Few-Shot Learning - Assignment1

Nikhila KN

February 20, 2022

1 Regularization

Machine learning models are designed to learn the relationship between dependent variable(s) and independent variables(s). To build such models, we need a set of data to train the model. The performance of the model depends on how it works for new data not seen during training. That is, a good model learns the relationship between the dependent variable(s) and the independent variable(s) from the data during model training and makes maximum error-free predictions when the new data arrives. However, in many circumstances, the model performs well during the training phase but fails miserably when new data is introduced. This problem is called *over fitting*. That is the test error is high in the evaluation phase of the model.

Regularization is a set of techniques used in machine learning to reduce test error, which may come at the expense of higher training error [1]. The majority of regularization techniques are focused on regularizing estimators. Regularization of an estimators works by trading increased bias for reduced variance.

The Figures 1,2 & 3 depict the under-fitting, appropriate fitting, and over-fitting scenarios, respectively. The purpose of regularization is to change the model from an over-fitting to an appropriate-fitting state.

1.1 Classical Regularization: Parameter Norm Penalty

We refer to the techniques utilized in the general machine learning and statistics literature as *classical regularization*. The majority of traditional regularization methods rely on adding a parameter norm penalty $\Omega(\theta)$ to the loss function J to limit the capacity of models like neural networks, linear regression, or logistic regression. The regularized loss function is denoted as follows:

$$\tilde{J}(\theta, X, y) = J(\theta\ X, y) + \alpha\Omega(\theta) \tag{1}$$

where α is a hyper parameter that weighs the relative contribution of the norm penalty term, Ω, relative to the standard loss function $J(x; \theta)$. The hyper parameter α should be a non-negative real number, with $\alpha = 0$ corresponding to no regularization, and larger values of α corresponding to more regularization.

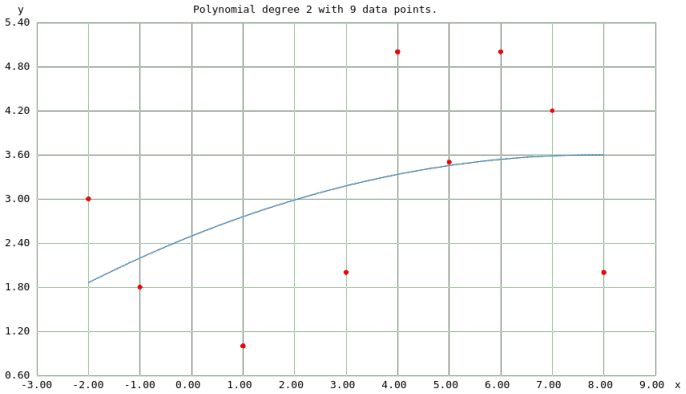

Figure 1: Under-fitting

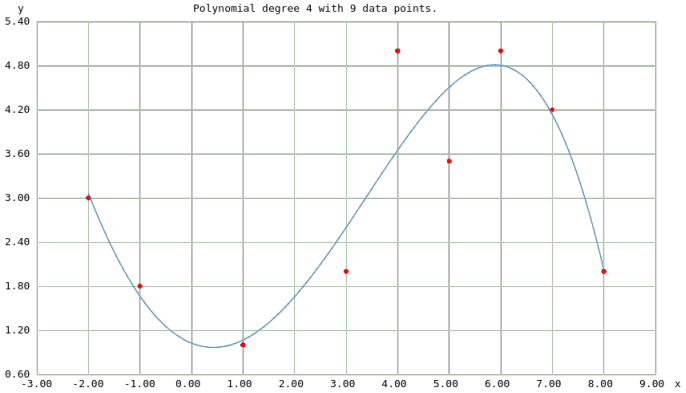

Figure 2: Appropriate-fitting

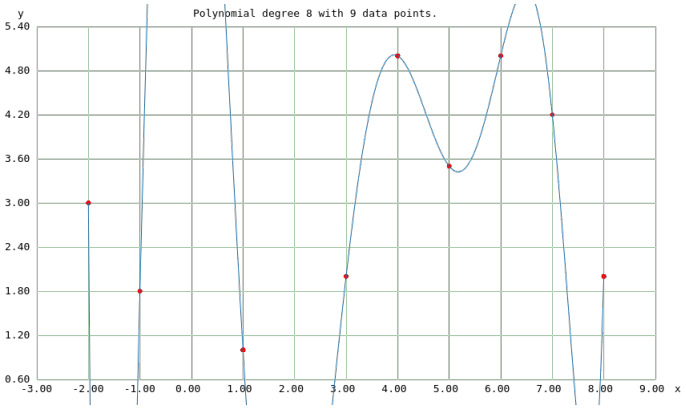

Figure 3: Over-fitting

1.2 L^2 Parameter Regularization

The L^2 parameter norm penalty is one of the simplest and most prevalent types of classical regularization. *Ridge regression* is another name for this type of regularization. It's also applicable to neural networks, where the penalty is equal to the total of all weight vectors' squared L^2. This is referred to as weight decay in neural networks. For neural networks, we typically employ a different coefficient α for each layer of the network's weights. A validation set should be used to fine-tune this coefficient.

1.3 L^1 Regularization

L^2 regularization is the most frequent type of regularization for model parameters, such as neural network weights. It isn't the only method of regularization used in common. L^1 regularization is a type of model parameter penalization that differs from L^2. L^1 regularization on the model parameter w could be defined as follows in formal terms:

$$\Omega(\theta) =\parallel w \parallel_1 = \sum_i \mid w_i \mid \tag{2}$$

1.4 Data Augmentation

Training a model with a variety of data is the best technique to generate a generalised model. In the real world, getting more data is challenging. Producing more fake data is one way to get around this problem.

Data augmentation is one of the most often utilized pre-processing approaches for training neural networks. The most essential effect is that it broadens or expands the data's diversity. Because of the increased diversity, the model encounters a distinct version of the original data at each training cycle. As a result, the increased diversity from data augmentation lowers the model's variance by improving its generalization ability.

1.5 Bagging (short for bootstrap aggregating)

Bagging is a strategy that combines many models to reduce generalization error. The goal is to train several different models individually, then have all of them vote on the output for test examples. This is an example of a general strategy in machine learning called *model averaging*. Model averaging works because different models will typically make different errors on the test set to some extent. Constructing k different datasets is what bagging entails. The number of examples in each dataset is the same as in the original dataset, however each dataset is built by sampling with replacement from the original dataset. This means that each dataset is likely to be missing part of the original dataset's samples, as well as having multiple duplicates. Then, on dataset i model i is trained. The trained models differ as a result of the differences in which

instances are included in each dataset. The use of model averaging to reduce generalization error is a very powerful strategy.

1.6 Dropout Regularization

Dropout is a computationally simple yet effective way of regularizing a wide range of models. Dropout can be regarded of as a way to make neural network bagging more practical. Bagging entails simultaneously training and evaluating many models for each test sample. When each model is a neural network, this appears unrealistic, because training and assessing a neural network takes time, and keeping a neural network takes memory. Dropout is a low-cost approximation for training and assessing an exponentially large bagged ensemble of neural networks. Dropout specifically trains the ensemble, which is made up of all sub-networks that may be created by deleting units from an underlying base network.

2 Transfer Learning

Transfer learning is an approach that facilitates a system to apply the knowledge and skills acquired in prior tasks to new ones. Which means a model built for a source task $\mathcal{T}_1$ can be reused to perform target task $\mathcal{T}_2$, thus we can save a significant amount of effort and time. In other words, instead of building a model from scratch, transfer learning allows us to design a model to accomplish a task using the learning experience from another model. Consider the case where we require a model to classify objects based on their color. We need a lot of labeled data to develop this model. Simultaneously, we have models that classify cars and flowers based on their color. We may leverage the knowledge from the car and flower classification models to create an object classification model from scratch. As a result, labor-intensive processes such as labeling can be reduced.

2.1 Definition

Suppose $\mathcal{D}$ is a domain with feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, x_2, ...x_n\} \in \mathcal{X}$. Given a specific domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ with two components: label space $\mathcal{Y}$ and an objective predictive function $f : \mathcal{X} \to \mathcal{Y}$. The function $\boldsymbol{f}$ is used to predict the corresponding label $f(x)$ of a new instance x. This task, denoted by $\mathcal{T} = \{\mathcal{Y}, f(x)\}$, is learned from the training data consisting of pairs $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in \mathcal{Y}$ [2].

Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, where $\mathcal{D}_T \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_\mathcal{T}$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$ [2].

2.2 Negative Transfer

Some of the knowledge in the domains will be domain-specific, while others will be shared across domains. When the source and target domains are unrelated, brute-force transfer may fail in some cases. In the worst-case scenario, it may even harm learning performance in the target domain, a circumstance known as *negative transfer*.

2.3 Categorization of Transfer Learning

According to label-setting aspect, transfer learning mainly divided into three categories.

- Inductive transfer learning
- Transductive transfer learning
- Unsupervised transfer learning

2.3.1 Inductive Transfer Learning

The scenario can be classified as inductive transfer learning if the target task $\mathcal{T}_T$ is different from the source task $\mathcal{T}_S$. The domains may or may not be the same in this scenario. The tasks can be different either due to difference in the labeled space, i.e., $\mathcal{Y}_S \neq \mathcal{Y}_T$ or difference in the conditional probability distributions between the domains. Depending on the labeled and unlabeled data in the source domain, *inductive transfer learning* further classified into two.

- Case 1: The source domain contains a large amount of labeled data, then the multitask learning environment is comparable to the inductive transfer learning setting in this scenario. Inductive transfer learning, on the other hand, simply strives to improve performance in the target task by transferring knowledge from the source task, whereas multitask learning tries to learn both the target and source tasks at the same time.
- Case 2: When labeled data is not available in the source domain, inductive transfer learning is similar to self-taught learning. In self-taught learning, the label spaces between the source and target domains may differ, meaning that the source domain's side information cannot be used directly. As a result, in the absence of labelled data in the source domain, it's analogous to inductive transfer learning.

2.3.2 Transductive Transfer Learning

The scenario known as transductive transfer learning occurs when the source and target domains are distinct but the source and target tasks are the same. The domains are different because of the any two following reasons,

1. The feature space $\mathcal{X}_S$ in the source domain is different from the feature space $\mathcal{X}_T$ in the target domain.

2. Although the feature spaces of the domains are similar, the marginal probability distributions of domain data are not;i.e., $P(X_S) \neq P(X_T)$, where $X_{S_i} \in \mathcal{X}_S$ and $X_{T_i} \in \mathcal{X}_T$.

2.3.3 Unsupervised Transfer Learning

When the labeled data not available in the both source and target training, the scenario is known as unsupervised transfer learning. While similar to inductive transfer learning, unsupervised learning focuses entirely on unsupervised tasks such as clustering, dimensionality reduction, and so on.

3 Data Size, Model Complexity and Generalizability

When we talk about machine learning, we frequently use terminology like data size, model complexity, and generalization. In the context of neural networks and deep learning, data size is especially important. When we talk about huge data, we're talking about model complexity. Model complexity is a measure of how complicated a model is based on how close a hypothesis may be made to the actual probability distribution. This brings us to the concept of generalization, which is a key component of machine learning models. This section tries to answer the relationship between data size, model complexity and generalizability.

1. Why larger data need larger models?
   The larger data always helps to understand the underlying probability distribution better. But if we only have a small model, the model's ability to learn the underlying probability distribution is limited. As a result, bigger data necessitates larger models in order to reflect the underlying probability distribution as nearly as possible. For more clarity on this situation, consider a case with a linear model and a large data set with a complex distribution. In this scenario, the model performs badly and a larger model than a linear model is required to accurately comprehend the distribution. Therefore having more data should large model to improve the quality of the learning.

2. Why larger models learn from larger data?
   It is well known that a poor approximation is caused by insufficient training data. An over-constrained model will likely underfit the limited training dataset, while an under-constrained model would likely overfit the training data, resulting in poor performance in both cases. When there isn't enough test data, the model's performance will be overestimated with a high variance. Larger data tends to provide a better sampling of the

underlying probability distribution, giving the model a better chance of approximating the actual distribution closely.

3. Why larger models need larger data?
   Assume a model is does not to get sufficient data to learn. As a result, there's a probability that a lot of different hypotheses could fit the data. Regardless of the fact that these hypotheses do not fit the underlying distribution. Therefore, the model fails to perform well when dealing with new data that hasn't been seen before. This indicates that in order for a model to perform better on test data, it must be properly trained with adequate data.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[2] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.