CMPE259

F22, HL

August 23 (Tue)

First Day of the Class

1. Organizational meeting
   "Greensheet"

Repo: github/Hualili/opencv/deep-learning-2022S/

Email: hua.li@sjsu.edu

(650) Hua-Lili Cellphone for Text message Only.

Office Hours: M.W. On Zoom. (See Syllabus for the Zoom Link).

2. Software Tools:

Anaconda — Install it By the end of this week;

TensorFlow, TF 2.0

OpenCV.

3. Prerequisites: CMPE255 & CMPE257

Homework: To upload a copy of your un-official Transcript to show the required courses satisfied.

On CANVAS.

4. Textbook. Deep Learning with Python.

Keras (API) for TF.

Robot Vision Book By Horn (Heavy Theoretical Book, Good Reference for OpenCV Algorithms. Good Theoretical Foundations)

5. Projects { Mandatory Assigned Project
   Team Project (Mandatory)

4-Person Team. Presentation By the End of the Semester.

August 25 (Wed)

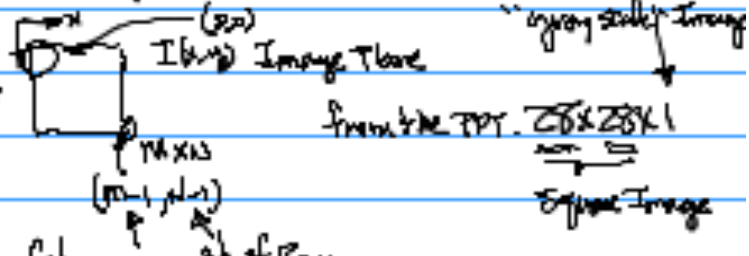
Note: 1- The lecture Note will be posted on to the github.

2- Zoom Recording will be posted on the the github.

Homework: By A week from today. 1. Anaconda Installation; 2. OpenCV Installation. Submission On Canvas. Jpg/Png Image → pdf → Zip file.

Example: (github: 2022F-183_)

MNIST Architecture for Handwritten Digits Recognition.

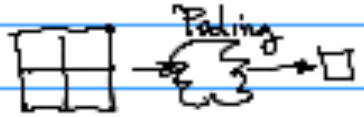


(gray scale image → 1 channel → 8 bit → [0, 255])

First Layer of the MNIST Architecture

n1 channel/plane of the 1st Convolution Layer C1

Next, pooling → Reduction of Resolution → 252 becomes



M1

From the Architecture diagram:

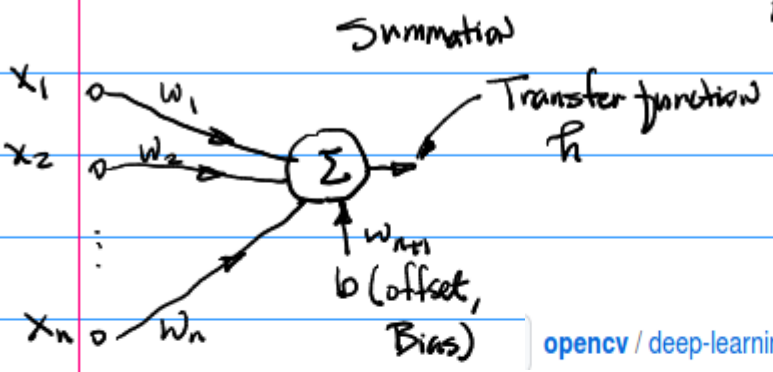
C1 M1 → C2 P2 → Flatten → (FF) FFNN

To generalize the quick inspection of the the CNNs, we have to investigate the Behavior of Each Single Neuron as the Basic Building Block.

CMPE258
Aug. 25, 22

August 30, Tue

2/



Ref.1
https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022F-103b-NN-Intro-Python-v5-2022-8-25.pdf

opencv / deep-learning-2022s / 2022S-103a-notation-neuro-loss-function-2022-2-8.pdf

Input/Excitation in Vector Form: $X=(x_1, x_2, \cdots, x_n)$ ...1)

Weights, links each excitation to the Neuron

Ref2. Code
https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022S-110b-%232019S-31-6mnist-numerals-ch02.py

$W=(w_1, w_2, \cdots, w_n)$ ... (2)

$x_1w_1 + x_2w_2 + \cdots + x_iw_i + \cdots + x_nw_n + bw_{n+1} = h$ ... (3)

$\sum_{i=1}^{n} x_iw_i + bw_{n+1} = h(x_iw_i)$ or simply $h(Xw)$

$h(Xw; b)$, $h(X, w)$

Example:

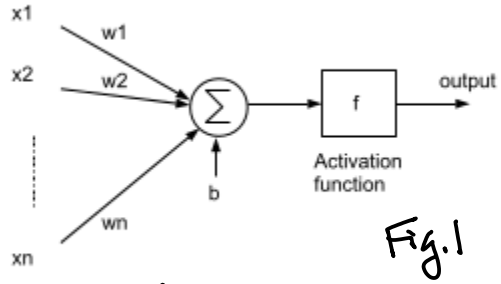
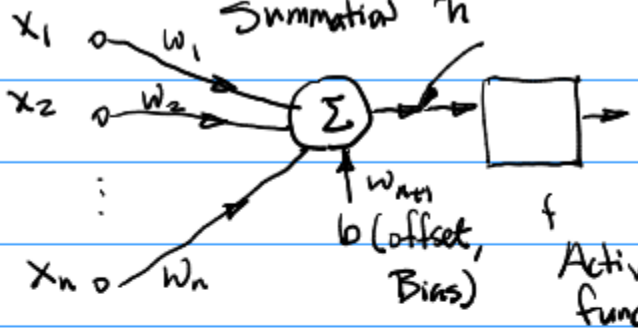


Fig.1

$(x_1, x_2, \cdots, x_n)$ Feature Vector with Dimension N.



$$h = \sum_{i=1}^{N} w_i x_i = W \cdot X + b \quad (11)$$

Transfer function $h(\cdot)$

$w_{n+1} b = b'$

$f(h(Xw)) = f\left(\sum_{i=1}^{n} x_iw_i + bw_{n+1}\right)$ ... (4)

$h(Xw) = \sum_{i=1}^{n} x_iw_i + bw_{n+1}$

$$y = f(\sum_{i=1}^{N} w_i x_i = W \cdot X + b). \quad (17)$$

Activation function. Its output is the Response of the Neuron.

Examples of Different Activation functions include RELU, Piecewise Linear.

Note: Be Able to Build A Single Neuron per a technical Specification, Such as n=11, Activation $f(\cdot)$, Draw a Block

CMPE258

Aug 30.

Consider the output of the Neuron y from Eqn(17).

Output of A Single Neuron.

For multiple Neuron Output, see Fig.2

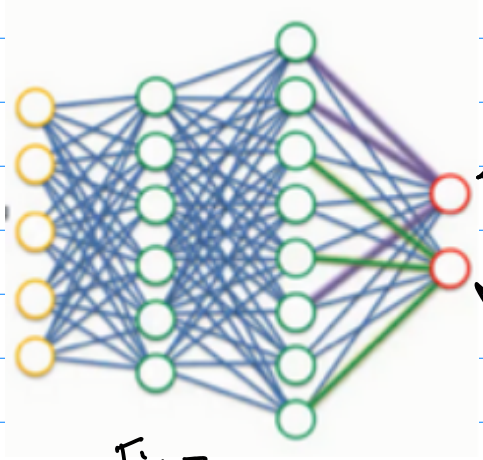


Fig.2

y_i ... (1)

Subscript: $i=1,2$

No. of Output at the Output Layer.

y_i, $i=1,2,\cdots,M$.

In practical Application,

y_i^j ... (2)

Superscript

$j=1,2,\cdots,P$ No. of Experiments performed, Training Performed.

Look at the Concept & Definition of Loss function. 3/

Mathematically To Compare a Neural Network Output (Single Neuron Output)

function f. function g

Comparision of the Similarity or difference between f and g.

$f-g$

f/g

Difference Between Two Functions.

Take this Approach to define Loss function,

$y - \hat{y}$ ... (3)

Ground Truth. Output (prediction) from the Neuron

Outputs

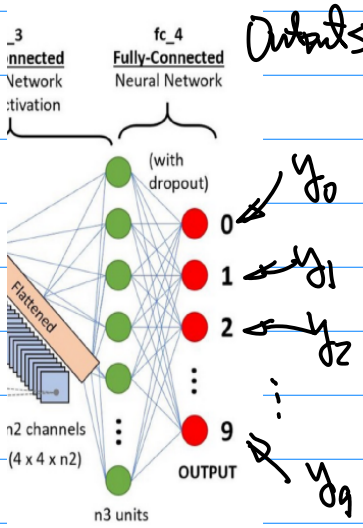


Fig3.

$y_i - \hat{y}_i$ ...(4-a)

To measure All the output for Each Training/Experiment

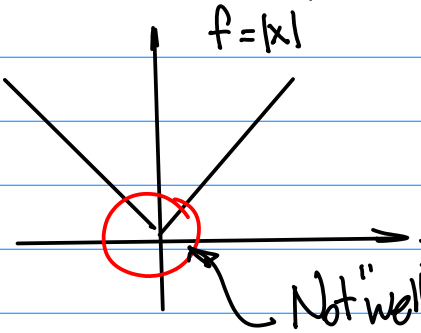
$$\sum_{i=0}^{q} (y_i - \hat{y}_i) \quad \text{...(4-b)}$$

Expand this to Experiment/Training up to "P" Times

$$\sum_{j=1}^{P} \left[\sum_{i=0}^{q} (y_i^j - \hat{y}_i^j) \right] \quad \text{...(4-c)}$$

Note: Eqn(4-c) may Lead to positive & Negative Terms Cancellation.

Fix: Absolute Value ? → Squared Instead,



Not "well-behaved" !

∴ "Well-Behaved" System (Function) → derivative/Partial Derivative up to order "K".

$$L = \sum_{j=1}^{P} \left[\sum_{i=0}^{q} (y_i^j - \hat{y}_i^j)^2 \right] \quad \text{...(4-d)}$$

→ J, or Φ

$$L_{total} = \frac{1}{2} \sum_{j=1}^{P} (\tilde{y}^j - y^j)^2. \quad (23)$$

Ground Truth

For a Single Neuron @ the Output Layer

Training Based "Steepest Gradient Descent" = SGD.

Example: Given A function $f(x) = x^2$, Find its Derivative

$$\frac{df}{dx} = 2 \cdot x$$

To get rid of Coefficient from the derivative,

Let's define $f(x) \triangleq \frac{1}{2} x^2$

then, $\frac{d}{dx} f(x) = \frac{1}{2} \cdot 2 \cdot x = x$

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i^j)^2 \quad (24)$$

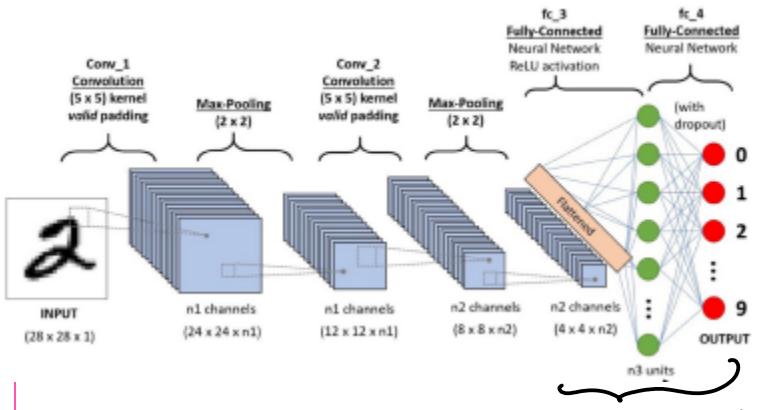
Sept 1st (Th).

Ref:

2022S-103a-notation-neuro-loss-function-2022-2-8.pdf

Example: Background on "Learning" of ANN.

2022F-103b-NN-Intro-Python-v5-2022-8-25.pdf



Fully Connected Feedforward NN

To Train A NN, we take eqn(23), e.g. Loss function (error function),

$L(\cdot) \rightarrow L(W_{ij})$

↑ Independant Variables

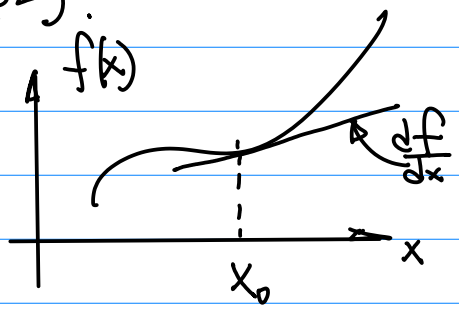
Minimize the Loss $L(\cdot)$, which is the process of Training, which leads to Learning for the NN.

↓

Since the Loss function in Eqn(23) is formulated with a ground Truth y, this defines a Supervised Learning.

Math. Formula: Prediction of Function's Behavior, we like to know given the current function Value (Loss function) how this function is going to change at next moment, increase? stay the same? Or Decrease?

Derivative of A given function is a good indicator to give us the description of the function behavior Next Step Ahead (Very small tiny Step Δ).



$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \quad \ldots (1)$$

↓ Intuition

$f(x+\Delta x) - f(x) \rightarrow > 0$ (derivate if the derivative is greater than 0)

$\Delta x \simeq 1$ unit

then $f(x+\Delta x) > f(x)$, the next Step function $f(x)$ is increased;

if the derivative $\frac{df}{dx} < 0$, then

$f(x+\Delta x) < f(x)$

if the derivative $\frac{df}{dx} = 0$, then

$f(x+\Delta x) = f(x)$

Consider Two Dimensional Case as an Example for n-dimensional Case.

CMPE258

Sept. 1st.

$f(x_1, x_2)$ → $\frac{\partial f(x_1,x_2)}{\partial x_1}$, $\frac{\partial f(x_1,x_2)}{\partial x_2}$ → Single Neuron

In Case of Training They are "Weights" w_1, w_2.

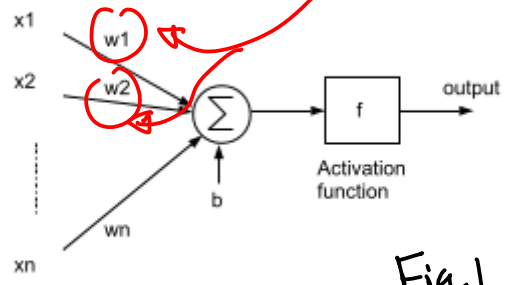


Fig.1

$\frac{\partial f}{\partial x_1} \rightarrow \frac{\partial f}{\partial w_1}$ In the Context of Training.

Conclusion: Use the Partial Derivate With respect to the weight W_i as an indicator to measure if the Loss function is got reduced or not.

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i^j)^2 \quad (24)$$

Example: Consider A Technique which allows training to be more effect, e.g., the to minize training and prediction error (Loss) function

Steps for Development of this technique:

Multi-dimensional Derivative

↓

gradient

↓

the Steepest gradient

↓

the Steepest Descent gradient

↓

The Core technique to train NN. SGD

Ref: from the github

2022S-105c-#20-2021S-4gradient-descent-v2-final-2021-2-8.pdf

Homework (Opt), Due A week from Today. Sept. 8th, Thursday. ON CANVAS.

1° Installation of TensorFlow. Version 2.0 or higher.

2° Screen Capture to show the installation is successful

Note: All Different Development Tools/Enviroments including google co-lab, jupyter Note Book etc. Are OK., However for the Deployment purpose, Projects Homework Submission must be in Python Stand-Alone form.

Sept. 6 (Th)

Homework: Due 1 week from Today Sept 13.

Cmpe258
Sept. 6.

1. OpenCV Installation, Python.
2. Use Smart phone to Capture 5~10 Seconds Video Clips. .avi, .mp4 (mpeg-4).
3. Sample Code, github. See CANVAS for the Detailed links & Requirements.
4. Submission to CANVAS.
   (1) Python Code;
   (2) Original & Processed image Side by Side with your Name + SID.
   (3) Create One pdf file to Cover the Source Code, And Screen Captured Images.
5. Naming Convention

   HW_CV_First_LastName_Cmpe258_SID.zip

https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022S-104d-%232-pdisplay-2019-1-30.py

Higher Dimensional Function

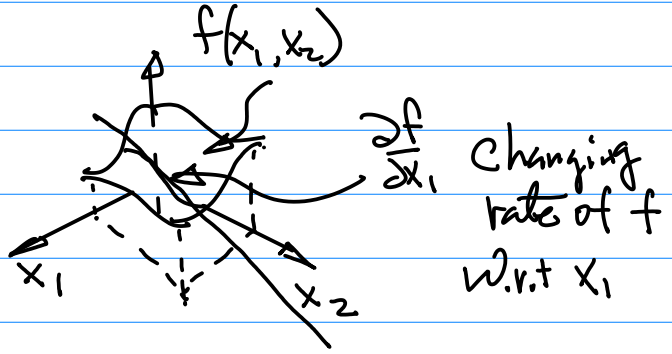
$$f(x_1, x_2, \cdots, x_n) \quad \cdots (2)$$

Weights, $w_1, w_2, \cdots, w_N$

Partial Derivatives:

$\frac{\partial f}{\partial x_1}$ w.r.t x_1, $\frac{\partial f}{\partial x_2}$ for x_2, ...

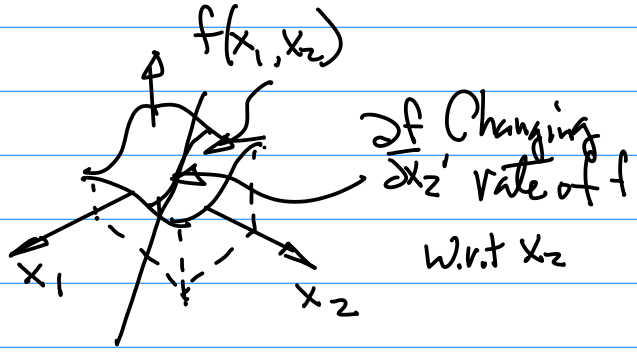
$\frac{\partial f}{\partial x_n}$ w.r.t x_n.



Example: Gradient Definition

Ref:

https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022S-105c-%2320-2021S-4gradient-descent-v2-final-2021-2-8.pdf



Loss function

Derivative, e.g. given $f(x)$, then

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \quad \cdots (1)$$

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i^j)^2 \qquad (24)$$

Consider the minization of function f (Loss Function) w.r.t. All possible weights.
Therefore, Put all the partial Derivatives together to form A vector, e.g., gradient.

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \frac{\partial f}{\partial x_i} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad \cdots (2a)$$

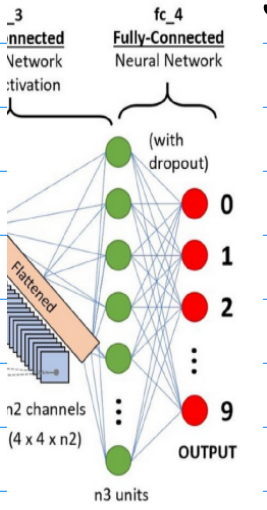
for n=2,

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \quad \cdots (2b)$$

for n=3

$$\nabla f(x_1, x_2, x_3) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} \quad \cdots (2c)$$

d. On the Right hand side of Eqn(5):
(x_1^k, x_2^k) Dimension n=2, (x_1, x_2)
Time Index "k", Superscript
Output of the NN with its weights at Step k (Time) is x_1^k, x_2^k



e. On the left (x_1^{k+1}, x_2^{k+2}), at the step k+1, to Reduce the Loss function, so update the new step By following

$$-\nabla f = -\eta \cdot \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} \cdots (3)$$

IV. Gredient Steepest Descent for Minimization

Conclusion:

$$(x_1^{k+1}, x_2^{k+1}) = (x_1^k, x_2^k) + [-\eta(\nabla f)^t] \quad (5)$$

a. Loss function f

b. n=2

e.g. $f(x_1, x_2, \cdots, x_n) \rightarrow f(x_1, x_2)$

c. Gradient $\nabla f(x_1, x_2)$, or ∇f

Background:
Given a function f(x), How do you Approximate this function By using Basic Building Blocks (B^3)?

$f(x)$ = Constant Term + A Linear Term + A Quadratic Term + A Cubic Term + ...(4)

CmpE258
Sept 6, 22

Taylor Expansion:

$$f(x) = f(x_0) + \frac{df}{dx}\cdot(x-x_0) + \frac{d^2f}{dx^2}(x-x_0)^2 + \cdots + R_n(x) \quad \cdots (4)$$

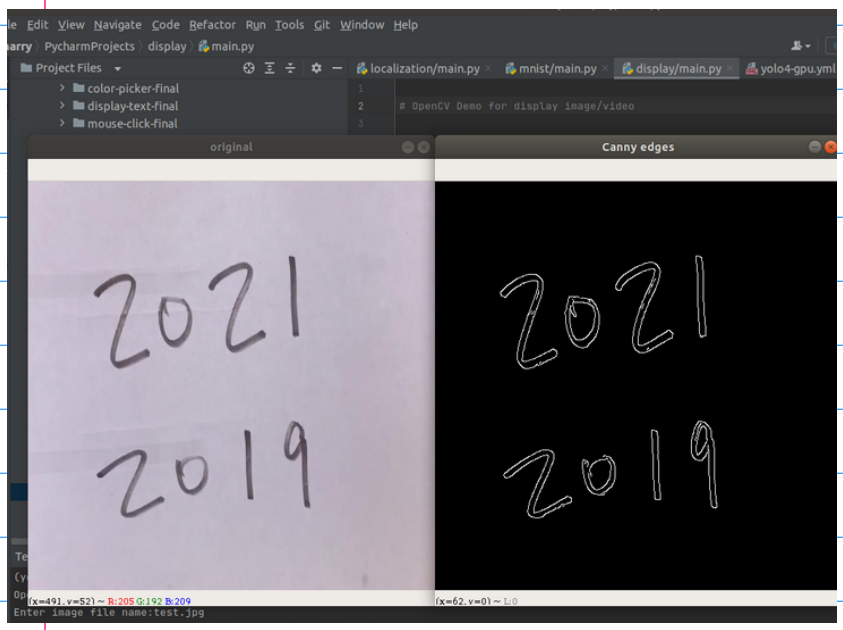
$$f(x_1,x_2) = f(x_1,x_2)\Big|_{\substack{x_1=x_{10}\\x_2=x_{20}}} + \frac{\partial f}{\partial x_1}(x_1-x_{10}) + \frac{\partial f}{\partial x_2}(x_2-x_{20})$$

($f(x_1,x_2)\Big|_{x_1=x_{10},\,x_2=x_{20}}$: Constant; $\frac{\partial f}{\partial x_2}(x_2-x_{20})$: Linear Term)

$$+ \frac{\partial^2 f}{\partial x_1^2}(x_1-x_{10})^2 + \frac{\partial^2 f}{\partial x_2^2}(x_2-x_{20})^2 + \cdots + R_n(x_1,x_2) \quad \cdots (5)$$

($\frac{\partial^2 f}{\partial x_1^2}(x_1-x_{10})^2 + \frac{\partial^2 f}{\partial x_2^2}(x_2-x_{20})^2$: 2nd Order; $R_n(x_1,x_2)$: Higher Order Terms)

Note: The screen Capture for your homework refereme.



The goal is to verify the formula for updating the weights of A given NN. (see Handout 1, Eqn(5)).

Step1. Taylor Expansion → Step2. Simplify the Taylor Expansion By just using up to the Linear terms → Step3. Re-arrange the Taylor Expansion in the form of Trainning formula (In Eqn(5), in Handout 1) → Step4. Analyze the re-arranged formula, to Reach the Observation which lead to the Conclusion, e.g., using gradient descent, we can Redue the Loss

Sept. 8 (Thu)

Note: 1° Check the CANVAS for Both Homeworks.

Example: From 1D case in Eqn(4), we can expand the Taylor Expansion to higher Dimension n. to Capture multiple excitations, multiple Weights W_i, $i=1,2,\cdots,n$.

Consider n=2

function through each step of the trainning.

From the Handout, we have as step 1 & 2:

$$f(x_1, x_2) \simeq f(a,b) + \frac{\partial f}{\partial x_1}(x_1 - a) + \frac{\partial f}{\partial x_2}(x_2 - b) \quad (6)$$

$$f(x_1,x_2) - f(a,b) \simeq f_{x_1}(x_1-a) + f_{x_2}(x_2-b)$$

Comparison of A "loss" function, write

$$\Delta x_1 = x_1 - a$$
$$\Delta x_2 = x_2 - b$$

And $\nabla f = \begin{pmatrix} f_{x_1} \\ f_{x_2} \end{pmatrix}$

Hence, we have

$$f(x_1,x_2) - f(a,b) = (\Delta x_1, \Delta x_2)\nabla f$$

Let $\Delta x_1 = -f_{x_1}$, $\Delta x_2 = -f_{x_2}$

Therefore,

$$f(x_1,x_2) - f(a,b) = (-f_{x_1}, -f_{x_2}) \begin{pmatrix} f_{x_1} \\ f_{x_2} \end{pmatrix}$$

$$= -(f_{x_1}^2 + f_{x_2}^2) \le 0$$

$$\Downarrow$$

Hence, $f(x_1,x_2) - f(a,b) \le 0$

OR, $f(x_1,x_2) \le f(a,b)$

Loss function Updated at the next step by Eqn(5) in Handout 1.

Loss function at the Current Step.

Note: The Requirement for this discussion on Notations, And Formulation, especially, the Eqn(5) in Handout 1 is required To Be Able to use these tools to Analyze the problem, And to Perform Verification (e.g. design).

Example: 2022F-103b-NN-Intro-Python-v5-2022-8-25.pdf

To Be Continued.

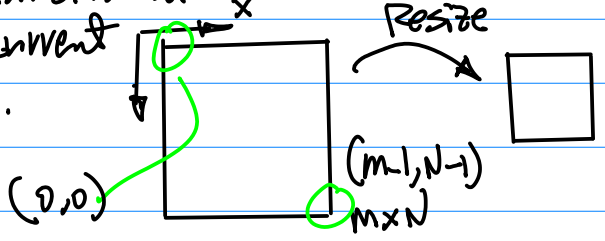
OpenCV Homework. Sample code

opencv/2022S-104d-#2-pdisplay-2

```
10  window_name = 'Display Image'
11
12  imageName = sys.argv[1] #get file name from command line
13
14  src = cv2.imread(imageName, cv2.IMREAD_COLOR)
15
16  if src is None:
17      print ('Error opening image!')
18      print ('Usage: pdisplay.py image_name\n')
19
20  ind = 0
21
22  while True:
23      cv2.imshow(window_name, src)
24
25      c = cv2.waitKey(500)
26      if c == 27:  #ESC
```

a. b. c.

Note: OpenCV Resize Function.



```python
import numpy as np
#import argparse
import cv2

img = input('Enter image file name:')

image = cv2.imread(img, cv2.IMREAD_COLOR)

if image is None:
    print('Error opening image!')
    print('Usage: pdisplay.py image_name\n')

image = cv2.resize(image, (512, 512))  a.

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,100,200)  b.
```

a. Conversion to Gray-scale image;
b. Canny Edge Detection.

Sept.13 (Tue)

Homework: (opt) Due to A week from today. Capture the screen to the T.F. Installation is successful.

Example: 2022F-103b-NN-Intro-Python-v5-2022-8-25.pdf

a. Feature Vectors $V=(v_1, v_2)$

$\Rightarrow X=(x_1, x_2)$

$\Rightarrow X_i=(x_{i1}, x_{i2})$

Example: Collecting data for training

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

X_1
X_2
$\vdots$

Signs	Mij	Mpq	Sign
V1	133	65	Stop
V2	160	72	Right
V3	152	70	Right
V4	120	60	Stop

b. Ground Truth y_i Corresponds to X_i

c. Preprocessing of the dataset

Name	x_1 Weight (minus 135)	x_2 Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

"Shift" Down the data w.r.t "0"

You can find the mean for x_i, $i=1,2,$ then "Shift" the Data Accordingly By Subtracting it from the mean.

Make the ground truth value as Numerical value for easy handling.

Now, Consider Design for a Simple feedforward NN.

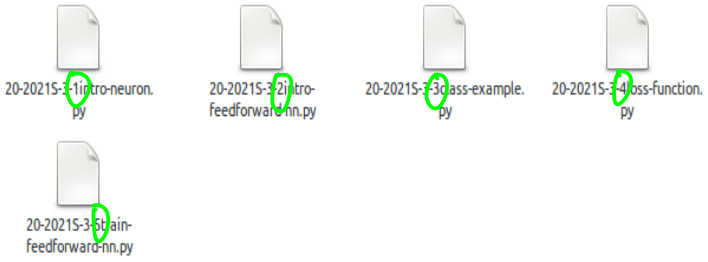
Step1. Match the Dimension of the Feature Vector X_i to the Input Neurons

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i{}^j)^2 \qquad (24)$$

Also, Determine the Output Layer. Inspection of the Nature of this Prediction, Hence, Choose One Single Neuron Output

Hidden Layer(s) → Affects the Behavior of Decision-making function.

Now, Let's use Python Code to Build this Simple FFNN



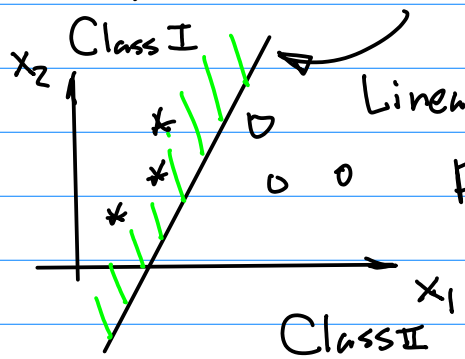


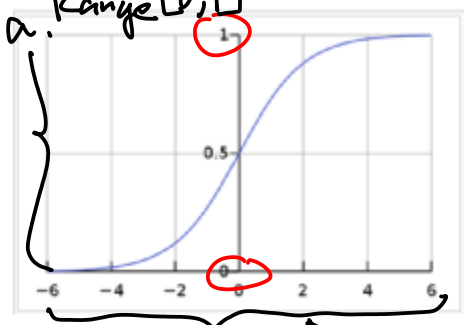
Fig.1.

Define An Activation function.

$y = f(h(w_i x_i; b)) = f(IW)$

```
def sigmoid(x):
  # Sigmoid activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))
```

a. Range [0,1]



https://en.wikipedia.org/wiki/Sigmoid_function

b. Excitation Range.

$f(h(IW))$

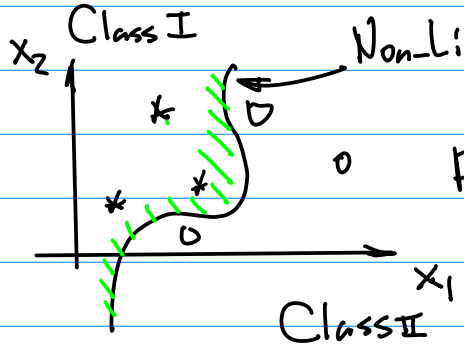
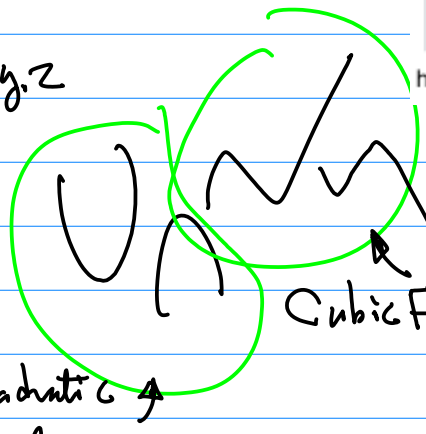


Fig.2

No. of Hidden Layers →



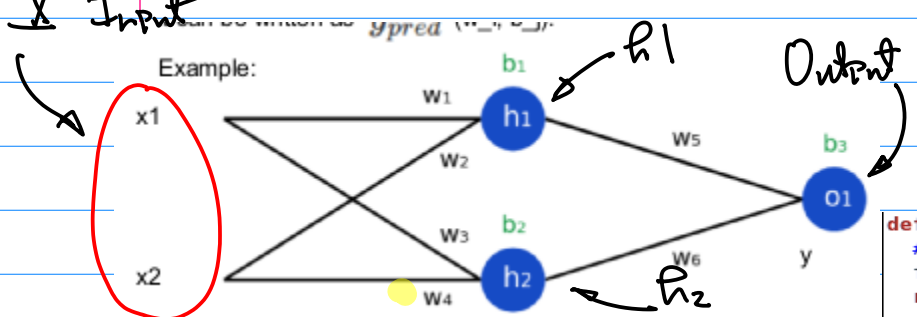
For the Simplicity in this example, Let's choose just 1 Hidden Layer.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Note: A Sigmoid Function. (1)

A sigmoid function is a

The Derivatives of the Activation Function. Done By Hand Calculation of the Derivative.




```
def deriv_sigmoid(x):
  # Derivative of sigmoid: f'(x) = f(x) * (1 - f(x))
  fx = sigmoid(x)
  return fx * (1 - fx)
```

CMPE258
Sept 13, 22

$$\frac{d}{dx}\,Sig(x) = \frac{d}{dx}\,\frac{1}{1+e^{-x}}$$

Define the Loss function

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i^j)^2$$

```
def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()
```

Mean Square Error

Ground Truth

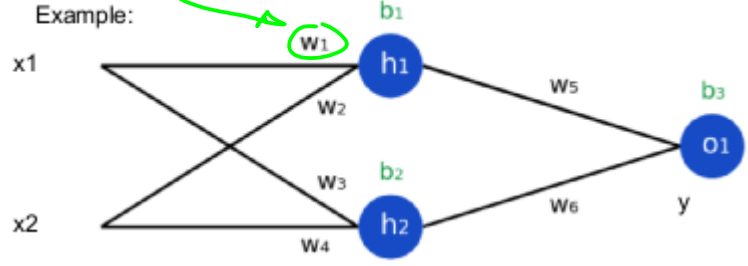
NN output ~ y_i

Note: The Derivative Code in this example Allows Training Implement By Gradient Descent.

The Weight initialization.

```
def __init__(self):
  # Weights
  self.w1 = np.random.normal()
  self.w2 = np.random.normal()
  self.w3 = np.random.normal()
  self.w4 = np.random.normal()
  self.w5 = np.random.normal()
  self.w6 = np.random.normal()
```

Example:




```
  # Biases
  self.b1 = np.random.normal()
  self.b2 = np.random.normal()
  self.b3 = np.random.normal()

def feedforward(self, x):
  # x is a numpy array with 2 elements.
  h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
  h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
  o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
  return o1
```

(24)

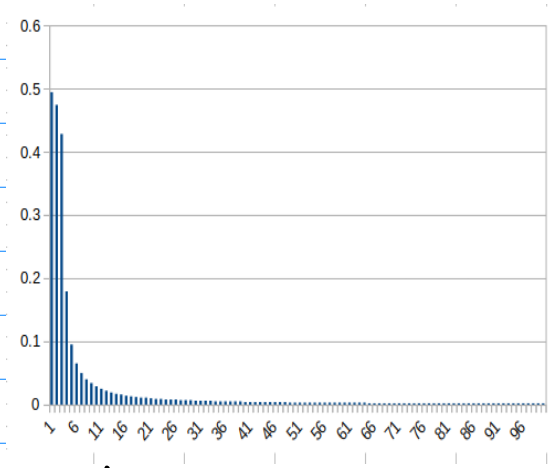
Define the F.F. Neural Network.

Output:

$$y = f(wx)\Big|_{f=\frac{1}{1+e^{-x}}}$$

where x is from the hidden Neurons, h_1 & h_2

https://github.com/hualili/opencv/blob/master/deep-learning-2022s/2022S-103c-%23nn_sample_2022.py



Sept 15 (Thu)

Today's Topics:

1° Preliminary Sample Code (Python)

2° Preprocessing Technique (Computer Vision / OpenCV) for Deep Convolutional NN. Handwritten Digits Recognition

Cmpe258 Sept.15,22

Ref: From Python Sample Code on the github

```
74  def train(self, data, all_y_trues):
```

a. Training will take the input from here

```
157 #-----------------------------------
158 # Define dataset and all_y_trues
159 #-----------------------------------
160 data = np.array([
161   [1, 2.5],     # person A
162   [1, 3],       # person A
163   [2.1, 3.4],   # Person A
164   [2.1, 1], # person B
165   [3.3, 1], # person B
166   [3, 2.3], # person B
167 # HL 2020-9-7 Part E
168 # for the testing of adaptive learning
```

b. Initialization. η

```
80      learn_rate = 0.1
81      epochs = 1000 # number of times to loop through
```

The Number of Trainings is defined with this as a upper Bound if the Loss function during the training drops below the pre-set threshold then the training Process will terminate

The Data Point (e.g. $X(x_1,x_2)$) is substituted into the code to evaluate Transfer function $h(w,x)$, then evaluate the Activation function $f(h(w,x)) = \frac{1}{1+e^{-x}}\Big|_{x=h(w,x)}$

```
85      # --- Do a feedforward (we'll need these values later)
86      sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
87      h1 = sigmoid(sum_h1)
```

The process propagates through each neuron & each layer till reaches the Output

```
92      sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
93      o1 = sigmoid(sum_o1)
94      y_pred = o1
```

Loss is being computed

```
98      d_L_d_ypred = -2 * (y_true - y_pred)
```

then, Compute partial derivatives to form gradient

```
96      # --- Calculate partial derivatives.
97      # --- Naming: d_L_d_w1 represents "partial L / partial w1"
98      d_L_d_ypred = -2 * (y_true - y_pred)
99
```

then, update the weight(s) Based on gradient descent

```
136
137     self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
138     self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
139     self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1
```

Compute total Loss function

```
151     # --- Calculate total loss at the end of each epoch
152     if epoch % 10 == 0:
153       y_preds = np.apply_along_axis(self.feedforward, 1, data)
154       loss = mse_loss(all_y_trues, y_preds)
155       print("Epoch %d loss: %.3f" % (epoch, loss))
```

$$\frac{\partial L}{\partial w_{i,k}} = \frac{\partial}{\partial w_{i,k}} \frac{1}{2} \sum_{j=1}^{P} \sum_{i=1}^{M} (\tilde{y}_i^j - y_i^j)^2 \qquad (24)$$

Note: the ground truth is given in the code

```
178 all_y_trues = np.array([
179   1, # person A
180   1, # person A
181   1, # person A
182   0, # person B
183   0, # person B
184   0, # person B
185 ])
```

Annotation

CmpE258
Sept. 15, 22

Example: Pre-processing Techniques for Handwritten Digits Recognition.
Objective: To Be Able to Localize the ROIs (Region of Interests).

a. ROI: Bounding Boxes

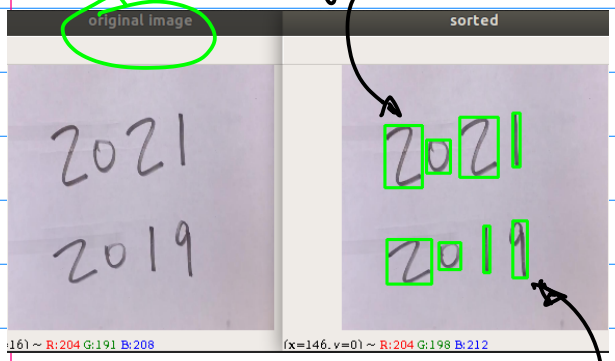


Fig.1.

b. localization: localized on Each hand written digits

Ref: 1. PP.18 (Starting from)

20225-101-cmpe258-2022-03-15-Note.pdf

2. 20225-108-Intro-Binary.pdf PPT. with math. Formulation.

In OpenCV, Image Planes (Color) is organized as B, G, R.

$I_r(x,y) \in [0,255]$, $I_g(x,y) \in [0,255]$, $I_b(x,y) \in [0,255]$ ... (2)

8 bit
$2^8 = 256$

Color Image Example in Fig.1. → Gray Scale Image.

$I(x,y)$ → $I_{gray}(x,y)$

$$I_{gray}(x,y) = \frac{1}{3}\left(I_r(x,y) + I_g(x,y) + I_b(x,y)\right) \quad \ldots (3)$$

Binary Image.

$$B(x,y) = \begin{cases} 255 & \text{if } I_{gray}(x,y) \geq T \\ 0 & \text{o/w} \end{cases} \quad \ldots (4)$$

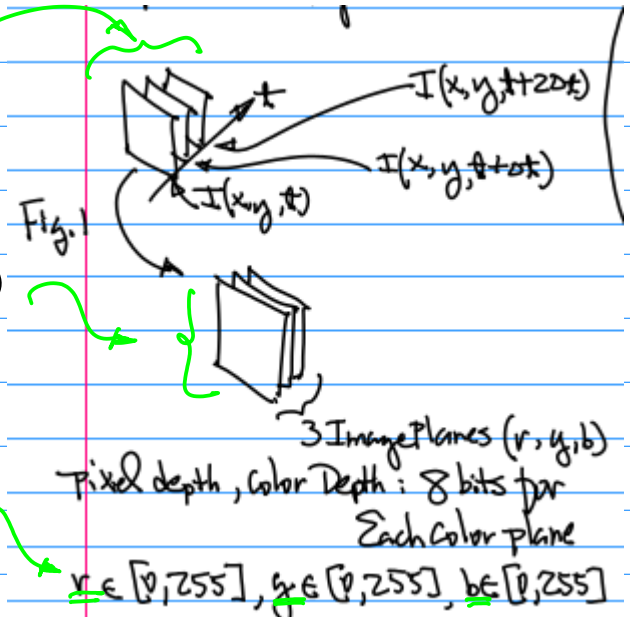
Background on Color Images.

a. Video Clip. (A Sequence of Images) From Ref. 1.

b. A Single frame of an image $I(x,y,t) \rightarrow I(x,y)$ @ time t ... (1)

c. Each Frame of A color image $I(x,y)$ Consists of 3 channels.

Or. Primitive Color planes, red, green, Blue



Pixel depth, Color Depth: 8 bits per Each Color plane

$r \in [0,255]$, $g \in [0,255]$, $b \in [0,255]$

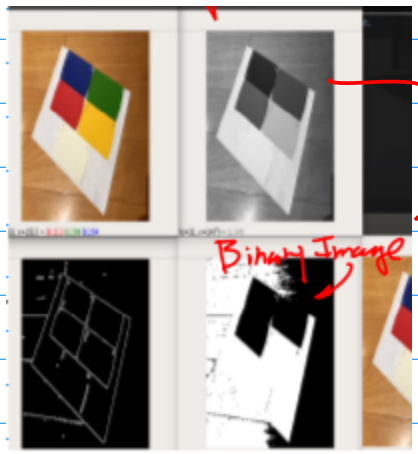
CmpE258
Sept. 15, 22

Example of A Binary Image
(This Binarized image is Canny Edge Detection) NOT Commonly used Binarization Image.

From Eqn (4)

Example:

2022S-108c-example-binary.pdf



By Eqn (4)

2022S-108-Intro-Binary.pdf

Operators for Preprocessing (1~6)

Pattern Recognition For Binary Images

The tool box for pattern recognition for binary images

1. Size
2. Moments
   - $\bar{x}$
   - $\bar{y}$
   - $\bar{x^2}$
   - $\bar{y^2}$ etc.
3. Perimeter
4. Orientation
5. Compositions of the above
   Perimeter and moments: vector
6. Invariant operators
   - size invariant
   - orientation invariant
   - illumination invariant

Biologically inspired techniques

- Rule 1. Proximity
- Rule 2. Similarity
- Rule 3. Closure
- Rule 4. Good continuation
- Rule 5. Symmetry
- Rule 6. Simplicity

Note: 'Proximity' usage for clean up binary image and remove noise, as well as growing boundary points per 'good continuation' rule to form a better edge map.

Note: Similarity defines a interesting question, how to describe one object is similar, or somewhat similar to others, neural network and fuzzy logic may help.

Harry Li, PhD, SJSU