Exercises
**Deep Learning**
Fall 2020

**Machine Learning Institute**
Dept. of Computer Science, ETH Zürich
**Thomas Hofmann**
Web http://www.da.inf.ethz.ch/teaching/2018/DeepLearning/

# Series Wed, Oct 21, 2020
# (Deep Learning, Exercise series 5)

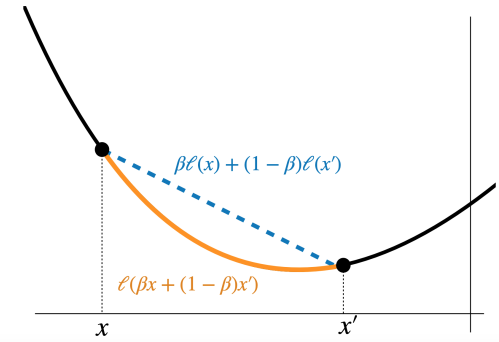
**Problem 1 (Characterizations of convex functions):**
In class, you have seen that a function $\ell : \mathbb{R}^d \to \mathbb{R}$ is convex if

$$\forall w, w' \in \mathbb{R}^d,\ \forall \lambda \in [0,1] : \ell(\lambda w + (1-\lambda)w') \leq \lambda \ell(w) + (1-\lambda)\ell(w'). \tag{1}$$

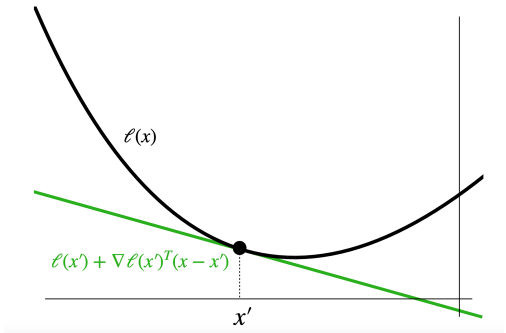
1. Show that, if $\ell$ is differentiable, this condition implies that

$$\forall w, w' \in \mathbb{R}^d : \ell(w) \geq \ell(w') + \nabla \ell(w')^T (w - w'). \tag{2}$$

2. (Home exercise) Show that, if $\ell$ is differentiable, conditions (1) and (2) are actually equivalent.



Definition in the lecture



This exercise

Now we assume that $\ell$ is also Lipschitz-smooth, i.e. $\|\nabla\ell(w) - \nabla\ell(w')\| \leq L\|w - w'\|, \forall w, w' \in \mathbb{R}^d$.
3. Show that this implies

$$\ell(w) \leq \ell(w') + \nabla\ell(w')^\top (w - w') + \frac{L}{2}\|w - w'\|^2 \tag{3}$$

*Hint: Use the fundamental theorem of calculus as well as Cauchy-Schwarz inequality.*

**Problem 2 (Gradient Descent step derivation):**
In iteration t, Gradient Descent takes the following step

$$s_t := w_{t+1} - w_t = -\frac{1}{\lambda}\nabla\ell(w_t), \quad \lambda > 0. \tag{4}$$

1. Show that this step minimizes a quadratically regularized first order Taylor expansion of the objective $f$ around the current iterate $w_t$. That is, show that the step $s_t^*$ defined as

$$s_t^* := \arg\min_{s \in \mathbb{R}^d} \left[ m_t(s) := \ell(w_t) + s^\mathsf{T}\nabla\ell(w_t) + \frac{\lambda}{2}\|s\|_2^2 \right] \tag{5}$$

equals $s_t$ as given in Eq. (7).
2. For what values of $\lambda$ is this step guaranteed to lower the objective $\ell$?

**Problem 3 (Stochastic Gradient Descent):**

In machine learning, our objective functions usually have an interesting finite-sum structure. That is,

$$\ell(w) = \frac{1}{n}\sum_{i}^{n} \ell_i(w, x_i) \tag{6}$$

where $x_i$ is the $i-$th datapoint in a given dataset. In this case, the computational cost of one GD step scales as $\mathcal{O}(d)$. One, obviously cheaper, alternative is to only compute the update based on the gradient of one specific datapoint. This is the updated of *stochastic* gradient decent (SGD), which is arguably the most widely used optimizer in ML:

$$s_t^{SGD} := -\frac{1}{\lambda}\nabla\ell_i(w_t), \quad i = 1, ..., n;\ \lambda > 0. \tag{7}$$

In each iteration, the datapoint $i$ is chosen uniformly at random such that $E[\nabla\ell_i(w_t)] = \nabla\ell(w_t)$.

1. In this regime, how many samples are left unseen in expectation after one epoch ($n$ iterations)?

As in Problem 1, assume that $l$ is lipschitz smooth.

2. Show that given $w_t$ and a constant stepsize $\alpha = 1/(2L)$, SGD does not converge to a critical point $w^*$, i.e.

$$E\left[\|w_{t+1} - w^*\|_2^2\right] \geq \frac{1}{(2L)^2} E\left[\|\nabla\ell_i(w_k)\|_2^2\right] \tag{8}$$

3. Name two possibilities to retain convergence.

**Problem 4 (Coding exercise: Adaptive gradient methods in TensorFlow (Home exercise)):**

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

$$\nabla_{t+1} = \nabla_t - \eta\nabla_{\nabla_t} l(x_t; \nabla_t) \tag{9}$$

   where $\nabla \in \mathbb{R}^n$ are the model parameters, $x_t$ 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** in the corresponding cell. Its update step is computed by

$$g_t = \nabla_{\nabla} l(x_t; \nabla_t) \tag{10}$$

$$G_t = G_{t-1} + g_t g_t^T \in \mathbb{R}^{n\times n} \tag{11}$$

$$\nabla_{t+1} = \nabla_t - \eta\,\mathrm{diag}(G_t + \epsilon I)^{-0.5} g_t \tag{12}$$

   where $\nabla \in \mathbb{R}^n$ are the model parameters, $x_t$ 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** in the corresponding cell. Its update step[1] is computed by

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\nabla l(x_t; \nabla_t) \tag{13}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\left(\nabla_\nabla l(x_t; \nabla_t)\right)^2 \tag{14}$$

$$\eta_t = \eta * \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \tag{15}$$

$$\nabla_{t+1} = \nabla_t - \frac{\eta_t}{\sqrt{v_t} + \epsilon} m_t \tag{16}$$

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

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