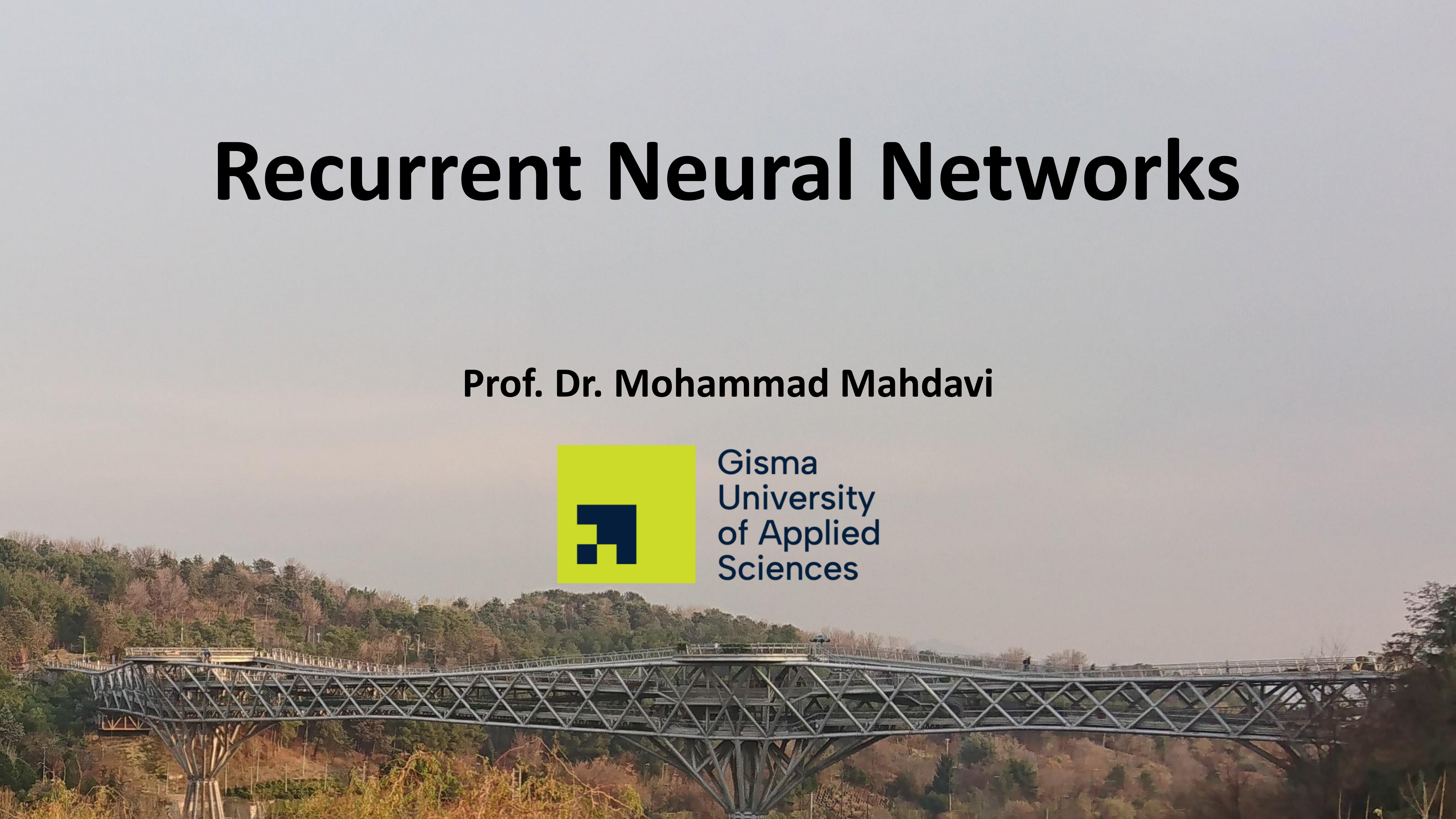


Recurrent Neural Networks

Prof. Dr. Mohammad Mahdavi

Gisma University of Applied Sciences

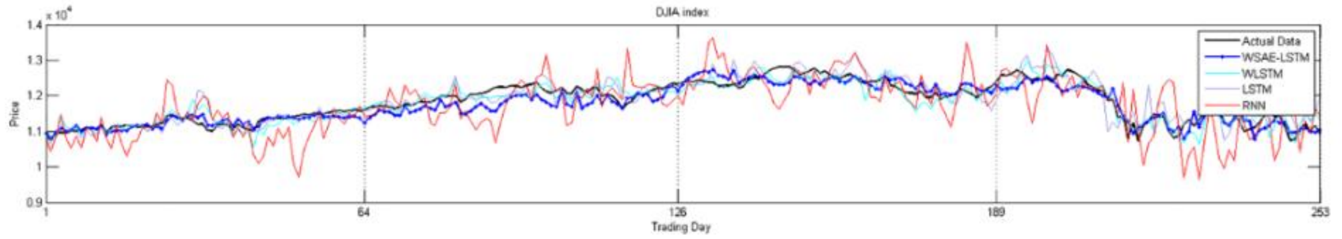
Gisma University of Applied Sciences

Sequence Data

[Burkov 2019]

https://deeplearningmath.org/sequence-models.html

- Sequence is one the most frequently observed types of structured data
  - Sequences of words and sentences
  - Sequences our genes
  - Music and videos are sequential
  - Time series



- A sequence is a matrix each row of which is a feature vector, and the order of rows matters
  - The key point for sequence models is that
    - The data we are processing are not anymore independently and identically distributed (i.i.d.) samples
    - The data carry some dependency due to the sequential order of the data

Jan 1	5˚	…	11%
Jan 2	7˚	…	6%
Jan 3	6˚	…	9%
Jan 4	4˚	…	2%

Input Sequence Matrix

- Each training example is a matrix
  - In which each row is a feature vector
- $\boldsymbol{X} = [\boldsymbol{x}^1, \dots, \boldsymbol{x}^t, \dots \boldsymbol{x}^{length_{\boldsymbol{X}}}]$
  - Let us consider this matrix as a sequence of vectors
  - $length_{\boldsymbol{X}}$ is the length of the input sequence
- Suppose our input example $\boldsymbol{X}$ is a text sentence
  - Then each feature vector x^t represents a word in the sentence at position t

I	0.1	…	0.34
go	0.83	…	0.61
to	0.44	…	0.22
school	0.23	…	0.5

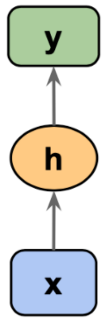
Gisma University of Applied Sciences

Sequence Modeling Tasks

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

https://deeplearningmath.org/sequence-models.html

- One-to-one is the vanilla mode of processing without any sequence
  - From fixed-sized input to fixed-sized output
  - E.g., image classification
  - Each rectangle is a vector and arrows represent functions
  - E.g., matrix multiply
- Sequence modeling tasks
  - Many-to-one
  - One-to-many
  - Many-to-many (synced sequence input and output)
  - Many-to-many (sequence input and sequence output)



one-to-one

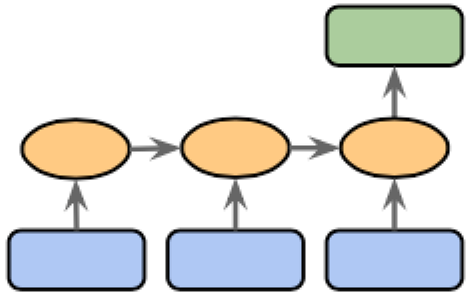
Gisma University of Applied Sciences

Many-to-One Sequence Modeling

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

https://deeplearningmath.org/sequence-models.html

- In many-to-one sequence modeling, the input is a sequence of elements, but the output is a single value or label
  - It is also known as sequence classification
  - E.g., text classification
  - E.g., action prediction
    - Which takes a sequence of video frames and outputs an action class


many-to-one

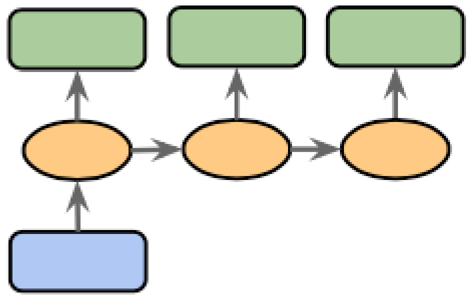
Gisma University of Applied Sciences

One-to-Many Sequence Modeling

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

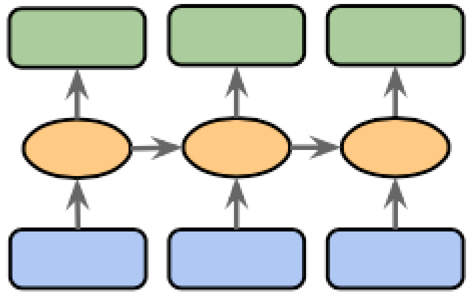
https://deeplearningmath.org/sequence-models.html

- In one-to-many sequence modeling tasks, the input is a single data element, but the output is a sequence of elements
  - E.g., image captioning
    - Which takes an image and outputs a sequence of words


one-to-many

Many-to-Many (Synced) Sequence Modeling

- In many-to-many (synced sequence input and output), the model processes a sequence of inputs and produces a sequence of outputs
  - With each input element corresponding directly to an output element in a synchronized fashion
  - It is also known as sequence labeling
  - E.g., video classification on frame level
    - Where we wish to label each frame of the video
  - E.g., part-of-speech (POS) tagging
  - E.g., named entity recognition (NER)



many-to-many

Gisma University of Applied Sciences

Many-to-Many (Sequence Input and Sequence Output) Sequence Modeling

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

https://deeplearningmath.org/sequence-models.html

- In many-to-many (sequence input and sequence output), the model processes a sequence of inputs and produces a sequence of outputs
  - Unlike the synchronized version, this variant does not require a one-to-one alignment between input and output elements
  - The input and output sequences can have different lengths
    - And the relationship between input elements and output elements is generally more complex
  - It is also known as sequence-to-sequence (seq2seq) learning
  - E.g., machine translation
  - E.g., text summarization
  - E.g., video captioning
    - Which takes a sequence of video frames and outputs a caption

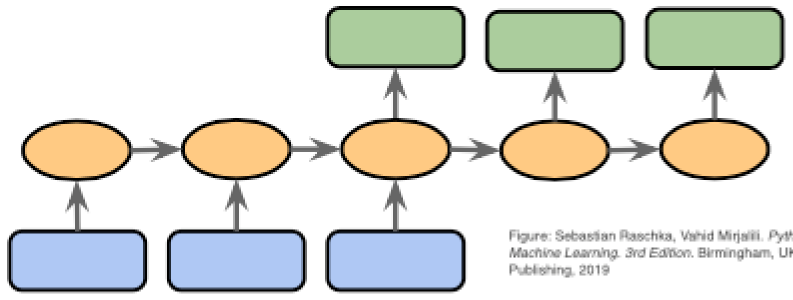


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

many-to-many

Recurrent Neural Network (RNN)

- RNNs are used to label, classify, or generate sequences
  - To label a sequence is to predict a class for each feature vector in a sequence
  - To classify a sequence is to predict a class for the entire sequence
  - To generate a sequence is to output another sequence (of a possibly different length) somehow relevant to the input sequence
- RNNs are often used in text/speech processing
  - Because sentences and texts are naturally sequences of either words/punctuation marks or sequences of characters

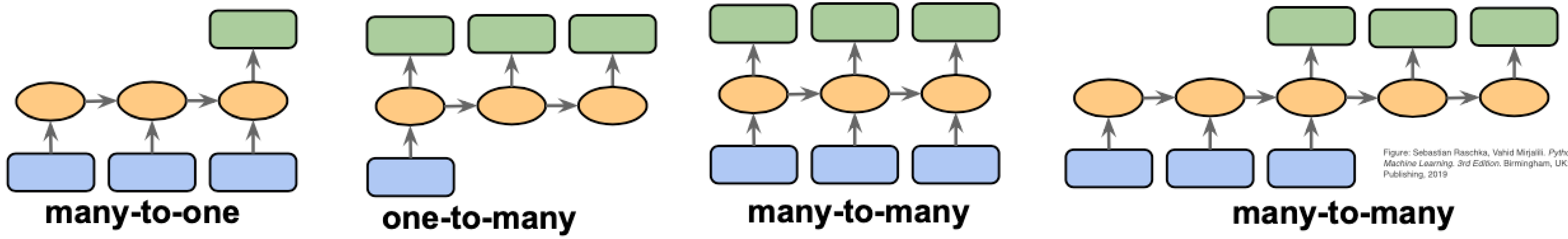


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

Gisma University of Applied Sciences

RNN vs FFNN

https://builtin.com/data-science/recurrent-neural-networks-and-lstm

https://deeplearningmath.org/sequence-models.html

- Feed forward neural network (FFNN)
  - Also called vanilla neural network
  - Moves the information in one direction
    - From the input layer, through the hidden layers, to the output layer
  - Have no memory of the input they receive and are bad at predicting what's coming next
    - Because a FFNN only considers the current input, it has no notion of order in time
    - It simply cannot remember anything about what happened in the past except its training
- Recurrent Neural Network (RNN)
  - Cycles the information through a loop
    - Which allows information to be passed from one step of the network to the next
    - Recurrent neural networks add the immediate past to the present
  - When it makes a decision, it considers
    - The current input
    - What it has learned from the inputs it received previously

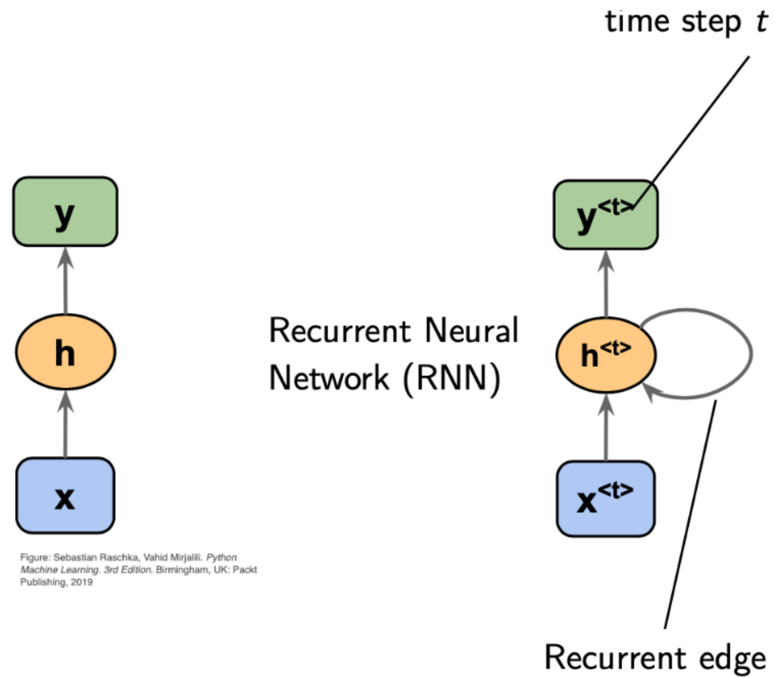


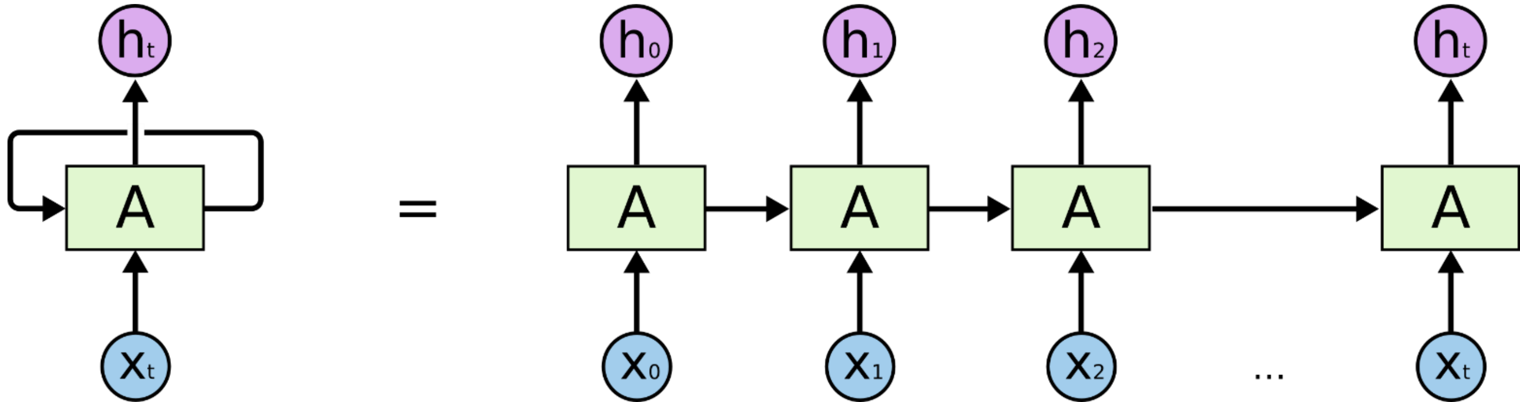
Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

Gisma University of Applied Sciences

RNN Architecture

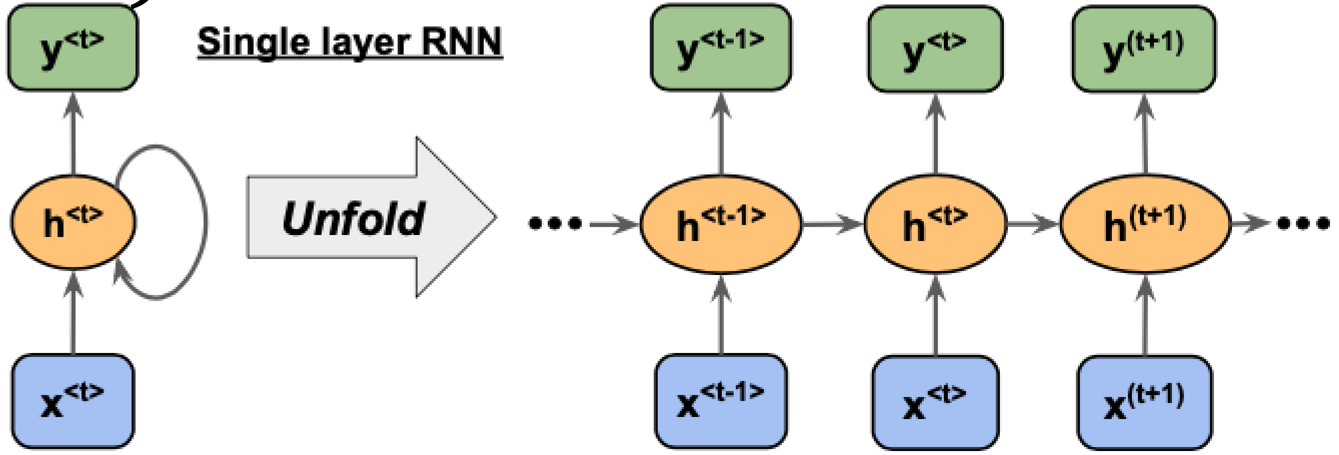
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- RNNs are networks with loops in them, allowing information to persist
  - In the below diagram, a chunk of neural network, A, looks at some input $\boldsymbol{x}_t$ and outputs a value $\boldsymbol{h}_t$
  - A loop allows information to be passed from one step of the network to the next
  - These loops make recurrent neural networks seem kind of mysterious
    - However, if you think a bit more, it turns out that they are not all that different than a normal neural network
- An RNN can be thought of as multiple copies of the same network
  - Each passing a message to a successor



RNN Architecture

- The idea is that each unit u of recurrent layer l has a real-valued state $h_{l,u}$
  - The state can be seen as the memory of the unit
  - In RNN, each unit u in each layer l receives two inputs
    - A vector of states from the previous layer $l-1$
    - The vector of states from this same layer l from the previous time step
- $h^{<t>} = f_W(h^{<t-1>}, x^{<t>})$
  - f_W is a function parameterized by the weight W
  - Every time step t the same function f_W is used
    - With the same set of weight parameters



Gisma University of Applied Sciences

Backpropagation Through Time (BPTT)

[Burkov 2019]

https://builtin.com/data-science/recurrent-neural-networks-and-lstm

https://deeplearningmath.org/sequence-models.html

- BPTT is a special version of backpropagation
  - It is used to train RNN models
- We can view a RNN as a sequence of neural networks
  - That you train one after another with backpropagation
  - Thus, BPTT is basically just a fancy buzz word for doing backpropagation on an unrolled recurrent neural network
  - If you do BPTT, the conceptualization of unrolling is required since the error of a given time step depends on the previous time step
  - Note that BPTT can be computationally expensive when you have a high number of timesteps

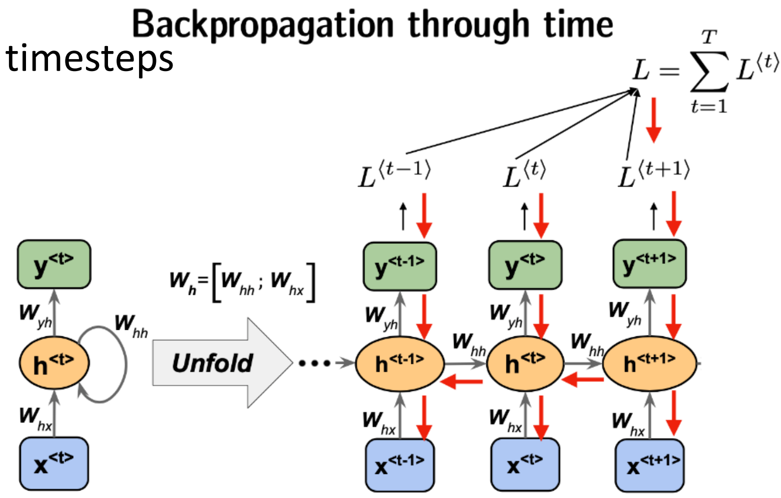


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

Limitations of RNNs

[Burkov 2019]

- Both tanh and softmax suffer from the vanishing gradient problem
  - Even if our RNN has just one or two recurrent layers, because of the sequential nature of the input, backpropagation has to "unfold" the network over time
  - From the point of view of the gradient calculation, in practice this means that the longer is the input sequence, the deeper is the unfolded network
- Another problem RNNs have is that of handling long-term dependencies
  - As the length of the input sequence grows, the feature vectors from the beginning of the sequence tend to be "forgotten"
    - Because the state of each unit, which serves as network's memory, becomes significantly affected by the feature vectors read more recently
  - Therefore, in text or speech processing, the cause-effect link between distant words in a long sentence can be lost

Gisma University of Applied Sciences

Recap: Language Model

https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/

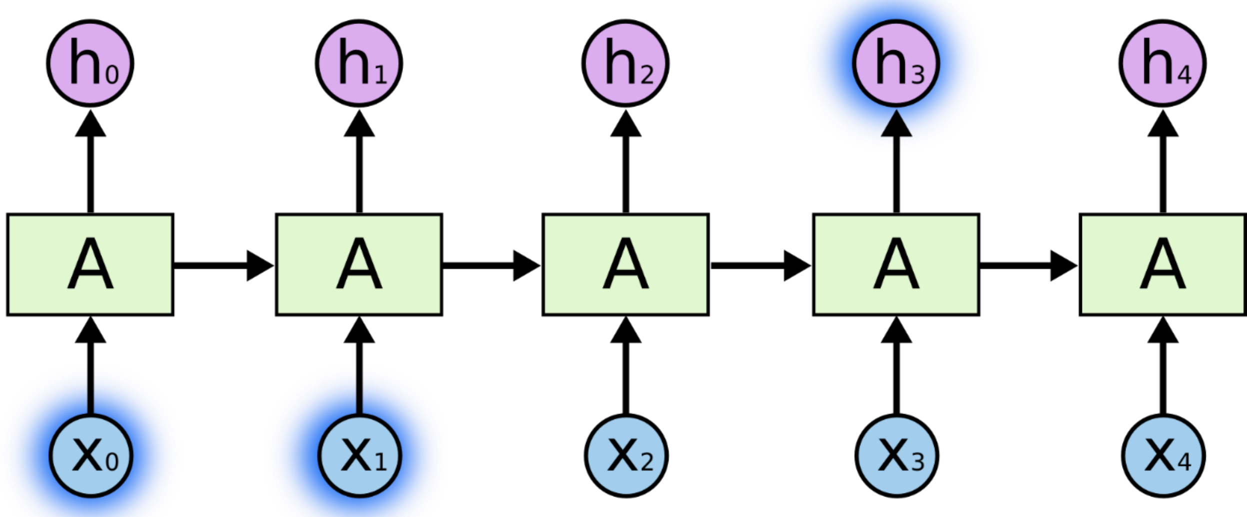
- A language model allows us to predict the probability of observing the sentence
- $P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1})$
  - In words, the probability of a sentence is the product of probabilities of each word given the words that came before it
- The applications of language models are two-fold
  - First, it allows us to score arbitrary sentences based on how likely they are to occur in the real world
    - This gives us a measure of grammatical and semantic correctness
  - Second, a language model allows us to generate new text
    - It is a generative model

Gisma University of Applied Sciences

The Problem of Long-Term Dependencies

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- RNNs can learn to use the past information
  - Where the gap between the relevant information and the place that it is needed is small
  - We only need to look at recent information to perform the present task
- “the clouds are in the …”
  - If we are trying to predict the last word, we do not need any further context
    - It is pretty obvious the next word is going to be “sky”

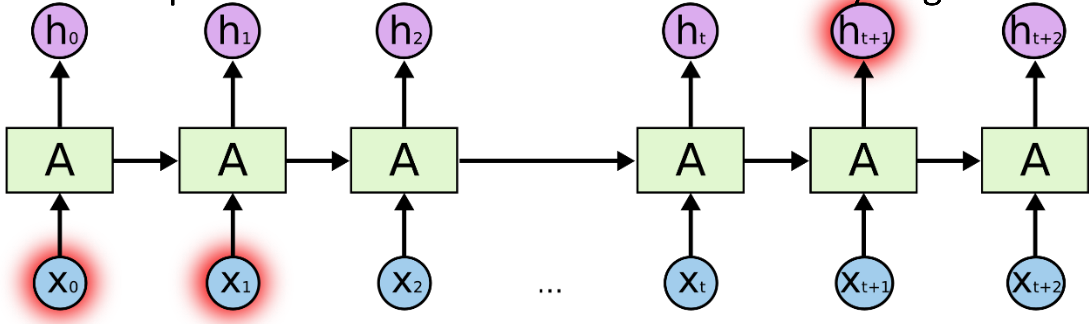


Gisma University of Applied Sciences

The Problem of Long-Term Dependencies

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- RNNs become unable to learn to connect the information as that gap grows
  - In theory, RNNs are absolutely capable of handling such “long-term dependencies”
  - Sadly, in practice, RNNs do not seem to be able to learn them
- “I grew up in France… I speak fluent …”
  - Consider trying to predict the last word in the text
    - Recent information suggests that the next word is probably the name of a language
      - But if we want to narrow down which language, we need the context of France, from further back
  - It is entirely possible for the gap between the relevant information and the point where it is needed to become very large
    - There are also cases where we need more context



Gated RNNs

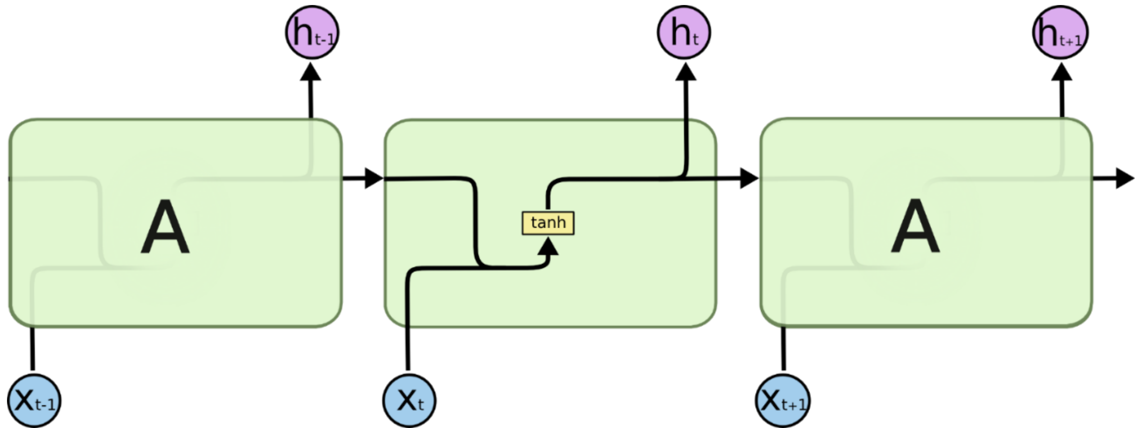
- Gated RNNs are the most effective RNN models used in practice
  - Long short-term memory (LSTM) networks
  - Networks based on the gated recurrent unit (GRU)
- These networks can store information in their units for future use like bits in a computer's memory
  - The trained neural network can "read" the input sequence of feature vectors and decide at some early time step t to keep specific information about the feature vectors
  - That information about the earlier feature vectors can later be used by the model to process the feature vectors from near the end of the input sequence
  - For example, if the input text starts with the word "she", a language processing gated RNN model could decide to store the information about the gender to interpret correctly the word their seen later in the sentence

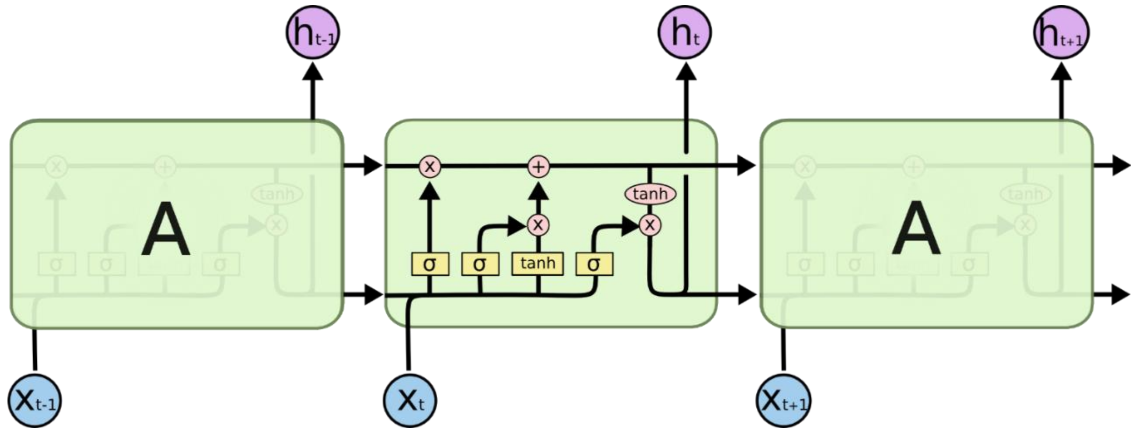
Long Short Term Memory (LSTM)

- LSTMs are a special kind of RNN, capable of learning long-term dependencies
  - Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
  - They work tremendously well on a large variety of problems, and are now widely used
- LSTMs are explicitly designed to avoid the long-term dependency problem
  - Remembering information for long periods of time is practically their default behavior, not something they struggle to learn

LSTM vs RNN

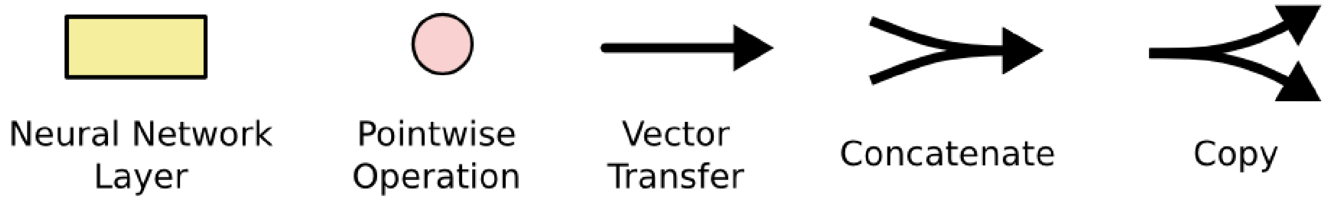
- All recurrent neural networks have the form of a chain of repeating modules of neural network
  - In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer
  - LSTMs also have this chain like structure, but the repeating module has a different structure
    - Instead of having a single neural network layer, there are four, interacting in a very special way

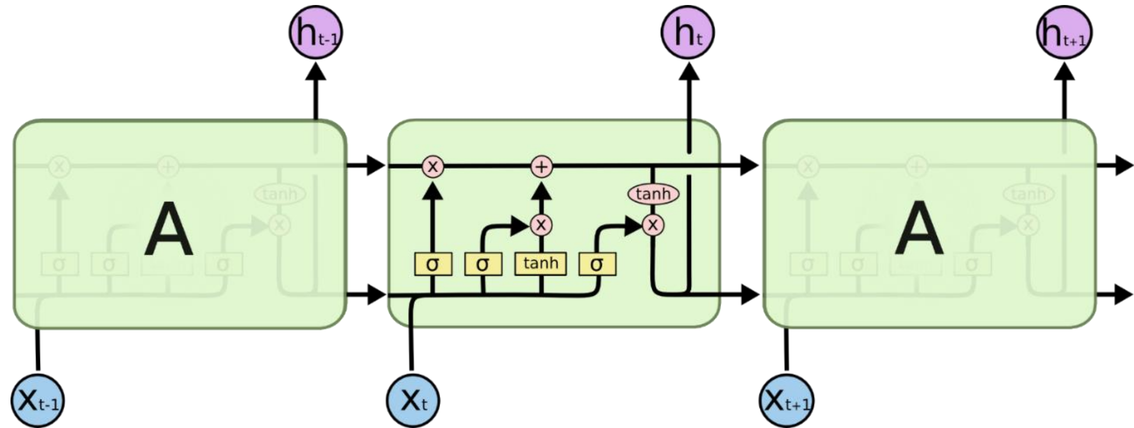




LSTM Architecture

- For now, let us just try to get comfortable with the notation
  - Each line carries an entire vector, from the output of one node to the inputs of others
  - The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers
  - Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations



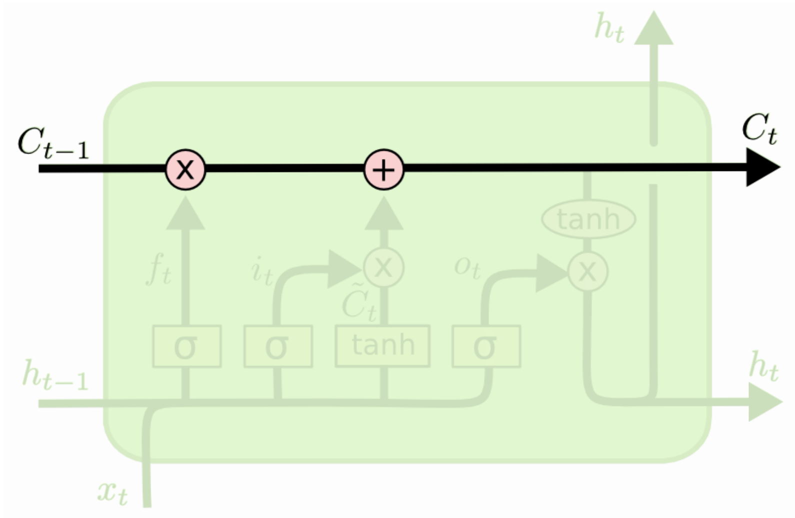


Gisma University of Applied Sciences

LSTM Architecture

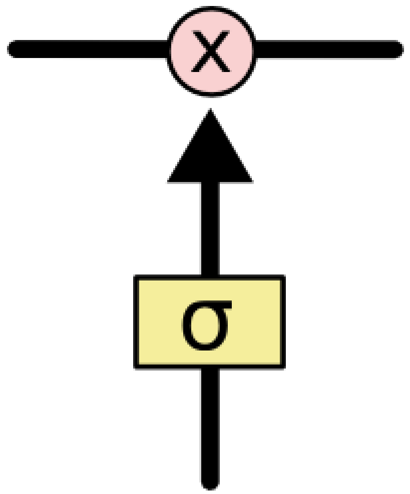
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram
    - The cell state is kind of like a conveyor belt
    - It runs straight down the entire chain, with only some minor linear interactions
    - It is very easy for information to just flow along it unchanged



LSTM Architecture

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates
  - Gates are a way to optionally let information through
  - They are composed out of a sigmoid neural net layer and a pointwise multiplication operation
  - The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through
  - A value of zero means “let nothing through,” while a value of one means “let everything through”
  - An LSTM has three of these gates, to protect and control the cell state

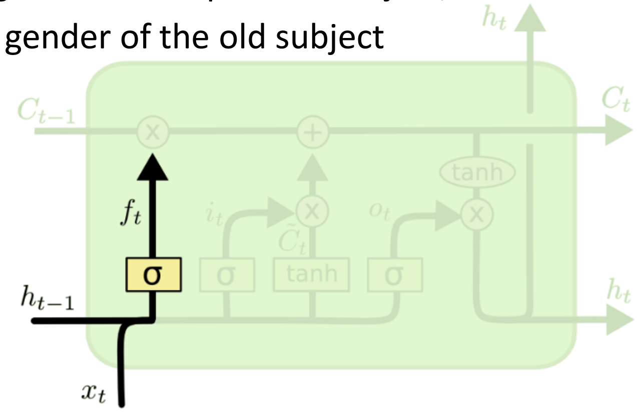


Gisma University of Applied Sciences

Forget Gate Layer

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The first step in our LSTM is to decide what information we are going to throw away from the cell state
  - This decision is made by a sigmoid layer called the “forget gate layer”
  - It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}
  - A 1 represents “completely keep this” while a 0 represents “completely get rid of this”
  - Let us go back to our example of a language model trying to predict the next word based on all the previous ones
    - In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used
    - When we see a new subject, we want to forget the gender of the old subject



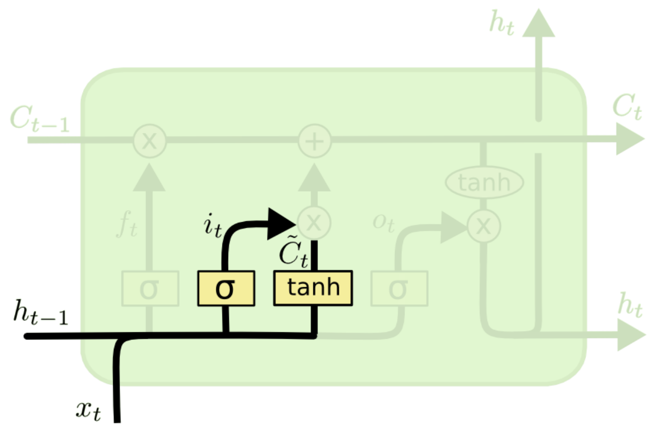
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Gisma University of Applied Sciences

Input Gate Layer

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The next step is to decide what new information we are going to store in the cell state
  - This has two parts
    - First, a sigmoid layer called the “input gate layer” decides which values we will update
    - Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state
  - In the next step, we will combine these two to create an update to the state
  - In the example of our language model, we would want to add the gender of the new subject to the cell state, to replace the old one we are forgetting

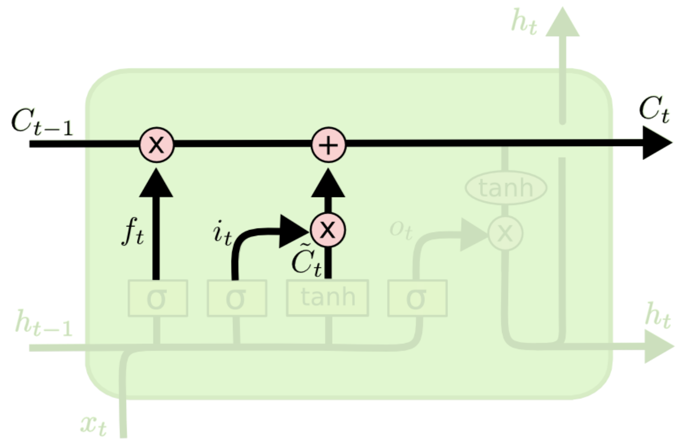


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Gisma University of Applied Sciences

Input Gate Layer

- It is now time to update the old cell state, $\boldsymbol{C}_{t-1}$, into the new cell state $\boldsymbol{C}_t$
  - The previous steps already decided what to do, we just need to actually do it
  - We multiply the old state by f_t, forgetting the things we decided to forget earlier
  - Then we add $\boldsymbol{i}_t * \widetilde{\boldsymbol{C}}_t$. This is the new candidate values, scaled by how much we decided to update each state value
  - In the case of the language model, this is where we would actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps

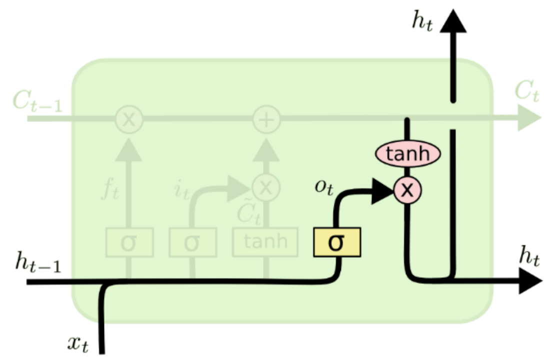


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Gisma University of Applied Sciences

Output Gate Layer

- Finally, we need to decide what we are going to output
  - This output will be based on our cell state, but will be a filtered version
  - First, we run a sigmoid layer which decides what parts of the cell state we are going to output
  - Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to
  - For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that is what is coming next
    - For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next

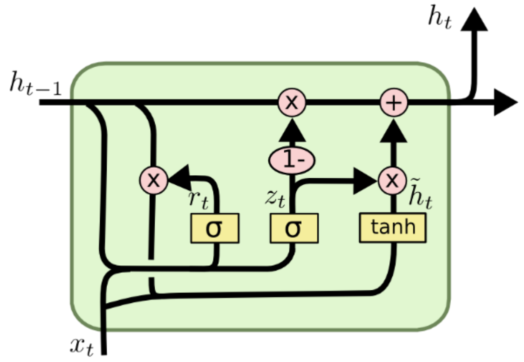


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Gisma University of Applied Sciences

GRU Architecture

- A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU
  - Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint.
  - It combines the forget and input gates into a single "update gate"
  - It also merges the cell state and hidden state, and makes some other changes
  - The resulting model is simpler than standard LSTM models, and has been growing increasingly popular



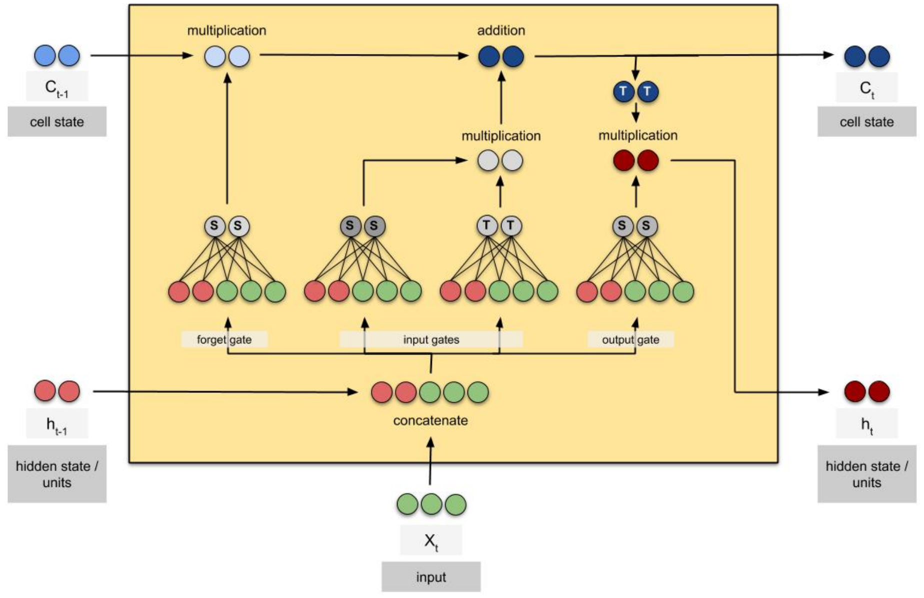
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gisma University of Applied Sciences

The Number of Units

https://tung2389.github.io/coding-note/unitslstm

- The number of units is the dimension of the hidden state (or the output)
    - E.g., keras.layers.LSTM(32) with 32 "units“
- For example, the hidden state here (the red circles) has length 2
    - The number of units is the number of neurons connected to that layer
        - Which are 2 neurons


multiplication
addition
C_{t-1}
cell state
C_t
cell state
multiplication
multiplication
forget gate
input gates
output gate
concatenate
h_{t-1}
hidden state / units
h_t
hidden state / units
X_t
input