Homework 1

CS 329T: Trustworthy Machine Learning Spring 2021

Due: April 9, 11:59 PM PST
Late Due Date: April 11, 11:59 PM PST.

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to Stanford Honor Code.

Late Day Policy

You will have **5** late days to use during the whole course, but no more than **2** late days can be used for a single homework.

Written Exercises

This homework contains some written exercises. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: `https://www.overleaf.com`) and Markdown are two recommended languages to generate the clean layout but you are free to use other software. You will need to submit the written part to **Homework1-PDF** on Gradescope, separately from you code submission. You can use the LaTeXtemplate provided to fill in the answers. Mention your name and SUNet Id at the top of the pdf.

Coding Exercises

You will be doing the coding part of the homwework in a jupyter notebook `solutions.ipynb`, provided as part of the zip folder. We recommend using Google Colab since one of the exercises needs a GPU, which is available for free via Colab. To open the notebook in Colab, go to `http://colab.research.google.com/`, and upload the notebook using *File* → *Upload Notebook*. Submit the completed `solutions.ipynb` as a jupyter notebook to **Homework1-Code** on Gradescope. **Make sure that the file is called `solutions.ipynb` or else the autograder will fail**.

Contents

1 Logistic Regression **3**

1.1 Hand-made SGD 4

2 Applications with Linear Models **4**

2.1 Explanations 4

2.2 Model stealing 5

2.3 Adversarial attacks 5

3 Intensive training using GPU **5**

1 Logistic Regression

For this homework you will need to review the slides and background reading on logistic regression, softmax, cross-entropy loss, and gradient descent.

Logistic regression is a probabilistic, linear classifier defined by

- a weight matrix $\mathbf{W}$
- a bias vector $\mathbf{b}$
- scoring function: given input instance $\mathbf{x}$, the score for class j is $\mathbf{s}_j \overset{\text{def}}{=} \mathbf{W}_j^T\mathbf{x} + \mathbf{b}_j$,
- softmax normalization: given scores $\mathbf{s} = \langle \mathbf{s}_1, \cdots, \mathbf{s}_n \rangle$, the class probabilities over n classes, denoted $\mathbf{y}'$, are the vector $\mathbf{y}' \overset{\text{def}}{=} \text{softmax}(\mathbf{s}) \overset{\text{def}}{=} \frac{1}{\sum_k \exp(\mathbf{s}_k)} \exp(\mathbf{s})$

Putting all this together, the probability of class j given input $\mathbf{x}$ is:

$$P(y' = j|\mathbf{x}, \mathbf{W}, \mathbf{b}) = \frac{\exp\left(\mathbf{W}_j^T\mathbf{x} + \mathbf{b}_j\right)}{\sum_k \exp\left(\mathbf{W}_k^T\mathbf{x} + \mathbf{b}_k\right)}$$

In the probabilistic notation, we use y' to refer to the class instead of one-hot vector encoding of class or class probabilities. Prediction is done by taking the class of highest probability:

$$y_{pred} \overset{\text{def}}{=} \text{argmax}_i P(y' = i|\mathbf{x}, \mathbf{W}, \mathbf{b})$$

For each input vector $\mathbf{x}$ and one-hot-encoded ground truth vector $\mathbf{y}$ in dataset $\mathbf{X},\ \mathbf{Y}$, the cross-entropy loss is

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b}) &\overset{\text{def}}{=} -\sum_j \mathbf{y}_j \log \mathbf{y}'_j \\ &= -\mathbf{y} \cdot \log \mathbf{y}' \end{aligned}$$

where $\cdot$ refers to dot product.

In this homework, you will not be processing each instance at a time but instead handle a subset ("batch") of the training dataset. Let $\mathbf{X}$ be the input matrix consisting of all the images in one training batch. $\mathbf{X}_i$ is the i-th image represented by the i-th column. Let $\mathbf{Y}$ be the ground truth matrix one-hot-encoded and $\mathbf{Y}'$ be the estimated probability matrix. Let N be the batch size. For one batch of 100 images of size 28*28 in 10 classes, the dimensions of $\mathbf{X}$ and $\mathbf{Y}$ are $(784, 100)$ and $(10, 100)$, respectively. The loss function for a batch is thus:

$$\begin{aligned} \mathbf{L}(\mathbf{X}, \mathbf{Y}; \mathbf{W}, \mathbf{b}) &\overset{\text{def}}{=} -\frac{1}{N}\sum_i \mathbf{Y}_i \cdot \log \mathbf{Y}'_i \\ &= -\frac{1}{N}\sum_i\sum_j \mathbf{Y}_{i,j} \log \mathbf{Y}'_{i,j} \end{aligned}$$

Now we have the relationship between $\mathbf{L}$ and $\mathbf{W}$,$\mathbf{b}$, by using the chain rule, we can derive the gradient $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{L}}{\partial \mathbf{b}}$. Most deep learning software calculates the gradients automatically for you. For the numpy version of the exercises in this homework, you will need to derive it by yourself. After deriving the gradient values, we can perform stochastic gradient descent to find the optimal $\mathbf{W}$ and $\mathbf{b}$.

1.1 Hand-made SGD

In some highly-specialized settings, automated tools may be unable to compute the gradient of your loss function. In these cases you would need to further lower your level of abstraction to implement backpropagation. The coding exercises call for a hand-made implementation of SGD for logistic regression. To help you get started, lets derive the gradients you will need to employ in the procedure which you will then hard-code into your code.

Written Exercise 1. *Derivation*

Derive the gradient of loss in terms of $\boldsymbol{W}$ and $\boldsymbol{b}$: $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}}$ and $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{b}}$. Show your work and make sure the dimensions of your vectors are consistent with the ones in the problem description.

Coding Exercise 2. *Complete the methods for logistic regression using scikit-learn, keras, and numpy in `solutions.ipynb`. Your solution to the derivation exercise should be helpful in your numpy implementation.* ***Note: The formulations for X and Y are flipped in the notebook. X is of size (N, 784) and Y is of size (N, 10), where N is the batch size.***

2 Applications with Linear Models

The simplicity and definition of linear models are convenient for a variety of reasons: methods which we will cover later in the course are trivial when applied to linear models. In the next set of exercises, you will cover some of those applications.

2.1 Explanations

A form of explanation for a prediction of a model is the *attribution* which assigns to each input a real value representing how important that input was to a prediction. Given a model $f : \mathcal{X}^n \to \mathbb{R}$, the *attribution for* $f(\boldsymbol{x}) = y$ is a real vector/matrix of the same shape as $\mathbf{x}$. Not just any real vector would do, however. Generally we would like to interpret positive values as indicating input dimensions that contributed to the outcome whereas negative values as indicating to diminishing the outcome. Further, larger magnitude values should indicate greater importance.

Another property that is sought for attributions is *completeness*: Given an input $\mathbf{x}$ and a baseline input $\mathbf{x'}$, an attribution $\mathbf{a} \in \mathbb{R}^n$ for $f(\mathbf{x}) = y$ (for example, probability of a certain class in logistic regression) is complete with respect to baseline $\mathbf{x'}$ iff $\mathbf{a} \odot (\mathbf{x} - \mathbf{x'}) \overset{\text{def}}{=} \sum_i \mathbf{a}_i(\mathbf{x}_i - \mathbf{x'}_i) = f(\mathbf{x}) - f(\mathbf{x'})$. That is, the attribution is a *complete* account of the difference in outcomes between $\mathbf{x}$ and the baseline $\mathbf{x'}$.

In general it may be impossible to define an attribution which is complete for multiple baselines. For linear models, however, this is not a problem.

Written Exercise 3. *Given a pre-softmax logistic regression model $f : \boldsymbol{x} \mapsto \left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right)$, an input $\boldsymbol{x}$ and, class index c, define an attribution $\boldsymbol{a}$ for $f(\boldsymbol{x})_c = y$ that is complete for all baselines.*

Coding Exercise 4. *Complete the methods `attribution` and `explain` in Part 2 of `solution.ipynb` that produces attributions and explanations for pre-softmax logistic model predictions. An explanation is the element-wise product of an input $\boldsymbol{x}$ and the attribution $\boldsymbol{a}$ for a given prediction. The attribution used should be the complete one from the prior exercise.*

2.2 Model stealing

Machine learning as a service is a means of monetizing machine-learnt models. In MLaaS, users pay to access a model's predictions without getting access to the model itself (i.e. the architecture and parameters that define it). However, it may be possible to recover a model using its predictions or perform the *model stealing* attack.

Coding Exercise 5. *Implement a model stealing attack in Part 2 of* `solutions.ipynb` *by completing the* `invert` *method. The method is given functional access to the pre-softmax score of a logistic regression and should output the model with parameters identical to the one model implementing the given functional interface.*

Written Exercise 6. *Is it possible to implement the attack in the prior exercise given access to post-softmax probabilities? If no, how would you adjust the exercise to make it possible while still being able to call it a "model stealing" attack?*

2.3 Adversarial attacks

Deep vision networks are susceptible to adversarial attacks: inputs that are visually indistinguishable from benign images that nonetheless fool a model into making a wrong prediction. In the context of this homework, given a model f, an input image $\mathbf{x}$, and target class c, an adversarial image $\mathbf{x'}$ is one which does not differ from original $\mathbf{x}$ but which has $f(\mathbf{x'})_c \geq max_i f(\mathbf{x'})_i$.

Coding Exercise 7. *Complete the* `attack` *method in Part 2 of* `solution.ipynb` *to implement an adversarial attack. There are many ways of doing this. We only ask that your solution is not trivial in that it attempts to limit in some way the difference between the given image and the adversarial image. The L_* metrics in* `test_attack` *may be useful to gauge progress towards this goal.*

Written Exercise 8. *Noted above, we can use $L_*(\boldsymbol{x} - \boldsymbol{x'})$, for various bases $*$, to measure how close the adversarial example is to the original. Pick a base from $* \in 0, 1, 2, \infty$ and describe a pair of images which are different according to the L_* but are actually close when it comes to human perception (i.e. they are close to indistinguishable).*

3 Intensive training using GPU

Real-world deep models are typically trained over large datasets and feature millions of parameters. Training them or even performing forward passes is computationally intensive. In this exercise we ask you to train a model using a GPU, achieving a significant speedup over using your general CPU hardware. For Colab, you can do this by going to *Runtime→Change Runtime Type→GPU.*

Coding Exercise 9. *Complete* `train_cifar_model` *to train a model for classifying CIFAR images. CIFAR images are slightly larger than MNIST and are of a more difficult domain.*

Your solution should be able to complete training under the given time constraints and achieve the given test accuracy results as specified in `test_cifar_model`*. The CIFAR dataset used in this test is not flattened as was in the case for MNIST which makes it immediately suitable for models with convolutional layers.*

Not using the GPU for this exercise is not expected to achieve this goal.

References

[1] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Mueller. How to explain individual classification decisions, 2009.

[2] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016. doi: 10.1109/SP.2016.42.

[3] Klas Leino, Linyi Li, Shayak Sen, Anupam Datta, and Matt Fredrikson. Influence-directed explanations for deep convolutional networks. *arXiv preprint arXiv:1802.03788*, 2018.

[4] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[5] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[6] Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation, 2020.

[7] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[8] Zifan Wang, PiotrPiotr Mardziel, Anupam Datta, and Matt Fredrikson. Interpreting interpretations: Organizing attribution methods by criteria, 2020.