Intro to NNs for Sequential Data

Recurrent Neural Networks (RNNs)

Sources: A. Ng, F. Lee, J. Johnson, Mach. Learn. Mastery (book)

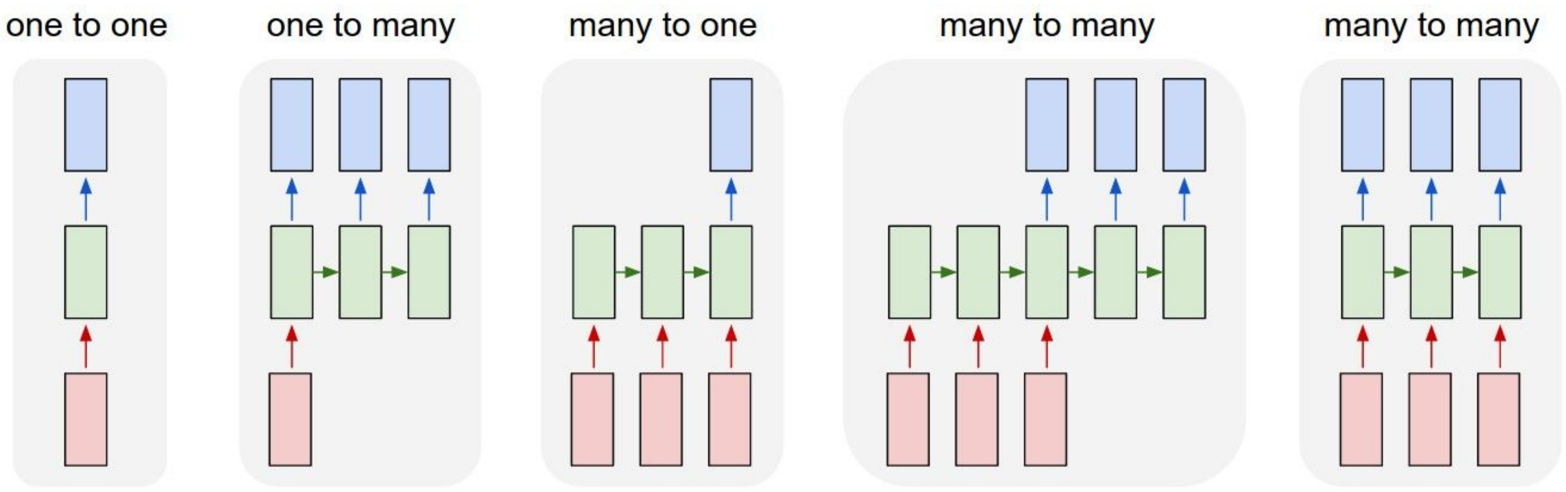
Intro to NNs for Sequential Data

- Thus far, we have considered supervised learning models with iid data.

    - I.I.D: Independent and identically distributed observations
        - Key assumption - observations are not correlated with one another
    - Problem: Lots of data exists that doesn't meet this assumption!
        - Examples of data with sequential structure:
            - Time series (our main focus)
            - Text (with one character related to sequence before and after)
            - Audio data (with time sequence)
            - Video data (with time sequence), etc.

Intro to NNs for sequential data

- NNs for sequential data (e.g.- Recurrent NNs) are flexible with regard to inputs and outputs!
- We will focus on typical supervised learning examples with sequential data, but these models can be bent to your will!



NNs for sequential data

Objectives for this lecture:

- Understand basics of sequential modelling using Recurrent Neural Network cells
- Start to think about time series oriented sequential models
- Next week we will discuss LSTMs and GRUs, which improve upon RNNs.

NNs for Sequential Data Examples

Green: Code examples we will cover

Univariate or Multivariate Time-Series for Tabular data: Input: Tabular Features-> Output: Target prediction

Speech recognition: *Input*: audio clip -> *Output*: text

Music generation: *Input*: integer referring to genre (or an empty set) -> *Output*: music

Sentiment classification: *Input*: text -> *Output*: ratings

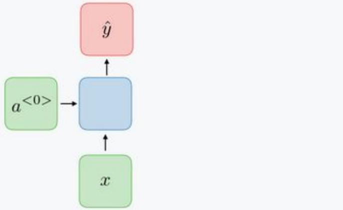
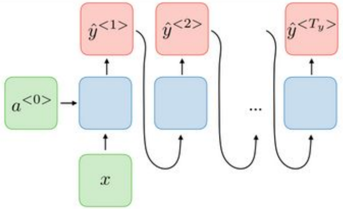
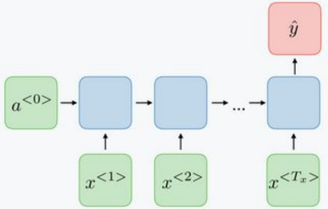
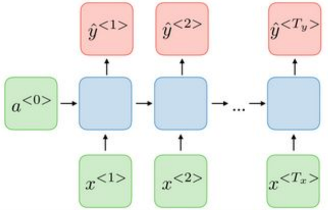
DNA sequence analysis: *Input*: DNA (alphabet) -> *Output*: label part of the DNA sequence

Machine translation: *Input*: text -> *Output*: text translation

Video activity recognition: *Input*: video frames -> *Output*: identification of the activity

Name entity recognition: *Input*: sentence -> *Output*: identify people within it.

Model Architecture Examples

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition

This is what we've been using.

Most of our in-class focus will be here for sequential models (all green highlighted examples fit this structure in previous slide)

Sequential data is different

Let's think about typical non-sequential tabular data:

	Feature 1	Feature 2...	Feature N	Target
Observation 1	*Obs. 1 data*	*Obs. 1 data*	*Obs. 1 data*	*Obs. 1 data*
Observation 2	*Obs. 2 data*	*Obs. 2 data*	*Obs. 2 data*	*Obs. 2 data*
Observation N	*Obs. 3 data*	*Obs. 3 data*	*Obs. 3 data*	*Obs. 3 data*

Observations are assumed to be independent of one another.

E.g.- Countries = Observations

No timesteps!

Sequential data is different

Sequential data adds the idea of multiple time steps (a sequence):

- **Data can have repeated observations that correlate with one another through time**

- **For our purposes a single observation's dimensions will now include time steps in a sequence:**
  - **Notation for observation i could be:**
    - **Xi<t>**
    - **Where i equals observation element number and t equals time step sequence number**

Sequential data is different

Sequential data adds the idea of multiple time steps (a sequence):

- **For our purposes a single observation’s dimensions will now include time steps in a sequence:**
  - **Notation for observation i could be:**
    - **Xi<t>**
    - **Where i equals observation element number and t equals time step sequence number**

New approach allows us to:

Predict y at different timesteps using data from x at different time steps

RNNs as for loops

-Via Francois Chollet

RNNs maintain a kind of memory (or "state") from one time step to the next.
1.Set "state" to zero.
2. Update "state" with X input data, params, & activation from time sequence 1.
3. Use new state value to transform X input in sequence step 2.
4. Loop through each sequence step to build network w/ memory of previous states.

Listing 6.19 Pseudocode RNN

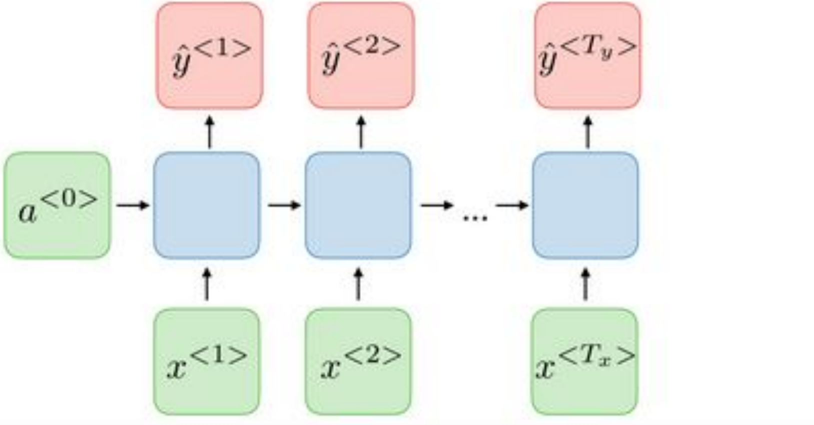
```
state_t = 0                           <-- The state at t
for input_t in input_sequence:        <-- Iterates over sequence elements
    output_t = f(input_t, state_t)
    state_t = output_t                <-- The previous output becomes the state for the next iteration.
```

Building a new type of network…
That addresses sequential problems

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Introduction to Recurrent Neural Networks



Notation Reference:

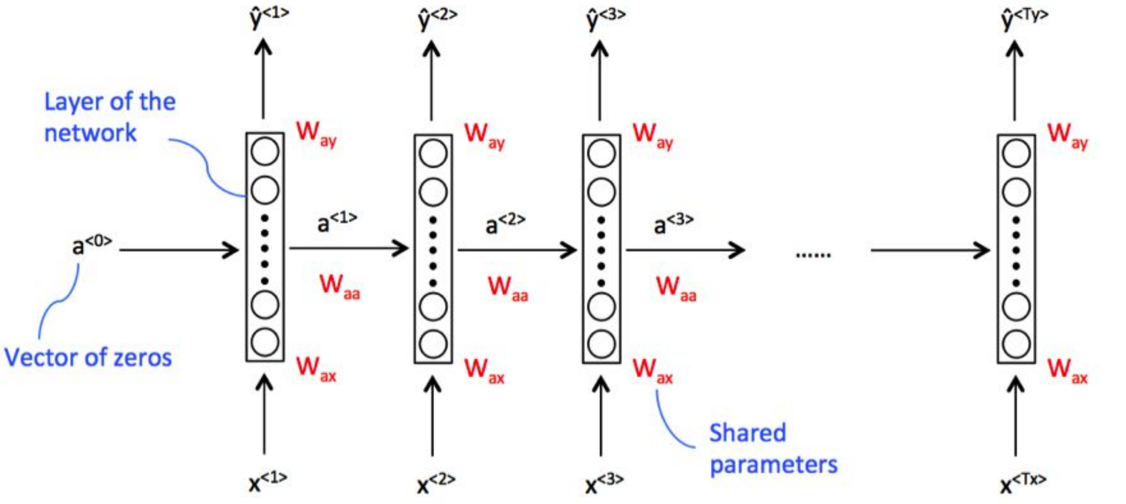
$x^{<t>}$: t-th element in the input sequence
$x^{(i)<t>}$: t-th element in the input sequence of training examples i
T_x : length of the input sequence
T_y : length of the output sequence
$T_x^{(i)}$: input sequence lenght for training example i
$y^{<t>}$: t-th element in the output sequence
$y^{(i)<t>}$: t-th element in the output sequence of training examples i

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Introduction to Recurrent Neural Networks Step by step

Overview:



○= RNN cell hidden layer nodes

a = activation of hidden layer nodes

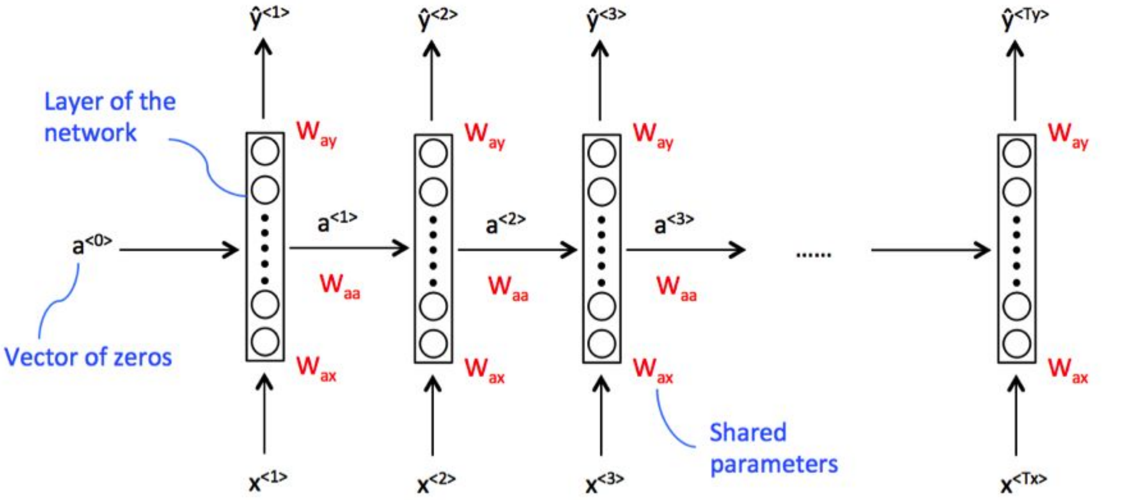
Wax=parameters fit to X input data combined with hidden nodes

Waa=parameters fit to data within hidden nodes

Way=parameters fit to y ouput data with hidden nodes

Introduction to Recurrent Neural Networks
Step by step

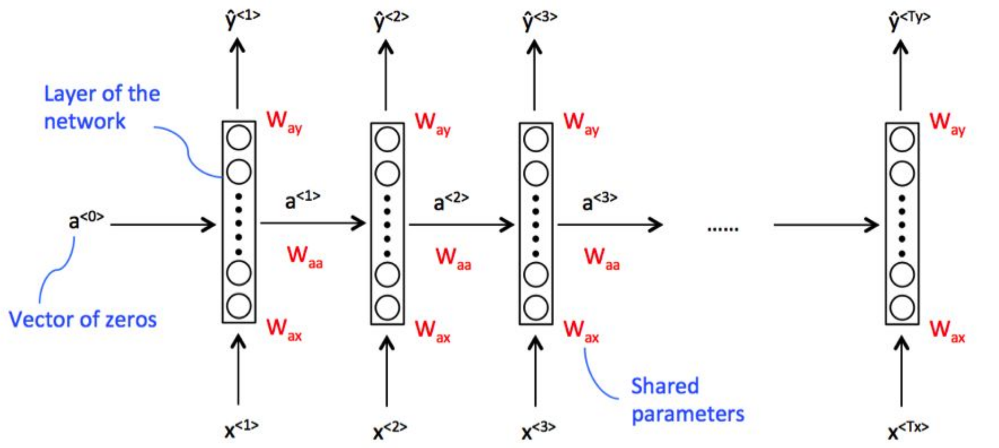
Key idea: Future predictions rely on data from past steps



For example, for prediction ŷ<3> it gets information not only from x<3>, but also from x<1> and x<2>.

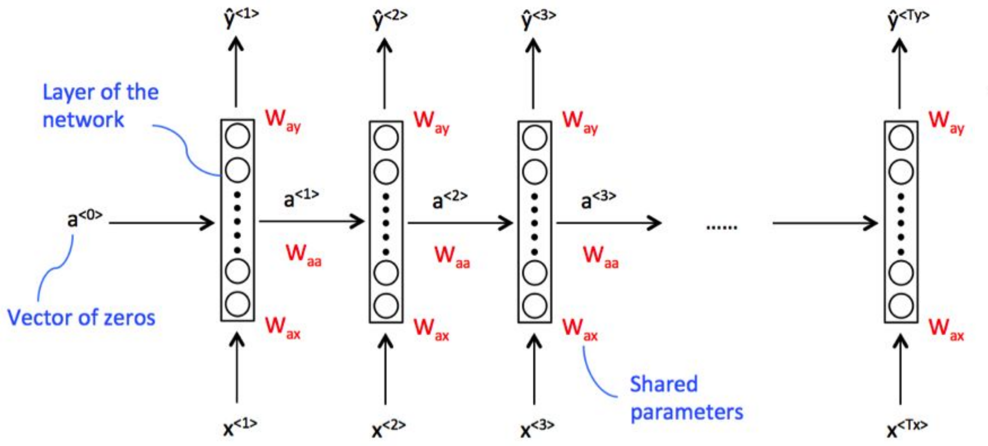
How to calculate forward propagation for RNN

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$
$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

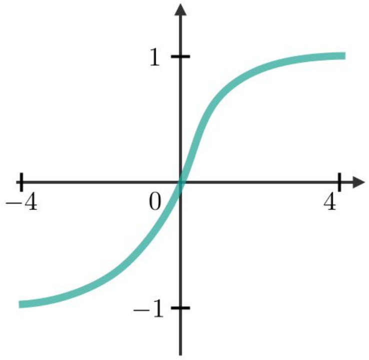


1) *Input a<0> as vector of all zeros*
2) *To compute a<1>*
   - $\boxed{\textit{a<1> = g(Waa*a<0> + Wax*x<1> +ba)}}$ *# g usually uses tanh or relu to transform output*
   -
3) *To comput yhat<1>*
   - $\boxed{\textit{yhat<1> = g(Wya*a<1> + by)}}$ *# g uses softmax or sigmoid depending on # of output categories*

How to calculate forward propagation for RNN

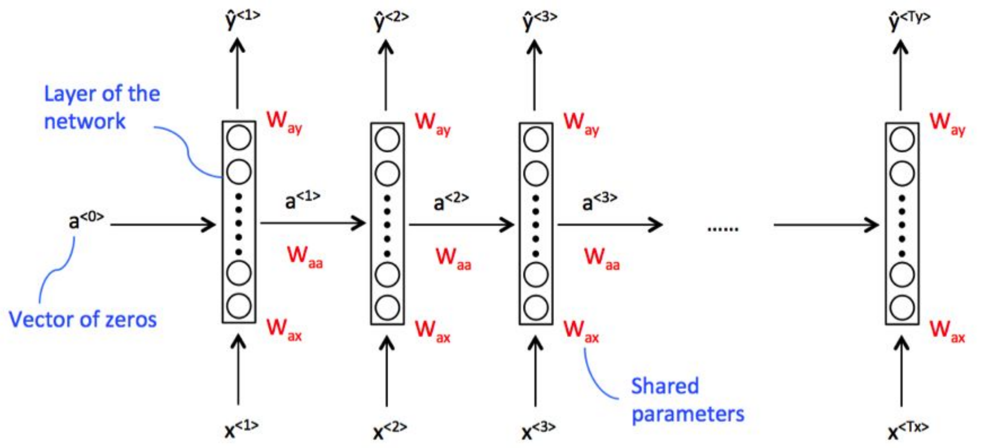


$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



1) *Input a<0> as vector of all zeros*
2) *To compute a<1>*
   - *a<1> = g(Waa*a<0> + Wax*x<1> +ba)* *# g usually uses* ***tanh*** *or relu to transform output*
   -
3) *To comput yhat<1>*
   - *yhat<1> = g(Wya*a<1> + by)* *# g uses softmax or sigmoid depending on # of output categories*

How to calculate forward propagation for RNN



More generally:

$$a^{\langle t\rangle} = \tanh(W_{ax}x^{\langle t\rangle} + W_{aa}a^{\langle t-1\rangle} + b_a)$$

$$\hat{y}^{\langle t\rangle} = softmax(W_{ya}a^{\langle t\rangle} + b_y)$$

Wa, ba, Wy, and by are coefficients shared temporally:

Parameters <u>Wa and ba</u> reused to calculate a<n>

Parameters <u>Wy and by</u> reused to calculate yhat<t>

How is loss calculated in back propagation

Back propagation = back propagation through time or bptt

Weights are updated from final time step backwards one step at a time.

Categorical loss is calculated for each yhat output and then summed to calculate total loss.

Keras Example: Univariate RNN Example
Step by step

Data preparation

Consider a given univariate sequence:

[10, 20, 30, 40, 50, 60, 70, 80, 90]

Keras Example: Univariate RNN Example
Step by step

Data preparation

Consider a given univariate sequence:

[10, 20, 30, 40, 50, 60, 70, 80, 90]

Need to restructure data to take sequence of timesteps (x) and predict future timestep or timestep(s) y

Keras Example: Univariate RNN Example Step by step

For univariate time series problem

[10, 20, 30, 40, 50, 60, 70, 80, 90]

Example of appropriate data structure:

Use three previous timesteps (x) to predict next time step (y)

```
X,                          y
10, 20, 30          40
20, 30, 40          50
30, 40, 50          60
...
```

Use pre-written functions to reshape data

Via tensorflow

The function below returns windows of time for the model to train on. The parameter history_size is the size of the past window of information. The target_size is how far in the future does the model need to learn to predict. The target_size is the label that needs to be predicted.

```
def univariate_data(dataset, start_index, end_index, history_size, target_size):
  data = []
  labels = []

  start_index = start_index + history_size # changing for iteration and data reshaping below
  if end_index is None:
    end_index = len(dataset) - target_size

  for i in range(start_index, end_index):
    indices = range(i-history_size, i)
    # Reshape data from (history_size,) to (history_size, 1)
    data.append(np.reshape(dataset[indices], (history_size, 1)))
    labels.append(dataset[i+target_size]) #returns original dataset values +1 given changes to start_index above
  return np.array(data), np.array(labels)
```

Use pre-written functions to reshape data

Via tensorflow

The function below returns windows of time for the model to train on. The parameter history_size is the size of the past window of information. The target_size is how far in the future does the model need to learn to predict. The target_size is the label that needs to be predicted.

Start index and end index tell function which observations to read in. These are used to create training and validation datasets

```
uni_data = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90])

univariate_past_history = 3
univariate_future_target = 0 # set to zero if you want to predict next time step

x_train_uni, y_train_uni = univariate_data(uni_data, 0, 8,
                                           univariate_past_history,
                                           univariate_future_target)
```

Use pre-written functions to reshape data

Via tensorflow

The function below returns windows of time for the model to train on. The parameter history_size is the size of the past window of information. The target_size is how far in the future does the model need to learn to predict. The target_size is the label that needs to be predicted.

Start index and end index tell function which observations to read in. These are used to create training and validation datasets

Printing out first element of result

```
print(x_train_uni[0],y_train_uni[0])
```

```
(array([[10],
        [20],
        [30]]), 40)
```

Keras Example: Univariate RNN Example Step by step

Fitting a simple RNN model with Keras:

```
# 8 hidden nodes in RNN layer.
# input shape lists time steps and number of X features

simple_rnn_model = tf.keras.models.Sequential([
    tf.keras.layers.SimpleRNN(8, input_shape=(3,1)),
    tf.keras.layers.Dense(1)
])

simple_rnn_model.compile(optimizer='adam', loss='mae')
```

Keras Example: Univariate RNN Example Step by step

Use model.predict() to predict new data with Keras:

```
# data must be same shape as original input shape
```

x_input= array([10],[20],[30])

```
prediction=model.predict(x_input)
```

Validation data for sequential models

Validation (test) data needs to extend sequence

Means we should keep test data in same sequential order

Simply split data such that you are keeping original order to create correctly structured validation data.

(So shuffling when you split training and test data is bad!)