3.3 Word Embedding

The output from the Embedding layer will be 4 vectors of 8 dimensions each, one for each word. We flatten this to a one 32-element vector to pass on to the Dense output layer.

In [126]:

```
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length)) #max_length=4

model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura
# summarize the model

print(model.summary())

# fit the model
model.fit(padded_docs, labels, epochs=200, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=1)
print('Accuracy: %f' % (accuracy*100))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 4, 8)              200       
_________________________________________________________________
flatten (Flatten)            (None, 32)                0         
_________________________________________________________________
dense (Dense)                (None, 1)                 33        
=================================================================
Total params: 233
Trainable params: 233
Non-trainable params: 0
_________________________________________________________________
None
1/1 [==============================] - 0s 1ms/step - loss: 0.3156 - accura
cy: 1.0000
Accuracy: 100.000000
```

In [127]:

```
probabilities=model.predict(padded_docs)
predictions = [float(np.round(x)) for x in probabilities]
accuracy = np.mean(predictions == labels)
print('Accuracy: %f' % (accuracy*100))
```

```
Accuracy: 100.000000
```

```
In [128]: docs
```

```
Out[128]: ['Well done!',
           'Good work',
           'Great effort',
           'nice work',
           'Excellent!',
           'Weak',
           'Poor effort!',
           'not good',
           'poor work',
           'Could have done better.']
```

```
In [129]: padded_docs
```

```
Out[129]: array([[18, 19,  0,  0],
                 [ 1, 10,  0,  0],
                 [12, 10,  0,  0],
                 [ 5, 10,  0,  0],
                 [23,  0,  0,  0],
                 [ 2,  0,  0,  0],
                 [24, 10,  0,  0],
                 [22,  1,  0,  0],
                 [24, 10,  0,  0],
                 [19, 23, 19,  6]])
```

```
In [130]: idx=0
          print(padded_docs[idx])
          s1=padded_docs[idx].reshape(1,max_length)  #(1,4)
          s1
```

```
          [18 19  0  0]

Out[130]: array([[18, 19,  0,  0]])
```

```
In [131]: model.predict(s1)
```

```
Out[131]: array([[0.8058486]], dtype=float32)
```

```
In [132]: model.predict(padded_docs)>0.5
```

```
Out[132]: array([[ True],
                 [ True],
                 [ True],
                 [ True],
                 [ True],
                 [False],
                 [False],
                 [False],
                 [False],
                 [False]])
```

3.4 word embeddeing

https://keras.io/api/layers/core_layers/embedding/
如果的訓練很好的話，可以將權重存下來，之後可以丟入model文字而得到word vector,此及為 word embeddeing

```
In [134]: from keras.models import Model

          word_model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
          word_model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_input (InputLayer) [(None, 4)]               0
_________________________________________________________________
embedding (Embedding)        (None, 4, 8)              200
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
=================================================================
Total params: 200
Trainable params: 200
Non-trainable params: 0
_________________________________________________________________
```

```
In [135]: sentence_feature=word_model.predict(padded_docs)
```

```
 In [51]: sentence_feature.shape

Out[51]: (10, 32)
```

```
In [136]: sentence_feature[0]

Out[136]: array([ 0.18417265, -0.22292025,  0.16876097,  0.2389457 , -0.17995703,
                 -0.2584377 ,  0.25680435,  0.2329329 , -0.18565285,  0.22311208,
                 -0.1610491 ,  0.20349775,  0.27811158,  0.27742064, -0.25657344,
                  0.17421931, -0.0709856 , -0.09383022, -0.09006649, -0.10515735,
                  0.08793452, -0.08655199, -0.06868622, -0.04377315, -0.0709856 ,
                 -0.09383022, -0.09006649, -0.10515735,  0.08793452, -0.08655199,
                 -0.06868622, -0.04377315], dtype=float32)
```

```
In [146]: def cosine_similarity(x, y):
              return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))

          def euclidean_distance(x, y):
              return np.sqrt(np.sum((x - y) ** 2))
```

```
In [138]: for i in range(len(docs)):
              ang=cosine_similarity(sentence_feature[0],sentence_feature[i])
              print(ang)
```

```
1.0000001
0.6868618
0.85712045
0.8530629
0.64138794
-0.41919726
-0.25246334
-0.5339921
-0.25246334
-0.4035426
```

In [147]:

```
for i in range(len(docs)):
    ang=euclidean_distance(sentence_feature[0],sentence_feature[i])
    print(ang)
```

```
0.0
0.69748384
0.49061915
0.49515063
0.7321967
1.4633414
1.4093796
1.679208
1.4093796
1.7650622
```

In [151]:

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
     'Python is a good programming language.',
     'C is a good programming language.',
     'Python Python Python Python Python.',
     ]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
word=vectorizer.get_feature_names()
print(word)
print(len(word))
print(X.toarray())
```

```
['good', 'is', 'language', 'programming', 'python']
5
[[1 1 1 1 1]
 [1 1 1 1 0]
 [0 0 0 0 5]]
```

In [154]:

```
for i in range(len(corpus)):
    ang=cosine_similarity(X.toarray()[0],X.toarray()[i])
    print(ang)
```

```
0.9999999999999998
0.8944271909999159
0.4472135954999579
```

In [152]:

```
for i in range(len(corpus)):
    ang=euclidean_distance(X.toarray()[0],X.toarray()[i])
    print(ang)
```

```
0.0
1.0
4.47213595499958
```

```
In [139]: docs
```

```
Out[139]: ['Well done!',
           'Good work',
           'Great effort',
           'nice work',
           'Excellent!',
           'Weak',
           'Poor effort!',
           'not good',
           'poor work',
           'Could have done better.']
```

3.5 Create Input Text Sequence

```
"A little girl running in field"

X1        X2(text sequence)                        y(word)
-----------------------------------------------------------------
image    a                                        little
image    a,little,                                girl
image    a,little, girl,                          running
image    a,little, girl, running,                 in
image    a,little, girl, running, in,             field
image    a,little, girl, running, in, field       endseq
```

In [140]:

```
# Create sequences input sequences and output words for an image
def create_sequences(tokenizer, max_length, captions_list):
    # X1 : input for image features
    # X2 : input for text features
    # y  : output word
    X1, X2, y = list(), list(), list()
    vocab_size = len(tokenizer.word_index) + 1  # +1 for corresponding to in
    # Walk through each caption for the image
    for caption in captions_list:
        # Encode the sequence

        seq = tokenizer.texts_to_sequences([caption])[0] # t=[[1, 3, 4, 2]],
        #print(caption,'->',seq)
        # Split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # Split into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            #print('X=',in_seq,'Y=',out_seq)
            # Pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0] #[1,2]--
            # Encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # Store

            X2.append(in_seq)
            y.append(out_seq)

    return X2, y
```

In [141]:

```
input_sequence, output_word=create_sequences(tokenizer,4,texts)
```

In [142]:

```
input_sequence
```

Out[142]:

```
[array([0, 0, 0, 1]),
 array([0, 0, 1, 2]),
 array([0, 1, 2, 3]),
 array([1, 2, 3, 5]),
 array([0, 0, 0, 1]),
 array([0, 0, 1, 4]),
 array([0, 1, 4, 2]),
 array([1, 4, 2, 3]),
 array([4, 2, 3, 6]),
 array([0, 0, 0, 7]),
 array([0, 0, 7, 1]),
 array([0, 7, 1, 2]),
 array([7, 1, 2, 3]),
 array([1, 2, 3, 8]),
 array([0, 0, 0, 2]),
 array([0, 0, 2, 1]),
 array([0, 2, 1, 3]),
 array([2, 1, 3, 5]),
 array([ 0,  0,  0, 10])]
```

```
In [26]: output_word
```

```
Out[26]: [array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32),
          array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
          array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.], dtype=float32)]
```

3.6 TO DO: slot filling

a little girl is **running**

Hint: use RNN to do train this model