Homework 1: Backpropogation and Neural Networks

Deep Learning (84100343-0)
Autumn 2022
Tsinghua University

多任务学习（MTL）的目的是通过联合学习，利用相关任务的共性和差异。任务之间的共同点和差异，对它们进行联合学习。它可以在各种相关的任务之间传递有用的信息，并且 已被应用于广泛的领域，如自然语言处理（NLP）和计算机视觉（CV）。视觉（CV）。在NLP中的一个例子是将多个子任务和有向无环图（DAG）的 依赖关系到判断预测[18]。另一个典型的例子在CV中被命名为 Multi-task 的多任务网络级联（MNC）[1]，它由三个网络组成。它由三个网络组成，分别是区分实例、估计掩码和分类对象，如图1所示。图1中所示。这些网络形成一个级联结构，并被设计为共享其卷积 特征。在MNC模型中，网络将任意大小的图像作为输入，并输出 实例感知的语义分割结果。级联有三个阶段：提出盒子级的 实例，回归面具级实例，以及对每个实例进行分类。每个阶段都涉及一个损失 项，但是后面阶段的损失依赖于前面阶段的输出，所以这三个损失项并不是独立的。整个网络是用一个统一的损失函数进行端到端的训练。

1 Part One: Backpropogation

1.1 Introduction

Multi-task learning (MTL) aims at exploiting the commonalities and differences across relevant tasks by learning them jointly. It can transfer useful information among various related tasks and has been applied to a wide range of areas such as natural language processing (NLP) and computer vision (CV). One example in NLP incorporates multiple subtasks and Directed Acyclic Graph (DAG) dependencies into judgment prediction [18]. Another typical example in CV is named Multi-task Network Cascades for instance-aware semantic segmentation (MNC) [1], which consists of three networks, respectively differentiating instances, estimating masks, and categorizing objects, as shown in Figure 1. These networks form a cascaded structure and are designed to share their convolutional features. In the MNC model, the network takes an image of arbitrary size as the input and outputs instance-aware semantic segmentation results. The cascade has three stages: proposing box-level instances, regressing mask-level instances, and categorizing each instance. Each stage involves a loss term, but a later stage's loss relies on the output of a previous stage, so these three loss terms are not independent. The entire network is trained end-to-end with a unified loss function.

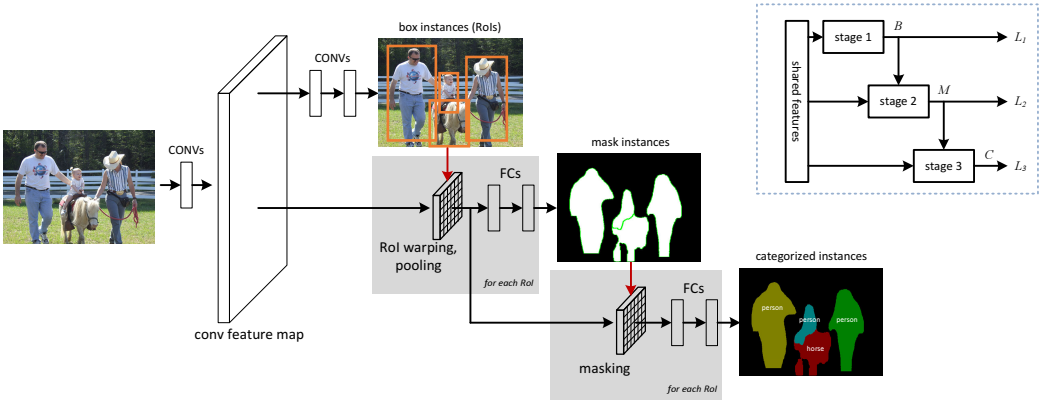

Figure 1: Multi-task Network Cascades for instance-aware semantic segmentation. The top right corner is a simplified illustration.

Compared with common multi-task learning, this new kind of multi-task cascade mechanism further explores the dependencies of several tasks. Thanks to the end-to-end training and the independence of external modules, the three sub-tasks and the entire system easily benefit from stronger features learned by deeper models [1]. As reported by the paper, this method achieves a mean Average Precision (mAP) of 63.5% on the PASCAL VOC dataset, about 3.0% higher than the previous best results using the same VGG network.

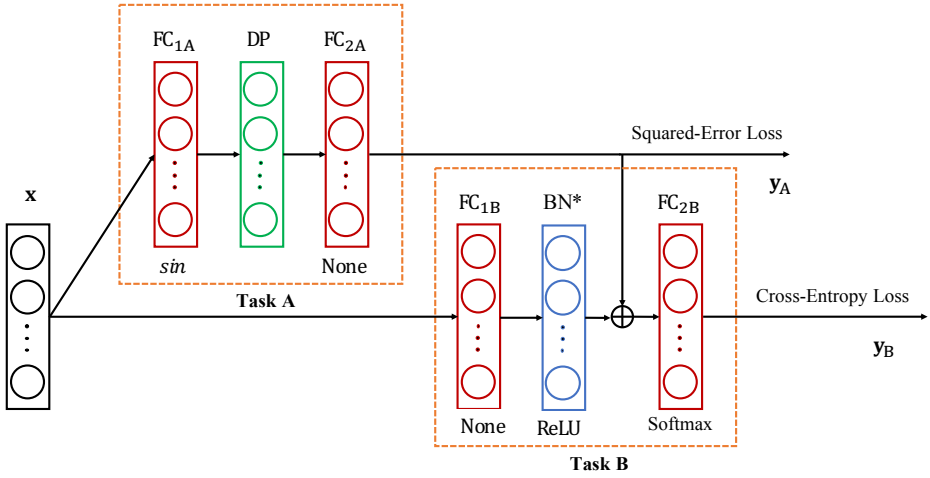

Figure 2: Schematic of the feed-forward network for multi-task learning in this homework.

1.2 Network Details

Figure 2 presents a simplified multi-task cascade network for this homework. For simplicity, we only use fully connected layers and leave out convolutional and pooling layers.

For a mini-batch of training samples ($\mathbf{x}$, $\mathbf{y}_\mathrm{A}$, $\mathbf{y}_\mathrm{B}$) with a batch size of m:

1. For each input sample $\mathbf{x}^i \in \mathbb{R}^d$ in a mini-batch, the prediction outputs for task A and task B are $\mathbf{y}_\mathrm{A}^i \in \mathbb{R}^a$ and $\mathbf{y}_\mathrm{B}^i \in \mathbb{R}^b$ respectively. $\mathbf{y}_\mathrm{A}^i$ is usually a vector with continuous value for regression task, while $\mathbf{y}_\mathrm{B}^i$ is a one-hot vector for classification task.
2. As illustrated by the boxes colored in red, $\mathrm{FC}_{1\mathrm{A}}$, $\mathrm{FC}_{2\mathrm{A}}$, $\mathrm{FC}_{1\mathrm{B}}$ and $\mathrm{FC}_{2\mathrm{B}}$ are fully-connected layers, in which $\mathrm{FC}_{1\mathrm{A}}$ and $\mathrm{FC}_{1\mathrm{B}}$ serve as hidden layers with s, t neurons respectively, while $\mathrm{FC}_{2\mathrm{A}}$ and $\mathrm{FC}_{2\mathrm{B}}$ are output layers with a, b neurons respectively.
3. Denote weight matrixs of all fully connected layers as $\theta_{1\mathrm{A}}$, $\theta_{2\mathrm{A}}$, $\theta_{1\mathrm{B}}$ and $\theta_{2\mathrm{B}}$, whose bias vectors are $\mathbf{b}_{1\mathrm{A}}$, $\mathbf{b}_{2\mathrm{A}}$, $\mathbf{b}_{1\mathrm{B}}$ and $\mathbf{b}_{2\mathrm{B}}$ respectively. Note that, $\mathrm{FC}_{1\mathrm{B}}$ is slightly different from the standard fully connected layer, detailed in (vi).
4. As illustrated by the green box, DP is the dropout layer with a random mask vector $\mathbf{M}$. In the training stage, the probability that a neuron of $\mathrm{FC}_{1\mathrm{A}}$ will be dropped is denoted as p. Formally,
$$\mathbf{M}_j = \begin{cases} 0, & r_j < p \\ 1/(1-p), & r_j \geq p \end{cases},$$
where r_j is a random value between 0 and 1 for the j^{th} neuron.
5. $\mathrm{FC}_{1\mathrm{A}}$ uses the *sine* as a periodic activation function, proposed by a recent paper [13] published in NeurIPS 2020 as an oral presentation. Neural networks with this loss are demonstrated to fit complicated signals, such as natural images and 3D shapes, and their derivatives robustly.
6. As illustrated by the blue box, BN^* is a mean-only batch normalization layer [11] for $\mathrm{FC}_{1\mathrm{B}}$. Different from the standard batch normalization [5], inputs are only subtracted by the mini-batch means, without being divided by the mini-batch standard deviations. Formally,
$$\begin{aligned} \mathbf{x}_{1\mathrm{B}} &= \theta_{1\mathrm{B}}\mathbf{x} \\ \mu &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_{1\mathrm{B}}^i \\ \mathbf{x}_{\mathrm{BN}} &= \mathbf{x}_{1\mathrm{B}} - \mu + \mathbf{b}_{1\mathrm{B}}. \end{aligned} \tag{1}$$
After that, $\mathbf{x}_{\mathrm{BN}}$ is fed into a ReLU activation function.

各分量相加

7. For each sample, the notation $\bigoplus$ in Task B means *element-wise addition* of two vectors. To enable element-wise addition, a is equal to t.
8. For each input sample $\mathbf{x}^i$, the final predictions of two tasks are $\widehat{\mathbf{y}}^i_{\mathrm{A}}$ and $\widehat{\mathbf{y}}^i_{\mathrm{B}}$. The overall loss functions include the squared-error loss of Task A and the cross-entropy loss of Task B:

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{2}\|(\widehat{\mathbf{y}}^i_{\mathrm{A}} - \mathbf{y}^i_{\mathrm{A}})\|_2^2 - \sum_{k=1}^{b}\mathbf{y}^i_{\mathrm{B},k}\log\widehat{\mathbf{y}}^i_{\mathrm{B},k}\right]. \tag{2}$$

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.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3: Batch Normalization Transform, applied to activation x over a mini-batch.

1. Given a standard BatchNorm layer, please calculate the gradients of the **output** $y_i = \mathbf{BN}_{\gamma,\beta}(x_i)$ with respect to the **parameters** of γ, β shown in Figure 3. (**5 points**)
2. Given a softmax function, please calculate the gradients of **the output of a softmax function** with respect to **its input**. (**5 points**)
3. Finish the detailed **feed-forward computations** of a batch samples ($\mathbf{x}$, $\mathbf{y}_{\mathrm{A}}$, $\mathbf{y}_{\mathrm{B}}$) during a training iteration, coming with final predictions ($\widehat{\mathbf{y}}_{\mathrm{A}}$ and $\widehat{\mathbf{y}}_{\mathrm{B}}$) of Task A and Task B. (**10 points**)
4. 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**)

Note that:

1. Please show necessary derivations of the gradients.
2. Please attach variable notations in the gradients expressions.
3. A vectorial form of gradients expressions are highly encouraged.

2 Part Two: Code Implementation

2.1 Multilayer Perceptron (MLP)

Figure 4 presents a simplified MLP for each branch of the multi-task cascade network mentioned above. In this part, you are required to implement and train a 3-layer neural network to classify images of hand-written digits from the MNIST dataset. For each example, the input to the network is a 28×28 pixel image, which is converted into a 784-dimensional vector and then the output is a vector of 10 probabilities (one for each digit). And here are some concepts you need to be familiar with.

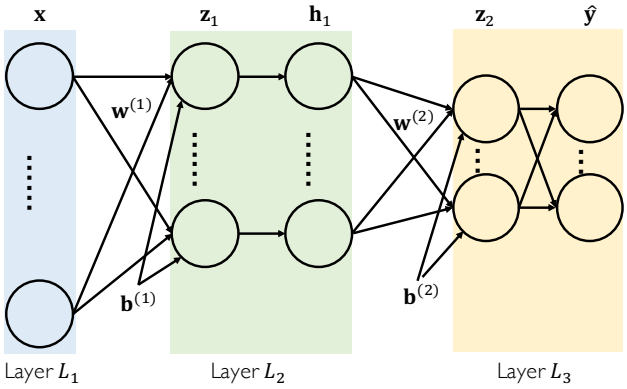

Figure 4: The network architecture of the Multilayer Perceptron (MLP).

2.1.1 Forward Propagation

The intermediate outputs $\mathbf{z}_1$, $\mathbf{h}_1$, $\mathbf{z}_2$, and $\widehat{\mathbf{y}}$ as the directed graph are shown below:

$$\begin{aligned}\mathbf{z}_1 &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\ \mathbf{h}_1 &= \mathrm{ReLU}(\mathbf{z}_1) \\ \mathbf{z}_2 &= \mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)} \\ \widehat{\mathbf{y}} &= \mathrm{Softmax}(\mathbf{z}_2).\end{aligned} \tag{3}$$

2.1.2 Loss function

After forward propagation, you should use the cross-entropy loss function:

$$f_{\mathrm{CE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{10}\mathbf{y}_k^i \log \widehat{\mathbf{y}}_k^i, \tag{4}$$

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

2.1.3 Backward Propagation

The individual gradient for each parameter term can be shown as follows:

$$\begin{aligned}\frac{\partial f_{\mathrm{CE}}}{\partial \mathbf{W}^{(2)}} &= \frac{1}{m}\sum_{i=1}^{m}(\widehat{\mathbf{y}}^i - \mathbf{y}^i)(\mathbf{h}_1^i)^\mathsf{T} \\ \frac{\partial f_{\mathrm{CE}}}{\partial \mathbf{b}^{(2)}} &= \frac{1}{m}\sum_{i=1}^{m}(\widehat{\mathbf{y}}^i - \mathbf{y}^i) \\ \frac{\partial f_{\mathrm{CE}}}{\partial \mathbf{W}^{(1)}} &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{W}^{(2)^\mathsf{T}}(\widehat{\mathbf{y}}^i - \mathbf{y}^i) \circ \mathrm{sgn}(\mathbf{z}_1^i)(\mathbf{x}^i)^\mathsf{T} \\ \frac{\partial f_{\mathrm{CE}}}{\partial \mathbf{b}^{(1)}} &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{W}^{(2)^\mathsf{T}}(\widehat{\mathbf{y}}^i - \mathbf{y}^i) \circ \mathrm{sgn}(\mathbf{z}_1^i).\end{aligned} \tag{5}$$

2.1.4 Task Description

We provide the starter code in `MLP` folder. To get full scores, you **only** need to complete the `mlp.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 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 and plot the change in loss during training. (**10 points**)
2. Train the network using proper hyper-parameters (batch size, learning rate, etc), and report the training accuracy and test accuracy. The test accuracy should exceed 90%. (**10 points**)

2.2 Convolutional Neural Network (CNN)

In this part, you will implement a convolutional neural network [6] using PyTorch [10]. 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 bias. 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 [12], GoogleNet [14], ResNet [2], DenseNet [4], EfficientNet [15] and so on, have achieved increasingly better performance on ImageNet dataset and have been leveraged to diverse applications across many fields. In this homework, a remote sensing dataset named EuroSAT [3] is provided. As shown in Figure 5, EuroSAT is a dataset based on Sentinel-2 satellite images covering 13 spectral bands and consisting out of 10 classes within a total $27,000$ labeled and geo-referenced images. You are required to solve the problem of land cover image classification by modern convolutional neural networks (CNNs).

Figure 5: Example images of EuroSAT

2.2.1 Dataset

Dataset Description. The dataset contains 10 categories of remote sensing images with different land like "*AnnualCrop*", "*River*", and etc. We will use 20% of the dataset for testing, and the remaining data for training in which each class has only 100 samples.

Directory structure. In this part, we specify the directory structure of each task.

1. The *test* set: "`./dataset/test/{AnnualCrop,...,River}/xxx.jpg`"
2. The *training* set: "`./dataset/train/{AnnualCrop,...,River}/xxx.jpg`"

Download Link. https://cloud.tsinghua.edu.cn/f/1ed981465c5d4282ab29/.

2.2.2 Tutorials

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

2.2.3 Tasks Description

Task A You are required to use the architecture of ResNet-18 to train a new model from scratch on the *training* set and then evaluate on the given *test* set. 白手起家

Task B 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. Your proposed model needs to be different from ResNet [2]. Some recipes can be borrowed from GoogleNet [14], DenseNet [4], ResNeXt [17], ConvNeXt [8], MLP-Mixer [16], and etc. Good ideas and advanced structural design will be encouraged for high scores.

To be more precisely, you need to finish the following tasks:

1. Train a model **A** for Task A, evaluate its accuracy on *test* set, and report training and test curves. Note that, the *start code* of this task is provided in `CNN` folder and it is runnable, you can directly run it and report the results with "`python main.py`". (**10 points**)
2. Train a model **B** for Task B, evaluate its accuracy on *test* set, and report training and test curves. (**10 points**).
3. Visualize the features before the last *fully-connected* layer of model **A** using t-SNE [9]. (**10 points**)
4. Use **data augmentation** and **learning rate strategy** to improve the performance of your model **B** and give a detailed ablation study in your report. (**15 points**)

3 Submit Format

1. For Part One, you should submit a report (in PDF) with detailed derivation.
2. For Part Two, you should submit the filled *start code* and ensure that it is runnable. Note that you also need to write a report for Section 2.2, which is supposed to cover your design of your model, technical details, visualization, experimental results (including training and validation curves), and the necessary references.

References

[1] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[3] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J-STARS*, 2019.

[4] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[6] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[8] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *CVPR*, 2022.

[9] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[11] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, 2016.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[13] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[15] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

[16] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.

[17] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CVPR*, 2017.

[18] H. Zhong, Z. Guo, C. Tu, C. Xiao, Z. Liu, and M. Sun. Legal judgment prediction via topological learning. In *EMNLP*, 2018.