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.

$f(x;w) = w^T x$

$\Rightarrow \nabla_x f(x;w) = w$

$\Rightarrow x' = x - \varepsilon w$

$\Rightarrow f(x') = w^T x' = w^T(x - \varepsilon w) = w^T x - \varepsilon \|w\|^2$

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.

$g(w; X, t, \lambda) = \frac{1}{2n} \|Xw - t\|^2 + \lambda \|w\|^2$

$\nabla_w g = \frac{1}{2n} \cdot X^T \cdot 2(Xw - t) + \lambda \cdot 2w = 0$

$\Rightarrow w\left(\frac{1}{n} \cdot X^T X + 2\lambda I\right) = \frac{1}{n} X^T t$

$\Rightarrow w^* = \left(X^T X + 2\lambda n \cdot I\right)^{-1} \cdot X^T t$

$\nabla_w^2 g = \frac{1}{n} \cdot X^T X$ is positive semi-definite, so w^* from $\nabla_w g = 0$ is a global minimum.

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.)

Let $f(x'; w^*_{ridge}) = w^{*T}_r x - \varepsilon \|w^*_r\|^2 = 0$

$\Rightarrow \varepsilon^* = \dfrac{w^{*T}_r x}{\|w^*_r\|^2}$

In the 1-dimensional case, $X \in \mathbb{R}^n$, $t \in \mathbb{R}^n$, $w \in \mathbb{R}^1$, $x \in \mathbb{R}^1$

so $w^*_r = (X^T X + 2\lambda n)^{-1} X^T t$.

$$\varepsilon^* = \frac{w^{*}_r x}{(w^*_r)^2} = \frac{x}{w^*_r} = \frac{x}{(X^T X + 2\lambda n I)^{-1} X^T t}$$

$$= \frac{(X^T X + 2\lambda n I) x}{X^T t}.$$

Assume $X^T t > 0$:

Yes, weight decay makes the model more robust to FGM attack, because $|\varepsilon^*|$ needs to be bigger in the case of weight decay. ($\lambda > 0$ v.s. $\lambda = 0$)

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:

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

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

$f(x') = w^T x - \varepsilon \cdot w^T \cdot \operatorname{sign}(w)$

In 1-D case, $\varepsilon = \frac{w \cdot x}{w \cdot \operatorname{sign}(w)} = \frac{\operatorname{sign}(x^T x + 2\lambda n)x}{\operatorname{sign}(x^T t)}$

No, because the sign of $x^T x + 2\lambda n$ remains to be positive when λ increases from 0 to ∞.

As such, ε is fixed, meaning that weight decay model is equally robust under FGSM attack as before.

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 $\mathcal{L}(w) = \frac{1}{2}aw^2$, 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:

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

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.)

$$\mathrm{var}(g_B(w)) = \frac{1}{B^2}\sum_{i=1}^{B}\mathrm{var}(g_i(w)) = \frac{1}{B^2}\cdot B\cdot\sigma^2 = \frac{1}{B}\cdot\sigma^2$$

As batch size increases, gradient noise decreases such that it is inversely proportional to batch size. Since smaller noise give us more confidence that the network is being trained in the correct dimension, we can expect the optimal learning rate to increase.

2.1.2 Training steps vs. batch size

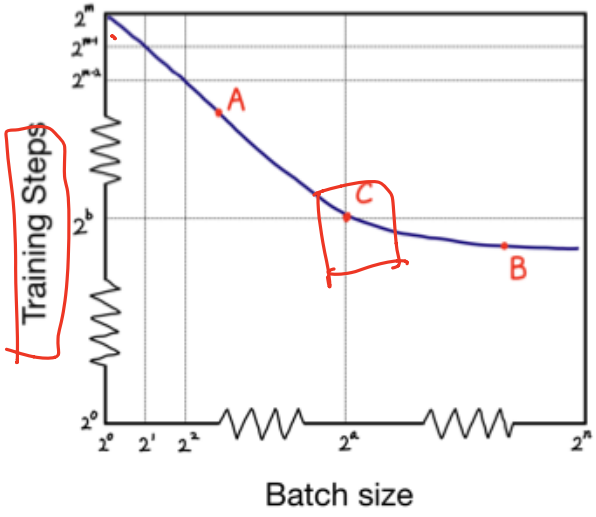

Figure 2: A cartoon illustration of the typical relationship between training steps and the batch size for reaching a certain validation loss (based on [Shallue et al., 2018]). Learning rate and other related hyperparameters are tuned for each point on the curve.

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.~~

Here we assume batches are parallelized so that each update takes constant time no matter the batch size, and total training time can be represented by # of training steps.

Point C is most efficient because at point A, we could better off by trading resource for larger batch size and smaller training steps. Slope becomes more gentle as we go beyond point C. The marginal decrement in loss is overlooked by the huge cost of computation resources, so point B is less efficient than point C.

(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: noise dominated. Potential way to accelerate training: seek parallel compute.

Point B: Regime: curvature dominated. Potential way to accelerate training: use higher order optimizer.

Options:

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

2.2

(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.

As can be seen from the right figure in Figure 3, continued training for a well-adapted model will only decrease the loss by a little. So A is not a good choice

Increasing batch size incurs the same problem as training for more steps — the bottleneck of the model has been reached so there's not much room for improvement. Thus B is not a good choice.

Figure 3-2 suggests that when more computing resources are available, choosing a more complex model will boost the performance to a new level. Thus C is the best choice.

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.

$$E_m(J) = \frac{1}{2N} \sum_{i=1}^{N} E_m\left[\left(\tilde{y}^{(i)} - t^{(i)} \right)^2 \right]$$

$$= \frac{1}{2N} \sum_{i=1}^{N} E_m\left[\left(\tilde{y}^{(i)} - E_m(\tilde{y}^{(i)}) + E_m(\tilde{y}^{(i)}) - t^{(i)} \right)^2 \right]$$

$$= \frac{1}{2N} \sum_{i=1}^{N} E_m\left[\left(\tilde{y}^{(i)} - E_m(\tilde{y}^{(i)}) \right)^2 \right] + \frac{1}{2N} \sum_{i=1}^{N} \left[E_m(\tilde{y}^{(i)}) - t^{(i)} \right]^2 + \frac{1}{2N} \sum_{i=1}^{N} \underbrace{E_m\left(\tilde{y}^{(i)} - E_m(\tilde{y}^{(i)}) \right)}_{0} \cdot \left(E_m(\tilde{y}^{(i)}) - t^{(i)} \right)$$

$$= \frac{1}{2N} \sum_{i=1}^{N} \underbrace{\text{var}\left(\tilde{y}^{(i)} \right)}_{\text{variance}} + \frac{1}{2N} \cdot \sum_{i=1}^{N} \left[\underbrace{E_m(\tilde{y}^{(i)}) - t^{(i)}}_{\text{Bias}} \right]^2$$

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.

⇓ test loss is equivalent

$$E(\hat{y}_{\pi}^{(i)}) = \sum_j E(1 + \pi_j^{(i)}) \cdot w_j x_j^{(i)}$$

$$= \sum_j w_j x_j^{(i)}$$

$$E(\hat{y}_m^{(i)}) = \frac{1}{p} \sum_j \underbrace{E(m_j^{(i)})}_{=p} \cdot w_j x_j^{(i)}$$

$$= \sum_j w_j x_j^{(i)}$$

So $E(\tilde{y}_{\pi}^{(i)}) = E(\tilde{y}_m^{(i)})$ ①

$$\text{var}(\tilde{y}_{\pi}^{(i)}) = \sum_j \text{var}(1 + \pi_j^{(i)}) \cdot [w_j \cdot x_j^{(i)}]^2$$

$$= \sum_j 6^2 \cdot [w_j \cdot x_j^{(i)}]^2$$

$m_j^{(i)} = \begin{cases} 1, \text{ with prob. } p \\ 0, \text{ with prob. } 0 \end{cases} \Rightarrow \text{var}(m_j^{(i)}) = E[(m_j^{(i)2})] - E^2(m_j^{(i)})$

$$= p - p^2$$

$$\text{var}(\tilde{y}_m^{(i)}) = \frac{1}{p^2} \cdot \sum_j \text{var}(m_j^{(i)}) \cdot [w_j \cdot x_j^{(i)}]^2$$

$$= \frac{p-p^2}{p^2} \sum_j \left[w_j \cdot x_j^{(i)} \right]^2$$

$$= \frac{1-p}{p} \sum_j \left[w_j \cdot x_j^{(i)} \right]^2$$

Thus if $\sigma = \sqrt{\frac{1-p}{p}}$, then $\mathrm{var}(\tilde{y}_\pi^{(i)}) = \mathrm{var}(\hat{y}_m^{(i)})$ ②

As such, if $\sigma = \sqrt{\frac{1-p}{p}}$,

$$E(J_m) = \frac{1}{2N} \sum_{i=1}^{N} \mathrm{var}(\hat{y}_m^{(i)}) + \frac{1}{2N} \cdot \sum_{i=1}^{N} \left[E(\hat{y}_m^{(i)}) - t^{(i)} \right]^2$$

$$= \frac{1}{2N} \sum_{i=1}^{N} \mathrm{var}(\tilde{y}_\pi^{(i)}) + \frac{1}{2N} \cdot \sum_{i=1}^{N} \left[E(\tilde{y}_\pi^{(i)}) - t^{(i)} \right]^2 \text{ by ① and ②}$$

$$= E(J_\pi)$$

Thus given $\sigma = \sqrt{\frac{1-p}{p^2}}$, the multiplicative Gaussian noise has same expected performance on test data as Bernouillie drop-out.