Name : Yash Manharbhai Patel

Student ID : A20451170

Course : CS512 – Computer Vision

Semester : Fall20

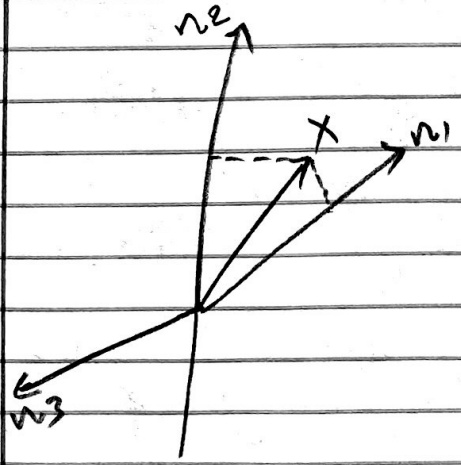
CS-512 - Assignment-3

Ans-1 a) Linear classifiers

$$\hat{y} = f(x; \Theta) = \Theta^T x + \Theta_0$$

-> rows of Θ^T (columns of Θ) are templates with K rows of Θ^T we have K templates. (one template per class)
$\Theta^T x$ measures how well x matches each of the K templates
(dot product of x with rows of Θ^T)
High similarity to a template of a particular class indicates high membership in this class.

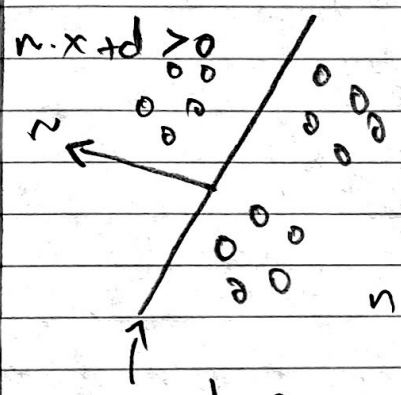
$$\hat{y} = \Theta^T x + \Theta_0 = \begin{bmatrix} n_1^T \\ \vdots \\ n_K^T \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} d_0 \\ \vdots \\ d_K \end{bmatrix}$$



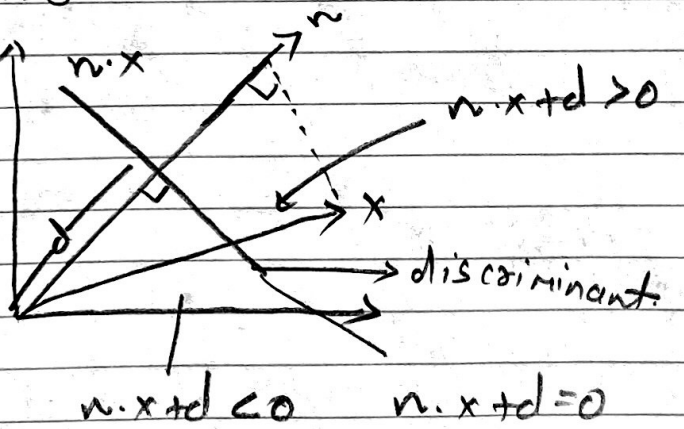
$$= \begin{bmatrix} n_1 \cdot x \\ \vdots \\ n_2 \cdot x \end{bmatrix} + \begin{bmatrix} d_0 \\ \vdots \\ d_K \end{bmatrix}$$

-> rows of Θ^T are parameters of K linear discriminant function.
Each linear discriminant separates one class from all others.

The value of a linear discriminant is positive for examples belonging to the class and negative otherwise.
It measures distance from the decision boundary.



n = normal
d ~ negative distance from origin

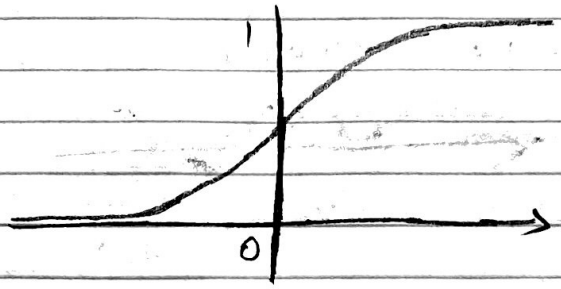


b) given similarity score (or decision boundary distance s_i) from a linear classifier (higher better), complete:

$$\hat{y}_j^{(i)} = P(y=j \mid x^{(i)})$$

In a 2-class classification case:

$$\hat{y}_j^{(i)} \equiv P(y=j \mid x^{(i)}) = \text{Sigmoid}(s_j^{(i)})$$



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

In a K-class classification.

$$\hat{y}_j^{(i)} \equiv P(y=j \mid x^{(i)}) = \frac{\exp(s_j^{(i)})}{\sum_{i=1}^{K} \exp(s_j^{(i)})} \leftarrow \text{Softmax}$$

c)

L_1 loss : $L_1(\theta) = \sum_{j=1}^{K} |\hat{y}_j^{(i)} - y_j^{(i)}|$

L_2 loss : $L_2(\theta) = \sum_{j=1}^{K} |\hat{y}_j^{(i)} - y_j^{(i)}|^2$

Huber loss :

$$\rho_\sigma(d) = \begin{cases} \frac{1}{2}d^2 & \text{if } |d| \le \sigma \\ \sigma(d - \frac{1}{2}\sigma) & \text{otherwise} \end{cases}$$

Cap loss of outliers (robust regression)

quadratic for small d, linear for large d.

$$L_i(\theta) = \sum_{j=1}^{K} \delta_{\sigma}(\hat{y}_j^{(i)} - y_j^{(i)})$$

Cross-entropy loss:

Convert similarity scores to probability:

$$\hat{y}_j^{(i)} = P(y=j \mid x^{(i)}) \qquad j \in [1, K]$$

Likelihood: $L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{K} (P(y=j \mid x^{(i)}))^{y_j^{(i)}}$

Log-likelihood:

$$l(\theta) = -\log L(\theta)$$

$$= -\sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log(P(y=j \mid x^{(i)}))$$

$$= -\sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log(\hat{y}_j^{(i)})$$

Sample loss: $L_i(\theta) = -\sum_{j=1}^{K} y_j^{(i)} \log(\hat{y}_j^{(i)})$

d) Occam's razor: simpler explanations are better.

A simpler solution is when the weights θ are lower (e.g. when $\theta_{ij} < 0$ we remove coefficient)

smaller coefficients → more stable solution that will generalize better

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_i(f(x_i, w), y_i) + \lambda R(\theta)$$

λ → weight of regularization (hyper parameter)

$R(\theta) = \sum_i \sum_j \theta_{ij}^2$ → regularization term

L_1 - Regularization

$R(\theta) = \sum_i \sum_j |\theta_{ij}|$

L_2 - Regularization minimizes weights while spreading them (e.g. $0.5^2 + 0.5^2 < 1^2 + 0^2$)

L_1 - Regularization does not have this property and may concentrate weights (e.g. $|0.5| + |0.5| = |1| + |0|$)

e) To minimize the loss (objective) solve:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ L(\theta)$$

$$\nabla L(\theta) = 0 \Rightarrow \theta^*$$

If $\nabla L(\theta)$ is not Linear it is hard to find explicit solution.

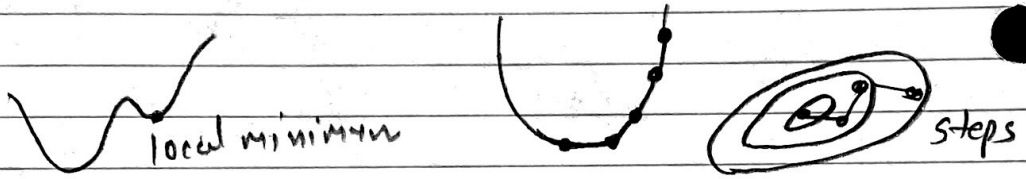
numerical iterative solution.
(e.g. gradient descent)

start with initial guess θ_0

repeat : $\theta^{i+1} = \theta^i - \eta \nabla L(\theta^i)$

η = learning rate (hyper parameter)

stop when : $(L(\theta^{i+1}) - L(\theta^i)) < \varepsilon$

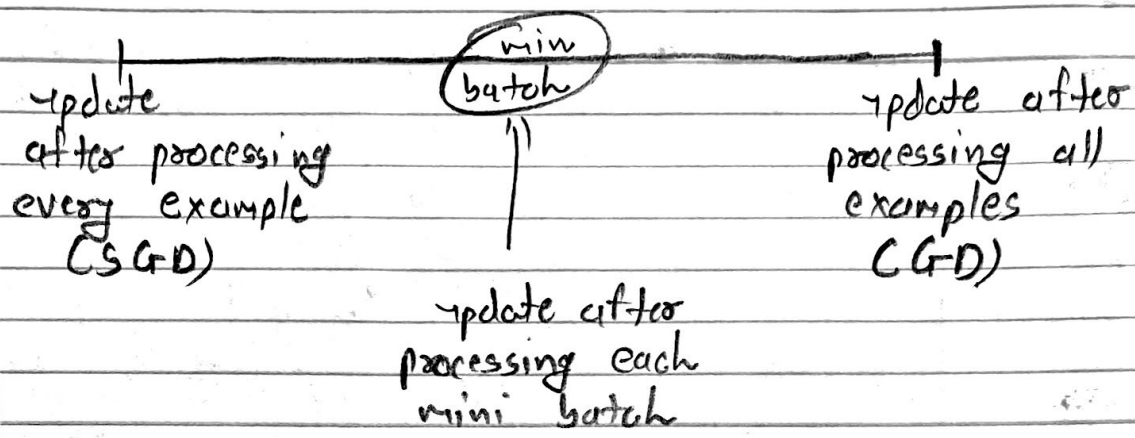


f) In gradient descent:
we find gradient of all the examples (Summation) in current iteration and then update weights based on this gradient of all examples

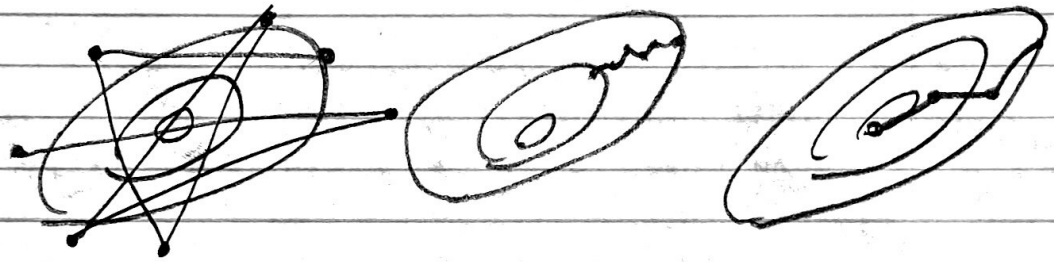
In stochastic gradient descent:

$\theta^{(i+1)} = \theta^i + \eta \nabla L_j(\theta^i)$ ← gradient based on example j

we update weights after every example.



g) learning rate has to be selected correctly



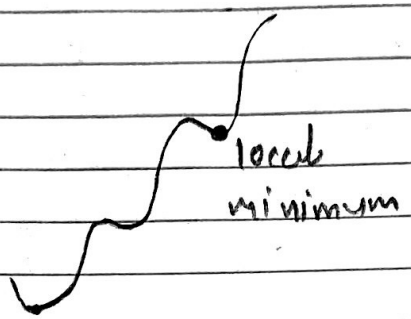
η to large η to small good η

learning rate need to be fixed
make learning smaller as iterations progress

- step decay: every κ iteration
  $\eta \leftarrow \eta/2$
- exponential decay: $\eta = \eta \cdot e^{-k/t}$

  $k \rightarrow$ decay rate
  $t \rightarrow$ iteration index
- fractional decay: $\eta = \eta_0 / (1 + k \cdot t)$

h) In gradient descent, momentum helps to reach to the global minimum.



without momentum, we stuck at local minimum with momentum, it helps to cross the local minimum and to reach the global minimum.

ADAM is an example of using momentum.

i) - start with guess for parameters: $v, \{w_j\}$ (at random close to zero)

- use $x^{(i)}$ and current parameters to compute outputs: $z^{(i)}, \hat{y}^{(i)}$ } forward pass

- use outputs $z^{(i)}, \hat{y}^{(i)}$ to update parameters: $v, \{w_j\}$ } backword pass.

- continue above two steps while loss changes.

gradients are propagated in the backword pass.

i)

In fully connected layer each neuron gets inputs from neurons of the previous layer. each neuron has it's own weights. it's very expensive in terms of memory and computation for large networks.

In convolution layer neurons receive inputs from only a window which is a subarea of previous layer and have same set of weights for each filter. It's used for image data where the features are local and equally likely to occur anywhere. less number of inputs and weights make convolution layers relatively cheap in terms of memory and computation.

K) large number of parameters tend to overfit. but, we need to prevent overfitting for regularization. we can get that by using dropout.

at each training stage drop out units in fully connected layers with probability of $(1-p)$, where p is hyperparameter.

removed nodes are reinstated with original weights in the sub sequent stage.

Dropout during training $p = 0.5$
(Lower for input nodes - not to loose data).

Dropout during testing:

- each node's output weighted by a factor of p.

- equivalent to averaging 2^n dropped out networks and computing expected value.

Advantages of dropout are:
- reduces nodes interactions (co-adaptation),
- reduces overfitting
- increases training speed.

Ans-2| a) I = 4×4 RGB image.

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad G = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Convolve with 3×3×3 filter having all ones.

without zero padding, we get 2×2 output.

$$\text{output} = \begin{bmatrix} 9\cdot1+9\cdot2+3\cdot1+3\cdot2+3\cdot3 & 9\cdot1+9\cdot2+3\cdot1+3\cdot2+3\cdot3 \\ 9\cdot1+9\cdot2+3\cdot2+3\cdot3+3\cdot4 & 9\cdot1+9\cdot2+3\cdot2+3\cdot3+3\cdot4 \end{bmatrix}$$

$$= \begin{bmatrix} 45 & 45 \\ 54 & 54 \end{bmatrix}$$

b) with zero padding, we get 4×4 output.

$$\text{output} = \begin{bmatrix} 4+8+6 & 6+12+9 & 6+12+9 & 4+8+6 \\ 6+12+12 & 9+18+18 & 9+18+18 & 6+12+12 \\ 6+12+18 & 9+18+27 & 9+18+27 & 6+12+18 \\ 4+8+14 & 6+12+21 & 6+12+21 & 4+8+14 \end{bmatrix}$$

$$= \begin{bmatrix} 18 & 27 & 27 & 18 \\ 30 & 45 & 45 & 30 \\ 36 & 54 & 54 & 36 \\ 26 & 39 & 39 & 26 \end{bmatrix}$$

c) dilation rate of 2.
using zero padding.

we get 4x4 output.

$$\text{output} = \begin{bmatrix} 4+8+8 & 4+8+8 & 4+8+8 & 4+8+8 \\ 4+8+12 & 4+8+12 & 4+8+12 & 4+8+12 \\ 4+8+8 & 4+8+8 & 4+8+8 & 4+8+8 \\ 4+8+12 & 4+8+12 & 4+8+12 & 4+8+12 \end{bmatrix}$$

$$= \begin{bmatrix} 20 & 20 & 20 & 20 \\ 24 & 24 & 24 & 24 \\ 20 & 20 & 20 & 20 \\ 24 & 24 & 24 & 24 \end{bmatrix}$$

d) - extract image patches (windows)
- vectorize image window and filter and perform dot product (plus bias)
- filter extends full depth of image
- multiple convolution filters per location (e.g. oriented edges)
- use stride to move filter → activation map may be smaller.

Instead of ordinary convolution (1D):

$$(I * K)(t) = \sum_{\tau} I(t-\tau) K(\tau)$$

use: $(I *_2 K)(t) = \sum_{\tau} I(2t-\tau) K(\tau)$

take steps of size 2 in image when performing the convolution.

e) multiple scale analysis can be achieved by using convolution and pooling layers.

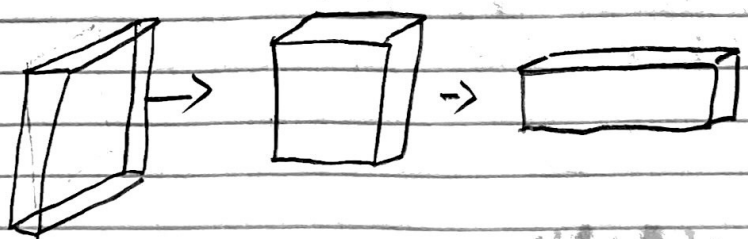
for multiple scale analysis we reduce spetial dimensions.
to reduce spetial dimensions we can use pooling or stride in convolution.

but in convolution it is random, so, we use pooling to reduce spetial dimensions and convolution to extract features.

for multiple scale analysis we can use convolution and pooling layers alternatively at different depth of convolution we can use same window size that means we keep the same number of coefficients. that retains more information.

f) when we use multiple convolution layer with no-padding or stride value more than 1 or we use pooling layer, the spetial dimensions decrease.

To compensate for reduced dimensions we increase depth (channels). so, we can keep same number of coefficients



multiple scale analysis with higher complexity.

when we apply convolution layer, spetial dimensions might get decrease and we can loose some information from the image

but as we increase depth or channels or units in convolution layer then we can store more information of these features in the image as it's spetial dimensions get decrease.

g) Given,
128 x 128 x 32 tensor,
16 convolution filters of size 3x3x32,

convolving,
without zero padding,
stride of 1,
dilation of 1,

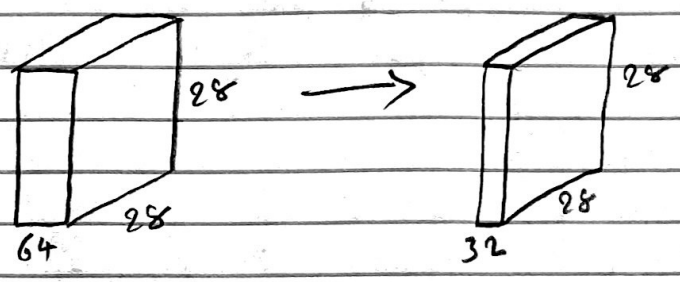
size of the resulting tensor will be,
126 x 126 x 16.

h) convolving,
without zero padding,
stride of 2,

size of the resulting tensor will be,
63 x 63 x 16.

i) 1×1 convolution is used to reduce dimensions

In this reduced dimensions Height and width will be same but number of channels will be less.



Here input dimensions are 28×28×64 but when we apply 32 convolution filters of size 1×1×64 we get output of dimensions 28×28×32.
Here, we can see 1×1 dimension of filter gives us same height and width and 64 dimension of filter gives us one single value but we used 32 units so we get 32 values. It means 1×1 convolution changes number of channels.

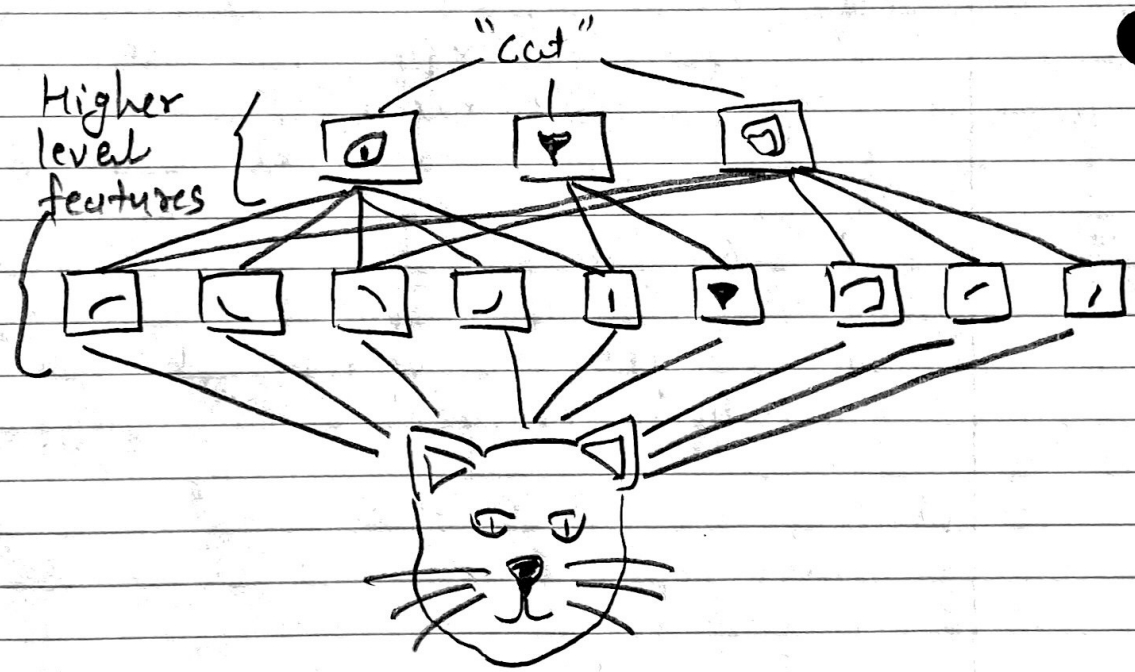
ii) convolution layer is used to extract features from input image. It finds the relationship between pixels by learning image features using small window of input data.
for multiple scale analysis we can use more convolution layers in the model.

early convolution layer extracts simple features like edge detection, different orientation of the features.

deeper convolution layer extracts complex features using simple features of early convolution layer like complex patterns, textures

As we go deeper in convolution layers we get lower to higher level features

Hierarchical features:



k) pooling is used to downsample the spetial dimensions without changing depth

In pooling we only select a value from local window

different ways of pooling:

- max pooling: partition non-overlapping regions and choose max value in each region.

Alternatives:
- average pooling
- L2 norm pooling
- ROI pooling (output size is fixed and input variable)

we choose max value in max pooling because max value of the feature helps to detect object in local window with lesser dimensions.

pooling is a layer without parameters so learing is not required for it

pooling also supports multiple scale analysis and helps to reduce number of coefficients amount of pooling is hyper parameter.

1) apply max pooling with 2x2 filter with a stride of 2.

Input image is a 4x4 RGB so, dimensions are 4x4x3

after applying max pooling output dimensions are 2x2x3.

Output

channel-1 = $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

channel-2 = $\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$

channel-3 = $\begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix}$