EEP 568 Homework 1 (100%) due: 10/13 11:59 pm

Submission Instructions

You will need to submit the following materials:

1. **Questions and Answers** in PDF format. Remember to put your name in your PDF file.
2. A folder containing all ipynb files with your results for the the problems. Please remember to keep all results and logs in the submitted ipynb files to get full credit.

All submitted files should be put into one single zip file named as HW#_xxx.zip, e.g. HW1_George_Clooney.zip**,** including all ipynb files with answers (both results and discussions), and the **Questions and Answers** PDF file.

Problem 1: Classification on Digital Images Using Traditional Machine Learning (50%)

We will go through a definition called Principal Component Analysis (PCA) for dimensionality reduction for high dimensional data. We do dimensionality reduction to convert the high d-dimensional dataset into k-dimensional data where $k < d$. Data variance on one axis may be very large but relatively smaller on another axis. Higher variance usually means greater information in this direction. Therefore, we can skip the dimensions having less variance because having less information.

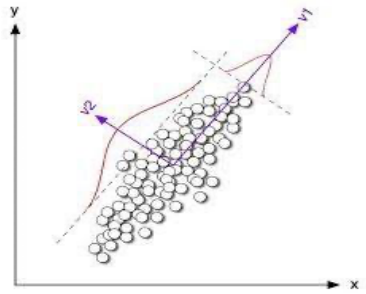


Fig. 1: Variances of 2D data.

Fig. 1 is an example of a set of 2D data. The direction of v1 is maximum while v2 is minimum, so that v1 has more information about the dataset. Thus, the 2D data with (x,y) variables can be converted to 1D variables in the direction of v1.

Projections (Orthogonal)

Input: $x_1, x_2, \ldots, x_n \in R^d$, target dim k.

Output: a k-dim subspace by orthonormal basis $q_1, q, \ldots, q_k \in R^d$.

Orthogonal projection:

$$\underbrace{\left(\sum_{i=1}^{k} \boldsymbol{q}_i \boldsymbol{q}_i^\top\right)}_{\Pi} \boldsymbol{x} \;=\; \sum_{i=1}^{k} \langle \boldsymbol{q}_i, \boldsymbol{x} \rangle \boldsymbol{q}_i \;\in\; \mathbb{R}^d .$$

It can be also represented in terms of coefficients w.r.t. the orthonormal basis $q_1, q, \ldots, q_k \in R^d$:

$$\boldsymbol{\phi}(\boldsymbol{x}) \;:=\; \begin{bmatrix} \langle \boldsymbol{q}_1, \boldsymbol{x} \rangle \\ \langle \boldsymbol{q}_2, \boldsymbol{x} \rangle \\ \vdots \\ \langle \boldsymbol{q}_k, \boldsymbol{x} \rangle \end{bmatrix} \;\in\; \mathbb{R}^k .$$

Minimize residual squared error

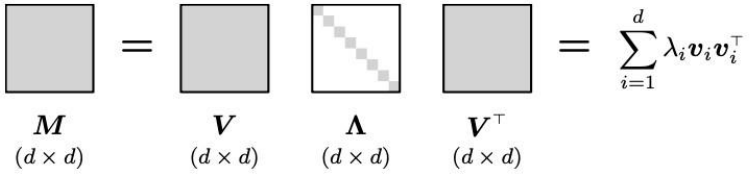
$$\underset{\substack{\boldsymbol{Q} \in \mathbb{R}^{d \times k}: \\ \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}}}{\arg\min} \; \frac{1}{n} \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \boldsymbol{Q}\boldsymbol{Q}^\top \boldsymbol{x}_i \right\|_2^2 \quad \equiv \quad \underset{\substack{\boldsymbol{Q} \in \mathbb{R}^{d \times k}: \\ \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}}}{\arg\max} \; \sum_{i=1}^{k} \boldsymbol{q}_i^\top \left(\frac{1}{n} \boldsymbol{A}^\top \boldsymbol{A} \right) \boldsymbol{q}_i .$$

(where $\boldsymbol{x}_i^\top$ is i-th row of $\boldsymbol{A} \in \mathbb{R}^{n \times d}$).

Solution: k eigenvectors of $A^T A$ corresponding to k largest eigenvalues.

Eigen Decompositions

Every symmetric matrix $\boldsymbol{M} \in \mathbb{R}^{d \times d}$ guaranteed to have eigendecomposition with real eigenvalues:



real **eigenvalues**: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \quad (\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d))$;
corresponding orthonormal **eigenvectors**: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d \quad (\boldsymbol{V} = [\boldsymbol{v}_1 | \boldsymbol{v}_2 | \cdots | \boldsymbol{v}_d])$.

Please follow the following steps to implement PCA using numpy

Step 1: Standardize the dataset.
Step 2: Calculate the covariance matrix for the features in the dataset.
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.
Step 4: Sort eigenvalues and their corresponding eigenvectors.
Step 5: Pick k eigenvalues and form a matrix of eigenvectors.
Step 6: Transform the original matrix.

Handwritten classification is a basic task for the early stage of image processing field. In this problem, we will work with MNIST dataset, which contains 60,000 training and 10,000 testing images of handwritten numbers 0-9.

Fig. 1: Some sample images in MNIST dataset.

(a) Prepare MNIST dataset

Download the dataset from Hugging Face and follow the instruction in HW1.ipynb to load and visualize the dataset.

(b) PCA on MNIST

Implement PCA function from scratch.

Apply PCA on MNIST dataset to reduce the dimension of the digit images.

(c) Logistic Regression

Follow the steps on the notebook HW1.ipynb to build a logistic regression model for MNIST dataset. Here you should use the representative data (after PCA) for training and inference. For more details, please refer to https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

(d) Support Vector Machine (SVM) Classifier

Build a SVM Classifier for MNIST dataset. Here you should continue to use the representative data (after PCA) for training and inference. For more details, please refer to https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

1) Implement a SVM classifier using the scikit-learn package: sklearn.svm.SVC with L2 regularization parameters C = 1.0, kernel type ‘linear’.

2) Evaluate the classification accuracies on the validation set.

3) Try using a different kernel type, change the kernel from ‘linear’ to ‘rbf’ (radial

basis function) and evaluate the classification accuracy on the validation set. Which one ('linear' or 'rbf') can give you higher accuracy?

4) Fix the kernel type to be 'rbf' and try different sets of regularization parameters $C \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$, and report all the classification accuracies on the validation set. What's the meaning of changing the C here? Which C in your case can give your the best accuracy? Compared with the logistic regression, which method performs better? Please briefly explain it.

5) Run your classifier on the testing set with the model which achieves the best performance on the validation dataset and visualize some of the images with their predicted labels.

Problem 2 Autograd – implement your backpropagation

Modern deep learning frameworks, such as PyTorch's Autograd, incorporate automatic differentiation systems essential for training neural networks. Leveraging dynamic computation graphs, these systems record tensor operations executed in the forward pass. During the backward pass, usually initiated on a loss tensor, PyTorch meticulously navigates this graph in reverse to calculate and retain gradients. Autograd not only grant adaptability in crafting models but also optimize memory usage. In this problem, we are going to implement a simplified engine for auto differentiation or Autograd. We will walk through the process of constructing computational graph, implementing backpropagation, defining nuerons and layers, building a multi-layer perceptron and eventually training our MLP on simple data.

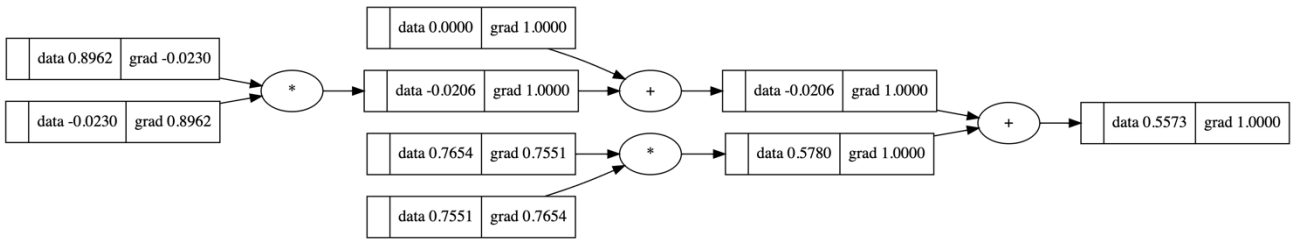


Fig. 2 Visualization of intermediate gradients from the backpropagation we will implement.

Problem 3: Image Classification on CIFAR-10 Using Multilayer Perceptron (MLP)

Image classification using a Multilayer Perceptron (MLP) involves training a neural network to categorize images into predefined classes. MLPs consist of fully connected layers where each neuron in one layer is connected to every neuron in the next layer. In image classification, the image is first flattened into a one-dimensional vector before being fed into the network.

In this problem, we will solve a simple image classification problem on the CIFAR-10 dataset. It has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size. There are 50000 images in the training set and 10000 images in the testing set.

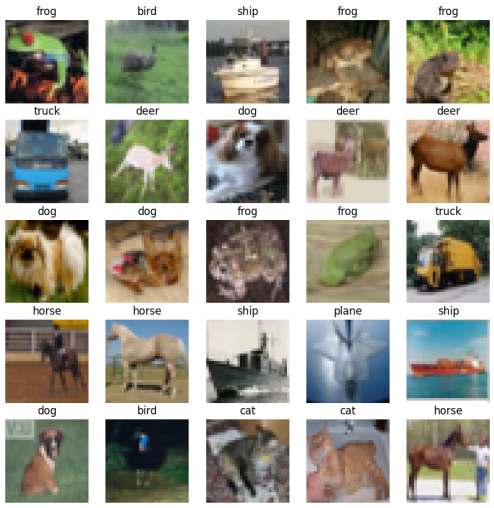


Fig. 3 CIFAR-10 Examples

We are going to implement MLP on Pytorch. If you are not familiar with Pytorch, you can find the provided Pytorch tutorials from the course Canvas and easily run them on Colab.

Please find more details in **HW1.ipynb**

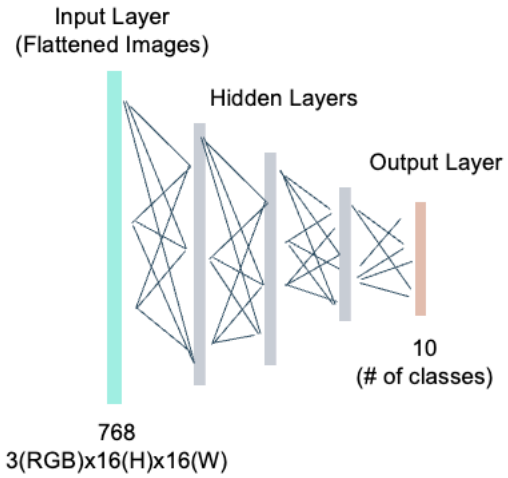


Fig. 4 An example MLP architecture with three hidden layers