Shivali Choudhary, Dikshant Jain, Emmanuel Gallegos
Computer Vision
CSU East Bay
29 October 2020

# Exercise Explore: Overtraining and the Importance of Checkpoints

## What is Overtraining?

"In [the case of overfitting], rather than 'learning' to generalize from a trend, an overfitting model may 'memorize' non-predictive features of the training data" (Bilbao). Essentially, overfitting or overtraining occurs when a supervised classifier becomes too closely attuned to a training dataset, and loses the generality that is sought when constructing classifiers. In the case of overtraining, while the training error might approach zero, the validation error will not, and may possibly begin to increase instead. This signifies that the classifier will perform poorly when exposed to unseen data because it is too attuned to classifying the features of the training set which may not be representative of the true sample possibility space.

In the case of ANN's, often the term 'overtraining' is used as opposed to overfitting, though the basic underlying concept between the two terms is similar. The model, whether it is a complex neural network, or a simple polynomial regression, is fitted to minimize training error at the cost of potential increased complexity and poorer validation error. The ultimate result is a model that is less robust.

To visualize the concept of overfitting, examine the following regression diagrams:

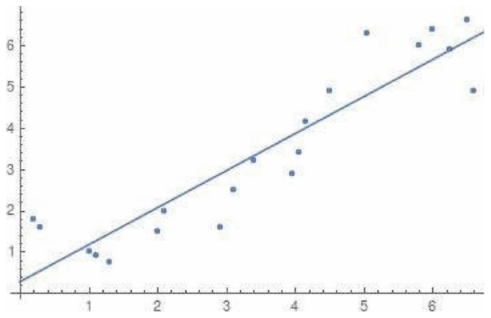

Fig 1: 1st Order Polynomial

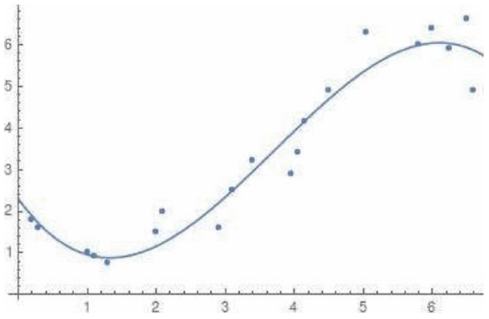

Fig 2: 3rd Order Polynomial

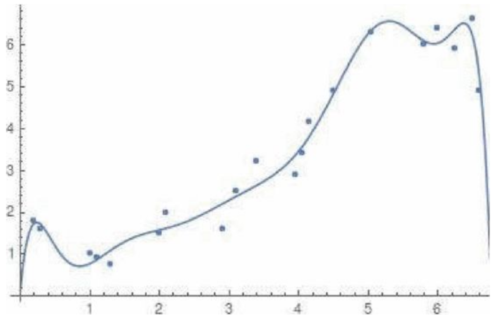

Fig 3: 6th Order Polynomial

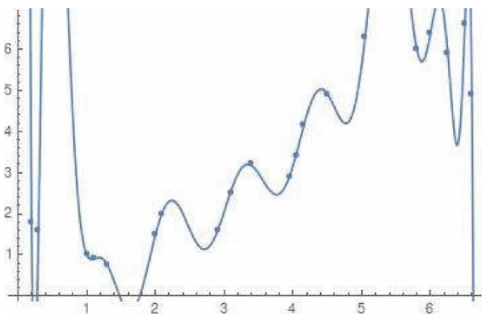

Fig 4: 20th Order Polynomial

In these diagrams, from Bilbao et al, we see examples of a polynomial regression on a training dataset with various order polynomial functions. The objective of the regression is to return a model, in this case a polynomial function, that can be used to predict the value of a new data point. As the order of the polynomial function increases, we can see that the model begins to more tightly hug the training data set and the training error should consequently go down as the complexity increases. However, looking in particular at the 20th order polynomial in figure 4, we can see an example of extreme overfitting, where the training error has approached zero, but the resulting model is extremely complex and will likely lead to high test errors when exposed to data not a part of the original training set.

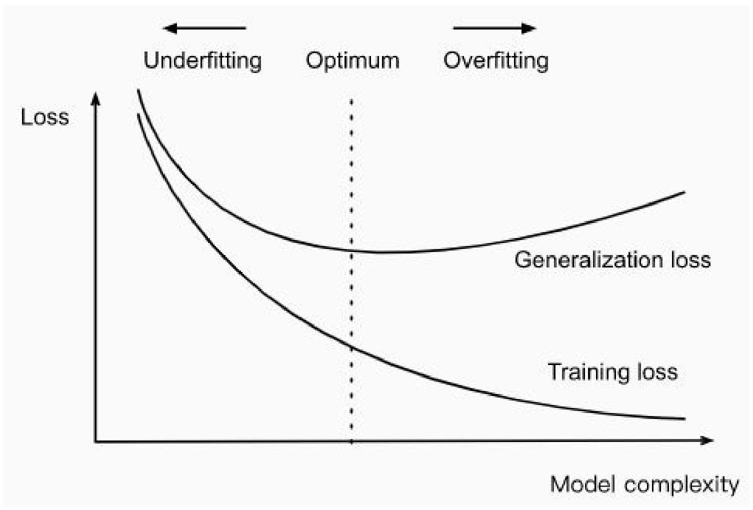

Fig 5: Underfitting vs Overfitting

Figure 5, from *Dive Into Deep Learning*, illustrates an example of under and over fitting. In this case, the figure refers to another regression based model, and thus the x-axis is labelled model complexity. For an object detection neural network, this axis might instead be labelled “time”, “steps”, or “epochs”, depending on the training schema. As the model is trained, eventually its generalization / validation / test loss will begin to increase at a certain point, while the training loss continues to decrease. Beyond this point, the model is considered overfit or overtrained on the dataset. Before that point, it is underfit or undertrained. This gives some intuition into why checkpoints are so important in training, as after training, one can look at validation loss during training and find a prior checkpoint closest to the ‘optimum’ location, where the derivative of loss becomes positive. But we will elaborate more on this later.

Unless the dataset is literally a perfect representation of the variety that is contained in the sample possibility space, the model will instead become attuned to the unique features of the sample data and lose the ability to generalize its predictions. While one should always try to train on a large enough dataset that appropriately represents the possibility space, overfitting is still a danger.

**How to identify if overtraining has occurred: 2 (shivali)**

During training, the model is evaluated on a holdout validation dataset after each epoch. If the performance of the model on the validation dataset starts to degrade (e.g. loss begins to increase or accuracy begins to decrease), then the training process is stopped.

**Monitoring Performance**
The performance of the model must be monitored during training.
This requires the choice of a dataset that is used to evaluate the model and a metric used to evaluate the model.
It is common to split the training dataset and use a subset, such as 30%, as a validation dataset used to monitor performance of the model during training. This validation set is not used to train the model. It is also common to use the loss on a validation dataset as the metric to monitor, although you may also use prediction error in the case of regression, or accuracy in the case of classification.
The loss of the model on the training dataset will also be available as part of the training procedure, and additional metrics may also be calculated and monitored on the training dataset.
Performance of the model is evaluated on the validation set at the end of each epoch, which adds an additional computational cost during training. This can be reduced by evaluating the model less frequently, such as every 2, 5, or 10 training epochs.
**Early Stopping Trigger**
Once a scheme for evaluating the model is selected, a trigger for stopping the training process must be chosen.
The trigger will use a monitored performance metric to decide when to stop training. This is often the performance of the model on the holdout dataset, such as the loss.
In the simplest case, training is stopped as soon as the performance on the validation dataset decreases as compared to the performance on the validation dataset at the prior training epoch (e.g. an increase in loss).
More elaborate triggers may be required in practice. This is because the training of a neural network is stochastic and can be noisy. Plotted on a graph, the performance of a model on a validation dataset may go up and down many times. This means that the first sign of overfitting may not be a good place to stop training.

**How to avoid overtraining: 2 (dikshant)**

- **Use a Train/Validation/Test Partition**
  If there is an ample amount of data available, the data can be partitioned into three sets. The training set is used to train the model. The model is tested on the validation set during training to determine error on unseen data. That is, data that was not used to train the model. This method will save the model at the point where validation error is lowest. Finally, the test set is used to verify generalization performance of the model.
- **Regularization**
  Adding a penalty training error depending on the magnitude of weights. Larger magnitude weights may be a sign of overtraining where the network is trying to learn noisy features of the training data set. Regularization steers the model toward smaller weights while minimizing error. The lambda constant is a multiplier on the penalty.

L1: total error = training error + $\lambda\Sigma|\text{weight}|$
L2: total error = training error + $\lambda\Sigma\ \text{weight}^2$

- **Bagging**
  Bagging (bootstrap aggregating) is an ensemble method that combines multiple models into a meta-model. For each model, the training data is randomly sampled with replacement from all data flagged as training. This means that some data points can be represented in the "bag" more than once for a model. Other training data may not be represented at all in a model. This sampling method creates different training sets for each model which increases model diversity and helps avoid overtraining.
- **Early Stopping with Cross-validation**
  Early stopping could be used with k-fold cross-validation, although it is not recommended.The k-fold cross-validation procedure is designed to estimate the generalization error of a model by repeatedly refitting and evaluating it on different subsets of a dataset. Early stopping is designed to monitor the generalization error of one model and stop training when generalization error begins to degrade.

  They are at odds because cross-validation assumes you don't know the generalization error and early stopping is trying to give you the best model based on knowledge of generalization error.

  It may be desirable to use cross-validation to estimate the performance of models with different hyperparameter values, such as learning rate or network structure, whilst also using early stopping.

  In this case, if you have the resources to repeatedly evaluate the performance of the model, then perhaps the number of training epochs may also be treated as a hyperparameter to be optimized, instead of using early stopping. Instead of using cross-validation with early stopping, early stopping may be used directly without repeated evaluation when evaluating different hyperparameter values for the model (e.g. different learning rates). One possible point of confusion is that early stopping is sometimes referred to as "cross-validated training." Further, research into early stopping that compares triggers may use cross-validation to compare the impact of different triggers.

**Why checkpoints are important: (shivali)**
Checkpointing is a technique that provides fault tolerance for computing systems. It basically consists of saving a snapshot of the application's state, so that applications can restart from that point in case of failure. This is particularly important for the long running applications that are executed in the failure-prone computing systems.

What is Checkpoint
1. The architecture of the model, allowing you to re-create the model
2. The weights of the model

3. The training configuration (loss, optimizer, epochs, and other meta-information)
4. The state of the optimizer, allowing you to resume training exactly where you left off.

Need of Checkpoint

- A good use of checkpointing is to output the model weights each time an improvement is observed during training.
- **Resilience**: If you are training for a very long time, or doing distributed training on many machines, the likelihood of machine failure increases. If a machine fails, TensorFlow can resume from the last saved checkpoint instead of having to start from scratch. This behavior is automatic — TensorFlow looks for checkpoints and resumes from the last checkpoint.
- **Generalization**: In general, the longer you train, the lower the loss on the training dataset. However, at some point, the error on the held-out, evaluation dataset might stop decreasing. If you have a very large model, and you are not doing sufficient regularization, the error on the evaluation dataset might even start to increase. If this happens, it can be helpful to go back and export the model that had the best validation error. This is also called early stopping because you could stop if you see the validation error start to increase. (A better idea is, of course, to decrease model complexity or increase the regularization so that this scenario doesn't happen). The only way you can go back to the best validation error or do early stopping is if you have been periodically evaluating and checkpointing the model.
- **Tunability**: In a well-behaved training loop, gradient descent behaves such that you get to the neighborhood of the optimal error quickly on the basis of the majority of your data and then slowly converge towards the lowest error by optimizing on the corner cases. Now, imagine that you need to periodically retrain the model on fresh data. You typically want to emphasize the fresh data, not the corner cases from last month.

**Citations:**

I. Bilbao and J. Bilbao, "Overfitting problem and the over-training in the era of data: Particularly for Artificial Neural Networks," 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, 2017, pp. 173-177, doi: 10.1109/INTELCIS.2017.8260032.

Zhang, A., Lipton, Z., Li, M., & Smola, A. (2020). 4.4 Model Selection, Underfitting, and Overfitting. In *Dive Into Deep Learning* (0.15.0 ed.).

https://ieeexplore-ieee-org.proxylib.csueastbay.edu/stamp/stamp.jsp?tp=&arnumber=488180

https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

https://vortarus.com/machine-learning-overtraining/

https://blog.floydhub.com/checkpointing-tutorial-for-tensorflow-keras-and-pytorch/#:~:text=Checkpoints%20in%20machine%20learning%20and,from%20where%20you%20left%20off.

https://machinelearningmastery.com/check-point-deep-learning-models-keras/
https://towardsdatascience.com/ml-design-pattern-2-checkpoints-e6ca25a4c5fe