Data C182 Designing, Visualizing & Understanding DNN
Fall 2024 Eric Kim, Naveen Ashish

Discussion 8

This discussion covers select questions from the midterm exam.

1. Initialization

In class, we discussed how, when initializing neural network weights, we tend to choose them randomly from e.g., a Gaussian distribution of a certain variance and mean. Why is this the case? Let's walk through some alternatives.

For the sake of simplicity, assume that your neural network consists only of consecutive affine layers and ReLU non-linearities, and that there is at least one such non-linearity. All hidden layers can have an arbitrary number of elements ≥ 1. You can also assume batch sizes of 1 for training (though your answers should hold for arbitrary batch size). Finally, assume that there is some loss function $L(y)$ that takes in the output of your neural network y, and that loss is used to train your neural network with standard gradient descent (i.e., no momentum, gradient clipping, RMSProp, etc).

For this problem, use this small two affine layer neural network, where x is a two-element column vector:

$$\text{out} = W_2\left[\text{ReLU}\left(W_1 x + b_1\right)\right] + b_2$$

Where $W_1 \in \mathbb{R}^{2\times 2}$, $b_1 \in \mathbb{R}^2$, $W_2 \in \mathbb{R}^{2\times 2}$, $b_2 \in \mathbb{R}^2$.

0.1 Part A.i

Suppose that all weights W_1, W_2 and biases b_1, b_2 for all layers are initialized to zero. The input $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

What is the output of the neural network?

Solution:

Solution: Zero Initialization

(a) **Forward Pass:**

Given all weights and biases are initialized to zero:

$$W_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Compute the activations:

$$h = W_1 x + b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$a = \text{ReLU}(h) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\text{out} = W_2 a + b_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Therefore, the output of the neural network is:

$$\text{out} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

0.2 Part A.ii

Suppose that the final gradient $\frac{dL}{d_{\text{out}}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$. What are the gradients over final Linear layer's weights and biases $\frac{dL}{dW_2}$ and $\frac{dL}{db_2}$?

Solution:

Solution: Gradients for Final Linear Layer

Given:

$$\frac{dL}{d_{\text{out}}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}, \quad a = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

During backpropagation, we compute gradients with respect to weights and biases.
- For the final layer:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \text{out}} \cdot a^T = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \text{out}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

- For the first layer:

$$\frac{\partial L}{\partial W_1} = \left(W_2^T \frac{\partial L}{\partial a} \right) \odot \text{ReLU}'(h) \cdot x^T = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial L}{\partial b_1} = \left(W_2^T \frac{\partial L}{\partial a} \right) \odot \text{ReLU}'(h) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

0.3 Part A.iii

What are the gradients over first Linear layer's weights and biases $\frac{dL}{dW_1}$ and $\frac{dL}{db_1}$?

Solution:

Solution: Gradients for First Linear Layer

Given:

$$a = \text{ReLU}(W_1 x + b_1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad W_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(a) **Gradient with respect to W_1:**

$$\frac{dL}{dW_1} = \left(W_2^T \frac{dL}{da}\right) \odot \text{ReLU}'(h) \cdot x^T$$

Compute $\frac{dL}{da}$:

$$\frac{dL}{da} = W_2^T \frac{dL}{d_{\text{out}}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Compute $\text{ReLU}'(h)$:

$$h = W_1 x + b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \text{ReLU}'(h) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Therefore:

$$\frac{dL}{dW_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

(b) **Gradient with respect to b_1:**

$$\frac{dL}{db_1} = \left(W_2^T \frac{dL}{d_{\text{out}}}\right) \odot \text{ReLU}'(h) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

0.4 Part A.iv

Describe why would this be a problem? (Hint: think about the next forward and backward pass. Your answer should contain little-to-no complicated math and should be at most a few sentences.)

Solution:

Solution: Implications of Zero Initialization

Initializing all weights and biases to zero causes the following issues:

- **No Weight Updates:** Since the gradients with respect to W_1 and W_2 are zero, these weights do not get updated during training.
- **Constant Output Independent of Input:** The network's output remains a constant vector determined solely by b_2, regardless of the input x.
- **Lack of Learning Capability:** The network cannot learn meaningful patterns or dependencies

from the data, rendering it ineffective for tasks like classification or regression.

As a result, the network remains static after initialization, unable to adapt or improve based on the training data.

0.5 B

(a) Now, suppose that for each weight matrix and bias vector, all elements are set to the same constant

$$W_1 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \text{and} \quad b_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The input $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. What is the output of the neural network?

Solution:

Solution: Implications of Zero Initialization

Step 1: Compute the output of the first layer:

$$z_1 = W_1 x + b_1 = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \cdot 1 + 2 \cdot 2 \\ 2 \cdot 1 + 2 \cdot 2 \end{bmatrix} + \begin{bmatrix} -2 \\ -2 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix} + \begin{bmatrix} -2 \\ -2 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

Step 2: Apply the ReLU activation function, which does not change the values since they are positive:

$$a_1 = \text{ReLU}(z_1) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

Step 3: Compute the output of the second layer:

$$z_2 = W_2 a_1 + b_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \cdot 4 + 1 \cdot 4 \\ 1 \cdot 4 + 1 \cdot 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 8 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 9 \end{bmatrix}$$

The output of the neural network is:

$$\text{Output} = \begin{bmatrix} 9 \\ 9 \end{bmatrix}$$

(b) Suppose that the final gradient $\frac{\partial L}{\partial d_{\text{out}}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$. What are the gradients over the final Linear layer's weights and biases $\frac{\partial L}{\partial W_2}$ and $\frac{\partial L}{\partial b_2}$?

Solution:

Solution: Implications of Zero Initialization

Step 1: Compute $\frac{\partial L}{\partial b_2}$, which is just the final gradient:

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial d_{\text{out}}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

Step 2: Compute $\frac{\partial L}{\partial W_2}$:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial d_{\text{out}}} \cdot a_1^T = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \begin{bmatrix} 4 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 0.1 \cdot 4 & 0.1 \cdot 4 \\ 0.2 \cdot 4 & 0.2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.4 \\ 0.8 & 0.8 \end{bmatrix}$$

(c) **(3 Points)** What are the gradients over the first Linear layer's weights and biases $\frac{\partial L}{\partial W_1}$ and $\frac{\partial L}{\partial b_1}$?

Solution:

Solution: Implications of Zero Initialization

Step 1: Compute the gradient with respect to a_1 using W_2 and $\frac{\partial L}{\partial d_{\text{out}}}$:

$$\frac{\partial L}{\partial a_1} = W_2^T \cdot \frac{\partial L}{\partial d_{\text{out}}} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}^T \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$$

Step 2: Compute $\frac{\partial L}{\partial z_1}$:

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1} \odot \text{ReLU}'(z_1) = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$$

Step 3: Compute $\frac{\partial L}{\partial b_1}$:

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$$

Step 4: Compute $\frac{\partial L}{\partial W_1}$:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z_1} \cdot x^T = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0.3 & 0.6 \\ 0.3 & 0.6 \end{bmatrix}$$

(d) Describe why this would be a problem? Hint: think about the next forward and backward pass. How do the results of the previous question impact the expressiveness of the model? Your answer should contain little-to-no complicated math and should be at most a few sentences.

Solution:

Solution: Implications of Zero Initialization

In this case, all intermediate activations for each layer are constant (though the output of each layer can be a different constant). In the last layer, the bias for the last layer can get updated and the weight matrix is:

$$\nabla_W L = \frac{\partial y}{\partial W}\frac{\partial L}{\partial y} = \frac{\partial L}{\partial y}x^T$$

In this case, $\frac{\partial L}{\partial y}$ is an arbitrary vector, but the elements of x are all the same constant. Thus, the resulting gradient is a matrix where each row is a (possibly different) constant. After W is updated using gradient descent, its rows are likewise each a possibly different constant.

What about the gradient with respect to x, which gets passed to the preceding layer?

$$\nabla_x L = \frac{\partial y}{\partial x}\frac{\partial L}{\partial y} = W^T\frac{\partial L}{\partial y}$$

In this case, W either is all constant or has constant rows. However, in either case, $\nabla_x L$ evaluates to all constants.

For the second-to-last layer, the downstream derivative it receives $\frac{\partial L}{\partial \text{out}}$ is thus always constant. Since it also received a constant input, that means its weight matrix, bias vector, and input gradients are thus also going to be filled with constants (albeit in maybe a different constant from before). Thus, the weight and bias terms themselves will stay constant after gradient descent.

The only case where this isn't true is the very first layer, since it receives the actual input of the neural net x_{input} (which might NOT be constant). However, since it received constant $\frac{\partial L}{\partial \text{out}}$, the gradient of its weight matrix will be:

$$\frac{\partial L}{\partial \text{out}}x_{\text{input}}^T$$

which is a matrix with each *column* having a different value. Thus, W for that layer will likewise have constant columns. For the next forward pass, its outputs are thus going to be constant. Because of this, we can repeat this same argument for all subsequent gradient steps as well.

The final result is that each hidden layer's resulting activations will always be the same – that is, their neurons all work the same. Thus, the overall neural network behaves just like one wherein each hidden layer has only one neuron, thereby lowering its expressivity. It will still try to learn the task, but likely poorly.

2. Multiple Choice

0.6 Q1

A model for classifying different objects is getting a high training set error. Which of the following is the most likely way to improve the classifier?

- ◯ **A**: Use more training data.
- ◯ **B**: Increase the regularization being used.
- ◯ **C**: Use a bigger network.
- ◯ **D**: Use a smaller network.

Solution:

Solution: Addressing High Training Error

The model is experiencing high training set error, which indicates that it is underfitting the data. Underfitting occurs when the model is too simple to capture the underlying patterns in the data. To address underfitting, we need to increase the model's capacity to learn more complex representations.

(a) **A: Use more training data.**
Incorrect. The primary issue here is that the model itself lacks sufficient capacity to learn from the data. While more data can help, the most direct approach to reduce training error is to increase the model's complexity.

(b) **B: Increase the regularization being used.**
Incorrect. Increasing regularization (such as L1 or L2 regularization) typically helps prevent overfitting by penalizing large weights. In this case, the model is underfitting, so increasing regularization would likely exacerbate the problem by further restricting the model's capacity.

(c) **C: Use a bigger network.**
Correct. A bigger network, with more layers or more neurons per layer, increases the model's capacity to learn complex patterns in the data. This directly addresses underfitting by allowing the model to better fit the training data, thereby reducing training error.

(d) **D: Use a smaller network.**
Incorrect. Using a smaller network would decrease the model's capacity, which is counterproductive in a situation where the model is already underfitting. This would likely increase the training error further.

0.7 Q2

How many model parameters are in a Convolution2D layer that uses a 4x4 filter with 5 output channels and a bias, and takes as input a three-channel color RGB image with height=32 pixels, width=32 pixels?

- ◯ **A**: 16
- ◯ **B**: 245

◯ **C**: 80
◯ **D**: 240
◯ **E**: 21
◯ **F**: 85

Solution:

Solution: Calculating Convolutional Layer Parameters

To calculate the number of parameters in a Convolution2D layer, consider both the weights of the filters and the biases.
Given:

- Filter size: 4×4
- Number of output channels: 5
- Number of input channels: 3 (RGB)
- Bias: 1 per output channel

Calculation:

$$\text{Number of weights} = \text{filter height} \times \text{filter width} \times \text{input channels} \times \text{output channels}$$

$$= 4 \times 4 \times 3 \times 5 = 240$$

$$\text{Number of biases} = \text{output channels} = 5$$

$$\text{Total parameters} = 240 + 5 = 245$$

0.8 Q3

Which of the following can lead to vanishing gradients?

◯ **A**: Sigmoid activations.
◯ **B**: Very deep neural network with skip connections.
◯ **C**: Batch normalization layers.
◯ **D**: Leaky ReLU activations.

Solution:

Solution: Factors Leading to Vanishing Gradients

Vanishing gradients occur when gradients become too small during backpropagation, hindering effective learning, especially in deep networks.
Explanation of Answer Choices:

(a) **A: Sigmoid activations.**
Correct. Sigmoid activation functions squash input values into the range (0, 1). For inputs

with large magnitudes, the sigmoid function saturates, leading to very small gradients (derivatives close to zero). This causes the gradients to diminish as they propagate back through the network, resulting in vanishing gradients.

(b) **B: Very deep neural network with skip connections.**
Incorrect. Skip connections (as used in architectures like ResNet) are designed to alleviate the vanishing gradient problem by providing alternative pathways for gradients to flow through the network. This facilitates the training of very deep networks.

(c) **C: Batch normalization layers.**
Incorrect. Batch normalization helps stabilize and accelerate training by normalizing layer inputs, which can mitigate issues like vanishing and exploding gradients. It does not contribute to vanishing gradients; instead, it often helps prevent them.

(d) **D: Leaky ReLU activations.**
Incorrect. Leaky ReLU activations address the "dying ReLU" problem by allowing a small, non-zero gradient when the unit is not active. They help in maintaining gradient flow and do not lead to vanishing gradients.

0.9 Q4

What is the primary motivation for adding masks in “masked self-attention” in the Transformer decoder?

- ◯ **A**: To better-condition the intermediate activation values to avoid the vanishing/exploding gradient problem.
- ◯ **B**: To avoid the decoder from “cheating” and using information from future token positions.
- ◯ **C**: To avoid the decoder from “cheating” and using information from other batch samples.
- ◯ **D**: To improve representation power by adding more model parameters.

Solution:

Solution: Purpose of Masks in Masked Self-Attention

Masks in masked self-attention are crucial for maintaining the autoregressive property of the Transformer decoder, ensuring that predictions for a given position depend only on the known outputs at positions before it.
Explanation of Answer Choices:

(a) **A: To better-condition the intermediate activation values to avoid the vanishing/exploding gradient problem.**
Incorrect. While conditioning intermediate activations is important, masks are not primarily used for preventing vanishing or exploding gradients. Instead, techniques like proper initialization and normalization address gradient issues.

(b) **B: To avoid the decoder from “cheating” and using information from future token positions.**
Correct. In masked self-attention, masks are applied to prevent the model from accessing future tokens during training. This ensures that the prediction for a particular position can only depend on the tokens before it, maintaining the causality required for tasks like language

modeling and translation.

(c) **C: To avoid the decoder from "cheating" and using information from other batch samples.**
Incorrect. Masks in self-attention are not related to interactions between different batch samples. Batch interactions are generally managed by batching mechanisms, not by attention masks.

(d) **D: To improve representation power by adding more model parameters.**
Incorrect. Masks do not add additional model parameters. Their primary role is to control the flow of information during attention computation, not to enhance representation capacity through additional parameters.

0.10 Q5

In the Transformer self-attention block, when computing the attention weights $A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$, what is the primary motivation for dividing by $\sqrt{d_k}$, where d_k is the embedding dimensionality?

- ◯ **A**: To better-condition the intermediate activation values to avoid the vanishing/exploding gradient problem.
- ◯ **B**: To avoid dividing by 0.
- ◯ **C**: To add additional regularization to avoid overfitting.
- ◯ **D**: To improve representation power by adding more model parameters.

Solution:

Solution: Scaling in Attention Mechanism

The scaling factor $\sqrt{d_k}$ is introduced to counteract the effect of the dot product's magnitude as the dimensionality increases.
Explanation of Answer Choices:

(a) **A: To better-condition the intermediate activation values to avoid the vanishing/exploding gradient problem.**
Correct. When computing the dot product QK^T, the variance of the result increases with the dimensionality d_k. Without scaling, for large d_k, the softmax function can push values into regions with very small gradients, exacerbating the vanishing gradient problem. Dividing by $\sqrt{d_k}$ normalizes the dot product, ensuring that the softmax input remains in a range that maintains meaningful gradient magnitudes.

(b) **B: To avoid dividing by 0.**
Incorrect. The dimensionality d_k is a positive integer, so $\sqrt{d_k}$ is never zero. This is not the reason for scaling.

(c) **C: To add additional regularization to avoid overfitting.**
Incorrect. The scaling factor does not serve as a regularizer. Regularization techniques involve adding penalties to the loss function or modifying the network architecture to prevent

overfitting.

(d) **D: To improve representation power by adding more model parameters.**
Incorrect. Scaling by $\sqrt{d_k}$ does not introduce new parameters or enhance representation power. It is purely a normalization technique to stabilize training.

