When More Parameters Reduce Training Performance: Linear Neural Networks

Anish Lakkapragada

Lynbrook High School
San Jose, CA 95129
alakkapragada176@student.fuhsd.org

Abstract

Linear Regression and neural networks are widely used algorithms for modeling data. Neural networks in particular have the ability to model complex functions beyond lines due to nonlinear activation functions between their subsequent layers. Neural networks without such activation functions just model lines, and thus are not used as they model the same form as linear regression. However, another explanation that we propose in this paper for the impracticality of such neural networks compared to linear regression is that they actually reduce performance. Their excess of parameters make them harder to optimize, and thus they require more training iterations to converge to the optimal solution and are still less likely to do so. We prove this hypothesis through detailing the optimization of both algorithms and then empirically comparing the performance between both models on synthethic, noisy datasets.

Introduction

Neural networks (McCulloch and Pitts 1943) distinguish themselves from linear regression by their ability to model nonlinear data. Their activation functions which transform output from one layer before feeding it to the next give them this unique ability. The common explanation for why not to use neural networks without such activation functions, which we refer to as linear neural networks (LNNs), is that they only can model lines and thus there is no advantage to them compared to a linear regression.

In addition to this, we propose another reason for the impracticality of LNNs is that they actually perform worse than linear regression on the same set of data, despite the fact that they model the same form. Our reasoning is that the excess of parameters in LNNs corrupts the optimization process as, like in all neural networks, currently suboptimal parameters will be used in the calculation for how other current parameters should update to increase the model's performance.

We test our hypothesis through a debrief of optimization procedures on both models and experiments on synthethic datasets of varying noise levels. We detail our reasoning in the next section.

Methods

If we have a univariate dataset X and associated labels y, assuming the relationship between X and y is linear, a linear regression model given by the equation $\hat{y}_i = ax_i + b$ can be created where $\hat{y}_i$ is the ith prediction for the input x_i. In this case, a and b would be the optimal weight and bias respectively to minimize the mean of the squared residuals (they can be solved through a closed-form solution (i.e. normal equation.)

Neural networks for univariate data can similarly be constructed as the following. The output for the first layer z_1 (represented as a vector over the entire dataset) is given by $z_1 = w_1 x + b_1$. In this case, w_n and b_n denote the weight and bias for the nth layer respectively. The output of an LNN with a second layer would then be $w_2 z_1 + b_2$ or $w_2 w_1 x + w_2 b_1 + b_2$. Note that an LNN, regardless of the amount of layers, only creates the form of a line and that an LNN with one layer is a linear regression. For any LNN, the output of the last layer would be the predictions.

LNNs require iterative optimization, namely gradient descent, to be solved. Gradient descent updates each of current parameters based on the derivative of the objective function with respect to that parameter. For any parameter p at time step t, gradient descent will update the parameter as such[1]: $p_{t+1} = p_t - \alpha \frac{dJ}{dp_t}$. In our case, our objective function J to be minimized is the mean squared error (MSE) given by $J = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$. The derivatives used to optimize the linear regression parameters weight m and bias b through such optimization are shown in Equation 1.

$$\frac{dJ}{dm} = \frac{2}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) x_i; \frac{dJ}{db} = \frac{2}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) \quad (1)$$

= LNN optimization to the optimal parameters a, b is more cumbersome because of the increased amount of parameters. For the two-layered neural network given by $w_2 w_1 x + w_2 b_1 + b_2$, the only guaranteed solution for this neural network resemble the optimal model $\hat{y}_i = ax_i + b$ is for $w_2 = a; w_1 = 1; b_1 = 0; b_2 = b$. The derivative of parameter w_2 used to optimize w_2 is given below:

$$\frac{dJ}{dw_2} = \frac{2}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)(w_1 x_i + b_1) \quad (2)$$

[1] α is the learning rate which controls the rate of change of the parameters by their derivatives and is typically from 0.01 to 0.0001.

Similarly, the derivative of the J with respect to b_1 would be the same as above except with w_2 and not z_1. In order for the optimal solution $w_2 = a$ to be met, we would need to have $\frac{dJ}{dw_2}$ be equal to $\frac{dJ}{dm}$ and the current value of w_2 to be the same. This will only ever happen when the current value of w_1 and b_1 are at their optimal values of 1 and 0 respectively. Gradient descent initializes parameters randomly before optimization, so this particular arrangement is extremely unlikely and thus convergence over iterations of all parameters to the ideal solution that best minimizes the objective function (MSE) is unlikely to happen just as fast as linear regression. Through this demonstration, it can be seen how this problem will only be further exacerbated if further layers (and parameters) were added to the LNN.

Empirical Experiments

We test the performance of linear regression compared to LNNs from 2 to 10 layers on synthetic datasets with varying levels of noise.

Data

We detail our procedure to generate synthetic data that has a linear form in this section. For simplicity, all of our data in our experiments are univariate, or one-dimensional. Note that even if our input data was multivariate, the same results would occur as linear regression or LNNs on multivariate data just leads to linear regression on each dimension of the input data.

We first sample the input data vector x from a standard normal distribution. We randomly sample scalars a and b from the same distribution as the respective weight and bias of the data. This gives us y, the label vector, is equal to $ax + b$.

However, because no practical data is perfectly linear we add noise to our dataset. We sample noise from a standard normal distribution and then scale the noise to the size of the pre-existing data by multiplying it by the average of $ax + b$. This scaled noise is then multiplied by a noise coefficient β, which controls the extent of corruption given by the noise. Finally this noise adjusted by magnitude and scaled to the size of the data is added to the pre-existing labels. The full process to generate the noisy label vector y_{noise} is given below in Equation 3.

$$y_{noise} = ax + b + \beta * \mathcal{N}(0, 1) * \mathbb{E}(ax + b) \qquad (3)$$

Note that when the noise is applied, the parameters of the line of best fit changes - we denote the new optimal weight as a^* and bias as b^*.

Results

We compare the performances of a linear regression model (or an LNN with one layer) to LNNs with 2 to 10 layers. For each experiment, using the dataset procedure above, we generate 1000 points for training the model and 200 for which the model is tested on. Both datasets are generated with the same noise coefficient. We first train each model on the the training data for an ample 10,000 iterations. At each iteration, we track the model's MSE on the train and validation datasets and the model's parameters deviation from the optimal weight and intercept.

We calculate this optimal parameter deviation D of the model by first solving the optimal weight and intercept by solving the normal equation (a closed-form solution) on the training data. Because all models are a linear function, we can simplify all models to a linear function $mx + b$ and then measure D as $|m - a^*| + |b - b^*|$. Over the iterations, this deviation should reduce.

We perform this experiment 100 times for each of the noise coefficient values 0.05, 0.15, 0.3, and 0.5. All our models are written in PyTorch, and our models are trained with stochastic gradient descent (SGD) (Robbins and Monro 1951), an iterative optimization method, using a learning rate of 0.001. We report the mean and standard deviations of the MSE (across all 100 experiments) at the end of training for all models and noise coefficients in Table 1. Figure 1 shows the average optimal parameter deviation D throughout training over the 100 experiments with a noise coefficient of 0.05 for each model. We note that the ordering of the individual line plots is the same across all noise coefficients.

Model Type and Number of Layers	Noise Coefficient β = 0.05	0.15	0.30	0.50
LinReg	0.0028 ±0.005	0.0197 ±0.025	0.086449 ±0.1197	0.2840 ±0.4667
LNN-2	0.0033 ±0.006	0.0199 ±0.025	0.086451 ±0.1197	0.2842 ±0.4668
LNN-3	0.004 ±0.007	0.023 ±0.04	0.09 ±0.1194	0.2844 ±0.4665
LNN-4	0.05 ±0.27	0.03 ±0.05	0.101 ±0.13	0.3059 ±0.47
LNN-5	0.08 ±0.28	0.09 ±0.26	0.196 ±0.42	0.358 ±0.61
LNN-6	0.21 ±0.55	0.19 ±0.58	0.26 ±0.59	0.551 ±0.9
LNN-7	0.39 ±0.85	0.40 ±0.98	0.52 ±1.02	0.8172 ±1.32
LNN-8	0.69 ±1.48	0.74 ±1.14	0.61 ±0.87	1.0087 ±1.35
LNN-9	0.87 ±1.27	0.74 ±1.08	0.72 ±1.06	1.0772 ±1.45
LNN-10	0.98 ±1.35	0.90 ±1.33	0.94 ±1.17	1.108 ±1.296

Table 1: Table of the mean and standard deviations of the testing MSE after training across all 100 runs for all different models and noise coefficients. LNN-n resembles a linear neural network with n layers. More significant figures are given when required to draw comparisons.

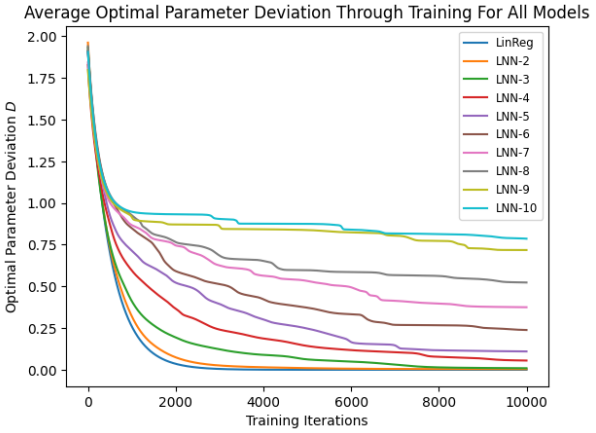

Figure 1: Plot of the average optimal parameter deviation D across all 100 experiments for each model during training.

From the figure above, it is clear that the optimal parameter solution is achieved only by linear regression and an LNN with two or three layers. Models with less layers typically converge to the optimal parameters ($D = 0$) faster. As the amount of layers in the LNN increases, the model typically converges and plateaus at increasingly suboptimal solutions. LNNs with the highest amount of layers are likely never going to converge to the optimal parameters. This is to be expected as the excess of parameters in LNNs compared to linear regression likely leads to local minimas in the objective function; it is these local minimas which cause the model to be stuck in suboptimal parameters instead of the parameters that lead to a global minima in the objective function. In contrast, linear regression has a convex objective function with only one minima - thus being very easy to minimize.

This is also reflected in the testing MSE, where models with the least amount of layers consistently have the lowest average MSE compared to those with more layers.

Conclusion

We first prove the superiority of linear regressions compared to LNNs by a comparison of their optimization. We then validate this proof by testing linear regression models and LNNs on 4 different levels of noise across 100 datasets for each level. We conclude LNNs are unable to perform the same as linear regressions due to being harder to optimize because of their excessive parameters.

References

McCulloch, W. S.; and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.