statinfer

Recurrent Neural Network(RNN) and LSTM

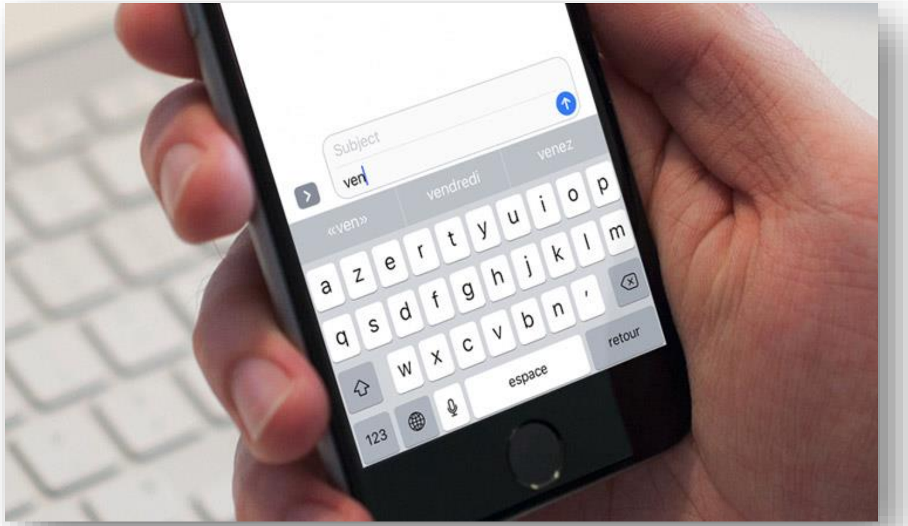
statinfer.com

Contents

- Sequential Models
- RNN Introduction
- Back Propagation Through Time
- RNN Model Building
- The problem of Vanishing Gradients
- LSTM models
- LSTM Model building

Can I have your number….

- Take your smart phone. Open a notepad or new message or mail.
- You need to type “**Can I have your number**”
- Type “Can” then start choosing the words from the suggestions made by your smart phone



What was the model behind text prediction?

Can
Can I
Can I have
Can I have your
Can I have your number

Input	Output
Can	I
Can I	have
Can I have	your
Can I have your	number
Sequence of words	next word

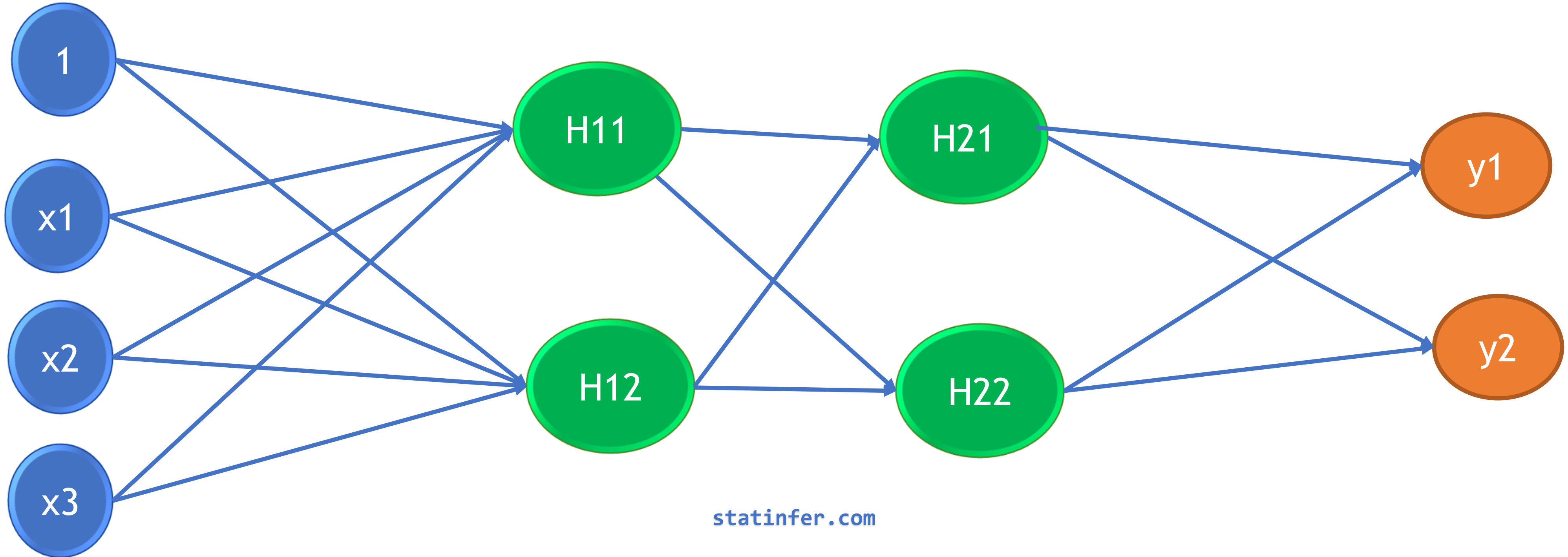
statinfer

A model for sequences

- What model was used for predicting the next word? – A sequential model
- Model accepts sequence of inputs and predicts the output/next-item in the sequence.
- Was it ANN model? – A universal function approximation model.
- Or was it CNN model? – A model that preserves spatial dependency.
- Or some other model?

ANN for sequential data

- To train this model, we need to supply x1,x2,x3 and y. At all points.
- In ANN the x1, x2 and x3 are not sequential. i.e x3 doesn't depend on x2 and x2 doesn't depend on x1
- In ANN y ~ x1+x2+x3 is same as y ~ x3+x2+x1



ANN for sequential data

- ANN doesn't assume any order in input variables.
- In a sequential model, the order is critical.
- In a sequential model, the output of previous prediction is input for the next prediction.
- In ANN, the outputs are independent of each other

Input	Output
Can	I
Can I	have

Two inputs for predicting this

ANN is not suitable for sequential models

- ANN is good for predicting independent text. But not for sequential text.
- ANN is best suited for non-sequential data
- ANN might do a good job in predicting next word, given a word(or words)
- We can somehow change the shape of the data, transform it and finally build an ANN. But it is very inefficient.

Input	Output
Can	I
Can I	have
Can I have	your
Can I have your	number
Sequence of words	next word

ANN doesn’t work

Input	Output
Can	I
I	have
have	your
your	number
One word	next word

ANN works

CNN is not suitable for sequential models

- CNN doesn't look at each word at a time.
- It preserves the spatial dependence. But, CNN doesn't really preserve the sequence.
- Kernel filters in CNN nullify the sequential ordering in the data. CNN doesn't work for sequential data

Input	Output
Can	I
Can I	have
Can I have	your
Can I have your	number
Sequence of words	next word

CNN doesn't work

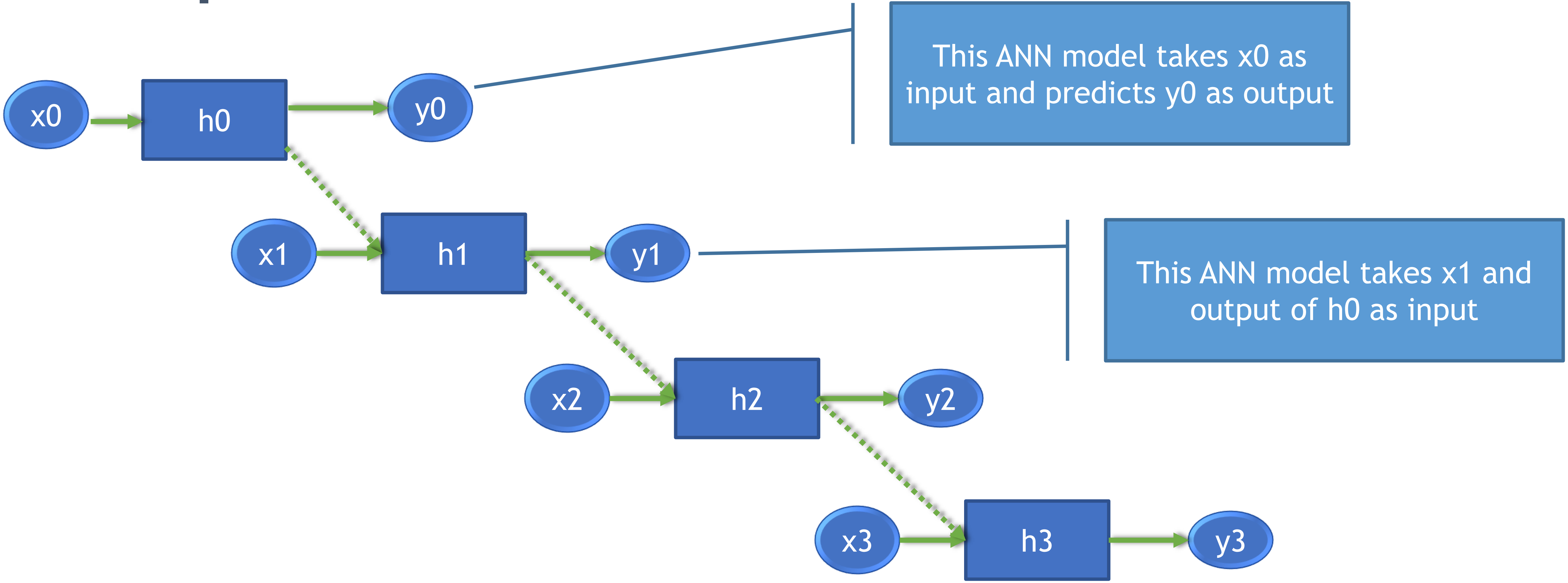
Input	Output
I Can have your	number
Your can I have	number
Can your I have	number
Can I have your	number
Words cluster	related word

CNN works

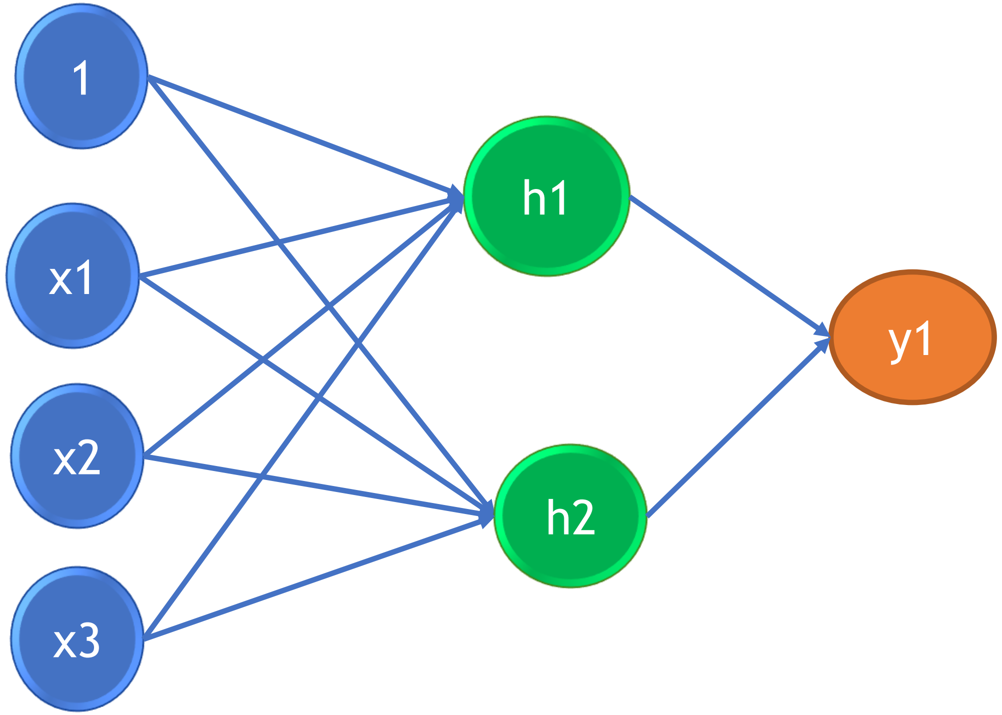
Sequential ANNs for Sequential data

- To build a model for sequential data, we need several dependent models in a sequence.
- We may have to build a model for single word prediction and use the output as the input for the next word prediction
- Remember, ANN does a good job for prediction of next word.
- We can use ANN for predicting the next word. We may have to take the output of ANN and use it as input in the next ANN

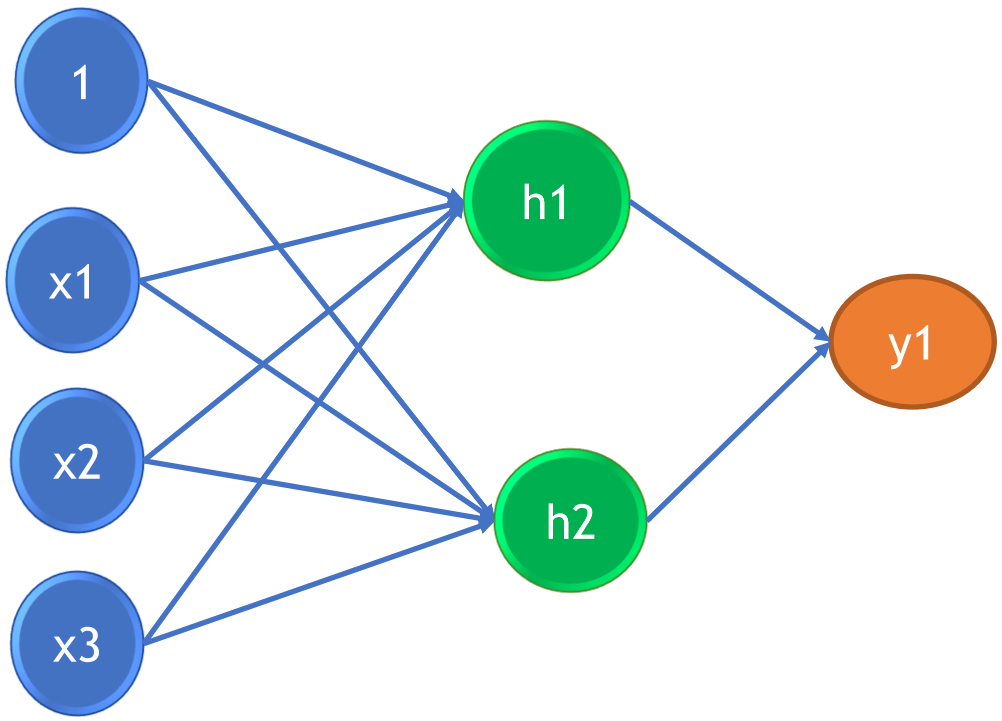
Sequential Models

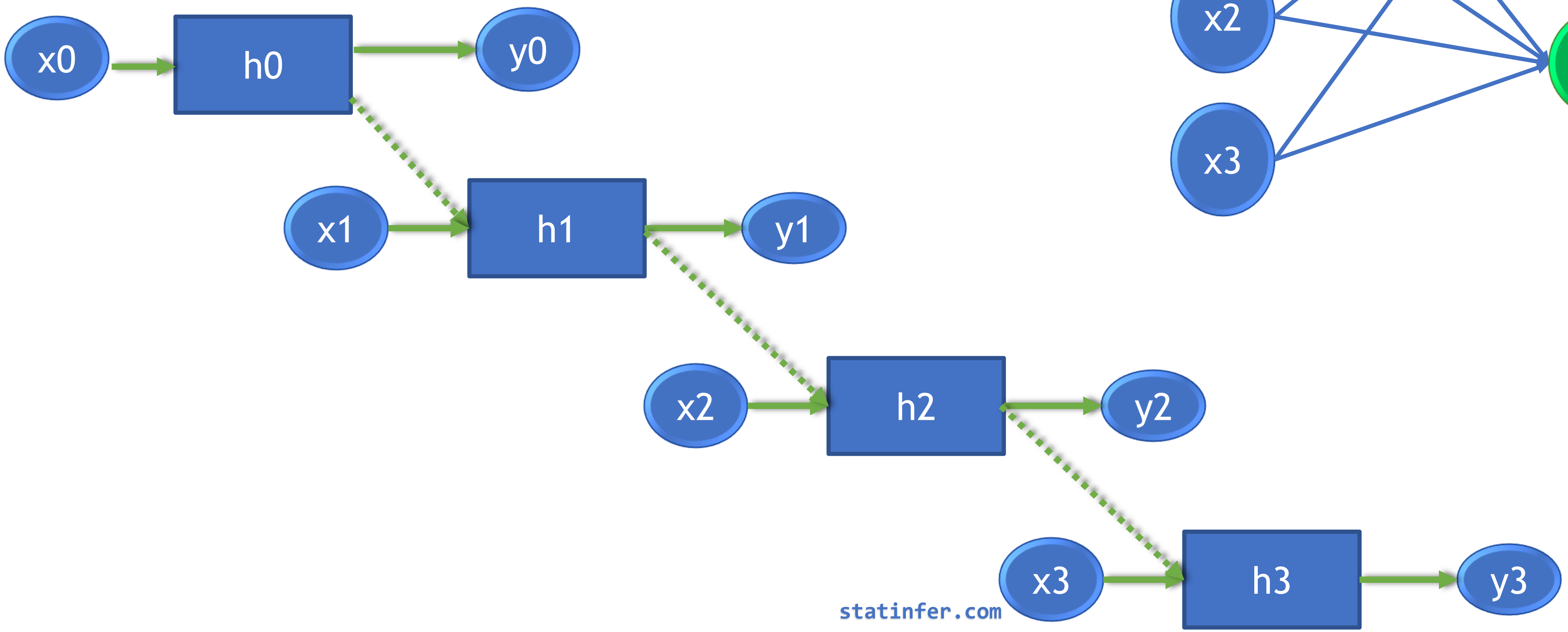


ANN

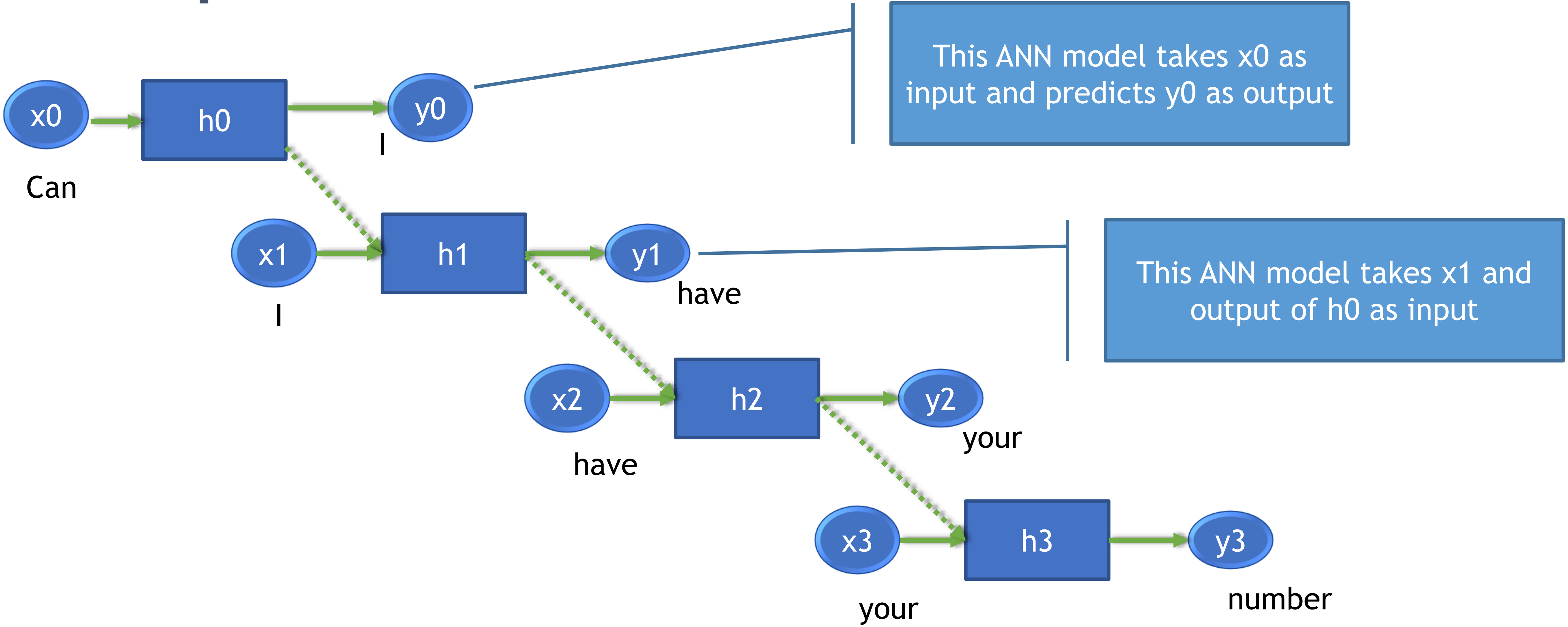


ANN vs Sequential ANN

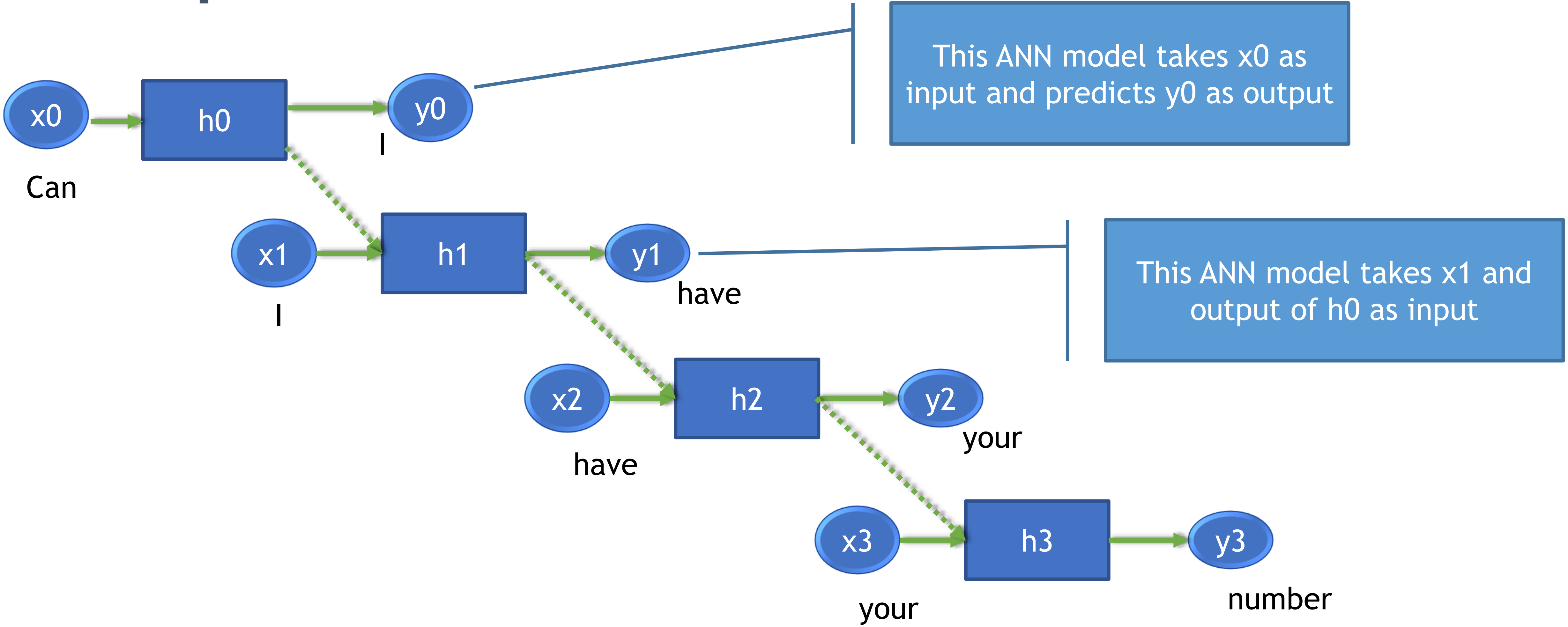




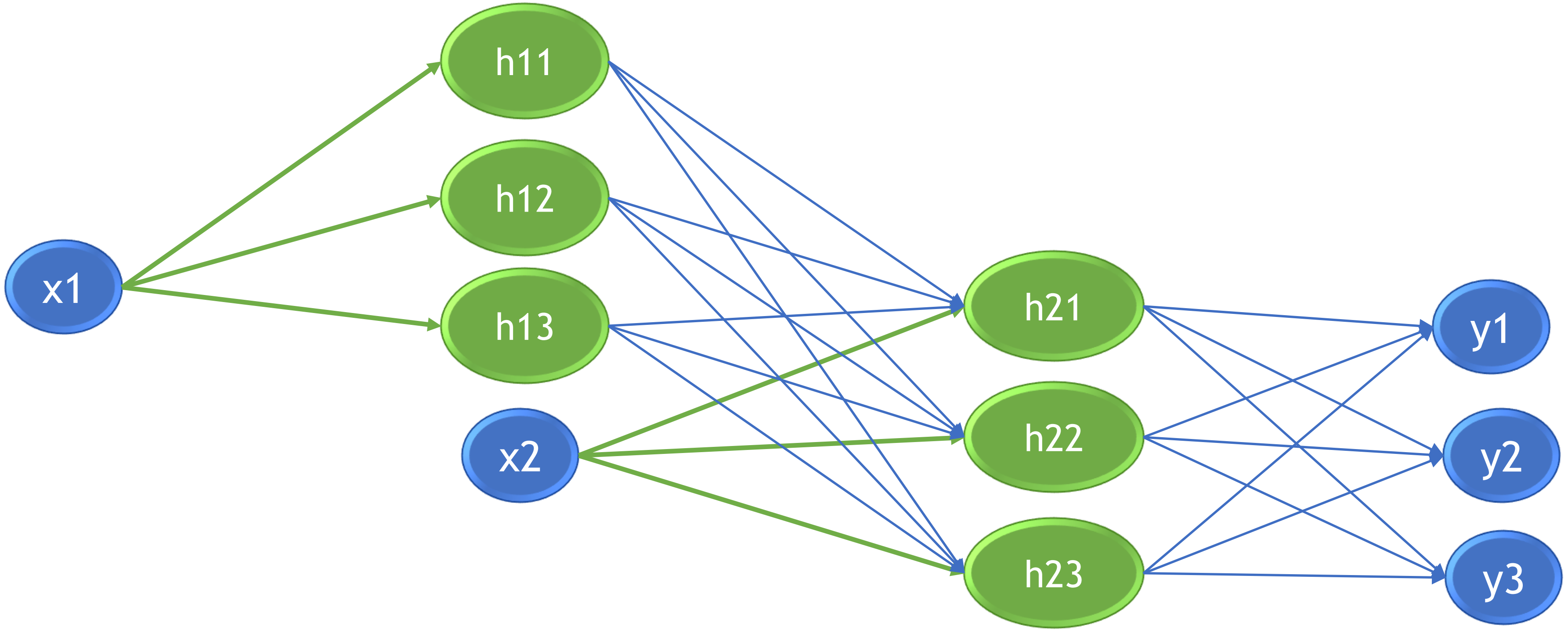
Sequential Models



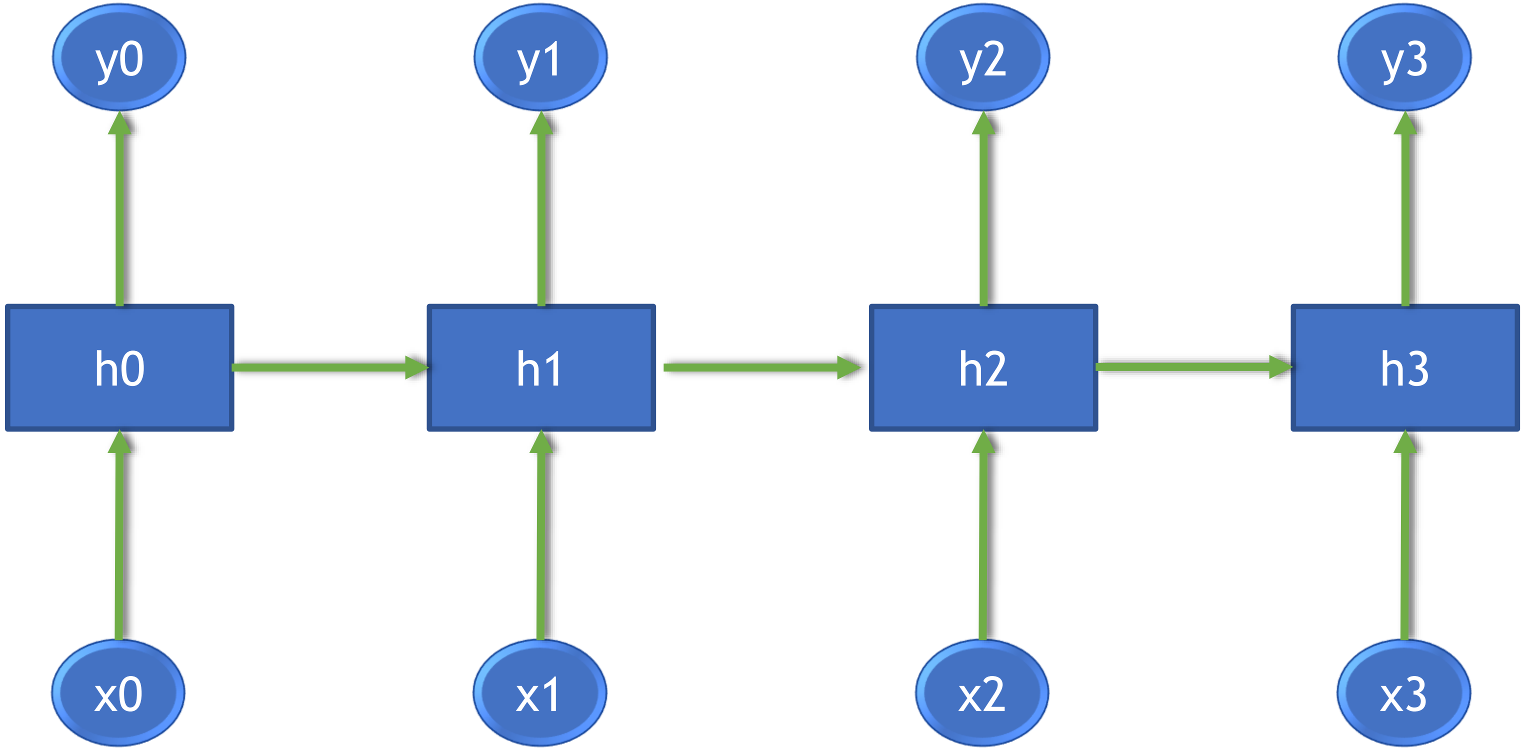
Sequential Models



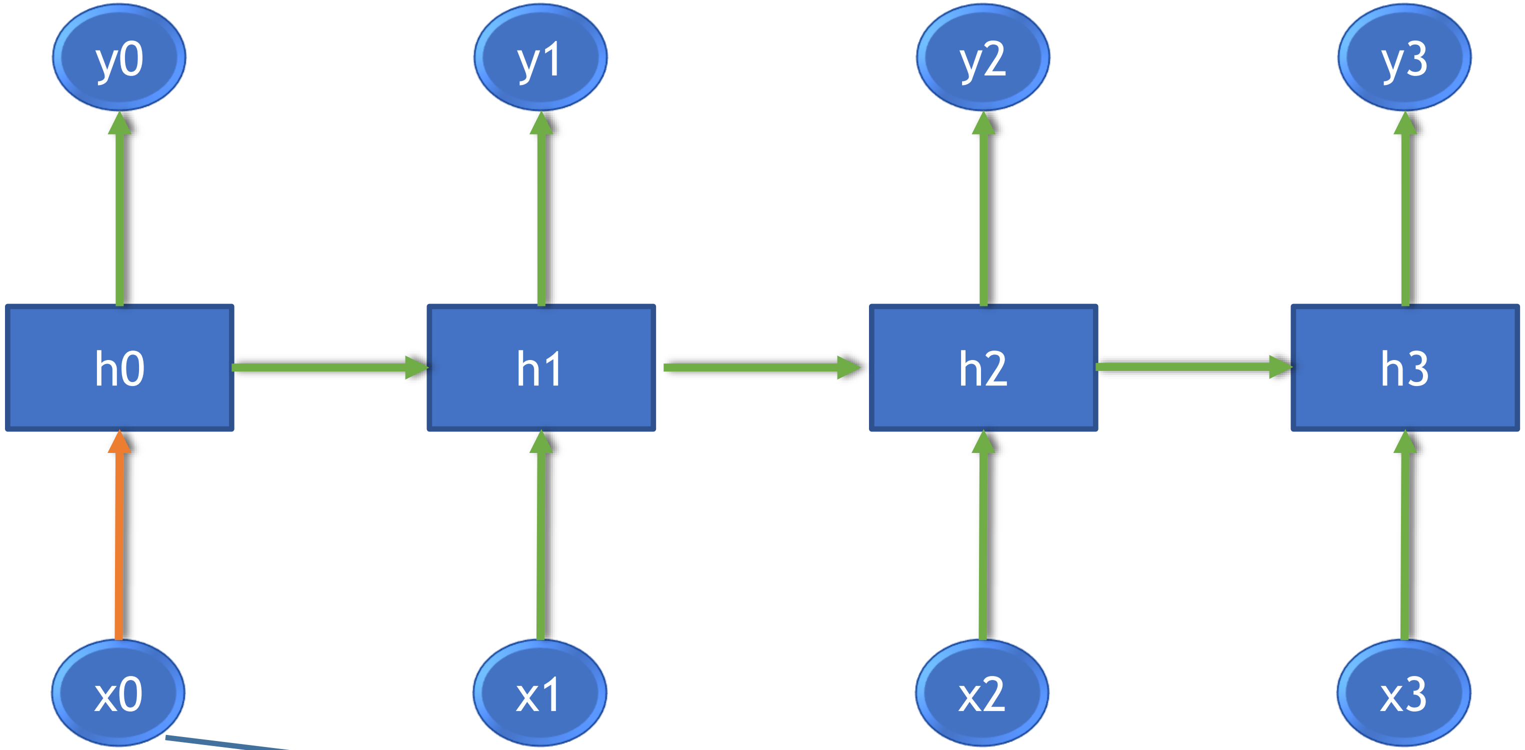
Sequential Models – two time steps



Sequential Models – Different Representation



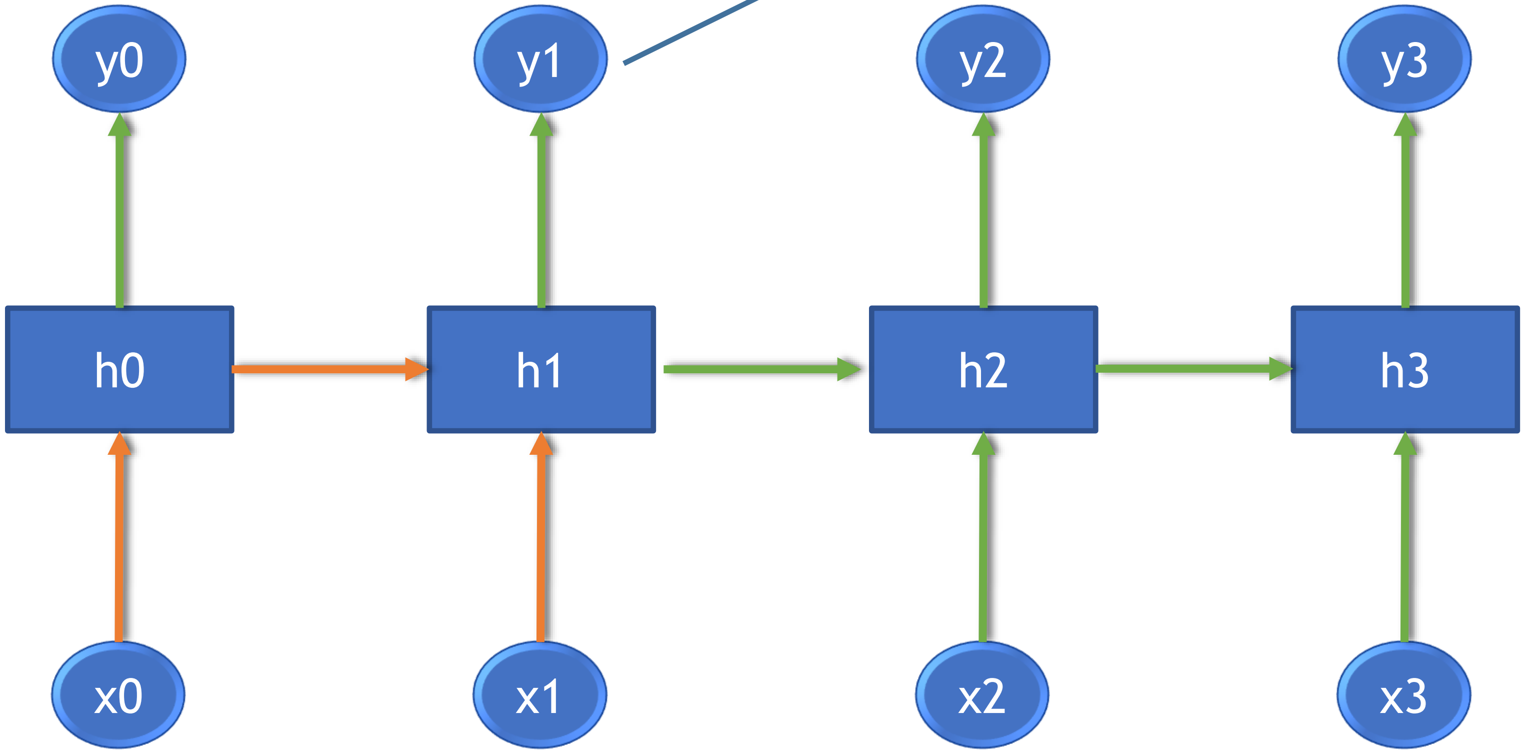
Sequential Models – Different Representation



This ANN model takes x0 as input and predicts y0 as output

Sequential Models – Different Representation

This ANN model takes x1 and output of h0 as input



LAB: Manual Sequential Model

- Imagine that you have just 3 words in every sentence. Given a word you need predict the next word. Given those two words, predict the third word.
- Take 4 gram data as input. Load the data.
- Model-1 – First word (x1)→ Second word (x2)
- Model-2 – {Hidden layer from model1 (h1) + Second word(x2) }→ Third word

LAB: Manual Sequential Model

- Model-1: Take first two columns from the data. Build an ANN model to predict the second word, given one word.
- Model-2: Take first three columns from the data. Build an ANN model to predict the third word, given first two words. Take the output of hidden node from the first model as input while predicting the third word
- Get the predictions for below data points
  - Love it
  - Love to

Approach: Manual Sequential Model

1. Download Data
2. Create Word to Number dictionary
3. Prepare one hot encoding vectors
4. Build model -1(m1) by taking word1 as input and word2 as output
5. Get m1 hidden layer output (h1)
6. Get word2 data
7. Append word2 and hidden layer output of previous model
8. Build model-2(m2) by taking word2 and h2 as input and word3 as output

Code: Manual Sequential Model

```
import pandas as pd
column_names = ['word1', 'word2', 'word3', 'word4']
gram2 = pd.read_csv('Datasets\\love_gram.txt', delimiter='\t', names=column_names)
gram2 = gram2.drop(['word4'], axis=1)
print("Few sample records from data \n", gram2.sample(10))
print("\nFrequency of word1 vlaues \n", gram2["word1"].value_counts())
print("\nFrequency of word2 vlaues \n", gram2["word2"].value_counts())
```

Data Importing

Dropping extra column

```
Few sample records from data
       word1 word2  word3
3413   love    to    see
1650   love  with   each
401   hated    to     do
4220   love    it   when
1684   love    to   have
33     hate    to     do
290    hate    to  think
3263   love    to    see
1773   love    to   find
3848   love    it   when
```

```
Frequency of word1 vlaues
 love       4327
loved       416
hate        400
hated        80
loves        72
lovely       24
loving       24
hates         8
Name: word1, dtype: int64
```

```
Frequency of word2 vlaues
 to          1866
it          1361
the          548
with         240
him          144
you          144
of           136
her          104
for           96
and           88
what          56
is            48
in            40
each          40
```

Code: Manual Sequential Model

- Finding unique words to create a word dictionary

```
chars = []
for i in list(gram2.columns.values):
    for j in pd.unique(gram2[i]):
        chars.append(j)
chars = np.unique(chars)

print('Count of unique words overall:', len(chars))
print('unique words list:', chars)
```

Iterating through each column to find unique words

```
Count of unique words overall: 139
unique words list: ['a' 'able' 'about' 'admit' 'affair' 'affection' 'all' 'and' 'another'
 'answer' 'as' 'at' 'be' 'because' 'being' 'better' 'between' 'bother'
 'break' 'care' 'cared' 'come' 'concern' 'country' 'cut' 'disappoint' 'do'
 'each' 'every' 'fact' 'feel' 'feeling' 'find' 'first' 'for' 'from' 'get'
 'go' 'god' 'going' 'got' 'hate' 'hated' 'hates' 'have' 'he' 'hear'
 'hearing' 'her' 'here' 'him' 'his' 'husband' 'i' 'idea' 'if' 'in'
 'interrupt' 'is' 'it' 'kind' 'know' 'leave' 'letter' 'life' 'like'
 'listen' 'look' 'lost' 'lot' 'love' 'loved' 'lovely' 'loves' 'loving'
 'make' 'makes' 'man' 'marriage' 'me' 'minute' 'more' 'most' 'much'
 'music' 'my' 'nature' 'neighbor' 'not' 'nothing' 'of' 'on' 'one' 'ones'
 'or' 'other' 'over' 'play' 'respect' 'say' 'see' 'sit' 'smell' 'so'
 'someone' 'song' 'sound' 'story' 'stronger' 'support' 'take' 'talk'
 'tell' 'than' 'that' 'the' 'them' 'they' 'think' 'this' 'thought' 'thy'
 'to' 'too' 'united' 'use' 'very' 'view' 'watch' 'way' 'we' 'what' 'when'
 'wife' 'will' 'with' 'work' 'you' 'your']
```

Code: Manual Sequential Model

- Creating a word to indices dictionary and reverse

```
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

print("char_indices dictionary \n",char_indices)
print("char_indices.keys \n", char_indices.keys())
print("char_indices.values \n", char_indices.values())
print("\n ##################################\n")
print("indices_char dictionary \n", indices_char)
print("indices_char keys \n",indices_char.keys())
print("indices_char values \n",indices_char.values())
```

words to indices and inverse

```
char_indices dictionary
 {'a': 0, 'able': 1, 'about': 2, 'admit': 3, 'affair': 4, 'affection': 5, 'all': 6, 'and': 7, 'another': 8, 'ans
wer': 9, 'as': 10, 'at': 11, 'be': 12, 'because': 13, 'being': 14, 'better': 15, 'between': 16, 'bother': 17, 'b
reak': 18, 'care': 19, 'cared': 20, 'come': 21, 'concern': 22, 'country': 23, 'cut': 24, 'disappoint': 25, 'do':
26, 'each': 27, 'every': 28, 'fact': 29, 'feel': 30, 'feeling': 31, 'find': 32, 'first': 33, 'for': 34, 'from':
35, 'get': 36, 'go': 37, 'god': 38, 'going': 39, 'got': 40, 'hate': 41, 'hated': 42, 'hates': 43, 'have': 44, 'h
e': 45, 'hear': 46, 'hearing': 47, 'her': 48, 'here': 49, 'him': 50, 'his': 51, 'husband': 52, 'i': 53, 'idea':
54, 'if': 55, 'in': 56, 'interrupt': 57, 'is': 58, 'it': 59, 'kind': 60, 'know': 61, 'leave': 62, 'letter': 63,
'life': 64, 'like': 65, 'listen': 66, 'look': 67, 'lost': 68, 'lot': 69, 'love': 70, 'loved': 71, 'lovely': 72,
'loves': 73, 'loving': 74, 'make': 75, 'makes': 76, 'man': 77, 'marriage': 78, 'me': 79, 'minute': 80, 'more': 8
1, 'most': 82, 'much': 83, 'music': 84, 'my': 85, 'nature': 86, 'neighbor': 87, 'not': 88, 'nothing': 89, 'of':
90, 'on': 91, 'one': 92, 'ones': 93, 'or': 94, 'other': 95, 'over': 96, 'play': 97, 'respect': 98, 'say': 99, 's
ee': 100, 'sit': 101, 'smell': 102, 'so': 103, 'someone': 104, 'song': 105, 'sound': 106, 'story': 107, 'stronge
r': 108, 'support': 109, 'take': 110, 'talk': 111, 'tell': 112, 'than': 113, 'that': 114, 'the': 115, 'them': 11
6, 'they': 117, 'think': 118, 'this': 119, 'thought': 120, 'thy': 121, 'to': 122, 'too': 123, 'united': 124, 'us
e': 125, 'very': 126, 'view': 127, 'watch': 128, 'way': 129, 'we': 130, 'what': 131, 'when': 132, 'wife': 133, '
will': 134, 'with': 135, 'work': 136, 'you': 137, 'your': 138}
char_indices.keys
 dict_keys(['a', 'able', 'about', 'admit', 'affair', 'affection', 'all', 'and', 'another', 'answer', 'as', 'at',
'be', 'because', 'being', 'better', 'between', 'bother', 'break', 'care', 'cared', 'come', 'concern', 'country',
'cut', 'disappoint', 'do', 'each', 'every', 'fact', 'feel', 'feeling', 'find', 'first', 'for', 'from', 'get', 'g
o', 'god', 'going', 'got', 'hate', 'hated', 'hates', 'have', 'he', 'hear', 'hearing', 'her', 'here', 'him', 'his
', 'husband', 'i', 'idea', 'if', 'in', 'interrupt', 'is', 'it', 'kind', 'know', 'leave', 'letter', 'life', 'like
', 'listen', 'look', 'lost', 'lot', 'love', 'loved', 'lovely', 'loves', 'loving', 'make', 'makes', 'man', 'marri
age', 'me', 'minute', 'more', 'most', 'much', 'music', 'my', 'nature', 'neighbor', 'not', 'nothing', 'of', 'on',
'one', 'ones', 'or', 'other', 'over', 'play', 'respect', 'say', 'see', 'sit', 'smell', 'so', 'someone', 'song',
'sound', 'story', 'stronger', 'support', 'take', 'talk', 'tell', 'than', 'that', 'the', 'them', 'they', 'think',
'this', 'thought', 'thy', 'to', 'too', 'united', 'use', 'very', 'view', 'watch', 'way', 'we', 'what', 'when', 'w
ife', 'will', 'with', 'work', 'you', 'your'])
char_indices.values
 dict_values([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 13
1, 132, 133, 134, 135, 136, 137, 138])
```

words to Indices

Indices to words

```
indices_char dictionary
 {0: 'a', 1: 'able', 2: 'about', 3: 'admit', 4: 'affair', 5: 'affection', 6: 'all', 7: 'and', 8: 'another', 9: '
answer', 10: 'as', 11: 'at', 12: 'be', 13: 'because', 14: 'being', 15: 'better', 16: 'between', 17: 'bother', 18
: 'break', 19: 'care', 20: 'cared', 21: 'come', 22: 'concern', 23: 'country', 24: 'cut', 25: 'disappoint', 26: '
do', 27: 'each', 28: 'every', 29: 'fact', 30: 'feel', 31: 'feeling', 32: 'find', 33: 'first', 34: 'for', 35: 'fr
om', 36: 'get', 37: 'go', 38: 'god', 39: 'going', 40: 'got', 41: 'hate', 42: 'hated', 43: 'hates', 44: 'have', 4
5: 'he', 46: 'hear', 47: 'hearing', 48: 'her', 49: 'here', 50: 'him', 51: 'his', 52: 'husband', 53: 'i', 54: 'id
ea', 55: 'if', 56: 'in', 57: 'interrupt', 58: 'is', 59: 'it', 60: 'kind', 61: 'know', 62: 'leave', 63: 'letter',
64: 'life', 65: 'like', 66: 'listen', 67: 'look', 68: 'lost', 69: 'lot', 70: 'love', 71: 'loved', 72: 'lovely',
73: 'loves', 74: 'loving', 75: 'make', 76: 'makes', 77: 'man', 78: 'marriage', 79: 'me', 80: 'minute', 81: 'more
', 82: 'most', 83: 'much', 84: 'music', 85: 'my', 86: 'nature', 87: 'neighbor', 88: 'not', 89: 'nothing', 90: 'o
f', 91: 'on', 92: 'one', 93: 'ones', 94: 'or', 95: 'other', 96: 'over', 97: 'play', 98: 'respect', 99: 'say', 10
0: 'see', 101: 'sit', 102: 'smell', 103: 'so', 104: 'someone', 105: 'song', 106: 'sound', 107: 'story', 108: 'st
ronger', 109: 'support', 110: 'take', 111: 'talk', 112: 'tell', 113: 'than', 114: 'that', 115: 'the', 116: 'them
', 117: 'they', 118: 'think', 119: 'this', 120: 'thought', 121: 'thy', 122: 'to', 123: 'too', 124: 'united', 125
: 'use', 126: 'very', 127: 'view', 128: 'watch', 129: 'way', 130: 'we', 131: 'what', 132: 'when', 133: 'wife', 1
34: 'will', 135: 'with', 136: 'work', 137: 'you', 138: 'your'}
indices_char keys
 dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27
, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55
, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83
, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 1
09, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131
, 132, 133, 134, 135, 136, 137, 138])
indices_char values
 dict_values(['a', 'able', 'about', 'admit', 'affair', 'affection', 'all', 'and', 'another', 'answer', 'as', 'at
', 'be', 'because', 'being', 'better', 'between', 'bother', 'break', 'care', 'cared', 'come', 'concern', 'countr
y', 'cut', 'disappoint', 'do', 'each', 'every', 'fact', 'feel', 'feeling', 'find', 'first', 'for', 'from', 'get'
, 'go', 'god', 'going', 'got', 'hate', 'hated', 'hates', 'have', 'he', 'hear', 'hearing', 'her', 'here', 'him',
'his', 'husband', 'i', 'idea', 'if', 'in', 'interrupt', 'is', 'it', 'kind', 'know', 'leave', 'letter', 'life', '
like', 'listen', 'look', 'lost', 'lot', 'love', 'loved', 'lovely', 'loves', 'loving', 'make', 'makes', 'man', 'm
arriage', 'me', 'minute', 'more', 'most', 'much', 'music', 'my', 'nature', 'neighbor', 'not', 'nothing', 'of', '
on', 'one', 'ones', 'or', 'other', 'over', 'play', 'respect', 'say', 'see', 'sit', 'smell', 'so', 'someone', 'so
ng', 'sound', 'story', 'stronger', 'support', 'take', 'talk', 'tell', 'than', 'that', 'the', 'them', 'they', 'th
ink', 'this', 'thought', 'thy', 'to', 'too', 'united', 'use', 'very', 'view', 'watch', 'way', 'we', 'what', 'whe
n', 'wife', 'will', 'with', 'work', 'you', 'your'])
```

Code: Manual Sequential Model

```
char_indices dictionary
 {'a': 0, 'able': 1, 'about': 2, 'admit': 3, 'affair': 4, 'affection': 5, 'all': 6, 'and': 7, 'another': 8, 'ans
wer': 9, 'as': 10, 'at': 11, 'be': 12, 'because': 13, 'being': 14, 'better': 15, 'between': 16, 'bother': 17, 'b
reak': 18, 'care': 19, 'cared': 20, 'come': 21, 'concern': 22, 'country': 23, 'cut': 24, 'disappoint': 25, 'do':
26, 'each': 27, 'every': 28, 'fact': 29, 'feel': 30, 'feeling': 31, 'find': 32, 'first': 33, 'for': 34, 'from':
35, 'get': 36, 'go': 37, 'god': 38, 'going': 39, 'got': 40, 'hate': 41, 'hated': 42, 'hates': 43, 'have': 44, 'h
e': 45, 'hear': 46, 'hearing': 47, 'her': 48, 'here': 49, 'him': 50, 'his': 51, 'husband': 52, 'i': 53, 'idea':
54, 'if': 55, 'in': 56, 'interrupt': 57, 'is': 58, 'it': 59, 'kind': 60, 'know': 61, 'leave': 62, 'letter': 63,
'life': 64, 'like': 65, 'listen': 66, 'look': 67, 'lost': 68, 'lot': 69, 'love': 70, 'loved': 71, 'lovely': 72,
'loves': 73, 'loving': 74, 'make': 75, 'makes': 76, 'man': 77, 'marriage': 78, 'me': 79, 'minute': 80, 'more': 8
1, 'most': 82, 'much': 83, 'music': 84, 'my': 85, 'nature': 86, 'neighbor': 87, 'not': 88, 'nothing': 89, 'of':
90, 'on': 91, 'one': 92, 'ones': 93, 'or': 94, 'other': 95, 'over': 96, 'play': 97, 'respect': 98, 'say': 99, 's
ee': 100, 'sit': 101, 'smell': 102, 'so': 103, 'someone': 104, 'song': 105, 'sound': 106, 'story': 107, 'stronge
r': 108, 'support': 109, 'take': 110, 'talk': 111, 'tell': 112, 'than': 113, 'that': 114, 'the': 115, 'them': 11
6, 'they': 117, 'think': 118, 'this': 119, 'thought': 120, 'thy': 121, 'to': 122, 'too': 123, 'united': 124, 'us
e': 125, 'very': 126, 'view': 127, 'watch': 128, 'way': 129, 'we': 130, 'what': 131, 'when': 132, 'wife': 133, '
will': 134, 'with': 135, 'work': 136, 'you': 137, 'your': 138}
char_indices.keys
 dict_keys(['a', 'able', 'about', 'admit', 'affair', 'affection', 'all', 'and', 'another', 'answer', 'as', 'at',
'be', 'because', 'being', 'better', 'between', 'bother', 'break', 'care', 'cared', 'come', 'concern', 'country',
'cut', 'disappoint', 'do', 'each', 'every', 'fact', 'feel', 'feeling', 'find', 'first', 'for', 'from', 'get', 'g
o', 'god', 'going', 'got', 'hate', 'hated', 'hates', 'have', 'he', 'hear', 'hearing', 'her', 'here', 'him', 'his
', 'husband', 'i', 'idea', 'if', 'in', 'interrupt', 'is', 'it', 'kind', 'know', 'leave', 'letter', 'life', 'like
', 'listen', 'look', 'lost', 'lot', 'love', 'loved', 'lovely', 'loves', 'loving', 'make', 'makes', 'man', 'marri
age', 'me', 'minute', 'more', 'most', 'much', 'music', 'my', 'nature', 'neighbor', 'not', 'nothing', 'of', 'on',
'one', 'ones', 'or', 'other', 'over', 'play', 'respect', 'say', 'see', 'sit', 'smell', 'so', 'someone', 'song',
'sound', 'story', 'stronger', 'support', 'take', 'talk', 'tell', 'than', 'that', 'the', 'them', 'they', 'think',
'this', 'thought', 'thy', 'to', 'too', 'united', 'use', 'very', 'view', 'watch', 'way', 'we', 'what', 'when', 'w
ife', 'will', 'with', 'work', 'you', 'your'])
char_indices.values
 dict_values([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 13
1, 132, 133, 134, 135, 136, 137, 138])
```

words to Indices

Code: Manual Sequential Model

Indices to words

```
indices_char dictionary
 {0: 'a', 1: 'able', 2: 'about', 3: 'admit', 4: 'affair', 5: 'affection', 6: 'all', 7: 'and', 8: 'another', 9: '
answer', 10: 'as', 11: 'at', 12: 'be', 13: 'because', 14: 'being', 15: 'better', 16: 'between', 17: 'bother', 18
: 'break', 19: 'care', 20: 'cared', 21: 'come', 22: 'concern', 23: 'country', 24: 'cut', 25: 'disappoint', 26: '
do', 27: 'each', 28: 'every', 29: 'fact', 30: 'feel', 31: 'feeling', 32: 'find', 33: 'first', 34: 'for', 35: 'fr
om', 36: 'get', 37: 'go', 38: 'god', 39: 'going', 40: 'got', 41: 'hate', 42: 'hated', 43: 'hates', 44: 'have', 4
5: 'he', 46: 'hear', 47: 'hearing', 48: 'her', 49: 'here', 50: 'him', 51: 'his', 52: 'husband', 53: 'i', 54: 'id
ea', 55: 'if', 56: 'in', 57: 'interrupt', 58: 'is', 59: 'it', 60: 'kind', 61: 'know', 62: 'leave', 63: 'letter',
64: 'life', 65: 'like', 66: 'listen', 67: 'look', 68: 'lost', 69: 'lot', 70: 'love', 71: 'loved', 72: 'lovely',
73: 'loves', 74: 'loving', 75: 'make', 76: 'makes', 77: 'man', 78: 'marriage', 79: 'me', 80: 'minute', 81: 'more
', 82: 'most', 83: 'much', 84: 'music', 85: 'my', 86: 'nature', 87: 'neighbor', 88: 'not', 89: 'nothing', 90: 'o
f', 91: 'on', 92: 'one', 93: 'ones', 94: 'or', 95: 'other', 96: 'over', 97: 'play', 98: 'respect', 99: 'say', 10
0: 'see', 101: 'sit', 102: 'smell', 103: 'so', 104: 'someone', 105: 'song', 106: 'sound', 107: 'story', 108: 'st
ronger', 109: 'support', 110: 'take', 111: 'talk', 112: 'tell', 113: 'than', 114: 'that', 115: 'the', 116: 'them
', 117: 'they', 118: 'think', 119: 'this', 120: 'thought', 121: 'thy', 122: 'to', 123: 'too', 124: 'united', 125
: 'use', 126: 'very', 127: 'view', 128: 'watch', 129: 'way', 130: 'we', 131: 'what', 132: 'when', 133: 'wife', 1
34: 'will', 135: 'with', 136: 'work', 137: 'you', 138: 'your'}
indices_char keys
 dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27
, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55
, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83
, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 1
09, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131
, 132, 133, 134, 135, 136, 137, 138])
indices_char values
 dict_values(['a', 'able', 'about', 'admit', 'affair', 'affection', 'all', 'and', 'another', 'answer', 'as', 'at
', 'be', 'because', 'being', 'better', 'between', 'bother', 'break', 'care', 'cared', 'come', 'concern', 'countr
y', 'cut', 'disappoint', 'do', 'each', 'every', 'fact', 'feel', 'feeling', 'find', 'first', 'for', 'from', 'get'
, 'go', 'god', 'going', 'got', 'hate', 'hated', 'hates', 'have', 'he', 'hear', 'hearing', 'her', 'here', 'him',
'his', 'husband', 'i', 'idea', 'if', 'in', 'interrupt', 'is', 'it', 'kind', 'know', 'leave', 'letter', 'life', '
like', 'listen', 'look', 'lost', 'lot', 'love', 'loved', 'lovely', 'loves', 'loving', 'make', 'makes', 'man', 'm
arriage', 'me', 'minute', 'more', 'most', 'much', 'music', 'my', 'nature', 'neighbor', 'not', 'nothing', 'of', '
on', 'one', 'ones', 'or', 'other', 'over', 'play', 'respect', 'say', 'see', 'sit', 'smell', 'so', 'someone', 'so
ng', 'sound', 'story', 'stronger', 'support', 'take', 'talk', 'tell', 'than', 'that', 'the', 'them', 'they', 'th
ink', 'this', 'thought', 'thy', 'to', 'too', 'united', 'use', 'very', 'view', 'watch', 'way', 'we', 'what', 'whe
n', 'wife', 'will', 'with', 'work', 'you', 'your'])
```

Code: Manual Sequential Model

- One Hot encoded representation

```
#Lets take example of two words
print("The word is -->"+gram2['word1'][0])
print("The one hot encoded version of the word is \n",X1[0])

print("\nThe word is --> "+gram2['word1'][500])
print("The one hot encoded version of the word is \n",X1[500])
```

```
The word is -->hate
The one hot encoded version of the word is
 [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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 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. 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. 0.]

The word is --> love
The one hot encoded version of the word is
 [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. 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. 0. 0. 0. 0. 0. 0. 1. 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. 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. 0. 0. 0. 0.]
```

This is how word ‘hate’ looks like after conversion to indices and then to one-hot encoding

Hate is at index 41, Which is denoted as 1, rest 138 values being zero

Code: Manual Sequential Model

- Defining our model

```
model1 = Sequential()
model1.add(Dense(10, input_dim=X1.shape[1], activation='sigmoid'))
model1.add(Dense(y1.shape[1] ,kernel_initializer="uniform", activation='softmax'))
model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                1400
_________________________________________________________________
dense_2 (Dense)              (None, 139)               1529
=================================================================
Total params: 2,929
Trainable params: 2,929
Non-trainable params: 0
_________________________________________________________________
```

Hidden node in layer1 = 10
input shape: X1.shape[1] = 139
Activation function = sigmoid

Output layer nodes = Output shape:y1.shape[1]=139

Activation function = SoftMax for probability of each char

Code: Manual Sequential Model

- Compiling and training the model

```
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model1.fit(X1, y1, epochs=20, batch_size=50,  verbose=1)

scores = model1.evaluate(X1, y1)
print("%s: %.2f%%" % (model1.metrics_names[1], scores[1]*100))
```

```
5351/5351 [==============================] - 0s 38us/step - loss: 0.0224 - acc: 0.9928
Epoch 13/20
5351/5351 [==============================] - 0s 35us/step - loss: 0.0224 - acc: 0.9928
Epoch 14/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0223 - acc: 0.9928
Epoch 15/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0223 - acc: 0.9928
Epoch 16/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0223 - acc: 0.9928
Epoch 17/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0223 - acc: 0.9928
Epoch 18/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0223 - acc: 0.9928
Epoch 19/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0223 - acc: 0.9928
Epoch 20/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0223 - acc: 0.9928
5351/5351 [==============================] - 0s 29us/step
acc: 99.28%
```

Loss function = 'binary_crossentropy'(for 0,1 kind output)
Optimizer = 'adam'
Scoring matrix = 'Accuracy'

Training for:
20 Epochs
With a batch_size of 50

Code: Manual Sequential Model

- Getting intermediate hidden states from model 1 to be appended to word2

```
model1h = Sequential()
model1h.add(Dense(10, input_dim=y1.shape[1], weights=model1.layers[0].get_weights()))
model1h.add(Activation('sigmoid'))
```

```
# Getting the hidden layer activations
h1 = model1h.predict(X1)
#peak into our hidden layer activations
print(h1.shape)
print(h1[:5])
```

```
(5351, 10)
[[0.8334678  0.83259743 0.8430732  0.8439461  0.85551775 0.84998465
  0.8653672  0.85076314 0.86628264 0.8691472 ]
 [0.8334678  0.83259743 0.8430732  0.8439461  0.85551775 0.84998465
  0.8653672  0.85076314 0.86628264 0.8691472 ]
 [0.8334678  0.83259743 0.8430732  0.8439461  0.85551775 0.84998465
  0.8653672  0.85076314 0.86628264 0.8691472 ]
 [0.8334678  0.83259743 0.8430732  0.8439461  0.85551775 0.84998465
  0.8653672  0.85076314 0.86628264 0.8691472 ]
 [0.8334678  0.83259743 0.8430732  0.8439461  0.85551775 0.84998465
  0.8653672  0.85076314 0.86628264 0.8691472 ]]
```

Hidden state nodes: 10

Input_dim = y1.shape[1]: same as model1 output shape

Initialized weights for hidden states = output weights from model1

Getting the hidden state nodes values

Code: Manual Sequential Model

- Preparing the data from model2, appending hidden states with word2

```
X2_2 = gram2['word2'].map(char_indices)
X2_2 = keras.utils.to_categorical(np.array(X2_2), num_classes=len(char_indices))
```

```
X2 = np.append(h1,X2_2, axis=1)
print(X2.shape)
```

```
(5351, 149)
```

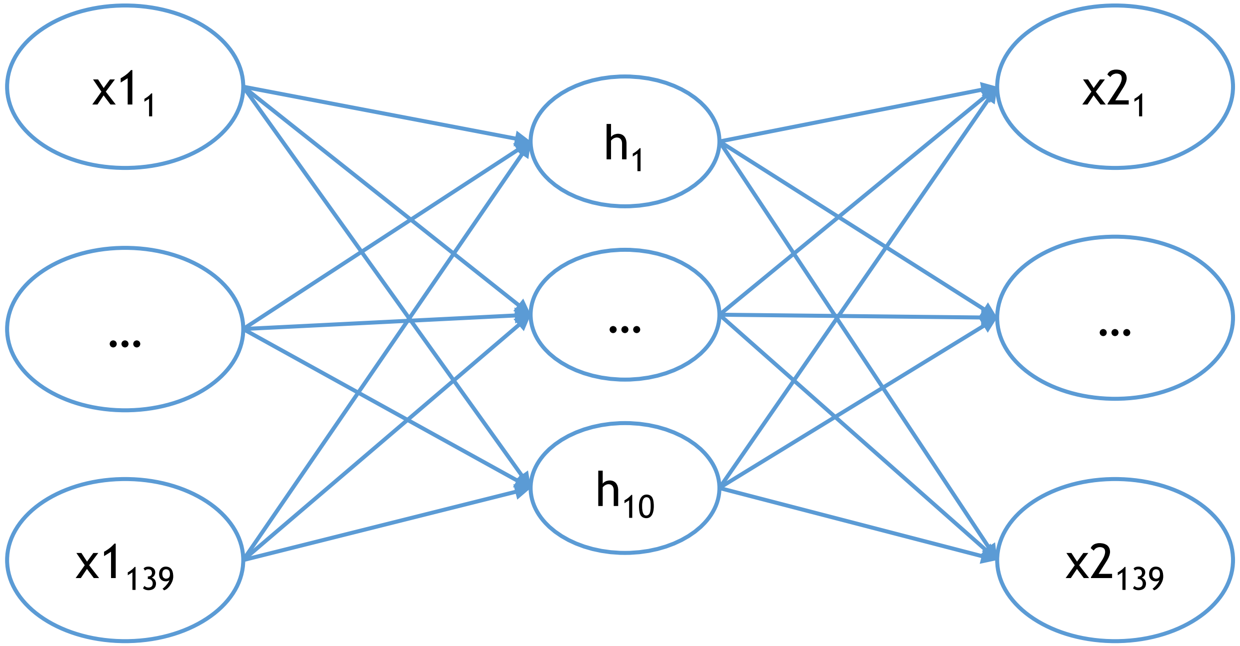
```
y2 = gram2['word3'].map(char_indices)
y2 = keras.utils.to_categorical(np.array(y2), num_classes=len(char_indices))
print(y2.shape)
print(y2[:2])
```

```
(5351, 139)
[[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. 1. 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. 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. 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. 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. 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.
  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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0.]]
```

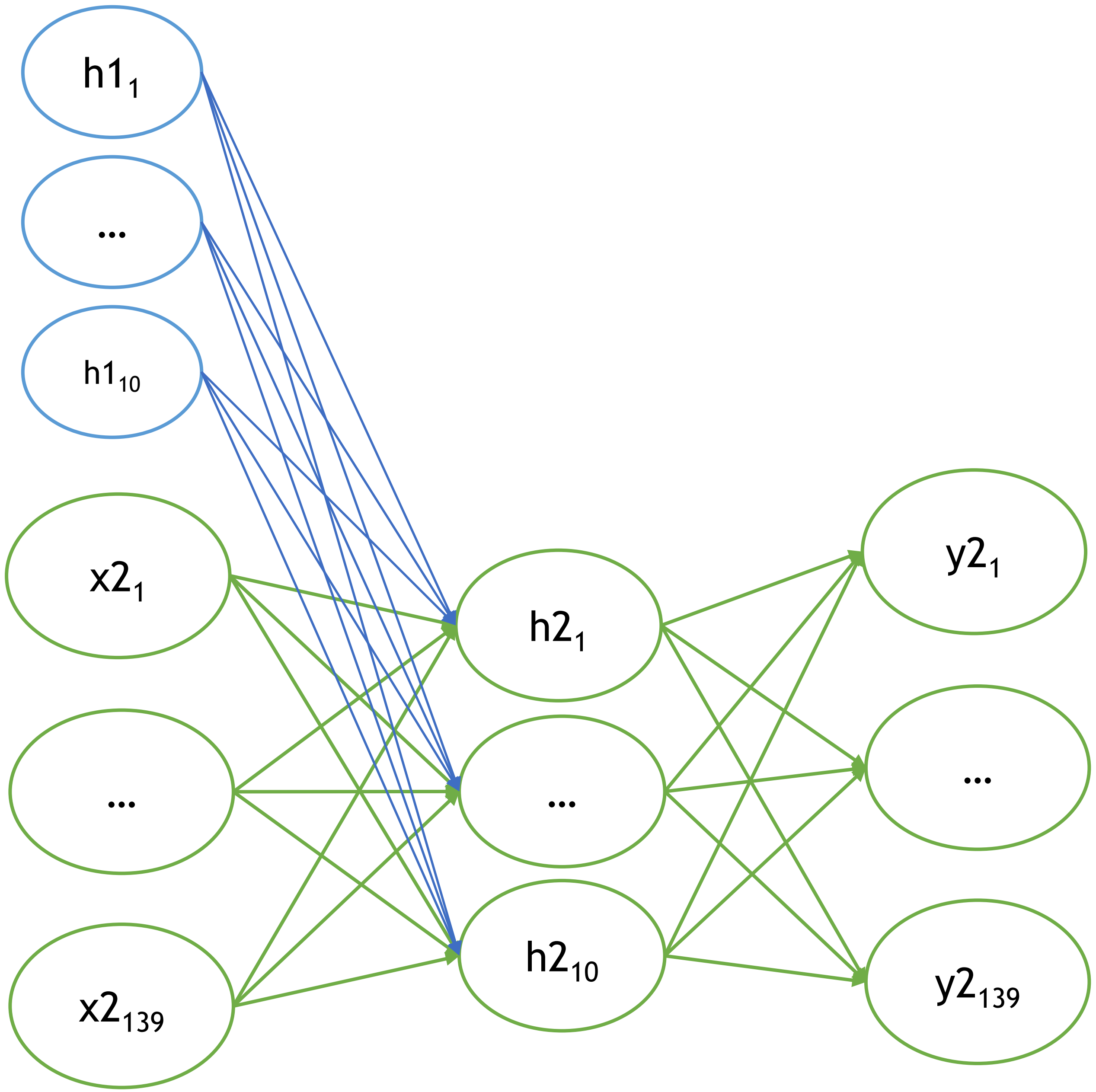
Mapping and Onehot encode word2

Appending word2 with Hidden states from model1, This is Input for model2

Mapping and Onehot encode word3, which is Output for model2



x1₁
...
x1₁₃₉
h₁
...
h₁₀
x2₁
...
x2₁₃₉



h1₁
...
h1₁₀
x2₁
...
x2₁₃₉
h2₁
...
h2₁₀
y2₁
...
y2₁₃₉

Code: Manual Sequential Model

- Defining model2

```
model2 = Sequential()
model2.add(Dense(10, input_dim=X2.shape[1], activation='sigmoid'))
model2.add(Dense(y2.shape[1], kernel_initializer='uniform', activation='softmax'))
model2.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 10)                1500
_________________________________________________________________
dense_5 (Dense)              (None, 139)               1529
=================================================================
Total params: 3,029
Trainable params: 3,029
Non-trainable params: 0
_________________________________________________________________
```

Nodes in layer1 = 10

input shape: X2.shape[1] = 10(from Hidden state)+139(from word2)

Activation function = sigmoid

Output shape: output shape of word3

Code: Manual Sequential Model

- Compiling and training model2

```
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model2.fit(X2, y2, epochs=20, batch_size=50,  verbose=1)

scores = model2.evaluate(X2, y2)
print("%s: %.2f%%" % (model2.metrics_names[1], scores[1]*100))
```

```
5351/5351 [==============================] - 0s 35us/step - loss: 0.0281 - acc: 0.9928
Epoch 13/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0276 - acc: 0.9943
Epoch 14/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0271 - acc: 0.9945
Epoch 15/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0266 - acc: 0.9945
Epoch 16/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0260 - acc: 0.9945
Epoch 17/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0254 - acc: 0.9945
Epoch 18/20
5351/5351 [==============================] - 0s 35us/step - loss: 0.0249 - acc: 0.9945
Epoch 19/20
5351/5351 [==============================] - 0s 38us/step - loss: 0.0244 - acc: 0.9945
Epoch 20/20
5351/5351 [==============================] - 0s 41us/step - loss: 0.0239 - acc: 0.9945
5351/5351 [==============================] - 0s 32us/step
acc: 99.45%
```

Loss function = 'binary_crossentropy'(for 0,1 type output)
Optimizer = 'adam'
Scoring matrix = 'Accuracy'

Training for:
20 Epochs
With a batch_size of 50

Code: Manual Sequential Model

- Custom output function to get combined results from model1 and model2

```
def two_step_pred(words_in):

    index_input=char_indices[words_in[0]]
    indices_in = keras.utils.to_categorical(index_input, num_classes=len(char_indices))
    indices_in=indices_in.reshape(1,len(char_indices))
    h1_test = model1h.predict(indices_in)

    index_input2=char_indices[words_in[1]]
    indices_in2 = keras.utils.to_categorical(index_input2, num_classes=len(char_indices))
    indices_in2= indices_in2.reshape(1,len(char_indices))
    X2_test = np.append(h1_test, indices_in2, axis=1)

    yhat = model2.predict_classes(X2_test)
    return indices_char[yhat[0]]
```

First word, getting one hot encoding,
Predicting hidden state nodes

Appending hidden state to encoded word2

Making predictions using combination of:
Hidden states from M1+Word2

```
print(two_step_pred(['love', 'it']))
```

```
when
```

```
print(two_step_pred(['love', 'to']))
```

```
see
```

Making the predictions

The sequential models

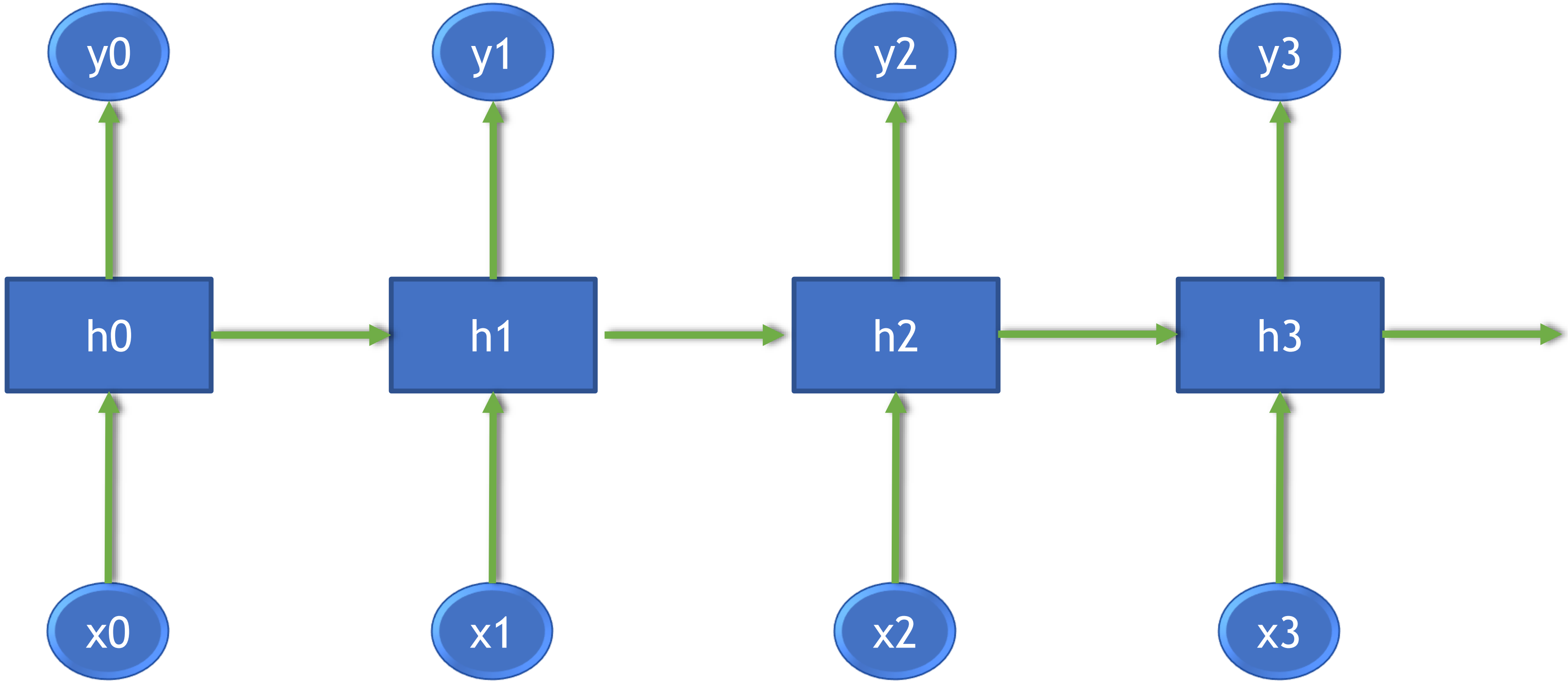
- We manually created two ANNs and combined them.
- Since we are working with only 3 words, we crated two ANN models.
- How may ANNs are required if are working senesces having 4 words.

- What if there is no limitation on the number of words.
- Is there a way to automatically build the sequential models for any variable input size?

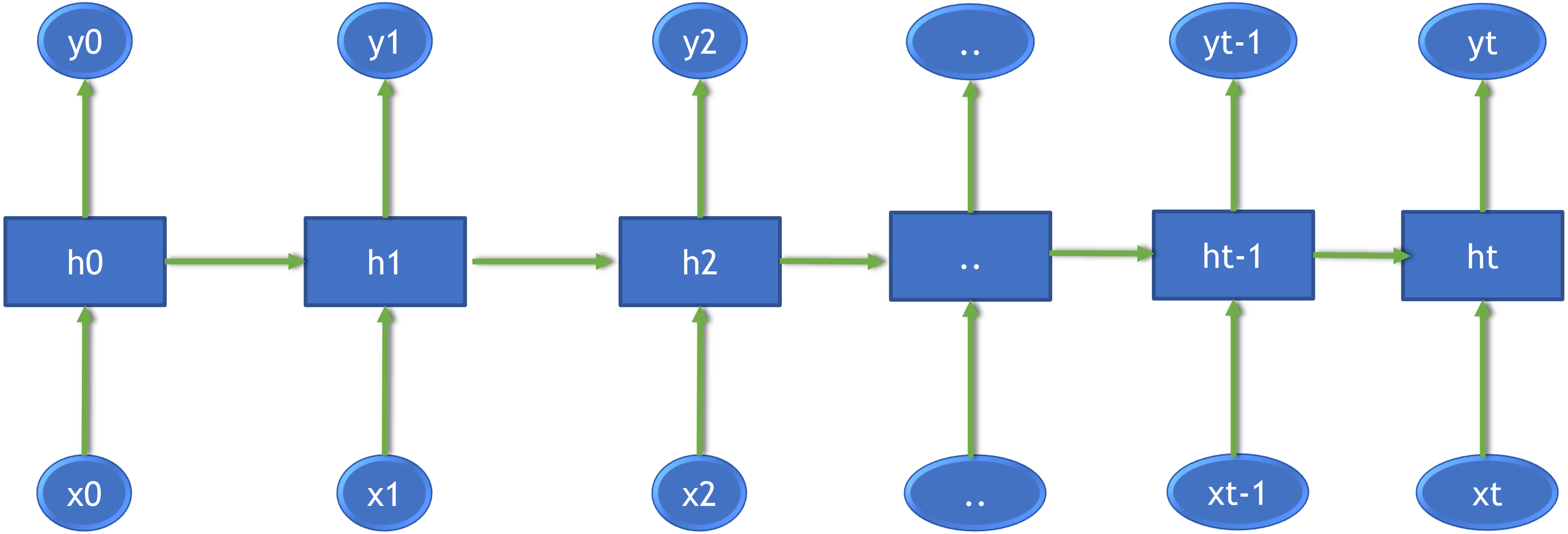
RNNs - The (programmed)sequential models

- Recurrent Neural Networks
- RNNs are very similar to the manual sequential model that we built in the previous lab
- RNNs are built for sequential input data
- RNNs will automatically build multiple ANNs in sequence
- RNNs also take care of sequential dependency
- RNNs are ANNs with memory
- RNN builds multiple ANN models sequentially and connect the ANN at time ‘t’ with ANN at time ‘t+1’

RNN Architecture



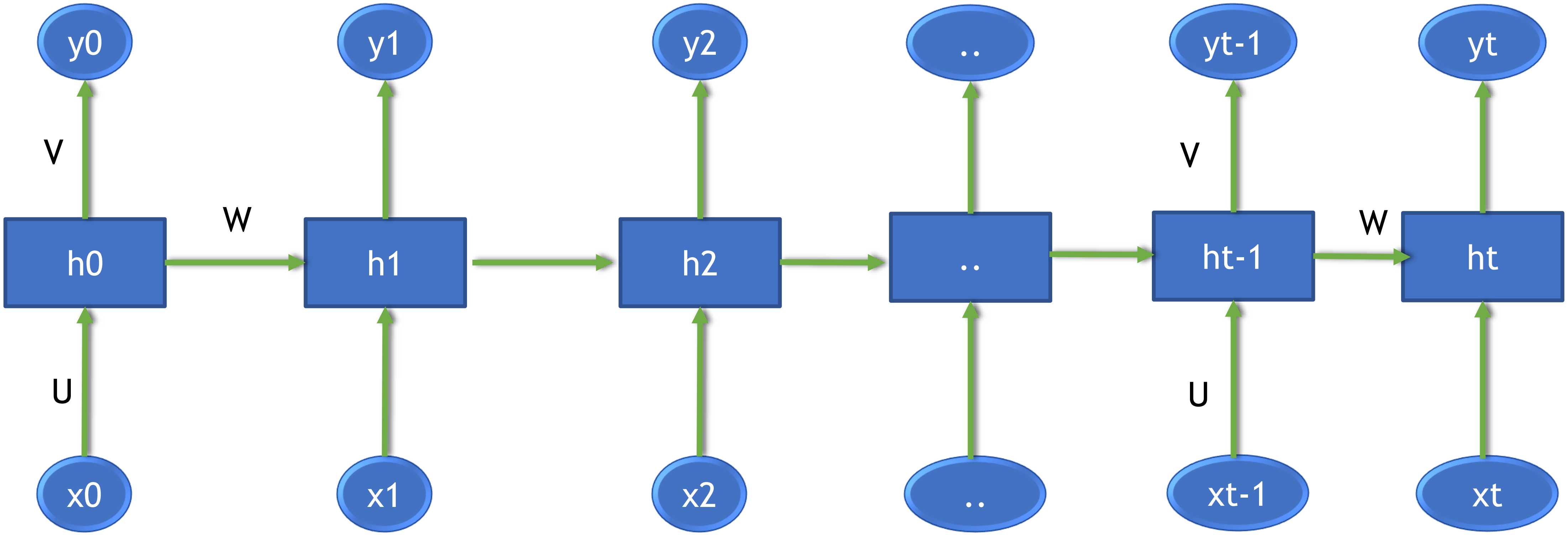
RNN – Layered ANN's over time



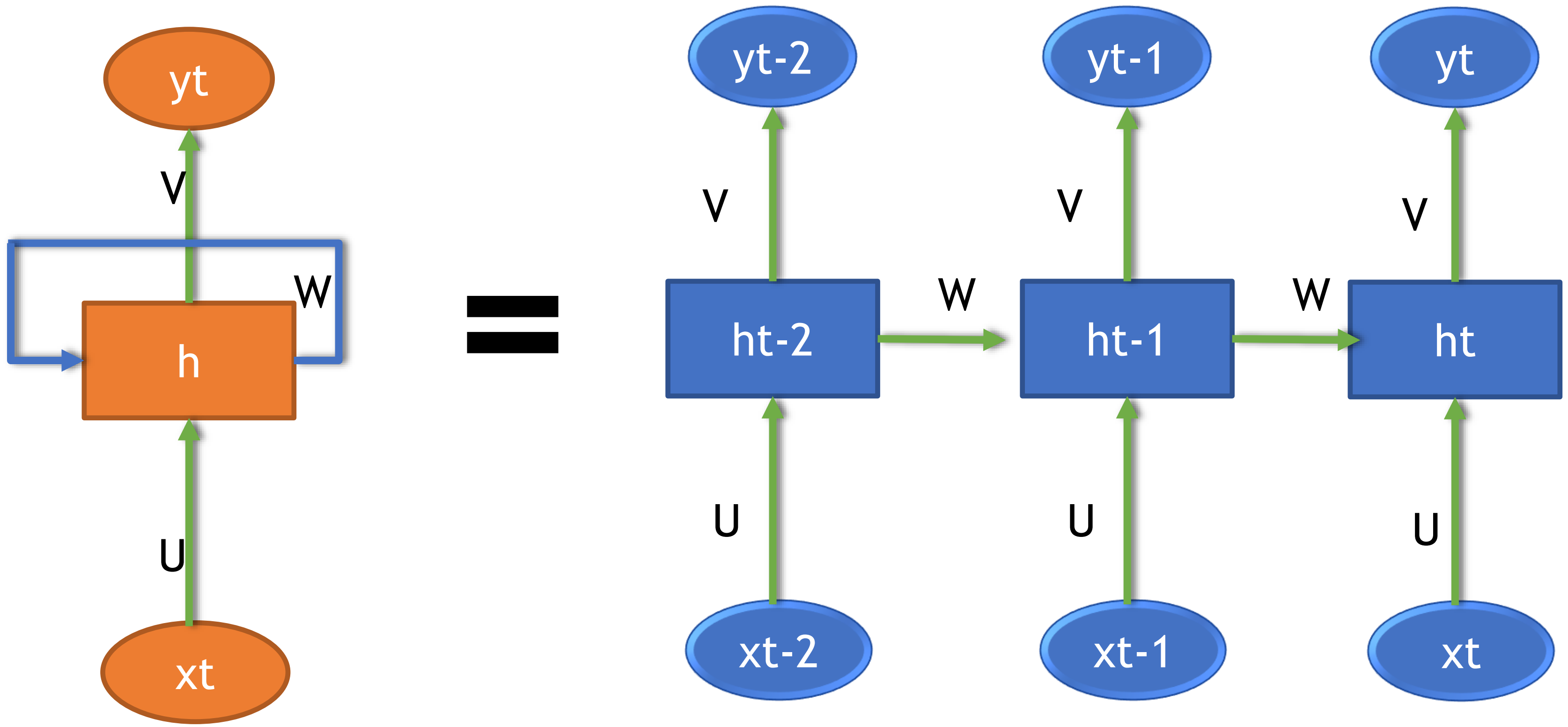
RNN – Layered ANN's over time

- At every time point 't', RNN is taking input xt and output from previous hidden state ht-1
- There are three different weights that we need to calculate
- Weights going from xt to ht (Ut)
- Weights going from ht to yt (Vt)
- Weights going from ht-1 to ht(Wt)
- Remember …ANN uses back propagation to find its weights. RNN uses BPTT (back propagation through time) to find all these weights(U,V,W) automatically

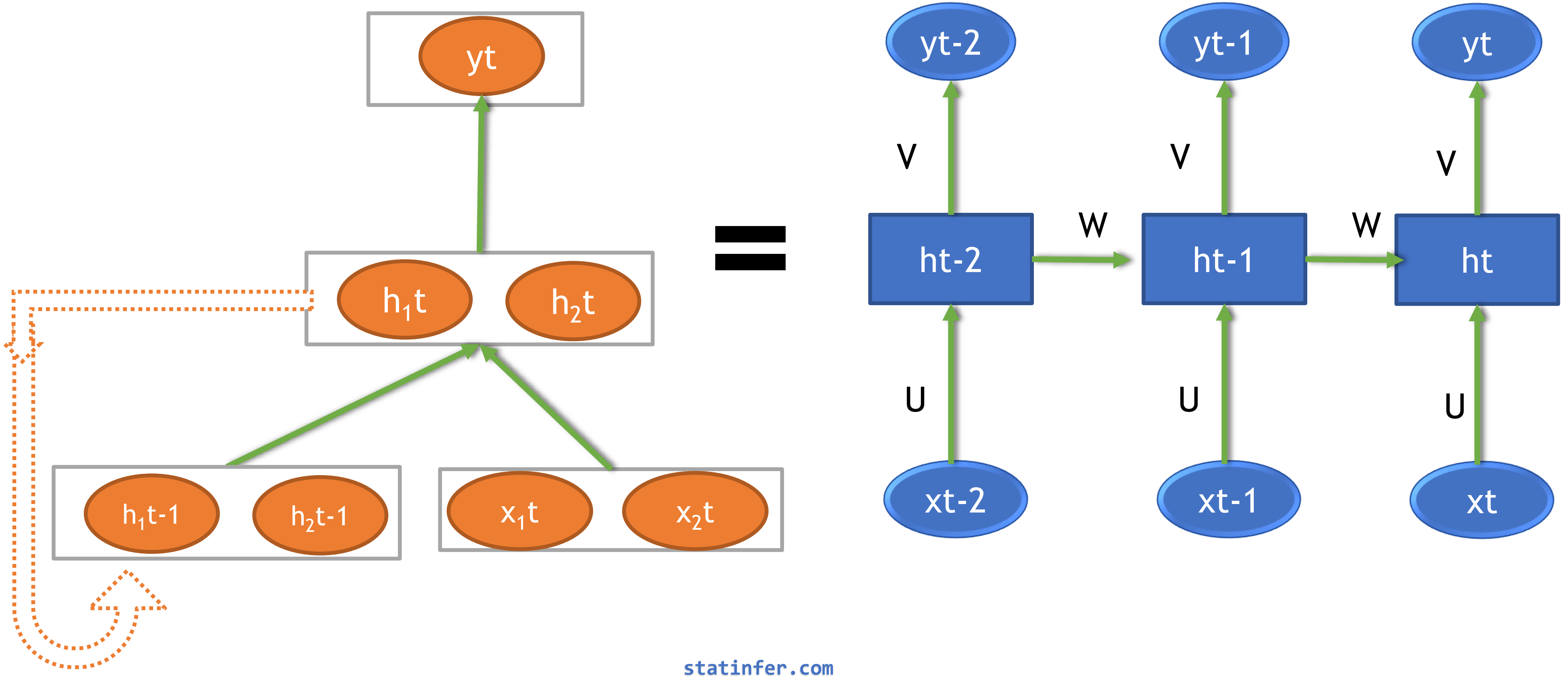
RNN Architecture



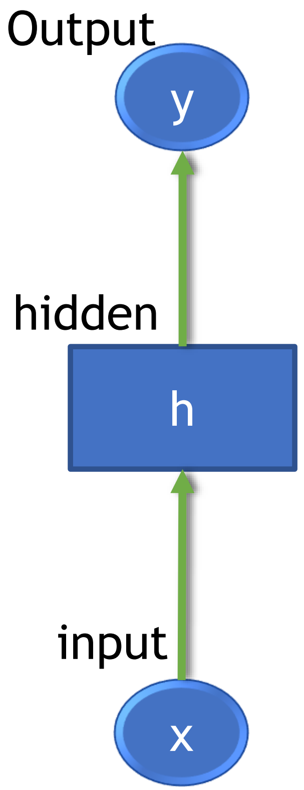
Many ways to visualize RNN models



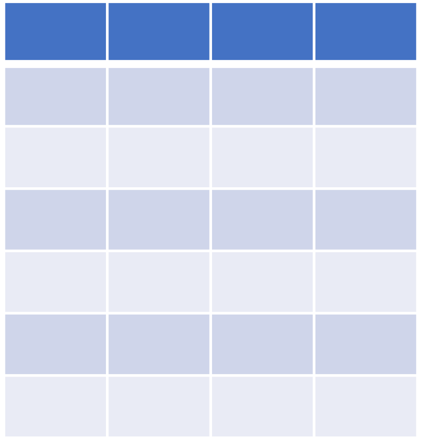
Many ways to visualize RNN models

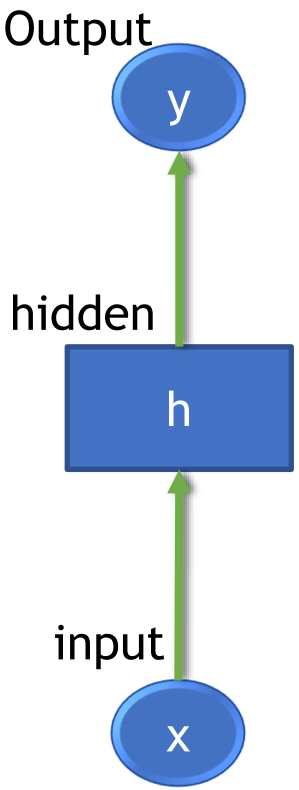


Recap-Back Propagation in ANN



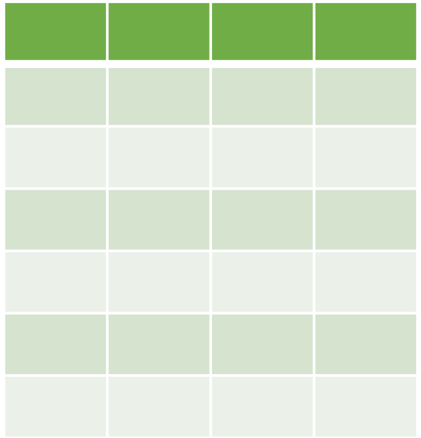
Recap-Back Propagation in ANN

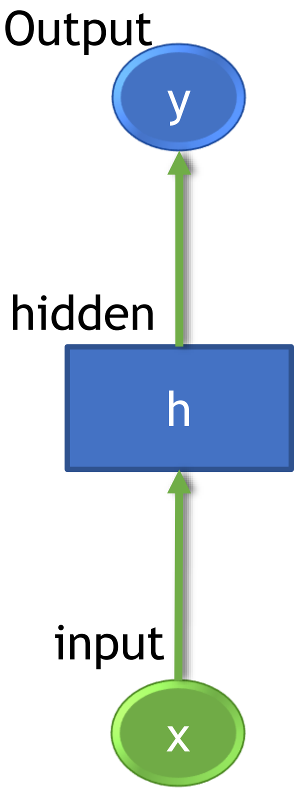




Input training data and perform feed forward calculations

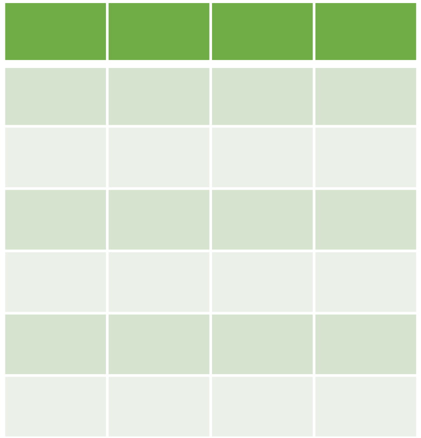
Recap-Back Propagation in ANN

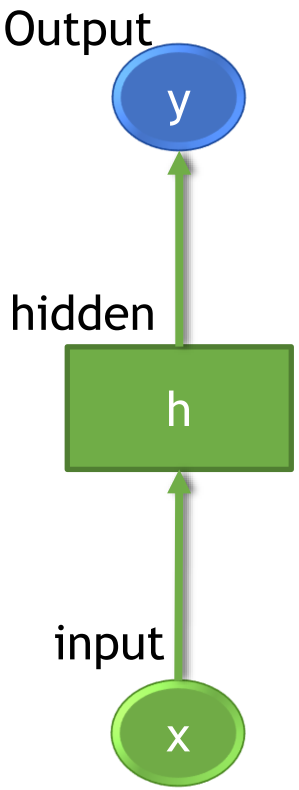




Input training data and perform feed forward calculations

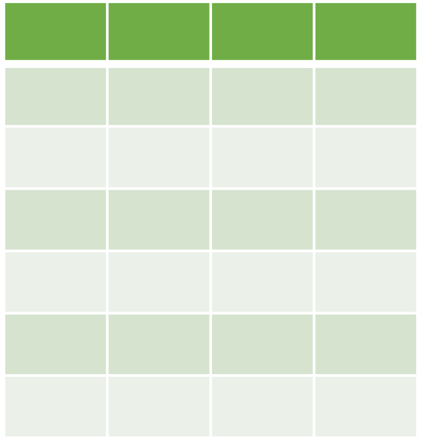
Recap-Back Propagation in ANN

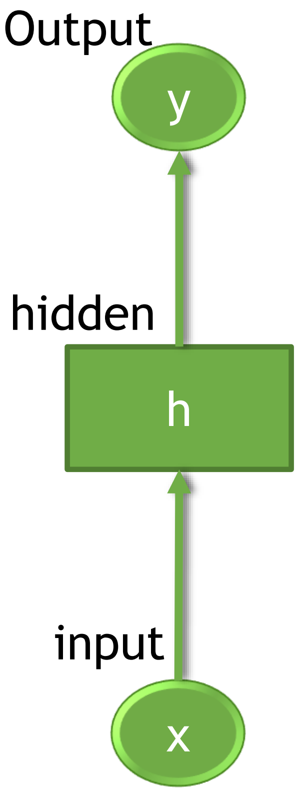




Input training data and perform feed forward calculations

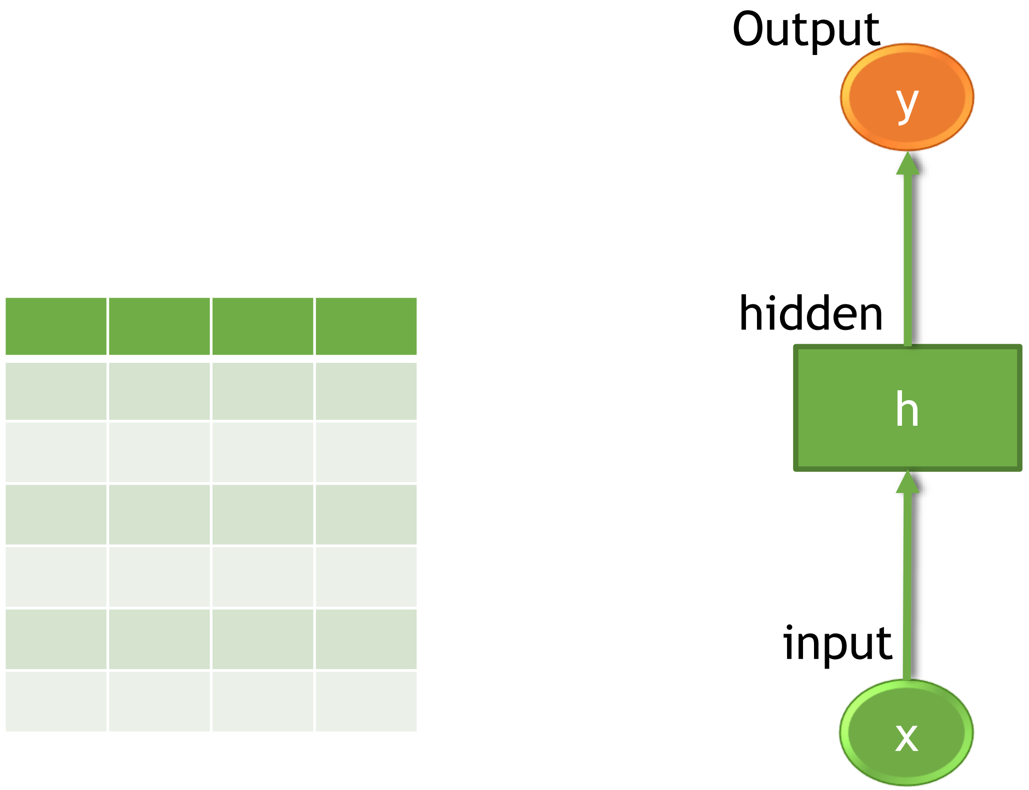
Recap-Back Propagation in ANN





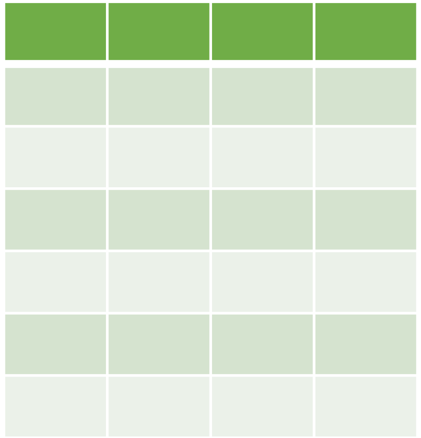
Input training data and perform feed forward calculations

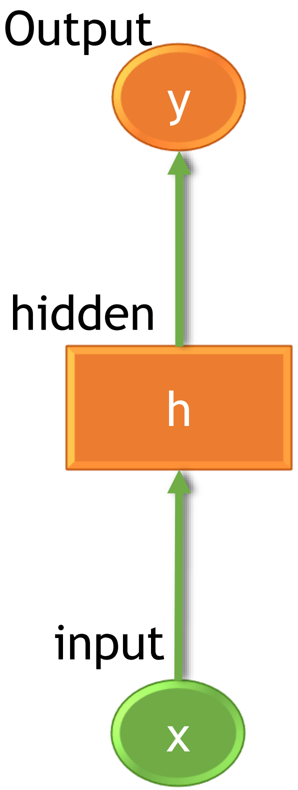
Recap-Back Propagation in ANN



Calculate error at output layer and propagate it backwords.

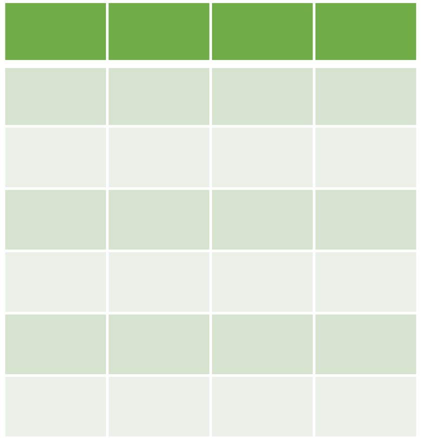
Recap-Back Propagation in ANN

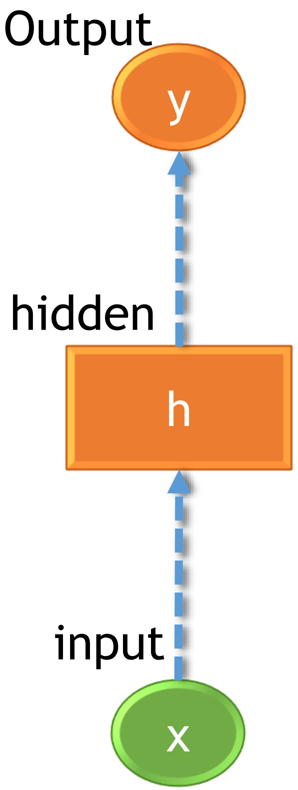




Calculate error at output layer and propagate it backwords.

Recap-Back Propagation in ANN





Weight corrections to reduce the error

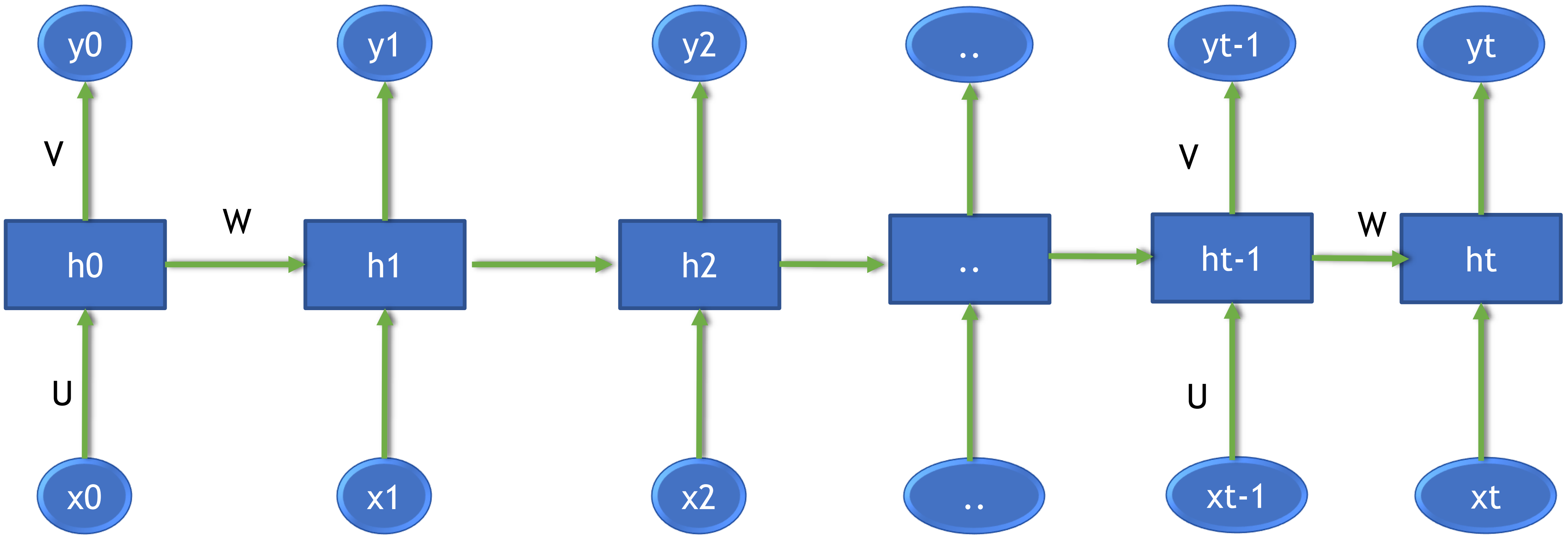
RNN – Back Propagation Through Time

- RNN has multiple ANNs stacked over time.
- RNN incorporates sequential back propagation
- Feed Forward is done at discrete time points
- Error is calculated at the final node (say time t)
- Back propagated into previous layers through all previous time points

This algorithm is known as Back Propagation Through Time

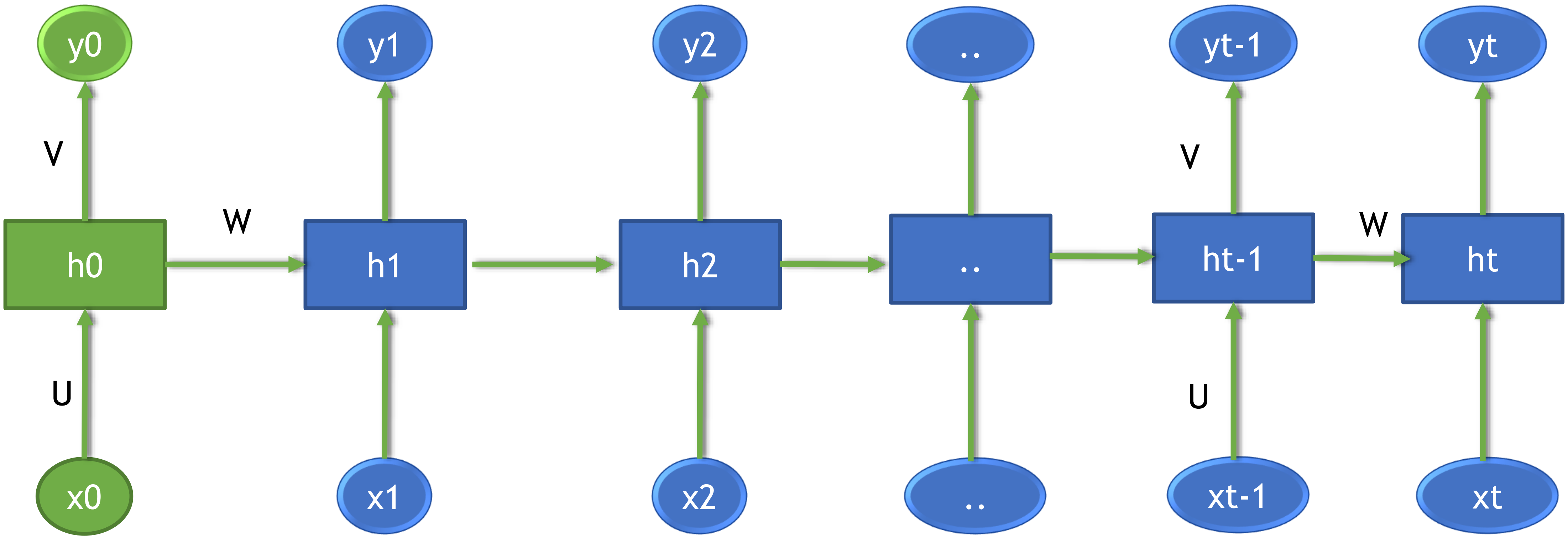
Back Propagation Through Time

Input training data and perform feed forward calculations



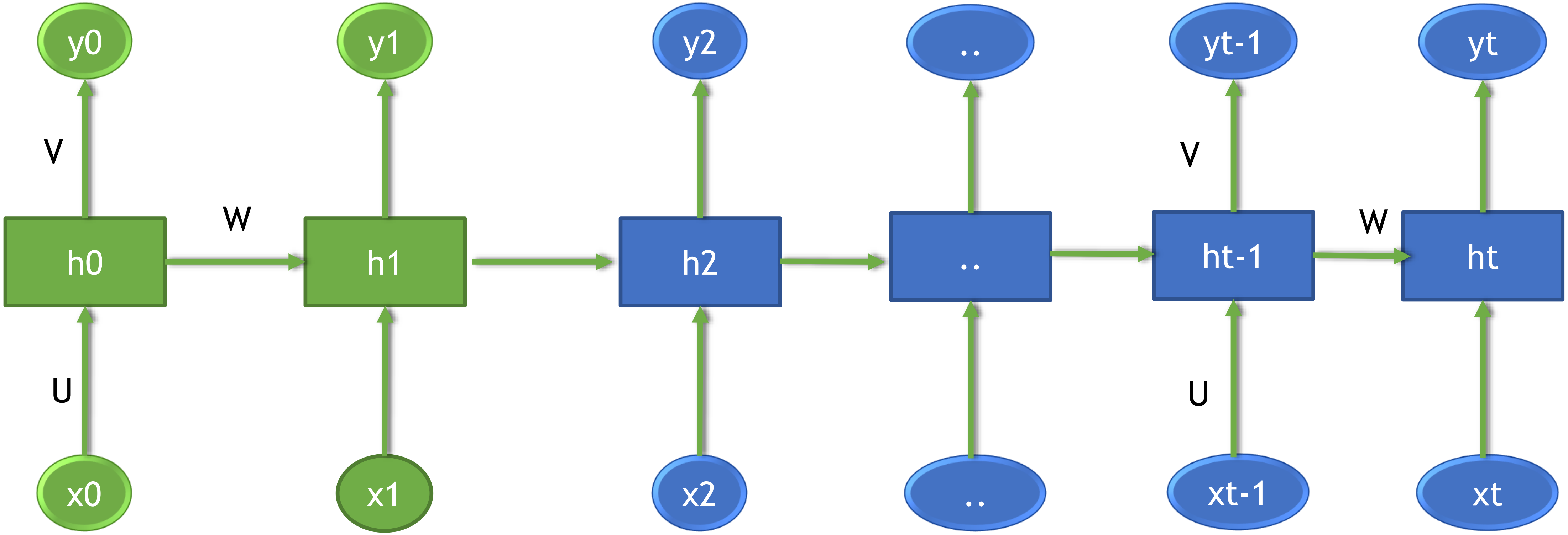
Back Propagation Through Time

Input training data and perform feed forward calculations



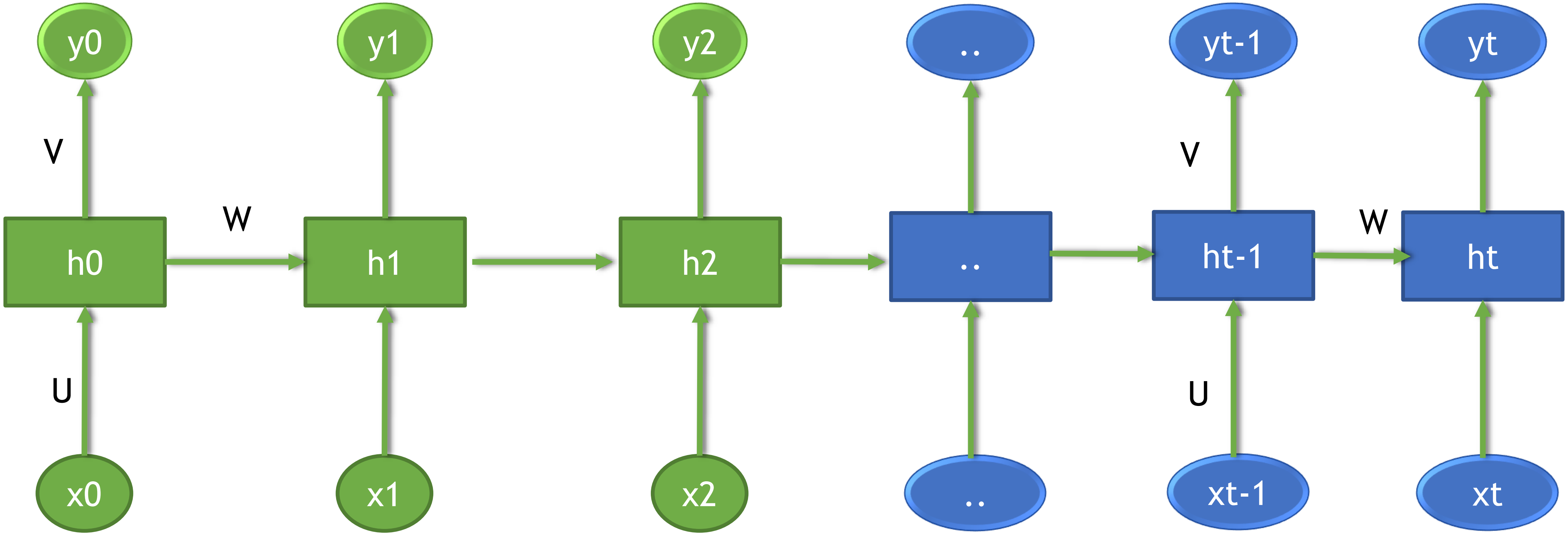
Back Propagation Through Time

Input training data and perform feed forward calculations



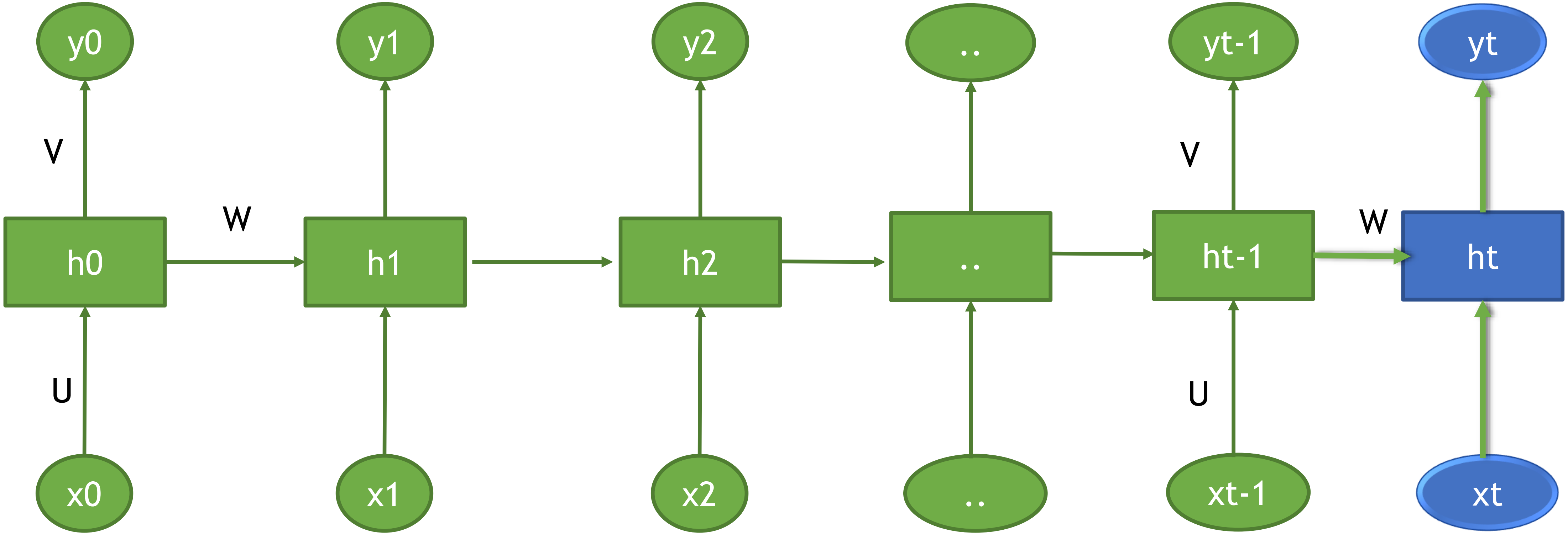
Back Propagation Through Time

Input training data and perform feed forward calculations



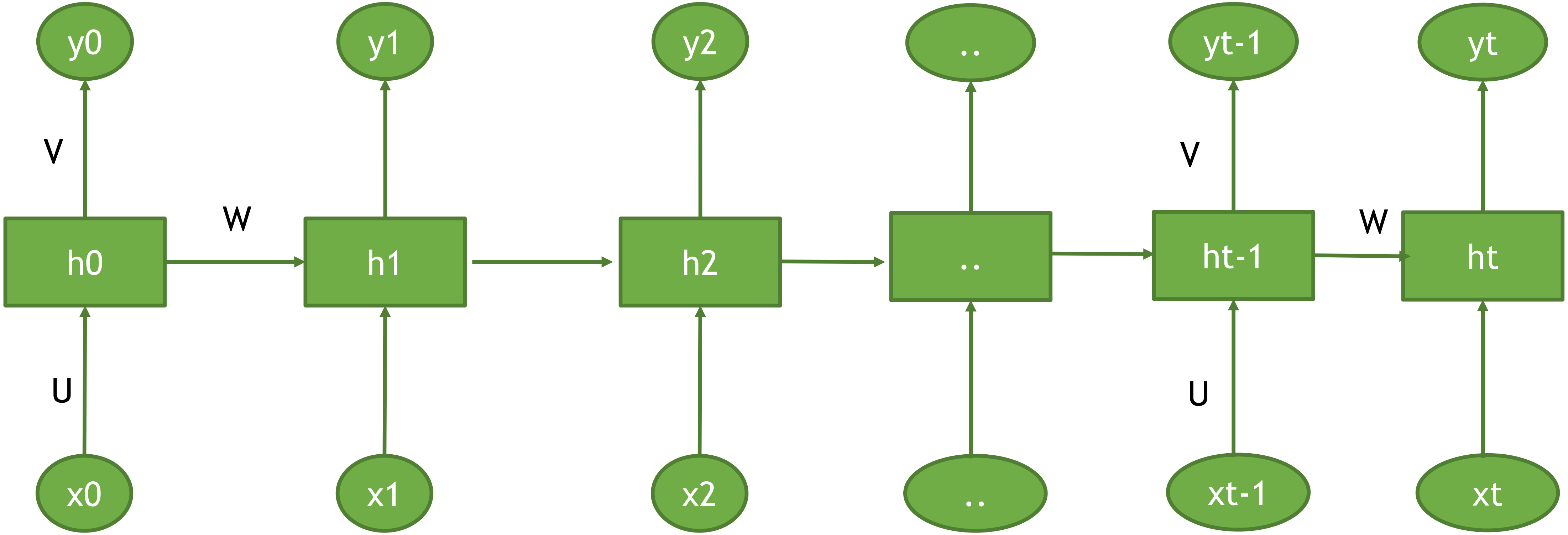
Back Propagation Through Time

Input training data and perform feed forward calculations



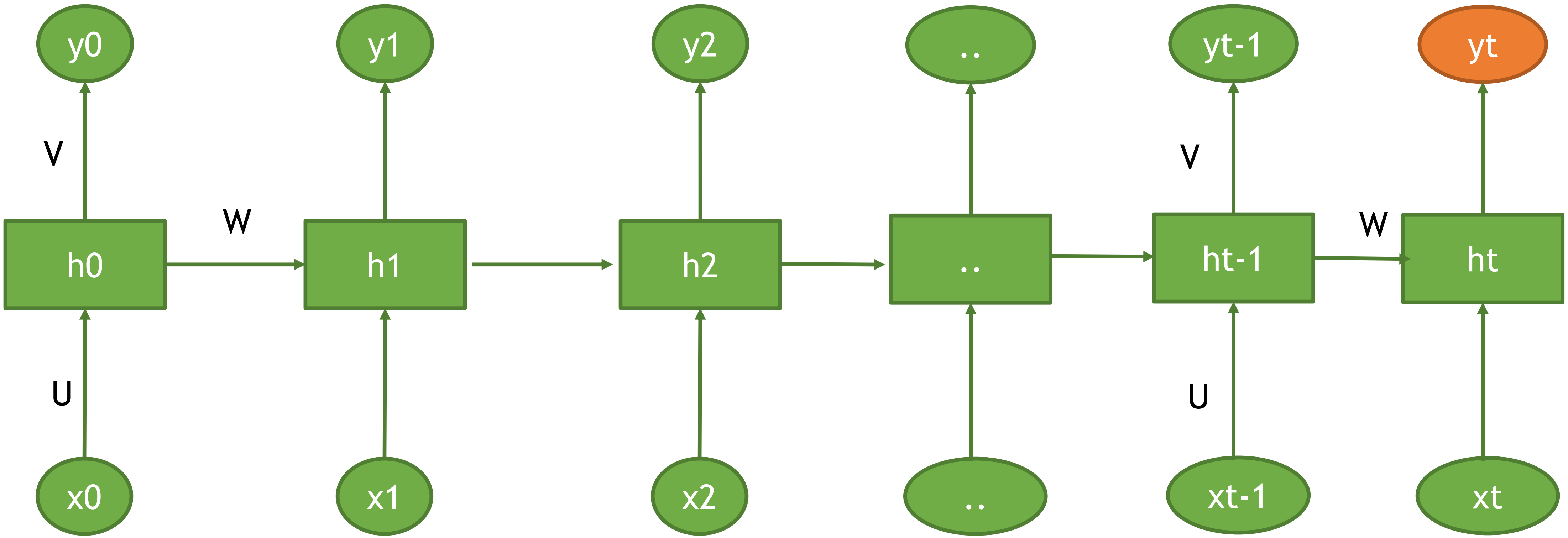
Back Propagation Through Time

Input training data and perform feed forward calculations



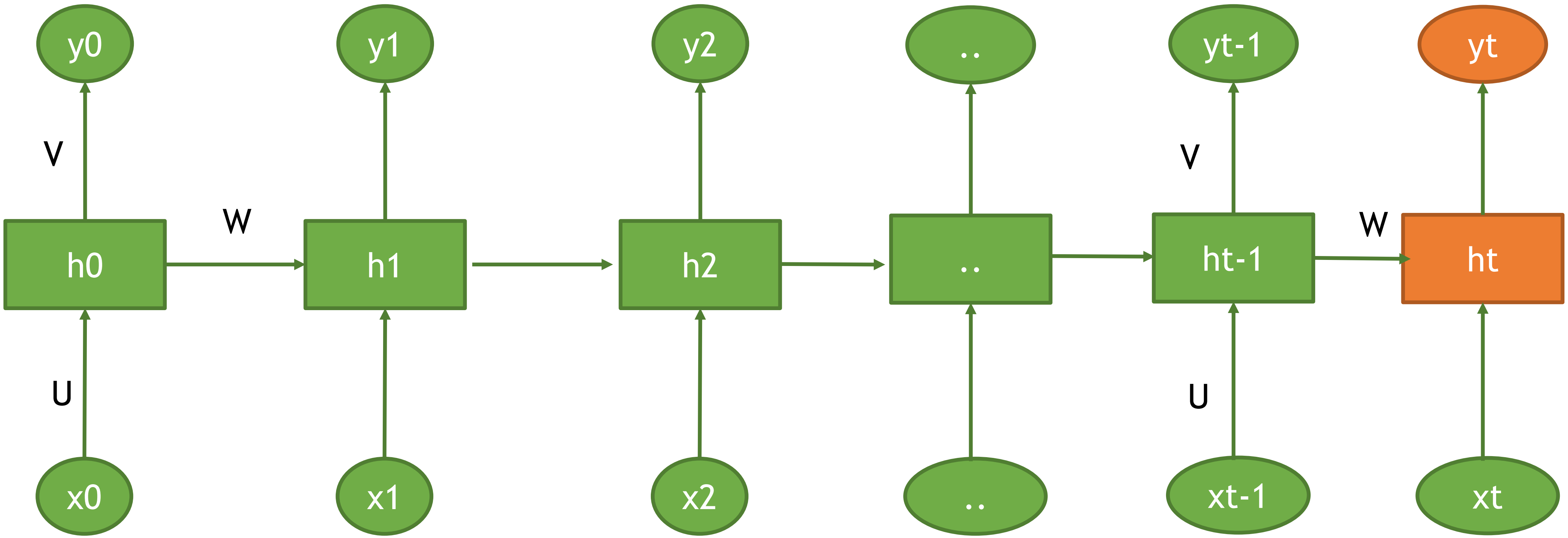
Back Propagation Through Time

Calculate error at output layer and propagate it backwords.



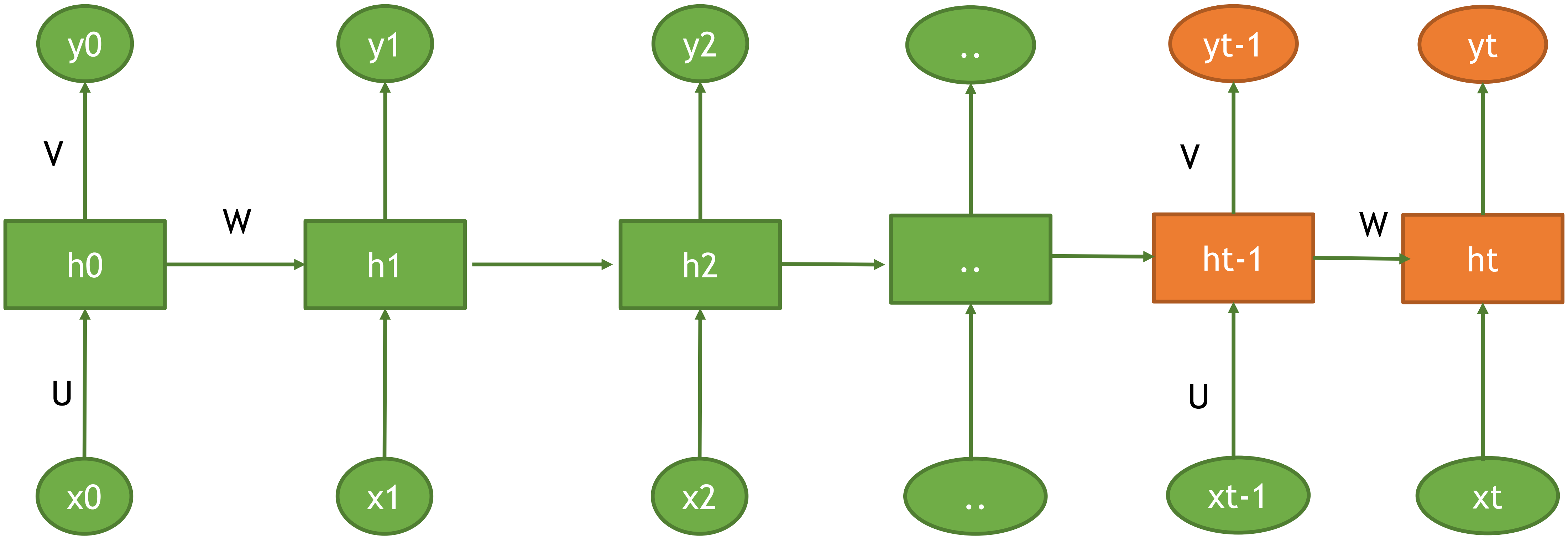
Back Propagation Through Time

Calculate error at output layer and propagate it backwords.



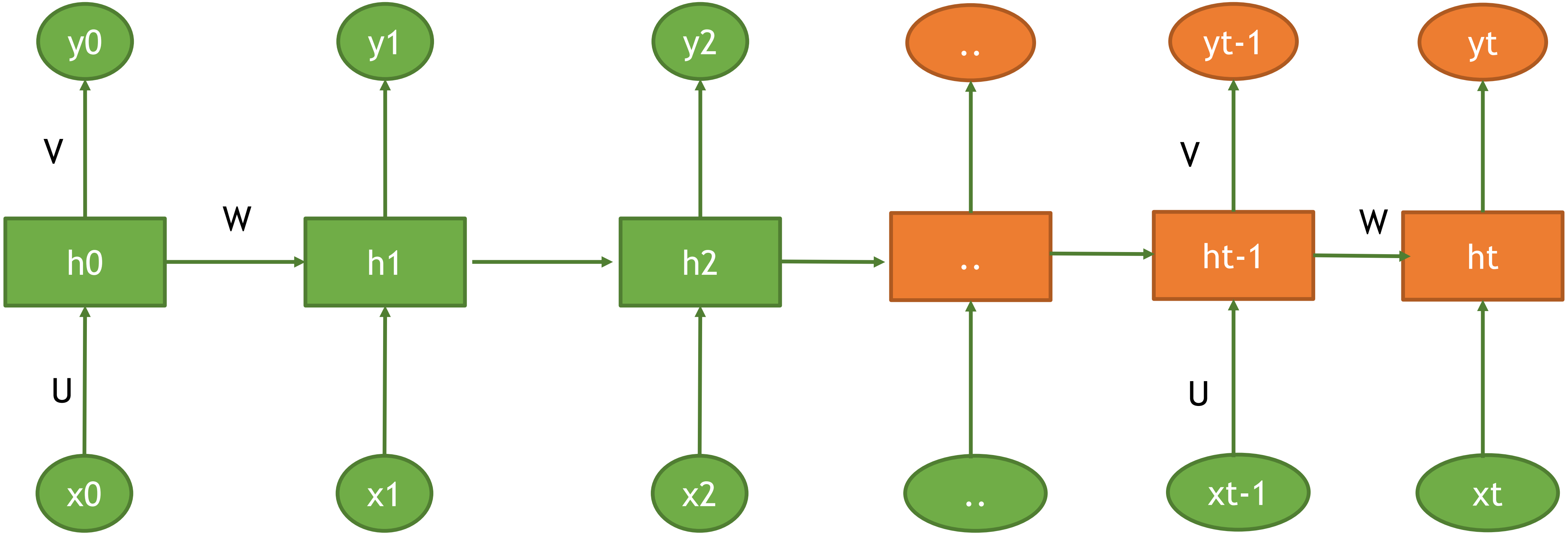
Back Propagation Through Time

Calculate error at output layer and propagate it backwords.



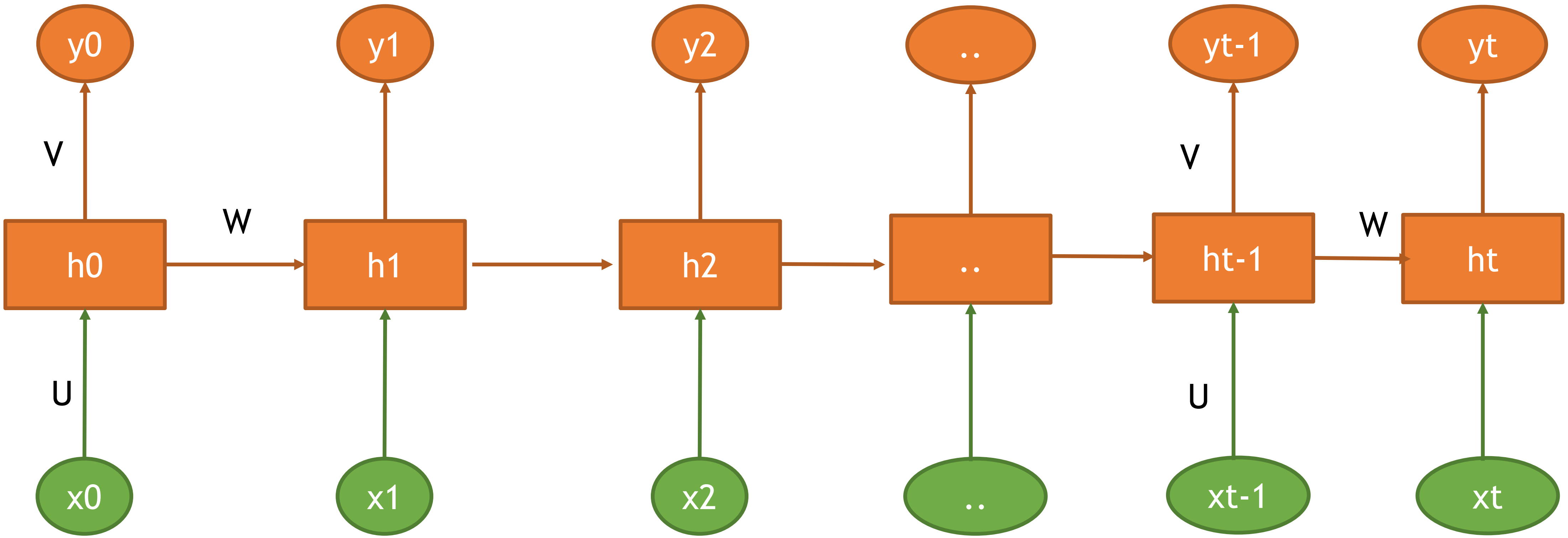
Back Propagation Through Time

Calculate error at output layer and propagate it backwords.



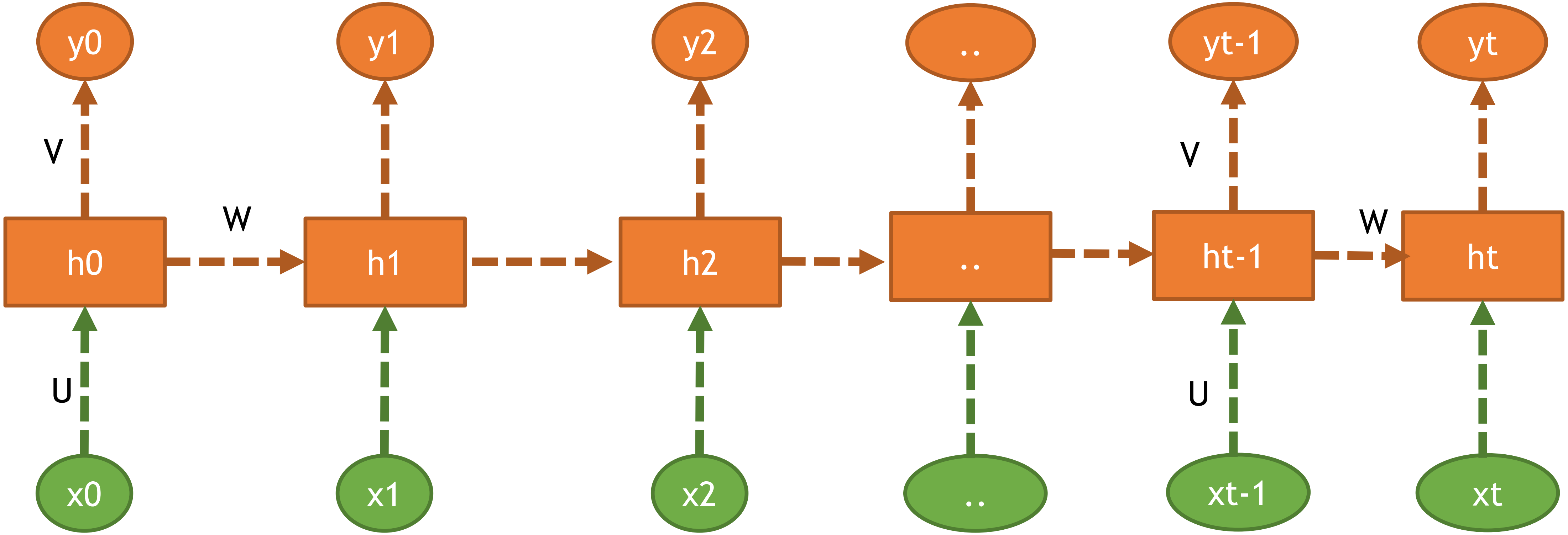
Back Propagation Through Time

Calculate error at output layer and propagate it backwords.



Back Propagation Through Time

Finally, weight corrections to reduce the error



Building RNN Models in Keras

- We have to mention time stamps
- Number of hidden nodes at each time stamp

LAB: RNN for word prediction

- Take Love gram data as input. Load the data. Build RNN model
- Generate text starting with below words
  - Love to
  - Love the
  - Love it

Code: RNN for word prediction

- Preparing the data
  - X3= [word1, word2]; y3= word3
  - Mapping and encoding X3 and y3

```
X3 = gram2[['word1','word2']]
for i in list(X3.columns.values):
    X3[i] = X3[i].map(char_indices)

X3=np.array(X3)
X3=np.reshape(X3,(X3.shape[0],2,1))
X3 = keras.utils.to_categorical(np.array(X3), num_classes=len(char_indices))
print(X3.shape)

y3 = gram2['word3'].map(char_indices)
y3 = keras.utils.to_categorical(np.array(y3), num_classes=len(char_indices))
print(y3.shape)
```

Creating array of X3

Creating array of y3

Code: RNN for word prediction

- This is how an observation of encoded X3 looks like:
  - 0^{th} observation

```
x3[0]
```

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

Word1

Word2

Code: RNN for word prediction

- Defining our SimpleRNN model
  - SimpleRNN('number of hidden nodes in each rnn cell', input_shape=(timesteps, input_data_dim))

```
model3 = Sequential()
model3.add(SimpleRNN(30, input_shape=(X3.shape[1],X3.shape[2])))
model3.add(Dense(len(char_indices)))
model3.add(Activation('softmax'))
model3.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 30)                5100
_________________________________________________________________
dense_6 (Dense)              (None, 139)               4309
_________________________________________________________________
activation_2 (Activation)    (None, 139)               0
=================================================================
Total params: 9,409
Trainable params: 9,409
Non-trainable params: 0
```

Number of hidden nodes in each RNN cell = 30

Time steps/length of sequence = X3.shape[1] = 2

Dimension of each variable: X3.shape[2] = 139

Output layer dim: size of y= 139(same as number of words)

Code: RNN for word prediction

- Enabeling checkpoints, compiling and training the model
  - filepath: where weights will be saved(make sure the path exists)
  - Monitor: evaluation matrix
  - Mode: min, max, auto; to decide whats best for the evaluation matrix. Eg:
    - Accuracy: we need 'max'
    - Error: we need 'min'
  - Save_best_only: to save weights only for the epoch with best monitor value

```
from keras.callbacks import ModelCheckpoint
import h5py

filepath="Datasets\\Other\\Weights_trained.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             mode='auto', save_best_only=True, save_weights_only=True)
callbacks_list = [checkpoint]
```

H5py helps us save weights of model in hdf5 format

Passing checkpoints to callbacks_list

* Please install h5py using !conda install h5py or !pip install h5py

Code: RNN for word prediction

- Enabling checkpoints, compiling and training the model

```
# compile network
model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
model3.fit(X3, y3, epochs=20, verbose=1, validation_data=(X3, y3),callbacks=callbacks_list)
```

Callbacks to make model save the epochs according to our configuration

```
666

Epoch 00017: val_acc improved from 0.66511 to 0.66660, saving model to Datasets\Other\Weights_trained.hdf5
Epoch 18/20
5351/5351 [==============================] - 1s 102us/step - loss: 1.2111 - acc: 0.6636 - val_loss: 1.1953 - val_acc: 0.6
666

Epoch 00018: val_acc did not improve from 0.66660
Epoch 19/20
5351/5351 [==============================] - 1s 108us/step - loss: 1.2027 - acc: 0.6625 - val_loss: 1.1878 - val_acc: 0.6
666

Epoch 00019: val_acc did not improve from 0.66660
Epoch 20/20
5351/5351 [==============================] - 1s 120us/step - loss: 1.1948 - acc: 0.6629 - val_loss: 1.1816 - val_acc: 0.6
666

Epoch 00020: val_acc did not improve from 0.66660

<keras.callbacks.History at 0xe3616a0>
```

Code: RNN for word prediction

- Loading saved model weights and running for a few more epochs

```
weightsfile= "datasets\\other\\Weights_trained.hdf5"
model3.load_weights(weightsfile)
```

Load saved weights to model

```
# compile network
model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
model3.fit(X3, y3, epochs=10, verbose=1, validation_data=(X3, y3))
```

Train the model for 10 more epochs

```
Epoch 5/10
5351/5351 [==============================] - 1s 102us/step - loss: 1.1811 - acc: 0.6625 -
666
Epoch 6/10
5351/5351 [==============================] - 1s 99us/step - loss: 1.1757 - acc: 0.6627 -
66
Epoch 7/10
5351/5351 [==============================] - 1s 105us/step - loss: 1.1717 - acc: 0.6614 -
666
Epoch 8/10
5351/5351 [==============================] - 1s 117us/step - loss: 1.1687 - acc: 0.6608 -
666
Epoch 9/10
5351/5351 [==============================] - 1s 110us/step - loss: 1.1653 - acc: 0.6612 -
681
Epoch 10/10
5351/5351 [==============================] - 1s 99us/step - loss: 1.1632 - acc: 0.6627 -
81

<keras.callbacks.History at 0x121b5eb8>
```

Code: RNN for word prediction

- Writing a custom prediction function and making predictions

```
def rnn_word_pred(in_text):
    print("Input is - " , in_text)
    encoded = [char_indices[i] for i in in_text]
    encoded = np.array(encoded).reshape(1,2,1)
    encoded =keras.utils.to_categorical(np.array(encoded), num_classes=len(char_indices))
    yhat = model3.predict_classes(encoded, verbose=0)[0]
    print("Output is --> " ,indices_char[yhat])
```

Map test text into char_to_indices, then Onehot-encode

Make prediction by passing through model

```
rnn_word_pred(["love","the"])
rnn_word_pred(["love","it"])
rnn_word_pred(["love","to"])
```

Making some predictions

```
Input is -  ['love', 'the']
Output is -->  way
Input is -  ['love', 'it']
Output is -->  when
Input is -  ['love', 'to']
Output is -->  see
```

RNN - Issues

Her heart was heavy because it was open, and so things filled it, and so things rushed out of it, but still the heart kept beating, tough and frighteningly powerful and meaning to shrug off the rest of her and continue on its own

My heart was heavy because it was open, and so things filled it, and so things rushed out of it, but still the heart kept beating, tough and frighteningly powerful and meaning to shrug off the rest of me and continue on its own

His heart was heavy because it was open, and so things filled it, and so things rushed out of it, but still the heart kept beating, tough and frighteningly powerful and meaning to shrug off the rest of him and continue on its own

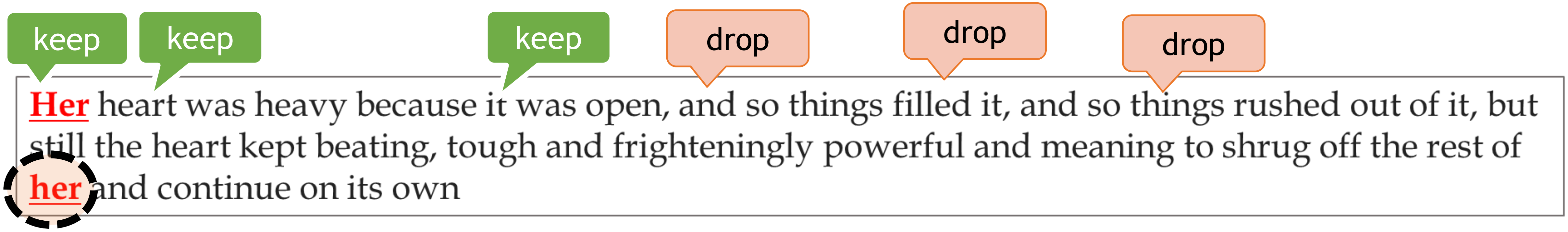
- “her” in the last line depends on “Her” in the beginning.
- If the sentence starts with “My” then it will end up with “me”
- Standard RNNs **fail** to train such long sequences

Standard RNNs can't learn long sequences

- In real world applications, RNNs trained with BPTT have difficulties in learning long-term dependencies.
- RNN in theory - Should learn very long sequences
- RNN in practise - limited to looking back at only a few steps.

LSTM – main idea

- In standard RNNs every hidden unit and input from time stamp is given importance. This may not be necessary in every scenario
  - We may want to ignore (forget) few intermediate inputs
  - We may want to keep some specific information for long interval
- In the below example, to predict last few words, we may need only few key words from beginning



LSTM - Calculations

$$C_t = C_{t-1} \circ f + g \circ i$$

C_t memory cell state at time t

C_{t-1} Cell state at time t-1

f forget irrelevant info

g Information at time t

i Allow relevant information as input

$$h_t = \tanh(C_t) \circ O$$

h_t RNN hidden state at time t

$\tanh(C_t)$ memory cell info at time t

O output only relevant information

LSTM - Calculations

x_t input at time t

Input gate

$$i = \sigma(x_t U^i + h_{t-1} W^i)$$

h_{t-1} hidden state at time t-1

Forget gate

$$f = \sigma(x_t U^f + h_{t-1} W^f)$$

Output gate

$$o = \sigma(x_t U^o + h_{t-1} W^o)$$

actual input at time t

$$g = tanh(x_t U^g + h_{t-1} W^g)$$

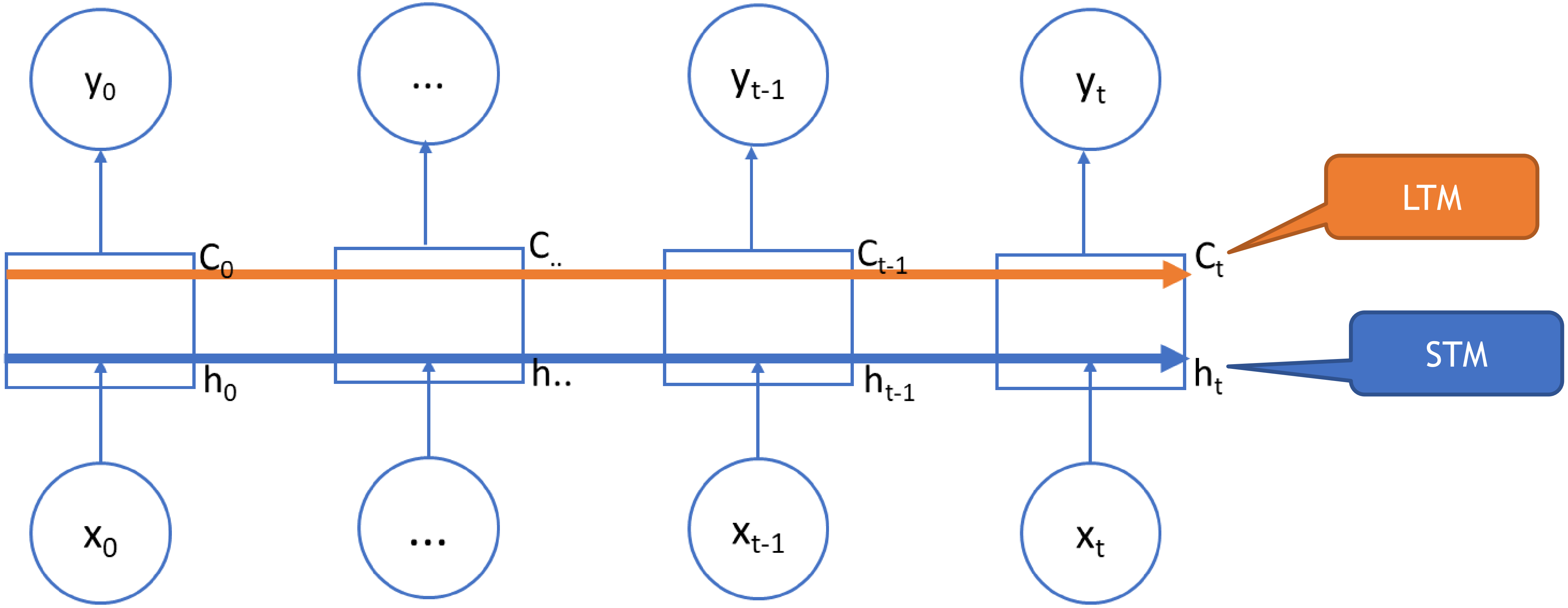
C_t memory cell state at time t

$$C_t = C_{t-1} \circ f + g \circ i$$

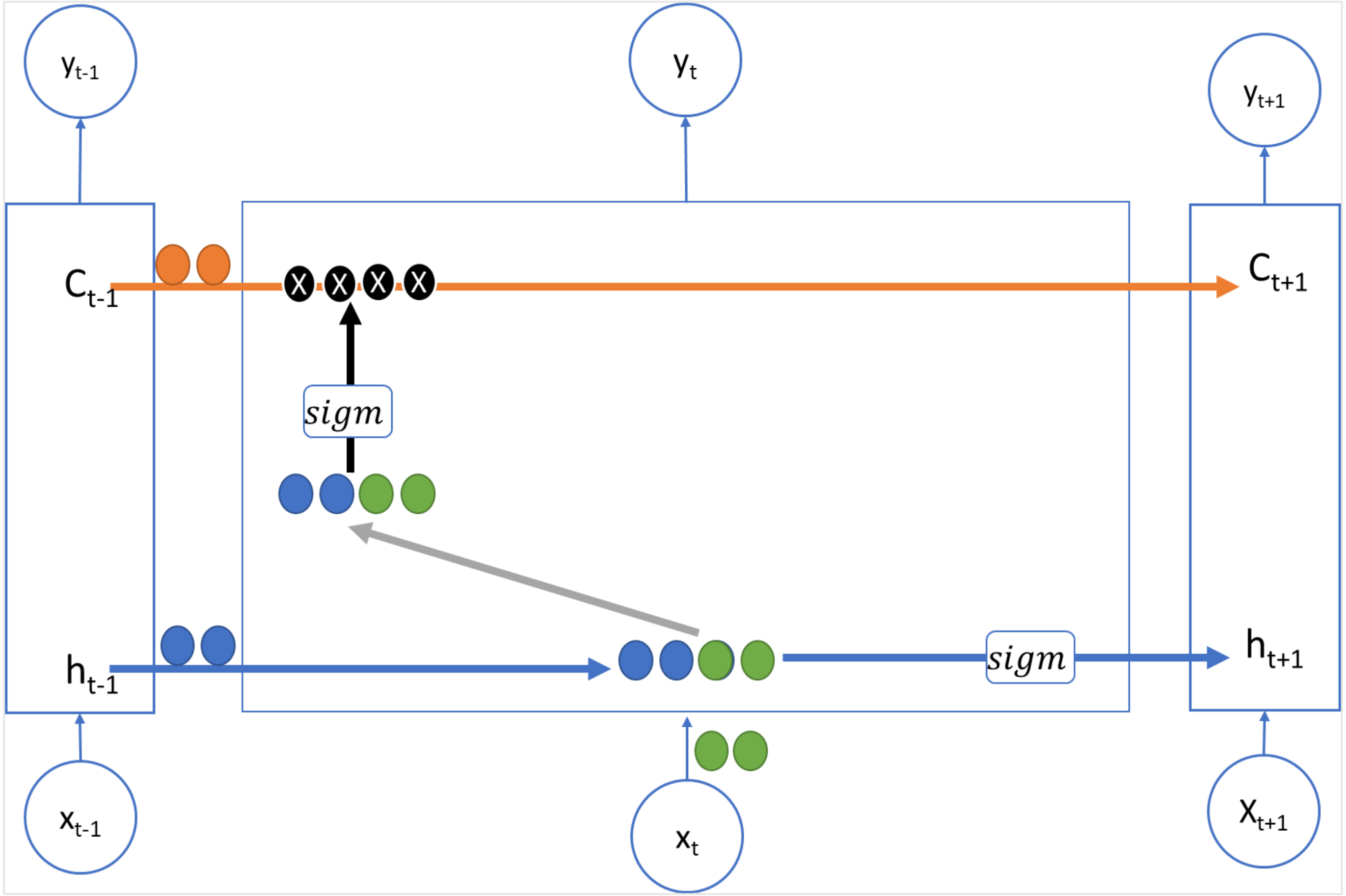
h_t RNN hidden state at time t

$$h_t = \tanh(C_t) \circ O$$

LSTM – main idea



Forget Gate –
To erase or retain information from cell state



$$f = \sigma(x_t U^f + h_{t-1} W^f)$$

Weights associated with forget gate

$$C_t = C_{t-1} \circ f$$

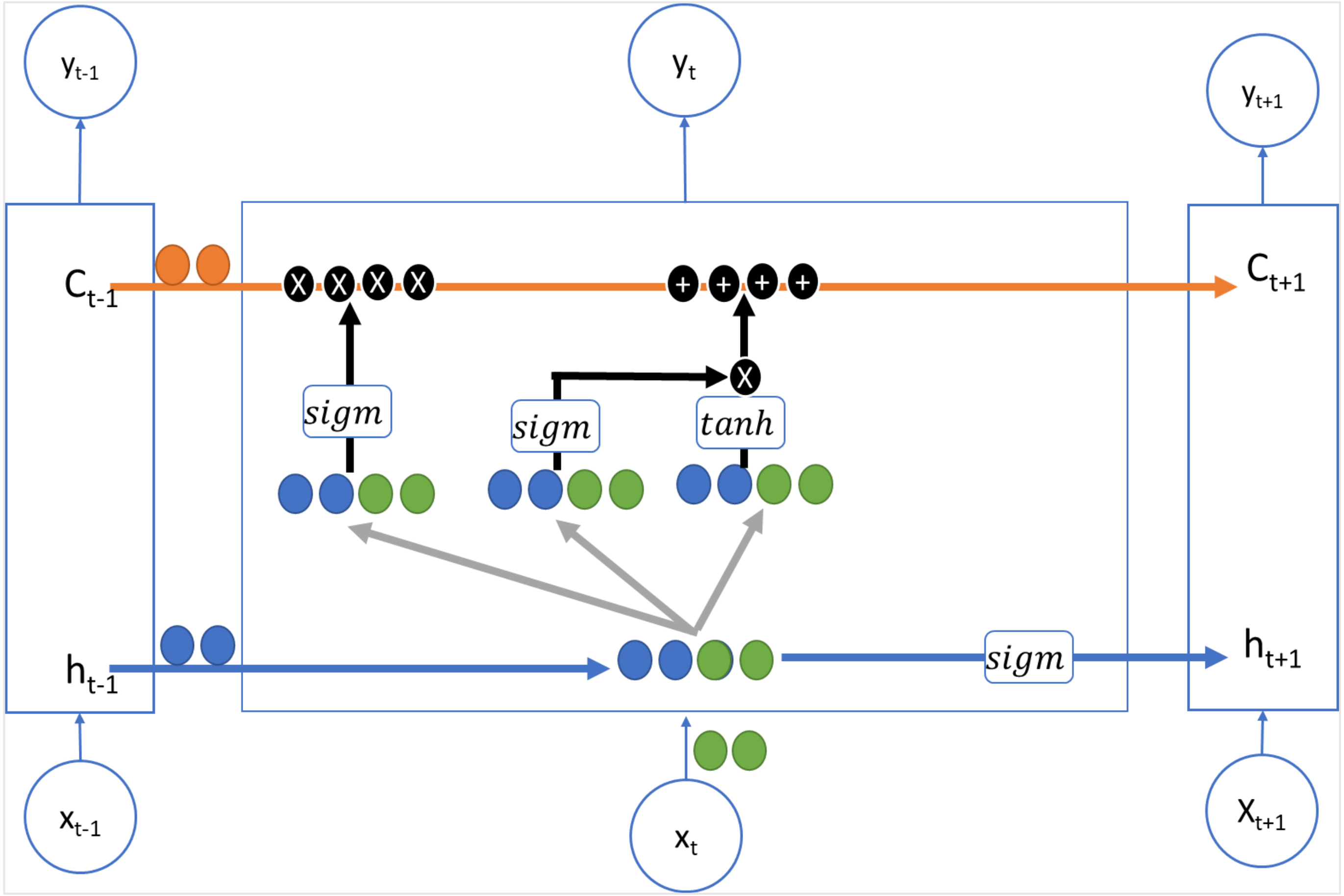
Updating the cell state

$$h_t = \sigma(x_t U + h_{t-1} W)$$

Regular Weights U and W

Input gate

To input information into cell state



$$f = \sigma(x_t U^f + h_{t-1} W^f)$$

Weights associated with forget gate

$$C_t = C_{t-1} \circ f$$

Updating the cell state

$$i = \sigma(x_t U^i + h_{t-1} W^i)$$

Weights associated with the input gate

$$g = tanh(x_t U^g + h_{t-1} W^g)$$

Weights associated with current input updater

$$C_t = C_{t-1} \circ f + g \circ i$$

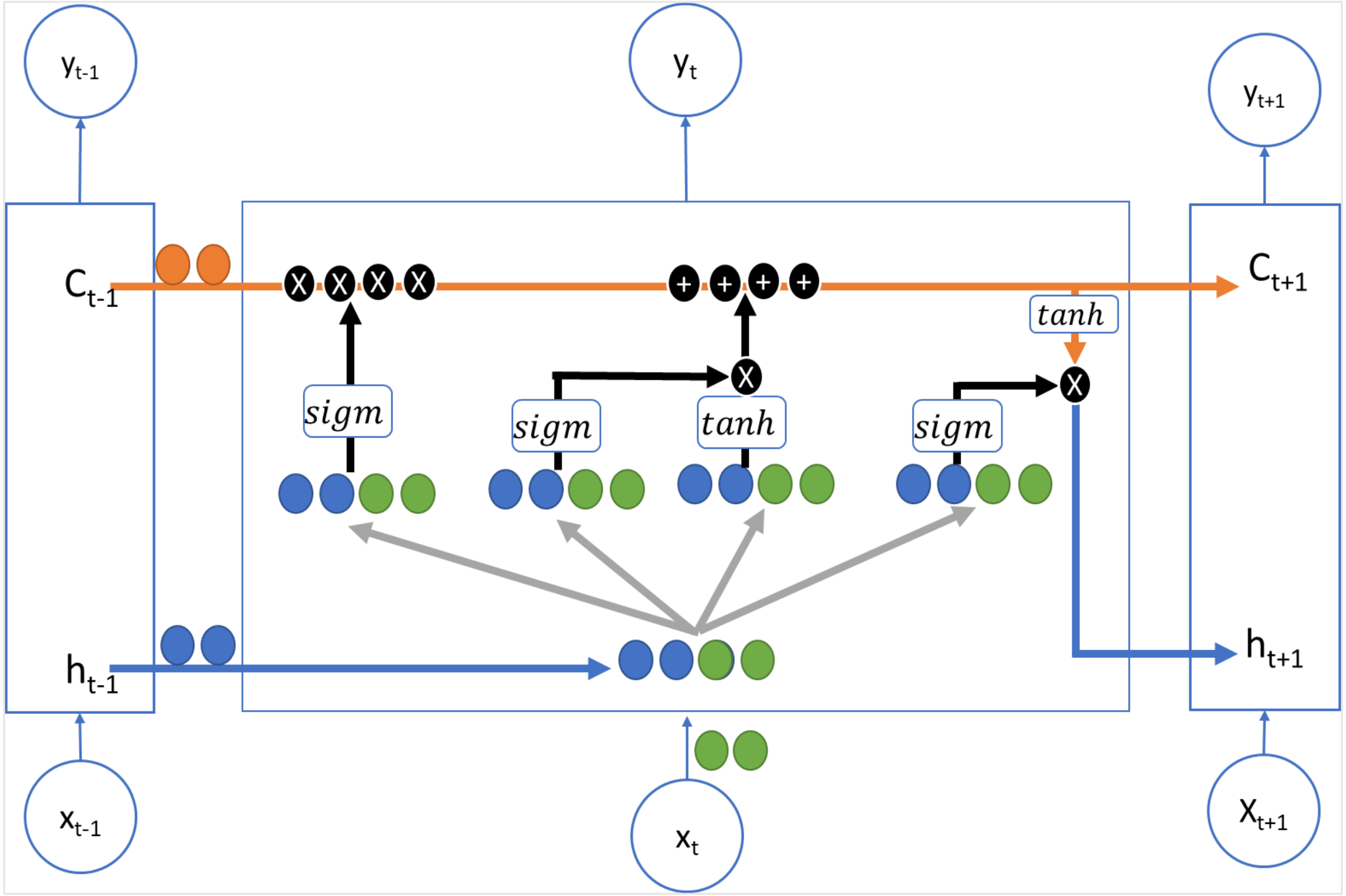
Write current input into cell state

$$h_t = \sigma(x_t U + h_{t-1} W)$$

Regular Weights U and W

Output gate

To output relevant information from cell state



$$f = \sigma(x_t U^f + h_{t-1} W^f)$$

Weights associated with forget gate

$$C_t = C_{t-1} \circ f$$

Updating the cell state

$$i = \sigma(x_t U^i + h_{t-1} W^i)$$

Weights associated with the input gate

$$g = tanh(x_t U^g + h_{t-1} W^g)$$

Weights associated with current information updater

$$C_t = C_{t-1} \circ f + g \circ i$$

Write current input into cell state

$$O = \sigma(x_t U^o + h_{t-1} W^o)$$

Weights associated with output gate

$$h_t = tanh(C_t) \circ O$$

The final output from the memory cell.

How gates work ? – Example text data

- Predict next word
  - Jim is a software engineer. He works for an IT company. Lynda is a teacher. --------
- Historical data - Training data examples
  - "Lynda was late that day. She apologized"
  - "Lynda's alarm goes off at 5 am. She gets up early."
  - "Jim told Lynda – 'you have such beautiful eyes.' Lynda smiled at him. She continued to walk."
  - "Jim is a software engineer. He works for an IT company. Lynda is a teacher. She teaches in a school."
  - "Lynda likes exploring new cities. She traveled to Paris last month."
  - "Lynda got a promotion last month. She got a good pay hike."

How gates work ? – Example text data

- Predict next word
  - Jim is a software engineer. He works for an IT company. Lynda is a teacher. --------
- Forget gate
  - Based on historical data, deletes the current subject from cell state
- Input gate
  - Based on historical data, input gate inputs the subject(Lynda) into cell state.
- Output gate
  - Based on the historical data, the output gate will decide to output to be “she” or “her”
- Cell state
  - Lynda- the subject will be carried to next cell state.

LAB: LSTM Model building

- Data Set: 3Gram_12Chars.csv
- Prepare data for the model.
- Use each sentence to make 14 characters as input and next character as output.
- Build an LSTM model try predicting next few letters

Conclusion

- We discussed about sequential models and details of RNN algorithm
- Though Back Propagation Through Time is a good algorithm, most of the times it fails due to vanishing and exploding gradients
- Standard RNNs can be used for short term dependencies.
- We may need to use LSTMs for long sequences.

Appendix