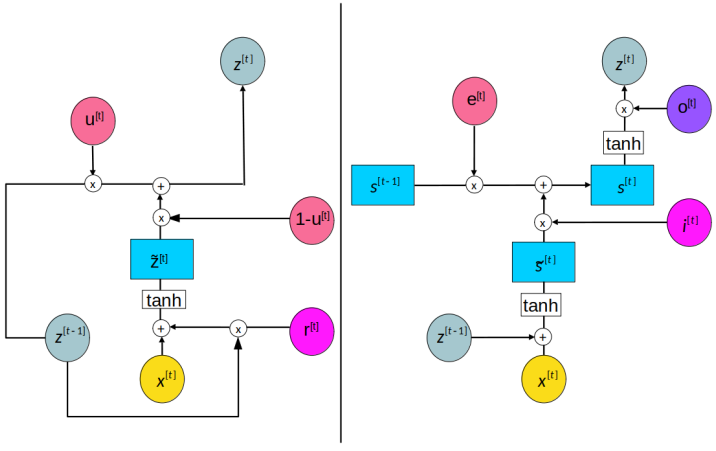
Deep Learning

Applications of RNNs

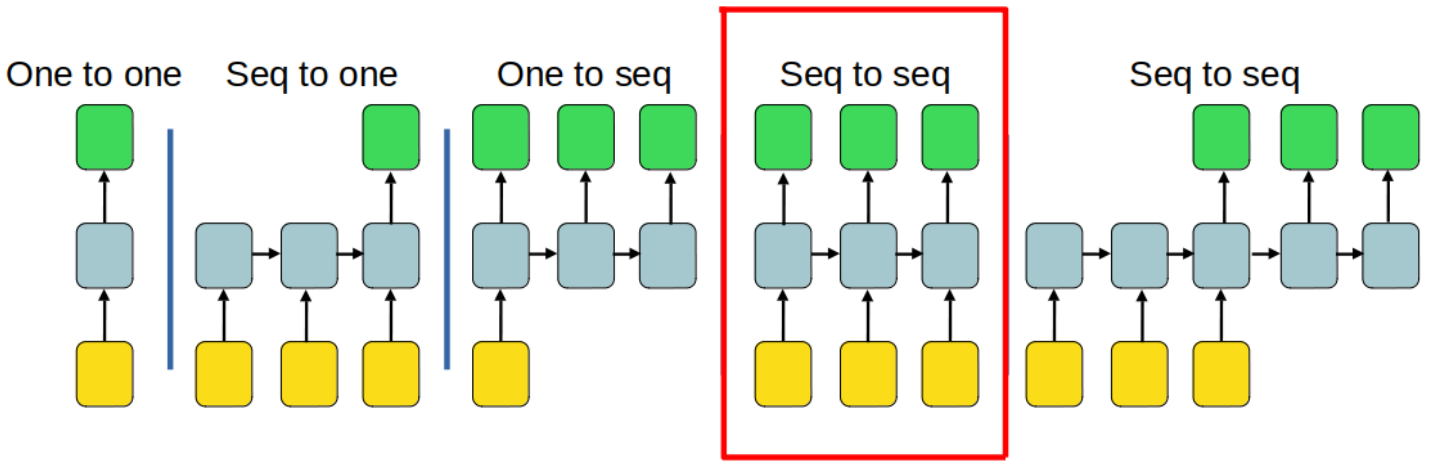


Learning goals

- LSTM
- GRU
- Bidirectional RNNs

Application: Language modelling

Seq-to-Seq (Type I)



RNNS - LANGUAGE MODELLING

- In an earlier example, we built a 'sequence-to-one' RNN model to perform 'sentiment analysis'.
- Another common task in Natural Language Processing (NLP) is **'language modelling'**.
- Input: word/character, encoded as a one-hot vector.
- Output: probability distribution over words/characters given previous words

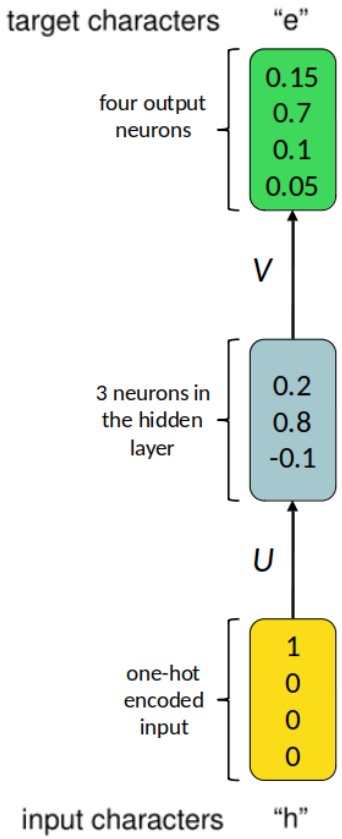
$$\mathbb{P}(y^{[1]}, \ldots, y^{[\tau]}) = \prod_{i=1}^{\tau} \mathbb{P}(y^{[i]} | y^{[1]}, \ldots, y^{[i-1]})$$

$\rightarrow$ given a sequence of previous characters, ask the RNN to model the probability distribution of the next character in the sequence!

RNNS - LANGUAGE MODELLING

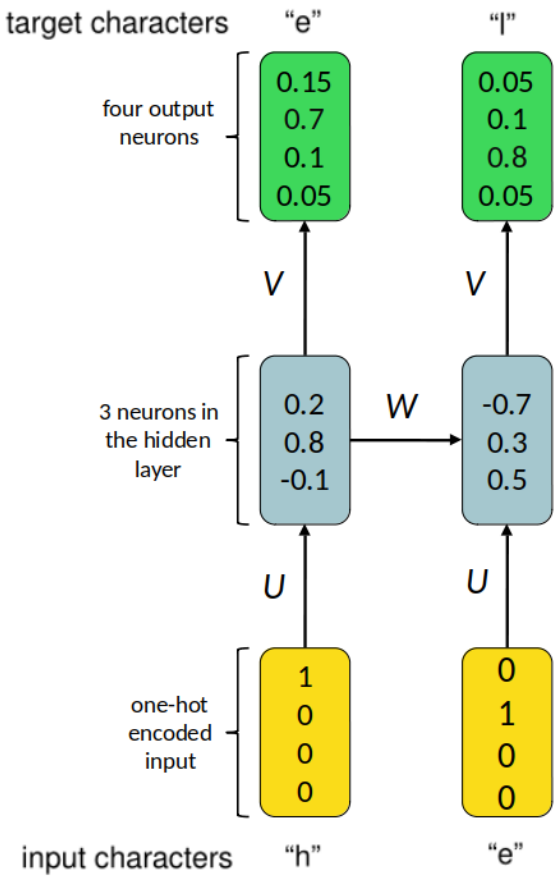
- In this example, we will feed the characters in the word "hello" one at a time to a 'seq-to-seq' RNN.
- For the sake of the visualization, the characters "h", "e", "l" and "o" are one-hot coded as a vectors of length 4 and the output layer only has 4 neurons, one for each character (we ignore the <eos> token).
- At each time step, the RNN has to output a probability distribution (softmax) over the 4 possible characters that might follow the current input.
- Naturally, if the RNN has been trained on words in the English language:
  - The probability of "e" should be likely, given the context of "h".
  - "l" should be likely in the context of "he".
  - "l" should **also** be likely, given the context of "hel".
  - and, finally, "o" should be likely, given the context of "hell".

RNNS - LANGUAGE MODELLING



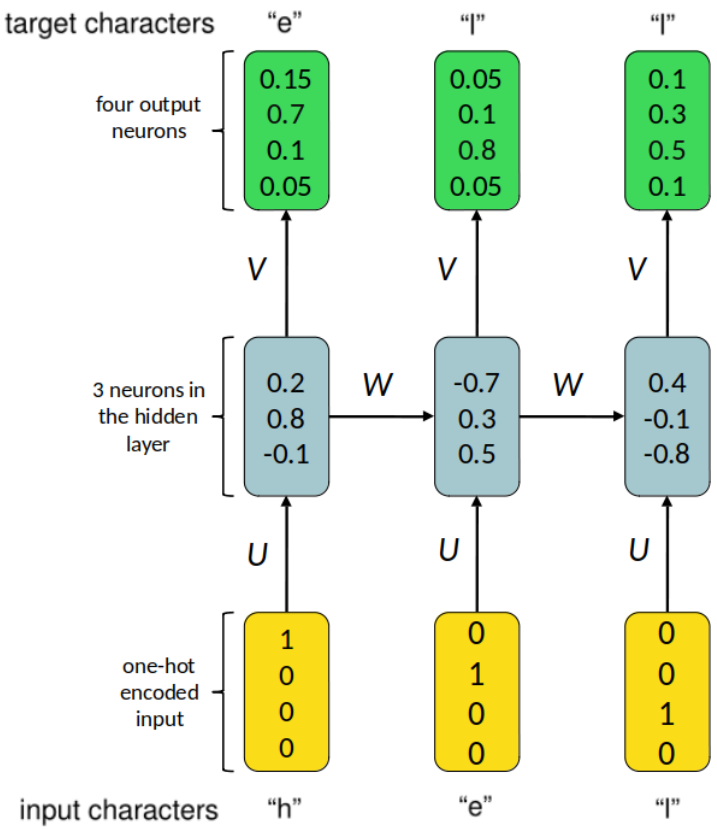
The probability of “e” should be high, given the context of “h”.

RNNS - LANGUAGE MODELLING



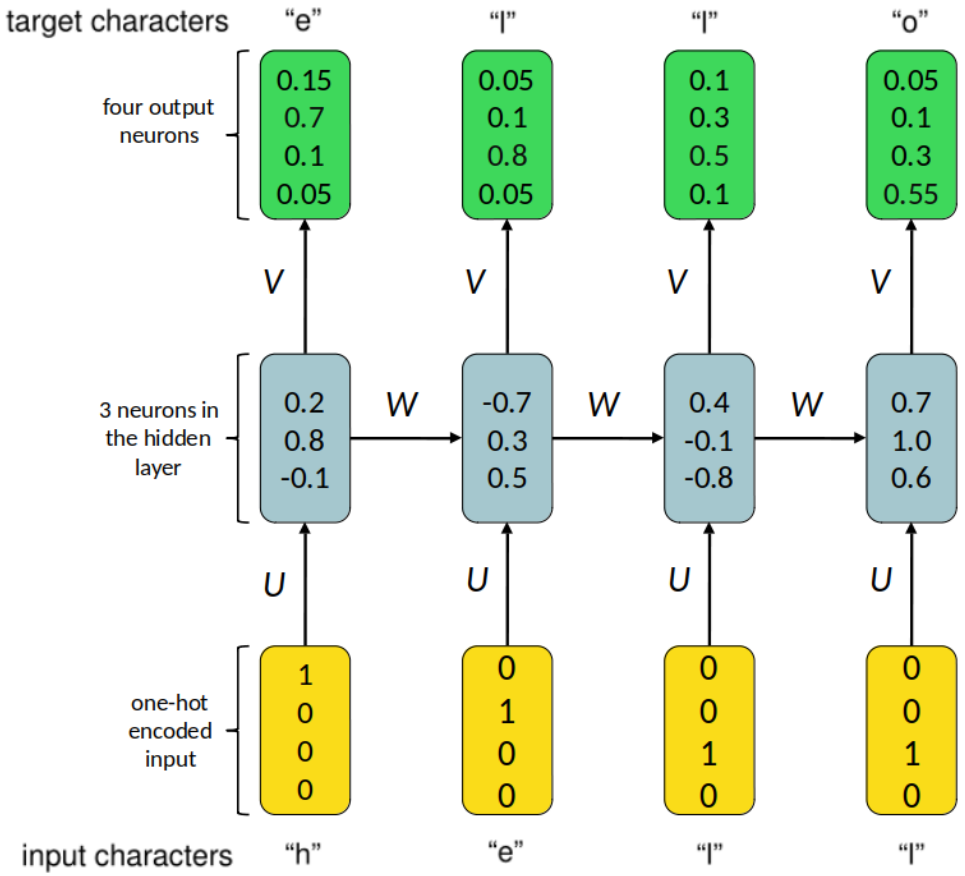
The probability of "l" should be high, given in the context of "he".

RNNS - LANGUAGE MODELLING



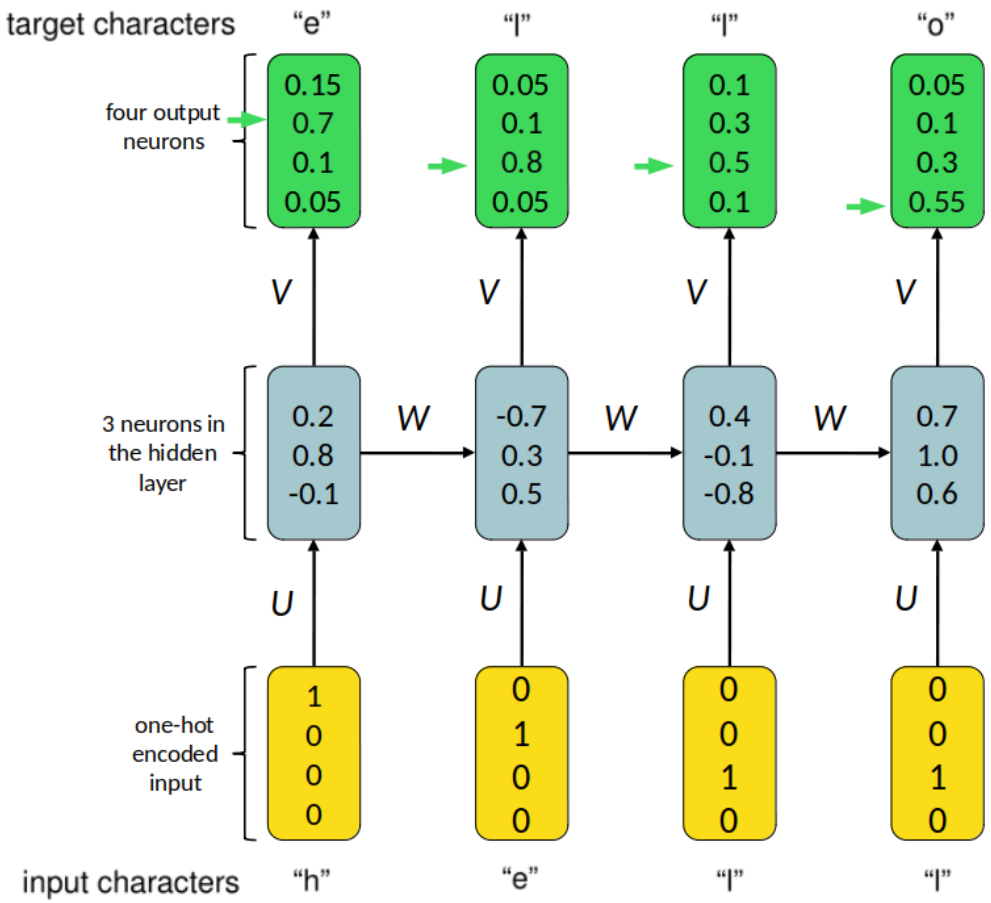
The probability of “l” should **also** be high, given in the context of “hel”.

RNNS - LANGUAGE MODELLING



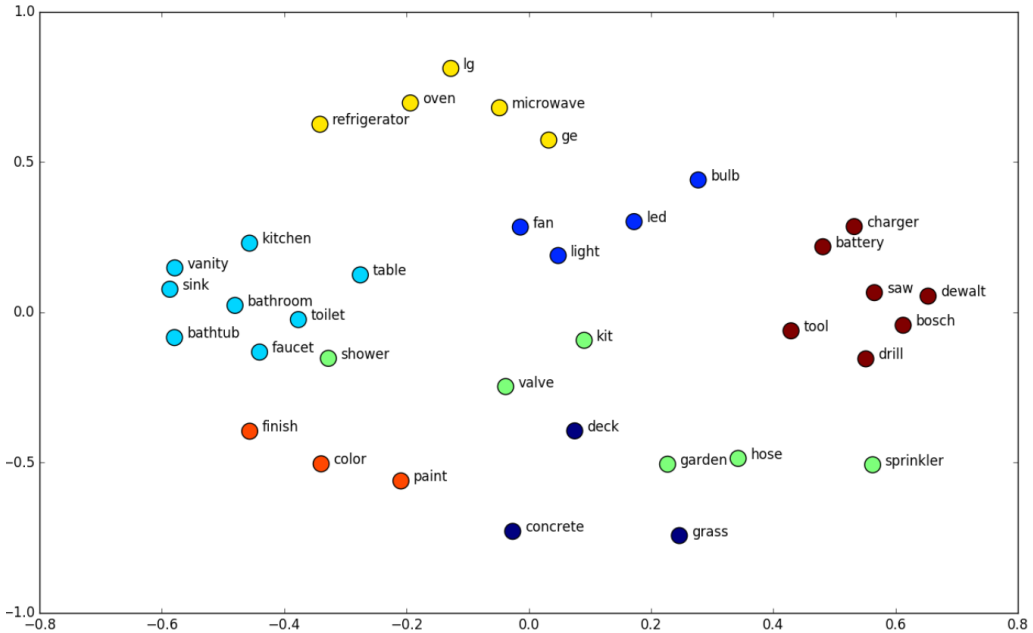
The probability of "o" should be high, given the context of "hell".

RNNS - LANGUAGE MODELLING



During training, our goal would be to increase the confidence for the correct letters (indicated by the green arrows) and decrease the confidence of all others.

WORD EMBEDDINGS



Source: Kaggle

Figure: Two-dimensional embedding space. Typically, the embedding space is much higher dimensional.

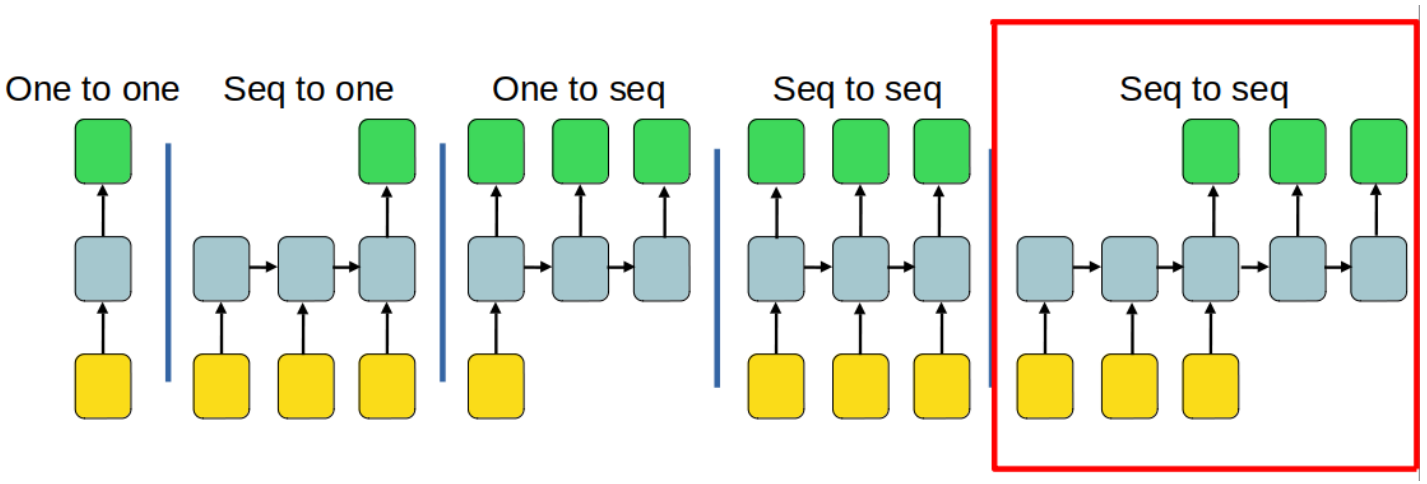
- Instead of one-hot representations of words it is standard practice to encode each word as a dense (as opposed to sparse) vector of fixed size that captures its underlying semantic content.
- Similar words are embedded close to each other in a lower-dimensional embedding space.

WORD EMBEDDINGS

- The dimensionality of these embeddings is typically much smaller than the number of words in the dictionary.
- Using them gives you a "warm start" for any NLP task. It is an easy way to incorporate prior knowledge into your model and a rudimentary form of **transfer learning**.
- Two very popular approaches to learn word embeddings are **word2vec** by Google and **GloVe** by Facebook. These embeddings are typically 100 to 1000 dimensional.
- Even though these embeddings capture the meaning of each word to an extent, they do not capture the *semantics* of the word in a given context because each word has a static precomputed representation. For example, depending on the context, the word "bank" might refer to a financial institution or to a river bank.

Encoder-Decoder Architectures

Seq-to-Seq (Type II)



ENCODER-DECODER NETWORK

- For many interesting applications such as question answering, dialogue systems, or machine translation, the network needs to map an input sequence to an output sequence of different length.
- This is what an encoder-decoder (also called sequence-to-sequence architecture) enables us to do!

ENCODER-DECODER NETWORK

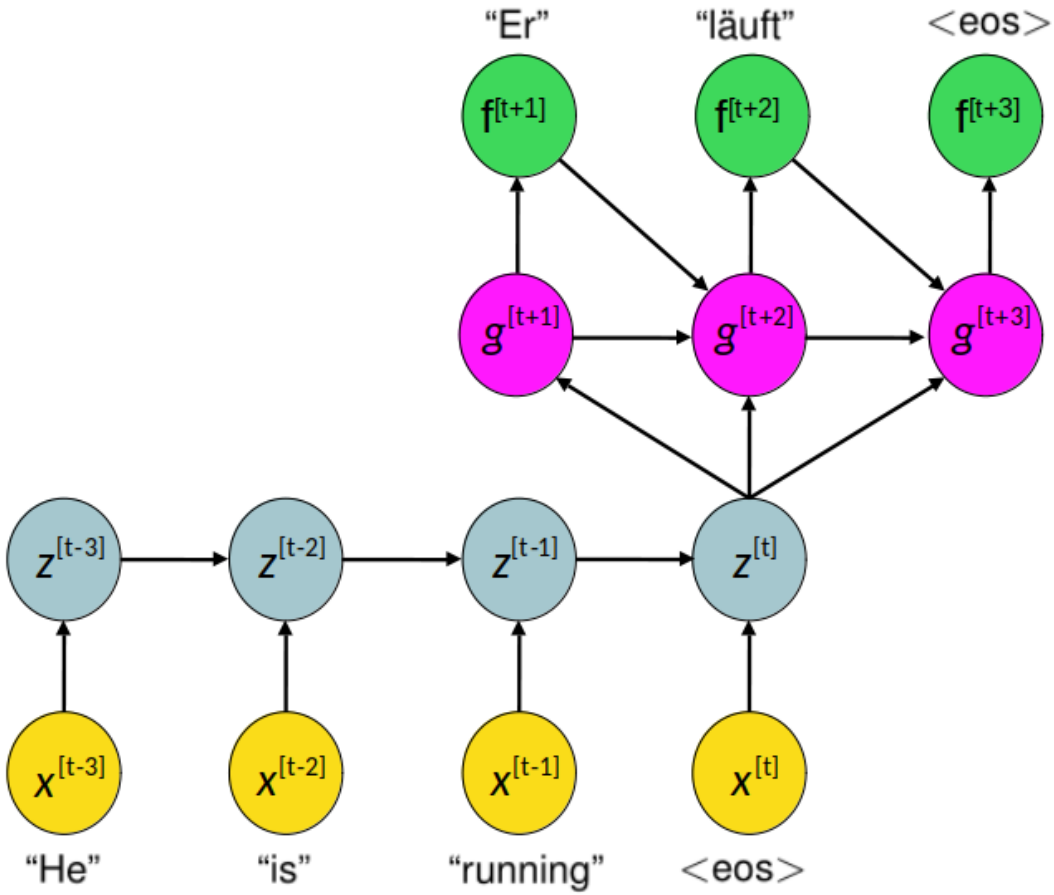


Figure: In the first part of the network, information from the input is encoded in the context vector, here the final hidden state, which is then passed on to every hidden state of the decoder, which produces the target sequence.

ENCODER-DECODER NETWORK

- An input/encoder-RNN processes the input sequence of length n_x and computes a fixed-length context vector C, usually the final hidden state or simple function of the hidden states.
- One time step after the other information from the input sequence is processed, added to the hidden state and passed forward in time through the recurrent connections between hidden states in the encoder.
- The context vector summarizes important information from the input sequence, e.g. the intent of a question in a question answering task or the meaning of a text in the case of machine translation.
- The decoder RNN uses this information to predict the output, a sequence of length n_y, which could vary from n_x.

ENCODER-DECODER NETWORK

- In machine translation, the decoder is a language model with recurrent connections between the output at one time step and the hidden state at the next time step as well as recurrent connections between the hidden states:

$$\mathbb{P}(y^{[1]}, \ldots, y^{[y_n]} | \mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[x_n]}) = \prod_{t=1}^{n_y} p(y^{[t]} | C; y^{[1]}, \ldots, y^{[t-1]})$$

  with C being the context-vector.
- This architecture is now jointly trained to minimize the translation error given a source sentence.
- Each conditional probability is then

$$p(y^{[t]} | y^{[1]}, \ldots, y^{[t-1]}; C) = f(y^{[t-1]}, g^{[t]}, C)$$

  where f is a non-linear function, e.g. the tanh and $g^{[t]}$ is the hidden state of the decoder network.

Attention

ATTENTION

- In a classical decoder-encoder RNN all information about the input sequence must be incorporated into the final hidden state, which is then passed as an input to the decoder network.
- With a long input sequence this fixed-sized context vector is unlikely to capture all relevant information about the past.
- Each hidden state contains mostly information from recent inputs.
- Key idea: Allow the decoder to access all the hidden states of the encoder (instead of just the final one) so that it can dynamically decide which ones are relevant at each time-step of the decoding process.
- This means the decoder can choose to "focus" on different hidden states (of the encoder) at different time-steps of the decoding process similar to how the human eye can focus on different regions of the visual field.
- This is known as an **attention mechanism**.

ATTENTION

- The attention mechanism is implemented by an additional component in the decoder.
- For example, this can be a simple single-hidden layer feed-forward neural network which is trained along with the RNN.
- At any given time-step i of the decoding process, the network computes the relevance of encoder state $\boldsymbol{z}^{[j]}$ as:

$$rel(\boldsymbol{z}^{[j]})^{[i]} = \boldsymbol{v}_a^\top \tanh(\boldsymbol{W}_a[\boldsymbol{g}^{[i-1]}; \boldsymbol{z}^{[j]}])$$

  where $\boldsymbol{v}_a$ and $\boldsymbol{W}_a$ are the parameters of the feed-forward network, $\boldsymbol{g}^{[i-1]}$ is the decoder state from the previous time-step and ';' indicates concatenation.
- The relevance scores (for all the encoder hidden states) are then normalized which gives the *attention weights* $(\alpha^{[j]})^{[i]}$:

$$(\alpha^{[j]})^{[i]} = \frac{\exp(rel(\boldsymbol{z}^{[j]})^{[i]})}{\sum_{j'} \exp(rel(\boldsymbol{z}^{[j']})^{[i]})}$$

ATTENTION

- The attention mechanism allows the decoder network to focus on different parts of the input sequence by adding connections from all hidden states of the encoder to each hidden state of the decoder.

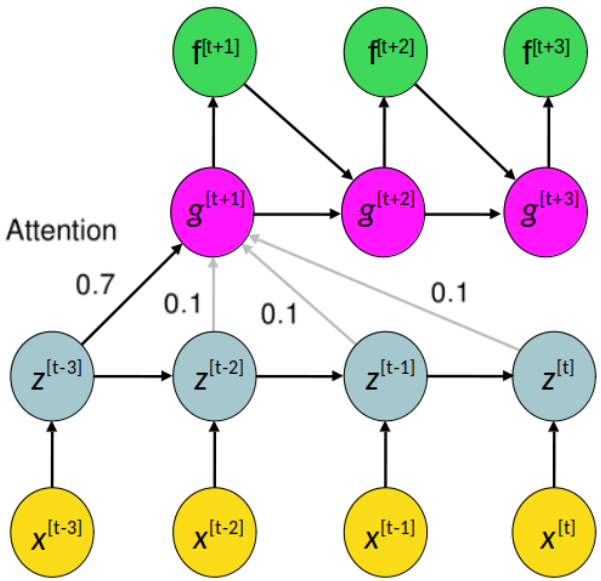


Figure: Attention at $i = t + 1$

ATTENTION

- At each time step i, a set of weights $(\alpha^{[j]})^{[i]}$ is computed which determine how to combine the hidden states of the encoder into a context vector $c^{[i]} = \sum_{j=1}^{n_x} (\alpha^{[j]})^{[i]} \boldsymbol{z}^{[j]}$, which holds the necessary information to predict the correct output.

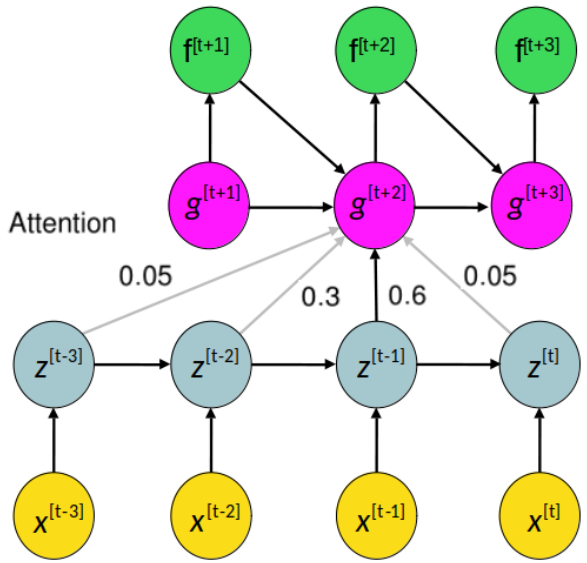
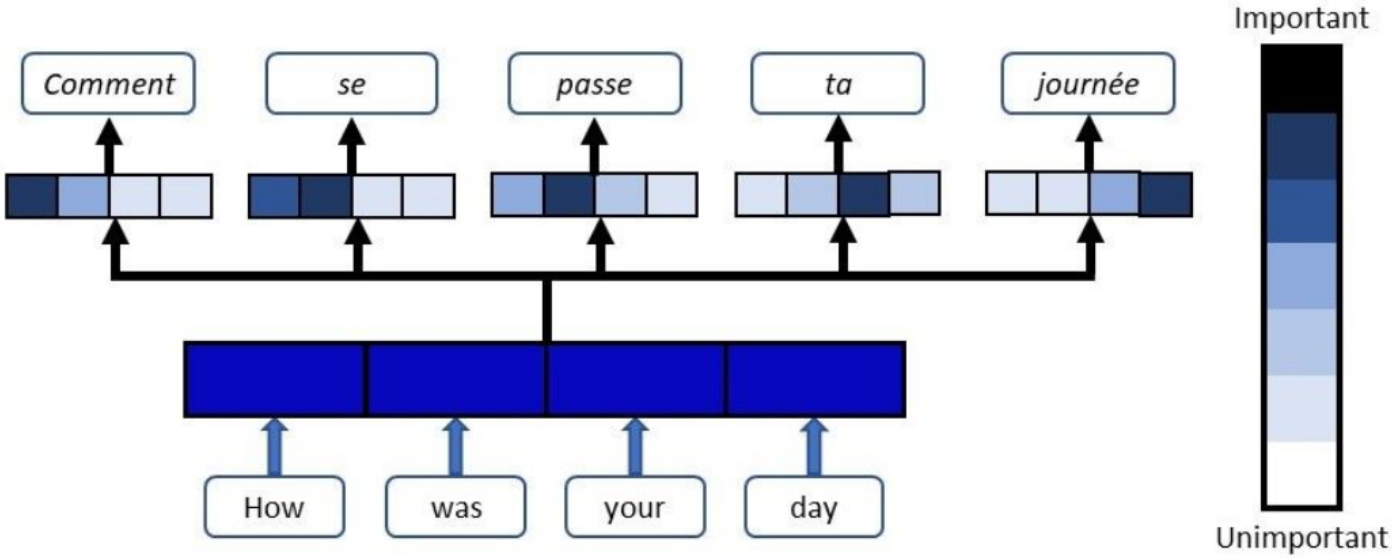


Figure: Attention at $i = t + 2$

ATTENTION



Credit: Gabriel Loye

Figure: An illustration of a machine translation task using an encoder-decoder model with an attention mechanism. The attention weights at each time-step of the decoding/translation process indicate which parts of the input sequence are most relevant. (There are 4 attention weights because there are 4 encoder states.)

TRANSFORMERS

- Advanced RNNs have similar limitations as vanilla RNN network.
  - Lack of parallelization, RNNs process the input data sequentially, as the output of a step depends on the output of the previous step.
  - Difficulties in learning long term dependency, Although GRU or LSTM perform superior compared to vanilla RNN, even they struggle to remember the context from a word which stands much earlier in long sequences.
- These challenges are successfully tackled by transformer networks.

TRANSFORMERS

- Transformers are solely based on attention (no RNN or CNN).
- In fact, the paper which coined the term *transformer* is called *Attention is all you need*.
- They are the state-of-the-art networks in natural language processing (NLP) tasks since 2017.
- Transformer architectures like BERT (Bidirectional Encoder Representations from Transformers, 2018) and GPT-3 (Generative Pre-trained Transformer-3, 2020) are pre-trained in a large text and can be fine-tuned (quickly adapt) to specific language tasks.

More application examples

SOME MORE SOPHISTICATED APPLICATIONS

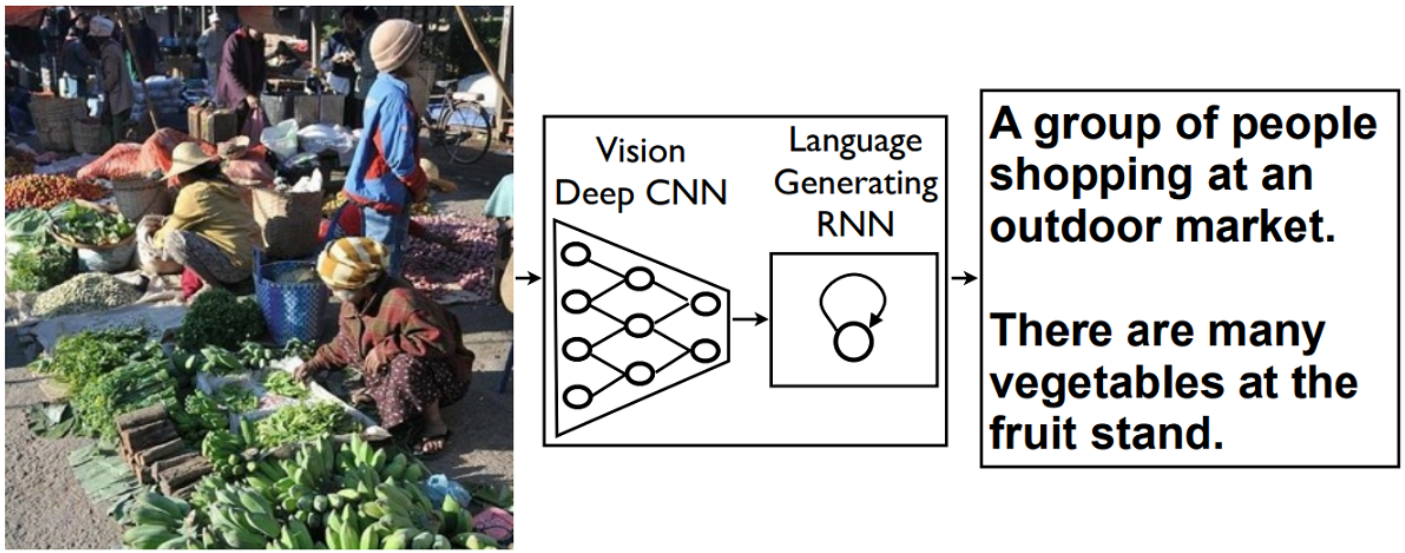


Figure: Show and Tell: A Neural Image Caption Generator (Oriol Vinyals et al. 2014). A language generating RNN tries to describe in brief the content of different images.

SOME MORE SOPHISTICATED APPLICATIONS

Figure: Show and Tell: A Neural Image Caption Generator (Oriol Vinyals et al. 2014). A language generating RNN tries to describe in brief the content of different images.

SOME MORE SOPHISTICATED APPLICATIONS

A woman is throwing a frisbee in a park.

A stop sign is on a road with a mountain in the background.

Figure: Attention for image captioning: the attention mechanism tells the network roughly which pixels to pay attention to when writing the text (Kelvin Xu al. 2015)

SOME MORE SOPHISTICATED APPLICATIONS

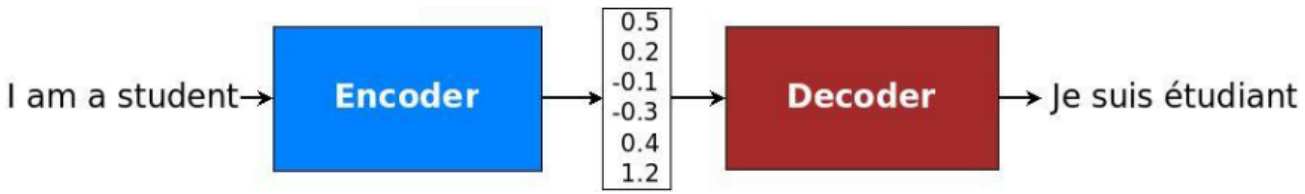


Figure: Neural Machine Translation (seq2seq): Sequence to Sequence Learning with Neural Networks (Ilya Sutskever et al. 2014). As we saw earlier, an encoder converts a source sentence into a “meaning” vector which is passed through a decoder to produce a translation.

SOME MORE SOPHISTICATED APPLICATIONS

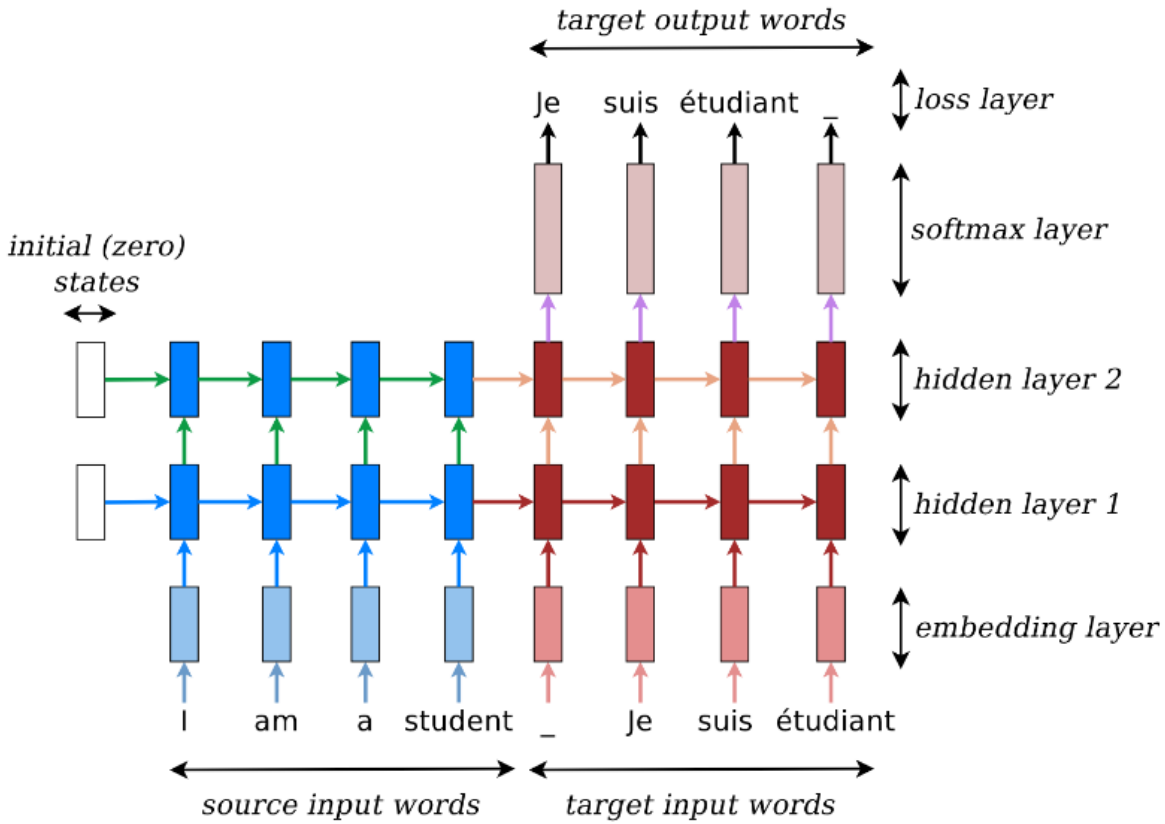


Figure: Neural Machine Translation (seq2seq): Sequence to Sequence Learning with Neural Networks (Ilya Sutskever et al. 2014). As we saw earlier, an encoder converts a source sentence into a “meaning” vector which is passed through a decoder to produce a translation.

SOME MORE SOPHISTICATED APPLICATIONS

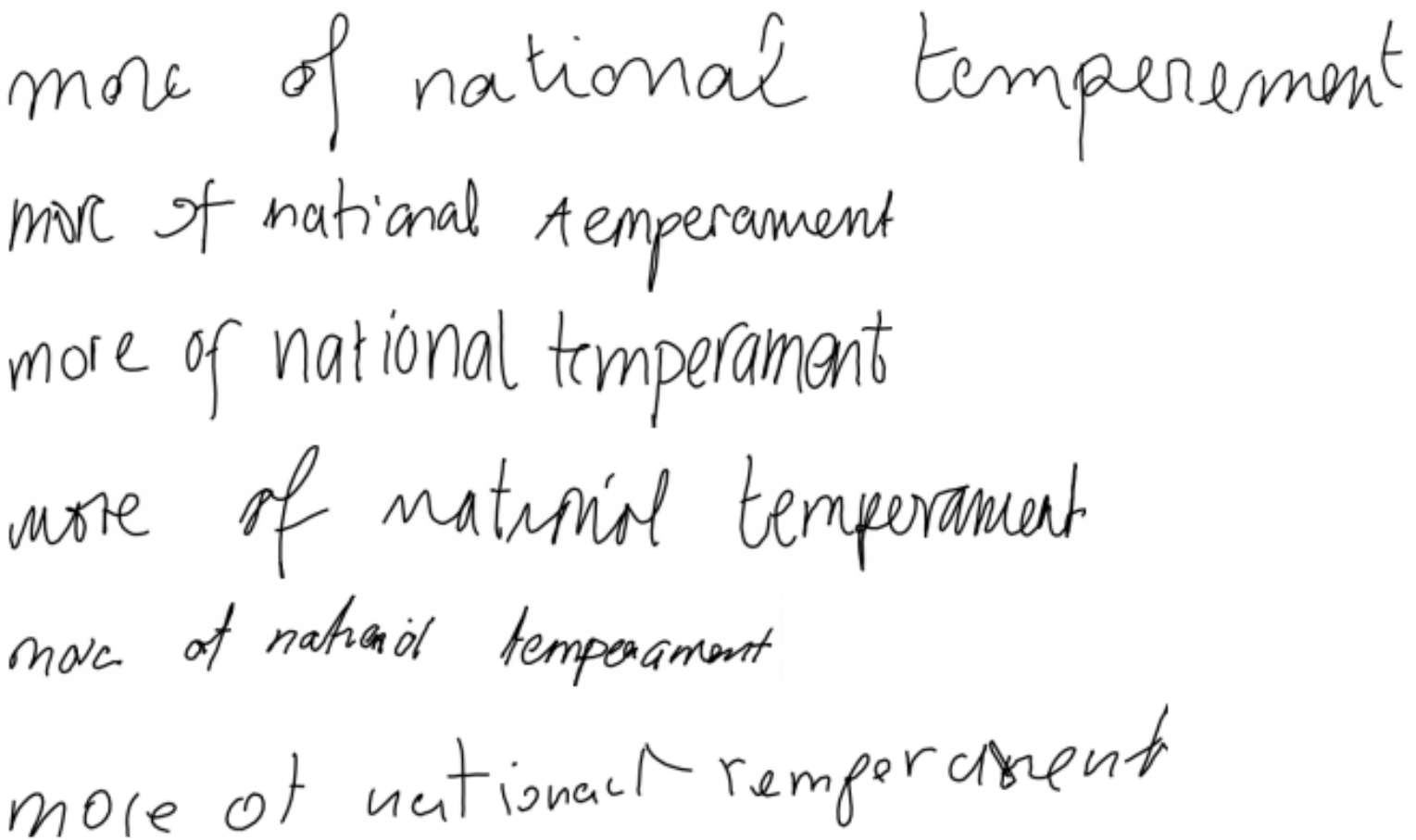


Figure: Generating Sequences With Recurrent Neural Networks (Alex Graves, 2013). Top row are real data, the rest are generated by various RNNs.

SOME MORE SOPHISTICATED APPLICATIONS

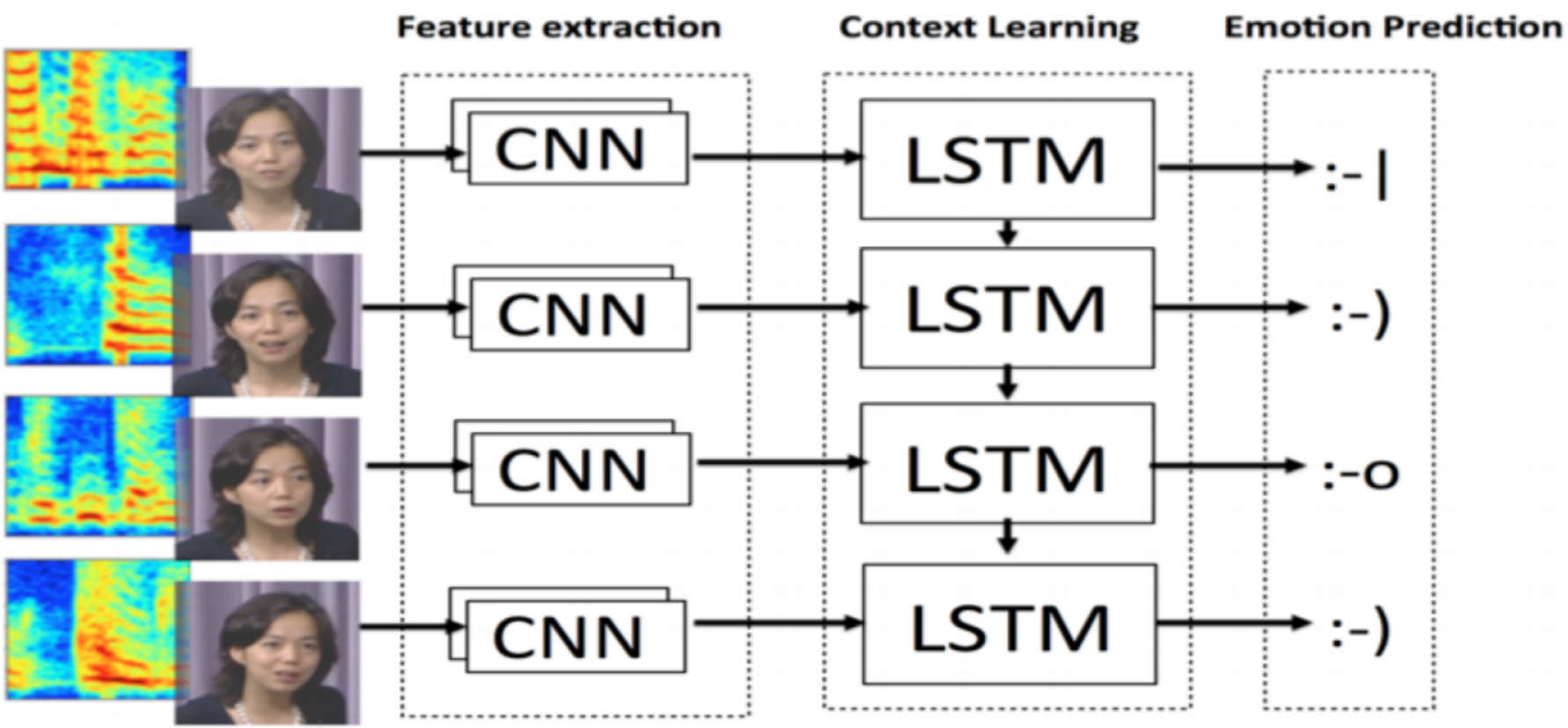


Figure: Convolutional and recurrent nets for detecting emotion from audio data (Namrata Anand & Prateek Verma, 2016). We already had this example in the CNN chapter!

SOME MORE SOPHISTICATED APPLICATIONS

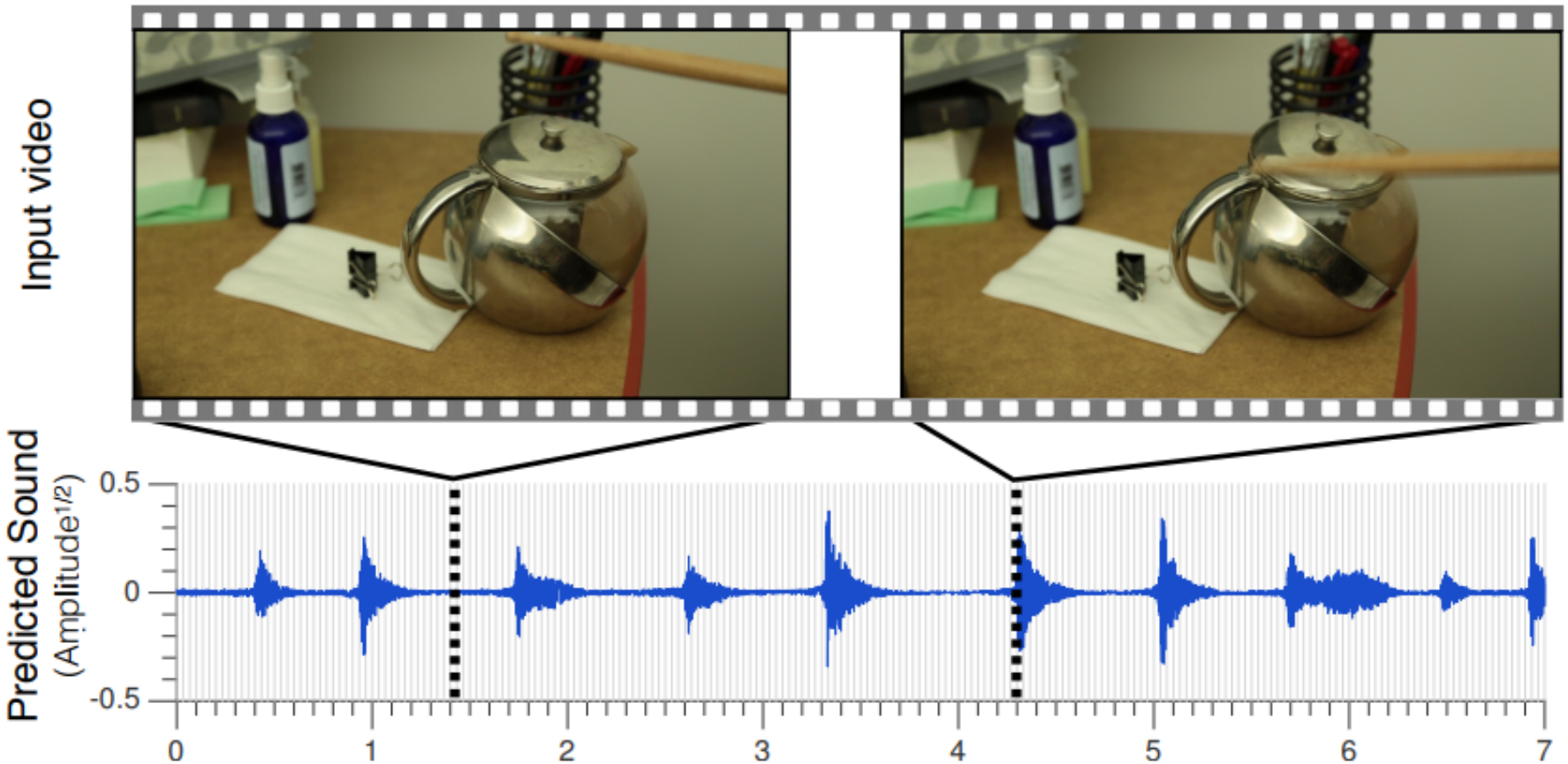


Figure: Visually Indicated Sounds (Andrew Owens et al. 2016). A model to synthesize plausible impact sounds from silent videos. Click here

CNNs or RNNs?

CNNS OR RNNS?

- Historically, RNNs were the default models used in sequence processing tasks.
- However, some families of CNNs (especially those based on Fully Convolutional Networks (FCNs)) *can* be used to process variable-length sequences such as text or time-series data.
- If a CNN doesn't contain any fully-connected layers, the total number of weights in the network is independent of the spatial dimensions of the input because of weight-sharing in the convolutional layers.
- Recent research [Bai et al. , 2018] indicates that such convolutional architectures (which the authors term Temporal Convolutional Networks (TCNs)) can outperform RNNs on a wide range of tasks.
- A major advantage of TCNs is that the entire input sequence can be fed to the network at once (as opposed to sequentially).

CNNS OR RNNS?

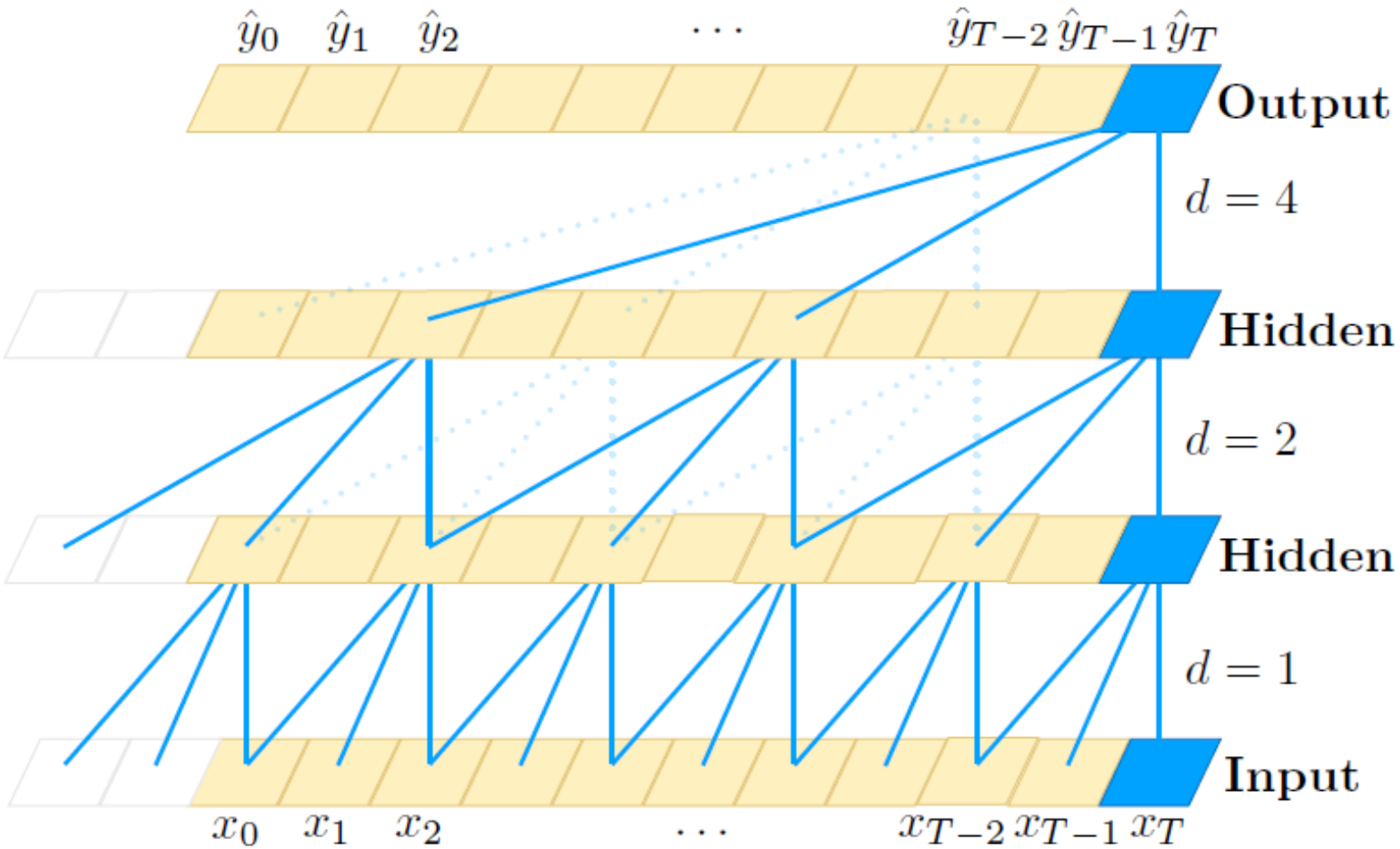


Figure: A TCN (we have already seen this in the CNN lecture!) is simply a variant of the one-dimensional FCN which uses a special type of dilated convolutions called **causal dilated** convolutions.

CNNS OR RNNS?

Sequence Modeling Task	Model Size ($\approx$)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

Figure: Evaluation of TCNs and recurrent architectures on a wide range of sequence modelling tasks. ^{h} means higher is better and $^\ell$ means lower is better. Note: To make the comparisons fair, all models have roughly the same size (for a given task) and the authors used grid search to find good hyperparameters for the recurrent architectures.

SUMMARY

- RNNs are specifically designed to process sequences of varying lengths.
- For that recurrent connections are introduced into the network structure.
- The gradient is calculated by backpropagation through time.
- An LSTM replaces the simple hidden neuron by a complex system consisting of cell state, and forget, input, and output gates.
- An RNN can be used as a language model, which can be improved by word-embeddings.
- Different advanced types of RNNs exist, like Encoder-Decoder architectures and bidirectional RNNs.[1]

1. A bidirectional RNN processes the input sequence in both directions (front-to-back and back-to-front).

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Oriol Vinyals, Alexander Toshev, Samy Bengio and Dumitru Erhan (2014)

Show and Tell: A Neural Image Caption Generator

https://arxiv.org/abs/1411.4555

Alex Graves (2013)

Generating Sequences With Recurrent Neural Networks

https://arxiv.org/abs/1308.0850

Namrata Anand and Prateek Verma (2016)

Convolutional and recurrent nets for detecting emotion from audio data

http://cs231n.stanford.edu/reports/2015/pdfs/Cs_231n_paper.pdf

Gabriel Loye (2019)

Attention Mechanism

https://blog.floydhub.com/attention-mechanism/

REFERENCES

Andrew Owens, Phillip Isola, Josh H. McDermott, Antonio Torralba, Edward H. Adelson and William T. Freeman (2015)

Visually Indicated Sounds

https://arxiv.org/abs/1512.08512

Andrej Karpathy (2015)

The Unreasonable Effectiveness of Recurrent Neural Networks

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel and Yoshua Bengio (2015)

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

https://arxiv.org/abs/1502.03044

Shaojie Bai, J. Zico Kolter, Vladlen Koltun (2018)

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

https://arxiv.org/abs/1803.01271

REFERENCES

Lilian Weng (2018)

Attention? Attention!

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html