Important Q&A

Q. What's the intuition behind the fact that linear functions stringed together with non linear functions can approximate any type of function. Meaning why can deep learning learn a universal approximation function?

A. Traditionally in ML, we have been coming up with our **own approximate function** (the model) that we think would mimic the function we are trying to predict. The shortcoming is that the approximate function we come up with, is based on the best of our knowledge and it **does not** have the **freedom to change its form**. It's form is fixed. For e.g. if we think the model for a particular problem should be **f(x) = w1*x*^2 + *w2*x+b**, the form remains so and never changes.
In deep learning, by **stacking weighted sum of inputs** and **non-linear activation functions**, we let the network figure out *by **itself*** what form the approximate function should be - it does so by **adjusting its own weights**. By adding **more layers and nodes**, and by **choosing the right activation function**, we **give the network enough wiggle room** to **approximate** any function - given the **loss function**.

Q. In the lecture you mention that you prefer to build a model first and then analyze the results to clean the data. This approach sounds good for images (and maybe audio and text too). What about data which is just a bunch of features? Is it better to do the reverse approach in this case too?

A. You may still find it useful. You can see what **type of inputs** are giving you trouble by looking at the value of the features.

Q. So, calling the lr_find() is a better approach compared to manually setting up the learning rates?

A. Yes

Q.How about if we have three classes unlike in class example in which probability of cat is close to 0 and for dog to 1. Then how can we detect which probability belong to which of three classes.

A. You will have a network with 3 outputs in this case (for 3 classes). Each output will be labelled - that's how you will know which probability belongs to which class (even for the cat/dog e.g., the two outputs are labeled)

They will each output a probability number, which should add up to 1. You pick the one with the highest probability.
For e.g., if you have trained a model for detecting dog, cat and mouse: given an image the network might output 0.8, 0.1 and 0.1 respectively - this suggests it thinks the image is that of a dog.

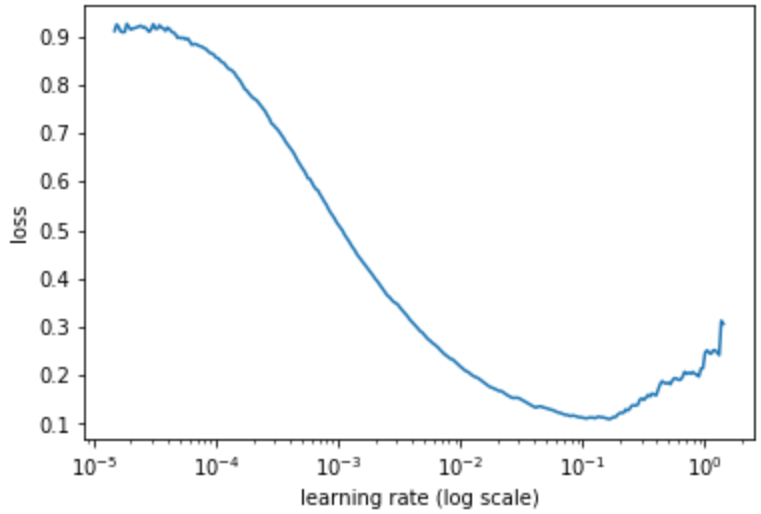
Q. Is there a logic behind calculating the batch size? Say, if I have this much Gig of vRAM, then I should set y amount of batch size?

A. **Nerdy way** (not required)- You can actually calculate how much memory your network will occupy in the GPU - by calculating all the weights and biases and gradients etc. (don't do it, there will be millions and millions of them in a serious network)
Practical way - You run your program, if it reports OOM error, reduce batch size. Good thing is, you will get to know if your batch size is big or not in the first epoch itself.

Note: Once, you encounter a OOM error or CUDA could not find memory typish error, reduce the batch size and restart the kernel. The reason is that, the GPU does not recover the memory it had used and hence, you might still be stuck with the OOM error even after reducing the batch sizes. Remember that, in the neural network a lot of calculations happen from layer to layer which are Linear Algebra based and hence a lot of memory is used.

Q. I had this question as well! Why pick a learning rate point where the loss was still improving and not the lowest point?

A. I suggest trying a higher learning rate (the learning rate at the lowest point) and comparing your results to the rate Jeremy used. The idea is we want to converge as fast as possible to optimize the function we are approximating without going too fast.
In p1v1, Jeremy explains using too high of a learning rate as if we are jumping over a valley, instead of further down into the valley.
The `lrf=learn.lr_find()` function is increasing the learning rate gradually during training. The plot you see is the loss, which we generally want to minimize, as a function of the learning rate.



If you want to learn more about what is being done, I suggest reading the paper referenced (https://arxiv.org/abs/1506.01186). ****note**, in the paper they tell you to **plot accuracy vs. learning rate** and **pick the learning rate where the accuracy is still increasing**; at the end of the day this should have the same outcome as plotting against loss and picking the learning rate where loss is still decreasing.
Maybe someone else out there has a better explanation of "why" this works. To be perfectly honest, I'm not entirely sure. It might be one of those things where theory hasn't caught up with best practices yet.

Q. Why does the learning rate find work? Does it pick 30 or N mini batches and trains it and gets the loss? Isn't that getting a local error per mini batch? How does using this learning rate optimal for the global optimization?

A. The idea is for us to plot and visually pick the learning rate giving low enough loss but fast enough. Since you visually look at it, any noise due to mini batch local optima can be ignored and we can pick a value from a smooth plot.

Q. Is the learning rate plot created from randomized mini-batches?

A. No. There are two plots. In one plot you have **x=mini-batch, y=learning-rate**. This is showing how learning rate is increasing as a function of mini-batch. This is for the algorithm that find best learning rate.
The other plot is learning rate as a function of loss.

Q. In CS231n Lecture 7 - Convolutional Neural Networks Andrej Karpathy says that “Number of filters in CNN are typically chosen as a powers of 2 (64/128/256/512/1024 etc) for computational reasons. Some libraries if they see powers of two, they might go into a spatial subroutine which is very efficient to perform in vectorized form. So, sometimes this might give you an improvement in performance”. (22 min of lecture). No conclusions so far, just an information

A. Well spotted. I don’t think I’ve seen that in practice, however.

Q. Write down details about each function

A.

References

[1] Lesley N Smith, Cyclical Learning Rates for Training Neural Networks:- https://arxiv.org/abs/1506.01186

[2] Alex J. Champandard, Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks:- https://arxiv.org/abs/1603.01768

[3] Brendan Fortuner, Linear Algebra cheat sheet for Deep Learning:- https://towardsdatascience.com/linear-algebra-cheat-sheet-for-deep-learning-cd67aba4526c

[4] Explained Visually, Image Kernels:- http://setosa.io/ev/image-kernels/

[5] Matthew D. Zeiler et al, Visualizing and Understanding Convolutional Networks:- https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

[6]