Deep Learning Assignment 10 in FS 2022

Learn to Write Like Shakespeare

Microsoft Forms Document:
https://forms.office.com/r/xs1Xb1pe3g

Manuel Günther

Distributed: Friday, May 13, 2022
Discussed: Friday, May 20, 2022

In this assignment we will implement a simple recurrent Elman network with one hidden layer. We train this network on a medium-size poem "The Sonnet" written by William Shakespeare and use it for auto-completing sentences/phrases starting from different seeds.

For training the network, we will need to transform the text into something machine-processable. Basically, for each of the characters in the text, we provide a D-element one-hot encoding vector, where D is the total number of unique characters in the dataset. Character sequences of length S will, hence, be turned into matrices of size $\mathbf{X} = \{\vec{x}^{\{s\}}, 1 \leq s \leq S\} \in \mathbb{R}^{S\times D}$. For each input, we provide the target values $\mathbf{T}$ of the same size, where the target for each sample is the next character: $\vec{t}^{\{s\}} = \vec{x}^{\{s+1\}}$.

To speed up processing, these sequences will be put into batches, i.e., $\mathcal{X}, \mathcal{T} \in \mathbb{R}^{B\times S\times D}$. This will automatically be achieved using the default PyTorch `DataLoader`.

1 Data and Targets

First, we need to load the whole dataset $\vec{c} \in \mathbb{R}^N$, a vector of characters, and turn the data sequence into one-hot encodings. For this purpose, we need to know that number D of unique characters in our text. For simplicity, we only consider lower-case characters and special symbols such as punctuation marks. Also, the newline characters `\n` need to be handled – you can also leave them inside and see what happens.

Then, for each of the characters, we need to assign a one-hot encoding, and build sequences of encodings. For a given index n into out data and a given sequence length S, we provide the input $\mathbf{X}^{[n]}$ and the target $\mathbf{T}^{[n]}$ as follows:

$$\mathbf{X}^{[n]} = \{\mathrm{enc}(n - S + s - 1) | 1 \leq s \leq S\} \qquad \mathbf{T}^{[n]} = \{\mathrm{enc}(n - S + s) | 1 \leq s \leq S\} \tag{1}$$

where enc is a function that returns the one-hot encoding from Task 2 for the character at the specified location in the original dataset $\vec{c}$. In the case that the computation ($n - S + s - 1$ or $n - S +$) results in a negative value $\vec{0}$ should be used instead. For example, for the original text `abcde`, sequence length $S = 7$ and index $n = 4$, we would have the representations for $x =$`000abcd` and $t =$`00abcde`.

Finally, we implement our own `Dataset` that returns the input and target encodings for any element of our input text.

Task 1: Data Characteristics Iterate through the poem file and collect all the data that we want to learn from. Make sure to handle newline characters in one way or the other. Create a list of unique characters contained in our data and obtain the dimension D.

Task 2: One-hot Encoding Implement a dictionary that provides a unique one-hot encoding for each of the characters in the dataset. The dictionary takes as the key a character and its value is its one-hot vector representation of dimension D.

Task 3: Sequence Coding Implement a function that provides the sequence encoding of the input $\mathbf{X}^{[n]}$ and the targets $\mathbf{T}^{[n]}$ for a given index n and a sequence length S according to (1).

Test 1: Sequences Get the sequences $\mathbf{X}^{[n]}$ and $\mathbf{T}^{[n]}$ of length $S = 5$ at index $n = 2$ using the function from Task 3. Assure that the provided data is correct, i.e., prepended with 0 vectors and using one-hot encodings afterward.

Task 4: Dataset and Data Loader Implement a `Dataset` class derived from `torch.utils.data.Dataset` that provides $\mathbf{X}^{[n]}$ and the targets $\mathbf{T}^{[n]}$. Implement three functions. The constructor `__init__(self, data, S)` takes the dataset $\vec{c}$ and an (initial) sequence length S. The index function `__getitem__(self, index)` should return the sequences $\mathbf{X}^{[n]}$ and $\mathbf{T}^{[n]}$ for the given `index` n; the function from Task 3 can be employed for this. Finally, `__len__(self)` should return N, i.e., the number of samples in out dataset. After implementing the `Dataset`, instantiate a `DataLoader` for the dataset with batch size of $B = 256$.

Test 2: Data Sizes For all batches in the data loader, check that the shape of the input batch $\mathcal{X}$ and target batch $\mathcal{T}$ are appropriate. Also make sure that the inputs and target contents align according to (1).

2 Elman Network Training

While there are implementations for recursive networks available in PyTorch, we here attempt our own implementation of the Elman network. The input to our network is a batch of sequences of dimension $\mathcal{X} \in \mathbb{R}^{B\times S\times D}$. Our network contains three fully-connected layers with dimensions $\mathbf{W}^{(1)} \in \mathbb{R}^{K\times D}$, $\mathbf{W}^{(r)} \in \mathbb{R}^{K\times K}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{D\times K}$ (plus bias neurons as handled by PyTorch). The network processing will iterate through our sequence, and processes all elements in the batch simultaneously. First, the hidden activation $\mathbf{H} \in \mathbb{R}^{B,K}$ is initialized with 0. Then, we iterate over $1 \leq s \leq S$ to process:

$$\mathbf{A}^{\{s\}} = \mathbf{W}^{(1)}\mathbf{X}^{\{s\}} + \mathbf{W}^{(r)}\mathbf{H}^{\{s-1\}} \qquad \mathbf{H}^{\{s\}} = g\big(\mathbf{A}^{\{s\}}\big) \qquad \mathbf{Z}^{\{s\}} = \mathbf{W}^{(2)}\mathbf{H}^{\{s\}} \tag{2}$$

where g is the activation function, PReLU in our case, and $\mathbf{X}^{\{s\}}$ is the data matrix stores as $\mathcal{X}_{:,s,:}$. The final output of our network $\mathcal{Z}$ is a combination of all $\mathbf{Z}^{\{s\}}$ matrices in dimension as our input $\mathcal{Z} \in \mathbb{R}^{B\times S\times D}$.

For training, we need to compare the output $\mathcal{Z}$ of our network with our target batch $\mathcal{T}$. We will make use of the categorical cross-entropy loss as implemented in PyTorch's `torch.nn.CrossEntropyLoss`. In our case, we will implicitly compute:

$$\mathcal{J}^{\text{CCE}} = \frac{1}{SB}\sum_{b=1}^{B}\sum_{s=1}^{S}\sum_{d=1}^{d}\mathcal{T}_{b,s,d}\log\mathcal{Y}_{b,s,d} \tag{3}$$

where $\mathcal{Y}_{b,s,d}$ is the result of SoftMax over the dimension D, which is the last index of our tensor. As the documentation of `torch.nn.CrossEntropyLoss` states, the SoftMax is always computed across the **second** dimension of the data tensor (which would be s in our case). Hence, we need to make sure to reorder the dimensions of the tensors $\mathcal{X}$ and $\mathcal{T}$ such that the computation is correct.

Task 5: Elman Network Implementation Implement the Elman network by deriving a class from `torch.nn.Module`. In the constructor, instantiate all required layers and activation functions for given values of D and K. In the `forward` function, implement the iterative processing according to (2). Store all resulting logits in matrix $\mathcal{Z}$, which is finally returned.

Test 3: Network Output Instantiate an Elman network with arbitrary numbers for D and K. Provide an exemplary batch for any value of S and B to the network. Assure that the network output is in the expected shape.

Task 6: Training Loop Instantiate an Elman network with the input dimension D obtained in Task 1, and hidden layer dimension $K = 1000$. Instantiate an Adam optimizer (SGD takes much longer to reduce the loss) with an appropriate learning rate η. Instantiate the loss function. Implement the training loop for 10 epochs (more epochs will generate better results). Assure that the loss function obtains the data in the correct format. Compute the training set loss over the epoch and report it at the end. Possibly, change the sequence length $S \in [5, 20]$ after each training batch by adapting parameters in the `Dataset`.

seed	**best output**	**sampled output**
`th`	the story of thee this strangely pass, and scarcely greststt converted from the sta	thou is his diss and and meant the deach of beauty os my loves as end meas. form a
beau	beauty thou wilt take, thou best of deains the tillage of the truth'n of both, and t	beauty shows the worst was thy humour doth and with heat, no, i that power ald are,
mothe	mother's glass hid err that thou shalt find thus makes but cours morr worts and to hi	mother's would by ill believing thingsteration of hers buttone, cansed of self grow.
bloo	blood which youngly those that stell of good, or evil luck, of plagues, of deaves sh	blooms have full as deeptate of youth in each st that i come so near, sweet to thee
q	quite, for i me deathroom thou wilt swift-foot shale rone, nor east sure i heat a	quity good a poor heam and men. yet this might, and by and by clespest of my pime
wh	when i against my self with thee shall not be tomb ex then seems? thou the beauty	when i (perhaps) compound so thou bear'st love to any whe hash on love to thy and

Table 1: Exemplary output of the text production for some seed strings of an Elman network trained for 20 epochs.

3 Writing a Poem

With the trained network, we will turn that network into a poet. Given some initial seed strings, we let the network predict the next character, which we append to our text. We repeat this process until we have produced a given string size.

For this purpose, we need to implement three functions. The first function needs to turn a given text into something that the network understands as input. The second function needs to interpret the network output, i.e., it needs to select a character from the predicted logits. There, we can implement two ways: First, we take the character with the highest predicted class:

$$o^* = \arg\max_o \vec{y}_o^{\{S\}} \tag{4}$$

Second, we can also perform probability sampling, where each of the sample is drawn based on the probability that SoftMax provides – such a function is for example implemented in `random.choices`.

Task 7: Text Encoding Write a function that produces the encoding $\mathcal{X} \in \mathbb{R}^{1\times S\times D}$ according to (1) for a given text, i.e., a sequence of S characters, with a batch size of 1.

Task 8: Next Element Prediction Write a function that predicts the next character of the sequence based on the logit values obtained from the network. Implement both ways, i.e., using the maximum probability or using probability sampling – use a Boolean parameter of your function to distinguish the two cases. Note that in our case, we are only interested in the logit for the last element of our sequence, i.e., $\vec{y}^{\{S\}}$.

Task 9: Sequence Completion Write a function that takes a seed text and appends characters to the text as predicted by the network using the two functions from above. Append as many as 80 characters to the seed text and return it.

Task 10: Text Production Define several seed strings (such as `"the ", "beau", "moth"` or alike). Call the function of Task 9 to complete the text twice, using the best prediction or probability sampling. Write the results to console. Exemplary outputs can be found in Table 1.