M22AIE233, ISHNOOR SINGH | ML - ASSIGNMENT

Prob1

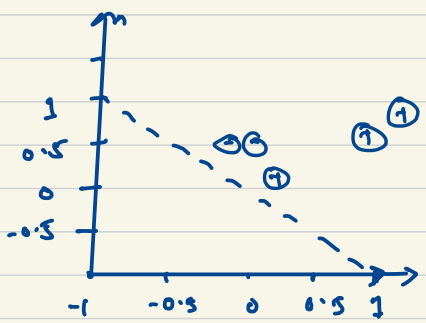
Perceptron

x_1	x_2	Class
1	1	1
-1	-1	-1
0	0.5	-1
0.1	0.5	-1
0.2	0.2	1
0.9	0.5	1

(i) In 3 Steps the learning algorithm will converge, 2 step of w updates & last step to check the correct predicm.

(ii)

Initial Decision boundry



Step 1: $w = [1, 1]$

Pred = $W^T x$

Class +1 for pred ≥ 0

Class -1 for pred < 0

Step 1

$w = [1, 1]$

$p = \text{pred} = w^T x$

Class +1 if $p \geq 0$

Class -1 if $p < 0$

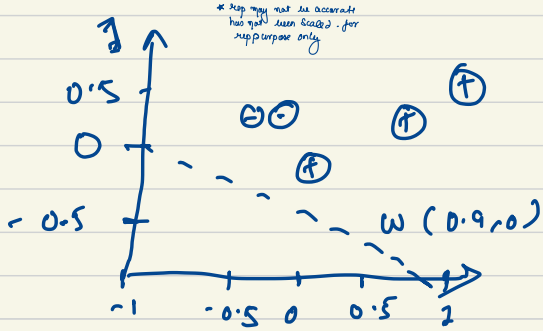
x_1	x_2	Class	pred ($w^T x$)
1	1	+1	2 (Correct)
0.2	0.2	+1	0.4 (Correct)
0.9	0.5	+1	1.4 (Correct)
-1	-1	-1	-2 (Correct)
0.0	0.5	-1	0.5 (wrong)
0.1	0.5	-1	0.55 (wrong)

Correct Class pred = 4

wrong " " = 2

$\rightarrow w = [1, 1] - [0, 0.5] = [1, 0.5]$

$\rightarrow w = [1, 0.5] - [0.1, 0.5] = [0.9, 0]$



C = Correct

WR = wrong

Step 2

$w = (0.9, 0)$

$p = \text{pred} = w^T x$

Class → +1 $p \geq 0$

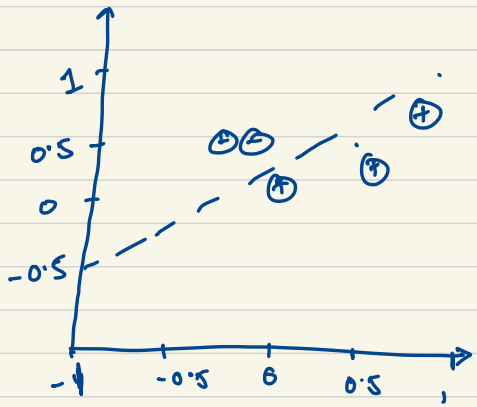
-1 $p < 0$

$\Rightarrow$ Correct: 5

wrong: 1

x_1	x_2	Class	Pred ($w^T x$) (C/WR)
1	1	1	0.9 (C)
0.2	0.2	1	0.18 (C)
0.9	0.5	1	0.81 (C)
-1	-1	-1	-0.9 (C)
0.0	0.5	-1	0.0 (W)
0.1	0.5	-1	0.15 (C)

$\rightarrow w = [0.9, 0] - [0, 0.5] = [0.9, 0.5]$



* rep may not be accurate has not been scaled. for rep purpose only

Step 3

(C/wr)

$W^T x$

$W = [0.9, 0.5]$

Correct prediction = 6

p = pred = $W^T x$

Class → +1 $p \geq 0$

→ -1 $p < 0$

x_1	x_2	Class	pred	
1	1	1	0.4	C
0.2	0.2	1	0.08	C
0.9	0.5	1	0.56	C
-1	-1	-1	-0.4	C
0.0	0.5	-1	-0.25	C
0.1	0.5	-	-0.13	C

Problem 2:

GitHub Repo Link:
Code provided in Repo

OBSERVATIONS:

The Gurmukhi digit Classifier is a simple Neural Network trained to recognize handwritten Gurmukhi digits (0-9). These are my observations.

(i) Data Loading & preprocessing:

- Data has been loaded from two folders 'train' and 'val' with 'train' folder having subfolders for each class.
- The train is used for training, 'val' is used to test/val & tune the folders
- There is a seperate function to test images placed in a seperate folder/given a path.
- The images are read in grey scale flattened and normalized.

(ii) Model Architecture:

- The classifier is designed in pytorch, it is a feed forward neural network with 4 fully connected layers
- The dropout rate is 0.2, as this prevents overfitting by out a propotion of neurons during training. This parameter has been tuned from different values.

- The hidden layers have size of [128, 64, 32] i.e the first H layer has 128 neurons. 2nd H layer has 64 and 3rd H layer has 32 neurons
- Since there are 10 classes, the output size of 10 is mentioned in last layer. Each image is flattened to a size of 32*32 which becomes input size.

(3) Training & Validation:

- The model is trained with ADAM optimizer with a learning rate of 3.e-3 and L2 regularisation (weight decay of) 9.e-4. The L2 regularization is used to prevent overfitting.
- Loss function used is cross-entropy loss, which is suitable for multiclass classification.
- Model is trained for 100 epochs and training & Validation loss is mentioned at each epoch.

(4) Final performance & further improvement.

- The training & Validation loss suggested that model did not overfit and generalizes well.
- Despite with less training examples, the model utilizes L2 regularization & dropout to prevent overfitting.
- However for even better results we can utilize a CNN along with a better GPU to generate a better model.

Problem 3

Git hub repo link:

(1) Model summary

(A) Chart image classification using CNN

Architecture:

- 2 convolution layers each followed by batch Normalization ReLU activation & max pooling.
  - first Conv layer: 4 input, 4 output channels, Kernel size: 3 Stride: 1, padding: 1
  - Second Conv layer: 4 input, 4 output, kernel: 3, stride: 1 padding: 1
- CNN layer are followed by a linear layer for classificam
  - Linear Layer: 4*32*32, output features (6)
- Maxpooling is used twice right after each ReLU activation function.

Training:

- Trained for 25 epochs, ADAM optimizer, learning rate: 0.001
- Loss func^n: Cross Entropy loss

task3: finetuning pretrained 'Alex Net'.

- Imported necessary libraries including Pytorch, torchvision, numpy & pandas.
- Defined a Custom Dataset Class 'Chart Dataset' to handle the chart data
- Loading, preprocessing / transforming the data using torchvision transform. Compose().
- Creating data sets and data loader by instiating the Custom dataset Class.
- Load pre trained 'Alex Net' & modify the first layer to accept 4 input channels and replace the last layer to match the no of classes in dataset. (5 classes)
- prepare model for training: Move to GPU if there. loss: Cross Entropy optimizer: SGD, lr: 0.001, momentum = 0.9
- Trained for 25 epochs & validated the model.

Problem 4

(a) Stochastic Gradient Descent (SGD) and Mini Gradient Descent are optimization algorithms used to minimize loss function. They differ in how they process the training data to update models params.

- SGD (Stochastic Grad Descent):
  - => Models parameters are updated after processing each individual training example.
  - => This means Gradient is computed and the weights are updated for every single data point
  - => Can be expensive but can make model escape local min.

- Mini-Batch Gradient Descent:
  - => This can be considered as in between Batch Gradient Descent (where gradient computation & weights are updated after the entire dataset) and SGD.
  - => In MBGD, the models parameters are updated after processing a small subset of training data called a mini batch.
  - => This has computationally efficiency of Batch GD and escapes local minimas like SGD.

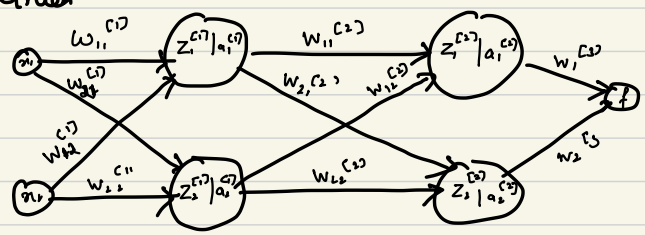
(2)

Given $f = w_1^{[2]} a_1^{[2]} + w_2^{[2]} a_2^{[2]}$

Compute

$$\frac{\partial f}{\partial z_1^{[2]}}, \frac{\partial f}{\partial z^{[2]}}, \frac{\partial f}{\partial z^{[1]}}, \frac{\partial f}{\partial w_{11}}$$

Given



$$A^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \end{bmatrix}$$

$$A^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[2]}) \\ \sigma(z_2^{[2]}) \end{bmatrix}$$

(1) $$\frac{\partial z}{\partial z_1^{[2]}} = \frac{\partial z}{a_1^{[2]}} \cdot \frac{\partial a_1^{[2]}}{\partial z_1^{[2]}}$$

$$f : w_1^{[2]} a_1^{[2]} + w_2^{[2]} a_2^{[2]}$$

$$a_1 : \sigma(z_2^{[2]})$$

$$: \sigma(w_{21}^{[2]} \cdot a_1^{[1]} + w_{22}^{[2]} \cdot a_2^{[1]})$$

↳ Activation function

$$\Rightarrow \frac{\partial z}{\partial a_1^{(2)}} = w_1^{(3)}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = \sigma'(z_1^{(2)}) = \sigma(z_1^{(2)}) * (1 - \sigma(z_1^{(2)}))$$

Hence: $\frac{\partial z}{\partial z_1^{(2)}} = w_1^{(2)} * \sigma(z_1^{(2)}) * (1 - \sigma(z_1^{(2)}))$

(3) $\frac{\partial L}{\partial z^{(1)}} = \left(\frac{\partial L}{\partial z^{(1)}} \cdot \frac{\partial L}{\partial z_2^{(1)}} \right)$

$$= \frac{\partial z}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}$$

$$+ \frac{\partial L}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} \cdot \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}$$

$$= w_1^{(3)} \cdot (w_1^{(3)} \cdot \sigma(z_1^{(2)}) * (1 - \sigma(z_1^{(1)}) \cdot w_{11}^{(2)}$$

$$\cdot \sigma(z_1^{(1)}) * (1 - \sigma(z_1^{(1)}))$$

$$= w_1^{(2)} \cdot \Big(w_1^{(2)} \cdot \sigma(z_1^{(2)}) * (1 - \sigma(z_1^{(2)}))$$

$$(w_{11}^{(2)} \cdot \sigma(z_1^{(1)}) * (1 - \sigma(z_1^{(1)}))$$

$$w_{12}^{(2)} \cdot \sigma(z_2^{(1)}) * (1 - \sigma(z_2^{(1)}))$$

(2) $\frac{\partial f}{\partial z^{(2)}} = \left(\frac{\partial f}{\partial z_1^{(2)}}, \frac{\partial f}{\partial z_2^{(2)}} \right)$

$= \frac{\partial f}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} + \frac{\partial f}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_1^{(2)}}$

$= w_1^{(3)} \cdot \sigma(z_1^{(2)}) \cdot (1 - z_1^{(2)})$
$+ w_2^{(3)} \cdot \sigma(z_2^{(2)})(1 - z_2^{(2)})$

(4) $\frac{\partial f}{\partial w_{11}} = \frac{\partial f}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}}$

$= w_1^{(3)} \cdot \left(\sigma(z_1^{(2)})(1 - \sigma(z_1^{(2)})) \cdot w_{11}^{(2)} \right.$
$\left(\sigma(z_1^{(1)})(1 - \sigma(z_1^{(1)})) x_1 \right.$

$\left[z_1^{(1)} = w_{11}^{(1)} \cdot x_1 + w_{12}^{(1)} x_2 \right.$

Problems

(1) Tuning hyperparams on test dataset pose a risk of overfitting the model which would fail to generalize, as we would be essentially optimizing the values on a specific dataset. Thus as a result the model may capture noise/patterns unique to test dataset.

=> The usual best practice is to have Train, Validate & test dataset. Where train set is used to train, Validate is used to tune & test is the unseen dataset used to test. This would allow to obtain an unbaised model performance.

(2)

The two strategies to reduce overfitting are:

(1) Regularization:
- Reg. is added to reduce overfitting by adding a penalty term to loss function.
- The most common reg methods are L_1 & L_2 regularisation
- L_1 adds the absolute value of the weights while L_2 adds the squared value of the weights
- These penalty discourages model to learn overly complex representations

(2) Dropout:
- This technique is specific to NNs, During training dropout randomly "drops" or deactivates a certain percentage of neurons
- This prevents the model from relying too heavily on

on any single neuron or a feature

(3)

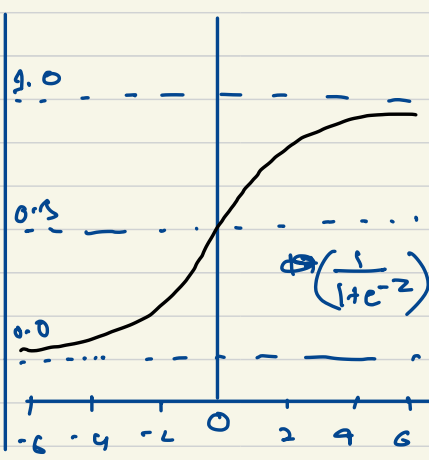
Input Layer Size:
- Depends on the no of features in the dataset used for the specific problem
- for ex, if we are dealing with image of size 28×28 pixels, the input layer would have 784 (28×28) nodes
- for ex for tabular data with 10 features input layer would have 10 nodes

Output Layer Size:
- Depends on the type of problem being solved & no of classes or target variable.
- for binary classificatn, we can have a single o/p neuron with sigmoid activatn
- for multiclass, o/p layer should have same neuron as classes with softmax
- for reg, o/p with one neuron with single activatn funcn.

(4)

The Sigmoid activation funcn: $\frac{1}{1+e^{-z}}$

- takes any real valued input and maps it to a value b/w 0 & 1
- It is S shaped smooth curve, with f(n) approaching 0 as $n \to -\infty$ and 1 as $n \to +\infty$ and centered around 0.5 when $n = 0$. It is differentiable
- Used in binary classificatn.



(5)

- The learning rate is a hyper param that determines the step size during optimization process. It controls how much of a change is applied in updating the parameters of the model
- A smaller learning rate leads to smaller updates while a larger learning rate leads to larger updates
- Choosing the right lr is appropriate for proper convergence of the model (decreasing cost w.r.t parameters)
- If lr is too small, it will take along time to converge can possibly be stuck in local min
- On the other hand a large lr may overshoot or miss the minima / optimal condition and may never converge.
- A learning rate given in ques of S.S is very very large and hence will cause instability in training process the network may not converge. It is recommended to start with smaller values glr (0.001, 0.0001 etc)