CSE402 Assignment #1: Matrix Calculus and Naive Bayes Classification

Due Date: Apri 12nd, Saturday, 11:59 PM KST.
In this assignment, you will review the Jacobian formulation for matrix derivatives, explore the mathematics behind Naive Bayes classification using word embeddings, and build a Naive Bayes classifier using PyTorch. Please note that this assignment involves self-directed study, including reviewing PyTorch tutorials and materials.

1. Part 1: You will study the Jacobian formulation for matrix derivatives.
2. Part 2: You will derive the mathematical foundations of the Naive Bayes classifier based on word embeddings.
3. Part 3: You will learn PyTorch, implement the Naive Bayes classifier using word embeddings, train the model, evaluate its performance, and explore possible extensions.

If you are using LaTeX, you can use `\ifans{}` to type your solutions.

1. Matrix Derivatives in Jacobian Formulation (22 points)

Suppose that $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is a column vector, defined as follows:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{1}$$

We now define a function of vector $\mathbf{x}$, i.e., $\mathbf{y} = \mathbf{f}(\mathbf{x}) \in \mathbb{R}^{m\times 1}$. The Jacobian matrix, $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m\times n}$ – that is, the partial derivative of $\mathbf{y}$ with respect to $\mathbf{x}$ – is given as follows:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \tag{2}$$

In addition, when $\mathbf{X} \in \mathbb{R}^{n\times m}$ is a parameter matrix and $y = f(\mathbf{X})$ is a scalar function of $\mathbf{X}$, the Jacobian matrix $\frac{\partial y}{\partial \mathbf{X}} \in \mathbb{R}^{m\times n}$ is defined as follows:

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{n1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1m}} & \frac{\partial y}{\partial x_{2m}} & \cdots & \frac{\partial y}{\partial x_{nm}} \end{pmatrix} \tag{3}$$

(a) (3 points) Suppose that $\mathbf{z} \in \mathbb{R}^k$, $\mathbf{y} \in \mathbb{R}^n$, and $\mathbf{x} \in \mathbb{R}^m$ are given as follows:

$$\begin{aligned} \mathbf{z} &= f(\mathbf{y}) \\ \mathbf{y} &= g(\mathbf{x}) \end{aligned} \tag{4}$$

Prove that the following chain rule holds:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \tag{5}$$

Note: Please specify the shapes of all matrices when they appear in the derivation.

(b) (3 points) Prove that if $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{y} \in \mathbb{R}^{m\times 1}$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$, and $\mathbf{A} \in \mathbb{R}^{m\times n}$, then the Jacobian matrix of $\mathbf{y}$ with respect to $\mathbf{x}$ is given by:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{A} \tag{6}$$

(c) (3 points) Prove that if $\mathbf{y} = f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{n\times 1}$ and f is an elementwise function, then the Jacobian matrix of $\mathbf{y}$ with respect to $\mathbf{x}$ is given by:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \text{diag}\left(f'(\mathbf{x})\right) \tag{7}$$

(d) (3 points) If $\mathbf{z} = f(\mathbf{U}g(\mathbf{W}\mathbf{x}))$, where $\mathbf{z} \in \mathbb{R}^{k\times 1}$, $\mathbf{U} \in \mathbb{R}^{k\times n}$, $\mathbf{W} \in \mathbb{R}^{n\times m}$, $\mathbf{x} \in \mathbb{R}^{m\times 1}$, and f and g are elementwise functions, what is the Jacobian matrix $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$?

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \tag{8}$$

(e) (5 points) Prove that if $J = h(\mathbf{z})$ and $\mathbf{z} = \mathbf{W}\mathbf{x}$, where $\mathbf{z} \in \mathbb{R}^{m\times 1}$ and $\mathbf{x} \in \mathbb{R}^{n\times 1}$, then the Jacobian matrix of J with respect to $\mathbf{W}$ is given by:

$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{x}\frac{\partial J}{\partial \mathbf{z}} \tag{9}$$

(f) (5 points) Prove that if the cross-entropy loss CE is expressed as:

$$\begin{aligned} J &= CE(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^T \log \hat{\mathbf{y}} \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{z}) \end{aligned} \tag{10}$$

where $\mathbf{y} \in \mathbb{R}^{k\times 1}$ is the one-hot label vector and $\mathbf{z} \in \mathbb{R}^{k\times 1}$ are the logits, then the Jacobian matrix of the cross-entropy loss J with respect to the logits $\mathbf{z}$ is given by:

$$\frac{\partial J}{\partial \mathbf{z}} = (\hat{\mathbf{y}} - \mathbf{y})^T \tag{11}$$

2. Naive Bayesian Text Classifier (24 points)

In text classification, a text $\mathbf{x}$ is treated as a sequence of words, i.e., $\mathbf{x} = (w_1, \cdots, w_n)$.

The Naïve Bayes model for text classification assumes that words are generated independently, and thus uses the posterior probability as follows:

$$P(y = k \mid \mathbf{x}) = \frac{\prod_{i=1}^{n} P(w_i \mid y = k) P(y = k)}{P(\mathbf{x})} \tag{12}$$

(a) (3 points) Consider the frequency-based estimator for the word generative probability $P(w_i \mid y = k)$, given by:

$$P(w_i \mid y = k) \approx \frac{c(w_i, y = k)}{\sum_j c(w_j, y = k)} \tag{13}$$

where $c(w_i, y = k)$ is the count (or **frequency**) of word w_i appearing in texts belonging to class k, $(y = k)$ in the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, and $\sum_j c(w_j, y = k)$ is the **total count** of all words appearing in texts belonging to class k, $(y = k)$.

What is the problem with the frequency-based estimator?

(b) (21 points) We model the word generative probability $P(w_i \mid y = k)$ using word embeddings as follows:

$$P(w_i \mid y = k) = \text{softmax}_i \left(\mathbf{v}_i^\top \mathbf{c}_k\right) = \frac{\exp\left(\mathbf{v}_i^\top \mathbf{c}_k\right)}{\sum_{j \in \mathcal{V}} \exp\left(\mathbf{v}_j^\top \mathbf{c}_k\right)} \tag{14}$$

Consider the negative log-likelihood as the objective (loss) function for the Naïve Bayes model, defined as follows:

$$\begin{aligned}
Loss &= -\sum_{(\mathbf{x},k)\in\mathcal{D}} \log P\left(\mathbf{x}, y = k\right) \\
&= -\sum_{(\mathbf{x},k)\in\mathcal{D}} \log P\left(\mathbf{x} | y = k\right) P(y = k) \\
&= -\sum_{(\mathbf{x},k)\in\mathcal{D}} \log \left(\prod_{w_i \in \mathbf{x}} P(w_i | y = k) \right) P(y = k) \\
&= \underbrace{-\sum_{(\mathbf{x},k)\in\mathcal{D}} \sum_{w_i \in \mathbf{x}} \log P(w_i | y = k)}_{\text{conditional likelihood part}} \underbrace{-\sum_{k=1}^{K} N_k \log P(y = k)}_{\text{prior part}}
\end{aligned} \tag{15}$$

i. (8 points) Suppose the word embedding matrix is given by:

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_{|\mathcal{V}|}^\top \end{bmatrix} \tag{16}$$

Derive the Jacobian matrix of the loss *Loss* in Eq. (15) with respect to $\mathbf{V}$, expressed as:

$$\frac{\partial Loss}{\partial \mathbf{V}} = \tag{17}$$

ii. (5 points) Derive the Jacobian matrix of the loss *Loss* in Eq. (15) with respect to $\mathbf{c}_k$, expressed as:

$$\frac{\partial Loss}{\partial \mathbf{c}_k} = \tag{18}$$

iii. (8 points) Note that the denominator in Eq. (14) involves a summation of exponentiated logits over all words. If you choose to use a negative sampling method, as in word2vec (i.e., skip-gram with negative sampling), how would you revise the loss function? Here, it is not necessary to preserve the precise form of the joint likelihood.

3. Naive Bayes Text Classification using PyTorch (56 points)

In this section, you'll be implementing a naive Bayes text classification by minimizing the loss function (i.e., the negative log likelihood).

Before you begin, please install `pytorch 2.6.0` under `python 3.12` from `https://pytorch.org/` for the assignment.

The official PyTorch website is a great resource that includes tutorials for understanding PyTorch's Tensor library and neural networks.

Please read Pytorch tutorial from `https://pytorch.org/tutorials/`, including the following materials:

- Learn the Basics
- Introduction to PyTorch - YouTube Series
- Deep Learning with PyTorch: A 60 Minute Blitz
- Learning PyTorch with Examples
- What is torch.nn really?
- Many other resources about Pytorch
- ...

(a) (23 points) We provide a jupyter notebook file (`nb_classifier.ipynb`) that implements a version of naive Bayes classifier based on word embedding in Section 2. You are required to complete the missing codes from `DenseMultinomialNaiveBayes` class, and the first line from `nb_finetuning` function. You don't need to modify other parts of the notebook files, except for settings and configurations based on your environment.

Once you correctly add the notebook file, you should see the following results.

Listing 1: Outputs after running nb_classifier.ipynb

```
    ...
Model Saved best loss: 1854.4124
Epoch 3/10:  30%|============================| 2346/7820 [01:07<01:16, 71.58it/s,
    ↪val_loss: 1699.5958]Validation Loss: 1699.5958
Epoch 4/10:  30%|============================| 2346/7820 [01:07<01:16, 71.58it/s,
    ↪val_loss: 1699.5958]Model Saved best loss: 1699.5958
Epoch 4/10:  40%|=======================================| 3128/7820 [01:28<00:58,
    ↪80.47it/s, val_loss: 1645.9739]Validation Loss: 1645.9739
Epoch 5/10:  40%|=======================================| 3128/7820 [01:29<00:58,
    ↪80.47it/s, val_loss: 1645.9739]Model Saved best loss: 1645.9739
Epoch 5/10:  50%|================================================| 3910/7820
    ↪[01:50<00:46, 84.11it/s, val_loss: 1616.1767]Validation Loss: 1616.1767
Epoch 6/10:  50%|================================================| 3910/7820
    ↪[01:50<00:46, 84.11it/s, val_loss: 1616.1767]Model Saved best loss: 1616.1767
Epoch 6/10:  60%|=========================================================| 4692/7820
    ↪[02:11<00:40, 78.01it/s, val_loss: 1596.4801]Validation Loss: 1596.4801
Epoch 7/10:  60%|=========================================================| 4692/7820
    ↪[02:12<00:40, 78.01it/s, val_loss: 1596.4801]Model Saved best loss: 1596.4801
Epoch 7/10:  70%|==================================================================|
    ↪5474/7820 [02:33<00:27, 84.28it/s, val_loss: 1582.0797]Validation Loss: 1582.0797
Epoch 8/10:  70%|==================================================================|
    ↪5474/7820 [02:34<00:27, 84.28it/s, val_loss: 1582.0797]Model Saved best loss:
    ↪1582.0797
Epoch 8/10:  80%|============================================================================
    ↪=====| 6256/7820 [02:55<00:18, 83.16it/s, val_loss: 1572.0885]Validation Loss:
    ↪1572.0885
```

```
Epoch 9/10:  80%|=================================================================================
    ↪=====| 6256/7820 [02:56<00:18, 83.16it/s, val_loss: 1572.0885]Model Saved best loss
    ↪: 1572.0885
Epoch 9/10:  90%|=================================================================================
    ↪=================| 7038/7820 [03:18<00:10, 73.80it/s, val_loss: 1565.1391]
    ↪Validation Loss: 1565.1391
Epoch 10/10:  90%|=================================================================================
    ↪==================| 7038/7820 [03:18<00:10, 73.80it/s, val_loss: 1565.1391]Model
    ↪Saved best loss: 1565.1391
Epoch 10/10: 100%|=================================================================================
    ↪==============================| 7820/7820 [03:40<00:00, 82.25it/s, val_loss:
    ↪1559.3582]Validation Loss: 1559.3582
Epoch 10/10: 100%|=================================================================================
    ↪==============================| 7820/7820 [03:40<00:00, 35.43it/s, val_loss:
    ↪1559.3582]
Model Saved best loss: 1559.3582
100%|=============================================================================================
    ↪==========================| 782/782 [00:07<00:00, 102.90it/s]
Accuracy: 0.6945
```

(b) (23 points) We would like to pretrain word embeddings using Word2Vec. To this end, we provide another Jupyter notebook file, `skipgram.ipynb`, which implements a version of the Skip-gram model with negative sampling. Similar to the previous problem, you are required to complete the missing code in the `SkipGram` class and fill in the first line of the `nb_finetuning` function.

Please note that some functions in `skipgram.ipynb` are the same as those in `nb_classifier.ipynb`. Again, you do not need to modify other parts of the notebook files, except for settings and configurations that may depend on your environment.

Listing 2: Outputs after running skipgram.ipynb

```
...
Epoch 1/10:  10%|=======| 937/9370 [03:37<29:29,  4.77it/s, val_loss: 1.8564]Validation
    ↪Loss: 1.8564
Epoch 2/10:  10%|=======| 937/9370 [03:38<29:29,  4.77it/s, val_loss: 1.8564]Model Saved
    ↪best loss: 1.8564
Epoch 2/10:  20%|=============| 1874/9370 [07:18<22:10,  5.64it/s, val_loss: 0.9520]
    ↪Validation Loss: 0.9520
Epoch 3/10:  20%|=============| 1874/9370 [07:18<22:10,  5.64it/s, val_loss: 0.9520]Model
    ↪ Saved best loss: 0.9520
Epoch 3/10:  30%|===================| 2811/9370 [10:57<19:26,  5.63it/s, val_loss:
    ↪0.5876]Validation Loss: 0.5876
Epoch 4/10:  30%|===================| 2811/9370 [10:57<19:26,  5.63it/s, val_loss:
    ↪0.5876]Model Saved best loss: 0.5876
Epoch 4/10:  40%|=========================| 3748/9370 [14:34<16:37,  5.64it/s, val_loss
    ↪: 0.4557]Validation Loss: 0.4557
Epoch 5/10:  40%|=========================| 3748/9370 [14:35<16:37,  5.64it/s, val_loss
    ↪: 0.4557]Model Saved best loss: 0.4557
Epoch 5/10:  50%|==============================| 4685/9370 [18:14<13:49,  5.65it/s,
    ↪val_loss: 0.4033]Validation Loss: 0.4033
Epoch 6/10:  50%|==============================| 4685/9370 [18:15<13:49,  5.65it/s,
    ↪val_loss: 0.4033]Model Saved best loss: 0.4033
Epoch 6/10:  60%|====================================| 5622/9370 [21:53<11:06,  5.62it/s,
    ↪ val_loss: 0.3795]Validation Loss: 0.3795
Epoch 7/10:  60%|====================================| 5622/9370 [21:54<11:06,  5.62it/s,
    ↪ val_loss: 0.3795]Model Saved best loss: 0.3795
Epoch 7/10:  70%|==========================================| 6559/9370 [25:34<08:19,
    ↪5.63it/s, val_loss: 0.3680]Validation Loss: 0.3680
Epoch 8/10:  70%|==========================================| 6559/9370 [25:34<08:19,
    ↪5.63it/s, val_loss: 0.3680]Model Saved best loss: 0.3680
Epoch 8/10:  80%|================================================| 7496/9370
    ↪[29:12<05:32,  5.64it/s, val_loss: 0.3616]Validation Loss: 0.3616
```

```
Epoch 9/10:  80%|=============================================| 7496/9370
    ↪[29:12<05:32,  5.64it/s, val_loss: 0.3616]Model Saved best loss: 0.3616
Epoch 9/10:  90%|==================================================| 8433/9370
    ↪[32:53<02:46,  5.64it/s, val_loss: 0.3580]Validation Loss: 0.3580
Epoch 10/10:  90%|=================================================| 8433/9370
    ↪[32:53<02:46,  5.64it/s, val_loss: 0.3580]Model Saved best loss: 0.3580
Epoch 10/10: 100%|=================================================================
    ↪===| 9370/9370 [36:33<00:00,  5.61it/s, val_loss: 0.3556]Validation Loss: 0.3556
Epoch 10/10: 100%|=================================================================
    ↪===| 9370/9370 [36:33<00:00,  4.27it/s, val_loss: 0.3556]Model Saved best loss:
    ↪0.3556
```

(c) (10 points) Revise the jupyter notebook file (`nb_classifier.ipynb`) to create (`nb_classifier_pretrained.ipynb`), where the word embedding parameterss of the classifier model is first initialized by the pretrained word embedding obtained from `skipgram.ipynb`. Please note that you will need to use a different directory to store the best model, distinguishing it from the run without the pretrained word embeddings.

Submission Instructions

You shall submit this assignment on Blackboard as two submissions – one for coding questions and another for "written" questions:

1. Run the `collect_submission.sh` script to produce your `assignment1.zip` file, and Upload your `assignment1_coding.zip` file. Please note that it is your responsibility to ensure that your code is runnable. If the code does not run, no points will be awarded.

2. Upload your written solutions `assignment1_written.pdf`.