In this homework you will learn:

```
- Forward propagation of a CNN network
- Backward propagation of a CNN network
- Numerical gradient checking
- Use Keras and TensorFlow to implement more complex CNN networks
```

In [1]:
```
from tools import load_data, read_vocab, sigmoid, tanh, show_model
```

```
Using TensorFlow backend.
```

In [97]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import WordNetLemmatizer
from nltk import WordNetLemmatizer, word_tokenize,download
from tools import load_data, save_prediction
```

In [2]:
```
import sklearn.utils
print(sklearn.__version__)
sklearn.utils.__all__
```

```
0.20.0
```

Out[2]:
```
['murmurhash3_32',
 'as_float_array',
 'assert_all_finite',
 'check_array',
 'check_random_state',
 'compute_class_weight',
 'compute_sample_weight',
 'column_or_1d',
 'safe_indexing',
 'check_consistent_length',
 'check_X_y',
 'indexable',
 'check_symmetric',
 'indices_to_mask',
 'deprecated',
 'cpu_count',
 'Parallel',
 'Memory',
 'delayed',
 'parallel_backend',
 'register_parallel_backend',
 'hash',
 'effective_n_jobs']
```

CNN model

Complete the code block in the cells in this section.

- step1: Implement the pipeline method to process the raw input

- step2: Implement the forward method
- step3: Implement the backward method
- step4: Run the cell below to train your model

In [94]:

```
"""
This cell shows you how the model will be used, you have to finish the cell belo
can run this cell.

Once the implementation is done, you should hype tune the parameters to find the
"""
from sklearn.model_selection import train_test_split
data = load_data("train.txt")
vocab = read_vocab("vocab.txt")

X, y = data.text, data.target
X_train, X_dev, y_train, y_dev = train_test_split(X, y, test_size=0.3)
cls = CNNTextClassificationModel(vocab)
cls.train(X_train, y_train, X_dev, y_dev, nEpoch=10)
```

```
Epoch: 0        Train accuracy: 0.657   Dev accuracy: 0.622
Epoch: 1        Train accuracy: 0.786   Dev accuracy: 0.679
Epoch: 2        Train accuracy: 0.837   Dev accuracy: 0.693
Epoch: 3        Train accuracy: 0.931   Dev accuracy: 0.720
Epoch: 4        Train accuracy: 0.956   Dev accuracy: 0.711
Epoch: 5        Train accuracy: 0.971   Dev accuracy: 0.722
Epoch: 6        Train accuracy: 0.983   Dev accuracy: 0.735
Epoch: 7        Train accuracy: 0.991   Dev accuracy: 0.737
Epoch: 8        Train accuracy: 0.998   Dev accuracy: 0.738

C:\Users\Chandler\.conda\envs\tf\lib\site-packages\ipykernel_launcher.py:176: R
untimeWarning: divide by zero encountered in log
C:\Users\Chandler\.conda\envs\tf\lib\site-packages\ipykernel_launcher.py:176: R
untimeWarning: invalid value encountered in multiply

Epoch: 9        Train accuracy: 0.999   Dev accuracy: 0.739
```

In [11]:

```
def window(iterable, size=2):
    """
    Sliding Window
    """
    i = iter(iterable)
    win = []
    for e in range(0, size):
        win.append(next(i))
    yield win
    for e in i:
        win = win[1:] + [e]
        yield win
```

In [93]:
```
import numpy as np

class CNNTextClassificationModel:
    def __init__(self, vocab, window_size=2, F=100, alpha=0.1):
        """
        F: number of filters
        alpha: back propagatoin learning rate
        """
        self.vocab = vocab
        self.window_size = window_size
        self.F = F
        self.alpha = alpha

        # U and w are the weights of the hidden layer, see Fig 1 in the pdf file
        # U is the 1D convolutional layer with shape: voc_size * num_filter * wi
        self.U = np.random.normal(loc=0, scale=0.01, size=(len(vocab), F, window_
        # w is the weights of the activation layer (after max pooling)
        self.w = np.random.normal(loc=0, scale=0.01, size=(F + 1))

    def pipeline(self, X):
        """
        Data processing pipeline to:
        1. Tokenize, Normalize the raw input
        2. Translate raw data input into numerical encoded vectors

        :param X: raw data input
        :return: list of lists

        For example:
        X = ["Apples orange banana",
         "orange apple bananas"]
        returns:
        [[0, 1, 2,
         1, 0, 2]]
        """

        """
        Implement your code here
        """
        X2 = []
        unknown = vocab['__unknown__']
        default = vocab['.']
        wnet = WordNetLemmatizer()

        for i in range(len(X)):
            cleaned_tokens = [self.vocab.get(wnet.lemmatize(w), unknown) for w i
            if len(cleaned_tokens) < self.window_size:
                cleaned_tokens = cleaned_tokens + [default] * (self.window_size
            X2.append(cleaned_tokens)

        return X2

    @staticmethod
    def accuracy(probs, labels):
        assert len(probs) == len(labels), "Wrong input!!"
        a = np.array(probs)
```

```
        b = np.array(labels)

        return 1.0 * (a==b).sum() / len(b)

    def train(self, X_train, y_train, X_dev, y_dev, nEpoch=50):
        """
        Function to fit the model
        :param X_train, X_dev: raw data input
        :param y_train, y_dev: label
        :nEpoch: number of training epoches
        """
        X_train = self.pipeline(X_train)
        X_dev = self.pipeline(X_dev)

        for epoch in range(nEpoch):
            self.fit(X_train, y_train)

            accuracy_train = self.accuracy(self.predict(X_train), y_train)
            accuracy_dev = self.accuracy(self.predict(X_dev), y_dev)

            print("Epoch: {}\tTrain accuracy: {:.3f}\tDev accuracy: {:.3f}"
                  .format(epoch, accuracy_train, accuracy_dev))

    def fit(self, X, y):
        """
        :param X: numerical encoded input
        """
        for (data, label) in zip(X, y):
            self.backward(data, label)

        return self

    def predict(self, X):
        """
        :param X: numerical encoded input
        """
        result = []
        for data in X:
            if self.forward(data)["prob"] > 0.5:
                result.append(1)
            else:
                result.append(0)

        return result

    def forward(self, word_indices):
        """
        :param word_indices: a list of numerically ecoded words
        :return: a result dictionary containing 3 items -
        result['prob']: \hat y in Fig 1.
        result['h']: the hidden layer output after max pooling, h = [h1, ..., hf
        result['hid']: argmax of F filters, e.g. j of x_j
        e.g. for the ith filter u_i, tanh(word[hid[j], hid[j] + width]*u_i) = h_i
        """

        assert len(word_indices) >= self.window_size, "Input length cannot be sho
```

```
        h = np.zeros(self.F + 1, dtype=float)
        hid = np.zeros(self.F, dtype=int)
        prob = 0.0



        # layer 1. compute h and hid
        # loop through the input data of word indices and
        # keep track of the max filtered value h_i and its position index x_j
        # h_i = max(tanh(weighted sum of all words in a given window)) over all w
        """
        Implement your code here
        """
        for filterIndex in range(self.F):
            uxList = []
            for xIndex in range(len(word_indices)-self.window_size+1):
                uxSum = 0.0
                for windowIndex in range(self.window_size):
                    uxSum += self.U[word_indices[xIndex + windowIndex]][filterIn
                uxList.append(tanh(uxSum))
            h[filterIndex] = np.max(uxList)
            hid[filterIndex] = np.argmax(uxList)
        h[-1] = 1


        # layer 2. compute probability
        # once h and hid are computed, compute the probabiliy by sigmoid(h^TV)
        """
        Implement your code here
        """
        prob_sum = 0.0
        for w_i, h_i in zip(self.w, h):
            prob_sum += w_i * h_i

        prob = sigmoid(prob_sum)
        # return result
        return {"prob": prob, "h": h, "hid": hid}

    def backward(self, word_indices, label):
        """
        Update the U, w using backward propagation

        :param word_indices: a list of numerically ecoded words
        :param label: int 0 or 1
        :return: None

        update weight matrix/vector U and V based on the loss function
        """

        pred = self.forward(word_indices)
        prob = pred["prob"]
        h = pred["h"]
        hid = pred["hid"]


        # update U and w here
        # to update V: w_new = w_current + d(loss_function)/d(w)*alpha
```

```
        # to update U: U_new = U_current + d(loss_function)/d(U)*alpha
        # Hint: use Q6 in the first part of your homework
        """
        Implement your code here
        """
        L = -(label) * np.log(prob) - (1 - label) * np.log(1 - prob)
        #print(word_indices[0])
        old_w = self.w[:]
        self.w = self.w + (label - prob) * h * self.alpha
        for filterIndex in range(self.F):
            incre = (label - prob) * old_w[filterIndex] * (1 - h[filterIndex] **
            for i in range(self.window_size):
                self.U[word_indices[hid[filterIndex] + i]][filterIndex][i] += in
```

In [111]:

```
"""
Run this cell to save weights and the prediction
"""
X_test = load_data("test.txt").text
save_prediction(cls.predict(cls.pipeline(X_test)),filename="CNN.csv")
```

Optional: Build your model using Keras + Tensorflow

So far we have always forced you to implement things from scratch. You may feel it's overwhelming, but fortunately, it is not how the real world works. In the real world, there are existing tools you can leverage, so you can focus on the most innovative part of your work. We asked you to do all the previous execises for learning purpose, and since you have already reached so far, it's time to unleash yourself and allow you the access to the real world toolings.

Sample model

In [ ]:

```
# First let's see how you can build a similar CNN model you just had using Keras
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

MAX_LENGTH = 100
```

In [ ]:
```
# Yes! it is a good practice to do data processing outside the ML model
wnet = WordNetLemmatizer()
# Numerical encode all the words
unknown = vocab['__unknown__']
X_train2 = [[vocab.get(wnet.lemmatize(w), unknown) for w in word_tokenize(sent)]
X_dev2 = [[vocab.get(wnet.lemmatize(w), unknown)for w in word_tokenize(sent)] for

# Tensorflow does not handle variable length input well, let's unify all input to
def trim_X(X, max_length=100, default=vocab['.']):
    for i in range(len(X)):
        if len(X[i]) > max_length:
            X[i] = X[i][:max_length]
        elif len(X[i]) < max_length:
            X[i] = X[i] + [default] * (max_length - len(X[i]))

    return np.array(X)

X_train2 = trim_X(X_train2, MAX_LENGTH)
X_dev2 = trim_X(X_dev2, MAX_LENGTH)


# Now we have all the input data nicely encoded with numerical label, and each of
# to have the same length. We would have needed to further apply one-hot encode f
# would be very expensive, since each word will be expanded into a len(vocab) (~1
# not support sparse matrix input at this moment. But don't worry, we will use an
# layer. This concept will be introduced in the next lesson. At this moment, you
```

In [ ]:
```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Dense, GlobalMaxPooling

model = Sequential()
model.add(Embedding(input_dim=len(vocab), input_length=MAX_LENGTH, output_dim=512
model.add(Conv1D(filters=100, kernel_size=2, activation="tanh", name="Conv1D-1"))
model.add(GlobalMaxPooling1D(name="MaxPooling1D-1"))
model.add(Dense(1, activation="sigmoid", name="Dense-1"))
print(model.summary())

show_model(model)
```

In [ ]:
```
# Train the model
model.compile(loss="binary_crossentropy", optimizer='adam', metrics=['accuracy']
model.fit(X_train2, y_train, epochs=10, validation_data=[X_dev2, y_dev])
```

Try your own model

We have shown you have to use an industry level tool to build a CNN model. Hopefully you think it is simpler than the version we built from scratch. Not really? Read Keras Documentation and learn more: [https://keras.io/ (https://keras.io/)](https://keras.io/)

Now it's your turn to build some more complicated CNN models

""" Implement your code here """

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ (http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/)

https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/ (https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)

https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/ (https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/)