Name:Dina Tarek

Assignment-Course 2- Week 1

Week 1: Introduction to Deep Neural Networks

Outlines:

- types of **initializations** with different results & their importance
- difference between **train/dev/test sets**
- **bias** & **variance Issues** in your model
- **regularization methods:dropout** or **L2 regularization.**
- how to deal with them experimental issues : **Vanishing** or **Exploding gradients**
- **gradient checking** to verify the **correctness** of **back propagation** implementation

Setting up your Machine Learning Application

1.Train ,Dev & Test sets

- Ranging from hyperparameter tuning to, how to set up your data, to how to make sure your optimization algorithm runs quickly so that you get your learning algorithm to learn in a reasonable time.

- Randomization problem

- how to set up training, development, and test sets can make a huge difference for good high performance neural network

- To train a neural network, a lot of decisions are taken , such as:
  - **how many layers will your neural network have?**
  - **And how many hidden units do you want each layer to have?**
  - **And what's the learning rates?**
  - **And what are the activation functions you want to use for the different layers?**
  - it's almost impossible to correctly guess the right values for all of these, and for other hyperparameter choices, on your first attempt.

- machine learning is a highly iterative process:
  - start with an **idea**, such as you want to build a neural network of a certain number of layers, certain number of hidden units, maybe on certain data sets
  - then **code** it up and try it by running your code.
  - **run** and **experiment**
  - a result that tells you how well this particular network, or this particular configuration works.
  - based on the outcome, **refine** your ideas and change the choices and maybe keep iterating in order to try to find a better and a better neural network.

- the best choices may depend on the **amount of data** you have, the **number of input features** you have through your computer configuration and whether you're training on GPUs or CPUs.

- determine how quickly to make progress is how efficiently you can go around this cycle. And setting up your data sets well in terms of the train, development and test sets can make much more efficient at that.
- In training data process , take all the data you have and carve off some portion of it to be the **training set.**
- Some portion of it to be the **hold-out cross validation set**, and this is called the **development set**. **(dev set)**
- the carve out some final portion of it to be your **test set**.

The workflow :

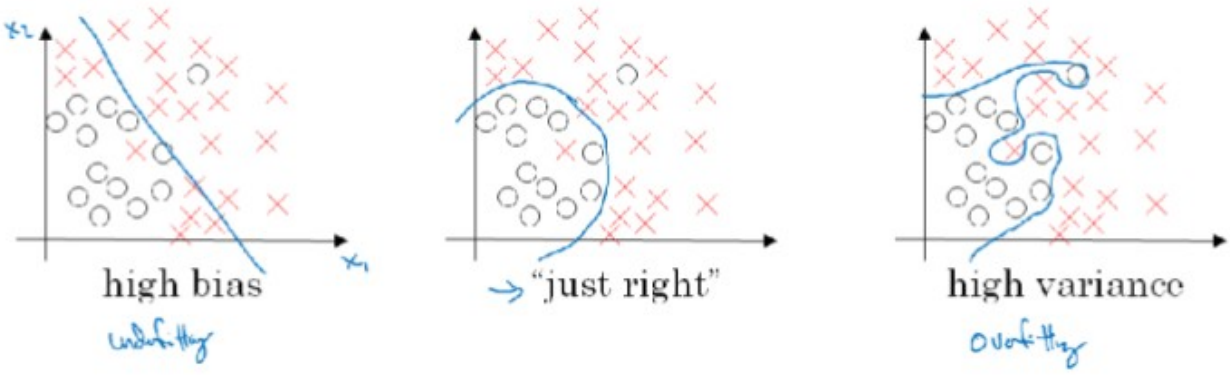
- keep on **training algorithms on your training sets**.
- use **dev set** to see which of **many different models performs best** on your dev set.
- After having done this long enough, having a final model that you want to evaluate,
- **take the best model** found and **evaluate it on your test set** to get an unbiased estimate of how well your algorithm is doing.

Notes:

- common practice to take all your data and split it according to maybe a 70/30% in terms of a people often talk about the 70/30 train test splits. If you
- don't have an explicit dev set or maybe a 60/20/20% split in terms of 60% train, 20% dev and 20% test
- the goal of the dev set or the development set is that you're going to test different algorithms on it and see which algorithm works better. So the dev set just needs to be big enough to evaluate
- if you have a relatively small dataset, these traditional ratios might be okay.
- if you have a much larger data set, it's also fine to set your dev and test sets to be much smaller than your 20% or even 10% of the data.

- It is better that **dev and test sets come from the same distribution**

2.Bias , Variance & Fitting



- **Under-fitting the data:** if you fit a straight line to the data, maybe you get a logistic regression fit to that. not a very good fit to the data (**high bias**)

- **just right:** more reasonable fit to the data.
- **Overfitting the data:**fit **complex classifier,(high variance)**

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	High Variance (Over Fitting)	High bias (Under Fitting)	High bias High Variance	Low bias Low Variance

looking at the **training set error**

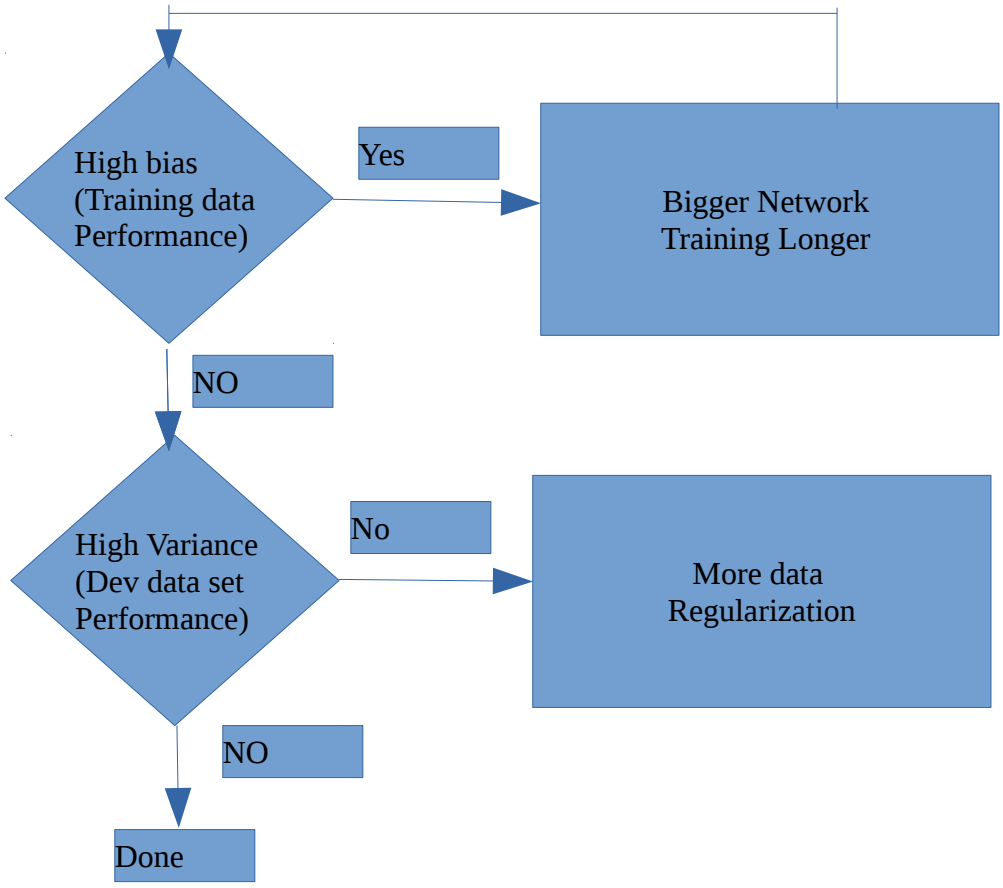
- get a **sense of how well you are fitting**, at
- **least the training data**
- tells you if you have a bias problem.

looking at how much higher your error goes, when you go from the training set to the dev set, that should give you a **sense of how bad is the variance problem**, so you'll be doing a good job generalizing from a training set to the dev set, that gives you sense of your variance. All this is under the assumption that the **bayes error** is quite small and that your training and your dev sets are drawn from the same distribution.

Bayes Error (**Optimal error**) (ex. human level performance gets nearly 0% error)

3.Basic recipe for machine learning

getting a bigger network almost always just reduces your bias without necessarily hurting your variance, so long as you regularize appropriately and getting more data pretty much always reduces your variance and doesn't hurt your bias much.



4.Regularization

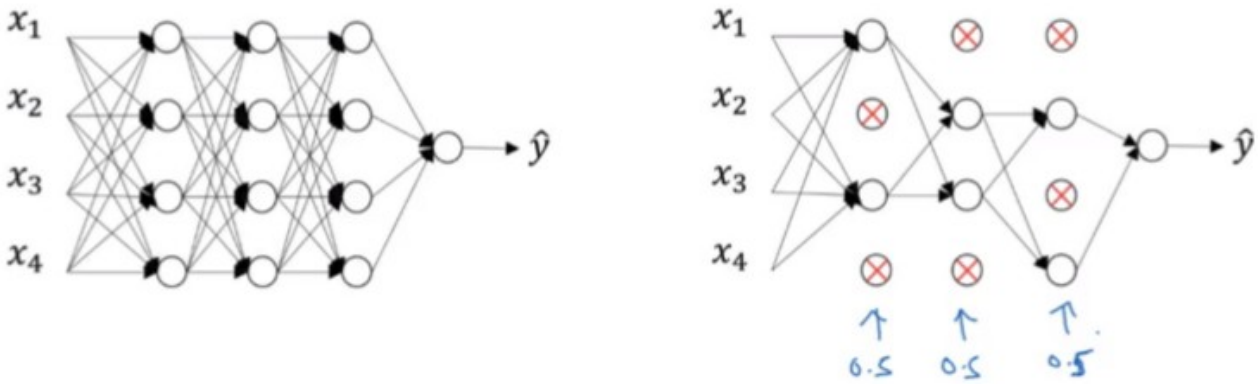
$$\min_{w,b} J(w,b)$$

- lambda Regularization Parameter , set using your development set, or using cross validation. (hyper parameter)
- minimize the cost function J
- w is usually pretty high dimensional parameter vector, especially with a high variance problem.
- b is just a single number.

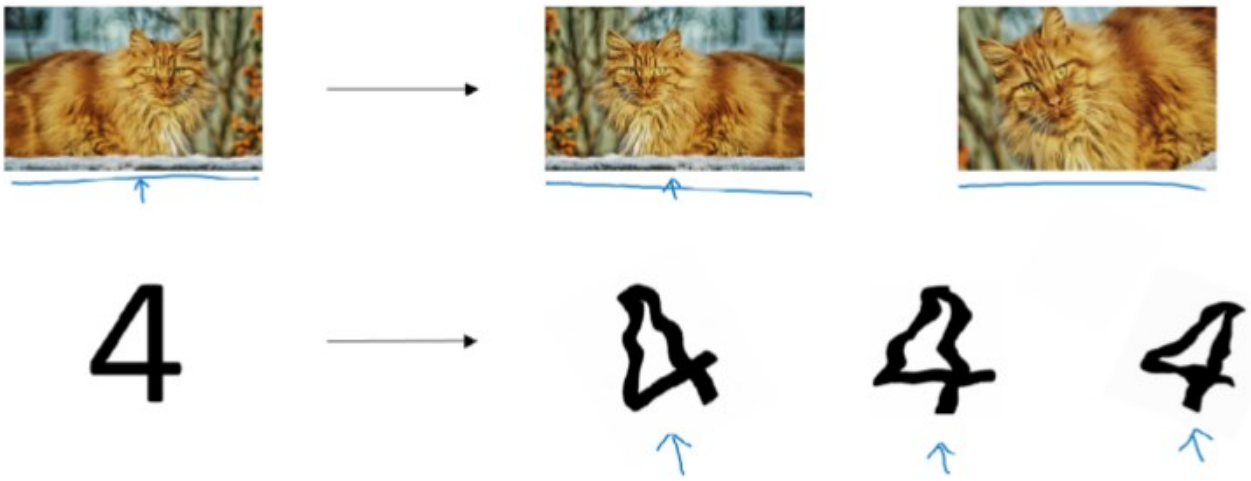
Why regularization reduces overfitting?

- matrices **W** is set to be reasonably **close to zero** take you from this overfitting case much closer to the left to other high bias case.
- using the **tanh relatively linear less able to overfit.**

5.Dropout



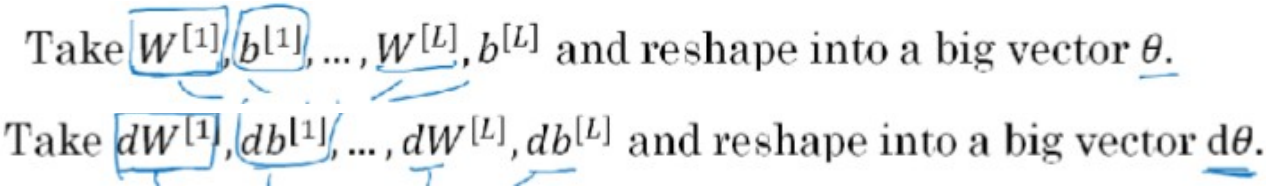
6.Data Augmentation



inexpensive way to give your algorithm more data and therefore sort of regularize it and reduce over fitting by flipping horizontally the image .

7.Gradient checking

a technique that can help in saving lots of time to find bugs in implementations of back propagation.



W is a matrix, and reshape it into a vector using reshaping and concatenation operation .