CS 189 Introduction to Machine Learning
Fall 2017 HW13

This homework is due **Friday, December 1 at 10pm.**

# 1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it:

*I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.*

## 2 Function approximation via neural networks

In this problem, you will prove a special case of a classical result in the approximation theory of neural networks. In particular, you will show that given a natural class of functions, a two-layer neural network (having just one hidden layer) with certain activation functions approximates every function in the class *exponentially* better than polynomial regression using the same number of coefficients. Clearly then, neural networks incur a smaller bias than polynomials for the same number of parameters. The related question of variance of neural network models is also interesting, but will not addressed by this problem.

The class of functions that we will be interested in approximating is given by the set

$$\mathscr{F}_{\mathsf{cos}} = \left\{ f : \mathbb{R}^d \to \mathbb{R} \text{ s.t. } f(x) = \frac{1}{2\|w\|_1} \cos(\langle w, x \rangle) \text{ for some } w \neq \mathbf{0} \right\}.$$

Here, $\mathbf{0}$ denotes the zero vector in $\mathbb{R}^d$.

Define a thresholding function $\tau : \mathbb{R} \mapsto \mathbb{R}$ as any bounded function for which $\tau(z) \to 1$ as $z \to \infty$ and $\tau(z) \to 0$ as $z \to -\infty$. We will consider a neural network function $h : \mathbb{R}^d \mapsto \mathbb{R}$ of the form

$$h(x) = \sum_{k=1}^{p} c_k \tau(\langle w_k, x \rangle + b_k),$$

where $(w_k, b_k, c_k) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$ for each $k = 1, 2, \ldots, p$. Denote the class of such functions having $p$ parameters and $\sum_{k=1}^{p} |c_k| \leq 1$ by $\mathscr{H}_p$.

Given any function $f \in \mathscr{F}_{\mathsf{cos}}$, we will show that it can be approximated effectively by a linear combination of a small number of threshold functions. In particular, we will measure the quality of approximation by the average error over the set $[0,1]^d$ as

$$E(f, p) = \inf_{h \in \mathscr{H}_p} \int_{x \in [0,1]^d} (f(x) - h(x))^2 dx.$$

In this problem, you will show that for all functions $f \in \mathscr{F}_{\mathsf{cos}}$, we have

$$E(f, p) \leq \frac{1}{p},$$

thus showing that we can approximate any function in $\mathscr{F}_{\mathsf{cos}}$ to within error $\varepsilon$ using at most $1/\varepsilon$ linear combinations of threshold functions. Since each threshold function involves $d+2$ parameters, we will have shown that an $\varepsilon$-approximation is achievable using $O\left(\frac{d}{\varepsilon}\right)$ scalar parameters.

In contrast, for polynomial approximators of the form

$$\phi(x) = \sum_{\substack{(k_1, k_2, \ldots, k_d) \\ m := \sum_j k_j \leq \ell}} w_m \left( \prod_{i=1}^{d} x_i^{k_i} \right)$$

for some $w \in \mathbb{R}^p$ with $p = \binom{d+\ell}{\ell}$, we will show that at least $\left(\frac{1}{\varepsilon d}\right)^d$ parameters are necessary to obtain a similar $\varepsilon$-approximation.

Parts (c-h) will walk you through a proof that linear combinations of thresholding functions (examples of which you will see in part (a)) are good approximators of functions in $\mathscr{F}_{\mathsf{cos}}$. You will see examples of functions in $\mathscr{F}_{\mathsf{cos}}$ in part (b). We will release a note on Piazza showing the approximation lower bound for polynomial functions mentioned above, so you have an idea why polynomials perform worse.

**Disclaimer:** We will deliberately avoid some mathematical subtleties in the following proof that are essential to handle in order for it to qualify as a complete argument. The goal here is for you to understand the concrete higher order message.

(a) **Verify that the following commonly used functions are thresholding functions.**

  (a) $\sigma(t) = \frac{1}{1+e^{-t}}$

  (b) $\mathsf{ReLu}(t) - \mathsf{ReLu}(t-1)$

(b) Fix $d = 2$, and **show 3D plots of functions in the class $\mathscr{F}_{\mathsf{cos}}$, for five sufficiently different choices of** $w$; you will be plotting $f(x)$ for $x \in [0,1]^2$. The goal is for you to convince yourself that these are not pathologically constructed functions; they look like many underlying regression functions we might want to approximate in practice. What is true but will not be shown in this problem is that many practical functions of interest including Gaussian PDFs and functions with sufficiently many derivatives can be expressed as a linear combination of functions in $\mathscr{F}_{\mathsf{cos}}$.

(c) Define the closure $\mathsf{cl}(T)$ of a set $T$ as the union of $T$ with its limit points, e.g. the closure of the set $\{x \in \mathbb{R}^d : \|x\|_2 < 1\}$ is given by the set $\{x \in \mathbb{R}^d : \|x\|_2 \leq 1\}$.

Define the step function $S(t)$ to be the indicator function that $t \geq 0$, i.e., $S(t) = 1$ for all $t \geq 0$, and 0 otherwise. **Show that for each $(w', b') \in \mathbb{R}^d \times \mathbb{R}$, we have**

$$S(\langle w', x\rangle + b') \subseteq \mathsf{cl}\left(\{\tau(\langle w, x\rangle + b) \text{ for some } w, b\}\right).$$

Hint: It should be sufficient to show that threshold functions with appropriately scaled arguments are step functions in the limit. Show this, and argue why it is sufficient.

(d) We will now try to approximate the function $\cos(\langle w, x\rangle)$ with a linear combination of step functions of the form $S(\langle w, x\rangle)$. Notice that it is sufficient to simply approximate $\cos(y)$ by $S(y)$, since the arguments of the two functions are identical. Also notice that since we only care about approximating the function within the set $\|x\|_\infty \leq 1$, we have by Holder's inequality that $y = \langle w, x\rangle \leq \|w\|_1 \|x\|_\infty = 1$. Executing a change of variables $y' = y/\|w\|_1$, it suffices now approximate the function $c(y') = \cos(\|w\|_1 y')$ within the range $y' \in [-1, 1]$.

**Show that for any fixed $\delta > 0$, we can design a linear combination of step functions that is $\delta$-close to $c(y')$ for every $y' \in [-1, 1]$.**

Hint: Recall how a continuous function can be successively approximated by finer and finer piecewise constant functions. It may be helpful to define a finite collection of points $P_\delta$ on the set $[-1, 1]$ such that $c(y')$ changes its value by at most $\delta$ for successive points.

(e) Let us consider the absolute sum of coefficients of the linear combination you computed above. You should have computed it to be

$$\sum_i |c(z_i) - c(z_{i-1})|,$$

where the points $\{z_i\}_i$ are the points defined in $P_\delta$. **Show that this sum is bounded by $\|w\|_1$.** Conclude that for every $w \neq \mathbf{0}$, we have approximated $\cos(\langle w, x\rangle)$ by a linear combination of step functions having sum of absolute coefficients at most $\|w\|_1$.

Hint: Finite differences are derivatives in the limit. The sum can be upper bounded by an appropriate integral.

(f) Denote the closed convex hull of a set $T$ by $\overline{\mathsf{conv}}(T)$; this is simply the closure of the convex hull of $T$. Using the previous parts, **show that $\mathscr{F}_{\mathsf{cos}} \subseteq \overline{\mathsf{conv}}\left(\{\tau(\langle w, x\rangle + b)$ for some $w, b\}\right)$.** We have thus shown that the function class of interest can be represented by a convex combination of threshold functions.

(g) How many threshold functions do you need to approximate any function $f \in \mathscr{F}_{\mathsf{cos}}$ to within some error $\varepsilon$? This will be shown in the next two parts via an existence argument.

Notice that roughly speaking, we have by definition of the convex hull that $f = \sum_{i=1}^m c_i g_i^*$ for some $m$ functions $g_i^* = \phi(\langle w_i, x\rangle + b)$, and $c_i \geq 0$ with $\sum_i c_i = 1$. Let $G$ be a random variable such that $G = g_i^*$ with probability $c_i$. With $p$ independent samples $G_1, G_2, G_3, \ldots$, define

$$f_p = \frac{1}{p} \sum_{i=1}^p G_i.$$

Let $\|g - h\|^2 := \int_{x \in [0,1]^d} (g(x) - h(x))^2 dx$ for any two functions $g$ and $h$. **Show that**

$$\mathbb{E}[\|f_p - f\|^2] \leq \frac{1}{p}, \tag{1}$$

where the expectation is taken over the random samples $G_1, \ldots, G_p$. You may assume that expectations commute with integrals, i.e., you can switch the order in which you take them.

(h) Use the above part to argue that **there must exist a convex combination of $p$ threshold functions such that equation** (1) **is true for a deterministically chosen $f_p$.** Conclude that for all functions $f \in \mathscr{F}_{\mathsf{cos}}$, we have

$$E(f, p) \leq \frac{1}{p}.$$

# 3 CNNs on Fruits and Veggies

In this problem, we will use the dataset of fruits and vegetables that was collected in HW5 and HW6. The goal is to accurately classify the produce in the image. In prior homework, we explored how to select features and then use linear classification to learn a function. We will now

explore using Convolutional Neural Networks to optimize feature selection jointly with learning a classification policy.

Denote the input state $x \in \mathbb{R}^{90\times90\times3}$, which is a down sampled RGB image with the fruit centered in it. Each data point will have a corresponding class label, which corresponds to their matching produce. Given 25 classes, we can denote the label as $y \in \{0,...,24\}$.

The goal of this problem is twofold. First you will learn how to implement a Convolutional Neural Network (CNN) using TensorFlow. Then we will explore some of the mysteries about why neural networks work as well as they do in the context of a bias variance trade-off.

Note all python packages needed for the project, will be imported already. **DO NOT import new Python libraries.** Also, this project will be computationally expensive on your computer's CPU. Please message us on Piazza if you do not have a strong computer and we can arrange for you to use EC2.

(a) To begin the problem, we need to implement a CNN in TensorFlow. In order to reduce the burden of implementation, we are going to use a TensorFlow wrapper known as *slim*. In the starter code is a file named *cnn.py*, the network architecture and the loss function are currently blank. Using the slim library, you will have to write a convolutional neural network that has the following architecture:

   (a) Layer 1: A convolutional layer with 5 filters of size 15 by 15
   (b) Non-Linear Response: Rectified Linear Units
   (c) A max pooling operation with filter size of 3 by 3
   (d) Layer 2: A Fully Connected Layer with output size 512.
   (e) Non-Linear Response: Rectified Linear Units
   (f) Layer 3: A Fully Connected Layer with output size 25 (i.e. the class labels)
   (g) Loss Layer: Softmax Cross Entropy Loss

   In the file *example_cnn.py*, we show how to implement a network in TensorFlow Slim. Please use this as a reference. Once the network is implemented **run the script** *test_cnn_part_a.py* **on the dataset and report the resulting confusion matrix**. The goal is to ensure that your network compiles, but we should not expect the results to be good because it is randomly initialized.

(b) The next step to train the network is to complete the pipeline which loads the datasets and offers it as mini-batches into the network. **Fill in the missing code in** *data_manager.py* **and report your code.**

(c) We will now complete the iterative optimization loop. Fill in the missing code in *trainer.py* to iteratively apply SGD for a fix number of iterations. In our system, we will be using an extra Momentum term to help speed up the SGD optimization. **Run the file** *train_cnn.py* **and report the resulting chart.**

(d) To better understand, how the network was able to achieve the best performance on our fruits and veggies dataset. It is important to understand that it is learning features to reduce the dimensionality of the data. We can see what features were learned by examining the response maps after our convolutional layer.

The response map is the output image after the convolutional has been applied. This image can be interpreted as what features are interesting for classification. **Fill in the missing code in** $viz\_features.py$ **and report the images specified**.

(e) Given that our network has achieved high generalization with such low training error, it suggests that a high variance estimator is appropriate for the task. To better understand why the network is able to work, we can compare it to another high variance estimator such as nearest neighbors. **Fill in the missing code in** $nn\_classifier.py$ **and report the performance as the numbers of neighbors is swept across when** $train\_nn.py$ **is run.**

# 4 Your Own Question

**Write your own question, and provide a thorough solution.**

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.