ANN ASSIGNMENT 2

Atreya Goswami

May 23, 2020

1 Linear Regression and Logistic Regression

1.1 Linear Regression as MLE

We can frame the problem of fitting a linear regression model as the problem of probability density estimation. We think about a probabilistic procedure that might've given rise to the data. This probabilistic procedure would have parameters and then we can take the approach of trying to identify which parameters would assign the highest probability to the data that was observed. We observed that there appears to be a linear relationship between x and y, but it's not perfect. We think of these imperfections as coming from some error or noisy process. We assume that these noise are sampled from a gaussian distribution of mean 0 and variance σ^2.

$\epsilon \sim \mathcal{N}(0, \sigma^2)$

$y = \theta_1 x + \theta_0 + \epsilon$ (Assuming one feature only)

By adding a constant to a gaussian we just change the mean of the distribution.So the resulting conditional probability distribution for our generative probabilistic process is

$y \sim \mathcal{N}(\theta_1 x + \theta_0, \sigma^2)$

$p(y_n|x_n, \theta_1, \theta_0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y_n - \theta_1 x_n - \theta_0)^2}$

Each point is i.i.d so we can write the likelihood function with respect to all of our observed points as the product of each individual probability density.

$L_X(\theta_0, \theta_1, \sigma^2) = \prod_{i=1}^{m} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(y^{(i)} - \theta_1 x^{(i)} - \theta_0)^2}$

We then try to find the parameters that maximise this function or instead we choose to maximise the log likelihood (MLE). The log-likelihood function :

$l_X(\theta_0, \theta_1, \sigma^2) = log L_X(\theta_0, \theta_1, \sigma^2) = -m log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta_0 - \theta_1 x^{(i)})^2$

To maximise this log-likelihood function we thus need to minimise $\sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$ where $\hat{y} = \theta_0 + \theta_1 x$

Thus the Maximum Likelihood Estimation in our probabilistic model is the same as minimising the sum of squared errors as is done in Linear Regression.

1.2 Modelling Conditional Probabilities in Logistic Regression

If we had learned the binary outputs directly instead of the probabilities of the class labels, then there is no proper criterion to predict these discontinuous binary labels directly. If we had led the gradient descent algorithm to learn class labels, they would have led to large costs which would have taken long time to converge or might not. There isnt any function which maps the numerical input features to class labels, instead they can be mapped to some continous random variable which can then be mapped to the class labels. In that case if we had used linear regression to predict class labels, the predicted values wouldn't be limited between 0 and 1. Also had we implemented linear regression, we would have assumed that the noise belongs to a gaussian distribution which is not the case. In order to predict a continuous variable between 0 and 1 and then map it to the class labels we are using a logistic regression classifier. The variable we are predicting turns out to be the probability distribution over classes.

2 Gradient Descent With Momentum

Gradient Descent with Momentum is a variant of the Gradient Descent Optimization algorithm that we commonly use for updating the parameters of our network. It is generally used when weights are updated after training on mini-batches. As the weights are updated after training on a few examples in the mini-batch (as compared to Batch Gradient Descent) there is a lot of variance in the direction of the update and the path taken to optimise will oscillate toward the global minima. Momentum aims to dampen these oscillations.

Momentum takes into account past history of updates to smooth out the updates. We store the previous direction of the gradients by an exponentially weighted average of the gradients denoted by v. Thus we actually use a moving average of the past gradients to update our parameters.

$v_{d\theta} := \beta v_{d\theta} + (1 - \beta)\nabla_\theta J(\theta)$

$\theta := \theta - \alpha . v_{d\theta}$

where,

θ are the parameters of our model

$v_{d\theta}$ is the exponentially weighted average of the gradients

$\nabla_\theta J(\theta)$ is the gradient of the cost function w.r.t to the parameters.

α is the learning rate of our algorithm

β is a hyperparameter s.t $v_{d\theta}$ roughly averages over the last $\frac{1}{1-\beta}$ iterations of the descent.

We can relate Gradient Descent with Momentum with the concept pf Momentum in Physics. We can think of the analogy of a ball rolling down from some height on a hill. Let $J(\theta)$ represent potential energy of the ball on the hill. Then its gradient represents a term similar to acceleration. $v_{d\theta}$ represents the velocity or momentum of the ball and β acts like a friction coefficient. The ball rolls down the hill as per the direction of slope of the hill. If the acceleration is in the direction of the velocity then the momentum increases in that direction. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions

whose gradients change directions due to the weighted measure of the initial velocity in that iteration. As a result, we gain faster convergence and reduced oscillation. β damps the velocity and reduces the kinetic energy of the system, or otherwise the particle would never come to a stop at the bottom of a hill.

3 Short Questions

3.1 Mini-Batches

Mini-Batch are used to make the process of Gradient Descent efficient and faster. In the process of Batch Gradient Descent , if there are a lot of training examples then it takes a lot of time to update the parameters as they get updated after a complete iteration of the training set. Also the error gradient is very stable, it may also lead to unwanted convergences and very slow learning. If we use the Stochastic Gradient Descent which updates parameters after considering one training example at a time, then it is computationally inefficient due to the large number of explicit for loops , unstable and fluctuating gradients and expensive due to updates after every step. This would mean giving up all the computational advantage gained by vectorization. Mini Batch Gradient Descent is an often-preferred method since it uses a combination of Stochastic Gradient Descent and Batch Gradient Descent. It simply separates the training set into small batches and performs an update for each of these batches. It thus creates a balance between the efficiency of Batch Gradient Descent and the robustness of Stochastic Gradient Descent.

3.2 Assymetric Initialization of Weights

If we initialise weights symmetrically, then the very purpose of introducing several nodes in a layer is destroyed. As the weights are symmetrically connected to the previous and next layers of the network, hence the gradients are the same for each of the nodes and thus the weights of each node gets updated similarly. Hence throughout the training procedure the weights of all the nodes remain same and the whole layer acts as a single node. So it is important to randomly initialize the weights to break the symmetry of the network.

3.3 Regularization and Overfitting

$J(w,b) = \frac{1}{m}\sum_{i}^{m} l(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} ||W^{[l]}||_F^2$

Minimizing the cost function consists of reducing both terms in the right: the MSE term and the regularization term. So each time some parameter is updated to become significantly large, it will increase the value of the cost function by the regularization term, and as a result, it will be penalized and updated to a small value. So we thus try to make the weights as close as possible to zero. So effectively our networks become simpler as some of the nodes become insignificant. Also as the weights are diminished, the linear forward z in the nodes tend to lie in the linear regions of the activation curves thereby making the whole network tend towards linearity. Thus it reduces the complexity of the model by avoiding complex decision boundaries and minimises overfitting.

The other methods to remove overfitting are :

1. Getting more training data to solve the problem of variance.
2. Adding some dropout layers in our network randomly
3. Reducing the number of training features by removing the least important and remotely connected features from the model.
4. Cross Validation Sampling
5. Data Augmentation-Using image data the using the same data by flipping the images and rotating the image as by doing so the array change and thus the input order change.

3.4 Batch Normalisation

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it difficult to train models with saturating nonlinearities. We address this problem, known as internal covariance shift, by the method of batch normalisation. We normalise the linear parts/activations of the hidden layers to a desired value of mean and variance so as to speed up learning.

Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift) and allows each layer of a network to learn by itself a little bit more independently of other layers.It also helps reducing overfitting. It makes the landscape of the corresponding optimization problem significantly more smooth. This ensures that the gradients are more predictive and thus allows for use of larger range of learning rates and faster network convergence.

4 Analysing CNN

Input : $N1 \times N2 \times 3$

Kernel : 3×3

Padding(p) : 1

Stride(s) : 2

$n_{c,in} = 3$

$n_{c,out} = 8$

Output : $\lfloor \frac{N1+1}{2} \rfloor \times \lfloor \frac{N2+1}{2} \rfloor \times 8$

Weights : $3 \times 3 \times 3 \times 8$

Number of Parameters : 216 (excluding bias terms)