EECS 182/282 Deep Neural Networks **Spring 2023**

Lecture 5: Survey of Problems and Architectures

Lecturer: Anant Sahai *Scribe: Kai He, Nianrong Fang*

1. Recap

1.1 Understood of Simple Linear Models: Local Linearization

We will leverage something we already know to understand other things that we do not know. This is why we need to review simple linear models. The method we use is called *Local Linearization*, which will be the bridge between linear and non-linear content intuitively. Local linearization is a method used to approximate the behavior of a function near a specific point by using its first-order Taylor series expansion. This involves finding the slope (derivative) of the function at the given point and using it to construct a linear equation that approximates the function's behavior in the immediate vicinity of that point.

1.2 Understood of Simple Linear Models: Parameters Learning

Ordinary Least Squares

The most used and simplest case that anchors most of our understanding is the *least squares*, which could be expressed mathematically as follows:

$$\mathbf{X}\overrightarrow{w} \approx \overrightarrow{y}$$

, where $\mathbf{X} \in \mathbb{R}^{x \times d}$, $\overrightarrow{w} \in \mathbb{R}^d$, $\overrightarrow{y} \in \mathbb{R}^n$.

One should recall that we could use *Ordinary Least Squares*(OLS) to learn the weights $\overrightarrow{w}$ by computing

$$\arg\min_{\overrightarrow{w}} ||\overrightarrow{y} - \mathbf{X}\overrightarrow{w}||^2$$

The closed-form solution of OLS is given by:

$$\overrightarrow{w}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\overrightarrow{y}$$

However, this can give us useless values since the model we learned "overfits" the training set with abnormally enormous parameters when there is some noise in the dataset.

Ridge Regressions

In the case of data with Gaussian noise, "explicit regularization" of the parameters during training is more likely to maintain a stable and reasonable system with good generalization capacity of the model. The idea we mentioned above is so called *Ridge Regression*, which is simply adding a regularization (penalty) on the original loss term. The penalty on the parameters can prevent the weights from diverging to unbelievable values, which means abnormally enormous. The mathematical definition of this is as follows:

$$\arg\min_{\overrightarrow{w}} ||\overrightarrow{y} - \mathbf{X}\overrightarrow{w}||^2 + \lambda||\overrightarrow{w}||^2$$

, where λ is a hyperparameter that could be set manually. By increasing the value of λ, we increase the amount of regularization and shrink the regression coefficients toward zero.

The closed-form solution of Ridge Regression is given by:

$$\overrightarrow{w}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^T\overrightarrow{y}$$

This formula is usually called *normal solutions* or *solution of normal equation.*

Probabilistic Justification for Ridge Regression

One probabilistic justification for ridge regression is based on the assumption that the predictor variables are generated from a multivariate normal distribution. Under this assumption, the **maximum likelihood estimates** of the regression coefficients in OLS can be written as:

$$\overrightarrow{w}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\overrightarrow{y}$$

However, when the data points are ill-conditioned, the matrix $\mathbf{X}^T\mathbf{X}$ can be nearly singular or even singular, which leads to unstable estimates of $\overrightarrow{w}_{OLS}$. Ridge regression addresses this problem by adding a regularization term to the OLS objective function, which has the form:

$$\overrightarrow{w}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^T\overrightarrow{y}$$

where λ is a hyperparameter that controls the strength of the regularization, and $\mathbf{I}_n$ is the identity matrix.

The ridge regression solution can be interpreted as the **maximum a posteriori (MAP) estimate** of the regression coefficients, where the prior distribution of the coefficients is assumed to be normal with mean zero. This prior distribution imposes a penalty on the size of the coefficients, which favors solutions with smaller coefficients and hence helps to reduce the effect of multicollinearity and noise.

Kernel Ridge

One should also note that there is another equivalent closed-form solution called *kernel ridge* which is the dual perspective on Ridge Regression. The alternative solution is given by:

$$\overrightarrow{w}_{Rigde} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n)^{-1}\overrightarrow{y}$$

Equivalence Proof:

Proof Scratch:

Plug the full SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ into those two closed-form solutions then simplify the formulas. Note that $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. In the end, one should see the simplified results are identical.

Details of Proof:

For the kernel ridge solution, we have:

$$\begin{aligned}
\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n)^{-1} &= \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T + \lambda\mathbf{I}_n\right)^{-1} \\
&= \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \left(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I}_n\right)^{-1} \\
&= \mathbf{V}\left(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I}_n\right)^{-1} \mathbf{\Sigma}^T\mathbf{U}^T \\
&= \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^T
\end{aligned}$$

For the normal ridge solution, we have:

$$\begin{aligned}
(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^T &= \left(\left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T + \lambda\mathbf{I}_n\right)^{-1} \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T \\
&= \left(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T + \lambda\mathbf{I}_n\right)^{-1} \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= \left(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I}_n\right)^{-1} \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T \\
&= \mathbf{V}\left(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I}_n\right)^{-1} \mathbf{\Sigma}^T\mathbf{U}^T \\
&= \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^T
\end{aligned}$$

, where $\mathbf{\Sigma}^{\dagger} = \left(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I}_n\right)^{-1} \mathbf{\Sigma}^T$ is a diagonal matrix that equals to

$$\begin{bmatrix}
\frac{\sigma_1}{\sigma_1^2+\lambda} & 0 & 0 & \dots & 0 \\
0 & \frac{\sigma_2}{\sigma_2^2+\lambda} & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & \dots & \vdots \\
0 & 0 & \dots & \dots & \frac{\sigma_d}{\sigma_d^2+\lambda}
\end{bmatrix}$$

How to Understand the Kernel Ridge:

We can easily get the normal solution since it is just an OLS-like formula with slight modifications. However, we may feel tough to understand the kernel ridge solution. In fact, we could treat the kernel ridge as a multiplication of matrices and vectors. Recall that the multiplication of matrices and vectors is equivalent to the linear combination of matrices. **This reveals that the parameters our models learned are the linear combination of data points.**

$$\overrightarrow{w}_{Rigde} = \underbrace{\mathbf{X}^T}_{\text{A matrix in } \mathbb{R}^{d\times n}} \underbrace{(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n)^{-1}\overrightarrow{y}}_{\text{A vector in } \mathbb{R}^n}$$

Connection between Kernel Approaches and Kernel Ridge:

To understand the connection, we may want to first focus on $\mathbf{X}\mathbf{X}^T$, which is a matrix in $\mathbb{R}^{n\times n}$ and it is the inner product of input data points.

$$\underbrace{\mathbf{X}\mathbf{X}^T}_{\text{A matrix in } \mathbb{R}^{n\times n}} = [\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle]_{i,j=1}^n$$

So, from a practical use perspective, that's why it's called a kernel ridge. When learning kernel methods (in machine learning courses like CS189), the general idea of the kernel is that anywhere there is an inner product matrix, you should think of it as a **similarity metric** between inputs. One can choose to include any other similarity measure than the Euclidean inner product.

Another reason for it is when we do prediction on the test points, we have to compute:

$$\overrightarrow{w}_{Ridge}^T \overrightarrow{x} = \overrightarrow{x}^T \overrightarrow{w}_{Ridge} = \overrightarrow{x}^T \mathbf{X}^T (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_n)^{-1} \overrightarrow{y}$$

, where $\overrightarrow{x}^T \mathbf{X}^T$ is the test data point of all training points that:

$$\overrightarrow{x}^T \mathbf{X}^T = [\langle \overrightarrow{x}, \overrightarrow{x}_1 \rangle, ..., \langle \overrightarrow{x}, \overrightarrow{x}_n \rangle]$$

This means that we may want to evaluate and predict the test data **based on the similarity of it to all training points**.

2. Implicit Regularization with Induction Bias

Recall the Gradient Descent for OLS, we have

$$\vec{w}_{t+1} = \vec{w}_t + 2\eta \mathbf{X}^T (\vec{y} - \mathbf{X}\vec{w}_t)$$

where $\vec{w}_t$ represents the parameter vector at the current step, and $\vec{w}_{t+1}$ represents the updated parameter vector.

Then we rewrite this equation in SVD coordinates as following

$$\begin{aligned} \tilde{w}_{t+1}[i] &= \tilde{w}_t[i] + 2\eta\sigma_i (\tilde{y}[i] - \sigma_i \tilde{w}_t[i]) \\ &= (1 - 2\eta\sigma_i^2)\tilde{w}_t[i] + 2\eta\sigma_i \tilde{y}[i] \end{aligned}$$

Observing the update equation shows that updating each component of the weights is totally independent, and they don't interact with each other during the gradient descent.

Let the optimal solution is w^*, if we consider the deviation from the optimum, we have

$$\tilde{w}_{t+1}[i] - w^*[i] = (1 - 2\eta\sigma_i^2)(\tilde{w}_t[i] - w^*[i])$$

(the detailed derivation process mentioned in Dis3). It shows that the speed of each component to coverage depends on $1 - 2\eta\sigma_i^2$. So it tells us when we do gradient descent, the inductive bias of gradient descent itself is during the training path to first fit the largest singular direction, which means it first fits the subspace of the largest, then the next, and so on. It implies we are smoothly doing an ordered fit to subspaces. It is like when different data patterns come into and excites the model so that the model resonates, we learn the combination of these patterns and where the model resonates; instead of considering the capacity of models, here we consider the inductive bias of the model, which is another perspective in deep learning.

So gradient descent has an implicit inductive bias that says the path of gradient descent will walk through a sequence of subspaces, from a simple model to a more and more complicated model. Also, different choices for λ also effectively cut off different complexity subspaces.

Besides, note that when σ_i is very small, the stationary point $\tilde{w}_t[i] = \frac{1}{\sigma_i}\tilde{y}[i]$ will go to extremely large. However, in this situation, we can consider the whole update as an integrator of $2\eta\sigma_i\tilde{y}[i]$. Thus gradient descent updates each time with a very small slope even though it will reach an extremely large value eventually. Together with early stopping, it implies that gradient descent is trying to do something like Ridge regularization and that's why gradient descent doesn't go nuts.

What Is the Early Stopping?

Early stopping is in the training stage when we stop the training process because validation performance has gotten worse or has not improved for a long time.

The intuitive implication is since gradient descent prioritizes learning directions with large singular values, learning these directions will improve the accuracy of the model on both the training and test sets. After this, gradient descent may learn directions with smaller singular values (perhaps noise) in the (noisy) dataset. And while learning noise will lead to improved accuracy on the training set, it will cause overfitting. This is reflected by no significant improvement or even a decrease in the accuracy of the test set. The early stop will prevent the gradient descent method from learning a part of the parameters, resulting in a decrease in the mode of the parameters, so the early stop is regarded as a kind of regularization.

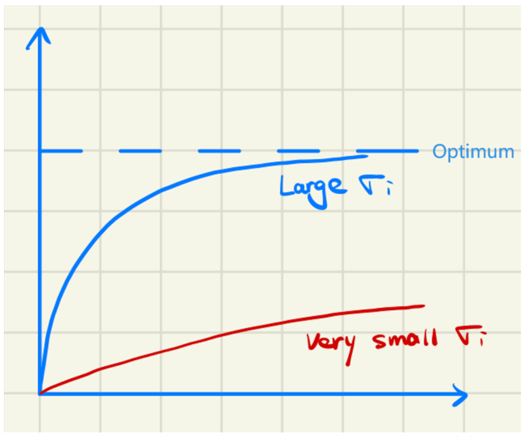

Figure 5.1: Different singular values in GD

3. Two Views of Learned Features

Consider a general model, which could be deep and include nonlinear functions, and we want to learn the parameters $\vec{\Theta}$ using training data $(\vec{x}_i, y_i)$.

$$f_{\vec{\Theta}_0+\Delta\vec{\Theta}}(\vec{x}) \approx f_{\vec{\Theta}_0}(\vec{x}) + \left.\frac{\partial f}{\partial \vec{\Theta}}\right|_{\vec{\Theta}_0} \Delta\vec{\Theta}$$

Specifically, here $\frac{\partial f}{\partial \vec{\Theta}} = \left[\frac{\partial F}{\partial \vec{\Theta}[1]}(\vec{x}), \frac{\partial F}{\partial \vec{\Theta}[2]}(\vec{x}), \ldots, \frac{\partial F}{\partial \vec{\Theta}[d]}(\vec{x})\right]$ which are nonlinear features of $\vec{x}$, then the only learnable parameters in this approximation formula is $\Delta\vec{\Theta}$, which are called locally learnable parameters. So it comes out that the product of lifting of x as a set of features with the weights is a linear combination of the function of $\vec{x}$.

Thus, we have two views of what features we actually learned in the deep models,

- Local features for learning
- Penultimate layer outputs

The first perspective is that we trained a $\vec{\Theta}_0$ as an initial condition as a starting point for fine-tuning. So when we use this model on the new data, the features we've actually learned are the derivatives $\frac{\partial f}{\partial \vec{\Theta}}$.

In another view, if we don't care about the specific layer before the linear layer, we consider the whole deep network from the perspective of a generalized linear model. We have a featurizer, some linear functions of those new features, and a loss that we are optimizing. The featurizer lifts the input $\mathbf{X}$ into the feature space when training. In this perspective, the "learned" features are the outputs of the featurizer, which is the penultimate layer of the whole network. If we write $f = \sum g_i(\vec{x}) w_{i,f}$, where g_i is the output of the penultimate layer and $w_{i,f}$ is the weight of the linear layer, the derivative of it is exactly the strict subset of the local features. Also, the output of the penultimate layer is the derivative since the derivative of something linear is the multiplies of it.

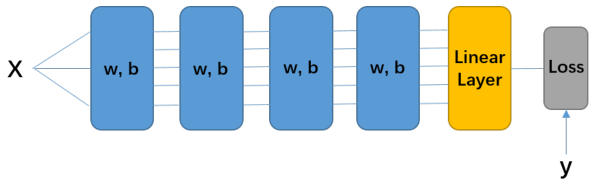

Figure 5.2: Example of a deep network