Dependency Parsing

Shubham Gupta

January 19, 2020

1 ML and Neural Networks

1.1 Optimizers

- Adam Optimizer
  - Since momentum accumates the gradient from the previous steps, it can help the algorithm get out **pathological curvature** areas. These are areas where the gradient decerasses slowly because the gradient keeps bouncing around the edges of this area, thereby leading to slower convergence. The momentum parameter m helps add momentum from the previous steps and do an exponential average. This is useful because it can use this momentum to go through these curvatures faster, thereby leading to convegence at the global minima. **TLDR: We get to the local minima faster and do not oscillate on the y-axis while searching for the optimal path as we add a term to help move faster along the x-axis**
  - My guess is the v term has accumated gradients over time. It gives more preference to gradient updates, thus affecting weights that have not been changed in a long time to escape from the local minima area the optimizer is stuck inside. I could be wrong.
- Dropout
  - γ looks like a regularization term. In terms of probability, it should be expressed as: $\gamma = p_{drop} * (1 - p_{drop})$. This value will be maximum whenthe value for p_{drop} is 0.5
  - Dropout essentially helps us train an ensemble of models together on the same dataset. This is important because it helps learn a general function and prevents the occurence of *coadaptation*, which is a process of learning signals that make only a particular set of connections strong. These strong connections are used during prediction as well, thereby ignoring all the other weaker connections which could have learned other important features. In training, because some of the neurons are switched off, the output is usually a function of the probability by which the neurons were switched off. Using the video of CS231 available:
    * Dropout makes output random: $y = fw(x, z)$ where z is random
    * Randomness at test time is bad

* Elimate this randomness
* Average out this randomness i.e $y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$
* Difficult to compute the above integral
* **Solution**: Approximate integrate
* average out the value across all dropout masks. This translates to multiplying the weights at each layer while predicting with the probability p_{drop}. Cheap approximation to the integral