

↳ 1 if correct, -1 if not correct

★ Regularization: towards max margin

$$\max \frac{1}{\|\theta\|} \rightarrow \min \frac{1}{2}\|\theta\|^2$$

★ The objective:

$$J(\theta, \theta_0) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0))}_{\text{Avg Loss}} + \underbrace{\frac{\lambda}{2}\|\theta\|^2}_{\text{Reg}}$$

Reg Parameter

Nawar

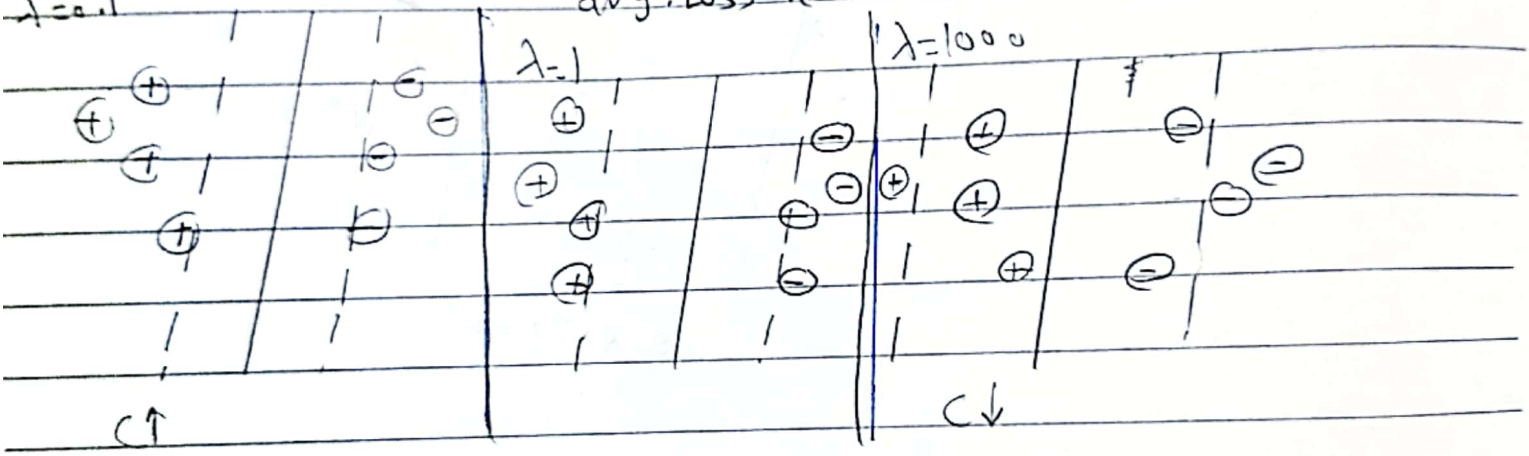
Lecture 4
Linear Classifier and Generalization

2. Review and the Lambda Parameter

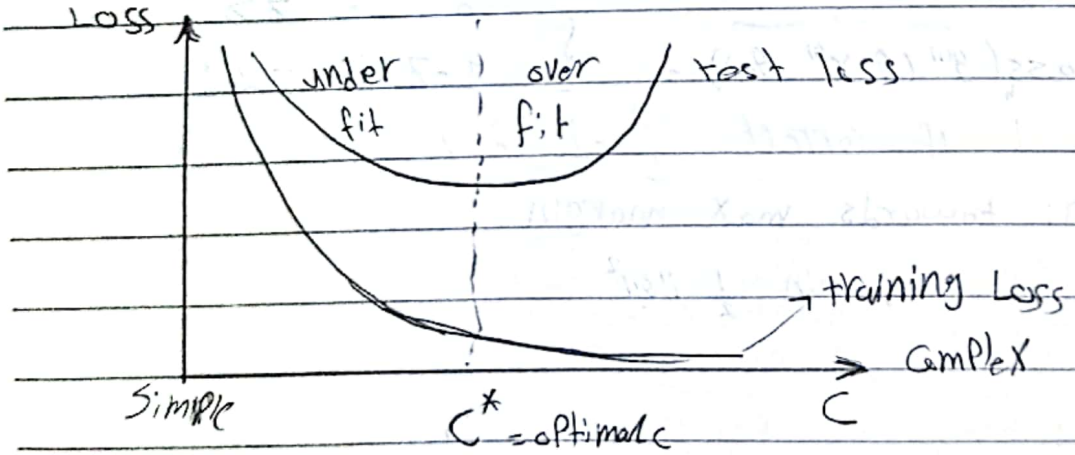
* Lambda: "regularization Parameter": chang the balance between the two terms, Larger λ is the more wetry to Push the margin boundary apart, the smaller λ the more emphasis on minimizing the avg Loss

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} (Loss_h((y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2} \|\theta^2\|$$

avg. Loss term — reg. term



3. Regularization and Generalization:



$$\frac{1}{\lambda} J(\theta, \theta_0) = \left(\frac{1}{\lambda n}\right) \sum_{i=1}^{n} Loss_h(y^{(i)}(x\theta + \theta_0)) + \frac{\lambda}{2} \|\theta\|^2 \quad (\times \frac{1}{\lambda})$$

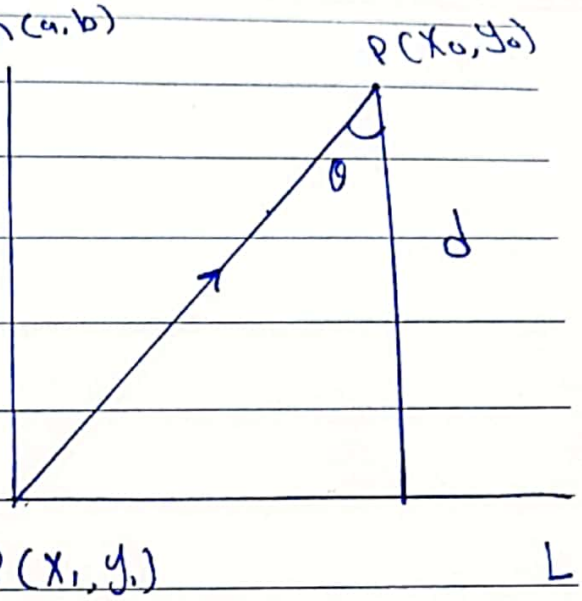
$\frac{1}{\lambda n} \rightarrow C$

Line $L: aX + bY + C = 0$

$P: (X_0, Y_0)$

→ let $\vec{n}$ be a vector normal to the line

→ The distance d is the orthogonal projection of the vector $\vec{QP}$



→ $d = \|\vec{QP}\| \cos\theta$ — ①

$d = \|\vec{QP}\| \cos\theta \cdot \frac{\|\vec{n}\|}{\|\vec{n}\|}$ — ②

$\because \vec{QP} \cdot \vec{n} = \|\vec{QP}\| \|\vec{n}\| \cos\theta \quad \therefore d = \frac{\vec{QP} \cdot \vec{n}}{\|\vec{n}\|}$ — ③

$\because \vec{QP} = (X_0 - X_1, Y_0 - Y_1), \vec{QP} \cdot \vec{n} = (X_0 - X_1, Y_0 - Y_1) \cdot (a, b)$

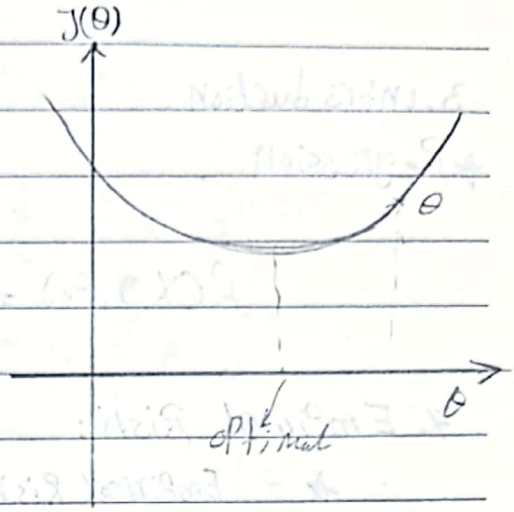
$\|\vec{n}\| = \sqrt{a^2 + b^2} \quad \therefore d = \frac{|a(X_0 - X_1) + b(Y_0 - Y_1)|}{\sqrt{a^2 + b^2}}$

$\because C = -aX_1 - bY_1 \quad \& \quad d = \frac{|aX_0 - aX_1 + bY_0 - bY_1|}{\sqrt{a^2 + b^2}}$

$\therefore d = \frac{|aX_0 + bY_0 + C|}{\sqrt{a^2 + b^2}}$

PAGE
DATE

4. Gradient Descent



$$\theta \leftarrow \theta - \eta \frac{\partial J}{\partial \theta}$$

step size ↵

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial J}{\partial \theta_j}$$

$$\theta \leftarrow \theta - \eta \nabla J(\theta)$$

5. Stochastic Gradient Descent

* It's more efficient way to do optimization.
* Perform gradient descent update with respect to the selected sampled term

$$\theta \leftarrow \theta - \eta \nabla J_i(\theta)$$

→ Random sample i

$$\theta \leftarrow \theta - \eta \nabla_\theta \left[Loss_h(y^{(i)} \theta \cdot x^{(i)}) + \frac{\lambda}{2} \|\theta\|^2 \right]$$

$$\theta \leftarrow \theta - \eta \begin{cases} 0 \quad \text{if } Loss = 0 & + \lambda\theta \\ -y^{(i)} x^{(i)} \text{ if } loss > 0 & + \lambda\theta \end{cases}$$

6. The Realizable Case - Quadratic Program

* support vector machine finds the maximum margin linear separator by solving the quadratic program that corresponds to $J(\theta, \theta_0)$
* In the realizable case, if we disallow any margin violations, the quadratic program we have to solve is:

→ Find θ, θ_0 that minimize $\frac{1}{2}\|\theta\|^2$ subject to

$$y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1$$

Nawar

PAGE
DATE

unit 2
Lecture 5
Linear Regression

3. Introduction:

* Regression :

$$f(X, \theta, \theta_0) = \sum_{i=0}^{d} \theta_i X_i + \theta_0 = \theta X + \theta_0$$

4. Empirical Risk:

* Empirical Risk " The objective"

$$R_n(\theta) = \frac{1}{n} \sum_{t=1}^{n} (y^{(t)} - \theta x^{(t)})^2 / 2 \quad \text{, assume } \theta_0 = 0$$

* There are 2 types of mistakes
  - → structural : when the mapping between vectors and y's is non linear
  - → Estimation : when the mapping is linear but you have a very limited training data, so you can't estimate it correctly

5. Gradient-Based Approach

* $\nabla_\theta (y^{(t)} - \theta x^{(t)})^2 / 2 = (y^{(t)} - \theta x^{(t)}) \nabla_\theta (y^{(t)} - \theta x^{(t)})$
  $= -(y^{(t)} - \theta x^{(t)}) \cdot x^{(t)}$
- → intializ $\theta = 0$ then randomally pick $t = \{1, \dots n\}$
- → update $\theta := \theta + \gamma (y^{(t)} - \theta x^{(t)}) \cdot x^{(t)}$

PAGE
DATE

6. Closed Form solution

$$\nabla_\theta R_n(\theta)_{|\theta=\hat{\theta}} = -\frac{1}{n}\sum_{t=1}^{n} (y^{(t)} - \hat{\theta} x^{(t)})\, x^{(t)}$$

$$= \underbrace{-\frac{1}{n}\sum_{t=1}^{n} y^{(t)} x^{(t)}}_{b} + \frac{1}{n}\sum_{t=1}^{n} \hat{\theta}\, x^{(t)}\, x^{(t)}$$

$$= -b + \underbrace{\frac{1}{n}\sum_{t=1}^{n} x^{(t)} (x^{(t)})^T}_{A}\, \hat{\theta}$$

$$= -b + A\hat{\theta} = 0, \qquad \hat{\theta} = A^{-1} b$$

→ Caveats : - the Matrix should be reversible, it'll be reversible if $(x^{(1)}, x^{(2)}, \dots x^{(n)})$ span $\mathbb{R}^d$

- $n \gg d$, d → dimensions of the feature vector
- Cost, the operation is $O(d^3)$

Matrix is non invertable when $\det(x) = 0$

$x = \begin{bmatrix} 4 & 5 \\ 3 & 2 \end{bmatrix}$ $\det(x) = (4\times2) - (5\times3) \neq 0$ → invertible

8. Regularization :

$$J_{\lambda,\theta}(\theta) = \underbrace{\frac{\lambda}{2}\|\theta\|^2}_{\theta} + R_n(\theta)$$

- Keep all the θs very Close to zero, ↓ Control how much loss we are incurring from training set, to find θs as good as possible match training examples

So because it Pulls the to be in some area, it'll not jumping for every noise in any training example

$$\nabla_\theta J_{\lambda,\theta}(\theta) = \lambda\theta - (y^{(t)} - \theta \cdot x^{(t)})\, x^{(t)}$$

$$\theta := \theta - \eta(\lambda\theta - (y^{(t)} - \theta x^{(t)}) x^{(t)}) = \underbrace{(1-\eta\lambda)}_{\text{small #}}\theta + \eta(y^{(t)} - \theta x^{(t)}) x^{(t)}$$

→ ~~Ridge~~ achieves 2 goals: - Pushs θs in the right direction to minimize the Loss, also Pushes θs down

Nawar

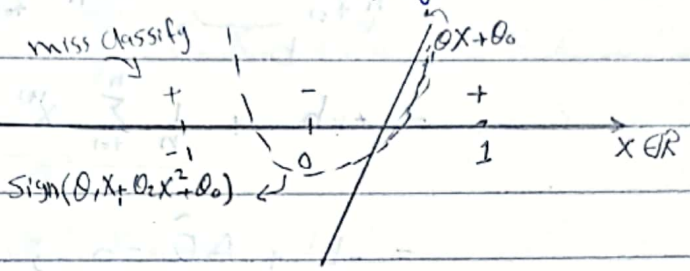
PAGE
DATE

Lecture 6
Non-Linear classification

2. Higher order Feature Vector

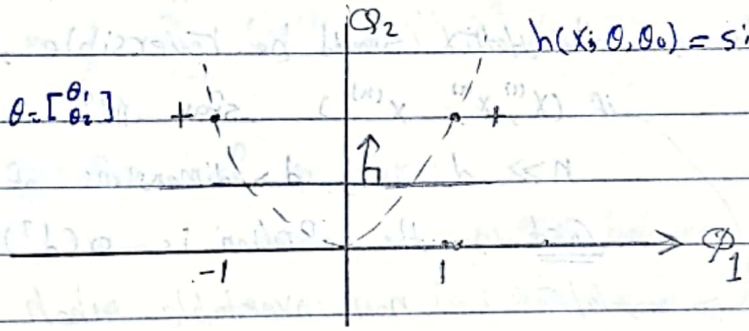
linear classification on the real line: $h(x;\theta,\theta_0) = \text{sign}(\theta x + \theta_0)$

$x \in \mathbb{R}$, θ



Feature space

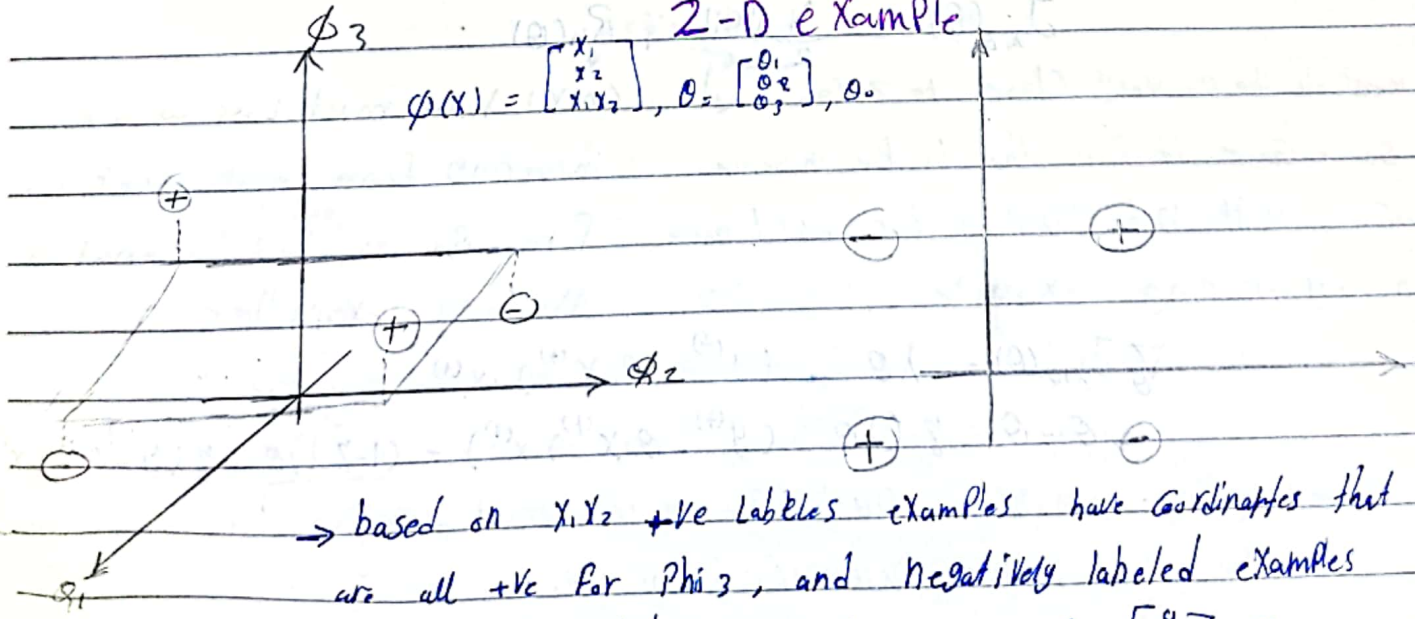
$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix} \in \mathbb{R}^2$, $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$



→ Linear classifier in the new feature coordinates implies a nonlinear classifier in the original x space

2-D example

$\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}$, $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$, θ_0



→ based on $x_1 x_2$ +ve labeled examples have coordinates that are all +ve for Phi 3, and negatively labeled examples have negative values for that coordinates → $\theta = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, $\theta_0 = 0$

Nawar

DATE

3. Introduction to nonlinear classification:

* Polynomial Features:

→ we can add more Polynomial terms $x \in \mathbb{R}$, $\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \\ \vdots \end{bmatrix}$

→ Means Lots of features in high dimensions

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, $\phi(x) = [\, x_1 \;\; x_2 \;\; x_1^2 \;\; x_2^2 \;\; \sqrt{2}x_1x_2 \,]^T \in \mathbb{R}$

→ we can do the same thing with Linear regression

* why not Feature Vector?

→ The vectors can be quite high dimensional.

→ so we will use Kernels

4. Motivation for Kernels: Computationals Efficiency

* Computing inner Products between 2 feature vectors can be cheap even if the vectors are high dimensional

$\phi(x) = [\, x_1, \; x_2, \; x_1^2, \; \sqrt{2}\, x_1 x_2, \; x_2^2 \,]^T$

$\phi(x') = [\, x_1', \; x_2', \; x_1'^2, \; \sqrt{2}\, x_1' x_2', \; x_2'^2 \,]^T$

$K(x, x') = \phi(x) \cdot \phi(x') = (x \cdot x') + (x \cdot x')^2 = x_1x_1' + x_2x_2' + x^2x'^2 + 2x_1x_2x_1'x_2' + x^2x'^2$

$= (x_1x_1' + x_2x_2'), (x_1x_1' + x_2x_2')^2$

→ for some feature maps we can evaluate the inner Products very efficiently $K(x, x') = \phi(x) \cdot \phi(x') = (1 + x \cdot x')^p$, $p = 1, 2, \ldots$

→ in those cases it's advantageous to express the linear classifier in terms of Kernels rather than explicity constructing feature vector

→ we need an equation to solve the classification problem using kernel trick, so we ReCall Perceptron with some difference

Nawar

PAGE
DATE

5. The Kernel Perceptron Algorithm

$\theta = 0$

run through $i = 1, \ldots, n$

if $y^{(i)} \theta \cdot \phi(x^{(i)}) \leq 0$

$\theta \leftarrow \theta + y^{(i)} \phi(x^{(i)})$

from $\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi(x^{(j)})$ → represented the Parameter vector that lives in high-dim feature space in times of number of Mistakes

↳ # of Mistakes Per input j

↳ for prediction → $\theta \phi(x^{(i)})$

$$\theta \cdot \phi(x^{(i)}) = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi(x^{(j)}) \cdot \phi(x^{(i)}) = \sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x^{(i)})$$

How similar the j-th example is to the i-th example

The kernel Perceptron will be:

$\alpha_1 = \ldots \alpha_n = 0$

run through $i = 1 \ldots n$

if $y^{(i)} \sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x^{(i)}) \leq 0$

$\alpha_i \leftarrow \alpha_i + 1$

PAGE
DATE

6. Kernel Composition rules:

* Instead of directly constructing feature vectors by adding coordinates and then taking it in the product and seeing how it collapses into a kernel we can construct kernels directly from simpler kernels.

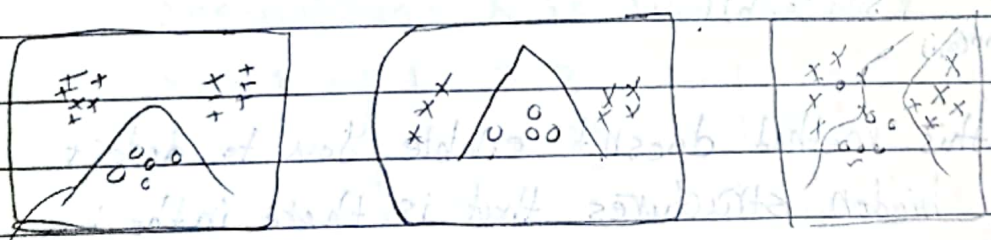
* Composition Rules:

1) $K(x, x') = 1$ is a kernel function $\phi(x) = 1$
2) let $f : \mathbb{R}^d \to \mathbb{R}$ scaler function, $k(x, x')$ is a kernel.
   Then: $\tilde{K}(x, x') = f(x)\, K(x, x')\, f(x') \to \tilde{\phi}(x) = f(x)\, \phi(x)$
3) if $K_1(x, x')$ and $K_2(x, x')$ are kernels, then
   $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel $\phi(x) = \begin{bmatrix} \phi^{(1)}(x) \\ \phi^{(2)}(x) \end{bmatrix}$
4) If $K_1(x, x')$ and $K_2(x, x')$ are kernels then
   $K(x, x') = K_1(x, x')\, K_2(x, x')$ is a kernel

7. The Radial basis Kernel

* The feature vector can be infinite dimensional... this means that they have unlimited expressive power

$$K(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|^2\right)$$



the DB now mapped into the original coordinates, it's linear boundary in ifinite dim space. But Mapped into an original space as (nonlinear)

$$\left\{ x_i \sum_{j=1}^{n} \alpha_j y^{(j)} K(x_i^{(j)}, x) = 0 \right.$$

training example, → * time we have made a mistake

* we can also use Random forest to create non linear classification

Nawar

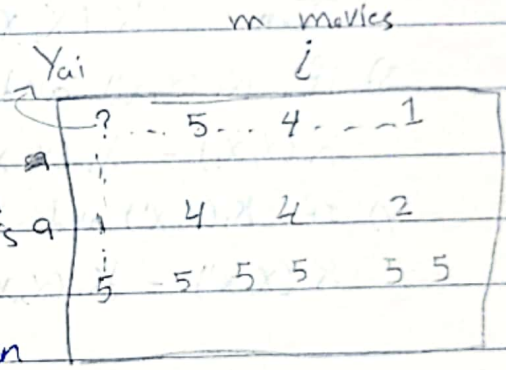
PAGE
DATE

Lecture 7
Recommender systems

3. K-Nearest Neighbors

* How many neighbors you want to look at and this can be one of the hyperparameters
* we will find K number of users who are similar to the desired user, and aggregate them all and take the average

$$\hat{Y}_{ai} = \frac{\sum_{b \in knn(a,i)} Y_{bi}}{K}$$



* we can use any technique to calculate the similiraty like: Cos-similarity, Euclidean

* weighting how similar ~~this~~ the users is to me?

$$\hat{Y}_{ai} = \frac{\sum_{b \in knn(a,i)} sim(a,b)\, Y_{bi}}{\sum_{b \in knn(a,i)} |sim(a,b)|}$$

* The problem: this method doesn't enable you to detect the hidden structures that is there in the data
  - you may be similar to some pool of users in one dim, but similar to some other users in different dim

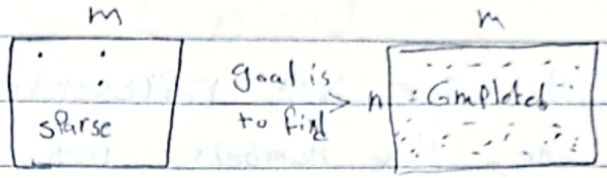
Nawar

PAGE
DATE

4. Collaborative Filtering : The Naive approach.

$$J(X) = \sum_{(a,i)\in D} \frac{1}{2}(Y_{ai} - X_{ai})^2 + \frac{\lambda}{2}\sum_{(a,i)} X_{ai}^2 \quad , \|X\|^2$$

$D = \{(a,i) \mid Y_{ai} \text{ is given}\}$



given $\quad$ X_{output}

* assumption: ① $(a,i) \in D \rightarrow \frac{\partial J(X_{ai})}{\partial X_{ai}} = \frac{\partial}{\partial X_{ai}}(Y_{ai} - X_{ai})^2/2 + \frac{\lambda}{2} X_{ai}^2 = 0$

$$\hookrightarrow X_{ai} = \frac{Y_{ai}}{1+\lambda}$$

② $(a,i) \notin D \rightarrow X_{ai} = 0$

* this solution is worse than before
* there some thing here : the derivative was taken for each entry independently

5. Collabrative Filtering with matrix Factorization:

* The problem with the naive approach was that there is no connection between for all differen entries of X
* we want to decrease number of Paramaters so we will us Factorization.
* Assumption: X is Low Rank

rank 1 : $\begin{bmatrix} 1 & 2 & 3 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ (U: users, V)

$= \begin{bmatrix} 10 \\ 50 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}$

$D(n \times m) \qquad D(n+m)$

$X = UV^T$, $X_{ai} = U_a V_i$ (numbers)

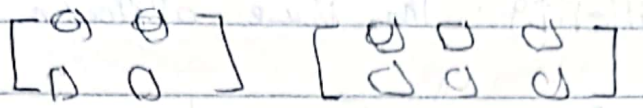
V_i we present every single movie in one number, this is assumption of how all the users like or didn't like the movies

U_i, the general sentiment about the movies for each user we are recording

Nawar

PAGE
DATE

Rank 2 : U V



U: the user here represented by 2 numbers instead of one, these numbers may represent their opinion about the first class & second class of movies

V: now we have the movies represented in 2 numbers so we will have 2 groups (movies) / kind of aspects

★ Selecting "K" the rank by try different values on the validation set

6. Alternating Minimization

$X_{ai} = u_a v_i$: rank 1

$$J(u,v) = \sum_{(a,i)\in D} (Y_{ai} - u_a v_i)^2/2 + \frac{\lambda}{2}\sum_{a=1}^{n} u_a^2 + \frac{\lambda}{2}\sum_{i=1}^{m} v_i^2$$

example : $Y = \begin{bmatrix} 5 & ? & 7 \\ 1 & 2 & ? \end{bmatrix}$, $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$, $v^T = [v_1\ v_2\ v_3]$

Initialize $v = \begin{bmatrix} 2 \\ 7 \\ 8 \end{bmatrix}$ → $uv^T = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}[2\ 7\ 8] = \begin{bmatrix} 2u_1 & 7u_1 & 8u_1 \\ 2u_2 & 7u_2 & 8u_2 \end{bmatrix}$

→ trying to find u_1 & u_2 to get value as (Y)

$\therefore u_1 = \frac{(5-2u_1)^2}{2} + \frac{(7-8u_1)^2}{2} + \frac{\lambda}{2}u_1^2$

$\frac{\partial w}{\partial u_1} = -66 + (68+1)u_1 = 0$ → $u_1 = 66/1+68 = 66/68$ (assume $\lambda = 1$)

$u_2 = 16/1+53 = 16/53$

Nawar

PAGE
DATE

* Then we take the new u_1 & u_2 and Compute V_1 & V_2 & V_3
* Reapet until Convergence
* Local optimum ادنا بس بنوصل لـ optimal مش مضمون انا نوصل لـ

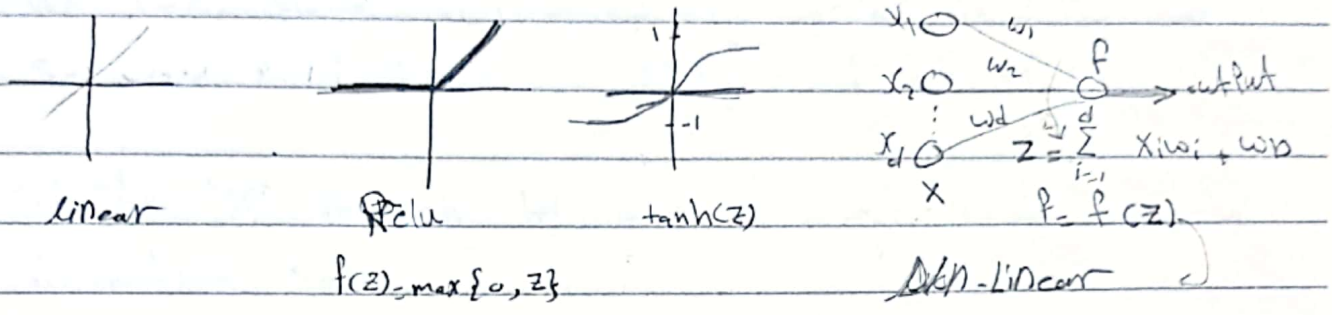
PAGE
DATE

Lecture 8
Intro to FF Neural Networks

3. Motivation:

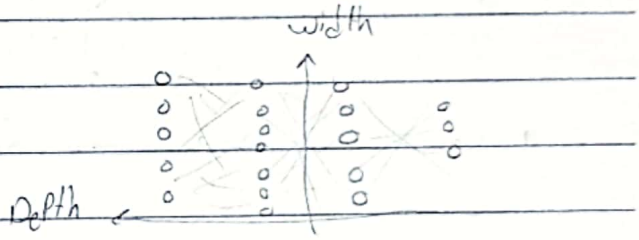
→ Neural Nets tries to optimize the feature representation for the task that we are trying to solve.

4.



5. Deep Neural Nets

Deep: many layers

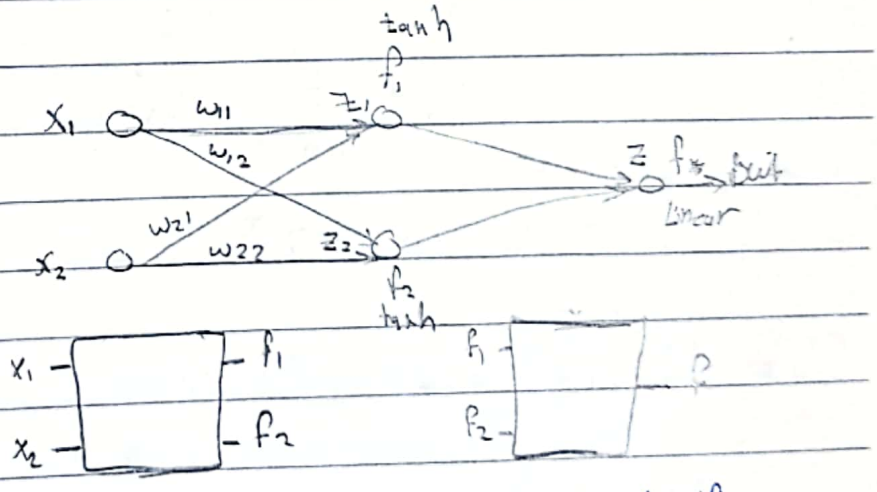


6. Hidden Layer Models

$z_1 = \sum_{i=1}^{2} x_i w_{i1} + w_{01}$

$f_1 = f(z_1) = \tanh(z_1)$

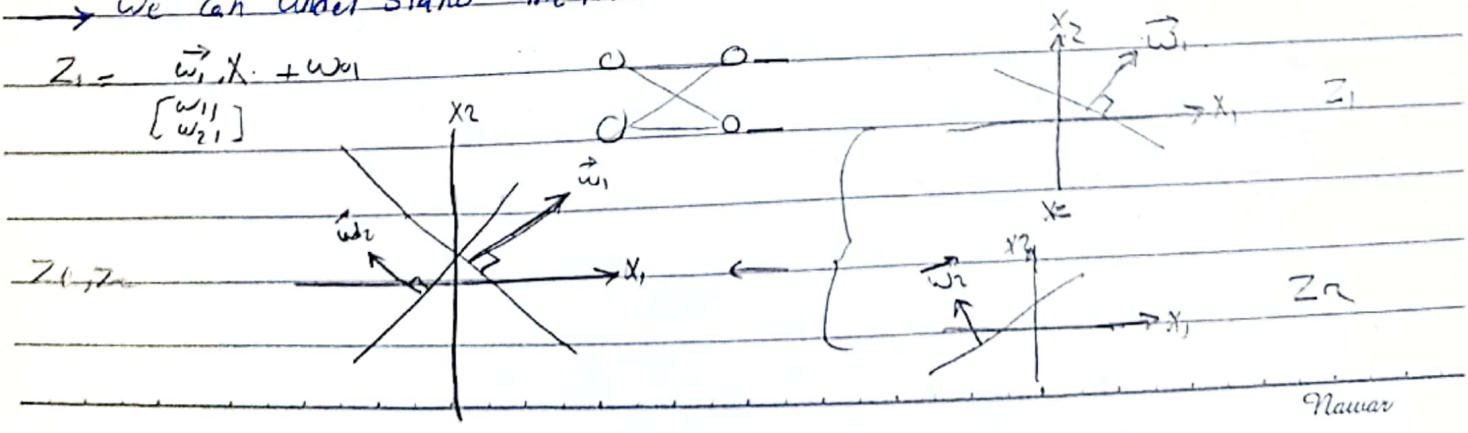
$z_2 = \quad f_2 =$



→ We can understand the first hidden unit as a linear classifier

$z_1 = \vec{w}_1 \cdot \vec{x} + w_{01}$

$\begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix}$

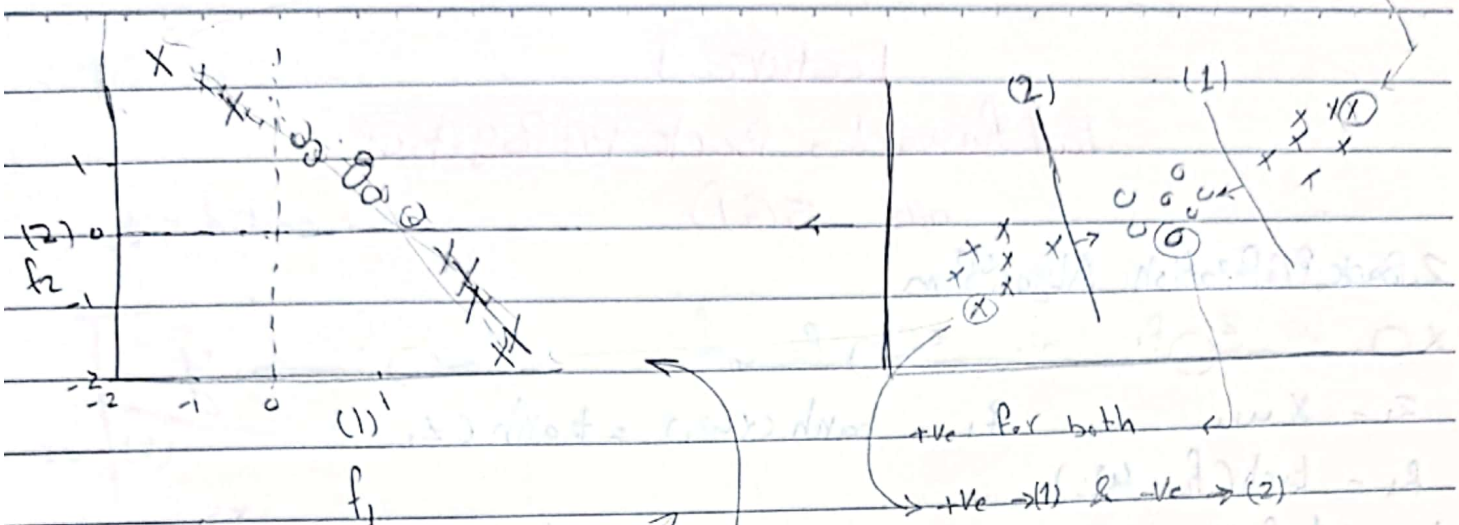


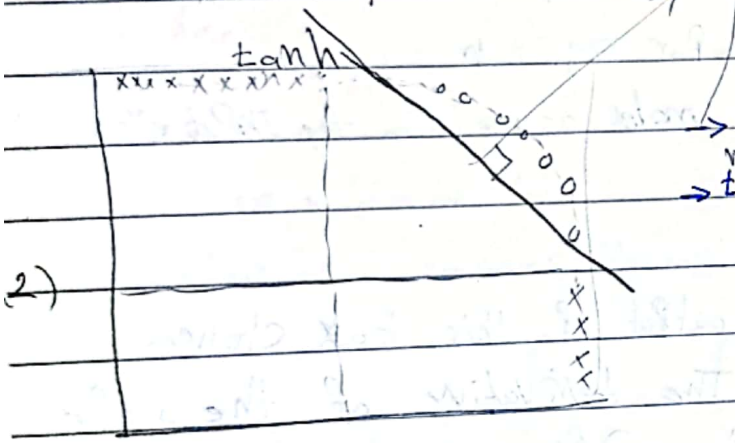
Nawar

PAGE
DATE

Linear

-Ve value for (1) & +Ve for (2)

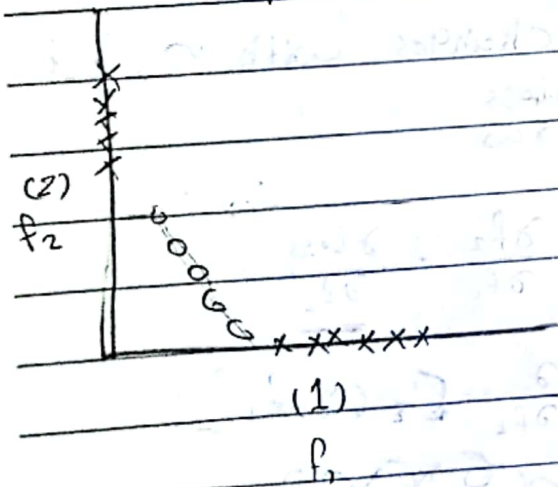




→ Linear activation : not Linearly seperable

→ non-linear tanh activation. the Problem becomes seperable with respect to the hidden unit activation

Relu

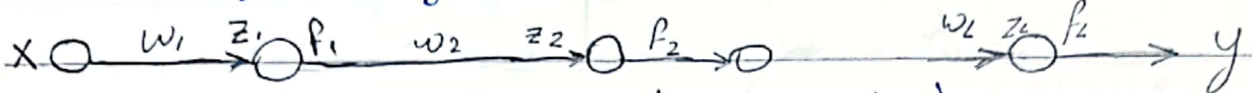


→ this is not actually strictly Linearly seperable

PAGE
DATE

Lecture 1
Feed forward, Back Propagation and SGD

2. Back Propagation Algorithm

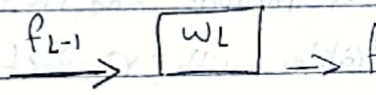


$z_1 = X w_1$ $\quad f_1 = \tanh(X w_1) = \tanh(z_1)$

$f_L = \tanh(f_{L-1} w_L)$

$\text{Loss}(y, f_L) = \frac{1}{2}(y - f_L)^2$ → for regression

Loss is function of all weights in the model as well as the Input example X



→ we need to understand how the output of this box changes as a function of the Input : The derivative of the output with respect to the Input : $\frac{\partial f_L}{\partial f_{L-1}}$

→ we need to evaluate how the Loss changes with respect to changing the Parameters $\frac{\partial \text{Loss}}{\partial w}$

$$\frac{\partial \text{Loss}}{\partial w_1} = \frac{\partial f_1}{\partial w_1} \cdot \frac{\partial \text{Loss}}{\partial f_1} \qquad \frac{\partial \text{Loss}}{\partial f_1} = \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial \text{Loss}}{\partial f_2}$$

$$\frac{\partial \text{Loss}}{\partial f_2} = \frac{\partial}{\partial f_2}\left[\frac{1}{2}(y - f_2)^2\right]$$

Chain Rule

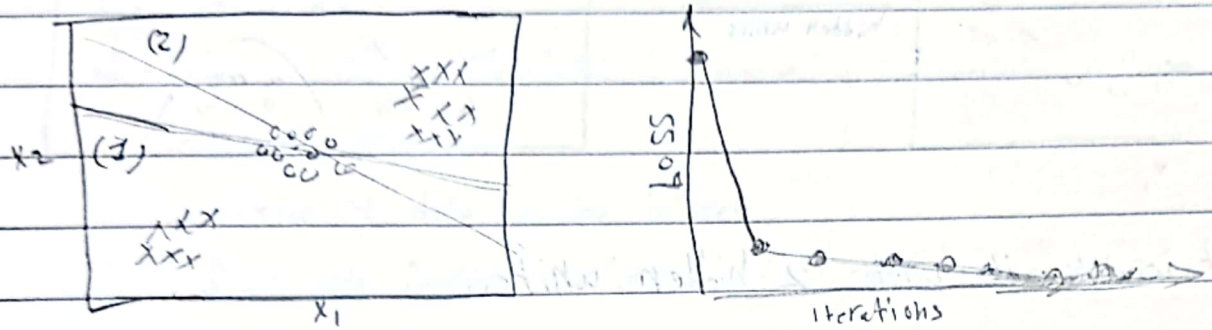
$$\text{So } \frac{\partial \text{Loss}}{\partial w_1} = \frac{\partial f_1}{\partial w_1} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial \text{Loss}}{\partial f_2}$$

Nawar

PAGE
DATE

3. Training Models with 1 hidden Layer:

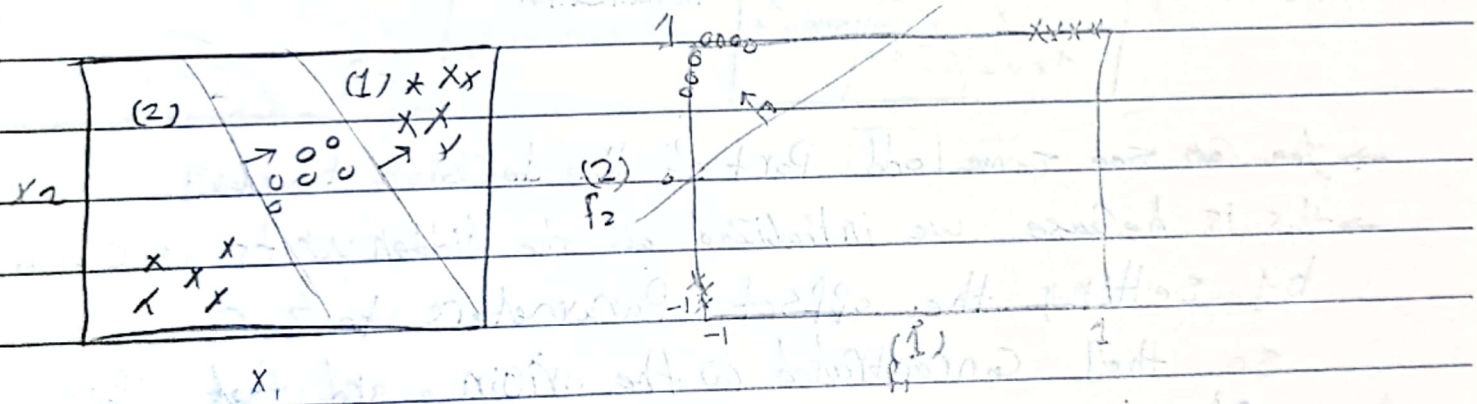
* 2 hidden units



initial network (2 hidden units)

A VG hinge loss

→ after ~10 Passes through the data



hidden units activations

* 10 hidden units

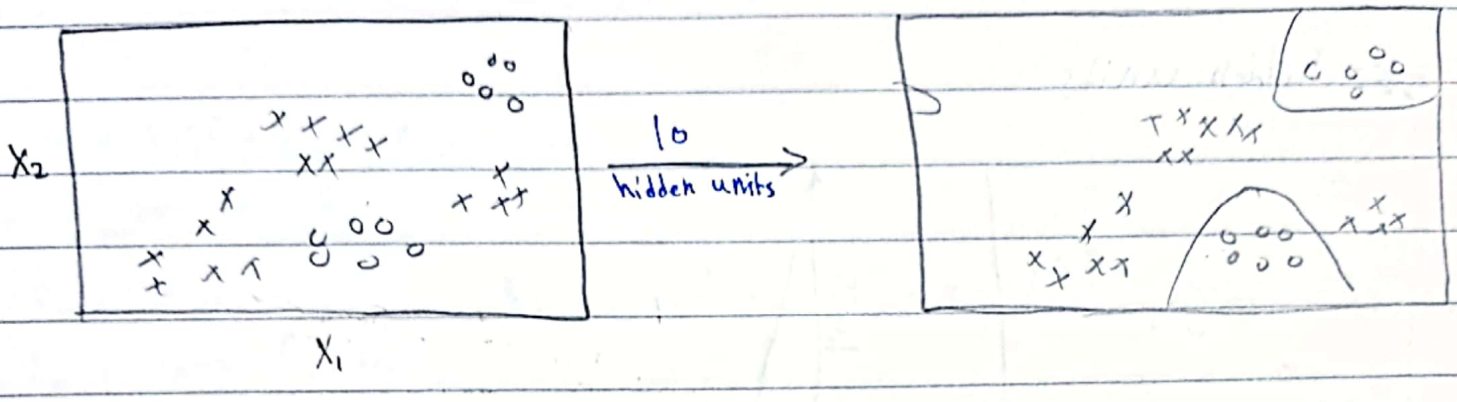
→ after training many of those hidden units end up doing something useful for the classification task

But some of the others doing random things

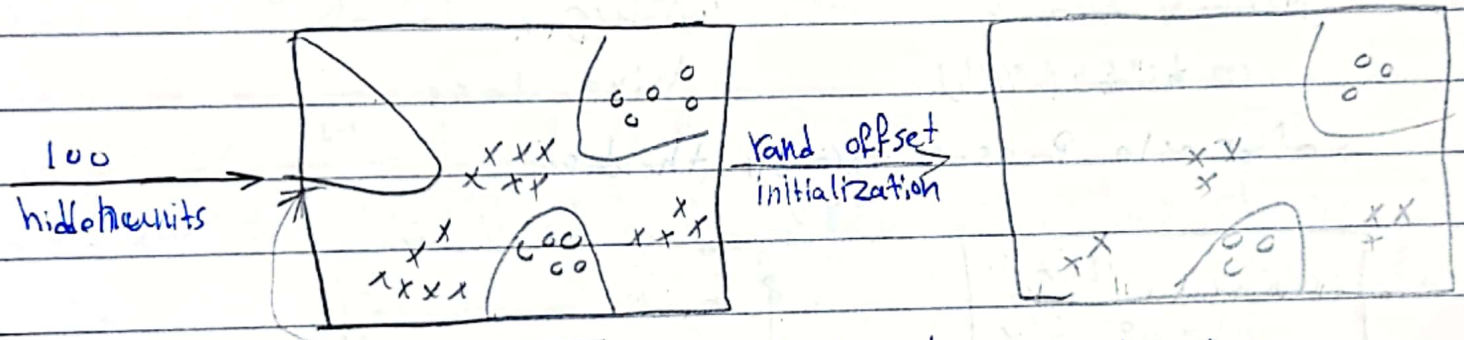
→ this issue called "over capacity"

PAGE
DATE

* 2 hidden units with harder task



→ Can't solve it with 2 hidden units



→ You can see some odd part of the decision boundary
→ this is because we initialized all the hidden units as classifiers by setting the offset parameters to zero so they concentrated @ the origin, and that effect persists in the solution because there is nothing to change that

* Size Vs optimization
→ RELU is (cheap to evaluate, sparsity)
→ easier to learn as Large models.

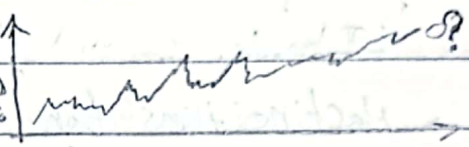
PAGE
DATE

Lecture 10
Recurrent Neural Networks

2. Introduction:

* How to cast (Temporal / sequance) Problems as a supervised learning problem?

→ Historical data can be broken down into feature vectors and target values (sliding window)



$$x(t) = \begin{bmatrix} 0.82 \\ 0.8 \\ 0.73 \\ 0.71 \end{bmatrix} \begin{matrix} y(t-1) \\ y(t-2) \\ \\ y(t-4) \end{matrix} \quad , \quad y(t) = 0.89$$

* Language modeling: what comes next?

"This Curse has been a tremendous ..."

$$\text{tremendous} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad y(t) = ?$$

$$a = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$x(t)$

* → The problem here is that we don't know exactly the length of the history that we should pay attention to.

→ The more history we include the longer the feature vectors and the more complex the learning and prediction.

→ Another problem is that sometimes the history is variable length, There may be words @ the beginning of the sentence that are quite relevant for predicting what happens towards the end.

→ we need more a more flexible way of turning sequences into feature vector.

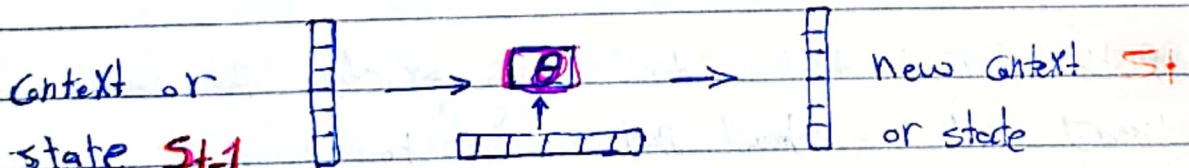
PAGE
DATE

3. Why we need RNNs ?

* Learning to encode/decode
  - → Language modeling
    "This Course has been al ?" → Success
  - → Sentiment Classification
    "I have seen better lectures" → +1
  - → Machine translation
    "I have seen better lectures" → "لقد رأيت محاضرات أفضل"
    Encoding → Decoding
* Encoding is mapping a sequance to a vector.
* Decoding is mapping a vector to a sequance.
* we can encode everything as vectors (Images, sentences, events, etc..), so we can turn the text into Image and vice versa.

4. Encoding with RNN

* Encoding sentences:



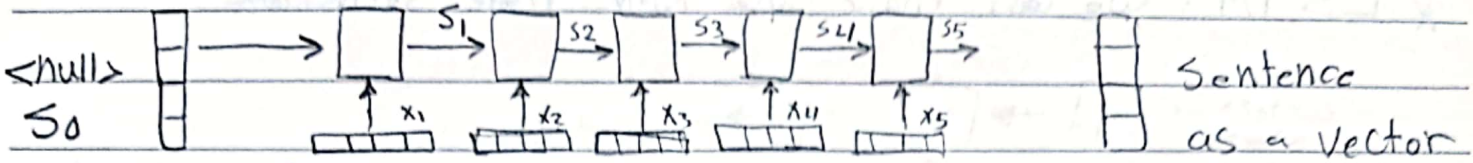
$$S_t = \tanh(W^{S,S} \cdot S_{t-1} + W^{S,X} \cdot X_t)$$

S_t: m×1, $W^{S,S}$: m×m, S_{t-1}: m×1, $W^{S,X}$: m×d, X_t: d×1

→ This is simple transformation that takes previous state and new information, has some parameters represented by these two matrices ($W^{S,S}$, $W^{S,X}$) which are adjustable

→ we can adjust these parameters (θ) so that this box and this transformation behaves as we want it to

PAGE
DATE



"Efforts and Curage are not ....."

→ the sentence vector representation depends on theta, as well as the sequance and we can adjust those parameter to make this representation appropriate for our task

5. Gating and LSTM

* The Problem ~~with~~ with the Previous is that we forget the information that we had, we override it completely to get the new information
* we need to control over what we write and retain in the summaries or the state is typicaly done via another network called a gating network

$$g_t = \text{Sigmoid}(W^{g,s} s_{t-1} + W^{g,x} \cdot x_t)$$

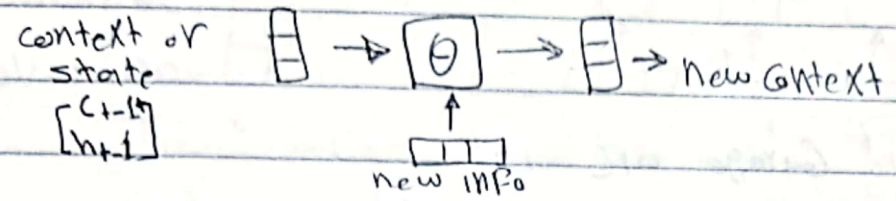
$$s_t = \underbrace{(1 - g_t)}_{\begin{bmatrix}0\\1\\0\\\vdots\end{bmatrix}} \odot s_{t-1} + \underbrace{g_t}_{\begin{bmatrix}1\\0\\1\\\vdots\end{bmatrix}} \odot \tanh(W^{s,s} s_{t-1} + W^{s,x} \cdot x_t)$$

→ g_t control what we should write and retain into the state

→ when $(g_t - 1)$ is zero → that means that we don't retain any information of the Previous state

PAGE
DATE

✱ LSTM: we can make the RNN more sophisticated



$f_t = \text{Sigmoid}(W^{f,h} \cdot h_{t-1} + W^{f,x} \cdot x_t)$ Forget gate

$i_t = \text{Sigmoid}(W^{i,h} h_{t-1} + W^{i,x} \cdot x_t)$ Input gate

$o_i = \text{Sigmoid}(W^{o,h} \cdot h_{t-1} + W^{o,x} \cdot x_t)$ output gate

$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{c,h} \cdot h_{t-1} + W^{c,x} x_t)$ Memory cell

$h_t = o_t \odot \tanh(c_t)$ visible state

→ Forget, input and output gates control how to read information into the memory cell, how to forget information that we've had previously, and how to output information from the memory cell to the visible form

→ The previous state now in 2 parts: c_{t-1} & h_{t-1}

→ input gate: controls to what degree of these coordinates of this suggested signal should enter the memory cell

→ Forget gate: controls which of the coordinates from the previous memory cell we should erase information from.

PAGE
DATE

Lecture 11
Recurrent Neural Networks 2

2. Markov Models:

★ Next word in a sentence depends on Previous symbols already written (history = one, two, or more words)

★ we assume that a limited Vocabulary V, the words that are not in our V are introducing an UNK symbol

→ <beg> symbol specifying the start of the sentence

→ <end> " " " end " " "

<beg> The Lecture leaves me UNK <end>

w_0 w_1 w_2 w_3 w_4 w_5 w_6

★ In first order Markov model (bigram) the next symbol only depends on the Previous one

$$P(w_1, \ldots, w_6) = P(w_1|\underbrace{w_0}_{\text{<beg>}}) \ldots P(w_6|w_5)$$

★ A first order Markov model:

→ Each symbol (except <beg>) is Predicted using the same Conditional Probability table until an <end> symbol is seen

	ML	Course	is	UNK	<end>
<beg>	0.7	0.1	0.1	0.1	0
ML	0.1	0.5	0.2	0.1	0.1
Course	0	0	0.7	0.1	0.2
is	0.1	0.3	0	0.6	0
UNK	0.1	0.2	0.2	0.3	0.2

→ Row sums to 1 and defines the Probability of the next word

<beg> " Course is great UNK " <end>

The Probability	$P(\text{Course}\|\text{beg})$	$P(\text{is}\|\text{Course})$	$P(\text{UNK}\|\text{is})$	$P(\text{<end>}\|\text{UNK})$
of this sentence	0.1	0.7	0.6	0.2

Nawar

PAGE
DATE

* Simple example of how would we use the markov model:

→ Zeroth symbol is always <beg>
→ sample from first row "Probability of distribution over the first word" which comes after the <beg> symbol by throwing a weighted five-way dice
→ we most likely get the word "ML" but we might also get the word "Course"
→ Now move to the "Course" row, we might get the most likely outcome "is" and so on..

* Maximum likelihood estimation
→ The goal is to maximize the Probability that the model can generate all the observed sentences (corpus S)

$$s \in S, \quad s = \{w_1^s, w_2^s, w_3^s, \ldots, w_{|s|}^s\}$$

$$\text{Log}\left\{ \prod_{s \in S} \left[\underbrace{\prod_{i=1}^{|s|} P(w_i^s \mid w_{i-1}^s)}_{\text{for a single sentence}} \right] \right\} \rightarrow \text{The Max Likelihood estimation}$$

* Count (w, w')

$$\hat{P}(w', w) = \frac{\text{Count}(w, w')}{\sum_{\tilde{w}} \text{Count}(w, \tilde{w})}$$

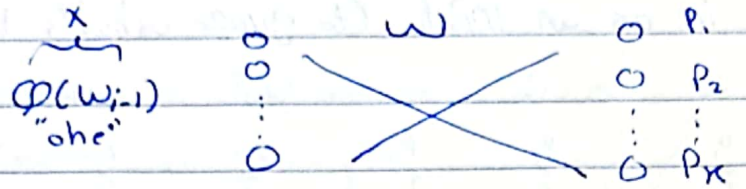
The Max Likelihood estimate of its row in the normalized Probability table.

Nawar

PAGE
DATE

3. Markov Model to Feedforward Neural Nets

→ we can represent the Markov model as a FFNN



$P_K = P(w_{i,K} \mid w_{i-1})$ $\quad P_K > 0$ $\quad \sum_K P_K = 1$

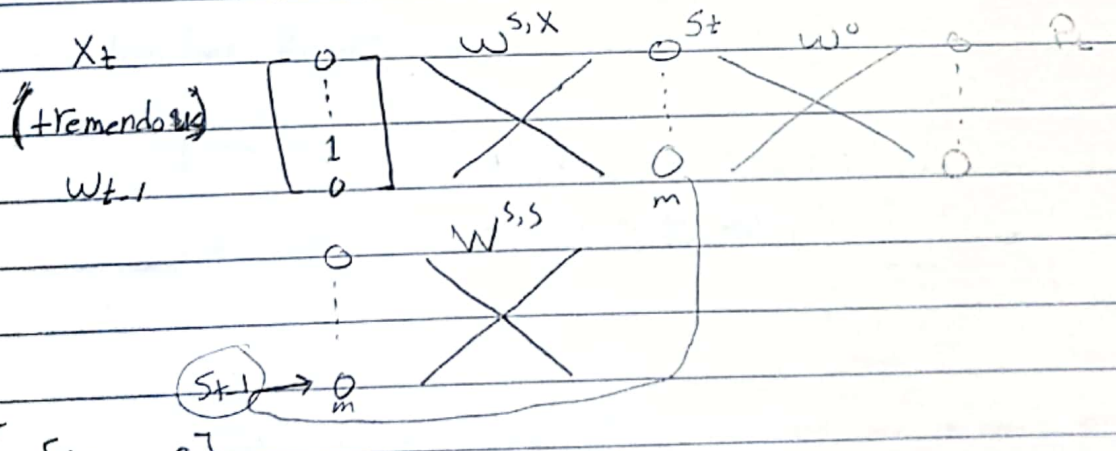
$z_K = \sum_j x_j w_{jK} + w_{0K} \quad \in \mathbb{R}$

$P_K = \frac{e^{z_K}}{\sum_j e^{z_j}} \rightarrow$ softmax

* with FFNN we can extend the Input by adding $\varphi(w_{i-2})$ and add more hidden Units

4. RNN Deeper Dive

"This course has been a tremendous!..."

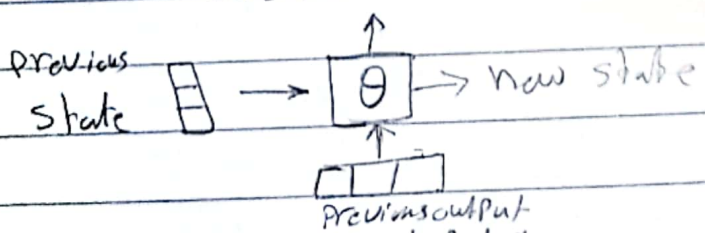


$s_0^T = [0\ 0 \ldots 0]$

$s_t = \tanh(W^{s,s} s_{t-1} + W^{s,x} x_t)$ state

$P_t = \text{softmax}(W^{o} s_t)$

P_t : output distribution

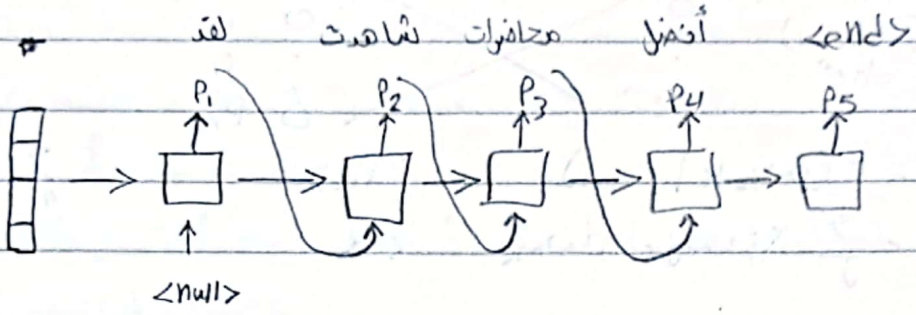


Nawar

PAGE
DATE

5. RNN Decoding :

* The output is fed in as an input (to guage what's left)



Vector encoding of the sentence "I have seen better Lectures"

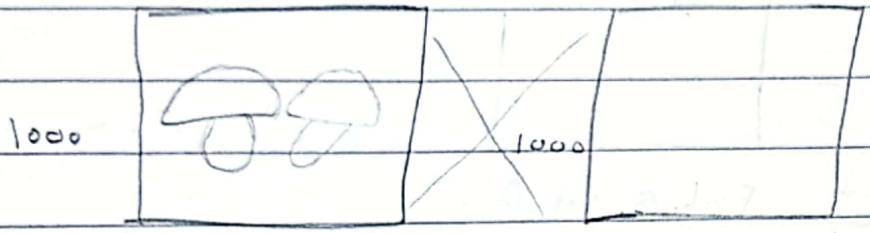
PAGE
DATE

Lecture 12

Convolution Neural Networks

2. Convolution Neural Networks

★ Feed-forward Networks



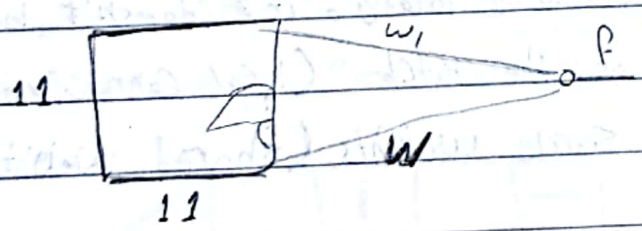
1000 W 1000

input $10^6 \times 10^6$ layer 1

→ Problems: - There is alot of weights are shared $10^6 \times 10^6$

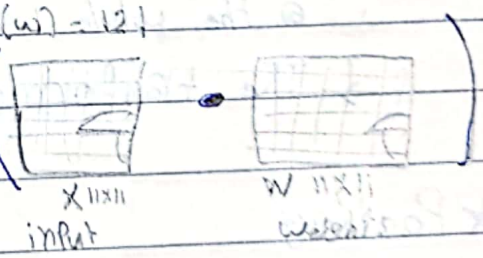
- same object that appears in slightly different locations would have to ~~learn~~ be learned anew with weights.

★ Patch classifier/filter



dim(w) = 121

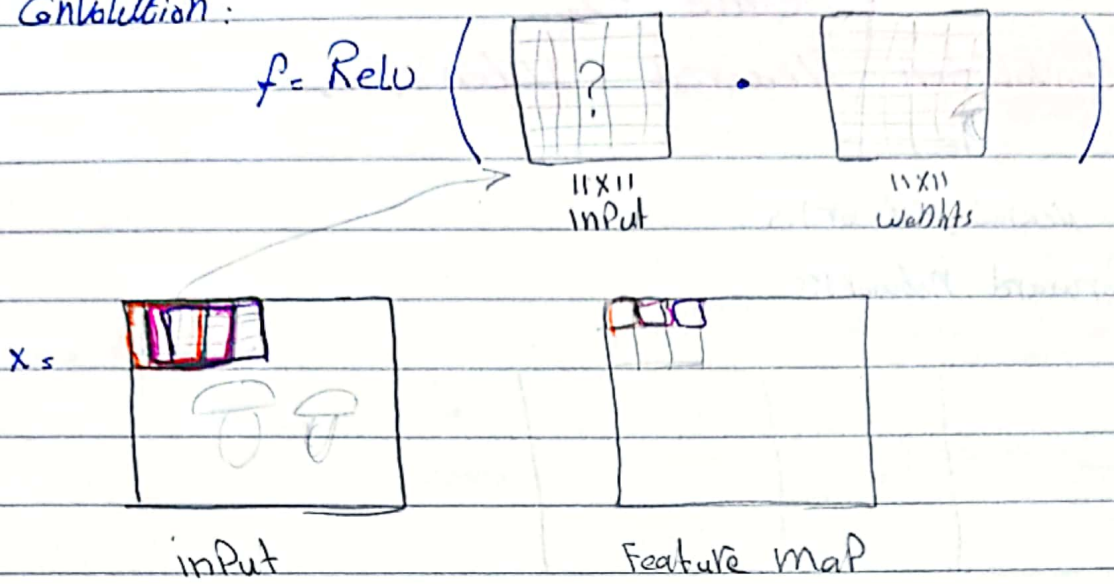
$f = Relu\left(\right)$



→ At this way we can think of the weights as actually an image

PAGE
DATE

✱ Convolution:



→ If we have looking for edges and we pass the Patch ~~Patch~~ on Mashram we might get a higher value in locations where such features appears (Mashrams)

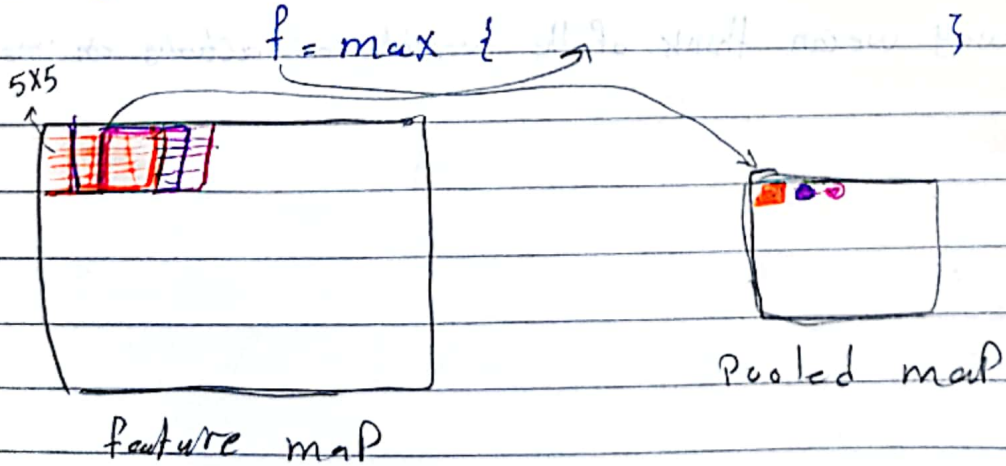
→ The transformation from the input to a feature map involves only 121 weights

→ Each location on feature map "activation of units" tells us how much that feature present in the corresponding location

→ each unit on feature map connects only locally, it doesn't look @ the whole image, it only looks at the Patch (local connectivity)

→ The tranformation uses the same weights (shared weights)

✱ Pooling



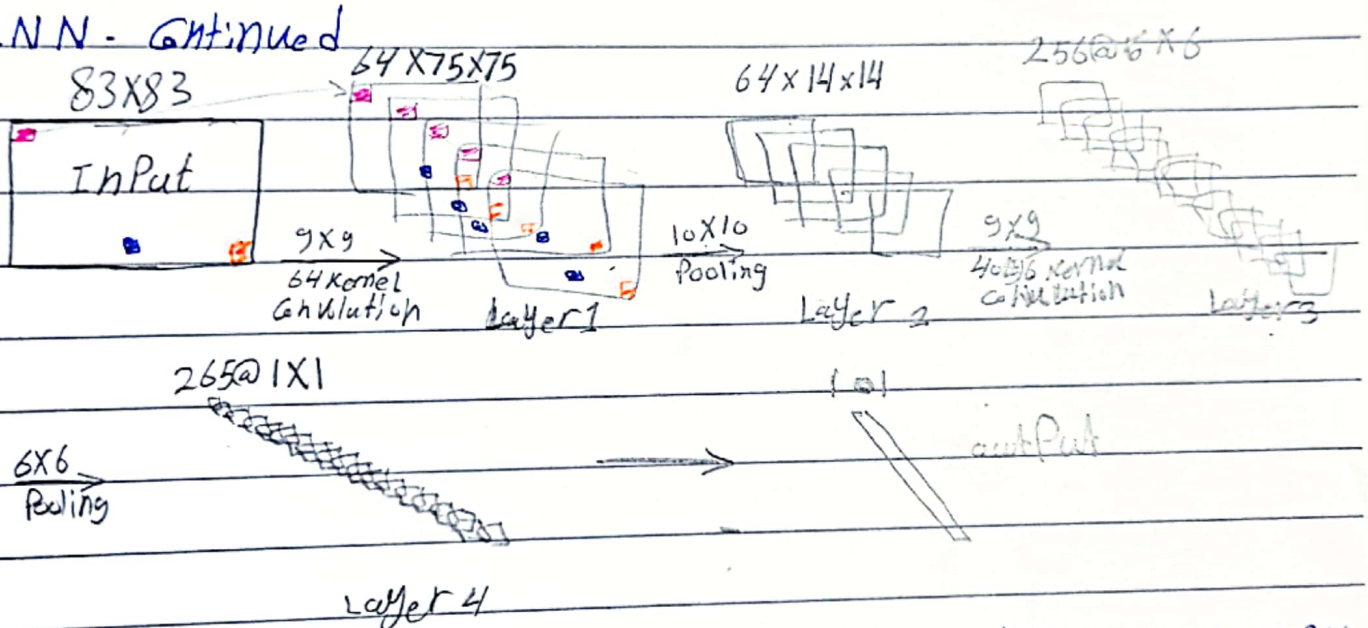
PAGE
DATE

→ we wish to know whether the feature was there, but not exactly where it was.

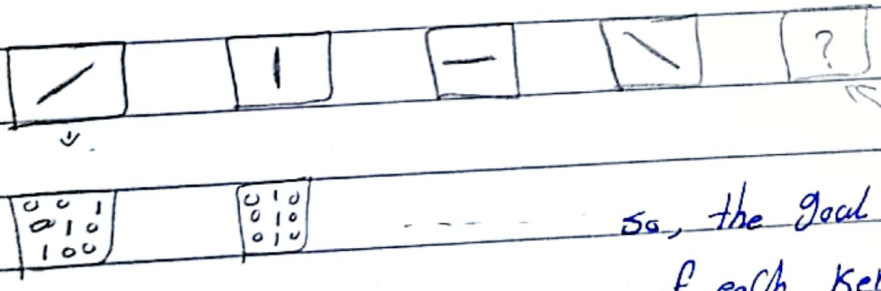
→ The image we will get has 4 times less pixels

→ Each location in the new image corresponds to the maximum of the 25 pixels that it is looking at in the original image

3. CNN - Continued



→ each convolution corresponds to some weights matrix, it corresponds to looking for a particular type of features like edges below:



So, the goal is find the weights of each kernel.