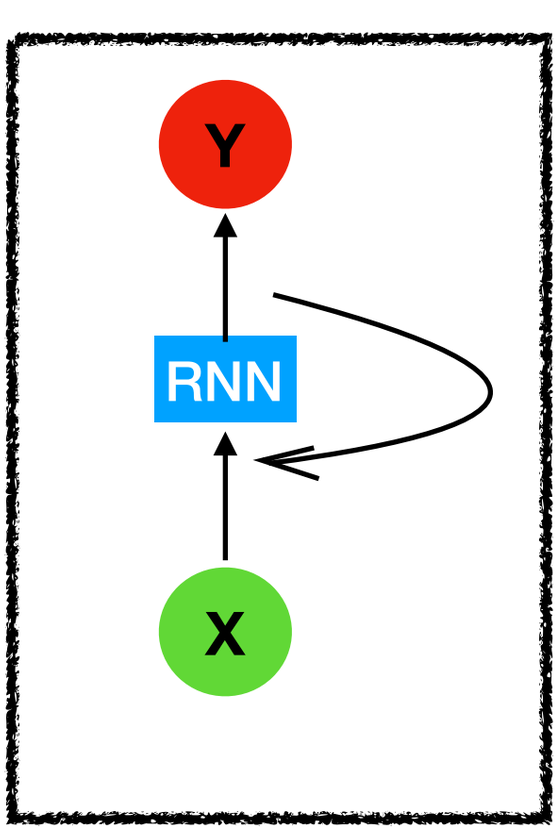
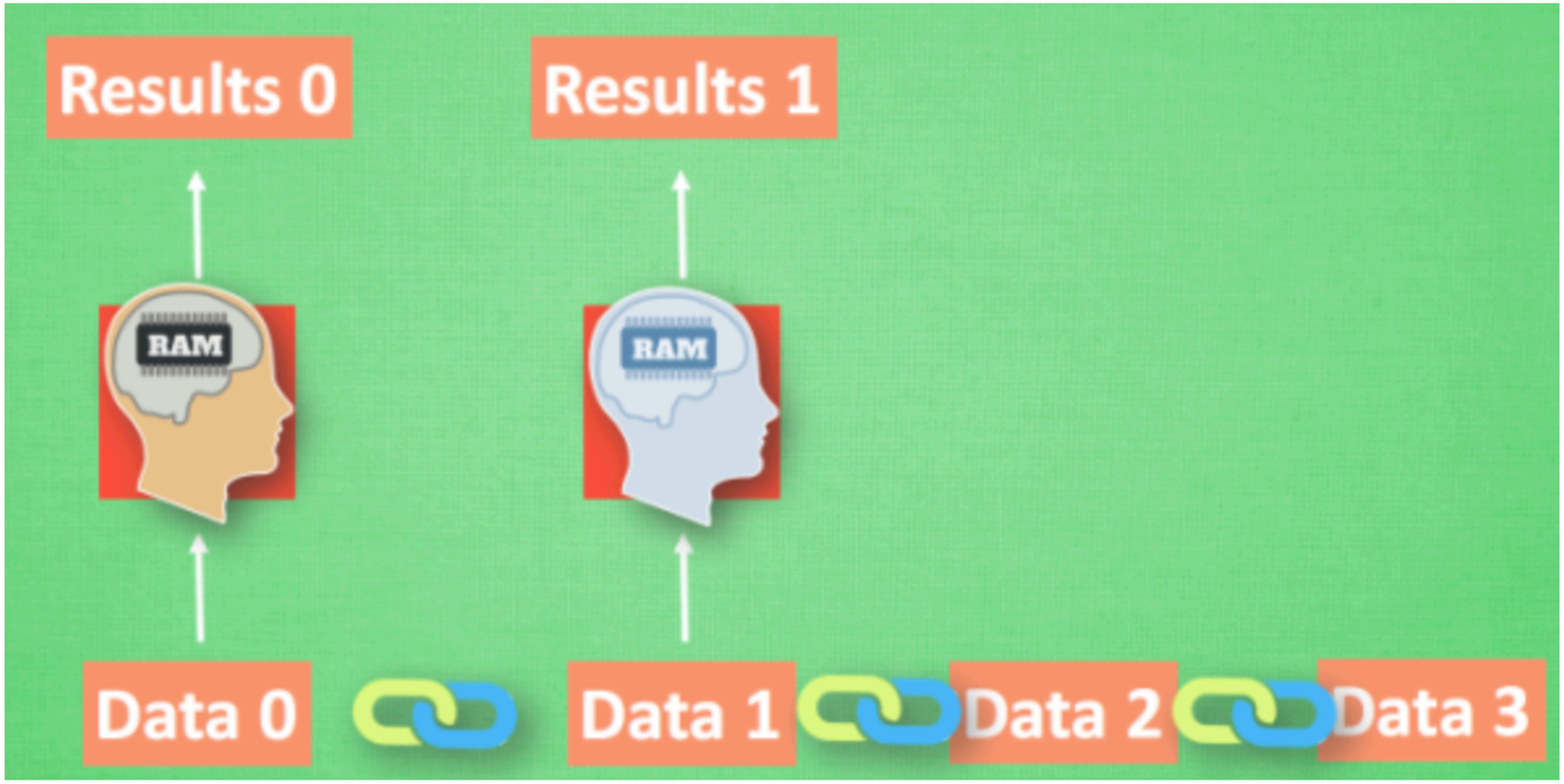
LSTM
長短期記憶網絡
(Long Short Term Memory)

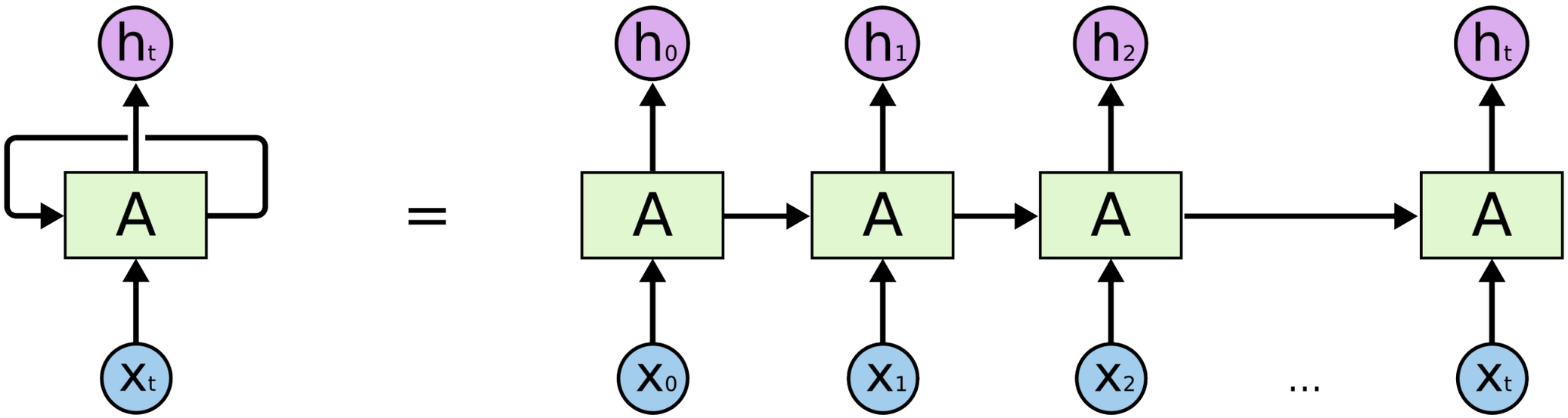
Before LSTM, Let's talk about RNN

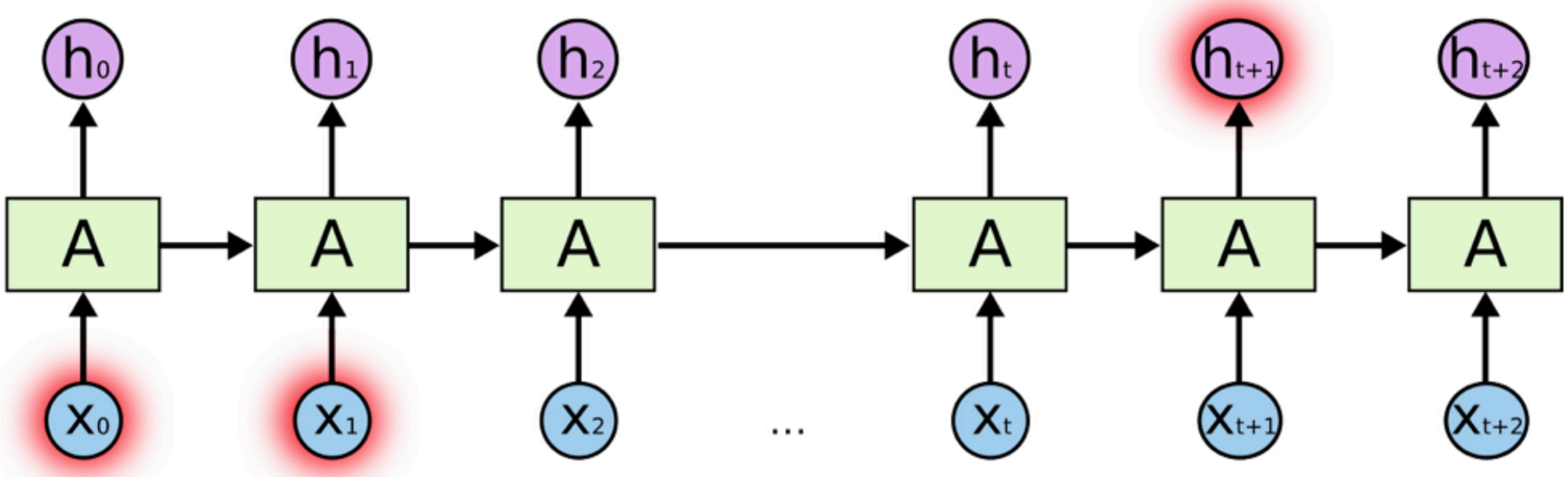
RNN的特點是利用序列的信息，如果你想預測一個句子的下一個詞，知道之前的詞是有幫助的。

RNN被成為遞歸的(recurrent)原因就是它會對一個序列的每一個元素執行同樣的操作，並且之後的輸出依賴於之前的計算。

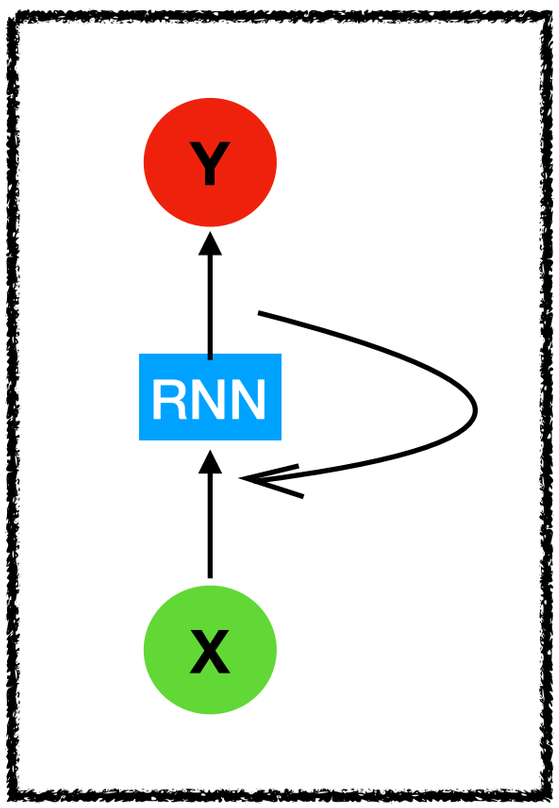





h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t


h_0
h_1
h_2
h_t
h_{t+1}
h_{t+2}
A
A
A
A
A
A
x_0
x_1
x_2
...
x_t
x_{t+1}
x_{t+2}

RNN的問題



梯度消失 + 梯度爆炸

*W *W *W ..... *W

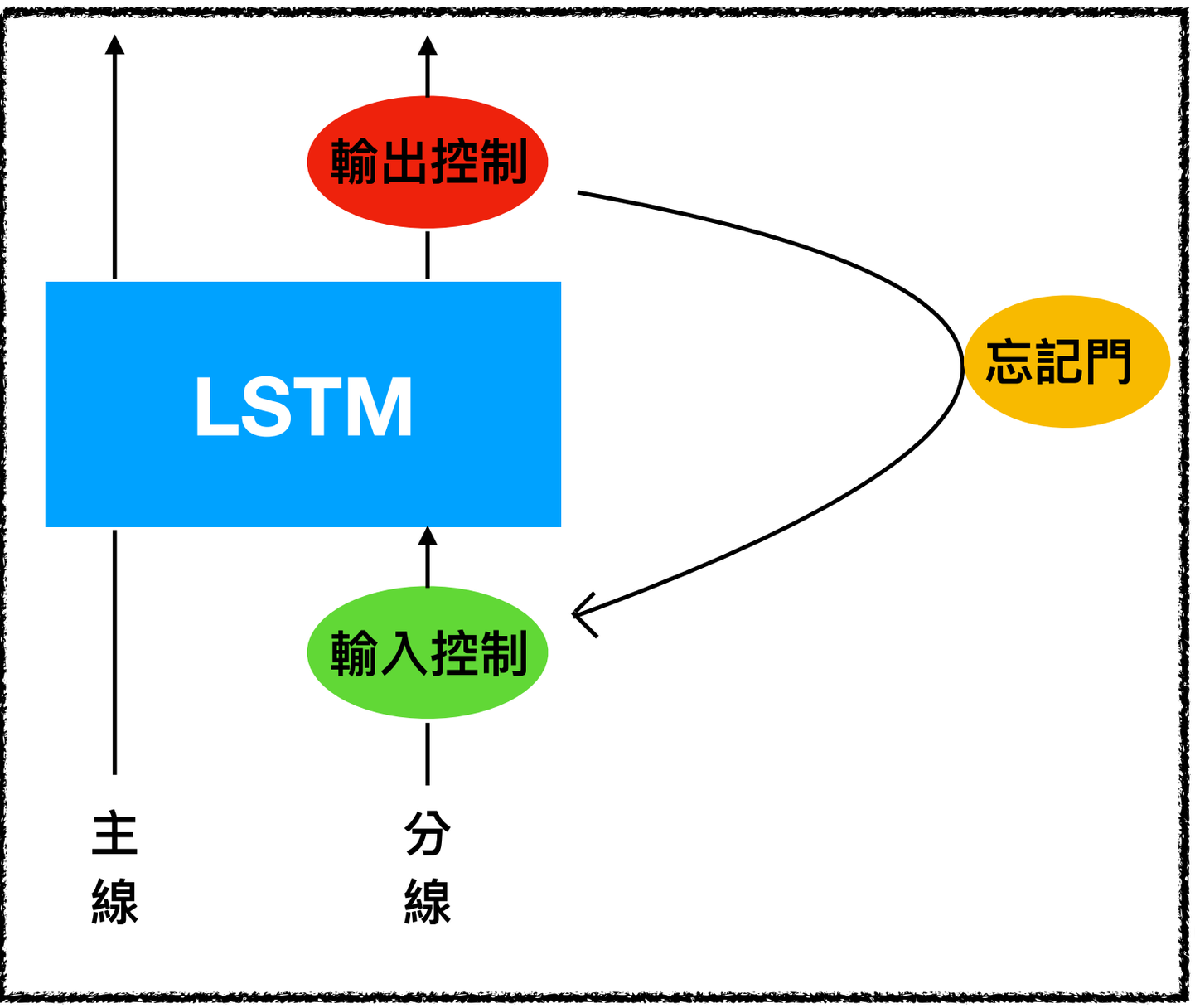
t_1 t_2 t_3 t_4 ..... t_n

在反向傳遞時，每個時間點得訊息都會乘上一個W

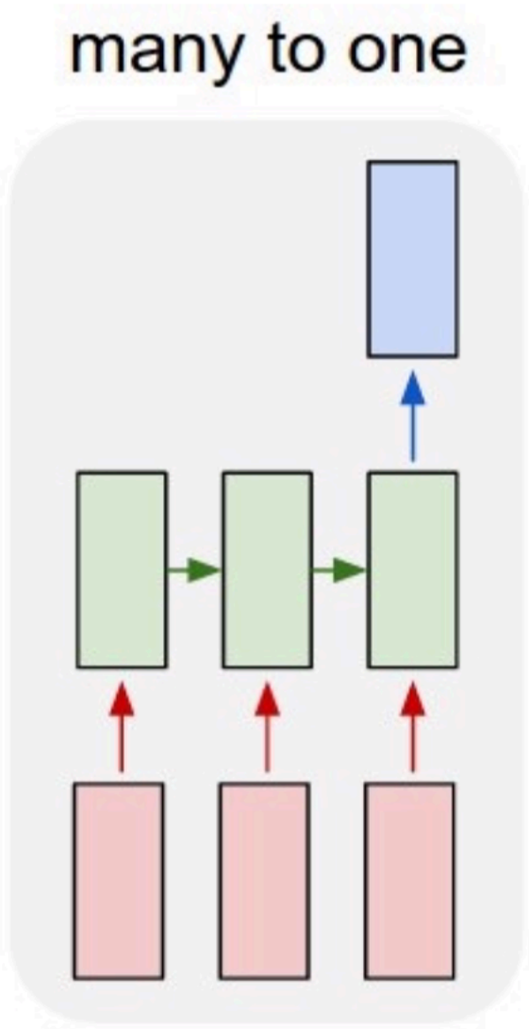
當 W < 1 (梯度小) —> 造成之前的訊息不容易記住，形成梯度消失

當 W > 1 (梯度大) —> 造成神經網路不穩定，無法從中得取資訊，梯度爆炸

RNN健忘的救星：LSTM



- 輸入控制：依照當前訊息對整體的重要性按照比例寫進主線中(Sigmoid)
- 忘記控制：如果新訊息改變了對之前訊息的想法，則忘記控制就會將之前某些主線訊息遺忘，替換成新的訊息(Sigmoid)
- 輸出控制：根據主線以及分線的訊息，判斷要輸出什麼訊息(Tanh)



時間序列預測

序列分類

Model 1:

1 Feature 1 Time_step

匯入 Keras 及相關模組

In [2]:
```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences

# 給定隨機的種子, 以便讓大家跑起來的結果是相同的
np.random.seed(7)
```

定義數據集，字母表(alphabet)

In [2]:
```
# 定義序列數據集
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))

print("字母對應到數字編號:\n", char_to_int)
print("\n")
print("數字編號對應到字母:\n", int_to_char)
```

```
字母對應到數字編號:
 {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13,
'O': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25}


數字編號對應到字母:
 {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N',
14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z'}
```

準備Input

In [7]:
```
# 準備輸入數據集
seq_len = 1
datax = []
datay = []

for i in range(0, len(alphabet)-seq_len, 1):
    seq_in = alphabet[i:i+seq_len]
    seq_out = alphabet[i+seq_len]
    datax.append([char_to_int[char] for char in seq_in])
    datay.append(char_to_int[seq_out])
    print(seq_in, "->", seq_out)
```

```
A -> B
B -> C
C -> D
D -> E
E -> F
F -> G
G -> H
H -> I
I -> J
J -> K
K -> L
```

(samples, time_steps, features)

(n, 1, 1)

資料前處理

In [5]:
```
# (samples, time_steps, features)
# 這裡的1個字符會變成1個時間步裡頭的1個element的"feature"向量
x = np.reshape(datax, (len(datax), seq_len, 1))

# normalize
x = x/float(len(alphabet))

# one-hot
y = np_utils.to_categorical(datay)

print("x shape: ", x.shape)  # (25筆samples, "1"個時間步長, 1個feature)
print("y shape: ", y.shape)
```

```
x shape:  (25, 1, 1)
y shape:  (25, 26)
```

重塑維度成lstm的格式

normalize

建立模型

In [6]:
```
# model
model = Sequential()
model.add(LSTM(32, input_shape=(x.shape[1], x.shape[2])))
model.add(Dense(y.shape[1], activation="softmax"))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                4352
_________________________________________________________________
dense_1 (Dense)              (None, 26)                858
=================================================================
Total params: 5,210
Trainable params: 5,210
Non-trainable params: 0
_________________________________________________________________
```

訓練

In [12]:
```
# train
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
model.fit(x, y, epochs=500, batch_size=1, verbose=2)
```

```
Epoch 1/500
 - 1s - loss: 3.2660 - acc: 0.0000e+00
Epoch 2/500
 - 0s - loss: 3.2582 - acc: 0.0000e+00
Epoch 3/500
 - 0s - loss: 3.2551 - acc: 0.0400
Epoch 4/500
 - 0s - loss: 3.2524 - acc: 0.0400
Epoch 5/500
 - 0s - loss: 3.2495 - acc: 0.0400
```

評估model的準確度

In [13]:
```
scores = model.evaluate(x, y, verbose=0)
print("model acc: %.2f%%" % (scores[1]*100))
```

```
model acc: 88.00%
```

預測結果

In [10]:
```
for pattern in datax:
    x = np.reshape(pattern, (1, len(pattern), 1))
    x = x/float(len(alphabet))

    pred = model.predict(x, verbose=0)
    index = np.argmax(pred)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print(seq_in, "->", result)
```

```
['A'] -> B
['B'] -> C
['C'] -> D
['D'] -> E
['E'] -> F
['F'] -> G
['G'] -> H
['H'] -> I
['I'] -> J
['J'] -> K
['K'] -> L
['L'] -> M
['M'] -> N
['N'] -> O
['O'] -> P
['P'] -> Q
['Q'] -> R
['R'] -> S
['S'] -> T
```

Run看看結果

Model 2:

3 Features 1 Time_step

準備Input

In [14]:
```
# model2 with seq_len=3
seq_len = 3
datax = []
datay = []

for i in range(0, len(alphabet)-seq_len, 1):
    seq_in = alphabet[i:i+seq_len]
    seq_out = alphabet[i+seq_len]
    datax.append([char_to_int[char] for char in seq_in])
    datay.append(char_to_int[seq_out])
    print(seq_in, "->", seq_out)
```

```
ABC -> D
BCD -> E
CDE -> F
DEF -> G
EFG -> H
FGH -> I
GHI -> J
HIJ -> K
IJK -> L
JKL -> M
KLM -> N
LMN -> O
```

(samples, time_steps, features)

(n, 1, 3)

資料前處理

In [15]:
```
# preprocessing
# (samples, time_steps, features)
x = np.reshape(datax, (len(datax), 1, seq_len))

# normalization
x = x/float(len(alphabet))

# one-hot
y = np_utils.to_categorical(datay)

print("x shape", x.shape)
print("y shape", y.shape)
```

重塑維度成lstm的格式

三個字會變成一個有3個element的"feature" vector。
1筆訓練資料只有”1"個time_steps, 裡頭存放著"3"個字符的資料"features"向量。

```
x shape (23, 1, 3)
y shape (23, 26)
```

建立模型

In [16]:
```
# build the model
model = Sequential()
model.add(LSTM(32, input_shape=(x.shape[1], x.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 32)                4608
_________________________________________________________________
dense_2 (Dense)              (None, 26)                858
=================================================================
Total params: 5,466
Trainable params: 5,466
Non-trainable params: 0
_________________________________________________________________
```

訓練

In [*]:
```
# start training
model.compile(loss="categorical_crossentropy",
              optimizer="adam", metrics=["accuracy"])
model.fit(x, y, epochs=500, batch_size=1, verbose=2)
```

```
Epoch 1/500
 - 1s - loss: 3.2754 - acc: 0.0000e+00
Epoch 2/500
 - 0s - loss: 3.2629 - acc: 0.0000e+00
Epoch 3/500
 - 0s - loss: 3.2556 - acc: 0.0000e+00
Epoch 4/500
 - 0s - loss: 3.2487 - acc: 0.0435
Epoch 5/500
 - 0s - loss: 3.2420 - acc: 0.0435
```

評估model的準確度

In [18]:
```
# evaluate the model
scores = model.evaluate(x, y, verbose=0)
print("Model ACC: %.2f%%" % (scores[1]*100))
```

```
Model ACC: 82.61%
```

預測結果

In [19]:
```
# making prediction
for pattern in datax:
    x = np.reshape(pattern, (1, 1, len(pattern)))
    x = x/float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = np.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print(seq_in, "->", result)
```

```
['A', 'B', 'C'] -> D
['B', 'C', 'D'] -> E
['C', 'D', 'E'] -> F
['D', 'E', 'F'] -> G
['E', 'F', 'G'] -> H
['F', 'G', 'H'] -> I
['G', 'H', 'I'] -> J
['H', 'I', 'J'] -> K
['I', 'J', 'K'] -> L
['J', 'K', 'L'] -> M
['K', 'L', 'M'] -> N
```

Run看看結果

比較Model 1 & Model 2

“Model 2” 相較於 "Model 1” 在預測的表現上只有小幅提升。

簡單的序列問題，即使使用Window方法，我們仍然無法讓LSTM學習到預測正確的順序。

以上的範例也是一個誤用LSTM網絡的糟糕的張量結構。

事實上，字母序列是一個特徵的”時間步驟(timesteps)"，而不是單獨特徵的一個時間步驟。

我們已經給了網絡更多的上下文，但是沒有更多的 ”順序” 上下文(context)。

下一範例中，我們將以”時間步驟(timesteps)”的形式提升模型的表現。

Model 3 :

3 Features —> 3 Time_steps

準備Input

In [20]:
```
# model with time_steps=3
seq_length = 3
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print(seq_in, '->', seq_out)
```

```
ABC -> D
BCD -> E
CDE -> F
DEF -> G
EFG -> H
FGH -> I
GHI -> J
HIJ -> K
IJK -> L
JKL -> M
KLM -> N
LMN -> O
```

(samples, time_steps, features)

(n, 3, 1)

資料前處理

In [21]:
```
x = np.reshape(dataX, (len(dataX), seq_length, 1))
x = x/float(len(alphabet))
y = np_utils.to_categorical(dataY)
```

重塑維度成lstm的格式

```
print("x shape: ", x.shape)
print("y shape: ", y.shape)
x
```

```
x shape:  (23, 3, 1)
y shape:  (23, 26)
```

Out[21]:
```
array([[[ 0.        ],
        [ 0.03846154],
        [ 0.07692308]],

       [[ 0.03846154],
        [ 0.07692308],
        [ 0.11538462]],
```

準備訓練資料集的時候要把資料的張量結構轉換成,
1筆訓練資料有"3"個時間步, 裡頭存放著"1"個字符的資料"features"向量。

建立模型

In [22]:
```
# model3
model = Sequential()
model.add(LSTM(32, input_shape=(x.shape[1], x.shape[2])))
model.add(Dense(y.shape[1], activation="softmax"))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 32)                4352
_________________________________________________________________
dense_3 (Dense)              (None, 26)                858
=================================================================
Total params: 5,210
Trainable params: 5,210
Non-trainable params: 0
_________________________________________________________________
```

訓練

In [*]:
```
# training
model.compile(loss="categorical_crossentropy",
              optimizer="adam", metrics=["accuracy"])
model.fit(x, y, epochs=500, batch_size=1, verbose=2)
```

```
Epoch 1/500
 - 1s - loss: 3.2633 - acc: 0.0000e+00
Epoch 2/500
 - 0s - loss: 3.2500 - acc: 0.0000e+00
Epoch 3/500
 - 0s - loss: 3.2424 - acc: 0.0435
Epoch 4/500
 - 0s - loss: 3.2357 - acc: 0.0000e+00
Epoch 5/500
 - 0s - loss: 3.2285 - acc: 0.0000e+00
Epoch 6/500
 - 0s - loss: 3.2207 - acc: 0.0435
Epoch 7/500
 - 0s - loss: 3.2125 - acc: 0.0435
```

評估model的準確度

In [24]:
```
# evaluating model
scores = model.evaluate(x, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model Accuracy: 100.00%
```

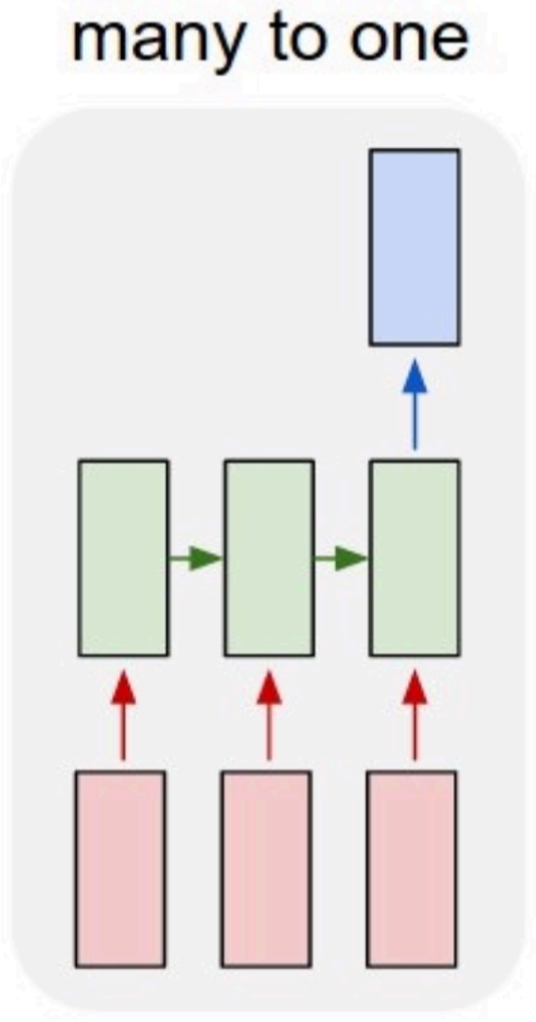
預測結果

In [25]:
```
# making prediction
# 讓我們擷取3個字符轉成張量結構 shape:(1,3,1)來進行infer
for pattern in datax:
    x = np.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = np.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print(seq_in, "->", result)
```

```
['A', 'B', 'C'] -> D
['B', 'C', 'D'] -> E
['C', 'D', 'E'] -> F
['D', 'E', 'F'] -> G
['E', 'F', 'G'] -> H
['F', 'G', 'H'] -> I
['G', 'H', 'I'] -> J
['H', 'I', 'J'] -> K
['I', 'J', 'K'] -> L
['J', 'K', 'L'] -> M
['K', 'L', 'M'] -> N
['L', 'M', 'N'] -> O
['M', 'N', 'O'] -> P
['N', 'O', 'P'] -> Q
```

Run看看結果

比較Model 3



當我們以”時間步驟(timesteps)”的形式給出更

多的上下文(context)來訓練LSTM模型時,

這時候循環神經網絡在序列資料的學習的效果就

可以發揮出它的效用。