CS231n: CNNs for Visual Recognition **Spring 2017**

Lecture 7

Course Coordinator: Prof. Fei-Fei Li *Scribes: Akash Gupta*

Further discussion -

- Fancier Optimization
- Regularization
- Transfer Learning

Optimization -

Problems with SGD -

- Loss changing quickly in one direction and slowly in another, this is referred to as the loss having high condition number. Formally, condition number is the ratio of largest to smallest value of the Hessian matrix. So, in this case very slow progress along shallow dimension and zig-zaggy along steep direction
- Local minima or saddle point
- This problem is not solved even by normal GD.

SGD + Momentum - Maintain a velocity everytime and we add the gradient estimates to this velocity. So, we step in the direction of the velocity. New hyperparameter $\rho = 0.9 \, or \, 0.99$

$$v_{t+1} = \rho v_t + \nabla f(x_t) \tag{7.1}$$
$$x_{t+1} = x_t - \alpha v_{t+1} \tag{7.2}$$

Q) *How does SGD+momentum help with poorly conditioned coordinates?*
A) If gradient is relatively small and ρ is well behaved then the velocity can monotonically increase at this and actually be larger than the actual gradient and so making faster progress along poorly conditioned dimension.

Nesterov Momentum - Here we first step in the direction of the velocity, we evaluate the gradient at that point and then go back to original point and kind of mix them.

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t) \tag{7.3}$$
$$x_{t+1} = x_t + v_{t+1} \tag{7.4}$$

Can rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x_t}) \tag{7.5}$$
$$\tilde{x_{t+1}} = \tilde{x_t} + v_{t+1} + \rho(v_{t+1} - v_t) \tag{7.6}$$

AdaGrad - During optimization, we will keep a running sum of squared gradients. So instead of velocity we have grad_squared term. On updating the params, we divide by this grad_squared term. We control the gradient updates, since by dividing we take shorter steps in zigzaggy dimension and large steps in shallow dimension.

Problem with AdaGrad: The gradients become smaller with time. Okay with convex case but with non-convex case we are stuck at saddle point.

RMSProp: Solves the above problem by adding a decay_rate to the grad_squared term.

Adam: SGD-Momentum + RMSProp - Moving estimate of squared gradients. But at very first timestep since second moment is initialized to 0 and we are dividing by that in param update. So we might take very large steps initially. To solve this we add a bias correction term. (NOTE: beta1 = 0.9, beta2 = 0.999, learning_rate = 1e-3 or 1e-4, a great start for models!)

$->$ Adam does not work in a rotated picture of poor conditioning.

Learning Rate - Apply decay over time.

1. Step decay $->$ Decay LR after a few epochs.
2. exponential decay $->$ $\alpha = \alpha \exp^{-kt}$
3. 1/t decay $->$ $\alpha = \alpha_0/(1+kt)$

First Order Optimization: Going in direction based on first-derivative.

Second Order Optimization: Take into account both first and second order derivative. Newton Step - generalized to multiple dimensions. But these are computationally expensive - $O(n^3)$

Quasi-Newton methods - One example is L-BFGS - keeps approximate of Hessian matrix.

In practice -

- Adam is good default choice.
- If you can afford to do full batch updates then try L-BFGS (E.g - Used sometimes in Style Transfer)

Model Ensembles - Train multiple independent models and at test time average their results. (Extra 2% performance)

Regularization - L1, L2, L1+L2 (Elastic Net), Dropout...

Dropout -

- Set some activations randomly to zero.
- Why is this a good idea? In making final decision, it does not depend on one feature too much whereas spreads the idea over different feature and so might prevent overfitting.
- It is also like training a large ensemble of models (that share params)
- At test time: multiply the output of the layer by dropout probability.
- Inverted Dropout: At test time use entire weight matrix but at training time you divide by Dropout probability.
- When using this gradients are only computed for units that are not dropped and so it takes more time to train during Dropout.

Data Augmentation -

- Horizontal Flips.
- Random Crops and scales
- Color Jitter
- and many more..

DropConnect - Remove weights instead of activations.

Fractional Max Pooling - Randomize the regions over which we pool.

Stochastic Depth - At training time, use a subset of layers and at Test time use all of them.

Q) Do we use more than one Regularization techniques?
A) Mostly Batch_Norm alone tends to be enough. If not working, apply according to the situation.

Transfer Learning -

- Small dataset - Reinitialize last matrix and freeze the weights of the initial layers and basically train a linear classifier and let it converge on the small data.
- Large dataset - Fine Tune - take some more of the last layers.

$->$ Transfer Learning with CNNs is pervasive (Norm, not exception)

$->$ Deep Learning software packages provide a "Model Zoo" of pretrained models so we can just download and use them.