Computer Vision *Abhay Doke, 29552668*

HW09: Linear models, neural networks

1. The perceptron will only converge if the data is linearly separable. For linearly separable data with margin δ, if $||\mathbf{x}|| \leq R$ for all data points, then it will converge in at most $\frac{R^2}{\delta^2}$ iterations (In class we assumed that $||\mathbf{x}|| \leq 1$). It is possible to force your data to be linearly separable as follows. If you have N data points in D dimensions, map data point $\mathbf{x}_n$ to the (D + N)-dimensional point $(\mathbf{x}_n, \mathbf{e}_n)$, where $\mathbf{e}_n$ is a N-dimensional vector of zeros, except for the n^{th} position, which is 1. (Eg., $\mathbf{e}_4 = (0, 0, 0, 1, 0, ...)$.)

   (a) Show that if you apply this mapping the data becomes linearly separable (you may wish to do so by providing a weight vector $\mathbf{w}$ in (D + N)-dimensional space that successfully separates the data).

   (b) How long will it take the perceptron algorithm to converge on this augmented data?

   (c) How does this mapping affect generalization (i.e., test performance)?

   i. By adding the N dimensions to the data we are separating all the data points with every point having a unique value that no other data point has and this makes the data separable. For a data that has been augmented as $(\mathbf{x}_n, \mathbf{e}_n)$ the learned weight vector W_i will be $[0_1, 0_2, ..., 0_d, y_1, y_2, ..., y_n]$. It can be written as [D x 0, y_i for i = 1:N] which is a (D + N) - dimensional vector with first D values equal to zero and next N values representing the class labels of the respective N data points.

   So, for every training data point, the value of $X_i.W$ will be equal to y_i which is the class label of the ith data sample. The unique dimension that we embedded into the original data helps the perceptron to learn the weight vector. As first D values in the weight vector are zero, weight vector doesn't care about the input data anymore. It is only dependent on the next N values which makes the data separable.

   For example, consider data points for the XOR gate as [[0, 0], [0, 1], [1, 0], [1, 1]] with output labels as [0, 1, 1, 0]. Augmented data will be [[0, 0, 1, 0, 0, 0], [0, 1, 0, 1, 0, 0], [1, 0, 0, 0, 1, 0], [1, 1, 0, 0, 0, 1]]. Here the D = 2 and N = 4, so the learned vector will be [2 X 0, y_i for i in 1:4] = [0, 0, 0, 1, 1, 0] which separates the augmented data.

   ii. For a binary classification of this augmented data, the Perceptron algorithm will take one iteration to learn the augmented data.

   iii. This kind of augmentation during the training is overfitting the data. At the test time we don't know how to augment the data. The generalization ability of the perceptron is lost by augmenting the data.

2. Why do we minimize the surrogate loss such as logistic loss or hinge loss instead of minimizing classification error (zero/one loss) for training a linear classifier for classification tasks?

Zero-one loss is non-convex and very tough to optimize. It is very sensitive to the weight vector and small changes in weight vector can cause large changes in loss. Surrogate loss is covex and easy to optimize. Hence for the linear classifier we minimize the surrogate loss such instead of zero/one.

3. Considering training linear classifiers and neural networks with gradient descent, why is weight initialization a practical issue on neural networks but not on linear classifiers.

Neural networks use non linear activation functions such as Tanh or Sigmoid. These functions suffer by the problem of vanishing gradients when the absolute value of the input is very high. During backpropagation, the gradients for these very high or very low inputs are very low. If the neural network is having many layers and the initial weights are very small or very high, then during the backpropagation, very negligible or zero gradient flows back through these activation functions and learning by these activation function nearly stops. If many activation functions face this kind of vanishing gradient problem, then the network almost stops learning. So in neural networks, we use special kind of initialization such as Xavier initialization.
In linear classifiers, we do not have such problem of vanishing gradients and hence weight initialization a practical issue on neural networks but not on linear classifiers