ASSIGNMENT — 3

AKSHAY R
A20442409

Review questions:

1. Neural networks.

(a).

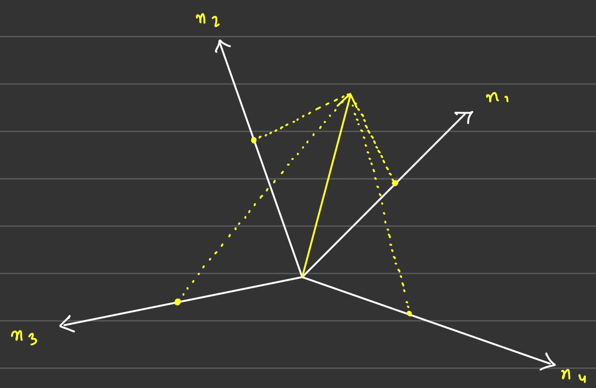
* Rows of θ^T (columns of θ) are templates
* With k rows of θ^T we have k templates (one template per class)
* $\theta^T x$ measures how well x matches with each of the k templates (dot product of x with rows of θ^T)
* High similarity of a template of a particular class indicates high membership in this class.

$$\hat{y} = \theta^T x + \theta_0$$

$$\hat{y} = \begin{bmatrix} n_1^T \\ \vdots \\ \vdots \\ n_k^T \end{bmatrix} \begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} d_1 \\ \vdots \\ \vdots \\ d_k \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} n_1 \cdot x \\ n_2 \cdot x \\ \vdots \\ n_k \cdot x \end{bmatrix} + \begin{bmatrix} d_1 \\ \vdots \\ \vdots \\ d_k \end{bmatrix}$$

n_1, n_2, n_3, n_4 are basis vectors.



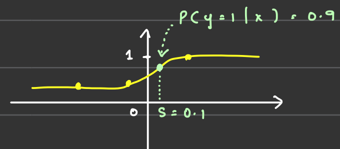
(b)

* Given similarity score (or decision boundary distance S_j) from a linear classifier (the higher the better), compute :

$$\underset{j^{th}\text{ unit}}{\hat{y}_j}\overset{i^{th}\text{ example}}{(i)} \equiv P(\underset{\text{LABEL}}{y = j} \mid \underset{\text{INPUT}}{x(i)})$$

* In a 2 class classification case :

$$\hat{y}_j^{(i)} \equiv P(y = 1 \mid x^{(i)}) = \text{sigmoid}(S_j^{(i)})$$



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

* In a k class classification case :

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

* Used at the last layer

(c) L_1 and L_2 loss

$$L_1 : L_i(\theta) = \sum_{j=1}^{k} \left| \hat{y}_j^{(i)} - y_j^{(i)} \right|$$

$$L_2 : L_i(\theta) = \sum_{j=1}^{k} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2$$

$\hat{y}_j^{(i)}$ is the predicted value : $y_j^{(i)} \in R$

Hubber loss :

$$\rho_\delta(d) : \begin{cases} \frac{1}{2}d^2 & \text{for normal errors. if } |d| \leq \delta \\ \delta(d - \frac{\delta}{2}) & \text{otherwise for outliers} \end{cases}$$

$d \Rightarrow$ distance b/w pred and actual.

$$L_i(\theta) = \sum_{j=1}^{k} \rho_\delta(\hat{y}_j^{(i)} - y_j^{(i)})$$

Cross entropy loss :

* Convert similarity scores to probabilities :

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

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

  Maximize

* Log - likelyhood: $-\log L(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \log(P(y = j \mid x^{(i)}))$

  Minimize (negative LL)

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

* 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 explaination is when the weigts θ are lower (eg: when $\theta_{ij} = 0$ we remove one coefficient)

* Smaller coefficients $\Rightarrow$ more stable solution that will generalize better.

ERROR TERM

$$* \quad L(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_i(f(x_{ij}, w), y_i) + \underbrace{\lambda}_{\text{Weight of Regularization}} \overbrace{R(\theta)}^{\text{Regularization term}}$$

HYPER-PARAMETER

$$R(\theta) = \underbrace{\sum_i \sum_j \theta_{ij}^2}_{L_2} \quad \text{or} \quad \underbrace{\sum_i \sum_j |\theta_{ij}|}_{L_1}$$

* L_2 regularization minimizes weights while spreading term (eg : $0.5^2 + 0.5^2 < 1^2 + 0^2$)
* L_1 regularization doesn't have this property and may concentrate terms (eg : $|0.5| + |0.5| = 1 + 0$)

(e) We move in the opposite direction of the gradient because we are trying to minimize the loss function and hence in a -ve direction with respect to the gradient.

(f) Stocastic gradient decent :

* Randomly order examples.
* for $\overbrace{j : 1 \ldots m}^{\text{1 EPOCH : Traverse through entire dataset.}}$

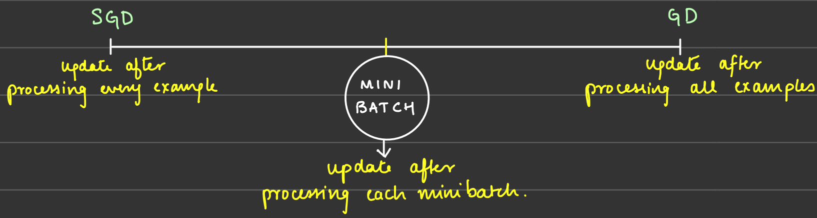
$$\theta^{(i+1)} \longleftarrow \theta^{(i)} - \eta \nabla L_j(\theta^i)$$

Based on example j

* Example for linear regression

$$\theta^{(i-1)} \longleftarrow \theta^{(i)} - \eta\left(x^{(j)^T}\theta - y^{(i)}\right)x^{(j)}$$

update after every example.



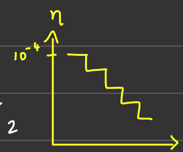
(g)

* Learning rate need not be fixed!

→ make learning rate smaller as iterations progress

* Stratagies :

① Step decay : every k iterations $\eta \leftarrow \eta/2$

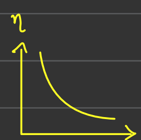


② Exponential decay :

Decay rate

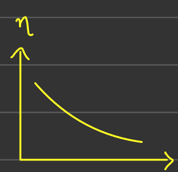
$$\eta_t = \eta \cdot e^{-\frac{k}{t}}$$

Iteration index



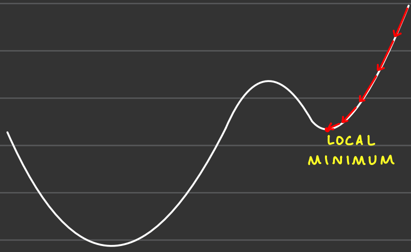
③ Fractional decay :

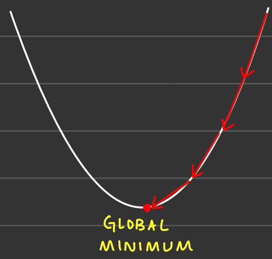
$$\eta = \eta_0 / (1 + kt)$$



→ Use when loss is not converging.

(h)

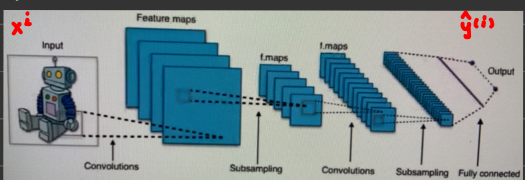




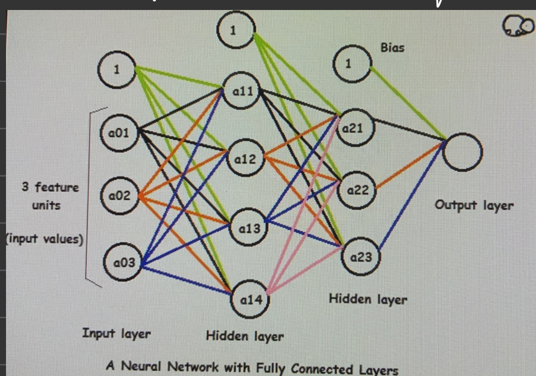
Not always will we have a loss function that will yield a global minimum while gradient decent. Sometimes we can get stuck in a local minima during gradient decent. Momentum is used to over the local minima problem.

(j) As the name states a fully connected layer is one where every neuron is connected to every other neurons in the preceding or the previous layer. Weights are calculated through forward and backward passes.

A CNN layer is one where weights are calculated by convolving a filter over an image and then using gradient decent.

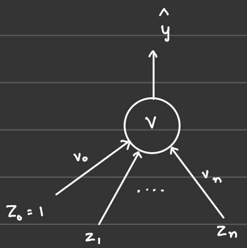


Convolution layers



Fully connected layers.

(i)



LOSS = E

chain rule:

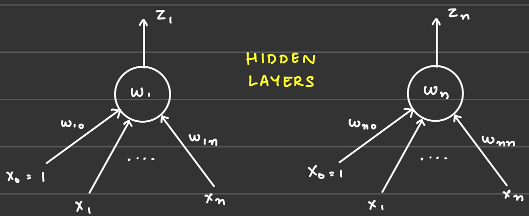
loss

$$\frac{\partial E}{\partial v} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial v} \longrightarrow \text{find } v$$

parameters

$$v^{(i+1)} \longleftarrow v^{(i)} - \eta \nabla \frac{\partial E}{\partial v}$$

HIDDEN LAYERS



$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial w_j} \longrightarrow \text{find } w_j$$

* Start with a guess of 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, $\{w_j\}$: backward pass (push back gradients)
* Continue while loss changes.

* Represent the network using a computational graph.
* Because each node is simple it has a simple explicit expression for its derivatives.
* Forward pass : Push input to compute all intermediate node values
* Backward pass : Starting with end nodes push gradients towards the begining.
* Multiply backpropogated gradients (from back) by current gradient and propogate this

Gradients are propogated in the backward pass.

(k) Regularization measures:

* Large number of parameters tend to overfit

Methods to prevent overfitting:

① Dropout:

- At each training stage drop out units in fully connected layers with probability of $(1-p)$ ← HYPER-PARAMETER
- Removed nodes are reinitiated with original weights in the subsequent stage.