

ESE 546, FALL 2020
HOMEWORK 3
INSTRUCTOR SOLUTIONS

Solution 1. (a)

$$\begin{aligned} \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|^2 &\leq \langle \nabla f(x) - \nabla f(y), x - y \rangle \\ \Rightarrow \|\nabla f(x) - \nabla f(y)\|^2 &\leq L \langle \nabla f(x) - \nabla f(y), x - y \rangle \end{aligned}$$

From Cauchy Shwarz inequality $\langle u, v \rangle \leq |\langle u, v \rangle| \leq \|u\| \|v\|$, we have

$$\begin{aligned} \langle \nabla f(x) - \nabla f(y), x - y \rangle &\leq \|\nabla f(x) - \nabla f(y)\| \|x - y\| \\ \Rightarrow \|\nabla f(x) - \nabla f(y)\|^2 &\leq L \|\nabla f(x) - \nabla f(y)\| \|x - y\| \\ \Rightarrow \|\nabla f(x) - \nabla f(y)\| &\leq L \|x - y\| \end{aligned}$$

(b) Let

$$\begin{aligned} g(z) &= f(z) - \langle \nabla f(x), z \rangle \\ h(z) &= f(z) - \langle \nabla f(y), z \rangle \end{aligned}$$

We see that

$$\begin{aligned} \nabla g(x) &= 0 \\ \nabla h(y) &= 0 \end{aligned}$$

Now considering $g(y) - g(x)$, as a consequence of descent lemma we have :

$$\begin{aligned} g(y) - g(x) &\geq \frac{1}{2L} \|\nabla g(y)\| \\ \Rightarrow f(y) - f(x) - \langle \nabla f(x), y - x \rangle &\geq \frac{1}{2L} \|\nabla g(y)\| \end{aligned}$$

Therefore,

$$f(y) - f(x) - \langle \nabla f(x), y - x \rangle \geq \frac{1}{2L} \|\nabla f(y) - \nabla f(x)\|$$

Similarly,

$$f(x) - f(y) - \langle \nabla f(y), x - y \rangle \geq \frac{1}{2L} \|\nabla f(x) - \nabla f(y)\|$$

Adding the two above inequalities, we have

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \frac{1}{L} \|\nabla f(x) - \nabla f(y)\|$$

(c) From part (a), we have:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$$

By Cauchy Schwartz inequality,

$$(\nabla f(x) - \nabla f(y))^T(x - y) \leq L(x - y)^T(x - y) \quad \text{for all } x, y$$

It leads to the monotocity of $Lx - \nabla f(x)$, which is

$$(Lx - \nabla f(x) - (Ly - \nabla f(y)))^T(x - y) \geq 0 \quad \text{for all } x, y.$$

The monotocity of $Lx - \nabla f(x)$ is equivalent to the convexity of $\frac{L}{2}\|x\|_2^2 - f(x)$. If f is twice differentiable, it's equivalent to $L\mathbf{I} - \nabla^2 f(x) \succcurlyeq 0$ or $\lambda_{\max}(\nabla^2 f(x)) \leq L$ for all x .

The definition of strongly convex is the Jensens inequality for the function

$$h(x) = f(x) - \frac{m}{2}\|x\|_2^2$$

Similarly, f is strongly convex if and only if h is convex. If f is twice differentiable, h is convex if and only $\nabla^2 f(x) - m\mathbf{I} \geq 0$ or $\lambda_{\min}(\nabla^2 f(x)) \geq m$ for all x .

Remark Here is a wrong proof for the LHS of the inequality:

Using the definition of strong convexity with parameter m , Cauchy Shwarz inequality, we have:

$$\begin{aligned} \langle \nabla f(x) - \nabla f(y), x - y \rangle &\leq \|\nabla f(x) - \nabla f(y)\| \|x - y\| \\ &\Rightarrow m\|x - y\|^2 \leq \|\nabla f(x) - \nabla f(y)\| \|x - y\| \\ &\Rightarrow m\|x - y\|^2 \leq \|\nabla f(x) - \nabla f(y)\| \|x - y\| \\ &\Rightarrow m \leq \frac{\|\nabla f(x) - \nabla f(y)\|}{\|x - y\|} \\ &\Rightarrow m \leq \lim_{y \rightarrow x^+} \frac{\|\nabla f(y) - \nabla f(x)\|}{\|y - x\|} = \|\nabla^2 f(x)\| \end{aligned}$$

Therefore :

$$m \leq \|\nabla^2 f(x)\|$$

The proof is correct when $f : \mathbb{R}^d \rightarrow \mathbb{R}$ where $d=1$. However, if $d > 1$, the hessian matrix $\nabla^2 f(x)$ is not a scalar. Thus, $\lim_{y \rightarrow x^+} \frac{\|\nabla f(y) - \nabla f(x)\|}{\|y - x\|} \neq \|\nabla^2 f(x)\|$.

Solution 2.

$$\begin{aligned} \|y - (R \odot X)w\|_2^2 &= y^\top y - 2w^\top (R \odot X)^\top y + w^\top (R \odot X)^\top (R \odot X)w \\ \mathbb{E}_R[\|y - (R \odot X)w\|_2^2] &= y^\top y - 2w^\top \mathbb{E}_R[R \odot X]^\top y + w^\top \mathbb{E}_R[(R \odot X)^\top (R \odot X)]w \end{aligned}$$

From lecture note, we know $\mathbb{E}_R[R \odot X] = (1-p)X$, then

$$\begin{aligned} &= y^\top y - 2(1-p)w^\top X^\top y + w^\top \mathbb{E}_R[(R \odot X)^\top (R \odot X)]w \\ &= \|y - (1-p)Xw\|_2^2 - (1-p)^2 w^\top X^\top X w + w^\top \mathbb{E}_R[(R \odot X)^\top (R \odot X)]w \\ &= \|y - (1-p)Xw\|_2^2 + w^\top (\mathbb{E}_R[(R \odot X)^\top (R \odot X)] - (1-p)^2 X^\top X)w \end{aligned}$$

From lecture note, we know

$$(\mathbb{E}_R[(R \odot X)^\top (R \odot X)])_{ij} = \begin{cases} (1-p)^2(X^\top X)_{ij} & \text{if } i \neq j \\ (1-p)(X^\top X)_{ii} & \text{otherwise} \end{cases}$$

Then, the off-diagonal elements of $\mathbb{E}_R[(R \odot X)^\top (R \odot X)] - (1-p)^2 X^\top X$ are zeros.

$$\begin{aligned} \mathbb{E}_R[(R \odot X)^\top (R \odot X)] - (1-p)^2 X^\top X &= [(1-p)(1-p)^2] \text{diag}(X^\top X) \\ &= p(1-p) \text{diag}(X^\top X) \\ \mathbb{E}_R[\|y - (R \odot X)w\|_2^2] &= \|y - (1-p)Xw\|_2^2 + p(1-p)w^\top \text{diag}(X^\top X)w \end{aligned}$$

Let $\tilde{w} = (1-p)w$, then

$$= \|y - X\tilde{w}\|_2^2 + \left(\frac{p}{1-p}\right) \tilde{w}^\top \text{diag}(X^\top X) \tilde{w}$$

Finally, we have

$$\min_w \mathbb{E}_R[\|y - (R \odot X)w\|_2^2] = \min_{\tilde{w}} \|y - X\tilde{w}\|_2^2 + \left(\frac{p}{1-p}\right) \tilde{w}^\top \text{diag}(X^\top X) \tilde{w}$$

Solution 3. Proof 1: It is possible to find the Bayes optimal predictor by lower bounding the population risk, as we saw in the lecture, but this proof requires us to know the solution a priori and basically verify it. We fill in the details from the lecture for clarity. If $f^*(x) = \mathbb{E}[y|x]$, then:

$$\begin{aligned} R(f) &= \mathbb{E}_{x,y}[(f(x) - y)^2] = \mathbb{E}_{x,y}[(f(x) - f^*(x) + f^*(x) - y)^2] \\ &= \mathbb{E}_{x,y}[(f(x) - f^*(x))^2] + \mathbb{E}_{x,y}[(f^*(x) - y)^2] + 2\mathbb{E}_{x,y}[(f(x) - f^*(x))(f^*(x) - y)] \end{aligned}$$

We can eliminate the cross-term and simplify the expression:

$$\begin{aligned} \mathbb{E}_{x,y}[(f(x) - f^*(x))(f^*(x) - y)] &= \mathbb{E}_x[\mathbb{E}_y[(f(x) - f^*(x))(f^*(x) - y)|x]] \\ &= \mathbb{E}_x[(f(x) - f^*(x))\mathbb{E}_y[(f^*(x) - y)|x]] \\ &= \mathbb{E}_x[(f(x) - f^*(x))(f^*(x) - \mathbb{E}_y[y|x])] = 0 \end{aligned}$$

Then, observing that the first term does not depend on y :

$$R(f) = \mathbb{E}_x[(f(x) - f^*(x))^2] + \mathbb{E}_{x,y}[(f^*(x) - y)^2]$$

The second term, is a constant for our minimization problem. As we have seen many times by now, it is called the *Bayes risk* and depends only on the unknown distribution. It is a lower bound for the population risk.

$$R(f) \geq \mathbb{E}_{x,y}[(f^*(x) - y)^2], \forall f$$

The equality above holds if and only if the first term is 0. Looking at the first term, we know that when the integrand is non-negative and the integral is 0, then the integrand has to be 0 everywhere. If $P(x, y)$ zeroes out the integrand at some point (x, y) then at this point the function $f(x)$ can take any value. We say $f_{min}(x) = f^*(x) = \mathbb{E}[y|x]$ "almost everywhere", or "up to measure zero", to account for places with no probability mass.

Remark: Remember that since the beginning we assumed that our predictor is deterministic, ie. for any x it predicts a single y . This is true for the Bayes optimal predictor too, that takes the expectation of y 's weighted by $P(y|x)$ and outputs a single y for each x . The problem is that we do not know $P(y|x)$ and thus, even if we know the form of the optimal predictor, we cannot compute it. Moreover, if the generative process for the labels is not deterministic we expect the Bayes risk to be non zero.

Easy Case 1: (deterministic label gen. process): $P(y|x) = \delta(y - y^*(x))$. Then,

$$f^*(x) = \mathbb{E}[y|x] = \int y\delta(y - y^*(x))dy = y^*(x)$$

and the Bayes error is:

$$\int [\int (f^*(x) - y)\delta(y - y^*(x))dy]P(x)dx = \int (f^*(x) - y^*(x))P(x)dx = 0$$

Easy Case 2: ($P(x|y)$ are non-overlapping for different y 's). This is a question from the recitation, convince yourselves that it is equivalent to the case above.

Proof 2 As we said, the proof above works great only in cases where we have already guessed the solution. Remember, we try to minimize a *functional*; a function whose arguments are again functions. Another direction is, to treat each $f(x)$ as a real number and try to minimize the functional by minimizing the pointwise error. This is a stricter condition than minimizing the expectation (in the sense that it implies our condition, but it is not implied by it) and generally not possible, but here it is applicable because the error splits across different $f(x)$'s independently. If it didn't, it would create intertwined equations unsolvable and maybe contradictory. By minimizing the pointwise error, we change the optimization problem from,

$$\min_{f:\mathbb{R}^k \rightarrow \mathbb{R}} R(f) = \mathbb{E}_{x,y}[l(f(x), y)] = \mathbb{E}_x[\mathbb{E}_y[l(f(x), y)|x]]$$

to a number of new ones,

$$\forall x \in \mathbb{R}^k \min_{f(x) \in \mathbb{R}} \mathbb{E}_y[l(f(x), y)|x]$$

or for a particular x_p ,

$$\min_{w \in \mathbb{R}} \mathbb{E}_y[l(w, y)|x_p] = w^2 - 2w\mathbb{E}[y|x_p] + \mathbb{E}[y^2|x_p]$$

This is of course a convex quadratic function with unique minimum at $f^*(x_p) = w^* = \mathbb{E}[y|x_p]$, and since this holds for each x , we get that $f^*(x) = \mathbb{E}[y|x]$ is the only optimum for the second problem, and thus an optimum for the first.

If we are looking for all the solutions of the first problem, we better verify that this is the only one ("up to measure 0") by either using the first proof, or by using known results from convexity theory on the expectation of convex functions.

Even in cases where this method is applicable, observe that in principle, it may give a more complicated solution, for example a discontinuous predictor, even when a simpler one exists, because there is no information shared or constraints between the subproblems.

Proof 3: (Calculus of Variations) The field of mathematics that is concerned with solving *functional optimization* problems is called *calculus of variations*. Such problems arise naturally in nearly any scientific field. Some examples are:

- (1) **Geodesics:** Find the (locally)-minimum-length curve that connects two fixed points on some surface/manifold. <https://www.youtube.com/watch?v=NfqrCdAjiks>
- (2) **Brachistochrone:** Find the shape of the curve down which a bead sliding from rest and accelerated by gravity will slip (without friction) from one point to another in the least time. <https://www.youtube.com/watch?v=skvnj67YGmw>
- (3) **Maximum Entropy Distribution** in a given class: Leading to the principle of maximum entropy that is sometimes used in machine learning https://en.wikipedia.org/wiki/Principle_of_maximum_entropy
- (4) **Principle of least action** in physics, **shape of a hanging rope** etc.

To understand what happens to the functional when the argument (function) changes by a small amount, we define the *functional derivative* $\frac{\delta R}{\delta f(x)}$, by the following relation:

$$R(f(x) + \epsilon\eta(x)) = R(f(x)) + \epsilon \int \frac{\delta R}{\delta f(x)} \eta(x) dx + \mathcal{O}(\epsilon^2)$$

Now if we want R to be stationary at $f^*(x)$ the second term vanishes. Since this holds for all perturbations $\eta(x)$, then: $\int \frac{\delta R}{\delta f(x)} \eta(x) dx = 0 \Rightarrow \frac{\delta R}{\delta f(x)}|_{f^*(x)} = 0$. (To understand it just

imagine $\eta(x)$ being a dirac around each point x .

$$\begin{aligned} R(f(x) + \epsilon\eta(x)) &= \mathbb{E}_{x,y}[(f(x) + \epsilon\eta(x) - y)^2] \\ &= \mathbb{E}_{x,y}[(f(x) - y)^2] + \epsilon\mathbb{E}_{x,y}[2(f(x) - y)\eta(x)] + \epsilon^2\mathbb{E}_{x,y}[\eta(x)^2] \\ &= R(f(x)) + \epsilon \int_x 2(f(x) - \mathbb{E}[y|x])\eta(x)dx + \mathcal{O}(\epsilon^2) \\ \Rightarrow f^*(x) &= \mathbb{E}[y|x] \end{aligned}$$

Remark: In the more general case where $R(f) = \int L(f(x), f'(x), x)dx$ the necessary first order condition becomes:

$$\frac{\partial R}{\partial f} = \frac{d}{dx} \frac{\partial R}{\partial f'}$$

where the partial derivatives involved are ordinary partial derivatives. These equations are called: **Euler-Lagrange** equations and they are very useful for deriving critical functions in calculus of variations.

Remark: Keep in mind that during this whole derivation we did not constraint $f(x)$. If there are external or natural constraints (like optimizing over probability distributions as in the maximum entropy problem above), we need to incorporate them using Lagrangian coefficients.

Remark: Many of you used the derivative $\frac{\delta R}{\delta f}$, like in finite-dimensional optimization and forgot about the dependency on x . Conceptually, this is like $f(x)$ changes by some constant function $f(x) + \epsilon c$, which is just a special case of the perturbations $\eta(x)$. This is why this derivation resulted in a much looser necessary condition:

$$\mathbb{E}_x[f(x)] = \mathbb{E}_y[y](= E_x[E_y[y|x]])$$

Remark: Using the necessary condition above, we do not know if the critical function is a minimum. We can again verify that from proof 1) or use a *second variation* criterion. A sufficient condition is for example:

$$A(x)\eta^2 + 2B(x)\eta\eta' + C(x)(\eta')^2 > 0, \forall x \in \text{dom}(f), \eta \neq 0$$

where:

$$A(x) = \frac{\partial^2 L}{\partial f^2}|_{f=f^*(x)}, \quad B(x) = \frac{\partial^2 L}{\partial f \partial f'}|_{f=f^*(x)}, \quad C(x) = \frac{\partial^2 L}{\partial f'^2}|_{f=f^*(x)}$$

which for our problem translates to $2\eta^2 > 0$, which holds true.

Solution 4. Understanding the Resnet 18 architecture

- (a) Dividing the code into batches: Refer to HW3_4.PY file for more details:

```
● ● ●

1 def get_vector(text):
2     indices = [one_hot_map[ch] for ch in text]
3     vectorized = np.eye(char_num)[indices]
4     return vectorized, indices
5
6
7 # Convert text to vectors
8 vectorized_text, index_text = get_vector(content)
9 vectorized_text = torch.Tensor(vectorized_text)
10 index_text = torch.Tensor(index_text).long()
11
12 def get_batch(counter, gpu, text, labels):
13     x_batch = text[counter * 25: (counter+1) * 25]
14     y_batch = labels[counter * 25 + 1: (counter+1) * 25 + 1]
15     x_batch = x_batch.reshape(-1, char_num)
16     ln = min(len(x_batch), len(y_batch))
17     x_batch, y_batch = x_batch[:ln], y_batch[:ln]
18     if gpu:
19         x_batch = x_batch.cuda()
20         y_batch = y_batch.cuda()
21     return (x_batch, y_batch)
```

FIGURE 1. Convert the text to one-hot vectors and also create sequences of length 25

- (b) The entire code is present in HW3_4.PY. Some important snippets are as added below

```

1 class charRNN(torch.nn.Module):
2     def __init__(self, hid_dim, out_dim):
3         super(charRNN, self).__init__()
4         self.rnn = torch.nn.RNN(input_size=char_num,
5                                hidden_size=hid_dim,
6                                num_layers=1,
7                                nonlinearity='tanh',
8                                bias=True)
9         # add another embedding layer to improve training
10        # self.embed is an extra layer that is typically added.
11        # Doesn't need to be added for this assignment
12        self.embed = torch.nn.Linear(char_num, char_num)
13        self.linear = torch.nn.Linear(hid_dim, out_dim)
14        self.hid_dim = hid_dim
15
16    def forward(self, x, hidden):
17        embed = self.embed(x)
18        embed = embed.view(-1, 1, char_num)
19        out, hidden = self.rnn(embed, hidden)
20        out = out.view(-1, self.hid_dim)
21        out = self.linear(out)
22        return out, hidden

```

FIGURE 2. Create an RNN model

```

1 for ep in range(EPOCHS):
2     for it in range(num_batches):
3         net.train()
4         hidden = hidden.cpu().detach()
5         if use_gpu:
6             hidden = hidden.cuda()
7         optim.zero_grad()
8         data, target = get_batch(it, use_gpu, train_x, train_y)
9         output, hidden = net(data, hidden)
10        loss = criterion(output, target)
11        loss.backward()
12        torch.nn.utils.clip_grad_norm_(net.parameters(), 0.1)
13        optim.step()
14        train_loss += loss.item()

```

FIGURE 3. Train the RNN model

```
1 for t_it in range(num_val_batches):
2     hidden_val = hidden_val.cpu().detach_()
3     if use_gpu:
4         hidden_val = hidden_val.cuda_()
5     val_data, val_target = get_batch(t_it, use_gpu, val_x, val_y)
6     output, hidden_val = net(val_data, hidden_val)
7     loss = criterion(output, val_target)
8     total_loss += loss.item_()
9     out_val = np.reshape(output.cpu().detach_().numpy_(),
10                         (-1, char_num))
11     total_acc = np.mean(np.argmax(out_val, axis=1) ==
12                         val_target.cpu().numpy_())
13 total_loss = total_loss / num_val_batches
14 total_acc = total_acc / num_val_batches * 100
15 print("Validation loss at %d: %f %f" % (it, total_loss, total_acc))
```

FIGURE 4. Validation Loop