Initialization:

Xavier initialization is designed to work well with tanh or sigmoid activation functions.

He initialization works with ReLU activations.

Regularization:

1. The importance of regularization:

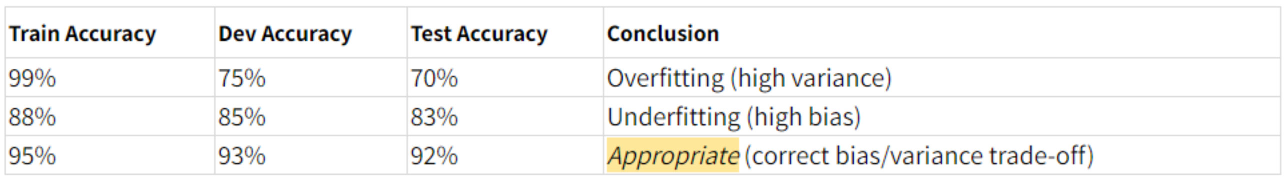
   Ultimately, you want your neural network to generalize to “unseen data” so that it can be used in the real world. A neural network's ability to generalize to unseen data depends on two factors:

   - The information in the training data. For example, a dataset of images taken from the front-facing camera of a car contains intuitively more complex information than a dataset of images of clouds in the sky.
   - The complexity of the network. By complexity, we mean the complexity of the function your neural network can mimic. Usually, the more parameters your network uses, the more complex it is.

   Case 1: The network is not complex enough, and the training data contains a lot of information: Your network is too simple to understand the training data’s salient features. This is called *underfitting* the training set.

   Case 2: The network is very complex, but the training data doesn’t contain too much information: Your network is *complex* enough to fully memorize the mapping between the training data and the training labels. However, it does not generalize well to unseen data because it has merely over-memorized the salient features of the training set. This is called *overfitting* the training set.

   Data Split (partioning a data set): Before we explore the main regularization methods, you must partition your dataset. In order to estimate the ability of your model to generalize, you will split your dataset into three (or sometimes more) sets: *training*, *dev* and *test*. If your model is trained on the training set, tuned on the dev set, and still performs well on the test set, it is able to generalize successfully.

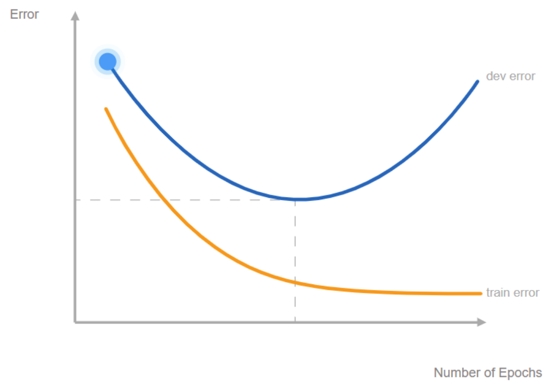


Train Accuracy	**Dev Accuracy**	**Test Accuracy**	**Conclusion**
99%	75%	70%	Overfitting (high variance)
88%	85%	83%	Underfitting (high bias)
95%	93%	92%	*Appropriate* (correct bias/variance trade-off)

   You want to close the performance gap between your test set and your dev and training set while keeping the training performance as high as possible. Let’s delve into the methods that will help you do so.

2. Early stopping:

   One of the widely used regularization method is called early stopping.



   Despite being practical, early stopping is not satisfying from a scientific standpoint. There exist other regularization methods, such as L1 (/L2) regularization and dropout, or optimization methods.

3. L1 and L2 regularizations

   L1 and L2 regularizations can be achieved by simply adding a term that penalizes large weights to the cost function.

$$J_{regularized} = J_{cross-entropy} + \lambda J_{\text{L1 or L2}}$$

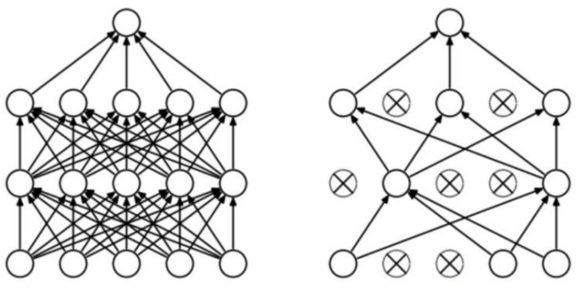
   where:

$$J_{L1} = \sum_{\text{all weights } w_k} |w_k| \text{ and } J_{L2} = ||w||_2^2 = \sum_{\text{all weights } w_k} |w_k|^2.$$

4. Dropout regularization

   Dropout regularization, that have been shown to be more effective at regularizing larger and more complex networks. If you had unlimited computational power, you could improve generalization by averaging the predictions of several different neural networks trained on the same task. The combination of these models will likely perform better than a single neural network trained on this task. However, with deep neural networks, training various architectures is expensive.

   Dropout is a regularization technique, that allows you to combine many different architectures efficiently by randomly dropping some of the neurons of your network during training.



(a) Standard Neural Net (b) After applying dropout.