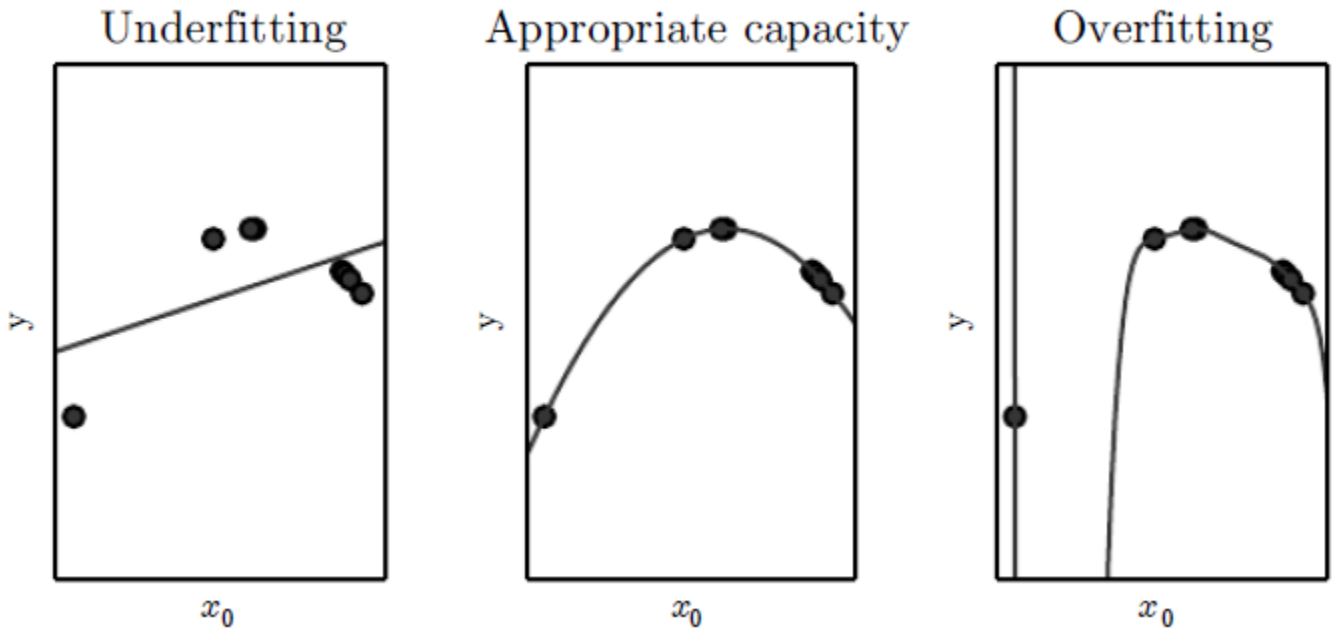
مهرداد برادران (99222020)

Neural network (AI First project)

Classification Fashion M-nist

1) Increasing the number of hidden layers might improve the accuracy or might not, it really depends on the complexity of the problem that you are trying to solve.

2) Increasing the number of hidden layers much more than the sufficient number of layers will cause accuracy in the test set to decrease, yes. It will cause your network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data. A picture taken from the aforementioned book gives a pretty good intuition for this concept



Where in the left picture they try to fit a linear function to the data. This function is not complex enough to correctly represent the data, and it suffers from a bias(underfitting) problem. In the middle picture, the model has the appropriate complexity to accurately represent the data

and to generalize, since it has learned the trend that this data follows (the data was synthetically created and has an inverted parabola shape). In the right picture, the model fits to the data, but it overfits to it, it hasn't learnt the trend and thus it is not able to generalize to new data.

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.

A simple and powerful regularization technique for neural networks and deep learning models is dropout.

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to ***complex co-adaptations***.

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

Batch normalization is a technique for training very deep neural networks that normalizes the contributions to a layer for every mini-batch. This has the impact of settling the learning process and drastically decreasing the number of training epochs required to train deep neural networks.

What are the advantages of Batch Normalisation?

- The model is less delicate to **hyperparameter tuning**. That is, though bigger **learning rates** prompted non-valuable models already, bigger LRs are satisfactory at this point
- Shrinks **internal covariant** shift
- Diminishes the reliance of **gradients** on the scale of the parameters or their underlying values
- **Weight initialization** is a smidgen less significant at this point
- **Dropout** can be evacuated for regularisation

For test the model on L1 and L2 we just need to add one new argument

In below function when we creating out model

L1:

Layer = keras.layers.Dense(num, activation='relu', kernel_regularizer=keras.regularizers.l1(0.1))

L2:

Layer = keras.layers.Dense(num, activation='relu', kernel_regularizer=keras.regularizers.l2(0.01))

L2

```
layer = keras.layers.Dense(100, activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l2(0.01))
```

L1

```
layer = keras.layers.Dense(100, activation="elu",
                           kernel_initializer="he_normal",
                           kernel_regularizer=keras.regularizers.l1(0.1))
```