Summary

sagisk

February 2, 2022

1 Chapter 4: Teaching Deep Learners to Generalize

- Because of the big number of parameters the NNs are prone to overfitting. - One thing to keep the eye on is the Bias-Variance tradeoff. - Some signs of overfitting are:

- For a model that is trained on different datasets the same test instance obtains drastically different predictions.
- The gap between the error of predicting trainining instances and test instances is rather large.

- Key methods to avoid overfitting:

- Penalty-based regularization: apply some constraints on the parameters
- Generic and tailored ensemble methods: bagging, subsampling, Dropout
- Early stopping: terminate training when error on the held-out data begins to rise.
- Pretraining: The weights in different layers are trained sequentially in a greedy fashion and then are used as a good starting points for the process of learning.
- Continuation and curriculum methods: Train simple models and provide their optimum points as a good initialization for a more complex models (which is related to the simpler model).
- Sharing parameters with domain-specific insights: Set some of the parameters in different pars of NN to the same value (like in cnn, rnn).

- Some of the above methods have similar effect as if adding some noise to the input or hidden layers. - In practise it is better to build complex models that can capture more features of data and apply regularization to overcame overfitting than to use simple models. - Supervised settings tend to be more prone to overfitting than unsupervised. Though in both cases regularization can/should be applied.

1.1 The Bias-Variance Trade-Off

The goal of the bias-variance trade-off is to quantify the expected error of the learning algorithm in terms of bias, variance and noise (data-specific).

1. Bias: It is the error caused by the simplifying assumptions in the model. No matter how big training instances we have certain test instances will have a consistent errors for our model. 2. Variance: Is the inability to learn all model's parameters in a statistically robust way especially given limited amount of data and large amount of parameters. 3. Noise: Is the irreducible error of the data.

Assume that the target is a numeric variable to capture the error by mean-squared error (MSE).

- $Z_i, i \in [1, t]$ be our feature variables
- Suppose $y_i = f(Z_i) + \epsilon_i, f$ is our unknown function.
- $g(Z_i, D)$ be our model for example in linear regression $g(Z_i, D) = WZ_i$.
- Let $\hat{y_i} = g(Z_i, D)$ be our model prediction.

MSE $= \frac{1}{t}\sum_{i=1}^{t}(\hat{y_i} - y_i)^2 = \frac{1}{t}\sum_{i=1}^{y}(g(Z_i, D) - f(Z_i) - \epsilon_i)^2$

Then,

$E[\text{MSE}] = \frac{1}{t}\sum_{i=1}^{t} E[(g(Z_i, D) - f(Z_i) - \epsilon_i)^2] = \frac{1}{t}\sum_{i=1}^{t} E[g(Z_i, D) - f(Z_i)^2] + \frac{\sum_{i=1}^{t} E[\epsilon_i^2]}{t} = \frac{1}{t}\sum_{i=1}^{t} E[g(Z_i, D) - E[g(Z_i, D)] + E[g(Z_i, D)] - f(Z_i)^2] + \frac{\sum_{i=1}^{t} E[\epsilon_i^2]}{t} = \frac{1}{t}\sum_{i=1}^{t} E[(f(Z_i) - E[g(Z_i, D))^2] + \frac{2}{t}\sum_{i=1}^{t}(f(Z_i) - E[g(Z_i, D)])(E[g(Z_i, D)] - E[g(Z_i, D)]) + \frac{1}{t}\sum_{i=1}^{t} E[(E[g(Z_i, D)] - g(Z_i, D))^2] + \frac{\sum_{i=1}^{t} E[\epsilon_i^2]}{t} = \frac{1}{t}\sum_{i=1}^{t}(f(Z_i) - E[g(Z_i, D))^2 + \frac{1}{t}\sum_{i=1}^{t} E[(E[g(Z_i, D)] - g(Z_i, D))^2] + \frac{\sum_{i=1}^{t} E[\epsilon_i^2]}{t} = \text{bias}^2 + \text{variance} + \text{noise}$

1.2 Generalization in Model Tuning and Evaluation

We should always divide the available data (usually with ration 2:2:1) into: 1. Training data: Here one can use different model designs, different hyperparameters and do the model selection. However, the actual evaluation of those models will be done on the validation set. 2. Validation data: Held-out data used for model selection and parameter tuning. For example, multiple models with various learning rates can be constructed on the training data and then the results will be evaluated on the validation data to determine the best learning rate. However, one should be cautios to not capture the noise in the validation data i.e. one can overfit even on the validation data. 3. Testing data: The testing data are used only once at the very end of the process.

Methods to divide the data:

- Hold-out: A fraction of data is used for training model, and the remaining for testing.
    1. Weaknesses: Can be that held-out examples have a higher presence of a certain class than the training data. Then the understimation of the true accuracy can occure.
    2. Strengths: Easy, fast to implement. Usually, the choice to go with in NN because of the big number of available data.
- Cross-validation: Divide the data into q equal segments where one of the q - segments is used for testing and the remaining for the training.
    1. Weaknesses: Takes lots of computational resources.

2 Methods to avoid overfitting

2.1 1) Penalty-Based Regularization

1. Penalize parameters:

- L_2 loss: Also called Tikhonov regularization. We define the regularized loss as $L(w) + \lambda\|w\|^2$ where $\lambda > 0$ is the regularization parameter. λ can be tuned by the model validation. The effect of introducing L_1 regularization is equivalent to introducing an equal amount of Gaussian noise to the single-layer NN with an identity activation. (See below*)
- L_1 loss: He we define the regularized loss as $L(w) + \lambda\|w\|_1 = L(w) + \lambda\sum_{i=0}^{d}|w_i|$. Since this function is not differentiable at 0 one can use the idea of sub-gradients to overcome this problem. Another way is to make an approximation to the sub-gradients by setting the weight update to be $w_i = w_i - \alpha\lambda s_i - \alpha\frac{dL}{dw_i}$. Where s_i is the partial derivative of $|w_i|$ defined to be -1 if $w_i < 0$, 1 if $w_i > 0$ and in the rear occasions when $w_i = 0$ to be 0.

1. Penalizing the activations: Imply L_1 penalty on the hidden units so that the original loss L becomes $L' = L + \lambda\sum_{i=1}^{M}|h_i|$ where h_i is the value of the i-th hidden unit.
    - Backpropagation becomes: $\frac{dL}{da_{h_r}} = \delta(h_r, N(h_r)) = \Phi'(a_{h_r})\sum_{h:h_r\to h}\delta(h, N(h))$ Immediately after making this update the value is adjusted for regularization: $\delta(h_r, N(h_r)) = \delta(h_r, N(h_r)) + \lambda\Phi'(a_{h_r})\text{sign}(h_r)$

L_1 vs L_2: In practise L_2 given better results than L_1. However, L_1 creates sparse solutions.

Connection of L_2 to noise*: Suppose, we change training case from (X, y) to $(X + \sqrt{\lambda}\epsilon, y)$ where $\epsilon = N(0, 1)$ then $\hat{y} = W(X + \sqrt{\lambda}\epsilon) = WX + \sqrt{\lambda}W\epsilon$.

$E[L] = E[(y - \hat{y})^2] = E[(y - WX - \sqrt{\lambda}W\epsilon)^2] = (y - WX)^2 - 2\sqrt{\lambda}E[W\epsilon](y - WX) + [(W\epsilon)^2] = (y - WX)^2 + \lambda\sum_{i=1}^{d} w_i^2$

2.2 2) Ensemble Methods

The idea is that in ideal world one has an infinite supply of data and one can create numerous training data sets to predict the same test instance using these data sets. However, in our ordinary world we don't have that privilage but we have two main methods to imitate that kind of behavour.

- Bagging: The training data is sampled **with replacement** m times. The sample size $s \leq n$ the number of data points. The predictions from different models are averaged.
- Subsampling: The training data is sampled **without replacement**. Is vital to take $s < n$.

Bagging vs Subsampling: Bagging is preferrable with limited data. Othrewise subsampling.
Dropping:

- Randomized Connection Dropping: Random connections between different layers are dropped. Different models are trained this way and the predictions are averaged. Note that unlike Dropout different models do not share their weights.
- Dropout: Uses node sampling instead of edge sampling. During dropout a different model is used for each mini-batch. So, a thousand NNs are sampled with shared weight and a tiny training data is used to update weights in each case. So, the main difference is that it shares weights between models.
    1. Sample a neural network from the base network. The input nodes are each sampled with probability p_i, and the hidden nodes are each sampled with probability p_h. Furthermore, all samples are independent of one another. When a node is removed from the network, all its incident edges are removed as well.
    2. Sample a single training instance or a mini-batch of training instances.
    3. Update the weights of the retained edges in the network using backpropagation on the sampled training instance or the mini-batch of training instances.
    4.

- Peculiarties: Dropout assumes that the outputs are in the for of probability. So, one should compute the geometric mean of the outputs and normalize them to get the predictions. Dropout prevents a phenomenon referred to as feature co-adaptation. Dropout can be considered an ensemble that adds noise to the data in an indirect way. - Weaknesses: since Dropout is a regularization method, it reduces the expressive power of the network. Therefore, one needs to use larger models and more units in order to gain the full advantages of Dropout.

2.3 3) Early stopping

The training is applied on the training data and at the same time the error of the model on the validation set is continuously monitored. When the error on the validation set starts to increase we stop the training even if the training error is still decreasing. It is important to keep the history of the learning process because one should not terminate immediately when the increase of error is detected in the validation set since it can be followed by a decrease.

2.4 4) Unsupervised Pretraining

A greedy approach is used to train the network one layer at a time by learning the weights of the outer hidden layers first and then learning the weights of the inner hidden layers. The resulting weights are used for a final phase of traditional NN backpropagation to fine-tune it. - Unsupervised pretraining: Example on autoencoders: Treat the outer hidden layers as a first-level reduced representation. Treat the inner hidden layer as a second-level reduced representation. 1. Learn the first-level reduced representation by training smaller NN with two hidden layers collapsed into one. The assumption is that the two outer hidden layers are symmetric. 2. Use first-level representation weights to learn the second-layer reduced representation in a similar fashion. 3. At the end the set of weights is used to train the entire NN.

- Variations of unsupervised pretraining:
  1. One can train multiple layers at one time instead of pretraining one layer at a time.
     - Advantages: Leads to more powerful initializations.
     - Disadvantages: Grouping too many layers can cause vanishing and exploding gradients.
  2. One can soften the restrictive symmetry assumption (for example sigmoid activations with only non-negative values in the encoder layer and tanh activations with both positive/negative values in the decoder) between hidden layers in encoder, decoder. So, the corresponding reductions between encoder-decoder can be different. To implement this one should add an additional layer of weigth between two layers during pretraining but after it one should use only the weights of encoder-decoder and discarding the additional layer weights.
     - Advantages: Adds flexibility during pretraining.
     - Disadvantages:

- Supervised pretraining: 1. The final output layer is removed and the representation of the final hidden layer is learned in an unsupervised way. So if the original NN has k hidden layers we add additional $(k-1)$ layers to build an autoencoder with $2k-1$ hidden layers and with final hidden layer as the middle layer. 2. Apply unsupervised pretraining on this autoencoder. 3. Use the weights of **only encoder portion** as initial weights for the original hidden layers. 4. Re-attach the output nodes to the final hidden layer. 5. Use the weights to train the original NN.

- Variations of supervised pretraining: No autoencoder is used here. Instead in the first iteration there are just the first hidden layer with the outputs and the weight connected them are learnt. In the second iteration the second hidden layers adds up. The first hidden layer is now treated as an input layer with its inputs as the transformed representations of the training points learned in the previous iteration. And so on.
  1. Advantages:
  2. Disadvantages: Does not work as good as unsupervised pretraining in many cases but not always.

2.5 5) Continuation and Curriculum Learning

The idea is that it is easier to optimize easy problems than to work with complex ones. So, instead of trying to deal with complex problem in one shot, one can build up to the solution gradually: from simple to complex. Two such methods are:

1. Continuation learning: Model-centric approach. One start with a simplified version of the problem (for example, a smooth loss function with single global optimum) and then continues to work on more complex refinement of the problem, updating the solutions (more complex loss functions). Let $L_1, ..., L_r$ be our loss functions we can think of L_i to be a smoothed version of L_{i+1}. So, solving each L_i brings the solution closer to the basin of optimal solutions. One way to construct loss functions is to use *blurring*. The idea is to compute loss function at sampled points in the vicinity of a given point and average the values to create the new loss function.

2. Data-centric approach. Starts training the model on a simpler data instances, and gradually adds more difficult instances to the training data. The main hypothesis is that different training data sets provide different levels of difficulty to a learner.

References