Glossary and general comments on ML terms

RNNs

- Quote from Karpathy: >If training vanilla NN is optimizing over functions, traning recurrent networks is optimizing over programs
- Even if the data is not sequential per se, you can still formulate the problem as a recurrent one (e.g. with the same input fed into the system multiple times or just once then letting the network run a few steps) Ladder networks
- **BPTT:** Back propagation through time
- **Softmax/Cross-Entropy** The softmax is given by

  $P_t(a) = \frac{\exp(q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(q_t(i)/\tau)}$

  where τ refers to the *temperature of the softmax*. The temperature is a hyperparameter: For high temperatures, all activations have similar probabilities (high entropy). For low τ, the highest input activation (even if it's only slightly larger than the other input activations) gets a disproportional probability while the other values vanish. Thus, it can be said that ***Higher temperatures give more variability in it's estimates but more mistakes (in case of k-classification) while low temperatures gives conservative estimates and only places probability mass on the largest value.***
- The main problem with RNNs is managing long-term dependencies (see ATML lecture).
- **End-to-end training** Training a network from raw input data (e.g. camera pixels) to the end task of the whole system (e.g. steering of a car) withouth having a **pipeline of networks** where each network completes a separate task and is trained independently (*divide-and-train*). The end-to-end training might be more beneficial because the network can be trained in one go without having to rely on human-defined subtasks. Instead, it can learn what SGD says is a good subtask.

Densely connected Conv. Nets

Philiosophy of DenseNets: Feature reuse

- CNNs were introduced by LeCun in 1998 (5 layers *LeNet5*). VGG in 2015 has 19 layers.
- **stochastic depth**: A training procedure where layers are randomly dropped during training (during inference, the whole network is present). Some ResNets have been using $L = 1202$ layers. According to the authors this shows that many of the layers are actually redundant and maybe it's

more important to have better connection schemes. Hence, DenseNets use fewer parameters than traditional CNNs and also they're sort of narrow (e.g. 12 feature maps per layer). There's only a small gain in *collective knowledge* per layer, but old information is also not lost. In addition, the gradient for early layers doesn't vanish because we have some direct connections.

Batch normalization

BN is a technique to use much higher learning rates and to be less careful about initialization. It also acts as a regularizer. According to Goodfellow: > One of the most exiting recent developments in optimizing deep learning. It's a method of adaptive reparametrization.

- It was motivated by the difficutly of training very deep models. One of the problems was that the weights all get updated at the same time but under the assumption that all the other weights are constant (partial derivatives)
- **Covariante shift** refers to changes in the distribution of the input data: This often happens in medicine, e.g. when we look at pig eyes for training but are actually interested in human eye data for inference.
- **Internal covariate shift** thus refers to this aforementioned shift happening between the layers of a NN.

Learning rate optimization

- **AdaGRAD** accumulates the squared gradients since the beginning and scales the learning rate by it's inverse. This means that the learning rate monotonically decays over time. It's important to notice that the learning rate is thus different for **all** parameters. Paramters with small gradients see a slow decay, while high gradients quickly vanish due to low learning rates. The drawback of AdaGRAD is that there is the possibility of premature and excessive decrease in the effective learning rate because it accumulates gradients *from the beginning of training*. It's very well suited for convex optimization.
- **RMSProp** is a modification to AdaGRAD which performs better in the non-convex world which is NNs. Instead of adding up the squared gradients, it forms an exponentially weighted moving average where the hyperparameter ρ in $r = \rho r + (1 - \rho)g$ controls how quickly the average forgets.
- **Adam: Adaptive moments