EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Discussion 1

1. Gradient Descent Mechanics

Gradient descent is the primary algorithm to search optimal parameters for our models. Typically, we want to solve optimization problems stated as

$$\min_{\theta \in \Theta} \mathcal{L}(f_\theta, \mathcal{D})$$

where $\mathcal{L}$ are differentiable functions. In this example, we look at a simple supervised learning problem where given a dataset $\mathcal{D} = \{(x_i, y_i)\}_i^N$, we want to find the optimal parameters θ that minimizes some loss. We consider different models for learning the mapping from input to output, and examine the behavior of gradient descent for each model.

(a) The simplest parametric model entails learning a single-parameter constant function, where we set $\hat{y}_i = \theta$. We wish to find

$$\hat{\theta}_{const} = \operatorname*{argmin}_{\theta \in \mathbb{R}} \mathcal{L}(f_\theta, \mathcal{D}) = \operatorname*{argmin}_{\theta \in \mathbb{R}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta)^2$$

i. What is the gradient of $\mathcal{L}$ w.r.t. θ ?

ii. What is the optimal value of θ ?

iii. Write the gradient descent update.

iv. *Stochastic Gradient Descent (SGD)* is an alternative optimization algorithm, where instead of using all *N* samples, we use single sample per optimization step to update the model. What is the contribution of each data-point to the full gradient update?

Solution:

- Taking the gradient of $\mathcal{L}$ w.r.t θ we have

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{2}{N} \sum_{i=1}^{N} (\theta - y_i)$$

- From the first-order optimality condition, at θ^* we have $\frac{\partial \mathcal{L}}{\partial \theta} = 0$. Using this, we get

$$\frac{2}{N} \sum_{i=1}^{N} (\theta^* - y_i) = 0$$

$$\implies \theta^* = \frac{1}{N} \sum_{i=1}^{N} y_i$$

- The gradient descent update follows the rule

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$$

where α is the step-size. Plugging in the gradient, we have

$$\theta \leftarrow \theta - \frac{2\alpha}{N}\sum_{i=1}^{N}(\theta - y_i)$$

- Using SGD, we update the model by uniformly sampling a single data-point per iteration, with the following update rule

$$\theta \leftarrow \theta - \frac{2\alpha}{N}(\theta - y_i)$$

The full gradient can be written as

$$\nabla_\theta \mathcal{L} = \frac{2}{N}(\theta - y_1) + \frac{2}{N}(\theta - y_2) + \frac{2}{N}(\theta - y_3) + \cdots + \frac{2}{N}(\theta - y_N)$$

which suggests that each data-point contributes equally to the full gradient.

(b) Instead of constant functions, we now consider a single-parameter **linear** model $\hat{y}(x_i) = \theta x_i$, where we search for θ such that

$$\hat{\theta} = \min_{\theta \in \mathbb{R}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta x_i)^2$$

i. What is the gradient of $\mathcal{L}$ w.r.t. θ ?

ii. What is the optimal value of θ ?

iii. Write the gradient descent update.

iv. Do all points get the same *vote* in the update? Why or why not? (*Hint*: repeat the derivation of the last bullet point in part(a).)

Solution:

- Taking the gradient of $\mathcal{L}$ w.r.t θ we have

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{2}{N} \sum_{i=1}^{N} (\theta x_i - y_i) x_i$$

- From the first-order optimality condition, at θ^* we have $\frac{\partial \mathcal{L}}{\partial \theta} = 0$. Using this, we get

$$\frac{2}{N} \sum_{i=1}^{N} (y_i - \theta x_i) x_i = 0$$

$$\implies \theta^* = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}$$

- The gradient descent update follows the rule

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$$

where α is the step-size. Plugging in the gradient, we have

$$\theta \leftarrow \theta - \frac{2\alpha}{N} \sum_{i=1}^{N} (\theta x_i - y_i) x_i$$

- Using SGD, we update the model by uniformly sampling a single data-point per iteration, with the following update rule

$$\theta \leftarrow \theta - \frac{2\alpha}{N} (\theta x_i - y_i) x_i$$

The full gradient can be written as

$$\nabla_\theta \mathcal{L} = \frac{2}{N}(\theta x_1 - y_1)x_1 + \frac{2}{N}(\theta x_2 - y_2)x_2 + \frac{2}{N}(\theta x_3 - y_3)x_3 + \cdots + \frac{2}{N}(\theta x_N - y_N)x_N$$

The contribution of each data-point to the full gradient depends on the size of the data-point x_i, i.e. data-points closer to the origin contribute less to the full gradient.

2. ReLU SGD Visualization

Work through the notebook to explore how a simple network with ReLU non-linearities adapts to model a function with SGD updates. Training the networks takes 5-10 minutes depending on whether you run locally or on a server, so you should start the training process (ie. run through the *train all layers* cell) then return to the theory part of the discussion while training occurs. The link to run the notebook on Google Colab is https://tinyurl.com/cs182-dis01-code

As you walk through the notebook, pay attention to how the slopes and the elbows of the ReLU functions change during training and how they impact the shape of the final learned function.

Solution:

Answers to notebook questions:

- *What is causing test error to be lower than train error?* **Ans:** We don't add noise in the test samples, which results in lower effective test error compared to training error.
- *Train Output Layer Weights Only*
  (a) *Hidden layer's impact?* **Ans**: The larger the hidden layer is, the lower the test error (no overfitting here if any student ask) and the better learnt function we get.
  (b) *Hand-pick the elbow locations?* **Ans**: At the discontinuities. Having an elbow at each kink is sufficient to capture the non-linearities in this example.
  (c) *How do the final test error and learned function compare to the ridge regression version of training?* **Ans**: Ridge regression version is faster and has a lower test error.
- *Train All Layers*
  (a) *Hidden layer's impact?* **Ans**: Same as the above.
  (b) *Elbow positions?* **Ans**: Gradually converge to the kinks of the function, although the elbows don't align to the exact positions of the kinks.
  (c) *How do the final test error and learned function compare to the ridge regression version of training? Which one is more efficient?* **Ans**: The answer differs depending on the width. For the test error, in the case of *width=10*, we see:

$$AllLayers \approx OuputLayerOnly > Ridge$$

In the case of *width=20*:

$$OuputLayerOnly > Ridge > AllLayers$$

As for the efficiency (ie. training time), *Training All Layers* is the slowest since it requires optimizing more parameters. *Ridge on Output Layer* is the fastest in this example since it can be solved in a closed form solution, whereas in *Training Output Layer* we are using SGD.

Contributors:

- Ashwin Pananjady.
- Josh Sanz.
- Yichao Zhou.

- Anant Sahai.
- Kumar Krishna Agrawal.
- Kevin Li.

