COMP4650/6490 Document Analysis

Deep Neural Networks — Part II

ANU School of Computing

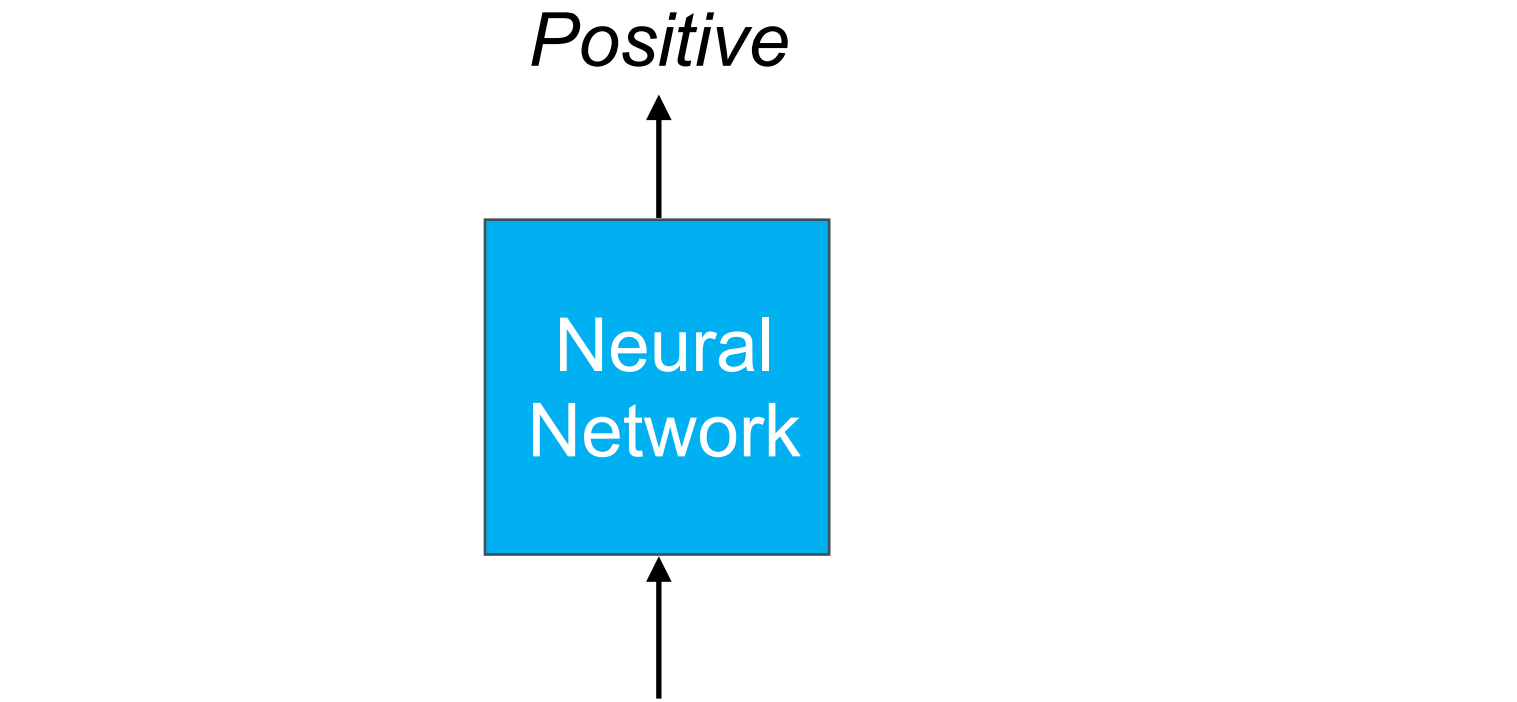
Administrative matters

- Assignment 1
  - Results released on Tuesday 29 August
  - Re-grade requests be posted on Ed forum
- Feedback survey 1
  - Closes on Friday 1 September

Recap

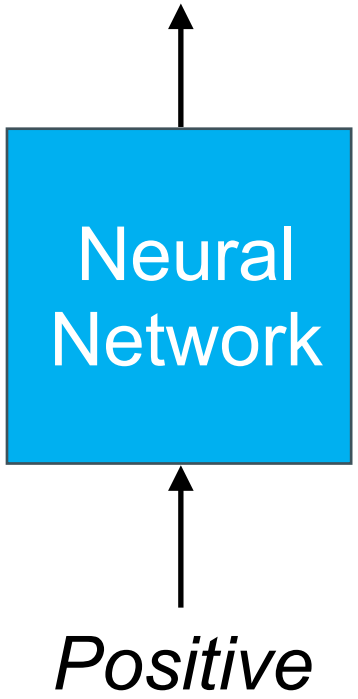
- Neural networks estimate a function which maps vectors to vectors
- Now we want neural networks that can handle structured input and structured output

NN for NLP: Structured input



NN for NLP: Structured output

The movie made me feel terrible but I loved it.



Text is structured

- We couldn't shuffle the words or characters of a sentence and expect to retain the meaning
- The meaning of a word, phrase or sentence is determined by the surrounding context, e.g.
  - The *bank* was grassy
  - The *bank* took my deposit
- Text has a sequential reading order (e.g. left to right in English)

Outline

- Simple neural network language models
  - Sequential output
- Recurrent neural network
  - Better sequential input and output

Neural network language models

Build a neural network that:

- Generates text
- Gives the likelihood of a text sequence

Auto-regressive:

- Take the previous k words and build a classifier to predict the next word.

Bigram language model

Let's choose $k = 1$ for this example:

Sentence: <bos> The best city in Australia.

Step 1: <bos> → The

Step 2: The → best

Step 3: best → city

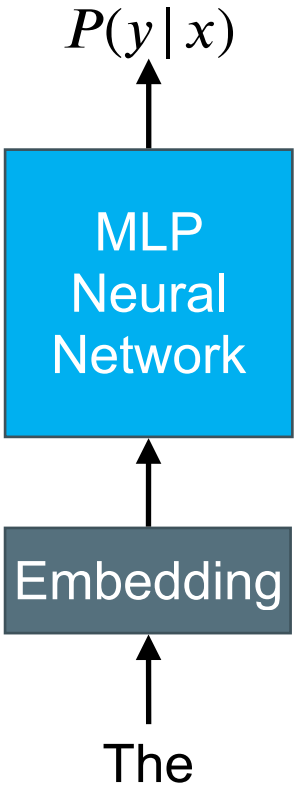
Step 3: city → in

Step 4: in → Australia.

This is the data for our classification model.

A neural network language model

Output: Probability distribution for the next word

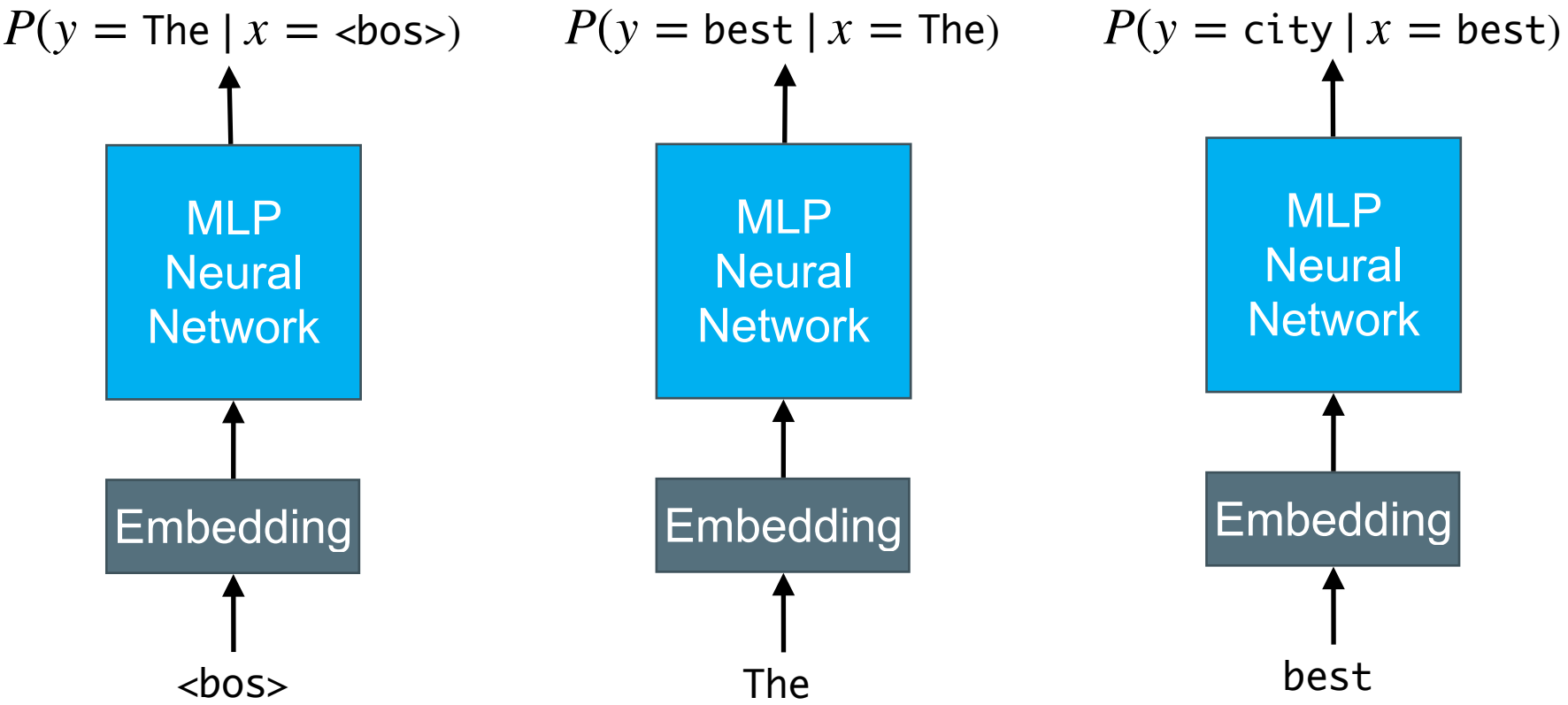


$P(y|x) =$

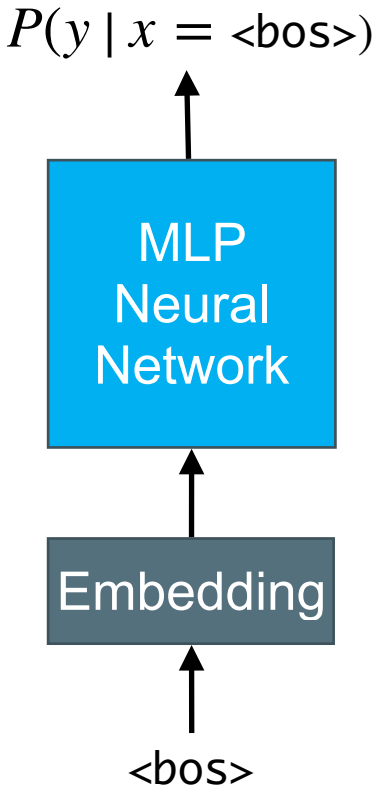
Word	Probability
a	0.01
any	0.01
auto	0.01
all	0.02
best	0.60
The	0.05
…	…

Probability of a sequence

$$P(\texttt{<bos> The best city}) = P(\texttt{<bos>})\, P(\texttt{The} \mid \texttt{<bos>})\, P(\texttt{best} \mid \texttt{The})\, P(\texttt{city} \mid \texttt{best})$$



Generating a sequence

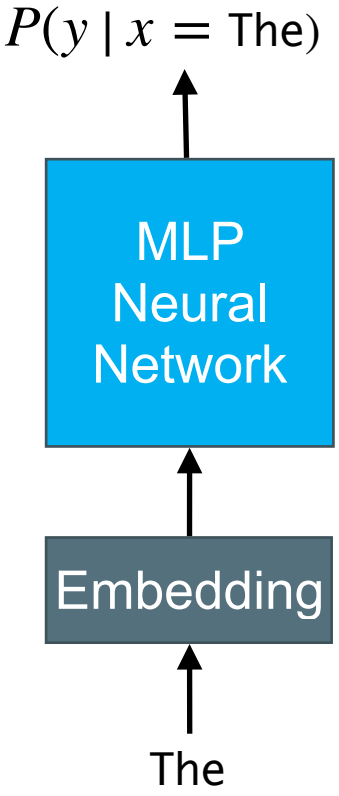


$P(y \mid x = \text{<bos>}) =$

Word	Prob
a	0.20
any	0.01
auto	0.01
all	0.02
best	0.01
The	0.30
…	…

$\text{argmax}_y P(y \mid x = \text{<bos>}) = \text{The}$

Generating a sequence



$P(y \mid x = \text{The}) =$

Word	Prob
a	0.20
any	0.01
auto	0.01
all	0.02
best	0.60
The	0.05
…	…

$\text{argmax}_y P(y \mid x = \text{The}) = \texttt{best}$

Generating a sequence

Choosing the largest probability at every step is called *argmax* decoding.

We can also *sample* from the probability distribution to generate random sequences.

Outline

- Simple Neural Network Language Models
  - Sequential output
- Recurrent neural network
  - Better sequential input and output

Neural networks of sequences

Major problem: sequences can have different lengths

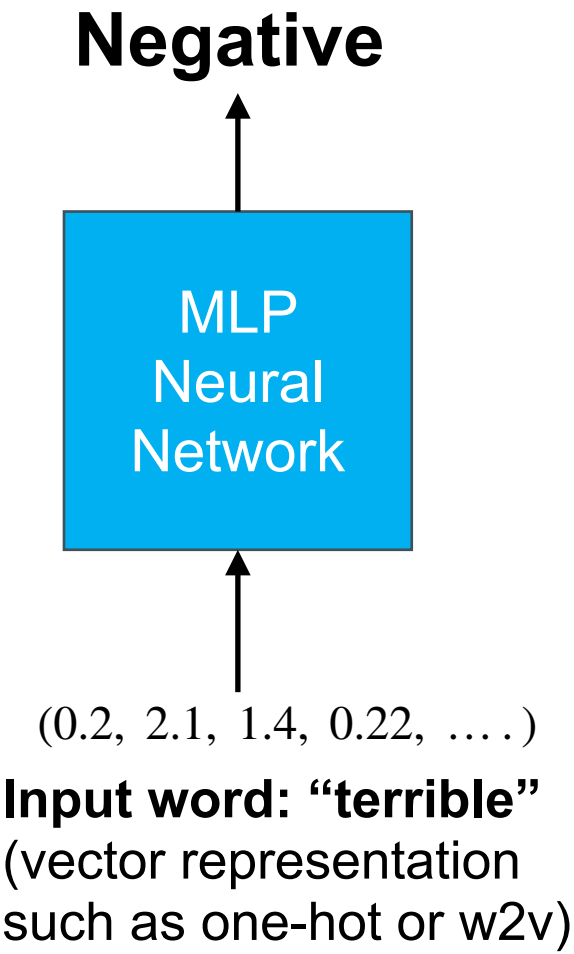
MLP Neural networks take a *fixed size* input.

We want neural networks to accept sequences as input.

Neural networks of sequences

We can input the vector representation of a word and do classification.

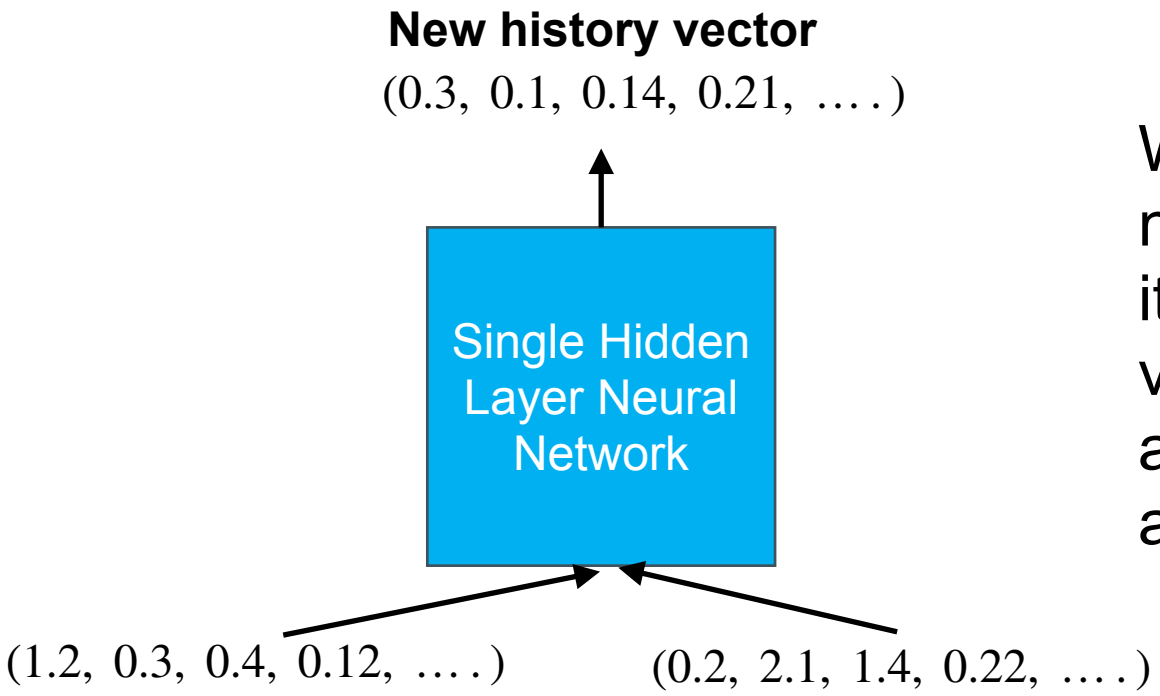
But what about the other words in the document, and the structure?



Recurrent neural network

- Process the input sequence **one word at a time**
- Assume we have a fixed length vector representation of all of the words up to the current index. Called the **history**.
- Then we use a neural network to map the current history and the current word to the **next history representation**.

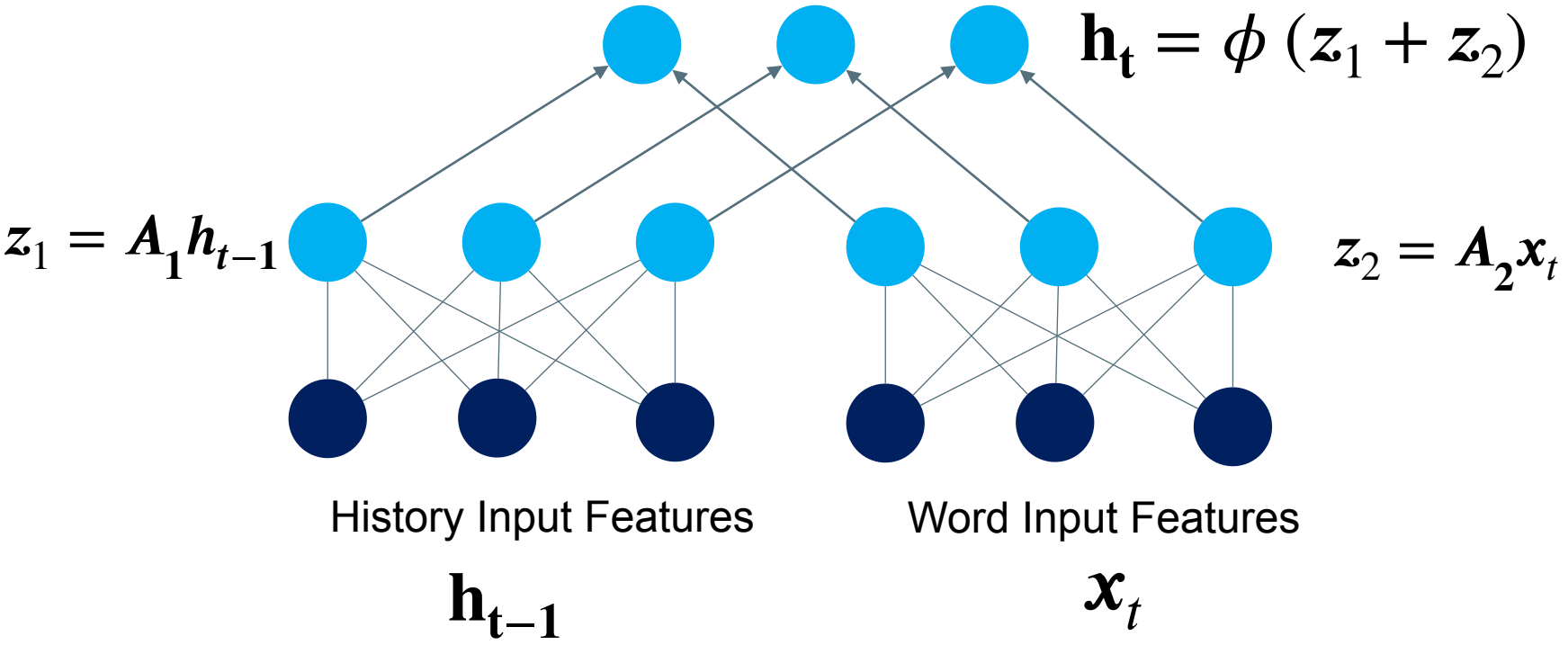
Recurrent neural network



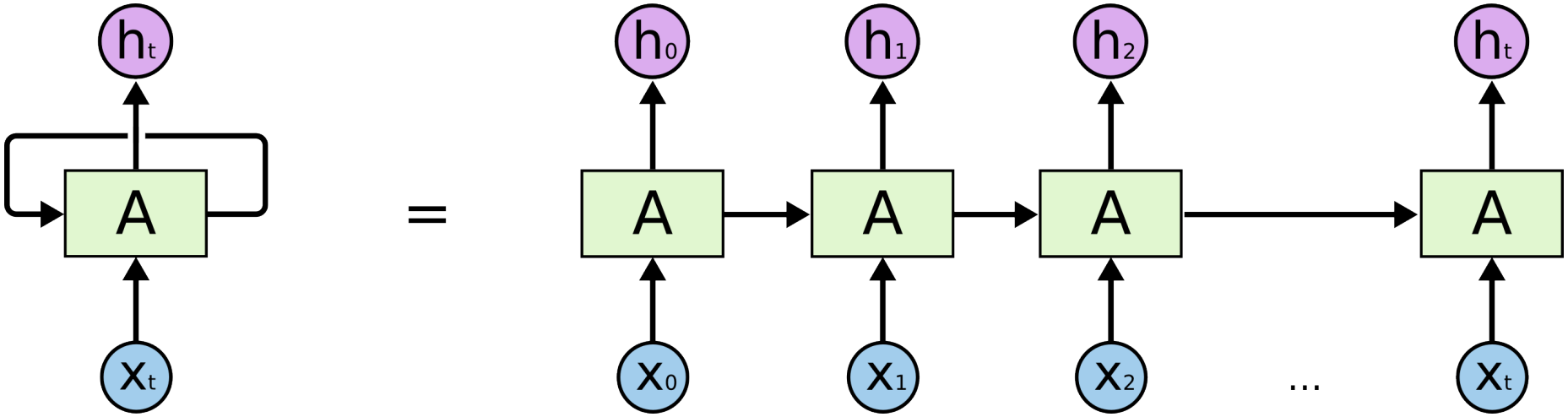
We change the neural network so it accepts two vectors as input and outputs another vector.

Recurrent cell

To accommodate two input vectors we use the following network (i.e. Elman RNN):



Recurrent neural network



The same network is repeated at each step.

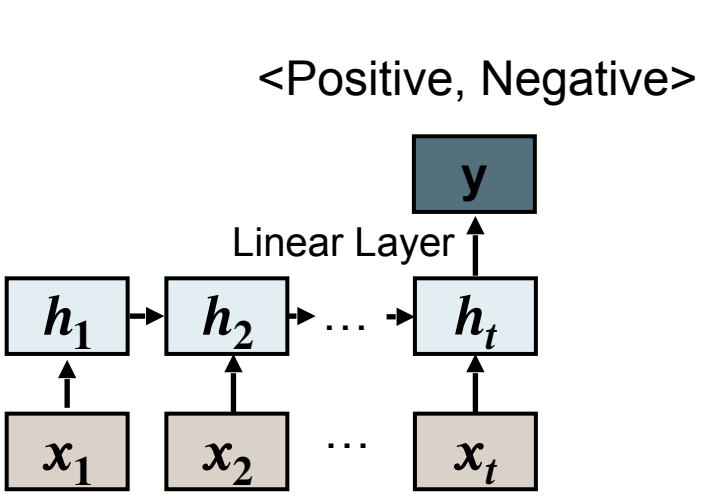
$$h_t = \phi(A_1 h_{t-1} + A_2 x_t + b)$$

h_n is the vector representation of the input elements up to x_n.

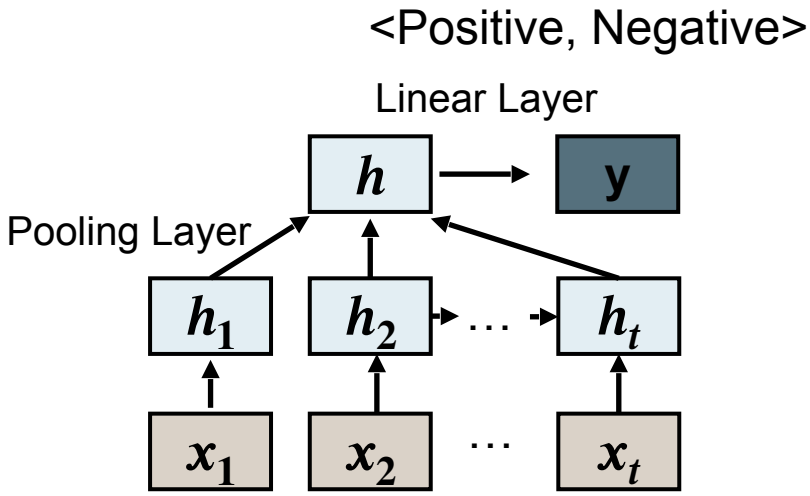
A, b are the (learnable) weights of the neural network.

ϕ is the activation function.

RNN for classification



We don't have to use the outputs at each step we can ignore them and use the last one.

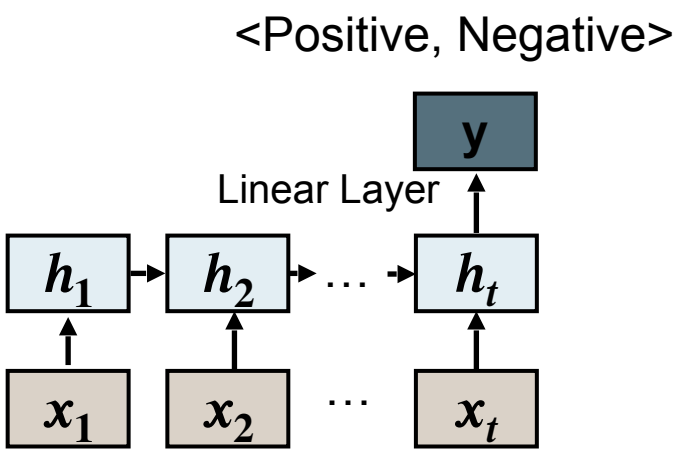


We could also use the outputs at each step and pool them using a function such as: sum, mean, max.

Back-propagation through time

The loss is calculated using the output of the network.

We back-propagate through the entire sequence to train the weights.



Problems with the simple RNN

We are using the *same* neural network at each step of the sequence.

At each step the previous representation gets multiplied by the weight matrix A.

Then the derivative $\frac{dh_n}{dh_0}$ is proportional to A^n.

Problems with the simple RNN

- If the weights in A are less than 1 then A^n decays exponentially (vanishing gradients).
- If the weights in A are greater than 1 then A^n grows exponentially (exploding gradients).
- This makes it very difficult for the RNN to learn representations of long sequences.

More advanced recurrent modules

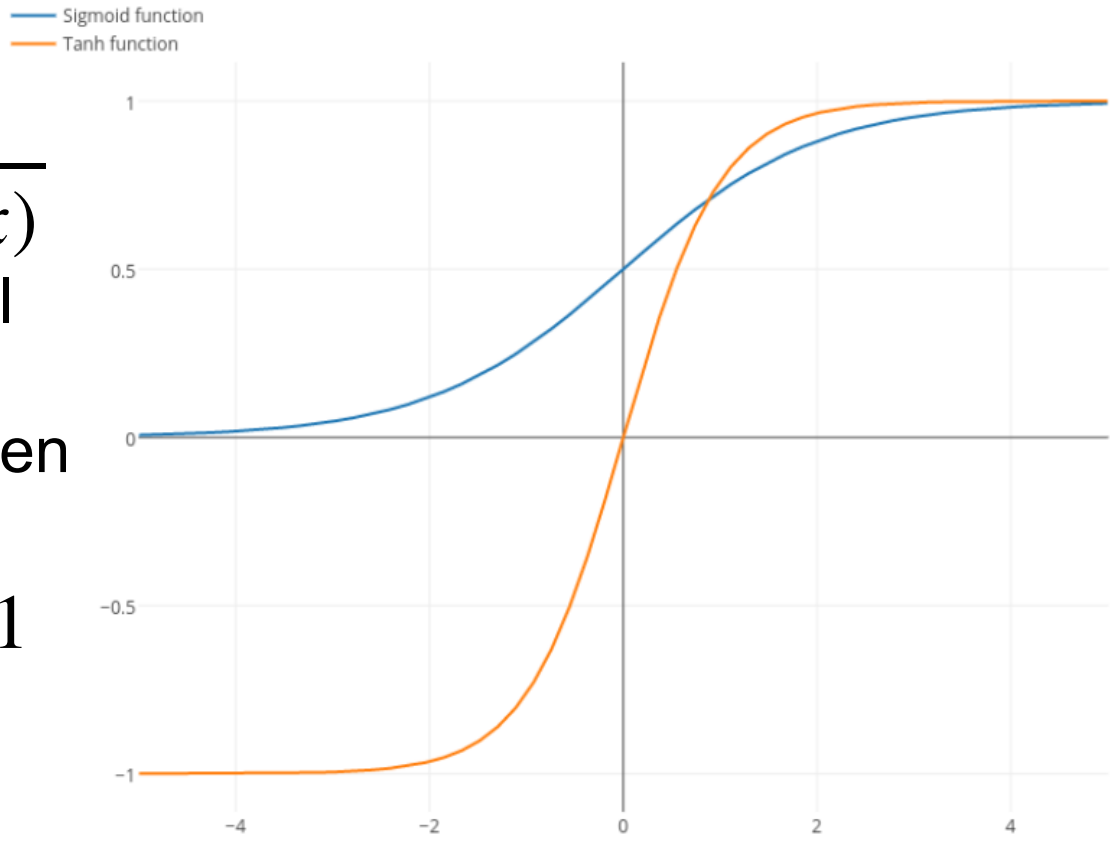
- Better architectures allow learning of longer temporal dependencies.
- They all make use of *residual connections* and *gating mechanisms*.
- Residual connections add the representation from the previous step to the output of the current step.
- Gates are vectors with elements in [0,1], they are multiplied with the output vectors.

- Sigmoid(x) =

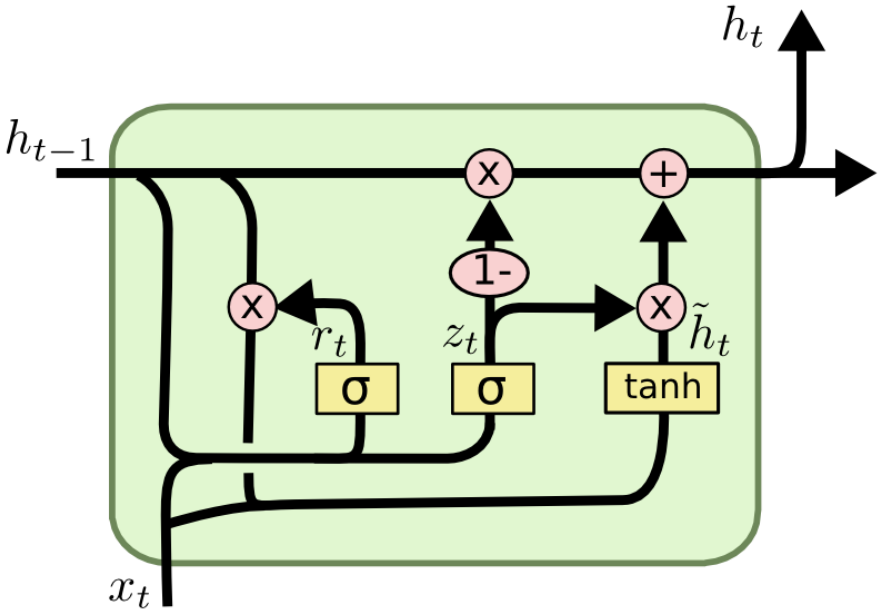
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

is the one dimensional version of softmax, it outputs a value between 0 and 1.

- $\mathrm{Tanh(x)} = 2\sigma(2x) - 1$ is a sigmoid that is rescaled to output between -1 and 1.



Gated Recurrent Unit (GRU)



update gate / reset gate / hidden state

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$
$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Update gate:
which parts of the history to update to new values.

Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Reset gate:
which parts of the history to forget.

Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Candidate history:
compute new history from input
ignoring update gate for now.

Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

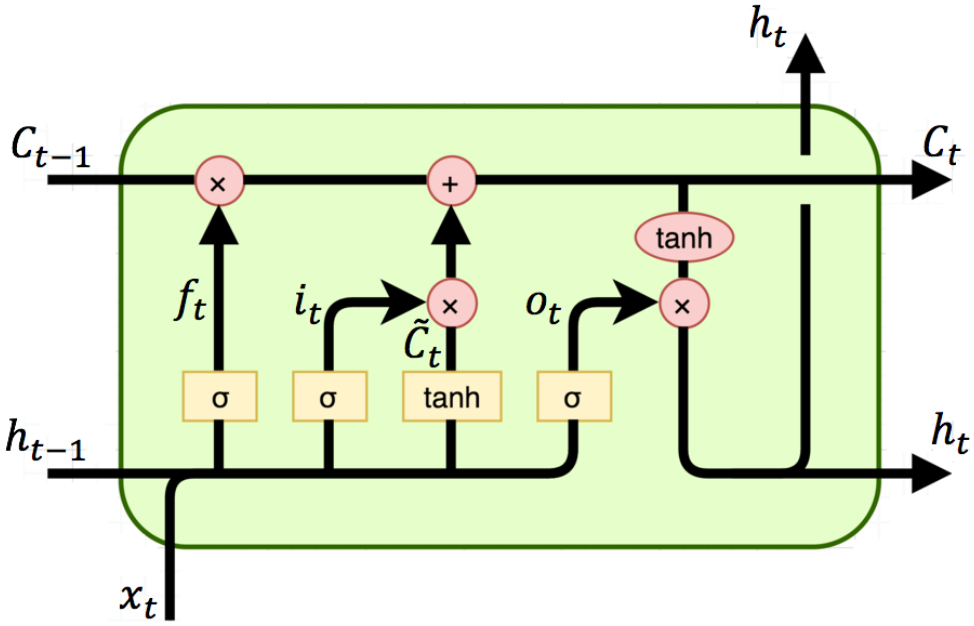
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

New history:
Apply the update gate to calculate the new history.

Long Short-Term Memory (LSTM)



Like the GRU except a separate cell memory C is kept. An output gate is used to output parts of the cell memory.

forget gate / input gate / output gate / cell state / hidden state

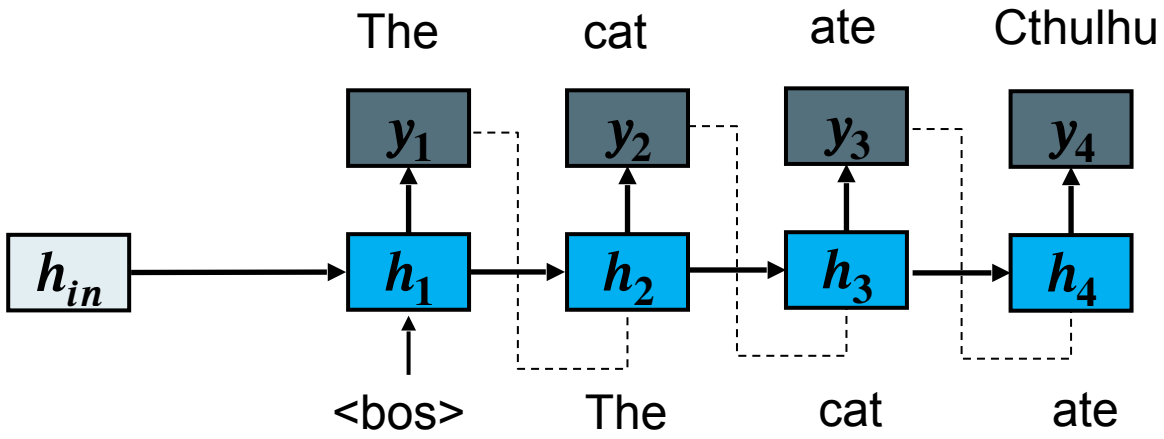
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \tanh(C_t)$$

RNN and terminology

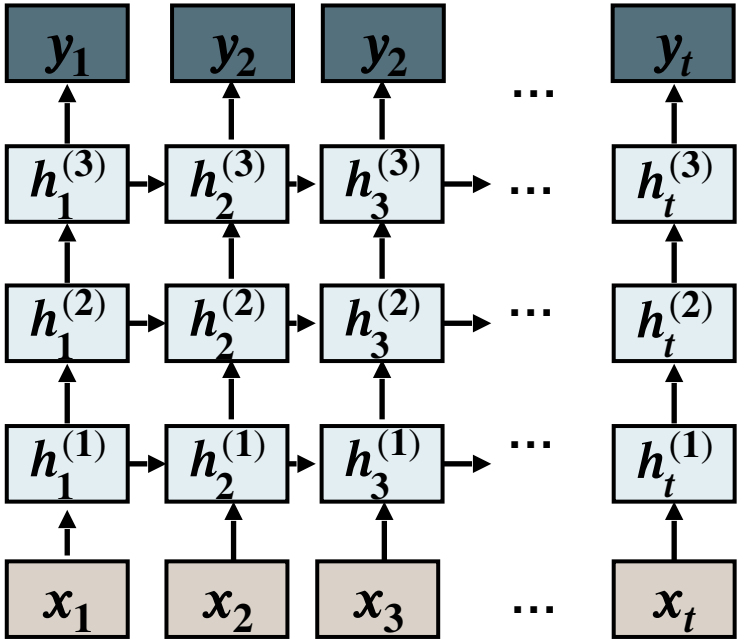
- LSTM and GRU are both drop-in replacements for the Simple RNN.
- The Simple RNN might also be called a Vanilla RNN or an Elman RNN.
- RNN is a generic term for any of these types of models.
- RNN refers to the entire sequential structure, to refer to the network that is repeated we often use RNN Cell or RNN Unit

RNN for language modelling

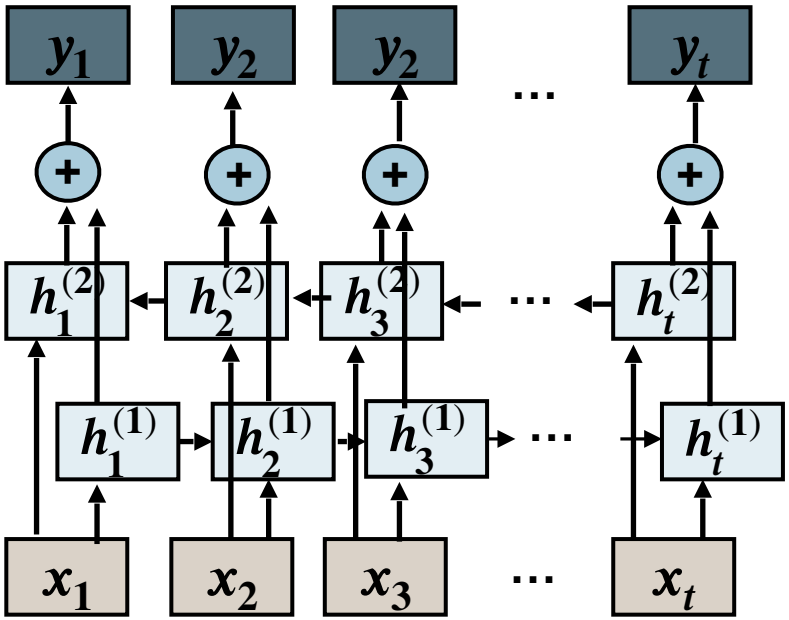
The RNN takes the word generated in the last step as the input to the next step.



Multi-layer RNN



Bi-directional RNN



Padding

When training on sequential data, different elements in the batch can have different lengths. This means they can't be concatenated into one tensor.

- A work-around is to pad sequences with a special '`pad`' token so that they all have the same length.

Padding

[4, 8, 4]
[1, 2]
[4, 3, 3, 4, 1]

[4, 8, 4, 0, 0]
[1, 2, 0, 0, 0]
[4, 3, 3, 4, 1]

Unknown Implicit Structure

Sequential structure (List of words)

Unknown structure?

- We infer structure with attention

Summary

- Recurrent Neural Network
  - Repeat the same network at every step
  - Takes history from previous step as input
  - Train with back-propagation through time

Reference

- Chapter 9, Speech and Language Processing