Machine Learning

for Design Engineers

5. Overview of lecture content

Lecture title	Instructor	Content
1. The magic of machine learning	Robert Shorten	Overview of course Introduction to team and assessment Some background material Reading list Background material for laboratories
2. The machine learning portfolio	Robert Shorten	What is Machine learning? Types of machine learning? Basic mathematical background Sprint on Markov chains
3. Key ideas in one lecture	Robert Shorten	The basics explained in an elementary manner.
4. Practicalities of real-life machine learning.	Robert Shorten	Some of the practical issues in dealing with data.
5. ANNs	Robert Shorten	An introduction to artificial neural networks.
6. ANNs	Robert Shorten	Learning algorithms for artificial neural networks.
7. CNNS and RNNS	Pietro Ferraro	An overview of convolutional neural networks
8. SVM	Robert Shorten	An overview of support vector machines and Kernel methods
9. Supervised learning	Robert Shorten	An introduction to K-means clustering
10. Reinforcement learning	P. Ferraro	What are Markov Decision processes?
11. Reinforcement learning	P.Ferraro	What is Q-learning?
13. Spectral graph theory.	Robert Shorten	An introduction to machine learning over graphs.
14. Spectral graph theory	Robert Shorten	An introduction to machine learning over graphs.
15. Spectral graph theory	Robert Shorten	An introduction to machine learning over graphs.
15. Advanced topic Algorithmic fairness	Ms. Quan Zhao	What is fairness in AI?
16. Advanced topic Quantum AI	Martin Mevissen – IBM Research	Quantum machine learning – an introduction

Lecture ①

ML is good at:

- Regression Problems
- Pattern Recognition Problems
- Sequential Decision Making Problems
- Finding Features of Graphs

Content of Design Engineering

- Designing Products (i.e. how virtual objects react in real-time)
- Part of a designed system
- ML Algorithms

Types of Learning

- Supervised Learning (w/ labels)
- Non-Supervised Learning (w/o labels)
- Exploration-Based Learning (finding Optimal Policies)
- Machine Learning on Graphs

Books:

1. The hundred-page machine learning book, A. Burkov
   ↳ surface level
2. Hands-on machine learning with SciKit-learn, Keras and TensorFlow, A. Geron
3. Decision making under uncertainty, M. Kovhendorfer
4. Google's page rank and beyond, A. Langville, C. Meyer
5. Mathematics for Machine Learning, M. Diesenroth *et. al.*

Lecture 2

2 types of Machine Learning

- Supervised (Regression & Classification)
- Unsupervised

Supervised learning

- provided labels

Unsupervised Learning

- learns on its own
- used in clustering applications

Semi-supervised Learning

- combination of labeled and non-labeled

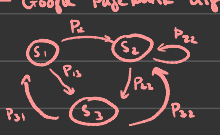
Reinforcement Learning

- "reward" system to optimise based on outcome

Built on 4 areas of mathematics: Optimisation, Probability, Linear Algebra, Markov Chains

- Lagrange Multipliers
- Probability Theory (random variable, mass & density function)
- Markov Chain (essential for graphs and reinforced learning)
  - dynamic system that evolves probabilistically with NO memories

Google PageRank Alg.



Lecture 3

- linear regression
- gradient descent
- building a simple prediction
- building a classifier

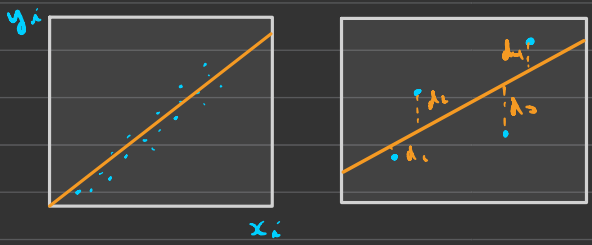
Predict $y(i)$ as $x(i)$ increases:

-fit a line $g(i) = mx(i) + c$

- least squares problem (optimisation)

$$J(m,c) = d_1^2 + d_2^2 + \cdots + d_n^2$$

$$= \sum_{i=1}^{n} (mx_i + c - y(i))^2$$



$$\theta = (U^T U)^{-1} U^T Y$$

derivation: $Y = [y(1), y(2), \ldots, y(m)]^T$, $\theta = [x(1), 1; x(2), 1; \ldots; x(m), 1]^T$

if linear relationship, $Y = U\theta$ $\therefore U^T Y = U^T U \theta$

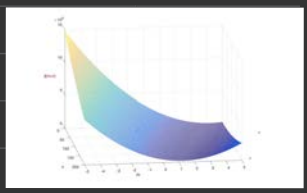
if UU^T is invertible, $(U^T U)^{-1} U^T Y = \theta$

assume: 1) linear relationship

2) squared distance $(d_i)^2$

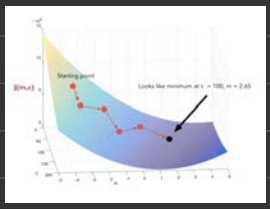
3) $U^T U$ is invertible (multiple unique solutions; dataset is rich enough)

can be used for nonlinear problems (i.e. $y = mx^2 + c$)



$J(m,c)$ as a function of m,c

Gradient Descent



method:

Guess initial value $m_0, c_0 \rightarrow$ calculate gradient $J(m,c)$ at the point:

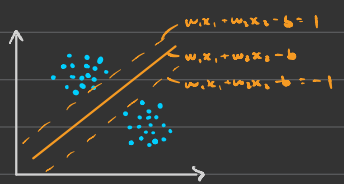
$\frac{\partial J}{\partial m}, \frac{\partial J}{\partial c} \Rightarrow$ update estimates of parameters:

$m_1 = m_0 - \alpha_m \frac{\partial J}{\partial m}$; $c_1 = c_0 - \alpha_c \frac{\partial J}{\partial c}$ where

α_c and α_m are small constants

repeat ↻

Classification



Minimise $w_1^2 + w_2^2$ subject to the constraints:

$w_1x_1 + w_2x_2 - b \geq 1$ if $y^{(i)} = 1$

$w_1x_1 + w_2x_2 - b \leq 1$ if $y^{(i)} = -1$

Lecture 4

- Anatomy of Machine learning
- Coding first learning alg.
- Practical Tips on ML

Anatomy of Machine Learning

- every ML algorithm has the following steps:

1) Cost function
2) Optimisation criterion based on a loss function
3) Algorithm to fit the optimisation

Gradient Descent ~ method of choice to implement optimisation

cost function: $J(m,c) = \frac{1}{n}\left(\sum_{i=1}^{n} (mx^{(i)} + c - y^{(i)})^2\right)$ given n data points

$\frac{\partial J}{\partial m} = \frac{1}{n}\sum_{i=1}^{n} -2x^{(i)}\left(y^{(i)} - (mx^{(i)} + c)\right)$

$\frac{\partial J}{\partial c} = \frac{1}{n}\sum_{i=1}^{n} -2\left(y^{(i)} - (mx^{(i)} + c)\right)$

Practical Comments: Consider 3 things

- getting data ready to learning ~ label the dataset ~ normalisation procedure ~ deal w/ missing data (data imputation) ~ learning (training, test, validation test)
- Selecting features and the model ~ too few parameters → underfitting → high bias ~ too many parameters → overfitting → high variance ~ bias variance dilemma ~ overfitting solution (regularisation) aka adding a term that constrains the parameter. (i.e. Ridge, LASSO)
- choosing the appropriate learning algorithm ~ many factors (i.e. hardware constraints, training speed, type of data & problem, nonlinearities, memory constraints)

Lecture 5

- ANN - artificial neural networks
  - ~ useful to solve regression and classification tasks
  - ~ best known is Multi-layer Perceptron

- Linear Seperation

$f(w_1x_1 + w_2x_2 + w_0) = y$ $\begin{cases} y = 1, & w_1x_1 + w_2x_2 + w_0 > 0 \\ y = -1, & \text{else} \end{cases}$ * labels each class 1 or -1

(issue) Single layer could not separate classes for following cases

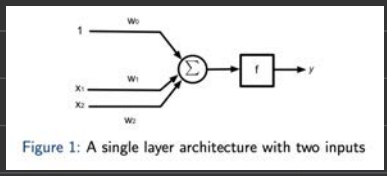


Figure 1: A single layer architecture with two inputs

$f(w_1x_1 + w_2x_2 + w_{10}) = y_1$

$f(w_1x_1 + w_2x_2 + w_{20}) = y_2$

$f(w_1y_1 + w_2y_2 + w_0) = y$

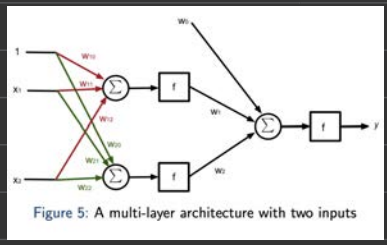
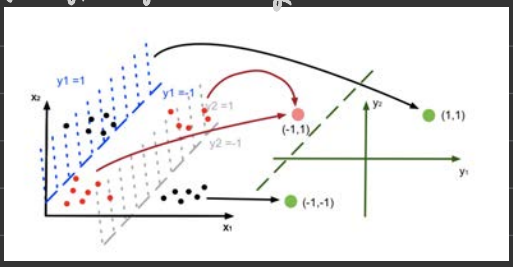


Figure 5: A multi-layer architecture with two inputs



Problems of the difficulty to apply gradient descent to learn the parameters => MLP

MLP - Multi layer perceptron

Multi-layer Perceptron

- when non-differentiable => $f(z) = \frac{1}{1+e^{-z}}$ logarithmic or

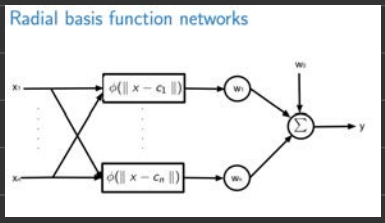
$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ tanh function

=> differential approximation to rectified linear function

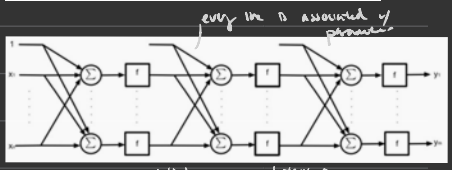
Network is a non-linear system - difficult to understand and certify.
outputs are also nonlinear

Radial Basis function Networks (RBF) - single layer network

$y = \sum_{i=1}^{n} w_i R_i(x) + w_0$ $\quad R_i(x) = \phi(\|x - c_i\|)$ $\quad c_i$: RBF centres



- estimating weights (w) is a linear regression problem
- many approximation methods take the form



Lecture 6

- how to find parameter of network → gradient descent
- 3 ways to calculate gradient: 1. Numerical Differentiation
  2. Symbolic Manipulation
  3. Algorithmic Differentiation

Numerical $\frac{\partial J}{\partial w_i} = \frac{J(w_i + h) - J(w_i)}{h}$ for small h

easy to program not exact, how to choose h?

Symbolic exact method by hand tedious, complicated functions

Automatic Back propagation and partial derivatives

Backpropagation - minimize error iteratively. Chain rule and partial der.

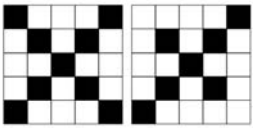
stochastic vs batch

All derivations are in Lecture 6 ✳

Lecture 7

CNN - convolutional neural networks

A human would have no issues saying that these two images look very similar.



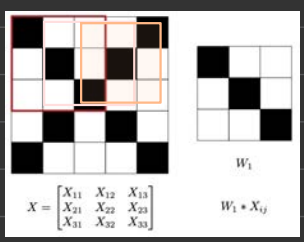
Why not LNN?

 $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [1\ 0\ 0\ 1]$

* loses spacial feature such as diagonals. applies to other formats.

Convolutional Kernels (filters)

$$X_{11} * W_1 = \sum_i \sum_j w_{ij} x_{ij}$$



1	−1	−1	−1	1
−1	1	−1	1	−1
−1	−1	1	−1	−1
−1	1	−1	1	−1
1	−1	−1	−1	1

$*$

1	−1	−1
−1	1	−1
−1	−1	1

$=$

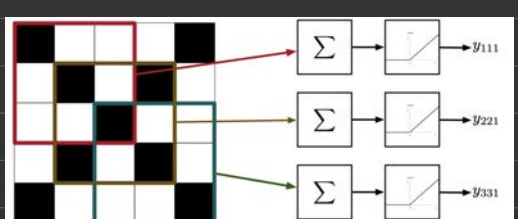
9	−3	1
−3	3	−3
1	−3	9

 higher → it matches

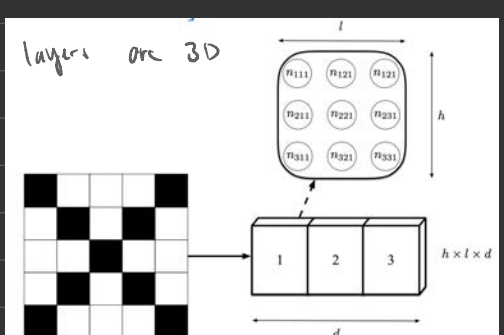
Convolutional Layers

$$h(s) \begin{cases} s, & \text{if } s \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

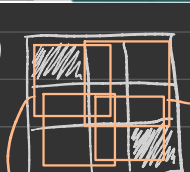
$$y_i = \sigma\left(\sum_j w_{ij} x_j + b_i\right)$$



layers are 3D



i.e.)

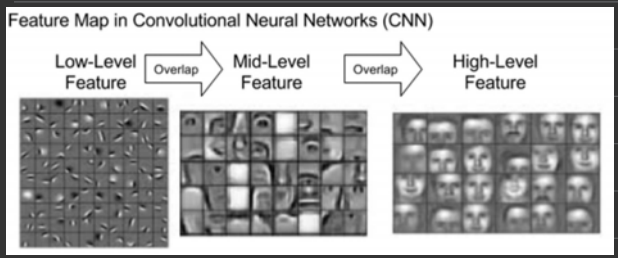


layers

$W_1 =$ $W_2 =$ 

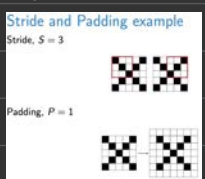
$$x = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

$W_1 * x_{11} - \sigma$ for each W and layer



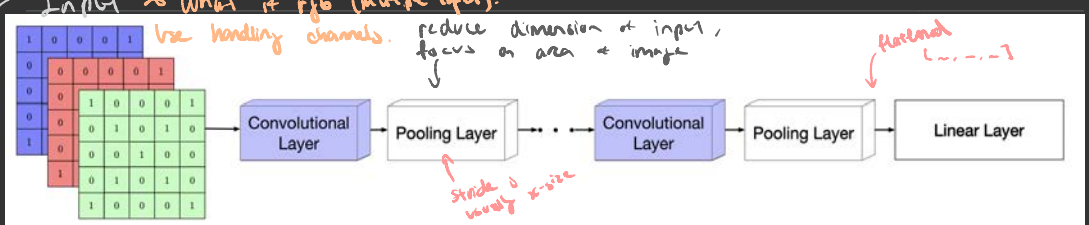
- layer is composed of hidden neurons
- output is fed to next layer
- "feature map" set of neurons that share the same feature.

- Kernel Dimensions ($k_{xi} * k_{yi}$): Dimensions of Matrices W_i
- Stride (s): distance btw current and next operation
- Padding (p): filling zeros of 0 on edges to avoid cropping effects.



CNN Architecture handling channels ⇒ [ ] ⇒ CNN ⇒ ...

Input ⇒ what it gets (multiple layers).



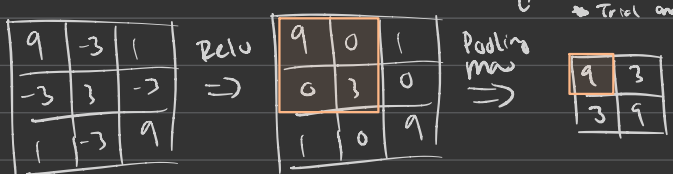
Use handling channels.

reduce dimension of input, focus on area of image

flattened [..., -, -]

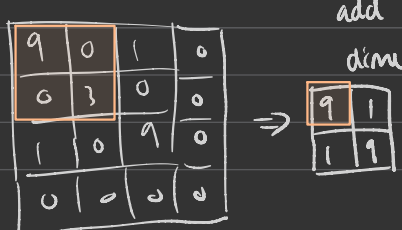
Pooling Layer

There are different types (MaxPool, AvgPool, Stride almost always = x-size)
* Trial and Error



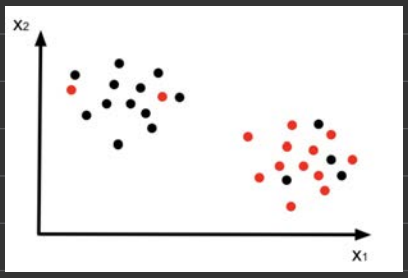
- eliminating pixels that are not too important
- reduce dimension

add two lines of 0s since dimension is odd



Support Vector Machine

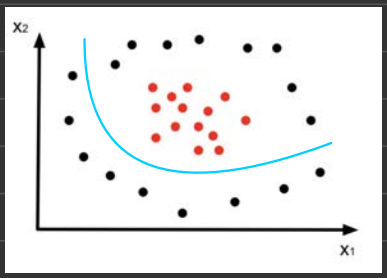
- limitations ~ constraint in every data point (too many constraints)
  ~ noisy data
  ~ non-linearities (not separable by a line)



Noisy Data

how to deal w/ it

- hinge function $max(0, 1 - y(i)(w_1x_1(i) + w_2x_2(i) - b))$
  - non-differentiable → subgradient



Non-Linearities

$b^2 w_1 x_1 + w_2 x_2 + w_3 x_1^2$

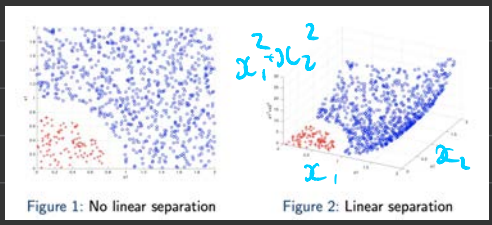


Figure 1: No linear separation

Figure 2: Linear separation

- adding another dimension, existing in 3 dimensions

Minimize over all data points

$$(w_1^2 + w_2^2) + \frac{1}{M}\sum_{i=1}^{M} max(0, 1 - y(i)(w_1x_1(i) + w_2x_2(i) - b))$$

Lecture 9

Unsupervised learning and K-means Clustering

~ dealing w/ unlabelled data

~ problems typically:

clustering - group similar patterns into clusters

anomaly detection - finding outliers

density estimation - estimating PDF for random process generating dataset.

What is k-means algorithm?

Inputs

k, number of clusters

X, training data $\{x(1), x(2), x(3), \dots, x(M)\}$

1) random initialization, choose k points to begin for center ($k < M$)

2) continue updating centre for each cluster assignment.

3) repeat until assignment doesn't change.

$c(i)$ = index of cluster where $x(i)$ is assigned

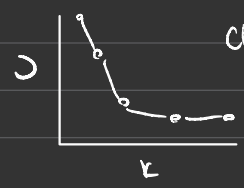
$\mu(j)$ = centre of cluster j

$\mu(c(i))$ = cluster centre for $x(i)$

minimize $J(c(1), \dots, c(M), \mu(1), \dots, \mu(k)) = \frac{1}{M} \sum_{i=1}^{M} ||x(i) - \mu(c(i))||^2$

can converge to a global optimum instead.

elbow method ← when choosing # of clusters



cross-validation: calculate cost J with test data. repeat w/ multiple subsets and K-values.

Markov Chain

Lecture 10

- dynamic system that evolves probabilistically
- NO MEMORY

today → tomorrow

i.e.)  ⇒ P_R = it will rain

- If Probabilities do not change (i.e wrt time) → homogeneous
- If if does → non-homogeneous

(1) Markov Property

$$P(X_{k+1}=S_j, X_k=S_i, \cdots, X_0=S_l) = P(X_{k+1}=S_j, X_k=S_i)$$

Kemeny Constant

$$K_i = \sum_{j=1}^{n} \pi_j m_{ij}$$

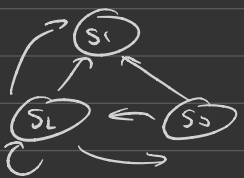
$$K = \sum_{j=2}^{n} \frac{1}{1-\lambda_j}$$

only if $j=1$ bc $\lambda=1$

- Can be used to find bridges by removing nodes
- Identifying different communities: calculate eigenvalues and see how close they are to 1

Sharing Economy

- Markov Chains are useful for modelling macroscopic behaviour of systems (i.e. congestions, how it'd evolve)



state transition diagram

$$P = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.1 & 0 & 0.9 \\ 0.5 & 0.5 & 0 \end{bmatrix}$$

- sums to 1

matrix

$$x^T(1) = x^T(0)P \quad , \quad x^T(2) = x^T(1)P = x^T(0)P^2$$

this is the 'evolution', where k in $x(k)^T$ denotes index

Stochastic : sums to 1 bc of probability

Non-negative : special properties, diagonalizable for eigenvectors

Finding Bridges

- means first passage time (MFPT)
  - how well a state is connected to a graph
- kemeny constant
  - how well connected is the graph

Bias in Machine Learning

3 types (Fairness, Transparency, Privacy/Security)

discriminatory outcomes — hard to know if change after outcome — issues for data

i.e.) Bank loan (using AI to speculate)

~ is it related to gender, race, etc.

~ even w hidden, AI can guess community (i.e. living in China town)

~ rejected by AI system (can it explain why?)

~ COMPAS in US Courts (reoffend likelihood)

~2016, bias against African American

Prediction	Truth	Caucasian	African American
High Risk	& Not Re-Offend	23.5%	44.9%
Low Risk	& Re-Offend	47.7%	28.0%

Figure: The bias in COMPAS. (from Angwin et al., 2016)

~ reasoning:

- under representation (minority subgroups)
- historical bias (i.e. past gender stereotypes)
- model used is to minimize sum of loss (majority group takes larger share, minimize for majority)

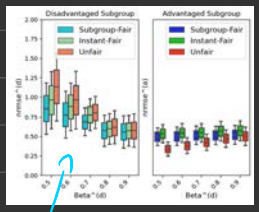
Fairness

- removing sensitive attribute does NOT work
- Statistical Parity (average guesses)
  - Independence (improved) — $P(F=+ \mid A=0) = P(F=+ \mid A=1)$ (binary sensitive attribute)
  - Separation — Same chance regardless of A
  - Sufficiency
- Individual Fairness (treating similar individuals similarly)
- Casual Fairness

(i.e. Counter-factual. changing attributes to see if "accepted")

Addressing Fairness

1) Pre-processing (pre-train) - resampling
2) In-processing (in-train) - minimizing problem
3) Post-processing (after)



suffers from higher loss

Reinforcement learning : optimal policy

Markov decision process : model the decision process
- to maximize reward, etc.

$$V^{*}(s) = E_{\pi}\left[G_t \mid S_t = s\right]$$

$$= R(s, \pi(s)) + \sum_{s' \in S} P(s, s', \pi(s)) \gamma V^{\pi}(s')$$

Dynamic Programming : Bellman to find convergent estimates

LAB NOTES

Lab 1 - Neural Network Basics

- Understand and code a simple neuron
- Understand how a neuron learns
- Understand its limitations

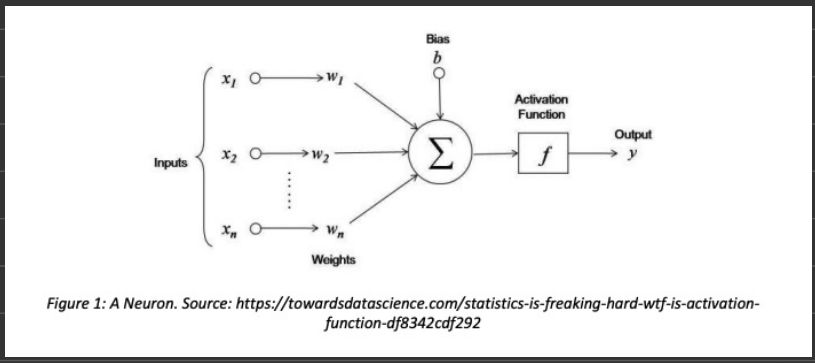


Figure 1: A Neuron. Source: https://towardsdatascience.com/statistics-is-freaking-hard-wtf-is-activation-function-df8342cdf292

$$\sigma_x = \underbrace{\frac{1}{1+e^{-(W^T x+b)}}}_{\text{Sigmoid Function}}$$

$W = (W_1, W_2, \dots, W_n)$ ~weights

$X = (x_1, x_2, \dots, x_n)$ ~inputs

for this lab, $n = 2$ (number of dimensions)

⇒ Set up the weight of the neurons

⇒ if $n=2$, $g(x_1, x_2) = y$ with $y \in \{0, 1\}$

⇒ Associated w/ "Truth Tables" OR & AND ↓

OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

⇓ Teach neuron to learn: update weights w/ gradient descent and error function.

$$E = (\hat{y} - o)^2 \qquad w_i' = w_i + \eta\left(\frac{dE}{dw_i}\right)$$

$$b' = b + \eta\left(\frac{dE}{db}\right)$$

$\hat{y}$: correct output; o: current output; η: learning rate

$E = (\hat{y} - o)^2$

Correct output — current output

$w_i' = w_i + \eta \left(\frac{dE}{dw_i}\right)$

$b' = b + \eta \left(\frac{dE}{db}\right)$

learning rate

$\sigma = \frac{1}{1 + e^{-(w^T x + b)}}$

$\frac{d\sigma(x)}{dx} = \sigma(x)\left(1 - \sigma(x)\right)$

$x = w^T x + b$

$w_i' = w_i + \eta \frac{dE}{dw_i}$

$\frac{dE}{dw} = \frac{dE}{d\sigma} \frac{d\sigma}{dx} \frac{dx}{dw}$

$\frac{dx}{dw} = x$

$\frac{d\sigma}{dx} = \sigma(x)\left(1 - \sigma(x)\right)$

$\therefore \frac{d\sigma}{dw} = \left(\sigma(x)\left(1 - \sigma(x)\right)\right) x$

$\frac{dE}{d\sigma} = 2(\hat{y} - \sigma)$

$\frac{dE}{dw} = \frac{dE}{d\sigma} * \frac{d\sigma}{dw}$

$= \underbrace{2(\hat{y} - \sigma)\left(\sigma(1 - \sigma)\right)}_{dw\ \ ds} x$

$w \pm \eta \boxed{\frac{dE}{dw_i}} \rightarrow$ diff_W.T $\rightarrow$ X.T.dot(dW)

$\therefore x^T \cdot dw = \sum_{i=1}^{n} x\, dw = x_1 dw_1 + x_2 dw_2 \cdots + x_n dw_n$

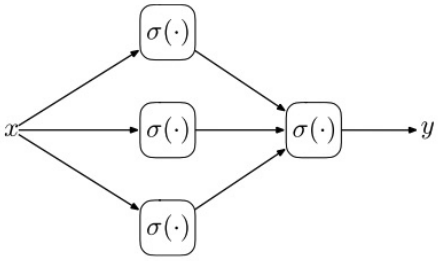
$b_i' = b_i + \eta \frac{dE}{db}$

$\frac{dE}{db} = \frac{dE}{d\sigma} \frac{d\sigma}{dx} \frac{dx}{db}$ $\quad \frac{dx}{db} = 1$

$= \frac{dE}{d\sigma} \frac{d\sigma}{dx} = \underbrace{2(\hat{y} - \sigma)\left(\sigma(1 - \sigma)\right)}_{dB\ \ ds}$

Lab 2 - Feedforward Network & Backpropagation

- Understand Backpropagation
- Write a neural network with ≥ 1 hidden layers
- Solve the XOR
- Understand how to build general classifiers



$$s = w^T x + b$$

$$\sigma(s) = \frac{1}{1+e^{-s}}$$

Use backpropagation due to hidden layer

$E = (z-y)^2$ $\quad w_i' = w_i + \eta \frac{dE}{dw_i}$ $\quad b' = b + \eta \frac{dE}{db}$

$\sigma = y$ $\quad \sigma(s) = \frac{1}{1+e^{-s}}$ $\quad s = w^T x + b$ $\quad \frac{d\sigma}{ds} = \sigma(1-\sigma)$

hidden layer

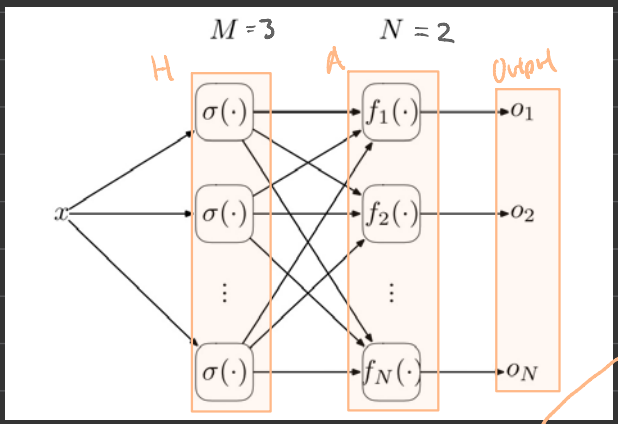
$db = b + \eta \frac{dE}{db}$ $\quad \frac{dE}{db} = \frac{dE}{d\sigma}\frac{d\sigma}{ds}\frac{ds}{db} = (z-y) \cdot \sigma(1-\sigma) \cdot (1)$

$\frac{dE}{dw} = \frac{dE}{d\sigma}\frac{d\sigma}{ds}\frac{ds}{dw} = (z-y) \cdot \sigma(1-\sigma) \cdot (x)$

final layer

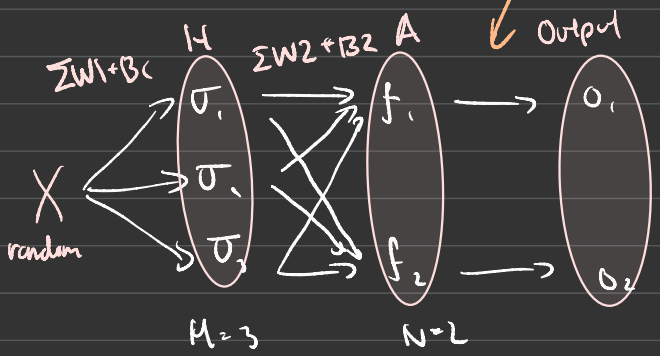
$db = b + \eta \frac{dE}{db} \cdot$

Unlike Lab 1, sigm function is not used to update.



$f_i = (\sum o_i w_i) + B2$

$o_i = f_i(s) = \frac{e^{s_i}}{\sum_{j=1}^{N} e^{s_j}}$



$A = \Sigma WH + B$

$S = w^T x + b$

Cross-Entropy Loss

new error function

$M = -\sum_j z_j \log(o_j)$

$o(1-o)$ (1)

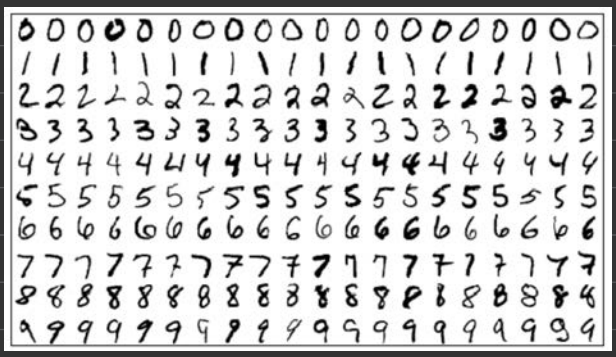
$\frac{dE}{db} = \frac{dE}{do} \frac{do}{ds} \frac{ds}{db} =$

Lab 3 - PyTorch and Deep Networks

- Understand PyTorch to write a Neural Network
- Write a Neural Network of Multiple Hidden Layers.

PyTorch "Autograd" uses tensor to automatically derive and keep track of operations.

dataset : MNIST



teach network to properly classify digits

- Input is 28 × 28 matrix (2-dimensional tensor)
  - corresponds to image pixels.
- Output is integer btw 0–9

Room E
15:45~55
March 10th

Oral Prep

2 Questions
-) Class
-) Lab Stuff