Autoregressive Models

THE UNIVERSITY OF
SOUTHERN MISSISSIPPI®

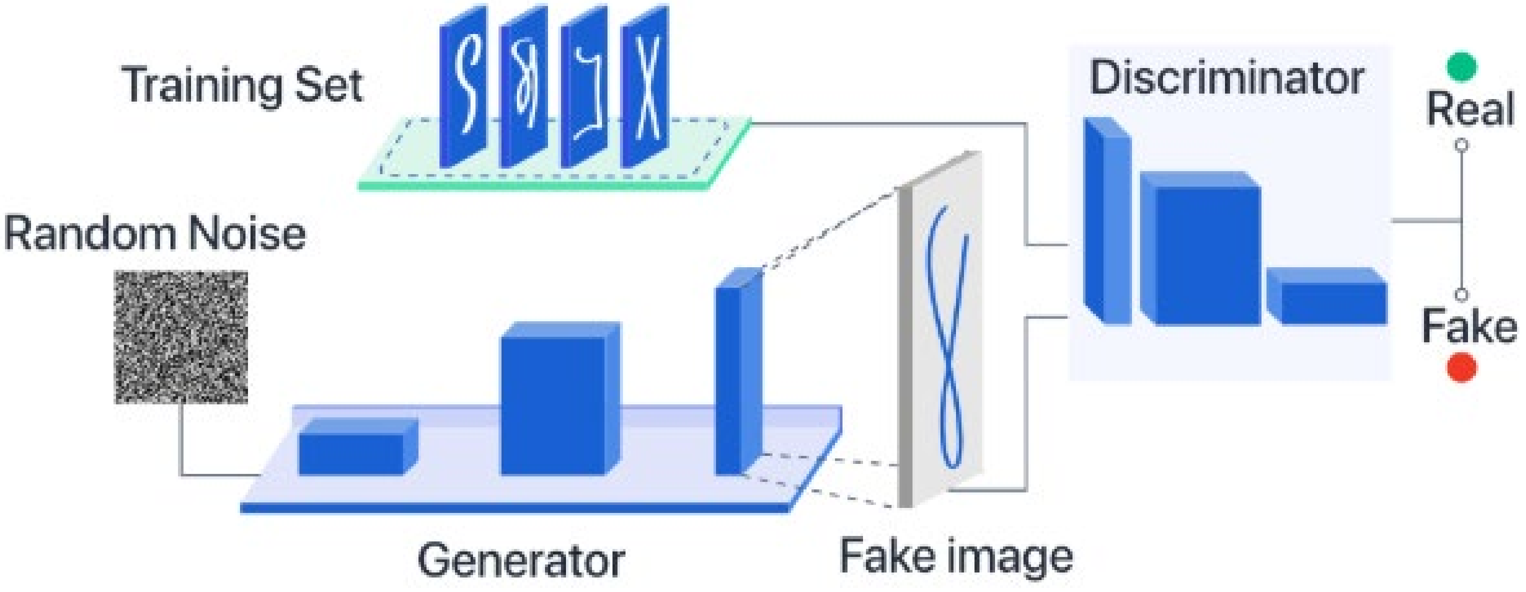
Autoencoders and GANs

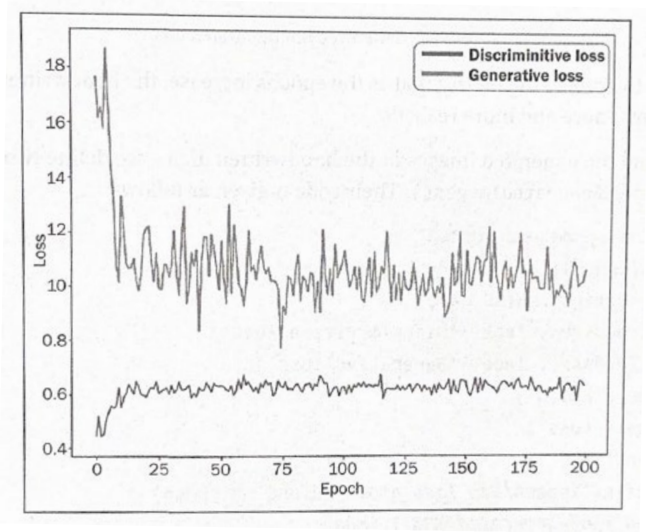
- So far, we have studied a family of generative models:
  - Autoencoder:
    - Samples from latent space
    - Learns how to decode back into the original domain
    - More advanced model: Variational Autoencoder (VAE)

  - Generative Adversarial Model (GAN):
    - Two adversaries:
      - Generator & Discriminator
      - Generator trues to convert random noise into observations that look as if they have been sampled from the original dataset (this is forgery)
        - Similar to the Decoder in AE, samples from latent space using multivariate standard normal distribution
      - Discriminator predicts if an image is real or fake

GAN

- In GAN models:
  - Two networks:
    - Discriminator network: a standard CNN
      - Tries to classify the image as real or generated
    - Generator network:
      - gets its updates from backpropagated values from the discriminator





At first, we would want the D_Loss to be high and G_Loss to be low.
However, a “good” trained model should have low D_Loss and low G_Loss.
This can be a sign of the Generator being able to “trick” the Discriminator.

Autoregressive Models

- Family of generative models that treats a problem as a sequential process.
- They condition predictions based on previous values in sequence rather than on latent space.
- In conclusion:
  - They attempt to **model** the data-generating distribution rather than an approximation of it (as in autoencoders)
  - Some models:
    - LSTM: Long short-term memory: Good for text
    - PixelCNN: Good for image
    - Transformers: Good for text

Autoregressive Models: LSTM

- LSTM is a RNN (recurrent neural network):
  - RNNs contain a recurrent layer (or cell) that can handle sequential data
    - an output of the layer at a particular timestep be part of the input in the next timestep
- Since LSTM is designed to work with sequential data, it fits well with text data.
- Key differences between working with text and image data:
  - Text data is composed of discrete chunks
    - We cannot change the word *cat* to more *cat*
  - Image data is composed of pixel values
    - Pixel data gets filtered, we can backpropagate, calculate loss, etc., to make a pixel more blue.

Autoregressive Models: LSTM

- Text data has a time dimension and no space dimension
  - Order of words is highly important

- Image data has two spatial dimensions but no time dimension
  - Images can be flipped without changing the content

- Text is highly sensitive to small changes in individual units
  - Changing a few words can change the context
  - Changing a few pixels in an image can still be recognizable

Autoencoders: LSTM – Tokenization

- Tokenization:
  - Process of splitting the text into individual units, i.e., words or characters

- When using word tokens:
  - Lower & upper case matters
  - Rare words can be replaced with a token for unknown word to reduce the number of weights in the NN.
  - Words are reduced to its simplest form
  - Tokenize punctuation or eliminate punctuation at all

Autoencoder: LSTM – Tokenization

- When using character token:
  - Generate new sequence of characters to form new words outside of the training vocabulary
  - The vocabulary is smaller – fewer number of weights to be learned by the NN

- Tokenization process:
  - For example: lowercase tokenization, without word stemming, punctuation is tokenized also:

```python
def pad_punctuation(s):
    s = re.sub(f"([{string.punctuation}])", r" \1 ", s)
    s = re.sub(" +", " ", s)
    return s
```

1. Pad punctuation marks to treat them as separate words

```python
text_data = [pad_punctuation(x) for x in filtered_data]
```

2. Convert to TensorFlow dataset

```python
text_ds = (
    tf.data.Dataset.from_tensor_slices(text_data).batch(BATCH_SIZE).shuffle(1000))
```

3. Create a Keras TextVectorization layer to convert text to lowercase
Give the most 10,000 prevalent words a corresponding integer token
Trim or pad the sequence to 201 tokens

```python
vectorize_layer = layers.TextVectorization(standardize="lower",
    max_tokens=10000,
    output_mode="int",
    output_sequence_length=200 + 1,)
```

4. Apply the TextVectorization layer to the training data

```python
vectorize_layer.adapt(text_ds)
```

5. Store a list of the word tokens

```python
vocab = vectorize_layer.get_vocabulary()
```

Autoencoder: LSTM – Tokenization

- For example:
- “Recipe for Ham Persillade with Mustard Potato Salad”
- Tokenized:
  - [26 16 557 1 8 298 335 189 …
  - TextVectorization Layer: creates a “map” (sort of an index, based on frequency)
    - 0:
    - 1: [UNK]
    - 2: .
    - 3: ,
    - 4: and
    - 5: to
    - 6: in
    - 7: the
    - 8: with
    - 9: a
    - …

Autoregressive Models: LSTM – Predict next sequence

- We need to train the LSTM model to predict the next word in a sequence, given a sequence of words preceding
- For example:
  - Feed model the tokens for: “grilled chicken with boiled ____”
  - Expected output: potatoes
  - Rather than: bananas
- One way to do it:
  - Shifting the entire sequence by one token to create our own target variable

Autoregressive Models: LSTM – Predict next sequence

- The LSTM architecture:
  - Input of the model: sequence of integer tokens
  - Output of the model: probability of each word in the 10,000-word vocabulary appearing next in the sequence.

  - We need two layers:
    - Embedding
    - LSTM

Autoregressive Models: LSTM – Predict next sequence

- Embedding layer:
  - In an essence, it is a look-up table that converts each integer token into a vector of length “embedding size”

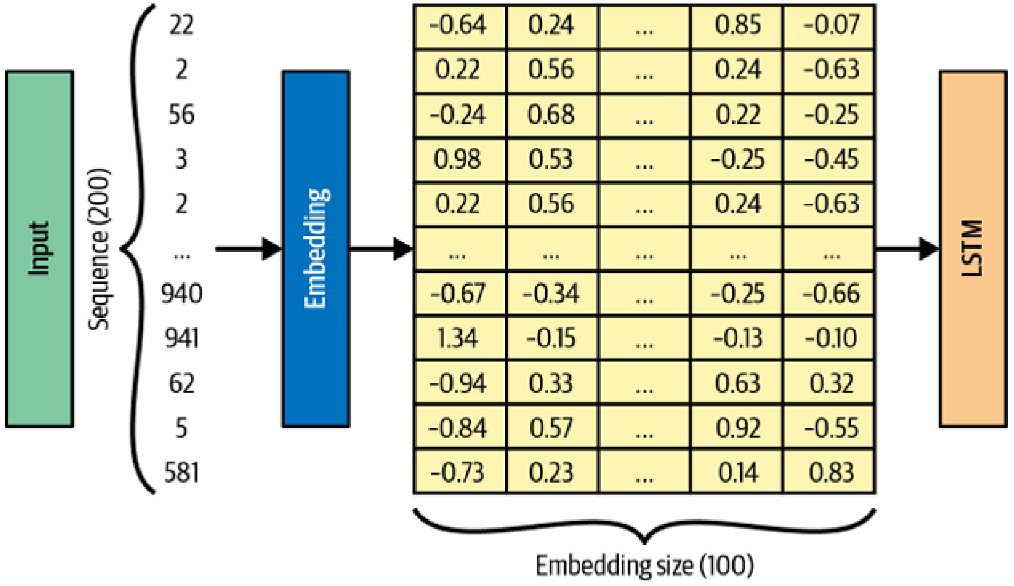
Token	Embedding				
0	-0.13	0.45	...	0.13	-0.04
1	0.22	0.56	...	0.24	-0.63
...	...	...	...	...	...
9998	0.16	-0.70	...	-0.35	1.02
9999	-0.98	-0.45	...	-0.15	-0.52

Vocabulary size (10,000)

Embedding size (100)

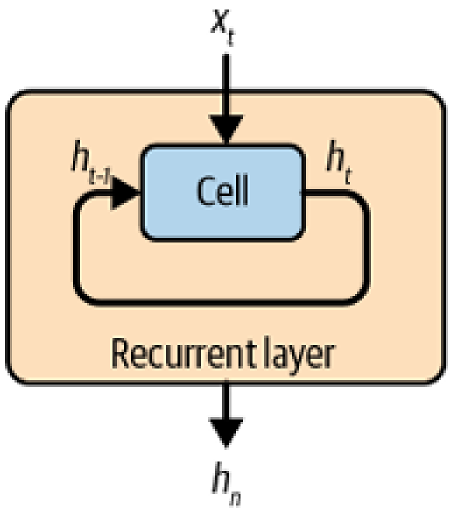
Autoregressive Models: LSTM – Predict next sequence

- The input layer passes a tensor of integer sequences to the embedding layer
- The embedding layer outputs a tensor that is passed to the LSTM layer



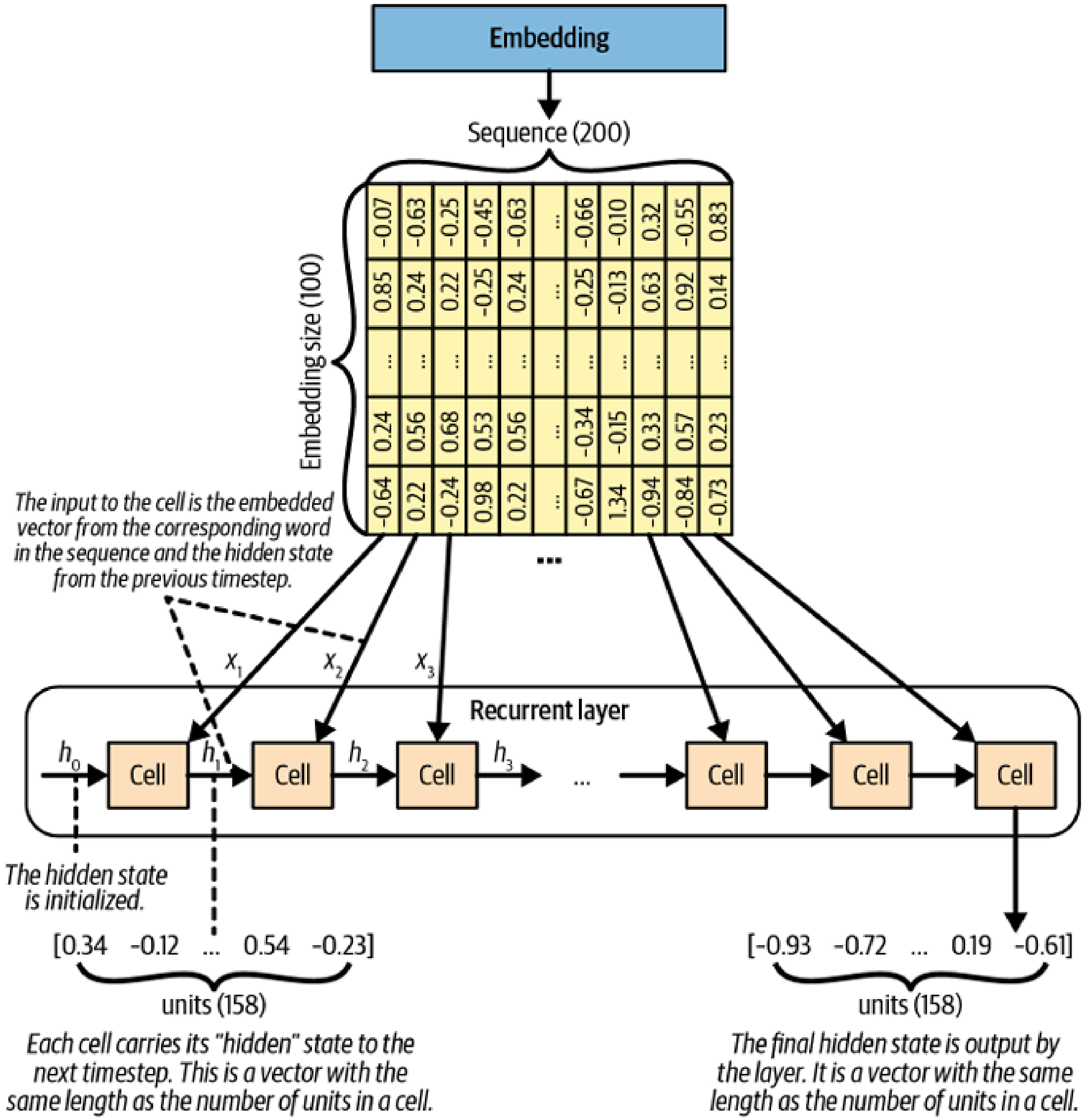
Autoregressive Models: LSTM – Predict next sequence

- LSTM layer
  - First, what is “recurrent”?
    - A recurrent layer consist of a cell that updates its hidden state as each element of the sequence is passed through it, one timestep at a time.
    - Hidden state:
      - A vector with length equal to the number of units in the cell
    - How does it work?
      - Given a sequential input: $x_1, \ldots, x_n$
      - A hidden state: h_t
      - At timestep t:
        - Use previous value of the hidden state, $_{ht\text{-}1}$
        - Data of the current timestep, x_1
        - Produce an updated state vector, h_t
        - Continue until the end of the sequence
        - Output the final hidden state, h_n , to the next layer of the network



Autoregressive Models: LSTM – Predict next sequence

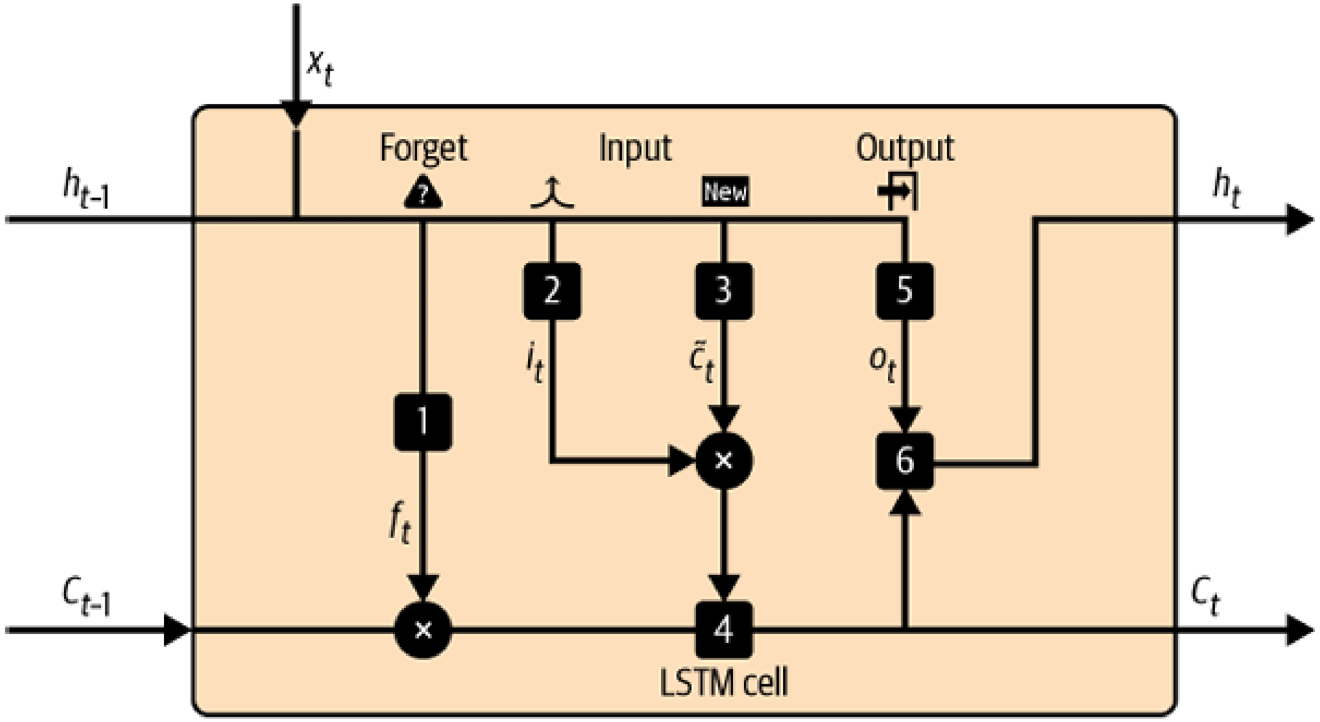
- Embedding and LSTM layers
  - Units and cells:
    - There is one cell defined by a number of units it contains. (think of a prison cell that holds multiple prisoners). The number of units is set when defining the layer.



Autoregressive Models: LSTM – Predict next sequence

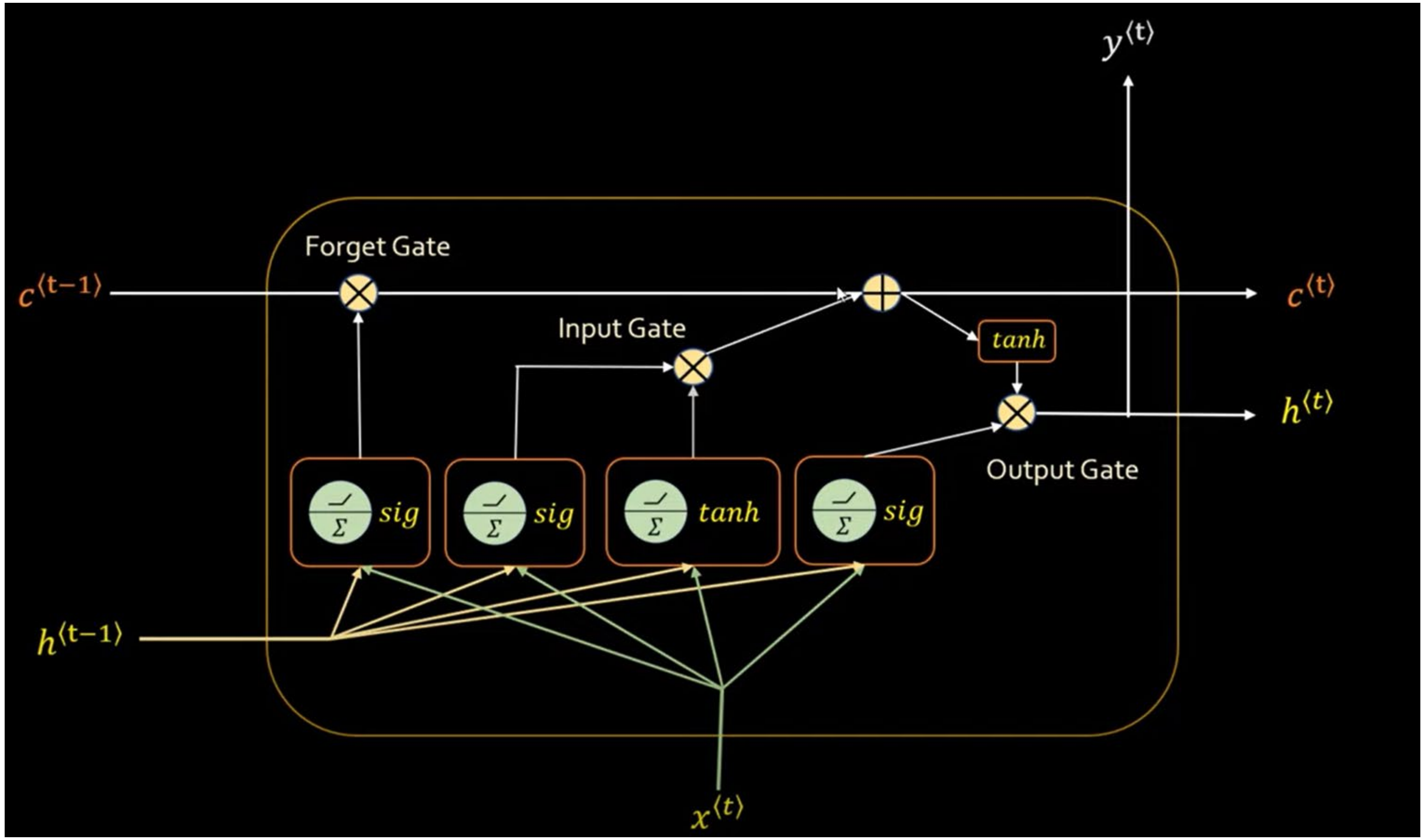
- The LSTM Cell:
  - The hidden state is updated in six steps (six neural networks):
    1. Hidden state of the previous timestep and the current word embedding x_t are concatenated and passed to the forget gate
       - The forget gate: NN with sigmoid activation
    2. Result of step 1, passed on an input gate:
       - Input gate: a NN with sigmoid activation
    3. Result of step 2, passed through a NN with tanh activation
    4. Resulting vector and cell state from step 1 are multiplied and added with the output of step 2 and step3 vector (element-wise)
    5. Result of step 4 is passed through an output gate:
       - Output gate: NN with sigmoid activation
    6. Result of step 5 is uses tanh activation to produce a new hidden state, h_t

Autoregressive Models: LSTM – Predict next seauence



1. $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2. $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
3. $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
4. $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
5. $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
6. $h_t = o_t * \tanh(C_t)$

Autoregressive Models: LSTM – Predict next sequence



Autoregressive Models: LSTM – Training

- Most important step in training:
  - The Dense layer transforms the hidden states at each timesteps into a vector of probabilities for the next token
  - The overall Model predicts the next token, given an input sequence of tokens. It does this for each token in sequence.

Autoregressive Models: LSTM – Training

- Notice the sequence of layers:
  - 1. input
  - 2. embedding
  - 3. LSTM
  - 4. Dense

```
inputs = layers.Input(shape=(None,), dtype="int32")
x = layers.Embedding(VOCAB_SIZE, EMBEDDING_DIM)(inputs)
x = layers.LSTM(N_UNITS, return_sequences=True)(x)
outputs=layers.Dense(VOCAB_SIZE,activation="softmax")(x)
lstm = models.Model(inputs, outputs)
lstm.summary()
```

Autoregressive Models: LSTM – Predict next sequence

- Example:
  - Train an LSTM model to generate new cooking recipies

THE UNIVERSITY OF
SOUTHERN MISSISSIPPI®