Large weight -> a sign of a more complex network that has overfit the training data.

Drop out -> simple and effective regularization method.

Small dataset can overfit the training data -> poor performance in new(test) data

**To reduce overfitting**

1. 같은 데이터 셋을 모든 neural network에 적용 후, 각 모델의 결과값 평균 내기

* Not feasible in practice

1. Randomly drop nodes

During training, some number of layer outputs are randomly ignored or “*dropped out*.”

Be treated-like a layer

**How to dropout**

Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer.

A common value is a probability of 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from the visible layer.

“The weights of the network will be larger than normal because of dropout. Therefore, before finalizing the network, the weights are first scaled by the chosen dropout rate. The network can then be used as per normal to make predictions” -> pytorch나 이런거에서 batch 끝날 때 알아서 해줌

**Tip**

- Dropout rate

A good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as of 0.8.

- # of nodes and dropout rate

A good rule of thumb is to divide the number of nodes in the layer before dropout by the proposed dropout rate and use that as the number of nodes in the new network that uses dropout. For example, a network with 100 nodes and a proposed dropout rate of 0.5 will require 200 nodes (100 / 0.5) when using dropout.

- Grid search…

- Weight constraint

- Use with smaller dataset

*the computational cost of using dropout and larger models may outweigh the benefit of regularization.*

Deep neural nets with a large number of parameters are very powerful machine learning systems

One of the drawbacks of dropout is that it increases training time. A dropout network typically takes 2-3 times longer to train than a standard neural network of the same architecture. A major cause of this increase is that the parameter updates are very noisy. Each training case effectively tries to train a different random architecture. Therefore, the gradients that are being computed are not gradients of the final architecture that will be used at test time. Therefore, it is not surprising that training takes a long time. However, it is likely that this stochasticity prevents overfitting. This creates a trade-off between overfitting and training time. With more training time, one can use high dropout and suffer less overfitting. However, one way to obtain some of the benefits of dropout without stochasticity is to marginalize the noise to obtain a regularizer that does the same thing as the dropout procedure, in expectation. We showed that for linear regression this regularizer is a modified form of L2 regularization. For more complicated models, it is not obvious how to obtain an equivalent regularizer. Speeding up dropout is an interesting direction for future work. -> 책에서는 왜 cheap하다고…?

**Weight decay**

*Reasonable values of lambda [regularization hyperparameter] range between 0 and 0.1.*

**In CNN**

Weight regularization does not seem widely used in CNN models, or if it is used, its use is not widely reported.

L2 weight regularization with very small regularization hyperparameters such as (e.g. 0.0005 or 5 x 10^−4) may be a good starting point.

*We found that this small amount of weight decay was important for the model to learn. In other words, weight decay here is not merely a regularizer: it reduces the model’s training error.*

**IN LSTM**

An often used configuration is L2 (weight decay) and very small hyperparameters (e.g. 10^−6). It is often not reported what weights are regularized (input, recurrent, and/or bias), although one would assume that both input and recurrent weights are regularized only.

*L2 weight decay is used with a weight of 10^−6*

**Overfit Multilayer Perceptron Model -> +0.2**

**7.1 Regularization from a Bayesian Perspective**

Bayesian estimation theory takes a fundamentally different approach to model estimation than the frequentist view by regarding the model parameters themselves as uncertain and therefore treating them as random variables.

1. Prior distribution

There is a deep connection between the Bayesian perspective on estimation and the process of regularization.

텍스트이(가) 표시된 사진

자동 생성된 설명 -> prior가 regularizer 와 같은 역할

**7.5 Dataset Augmentation**

The best way to make a machine learning model generalize better is to train it on more data.

To get around this problem

- create fake data

- add it to the training set

-> easiest for classification

* 근데 이것도 힘들어. density function 모르는데 가짜 데이터를 어찌 만들겠어~

Noise 약간 섞는게 좋은 방법이 될 수 있다

To compare the performance of one machine learning algorithm to another, it is necessary to perform controlled experiments. When comparing machine learning algorithm A and machine learning algorithm B, it is necessary to make sure that both algorithms were evaluated using the same hand-designed dataset augmentation schemes.

**7.6 Classical Regularization as Noise Robustness**

1. adding noise to the **input** -> add noise directly to the model parameters

2. noise has been used in the service of regularizing models is by adding it to the **weights** -> Bayesian

**7.10 Bagging and Other Ensemble Methods**

The idea is to train several different models separately, then have all of the models vote on the output for test examples.

….

**7.12 Multi-Task Learning**