Sapienza Training Camp 2020

Building an Image Search Engine

3 - 5 September, 2020

Agenda for today

- Word and sentence embeddings
- Recurrent neural networks

Roadmap

Image from the photo collection

I_k

→ Neural Network (CNN) → Image encoding φ_k → Recurrent neural network (RNN) → T_k

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"

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

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"

Q Query: "person walking with a dog on the beach"

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

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

Text embeddings

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

V is a fixed-size vocabulary, each word is mapped to its index in the vocabulary

$$Q \rightarrow \mathbf{e}(Q) \in \mathbb{R}^{|V|}$$ embedding vector

Text embeddings

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

V is a fixed-size vocabulary, each word is mapped to its index in the vocabulary

$$Q \rightarrow \mathbf{e}(Q) \in \mathbb{R}^{|V|}$$ embedding vector

$$\mathbf{e}(Q) = [0, \ldots, 1, 0, 1, \ldots, 0]$$

$\mathbf{e}(Q)_i$ - indicates presence/absence of a vocabulary term with index i

Text embeddings

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

V is a fixed-size vocabulary, each word is mapped to its index in the vocabulary

$$Q \rightarrow \mathbf{e}(Q) \in \mathbb{R}^{|V|}$$ embedding vector

$$\mathbf{e}(Q) = [0, \ldots, 1, 0, 1, \ldots, 0]$$

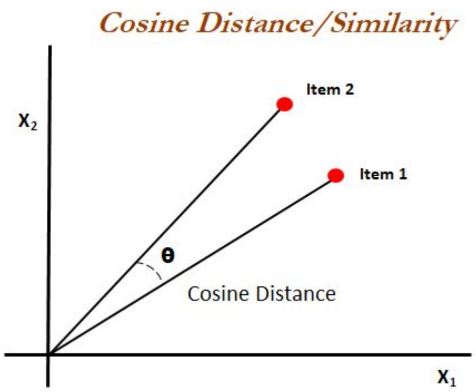
$\mathbf{e}(Q)_i$ - indicates presence/absence of a vocabulary term with index i

$\mathbf{e}(Q)^T \mathbf{e}(T)$ = how many words are in common between query Q and text T

Cosine similarity

- Define similarity between query and image description as

$$\mathrm{sim}(Q,T) = \frac{\mathbf{e}(Q)^T\mathbf{e}(T)}{\|\mathbf{e}(Q)\|\|\mathbf{e}(T)\|}$$



- https://en.wikipedia.org/wiki/Cosine_similarity

Image from https://www.oreilly.com

tf-idf

- Observation: not all words are equally important for comparing query and image description
  - “the”, “on” - not as important as “dog” or “beach”
  - problems with synonyms, e.g. “bike” vs. “bicycle”
  - different wording expresses the same concept: “riding on a bike” vs. “biking”

tf-idf

- Observation: not all words are equally important for comparing query and image description
  - "the", "on" - not as important as "dog" or "beach"
- Good step forward is to switch from binary embedding to tf-idf weights

how many times j occurs in T

$$\mathbf{e}(T)_i = \left[\frac{f_{i,T}}{\sum_j f_{j,T}}\right]\left[\log\frac{|\mathcal{D}|}{|d \in \mathcal{D} : d \text{ contains term i}|}\right]$$

- https://en.wikipedia.org/wiki/Tf–idf

tf-idf

- Observation: not all words are equally important for comparing query and image description
  - “the”, “on” - not as important as “dog” or “beach”
- Good step forward is to switch from binary embedding to tf-idf weights

some representative set of documents

$$\mathbf{e}(T)_i = \left[\frac{f_{i,T}}{\sum_j f_{j,T}}\right] \left[\log \frac{|\mathcal{D}|}{|d \in \mathcal{D} : d \text{ contains term i}|}\right]$$

- https://en.wikipedia.org/wiki/Tf–idf

tf-idf

- Observation: not all words are equally important for comparing query and image description
  - “the”, “on” - not as important as “dog” or “beach”
- Good step forward is to switch from binary embedding to tf-idf weights

term frequency

$$\mathbf{e}(T)_i = \left[\frac{f_{i,T}}{\sum_j f_{j,T}}\right] \left[\log \frac{|\mathcal{D}|}{|d \in \mathcal{D} : d \text{ contains term i}|}\right]$$

- https://en.wikipedia.org/wiki/Tf–idf

tf-idf

- Observation: not all words are equally important for comparing query and image description
  - “the”, “on” - not as important as “dog” or “beach”
- Good step forward is to switch from binary embedding to tf-idf weights

inverse document frequency

$$\mathbf{e}(T)_i = \left[\frac{f_{i,T}}{\sum_j f_{j,T}}\right]\left[\log \frac{|\mathcal{D}|}{|d \in \mathcal{D} : d \text{ contains term i}|}\right]$$

- https://en.wikipedia.org/wiki/Tf–idf

Word and sentence embeddings

- Ideally we want the embedding to stay similar in the presence of factors such as
  - synonyms, e.g. “bike” vs. “bicycle”
  - paraphrasing: e.g. “riding on a bike” vs. “biking”
- Consider more complex word embeddings:
  - word2vec
  - Glove
  - ...
- Many embeddings available:
  - https://github.com/chakki-works/chakin
- Construct sentence embeddings:
  - paper: “A simple but tough-to-beat baseline for sentence embeddings”, https://openreview.net/pdf?id=SyK00v5xx

Agenda for today

- Word and Sentence Embeddings
- Recurrent Neural Networks

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:

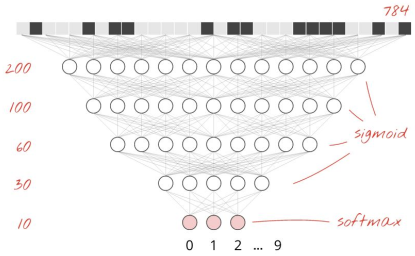
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/text/image_captioning.ipynb

```python
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:

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

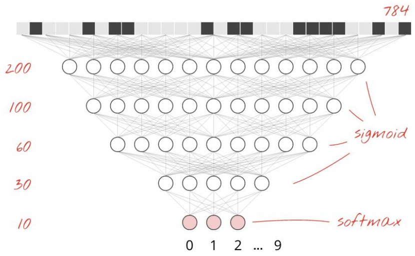


Images from "TensorFlow, Keras and deep learning, without a PhD"

Recap: dense and convolutional layers

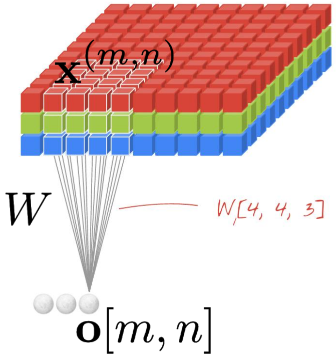
- Dense layer:

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



- Convolutional layer:

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

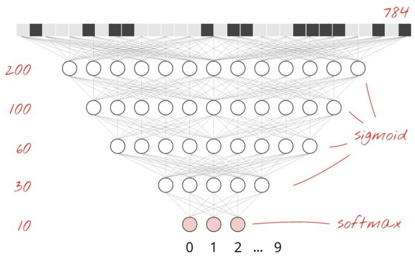


Images from ["TensorFlow, Keras and deep learning, without a PhD"]

Recap: dense and convolutional layers

- Dense layer:

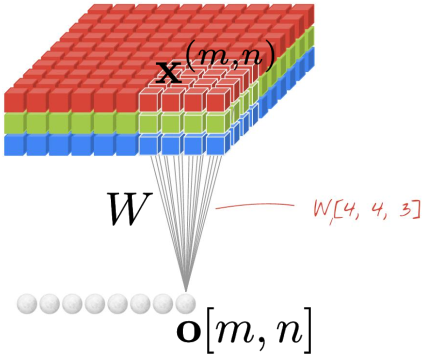
$$\mathbf{o} = \mathrm{g}(W\mathbf{x} + w_0), \quad \text{where } \mathbf{x} \in \mathbb{R}^d$$



- Convolutional layer:

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

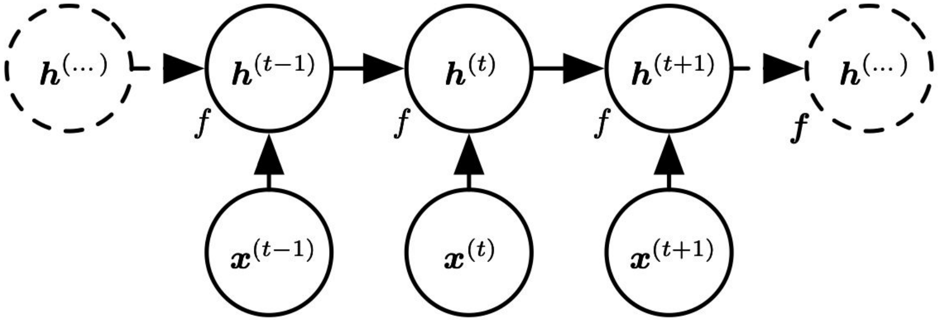
  - weight sharing: same weights used for all local windows $\mathbf{x}^{(m,n)}$
  - convolutional layer supports variable size input



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



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

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

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

Recurrent layer configurations

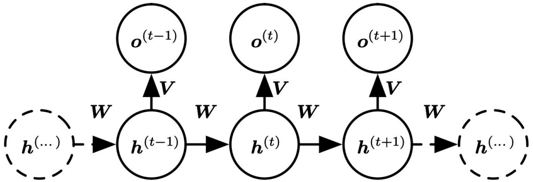


Image description (our application!)

Image from [“https://www.deeplearningbook.org/slides/10_rnn.pdf”](https://www.deeplearningbook.org/slides/10_rnn.pdf)

Recurrent layer configurations

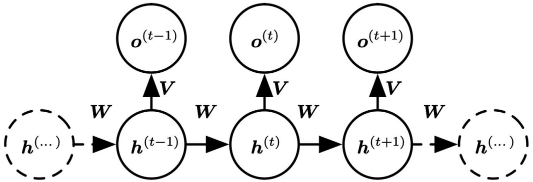
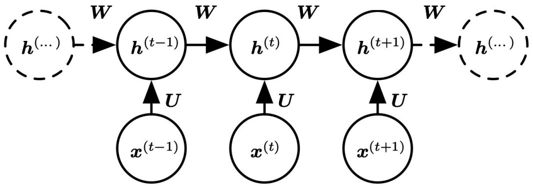


Image description (our application!)



Text classification

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

Recurrent layer configurations

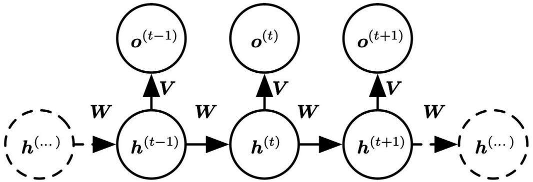
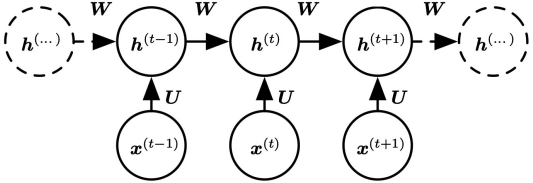
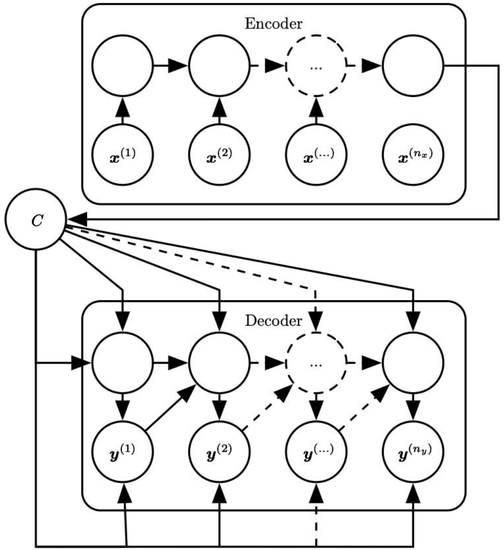


Image description (our application!)



Text classification



Machine translation

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

Image captioning model

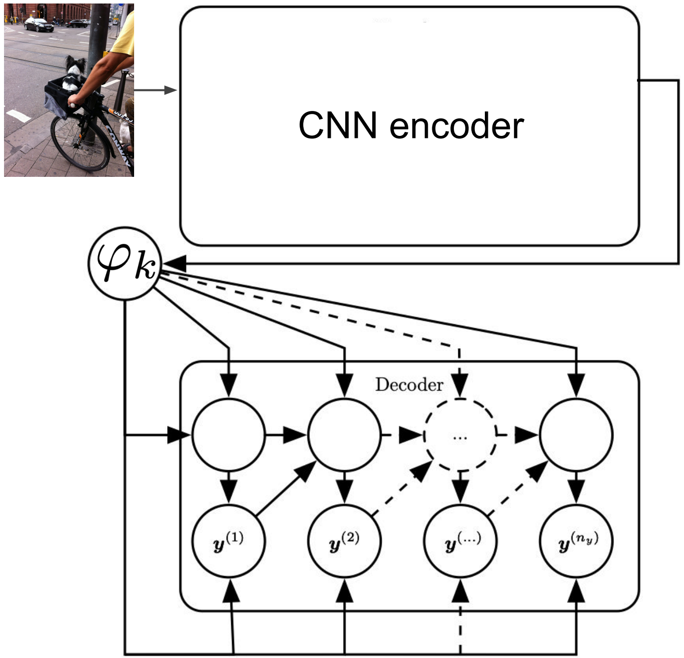


Image captioning model

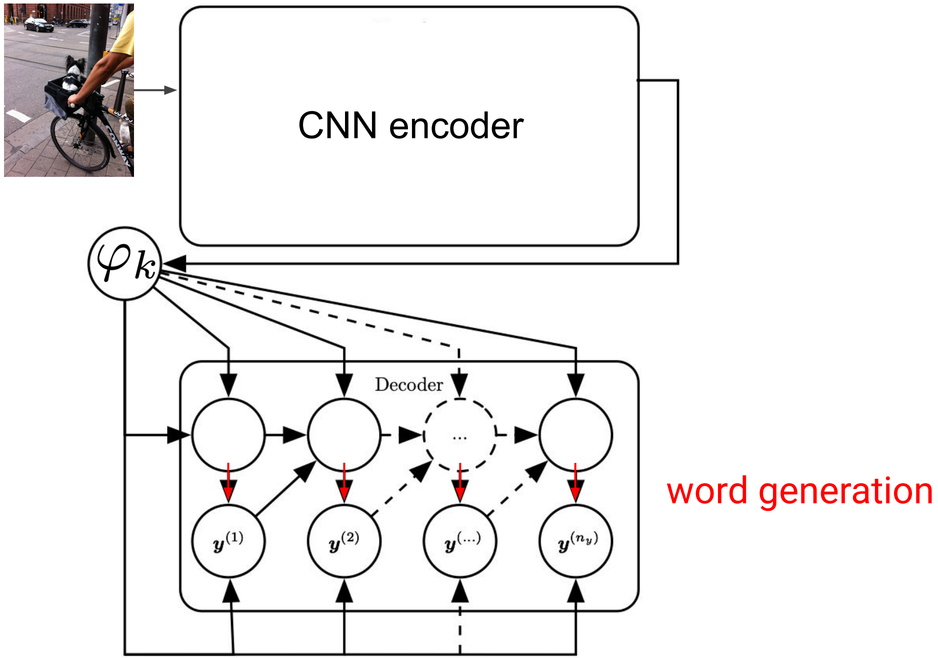
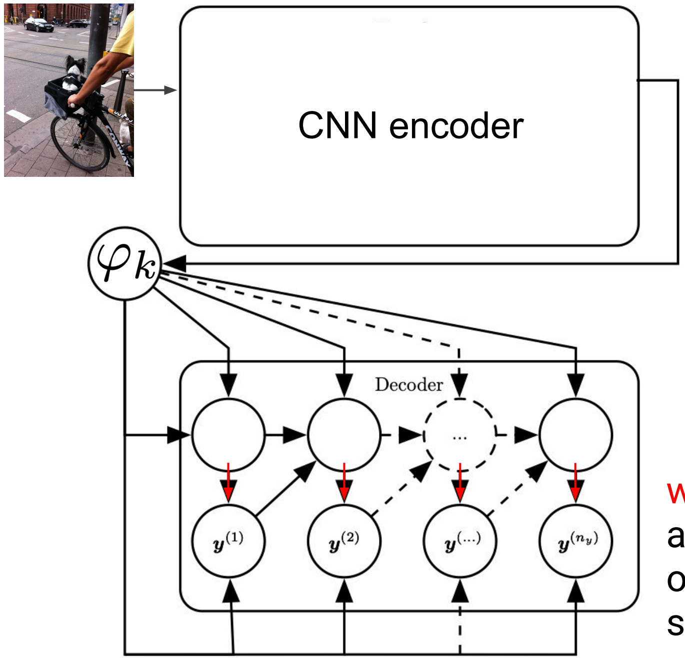
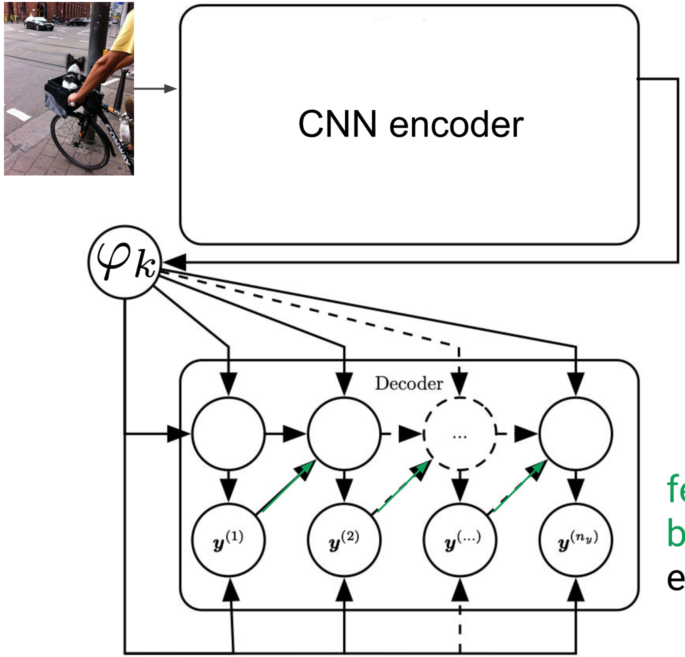


Image captioning model



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

Image captioning model




```python
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

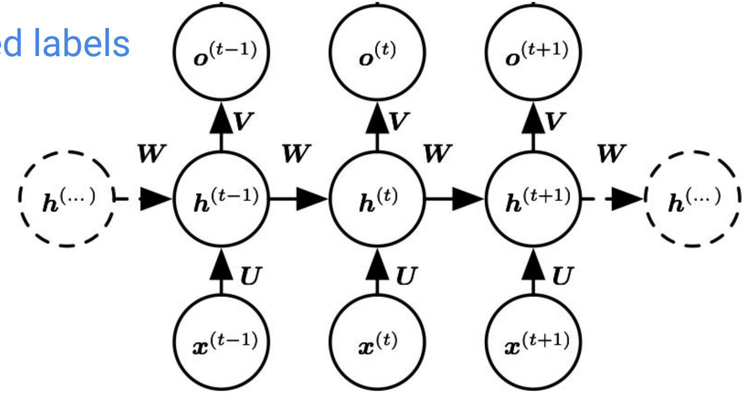


Image from
[“https://www.deeplearningbook.org/slides/10_rnn.pdf”](https://www.deeplearningbook.org/slides/10_rnn.pdf)

Model training

Ground-truth labels

Loss

Predicted labels

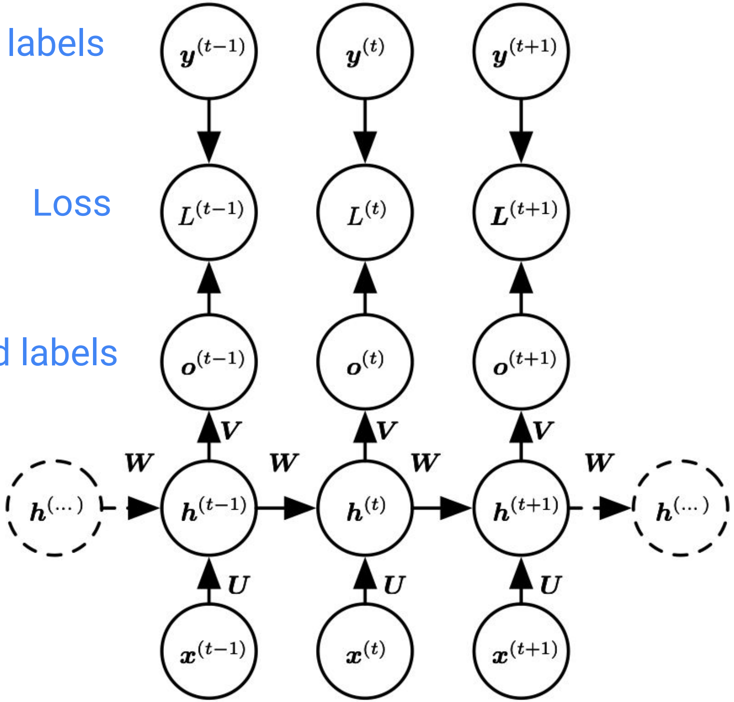


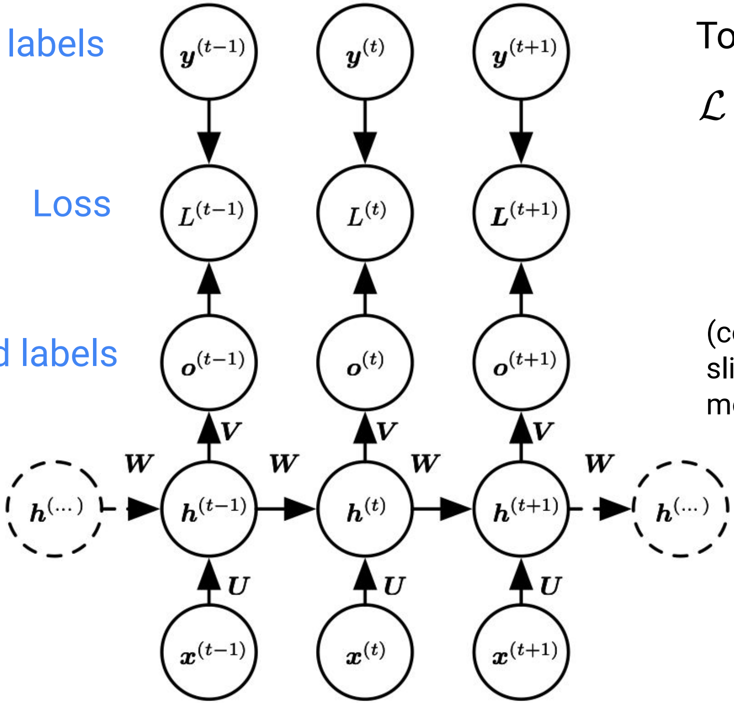
Image from
["https://www.deeplearningbook.org/slides/10_rnn.pdf"](https://www.deeplearningbook.org/slides/10_rnn.pdf)

Model training

Ground-truth labels

Loss

Predicted labels



Total loss:

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

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

(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”](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!