ANN - ASSIGNMENT 2 (Theory)

1. (a) Formulation of linear regression as maximization of a likelihood function:

First of all, let's declare our model as a Gaussian distribution centered around a line, with variance σ^2.

$$y = wx + b + \epsilon \quad \text{where } \epsilon \in (0, \sigma^2)$$

which is eqⁿ to $\quad \cancel{y} \; y \sim N(wx+b, \sigma^2)$

The probability distribution / likelihood funcⁿ is then given by:

$$f(y|x; w, b, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-wx-b)^2}{2\sigma^2}}$$

The likelihood funcⁿ over the entire data set is:

$$L_X(w, b, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{i=1}^{m} e^{-\frac{(y^{(i)} - wx^{(i)} - b)^2}{2\sigma^2}}$$

Let's represent $wx^{(i)} + b$ as $\hat{y}^{(i)}$

The log-likelihood becomes:

$$l_X(w, b, \sigma^2) = -\log(\sqrt{2\pi\sigma^2}) - \sum_{i=1}^{m} \left(\frac{1}{2\sigma^2} \times \cancel{\log}\,(y^{(i)} - \hat{y}^{(i)})^2\right)$$

$\Rightarrow$ ~~$l_x(w, b, \sigma$~~ $l_x(\hat{y}, \sigma^2) = - \left[\log(\sqrt{2\pi\sigma^2}) + \frac{1}{2\sigma^2} \sum_{i=1}^{m} [y^{(i)} - \hat{y}^{(i)}]^2 \right]$

we have to ~~minimi~~ maximise the likelihood function. ~~that~~ Since σ^2 is a constant, effectively, we have to minimize

$$\sum_{i=1}^{m} [y^{(i)} - \hat{y}^{(i)}]^2.$$

Hence, this leads us to a mean-squared loss function, which we have to minimise.

So, in this way, the linear regression problem can be seen as a maximum-likelihood function.

1. (b)

(i) While training the learning algorithms for classification problems, we make use of ~~##~~ loss functions derived ~~f~~ using maximum likelihood estimation, and the derivation of MLEs is based on conditional probabilities. Hence, the algorithms tend to learn probability distributions rather than direct outputs.

(ii) Suppose we are working on an MNIST dataset. We use a softmax function to calculate the outputs values, and these values are used in backpropagation. Hence, the entire calculation is done using the ~~probabi~~ value output by softmax, ~~and not the f exact ##~~ This in turn, allows for our model to make more informed decisions about the "likelihood", and in turn, learn features. For example if we have an example which looks ~~a be~~ like 4 and 9 both. In this case, learning direct outputs won't help, but learning probability distributions will help the model to learn the common features between 4 and 9.

2. SGD with momentum:

This algorithm makes use of the concept of exponentially weighted average.

Computing exponentially weighted avg:

~~Take~~ consider parameters $\theta_1, \theta_2, ---, \theta_n$. The corresponding exp. weighted avgs (say $v_1, v_2, ---, v_n$) are computed as follows →

$v_0 = 0$ (say)

$v_1 = \beta v_0 + (1-\beta)\theta_1$

$v_2 = \beta v_1 + (1-\beta)\theta_2$

⋮

$$\boxed{v_n = \beta v_{n-1} + (1-\beta)\theta_n}$$

Now, SGD uses the exp. weighted average of the gradients to update the parameters.

~~$V_{dw} = V$~~ compute v_{dw}, v_{db}.

$W = W - \alpha v_{dw}$ → can also use $\left(\frac{v_{dw}}{1-\beta^t}\right)$ instead of v_{dw}

$b = b - \alpha v_{db}$

Physical Analogy :

If we think of our cost-minimizing model as a ball rolling down a hill, we can draw the following analogies :

In the eqⁿ $V_{dW} = \beta V_{dW\_prev} + (1-\beta) dW$

~~the, (1-β)dW~~ let us think of 'V_{dW}' as the "velocity" of the ball rolling down the hill. In such a case, "$(1-\beta)dW$" term acts as an "acceleration" term, which increases the velocity of the ball in the direction of the steepest descent. Hence, the convergence, or in analogous terms, the descent of the ball down the hill becomes faster.

6.(a) ~~Mini~~ Using gradient descent on the entire batch at once has two disadvantages mainly :

(i) uses a lot of memory

(ii) May converge at a local optima (in case of optimization of non-convex cost func^n's)

But if we implement ~~the~~ gradient descent ~~after~~ on every training example one by one, ~~we lose the~~ it adds some noiseness to the descent, allowing it to avoid local optimas. However, it loses the speeding effects of a vectorized implementation.

So, using mini-batch provides a mid-way path between the two extremes. It adds noiseness to the descent while retaining the speeding effects of vectorization.

In terms of robustness, mini batch-GD is less robust than SGD, but more robust than Batch GD.

Interms of efficiency, minibatch-GD is less efficient than batch GD, but more than SGD. So, it is a balance bet^n robustness and efficiency.

(b) If we initialize the weights ~~to zero~~ symmetrically, then all the neurons in a particular layer end up having the same gradient (calculated using backprop). So, all the neurons in a layer end up doing the same thing. ~~as one another~~. Hence, the training fails to achieve anything significant if weights are initialized symmetrically.

(c) Regularization essentially aims at making the values of parameters smaller, so that the model ~~~~ doesn't ~~~~ change a lot with small changes in input (i.e. has a relatively lower variance).

Some Regularization involves adding the values of ~~~~ the parameters to the cost function so that while minimizing the cost function, it ensures that the parameters don't take large values.

Other ways to stop overfitting are:

(i) Reduction of features
(ii) Early stopping while training
(iii) Using more data to train

Scanned with CamScanner

(d) Batch normalization in Neural Networks:
is applied to the values of neurons before applying the activation values. The implementation is as follows:

for any layer, l;

$$\mu^{[l]} = \frac{1}{m} \sum_{i=1}^{m} z^{[l](i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} \left(z^{[l](i)} - \mu^{[l]}\right)^2$$

$$z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^2 + \varepsilon}}$$

Batch normalization makes sure that the values of activations don't get too large or too small. ~~This makes~~ also, the values are in a particular range, which makes training a bit easier.

Apart from this, if we train our model on a particular ~~set of~~ type of data, then our model won't perform well on different sets of the same data. (An example can be a model trained on only pictures of white cats).
Batch normalization helps in reducing this particular phenomenon, (known as covariate shift)

4. The number of parameters in the conv layer is calculated as follows:

$\therefore$ window size = 3 × 3
input channels = 3
Output channels = 8

$$\text{So \# trainable params} = \Big(\underbrace{(3\times3)}_{\text{filter size}} \times \overset{\text{input channels}}{3} \Big) \times \underset{\text{output channels}}{8}$$

$$= 216$$

The dimension of the output is:

$$\cancel{N1+2P} \left\lfloor \frac{N1+2-3}{2} + 1 \right\rfloor \times \left\lfloor \frac{N2+2-3}{2} + 1 \right\rfloor \times 8$$

$$= \left\lfloor \frac{N1+1}{2} \right\rfloor \times \left\lfloor \frac{N2+1}{2} \right\rfloor \times 8$$