

Part I

In the previous lecture the ADAM optimizer was discussed, we got reminded that it has momentum and adapts the weights yada-yada.

History of DL(?)

We discussed the frameworks that are used for DL, both low- and highlevel ones. We will mostly discuss high level frameworks like TF, NumPy, Torch etc. The *1st gen* frameworks implement NNs with config files — those are frameworks like Caffe. They are easy to delpoy, they have a lot of things done already, but are hard to debug and use for some of the newer things.

The second generation of frameworks used not layers, but functions as layers. So... symbolic graphs (although static). The gradients are automatically computed, ... (just see the respective slide). Some of *2nd gen* frameworks are TF (before 2.0) and Theano. It's biggest downside is that the graph optimization takes a lot of time and the training process does not commence before the graph is ready (i wish i had this problem at work...).

Gen 3 — dynamic graphs. Now the computations do not have to be defined before computing. Examples: Torch, Chainer, DyNet. Their value mostly comes from user being able to change things on the fly yet they are cumbersome to optimize. We are going to use Torch, but we are encouraged to try other frameworks.

The Jax framework is a super-duper math framework with jacobians, fancy gradients(huh?). Mostly used for research.

Note: there are things like Triton that allow you to run low-level instructions with python-like syntax.

Part 2

In terms of features we can process both features in data and an image by mixing the in the process of learning. To change the influence of one or the other we could change the amount of inputs of either.

It would be really good if we were able to use a NN that already knows something... E.g. if you are classifying people as fashionable or not it would be nice to be able to tell if the picture shows a human etc.

Dropout — “I don't want my NN trusting any single neuron too much”. It is a specific layer that zeroes some of the incoming weights with a specific probability and also scales the remaining weights so the average is the same. This is done only in training time. Dropout has to be used in conjecture with another

layer after it since otherwise you are losing out. Mostly it is just in case of your model overfitting.