Ronak Mehta **STAT 535**

Instructor: Marina Meila Autumn 2021

Project Report **Due:** December 15, 2021

---

1 Introduction and Problem Statement

In this project, we apply a machine learning model to predict the capacity of wind turbines capacity given a set of features. To formalize the problem, consider a standard supervised learning setup. Let (X, Y) be a random variable, where X takes values in $\mathbb{R}^d$ and Y takes values in $\mathbb{R}$. Our goal is to produce a function $f : \mathbb{R}^d \to \mathbb{R}$ that minimizes mean squared error risk. That is, to minimize

$$\mathcal{R}(f) = \mathbb{E}\left[(Y - f(X))^2\right]. \tag{1}$$

We are given training sample $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$, from which we estimate the risk as per

$$\hat{\mathcal{R}}(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2. \tag{2}$$

While optimizing $\hat{\mathcal{R}}(\cdot)$ to "train" the predictor f, we hope to find one that generalizes, in the sense of having a small value for $\mathcal{R}(f)$. This setup is instantiated via the the wind turbine dataset, described in Section 2.1.

2 Methods

2.1 Dataset Preprocessing

The given training dataset has $n = 50,000$ instances, with $d = 10$ features. Of the 10 features, 2 are categorical, while 8 are numerical. I used all of them. The following preprocessing steps were taken.

1. **Removal of columns that had a high number of null values.** Only one column, `retrofit_year`, has majority null values. This feature was dropped entirely.
2. **Removal of rows that had null values.** There was only one row with this problem, leaving $49,999$ training instances.
3. **Encoding the categorical variables as integers.** The number of categories for the categorical variables were 42 and 448 respectively. Because this is a large number of categories, we keep them as integers (as opposed to one-hot encoding them) and applying an embedding layer in `PyTorch` to handle them efficiently (see Section 2.2).

4. **Standardizing the numerical features.** Let $x^{(i)}$ be the i-th input instance, with its j-th feature given by $x_j^{(i)}$. Define

$$\overline{x}^{\text{train}} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$$

and

$$s_j^{\text{train}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)} - \overline{x}_j^{\text{train}})^2},$$
$$s^{\text{train}} = (s_1^{\text{train}}, ..., s_d^{\text{train}})^\top.$$

For $x \in \mathbb{R}^d$, the standardization function is defined by

$$\text{Standardize}(x) \equiv \frac{x - \overline{x}^{\text{train}}}{s^{\text{train}}},$$

where the division is applied element-wise. This subtracts the training column means and divides by the training column standard deviations.

5. **Saving the result as a `PyTorch tensor`.** This step is required for the data to be processed by the choice of model.

All steps were applied to the test set as well, noting that parameters of the standardization were estimated from the training set and then applied to the test set. Before training, 10% of the data was held out for validation.

2.2 Model Specification

We apply the neural network model, implemented in `PyTorch`. Specifically, rather than searching over all choices of f, we instead choose a parametrized function class $\{f_\theta : \theta \in \mathbb{R}^p\}$, and optimize over θ. The function f_θ will be fully specified by composing elements from the following common function classes in layers indexed by $l \in \mathbb{N}$, which are briefly reviewed below.

- For $z \in \mathbb{R}^k$, let

$$\text{FullyConnected}_l(z) \equiv \text{ReLU}\left(W_l x + b_l\right),$$

where $W_l \in \mathbb{R}^{d_l \times k}$, $b_l \in \mathbb{R}^{d_l}$, and $\text{ReLU}(x) = \max\{x, 0\}$ is taken element-wise. The parameters W_l and b_l refer to the *weight* and *bias*, respectively, whereas d_l is the *hidden dimension* of layer l.

- For $z \in \{1, ..., v\}$, let

$$\text{Embedding}_l(z) \equiv E_l e_z,$$

where $E_l \in \mathbb{R}^{m_l \times v}$ and $e_z \in \{0, 1\}^v$ is the z-th standard basis vector in $\mathbb{R}^v$. The parameter E_l is the *embedding matrix*, v is the *dictionary size*, and m_l is the *embedding dimension*. Because e_z is a standard basis vector, the operation $z \mapsto E_l e_z$ simply indexes the z-th column of E_l. Therefore, if we think of z is representing one out of v categories, this function associates to the category a vector $\mathbb{R}^{m_l}$.

Now, returning to $x = (x_1, x_2, ..., x_{10})$, the input vector in the given dataset, recalling that x_1 and x_2 are categorical while $x_{3:10} \equiv (x_3, ..., x_{10})$ are real-valued. Given additional parameters $w_{\text{out}} \in \mathbb{R}^{d_L}$ and $b_{\text{out}} \in \mathbb{R}$, the chosen function can be defined by $f_\theta(x) \equiv z_{\text{out}} \in \mathbb{R}$, which is constructed by the following procedure.

$$\begin{aligned} z_0 &= (\text{Embedding}_1(x_1), \text{Embedding}_1(x_2), \text{Standardize}\,(x_{3:10})) \\ z_L &= \text{FullyConnected}_L \circ \cdots \circ \text{FullyConnected}_1(z_0) \\ z_{\text{out}} &= w_{\text{out}}^\top z_L + b_{\text{out}}. \end{aligned}$$

In words, x_1 and x_2 are embedded into $\mathbb{R}^{m_1}$ and $\mathbb{R}^{m_2}$, and then concatenated with the standardized numerical features. Then, the resulting vector passes through L fully connected layers, and is finally projected onto $\mathbb{R}$. The full parameter vector θ is given by

$$\theta = (w_{\text{out}}, b_{\text{out}}, W_L, b_L, ..., W_1, b_1, E_1, E_2).$$

This vector contains the parameters of the linear projection, the weights and biases of each fully connected layer, and the embedding matrices of the two embedding layers. Similarly, $\mathcal{H}_{\text{model}}$ is the hyperparameter vector, given by

$$\mathcal{H}_{\text{model}} = (L, d_1, ..., d_L, m_1, m_2).$$

This vector contains the number of layers, the output dimension of each layer, and the dimension of each embedding. There are also hyperparameters associated to fitting the model, covered in Section 2.3, and selection of these hyperparameters is given in Section 2.4.

2.3 Training

The optimization algorithm used for training the neural network is AdamW, also known as Adam with decoupled weight decay regularization. The algorithm, as well as all of its hyperparameters, is described in detail in Figure 1. The algorithm can be thought of as an adaptive ℓ_2 regularization, where "adaptive" refers to the fact that parameters are rescaled to be in a similar range before applying the regularization constant. The momentum aspect was discussed in class.

In our particular problem, I kept the default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and tuned just the learning rate α and regularization (often called weight decay) parameter λ. The iterations were measured in epochs N, i.e. passes through the training set, where on each iteration a random batch of size $M = 64$ was sampled. Thus, the total number of iterations was approximately $N \cdot \frac{n}{M}$, where N was tuned. Thus, the tunable hyperparameter vector $\mathcal{H}_{\text{train}} = (\alpha, \lambda, N)$.

2.4 Hyperparameter Selection

The combined list of hyperparameters is given by $\mathcal{H} = (\mathcal{H}_{\text{model}}, \mathcal{H}_{\text{train}})$. I attempted $1,000$ choices for $\mathcal{H}$, where each setting was constructed by randomly sampling from the search space for each hyperparameter. The search spaces and their final values are given in Table 1. The best values were chosen by minimum MSE on a validation set of size $5,000$ (10% of the training data). I did not retrain on the entire training set before submitting the model, and the MSE from the best model was my prediction of generalization error.

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

1: **given** $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2: **initialize** time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3: **repeat**
4: $t \leftarrow t + 1$
5: $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
6: $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
7: $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
8: $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9: $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
10: $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
11: $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, or also be used for warm restarts
12: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
13: **until** *stopping criterion is met*
14: **return** optimized parameters $\boldsymbol{\theta}_t$

Figure 1: The objective function f_t changes with iterate t, as it is the empirical risk from Equation 2 estimated on a mini-batch $(x_1, y_1), ..., (x_M, y_M)$ of size M, selected via the "SelectBatch" function. The gradient $\nabla \theta_t$ is computed via backpropagation on the forward pass described in Section 2.2. The vector m_t is an exponential moving average of the gradients, implementing the momentum update. The vector v_t contains an estimate of the "scale" of each parameter, implimenting the RMSProp update, i.e. dividing the gradient for particular parameters by their scale (Line 12 is to be taken element-wise). The vectors $\hat{m}_t$ and $\hat{v}_t$ correct the bias induced by using only a finite average. The learning rate is given by α, while the regularization parameter is λ.

These search spaces are actually constrained from larger search spaces. After running the first iteration of $1,000$ networks, I selected the hyperparameters from the top performing networks and created a smaller search space, which is the one described above. I also attempted dropout regularization, but that significantly decreased performance so it was not included in the final model.

3 Results

The training curves for the model are shown in Figure 2.

The total training time for the model was approximately 48 seconds, while the prediction time on the test set was essentially instantaneous. The final estimate of the generalization error was 5309.48 ± 802.79 (95% confidence interval), and was computed as follows. Let $(x_1, y_1), ..., (x_m, y_m)$ denote the validation set, and let $f_{\hat{\theta}}$ denote the trained predictor. Let

$$\text{err}_i = (y_i - f_{\hat{\theta}}(x_i))^2 \text{ for } i = 1, ..., m.$$

Then, the predicted error estimate was the average of $\text{err}_1, ..., \text{err}_m$, whereas the confidence interval length was given by $1.96 \cdot \frac{s_{\text{err}}}{\sqrt{m}}$, where s_{err} is the standard deviation of $\text{err}_1, ..., \text{err}_m$. In our case, we had $m = 5,000$. Another scale-invariant metric computed was R^2, given by

$$R^2 \equiv \text{percent of explained variance} = 1 - \frac{\text{validation MSE}}{\text{variance of } y_1, ..., y_m} = 0.999,$$

Hyperparameter	Notation	Search Space	Final Value
Number of layers	L	$\{1, 2, 3, 4\}$	2
Hidden dimensions	$d_1, ..., d_L$	$\{4, 8, 16, 32, 64, 128\}$	$(128, 64)$
Embedding dimension 1	m_1	$\{4, 8, 16\}$	4
Embedding dimension 2	m_2	$\{4, 8, 16, 32\}$	8
Number of epochs	N	$\{4, 8, 16, 32\}$	32
Learning rate	α	$\{0.003, 0.01\}$	0.01
Weight decay	λ	$\{0, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1\}$	0.03
Momentum constant	β_1	$\{0.9\}$	0.9
RMSProp constant	β_2	$\{0.999\}$	0.999
Batch size	M	$\{64\}$	64

Table 1: Hyperparameter values.

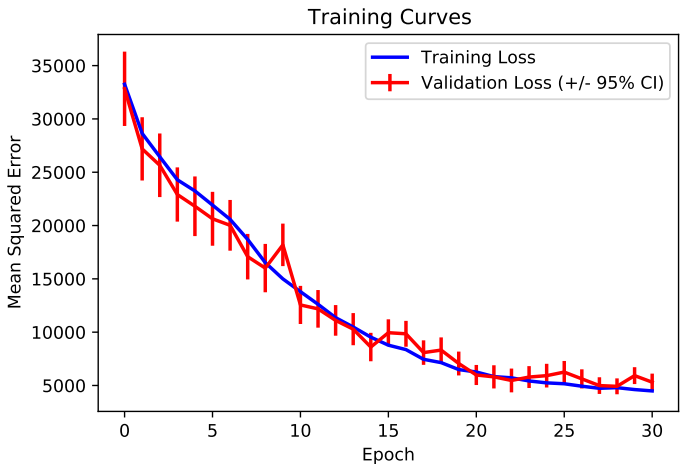

Figure 2: Training curve for neural network model. Even though the objective function is non-convex, the training loss decreased steadily, and generally mirrored the validation loss. The confidence intervals also tend to get smaller as we approach the final epoch.

indicating that the model performed well.

4 Discussion

Overall, this model handles the prediction task well. Areas of improvement include retraining on all of the data after choosing the best hyperparameters, although this might harm the accuracy of the generalization error estimate. Other optimization algorithms such as traditional stochastic gradient descent could have been considered, as well as simpler models such as linear regression or random forest. My particular choice of model was motivated by wanting to learn more about neural network optimization in practice, which was a satisfying result of this project.