Homework 3 - Version 1.0

Deadline: Thurs, Mar.11, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3]. *Note that 10% of the homework mark (worth 1 pt) may be removed for a lack of neatness.*

The teaching assistants for this assignment are Alex Adam and Jenny Xuchen Bao.
mailto:csc413-2021-01-tas@cs.toronto.edu

[1] https://markus.teach.cs.toronto.edu/csc413-2021-01
[2] https://csc413-uoft.github.io/2021/assets/misc/syllabus.pdf
[3] https://piazza.com/class?nid=kjt32fc0f7y3kb

1 Robustness and Regularization

Adversarial examples plague many machine learning models, and their existence makes the adoption of ML for high-stakes applications undergo increasingly more regulatory scrutiny. The simplest way to generate an adversarial examples is using the untargeted fast gradient sign method (FGSM) from [Goodfellow et al., 2014]:

$$\mathbf{x}' \leftarrow \mathbf{x} + \epsilon \, \text{sgn}(\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}\,;\mathbf{w}), y))$$

where $\mathbf{x} \in \mathbb{R}^d$ is some training example we want to perturb, y is the label for that example, and ϵ is a positive scalar chosen to be small enough such that the ground truth class of $\mathbf{x}'$ is the same as that of $\mathbf{x}$ according to human perception, yet large enough such that our classifier f misclassifies $\mathbf{x}'$ while correctly classifying $\mathbf{x}$. Read about how the sgn() function works here (`https://en.wikipedia.org/wiki/Sign_function`).

Note that we are taking the gradient of $\mathcal{L}(f(\mathbf{x}\,;\mathbf{w}), y)$ with respect to the *input* $\mathbf{x}$ instead of the weights $\mathbf{w}$, and that we are adding this gradient rather than subtracting it since the goal here is to increase the loss on $\mathbf{x}'$.

For the rest of the question, we assume we are dealing with a binary linear classifier that outputs a scalar logit as follows:

$$f(\mathbf{x}\,;\mathbf{w}) = \mathbf{w}^\top \mathbf{x},$$

where $\mathbf{w} \in \mathbb{R}^d$ where d is dimension of the input $\mathbf{x}$, so $f : \mathbb{R}^d \to \mathbb{R}$. For the remainder of the question, we ignore the loss function, and simply try to reduce the output predicted by the classifier f.

To simplify our analysis, assume that the linear classifier outputs a positive logit on the input $\mathbf{x}$, $\mathbf{w}^\top \mathbf{x} > 0$. The attack is performed on the outputted logits directly to change the model's prediction from positive to negative. The attack now becomes:

$$\mathbf{x}' \leftarrow \mathbf{x} - \epsilon \, \text{sgn}(\nabla_{\mathbf{x}} f(\mathbf{x}\,;\mathbf{w})),$$

where we are trying to decrease the outputted logit.

1.1 Adversarial Examples

1.1.1 Bounding FGSM [0pt]

To understand why the sgn() function is used, compute the ℓ^∞ norm of $\delta_1 = \epsilon \, \text{sgn}(\nabla_{\mathbf{x}} f(\mathbf{x}\,;\mathbf{w}))$ and $\delta_2 = \epsilon \, \nabla_{\mathbf{x}} f(\mathbf{x}\,;\mathbf{w})$. Under what conditions does $||\delta_1||_\infty = ||\delta_2||_\infty$? What guarantee does the sgn() function give us on the ℓ^∞ norm of the perturbation?

1.1.2 Prediction under Attack [1pt]

If we remove the sgn() function from the FGSM, we are left with just the FGM

$$\mathbf{x}' \leftarrow \mathbf{x} - \epsilon \, \nabla_{\mathbf{x}} f(\mathbf{x}\,;\mathbf{w})$$

Let us construct $\mathbf{x}'$ using the FGM. Write down the model output under the adversarial attack $f(\mathbf{x}'\,;\mathbf{w})$ as a function of $\epsilon, \mathbf{x}, \mathbf{w}$ in a closed form.

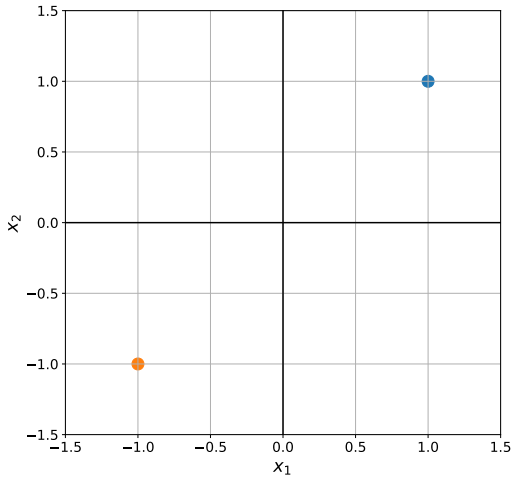

Figure 1: 2D plot of the training data.

1.2 Gradient Descent and Weight Decay

The most trivial though impractical way of making a classifier robust to adversarial examples is to set $\mathbf{w} = \mathbf{0}$ such that $f(\mathbf{x}\,;\mathbf{w}) = 0$ for any $\mathbf{x}$. However, this just computes a constant function, and is not useful. Intuitively, it looks like the smaller the norm of $\mathbf{w}$, then less the output will change when changing $\mathbf{x}$. We explore if this is always the case.

Suppose we have a design matrix $X \in \mathbb{R}^{n\times d}$ where n is the number of samples and d is the dimensionality, and a target vector $\mathbf{t} \in \mathbb{R}^n$. We can define the objective of linear regression as

$$\min_{\mathbf{w}} \frac{1}{2n}||X\mathbf{w} - \mathbf{t}||_2^2$$

If we penalize the squared ℓ^2 norm of the weights, we end up with ridge regression:

$$\mathbf{w}^*_{ridge} = \arg\min_{\mathbf{w}} \frac{1}{2n}||X\mathbf{w} - \mathbf{t}||_2^2 + \lambda||\mathbf{w}||_2^2,$$

where λ is the weight decay coefficient, $\lambda > 0$.

1.2.1 Toy Example [0pt]

Consider the following dataset $\mathcal{D} = \{(\underbrace{(1,1)}_{\mathbf{x}_1}, \underbrace{1}_{t_1})(\underbrace{(-1,-1)}_{\mathbf{x}_2}, \underbrace{-1}_{t_2})\}$ visualized in Figure 1 for your convenience. Draw the set of optimal solutions $\mathbf{w}^*$ (it's a line) in weight space (which is different from Figure 1 which is in data space) with $\mathbf{w}_1^*$ on the x-axis, and $\mathbf{w}_2^*$ on the y-axis. On the same plot, draw the contours of $\mathbf{w}^\top\mathbf{w}$. Is there a contour value for which the intersection between the set of optimal solutions and $\mathbf{w}^\top\mathbf{w}$ is a single point? If so, provide the coorindate of the point. What does this imply about the uniqueness of ridge regression solution for this dataset?

1.2.2 Closed Form Ridge Regression Solution [1pt]

Recall the solution to plain regression is $\mathbf{w}^* = (X^\top X)^{-1}X^\top\mathbf{t}$. Write down the closed-form solution to ridge regression in matrix form, $\mathbf{w}^*_{ridge}$. Show your work.

1.2.3 Adversarial Attack under Weight Decay [1pt]

Previously, we derived model output under the FGM adversarial attack $f(\mathbf{x}' ; \mathbf{w})$ without the sign function. Here, let us consider attacking the ridge regression solution. For any adversarial attacks, we first need to choose the appropriate amount of adversarial perturbation added to the original inputs. In FGM, the perturbation amount is decided by setting ϵ, larger ϵ corresponds to larger perturbation. So, how much perturbation is necessary to fool the model to output zero, that is $f(\mathbf{x}' ; \mathbf{w}^*_{ridge}) = 0$, with weight decay?

To answer this question concretely, let us consider a 1-D model that takes a scalar input $x \in \mathbb{R}$ and a scalar weight $w^*_{ridge} \in \mathbb{R}$,

$$x' \leftarrow x - \epsilon \, \nabla_x f(x ; w^*_{ridge}).$$

Derive the analytical closed form of ϵ as a function of the weight decay coefficient λ such that $f(x' ; w^*_{ridge}) = 0$. Show your work. Does weight decay make the model more robust under FGM attack? Why?

(Hint: Substitute your 1.2.2 solution into 1.1.2 final form then set the equation to zero. Simplify.)

1.2.4 The Adversary Strikes Back [0pt]

Now consider the 1-D case again under for the Fast Gradient Sign Method (FGSM) by including the sign function in the perturbation:

$$x' \leftarrow x - \epsilon \, \mathrm{sgn}\big(\nabla_x f(x ; w^*_{ridge})\big).$$

Does weight decay make the model more robust under FGSM attack? Why?

2 Trading off Resources in Neural Net Training

2.1 Effect of batch size

When training neural networks, it is important to select an appropriate batch size. In this question, we will investigate the effect of batch size on some important quantities in neural network training.

2.1.1 Batch size vs. learning rate

Batch size affects the stochasticity in optimization, and therefore affects the choice of learning rate. We demonstrate this via a simple model called the noisy quadratic model (NQM). Despite the simplicity, the NQM captures many essential features in realistic neural network training [Zhang et al., 2019].

For simplicity, we only consider the scalar version of the NQM. We have the quadratic loss $\mathcal{L}(w) = \frac{1}{2} a w^2$, where $a > 0$ and $w \in \mathbb{R}$ is the weight that we would like to optimize. Assume that we only have access to a noisy version of the gradient — each time when we make a query for the gradient, we obtain $g(w)$, which is the true gradient $\nabla \mathcal{L}(w)$ with additive Guassian noise:

$$g(w) = \nabla \mathcal{L}(w) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

One way to reduce noise in the gradient is to use minibatch training. Let B be the batch size, and denote the minibatch gradient as $g_B(w)$:

$$g_B(w) = \frac{1}{B} \sum_{i=1}^{B} g_i(w), \quad \text{where } g_i(w) = \nabla \mathcal{L}(w) + \epsilon_i, \quad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2).$$

(a) [1pt] As batch size increases, how do you expect the optimal learning rate to change? Briefly explain in 2-3 sentences.

(Hint: Think about how the minibatch gradient noise change with B.)

2.1.2 Training steps vs. batch size

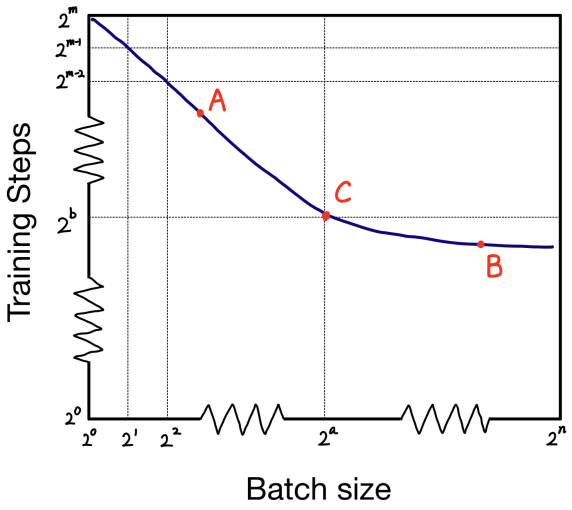

Figure 2: A cartoon illustration of the typical relationship between training steps and the batch size for reaching a certain validation loss (based on [Shallue et al., 2018]). Learning rate and other related hyperparameters are tuned for each point on the curve.

For most of neural network training in the real-world applications, we often observe the relationship of training steps and batch size for reaching a certain validation loss as illustrated in Figure 2.

(a) [1pt] For the three points (A, B, C) on Figure 2, which one has the most efficient batch size (in terms of best resource and training time trade-off)? Assume that you have access to scalable (but not free) compute such that minibatches are parallelized efficiently. Briefly explain in 1-2 sentences.

(b) [1pt] Figure 2 demonstrates that there are often two regimes in neural network training: the noise dominated regime and the curvature dominated regime. In the noise dominated regime, the bottleneck for optimization is that there exists a large amount of gradient noise. In the curvature dominated regime, the bottleneck of optimization is the ill-conditioned loss landscape. For points A and B on Figure 2, which regimes do they belong to, and what would you do to accelerate training? Fill each of the blanks with **one** best suited option.

Point A: Regime: _____. Potential way to accelerate training: _____.

Point B: Regime: _____. Potential way to accelerate training: _____.

Options:

- Regimes: noise dominated / curvature dominated.
- Potential ways to accelerate training: use higher order optimizers / seek parallel compute

2.2 Model size, dataset size and compute

We have seen in the previous section that batch size is an important hyperparameter during training. Besides efficiently minimizing the training loss, we are also interested in the test loss. Recently, researchers have observed an intriguing relationship between the test loss and hyperparameters such as the model size, dataset size and the amount of compute used. We explore this relationship for neural language models in this section. The figures in this question are from [Kaplan et al., 2020].

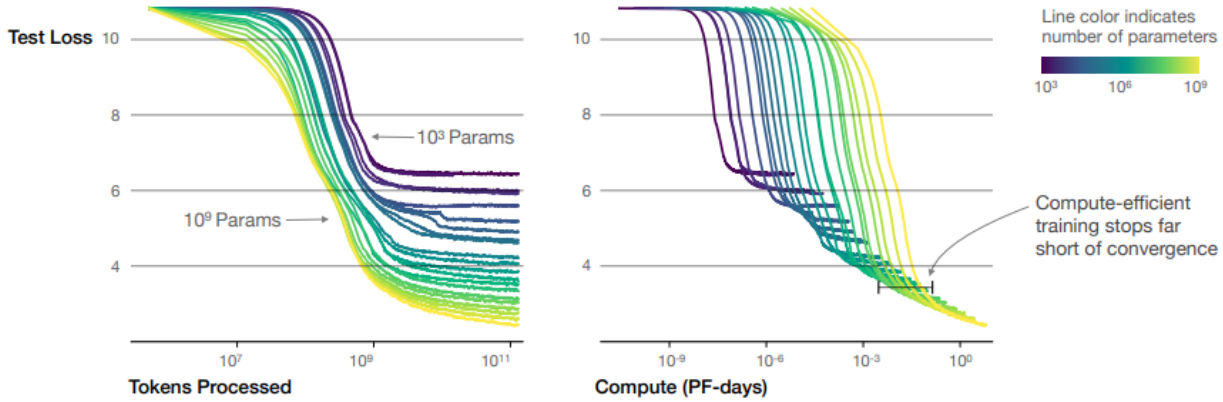

Figure 3: Test loss of language models of different sizes, plotted against the dataset size (tokens processed) and the amount of compute (in petaflop/s-days).

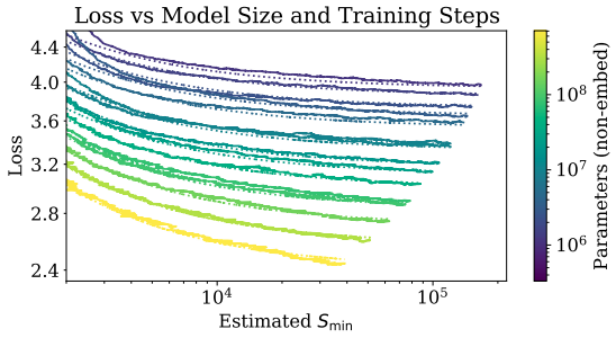

Figure 4: Test loss for different sized models after the initial transient period, plotted against the number of training steps (S_{min}) when using the critical batch sizes (the batch sizes that separate the two regimes in Question 2.1.2).

(a) [1pt] Previously, you have trained a neural language model and obtained somewhat adequate performance. You have now secured more compute resources (in PF-days), and want to improve the model test performance (assume you will train from scratch). Which of the following is the best option? Give a brief explanation (2-3 sentences).

A. Train the same model with the same batch size for more steps.

B. Train the same model with a larger batch size (after tuning learning rate), for the same number of steps.

C. Increase the model size.

3 Dropout as Gaussian noise

In Lecture 6 `https://csc413-uoft.github.io/2021/assets/slides/lec06.pdf`, we derived the expected loss of a linear regression model with input dropout. In this question, we show that dropout can be equivalently viewed as applying Gaussian noise.

3.1 Warm-up: linear regression with input dropout [0pt]

As a warm-up, consider linear regression with input dropout of probability $1 - p$ (the input is retained with probability p).

$$\tilde{y}_m^{(i)} = \frac{1}{p} \sum_j m_j^{(i)} w_j x_j^{(i)}, \qquad \text{where } m_j^{(i)} \overset{\text{i.i.d.}}{\sim} \text{Ber}(p).$$

Derive the bias-variance decomposition as in Lecture 6.

3.2 Multiplicative Gaussian noise [1pt]

Instead of dropout, we apply the multiplicative Gaussian noise as follows:

$$\tilde{y}_\pi^{(i)} = \sum_j (1 + \pi_j^{(i)}) w_j x_j^{(i)}, \qquad \text{where } \pi_j^{(i)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2).$$

Show that with an appropriate choice of σ, this is equivalent to applying input dropout with probability $1 - p$, and find such σ as a function of p.

References

[Goodfellow et al., 2014] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[Kaplan et al., 2020] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[Shallue et al., 2018] Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. (2018). Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*.

[Zhang et al., 2019] Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G. E., Shallue, C. J., and Grosse, R. (2019). Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *arXiv preprint arXiv:1907.04164*.