Assignment-2

Q1.

(a)

In order to perform probabilistic regression, we have to assign a label distribution over all $\mathbb{R}$ for every data point $\mathbf{x}$. Suppose we decide to do that using a Gaussian distribution need to decide on a mean $\mu_{\mathbf{x}}$and a variance $\sigma_{\mathbf{x}}^2 > 0$

Popular choice: Let $\mu_{\mathbf{x}} = \mathbf{w}^\top \mathbf{x}$and $\sigma_{\mathbf{x}}^2 = \sigma^2$i.e. $\mathcal{N}(\cdot | \mathbf{w}^\top \mathbf{x}, \sigma^2)$. We can also choose a different σ for every data point but that will be more complicated.

Likelihood function w.r.t a data point (x^i, y^i) then becomes:

$$\mathcal{N}\left(y^i \mid \mathbf{w}^\top \mathbf{x}^i, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\left(y^i - \mathbf{w}^\top \mathbf{x}^i\right)^2 / 2\sigma^2\right)$$

Negative log likelihood w.r.t a set of data points $\left(x^i, y^i\right)$ then becomes: $\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \left(y^i - \mathbf{w}^\top \mathbf{x}^i\right)^2$

(b)

- Probabilistic regression models are inherently different simple linear regression models. Prob. Models work on the principle of assigning a non-zero probability to every possible regression value in the range of the function.
- Limiting the result of the model to simply one value means that all the probability has been assigned to just one value but then the model is no longer probabilistic as it is not taking into account for possible deviations in the result due to fickle changes in the input.

Q2. We define a momentum, which is a moving average of our gradients. We then use it to update the weight of the network. This can be written as follows:

$$V_t = \beta V_{t-1} + (1-\beta) \nabla_w L(W, X, y)$$
$$W = W - \alpha V_t$$

Where L = loss function, alpha is the learning rate, beta is a hyperparameter, V is the velocity notation a measure of past values of the model used to depict momentum.
SGD is like a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through and not get stuck in narrow valleys, small humps and local minima and help reach global optima.

Q3.

(a) Reasons that mini-batch is used so widely are:

- Efficiency. Typically, especially early in training, the parameter-gradients for different subsets of the data will tend to point in the same direction. So gradients evaluated on 1/100th of the data will point roughly in the same general direction as on the full dataset, but only require 1/100 the computation. Since convergence on a highly-nonlinear deep network typically requires thousands or millions of iterations no matter how good your gradients are, it makes sense to do many updates based on cheap estimates of the gradient rather than few updates based on good ones.

- Optimization: Noisy updates may allow the user to bounce out of bad local optima

(b) If weight initialized symmetrically then will be updated by the same amount with each gradient update throughout training. This basically means that all hidden neurons (layers) will essentially doing the same calculation. This is not what we want since we desire different layers to compute different functions to realize the true potential of a neural network

(c) Regularization penalizes spurious features and helps reduce noise in the model thus helping tackle the problem of overfitting. Overfitting can be tackled by doing cross validation, early stopping or Pruning (for CART Models)

(d) To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. It helps stabilize the NN as:

- We can use higher learning rates because batch normalization makes sure that there's no activation that's gone really high or really low. And by that, things that previously couldn't get to train, it will start to train.
- It reduces overfitting because it has a slight regularization effect. Similar to dropout, it adds some noise to each hidden layer's activations. Therefore, if we use batch normalization, we will use less dropout, which is a good thing because we are not going to lose a lot of information. However, we should not depend only on batch normalization for regularization; we should better use it together with dropout.

Q4.

Number of parameters =
[(Size of Kernel * Stride) + 1] * Number of Input Channels

So, in this example number of parameters = [{(3*3*2) + 1} *3] = 57

Size of output image =
[(N1 – Kernel Height + 2*Padding) / Stride] + 1 **x** [(N2 – Kernel Width + 2*Padding) / Stride] + 1 **x** 3

So, size of output image = (N1+1)/2 **x** (N2+1)/2 **x** 3