ECE 449 Machine Learning (21Fa): Assignment 4

Xu Ke 3190110360

Nov. 30, 2021

1 Problem 1: CNN

For the given object function and $q = softmax(z)$, we can express E in terms of z, which is:

$$\begin{aligned} E = -\sum_i^N p_i ln(q_i) = -\sum_i^N p_i ln(\frac{e^{z_i}}{\sum_j^N e^{z_j}}) = \sum_i^N p_i(ln(\sum_j^N e^{z_j}) - z_i) \\ = \sum_i^N p_i ln(\sum_j^N e^{z_j}) - \sum_i^N p_i z_i = ln(\sum_j^N e^{z_j}) - \sum_i^N p_i z_i \end{aligned} \tag{1}$$

Then, let's take the derivative of E over z with the new expression of E above, which is:

$$\begin{aligned} \frac{\partial E}{\partial z_x} = \frac{\partial ln(\sum_j^N e^{z_j})}{\partial z_x} - p_x = \frac{\partial ln(\sum_j^N e^{z_j})}{\partial \sum_j^N e^{z_j}} \cdot \frac{\partial \sum_j^N e^{z_j}}{\partial z_x} - p_x \\ = \frac{e^{z_x}}{\sum_j^N e^{z_j}} - p_x = q_x - p_x, (1 \le x \le N) \end{aligned} \tag{2}$$

Hence proved.

2 Problem 2: CNN

From the question, we already know the following matrix:

$$X = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, W = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, P = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \tag{3}$$

2.1 Forward Process

For the forward process, with the Neural Network Graph, we will have:

$$X = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow[withW]{matrixmultiplication} X' = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{4}$$

$$X' = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \xrightarrow[withB]{matrixaddition} X'' = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{5}$$

$$X'' = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \xrightarrow{Relu(*)} Z = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6}$$

$$Z = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \xrightarrow{softmax} q = \begin{bmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{bmatrix} \tag{7}$$

$$q = \begin{bmatrix} \frac{e}{1+e} \\ \frac{1}{1+e} \end{bmatrix} \xrightarrow[E=-p_1 ln(q_1) - p_2 ln(q_2)]{objectfunction} E = 1.01326 \tag{8}$$

2.2 Backward Process

From the problem above, we already know that $\frac{\partial E}{\partial z_x} = q_x - p_x$, thus in this case we will have:

$$\frac{\partial E}{\partial z_x} = \begin{bmatrix} q_1 - p_1 \\ q_2 - p_2 \end{bmatrix} = \begin{bmatrix} \frac{e}{1+e} - 0.3 \\ \frac{1}{1+e} - 0.7 \end{bmatrix} \tag{9}$$

And from the forward process, we can easily get $\frac{\partial Z}{\partial X''}$, $\frac{\partial X''}{\partial B}$, and $\frac{\partial X''}{\partial X'}$, which are:

$$\frac{\partial Z}{\partial X''} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \frac{\partial X''}{\partial B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \frac{\partial X''}{\partial X'} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{10}$$

Thus, with chain rule, we can get $\frac{\partial E}{\partial B}$ and $\frac{\partial E}{\partial W}$, which are:

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Z} \cdot \frac{\partial Z}{\partial X''} \cdot \frac{\partial X''}{\partial B} = \begin{bmatrix} \frac{e}{1+e} - 0.3 \\ 0 \end{bmatrix} \tag{11}$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Z} \cdot \frac{\partial Z}{\partial X''} \cdot \frac{\partial X''}{\partial X'} \cdot \frac{\partial X'}{\partial X} = \frac{\partial E}{\partial Z} \cdot \frac{\partial Z}{\partial X''} \cdot \frac{\partial X''}{\partial X'} \cdot X^T = \begin{bmatrix} \frac{e}{1+e} - 0.3 & \frac{e}{1+e} - 0.3 \\ 0 & 0 \end{bmatrix} \tag{12}$$

2.3 Results

Then we can use the gradient descend formula to calculate the updated value of W' and B' with learning rate $\alpha = 0.02$, which are:

$$\boxed{B' = B - \alpha \cdot \frac{\partial E}{\partial B} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.02 \begin{bmatrix} \frac{e}{1+e} - 0.3 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.0086211 \\ 0 \end{bmatrix}} \tag{13}$$

$$\boxed{W' = W - \alpha \cdot \frac{\partial E}{\partial W} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} - 0.02 \begin{bmatrix} \frac{e}{1+e} - 0.3 & \frac{e}{1+e} - 0.3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.99138 & -0.00862117 \\ 0 & -1 \end{bmatrix}} \tag{14}$$

3 Problem 3: RNN

3.1 Write the formula

From the given shifted logistic activation function and the RNN graph, we can easily get the relation between h_{t+1} and h_t, which is:

$$h_{t+1} = \phi(z_t + h_t W) = \sigma(z_t + h_t W) - 0.5 \tag{15}$$

Thus, by taking the derivative at the both sides, we can easily get the relation between $\overline{h_{t+1}}$ and $\overline{h_t}$, which is:

$$\overline{h_{t+1}} = \sigma'(z_t + h_t W)\overline{h_t}W \tag{16}$$

Then, we can write the formula for the derivative h_t as a function of h_{t+1}, which is:

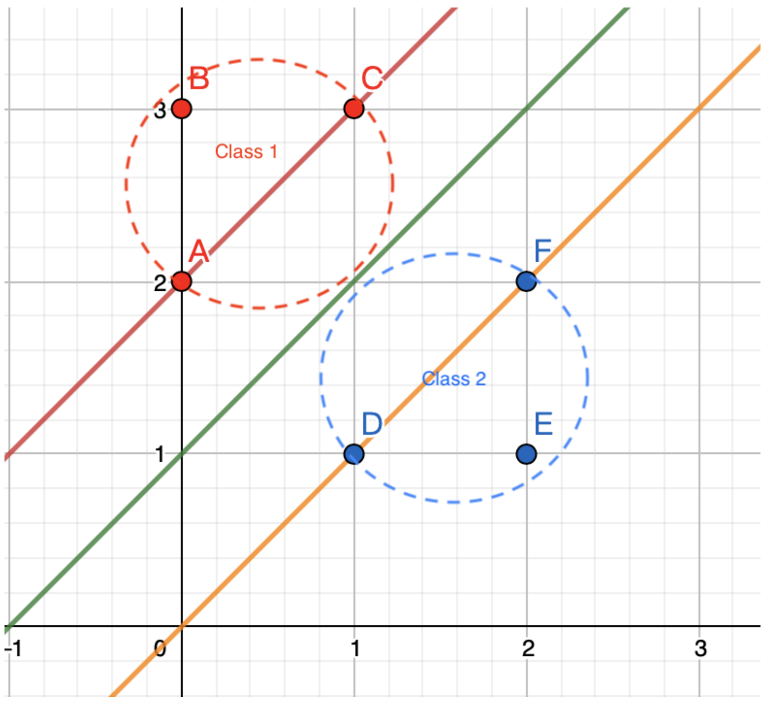
$$\boxed{\overline{h_t} = \frac{\overline{h_{t+1}}}{W\sigma'(z_t + h_t W)}} \tag{17}$$

3.2 Determine the value

From the part above, we already know that $\frac{\overline{h_{t+1}}}{\overline{h_t}} = W\sigma'(z_t + h_t W)$ and given that the input to the network is $x = 0$ and $h_t = 0$ holds for all t. Since if W ¡ α, the gradient vanishes and if W ¿ α, the gradient explodes, thus $\frac{\overline{h_{t+1}}}{\overline{h_t}}$ must be equal to 1, and since $\sigma'(0) = \frac{1}{4}$, we will have:

$$\boxed{\alpha = \hat{W} = \frac{1}{\sigma'(0)} = \frac{1}{\frac{1}{4}} = 4} \tag{18}$$

4 Problem 4: SVM



According to lecture slides, since they reach the optimum at the same time, in order to find the maximum margin, we just need to minimize $\frac{1}{2}||w||^2$, where we have $||w|| = \sqrt{w_1^2 + w_2^2}$ due to L2 norm, s.t. $y(i)(w \cdot x(i) + b) \geq 1$, for i = 1,2,3,4,5,6. Since this is a binary classification problem, we need to find one separating hyperplane. Assume Class 1 as 1 (i.e. positive) and Class as -1 (i.e. negative), then we can get the following:

$$\begin{cases} w_1 \cdot 0 + w_2 \cdot 2 + b \geq 1 \\ w_1 \cdot 0 + w_2 \cdot 3 + b \geq 1 \\ w_1 \cdot 1 + w_2 \cdot 3 + b \geq 1 \\ -w_1 \cdot 1 - w_2 \cdot 1 - b \geq 1 \\ -w_1 \cdot 2 - w_2 \cdot 1 - b \geq 1 \\ -w_1 \cdot 2 - w_2 \cdot 2 - b \geq 1 \end{cases} \tag{19}$$

the equation above can be simplified as:

$$\begin{cases} 1 - w_2 \cdot 2 - b \leq 0 \\ 1 - w_2 \cdot 3 - b \leq 0 \\ 1 - w_1 \cdot 1 - w_2 \cdot 3 - b \leq 0 \\ 1 + w_1 \cdot 1 + w_2 \cdot 1 + b \leq 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 1 + b \leq 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 2 + b \leq 0 \end{cases} \tag{20}$$

Then, we can easily get the loss function shown as following:

$$\begin{aligned} Loss(w) = \frac{1}{2}(w_1^2 + w_2^2) + \beta_1(1 - 2w_2 - b) + \beta_2(1 - 3w_2 - b) + \beta_3(1 - w_1 - 3w_2 - b) \\ + \beta_4(1 + w_1 + w_2 + b) + \beta_5(1 + w_1 + 2w_2 + b) + \beta_6(1 + 2w_1 + 2w_2 + b) \end{aligned} \tag{21}$$

Then let's take the derivative w.r.t w_1,w_2 and b respectively and assign all of $\beta_1(1 - 2w_2 - b), \beta_2(1 - 3w_2 - b), \beta_3(1 - w_1 - 3w_2 - b), \beta_4(1 + w_1 + w_2 + b), \beta_5(1 + w_1 + 2w_2 + b), \beta_6(1 + 2w_1 + 2w_2 + b)$ to be zero since they are all smaller or equal to 0 from the proof shown above as equation (22) and (23), then we will have:

$$\begin{cases} \frac{\partial Loss}{\partial w_1} = w_1 - \beta_3 + \beta_4 + 2\beta_5 + 2\beta_6 = 0 \\ \frac{\partial Loss}{\partial w_2} = w_2 - 2\beta_1 - 3\beta_2 - 3\beta_3 + \beta_4 + \beta_5 + 2\beta_6 = 0 \\ \frac{\partial Loss}{\partial b} = -\beta_1 - \beta_2 - \beta_3 + \beta_4 + \beta_5 + \beta_6 = 0 \\ 1 - w_2 \cdot 2 - b = 0 \\ 1 - w_2 \cdot 3 - b = 0 \\ 1 - w_1 \cdot 1 - w_2 \cdot 3 - b = 0 \\ 1 + w_1 \cdot 1 + w_2 \cdot 1 + b = 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 1 + b = 0 \\ 1 + w_1 \cdot 2 + w_2 \cdot 2 + b = 0 \end{cases} \tag{22}$$

Solve the equation above, we will get:

$$\begin{cases} w_1 = -1 \\ w_2 = 1 \\ b = -1 \end{cases} \tag{23}$$

Plug back to the general equation above, we will get the expression of 2-D hyperplane:

$$\boxed{-x_1 + x_2 - 1 = 0} \tag{24}$$

And from the figure above, we can easily figure out the support vectors to be the $\boxed{observation 1, 3, 4, 6}$.

5 Problem 5: Transformer

5.1 Describe the advantages of Transformer(attention mechanism) compared to RNN.

The biggest difference between Transformer and RNN is that RNN's training is iterative and serial, which means sentences must be processed word by word (i.e. the current word must be processed before the next word can be processed); While **Transformer's training is parallel**, with the help of Positional Encoding to understand language order, Self-Attention Mechanism and full-connection layer for computation, all words are trained at the same time, which greatly **increases the computational efficiency**.

The previously calculated hidden state preserves the information in the RNN, but the problem is that the encoding of a particular word is only retained at the next time-step, which means that the encoding of a word only strongly affects the representation of the next word, and its influence quickly disappears after a few time-steps. Using specific unit to have deeper processing to hide status (e.g. increase the number of parameters to train), can enlarge the learning-dependent range of RNN to some extent, and another way is to use a bidirectional model that encodes the same sentence in both directions from start to finish and from end to start, giving the word at the end of the sentence greater influence on the creation of hidden representations, which can indeed mitigate this problem but the true problem is recursion itself. **Transformer does not rely on past hidden states to capture dependencies on previous words, and does not use recursion**, but rather processes a sentence as a whole, which is why there is no risk of losing (or "forgetting") past information. In addition, multi-focus and positional embedding both provide information about the relationships between different words, **freeing Transformer from long-term dependency issues**.

5.2 Why do we need pre-trained model in NLP tasks(such as BERT)?

Pre-trained Models for Natural Language Processing: A Survey analyzes the current pre-training Language model from multiple perspectives and holds that the pre-training model has three advantages: **The pre-training model learns knowledge from a large corpus, which is of great help to downstream tasks**. **Pre-training provides a better way to initialize parameters, resulting in better generalization ability and faster convergence on the target task**.**Pre-training can be considered as a regularization method to prevent overfitting of models on small data sets**.

5.3 Why do we need positional encoding in Transformer?

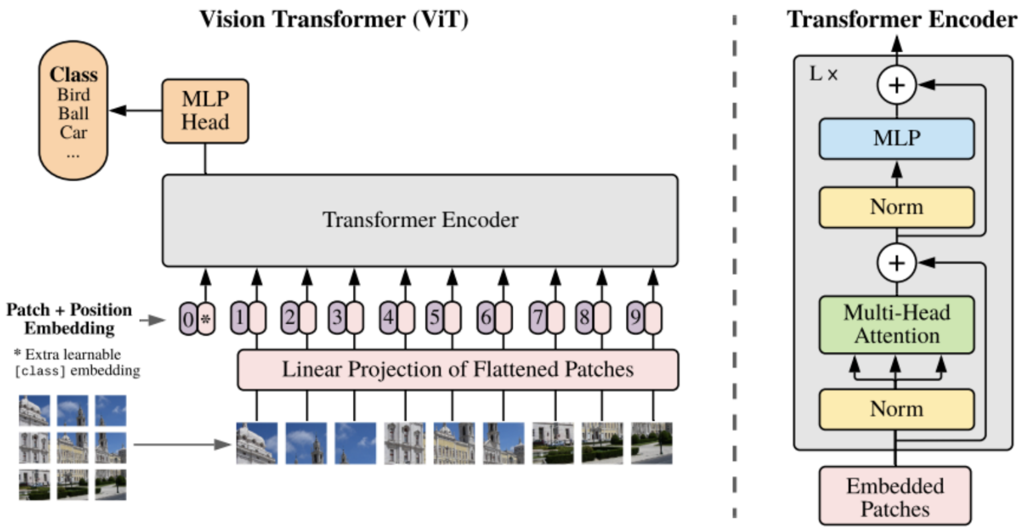
Transformer model completely abandoned the recurrent neural network in the RNN model, which is a sequential structure that naturally contains the location information between sentences. However, Transformer replaces recurrent neural network with the Self-Attention Mechanism, which **considers the permutations to be equivalent, thus leading to the loss of position information so that the model cannot know the relative and absolute position information of each word in the sentence**. In real life, there are situations in which exactly the same words are used but have opposite meanings, so it is necessary to include location information into Transformer to help the model learn this information. By the way, the experimental result shows that **Position Embedding can improve the accuracy and weaken the overfitting**.

5.4 Give an example of Transformer for vision and describe how it works.

5.4.1 Acknowledgement of the Model I Choose

Model Name: Vision Transformer
Paper Name: *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* from ICLR 2021
Author: Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby
Source Paper: http://arxiv.org/abs/2010.11929
Source Code: https://github.com/google-research/vision_transformer
Pytorch Version: https://github.com/lucidrains/vit-pytorch

5.4.2 Model Overview



5.4.3 Description of Mechanism

The figure shown above is the model overview for Vision Transformer (ViT) from *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. In order to transfer the image to corresponding 1-D sequential patch embedding, a 2-D image $x \in R^{H\times W\times C}$ will be divided into N $P \times P$ patches, where $x_P \in R^{N\times(p^2\cdot C)}$ with $N = \frac{HW}{P^2}$ to get N eigenvector with size $P^2 \cdot C$ and through MLP linear variation it will be mapped to a fix size hidden feature vector with dimension D and patch embedding $x'_P \in R^{N\times D}$ which is equivalent to the $P \times P$ convolution of x_P with step P (code is shown below). And for convenience, some relevant code copied from github are shown shown as following:

```
class PatchEmbed(nn.Module):
    """ Image to Patch Embedding
    """
    def __init__(self, img_size=224, patch_size=16, in_chans=3, embed_dim=768):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        num_patches = (img_size[1] // patch_size[1]) * (img_size[0] // patch_size
            [0])
        self.img_size = img_size
        self.patch_size = patch_size
        self.num_patches = num_patches

        self.proj = nn.Conv2d(in_chans, embed_dim, kernel_size=patch_size, stride=
            patch_size)

    def forward(self, x):
        B, C, H, W = x.shape
        assert H == self.img_size[0] and W == self.img_size[1], \
            f"Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{
                self.img_size[1]})."
        x = self.proj(x).flatten(2).transpose(1, 2)
        return x
```

Similar to the [class] token in BERT, ViT introduces the class token mechanism. As transformer input is a series of patch embedding, the output is also a sequence of patch features of the same length, while at the end it needs to be summarized as a category. A simple method can be avg-pool, in which all patch features are considered to calculate the image feature. However, instead of this approach, the author introduces a flag-like class token, whose output characteristics plus a linear classifier can achieve classification. When the class token embedding is trained, it is randomly initialized and added to the POS embedding. As shown in the figure, a new embedding is added at [0] when transformer is input, and the final input length is N+1. And for convenience, some relevant code copied from github are shown as following:

```
""" Random Initialization
"""
self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))

if num_classes > 0:
    self.head = nn.Linear(self.num_features, num_classes)
else:
    nn.Identity()

""" Forward process in detail
"""
B = x.shape[0]
x = self.patch_embed(x)
cls_tokens = self.cls_token.expand(B, -1, -1)
x = torch.cat((cls_tokens, x), dim=1)
x = x + self.pos_embed
```