Impact Factor: 9.9 ISSN-L: 2544-980X

Methods for Reducing Losses in Neural Networks

Abdukadirov Bakhtiyor [1]

Annotation: This article discusses training errors, validation errors, and plotting their correlation that occur when training data in deep neural networks. It is also assumed that the validation error in training deep neural networks is less than the error in training.

Keywords: neural network, network training, loss, mean square error, epoch, regularization.

1. Introduction

Decision making using neural networks is a necessary step towards mastering artificial intelligence [1]. In this aspect, the construction and optimization of the neural network architecture play a fundamental role. The key factors in using a neural network are its training level and the identification error measure (IEM), as well as the number of resources required for the normal functioning of the neural network [2, 3]. However, different approaches to implementing the neural network training process led to completely different values of these three factors, the second of which and partially the third are invariable attributes of the neural network. In this regard, the choice of the neural network training method should be carried out within the framework of coordinating the IEM with the neural network training time, which directly determines the level of its training, and the number of resources consumed, which is not always unambiguously feasible [4].

2. Materials and methods. Training a neural network is a process of minimization in the space of trained parameters of the evaluation function.

The need to use example weights during training may be due to the following reasons:

1) one of the examples is poorly trained;

2) the number of examples of different classes in the training set is very different;

3) the examples in the training set have different reliability [6].

To fix these reasons, you need to follow these methods:

1. Randomly initialize the network weights and biases.

2. Get a bunch of labeled training data (e.g. pictures of cats labeled "cats" and pictures of other things labeled correctly).

3. For each piece of training data, feed it to the network.

4. Check if the network gets it right (given an image labeled "cat", is the network's output also "cat", or is it "dog").

5. If not, how wrong was it? Or how right was it?

6. Tweak the weights slightly to make the network more confident in getting the right answer.

7. Repeat.

3. Literature Review. In the research works conducted by the authors [5, 7-10], various methods of weight optimization aimed at reducing training losses were analyzed. In the work [2], the author indicated that one of the key aspects of weight adjustment is the choice of optimization algorithm, the most common of which are: SGD (Stochastic Gradient Descent), Adam and RMSprop.

[1] Бухарский государственный медицинский институт, Бухара, Узбекистан

Suppose we are training the network to distinguish between cats and dogs. Therefore, we only need two output neurons - one for each classification. We feed the image of a cat to the network. In the meantime, imagine that each pixel of the image corresponds to one "input". Here, the probability is assigned 62% that the image depicts a dog, and 38% that it is a cat. Ideally, we want to say that this image is 100% a cat.

So we go backwards through the network, increasing the weights and biases to increase the likelihood that the network will classify it as a cat.

How do we know how wrong the network is? We measure the difference between the network's output and the correct output using a "loss function".

The best loss function will depend on your data and your intended application. A simple example of a loss function would be mean squared error - this is what, to fit a line to a data point, you are trying to minimize the square of the distance between each of the points and the line:

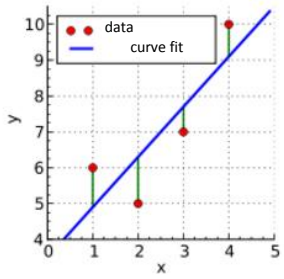

Fig. 1. Graph of the function for finding the mean square error

The loss function does the same thing, but in many more dimensions. They can be mathematically complex, but it is useful to think of it as the difference between what the network outputs and what it should output.

The error is a percentage value that reflects the discrepancy between the expected and the received answers. The error is generated every epoch and should decrease. If it does not, then you are doing something wrong.

The loss function is at the center of a neural network. It is used to calculate the error between the actual and received answers. Our main goal is to minimize this error. So the loss function effectively moves the training of the neural network closer to this goal. The loss function measures “how good” the neural network is with respect to the given training sample and the expected answers. It can also depend on variables such as weights and biases.

The loss function is one-dimensional and not a vector, since it estimates how well the neural network is performing overall. At the most basic level, the loss function determines how “good” or “bad” the predictor data is when classifying input data points in a dataset. The smaller the loss, the better the classifier performs in modeling the relationship between the input data and the output targets. However, there is a point where we can overfit our model - by modeling the training data too closely, our model loses its ability to generalize.

To do this we need:

1. Reduce our losses, thereby improving the accuracy of the model.

2. We need to do this as quickly as possible and with minimal hyperparameter updates.

3. All without overloading our network or overfitting the training data.

This is a balancing act, and our choice of loss function and model optimizer can have a significant impact on the quality, accuracy, and generality of our final model.

Typical loss functions, also called "objective functions" or "evaluation functions," include:

- Binary cross-entropy

- Categorical cross-entropy
- Sparse categorical cross-entropy
- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)

4. Analysis and results. The main goal when training a neural network is to find suitable weights and biases of the neural network in which there will be minimal values of the loss function. We can plot the loss as a function of weight. To do this accurately, we need to be able to visualize many dimensions to account for the many weights and biases in the network. Since it is difficult for us to visualize more than three dimensions, let's pretend we only need to find two weight values. We can then use the third dimension for the loss. Before training the network, the weights and biases are initialized randomly, so the loss function will likely be high since the network will make a lot of mistakes. Our goal is to find the lowest point of the loss function and then see which weight values it corresponds to.

So how do we help the network find the lowest point? To find the minimum point, the Gradient Descent method is used.

At first, the loss function will be high, and the network will make incorrect predictions. As the weights are adjusted and the loss function is reduced, the network will get better at outputting correct answers.

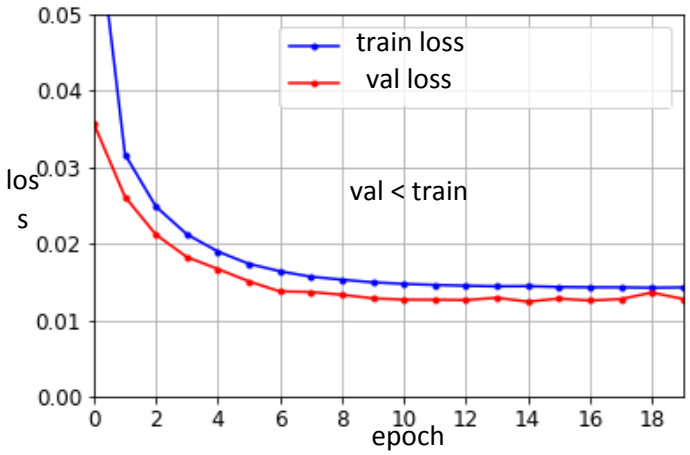

Fig. 2. Graph of the loss rate during testing and training

The most common reason is regularization (e.g. dropout), as it is applied during training but not during validation and testing. If we add the regularization loss to the validation loss, things will look different.

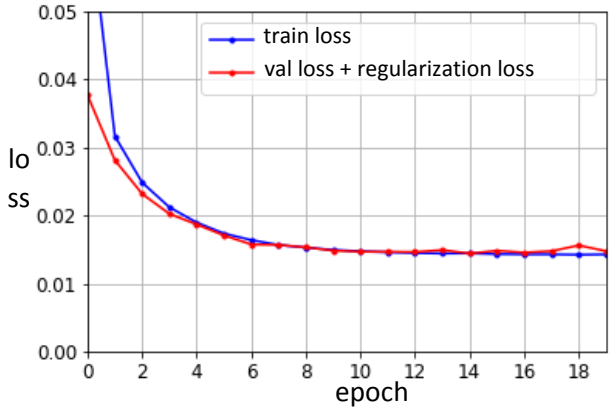

Fig. 3. Plot of the proportions of testing loss, regularization loss, and training loss

The first reason is that regularization is applied during training, but not during validation/testing. When training a deep neural network, we often apply regularization to help our model:

1. To get higher validation/validation accuracy

2. And ideally to better generalize the data beyond the validation and testing sets

Regularization methods often sacrifice training accuracy to improve validation/testing accuracy - in some cases, this can result in your validation loss being lower than your training loss.

Second, regularization methods like dropout are not applied during validation/testing.

As shown in Figure 6, accounting for regularization before validation loss (e.g. applying dropout during validation/testing) can make your training/validation loss curves more similar.

Oh, and training loss is measured during each epoch, while validation loss is validated after each epoch, so on average, training loss is measured 0.5 epochs earlier. If we shift it 0.5 epochs to the left, things will look different again.

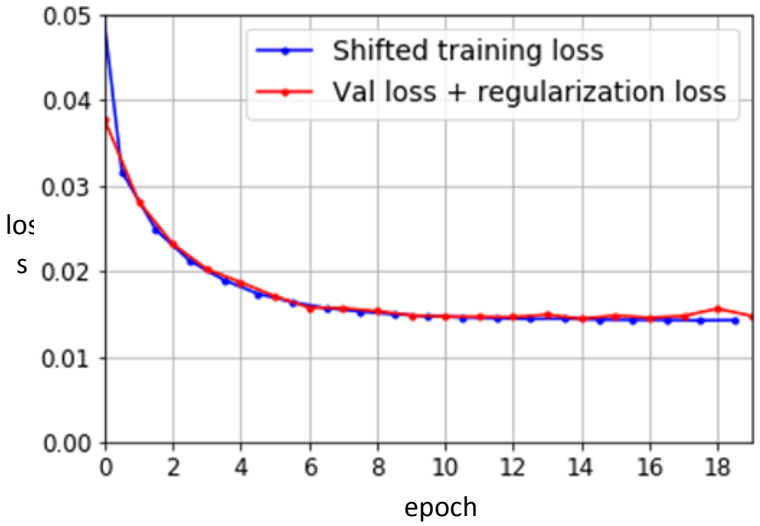

Fig. 4. Plot of the ratio of biased learning loss to regularization loss

The second reason you might see validation loss lower than training loss has to do with how the loss value is measured and reported:

1. Training loss is measured during each epoch
2. While validation loss is measured after each epoch

Our training loss is reported continuously throughout the epoch; however, the validation metrics are only computed on the validation set after the current training epoch has finished.

This means that on average, the training loss is measured half an epoch earlier.

If we shift the training loss half an epoch to the left, we can see that the gaps between the training and loss values are much smaller. This may also indicate leakage from the test data to the training data, so be careful here.

Even if the validation loss is close to the training loss, our model may still be overfitting. When comparing, we need to take into account the regularization loss, shift the training loss half an epoch, and make sure the validation set is large, drawn from the same distribution as the training, and without leakage.

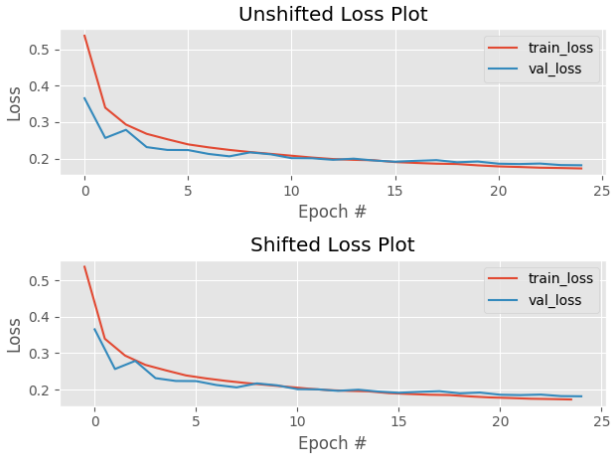

Fig. 5. Unmodified loss graphs (a), shift of the loss graph during training by 1/2 epoch to the left (b)

As you can see, shifting the training loss values to the left by half a period (bottom) makes the training/validation curves more similar compared to the unshifted plot (top).

Or maybe the val set is simpler than the training set. This can happen by chance if the val set is too small or if it was not chosen properly, e.g. too many simple classes. Or the training set has leaked into the val set. Or you are using data augmentation during training.

Conclusion. Regularization is applied during training, but not during validation or testing. If you add regularization loss during validation or testing, your loss values and curves will look more similar. Training loss is measured during each epoch, and validation loss is measured after each epoch. On average, training loss is measured half an epoch earlier. If you shift the training loss curve half an epoch to the left, your losses will align a little better. Your validation set may be simpler than your training set, or your code may be leaking your data. Make sure your validation set is large enough and sampled from the same distribution as your training set. You may be over-regularizing your model. Try reducing the regularization constraints, including increasing the capacity of the model i.e. making it deeper with more parameters, reducing dropout, reducing the strength of weight decay, etc.

References

1. Russell, S. Artificial Intelligence: A Modern Approach (AIMA) / S. Russell, P. Norvig. - [2nd ed.]. - M .: Williams Publishing House, 2007. p.1408
2. Khaikin, S. Neural networks: a complete course / S. Khaikin. - [2nd ed.]. - M .: Publishing house "Williams", 2006. p.1104.
3. Zuev, V.N., Kemaikin V.K. Modified algorithm for training neural networks // Software products and systems. 2019. T. 32. № 2. pp. 258-262.
4. Aurélien, J. Sometimes the verification loss is lower than the training loss. [Electronic resource]. Access mode: https: // twitter.com / aureliengeron / status /1110839223878184960
5. Abdukadirov, B. Methods for detecting video attacks in biometric systems // Current state of the pharmaceutical industry: problems and prospects, Proceedings of the international scientific and practical conference, Tashkent Pharmaceutical Institute – Tashkent. 2021. November 18-19. pp. 443-444.
6. Norinov M., Abdukadirov B., Gofurov M. Application of Fourier Methods and Discrete-Cosinus Transformation in the Process of Processing of TV Images // International Journal of Innovative Technology and Exploring Engineering. – 2019. – Vol. 8, Issue 9S3. – Pp. 1565-1568.
7. Niyozmatova N., Mamatov N., Samijonov A., Abdukadirov B., Abdullayeva B. Algorithm for determining the coefficients of the interpolation polynomial of Newton with separated differences // IOP Conference Series: Materials Science and Engineering. – 2020. – Vol. 862, Issue 042019. – Pp. 1-4.

8. Fazilov Sh., Radjabov S., Abdukadirov B. The problem of detecting fake access in biometric identification systems. Descendants of Muhammad al-Khwarizmi, 3(13): pp.16-23, Tashkent, 2020.
9. Abdukadirov B.A., Khashimov A.A., Nurilloev I.F. Tojiboeva Sh.Kh., Mamatov A.A. Approach to detecting false inputs in facial recognition systems // Uzbekistan Journal of Computer Science and Energy Problems. – 2020. – №3. –pp. 73-82.
10. Abdukadirov B. Methods for detecting false inputs in biometric systems // Scientific and Technical Journal of Namangan Institute of Engineering and Technology. – 2021. – Vol. 6, Issue 3. – Pp. 208-214.