Sapienza Training Camp 2021

Building an Image Search Engine

2 - 4 September, 2021

Roadmap

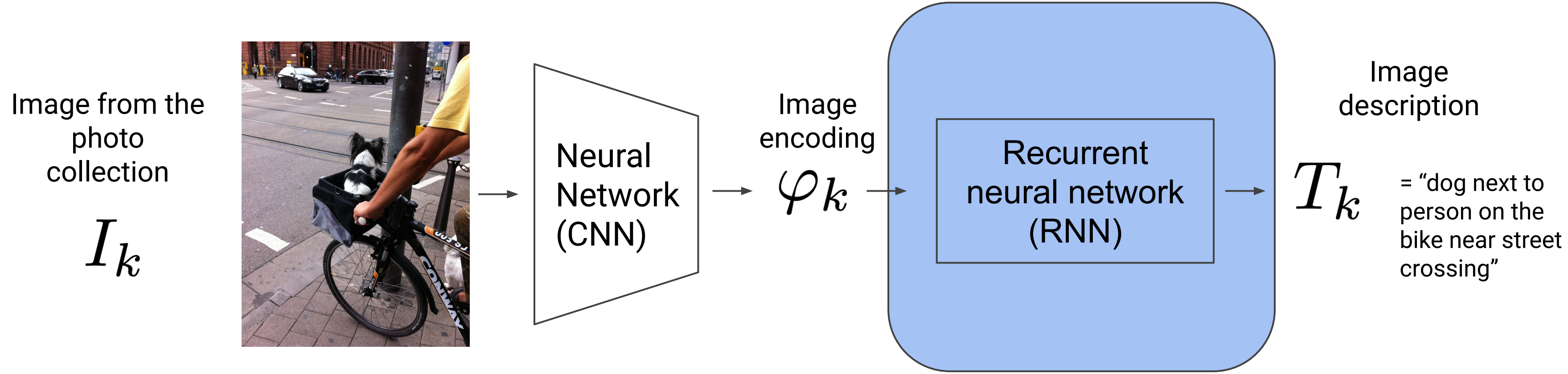


Image from the photo collection

I_k

Neural Network (CNN)

Image encoding

φ_k

Recurrent neural network (RNN)

Image description

T_k

= “dog next to person on the bike near street crossing”

Define similarity function. Order images according to similarity to the query.

Q Query: “person walking with a dog on the beach”

$$\mathrm{sim}(Q, T_1) > \mathrm{sim}(Q, T_2)$$

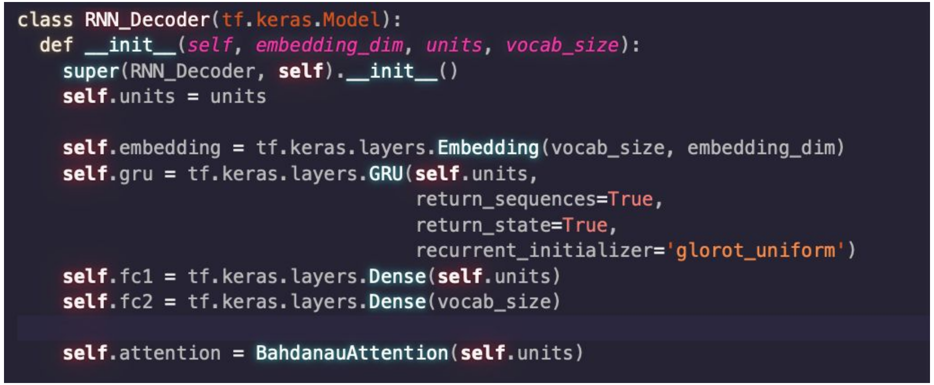
Highlights in Natural Language Processing (NLP)

- Google Translate
    - translate.google.com
- BERT language model used in Google search
    - Google uses AI to boost search engine ranking efficiency, FT.com, Oct. 2019
- OpenAI’s GPT language model:
    - https://openai.com/blog/better-language-models

Recurrent neural networks

Image captioning model from:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/text/image_captioning.ipynb

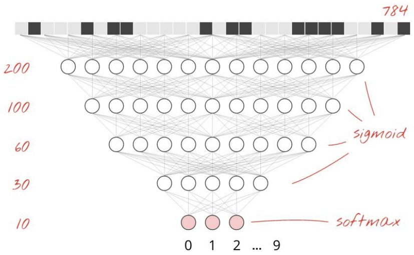


```
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)
```

Recap: dense and convolutional layers

- Dense layer:

$$\mathbf{o} = \mathrm{g}(W\mathbf{x} + w_0), \quad \text{where } \mathbf{x} \in \mathbb{R}^d$$

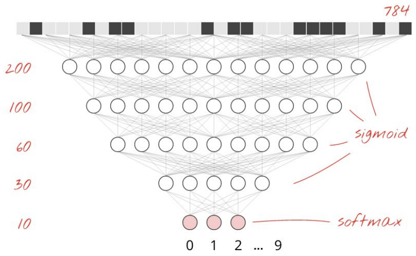


Images from "TensorFlow, Keras and deep learning, without a PhD"

Recap: dense and convolutional layers

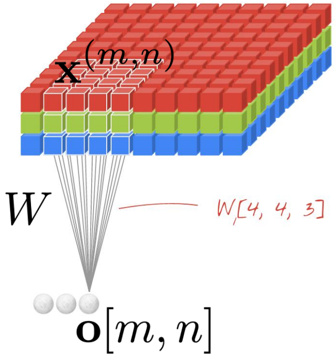
- Dense layer:

$$\mathbf{o} = \mathrm{g}(W\mathbf{x} + w_0), \quad \text{where } \mathbf{x} \in \mathbb{R}^d$$



- Convolutional layer:

$$\mathbf{o}[m,n] = \mathrm{g}(W\mathbf{x}^{(m,n)} + w_0), \quad \text{where } \mathbf{x}^{(m,n)} = \mathbf{x}[m\!:\!m+D, n\!:\!n+D]$$

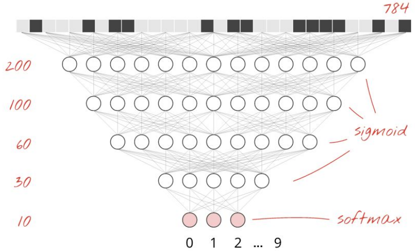


Images from "TensorFlow, Keras and deep learning, without a PhD"

Recap: dense and convolutional layers

- Dense layer:

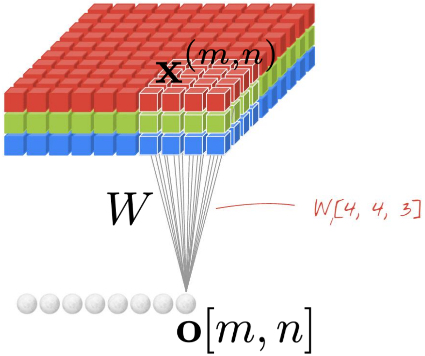
$$\mathbf{o} = \mathrm{g}(W\mathbf{x} + w_0), \quad \text{where } \mathbf{x} \in \mathbb{R}^d$$



- Convolutional layer:

$$\mathbf{o}[m,n] = \mathrm{g}(W\mathbf{x}^{(m,n)} + w_0), \quad \text{where } \mathbf{x}^{(m,n)} = \mathbf{x}[m\!:\!m{+}D, n\!:\!n{+}D]$$

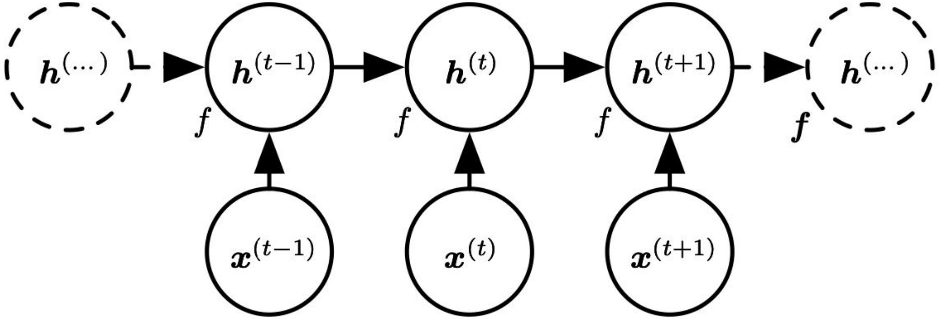
  - weight sharing: same weights used for all local windows $\mathbf{x}^{(m,n)}$
  - convolutional layer supports variable size input



Images from "TensorFlow, Keras and deep learning, without a PhD"

Recurrent layer

- Recurrent layer
  - weight sharing across time steps
  - recurrent layer supports sequences of variable size



$$\mathbf{h}^{(t)} = \mathrm{g}(W\mathbf{h}^{(t-1)} + U\mathbf{x}^{(t)} + w_0)$$

$$\text{where } \mathbf{h}^{(t)} \in \mathbb{R}^K$$

Image from "https://www.deeplearningbook.org/slides/10_rnn.pdf"

Recurrent layer configurations

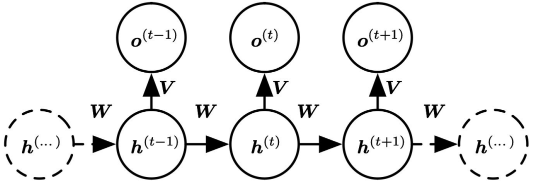


Image description (our application!)

Image from "https://www.deeplearningbook.org/slides/10_rnn.pdf"

Recurrent layer configurations

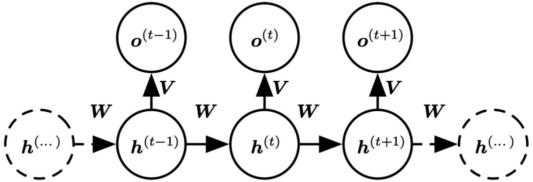
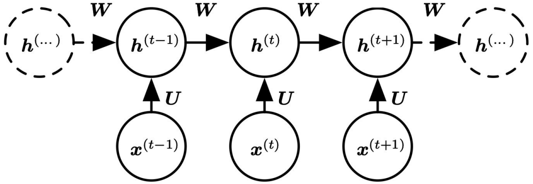


Image description (our application!)



Text classification

Image from “https://www.deeplearningbook.org/slides/10_rnn.pdf”

Recurrent layer configurations

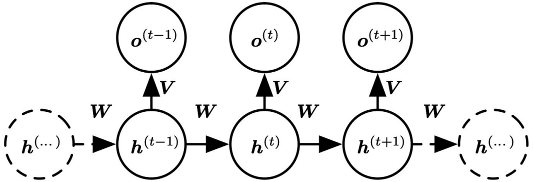
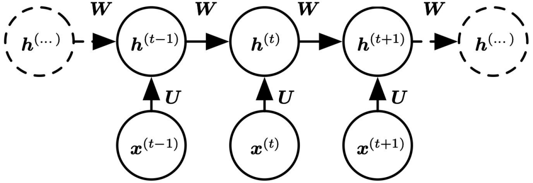
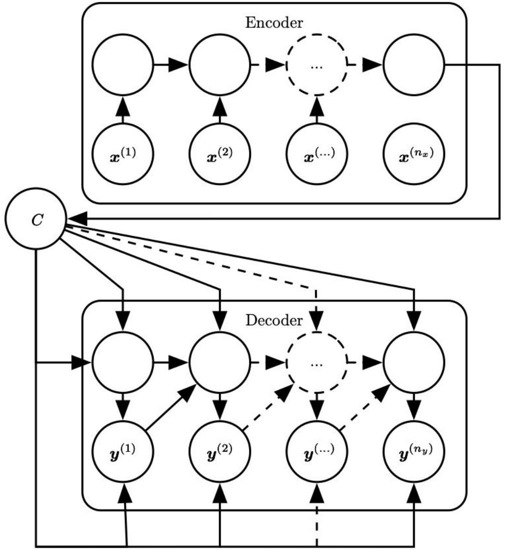


Image description (our application!)



Text classification



Machine translation

Image from "https://www.deeplearningbook.org/slides/10_rnn.pdf"

Image captioning model

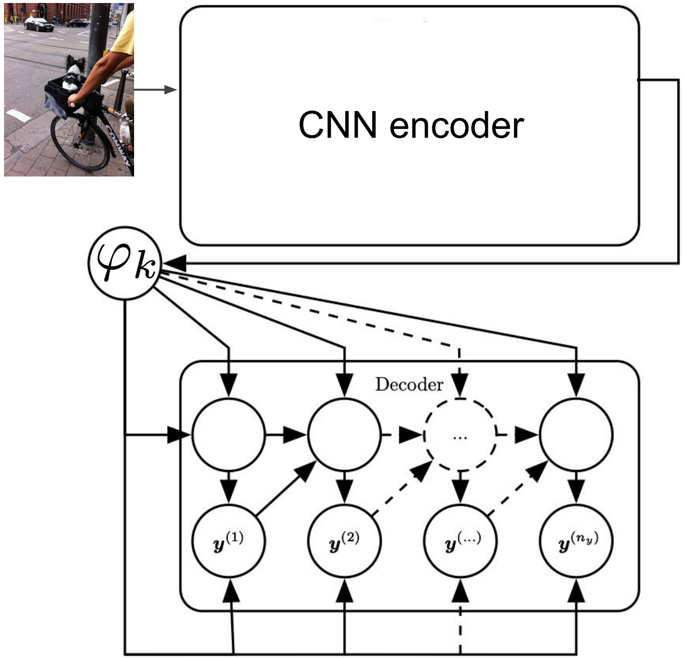
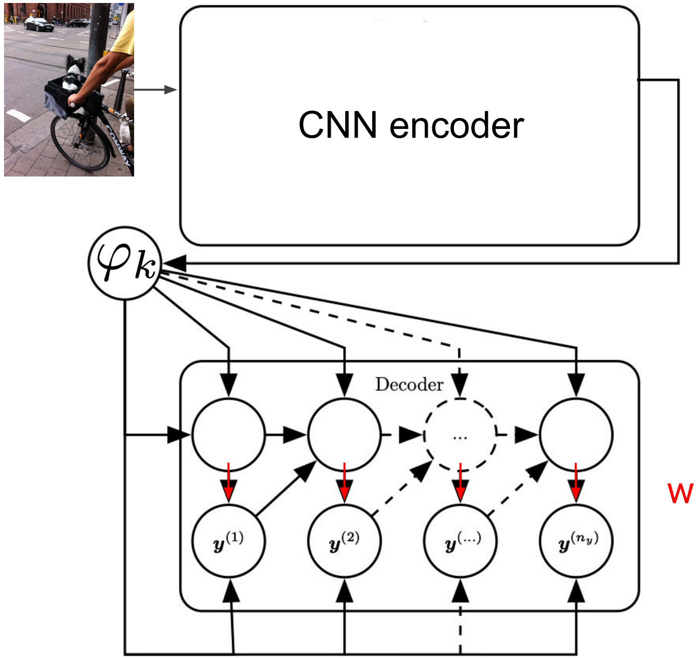
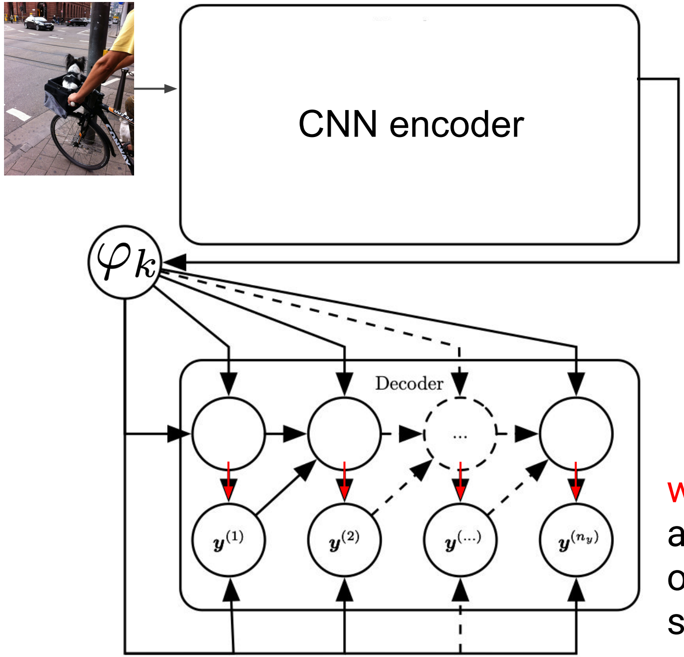


Image captioning model



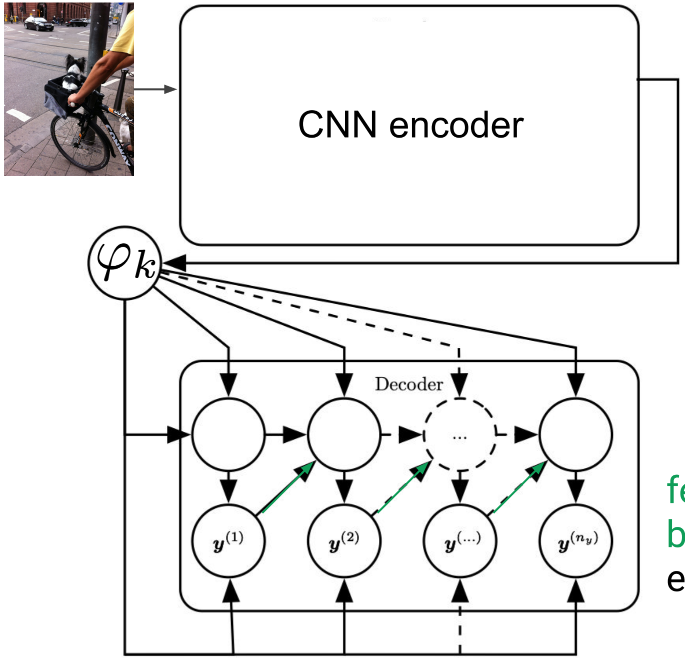
word generation

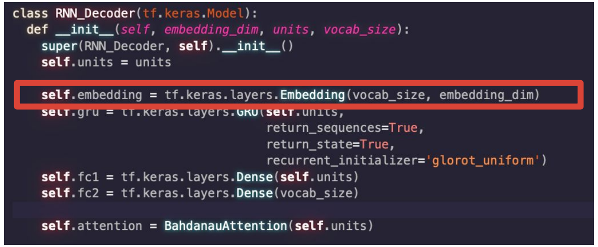
Image captioning model



word generation: output a vector with probability of each word (via softmax)

Image captioning model





```
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)
```

feed generated word back to RNN: use embedding layer

Model training

Predicted labels

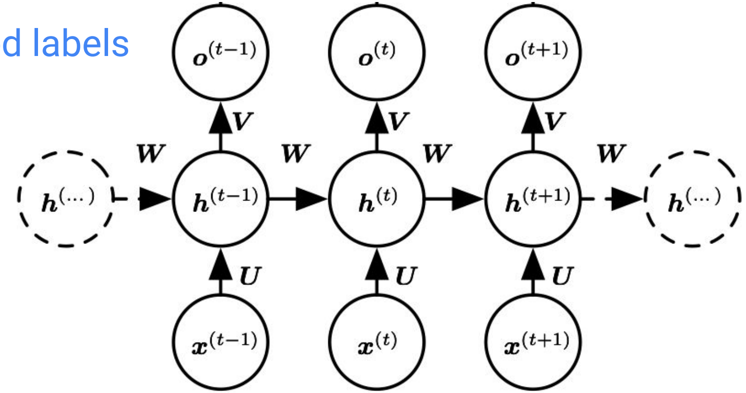


Image from
"https://www.deeplearningbook.org/slides/10_rnn.pdf"

Model training

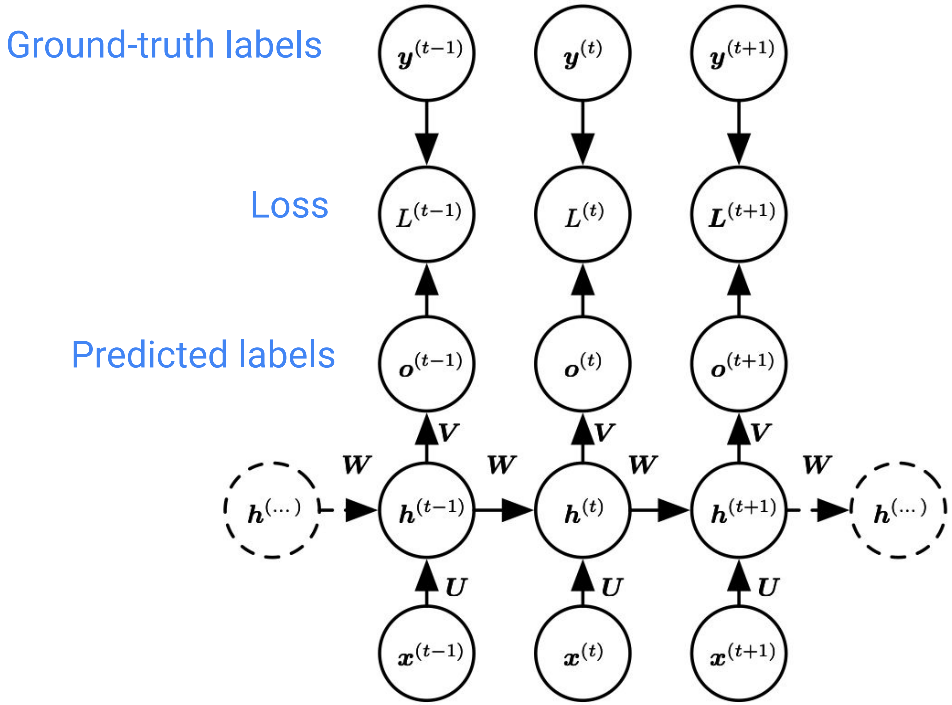
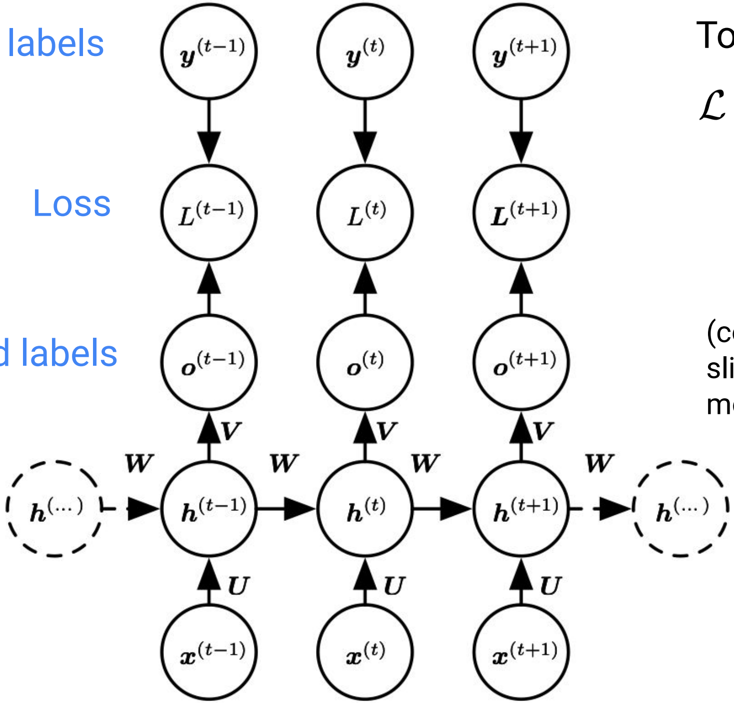


Image from "https://www.deeplearningbook.org/slides/10_rnn.pdf"

Model training



Total loss:

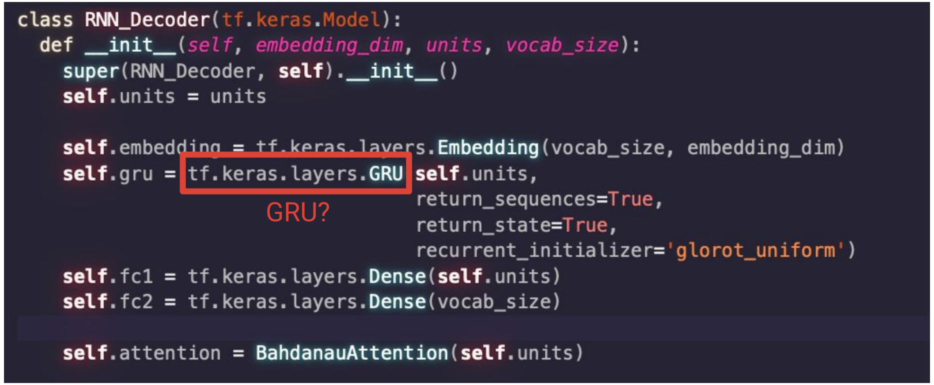
$$\mathcal{L} = \sum_t \mathcal{L}^{(t)}$$

$$= -\sum_t \log p_{\text{RNN}}(y^{(t)} | X^{(1:t)}; \theta)$$

(compare to the cross-entropy loss on slide 17 in the “introduction to NN” module)

Image from “https://www.deeplearningbook.org/slides/10_rnn.pdf”

More complex RNN units: LSTM and GRU



```
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)
```

GRU?

Take a quiz!