Assignment 4: Advanced Optimization and Initialization
MA-INF 2313: Deep Learning for Visual Recognition

Due Date: Sunday, 13.12.2020

1 Theoretical Exercises (15 points)

a) Adaptive Learning Rates (*5 points*)

To get to know optimizers with adaptive learning rates, we will study them on a simple net. Consider a neuron o with two inputs $x, y \in \mathbb{R}$, one output and the Sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ as the activation function. In other words, $o(x, y) = \sigma(w_x \cdot x + w_y \cdot y)$. Let t be the target value for training and use the squared difference as the loss.

1. (*1 point*) Compute the partial derivatives of the loss with respect to the weights w_x and w_y.
2. (*4 points*) Let $x = 1, y = -1, t = 1$ and $w_x = -1, w_y = 1$. Compute the updated weights for three training iterations by hand using the following optimizers:
   - Stochastic gradient descent without (Nesterov) momentum,
   - AdaGrad,
   - RMSProp (no momentum) with decay rate $\rho = 0.9$,
   - Adam with default decay rates $\rho_1 = 0.9$ and $\rho_2 = 0.999$.

   Choose a step size of $\epsilon = 1$ and the stabilization constant $\delta = 10^{-8}$. Report all values that you used for computation.

b) Unstable Gradient Problem (*10 points*)

Alongside the exploding gradient problem, there is also the danger of *vanishing gradients*. This means that gradient values come dangerously close to machine precision and therefore weight changes become minuscule at computation time. We now want to analyze this instability.

As a first model, consider a neural net consisting of an input $x \in \mathbb{R}$, n hidden layers $h_i = \sigma(w_i \cdot h_{i-1})$ where w_i is the weight of the layer, $\sigma : \mathbb{R} \to \mathbb{R}$ is any activation function, $h_1 = \sigma(x)$ and h_n is the output of the net.

1. (*1 point*) Compute the derivative of the output with respect to the weights of the hidden layers, i.e. compute $\frac{\partial h_n}{\partial w_i}$ for $i \in \{1, ..., n\}$.

2. (*2 points*) Consider σ to be the Sigmoid function. What is the maximum value of the gradient of weight w_i with respect to i, if $|w_i| < 1$ for all $i \in \{1, ..., n\}$? How does your result change if you instead consider the ReLU activation $\sigma(x) = \max(0, x)$?

Now let us consider an ordinary MLP with d inputs $X := (x_1, ..., x_d)^T$, and n layers of n_i neurons each. Then the weights become matrices $W^{(i)} \in \mathbb{R}^{n_i \times (n_i - 1)}$ and the layer activations the vectors $h^{(i)} \in \mathbb{R}^{n_i}$. For our purposes we assume σ is the identity, so that $h^{(i)} = W^{(i)} \cdot h^{(i-1)}$. Additionally suppose that input and weights have zero mean, that the input has unit variance, and that all weights of a layer share the same variance, i.e.

$$\text{for } i = 1, ..., n: \ \text{Var}\left(W^{(i)}_{j,k}\right) = \text{Var}\left(W^{(i)}\right)$$

3. (*3 points*) Show that for the product of two independent random variables $\hat{X}, \hat{Y}$ where $\mathbb{E}(\hat{X}) = \mathbb{E}(\hat{Y}) = 0$, the following holds:

   $$\text{Var}(\hat{X} \cdot \hat{Y}) = \text{Var}(\hat{X}) \cdot \text{Var}(\hat{Y})$$

   Use the result to show the equivalent result for n zero mean, independent random variables $\hat{X}_1, \hat{X}_2, ..., \hat{X}_n$.

4. (*1 point*) Now show that the variance of an entry of $h^{(i)}$ can be expressed with respect to the variance of the weights and the input, i.e.

   $$\text{Var}\left(h^{(i)}_j\right) = n_i \cdot \text{Var}\left(W^{(i)}\right) \cdot \text{Var}(X).$$

   As this expression is independent of j we can define $\text{Var}\left(h^{(i)}_j\right) = \text{Var}\left(h^{(i)}\right)$ as the shared variance of the outputs like before.

5. (*1 point*) Now let's say that we want that the variance between layers does not change, i.e. $\text{Var}\left(h^{(i-1)}\right) = \text{Var}\left(h^{(i)}\right)$. If the input is normalized to have unit variance, what simple condition can we derive for the shared variance of the weights in layer i with respect to the number of neurons?

6. (*2 points*) Using the same approach, give a similar condition for $\text{Var}\left(W^{(i)}\right)$, but from a backpropagation point-of-view, i.e. assume

   $$\text{Var}\left(\frac{\partial h^{(n)}}{\partial h^{(i)}}\right) = \text{Var}\left(\frac{\partial h^{(n)}}{\partial h^{(i+1)}}\right).$$

2 Programming Exercises (15 points)

a) Vanishing Gradients (*9 points*)

In the theoretical exercises we have seen that the Sigmoid function limits the gradient exponentially in the number of layers. We can empirically show that by constructing a net and looking at the norm of the gradient vectors as a measure of how much change happens in a given layer.

1. (*3 points*) Take your FashionMNIST classifier from the last exercise sheet and train it for 20 epochs with the following layer configurations: `(50,)`, `(50, 30)`, `(50, 30, 30)`, `(50, 30, 30, 30)`. Your net should have `torch.nn.Sigmoid` as the activation function between all hidden layers. Use the `torch.optim.SGD` optimizer with a learning rate of $\eta = 0.005$. Record the norms of the gradients (per layer) of the last batch each iteration as well as the test accuracy.

2. (*1 point*) Plot the test accuracy of all four configurations. In separate plots, show the gradient norms over the epochs for all configurations, i.e. each plot should show one curve for the norm of the gradient of each layer changing over the epochs.

3. (*1 point*) Rerun the above experiment, this time using `torch.nn.ReLU` as the activation function. Also plot the results like above. What do you observe?

Maybe there is still a way to rescue Sigmoid as an activation function. In fact our derivation in theoretical exercise b) motivates the *Xavier initialization*. Luckily, PyTorch offers this as a built-in method in the `torch.nn.init` package.

4. (*2 points*) Try out the `xavier_normal_` initialization for the non-bias weights on the deepest net from above, again using the Sigmoid function as the activation. Plot the values like before.

5. (*2 points*) Research what fancy initialization scheme PyTorch is using by default and compare your results to it. Can you modify the parameters of the default initialization method so they better fit our setting? Does it help?

b) Adaptive Learning Rates (*6 points*)

Finally we can use the optimizer everyone loves: *Adam*.

1. (*1 point*) Improve the best CIFAR-10 classifier from the last sheet by throwing out the rusty SGD optimizer and start using `torch.optim.Adam`. Don't worry about the initial learning rate or other parameters, because the moment estimation is *adaptive* anyway. Run your training for 50 epochs and plot the test accuracy.

2. (*1 point*) Compare the accuracy with the results from last sheet. How big was the performance improvement with Adam?

3. (*2 points*) It is entirely possible (and implied above) that off-the-shelf Adam might not improve your results and you have to fine-tune parameters after all. Try to find better values for the initial learning rate using a simple grid search, with values 10^{-k} for $k \in \{1, 2, 3, 4, 5\}$. Take the final **training** accuracy as your comparison metric.

4. (*2 points*) Apparently, random search outperforms grid search. Sample five values for k uniformly at random from the real interval $[1, 5]$ and compare the results to the ones from grid search.