.:: به نام خدا::.

تکلیف سری چهارم درس یادگیری ژرف

نام-نام خانوادگی

شماره دانشجویی

در قسمت اول این گزارش ، هدف، مشخص کردن نقش کلمات در جمله است و علاوه بر آن در شرایطی که مجموعه ای از کلمات یک نقش داشته باشند کلمه اول با پیشوند B و بقیه با پیشوند I مشخص گردند. ایده‌ای که در اینجا در نظر گرفته شده است این است که جملات به صورت دنباله‌ای از واژگان به شبکه داده شوند و خروجی برچسب هر واژه می‌باشد. در این صورت کل دنباله ی ورودی و خروجی هم بعد میگردد. هم چنین د راین قسمت، طول کل جملات داده های آموزشی را هم طول کرده (طول همه ی جملات را برابر با بزرگترین جمله داده اموزشی قرار داده و به عنوان ورودی داده اموزشی به شبکه میدهیم.) و باید توجه شود که هنگام تخمین جمله جدید الزامی نداریم که طول دنباله جدید هم اندازه با دنباله‌های آموزشی باشد.

In [2]:
```
import numpy as np
import keras
from keras.models import Sequential
from keras.models import Sequential, Model
from keras.layers import Dense
from keras.layers import LSTM, RNN, GRU, SimpleRNN, TimeDistributed, Activat
ion, Embedding, RepeatVector, Bidirectional
from keras.optimizers import RMSprop
from keras.utils.np_utils import to_categorical
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
```

```
Using TensorFlow backend.
```

ابتدا لازم است که دیتاهای آموزشی لود شوند. بدین منظور به جهت کم تر کردن نیاز به دیتاهای آموزشی بیشتر و از آن جا که حروف بزرگ و کوچک، تفاوتی در معنا و تشخیص نقش کلمه ایجاد نمی کند تمام حروف را به حروف کوچک تبدیل کرده و به عنوان ورودی به شبکه وارد میکنیم. هم چنین، طبق الگوی دیتاهای آموزشی داده شده، پایان هر جمله را با دستور "". . O" مشخص مینماییم.

In [3]:
```
# load data:
data = open('./train.data', 'r').read().lower()
# convert data into sentences:
sentences = data.split('. . o\n\n')[0:-1]
len(sentences)
```

Out[3]: 72

مشخص است که کل جملات داده های train یا آموزشی به تعداد 72 جمله است.

In [4]:
```
sentences[0]
```

Out[4]:
```
"confidence nn b-np\nin in b-pp\nthe dt b-np\npound nn i-np\nis vbz b-vp\nwid
ely rb i-vp\nexpected vbn i-vp\nto to i-vp\ntake vb i-vp\nanother dt b-np\nsh
arp jj i-np\ndive nn i-np\nif in b-sbar\ntrade nn b-np\nfigures nns i-np\nfor
in b-pp\nseptember nnp b-np\n, , o\ndue jj b-adjp\nfor in b-pp\nrelease nn b-
np\ntomorrow nn b-np\n, , o\nfail vb b-vp\nto to i-vp\nshow vb i-vp\na dt b-n
p\nsubstantial jj i-np\nimprovement nn i-np\nfrom in b-pp\njuly nnp b-np\nand
cc i-np\naugust nnp i-np\n's pos b-np\nnear-record jj i-np\ndeficits nns i-n
p\n"
```

ابتدا 4 لیست تعریف میگردد. به این صورت که X_data برای کلمات جمله Y_data به عنوان لیبل هر کلمه. words به منظور ذخیره کلمات و نیز متغیر chunks به منظور sort . و ذخیره کردن لیبل ها انتخاب میگردند. سپس کلمات و نقش های انان از یکدیگر جدا گشته و در متغیرهای مربوطه ذخیره میگردند. هم چنین ، همانگونه که در ابتدا بیان گردید کلمه دوم یک عبارت با توجه به الگوی منظمی که وجود دارد نیاز به ذخیره سازی ندارد.

In [21]:
```
X_data = []  # container for sentences
Y_data = []  # container for labels
words = []  # this is our vocabulary
chunks = [] # this is our chunk labels
for i in range(len(sentences)):
    x = [] # temporary containers
    y = []
    # each line in a sentence contain 3 parts: 1) word, 2) speech label  and
3) chunck label:
    line = sentences[i].splitlines()
    for j in range(len(line)):
        # some lines contain just one space, that doesn't follow any rule we
defined.
        # so those lines simple ignored.
        if len(line[j].split(' ')) == 3:
            w, _, c = line[j].split(' ')
            x.append(w)
            y.append(c)
            words.append(w)
            chunks.append(c)
    # add '.' after the last word of sentence, and add 'o' as last label for
'.':
    X_data.append(' '.join(x)+'.')
    Y_data.append(' '.join(y)+' o')
```

In [22]:
```
X_data[0]
```

Out[22]:
```
"confidence in the pound is widely expected to take another sharp dive if tra
de figures for september , due for release tomorrow , fail to show a substant
ial improvement from july and august 's near-record deficits."
```

In [23]:
```
Y_data[0]
```

Out[23]:
```
'b-np b-pp b-np i-np b-vp i-vp i-vp i-vp i-vp b-np i-np i-np b-sbar b-np i-np
b-pp b-np o b-adjp b-pp b-np b-np o b-vp i-vp i-vp b-np i-np i-np b-pp b-np i
-np i-np b-np i-np i-np o'
```

عبارت \n به عنوان padding و عبارت پایان دهنده جمله اضافه شده اند.

In [24]:
```
chars = list(set(''.join(words)))
chars.sort()
chars = [' '] + chars
# create targets (label)
label = list(set(chunks))
label.sort()
label = ['\n'] + label
```

In [26]:
```
print(label)
```

```
['\n', 'b-adjp', 'b-advp', 'b-np', 'b-pp', 'b-prt', 'b-sbar', 'b-vp', 'i-adjp
', 'i-advp', 'i-np', 'i-pp', 'i-sbar', 'i-vp', 'o']
```

برچسب‌های خروجی برای شبکه قابل فهم نیستند و لازم است به صورت داده‌های دسته بندی (categorical) تبدیل شوند. برای این کار ابتدا دنباله خروجی به دنباله‌ای از اعداد و سپس به بردارهای one-hot coded نگاشت می‌شوند. در مرحله اول این کار با استفاده از تابع زیر انجام می‌شود:

In [29]:
```
def label_encode(l):
    return [label.index(i) for i in l.split(' ')] + [0]

print(label_encode(Y_data[0]))
```

```
[3, 4, 3, 10, 7, 13, 13, 13, 13, 3, 10, 10, 6, 3, 10, 4, 3, 14, 1, 4, 3, 3,
14, 7, 13, 13, 3, 10, 10, 4, 3, 10, 10, 3, 10, 10, 14, 0]
```

در سلول زیر لیستی از شامل دنباله‌هایی از برچسب‌های خروجی برای جملات داده‌های آموزش تولید می‌شود.

In [30]:
```
Y = [label_encode(Y_data[i]) for i in range(len(Y_data))]
```

همچنین برای آماده سازی داده‌های آموزش نیاز داریم که جملات را از نظر واژگان هم طول نماییم. به این منظور می‌بایست بیشترین طول جملات را داشته باشیم. در سلول زیر بیشترین طول محاسبه می‌شود:

In [31]:
```
maxlen = max([len(Y[i]) for i in range(len(Y))])
maxlen
```

Out[31]:
```
57
```

در ادامه رشته کدهای خروجی را به صورت one-hot vector تبدیل می‌کنیم. توجه شود که به انتهای جملات یک کد توقف یا padding اضافه شده است. برای تبدیل خروجی‌ها به دنباله one-hot vector از تابع آماده کتابخانه keras یعنی دستور to_categorical استفاده شده است.

In [32]:
```
y = np.zeros((len(Y), maxlen+1, len(label)))
for i in range(y.shape[0]):
    T = Y[i]
    for j in range(len(T)):
        y[i, j]= to_categorical(T[j], num_classes=len(label))
```

In [33]:
```
to_categorical(2, num_classes=5)
```

Out[33]:
```
array([0., 0., 1., 0., 0.], dtype=float32)
```

برای بررسی کدینگ درست برچسب‌های خروجی، در سلول زیر برای جمله اول خروجی به همراه کد padding نشان داده شده است.

In [14]:
```
y[0].argmax(-1)
```

Out[14]:
```
array([3, 5, 3, 3, 8, 8, 8, 8, 8, 3, 3, 3, 7, 3, 3, 5, 3, 4, 1, 5, 3, 3,
       4, 8, 8, 8, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

پس از آماده سازی داده‌های خروجی برای آموزش، حال نوبت به داده‌های ورودی می‌رسد. این که داده‌های ورودی به چه صورتی انجام شوند. این که چگونه داده‌های ورودی را آماده کنیم، تأثیر مهمی در معماری شبکه خواهد داشت. راهکارهای مختلفی که می‌تواند در اینجا مطرح شود دو حالت character-level و word-level می‌باشد. در حالت اول طول دنباله ورودی و خروجی مشابه نمی‌باشد. بنابراین معماری مورد استفاده حالت پیچیده ساختار «ماشین ترجمه» پیشنهاد می‌شود. اما در حالت دوم با توجه به این که دنباله ورودی و خروجی هم طول و برابر تعداد واژگان یک جمله هستند، بنابراین ساختار شبکه بسیار ساده‌تر خواهد بود. در اینجا ساختار دوم پیاده سازی شده است.

اما اگر بخواهیم که مدل را در سطح واژگان پیاده سازی کنیم، نیاز است که واژه‌های ورودی ابتدا به بردارهایی کد شوند. می‌توانیم در ابتدای مدل خود یک لایه برای word embedding قرار دهیم. اما به دلیل این که برای واژگان انگلیسی این کار پیشتر انجام شده است، ما از کدینگ آماده word2vec استفاده می‌کنیم. برای این منظور از کتابخانه پایتونی gensim و مدل عمومی از پیش آموزش دیده glove.6B توسط دانشگاه استنفورد استفاده می‌کنیم. و سپس مدل glove را به مدل word2vec تبدیل می‌نماییم.

In [34]:
```
glove_input_file = 'glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'
glove2word2vec(glove_input_file, word2vec_output_file)
```

Out[34]:
```
(400000, 100)
```

In [35]:
```
filename = 'glove.6B.100d.txt.word2vec'
w2v = KeyedVectors.load_word2vec_format(filename, binary=False )
```

در اینجا از بردارهای ۱۰۰ تایی برای کد استفاده کردیم. در زیر یک مثال از این بردارها نشان داده شده است.

In [36]:
```
w2v['while']
```

Out[36]:
```
array([ 0.094157 ,  0.46457  ,  0.4535   , -0.15074  ,  0.27223  ,
        0.4545   , -0.14906  ,  0.15345  , -0.061775 , -0.080787 ,
        0.53914  , -0.39179  ,  0.083668 , -0.10328  ,  0.27425  ,
       -0.80995  , -0.11588  , -0.32288  , -0.23434  ,  0.19782  ,
        0.47749  ,  0.027463 ,  0.49629  ,  0.41455  ,  0.55198  ,
        0.13814  , -0.14193  , -0.65181  , -0.055301 , -0.026074 ,
       -0.26557  ,  0.16076  , -0.32292  , -0.10203  ,  0.08234  ,
        0.13615  ,  0.27754  ,  0.19405  , -0.2348   , -0.12201  ,
       -0.39889  , -0.6782   ,  0.42633  ,  0.21963  , -0.20309  ,
        0.16836  ,  0.013425 , -0.35281  , -0.069011 , -0.93563  ,
        0.16361  , -0.13117  ,  0.099808 ,  1.8998   , -0.26605  ,
       -2.4321   , -0.34386  , -0.46084  ,  1.3691   ,  0.72702  ,
       -0.18504  ,  0.18016  ,  0.085648 ,  0.46807  ,  0.12802  ,
        0.28034  ,  0.68951  ,  0.36221  ,  0.66845  ,  0.32295  ,
       -0.58005  , -0.27069  ,  0.15057  , -0.46084  , -0.21336  ,
        0.36952  , -0.23539  ,  0.075712 , -0.71302  , -0.27551  ,
        0.64845  ,  0.10345  , -0.64706  ,  0.29101  , -1.4154   ,
       -0.31586  , -0.26086  ,  0.24959  , -0.20852  , -0.28688  ,
       -0.075658 , -0.63833  , -0.0040848,  0.21971  , -0.91796  ,
        0.271    , -0.30677  , -0.23741  ,  0.69147  , -0.16581  ],
      dtype=float32)
```

با توجه به این که مدل glove با داده‌های عمومی ویکیپدیا آموزش داده شده است، بنابراین الزاما دارای همه واژگان پایگاه داده ما نمی‌باشد. در این صورت برای واژگانی که در مدل word2vec تعریف نشده‌اند ما یک کد ناشناس که یک بردار ۱۰۰ تایی منفی یک است استفاده می‌کنیم. نکته‌ای که باید در نظر گرفته شود این است که این روش کد گذاری تأثیر منفی آنچنانی در خروجی کار نمی‌گذارد، برای این که نقش واژگان در یک جمله به صورت سلسه مراتبی قابل پیش‌بینی است و ارتباط مستقیمی به جنس آن واژه ندارد. حال اگر در میان واژگان جمله‌ای عبارت ناشناس باشد، باز نقش ان بر اساس نقش کلمات پیش و پس از آن قابل شناسایی است.

In [37]:
```
UKN = -np.ones((1,100))
```

در سلول زیر، داده‌های ورودی آموزشی با استفاده از word2vec کدگذاری می‌شوند. طول جملات نیز در اینجا مشابه برچسب‌ها یکی بیشتر از طول جمله‌ها می‌باشد و آخرین عبارت معرف کد پایان جمله یا همان \n می‌باشد.

In [45]:
```
E = []
x = np.zeros((len(X_data), maxlen+1, 100))
for i in range(len(X_data)):
    # each words splited by single space:
    T = X_data[i].split(' ')
    # T is the words of i-th sentence
    for j in range(len(T)):
        # here just checked if the w2v model contained the input word or no
t:
        try:
            # T[j] is j-th word of i-th sentence
            x[i, j]= w2v[T[j]]
        except:
            E.append(T[j])
            x[i, j] = UKN
```

In [68]:
```
print(E)
# Unknown words on the data set:
```

```
['deficits.', 'week.', 'thursday.', 'say.', 'dillow', 'institute.', '-lrb-',
'-rrb-', 'august.', '1988.', 'exports.', 'inflows.', 'september.', 'dillow',
'stockbuilding', 'imports.', 'billion.', 'positions.', 'pound.', 'clouded.',
'effect.', 'years.', 'quickly.', 'ago.', '1988.', 'dillow', 'rises.', 'furthe
r.', 'necessary.', 'hollow.', 'friday.', '1.5890', '2.9495', '1.5940',
'2.9429', 'thursday.', 'unchanged.', 'abated.', 'points.', 'reserve.',
'1.8578', '1.8470', 'york.', '142.43', '141.70', 'thursday.', '141.95',
'141.35', 'yen.', '367.30', 'cents.', 'ounces.', '366.50', 'ounce.', 'shar
e.', '31.', 'financing.', 'filing.', '40-a-share', 'million.', '2,664,098', '
outstanding.', 'trading.', 'products.', '%.', 'share.', 'boston.', 'inflatio
n.', 'air-freight', 'transport.', 'years.', 'abroad.', 'freight-transport', '
year.', 'freight-transport', 'years.', 'columbus.', 'transportation.', 'fil
l.', '1981.', 'freight-transport', 'rates.', 'lalonde.', 'transportation-cost
', 'years.', 'freight-cost', 'reductions.', 'freight-rate', 'labor.', 'shippe
rs.', 'technology.', 'less-than-truckload', 'april.', 'share.', 'railroad-rat
e', 'rail-traffic', 'trucks.', 'less-than-truckload', 'increases.']
```

In [128]:
```
x.shape
```

Out[128]: (72, 58, 100)

In [129]:
```
y.shape
```

Out[129]: (72, 58, 15)

با توجه به این که برخی از واژگان ورودی در مدل word2vec موجود نیستند، بنابراین برای پیشبینی نقش آن‌ها می‌توان از کلمات پیش و پس از آن‌ها استفاده کرد، برای این منظور مدلی که تعریف می‌کنیم بجای استفاده از شبکه‌های بازگشتی یک‌طرفه، از ساختار دوطرفه یا bidirectional استفاده می‌کند. در مدل زیر از لایه Lstm استفاده شده است که به راحتی می‌توان آن را با GRU برای لایه‌های gru و SimpleRNN برای لایه‌های Rnn ساده عوض کرد. ولی شبکه در حالت‌هایی که لایه‌ها از جنس GRU و LSTM هستند زودتر و بهتر آموزش می‌بیند.

In [63]:
```
hidden_size =  32
model = Sequential()
model.add(Bidirectional(LSTM(hidden_size, return_sequences=True), input_shap
e=(None, 100)))
model.add(Bidirectional(LSTM(hidden_size, return_sequences=True)))
```

In [64]:
```
num_layers = 1
for _ in range(num_layers):
    model.add(Bidirectional(LSTM(hidden_size, return_sequences=True)))
model.add(TimeDistributed(Dense(len(label))))
model.add(Activation('softmax'))
optimizer = RMSprop(decay=1e-1)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

در لایه آخر، با توجه به این که اندازه خروجی شبکه برابر با ورودی شبکه می‌باشد، بنابراین از تمام دنباله لایه بازگشتی به لایه fully-connected متصل شده است. در Keras به این صورت عمل می‌شود که در لایه بازگشتی پارامتر return_sequences فعال می‌شود و سپس با استفاده با یک لایه TimeDistributed دنباله شبکه به طور سلسله مراتبی به دسته بند منتقل می‌شود. خلاصه‌ای از جزییات معماری شبکه در زیر نشان داده شده است.

In [130]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
bidirectional_14 (Bidirectio (None, None, 64)          34048     
_________________________________________________________________
bidirectional_15 (Bidirectio (None, None, 64)          24832     
_________________________________________________________________
bidirectional_16 (Bidirectio (None, None, 64)          24832     
_________________________________________________________________
time_distributed_6 (TimeDist (None, None, 15)          975       
_________________________________________________________________
activation_6 (Activation)    (None, None, 15)          0         
=================================================================
Total params: 84,687
Trainable params: 84,687
Non-trainable params: 0
_________________________________________________________________
```

In [65]:
```
model.load_weights('lastWfixed')
```

در زیر داده‌های آموزش را به دو دسته آموزش و ارزیابی تقسیم می‌کنیم و با آن‌ها شبکه را آموزش می‌دهیم.

In [41]:
```
idx = np.random.permutation(x.shape[0])
train_idx = idx[0:64]
val_idx = idx[64::]
```

آموزش شبکه در Keras با استفاده از دستور fit انجام می‌شود. پس از گذشت ۲۰۰ دوره زمانی، وزن‌های شبکه آموزش می‌بینند.

In [66]:
```
model.fit(x[train_idx], y[train_idx], epochs=200, validation_data=(x[val_id
x], y[val_idx]))
model.save_weights('lastWfixed')
```

```
Train on 64 samples, validate on 8 samples
Epoch 1/200
64/64 [==============================] - 15s 240ms/step - loss: 0.3097 - acc:
0.9084 - val_loss: 0.2076 - val_acc: 0.9397
Epoch 2/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0931 - acc:
0.9817 - val_loss: 0.1937 - val_acc: 0.9483
Epoch 3/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0754 - acc:
0.9863 - val_loss: 0.1959 - val_acc: 0.9461
Epoch 4/200
64/64 [==============================] - 1s 13ms/step - loss: 0.0723 - acc:
0.9871 - val_loss: 0.1991 - val_acc: 0.9483
Epoch 5/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0702 - acc:
0.9876 - val_loss: 0.1991 - val_acc: 0.9440
Epoch 6/200
64/64 [==============================] - 1s 11ms/step - loss: 0.0686 - acc:
0.9884 - val_loss: 0.1974 - val_acc: 0.9461
Epoch 7/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0677 - acc:
0.9884 - val_loss: 0.1943 - val_acc: 0.9440
Epoch 8/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0674 - acc:
0.9881 - val_loss: 0.1942 - val_acc: 0.9440
Epoch 9/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0657 - acc:
0.9898 - val_loss: 0.1972 - val_acc: 0.9440
Epoch 10/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0645 - acc:
0.9887 - val_loss: 0.2003 - val_acc: 0.9440
Epoch 11/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0625 - acc:
0.9895 - val_loss: 0.2018 - val_acc: 0.9440
Epoch 12/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0613 - acc:
0.9906 - val_loss: 0.2051 - val_acc: 0.9461
Epoch 13/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0609 - acc:
0.9898 - val_loss: 0.2055 - val_acc: 0.9461
Epoch 14/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0623 - acc:
0.9895 - val_loss: 0.2075 - val_acc: 0.9461
Epoch 15/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0600 - acc:
0.9892 - val_loss: 0.2075 - val_acc: 0.9461
Epoch 16/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0584 - acc:
0.9903 - val_loss: 0.2073 - val_acc: 0.9483
Epoch 17/200
64/64 [==============================] - 1s 9ms/step - loss: 0.0621 - acc:
0.9873 - val_loss: 0.2131 - val_acc: 0.9483
Epoch 18/200
64/64 [==============================] - 1s 11ms/step - loss: 0.0677 - acc:
0.9844 - val_loss: 0.2039 - val_acc: 0.9461
Epoch 19/200
64/64 [==============================] - 1s 11ms/step - loss: 0.0589 - acc:
0.9895 - val_loss: 0.1956 - val_acc: 0.9461
Epoch 20/200
64/64 [==============================] - 1s 11ms/step - loss: 0.0538 - acc:
0.9908 - val_loss: 0.1912 - val_acc: 0.9440
Epoch 21/200
64/64 [==============================] - 1s 11ms/step - loss: 0.0534 - acc:
0.9908 - val_loss: 0.1881 - val_acc: 0.9483
Epoch 22/200
64/64 [==============================] - 1s 10ms/step - loss: 0.0558 - acc:
0.9900 - val_loss: 0.1884 - val_acc: 0.9461
Epoch 23/200
```

حال ابتدا نتیجه شبکه را روی یکی از داده‌های ارزیابی، بررسی می‌کنیم. در سلول زیر خروجی پیشبینی شده شبکه مشاهده می‌شود:

In [131]:
```
index = np.random.permutation(val_idx)[0]
l = model.predict(x[index:index+1])[0].argmax(-1)
l
```

Out[131]:
```
array([ 3,  4,  3, 10, 10, 10, 10, 14,  3, 10,  7, 13,  4,  3, 10, 14,  3,
        4,  3, 10,  3, 10,  4,  3, 10, 14,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0])
```

در اینجا نیز خروجی مطلوب به ازای ورودی قبلی نشان داده شده است:

In [132]:
```
y[index].argmax(-1)
```

Out[132]:
```
array([ 2,  4,  3, 10, 10, 10, 10, 14,  3, 10,  7, 13,  4,  3, 10, 14,  2,
        4,  3, 10,  3, 10,  4,  3, 10, 14,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0])
```

برچسب‌های پیش‌بینی شده تا رسیدن به عبارت توقف:

In [75]:
```
a = []
for i in range(l.shape[0]):
    c = label[l[i]]
    a.append(c)
    if c == '\n':
        break
print(a)
```

```
['b-pp', 'b-np', 'b-pp', 'b-np', 'o', 'b-np', 'i-np', 'i-np', 'b-vp', 'b-pp',
'b-np', 'b-pp', 'b-np', 'i-np', 'o', 'b-advp', 'b-pp', 'b-np', 'b-np', 'i-np
', 'i-np', 'b-pp', 'b-np', 'i-np', 'o', '\n']
```

برچسب‌های مطلوب:

In [76]:
```
a = []
for i in range(len(Y[index])):
    a.append(label[Y[index][i]])
print(a)
```

```
['b-pp', 'b-np', 'b-pp', 'b-np', 'o', 'b-np', 'i-np', 'i-np', 'b-vp', 'b-pp',
'b-np', 'b-pp', 'b-np', 'i-np', 'o', 'b-advp', 'b-pp', 'b-np', 'b-np', 'i-np
', 'i-np', 'b-pp', 'b-np', 'i-np', 'o', '\n']
```

عبارت ارزیابی ورودی:

In [133]:
```
X_data[index]
```

Out[133]:
```
'late in the new york trading day , the dollar was quoted at 1.8578 marks , u
p from 1.8470 marks late thursday in new york.'
```

حال به طریق مشابه روی داده‌های تست نتیجه بررسی و گزارش می‌شود.

In [134]:
```
# test one sample:
# load data:
data = open('./test.data', 'r').read().lower()
# convert data into sentences:
sentences = data.split('. . o\n\n')[0:-1]
```

می‌توان مشابه عبارت داده شده در صورت مساله به کمک دستورات زیر ، خروجی را به صورت تصویر تبدیل کرد. به این ترتیب که کلمات هم نقش با رنگهای مشابهی نشان داده شوند.

In [80]:
```
img = np.ones((500, 2000, 3), dtype=np.uint8)*255
```

In [86]:
```
print(label)
```

```
['\n', 'b-adjp', 'b-advp', 'b-np', 'b-pp', 'b-prt', 'b-sbar', 'b-vp', 'i-adjp
', 'i-advp', 'i-np', 'i-pp', 'i-sbar', 'i-vp', 'o']
```

در سلول پایین یک تابع برای تخصیص رنگ به هر یک از ۱۵ برچسب مسأله تعریف شده است:

In [135]:
```
COLOR = ['black', 'magenta', 'blue', 'green', 'yellow', 'cyan',
(0.45,0.925,0.525), 'orange', 'red',
        (0.5,0.1, 0.1), (0.1,0.5,0.1), (0.1, 0.1, 0.5), (0.5, 0.1, 0.5),
(0.5, 0.5, 0.1), (0.1, 0.5, 0.5)]
def getcolor(pos):
    if label.count(pos) != 0:
        return COLOR[label.index(pos)]
    else:
        return COLOR[0]
getcolor('b-sbar')
```

Out[135]:
```
(0.45, 0.925, 0.525)
```

در سلول زیر، روی ۱۰ جمله نمونه از داده‌های تست، نتیجه پیش بینی شبکه به صورت رنگی و نمایشی بررسی می‌شود. به این صورت که در هر بلاک از تصویر دو خط نوشته می‌شود و در خط اول خروجی پیشبینی شبکه نمایش داده می‌شود و در خط دوم رنگ‌ها بر اساس برچسب‌ها مشخص می‌شوند.

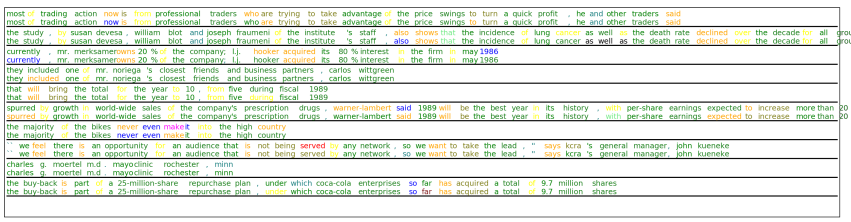
In [100]:
```
#index = np.random.permutation(len(sentences))[0:10].tolist()
index = [355, 324, 465, 224, 375, 68, 688, 491, 526, 451]
#
plt.figure(figsize=(20, 30), dpi=180)
plt.imshow(img);
for idx in range(len(index)):
    X_test = []
    Y_test = []
    xt = []
    yt = []
    line = sentences[index[idx]].splitlines()
    for j in range(len(line)):
        if len(line[j].split(' ')) == 3:
            w, _, c = line[j].split(' ')
            xt.append(w)
            yt.append(c)
            #words.append(w)
            #chunks.append(c.split('-')[-1])
    X_test.append(' '.join(xt))
    Y_test.append(' '.join(yt))

    x_test = np.zeros((1,len(X_test[0].split(' ') + ['\n']), 100))

    T = X_test[0].split(' ')
    for j in range(len(T)):
        try:
            x_test[0, j]= w2v[T[j]]
        except:
            E.append(T[j])
            x_test[0, j] = UKN
    l = model.predict(x_test)[0].argmax(-1)

    a = []
    for i in range(l.shape[0]):
        c = label[l[i]]
        a.append(c)


    s = X_test[0].split(' ')
    #t = int(2000/len(s))
    pos = Y_test[0].split(' ')
    plt.text(x=5, y=20+(45*idx), s=s[0], color=getcolor(a[0]))
    plt.text(x=5, y=40+(45*idx), s=s[0], color=getcolor(pos[0]))
    st_x = len(s[0])*10+10
    for i in range(1, len(s)):
        plt.text(x=5+st_x, y=20+(45*idx), s=s[i], color=getcolor(a[i]))
        plt.text(x=5+st_x, y=40+(45*idx), s=s[i], color=getcolor(pos[i]))
        st_x += len(s[i])*10
        st_x += 11
    plt.hlines(40+45*idx+5, 5, 1995)
    plt.xticks([])
    plt.yticks([]);
```



در نهایت با استفاده از ماتریس آشفتگی تخمین می‌توان عمل کرد شبکه را روی کلاس‌های مختلف بررسی کرد.

In [104]:

```
preds = []  # predicted labels
targets = [] # True labels
# loop over sentences:
for idx in range(len(sentences)):
    X_test = []
    Y_test = []
    xt = []
    yt = []
    line = sentences[idx].splitlines()
    # loop over words in one sentence and extract words and True labels
    for j in range(len(line)):
        if len(line[j].split(' ')) == 3:
            w, _, c = line[j].split(' ')
            xt.append(w)
            yt.append(c)

    X_test.append(' '.join(xt))
    Y_test.append(' '.join(yt))
    # word coding:
    x_test = np.zeros((1,len(X_test[0].split(' ') + ['\n']), 100))

    T = X_test[0].split(' ')
    for j in range(len(T)):
        try:
            x_test[0, j]= w2v[T[j]]
        except:
            x_test[0, j] = UKN
    # prediction:
    l = model.predict(x_test)[0].argmax(-1)

    # extract predicted labels
    a = []
    for i in range(l.shape[0]):
        a.append(l[i])

    # stack the predection in one list
    preds = preds + a

    # extract true labels
    pos = Y_test[0].split(' ') + ['\n']
    # some test labels arn't included in the train data set.
    # so here, they will be removed and considerd as \n
    for k in range(len(pos)):
        if label.count(pos[k]) == 0:
            pos[k] = '\n'
    # stack all targets in one list:
    t = [label.index(pos[i]) for i in range(len(pos))]
    targets = targets + t
```

In [139]:
```
# calculate and predict the confusion matrix:
CM = confusion_matrix(y_pred=preds, y_true=targets)
for i in range(CM.shape[0]):
    for j in range(CM.shape[1]):
        print('%4d'%(CM[i,j]), end=' ')
    print()
```

```
  56   58   11    9    6    0    3  184    3   12  300    0    0   57   79
   0   23   17   45    5    0    0   17    1    1   29    0    0   29    5
   0   11   70   78   19    1    8   82    1    1   31    0    0   14   18
   3   31  114 3900   89    2   31   80    4    5  638    0    0   49  133
   0    3   22  119 1709    3   12   30    3    2   41    0    0   22   22
   0    2    7    7   13    0    1    2    0    0    1    0    0    5    1
   0    0   15   39   35    0   62    9    0    0    4    0    0    2   26
   0   28   72  156   78    0   16 1215    2    2  220    0    1   93   20
   0   11    5   12    1    0    2    4    5    0   21    0    0    6   12
   0    5    5    3    1    0    2    3    1    0    8    0    0    1    7
   8   31   44  292   59    0    3  240    4    5 4759    0    0   11  235
   0    0    0    4    9    0    0    0    0    0    0    0    0    0    2
   0    0    1    0    0    0    0    0    0    0    0    1    0    0    0
   0   29   37  168   34    7   12   90    1    4   21    0    0  644   14
   0    0   11   43   19    0   20   13    0    0   95    0    0    6 1596
```

In [140]:
```
# calculate the accuracy of model:
acc = np.diag(CM) / CM.sum(1)
acc*100
```

Out[140]:
```
array([ 7.19794344, 13.37209302, 20.95808383, 76.78676905, 85.96579477,
        0.        , 32.29166667, 63.84655807,  6.32911392,  0.        ,
       83.6232648 ,  0.        ,  0.        , 60.69745523, 88.51913478])
```

In [141]:
```
# caluclate the precision of model:
prec = np.diag(CM) / CM.sum(0)
prec[6] = 0
prec*100
```

Out[141]:
```
array([83.58208955,  9.9137931 , 16.2412993 , 80.        , 82.2821377 ,
        0.        ,  0.        , 61.70644997, 20.        ,  0.        ,
       77.15629053,  0.        ,  0.        , 68.58359957, 73.5483871 ])
```

جمع بندی:

در اینجا یک مدل برای تشخیص نقش کلمات در یک جمله ارایه شد. مدل ارایه شده یک مدل ساده اما نسبتاً کارآمد بر مبنای شبکه‌های بازگشتی و یک دسته بند است. اما با توجه به این که داده‌های آموزشی به اندازه کافی در اختیار نداریم، بنابراین مدل به طور مناسب آموزش نمی‌بیند. مسأله دیگر بعد برچسب‌ها است. که تعداد آن‌ها نسبت به یکدیگر بسیار نا متوازن می‌باشد و تاثیر جدی در روند آموزشی کلاس‌ها می‌گذارد. این اختلاف در ماتریس آشفتگی داده‌های آزمون و نیز دقت خروجی مشخص می‌باشد.