Foundations of Deep Learning (AMMI)

Assignment 1 Answer Sheet

Due: Monday 12 November 2018 at 08:00

---

1 Nonlinear Activation Functions

Logistic function (sigmoid), hyperbolic tangent, and rectified linear unit (ReLU) are commonly used activation functions in deep learning. They are defined as:

`sigmoid`:

$$y = \sigma(x) = \frac{1}{1 + \exp(-x)}, \tag{1}$$

`tanh`:

$$y = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{\exp(2x) - 1}{\exp(2x) + 1}, \tag{2}$$

`ReLU`:

$$y = (x)^+ = \max(0, x), \tag{3}$$

where x is the input scalar and y is the output scalar. Assume the error back-propagated to y is $\frac{\partial E}{\partial y}$. For each activation function, write the expression for $\frac{\partial E}{\partial x}$ in terms of $\frac{\partial E}{\partial y}$.

Answers:

- $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \frac{\exp(-x)}{(1+\exp(-x))^2} = \frac{\partial E}{\partial y}(\sigma(x) - \sigma^2(x)) = \frac{\partial E}{\partial y}\sigma(x)(1 - \sigma(x))$.
- $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \frac{4\exp(2x)}{(\exp(2x)+1)^2} = \frac{\partial E}{\partial y}(1 - \tanh^2(x))$.
- $\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}$ if $x > 0$, 0 otherwise.

2 Vanishing and Exploding Gradients

Consider a fully-connected linear network with 51 layers (that considers input layer as one layer), where each layer has the *same* square $n \times n$ weight matrix $\boldsymbol{W}$. Inputs are vectors

$\boldsymbol{x} \in \mathbb{R}^n$ and outputs are vectors $\boldsymbol{y} \in \mathbb{R}^n$, and the value at hidden layer k is $\boldsymbol{h}_k = \boldsymbol{W}^k \boldsymbol{x}$. Let $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the singular value decomposition of $\boldsymbol{W}$.

Assume that we are given the vector $\frac{\partial E}{\partial \boldsymbol{y}}$, *i.e.* the gradient of the error with respect to the output, and that $\left\| \frac{\partial E}{\partial \boldsymbol{y}} \right\|_2 = 1$.

1. Write down $\frac{\partial E}{\partial \boldsymbol{h}_{k-1}}$ in terms of $\frac{\partial E}{\partial \boldsymbol{h}_k}$ and $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$, for $1 \leq k \leq 50$.
2. Let $\boldsymbol{\Sigma} = \mathbb{I}$, the identity matrix. What is the ℓ_2 norm of $\frac{\partial E}{\partial \boldsymbol{h}_1}$, *i.e.* the norm of the gradient with respect to the first layer hidden units? Explain why.
3. Let $\boldsymbol{\Sigma} = \frac{1}{2}\mathbb{I}$. What is the ℓ_2 norm of $\frac{\partial E}{\partial \boldsymbol{h}_1}$? Explain why.
4. Let $\boldsymbol{\Sigma} = 2\mathbb{I}$. What is the ℓ_2 norm of $\frac{\partial E}{\partial \boldsymbol{h}_1}$? Explain why.

Answers:

1. First note that

$$\begin{aligned} h_k &= W^k x \\ &= W \cdot W^{k-1} x \\ &= W h_{k-1} \end{aligned}$$

Therefore,

$$\frac{\partial E}{\partial h_{k-1}} = \frac{\partial E}{\partial h_k} W = \frac{\partial E}{\partial h_k} U \Sigma V^\top$$

2. Based on the above, we have

$$\frac{\partial E}{\partial h_{k-1}} = \frac{\partial E}{\partial h_k} U \Sigma V^\top = \frac{\partial E}{\partial h_k} U V^\top$$

Since U and V are orthogonal (by definition of the singular value decomposition), they do not change the norm of vectors they multiply. Therefore $||\frac{\partial E}{\partial h_{k-1}}|| = ||\frac{\partial E}{\partial h_k}||$. By induction, $||\frac{\partial E}{\partial h_1}|| = ||\frac{\partial E}{\partial h_{49}}|| = 1$.

3. Using a similar argument as above, we have

$$\frac{\partial E}{\partial h_{k-1}} = \frac{\partial E}{\partial h_k} U \Sigma V^\top = \frac{\partial E}{\partial h_k} \frac{1}{2} U V^\top$$

and $||\frac{\partial E}{\partial h_{k-1}}|| = \frac{1}{2}||\frac{\partial E}{\partial h_k}||$. By induction, $||\frac{\partial E}{\partial h_1}|| = \frac{1}{2^{49}}||\frac{\partial E}{\partial h_{49}}|| = \frac{1}{2^{49}}$

4. Using a similar argument as above, we have

$$\frac{\partial E}{\partial h_{k-1}} = \frac{\partial E}{\partial h_k} U \Sigma V^\top = \frac{\partial E}{\partial h_k} 2 U V^\top$$

and $||\frac{\partial E}{\partial h_{k-1}}|| = 2||\frac{\partial E}{\partial h_k}||$. By induction, $||\frac{\partial E}{\partial h_1}|| = 2^{49}||\frac{\partial E}{\partial h_{49}}|| = 2^{49}$

Another way of answering questions 2, 3, 4 at once is by first using the answer of 1) to show that:

$$\frac{\partial E}{\partial h_1} = \frac{\partial E}{\partial y} W^{49}$$

For $\Sigma = c\mathbb{I}$, where $c \in \{1, 1/2, 2\}$:

$$\begin{aligned}
\left\| \frac{\partial E}{\partial h_1} \right\|_2 &= \sqrt{\frac{\partial E}{\partial h_1} \frac{\partial E}{\partial h_1}^\top} \\
&= \sqrt{\frac{\partial E}{\partial h_y} (W^{49})(W^\top)^{49} \frac{\partial E}{\partial h_y}^\top} \\
&= \sqrt{\frac{\partial E}{\partial h_y} (U\Sigma V^\top) \cdots (U\Sigma V^\top)(V\Sigma U^\top) \cdots (V\Sigma U^\top) \frac{\partial E}{\partial h_y}^\top} \\
&= \sqrt{\frac{\partial E}{\partial h_y} (U(c\mathbb{I})V^\top) \cdots (U(c\mathbb{I})V^\top)(V(c\mathbb{I})U^\top) \cdots (V(c\mathbb{I})U^\top) \frac{\partial E}{\partial h_y}^\top} \\
&= \sqrt{c^{98} \frac{\partial E}{\partial h_y} (UV^\top) \cdots (UV^\top)(VU^\top) \cdots (VU^\top) \frac{\partial E}{\partial h_y}^\top} \\
&= c^{49} \sqrt{\frac{\partial E}{\partial h_y} \frac{\partial E}{\partial h_y}^\top} \\
&= c^{49} \cdot 1
\end{aligned}$$

and then plugging in $c = 1, 1/2, 2$. We used the fact that that $V^\top V = UU^\top = \mathbb{I}$ to reduce the big product in the third-to-last step.

3 Sentence classification

Sentence classification is a common problem in neural language processing (NLP). There are many ways to deal with this problem. As simplest, you could just use machine learning algorithms such as logistic regression, or any you can think of, such as a multi-layer perceptron (MLP). You could even get help from CNNs or RNNs, which can give you good sentence vectors, which you can feed into a linear classification layer. The goal of this question is to make sure that you have a clear picture in your mind about all these possible techniques.

Let's say you have a corpus of 50k words. For the simplicity of this question, assume you have the trained word embedding matrix of size 50k $\times$ 300 available to you, which can give you a word vector of size 1×300. Consider one sentence having 10 words for the classification task and describe your approach for the techniques below.

(a) CNN

1. Design a one-layer CNN which first maps the sentence to a vector of length 5 (with the help of convolution and pooling), then feeds this vector to a fully connected layer with soft(arg)max to get the probability values for possible 3 classes.

Answers:

(a) Input: $X_in \in R^{10*300}$, word embedding matrix of 10-word sentence.

(b) Convolution Layer: Choose more kernels and smaller size to improve training speed. Use 5 kernels, each of wich is $3 * 300$, zero padding size=1, stride size=1. This setting will map the input into a tensor of$X_c \in R^{10*1*5}$.

(c) Max Pooling Layer: Choose Max Pooling to capture features in words. With the size $10 * 1$ pooling layer, the input X_c turns into $X_p0 \in R^{1*5}$, transpose it into $X_p = X_p0^T \in R^{5*1}$.

(d) Fully Connected Layer: Since we need 3-class values, the FC Layer need to map tensor to R^3. So the wight matrix $w \in R^{3*5}$, and with bias item $b \in R^{3*1}$, we get $X_f = wX_p + b \in R^{3*1}$.

(e) Softmax Layer: Map $X_f \in R^{3*1}$ into $X_{out} \in R^{3*1}$, each item in X_f turns into $X_{out}[i] = \frac{e^{X_f[i]}}{\Sigma e^{X_f[i]}}$, thus $\sum X_{out}[i] = 1$.

2. Clearly mention the sizes for your input, kernel, outputs at each step (till you get the final 3×1 output vector from soft(arg)max).

   Answers:

   (a) Input size: $X_in \in R^{10*300}$.

   (b) Input for Conv: $X_in \in R^{10*300}$, Conv Kernel size: $3 * 300$, Conv Kernel number: 10, output: $X_c \in R^{10*1*5}$.

   (c) Input for Pooling: $X_c \in R^{10*1*5}$, Pooling kernel: $10*1$, output: $X_p = X_p0^T \in R^{5*1}$.

   (d) Input for FC Layer: $X_p \in R^{5*1}$, FC kernel: $w \in R^{3*5}$ FC output: $X_f \in R^{3*1}$.

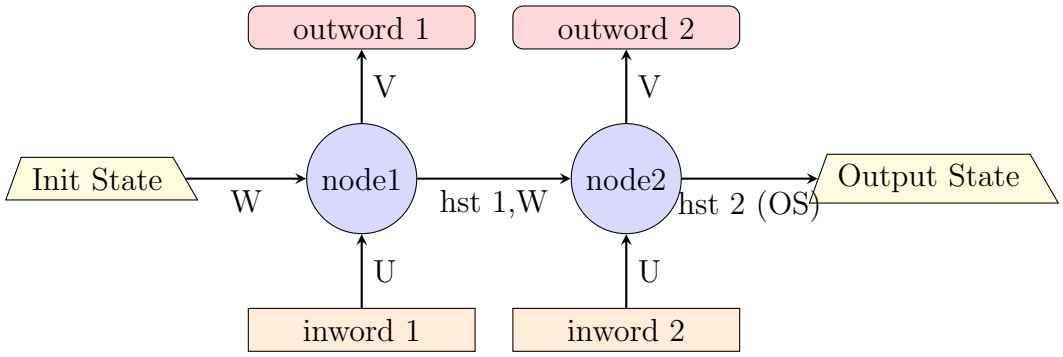
   (e) Input for SoftMax: $X_f \in R^{3*1}$, kernel:$\in R^{3*3}$, output: $X_out \in R^{3*1}$.

3. Please describe the effect of small filter size *vs.* large filter size during the convolution? What would be your approach to select the filter sizes for classification task? Answers: For the NLP problems, the second dimension of kernel(300) should be fixed since it's the size of word vector, and the first dimension shows how much information the kernel captures from the language. Generally speaking, larger kernel size means able to capture information from longer phrase. So when trying to classify sentences from literature which has long sentences with complicated Attributive and articles, we should try larger kernels (maybe 5*300 or 7*300). When dealing with chats, comments

from social network or online shopping sites, we should use smaller kernels to capture shorter phrases. In real-word application, I will try different sizes on one set and trade off between higher performance and faster training speed.

(b) RNN

1. How can a simple RNN which is trained for language modelling be used to get the sentence vector?



Answers: For a general RNN trained for language modelling, the flow chat is shown above, which takes one word and one state from former node as input, and give a word and a state as output. But for the classification problem which focuses on getting a sentence vector as output, we don't need the word output, only the hidden states in each step and the final state should be kept. So we won't need a V matrix, and a fully connected layer and softmax might be needed after the final layer to map output state to a probability vector.

2. Design a simple RNN which first maps the sentence to a vector of length 50, then feeds this vector to fully connected layer with soft(arg)max to get the probability values for possible 4 classes. Answers:

   (a) Structure: Totally 10 nodes in this layer. As a simple RNN, we just introduce one layer. Only one output which is the final state vector, i.e. vector of length 50.

   (b) Input: A vector of sentence $X_0 \in R^{10*300}$, each word $X_0^i \in R^{1*300}$ as a input to a RNN node. Also a initial state vector $s_0 \in R^5 0 * 1$ is needed as input state, initialize it as 0 vector.

   (c) W Matrix and U matrix: Since we need output state to be $50 * 1$, U should be $50 * 300$ and W should be $50 * 50$ so that $S_t = tanh(UX_t + WS_{t-1}) \in R^{50,1}$.

   (d) V matrix: Since we don't need output words, V could be 0.

(e) Fully connected layer: To map the vector to R^{4*1}, the FC kernel $w \in R^{4*50}$.

(f) SoftMax: Map $X_f \in R^{4*1}$ into $X_{out} \in R^{4*1}$, each item in X_f turns into $X_{out}[i] = \frac{e^{X_f[i]}}{\Sigma e^{X_f[i]}}$, thus $\sum X_o ut[i] = 1$.

3. Clearly mention the sizes of all the RNN components such as your input vector, hidden layer weight matrix, hidden state vectors, cell state vector, output layers (RNN components sizes would be same at each time step). Answers:

(a) Input size: $X_i n \in R^{10*300}$, $s_0 \in R^5 0 * 1$.

(b) RNN Kernel and output size: Input for W is $s_t \in R^5 0 * 1$, size W is $50 * 50$; input for U is $X_t \in R^{1*300}$, size U is $50 * 300$, output $S_t \in R^{50,1}$.

(c) Input for FC Layer: $s_{tout} \in R^{50,1}$, FC kernel size: $w \in R^{4*50}$, FC output: $s_f \in R^{4*1}$.

(d) Input for SoftMax Layer: $s_f \in R^{4*1}$, Softmax kernel: $\in R^{4*4}$, final output: $s_{out} \in R^{4*1}$.

4 Image classification

You would be using the MNIST dataset and building a CNN to classify the digits 0 to 9 in the dataset. In our practical session, the code we demonstrated uses hard-coded matrix sizes. However, this is not scalable when you expand your model's capacity through a more complex architecture (deeper, wider, etc.). In this programming exercise, we will be creating a class so we can change arguments easily without manually calculating our tensor sizes and hard-coding them into our code.

4.1 New Model Class

You are required to create a model that has the following architecture: (1) input, (2) convolution, (3) pooling, (4) convolution, (5) pooling, (6) linear, and (7) soft(arg)max.

Instead of a simple model class we demonstrated in class (please refer to that class to get a sense of what is expected), you are required to add a few more arguments and modify the class accordingly.

`CNN(**kwargs)`

1. `conv_kernel_size (int)`

   This is the size of the convolution kernel. In our class, we have used the size of 3.

2. `pooling_kernel_size (int)`

   This is the size of the pooling kernel.

3. `stride_size (int)`

   This is the stride size of the kernel. We have covered only a stride size of 1 in class.

4. `zero_padding (bool)`

   If True, the model uses zero padding (also called same padding). Else, the model uses valid padding (no padding).

5. `max_pooling (bool)`

   If True, the model uses Max Pooling. Else the model uses Average Pooling.

4.2 Zero Padding Model Tensors

After you have defined your model's class, you are required to instantiate the model's class with the following values for your new arguments and assign it to an object. Print out all of the model's parameters.

For the convolutional kernel size, the size of 5 implies a 5 by 5 kernel size, this logic applies to the pooling kernel size too.

```
model = CNN(**kwargs)
```

1. `conv_kernel_size (int) = 5`
2. `pooling_kernel_size (int) = 2`
3. `stride_size (int) = 1`
4. `zero_padding (bool)=True`
5. `max_pooling (bool)=True`

Answers:

1. Take note that a lot of students did not factor into account that the question required a stride size of 1 (on convolutions and pooling). For PyTorch, pooling layers assume a stride equal to your kernel size, so you have to pass an argument for stride to be equal to 1. The following size syntax follows this format: (depth x width x height).
2. Input size: 1 x 28 x 28
3. Output of first convolution: 16 x 28 x 28
4. Output of first pooling: 16 x 27 x 27
5. Output of second convolution: 32 x 27 x 27
6. Output of second pooling: 32 x 26 x 26
7. Hence before our final linear layer we have to flatten this tensor to 1 x 21632

4.3 Valid Padding Model Tensors

Here we will do the same task as above but with valid padding (no padding). Print out all of the model's parameters.

```
model = CNN(**kwargs)
```

1. `conv_kernel_size (int) = 5`
2. `pooling_kernel_size (int) = 2`
3. `stride_size (int) = 1`
4. `zero_padding (bool) = False`
5. `max_pooling (bool) = True`

Answers:

1. Take note that a lot of students did not factor into account that the question required a stride size of 1 (on convolutions and pooling). For PyTorch, pooling layers assume a stride equal to your kernel size, so you have to pass an argument for stride to be equal to 1. The following size syntax follows this format: (depth x width x height).
2. Input size: 1 x 28 x 28
3. Output of first convolution: 16 x 24 x 24
4. Output of first pooling: 16 x 23 x 23
5. Output of second convolution: 32 x 19 x 19
6. Output of second pooling: 32 x 18 x 18
7. Hence before our final linear layer we have to flatten this tensor to 1 x 10368

Intructions

For Section 1 to 3, you can write down your solutions on paper. However, for Section 4, you are required to send us your Jupyter Notebook for grading.

Submission

Evaluation

Your grade for this assignment will be based on:

- 10% - Section 1.
- 15% - Section 2.
- 25% - Section 3a.
- 25% - Section 3b.
- 25% - Section 4 (programming).

Submission

Send your notebook to ritchieng@u.nus.edu and pass your solutions to Ritchie at the start of Monday's class. When you send your notebook, please use the following format as your email header.

```
[Name Surname] Submission HW1
```