C3_W3_Assignment

September 20, 2022

1 Assignment 3 - Named Entity Recognition (NER)

Welcome to the third programming assignment of Course 3. In this assignment, you will learn to build more complicated models with Trax. By completing this assignment, you will be able to:

- Design the architecture of a neural network, train it, and test it.
- Process features and represents them
- Understand word padding
- Implement LSTMs
- Test with your own sentence

1.1 Important Note on Submission to the AutoGrader

Before submitting your assignment to the AutoGrader, please make sure you are not doing the following:

1. You have not added any *extra* `print` statement(s) in the assignment.
2. You have not added any *extra* code cell(s) in the assignment.
3. You have not changed any of the function parameters.
4. You are not using any global variables inside your graded exercises. Unless specifically instructed to do so, please refrain from it and use the local variables instead.
5. You are not changing the assignment code where it is not required, like creating *extra* variables.

If you do any of the following, you will get something like, `Grader not found` (or similarly unexpected) error upon submitting your assignment. Before asking for help/debugging the errors in your assignment, check for these first. If this is the case, and you don't remember the changes you have made, you can get a fresh copy of the assignment by following these instructions.

1.2 Outline

- Section **??**
- Section **??**
  - Section **??**
  - Section **??**
    - Section **??**
- Section **??**
  - Section **??**
- Section **??**

    - Section ??
- Section ??
    - Section ??
- Section ??

Introduction

We first start by defining named entity recognition (NER). NER is a subtask of information extraction that locates and classifies named entities in a text. The named entities could be organizations, persons, locations, times, etc.

For example:

Is labeled as follows:

- French: geopolitical entity
- Morocco: geographic entity
- Christmas: time indicator

Everything else that is labeled with an `O` is not considered to be a named entity. In this assignment, you will train a named entity recognition system that could be trained in a few seconds (on a GPU) and will get around 75% accuracy. Then, you will load in the exact version of your model, which was trained for a longer period of time. You could then evaluate the trained version of your model to get 96% accuracy! Finally, you will be able to test your named entity recognition system with your own sentence.

[1]:
```
import os
import numpy as np
import pandas as pd
import random as rnd
import w3_unittest
import trax

from utils import get_params, get_vocab
from trax.supervised import training
from trax import layers as tl

# set random seeds to make this notebook easier to replicate
rnd.seed(33)
```

Part 1: Exploring the data

We will be using a dataset from Kaggle, which we will preprocess for you. The original data consists of four columns: the sentence number, the word, the part of speech of the word, and the tags. A few tags you might expect to see are:

- geo: geographical entity
- org: organization
- per: person
- gpe: geopolitical entity
- tim: time indicator
- art: artifact

- eve: event
- nat: natural phenomenon
- O: filler word

[2]:
```
# display original kaggle data
data = pd.read_csv("data/ner_dataset.csv", encoding = "ISO-8859-1")
train_sents = open('data/small/train/sentences.txt', 'r').readline()
train_labels = open('data/small/train/labels.txt', 'r').readline()
print('SENTENCE:', train_sents)
print('SENTENCE LABEL:', train_labels)
print('ORIGINAL DATA:\n', data.head(5))
del(data, train_sents, train_labels)
```

```
SENTENCE: Thousands of demonstrators have marched through London to protest the
war in Iraq and demand the withdrawal of British troops from that country .

SENTENCE LABEL: O O O O O O B-geo O O O O O B-geo O O O O O B-gpe O O O O O

ORIGINAL DATA:
     Sentence #           Word  POS Tag
0  Sentence: 1      Thousands  NNS   O
1           NaN             of   IN   O
2           NaN  demonstrators  NNS   O
3           NaN           have  VBP   O
4           NaN        marched  VBN   O
```

1.1 Importing the Data

In this part, we will import the preprocessed data and explore it.

[3]:
```
vocab, tag_map = get_vocab('data/large/words.txt', 'data/large/tags.txt')
t_sentences, t_labels, t_size = get_params(vocab, tag_map, 'data/large/train/
 ↪sentences.txt', 'data/large/train/labels.txt')
v_sentences, v_labels, v_size = get_params(vocab, tag_map, 'data/large/val/
 ↪sentences.txt', 'data/large/val/labels.txt')
test_sentences, test_labels, test_size = get_params(vocab, tag_map, 'data/large/
 ↪test/sentences.txt', 'data/large/test/labels.txt')
```

`vocab` is a dictionary that translates a word string to a unique number. Given a sentence, you can represent it as an array of numbers translating with this dictionary. The dictionary contains a `<PAD>` token.

When training an LSTM using batches, all your input sentences must be the same size. To accomplish this, you set the length of your sentences to a certain number and add the generic `<PAD>` token to fill all the empty spaces.

[4]:
```
# vocab translates from a word to a unique number
print('vocab["the"]:', vocab["the"])
# Pad token
print('padded token:', vocab['<PAD>'])
```

```
vocab["the"]: 9
padded token: 35180
```

The `tag_map` is a dictionary that maps the tags that you could have to numbers. Run the cell below to see the possible classes you will be predicting. The prepositions in the tags mean: * I: Token is inside an entity. * B: Token begins an entity.

[5]:
```
print(tag_map)
```

```
{'O': 0, 'B-geo': 1, 'B-gpe': 2, 'B-per': 3, 'I-geo': 4, 'B-org': 5, 'I-org': 6,
'B-tim': 7, 'B-art': 8, 'I-art': 9, 'I-per': 10, 'I-gpe': 11, 'I-tim': 12,
'B-nat': 13, 'B-eve': 14, 'I-eve': 15, 'I-nat': 16}
```

If you had the sentence

"Sharon flew to Miami on Friday"

The tags would look like:

```
Sharon B-per
flew   O
to     O
Miami  B-geo
on     O
Friday B-tim
```

where you would have three tokens beginning with B-, since there are no multi-token entities in the sequence. But if you added Sharon's last name to the sentence:

"Sharon Floyd flew to Miami on Friday"

```
Sharon B-per
Floyd  I-per
flew   O
to     O
Miami  B-geo
on     O
Friday B-tim
```

your tags would change to show first "Sharon" as B-per, and "Floyd" as I-per, where I- indicates an inner token in a multi-token sequence.

[6]:
```
# Exploring information about the data
print('The number of outputs is tag_map', len(tag_map))
# The number of vocabulary tokens (including <PAD>)
g_vocab_size = len(vocab)
print(f"Num of vocabulary words: {g_vocab_size}")
print('The training size is', t_size)
print('The validation size is', v_size)
print('An example of the first sentence is', t_sentences[0])
print('An example of its corresponding label is', t_labels[0])
```

```
The number of outputs is tag_map 17
Num of vocabulary words: 35181
The training size is 33570
The validation size is 7194
An example of the first sentence is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 9, 15, 1, 16, 17, 18, 19, 20, 21]
An example of its corresponding label is [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0]
```

So you can see that we have already encoded each sentence into a tensor by converting it into a number. We also have 16 possible tags (excluding the '0' tag), as shown in the tag map.

1.2 Data generator

In python, a generator is a function that behaves like an iterator. It returns the next item in a pre-defined sequence. Here is a link to review python generators.

In many AI applications it is very useful to have a data generator. You will now implement a data generator for our NER application.

Exercise 01

Instructions: Implement a data generator function that takes in `batch_size, x, y, pad, shuffle` where x is a large list of sentences, and y is a list of the tags associated with those sentences and pad is a pad value. Return a subset of those inputs in a tuple of two arrays `(X,Y)`.

`X` and `Y` are arrays of dimension (`batch_size, max_len`), where `max_len` is the length of the longest sentence *in that batch*. You will pad the `X` and `Y` examples with the pad argument. If `shuffle=True`, the data will be traversed in a random order.

Details:

Use this code as an outer loop

```
while True:
...
yield((X,Y))
```

so your data generator runs continuously. Within that loop, define two `for` loops:

1. The first stores temporal lists of the data samples to be included in the batch, and finds the maximum length of the sentences contained in it.
2. The second one moves the elements from the temporal list into NumPy arrays pre-filled with pad values.

There are three features useful for defining this generator:

1. The NumPy `full` function to fill the NumPy arrays with a pad value. See full function documentation.
2. Tracking the current location in the incoming lists of sentences. Generators variables hold their values between invocations, so we create an `index` variable, initialize to zero, and increment by one for each sample included in a batch. However, we do not use the `index` to access the positions of the list of sentences directly. Instead, we use it to select one index

from a list of indexes. In this way, we can change the order in which we traverse our original list, keeping untouched our original list.

3. Since `batch_size` and the length of the input lists are not aligned, gathering a batch_size group of inputs may involve wrapping back to the beginning of the input loop. In our approach, it is just enough to reset the `index` to 0. We can re-shuffle the list of indexes to produce different batches each time.

[7]:
```
# UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: data_generator
def data_generator(batch_size, x, y, pad, shuffle=False, verbose=False):
    '''
      Input:
        batch_size - integer describing the batch size
        x - list containing sentences where words are represented as integers
        y - list containing tags associated with the sentences
        shuffle - Shuffle the data order
        pad - an integer representing a pad character
        verbose - Print information during runtime
      Output:
        a tuple containing 2 elements:
        X - np.ndarray of dim (batch_size, max_len) of padded sentences
        Y - np.ndarray of dim (batch_size, max_len) of tags associated with the␣
↪sentences in X
    '''

    # count the number of lines in data_lines
    num_lines = len(x)

    # create an array with the indexes of data_lines that can be shuffled
    lines_index = [*range(num_lines)]

    # shuffle the indexes if shuffle is set to True
    if shuffle:
        rnd.shuffle(lines_index)

    index = 0 # tracks current location in x, y
    while True:
        buffer_x = [0] * batch_size # Temporal array to store the raw x data␣
↪for this batch
        buffer_y = [0] * batch_size # Temporal array to store the raw y data␣
↪for this batch

  ### START CODE HERE (Replace instances of 'None' with your code) ###

        # Copy into the temporal buffers the sentences in x[index]
        # along with their corresponding labels y[index]
        # Find maximum length of sentences in x[index] for this batch.
```

```
        # Reset the index if we reach the end of the data set, and shuffle the␣
↪indexes if needed.
        max_len = 0
        for i in range(batch_size):
             # if the index is greater than or equal to the number of lines in x
            if index >= num_lines:
                # then reset the index to 0
                index = 0
                # re-shuffle the indexes if shuffle is set to True
                if shuffle:
                    rnd.shuffle(lines_index)

            # The current position is obtained using `lines_index[index]`
            # Store the x value at the current position into the buffer_x
            buffer_x[i] = x[lines_index[index]]

            # Store the y value at the current position into the buffer_y
            buffer_y[i] = y[lines_index[index]]

            lenx = len(buffer_x[i]) #length of current x[]
            if lenx > max_len:
                max_len = lenx #max_len tracks longest x[]

            # increment index by one
            index += 1


        # create X,Y, NumPy arrays of size (batch_size, max_len) 'full' of pad␣
↪value
        X = np.full((batch_size, max_len), pad)
        Y = np.full((batch_size, max_len), pad)

        # copy values from lists to NumPy arrays. Use the buffered values
        for i in range(batch_size):
            # get the example (sentence as a tensor)
            # in `buffer_x` at the `i` index
            x_i = buffer_x[i]

            # similarly, get the example's labels
            # in `buffer_y` at the `i` index
            y_i = buffer_y[i]

            # Walk through each word in x_i
            for j in range(len(x_i)):
                # store the word in x_i at position j into X
                X[i, j] =  x_i[j]
```

```
                # store the label in y_i at position j into Y
                Y[i, j] = y_i[j]

    ### END CODE HERE ###
        if verbose: print("index=", index)
        yield((X,Y))
```

[8]:
```
batch_size = 5
mini_sentences = t_sentences[0: 8]
mini_labels = t_labels[0: 8]
dg = data_generator(batch_size, mini_sentences, mini_labels, vocab["<PAD>"],␣
 ↪shuffle=False, verbose=True)
X1, Y1 = next(dg)
X2, Y2 = next(dg)
print(Y1.shape, X1.shape, Y2.shape, X2.shape)
print(X1[0][:], "\n", Y1[0][:])
```

```
index= 5
index= 2
(5, 30) (5, 30) (5, 30) (5, 30)
[    0     1     2     3     4     5     6     7     8     9    10    11
    12    13    14     9    15     1    16    17    18    19    20    21
 35180 35180 35180 35180 35180 35180]
 [    0     0     0     0     0     0     1     0     0     0     0     0
     1     0     0     0     0     0     2     0     0     0     0     0
 35180 35180 35180 35180 35180 35180]
```

Expected output:

```
index= 5
index= 2
(5, 30) (5, 30) (5, 30) (5, 30)
[    0     1     2     3     4     5     6     7     8     9    10    11
    12    13    14     9    15     1    16    17    18    19    20    21
 35180 35180 35180 35180 35180 35180]
 [    0     0     0     0     0     0     1     0     0     0     0     0
     1     0     0     0     0     0     2     0     0     0     0     0
 35180 35180 35180 35180 35180 35180]
```

[9]:
```
# Test your function
w3_unittest.test_data_generator(data_generator)
```

```
 All tests passed
```

Part 2: Building the model

You will now implement the model that will be able to determining the tags of sentences like the following:

The model architecture will be as follows:

Concretely, your inputs will be sentences represented as tensors that are fed to a model with:

- An Embedding layer,
- A LSTM layer
- A Dense layer
- A log softmax layer.

Good news! We won't make you implement the LSTM cell drawn above. You will be in charge of the overall architecture of the model.

Exercise 02

Instructions: Implement the initialization step and the forward function of your Named Entity Recognition system.
Please utilize help function e.g. `help(tl.Dense)` for more information on a layer

- tl.Serial: Combinator that applies layers serially (by function composition).
  - You can pass in the layers as arguments to `Serial`, separated by commas.
  - For example: `tl.Serial(tl.Embeddings(...), tl.Mean(...), tl.Dense(...), tl.LogSoftmax(...))`
- tl.Embedding: Initializes the embedding. In this case it is the dimension of the model by the size of the vocabulary.
  - `tl.Embedding(vocab_size, d_feature)`.
  - `vocab_size` is the number of unique words in the given vocabulary.
  - `d_feature` is the number of elements in the word embedding (some choices for a word embedding size range from 150 to 300, for example).
- tl.LSTM:`Trax` LSTM layer.
  - `LSTM(n_units)` Builds an LSTM layer with hidden state and cell sizes equal to `n_units`. In trax, `n_units` should be equal to the size of the embeddings `d_feature`.
- tl.Dense: A dense layer.
  - `tl.Dense(n_units)`: The parameter `n_units` is the number of units chosen for this dense layer.
- tl.LogSoftmax: Log of the output probabilities.
  - Here, you don't need to set any parameters for `LogSoftMax()`.

Online documentation

- tl.Serial
- tl.Embedding
- tl.LSTM
- tl.Dense
- tl.LogSoftmax

[10]:
```
# UNIT TEST COMMENT: Candidate for table-driven test
# The best way to eval the correctness is using the string representation of␣
 ↪the model.
# Just as in the expected output cell.

# UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
```

```
# GRADED FUNCTION: NER
def NER(tags, vocab_size=35181, d_model=50):
    '''
      Input:
        tag_map - dictionary that maps the tags to numbers
        vocab_size - integer containing the size of the vocabulary
        d_model - integer describing the embedding size
      Output:
        model - a trax serial model
    '''
    ### START CODE HERE (Replace instances of 'None' with your code) ###
    model = tl.Serial(
      tl.Embedding(vocab_size, d_model), # Embedding layer
      tl.LSTM(n_units=d_model), # LSTM layer
      tl.Dense(n_units=len(tags)), # Dense layer with len(tags) units
      tl.LogSoftmax() # LogSoftmax layer
      )
      ### END CODE HERE ###
    return model
```

[11]:
```
# initializing your model
model = NER(tag_map)
# display your model
print(model)
```

```
Serial[
  Embedding_35181_50
  LSTM_50
  Dense_17
  LogSoftmax
]
```

Expected output:

```
Serial[
  Embedding_35181_50
  LSTM_50
  Dense_17
  LogSoftmax
]
```

[12]:
```
# Test your function
w3_unittest.test_NER(NER)
```

```
 All tests passed
```

Part 3: Train the Model

This section will train your model.

Before you start, you need to create the data generators for training and validation data. It is important that you mask padding in the loss weights of your data, which can be done using the `id_to_mask` argument of `trax.data.inputs.add_loss_weights`.

[13]:
```
# Setting random seed for reproducibility and testing
rnd.seed(33)

batch_size = 64

# Create training data, mask pad id=35180 for training.
train_generator = trax.data.inputs.add_loss_weights(
    data_generator(batch_size, t_sentences, t_labels, vocab['<PAD>'], True),
    id_to_mask=vocab['<PAD>'])

# Create validation data, mask pad id=35180 for training.
eval_generator = trax.data.inputs.add_loss_weights(
    data_generator(batch_size, v_sentences, v_labels, vocab['<PAD>'], True),
    id_to_mask=vocab['<PAD>'])
```

3.1 Training the model

You will now write a function that takes in your model and trains it.

As you've seen in the previous assignments, you will first create the TrainTask and EvalTask using your data generator. Then you will use the `training.Loop` to train your model.

Exercise 03

Instructions: Implement the `train_model` program below to train the neural network above. Here is a list of things you should do: - Create the trainer object by calling `trax.supervised.training.Loop` and pass in the following:

```
- model = [NER](#ex02)
- [training task](https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#trax.supervise
    - loss_layer = [tl.CrossEntropyLoss()](https://github.com/google/trax/blob/e65d51fe584b10c
    - optimizer = [trax.optimizers.Adam(0.01)](https://github.com/google/trax/blob/e65d51fe584
- [evaluation task](https://trax-ml.readthedocs.io/en/latest/trax.supervised.html#trax.supervi
    - metrics for `EvalTask`: `tl.CrossEntropyLoss()` and `tl.Accuracy()`
    - in `EvalTask` set `n_eval_batches=10` for better evaluation accuracy
- output_dir = output_dir
```

You'll be using a cross entropy loss, with an Adam optimizer. Please read the trax documentation to get a full understanding. The trax GitHub also contains some useful information and a link to a colab notebook.

[14]:
```
# CODE REVIEW COMMENT: Unit test proposed for correctness

# UNQ_C3 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: train_model
def train_model(NER, train_generator, eval_generator, train_steps=1,␣
 ↪output_dir='model'):
```

```
    '''
    Input:
        NER - the model you are building
        train_generator - The data generator for training examples
        eval_generator - The data generator for validation examples,
        train_steps - number of training steps
        output_dir - folder to save your model
    Output:
        training_loop - a trax supervised training Loop
    '''
### START CODE HERE (Replace instances of 'None' with your code) ###
    train_task = training.TrainTask(
      train_generator, # A train data generator
      loss_layer = tl.CrossEntropyLoss(), # A cross-entropy loss function
      optimizer = trax.optimizers.Adam(0.01) # The adam optimizer
    )

    eval_task = training.EvalTask(
      labeled_data = eval_generator,  # A labeled data generator
      metrics = [tl.CrossEntropyLoss(), tl.Accuracy()], # Evaluate with␣
 ↪cross-entropy loss and accuracy
      n_eval_batches = 10, # Number of batches to use on each evaluation
    )

    training_loop = training.Loop(
        NER, # A model to train
        train_task, # A train task
        eval_tasks = [eval_task], # The evaluation task
        output_dir = output_dir # The output directory
    )

    # Train with train_steps
    training_loop.run(n_steps = train_steps)
### END CODE HERE ###
    return training_loop
```

[15]:
```
# Test your function
w3_unittest.test_train_model(train_model, NER(tag_map, vocab_size=35181,␣
 ↪d_model=50), data_generator)
```

```
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0
and rerun for more info.)

 All tests passed
```

On your local machine, you can run this training for 1000 train_steps and get your own model. This training takes about 5 to 10 minutes to run.

[16]:
```
train_steps = 100             # In coursera we can only train 100 steps
!rm -f 'model/model.pkl.gz'  # Remove old model.pkl if it exists

# Train the model
training_loop = train_model(NER(tag_map), train_generator, eval_generator,␣
 ↪train_steps)
```

```
Step      1: Total number of trainable weights: 1780117
Step      1: Ran 1 train steps in 2.41 secs
Step      1: train CrossEntropyLoss |  2.26452208
Step      1: eval  CrossEntropyLoss |  1.50368143
Step      1: eval           Accuracy |  0.81126399

Step    100: Ran 99 train steps in 52.13 secs
Step    100: train CrossEntropyLoss |  0.49678212
Step    100: eval  CrossEntropyLoss |  0.23554303
Step    100: eval           Accuracy |  0.93673450
```

Expected output (Approximately)

```
...
Step      1: Total number of trainable weights: 1780117
Step      1: Ran 1 train steps in 2.63 secs
Step      1: train CrossEntropyLoss |  4.49356890
Step      1: eval  CrossEntropyLoss |  3.41925483
Step      1: eval           Accuracy |  0.01685534

Step    100: Ran 99 train steps in 49.14 secs
Step    100: train CrossEntropyLoss |  0.61710459
Step    100: eval  CrossEntropyLoss |  0.27959008
Step    100: eval           Accuracy |  0.93171992
...
```

This value may change between executions, but it must be around 90% of accuracy on train and validations sets, after 100 training steps.

We have trained the model longer, and we give you such a trained model. In that way, we ensure you can continue with the rest of the assignment even if you had some troubles up to here, and also we are sure that everybody will get the same outputs for the last example. However, you are free to try your model, as well.

[17]:
```
# loading in a pretrained model..
model = NER(tag_map)
model.init(trax.shapes.ShapeDtype((1, 1), dtype=np.int32))

# Load the pretrained model
model.init_from_file('model.pkl.gz', weights_only=True)
```

```
[17]: ((array([[ 0.81504697,  0.41156578, -0.39470178, …,  1.8432791 ,
                 0.20336688, -1.5193548 ],
               [-0.8170589 , -0.6749423 ,  0.7024649 , …,  0.765976  ,
                -0.32910004,  0.3397966 ],
               [ 1.8445145 , -1.1847575 ,  0.44616678, …, -1.3649443 ,
                -0.26146498, -1.0452443 ],
               …,
               [-0.47112465,  0.39385355,  0.8544945 , …,  0.8979616 ,
                -0.4146749 ,  0.46532995],
               [ 0.06438114,  0.40173107, -0.81699884, …, -0.26729876,
                 0.06683028, -1.8318921 ],
               [ 1.490085  ,  0.25960454,  0.57180226, …, -0.8960343 ,
                 1.4659392 ,  0.51151174]], dtype=float32),
        ((), (), ())),
       ((array([[ 1.8205246e-02,  4.5587063e-02,  2.3308586e-01, …,
                  3.2658550e-01,  4.4989395e-01,  1.3649037e-01],
                [ 3.4135941e-02,  3.6623224e-04, -2.0645858e-01, …,
                 -3.7373897e-02,  1.3227654e-01,  3.6544618e-01],
                [ 2.4287397e-01, -4.1946974e-02,  2.5109309e-01, …,
                  3.0652905e-01,  1.8023208e-01, -4.4977060e-01],
                …,
                [-3.8651712e-02, -1.4279284e-01,  7.6946914e-02, …,
                  4.3700095e-02, -6.8984614e-03,  2.3343280e-02],
                [ 4.4559538e-03, -4.5776283e-03, -2.5545111e-01, …,
                 -1.9839586e-01, -2.2730356e-01,  2.8106436e-02],
                [ 8.7965906e-02,  1.5161000e-01, -1.3848971e-02, …,
                  1.6166535e-01,  2.1183728e-01, -6.4703353e-02]], dtype=float32),
         array([1.6854244 , 1.2979273 , 1.0111951 , 0.83587646, 2.2717557 ,
                1.174464  , 1.2097938 , 0.8333579 , 1.5129769 , 1.2865868 ,
                1.0122222 , 0.7420179 , 1.4616095 , 1.3716865 , 1.183128  ,
                2.0540466 , 0.7297637 , 1.7542948 , 1.4260938 , 1.0854443 ,
                0.86481327, 1.7100143 , 0.95269287, 1.4605767 , 1.8531597 ,
                1.4426551 , 0.96249163, 1.1918762 , 1.5750595 , 2.8395677 ,
                0.6104171 , 1.7534453 , 1.4870973 , 0.9973332 , 1.6693677 ,
                1.5615942 , 0.9934437 , 1.4966784 , 1.3405765 , 0.9236135 ,
                0.7409323 , 1.3020254 , 1.7708799 , 1.0030404 , 0.80193156,
                1.0716376 , 1.493366  , 2.1602647 , 0.8422125 , 1.6722763 ,
                1.4617225 , 1.1182904 , 0.7711745 , 0.86279124, 2.00725   ,
                1.2961555 , 1.1192496 , 1.2368942 , 1.696922  , 1.1817851 ,
                0.9268449 , 1.0085806 , 1.4859127 , 1.4285009 , 1.112943  ,
                1.739962  , 1.0019778 , 1.8289869 , 1.5200068 , 1.0460978 ,
                1.2855432 , 1.1556262 , 0.5179452 , 1.4512749 , 2.0955215 ,
                1.2157683 , 0.8909916 , 1.1306164 , 1.4892405 , 2.0666041 ,
                0.51464105, 1.6188123 , 1.3959543 , 1.3240963 , 1.398886  ,
                1.7381012 , 1.4586241 , 1.4561127 , 1.2323151 , 0.76638013,
                1.1020749 , 1.3235599 , 1.364442  , 1.2431955 , 0.8876695 ,
                0.7320451 , 1.2007953 , 1.5502919 , 0.99877405, 1.2763859 ,
```

```
          1.0426939 , 1.0525128 , 0.8009765 , 0.9399874 , 0.9907789 ,
          0.83222485, 1.0317605 , 0.91201544, 1.1327735 , 1.0524918 ,
          1.0372494 , 0.88949317, 1.0550225 , 1.1370779 , 0.7050474 ,
          1.2664902 , 0.93727463, 0.90847397, 0.95251304, 0.7721647 ,
          0.8260327 , 0.8332168 , 0.97942346, 1.0223993 , 1.0006895 ,
          1.060646  , 0.9042888 , 1.0394853 , 0.9925131 , 0.38989544,
          0.7230699 , 1.0208429 , 1.062438  , 1.0589039 , 1.0775026 ,
          1.0874754 , 1.1661438 , 0.9866324 , 1.0852152 , 0.9155094 ,
          1.0852442 , 0.9840062 , 0.75459987, 0.99201745, 0.9202928 ,
          0.7419778 , 1.0831951 , 1.0849274 , 0.8571488 , 0.9694658 ,
          1.194375  , 1.0539739 , 0.9547721 , 0.9202652 , 1.0917386 ,
          1.1022308 , 1.0888128 , 0.7548035 , 1.0795542 , 1.1811827 ,
          1.353696  , 0.90758467, 1.3962934 , 1.3079231 , 1.0871059 ,
          1.2455839 , 0.99197876, 0.9041552 , 1.0928652 , 1.0660387 ,
          0.94059443, 1.4588033 , 1.0957755 , 1.0757209 , 1.267846  ,
          1.1812643 , 0.94670117, 0.90446424, 1.0446624 , 2.7972755 ,
          0.91194886, 1.2462132 , 0.9716122 , 1.2372544 , 1.2981352 ,
          1.0619044 , 1.1879133 , 1.0467446 , 1.0551138 , 1.0373522 ,
          1.1695648 , 1.2904806 , 1.5025221 , 1.0758194 , 0.88997847,
          0.8845719 , 1.2349951 , 1.332449  , 1.0055327 , 1.2380725 ],
         dtype=float32)),),
 ()),
(array([[ 5.22492409e-01, -3.00267011e-01, -5.76043725e-01,
          5.18762350e-01, -3.99674535e-01,  1.64419889e-01,
         -3.61355692e-01, -1.72977015e-01, -8.64919871e-02,
         -1.06282525e-01, -3.91271472e-01, -4.38841224e-01,
         -1.28947124e-01, -8.25695843e-02, -4.35167342e-01,
         -1.68992266e-01, -1.52835160e-01],
        [ 4.04112905e-01, -1.67354211e-01, -2.51709402e-01,
         -1.43934548e-01,  2.43671998e-01,  4.71022487e-01,
         -1.73283696e-01,  2.03600943e-01,  4.25134748e-02,
         -8.73562768e-02, -4.97560620e-01, -2.58896202e-01,
          2.58559108e-01,  1.86556876e-01, -1.71378508e-01,
         -4.31724578e-01, -4.34081614e-01],
        [ 5.52614443e-02,  1.75628662e-01, -7.04351723e-01,
         -2.49994487e-01, -6.04376495e-01,  2.11223602e-01,
         -3.01960737e-01, -7.28549838e-01, -6.53434098e-02,
         -4.08937812e-01, -3.17913890e-01,  8.83366074e-03,
         -9.52698886e-02,  1.47018403e-01, -6.79332167e-02,
          3.54560018e-01,  2.11631730e-01],
        [-3.59147489e-02,  1.09010863e+00, -4.23903733e-01,
         -1.64979771e-01,  4.46146168e-02,  3.71380627e-01,
          3.77887279e-01, -2.62679487e-01, -2.77810395e-01,
         -5.90043589e-02,  2.83577114e-01,  7.52747478e-03,
         -1.87515095e-01, -5.66075921e-01, -3.26092213e-01,
          1.72540739e-01, -1.50614560e-01],
        [ 4.22573268e-01, -1.27358094e-01,  8.93090889e-02,
```

```
  4.05784428e-01, -5.10686934e-02, -4.64693904e-01,
  9.22672227e-02,  1.39183030e-01,  9.20567214e-02,
  2.52354592e-02, -7.87983537e-02, -3.28912959e-02,
 -2.01635107e-01, -4.41590816e-01, -3.99357140e-01,
 -5.43071628e-01,  3.45393308e-02],
[-2.21701145e-01,  1.05201030e+00,  3.19466263e-01,
  6.55409217e-01,  7.94601262e-01,  4.01203394e-01,
 -2.41930217e-01, -1.42703581e+00, -1.28299668e-01,
 -2.94186532e-01, -3.08514386e-01, -1.87155381e-01,
 -7.16864049e-01, -4.50788319e-01,  9.09690857e-02,
  8.71396512e-02, -3.75783056e-01],
[-3.94670628e-02, -6.68459356e-01,  5.44112682e-01,
 -2.37003058e-01, -7.75517046e-01, -3.03820342e-01,
  3.64599198e-01,  1.69453681e-01, -3.42120290e-01,
 -2.73573607e-01,  3.32077332e-02, -4.04292196e-01,
  2.61205554e-01, -3.58233213e-01, -7.17273206e-02,
 -1.00011356e-01, -1.35751124e-02],
[-1.26873897e-02,  6.28863692e-01,  4.45303433e-02,
 -6.63247883e-01, -5.81941545e-01,  6.78069174e-01,
  1.81016892e-01,  6.96243405e-01,  4.67899404e-02,
 -3.33413273e-01, -6.67748451e-01, -1.22301884e-01,
 -1.35667056e-01, -3.53891730e-01, -6.11209981e-02,
 -5.01504362e-01, -2.33926222e-01],
[ 2.86403835e-01, -1.24878570e-01,  2.58399136e-02,
 -5.88641942e-01, -5.48453555e-02, -1.91024333e-01,
 -3.46662372e-01, -6.81052446e-01, -3.42517942e-01,
 -1.60946369e-01,  5.04518926e-01,  3.82423238e-03,
  9.65177491e-02, -2.29205593e-01, -1.63260683e-01,
  1.73317656e-01, -4.02356744e-01],
[ 4.32505935e-01,  7.19232634e-02, -5.51506817e-01,
 -2.33325660e-01, -8.26672018e-02,  5.25125861e-01,
 -2.96465605e-01, -1.77367941e-01, -7.36130029e-02,
 -4.07649636e-01, -1.39186457e-01, -3.82155031e-01,
  4.94351611e-03,  2.20503062e-01, -2.77898759e-01,
 -1.52841955e-02, -4.33646590e-01],
[ 1.76988125e-01,  4.47825193e-01,  2.44967893e-01,
 -5.62304497e-01, -1.14976853e-01,  8.75651464e-02,
 -4.58624363e-02, -4.26192880e-01,  1.93301052e-01,
 -6.91710338e-02, -7.87903190e-01,  8.99584405e-03,
  6.64672107e-02, -4.76139337e-02,  3.92017603e-01,
  2.26357162e-01, -3.01785171e-01],
[-1.19927853e-01,  7.97016978e-01,  6.32443130e-01,
  5.61205387e-01, -5.99712670e-01, -1.92995548e-01,
 -5.06583691e-01, -6.36727691e-01, -1.22797422e-01,
  2.56529953e-02,  5.24826825e-01, -7.42853247e-03,
 -6.32434711e-02,  2.89041668e-01, -1.51149228e-01,
 -5.36483586e-01,  1.18176110e-01],
```

```
[ 4.57709074e-01, -2.45516673e-01, -7.03083992e-01,
  3.33185881e-01, -6.23180389e-01,  2.82489032e-01,
 -3.70383203e-01,  2.25939706e-01, -1.43565089e-01,
  1.67658851e-01,  8.68608207e-02, -1.21780619e-01,
  1.81653365e-01, -7.95933381e-02, -1.31369397e-01,
  1.36302680e-01, -2.53608823e-01],
[ 2.91132987e-01,  2.15994287e-02,  2.77102590e-01,
 -2.24864021e-01,  5.28830588e-01, -3.65532964e-01,
  3.55308592e-01, -2.67890215e-01, -2.47928411e-01,
  4.65085804e-02,  3.00757349e-01, -1.93124145e-01,
 -3.94805998e-01, -1.19426385e-01, -5.63039899e-01,
 -4.99142528e-01, -3.94075811e-01],
[-2.23453805e-01,  1.06039202e+00,  4.82775420e-01,
  5.67017198e-01, -8.55074763e-01,  6.61626220e-01,
 -6.88900888e-01,  6.55609131e-01, -2.31606901e-01,
 -4.72241521e-01,  2.30029985e-01,  5.07244840e-02,
 -3.97857964e-01, -3.21961761e-01, -7.28645563e-01,
 -5.97576618e-01, -2.85780191e-01],
[ 2.28473783e-01,  2.14988634e-01, -6.33245468e-01,
 -2.97629714e-01, -4.28691089e-01, -1.28603712e-01,
 -8.88055190e-02, -3.50240380e-01, -3.86497647e-01,
  1.21420436e-01,  5.07317781e-01,  4.42964733e-02,
  4.24934477e-01, -2.25339666e-01, -5.02384603e-01,
 -3.71262610e-01, -4.98380035e-01],
[-1.63695082e-01, -5.88303030e-01, -1.34929731e-01,
  6.93596601e-01, -3.95576358e-01,  4.76714909e-01,
  4.20461148e-02,  2.21985132e-01, -2.69573987e-01,
 -3.66358876e-01,  3.41131598e-01,  7.51820058e-02,
 -1.58592671e-01, -1.91948965e-01, -2.17588484e-01,
 -8.48015696e-02,  1.90072488e-02],
[ 1.02034263e-01,  2.79471040e-01,  4.97352272e-01,
 -2.43766621e-01, -5.60707390e-01,  2.63408780e-01,
 -5.91222286e-01, -2.88201898e-01, -2.82364190e-01,
 -3.60000461e-01, -4.40491796e-01, -2.59812236e-01,
 -5.25629282e-01, -1.16754659e-01,  7.34341741e-02,
 -5.84913492e-01, -1.80946719e-02],
[ 3.67322057e-01, -5.77189028e-01, -4.23639081e-02,
 -6.32877126e-02, -7.42029101e-02, -4.37420666e-01,
 -2.73921698e-01,  1.96201295e-01, -4.18877602e-01,
 -1.31240711e-01, -2.64525414e-01, -3.67617488e-01,
  4.76701051e-01, -3.19049627e-01, -2.48019516e-01,
 -1.11414842e-01, -2.14790419e-01],
[-1.29754096e-01,  2.48156134e-02, -1.48425445e-01,
  8.88699591e-01, -4.14236307e-01,  7.95713484e-01,
  7.85508454e-01, -8.26169431e-01, -3.73806097e-02,
 -1.25360399e-01,  9.11820233e-02,  4.29998040e-02,
  2.59737194e-01, -5.50461948e-01, -2.85380721e-01,
```

```
 -6.62189066e-01, -3.21492523e-01],
[-2.62812227e-01, -7.64828205e-01,  2.96108484e-01,
  6.43610954e-01, -9.54106376e-02, -9.53767121e-01,
 -3.02402675e-01,  4.00129169e-01, -3.86336535e-01,
 -1.66053250e-02,  3.58739883e-01,  6.12522149e-03,
  2.89238691e-01, -3.26216936e-01, -1.56232119e-01,
  3.83345261e-02, -2.34865382e-01],
[ 2.54942626e-01, -4.82052058e-01, -7.32252479e-01,
  4.74952646e-02, -5.61625183e-01, -5.20331860e-01,
 -2.24171385e-01,  4.17553186e-02,  1.70720324e-01,
 -3.45877595e-02, -9.01532471e-01, -2.00972378e-01,
  2.87372530e-01,  1.68675065e-01,  2.81985968e-01,
  1.08927846e-01, -1.56412438e-01],
[ 6.87182993e-02, -5.24033308e-01, -4.86984462e-01,
 -6.19755566e-01, -6.03793383e-01, -5.91538727e-01,
 -2.93026537e-01, -1.14599772e-01, -3.74767989e-01,
 -9.95257683e-03, -4.97348040e-01, -5.62906146e-01,
  8.75871360e-01, -4.41906989e-01, -4.59717900e-01,
 -1.90668285e-01, -2.51278251e-01],
[ 2.83860415e-01, -6.21597767e-01,  3.60506624e-01,
 -1.33115783e-01, -2.20012460e-02, -1.41365483e-01,
 -1.33905903e-01, -4.77693319e-01, -2.48354778e-01,
 -1.81036219e-01, -5.14351666e-01, -1.18651643e-01,
  1.70918792e-01, -1.33117363e-01, -1.15978524e-01,
 -3.00302684e-01, -3.32199812e-01],
[-2.63424993e-01,  4.40211475e-01,  3.91667366e-01,
  6.69456720e-01, -8.90326723e-02,  7.38337994e-01,
  2.89319992e-01, -7.13289440e-01,  1.29570678e-01,
 -3.32683802e-01,  3.51401448e-01, -3.54277313e-01,
 -6.47332251e-01, -1.37902081e-01,  7.81771019e-02,
 -4.59978938e-01, -3.43254894e-01],
[ 4.38510835e-01,  3.23934615e-01, -2.41020173e-01,
 -1.90129876e-01, -2.48373598e-01, -4.99090701e-02,
  9.83321518e-02, -3.23142290e-01, -3.04442763e-01,
 -1.13189556e-01, -1.59954906e-01, -2.11109340e-01,
 -2.68609703e-01, -5.04755020e-01,  2.63985068e-01,
  7.55190328e-02, -2.76284873e-01],
[-1.09361887e-01, -6.47492409e-01, -1.52858689e-01,
 -5.96112072e-01,  1.52102530e-01, -4.06115443e-01,
 -1.22687712e-01,  6.07140400e-02, -3.67303878e-01,
  3.36758196e-02, -6.23483062e-02,  3.59757662e-01,
 -3.32621008e-01, -9.32668567e-01,  9.95486826e-02,
  3.56584340e-01, -5.92220910e-02],
[ 3.31617087e-01, -5.03506124e-01,  2.61460215e-01,
 -1.77441731e-01, -6.55950487e-01, -1.40989184e-01,
 -5.33709191e-02, -3.44653875e-01,  1.91584319e-01,
 -6.53832406e-02, -6.13427758e-01, -2.21090987e-01,
```

```
 -2.45919347e-01, -1.95010722e-01, -2.77357161e-01,
 -3.02460015e-01, -4.70238894e-01],
[ 5.31833112e-01, -4.71094847e-01, -5.75433016e-01,
 -1.62334353e-01,  2.38227900e-02, -3.40687007e-01,
 -2.04606399e-01,  1.17438436e-01, -3.17493588e-01,
 -6.09255284e-02, -1.90586507e-01, -3.50243121e-01,
 -3.62981498e-01, -1.62433341e-01, -7.38168508e-02,
 -3.40081096e-01, -3.73968408e-02],
[ 5.40199101e-01,  5.99464953e-01,  2.64185220e-01,
  1.11854923e+00, -1.07264268e+00,  8.15554261e-01,
 -1.35643494e+00,  8.43949199e-01, -1.88045818e-02,
 -3.33660156e-01, -1.16771579e+00, -2.97938138e-01,
 -7.29957819e-01, -2.28346884e-01, -1.58297598e-01,
 -8.07544291e-01, -4.59286124e-01],
[ 3.05221751e-02, -3.53594273e-01, -2.41475329e-01,
 -3.41951340e-01, -9.85349596e-01,  7.05481887e-01,
 -1.13184273e-01, -7.16636360e-01,  9.91140828e-02,
 -1.89452142e-01, -9.07258019e-02,  1.08113259e-01,
 -3.24634790e-01,  5.54517098e-02, -1.41534880e-01,
  2.37111524e-01, -1.64440528e-01],
[ 3.62177491e-01,  2.16381714e-01,  3.70851994e-01,
  4.62692618e-01, -3.33623588e-01,  5.51848054e-01,
  5.78205250e-02, -5.25472641e-01,  2.04113826e-01,
 -1.99815780e-01, -2.74214923e-01, -4.07270014e-01,
 -2.48084068e-01, -4.16319549e-01, -3.32400531e-01,
 -5.12472868e-01, -4.17373240e-01],
[ 3.67569417e-01, -2.63302237e-01, -7.53237084e-02,
 -3.81765366e-01, -4.95616764e-01, -3.10622662e-01,
 -4.41325642e-02,  8.61776248e-02, -2.55106360e-01,
 -2.12387219e-01,  3.16616952e-01,  2.34824866e-02,
  1.32826909e-01, -6.44535422e-01, -3.18675429e-01,
 -6.68750405e-01, -1.29539510e-02],
[ 1.01901963e-01, -5.67999221e-02, -2.19238356e-01,
  9.71220374e-01, -9.46732700e-01,  7.30410695e-01,
  8.46439719e-01, -8.25150907e-01, -1.10785067e-01,
 -2.15972647e-01,  5.91678560e-01, -2.55152315e-01,
 -6.53665900e-01, -2.90349305e-01,  5.40182889e-02,
 -6.16701059e-02, -6.39539808e-02],
[ 4.19817120e-01, -5.16562238e-02, -5.17634332e-01,
  3.89064774e-02,  1.17138639e-01, -2.68300891e-01,
 -1.94658622e-01,  3.42325680e-02, -3.73620123e-01,
 -8.29097852e-02,  2.67258018e-01, -1.23883054e-01,
 -3.69409949e-01, -3.62399876e-01, -6.79629087e-01,
  1.45526290e-01, -1.08446449e-01],
[ 4.48970526e-01,  3.21831182e-02, -1.11019634e-01,
 -1.11492701e-01, -8.54404271e-02,  3.40578228e-01,
  8.03871304e-02,  5.26210427e-01, -3.28268081e-01,
```

```
  6.52397573e-02,  4.50341642e-01,  5.14808968e-02,
 -6.00921333e-01, -3.35734516e-01, -2.66357690e-01,
 -3.08603436e-01, -1.13995962e-01],
[ 1.09846825e-02,  3.56205478e-02,  1.76665619e-01,
 -5.03098547e-01,  3.28140318e-01,  4.42327231e-01,
  2.03654036e-01,  3.48726884e-02, -7.60957226e-02,
 -9.89460722e-02, -2.82514811e-01, -3.48744988e-01,
  2.80020852e-02, -7.88114607e-01, -3.52411598e-01,
 -4.70479995e-01, -5.92435658e-01],
[-6.22582884e-05, -2.40957081e-01, -3.05581689e-01,
  7.37112880e-01, -6.26345754e-01,  6.43571675e-01,
  1.12985708e-01,  4.57808197e-01, -1.66287459e-02,
 -2.88720906e-01, -4.21509832e-01, -4.08095300e-01,
  3.01908910e-01,  9.54955891e-02, -2.78529674e-01,
 -4.78266060e-01, -4.76607978e-01],
[ 5.24540365e-01, -5.02445400e-02, -3.52061003e-01,
 -4.43094224e-01, -3.01498264e-01, -6.29142284e-01,
  3.07413161e-01, -3.63631785e-01, -4.39681917e-01,
 -3.99245590e-01,  8.77478253e-03,  5.01419455e-02,
  2.55710501e-02, -7.00878143e-01, -1.09079190e-01,
 -1.03994653e-01, -8.10173154e-02],
[-1.19782045e-01,  5.95800161e-01,  6.55453622e-01,
  2.38709435e-01,  5.85567772e-01,  3.13297391e-01,
 -4.78802264e-01, -6.00795090e-01, -1.14559121e-01,
  3.57246213e-02, -6.54651523e-02, -2.67522752e-01,
 -5.73347390e-01, -5.60121536e-01, -6.64083779e-01,
 -3.62119913e-01,  1.00626983e-02],
[ 1.21727191e-01, -4.18006867e-01,  7.13811368e-02,
  7.97947824e-01, -4.02392060e-01, -5.18675745e-01,
 -6.82298183e-01, -1.09238148e+00, -1.78569824e-01,
 -7.93654621e-02,  9.50281799e-01,  1.33493260e-01,
 -8.58942151e-01, -4.63274196e-02, -7.98183918e-01,
 -8.05208564e-01,  4.72618677e-02],
[ 1.13224998e-01, -5.22088170e-01,  5.34476161e-01,
 -2.40443334e-01, -9.04432908e-02, -2.37525627e-01,
  7.32684016e-01, -1.52851716e-01, -1.08835265e-01,
 -8.98972601e-02, -2.55013071e-02,  1.40439808e-01,
 -6.15448177e-01, -3.34072262e-01, -6.00367308e-01,
 -4.53568399e-01, -3.25216532e-01],
[ 3.94972384e-01,  9.16032195e-02,  6.35005355e-01,
  6.20653570e-01, -8.77909303e-01,  3.39965731e-01,
 -6.30401492e-01, -5.64714134e-01,  1.24840133e-01,
 -3.12741995e-01, -5.87009728e-01, -2.54345924e-01,
 -7.82958984e-01,  3.19272041e-01,  3.00479591e-01,
 -3.77016723e-01, -2.64459908e-01],
[-3.91743220e-02,  1.10754780e-02, -2.84384102e-01,
  4.42395508e-01,  2.09858686e-01, -4.99879122e-01,
```

```
          -9.10706043e-01,  7.58859992e-01, -5.65062463e-02,
           1.03602551e-01,  5.74107528e-01, -3.93347889e-02,
           3.66998076e-01, -4.55251873e-01,  5.28707922e-01,
           8.80521908e-02, -3.98709625e-01],
         [-1.40591100e-01, -4.29417007e-03,  3.06523126e-02,
          -5.93358219e-01, -9.97789443e-01,  2.82077730e-01,
          -4.35722858e-01,  8.37878585e-01, -3.75346094e-01,
          -1.79947332e-01,  2.14621201e-01, -3.02261319e-02,
           1.96065083e-01, -5.44648468e-01,  1.85054287e-01,
           1.57266483e-01, -4.41018850e-01],
         [ 4.78619397e-01,  4.01760072e-01,  7.67934799e-01,
          -8.05917680e-01, -6.95648074e-01, -1.72201172e-01,
          -1.55523360e+00,  1.17291439e+00,  9.15536806e-02,
          -1.20233011e+00, -1.06491590e+00, -3.47370893e-01,
           3.66485298e-01,  7.67261684e-01,  2.37176105e-01,
          -7.63584018e-01,  2.68105388e-01],
         [ 4.51389760e-01, -5.52120507e-01, -4.23602015e-01,
          -2.22692907e-01,  3.81036788e-01, -3.43598247e-01,
           4.84142274e-01,  2.48648617e-02, -2.85672873e-01,
          -1.97633788e-01,  5.48268676e-01, -1.02515846e-01,
          -2.26549566e-01, -4.06838089e-01, -4.04224783e-01,
           1.33955255e-01,  1.22038936e-02],
         [ 3.88962060e-01,  1.81567445e-01, -7.17053652e-01,
           2.15425074e-01, -1.67034537e-01, -2.63409376e-01,
           9.80212986e-02, -2.58824915e-01, -1.61426812e-02,
           9.19656307e-02,  3.92467558e-01, -4.93264526e-01,
          -6.43476546e-01,  1.75854832e-01, -4.32162315e-01,
          -2.38268331e-01, -1.59788758e-01],
         [ 7.24074692e-02,  9.89253461e-01, -8.72149944e-01,
          -4.29509282e-01,  4.46410447e-01, -4.24668521e-01,
          -4.08271313e-01,  6.85685635e-01, -3.34988713e-01,
          -4.23630416e-01, -4.64578867e-01, -1.47811189e-01,
           7.02983916e-01, -6.06167316e-01, -5.19080877e-01,
          -7.26843596e-01, -2.54551142e-01],
         [ 2.97565311e-01,  3.26346308e-01, -2.74902105e-01,
           2.75862604e-01,  4.16763574e-02, -7.67334327e-02,
           3.63889307e-01,  5.30533910e-01,  1.30498469e-01,
          -2.77325734e-02, -4.11840200e-01, -1.68623656e-01,
          -1.00533199e+00, -2.06705406e-02,  4.96292524e-02,
          -3.56811970e-01, -1.76163614e-01]], dtype=float32),
  array([ 0.19099922,  0.02916801, -0.04365778,  0.29735366, -0.24531348,
          0.1660356 , -0.1513697 , -0.23446557, -0.40085268, -0.22247958,
         -0.23183118, -0.1637206 , -0.17303191, -0.18167539, -0.16953774,
         -0.32299468, -0.31691033], dtype=float32)),
 ()),
((), (((), ((), ())), ((), ()), ()), (), ()))
```

Part 4: Compute Accuracy

You will now evaluate in the test set. Previously, you have seen the accuracy on the training set and the validation (noted as eval) set. You will now evaluate on your test set. To get a good evaluation, you will need to create a mask to avoid counting the padding tokens when computing the accuracy.

Exercise 04

Instructions: Write a program that takes in your model and uses it to evaluate on the test set. You should be able to get an accuracy of 95%.

More Detailed Instructions

- *Step 1*: model(sentences) will give you the predicted output.
- *Step 2*: Prediction will produce an output with an added dimension. For each sentence, for each word, there will be a vector of probabilities for each tag type. For each sentence,word, you need to pick the maximum valued tag. This will require `np.argmax` and careful use of the `axis` argument.
- *Step 3*: Create a mask to prevent counting pad characters. It has the same dimension as output. An example below on matrix comparison provides a hint.
- *Step 4*: Compute the accuracy metric by comparing your outputs against your test labels. Take the sum of that and divide by the total number of **unpadded** tokens. Use your mask value to mask the padded tokens. Return the accuracy.

[18]:
```
#Example of a comparision on a matrix
a = np.array([1, 2, 3, 4])
a == 2
```

[18]:
```
array([False,  True, False, False])
```

[19]:
```
# create the evaluation inputs
x, y = next(data_generator(len(test_sentences), test_sentences, test_labels,␣
 ↪vocab['<PAD>']))
print("input shapes", x.shape, y.shape)
```

```
input shapes (7194, 70) (7194, 70)
```

[20]:
```
# sample prediction
tmp_pred = model(x)
print(type(tmp_pred))
print(f"tmp_pred has shape: {tmp_pred.shape}")
```

```
<class 'jaxlib.xla_extension.DeviceArray'>
tmp_pred has shape: (7194, 70, 17)
```

Note that the model's prediction has 3 axes: - the number of examples - the number of words in each example (padded to be as long as the longest sentence in the batch) - the number of possible targets (the 17 named entity tags).

[57]:
```
# UNQ_C4 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
# GRADED FUNCTION: evaluate_prediction
def evaluate_prediction(pred, labels, pad):
    """
    Inputs:
        pred: prediction array with shape
            (num examples, max sentence length in batch, num of classes)
        labels: array of size (batch_size, seq_len)
        pad: integer representing pad character
    Outputs:
        accuracy: float
    """
    ### START CODE HERE (Replace instances of 'None' with your code) ###
## step 1 ##
    outputs = np.argmax(pred, axis=-1)
    print("outputs shape:", outputs.shape)

## step 2 ##
    mask = labels != pad
    print("mask shape:", mask.shape, "mask[0][20:30]:", mask[0][20:30])
## step 3 ##
    accuracy = np.sum(np.sum((outputs == labels), axis =-1),axis =-1)/np.
 ↪sum(mask)
    ### END CODE HERE ###
    return accuracy
```

[58]:
```
accuracy = evaluate_prediction(model(x), y, vocab['<PAD>'])
print("accuracy: ", accuracy)
```

```
outputs shape: (7194, 70)
mask shape: (7194, 70) mask[0][20:30]: [ True  True  True False False False
False False False False]
accuracy:  0.9543761
```

Expected output (Approximately)

```
outputs shape: (7194, 70)
mask shape: (7194, 70) mask[0][20:30]: [ True  True  True False False False False False False
accuracy:  0.9543761
```

[59]:
```
# Test your function
w3_unittest.test_evaluate_prediction(evaluate_prediction)
```

```
outputs shape: (3, 3)
mask shape: (3, 3) mask[0][20:30]: []
outputs shape: (3, 3)
mask shape: (3, 3) mask[0][20:30]: []
outputs shape: (3, 3)
mask shape: (3, 3) mask[0][20:30]: []
```

```
outputs shape: (3, 4)
mask shape: (3, 4) mask[0][20:30]: []
 All tests passed
```

Part 5: Testing with your own sentence

Below, you can test it out with your own sentence!

[60]:
```
# This is the function you will be using to test your own sentence.
def predict(sentence, model, vocab, tag_map):
    s = [vocab[token] if token in vocab else vocab['UNK'] for token in sentence.
 ↪split(' ')]
    batch_data = np.ones((1, len(s)))
    batch_data[0][:] = s
    sentence = np.array(batch_data).astype(int)
    output = model(sentence)
    outputs = np.argmax(output, axis=2)
    labels = list(tag_map.keys())
    pred = []
    for i in range(len(outputs[0])):
        idx = outputs[0][i]
        pred_label = labels[idx]
        pred.append(pred_label)
    return pred
```

[61]:
```
# Try the output for the introduction example
#sentence = "Many French citizens are goin to visit Morocco for summer"
#sentence = "Sharon Floyd flew to Miami last Friday"

# New york times news:
sentence = "Peter Navarro, the White House director of trade and manufacturing␣
 ↪policy of U.S, said in an interview on Sunday morning that the White House␣
 ↪was working to prepare for the possibility of a second wave of the␣
 ↪coronavirus in the fall, though he said it wouldn't necessarily come"
s = [vocab[token] if token in vocab else vocab['UNK'] for token in sentence.
 ↪split(' ')]
predictions = predict(sentence, model, vocab, tag_map)
for x,y in zip(sentence.split(' '), predictions):
    if y != 'O':
        print(x,y)
```

```
Peter B-per
Navarro, I-per
White B-org
House I-org
Sunday B-tim
morning I-tim
White B-org
House I-org
```

```
coronavirus B-tim
fall, B-tim
```

Expected Results

```
Peter B-per
Navarro, I-per
White B-org
House I-org
Sunday B-tim
morning I-tim
White B-org
House I-org
coronavirus B-tim
fall, B-tim
```

[ ]: