CS 419M Introduction to Machine Learning **Spring 2021-22**

Lecture 13: Introduction to Non Linearity and Gradient Descent

Lecturer: Abir De *Scribe: Group 2s*

13.1 Introduction To Non-Linearity

Linearity here refers to modelling the Output as a linear function of the corresponding input, $\mathbf{y_i} \propto \mathbf{x_i}$.
We turn our attention to non-linear models as linear models aren't as accurate as we would like them to be, or rather do not *model* the output accurately. Here accuracy is used rather loosely, it just refers to the model not being able to trace the required output.

13.2 What are the choices for non linear models?

13.2.1 Polynomial Approximations

Here we try to model the output as a polynomial function

$$y = \sum_{i=1}^{\infty} a_i x^i$$

The space spanned by 1, x, x^2... does not include the set of all function and hence we can say some linear combination of these will not perfectly fit certain outputs. We also see that while modelling log(x), the polynomial fit causes issues at x=0. Moreover, as the exponents of x grow bigger the terms shoot up and cause stability issues.

13.2.2 Universal Approximators

Theorem 13.1. ***Universal approximation theorem (non affine activation, arbitrary depth and constrained width)****: Let χ be a compact subset of R^d. Let σ: $R \to R$ be any non-affine continuous which is continuously differentiable at at least one point, with nonzero derivative at that point. Let $N_{d,D:d+D+2}$ denote the space of feed-forward neural networks with d input neurons, D output neurons and an arbitrary number of hidden layers with d+D+2 neurons such that every input neuron has activation function σ and every output neuron has identity as activation function as identity, with input layer ϕ and output layer ρ. Then given an $\epsilon > 0$, and any function belonging to the set of continuous functions from $X \to R^d$ there exists an approximator $\hat{f}$ belonging to $N_{d,D:d+D+2}$ such that*

$$\sup_{x \in \chi} |f(x) - f\hat{(}x)| < \epsilon.$$

Source: Wikipedia-Universal approximation theorem

The above theorem states that there exists an approximator/neural network under appropriate conditions which can approximate any continuous function. However, it is important to note that this theorem does not give us a method to obtain such an approximator for a given function. One such approximator is an LRL (Linear-ReLU-Linear) unit. Here, the middle ReLU layer is sandwiched between 2 Linear layers. That is, $z_1 = w.X_{input} + b$
$z_2 = ReLU(z_1)$
$z_3 = p.z_2 + q$ where w and p are weight vectors of appropriate dimension (depending on input) and b,q are respective bias terms. Since, this satisfies the conditions of universal approximation theorem we are guaranteed to have an approximator of this form, but the number of layers (LRL units) so that we get a reasonably close approximation are usually found out by trial and error.

13.3 Introduction to Gradient Descent

Given an arbitrary function $f(x)$, parametrized by θ,and a data set $\mathcal{D}$ we wish to find the optimal parameter θ^* such that $y \approx f_{\theta^*}(x)$. More specifically, we wish to minimize the following objective

$$\min_{\theta} l_\theta = \min_{\theta} \sum_{i \in \mathcal{D}} (y_i - f_\theta(x_i))^2 \tag{13.1}$$

We start with an initial arbitary initial point θ_0, which we hope to be near the local minima. Then, θ_t is calculated iteratively according to the following update rule,

$$\theta_{t+1} = \theta_t - \gamma \left(\frac{\partial l_\theta}{\partial \theta} \right)_{\theta = \theta_t} \tag{13.2}$$

Here γ is called the learning rate.

Observe that, if we make γ high, although it would increase the learning speed, but it may so happen that the point escapes local minima or fail to converge, due to oscillations around the minima. Hence, we need to find appropriate bounds on γ.

By second order taylor approximation we have the following,

$$l_{\theta_{t+1}} = l_{\theta_t} + \frac{\partial l_\theta}{\partial \theta}(\theta_{t+1} - \theta_t) + \frac{1}{2}(\theta_{t+1} - \theta_t)^T H(\theta)(\theta_{t+1} - \theta_t) \tag{13.3}$$

$$\implies l_{\theta_{t+1}} \leq l_{\theta_t} + \frac{\partial l_\theta}{\partial \theta}(\theta_{t+1} - \theta_t) + \frac{\lambda_{max}}{2}||\theta_{t+1} - \theta_t||^2 \tag{13.4}$$

Since, $H(\theta) \leq \lambda_{max} I$,where λ_{max} is the maximum of the set of eigenvalues and I is the identity matrix.

$$\implies l_{\theta_{t+1}} - l_{\theta_t} \leq -\gamma \left| \frac{\partial l_\theta}{\partial \theta} \right|^2 + \frac{\lambda_{max}}{2} \gamma^2 \left| \frac{\partial l_\theta}{\partial \theta} \right|^2 < 0 \tag{13.5}$$

Since, we want the loss to decrease, ie. $l_{\theta_{t+1}} - l_{\theta_t} < 0$, hence we can bound the upper bound of the above inequality by 0 to get condition on γ

$$\implies \gamma < \frac{2}{\lambda_{max}} \tag{13.6}$$

Note that, if any of the eigenvalue of the hessian is very large, learning becomes very difficult due to a very low learning rate constraint.

In case of convexity, we can also have the following inequality,

$$l_{\theta_{t+1}} - l_{\theta_t} \geq \left(\frac{\partial l_\theta}{\partial \theta}\right)^T_{\theta=\theta_t} (\theta - \theta_t) \tag{13.7}$$

$$\implies l_{\theta_{t+1}} \geq l_{\theta_t} - \gamma \left|\frac{\partial l_\theta}{\partial \theta}\right|^2 \tag{13.8}$$

We can also represent gradient descent by the following optimization problem,

$$\theta_{t+1} = \arg\min_\theta \frac{1}{2}||\theta - \theta_t||^2 + \gamma \left(l_{\theta_t} + (\theta - \theta_t)^T \left(\frac{\partial l_\theta}{\partial \theta}\right)_{\theta=\theta_t}\right) \tag{13.9}$$

The first term, wants θ to be close to θ_t, whereas the second term forces a movement in the direction of gradient. The parameter γ controls the tradeoff between the two terms.

13.3.1 Taking data in batches

Consider

$$l_{\theta_t} = \sum_{x \in D} (y_i - f_{\theta_t}(x_i))^2 \tag{13.10}$$

Here we have taken complete dataset while calculating loss. But the problem with this loss function is that, if we are stuck in some local minima, then we can not get out of it.

Now consider

$$l_{\theta_t} = \sum_{x \in B_t} (y_i - f_{\theta_t}(x_i))^2 \tag{13.11}$$

where B_t is a batch of dataset. Now notice that, loss function will change in every batch and hence if we are stuck at local minima, then we can get out of it.

We may also use

$$l_{\theta_t} = \frac{\sum_{x \in B_t} (y_i - f_{\theta_t}(x_i))^2}{|B_t|} \tag{13.12}$$

where $|B_t|$ is batch size.

But then

$$\theta_{t+1} = \theta_t - \frac{\gamma}{|B_t|} \cdot \frac{\partial l_{\theta_t}}{\partial \theta} \tag{13.13}$$

Initially γ was small (in orders of 10^{-3}). $|B_t|$ is usually in orders of 100. Thus $\frac{\gamma}{|B_t|}$ will be in orders of 10^{-5} which is very small. We can increase γ itself such that $\frac{\gamma}{|B_t|}$ is in appropriate range.

Q. Why can't we take $\gamma = ||\theta_{t-1} - \theta_{t-2}||_{l_2}$?
Ans. Consider the case where we have divided our dataset into batches and we are training our neural network by giving batch by batch data.

$$\theta_{t+1} = \theta_t - \gamma \cdot \frac{\partial l_{\theta_t}}{\partial \theta} \tag{13.14}$$

Notice that $\frac{\partial l_{\theta_t}}{\partial \theta}$ will not be always zero as data changes according to batch.
Now if γ is a non zero constant, then learning will occur whenever $\frac{\partial l_{\theta_t}}{\partial \theta}$ is non zero.
But if γ is $||\theta_{t-1} - \theta_{t-2}||_{l_2}$, learning will stop when $\theta_{t-1} = \theta_{t-2}$. Because of zero learning, θ will not be updated and we will be stuck at local minima.

13.4 Group Details and Individual Contribution

1. Aaron Jerry Ninan 190100001 - Introduction to Gradient Descent
2. Bhavya Singhal 200010014 - Why we cant take $\gamma = ||\theta_{t-1} - \theta_{t-2}||_{l_2}$?
3. Kaiwalya Joshi 20D070043 - Taking data in batches
4. Rhishabh Suneeth 200260040 - Universal Approximators
5. Rishabh Ravi 200260041 - Introduction to non linear models