drawing

Application of Deep Learning to Text and Images

Module 2, Lab 4: Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are special types of networks that can capture the dynamics of sequences via repeating connections. In this exercise, you will learn how to use RNNs and apply them to a text classification problem.

You will learn:

- How to perform text transformation
- How to use pre-trained GloVe word embeddings
- How to set up a Recurrent Neural Network model
- How to train and test a RNN model

This lab uses a dataset from a small sample of Amazon product reviews.

Review dataset schema:

- **reviewText:** Text of the review
- **summary:** Summary of the review
- **verified:** Whether the purchase was verified (True or False)
- **time:** UNIX timestamp for the review
- **log_votes:** Logarithm-adjusted votes log(1+votes)
- **isPositive:** Whether the review is positive or negative (1 or 0)

You will be presented with two kinds of exercises throughout the notebook: activities and challenges.

Activity	**Challenge**
No coding is needed for an activity. You try to understand a concept, answer questions, or run a code cell.	Challenges are where you can practice your coding skills.

Important notes:

- One distinction between regular neural networks and recurrent neural networks (RNN) is that recurrent networks specialize in sequential data. With this dataset, you will use RNNs on the **reviewText** field. You will assume that the text is made of words or tokens that are placed in a grammatically logical order. The RNN will understand the associations between the words through the recurrent connections. Eventually, it will learn to classify the text correctly (up to a certain accuracy level).
- If you were interested in including the **summary** field, you would either have to append the summary to the review text or train a separate model. In this lab you will train a RNN using only the **reviewText** field so you can focus on learning the process and keep training time shorter.

Index

```
# installing libraries
!pip install -U -q -r requirements.txt
```

```
ERROR: Could not open requirements file: [Errno 2] No such file or directory: 'requirements.txt'
```

```
# installing libraries
!pip install -U -q -r requirements.txt

# Installing d2l from its GitHub repository and other required packages
!pip uninstall -y torch torchtext torchvision # uninstall all conflicting modules
!pip install torch==2.0.0+cu118 torchvision==0.15.1+cu118 torchtext==0.15.0 --index-url https://download.pytorch.org/whl/cu118 # install tor
```

```
!pip install git+https://github.com/d2l-ai/d2l-en.git
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install scikit-learn

# Import necessary modules
import os, re, time
import numpy as np
import torch
import pandas as pd
import matplotlib.pyplot as plt
# Correctly import d2l
from d2l import torch as d2l
from os import path
from collections import Counter
from torch import nn, optim
from torch.nn import BCEWithLogitsLoss
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import vocab
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from torchtext.vocab import GloVe
import random

# Check if torchtext is installed and get its version
try:
    import torchtext
    print(f"torchtext version: {torchtext.__version__}")
except ModuleNotFoundError:
    print("torchtext is not installed. Please check the installation steps.")

#After running the previous code cell, select 'Kernel' > 'Restart'
```

```
    return _compile(pattern, flags)
           ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.11/re/__init__.py", line 294, in _compile
    p = _compiler.compile(pattern, flags)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.11/re/_compiler.py", line 745, in compile
    p = _parser.parse(p, flags)
        ^^^^^^^^^^^^^^^^^^^^^^^
```

```
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "/usr/local/lib/python3.11/dist-packages/pip/_internal/operations/check.py", line 101, in check_install_conflicts
      package_set, _ = create_package_set_from_installed()
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "/usr/local/lib/python3.11/dist-packages/pip/_internal/operations/check.py", line 42, in create_package_set_from_installed
      dependencies = list(dist.iter_dependencies())
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    File "/usr/local/lib/python3.11/dist-packages/pip/_internal/metadata/importlib/_dists.py", line 222, in iter_dependencies
  ^C
  ^C
  ^C
  torchtext version: 0.15.0+cpu
```

Text Transformation

In this section, you will process the **reviewText** field and convert it into a form that works well with recurrent networks. To do this you will:

- Read the dataset, create train/validation split and fill-in the missing text fields.
- Create a vocabulary using the texts from the **reviewText** field.
  - This vocabulary has a unique integer value for each word in the vocabulary such as "car"->32, "house"->651, ...
- Transform the texts by replacing the words with their corresponding unique integer values.
  - For example: "Happy to own it" becomes [321, 6, 237, 8, 2].
- Use a fixed sequence length of 50 so that you can put the data into a memory efficient form and load it in batches.
  - Longer texts are cut short (to 50 tokens) and shorter ones are padded a special value (1) to complete to 50 token length. 0 is used for unknown words (assume the real-world scenarios involving unknown words).

Start by reading in the dataset and looking at the first five rows.

```
# Download the file and rename it
!wget https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-REVIEW-DATA-CLASSIFICATI
!mv AMAZON-REVIEW-DATA-CLASSIFICATION.csv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv
# Create the data directory if it doesn't exist
!mkdir -p data
# Move the downloaded file to the data directory
!mv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv data/
# Remove the import requirement for requirements.txt as the file is not found
#Import libraries
import os, re, time
import numpy as np
import torch
import pandas as pd
import matplotlib.pyplot as plt
# Correctly import d2l
from d2l import torch as d2l
from os import path
from collections import Counter
from torch import nn, optim
from torch.nn import BCEWithLogitsLoss
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import vocab
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from torchtext.vocab import GloVe
import random

# Check if torchtext is installed and get its version
try:
    import torchtext
    print(f"torchtext version: {torchtext.__version__}")
except ModuleNotFoundError:
    print("torchtext is not installed. Please check the installation steps.")
# verify that the file is now present.
!ls data/
```

```
--2025-02-27 04:28:38--  https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 404 Not Found
2025-02-27 04:28:38 ERROR 404: Not Found.

mv: cannot stat 'AMAZON-REVIEW-DATA-CLASSIFICATION.csv': No such file or directory
mv: cannot stat 'NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv': No such file or directory
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-15-f2babeae1803> in <cell line: 0>()
     10 import os, re, time
     11 import numpy as np
---> 12 import torch
     13 import pandas as pd
     14 import matplotlib.pyplot as plt

                                   3 frames

/usr/lib/python3.11/ctypes/__init__.py in __init__(self, name, mode, handle, use_errno, use_last_error, winmode)
    374 
    375         if handle is None:
--> 376             self._handle = _dlopen(self._name, mode)
    377         else:
    378             self._handle = handle

OSError: /usr/local/lib/python3.11/dist-packages/torch/lib/libtorch_global_deps.so: cannot open shared object file: No such file or
directory
```

Next steps: Explain error

Now, look at the range and distribution of the target column `isPositive`.

```
# Download the file and rename it
!wget https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-REVIEW-DATA-CLASSIFICATI
!mv AMAZON-REVIEW-DATA-CLASSIFICATION.csv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv
# Create the data directory if it doesn't exist
!mkdir -p data
# Move the downloaded file to the data directory
!mv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv data/
# Remove the import requirement for requirements.txt as the file is not found
#Import libraries
import os, re, time
import numpy as np
import torch
import pandas as pd
import matplotlib.pyplot as plt
# Correctly import d2l
from d2l import torch as d2l
from os import path
from collections import Counter
from torch import nn, optim
from torch.nn import BCEWithLogitsLoss
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import vocab
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from torchtext.vocab import GloVe
import random

# Check if torchtext is installed and get its version
try:
    import torchtext
    print(f"torchtext version: {torchtext.__version__}")
except ModuleNotFoundError:
    print("torchtext is not installed. Please check the installation steps.")
# verify that the file is now present.
!ls data/
```

```
--2025-02-27 04:28:35--  https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 404 Not Found
2025-02-27 04:28:35 ERROR 404: Not Found.

mv: cannot stat 'AMAZON-REVIEW-DATA-CLASSIFICATION.csv': No such file or directory
mv: cannot stat 'NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv': No such file or directory
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-14-f2babeae1803> in <cell line: 0>()
     10 import os, re, time
     11 import numpy as np
---> 12 import torch
     13 import pandas as pd
     14 import matplotlib.pyplot as plt

                                 3 frames
/usr/lib/python3.11/ctypes/__init__.py in __init__(self, name, mode, handle, use_errno, use_last_error, winmode)
    374 
    375         if handle is None:
--> 376             self._handle = _dlopen(self._name, mode)
    377         else:
    378             self._handle = handle

OSError: /usr/local/lib/python3.11/dist-packages/torch/lib/libtorch_global_deps.so: cannot open shared object file: No such file or
directory
```

Next steps: Explain error

It is always important that you check the number of missing values for each column.

```
# installing libraries
!pip install -U -q -r requirements.txt

# Installing d2l from its GitHub repository and other required packages
!pip uninstall -y torch torchtext torchvision # uninstall all conflicting modules
!pip install torch==2.0.0+cu118 torchvision==0.15.1+cu118 torchtext==
```

```
ERROR: Could not open requirements file: [Errno 2] No such file or directory: 'requirements.txt'
WARNING: Skipping torch as it is not installed.
WARNING: Skipping torchtext as it is not installed.
WARNING: Skipping torchvision as it is not installed.
ERROR: Invalid requirement: 'torchtext==': Expected end or semicolon (after name and no valid version specifier)
    torchtext==
             ^
```

Since there are missing values in the text fields, specifically in the **reviewText** field, you need to fill-in the missing values with an empty string.

```
# install modules
!pip install -U -q -r requirements.txt
# Installing d2l from its GitHub repository and other required packages
!pip uninstall -y torch torchtext torchvision # uninstall all conflicting modules
!pip install torch==2.0.0+cu118 torchvision==0.15.1+cu118 torchtext==0.15.0 --index-url https://download.pytorch.org/whl/cu118 # install tor
!pip install git+https://github.com/d2l-ai/d2l-en.git
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install scikit-learn

# Download the file and rename it
!wget https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-REVIEW-DATA-CLASSIFICAT
!mv AMAZON-REVIEW-DATA-CLASSIFICATION.csv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv
# Create the data directory if it doesn't exist
!mkdir -p data
# Move the downloaded file to the data directory
!mv NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv data/
# verify that the file is now present.
!ls data/
```

Show hidden output

Now, split the dataset into training and validation.

```
# install all necessary modules
!pip install -U -q torch==2.0.0+cu118 torchvision==0.15.1+cu118 torchtext==0.15.0 --index-url https://download.pytorch.org/whl/cu118
!pip install -U -q git+https://github.com/d2l-ai/d2l-en.git
!pip install -U -q numpy pandas matplotlib scikit-learn

# Import necessary modules
import os
import re
import time
import numpy as np
import torch
import pandas as pd
import matplotlib.pyplot as plt
from d2l import torch as d2l
from os import path
from collections import Counter
from torch import nn, optim
from torch.nn import BCEWithLogitsLoss
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import vocab
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from torchtext.vocab import GloVe
import random

# Check if torchtext is installed and get its version
try:
    import torchtext
    print(f"torchtext version: {torchtext.__version__}")
except ModuleNotFoundError:
    print("torchtext is not installed. Please check the installation steps.")

# --- Download and move file once ---
data_dir = "data"
file_name = "NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv"
file_path = os.path.join(data_dir, file_name)
download_url = "https://github.com/aws-samples/aws-machine-learning-university-accelerated-nlp/raw/master/data/examples/AMAZON-REVIEW-DATA-C
temp_file_name = "AMAZON-REVIEW-DATA-CLASSIFICATION.csv"
temp_file_path = temp_file_name
#check if the directory exists
if not os.path.exists(data_dir):
  !mkdir -p data

# Check if the file has already been downloaded
if not os.path.exists(file_path):
    print("Downloading file...")
    !wget {download_url} -O {temp_file_path}
    #check if it was properly downloaded
    if os.path.exists(temp_file_path) and os.path.getsize(temp_file_path) > 0:
        print(f"File downloaded and saved temporarily as {temp_file_path}.")
        !mv {temp_file_path} {file_name}
        !mv {file_name} data/
        print(f"File moved to {file_path}.")
    else:
        raise FileNotFoundError(f"Failed to download file from {download_url}")

else:
    print(f"File already exists at {file_path}. Skipping download.")

# --- Check if the file is present and not empty ---
if not os.path.exists(file_path):
    raise FileNotFoundError(f"File not found at {file_path}")
if os.path.getsize(file_path) == 0:
    raise ValueError(f"File at {file_path} is empty.")

# verify that the file is now present.
!ls data/

# --- Load the dataset ---
try:
    # Load the dataset into a pandas DataFrame.
    df = pd.read_csv(file_path)
    print("File successfully loaded.")
except pd.errors.EmptyDataError:
    print(f"Error: EmptyDataError when reading {file_path}. File might be empty or corrupted.")
except pd.errors.ParserError:
        print(f"Error: ParserError when reading {file_path}. File might not be a valid CSV.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# This separates 10% of the entire dataset into validation dataset.
# Now, train_test_split should be recognized.
train_text, val_text, train_label, val_label = train_test_split(
    df["reviewText"].tolist(),
    df["isPositive"].tolist(),
    test_size=0.10,
    shuffle=True,
    random_state=324,
)
```

Show hidden output

Next steps: Explain error

Creating a vocabulary:

Once your dataset is ready, you need to create a vocabulary with the tokens from the text data. To do this, use a basic English tokenizer and then use these tokens to create the vocabulary. In this vocabulary, tokens will map to unique ids, such as "car"->32, "house"->651, ...

```
try:
    # Your code that might raise an exception
    requirements = """
    Your multiline string or code here
    """
except Exception as e:
    print(f"An error occurred: {e}")
finally:
    print("Execution completed.")
```

```
Execution completed.
```

To see what the data now looks like, print some examples.

```
# --- Load the dataset ---
try:
    df = pd.read_csv(file_path)
    print("Dataset loaded successfully!")
    print(df.head())  # Show first few rows to confirm successful loading

except FileNotFoundError:
    print(f"Error: The file {file_path} was not found. Please check the file path and try again.")

except pd.errors.EmptyDataError:
    print(f"Error: The file {file_path} is empty. Please provide a valid dataset.")

except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
An unexpected error occurred: name 'file_path' is not defined
```

Now, print the words for the first 25 indexes in the vocabulary.

- `< unk >` is reserved for unknown words
- `< pad >` is used for the padded tokens (more about this in the next section)

```
from torchtext.vocab import build_vocab_from_iterator

# Assuming you have a list of tokenized sentences
tokenized_data = [["this", "is", "an", "example"], ["another", "sentence"]]

# Create a vocabulary from tokenized data
def yield_tokens(data):
    for text in data:
        yield text
```

```
vocab = build_vocab_from_iterator(yield_tokens(tokenized_data), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])  # Handle unknown words

# Print first 25 items in vocabulary
print(vocab.get_itos()[:25])  # ✅ This should work correctly
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-10-f3004ea66d3b> in <cell line: 0>()
----> 1 from torchtext.vocab import build_vocab_from_iterator
      2 
      3 # Assuming you have a list of tokenized sentences
      4 tokenized_data = [["this", "is", "an", "example"], ["another", "sentence"]]
      5 

ModuleNotFoundError: No module named 'torchtext'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

```
import pandas as pd
import os
from torchtext.data.utils import get_tokenizer

# Define file path
file_path = "data/NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv"

# --- Load the dataset ---
try:
    if os.path.exists(file_path) and os.path.getsize(file_path) > 0:
        df = pd.read_csv(file_path)
        print("File successfully loaded.")
        print(df.head())  # Show first few rows to confirm successful loading
    else:
        raise pd.errors.EmptyDataError

except pd.errors.EmptyDataError:
    print(f"Error: EmptyDataError when reading {file_path}. File might be empty or corrupted.")
    df = pd.DataFrame()  # Define df as an empty DataFrame to prevent further errors
except pd.errors.ParserError:
    print(f"Error: ParserError when reading {file_path}. File might not be a valid CSV.")
    df = pd.DataFrame()
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    df = pd.DataFrame()

# --- Tokenizing the loaded data ---
if not df.empty and 'reviewText' in df.columns:
    tokenizer = get_tokenizer("basic_english")
    sample_text = df['reviewText'].iloc[0]  # Get a sample text
    tokenized_text = tokenizer(sample_text)  # Tokenize the sample
    print(f"\nSample Text: {sample_text}")
    print(f"Tokenized Text: {tokenized_text}")
else:
    print("Error: The dataframe is empty or does not contain the column 'reviewText'.")
```

Show hidden output

Next steps: Explain error

Text transformation with defined vocabulary

Now, you can use the vocabulary and map tokens in the text to unique ids of the tokens.

For example: `["this", "is", "a", "sentence"] -> [14, 12, 9, 2066]`

```
# Let's create a mapper to transform our text data
text_transform_pipeline = lambda x: [vocab[token] for token in tokenizer(x)]
```

Once the mapping is complete, you can print some before and after examples.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset into a pandas DataFrame (ensure this step is executed first)
try:
    df = pd.read_csv(file_path)  # Ensure file_path is correctly defined
    print("File successfully loaded.")
    print("Columns in DataFrame:", df.columns)
except Exception as e:
    print(f"An error occurred while loading the file: {e}")
    df = None  # In case of failure, set df to None

# Proceed with splitting only if df is loaded successfully
if df is not None and not df.empty:
    # Check if 'reviewText' and 'isPositive' columns exist
    if 'reviewText' in df.columns and 'isPositive' in df.columns:
        train_text, val_text, train_label, val_label = train_test_split(
            df["reviewText"].tolist(),
            df["isPositive"].tolist(),
            test_size=0.10,
            shuffle=True,
            random_state=324,
        )
        print(f"Training and validation sets created with {len(train_text)} training samples and {len(val_text)} validation samples.")
    else:
        print("Error: DataFrame does not contain the required columns 'reviewText' and 'isPositive'.")
else:
    print("Error: DataFrame is empty or failed to load.")
```

```
An error occurred while loading the file: name 'file_path' is not defined
Error: DataFrame is empty or failed to load.
```

To make this process easier to use, create a function to do all the steps automatically.

Create the function to:

- Transform and pad (if necessary) the text data
- Cut the series of words at the point where it reaches a certain length
  - For this example, use `max_len=50`
  - If the text is shorter than max_len, pad `ones` to the start of the sequence

```
def pad_features(reviews_split, seq_length):
    # Transform the text
    # use the dict to tokenize each review in reviews_split
    # store the tokenized reviews in reviews_ints
    reviews_ints = []
    for review in reviews_split:
        reviews_ints.append(text_transform_pipeline(review))

    # getting the correct rows x cols shape
    features = np.ones((len(reviews_ints), seq_length), dtype=int)

    # for each review, I grab that review
    for i, row in enumerate(reviews_ints):
        features[i, -len(row):] = np.array(row)[:seq_length]

    return torch.tensor(features, dtype=torch.int64)
```

Let's look at two example sentences. Remember that 1 is used for each padded item and 0 is used for each unknown word in the text.

```
import numpy as np

# Assuming df has been loaded properly and train_test_split has been executed successfully
# Example: Ensure train_text and train_label are already defined from the previous code

if 'train_text' not in locals():  # Check if train_text is defined
    print("Error: train_text is not defined. Ensure the data split code is executed correctly.")
else:
    # Define pad_features function (assuming this is part of your preprocessing)
    def pad_features(texts, seq_length):
        # Pad sequences to the specified length using zeros (or any other method)
        padded_texts = np.array([text[:seq_length] + ['<PAD>'] * max(0, seq_length - len(text)) for text in texts])
        return padded_texts

    # Process and display a few samples from train_text
    for text in train_text[15:17]:
        print(f"Text: {text}\n")
        print(f"Original length of the text: {len(text)}\n")
        tt = pad_features([text], seq_length=50)
        print(f"Transformed text: \n{tt}\n")
        print(f"Shape of transformed text: {tt.shape}\n")
```

```
Error: train_text is not defined. Ensure the data split code is executed correctly.
```

Use the `pad_features()` function and create the data loaders and use `max_len=50` to consider only the first 50 words in the text.

```
# Import necessary modules from PyTorch
from torch.utils.data import TensorDataset, DataLoader
import torch
import numpy as np

# Define pad_features function if it's not already defined
def pad_features(texts, seq_length):
    # Pad sequences to the specified length using zeros (or any other method)
    padded_texts = np.array([text[:seq_length] + ['<PAD>'] * max(0, seq_length - len(text)) for text in texts])
    return padded_texts

max_len = 50
batch_size = 64

# Assuming train_text, train_label, val_text, and val_label are already defined

# Pass transformed and padded data to dataset
# Create data loaders
train_dataset = TensorDataset(
    torch.tensor(pad_features(train_text, max_len)),
    torch.tensor(train_label)
)
train_loader = DataLoader(train_dataset,
                          batch_size=batch_size,
                          drop_last=True)

val_dataset = TensorDataset(
    torch.tensor(pad_features(val_text, max_len)),
    torch.tensor(val_label)
)
val_loader = DataLoader(val_dataset,
                        batch_size=batch_size,
                        drop_last=True)
```

Show hidden output

Next steps: Explain error

Using pre-trained GloVe word embeddings

In this example, you will use GloVe word vectors `name="6B"` with `dim=300`. This gives 6 billion words/phrases vectors. Each word vector has 300 numbers.

The following code shows how to get the word vectors and create an embedding matrix from them. You will connect your vocabulary indexes to the GloVe embedding with the `get_vecs_by_tokens()` function.

```
glove = GloVe(name="6B", dim=300)
embedding_matrix = glove.get_vecs_by_tokens(vocab.get_itos())
```

```
.vector_cache/glove.6B.zip: 862MB [02:39, 5.42MB/s]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-20-5e536f0cff8d> in <cell line: 0>()
----> 1 glove = GloVe(name="6B", dim=300)
      2 embedding_matrix = glove.get_vecs_by_tokens(vocab.get_itos())

                    7 frames

/usr/lib/python3.11/zipfile.py in _read1(self, n)
   1040         elif self._compress_type == ZIP_DEFLATED:
   1041             n = max(n, self.MIN_READ_SIZE)
-> 1042             data = self._decompressor.decompress(data, n)
   1043             self._eof = (self._decompressor.eof or
   1044                          self._compress_left <= 0 and

KeyboardInterrupt:
```

Now you need to set your parameters such as number of epochs and the vocabulary size.

```
# Size of the state vectors
hidden_size = 128

# General NN training parameters
learning_rate = 0.001
num_epochs = 35

# Embedding vector and vocabulary sizes
embed_size = 300  # glove.6B.300d.txt
vocab_size = len(vocab.get_itos())
```

We need to put our data into correct format before the process.

Recurrent Neural Networks

Interact with the basic word-level RNN below. Each sequence in the RNN is predicted from information in the previous hidden layer, as well as the previous word in the sequence:

```
RNN()
```

Setting-up the Recurrent Neural Network model

The model is made of these layers:

- Embedding layer:
    - Words/tokens are mapped to word vectors
- RNN layer:
    - A simple RNN model
    - Stack 2 RNN layers
    - For more details about the RNN read the PyTorch RNN documentation
- Linear layer:
    - A linear layer with two neurons (for two output classes) is used to output the `isPositive` prediction

```
class Net(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, num_classes, num_layers=1):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size, padding_idx=1)
        self.rnn = nn.RNN(
            embed_size, hidden_size, num_layers=num_layers, batch_first=True
        )

        self.linear = nn.Linear(hidden_size, num_classes)
    def forward(self, inputs):
        embeddings = self.embedding(inputs)
        # Call the RNN layer
        outputs, _ = self.rnn(embeddings)

        # Output shape after RNN: (batch_size, max_len, hidden_size)
        # Get the output from the last time step with outputs[:, -1, :] below
        # The output shape becomes: (batch_size, 1, hidden_size)
        # Send it through the linear layer
        return self.linear(outputs[:, -1, :])

# Initialize the weights
def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.xavier_uniform_(m.weight)
    if type(m) == nn.RNN:
        for param in m._flat_weights_names:
            if "weight" in param:
                nn.init.xavier_uniform_(m._parameters[param])
```

Now you can initialize the network and then make the embedding layer use the GloVe word vectors.

```
# Our architecture with 2 RNN layers
model = Net(vocab_size, embed_size, hidden_size,
            num_classes=2, num_layers=2)

# We set the embedding layer's parameters from GloVe
model.embedding.weight.data.copy_(embedding_matrix)
# We won't change/train the embedding layer
model.embedding.weight.requires_grad = False
```

Training and testing the model

You are now ready to train the model. To do this, first define the evaluation and training functions.

```
def accuracy(y_hat, y):
    """Compute the number of correct predictions."""
    pred = torch.argmax(y_hat, axis=1)
    return torch.sum(pred == y)

def eval_accuracy(net, data_loader):
    # Use accumulator to keep track of metrics: correct predictions, num of predictions
    metric = d2l.Accumulator(2)

    net.eval()
    for X, y in data_loader:
        y_hat = net(X)
        metric.add(accuracy(y_hat, y), y.numel())

    return metric[0] / metric[1]

print("Classification Accuracy:", eval_accuracy(model, val_loader))
```

Finally! It is time to start the training process!

To help see what is happening, after each epoch the cross-entropy loss will be printed.

```
# Train the network
def train_net(net, train_loader, test_loader, num_epochs=1, lr=0.001):

    net.apply(init_weights)
    loss = nn.CrossEntropyLoss()
    trainer = torch.optim.SGD(net.parameters(), lr=lr)

    # Collect training times for each epoch
    train_times = []
    # Collect train losses after each epoch
    train_losses = []
    # Collect train and test accuracy
    train_accs, test_accs = [], []
    net.train()
    for epoch in range(num_epochs):
        train_loss = 0
        metric = d2l.Accumulator(3)

        timer = d2l.Timer()
        timer.start()
        # Training loop
        for X, y in train_loader:
            # Compute gradients and update parameters
            y_hat = net(X)
            l = loss(y_hat, y)
            trainer.zero_grad()
            l.backward()
            trainer.step()
            metric.add(l.item() * len(y), accuracy(y_hat, y), y.numel())
            train_loss, train_acc = metric[0]/metric[2], metric[1]/metric[2]
        timer.stop()

        # Store training times
        train_times.append(timer.sum())
        # Store the loss after one epoch of training
        train_losses.append(train_loss)
        # Store the train accuracy
        train_accs.append(train_acc)
        # Compute the test accuracy after one epoch
        test_acc = eval_accuracy(net, test_loader)
        test_accs.append(test_acc)

        print(f'epoch {epoch+1}, Train loss {train_loss:.4f}, Train accuracy {train_acc:.4f}, Val accuracy {test_acc:.4f},  Training time (s

    return train_losses, train_accs, test_accs
```

To add clarity, define a function to plot the losses and accuracies.

```
# Plot the training losses
def plot_losses(train_losses, train_accs, test_accs):

    plt.plot(train_losses, label="Training Loss")
    plt.title("Loss values")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    plt.plot(train_accs, "g", label="Train Accuracy")
    plt.plot(test_accs, "red", label="Validation Accuracy")
    plt.title("Accuracy values")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()
```

Now you can use the plotting function to display the results.

```
%%time

train_losses, train_accs, val_accs = train_net(model, train_loader,
                                               val_loader, num_epochs=num_epochs,
                                               lr=learning_rate)

plot_losses(train_losses, train_accs, val_accs)
```

Finally, you can use the `eval_accuracy()` function to calculate validation set performance.

```
print("Classification Accuracy on Validation set:", eval_accuracy(model, val_loader))
```

When you look at the plots, you probably noticed that the model hasn't reached a plateau for the validation set. This indicates that your model has not train long enough. With this setup, the way to have your model train longer is to increase the number of `epochs` it trains.

The number of `epochs` is set in the Using pre-trained GloVe word embeddings section.

Try it Yourself!

Challenge

Increase the `num_epochs` parameter to a larger value (25, 30, ...)

Then, re-run the notebook

Did your **Validation** accuracy **improve**?

Conclusion

RNN's are a very important tools, especially for problems involving sequential data. You have learned how to build a simple RNN and use it to solve a sample problem. If you are further interested in improving your model, you can try the following:

- Change your hyper-parameters: Learning rate, batch size, and hidden size
- Increase the number of layers: num_layers
- Switch to Gated Recurrent Units and Long Sort-term Memory Networks.

Next Lab: Finetuning the BERT model

Transformers have been extremely popular and successful models in Natural Language Processing problems. In the next lab you will learn how to use a previously trained transformer model called **"BERT"** to solve a text classification problem.