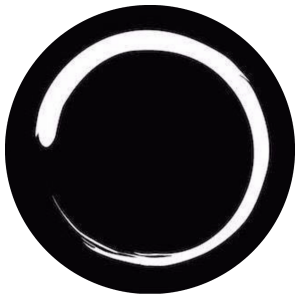
inzva DLSG

Homework 1



Open Questions

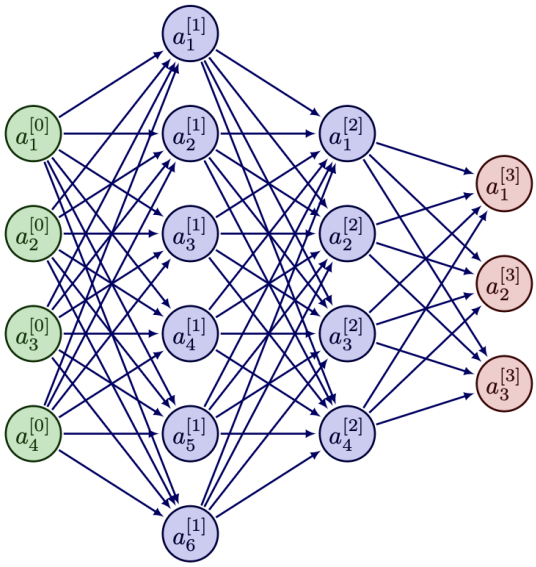
1. What is **deep learning**, and how does it differ from **traditional machine learning**?

2. What do we mean by **forward propagation**?

3. What is **backpropagation**, and why is it essential for neural networks?

4. Why do we use **activation functions** for the layers? What happens if we don't use activation functions?

5. Write **forward propagation equations** and corresponding dimensions of the **matrices** and **vectors** for the neural network below. Assume your batch size is 1. (Dimensions for weight matrices and input vectors for each layer, as we did in the session.)



6. When does our model have **underfitting** and **overfitting**? How do we determine these situations? Evaluate your answer.

7. What is the **learning rate** in the context of training neural networks? What happens if the learning rate is too high or too low?

8. What is **dropout**, and how does it help in training neural networks?

9. What is the **main difference** between model's weights and **hyperparameters**?

10. Explain the difference between **batch gradient descent**, **stochastic gradient descent**, and **mini-batch gradient descent**.

11. Explain the difference between **Adam** and **RMSprop** optimizers.

Coding Assignment

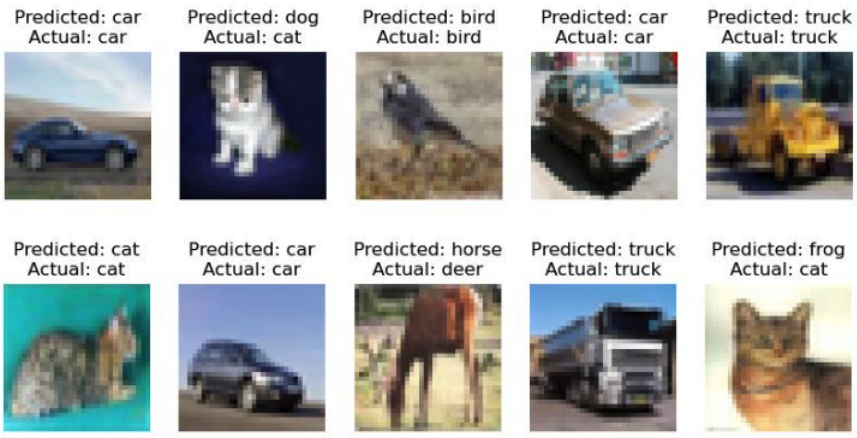
You will be working with a partially completed deep learning project that focuses on training a neural network on the CIFAR-10 dataset using PyTorch. The provided code includes structured comments and placeholder variables that you are required to fill out based on the instructions within the code. Your task is to define critical parameters such as the number of epochs, batch size, learning rate, the neural network architecture and so on.

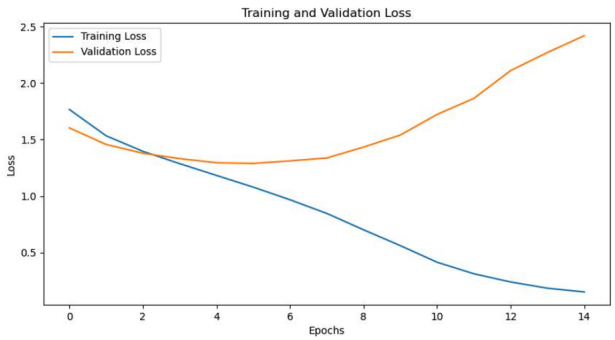
The *main.py* file serves as the entry point for the project and walks you through the steps of data loading, model initialization, training, and testing. The comments provided in the code will guide you in filling out necessary parts where values or function implementations are missing. There are additional utility files that handle the loading and processing of data as well as visualization tools to help you monitor model performance during training.

It is important to note that the goal of this assignment is not to achieve high accuracy, but rather to successfully train, test, and visualize your results using the provided codebase. Focus on understanding the workflow, completing the missing parts, and ensuring that the model can run without errors. Pay close attention to the provided comments, and ensure that your code is well-structured and functional.

Make sure to follow the instructions in the code carefully and utilize the PyTorch documentation if needed. Once the necessary parts are implemented, you should be able to train a neural network, visualize its performance, and evaluate its accuracy on the test set.

Visualization Examples





Bonus Questions (Optional)

1. Assume you have a linear regression model $\hat{y} = \mathbf{f_w}(x) = \mathbf{w}_1 x + \mathbf{w}_0$ for a dataset of pairs $(x_i, y_i)_{1 \leq i \leq N}$. You want to estimate your weights so that your model fits the data well. You have residuals (errors of your model for every pair) $e_i = (y_i - \hat{y}_i)$. Then, using the residual sum of squares (RSS) function $\mathbf{RSS} = \sum_{i=1}^{N} e_i^2$, show that your estimated weights can be found as

$$\mathbf{w}_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$
$$\mathbf{w}_0 = \bar{y} - \mathbf{w}_1 \bar{x}$$

   where $\bar{x} = \frac{1}{n}\sum_{i=1}^{N} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{N} y_i$ are the means of your samples.

2. In binary classification, where the output can either be 0 or 1, the binary cross-entropy loss function is used. The BCE loss measures the difference between the true labels and the predicted probabilities. Given a single data point, the binary cross-entropy loss can be defined as:

$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

   where:

   - y is the true label (0 or 1).
   - $\hat{y}$ is the predicted probability for the positive class (1).

   In forward propagation, we calculate the predicted probability using a logistic function (also known as the sigmoid function), followed by the binary cross-entropy loss.

   For a model output z (logit):

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

   This function converts the raw score (logit) z into a probability $\hat{y}$ between 0 and 1. Then assume you have the following neural network with 2 hidden layer and sigmoid function for every layer's activation function. Derive the **forward** and **backward propagation** equations for the binary cross-entropy loss function.