Exercises
Deep Learning
Fall 2018

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Fernando Pérez-Cruz
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

Series Monday, Nov 5, 2018
(Deep Learning, Exercise series 6)

Problem 1 (Fundamentals of Unconstrained Optimization):

1. Compute the gradient $\nabla f(w)$ and Hessian $\nabla^2 f(w)$ of the Rosenbrock function
$$f(w) = 100(w_2 - w_1^2)^2 + (1 - w_1)^2.$$
Show that $w^* = (1, 1)^\top$ is the only local minimizer of this function, and that the Hessian matrix at that point is positive definite.
2. Show that the function $f(w) = 8w_1 + 12w_2 + w_1^2 - 2w_2^2$ has only one stationary point, and that it is neither a maximum or minimum, but a saddle point. Sketch the contour lines of f.

Problem 2 (Gradient Descent step derivation):

Show that the Gradient Descent step in iteration t, namely

$$\boldsymbol{s}_t := \boldsymbol{w}_{t+1} - \boldsymbol{w}_t = -\frac{1}{L}\nabla f(\boldsymbol{w}_t), \quad L > 0 \tag{1}$$

minimizes a quadratically regularized first order Taylor expansion of the objective f around the current iterate $\boldsymbol{w}_t$. That is, show that the step $\boldsymbol{s}_t^*$ defined as

$$\boldsymbol{s}_t^* := \arg\min_{\boldsymbol{s}\in\mathbb{R}^d} \left[m_t(\boldsymbol{s}) := f(\boldsymbol{w}_t) + \boldsymbol{s}^\intercal \nabla f(\boldsymbol{w}_t) + \frac{L}{2}\|\boldsymbol{s}\|_2^2 \right] \tag{2}$$

equals $\boldsymbol{s}_t$ as given in Eq. (1). (As we will see next week, if we pick L greater or equal to the Lipschitz constant of the gradients, $m_t(\boldsymbol{s})$ is a global overestimator of f and hence s_t is guaranteed to be a descent step.)

Problem 3 (Programming exercise: Optimization methods in tensorflow):

In this exercise, you will implement three of the most common optimization methods in deep learning–SGD, Adagrad, and Adam–from scratch. In the supplementary material of this exercise, we provide a jupyter notebook skeleton for a simple MNIST classification model. The training and testing procedure is already implemented. The only thing left to do is the implementation of an optimization method. But instead of using the tensorflow API (e.g. tf.train.AdamOptimizer()) you are asked to implement the optimizers yourself. A skeleton of the necessary classes and methods is provided.

1. Implement Stochastic Gradient Descent (**SGD**) in the corresponding cell. The SGD update step is given by
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{w}_t} l(\boldsymbol{x}_t; \boldsymbol{w}_t) \tag{3}$$
where $\boldsymbol{w} \in \mathbb{R}^n$ are the model parameters, $\boldsymbol{x}_t$ is a randomly sampled *mini-batch* of the data and $l()$ your loss function. The provided signature of the method *apply_gradients* follows the naming convention for tensorflow optimizers. Try to first get familiar with how the gradients and variables are provided and how to assign new values to tensorflow variables.
2. Implement **Adagrad** [1] in the corresponding cell. Its update step is computed by
$$\boldsymbol{g}_t = \nabla_{\boldsymbol{w}} l(\boldsymbol{x}_t; \boldsymbol{w}_t) \tag{4}$$
$$G_t = G_{t-1} + \boldsymbol{g}_t \boldsymbol{g}_t^T \in \mathbb{R}^{n\times n} \tag{5}$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\, \mathrm{diag}(G_t + \epsilon I)^{-0.5} \boldsymbol{g}_t \tag{6}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ are the model parameters, $\boldsymbol{x}_t$ is a randomly sampled *mini-batch* of the data and $l()$ your loss function. As shown by the formulas, Adagrad uses individual step sizes for every parameter, which depend on the history of its gradients. The intuiton here is that the learning rate decays faster for dimensions that have already seen significatn updates.
Hint: For the implementation it is not necessary to compute the full matrix G_t but just its diagonal elements, which reduces to the sum of squared gradients in every dimension over time.

3. Implement **Adam** [2] in the corresponding cell. Its update step[1] is computed by

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{w}} l(\boldsymbol{x}_t; \boldsymbol{w}_t) \tag{7}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\left(\nabla_{\boldsymbol{w}} l(\boldsymbol{x}_t; \boldsymbol{w}_t)\right)^2 \tag{8}$$

$$\eta_t = \eta * \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \tag{9}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta_t}{\sqrt{\boldsymbol{v}_t} + \epsilon}\boldsymbol{m}_t \tag{10}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ are the model parameters, $\boldsymbol{x}_t$ is a randomly sampled *mini-batch* of the data and $l()$ your loss function.

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[1] Note that the following formula corresponds to Algorithm 1 in [2] with the proposed modification of section 2. This is the same algorithm that is used in the tensorflow implementation.