Homework 1: Backpropogation and Neural Networks

Deep Learning (84100343-0)
Autumn 2024
Tsinghua University

1 Part One: Backpropogation

1.1 Introduction

The combination of larger datasets and increased computational power often brings about major shifts in computer vision. Convolutional Neural Networks (CNNs) have long been the de-facto standard in this field, but Vision Transformers [2] (ViT), an alternative based on self-attention layers, have achieved state-of-the-art results. ViT continues the long-lasting trend of reducing reliance on hand-crafted features and inductive biases, focusing instead on learning directly from raw data.

In this trend, the *MLP-Mixer* [14] emerges as a novel architecture shown to be a competitive but conceptually and technically simple alternative, which does not use convolutions or self-attention. As shown in Figure 1, Mixer's architecture is based entirely on multi-layer perceptrons (MLPs) that are repeatedly applied across either spatial or feature dimensions. Mixer relies only on basic matrix multiplication routines, changes to data layout (reshapes and transpositions), and scalar nonlinearities.

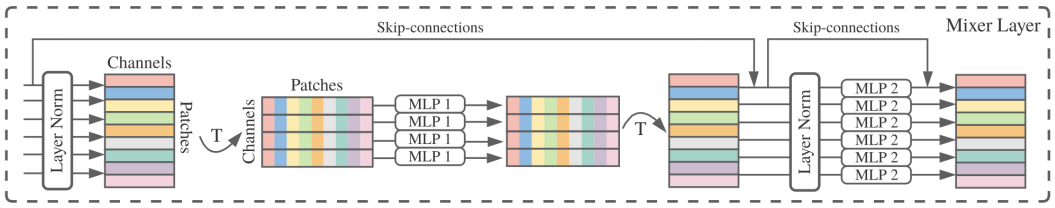

Figure 1: A mixer layer in the MLP-Mixer [14].

1.2 Network Details

Figure 2 presents a simplified mixer layer for this homework. For a mini-batch of training samples $(\mathbf{X}, \mathbf{Y})$ with a batch size of m:

1. Each input sample $\mathbf{X}^i \in \mathbb{R}^{L \times D}$ is in a mini-batch, where L represents the spatial dimension and D denotes the feature dimension. $\mathbf{Y}^i$ is a one-hot vector for classification task with K classes.
2. As illustrated by the boxes colored in *red*, FC-1, FC-2, and FC-3 represent fully connected layers. FC-1 serves as a hidden layer with L neurons, and FC-2 as another hidden layer with D neurons, both using ReLU as the activation function. FC-3 is the output layer with K neurons, employing the Softmax activation function.
3. As highlighted by the *green* boxes, $\mathrm{Transpose}$ operations swap the rows and columns of the input matrix, allowing the network to freely choose which dimension—spatial or feature—it wants to perform the mixing on.

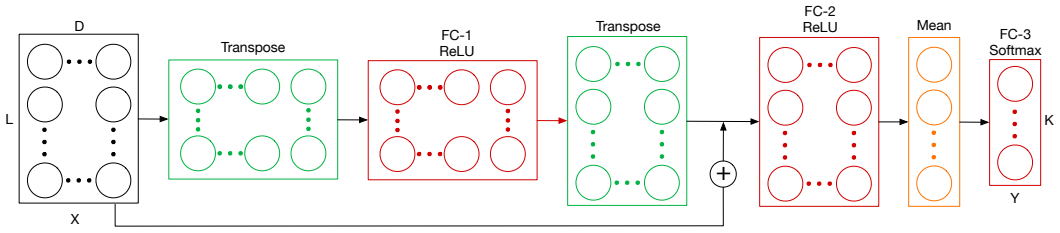

Figure 2: Schematic of the simplified mlp-mixer network in this homework.

4. As illustrated by the *orange* box, Mean operation calculates the average along the feature dimension, resulting in a vector with dimensions $L \times 1$.
5. For each sample, the notation $\bigoplus$ means *element-wise addition* of two matrices.
6. Denote weight matrices of all fully connected layers as Θ_1, Θ_2 and Θ_3, whose bias vectors are $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ respectively.
7. For each input sample $\mathbf{X}^i$, the final prediction is $\widehat{\mathbf{Y}}^i$. We use the **cross-entropy** loss:

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \left[-\sum_{k=1}^{K} \mathbf{Y}_k^i \log \widehat{\mathbf{Y}}_k^i \right] . \tag{1}$$

1.3 Task Description

For this part, please answer the following questions and submit a PDF (handwriting, Word, or LaTeX) with detailed derivation and computation.

1. Given a softmax function, please calculate the gradients of **the output of a softmax function** with respect to **its input**. (**5 points**)
2. Finish the detailed **feed-forward computations** of a batch of samples ($\mathbf{X}$, $\mathbf{Y}$) during a training iteration, coming with final predictions $\widehat{\mathbf{Y}}$. (**10 points**)
3. Use the backpropagation algorithm we have learned in class and give **the gradients of the overall loss in a mini-batch with respect to the parameters at each layer**. (**15 points**)
4. Write the pseudo-code for the stochastic gradient descent (SGD) algorithm to update the parameters at each layer during training. (**5 points**)

Note that:

1. Please show the necessary derivations of the gradients.
2. Please attach variable notations in the gradients expressions.
3. A **vectorial form** of gradient expressions is highly encouraged. The averaging within the mini-batch can be represented using the summation symbol Σ.
4. In the spirit of backpropagation, you can **use gradients of some parameters, in addition to inputs, outputs, and parameters, to represent gradients of other parameters**.

2 Part Two: Multilayer Perceptron (MLP)

Figure 3 presents a 3-layer MLP. In this part, you are required to implement and train this 3-layer neural network to forecast time series from the ETTh1 (Electricity Transformer Temperature) dataset [18], where each series consists of the target value "oil temperature" and 6 power load features.

2.1 Time Series Forecasting

We formulate the multivariate time series forecasting problem as follows. We denote historical observations of each data sample as $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\} \in \mathbb{R}^{T \times N}$ with T timesteps and N variates,

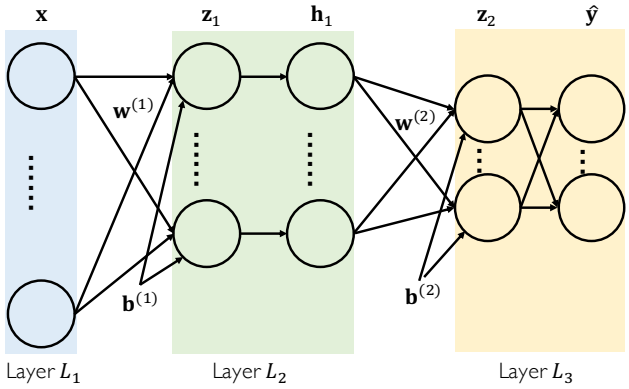

Figure 3: The network architecture of the Multilayer Perceptron (MLP).

and aim to predict the future S timesteps $\mathbf{Y} = \{\mathbf{y}_{T+1}, \ldots, \mathbf{y}_{T+S}\} \in \mathbb{R}^{S \times N}$. Recent literature [17] shows that MLPs or even linear projections can be competitive for this problem, as long as we utilize proper data preprocessing. Concretely, we decompose the series into a **trend component** and a **seasonal component** and use individual networks to predict their future:

$$\mathbf{X}' = \mathrm{Reshape}(\mathbf{X}) \in \mathbb{R}^{N \times T} \tag{2}$$

$$\mathbf{X}_\mathrm{t} = \mathrm{AvgPool}(\mathrm{Padding}(\mathbf{X}')), \quad \mathbf{X}_\mathrm{s} = \mathbf{X}' - \mathbf{X}_\mathrm{t} \tag{3}$$

$$\hat{\mathbf{Y}}_\mathrm{t} = \mathrm{MLP}_\mathrm{t}(\mathbf{X}_\mathrm{t}) \in \mathbb{R}^{N \times S}, \quad \hat{\mathbf{Y}}_\mathrm{s} = \mathrm{MLP}_\mathrm{s}(\mathbf{X}_\mathrm{s}) \in \mathbb{R}^{N \times S} \tag{4}$$

$$\hat{\mathbf{Y}} = \mathrm{Reshape}(\hat{\mathbf{Y}}_\mathrm{t} + \hat{\mathbf{Y}}_\mathrm{s}) \in \mathbb{R}^{S \times N} \tag{5}$$

Note that here we consider different variates in the same time series as independent but use the shared network to predict them.

2.2 Forward Propagation

MLP_s and MLP_t are two independent networks with the same architecture. The intermediate outputs $\mathbf{z}_1$, $\mathbf{h}_1$, $\mathbf{z}_2$, and $\widehat{\mathbf{y}}$ as the directed graph are shown below:

$$\begin{aligned} \mathbf{z}_1 &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h}_1 &= \mathrm{ReLU}(\mathbf{z}_1) \\ \mathbf{z}_2 &= \mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)} \\ \widehat{\mathbf{y}} &= \mathbf{z}_2. \end{aligned} \tag{6}$$

There is no activation function in the last layer since we are solving a regression problem.

2.3 Loss Function

After forward propagation, you should use the **mean squared error (MSE)** loss function:

$$\mathcal{L}_\mathrm{MSE} = \frac{1}{mSN} \sum_{i=1}^{m} \sum_{j=1}^{S} \sum_{k=1}^{N} \left(\mathbf{y}_{j,k}^{(i)} - \hat{\mathbf{y}}_{j,k}^{(i)} \right)^2, \tag{7}$$

where m is the number of samples in each mini-batch.

2.4 Backward Propagation

Then, similar to what we have done in Part One, you can calculate the gradient of all the parameters and use (stochastic) gradient descent to update the parameters.

2.5 Task Description

We provide the starter code in `MLP` folder. To get full scores, you **only** need to complete the `model.py` marked with **TODO**. Besides, you are free to modify other parts of the code for your convenience.

To get you comfortable with gradient derivation, in this part, we will be using `numpy`. Frameworks that support auto-derivation are **not** allowed. We highly encourage you to adopt a **vectorized** implementation to speed up computation. The concrete tasks and their scores are as follows:

1. Implement forward propagation and backward propagation algorithms. Train the network, plot the change in loss during training, and showcase some predictions from your models. (**10 points**)
2. Train the network using proper hyper-parameters (batch size, learning rate, etc) and report the test MSE, which should not exceed 0.6. (**10 points**)

3 Part Three: Convolutional Neural Network (CNN)

In recent years, scientific datasets have become increasingly important. For example, the PDB dataset [1] has played a critical role in protein structure prediction and was utilized by the notable AlphaFold, whose groundbreaking advancements in the field are recognized by the Nobel Prize. Other well-established datasets include SDSS-V [6] in astronomy and ERA5 [10] in meteorology and earth sciences.

In this part, you will implement a convolutional neural network [5] using PyTorch [9] to explore a scientific dataset. In addition to understanding the principles of CNN, we expect you to get a better grasp of PyTorch. You can learn more about the usage of PyTorch through the PyTorch Tutorial.

As we have learned in class, CNNs are made up of layers with learnable parameters, including weights and biases. Each layer takes the output of the previous layer to perform some operations and produces an output. In recent years, main-stream CNNs, such as AlexNet [7], VGG [11], GoogleNet [12], ResNet [3], DenseNet [4], EfficientNet [13] and so on, have achieved increasingly better performance on ImageNet dataset and have been leveraged to diverse applications across many fields.

In this homework, the interesting PathMNIST dataset from the MedMNIST benchmark [16] in the biomedical imaging area is provided for multi-class tissue type classification. As shown in Figure 4, it consists of 89,996 training samples, 10,004 validation samples, and 7,180 test samples, with each image resized to $3 \times 64 \times 64$ and annotated with **1 label** representing different tissue types such as *Adipose*, *Background*, and others. You are required to address the task of multi-class tissue type classification using modern convolutional neural networks (CNNs).

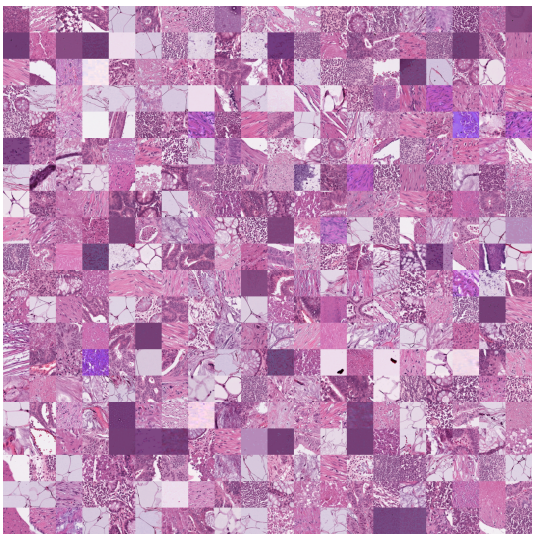

Figure 4: Example images of PathMNIST dataset.

3.1 Dataset

Dataset Description. The dataset comprises images annotated with 9 types of tissues, such as "*Adipose*", "*Background*", "*Debris*", among others. We will utilize the official split of the dataset into training, validation, and test sets for our experiments.

Download Link. `https://cloud.tsinghua.edu.cn/f/eb6b60137ef744ab881b/`.
You should place the downloaded data file under the CNN directory, or modify the data_path in the provided *start code* accordingly. The data_path refers to **the directory path** that contains the pathmnist_64.npz file.

3.2 Loss Function and Evaluation Metric

The task in this dataset is multi-class classification. We use the **cross-entropy (CE)** loss function, defined as follows:

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \left[-\sum_{k=1}^{K} \mathbf{Y}_k^i \log \widehat{\mathbf{Y}}_k^i \right]. \tag{8}$$

where $\widehat{\mathbf{Y}}^i$ is the output logits of neural networks and $\mathbf{Y}^i$ is the ground truth.

The evaluation metric is the **AUC and ACC**.

3.3 Tutorials

- **PyTorch**
  - `https://pytorch.org/tutorials/`
  - `https://github.com/yunjey/pytorch-tutorial`

3.4 Tasks Description

1. Train the model **A** (ResNet-18) from scratch on the training set, and evaluate its AUC and ACC on *test* set. Utilize a modern experiment management tool, such as *WandB* or *TensorBoard*, to track key variables throughout the training process. Report training, validation curves and test performance. Notably, the *start code* of this task is provided in `CNN` folder, and it is runnable; you can directly run it with "`python main.py`". (**5 points**)
2. Leverage a proper neural network visualization toolkit to visualize **conv features** or **saliency maps** of model **A**. (**5 points**)
   - **pytorch**: `https://github.com/utkuozbulak/pytorch-cnn-visualizations`
   - **others**: Any other toolkits if you think they are helpful...
3. Design and implement your own convolutional neural networks (CNNs) to train a new model from scratch on the *training* set and then evaluate on the given *test* set. Report training, validation curves and test performance. Your proposed model needs to be **different from ResNet** [3]. Some recipes can be borrowed from GoogLeNet [12], DenseNet [4], ResNeXt [15], ConvNeXt [8], MLP-Mixer [14], and etc. Good ideas and advanced structural design will be encouraged for higher scores. (**10 points**).
4. Use training techniques, including but not limited to **data augmentation** and **learning rate strategies**, either sourced from existing materials or proposed by yourself, to improve the performance of your model **B** and give a detailed ablation study in your report. A properly implemented model should achieve an ACC **greater than 0.9**. (**10 points**)
5. Nowadays, training a model from scratch is expensive and often less effective, making **fine-tuning a pre-trained model** on a given task crucial. You are required to fine-tune a vision model that has been pre-trained on another dataset to the **PathMNIST dataset**. When fine-tuning, you only need to replace the classification head. You should report the fine-tuning performance and discuss the differences between fine-tuning a pre-trained model and training a model from scratch, e.g. learning rate and convergence speed. (**10 points**)

   The pre-trained models can be sourced from:
   - **hugging face**: `https://huggingface.co/models`
   - **timm**: `https://github.com/rwightman/pytorch-image-models`

4 Submit Format

1. For Part One, you should submit a report (in PDF) with detailed derivation.
2. For Part Two and Three, you should submit the filled *start code* and ensure that it is runnable. Note that you also need to write a report for your experiments, which is supposed to cover the design of your model, technical details, visualization, experimental results (including training and validation curves), and the necessary references.

References

[1] Stephen K Burley, Charmi Bhikadiya, Chunxiao Bi, Sebastian Bittrich, Li Chen, Gregg V Crichlow, Cole H Christie, Kenneth Dalenberg, Luigi Di Costanzo, Jose M Duarte, et al. Rcsb protein data bank: powerful new tools for exploring 3d structures of biological macromolecules for basic and applied research and education in fundamental biology, biomedicine, biotechnology, bioengineering and energy sciences. *Nucleic acids research*, 49(D1):D437–D451, 2021.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[6] Juna Kollmeier, SF Anderson, GA Blanc, MR Blanton, KR Covey, J Crane, N Drory, PM Frinchaboy, CS Froning, JA Johnson, et al. Sdss-v pioneering panoptic spectroscopy. *Bulletin of the American Astronomical Society*, 2019.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[8] Zhuang Liu, Hanzi Mao, Chaozheng Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[10] Adrian Simmons, Cornel Soci, Julien Nicolas, Bill Bell, P. Berrisford, Rossana Dragani, Johannes Flemming, Leopold Haimberger, Sean Healy, Hans Hersbach, Andras Horányi, Antje Inness, J. Munoz-Sabater, Raluca Radu, and Dinand Schepers. Global stratospheric temperature bias and other stratospheric aspects of era5 and era5.1, 01/2020 2020.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[13] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[14] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021.

[15] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[16] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.

[17] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *AAAI*, 2023.

[18] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*, 2021.