In [1]:

```
# Please do not change this cell because some hidden tests might depend on it.
import os

# Otter grader does not handle ! commands well, so we define and use our
# own function to execute shell commands.
def shell(commands, warn=True):
    """Executes the string `commands` as a sequence of shell commands.

       Prints the result to stdout and returns the exit status.
       Provides a printed warning on non-zero exit status unless `warn`
       flag is unset.
    """
    file = os.popen(commands)
    print (file.read().rstrip('\n'))
    exit_status = file.close()
    if warn and exit_status != None:
        print(f"Completed with errors. Exit status: {exit_status}\n")
    return exit_status

shell("""
ls requirements.txt >/dev/null 2>&1
if [ ! $? = 0 ]; then
 rm -rf .tmp
 git clone https://github.com/cs187-2021/project1.git .tmp
 mv .tmp/requirements.txt ./
 rm -rf .tmp
fi
pip install -q -r requirements.txt
""")
```

In [2]:

```
# Initialize Otter
import otter
grader = otter.Notebook()
```

%%latex \newcommand{\vect}[1]{\mathbf{#1}} \newcommand{\cnt}[1]{\sharp(#1)} \newcommand{\argmax}[1]{\underset{#1}{\operatorname{argmax}}} \newcommand{\softmax}{\operatorname{softmax}} \newcommand{\Prob}{\Pr} \newcommand{\given}{\,|\,}

CS187

Project segment 1: Text classification

In this project segment you will build several varieties of text classifiers using PyTorch.

1. A majority baseline.
2. A naive Bayes classifer.
3. A logistic regression (single-layer perceptron) classifier.
4. A multilayer perceptron classifier.

Preparation {-}

In [3]:
```
import copy
import re
import wget
import torch
import torch.nn as nn
import torchtext.legacy as tt

from collections import Counter
from torch import optim
from tqdm.auto import tqdm
```

In [4]:
```
# Random seed
random_seed = 1234
torch.manual_seed(random_seed)

## GPU check
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cpu
```

The task: Answer types for ATIS queries

For this and future project segments, you will be working with a standard natural-language-processing dataset, the ATIS (Airline Travel Information System) dataset. This dataset is composed of queries about flights – their dates, times, locations, airlines, and the like.

Over the years, the dataset has been annotated in all kinds of ways, with parts of speech, informational chunks, parse trees, and even corresponding SQL database queries. You'll use various of these annotations in future assignments. For this project segment, however, you'll pursue an easier classification task: **given a query, predict the answer type**.

These queries ask for different types of answers, such as

- Flight IDs: "Show me the flights from Washington to Boston"
- Fares: "How much is the cheapest flight to Milwaukee"
- City names: "Where does flight 100 fly to?"

In all, there are some 30 answer types to the queries.

Below is an example taken from this dataset:

Query:

```
show me the afternoon flights from washington to boston
```

SQL:

```
SELECT DISTINCT flight_1.flight_id FROM flight flight_1 ,
airport_service airport_service_1 , city city_1 , airport_service
airport_service_2 , city city_2
   WHERE flight_1.departure_time BETWEEN 1200 AND 1800
     AND ( flight_1.from_airport = airport_service_1.airport_code
           AND airport_service_1.city_code = city_1.city_code
           AND city_1.city_name = 'WASHINGTON'
           AND flight_1.to_airport = airport_service_2.airport_code
           AND airport_service_2.city_code = city_2.city_code
           AND city_2.city_name = 'BOSTON' )
```

In this project segment, we will consider the answer type for a natural-language query to be the target field of the corresponding SQL query. For the above example, the answer type would be *flight_id*.

Loading and preprocessing the data

> Read over this section, executing the cells, and **making sure you understand what's going on before proceeding to the next parts.**

First, let's download the dataset.

In [5]:

```
data_dir = "https://raw.githubusercontent.com/nlp-course/data/master/ATIS/"
os.makedirs('data', exist_ok=True)
for file in ["train.nl",
             "train.sql",
             "dev.nl",
             "dev.sql",
             "test.nl",
             "test.sql"]:
    wget.download(f"{data_dir}{file}", out='data/')
```

```
100%
[..........................................................................] 2
50477 / 250477
```

We use `torchtext` to prepare the data, as in lab 1-5. More information on `torchtext` can be found at https://pytorch.org/text/0.8.1/data.html.

> You'll notice that we link to version 0.8.1 of the PyTorch documentation, because the `torchtext.data.Field` class is now deprecated. We therefore imported it as `torchtext.legacy` at the top of this notebook. Sadly, torchtext has no convenient replacement for `Field` at the moment.

To begin, `torchtext` requires that we define a mapping from the raw data to featurized indices, called a `Field`. We need one field for processing the question (`TEXT`), and another for processing the label (`LABEL`). These fields make it easy to map back and forth between readable data and lower-level representations like numbers.

In [6]:

```
TEXT = tt.data.Field(lower=True,              # lowercase all tokens
                     sequential=True,         # sequential data
                     include_lengths=False,   # do not include lengths
```

```
                        batch_first=True,       # batches will be batch_size X max_l
                        tokenize=tt.data.get_tokenizer("basic_english"))
LABEL = tt.data.Field(batch_first=True, sequential=False, unk_token=None)
```

We provide an interface for loading ATIS data, built on top of `torchtext.data.Dataset` .

In [7]:

```
class ATIS(tt.data.Dataset):
  @staticmethod
  def sort_key(ex):
    return len(ex.text)

  def __init__(self, path, text_field, label_field, **kwargs):
    """Creates an ATIS dataset instance given a path and fields.
    Arguments:
        path: Path to the data file
        text_field: The field that will be used for text data.
        label_field: The field that will be used for label data.
        Remaining keyword arguments: Passed to the constructor of
            tt.data.Dataset.
    """
    fields = [('text', text_field), ('label', label_field)]

    examples = []
    # Get text
    with open(path+'.nl', 'r') as f:
        for line in f:
            ex = tt.data.Example()
            ex.text = text_field.preprocess(line.strip())
            examples.append(ex)

    # Get labels
    with open(path+'.sql', 'r') as f:
        for i, line in enumerate(f):
            label = self._get_label_from_query(line.strip())
            examples[i].label = label

    super(ATIS, self).__init__(examples, fields, **kwargs)

  def _get_label_from_query(self, query):
    """Returns the answer type from `query` by dead reckoning.
    It's basically the second or third token in the SQL query.
    """
    match = re.match(r'\s*SELECT\s+(DISTINCT\s*)?(\w+\.)?(?P<label>\w+)', query)
    if match:
        label = match.group('label')
    else:
        raise RuntimeError(f'no label in query {query}')
    return label

  @classmethod
  def splits(cls, text_field, label_field, path='./',
             train='train', validation='dev', test='test',
             **kwargs):
    """Create dataset objects for splits of the ATIS dataset.

    Arguments:
        text_field: The field that will be used for the sentence.
        label_field: The field that will be used for label data.
        root: The root directory that the dataset's zip archive will be
            expanded into; therefore the directory in whose trees
            subdirectory the data files will be stored.
        train: The filename of the train data. Default: 'train.txt'.
        validation: The filename of the validation data, or None to not
            load the validation set. Default: 'dev.txt'.
        test: The filename of the test data, or None to not load the test
            set. Default: 'test.txt'.
        Remaining keyword arguments: Passed to the splits method of
            Dataset.
    """

    train_data = None if train is None else cls(
        os.path.join(path, train), text_field, label_field, **kwargs)
    val_data = None if validation is None else cls(
        os.path.join(path, validation), text_field, label_field, **kwargs)
    test_data = None if test is None else cls(
        os.path.join(path, test), text_field, label_field, **kwargs)
    return tuple(d for d in (train_data, val_data, test_data)
                 if d is not None)
```

We split the data into training, validation, and test corpora, and build the vocabularies from the training data.

In [8]:

```
import math
#Make splits for data
train_data, val_data, test_data = ATIS.splits(TEXT, LABEL, path='./data/')

# Build vocabulary for data fields
MIN_FREQ = 3 # words appearing fewer than 3 times are treated as 'unknown'
TEXT.build_vocab(train_data, min_freq=MIN_FREQ)
LABEL.build_vocab(train_data)

# Compute size of vocabulary
print(TEXT.vocab.itos)
vocab_size = len(TEXT.vocab.itos)
num_labels = len(LABEL.vocab.itos)
print(f"Size of vocab: {vocab_size}")
print(f"Number of labels: {num_labels}")
```

```
['<unk>', '<pad>', 'to', 'from', 'flights', 'the', 'on', 'what', 'me', 'flight',
'boston', 'show', 'i', 'san', 'denver', 'a', 'in', 'francisco', 'atlanta', 'an
d', 'pittsburgh', 'is', 'dallas', 'baltimore', 'all', 'philadelphia', 'like',
"'", 'are', 'list', 'airlines', 'of', 'that', 'between', 'washington', 'leavin
g', 'please', 'morning', 'would', 'fare', 'fly', 'for', 'first', 'oakland', 'aft
er', 'there', 'wednesday', 'd', 'ground', 'cheapest', 'you', 'transportation',
'does', 'class', 'need', 'trip', 'city', 'arriving', 'round', 'available', 'hav
e', 'before', 'with', 'afternoon', 'which', 'one', 'fares', 'way', 'american',
'new', 'leave', 'at', 'give', 'monday', 'want', 'dc', 'york', 'earliest', 'nonst
op', 'thursday', 'arrive', 'united', 'go', 'information', 'tuesday', 'can', 'air
port', 'find', 'how', '.', 'st', 'evening', 'twenty', 'newark', 'noon', 'miami',
'milwaukee', 'delta', 'sunday', 'any', 'august', 'vegas', 'charlotte', 'las',
's', 'continental', 'do', 'o', 'stop', 'chicago', 'clock', 'toronto', 'diego',
'july', 'orlando', 'seventh', 'airline', 'friday', 'fort', 'saturday', 'worth',
'next', 'us', 'air', 'early', 'phoenix', 'houston', 'indianapolis', 'seattle',
'tell', 'kansas', 'code', 'latest', 'tomorrow', 'aircraft', 'downtown', 'angele
s', 'lake', 'los', 'salt', 'cleveland', 'cost', 'going', 'around', 'stopover',
'see', 'montreal', 'may', '5pm', 'june', 'about', 'petersburg', 'by', 'get',
'm', 'memphis', 'an', 'many', 'mean', 'ticket', 'twa', 'expensive', 'minneapoli
s', 'tampa', 'departing', 'leaves', 'long', 'type', 'jose', 'or', 'could', 'kno
w', 'nashville', 'travel', 'daily', 'dollars', 'international', 'louis', 'much',
'okay', 'tacoma', 'than', '10am', '12', '6pm', 'beach', 'cincinnati', 'time', 'b
ook', 'least', 'depart', 'into', 'meal', 'service', 'california', 'coach', 'detr
oit', 'economy', 'last', 'november', 'northwest', 'serve', '7pm', 'columbus', 'l
owest', 'second', 'used', 'day', 'less', 'night', 'september', 'pm', 'flying',
'general', 'mitchell', 'paul', 'serves', 'kind', 'now', 'third', '5', 'am', 'arr
ives', 'breakfast', 'field', 'it', 'love', 'be', 'december', 'return', 'as', 'di
rect', 'make', 'out', 'possible', 'schedule', 'stops', 'times', 'april', 'burban
k', 'car', 'la', 'airports', 'connecting', 'has', 'interested', 'late', 'lookin
g', 'number', 'restriction', 'take', '1991', '8pm', 'business', 'dinner', 'fourt
h', 'goes', 'price', 'stopping', '2pm', '4pm', '6', 'eastern', 'eighth', 'fift
h', 'ontario', 'will', 'dl', 'this', 'airfare', 'also', 'arrangements', 'most',
'plane', 'using', 'via', 'your', '1000', '8', 'fifteenth', 'prices', 'rental',
'twentieth', 'week', 'westchester', '4', '8am', '9am', 'display', 'ninth', 'retu
rning', 'sixth', 'stand', 'thirtieth', 'through', 'county', 'flies', 'listing',
'today', 'ua', 'wednesdays', 'again', 'bwi', 'only', 'other', 'tenth', 'types',
'carolina', 'express', 'f', 'midwest', 'north', 'qx', 'smallest', 'tickets', 'wh
ere', '12pm', 'ap/57', 'back', 'classes', 'eleventh', 'florida', 'h', 'jersey',
'limousine', 'lunch', 'meals', 'served', 'should', 'twelfth', 'two', '10pm', 'ca
pacity', 'cities', 'colorado', 'guardia', 'hi', 'makes', 'mco', 'october', 'righ
t', 'same', 'seating', 'shortest', 'under', '9', 'dfw', 'either', 'explain', 'ja
nuary', 'landings', 'layover', 'march', 'my', 'nineteenth', 'seventeenth', 'som
e', 'thank', 'then', 'transport', 'y', '3', '3pm', '466', '7', 'airplane', 'cana
dian', 'codes', 'departure', 'distance', 'ewr', 'fourteenth', 'hours', 'hp', 'i
f', 'logan', 'over', 'planes', 're', 'sixteenth', 'thirty', 'when', '1', '10',
'1pm', '2', '281', '9pm', 'abbreviation', 'arrange', 'both', 'canada', 'each',
'february', 'let', 'no', 'people', 'q', 'serving', 'takeoffs', 'trying', 'use',
'uses', 'yn', '747', 'anywhere', 'arrival', 'booking', 'but', 'later', 'live',
'mornings', 'name', 'nationair', 'numbers', 'ohio', 'provided', 'qo', 'qw', 'ren
t', 'sfo', 'southwest', 'stopovers', 'their', 'thrift', 'traveling', 'yes', '110
0', '1115am', '11am', '1245pm', 'ap57', 'area', 'arrivals', 'boeing', 'coming',
'departs', 'departures', 'destination', 'during', 'ea', 'eighteenth', 'far', 'f
f', 'following', 'georgia', 'help', 'jfk', 'll', 'making', 'midnight', 'offer',
'passengers', 'pennsylvania', 'rates', 'requesting', 'these', 'total', 'tuesday
s', '1291', '1992', '21', '2100', '296', '430pm', '630am', '718am', '7am', '82
5', '838', '934pm', 'another', 'ap80', 'cheap', 'choices', 'close', 'connect',
'costs', 'dc10', 'describe', 'fit', 'fn', 'great', 'hello', 'include', 'kinds',
'land', 'lufthansa', 'meaning', 'minnesota', 'more', 'near', 'noontime', 'nw',
'options', 'reservation', 'saturdays', 'seats', 'services', 'six', 'sometime',
'sorry', 'stands', 'sundays', 'taxi', 'tennessee', 'three', 'thursdays', 'trip
s', 'turboprop', 'we', 'weekday']
Size of vocab: 512
Number of labels: 30
```

To get a sense of the kinds of things that are asked about in this dataset, here is the list of all of the answer types in the training data.

In [9]:
```python
for i, label in enumerate(sorted(LABEL.vocab.itos)):
    print(f"{i:2d} {label}")
```

```
 0 advance_purchase
 1 aircraft_code
 2 airline_code
 3 airport_code
 4 airport_location
 5 arrival_time
 6 basic_type
 7 booking_class
 8 city_code
 9 city_name
10 count
```

```
11 day_name
12 departure_time
13 fare_basis_code
14 fare_id
15 flight_id
16 flight_number
17 ground_fare
18 meal_code
19 meal_description
20 miles_distant
21 minimum_connect_time
22 minutes_distant
23 restriction_code
24 state_code
25 stop_airport
26 stops
27 time_elapsed
28 time_zone_code
29 transport_type
```

Handling unknown words

Note that we mapped words appearing fewer than 3 times to a special *unknown* token (we're using the `torchtext` default, `<unk>`) for two reasons:

1. Due to the scarcity of such rare words in training data, we might not be able to learn generalizable conclusions about them.
2. Introducing an unknown token allows us to deal with out-of-vocabulary words in the test data as well: we just map those words to `<unk>` .

In [10]:

```
unk_token = TEXT.unk_token
print (f"Unknown token: {unk_token}")
unk_index = TEXT.vocab.stoi[unk_token]
print (f"Unknown token id: {unk_index}")

# UNK example
example_unk_token = 'IAmAnUnknownWordForSure'
print (f"An unknown token: {example_unk_token}")
print (f"Mapped back to word id: {TEXT.vocab.stoi[example_unk_token]}")
print (f"Mapped to <unk>?: {TEXT.vocab.stoi[example_unk_token] == unk_index}")
```

```
Unknown token: <unk>
Unknown token id: 0
An unknown token: IAmAnUnknownWordForSure
Mapped back to word id: 0
Mapped to <unk>?: True
```

Batching the data

To load data in batches, we use `data.BucketIterator` . This enables us to iterate over the dataset under a given `BATCH_SIZE` which specifies how many examples we want to process at a time.

In [11]:

```
BATCH_SIZE = 32
```

```
train_iter = tt.data.BucketIterator(train_data, batch_size=BATCH_SIZE, device=de
val_iter = tt.data.BucketIterator(val_data, batch_size=BATCH_SIZE, device=device
test_iter = tt.data.Iterator(test_data, batch_size=BATCH_SIZE, sort=False, devic
```

Let's look at a single batch from one of these iterators.

In [12]:

```
batch = next(iter(train_iter))
text = batch.text
print(batch.text)
for arr in batch.text:
    print(" ".join(TEXT.vocab.itos[i] for i in arr))
print (f"Size of text batch: {text.size()}")
print (f"Third sentence in batch: {text[2]}")
print (f"Converted back to string: {' '.join([TEXT.vocab.itos[i] for i in text[2

print(train_iter.dataset[2].text)

label = batch.label
print(label)
# print(LABEL.vocab.itos[1])
# print (f"Size of label batch: {label.size()}")
# print (f"Third label in batch: {label[2]}")
# print(LABEL.vocab.itos)
# print (f"Converted back to string: {LABEL.vocab.itos[label[2].item()]}")
```

```
tensor([[221,   4,   3,  22,   2,  13,  17,   1,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  21, 276, 198,  55,   3,  34,   2,  10,   6, 100,  92, 115,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 12,  38,  26,   2, 145,   5,   4,   3,  23,   2,  25,  36,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 12,  54,   2,  82,   3,  10,   2,  18,  19, 318,  16,   5, 341, 207,
          87,   8,   5,  77,   9,   3,  10],
        [ 11,   8,  48,  51,  16,  23,   1,   1,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 28,  45,  99,   4,  61, 183,  94,   6, 149, 279,   3, 166, 185,   2,
         203,   1,   1,   1,   1,   1,   1],
        [  7,  27, 104,   5,  77,   9,  35,  14,  41,  20,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 29,  41,   8, 303,   5,  81,   4,  33,  14,  19,  43,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 12,  54,   2,  40,   3,  14,   2,  13,  17,  84,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 12,  27,  47,  26,  15,   9, 133,  91,   3, 172,   2, 126,  32,   0,
         254,  41, 133,   1,   1,   1,   1],
        [  7,  21,   5, 189, 161,  65,  67,  39,  33,  25,  19,  10,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,   4,  28,  45,   3,  18,   2,  23,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 72,   8,  24,   5,   4,   3,  69,  76,   2,  95,  58,  55,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 36,  29,   5,  37,   4,   3, 130,  56,   2,  18,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,   0,  31,  48,  51,  21,  45,  16,  34,  75,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  21,   5, 406, 187,  16,  13,  17,  41,   5,   0,   9,  35,  34,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  21,   5, 195, 197,  53, 209, 193,   3,  10,   2,  13,  17,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  63,   4,  28,  59,  33,  14,  19,  22, 118, 120,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  21, 299,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  3, 128,   2, 139, 137,  56,   1,   1,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 12,  38,  26,  15,   9,   3,  25,   2,  13,  17, 408,  12,  38,  26,
           2, 108,  16,  22,   1,   1,   1],
        [  7,  78,   4,  28,  59,   3,  43,   2,  20,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 29, 174,   4,   3,  43,   2,  10, 274,  97,  30,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,   4,  28,  45,  46,  37,   3,  18,   2,  25,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 36,  11,   8,   5,  58,  55,   4,   3,  90,  89, 151,   2, 111,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 36,  11,   8,  24,   5,   4,   3, 127,   2,  13, 112, 133,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  21,   5,  77,   9,  33, 375,  19, 302,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7,  28,   5,   4,   3,  10,   2,  13,  17,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 87,   5,  49,  65,  67,  39,   3,  20,   2,  13,  17,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  7, 200,   4,  70,  14,  61,  94,   1,   1,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [  4,  32,  82,   3,  43,   2,  25,   6, 117,   1,   1,   1,   1,   1,
           1,   1,   1,   1,   1,   1,   1],
        [ 11,   8,  24,   0,   4,   3,  34,  75,   2,  13,  17,  19,  29, 423,
          66,   1,   1,   1,   1,   1,   1]])
am flights from dallas to san francisco <pad> <pad> <pad> <pad> <pad> <pad> <pad
> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what is your last trip from washington to boston on august twenty seventh <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad>
i would like to see the flights from baltimore to philadelphia please <pad> <pad
> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
i need to go from boston to atlanta and back in the same day find me the earlies
t flight from boston
show me ground transportation in baltimore <pad> <pad> <pad> <pad> <pad> <pad> <
pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
are there any flights before 12 noon on june fifteenth from long beach to columb
us <pad> <pad> <pad> <pad> <pad> <pad>
what ' s the earliest flight leaving denver for pittsburgh <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
list for me only the united flights between denver and oakland <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad>
i need to fly from denver to san francisco tuesday <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad>
i ' d like a flight tomorrow evening from nashville to houston that <unk> dinner
for tomorrow <pad> <pad> <pad> <pad>
what is the least expensive one way fare between philadelphia and boston <pad> <
pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what flights are there from atlanta to baltimore <pad> <pad> <pad> <pad> <pad> <
pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
give me all the flights from new york to miami round trip <pad> <pad> <pad> <pad
> <pad> <pad> <pad> <pad> <pad>
please list the morning flights from kansas city to atlanta <pad> <pad> <pad> <p
ad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what <unk> of ground transportation is there in washington dc <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what is the arrival time in san francisco for the <unk> flight leaving washingto
```

```
n <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what is the coach economy class night service from boston to san francisco <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad>
what afternoon flights are available between denver and dallas fort worth <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what is ua <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad>
from seattle to salt lake city <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <
pad> <pad> <pad> <pad> <pad> <pad> <pad>
i would like a flight from philadelphia to san francisco but i would like to sto
p in dallas <pad> <pad> <pad>
what nonstop flights are available from oakland to pittsburgh <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
list daily flights from oakland to boston using delta airlines <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what flights are there wednesday morning from atlanta to philadelphia <pad> <pad
> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
please show me the round trip flights from st . petersburg to toronto <pad> <pad
> <pad> <pad> <pad> <pad> <pad> <pad>
please show me all the flights from indianapolis to san diego tomorrow <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what is the earliest flight between logan and bwi <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad>
what are the flights from boston to san francisco <pad> <pad> <pad> <pad> <pad>
<pad> <pad> <pad> <pad> <pad> <pad> <pad>
find the cheapest one way fare from pittsburgh to san francisco <pad> <pad> <pad
> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
what northwest flights leave denver before noon <pad> <pad> <pad> <pad> <pad> <p
ad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
flights that go from oakland to philadelphia on friday <pad> <pad> <pad> <pad> <
pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
show me all <unk> flights from washington dc to san francisco and list their far
es <pad> <pad> <pad> <pad> <pad> <pad>
Size of text batch: torch.Size([32, 21])
Third sentence in batch: tensor([ 12,  38,  26,   2, 145,   5,   4,   3,  23,
2,  25,  36,   1,   1,
          1,   1,   1,   1,   1,   1,   1])
Converted back to string: i would like to see the flights from baltimore to phil
adelphia please <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
['what', 'flights', 'from', 'tacoma', 'to', 'orlando', 'on', 'saturday']
tensor([ 0,  0,  0,  0,  2,  0,  0,  0,  0,  0,  1,  0,  0,  0,  2, 13,  0,  0,
         3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0])
```

You might notice some padding tokens `<pad>` when we convert word ids back to strings, or equivalently, padding ids `1` in the corresponding tensor. The reason why we need such padding is because the sentences in a batch might be of different lengths, and to save them in a 2D tensor for parallel processing, sentences that are shorter than the longest sentence need to be padded with some placeholder values. `torchtext` does all this for us automatically. Later during training you'll need to make sure that the paddings do not affect the final results.

In [13]:
```
padding_token = TEXT.pad_token
print (f"Padding token: {padding_token}")

padding_id = TEXT.vocab.stoi[padding_token]
print (f"Padding word id: {padding_id}")
```

```
Padding token: <pad>
Padding word id: 1
```

Alternatively, we can also directly iterate over the individual examples in `train_data`, `val_data` and `test_data`. Here the returned values are the raw sentences and labels instead of their corresponding ids, and you might need to explicitly deal with the unknown words, unlike using bucket iterators which automatically map unknown words to an unknown word id.

In [14]:
```
for example in train_iter.dataset[:5]: # train_iter.dataset is just train_data
  print(f"{example.label:10} -- {' '.join(example.text)}")
```

```
flight_id  -- list all the flights that arrive at general mitchell international
from various cities
flight_id  -- give me the flights leaving denver august ninth coming back to bos
ton
flight_id  -- what flights from tacoma to orlando on saturday
fare_id    -- what is the most expensive one way fare from boston to atlanta on
american airlines
flight_id  -- what flights return from denver to philadelphia on a saturday
```

Notations used

In this project segment, we'll use the following notations.

- Sequences of elements (vectors and the like) are written with angle brackets and commas ($\langle w_1, \ldots, w_M \rangle$) or directly with no punctuation ($w_1 \cdots w_M$).
- Sets are notated similarly but with braces, ($\{v_1, \ldots, v_V\}$).
- Maximum indices (M, N, V, T, and X in the following) are written as uppercase italics.
- Variables over sequences and sets are written in boldface ($\mathbf{w}$), typically with the same letter as the variables over their elements.

In particular,

- $\mathbf{w} = w_1 \cdots w_M$: A text to be classified, each element w_j being a word token.
- $\mathbf{v} = \{v_1, \ldots, v_V\}$: A vocabulary, each element v_k being a word type.
- $\mathbf{x} = \langle x_1, \ldots, x_X \rangle$: Input features to a model.
- $\mathbf{y} = \{y_1, \ldots, y_N\}$: The output classes of a model, each element y_i being a class label.
- $\mathbf{T} = \langle \mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(T)} \rangle$: The training corpus of texts.
- $\mathbf{Y} = \langle y^{(1)}, \ldots, y^{(T)} \rangle$: The corresponding gold labels for the training examples in T.

To Do: Establish a majority baseline

A simple baseline for classification tasks is to always predict the most common class. Given a training set of texts $\mathbf{T}$ labeled by classes $\mathbf{Y}$, we classify an input text $\mathbf{w} = w_1 \cdots w_M$ as the class y_i that occurs most frequently in the training data, that is, specified by

$$\operatorname*{argmax}_i \sharp(y_i)$$

and thus ignoring the input entirely (!).

Implement the majority baseline and compute test accuracy using the starter code below. For this baseline, and for the naive Bayes classifier later, we don't need to use the validation set since we don't tune any hyper-parameters.

In [15]:
```
# TODO
def majority_baseline_accuracy(train_iter, test_iter):
    """Returns the most common label in the training set, and the accuracy of
     the majority baseline on the test set.
    """
    labelCounter = {}
    for label in LABEL.vocab.itos:
        labelCounter[label] = 0
    for example in train_iter.dataset:
        labelCounter[example.label] += 1
    most_common_label = max(labelCounter, key = labelCounter.get)

    accuracyCount = 0
    for example in test_iter.dataset:
        if(example.label == most_common_label):
            accuracyCount += 1
    test_accuracy = accuracyCount/len(test_iter.dataset)

    return most_common_label, test_accuracy
```

How well does your classifier work? Let's see:

In [16]:
```
# Call the method to establish a baseline
most_common_label, test_accuracy = majority_baseline_accuracy(train_iter, test_i

print(f'Most common label: {most_common_label}\n'
      f'Test accuracy:     {test_accuracy:.3f}')
```

```
Most common label: flight_id
Test accuracy:     0.683
```

To Do: Implement a Naive Bayes classifier

Review of the naive Bayes method

Recall from lab 1-3 that the Naive Bayes classification method classifies a text $\mathbf{w} = \langle w_1, w_2, \ldots, w_M \rangle$ as the class y_i given by the following maximization:

$$\operatorname*{argmax}_i \Pr(y_i \mid \mathbf{w}) \approx \operatorname*{argmax}_i \Pr(y_i) \cdot \prod_{j=1}^{M} \Pr(w_j \mid y_i)$$

or equivalently (since taking the log is monotonic)

$$\begin{aligned}\operatorname*{argmax}_i \Pr(y_i \mid \mathbf{w}) &= \operatorname*{argmax}_i \log \Pr(y_i \mid \mathbf{w}) && (1)\\ &\approx \operatorname*{argmax}_i \left(\log \Pr(y_i) + \sum_{j=1}^{M} \log \Pr(w_j \mid y_i) \right) && (2)\end{aligned}$$

All we need, then, to apply the Naive Bayes classification method is values for the various log probabilities: the priors $\log \Pr(y_i)$ and the likelihoods $\log \Pr(w_j \mid y_i)$, for each feature (word) w_j and each class y_i.

We can estimate the prior probabilities $\Pr(y_i)$ by examining the empirical probability in the training set. That is, we estimate

$$\Pr(y_i) \approx \frac{\sharp(y_i)}{\sum_j \sharp(y_j)}$$

We can estimate the likelihood probabilities $\Pr(w_j \mid y_i)$ similarly by examining the empirical probability in the training set. That is, we estimate

$$\Pr(w_j \mid y_i) \approx \frac{\sharp(w_j, y_i)}{\sum_{j'} \sharp(w_{j'}, y_i)}$$

To allow for cases in which the count $\sharp(w_j, y_i)$ is zero, we can use a modified estimate incorporating add-δ smoothing:

$$\Pr(w_j \mid y_i) \approx \frac{\sharp(w_j, y_i) + \delta}{\sum_{j'} \sharp(w_{j'}, y_i) + \delta \cdot V}$$

Two conceptions of the naive Bayes method implementation

We can store all of these parameters in different ways, leading to two different implementation conceptions. We review two conceptions of implementing the naive Bayes classification of a text $\mathbf{w} = \langle w_1, w_2, \ldots, w_M \rangle$, corresponding to using different representations of the input $\mathbf{x}$ to the model: the index representation and the bag-of-words representation.

Within each conception, the parameters of the model will be stored in one or more matrices. The conception dictates what operations will be performed with these matrices.

Using the index representation

In the first conception, we take the input elements $\mathbf{x} = \langle x_1, x_2, \ldots, x_M \rangle$ to be the *vocabulary indices* of the words $\mathbf{w} = w_1 \cdots w_M$. That is, each word token w_i is of the word type in the vocabulary $\mathbf{v}$ at index x_i, so

$$v_{x_i} = w_i$$

In this representation, the input vector has the same length as the word sequence.

We think of the likelihood probabilities as forming a matrix, call it $\mathbf{L}$, where the i, j-th element stores $\log \Pr(v_j \mid y_i)$.

$$\mathbf{L}_{ij} = \log \Pr(v_j \mid y_i)$$

Similarly, for the priors, we'll have

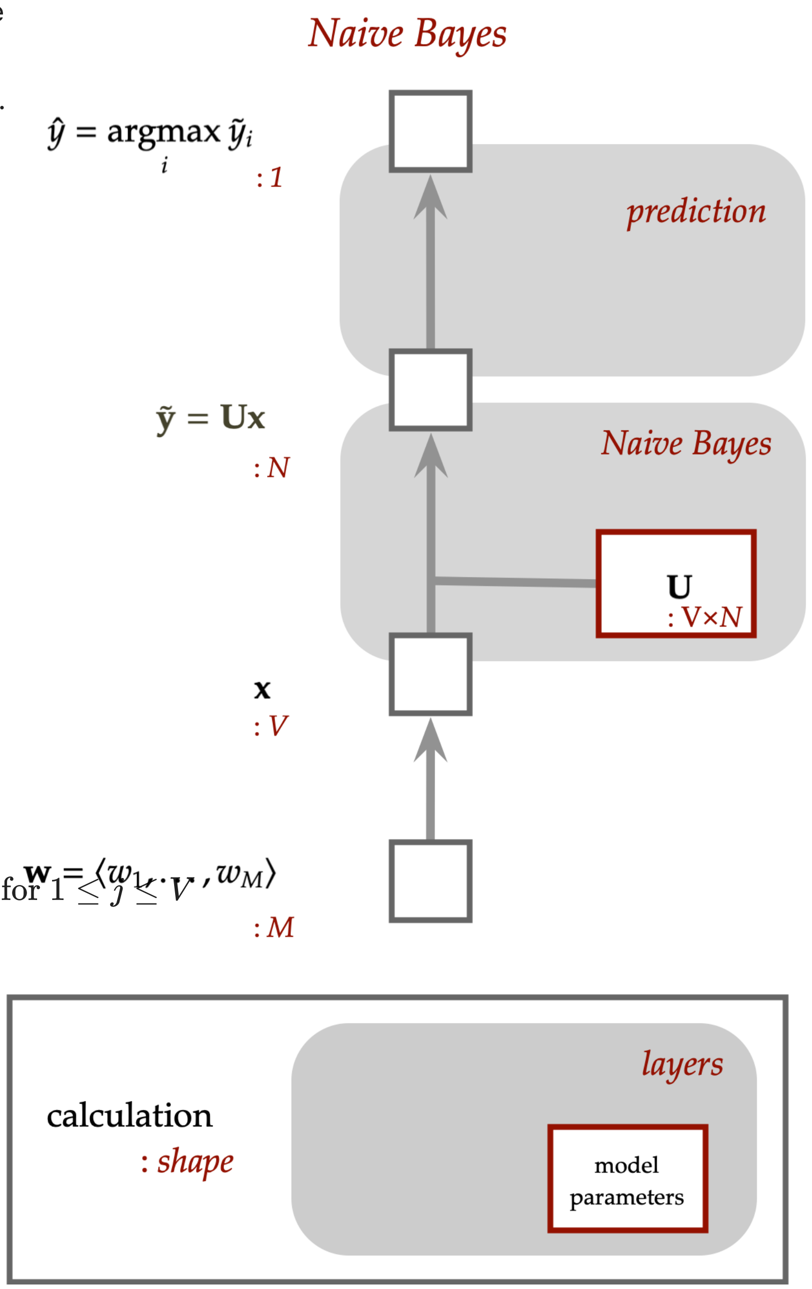
$$\mathbf{P}_i = \log \Pr(y_i)$$

Now the maximization can be implemented as

$$\operatorname*{argmax}_i \log \Pr(y_i) + \sum_{j=1}^{M} \log \Pr(w_j \mid y_i) = \operatorname*{argmax}_i \mathbf{P}_i + \sum_{j=1}^{M} \mathbf{L}_{x_j i} \tag{3}$$

Implemented in this way, we see that the use of each input x_i is as an *index* into the likelihood matrix.

Using the bag-of-words representation


Naive Bayes
$\hat{y} = \operatorname*{argmax}_i \tilde{y}_i$
: 1
prediction
$\tilde{\mathbf{y}} = \mathbf{U}\mathbf{x}$
: N
Naive Bayes
U
: V×N
x
: V
$\mathbf{w} = \langle w_1, \ldots, w_M \rangle$
: M
calculation
: shape
layers
model parameters

Notice that since each word in the input is treated separately, the order of the words doesn't matter. Rather, all that matters is how frequently each word type occurs in a text. Consequently, we can use the bag-of-words representation introduced in lab 1-1.

Recall that the bag-of-words representation of a text is just its frequency distribution over the vocabulary, which we will notate $bow(\mathbf{w})$. Given a vocabulary of word types $\mathbf{v} = \langle v_1, v_2, \ldots, v_V \rangle$, the representation of a sentence $\mathbf{w} = \langle w_1, w_2, \ldots, w_M \rangle$ is a vector $\mathbf{x}$ of size V, where

$$bow(\mathbf{w})_j = \sum_{i=1}^{M} 1[w_i = v_j] \qquad \text{for } 1 \leq j \leq V$$

We write $1[w_i = v_j]$ to indicate 1 if $w_i = v_j$ and 0 otherwise. For convenience, we'll add an extra $(V+1)$-st element to the end of the bag-of-words vector, a single 1 whose use will be clear shortly. That is,

$$bow(\mathbf{w})_{V+1} = 1$$

Under this conception, then, we'll take the input $\mathbf{x}$ to be $bow(\mathbf{w})$. Instead of the input having the same length as the text, it has the same length as the vocabulary.

As described in lecture, represented in this way, the quantity to be maximized in the naive Bayes method

$$\log \Pr(y_i) + \sum_{j=1}^{M} \log \Pr(w_j \mid y_i)$$

can be calculated as

$$\log \Pr(y_i) + \sum_{j=1}^{V} x_j \cdot \log \Pr(v_j \mid y_i)$$

which is just $\mathbf{Ux}$ for a suitable choice of $N \times (V+1)$ matrix $\mathbf{U}$, namely

$$\mathbf{U}_{ij} = \begin{cases} \log \Pr(v_j \mid y_i) & 1 \leq i \leq N \text{ and } 1 \leq j \leq V \\ \log \Pr(y_i) & 1 \leq i \leq N \text{ and } j = V+1 \end{cases}$$

Under this implementation conception, we've reduced naive Bayes calculations to a single matrix operation. This conception is depicted in the figure at right.

You are free to use either conception in your implementation of naive Bayes.

Implement the naive Bayes classifier

For the implementation, we ask you to implement a Python class `NaiveBayes` that will have (at least) the following three methods:

1. `__init__` : An initializer that takes two `torchtext` fields providing descriptions of the text and label aspects of examples.

2. `train` : A method that takes a training data iterator and estimates all of the log probabilities $\log \Pr(c_i)$ and $\log \Pr(x_j \mid c_i)$ as described above. Perform add-δ smoothing with $\delta = 1$. These parameters will be used by the `evaluate` method to evaluate a test dataset for accuracy, so you'll want to store them in some data structures in objects of the class.

3. `evaluate` : A method that takes a test data iterator and evaluates the accuracy of the trained model on the test set.

You can organize your code using either of the conceptions of Naive Bayes described above.

You should expect to achieve about an **86% test accuracy** on the ATIS task.

In [17]:
```python
def bagOfWords(text,sizeV,padding_id):
        c = Counter(text)
        bow = [0 for i in range(sizeV)]
        for j in range(sizeV):
```

```
        for k in c:
            if torch.tensor(j) == k:
                bow[j] += c[k]
    bow[padding_id] = 0
    return bow
```

In [18]:

```
class NaiveBayes():
    def __init__ (self, text, label):
        self.text = text
        self.label = label
        self.padding_id = text.vocab.stoi[text.pad_token]
        self.V = len(text.vocab.itos) # vocabulary size
        self.N = len(label.vocab.itos) # the number of classes
        # TODO: Add your code here
        # self.L = len(text)
        self.U = torch.zeros(self.N, self.V+1)


    def train(self, iterator):
        """Calculates and stores log probabilities for training dataset `iterato
        # TODO: Implement this method.
        labelList = [example.label for example in iterator.dataset]
        total = len(labelList)
        #print(total)
        c = Counter(labelList)
        print(c)
        for sentence in iterator.dataset:
            labelIndex = self.label.vocab.stoi[sentence.label]
            for word in sentence.text:
                wordIndex = self.text.vocab.stoi[word]
                self.U[labelIndex][wordIndex] += 1
        #print(self.U)
        countArr = []
        for i in range(self.U.size()[0]):
            count = 0
            for j in range(len(self.U[i])):
                count += self.U[i][j]
            countArr.append(count)
        print(countArr)

        for i in range(self.U.size()[0]):
            for j in range(len(self.U[i]) - 1):
                #print(countArr[i].item())
                self.U[i][j] = math.log2((self.U[i][j] + 1)/(countArr[i].item()
            self.U[i][self.V] = math.log2(c[LABEL.vocab.itos[i]]/total)
        print(self.U)




    def evaluate(self, iterator):
        """Returns the model's accuracy on a given dataset `iterator`."""
        # TODO: Implement this method.
        evalIterator = iter(iterator)
        numTotal = 0
        numCorrect = 0
        while True:
            try:
                batch = next(evalIterator)
                label = batch.label
                text = batch.text

                for i in range(len(label)):
                    bow = bagOfWords(text[i], self.V, self.padding_id)
                    bow.append(1)
                    y = torch.matmul(self.U, torch.tensor(bow, dtype = torch.flo
                    gold_label = label[i]
                    numTotal += 1
                    if gold_label == torch.argmax(y):
                        numCorrect += 1
            except StopIteration:
                break
        accuracy = numCorrect/numTotal
        return accuracy
```

In [19]:

```
# for word in train_iter.dataset:
#     print(word.text)
#     print(word.label)
# totalCount = 0
# for sentence in train_iter.dataset:
#     #print(sentence.label)
#     totalCount += len(sentence.text)
# sentenceList = [example.text for example in train_iter.dataset]
# #print(len(sentenceList))
# #print(sentenceList)
# labelList = [example.label for example in train_iter.dataset]
# #print(len(labelList))

# labelName = LABEL.vocab.stoi['flight_id']
# print(labelName)
# print(LABEL.vocab.stoi['time_elapsed'])
# print(TEXT.vocab.stoi['your'])
# print(TEXT.vocab.itos[2])
# print(LABEL.vocab.itos[0])
# print(U.size()[1])

#print(totalCount)


nb_classifier = NaiveBayes(TEXT, LABEL)
nb_classifier.train(train_iter)

# for example in test_iter.dataset:
#     print(example.text)

# Evaluate model performance
print(f'Training accuracy: {nb_classifier.evaluate(train_iter):.3f}\n'
      f'Test accuracy: {nb_classifier.evaluate(test_iter):.3f}')
```

```
Counter({'flight_id': 3210, 'fare_id': 344, 'transport_type': 228, 'airline_cod
e': 171, 'aircraft_code': 82, 'departure_time': 52, 'fare_basis_code': 43, 'airp
ort_code': 39, 'count': 38, 'state_code': 29, 'booking_class': 24, 'ground_far
e': 19, 'restriction_code': 18, 'miles_distant': 11, 'arrival_time': 11, 'city_c
ode': 10, 'meal_code': 10, 'meal_description': 5, 'basic_type': 5, 'minutes_dist
ant': 5, 'flight_number': 5, 'advance_purchase': 5, 'time_elapsed': 3, 'airport_
location': 3, 'day_name': 2, 'stop_airport': 2, 'stops': 2, 'minimum_connect_tim
e': 1, 'time_zone_code': 1, 'city_name': 1})
[tensor(37635.), tensor(4211.), tensor(1964.), tensor(1461.), tensor(1144.), ten
sor(714.), tensor(270.), tensor(182.), tensor(408.), tensor(378.), tensor(167.),
tensor(196.), tensor(103.), tensor(134.), tensor(112.), tensor(90.), tensor(9
2.), tensor(20.), tensor(63.), tensor(159.), tensor(36.), tensor(67.), tensor(1
5.), tensor(37.), tensor(6.), tensor(19.), tensor(31.), tensor(3.), tensor(9.),
tensor(6.)]
tensor([[ -7.0746, -15.2193,  -3.6287,  ..., -13.6343, -13.6343,  -0.4480],
        [ -6.8836, -12.2055,  -3.8089,  ..., -11.2055, -12.2055,  -3.6701],
        [ -6.9519, -11.2738,  -5.5734,  ..., -11.2738, -11.2738,  -4.2635],
        ...,
        [ -9.0084,  -9.0084,  -9.0084,  ...,  -9.0084,  -9.0084, -12.0964],
        [ -7.0251,  -9.0251,  -9.0251,  ...,  -9.0251,  -9.0251, -12.0964],
        [ -8.0168,  -9.0168,  -9.0168,  ...,  -9.0168,  -9.0168, -12.0964]])
Training accuracy: 0.897
Test accuracy: 0.864
```

In [ ]:

```

```

Implement the logistic regression classifier

For the implementation, we ask you to implement a logistic regression classifier as a subclass of `torch.nn.Module`. You need to implement the following methods:

1. `__init__`: an initializer that takes two `torchtext` fields providing descriptions of the text and label aspects of examples.

   During initialization, you'll want to define a tensor of weights, wrapped in `torch.nn.Parameter`, initialized randomly, which plays the role of $\mathbf{U}$. The elements of this tensor are the parameters of the `torch.nn` instance in the following special technical sense: It is the parameters of the module whose gradients will be calculated and whose values will be updated. Alternatively, you might find it easier to use the `nn.Embedding` module which is a wrapper to the weight tensor with a lookup implementation.

2. `forward`: given a text batch of size `batch_size X max_length`, return a tensor of logits of size `batch_size X num_labels`. That is, for each text $\mathbf{x}$ in the batch and each label y, you'll be calculating $\mathbf{Ux}$ as shown in the figure, returning a tensor of these values. Note that the softmax operation is absorbed into `nn.CrossEntropyLoss` so you won't need to deal with that.

3. `train_all`: A method that performs training. You might find lab 1-5 useful.

4. `evaluate`: A method that takes a test data iterator and evaluates the accuracy of the trained model on the test set.

Some things to consider:

1. The parameters of the model, the weights, need to be initialized properly. We suggest initializing them to some small random values. See `torch.uniform_`.

2. You'll want to make sure that padding tokens are handled properly. What should the weight be for the padding token?

3. In extracting the proper weights to sum up, based on the word types in a sentence, we are essentially doing a lookup operation. You might find `nn.Embedding` or `torch.gather` useful.

You should expect to achieve about **90%** accuracy on the ATIS classificiation task.

In [20]:
```
class LogisticRegression(nn.Module):
    def __init__ (self, text, label):
        super().__init__()
        self.text = text
        self.label = label
        self.padding_id = text.vocab.stoi[text.pad_token]
        # Keep the vocabulary sizes available
        self.N = len(label.vocab.itos) # num_classes
        self.V = len(text.vocab.itos)  # vocab_size
        # Specify cross-entropy loss for optimization
        self.criterion = nn.CrossEntropyLoss()
        # TODO: Create and initialize a tensor for the weights,
        #       or create an nn.Embedding module and initialize
        ...
        self.U = torch.nn.Parameter(torch.Tensor(self.N,self.V).uniform_(0,1))
        #self.embedding = nn.Embedding(self.N,self.V, padding_idx = self.padding
        #self.embedding.weight.data.uniform_(0,1)
        #self.linear = nn.Linear(self.V,self.N)
        #self.linear.weight.data.uniform_(0,1)
        #self.U = nn.Linear()

        #self.U = nn.Embedding(torch.Tensor(self.N,self.V).uniform_)

    def forward(self, text_batch):
    # TODO: Calculate the logits (Ux) for the `text_batch`,
    #       returning a tensor of size batch_size x num_labels
        ret = torch.zeros([len(text_batch),self.V])
        for i, sentence in enumerate(text_batch.text):
            bow = torch.zeros(self.V)
            for wordIndex in sentence:
                bow[wordIndex] += 1
            ret[i] = bow

        retVal = torch.matmul(ret,self.U.T)
        return retVal

    def train_all(self, train_iter, val_iter, epochs=8, learning_rate=3e-3):
    # Switch the module to training mode
        self.train()
        # Use Adam to optimize the parameters
        optim = torch.optim.Adam(self.parameters(), lr=learning_rate)
        best_validation_accuracy = -float('inf')
        best_model = None
        # Run the optimization for multiple epochs
        with tqdm(range(epochs), desc='train', position=0) as pbar:
            for epoch in pbar:
                c_num = 0
                total = 0
                running_loss = 0.0
                for batch in tqdm(train_iter, desc='batch', leave=False):
                    # TODO: set labels, compute logits (Ux in this model),
                    #       loss, and update parameters
                    ...
                    optim.zero_grad()
                    labels = batch.label
                    logits = self.forward(batch)
                    loss = self.criterion(logits,labels)
                    loss.backward()
                    optim.step()
                    ...
                    # Prepare to compute the accuracy
                    predictions = torch.argmax(logits, dim=1)
                    total += predictions.size(0)
                    c_num += (predictions == labels).float().sum().item()
                    running_loss += loss.item() * predictions.size(0)

                    # Evaluate and track improvements on the validation dataset
                    validation_accuracy = self.evaluate(val_iter)
                    if validation_accuracy > best_validation_accuracy:
                        best_validation_accuracy = validation_accuracy
                        self.best_model = copy.deepcopy(self.state_dict())
                        epoch_loss = running_loss / total
                        epoch_acc = c_num / total
                        pbar.set_postfix(epoch=epoch+1, loss=epoch_loss, train_a

    def evaluate(self, iterator):
        """Returns the model's accuracy on a given dataset `iterator`."""
        self.eval()   # switch the module to evaluation mode
        # TODO: Compute accuracy
        with torch.no_grad():
            numCorrect = 0
            numTotal = len(iterator.dataset)
            for sentence in iterator.dataset:
                bow = torch.zeros(self.V)
                for word in sentence.text:
                    bow[self.text.vocab.stoi[word]] += 1
                label = torch.argmax(torch.matmul(self.U,bow))
                if sentence.label == self.label.vocab.itos[label]:
                    numCorrect += 1

        return numCorrect / numTotal
```

In [21]:

```
model = LogisticRegression(TEXT, LABEL).to(device)
model.forward(batch)
```

Out[21]:

```
tensor([[ 8.2163, 15.5570,  8.8837,  8.6198, 15.6998,  9.0532,  4.1633,  9.7062,
         15.9774,  4.3654,  3.2959,  9.4384, 15.9067, 14.5179, 15.2123,  3.2148,
         14.7468, 15.9950, 10.5336,  7.1896,  7.4921, 10.5424, 13.5207,  5.5149,
          8.0534, 12.3653,  7.4518, 15.4852, 14.6244,  4.0875],
        [ 8.7474, 13.1596,  8.4309,  9.3893, 13.2008,  9.9894,  6.6734,  9.3189,
         14.1458,  8.0941,  6.4428,  9.4998, 14.0991, 10.9224, 12.3683,  6.1700,
         12.4263, 13.9541,  9.1823,  8.0108,  6.7542,  9.7368, 11.2559,  6.9367,
          8.7493, 11.2465,  9.8955, 13.5798, 13.1740,  9.2232],
        [ 9.7426, 12.9181,  9.9068, 10.3123, 14.7843,  9.6891,  6.2082, 11.0457,
         13.5136,  8.2502,  5.8746,  8.6225, 15.6152, 12.6178, 13.8781,  4.7702,
         12.4727, 14.7312, 10.0066,  7.5826,  7.6166, 11.2500, 10.4801,  7.6911,
          8.4950, 12.1162,  8.3370, 12.8192, 13.9425,  6.0688],
        [10.6882,  8.7890, 11.5213,  8.0070, 11.9126, 10.7590,  8.7202, 11.8735,
          7.2378, 11.0607,  8.1603,  8.9124,  9.4721,  9.2161, 10.7438,  8.4787,
         10.9692,  9.7742,  9.9156,  8.5933,  6.5196, 10.3650, 12.9091, 10.0637,
         13.1775,  9.5516, 10.0587, 11.2524, 12.3116, 12.8139],
        [ 9.9611, 14.1209, 10.0615,  9.1170, 16.3884,  8.9260,  4.8320,  8.1773,
         17.0124,  4.6206,  2.0188,  9.4344, 18.0289, 15.3061, 15.0416,  3.2809,
         13.8508, 17.9446, 12.6259,  7.0086,  7.4800, 10.3604, 12.2016,  6.4708,
          6.6081, 13.2147,  7.0716, 17.5058, 14.3247,  4.2703],
        [10.1677, 11.4172, 10.4830, 11.0726, 14.8133,  9.4854,  7.9345, 10.1890,
         12.4322,  7.6966,  5.8819,  8.8292, 13.3069,  8.9394, 12.2904,  5.4918,
         11.4506, 14.4572, 10.7791,  9.4656, 10.1353,  8.9651, 11.3274,  8.5702,
          8.6428, 12.1936,  7.6933, 13.5203, 13.7674,  6.6426],
        [ 9.3468, 14.7363,  9.3452,  7.6328, 14.4903,  9.7729,  6.3645, 10.8399,
         14.7623,  7.0622,  5.4532, 10.4070, 16.6219, 13.9118, 13.2481,  4.4682,
         12.6227, 16.5611, 10.7575,  9.1490,  8.6687,  9.9697, 11.4991,  7.5637,
          8.3193, 11.4018,  6.6519, 15.0224, 14.4234,  5.5884],
        [ 9.0085, 13.2247,  9.8896,  7.5171, 13.0350,  9.0105,  4.9528,  9.9456,
         15.4908,  6.9975,  4.5090, 10.9186, 13.4453, 12.2805, 12.9462,  4.3459,
         12.9709, 16.9837,  9.7485,  8.6590,  8.4027, 11.2723, 10.9786,  8.4443,
          8.5047, 10.6005,  9.1674, 14.8051, 12.4650,  7.6385],
        [ 9.2876, 13.6940,  9.3635,  9.6121, 15.5782,  8.9896,  4.3768,  9.9393,
         14.4328,  7.9796,  4.4255, 10.2923, 14.1697, 13.0691, 15.3244,  4.4122,
         14.5712, 15.1463, 10.2042,  5.5394,  8.1176, 10.9741, 13.2380,  7.1613,
          9.7282, 11.6559,  9.8939, 14.5798, 14.7485,  4.8456],
        [ 8.2191, 10.9360, 11.8178, 10.3342, 13.4709, 12.0143,  8.4833, 10.1191,
         10.8220, 11.7112,  7.9486, 10.6435, 11.0457,  9.9091, 11.2157,  9.9096,
         13.5504, 12.6099, 10.0412,  7.1773, 11.2489, 10.5331, 13.2725,  8.9684,
         10.7115, 10.5705,  8.7397, 12.3384, 11.2215,  9.5261],
        [ 9.4359, 12.8356,  9.8798,  8.9680, 15.0127,  9.8047,  6.1780, 11.0346,
         14.2423,  6.6206,  3.8423,  9.4795, 14.8269, 13.0073, 12.6766,  3.1088,
         12.7721, 14.0140, 10.9502,  9.6014,  6.9190,  8.4275, 11.9244,  7.5496,
          7.4082, 12.2570,  9.0602, 14.1736, 12.3852,  7.5570],
        [ 8.7376, 14.3141,  9.6395,  9.1512, 16.0617,  9.5763,  4.1724, 11.3121,
         15.1856,  5.2813,  3.4490,  9.4538, 17.2412, 13.3788, 14.6760,  3.7962,
         12.9921, 15.9107, 11.7759,  7.1569,  6.5850, 10.9185, 12.2303,  6.5321,
          8.3054, 13.4666,  6.7557, 15.5245, 13.9355,  5.3700],
        [ 8.2125, 13.6270, 10.0267,  9.5332, 13.1635,  9.7863,  5.0731, 10.2364,
         13.9030,  7.0311,  4.9611,  9.4223, 14.4755, 12.4870, 13.5516,  4.1165,
         12.3210, 14.8642,  8.8555,  7.9537,  7.8365, 10.3616, 12.4078,  7.1653,
          8.7921,  9.7511,  8.0283, 14.6658, 13.9302,  9.5977],
        [ 8.3694, 13.1105, 10.3854,  9.1567, 13.9802, 10.1938,  6.5206, 10.1164,
         14.6739,  7.4206,  4.4857,  7.7383, 16.4697, 13.1149, 14.2016,  3.8916,
         13.7837, 14.9768, 10.0534,  7.7129,  7.9441, 10.5174, 10.3817,  6.9799,
          8.6285, 12.0786,  8.6331, 15.2260, 13.1852,  6.0917],
        [ 9.1188, 13.9915, 10.9663,  9.2085, 15.2638, 10.2048,  6.7321,  9.1297,
         15.1257,  6.2735,  2.8960,  9.2061, 15.6819, 13.0723, 14.9646,  4.6511,
         13.6543, 15.3624, 10.9304,  8.8254,  6.9840, 10.0681, 11.2288,  7.9831,
          7.6897, 13.0712,  7.8000, 15.4189, 11.2552,  6.6875],
        [ 8.8714, 13.6364, 10.3837,  8.7747, 14.6250, 10.5099,  8.9074, 10.2727,
         12.4497,  6.8208,  6.4878,  9.5735, 13.1062, 13.7962, 14.5710,  5.5261,
         13.1994, 14.4324,  8.4637,  9.4017,  8.6282, 10.1419, 11.1165,  8.1643,
         10.8134, 11.4798,  9.2827, 12.6526, 13.5388,  8.6858],
        [10.5191, 14.4149, 10.2493,  8.6135, 14.3228,  9.8774,  4.8147,  9.7221,
         12.4343,  5.6042,  5.1220,  7.8906, 14.6528, 13.4503, 14.3565,  5.9961,
         12.0441, 14.2489,  9.1877,  8.3129,  5.9862,  9.5138, 10.8359,  8.2965,
          8.5893, 11.4491, 10.3151, 14.2779, 14.8497,  7.4824],
        [ 8.2590, 13.3010,  8.6160,  7.7839, 12.7974, 10.1897,  4.7226, 12.1197,
         14.8610,  6.4079,  4.0116, 12.4752, 15.4108, 12.5029, 13.6307,  5.0996,
         13.0181, 14.4542, 11.8609,  8.4094,  7.8504, 10.2850, 11.9100,  6.9591,
          6.6596, 11.8168,  8.6168, 13.8280, 13.5112,  5.1687],
        [ 8.8840, 15.4006,  9.2966,  7.8405, 16.5047,  8.7713,  3.0242,  9.3388,
         18.9458,  3.4491,  1.0257,  9.1489, 19.9277, 16.6334, 15.6813,  1.8208,
         14.8078, 18.8620, 11.5550,  6.4934,  6.9030,  7.8479, 11.1943,  4.3977,
          5.6943, 13.2890,  6.2084, 16.4351, 13.8821,  2.6708],
        [ 8.8765, 14.4660, 10.1246,  9.1157, 16.5966,  9.7413,  3.6155, 10.0764,
         17.0374,  5.4196,  3.5506,  7.5855, 17.7812, 14.9007, 14.4300,  3.0302,
         14.3812, 16.8244, 10.4008,  7.2479,  6.6629,  9.1338, 11.2778,  4.5905,
          6.2798, 12.2169,  7.7808, 16.5274, 13.9567,  5.2375],
        [10.8378, 11.3048,  9.6094, 10.6490, 12.8355, 10.9825,  6.8835,  9.8791,
         11.3466, 11.6661,  8.0685,  9.3624, 11.5333,  9.9464, 13.2338,  8.0976,
         12.3268, 10.4502, 11.5078,  8.6632,  9.8103, 11.0056, 13.0263,  8.4564,
         12.3371, 10.9287,  9.6994, 10.0210, 11.7424,  8.3920],
        [ 9.2330, 14.0603,  9.0161,  8.8186, 14.7513,  9.1048,  4.9825, 10.1278,
         15.6724,  5.4875,  4.8804,  9.8726, 16.9713, 11.9019, 13.7679,  4.2176,
         13.6047, 14.9137, 10.3765,  6.3331,  7.8323, 10.7234, 12.5283,  6.6376,
          8.4296, 12.3052,  8.5650, 15.1772, 13.7513,  5.3858],
        [ 9.7591, 14.1657,  8.4665,  9.6959, 14.0381,  7.7633,  5.4138,  9.3155,
         15.3867,  6.1389,  4.1072,  9.5859, 13.9308, 12.7991, 13.0722,  3.5383,
         12.2141, 14.5215, 10.6448,  8.1421,  8.7306,  9.4343, 12.4051,  6.5282,
          8.4338, 11.7687,  8.9635, 15.3781, 13.2078,  6.4001],
        [ 8.8898, 14.1618,  9.8352,  9.6118, 15.4378, 10.5221,  4.7656, 10.9376,
         14.3463,  6.2769,  3.8632,  9.4432, 16.9118, 11.8769, 14.1986,  3.5408,
         13.3373, 15.1120, 12.4522,  8.5253,  7.4686, 10.2621, 11.5600,  7.9161,
          9.5133, 12.3628,  7.8750, 15.4308, 12.6893,  6.3403],
        [ 9.9207, 13.4140, 10.8314,  8.8412, 12.8155,  8.1419,  7.1984,  9.4684,
         13.2746,  8.2588,  7.3683,  9.4084, 13.8243, 13.2387, 13.1847,  3.5105,
         12.2951, 14.5254, 10.6045,  8.0307,  7.9929, 10.7654, 11.7688,  8.6474,
          9.5767, 10.3128,  8.8280, 14.7221, 15.9045,  8.1797],
        [ 9.4832, 12.8108, 10.6771,  9.0299, 14.8113,  9.5072,  7.2549, 10.4794,
         12.5926,  6.6196,  4.5149,  9.8403, 15.2118, 12.6586, 14.2457,  3.9787,
         13.7151, 17.0319, 10.1327,  7.0150,  9.0348, 11.5867, 11.5970,  8.0336,
          9.1356, 11.3479,  9.0935, 14.2064, 15.0366,  7.8805],
        [ 9.8066, 13.3220,  8.6778,  7.5006, 14.5280,  9.7117,  4.9009,  9.8739,
         17.0453,  6.2675,  3.1059,  9.3305, 17.3702, 14.5244, 14.6382,  3.8795,
         12.1655, 16.3593, 11.4839,  8.7795,  5.9519,  9.3040, 11.5096,  4.7269,
          7.3230, 12.2226,  7.0033, 13.8213, 13.4694,  5.5493],
        [ 9.7218, 14.8967,  9.8045,  8.3300, 15.4171,  9.1065,  3.9607, 10.0608,
         15.7484,  4.5546,  3.2559,  9.4166, 15.3904, 13.2353, 14.6587,  3.3196,
         13.5684, 15.1742,  9.7364,  7.2240,  7.2773, 10.3865, 13.1196,  6.7661,
          9.1055, 12.3973,  8.5450, 14.3907, 15.2030,  5.5204],
        [ 9.0490, 12.5031, 10.6117, 10.7026, 15.3611,  8.2944,  5.4274,  9.9519,
         13.5066,  5.8184,  5.7906,  7.7163, 14.8494, 13.4542, 14.7517,  4.6954,
         14.4354, 14.3137, 10.2560,  6.7700,  8.3587,  9.9091, 11.8853,  7.7821,
          8.5936, 12.6030,  7.9789, 14.4699, 15.1003,  5.5110],
        [ 8.7004, 15.3505,  9.1155,  8.2349, 14.8805,  8.6129,  4.2162,  9.6519,
         16.2787,  4.0647,  2.8511,  9.5694, 16.4254, 14.3618, 14.3899,  3.1572,
         14.6450, 17.1423, 12.3638,  7.6996,  7.9265,  9.7305, 11.4614,  5.8863,
          6.6476, 11.2143,  8.3627, 15.0830, 13.1785,  3.8446],
        [ 8.2578, 13.2873,  8.7114, 10.3409, 15.4128,  9.3052,  5.9974,  9.2287,
         14.8226,  6.7615,  3.5054,  9.7094, 16.5918, 12.2393, 12.7162,  3.6964,
         13.2161, 16.3058, 11.1748,  8.1324,  8.1809,  9.6978, 12.9148,  6.4758,
          8.2691, 11.0883,  8.4649, 16.0384, 13.7080,  4.6779],
        [ 9.6900, 12.6010, 11.3850,  9.1744, 12.1166,  9.5996,  6.9852,  9.8281,
         11.6045,  8.6488,  5.8703,  9.4458, 11.2788, 10.8413, 14.0149,  6.1412,
         13.0673, 13.4686,  9.2553,  7.5468,  8.1053, 12.5553, 12.2589,  9.4940,
         11.0303, 10.0231, 10.6666, 12.3469, 13.6329,  8.3364]],
       grad_fn=<MmBackward>)
```

In [22]:
```
# Instantiate the logistic regression classifier and run it
N = len(LABEL.vocab.itos)
V = len(TEXT.vocab.itos)
```

```
x = torch.nn.Parameter(torch.Tensor(N,V).uniform_(0,1))
#print(torch.mm(x,x.T))
model = LogisticRegression(TEXT, LABEL).to(device)
model.train_all(train_iter, val_iter)
model.load_state_dict(model.best_model)
test_accuracy = model.evaluate(test_iter)
print (f'Test accuracy: {test_accuracy:.4f}')
```

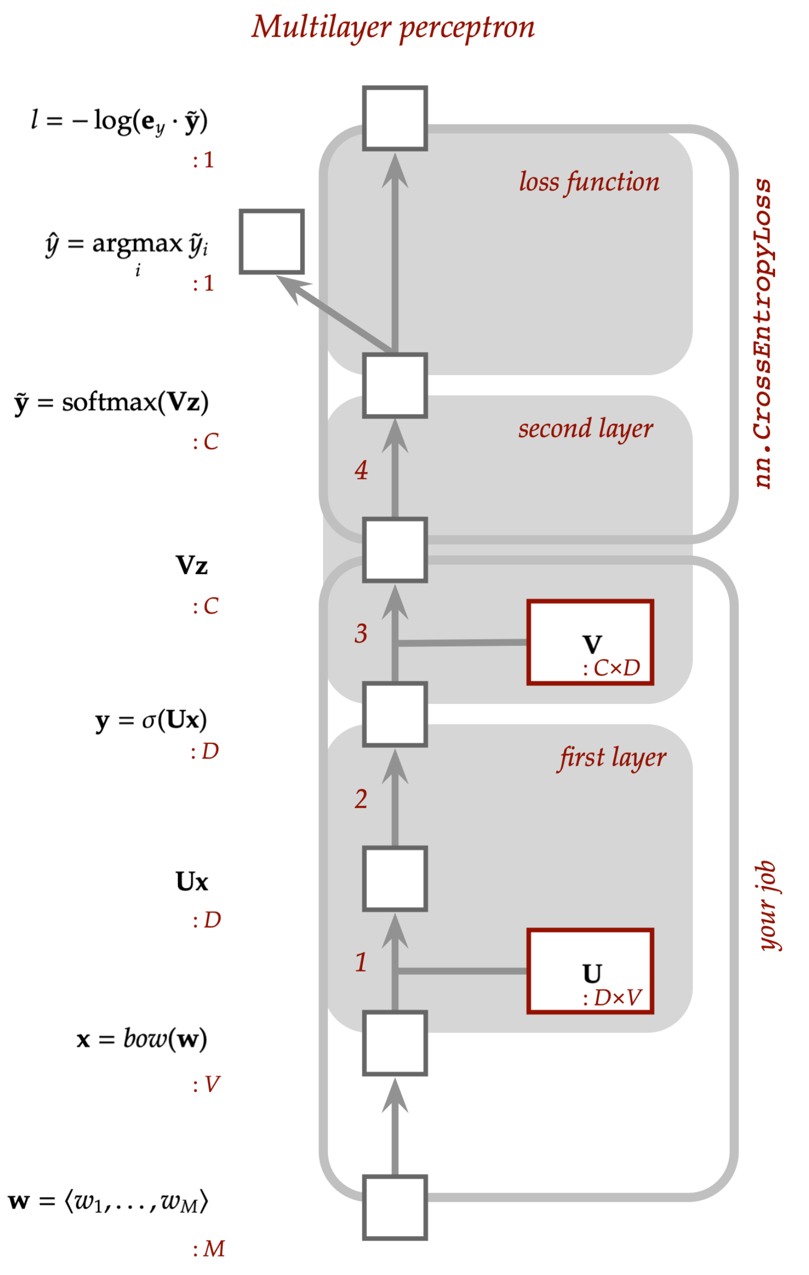
```
Test accuracy: 0.9085
```

To Do: Implement a multilayer perceptron

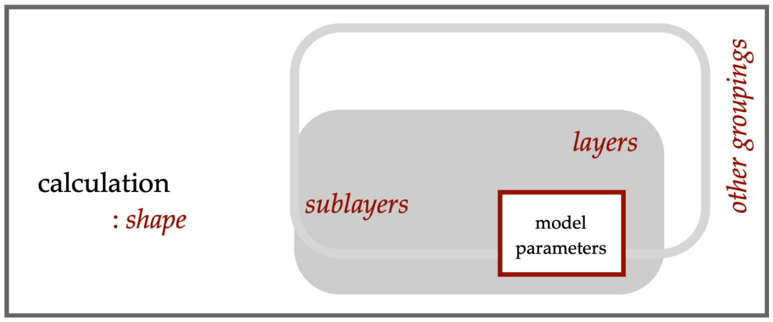
Review of multilayer perceptrons

In the last part, you implemented a perceptron, a model that involved a linear calculation (the sum of weights) followed by a nonlinear calculation (the softmax, which converts the summed weight values to probabilities). In a multi-layer perceptron, we take the output of the first perceptron to be the input of a second perceptron (and of course, we could continue on with a third or even more).

In this part, you'll implement the forward calculation of a two-layer perceptron, again letting PyTorch handle the backward calculation as well as the optimization of parameters. The first layer will involve a linear summation as before and a **sigmoid** as the nonlinear function. The second will involve a linear summation and a softmax (the latter absorbed, as before, into the loss function). Thus, the difference from the logistic regression implementation is simply the adding of the sigmoid and second linear calculations. See the figure for the structure of the computation.



Multilayer perceptron
$l = -\log(\mathbf{e}_y \cdot \tilde{\mathbf{y}})$
$: 1$
$\hat{y} = \operatorname{argmax}_i \tilde{y}_i$
$: 1$
$\tilde{\mathbf{y}} = \operatorname{softmax}(\mathbf{Vz})$
$: C$
$\mathbf{Vz}$
$: C$
$\mathbf{y} = \sigma(\mathbf{Ux})$
$: D$
$\mathbf{Ux}$
$: D$
$\mathbf{x} = bow(\mathbf{w})$
$: V$
$\mathbf{w} = \langle w_1, \ldots, w_M \rangle$
$: M$
loss function
second layer
first layer
nn.CrossEntropyLoss
your job
1
2
3
4
$\mathbf{V}$ $: C \times D$
$\mathbf{U}$ $: D \times V$



Implement a multilayer perceptron classifier

For the implementation, we ask you to implement a two layer perceptron classifier, again as a subclass of the `torch.nn` module. You might reuse quite a lot of the code from logistic regression. As before, you need to implement the following methods:

1. `__init__` : An initializer that takes two `torchtext` fields providing descriptions of the text and label aspects of examples, and `hidden_size` specifying the size of the hidden layer (e.g., in the above illustration, `hidden_size` is `D`).

   During initialization, you'll want to define two tensors of weights, which serve as the parameters of this model, one for each layer. You'll want to initialize them randomly.

   The weights in the first layer are a kind of lookup (as in the previous part), mapping words to a vector of size `hidden_size` . The `nn.Embedding` module is a good way to set up and make use of this weight tensor.

   The weights in the second layer define a linear mapping from vectors of size `hidden_size` to vectors of size `num_labels` . The `nn.Linear` module or `torch.mm` for matrix multiplication may be helpful here.

2. `forward` : Given a text batch of size `batch_size X max_length` , the `forward` function returns a tensor of logits of size `batch_size X num_labels` .

   That is, for each text $\mathbf{x}$ in the batch and each label c, you'll be calculating $MLP(bow(\mathbf{x}))$ as shown in the illustration above, returning a tensor of these values. Note that the softmax operation is absorbed into `nn.CrossEntropyLoss` so you don't need to worry about that.

   For the sigmoid sublayer, you might find `nn.Sigmoid` useful.

3. `train_all` : A method that performs training. You might find lab 1-5 useful.

4. `evaluate` : A method that takes a test data iterator and evaluates the accuracy of the trained model on the test set.

You should expect to achieve at least **90%** accuracy on the ATIS classificiation task.

In [23]:
```
class MultiLayerPerceptron(nn.Module):
    def __init__ (self, text, label, hidden_size=128):
        super().__init__ ()
        self.text = text
        self.label = label
        self.padding_id = text.vocab.stoi[text.pad_token]
        self.hidden_size = hidden_size
        # Keep the vocabulary sizes available
        self.N = len(label.vocab.itos) # num_classes
        self.V = len(text.vocab.itos)  # vocab_size
        # Specify cross-entropy loss for optimization
        self.criterion = nn.CrossEntropyLoss()
        # TODO: Create and initialize neural modules
        self.hidden1 = torch.nn.Embedding(self.V, hidden_size)
        self.hidden1.weight.data.uniform_(-0.05,0.05)
        self.hidden2 = nn.Sigmoid()
        self.hidden3 = nn.Linear(hidden_size, self.N)

    def forward(self, text_batch):
        # TODO: Calculate the logits for the `text_batch`,
        #       returning a tensor of size batch_size x num_labels
        embedding = self.hidden1(torch.tensor(text_batch)).sum(axis = 1)
        finalHidden = self.hidden2(embedding)
        return self.hidden3(finalHidden)

    def train_all(self, train_iter, val_iter, epochs=8, learning_rate=3e-3):
        # Switch the module to training mode
        self.train()
        # Use Adam to optimize the parameters
        optim = torch.optim.Adam(self.parameters(), lr=learning_rate)
        best_validation_accuracy = -float('inf')
        best_model = None
        # Run the optimization for multiple epochs
        with tqdm(range(epochs), desc='train', position=0) as pbar:
            for epoch in pbar:
                c_num = 0
                total = 0
                running_loss = 0.0
                for batch in tqdm(train_iter, desc='batch', leave=False):
                    # TODO: set labels, compute logits (Ux in this model),
                    #       loss, and update parameters
                    ...
                    optim.zero_grad()
                    labels = batch.label
                    logits = self.forward(batch.text)
                    loss = self.criterion(logits,labels)
                    loss.backward()
                    optim.step()
                    ...
                    # Prepare to compute the accuracy
                    predictions = torch.argmax(logits, dim=1)
                    total += predictions.size(0)
                    c_num += (predictions == labels).float().sum().item()
                    running_loss += loss.item() * predictions.size(0)

                # Evaluate and track improvements on the validation dataset
                validation_accuracy = self.evaluate(val_iter)
                if validation_accuracy > best_validation_accuracy:
                    best_validation_accuracy = validation_accuracy
                    self.best_model = copy.deepcopy(self.state_dict())
                    epoch_loss = running_loss / total
                    epoch_acc = c_num / total
                    pbar.set_postfix(epoch=epoch+1, loss=epoch_loss, train_acc =

    def evaluate(self, iterator):
        """Returns the model's accuracy on a given dataset `iterator`."""
        # TODO: Compute accuracy
        evalIterator = iter(iterator)
        numTotal = 0
        numCorrect = 0

        for batch in evalIterator:
            model = self.forward(batch.text)

            for i, sentence in enumerate(batch):
                correctLabel = batch.label[i]
                label = torch.argmax(model[i])
                if correctLabel == label:
                    numCorrect += 1
                numTotal += 1
        return numCorrect / numTotal
```

In [24]:
```
# Instantiate classifier and run it
model = MultiLayerPerceptron(TEXT, LABEL).to(device)
model.train_all(train_iter, val_iter)
model.load_state_dict(model.best_model)
test_accuracy = model.evaluate(test_iter)
print (f'Test accuracy: {test_accuracy:.4f}')
```

```
/var/folders/09/v1n7m90x4x19d0b5zn46swr80000gn/T/ipykernel_9298/1581480568.py:2
2: UserWarning: To copy construct from a tensor, it is recommended to use source
Tensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), r
ather than torch.tensor(sourceTensor).
  embedding = self.hidden1(torch.tensor(text_batch)).sum(axis = 1)
```

```
Test accuracy: 0.9286
```

Lessons learned

Take a look at some of the examples that were classified correctly and incorrectly by your best method.

Question: Do you notice anything about the incorrectly classified examples that might indicate *why* they were classified incorrectly?

The incorrectly classified examples require a lot more information to be gathered and understood in order to make a correct prediction on the class label. This was a trend that was present in a lot of the incorrectly classified examples because they were not only longer in length but also included additional required information to a lot of the simpler queries. For simple and short queries, the accuracy of the classification was pretty high. However, for longer and more complex queries the accuracy became a lot lower and it was more likely that it would be classified incorrectly in terms of the label.

In [25]:
```
...
```

Out[25]:
```
Ellipsis
```

Debrief

Question: We're interested in any thoughts you have about this project segment so that we can improve it for later years, and to inform later segments for this year. Please list any issues that arose or comments you have to improve the project segment. Useful things to comment on include the following:

- Was the project segment clear or unclear? Which portions?
- Were the readings appropriate background for the project segment?
- Are there additions or changes you think would make the project segment better?

I thought that the necessary computation for the evaluation functions was unclear along with the definition of why to use nn.Linear or nn.Embedding. I think there was some reading that was missing and going to section and office hours was necessary to do this problem set and would have been very hard to do without it. I think more information about each of the functions that are being used throughout the project would make the project segments a lot more accessible and easier.

Instructions for submission of the project segment

This project segment should be submitted to Gradescope at http://go.cs187.info/project1-submit, which will be made available some time before the due date.

Project segment notebooks are manually graded, not autograded using otter as labs are. (Otter is used within project segment notebooks to synchronize distribution and solution code however.) **We will not run your notebook before grading it.** Instead, we ask that you submit the already freshly run notebook. The best method is to "restart kernel and run all cells", allowing time for all cells to be run to completion.

We also request that you **submit a PDF of the freshly run notebook**. The simplest method is to use "Export notebook to PDF", which will render the notebook to PDF via LaTeX. If that doesn't work, the method that seems to be most reliable is to export the notebook as HTML (if you are using Jupyter Notebook, you can do so using `File -> Print Preview`), open the HTML in a browser, and print it to a file. Then make sure to add the file to your git commit. Please name the file the same name as this notebook, but with a `.pdf` extension. (Conveniently, the methods just described will use that name by default.) You can then perform a git commit and push and submit the commit to Gradescope.

End of project segment 1 {-}

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In [ ]:

```

```