Lecture 18

Overfit, Dropout, EarlyStop, Batchnorm

Overfitting

Are you happy with 100% accuracy with train data?

Probably, YES! but...

Train accuracy: 100%
Test accuracy: 10%

Overfitting

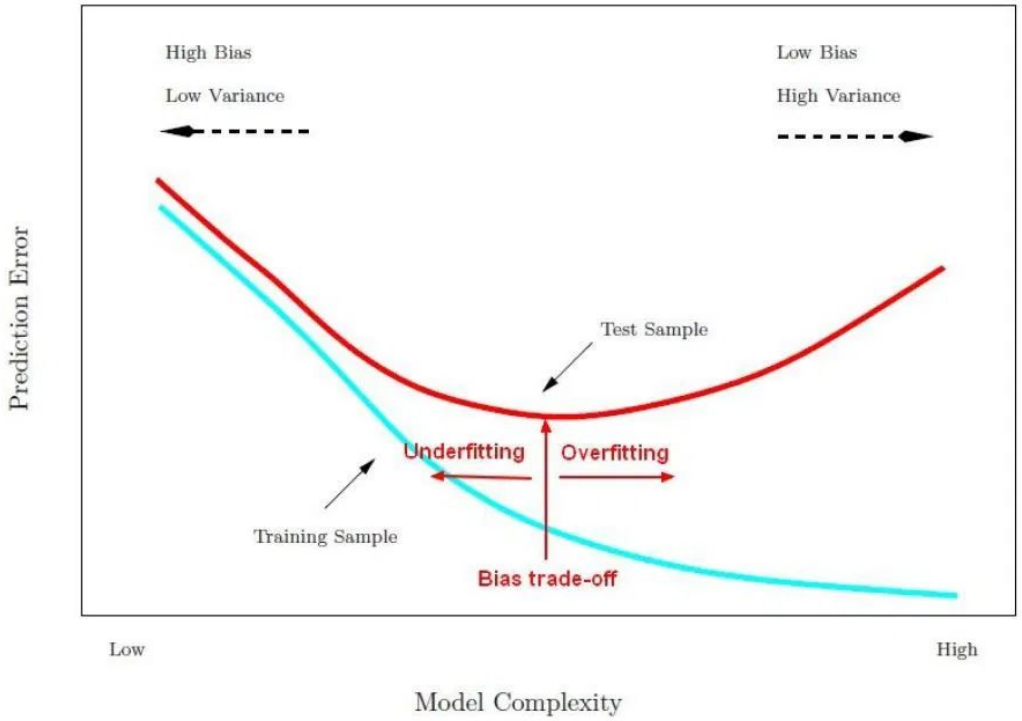
Overfitting occurs when a model becomes overly tuned to the training data, capturing not only the underlying patterns but also **the noise and random variations** present in the data.

Consequently, **the model's performance on new and unseen data is adversely affected**.

Essentially, the model learns to recognize and incorporate the specific details of the training data that are irrelevant or misleading when applied to new data.

This **lack of generalization** hampers the model's ability to make accurate predictions or classifications.

Overfitting vs Underfitting

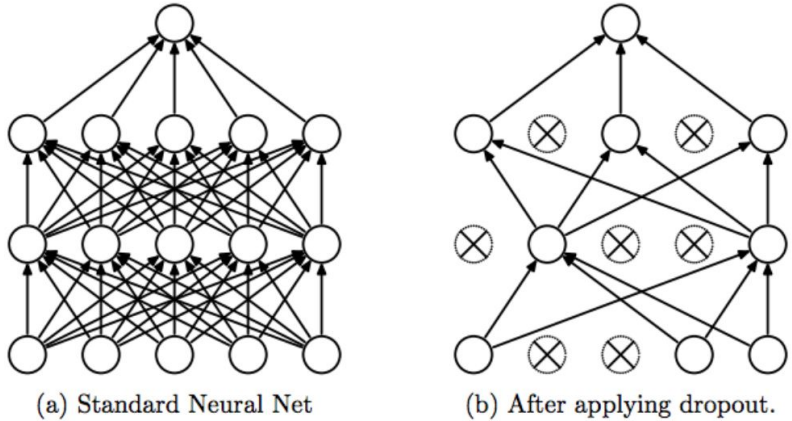


Dropout

Now, we know that the learning is not unconditionally improved by just adding more nodes or layers.

Dropout is to randomly turn off some of the nodes in the hidden layer.

By randomly turning off the nodes in this way, it is possible to prevent learning from being too biased to the learning data.



(a) Standard Neural Net

(b) After applying dropout.

Dropout in Pytorch

```python
import torch
import torch.nn as nn

# Create a simple model with a dropout layer
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(10, 20)
        self.dropout = nn.Dropout(0.5)  # Dropout layer with dropout probability of 0.5
        self.fc2 = nn.Linear(20, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.dropout(x)  # Apply dropout to the output of fc1
        x = torch.relu(x)
        x = self.fc2(x)
        return x
```

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

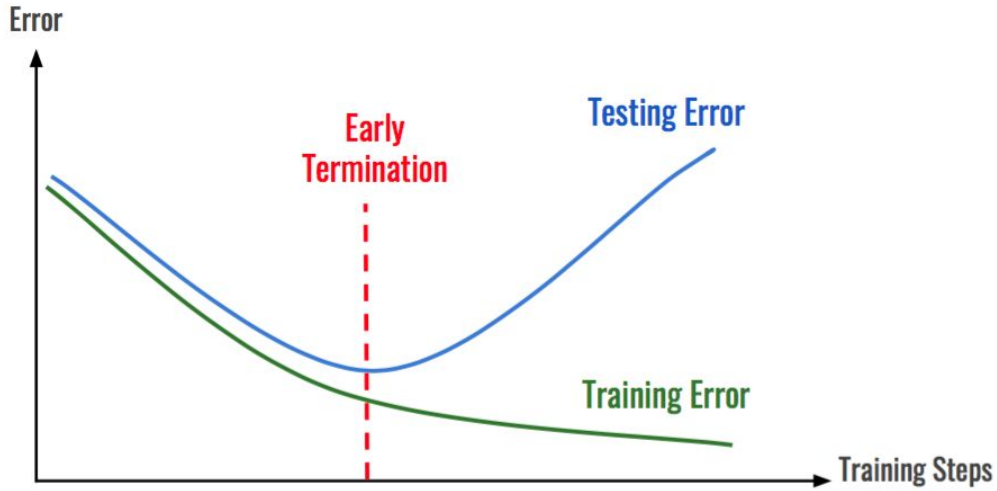
Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

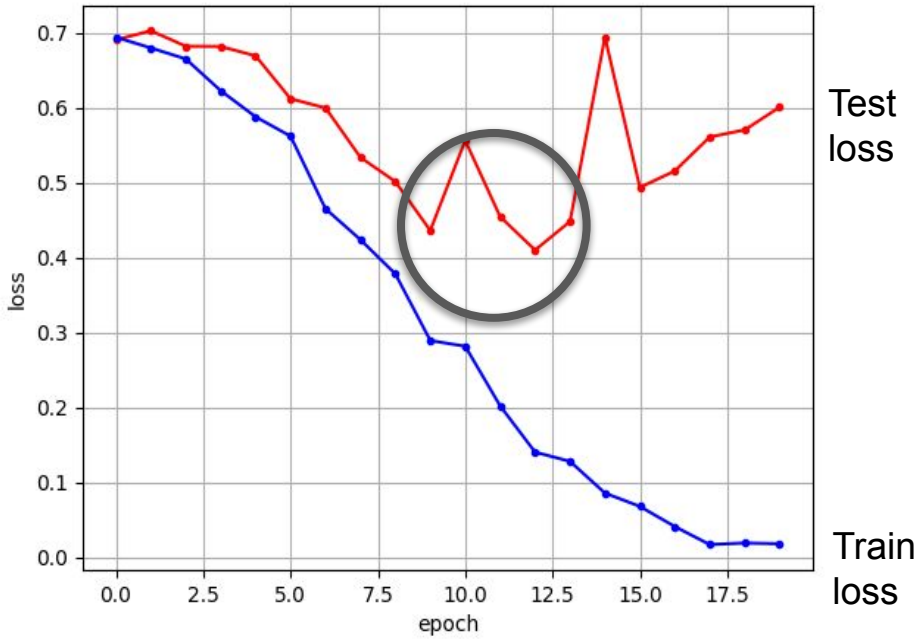
Keywords: neural networks, regularization, model combination, deep learning

Early Stopping

Having too much iteration may cause too much variance on train data, once validation loss stop decreasing, stop iteration

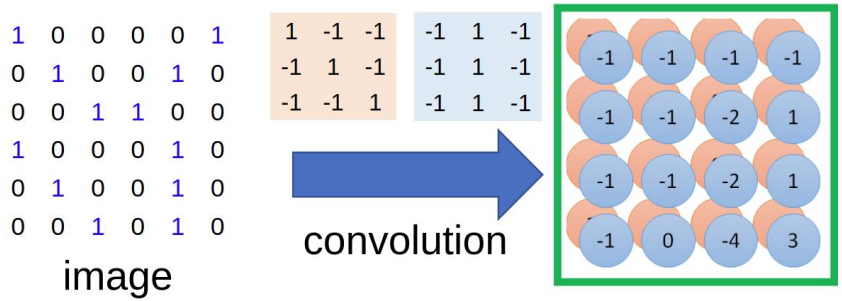


Dr. Kim’s example



Padding

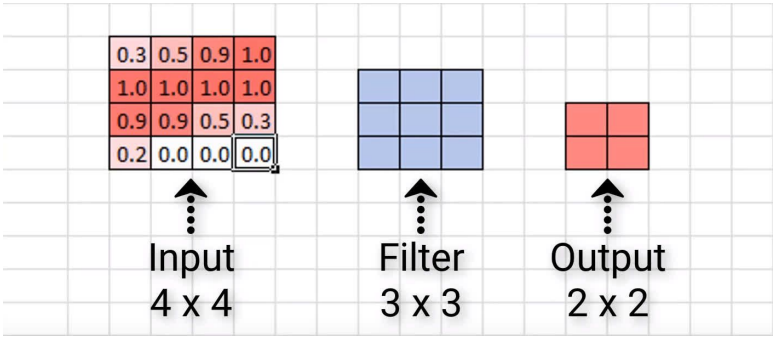
Remember that when you move the mask/filter and create the convolution layer, the size of the image decreases from the beginning.

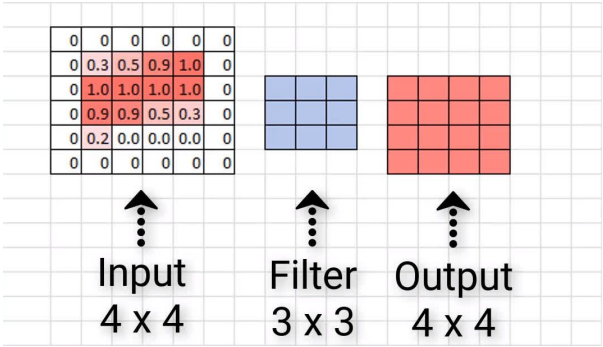


If you want to keep the image size, Keras' **padding** feature makes it easy to handle this problem.

Padding

Padding in PyTorch refers to the technique of adding extra elements or values to the input data or feature maps, usually around the borders, to preserve the spatial dimensions during convolutional operations. Padding is commonly used to ensure that **the output size matches the input size** or to prevent information loss at the edges of an image or feature map.





$$(N_h - K_h + 2*P_h)/S_h + 1 \quad x \quad (N_w - K_w + 2*P_w)/S_w + 1$$

Padding

```python
import torch
import torch.nn as nn

# Create a 2D convolutional layer with padding
conv_layer = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)

# Create a random input tensor
input_tensor = torch.randn(1, 3, 32, 32)  # Batch size of 1, 3 channels, 32x32 image

# Apply the convolutional layer to the input tensor
output_tensor = conv_layer(input_tensor)

# Print the shapes of the input and output tensors
print("Input tensor shape:", input_tensor.shape)
print("Output tensor shape:", output_tensor.shape)
```

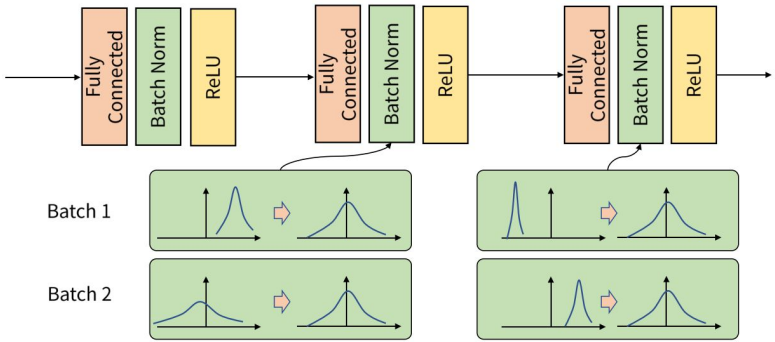
```
Input tensor shape: torch.Size([1, 3, 32, 32])
Output tensor shape: torch.Size([1, 16, 32, 32])
```

BatchNorm2d

Batchnorm2d refers to the process of normalizing batch data during training by using their mean and variance.

This technique ensures that regardless of varying data distributions within each batch, the normalization adjusts the distribution such that it has a mean of 0 and a standard deviation of 1.

This standardization is applied across all batch units to facilitate more efficient learning.



https://velog.io/@danbibibi/PyTorch-nn.BatchNorm2d