Week 10: Emerging topics

Introduction

In the final week of this course, you will learn about the basics of deep learning featuring recurrent and convolutional neural networks. You will cover some of the main themes in ethical aspects of machine learning including terminology and issues, and some ways in which they can be addressed.

Recommended readings

Before reading and engaging with this week's slides you should review the following from Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 2: End-to-End Machine Learning Project
- TBA

Recurrent Neural Networks

Recurrent neural network(**RNN**) is a type of artificial neural network featuring feedback connections that make them appropriate for modelling dynamical systems and temporal sequences. RNNs are based on feedforward neural networks, but they can use their internal state (memory) to process variable-length sequences of inputs.

Simple or vanilla RNNs use context units to store the output of the state neurons from the computation of the previous time steps. The context layer is used for the computation of present states as they contain information about the previous states. The Elman RNN architecture is a vanilla RNN that employs a context layer which makes a copy of the hidden layer outputs in the previous time steps. The dynamics of the change of hidden state neuron activation in Elman RNN.

Elman, J.L. (1990). Finding structure in time. *Cognitive science*, *14*(2), pp.179-211. Retrieved from https://crl.ucsd.edu/~elman/Papers/fsit.pdf.

Below is an example of a simple RNN that shows context layer that features additional memory to propagate information into future time steps from the current state given set of inputs in the input layer.

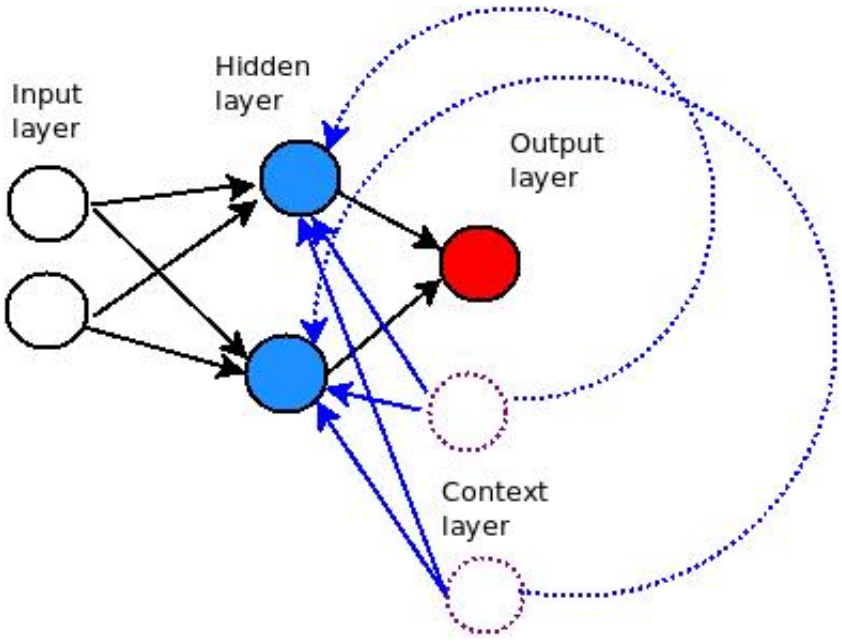


Figure. Simple RNN. Source: R. Chandra

The dynamics of the change of hidden state neuron activation in Elman style recurrent networks is given by the following.

$$y_i(t) = f\left(\sum_{k=1}^{K} v_{ik}\ y_k(t-1) + \sum_{j=1}^{J} w_{ij}\ x_j(t-1)\right)$$

where $y_k(t)$ and $x_j(t)$ represent the output of the context state neuron and input neurons respectively. v_{ik} and w_{ij} represent their corresponding weights. $f(.)$ is the transfer function.

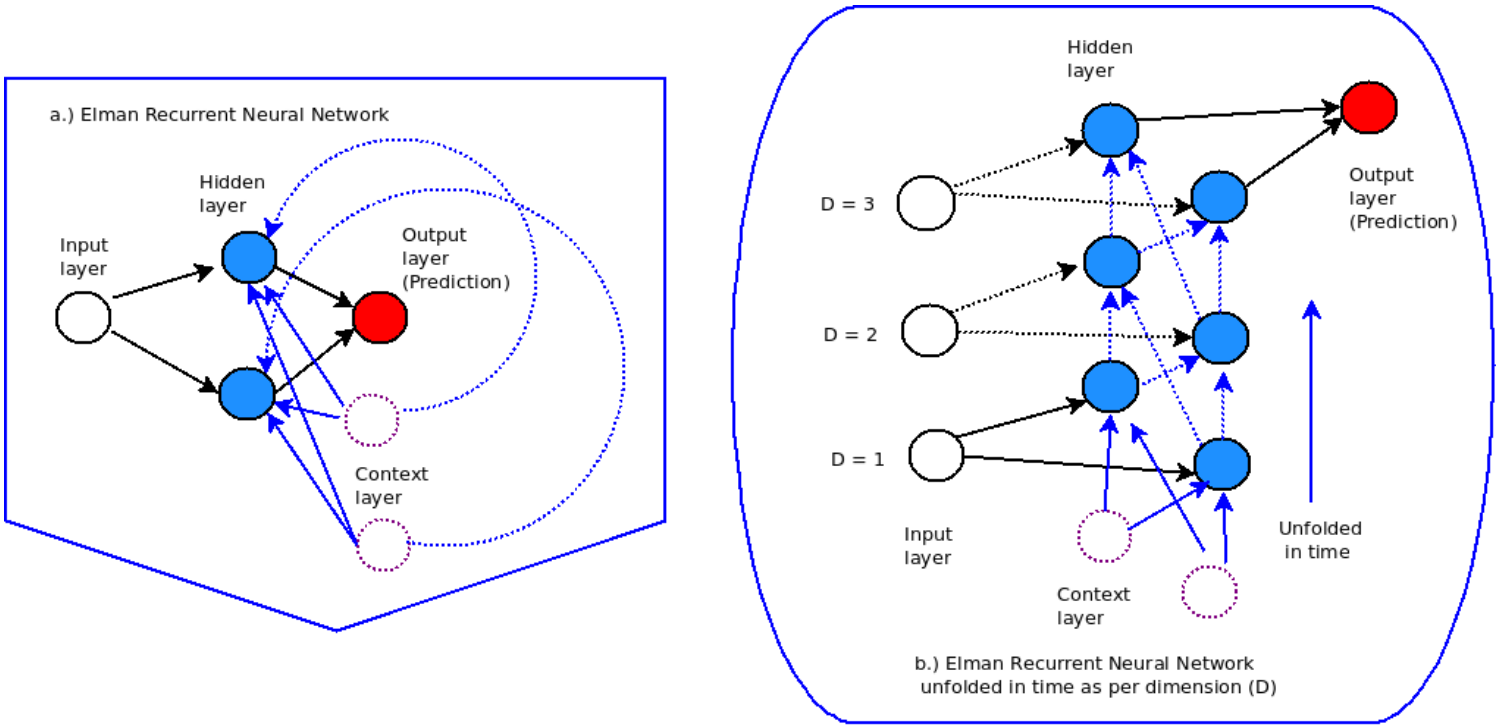


Figure. a) Elman RNN and b) Elman RNN unfolded in time. Chandra, R. (2015). Competition and collaboration in cooperative coevolution of Elman recurrent neural networks for time-series prediction. *IEEE transactions on neural networks and learning systems*, *26*(12), 3123-3136 http://repository.usp.ac.fj/8032/1/TNNLS2404823-FinalPublished.pdf

The below figure shows a simple recurrent neural network architecture: (a) Feedforward network, (b) recurrent network, and (c) how it unfolds in time. Note the similarities and differences between a feedforward neural network (a) and a RNN (Elman,1990, p 179).

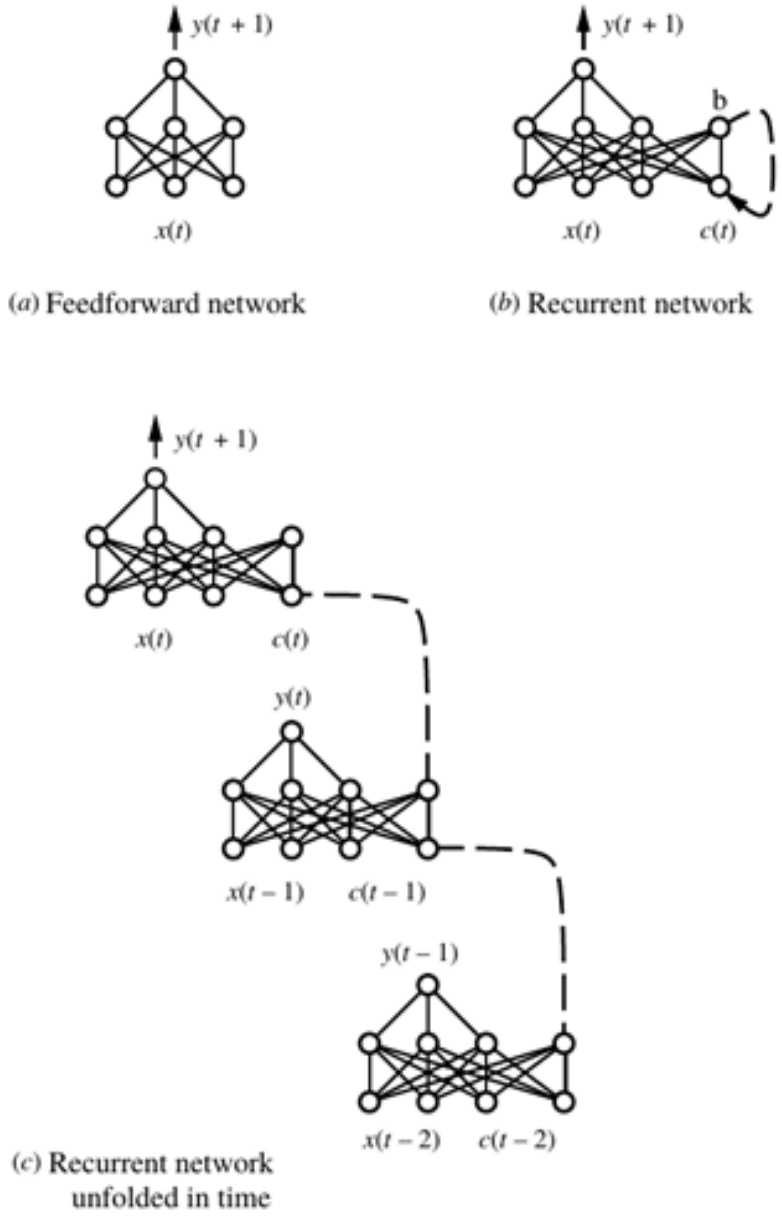


Figure: Recurrent networks. Adapted from *Machine Learning* by T. Mitchell, 1997, Maidenhead; U.K: McGraw Hill.

Backpropagation through time (BPTT)

BPTT begins by unfolding an RNN in time which contains k inputs and outputs, but every copy of the network shares the same parameters. Once unfolded, the backpropagation algorithm is used to find the gradient of the cost with respect to all the network parameters as done in conventional backpropagation for multi-layer perceptron.

Note Pseudocode below.

```python
Back_Propagation_Through_Time(a, y)   // a[t] is the input at time t. y[
    Unfold the network to contain k instances of f
    do until stopping criteria is met:
        x := the zero-magnitude vector // x is the current context
        for t from 0 to n - k do      // t is time. n is the length of t
            Set the network inputs to x, a[t], a[t+1], ..., a[t+k-1]
            p := forward-propagate the inputs over the whole unfolded ne
            e := y[t+k] - p;           // error = target - prediction
            Back-propagate the error, e, back across the whole unfolded
            Sum the weight changes in the k instances of f together.
            Update all the weights in f and g.
            x := f(x, a[t]);           // compute the context for the ne
```

Pseudocode source: https://en.wikipedia.org/wiki/Backpropagation_through_time

Below we see a basic code of BPTT for Elman RNN.

```python


   def ForwardPass(self, sample,slide):
        sample_time = sample[slide]
        layer = 0
        weightsum = 0.0
        StateWeightSum = 0.0
        forwardout=0.0
        for row in range(0,self.Top[0]):
            self.InlayerOutL0[slide+1][row] = sample_time[row]

        for y in range(0, self.Top[1]):
            for x in range(0, self.Top[0]):
                weightsum += self.InlayerOutL0[slide+1][x] * self.W1[x,y
```

A. Ashray and R. Chandra, Elman RNN from scratch in Python: https://github.com/sydney-machine-learning/Elman-RNN-python/tree/master/Elman_RNN

We note that we use the same set of weight for every time step of the computation as shown below.

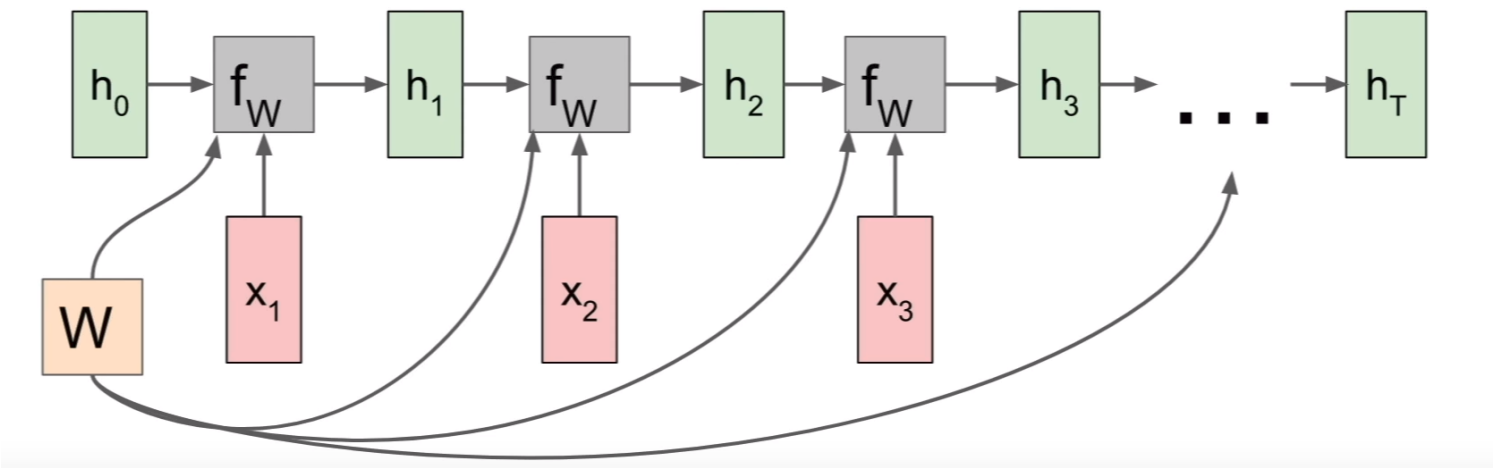


Figure. Computation in RNN. Source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

RNN architectures

Given we have covered simple or vanilla RNN with BTPP, there are many types of datasets and can summarise some major RNN architectures as follows.

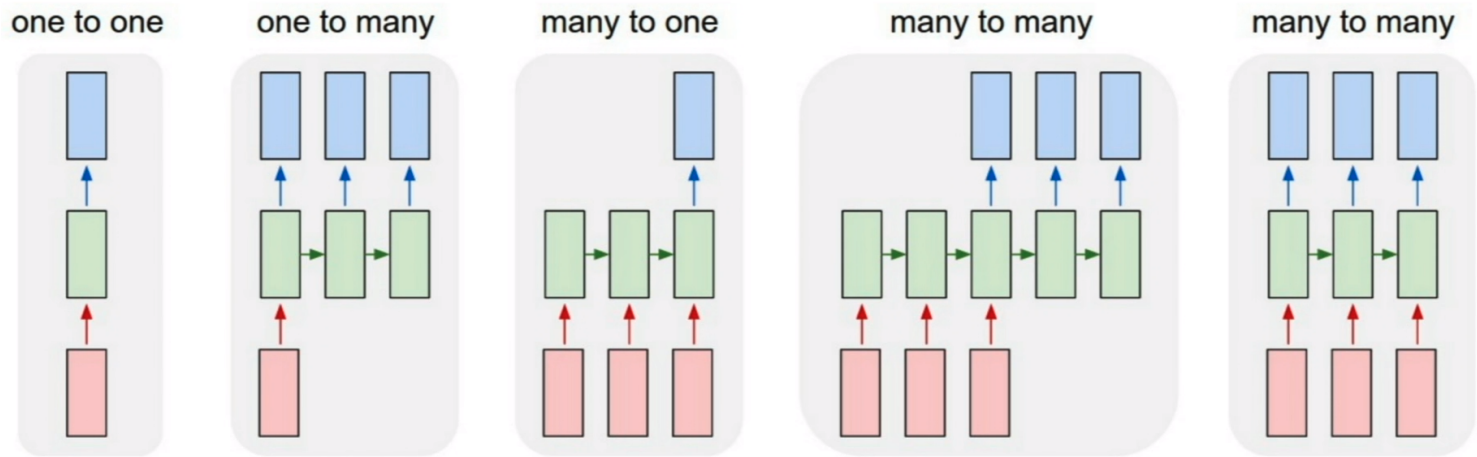


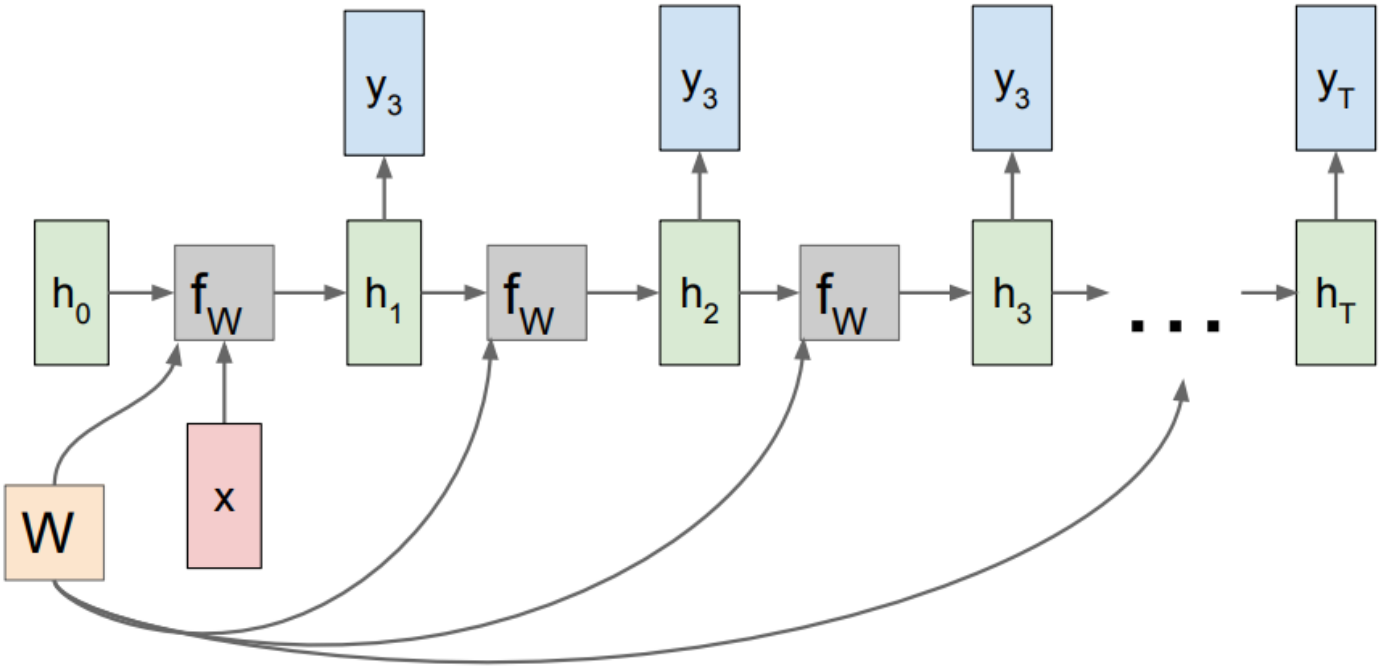
Figure. RNN for different datasets. Source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

1. **One-to-one:** A simple neural network architecture with one input and one output layer.
2. **One-to-many:** We have one input (e.g. image as a fixed size input) and the output can be words or sentences.
3. **Many-to-one:** The input is expected to be a word sequence or paragraph and output can be continuous values for sentiment analysis (classify sad, happy set of inputs).
4. **Many-to-many:** ideal for machine translation like the one we see on Google translate where

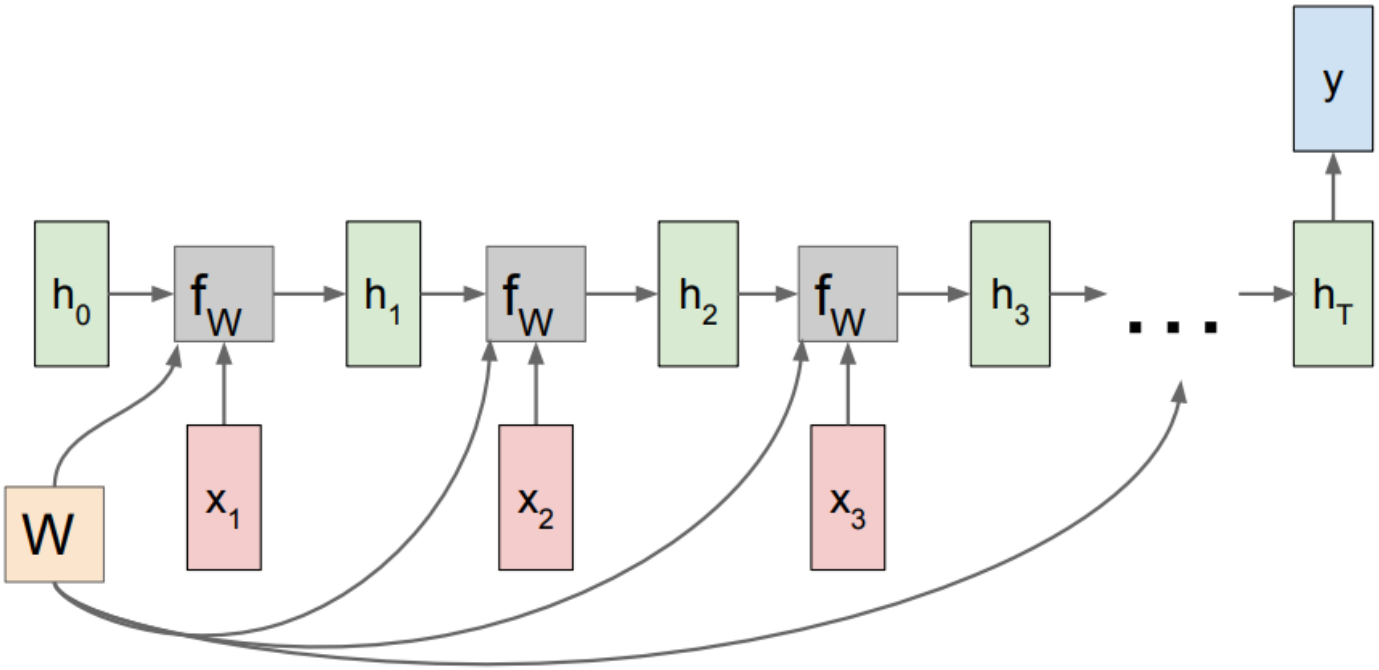
the input could an English sentence and output is a different language with variable lengths. It can also be used for video classification on frame level, i.e after certain frame inputs, recognise objects in video.

Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S. (2011, May). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 5528-5531). IEEE.

RNN: Computational Graph: One to Many



RNN: Computational Graph: Many to One



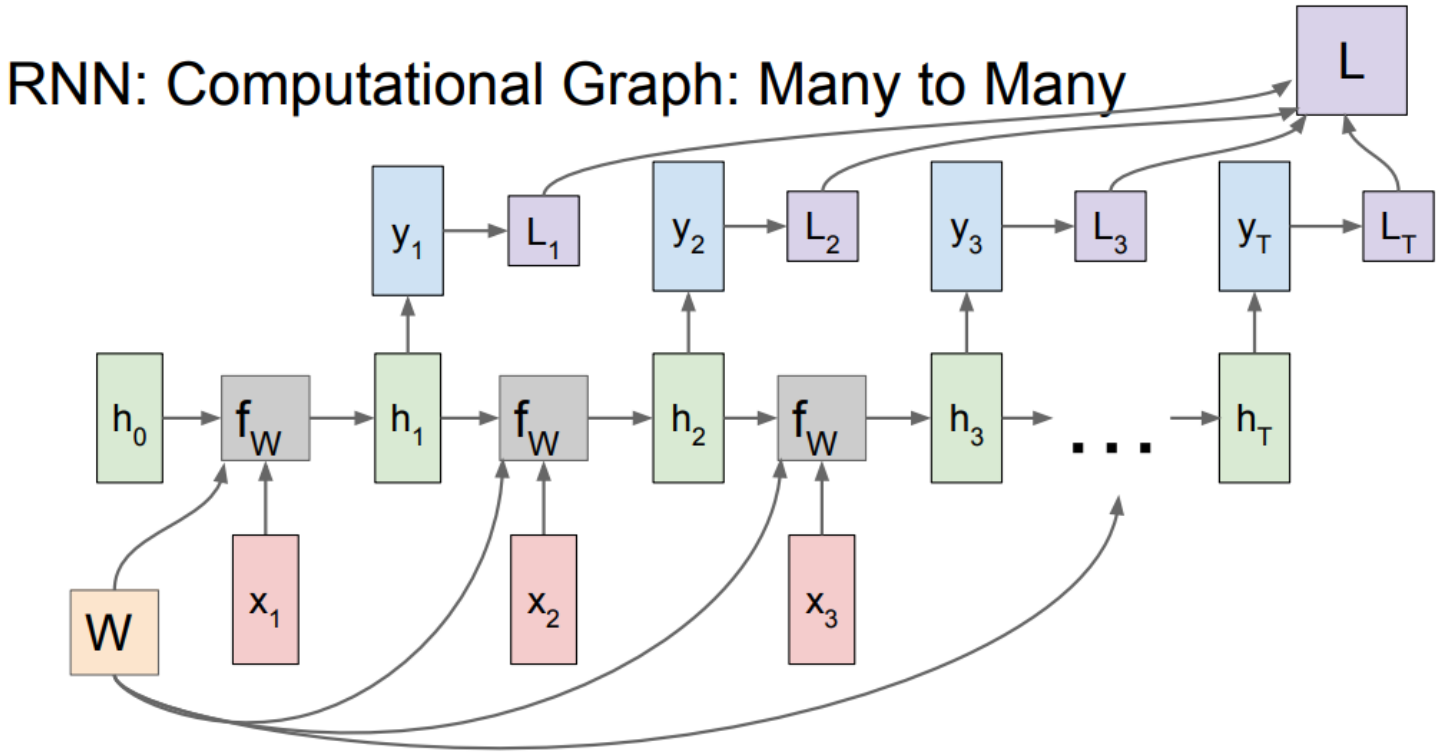


Figure. RNN models. Source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

We have a simple one-to-many vanilla RNN that produce output h_t given some weight matrices W_{hh} and W_{xh}, with previous outputs h_{t-1} with input x_t and a transfer function `tanh()`.

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + B_h)$$

Given a **one-to-many** model, we wish to produce an output y_t at every timestep and therefore, we need another weight matrix that accepts a hidden state and project it to an output.

$$y_t = W_{hy}h_t + B_y$$

Below we see basic idea wit code about how computation is done in one to many RNN model.

Run PYTHON

```python
import numpy as np


np.random.seed(0)
class RecurrentNetwork(object):
    """When we say W_hh, it means a weight matrix that accepts a hidden
    Similarly, W_xh represents a weight matrix that accepts an input vec
    notation can get messy as we get more variables later on with LSTM a
    LSTM notes.
    """
    def __init__(self):
        self.hidden_state = np.zeros((3, 3))
        self.W_hh = np.random.randn(3, 3)
        self.W_xh = np.random.randn(3, 3)
```

Code Source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Considering softmax loss function, our gradient for every time step would be:

$$\frac{\partial L}{\partial W_{hy}}, \frac{\partial L}{\partial W_{By}}, \frac{\partial L}{\partial h_t}, \frac{\partial L}{\partial B_h} \frac{\partial L}{\partial W_{hh}}, \frac{\partial L}{\partial W_{xh}}$$

BPTT training would become problematic when we want to train a sequence that is very long due to vanishing error gradient problem.

In the case of training a paragraph of words, we have to iterate through many layers to compute one simple gradient step. Truncated backpropagation is an approximation of BPTT where we can run forward and backwards through chunks of the sequence instead of the whole sequence.

Puskorius, G. V., & Feldkamp, L. A. (1994, June). Truncated backpropagation through time and Kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* (Vol. 4, pp. 2488-2493). IEEE.

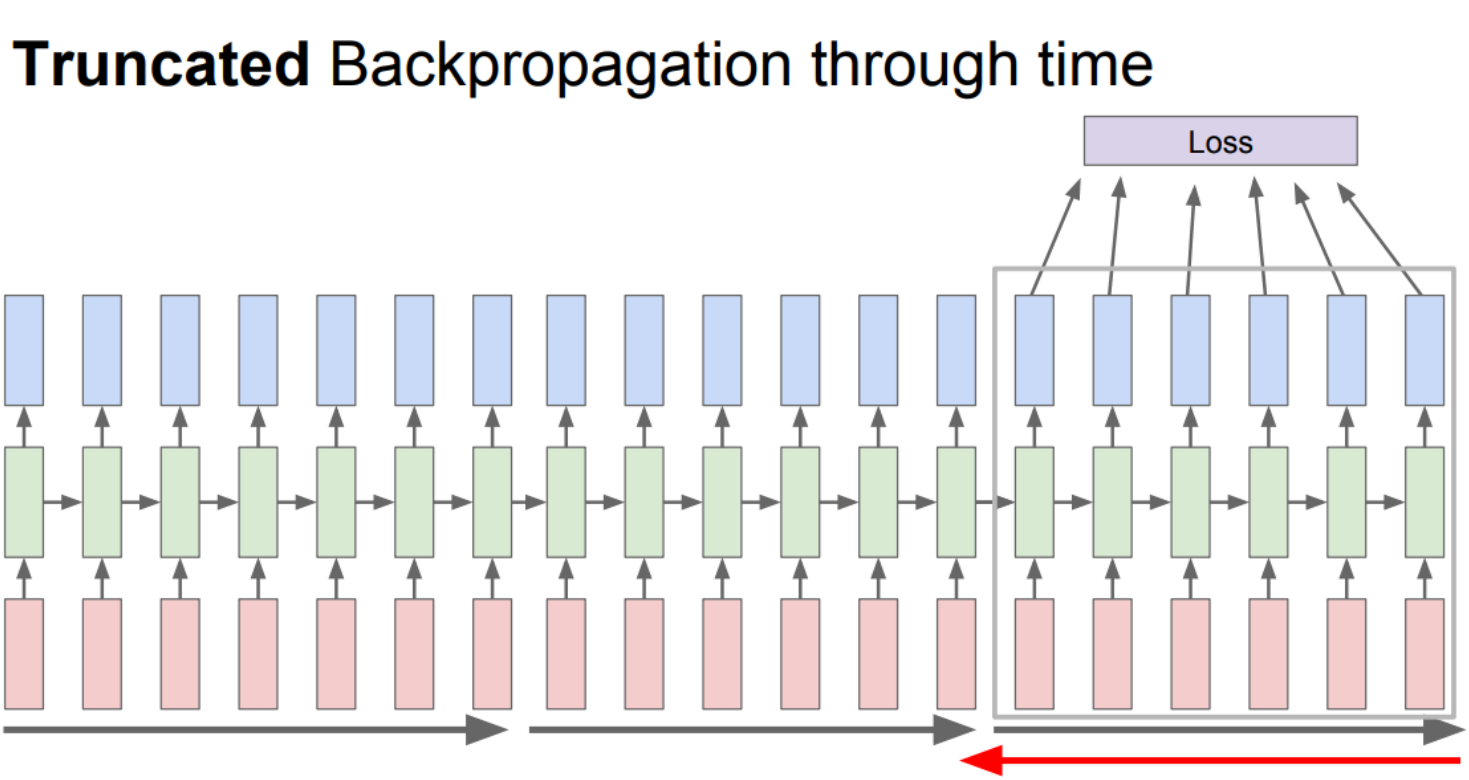


Figure. RNN models. Source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Although our input sequence can potentially be very long, we will step forward for some number of steps and compute a loss only over this subsequence of the data while training. We then backpropagate through the sub-sequence and make a gradient step on the weights.

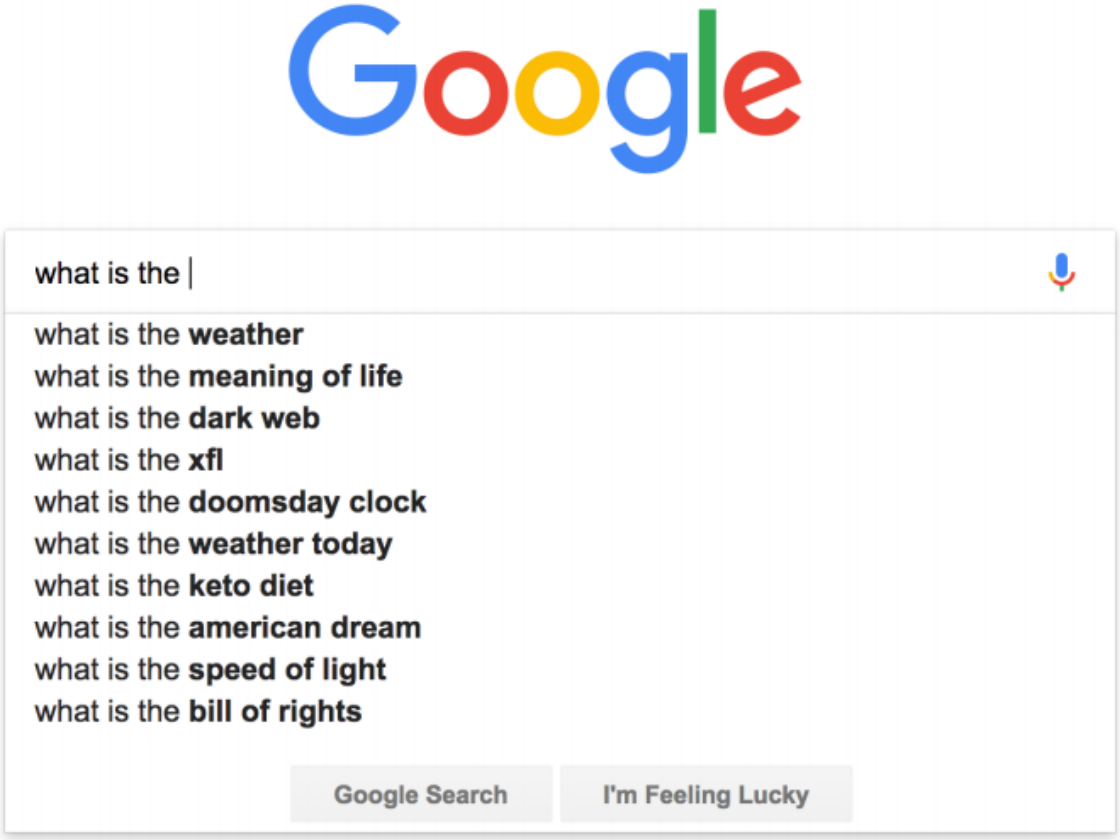
In the next batch, we still have this hidden state from the previous batch and carry hidden state forward. The forward pass is unaffected in this way and we only backpropgate again through the second batch.

Language model with RNN

RNNs have been prominent for language models in natural language process applications.

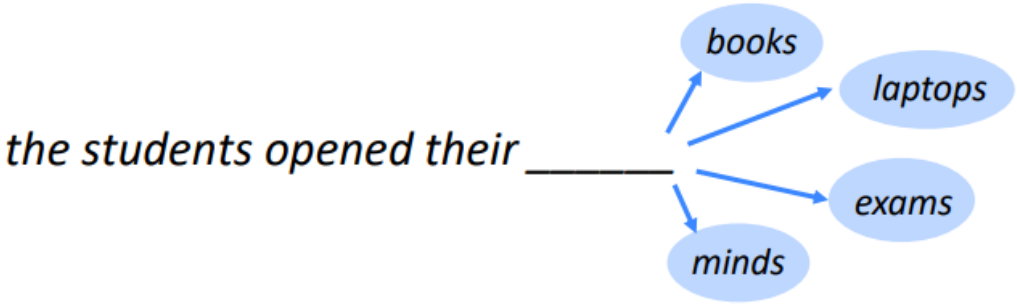
Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S. (2011, May). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 5528-5531). IEEE.

You use Language Models every day!



Language Modeling

- **Language Modeling** is the task of predicting what word comes next.



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

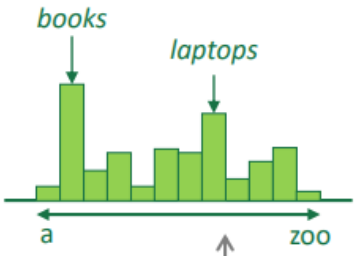
$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

  where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

A RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

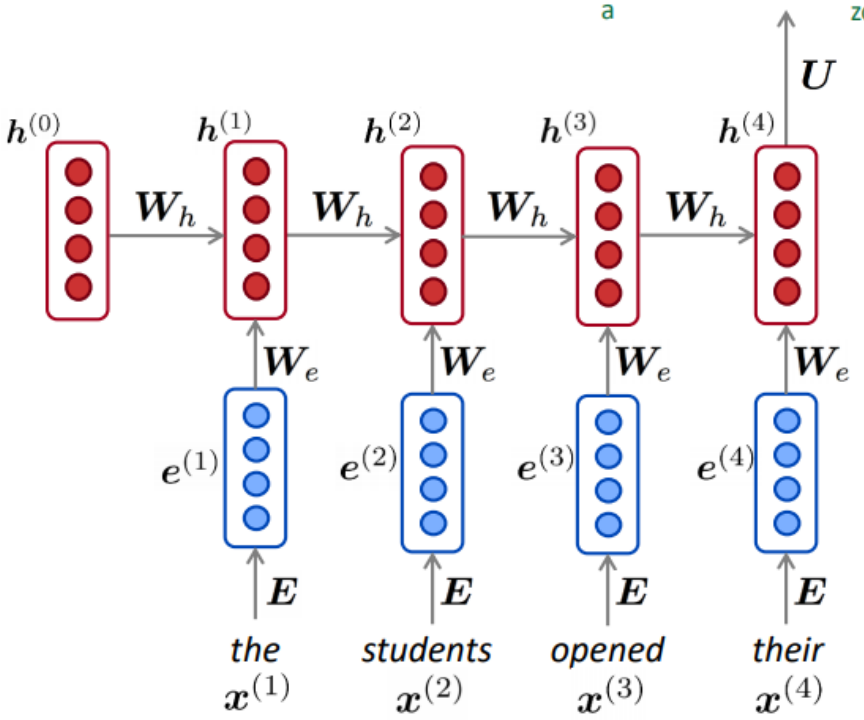
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Note: this input sequence could be much longer, but this slide doesn't have space!

A RNN Language Model

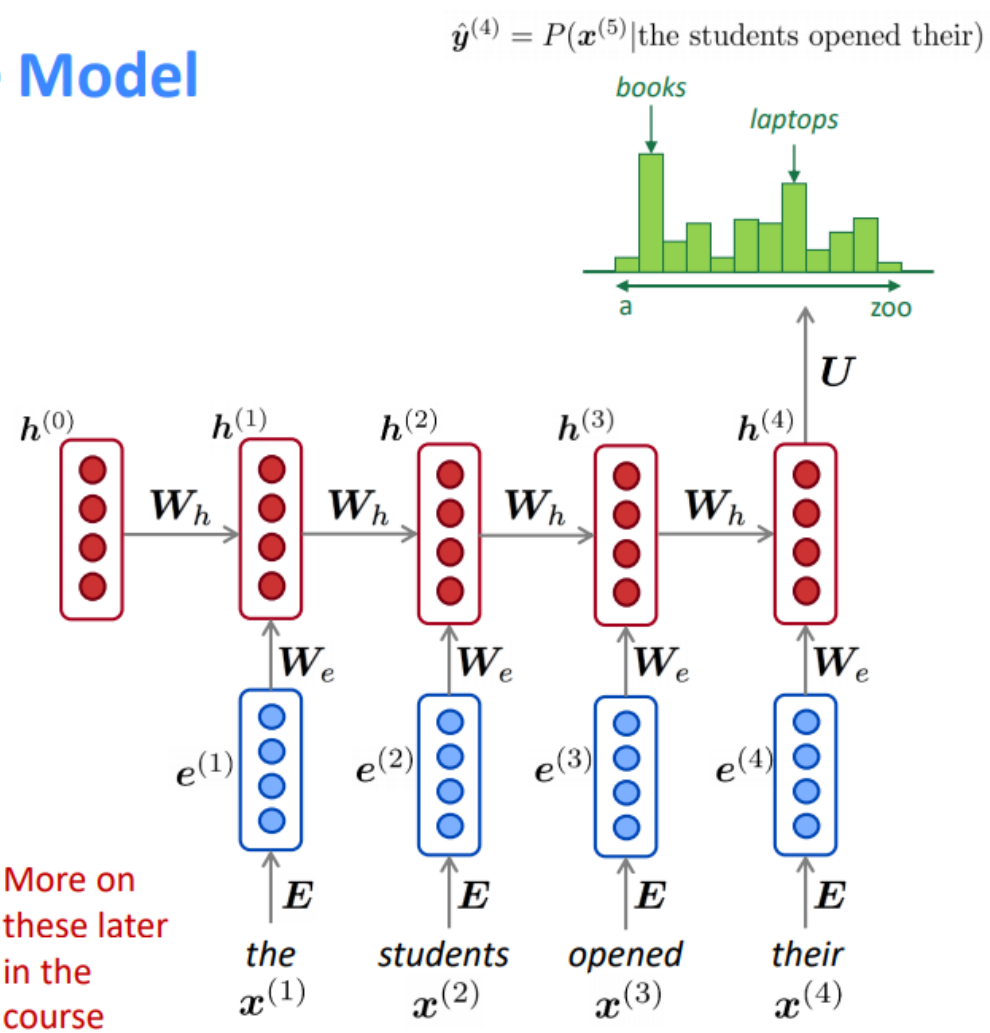
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course



24

Slide Source: RNN for Language Modelling. Stanford Lecture series: https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf

We note that traditional statistical methods and n-grams have been used typically for language modelling prior to RNNs.

Niesler, T. R., & Woodland, P. C. (1996, May). A variable-length category-based n-gram language model. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings* (Vol. 1, pp. 164-167). IEEE.

Below we can see a language model for characters that can help in typing and improving spelling. We note that a one-hot encoding is used as shown below for inputs to language model via RNN.

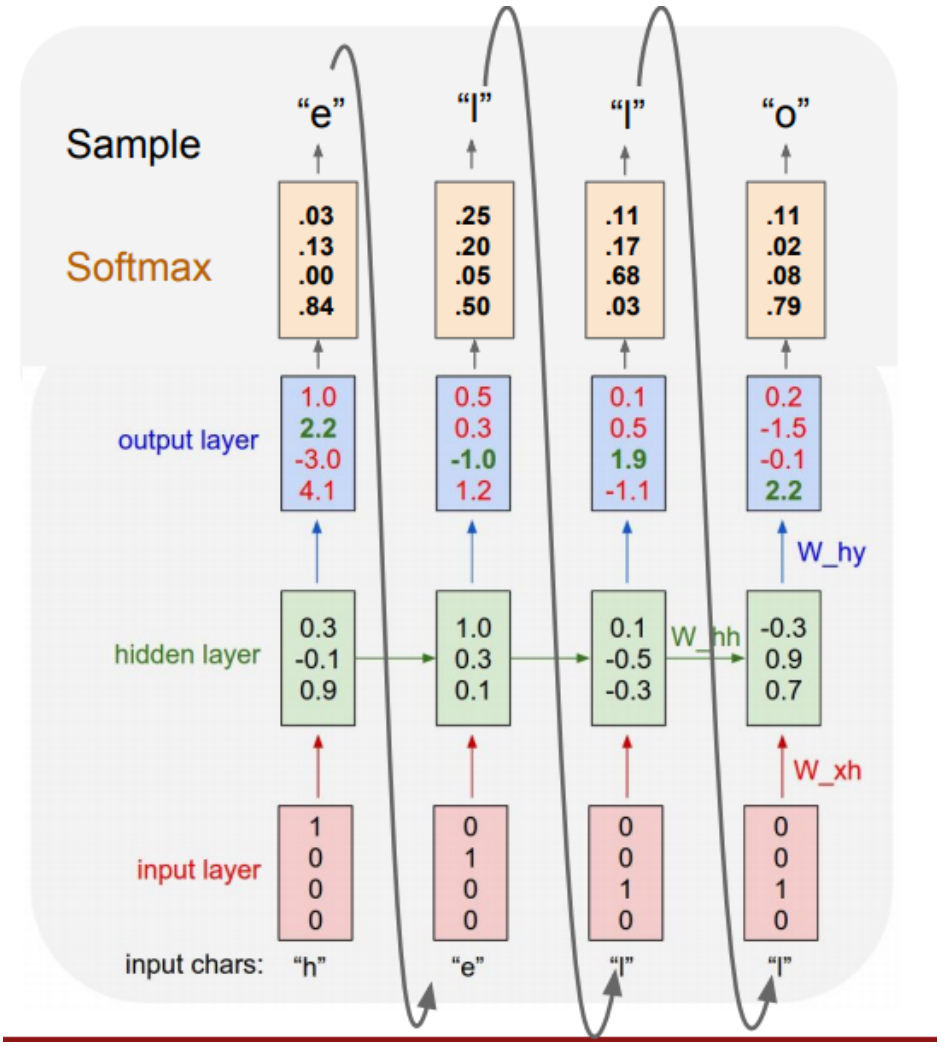
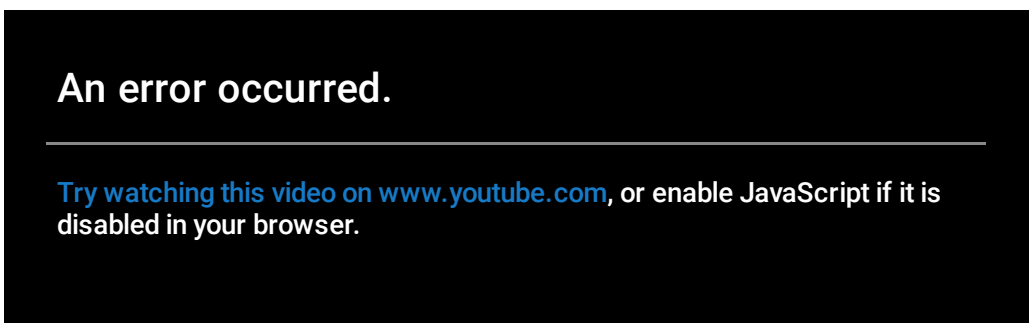


Figure Source. RNN language model for characters: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

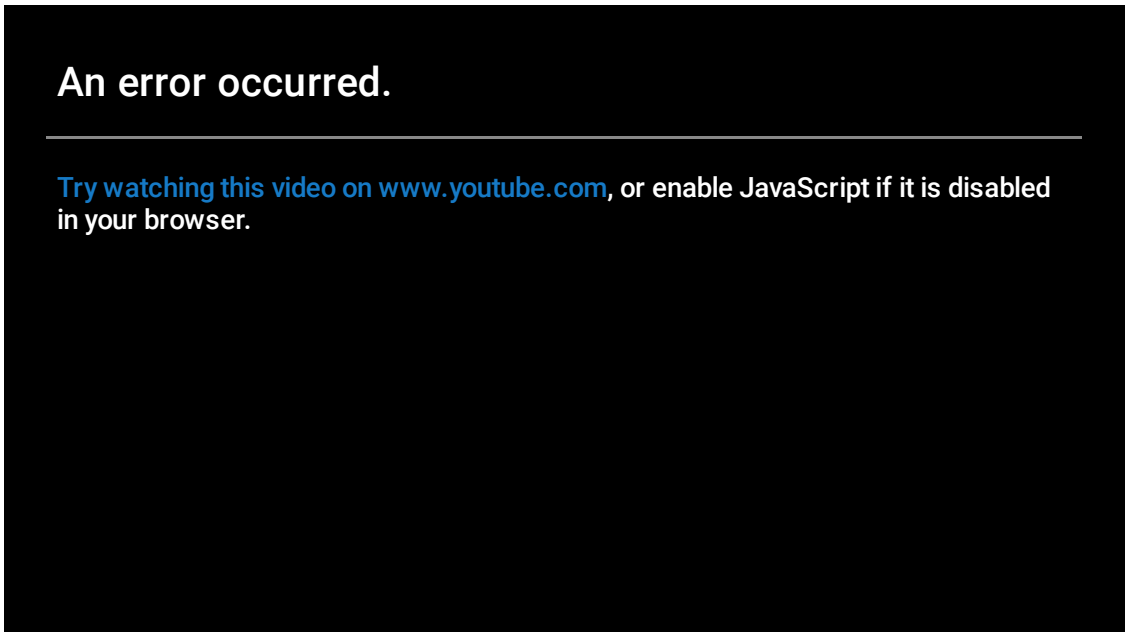
We seed the word with a prefix like a letter h in this case above and output is a softmax vector which represents probability.

Some videos

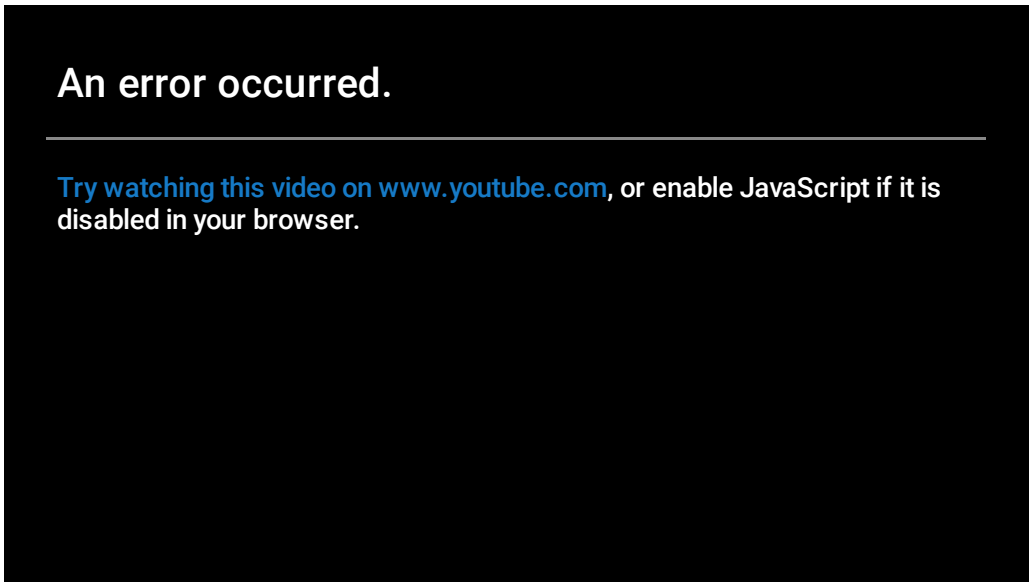
Sentiment analysis with simple RNNs



BPTT



Truncated BPTT



References

1. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.

2. Elman, J.L. (1990). Finding structure in time. *Cognitive science*, *14*(2), pp.179-211. Retrieved from https://crl.ucsd.edu/~elman/Papers/fsit.pdf.
3. https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf
4. Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550-1560.http://www.werbos.com/Neural/BTT.pdf
5. Williams, R. J., & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, *2*(4), 490-501.http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.7941&rep=rep1&type=pdf
6. Blanco, A., Delgado, M., & Pegalajar, M. C. (2001). A real-coded genetic algorithm for training recurrent neural networks. *Neural networks*, *14*(1), 93-105.
7. Chandra, R. (2015). Competition and collaboration in cooperative coevolution of Elman recurrent neural networks for time-series prediction. *IEEE transactions on neural networks and learning systems*, *26*(12), 3123-3136 http://repository.usp.ac.fj/8032/1/TNNLS2404823-FinalPublished.pdf
8. Sutskever, I. (2013). *Training recurrent neural networks*. Toronto, Canada: University of Toronto. http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf
9. Puskorius, G. V., & Feldkamp, L. A. (1994, June). Truncated backpropagation through time and Kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* (Vol. 4, pp. 2488-2493). IEEE.
10. Tallec, C., & Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*.https://arxiv.org/pdf/1705.08209.pdf
11. Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S. (2011, May). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 5528-5531). IEEE.
12. Mikolov, T., & Zweig, G. (2012, December). Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)* (pp. 234-239). IEEE.
13. Auli, M., Galley, M., Quirk, C., & Zweig, G. (2013). Joint language and translation modeling with recurrent neural networks: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/EMNLP2013RNNMT.pdf
14. Niesler, T. R., & Woodland, P. C. (1996, May). A variable-length category-based n-gram language model. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings* (Vol. 1, pp. 164-167). IEEE.

LSTM network models

At times, we only need to look at recent information to perform the present task, such as a language model trying to predict the next word based on the previous ones. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use past information.

There are also cases where we need more context where the gap between the relevant information and the point where it is needed to become very large. As that gap grows, RNNs have difficulty to learn to connect the information. The problem is known as the vanishing gradient problem where RNNs have difficulty to learn long term dependencies in sequences.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

Below we can see the problem of learning long term dependencies or the vanishing error gradient problem.

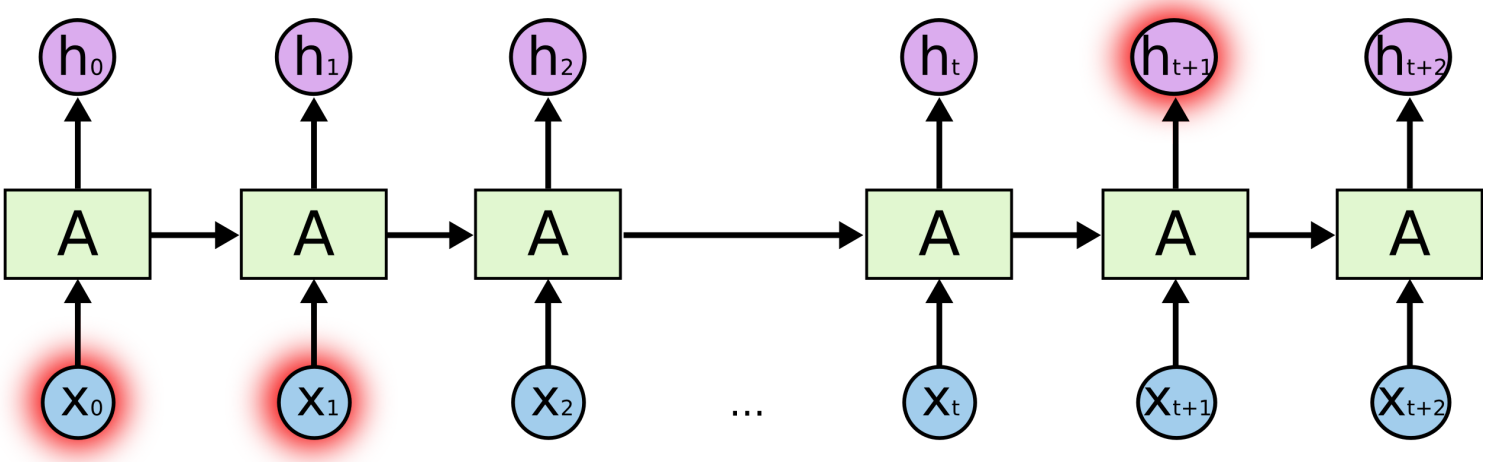


Figure Source. Understanding LSTM Networks: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

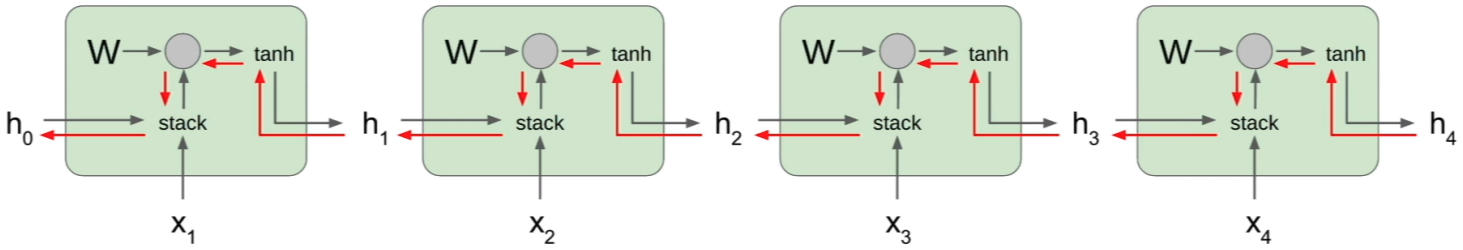


Figure Source. Understanding LSTM Networks: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

When we backpropagate from `h[t]` to `h[t-1]`, the gradient will flow through the `tan()` gate and then to matrix multiplication gate. Since we backpropagate into matrix multiplication gate, the upstream gradient is multiplied by the transpose of the `W` matrix, which takes place at every time step throughout the sequence.

The problem arises when the sequence is too long.

The final expression for the gradient on `h[0]` will involve many factors of this weight matrix which will either lead to an exploding gradient or vanishing gradient problem.

Long short-term memory networks (LSTM)

LSTM has been developed to address the problem of learning long term dependencies in data by RNNs, which recently gained popularity due to the deep learning revolution.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780. Retrieved from https://www.mitpressjournals.org/doi/pdfplus/10.1162/neco.1997.9.8.1735.

LSTM models use memory cells and gates to capture information in long term dependencies in data.

LSTM models feature an advanced recurrence relation than the vanilla RNN. LSTMs provided better capabilities in remembering the long-term dependencies using memory cells and gates.

LSTM has two states, where one is the hidden state `h[t]` as given in vanilla RNN and the other is called the cell state `c[t] which` is an internal vector that is not exposed to the outside world, i.e LSTM architecture.

We have major terminologies that define the components of the cell state.

1. `f` is the **forget gate** which determines whether to erase cell.
2. `i` is the **input gate** which determines whether to write to cell.
3. `g` is the **gate gate** which determines how much to write to cell.
4. o is the **output gate** which determines how much to reveal cell.

We take the previous cell state and the hidden state as the inputs to the current state cell in the LSTM model. The previous hidden state is combined with the input vector and multiplied with the weight matrix to produce `ifog`. The forget gate multiplies element-wise with the previous cell state, while the input and gate gate also multiply elementwise. The two results are combined through sum elementwise to produce a new cell state which is then squashed by a `tanh` and multiplied element-wise by the output gate to produce our next hidden state as show below.

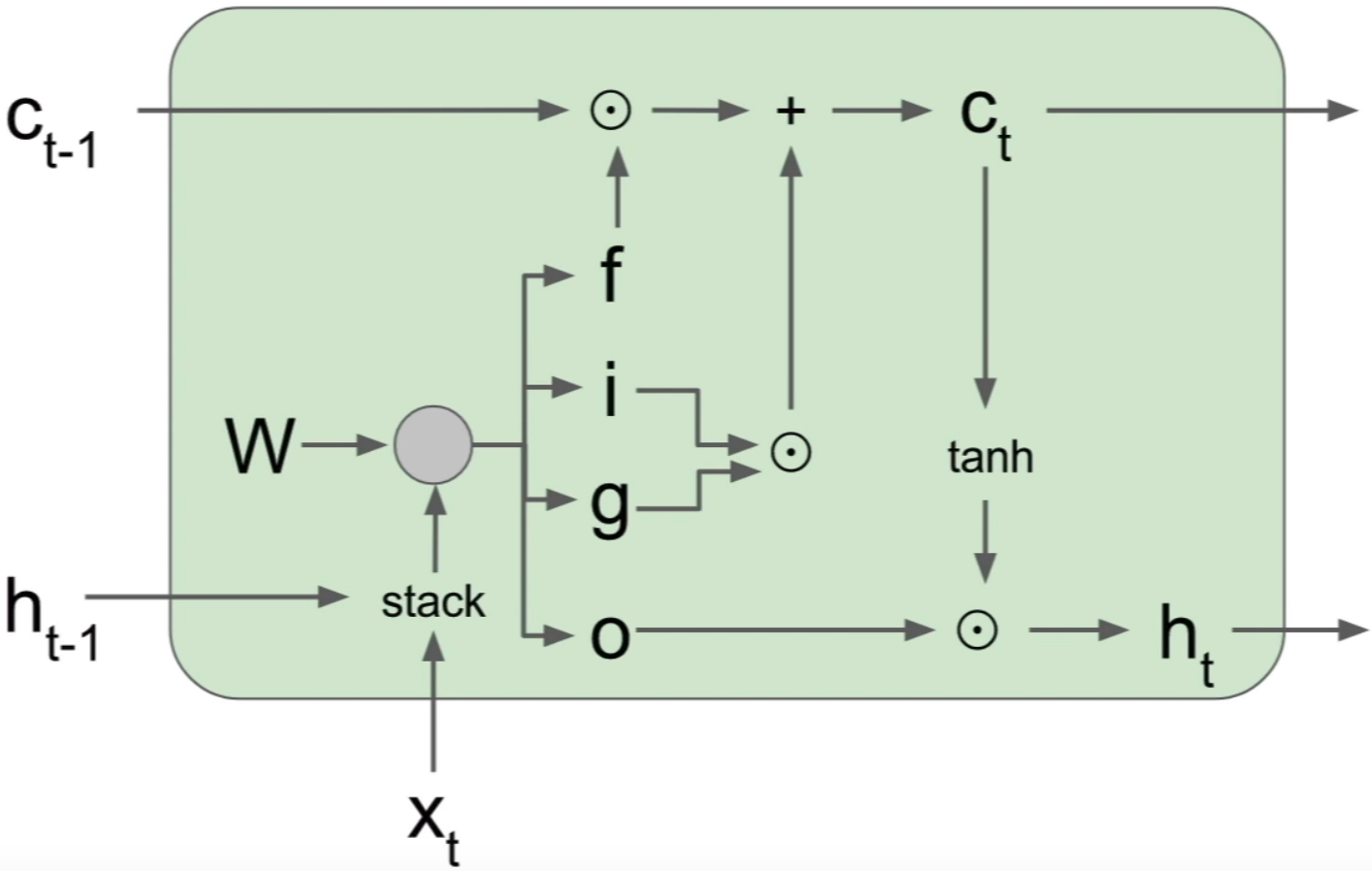


Figure. LSTM memory: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/long_short_term_memory

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

where σ represents sigmoid transfer function. The cell state is defined by the following.

$$c_t = f \odot c_{t-1} + i \odot g$$
$$c_t = \sigma(W_{hf} h_{t-1} + W_{xf} x_t) \odot c_{t-1} + \sigma(W_{hi} h_{t-1} + W_{xi} x_t) \odot tanh(W_{hg} h_{t-1} + W_{xg} x_t)$$

The hidden state is a function of the cell state is given below.

$$\begin{aligned} h_t &= o \odot tanh(c_t) \\ &= \sigma\left(W_{hho} h_{t-1} + W_{xho} x_t\right) \odot tanh(c_t) \end{aligned}$$

In terms of training, we backpropagate from `c[t]` to `c[t-1]` with element-wise multiplication by the `f` gate, and there is no matrix multiplication by W. The `f` gate is different at every time step, ranged between 0 and 1 due to sigmoid property, thus we have avoided the problem of multiplying the same thing over and over again.

Hence, in backpropagating `h[t]` to `h[t-1], we` go through only one single `tanh()` nonlinearity, rather than `tanh()` for every single step as shown below.

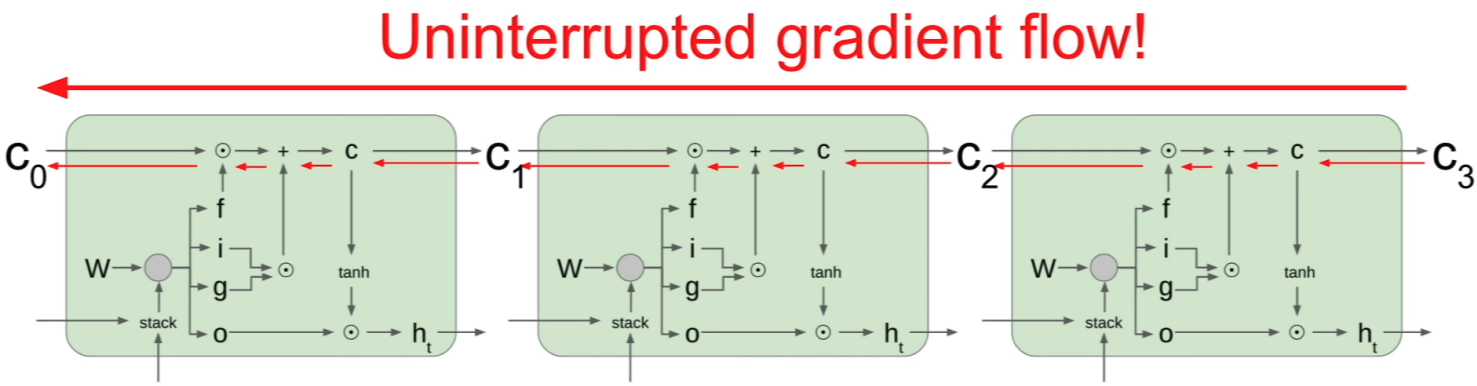


Figure. LSTM memory and gradient flow: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/long_short_term_memory

We note that finer details are beyond the scope of this lesson. Further information you can get with

practice and going over basic LSTM code.

LSTM code in python from scratch: https://www.kaggle.com/navjindervirdee/lstm-neural-network-from-scratch

LSTM in R via Keras

Below we see LSTM example in R for time series prediction problem with output shown below.

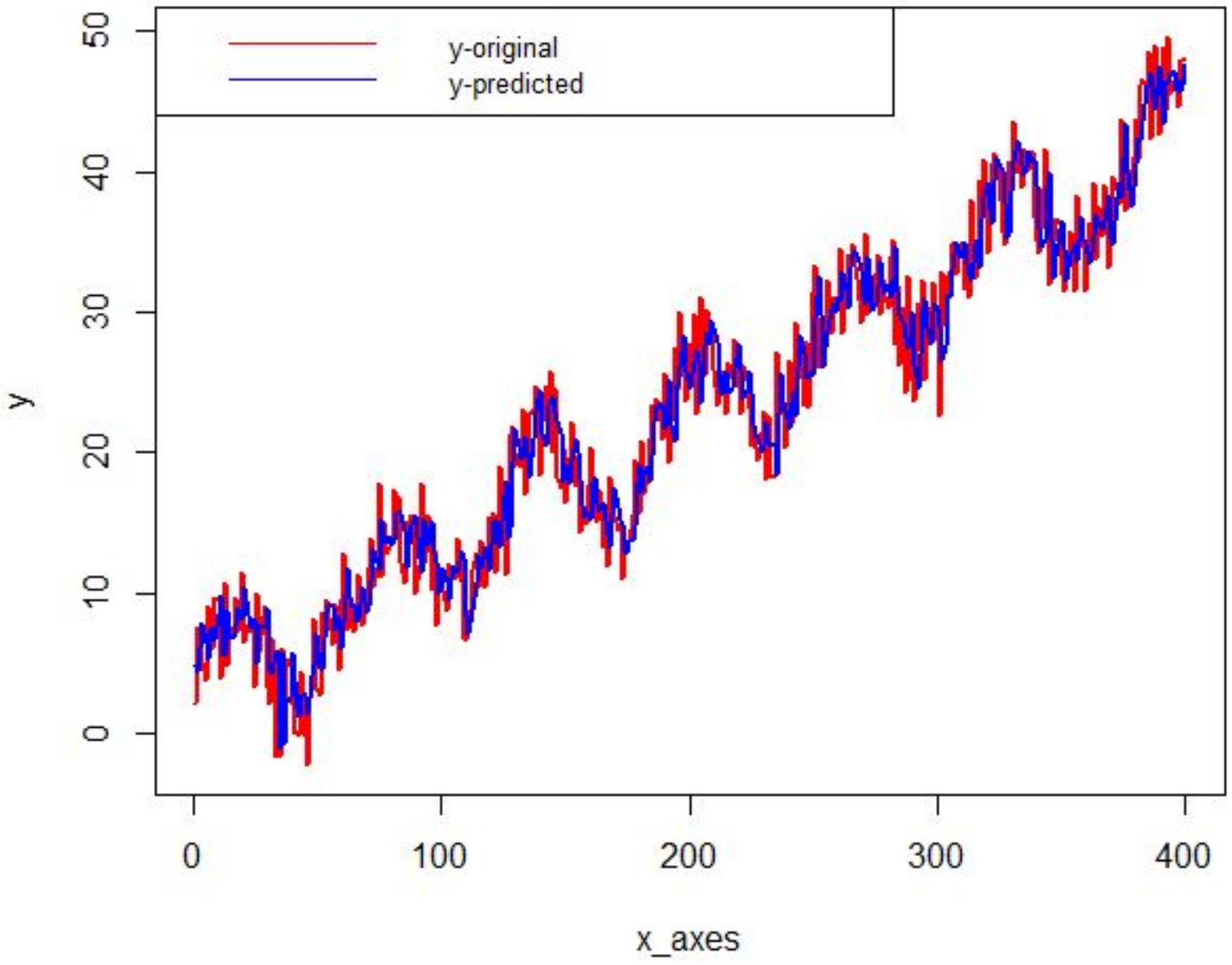


Figure. LSTM for prediction. Source: https://www.datatechnotes.com/2019/01/regression-example-with-lstm-networks.html#:~:text=The%20LSTM%20(Long%20Short%2DTerm,observed%20part%20of%20the%20elements.

You can change the step size and observe the prediction results.

Run

R

```r
#LSTM for time series prediction

library(keras)


N = 400
set.seed(123)
n = seq(1:N)
a = n/10+4*sin(n/10)+sample(-1:6,N,replace=T)+rnorm(N)
head(a,20)

#reshape - windowing

step = 3  # step is a window size
```

Code Source. LSTM: https://www.datatechnotes.com/2019/01/regression-example-with-lstm-networks.html#:~:text=The%20LSTM%20(Long%20Short%2DTerm,observed%20part%20of%20the%20elements.

Sentiment Analysis with LSTM

As mentioned, natural language processing (NLP) is major application for RNNs and LSTMs. Sentiment analysis systematically identifies, extract, quantify, and study affective states (emotions) and subjective information. Sentiment analysis is one of the most common applications in NLP.

Zhang, L., Wang, S., & Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1253. https://arxiv.org/ftp/arxiv/papers/1801/1801.07883.pdf

We use movie reviews in IMDB data set for sentiment analysis using LSTM model.

Note this does not execute due to Ed limitations, but you can try in your own computer.

Run PYTHON

```
from keras.datasets import imdb
#Set the vocabulary size and load in training and test data.
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = vocabu
print('Loaded dataset with {} training samples, {} test samples'.format(

word2id = imdb.get_word_index()
id2word = {i: word for word, i in word2id.items()}
print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[6]])
print('---label---')
print(y_train[6])

print('Maximum review length: {}'.format(
```

Code Source: https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e

We note that R example implementation is given here: https://keras.io/examples/nlp/text_classification_from_scratch/

Once processing is done, we can then build our LSTM model to develop a language model.

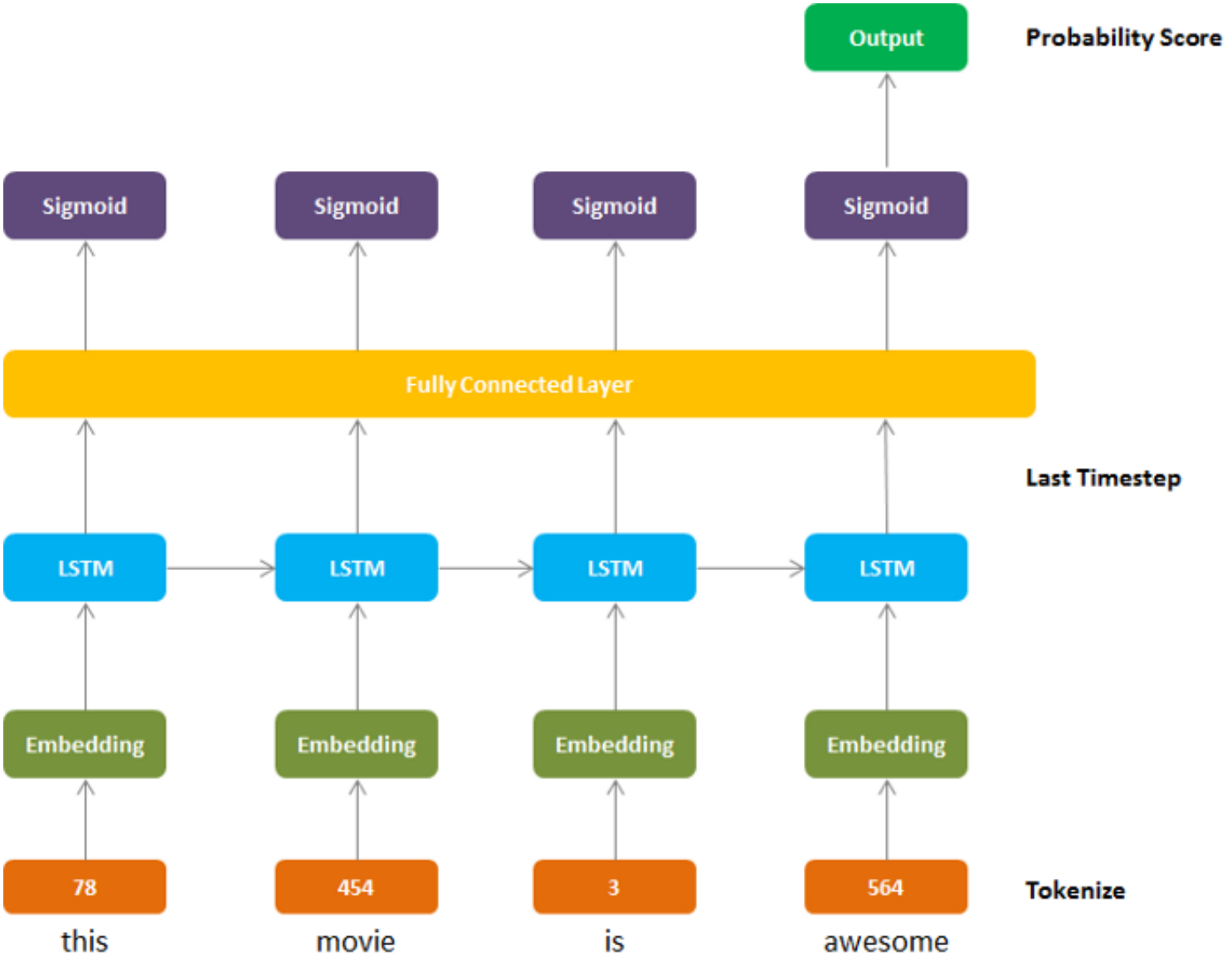


Figure. A step by step Guide on Sentiment Analysis with RNN and LSTM: https://medium.com/@lamiae.hana/a-step-by-step-guide-on-sentiment-analysis-with-rnn-and-lstm-3a293817e314

Text generation with LSTM

Similar to how Google search helps you in searches, LSTM models can be used to compete or generate sentences. Code below shows how LSTM model can be used to complete simple nursery rhymes.

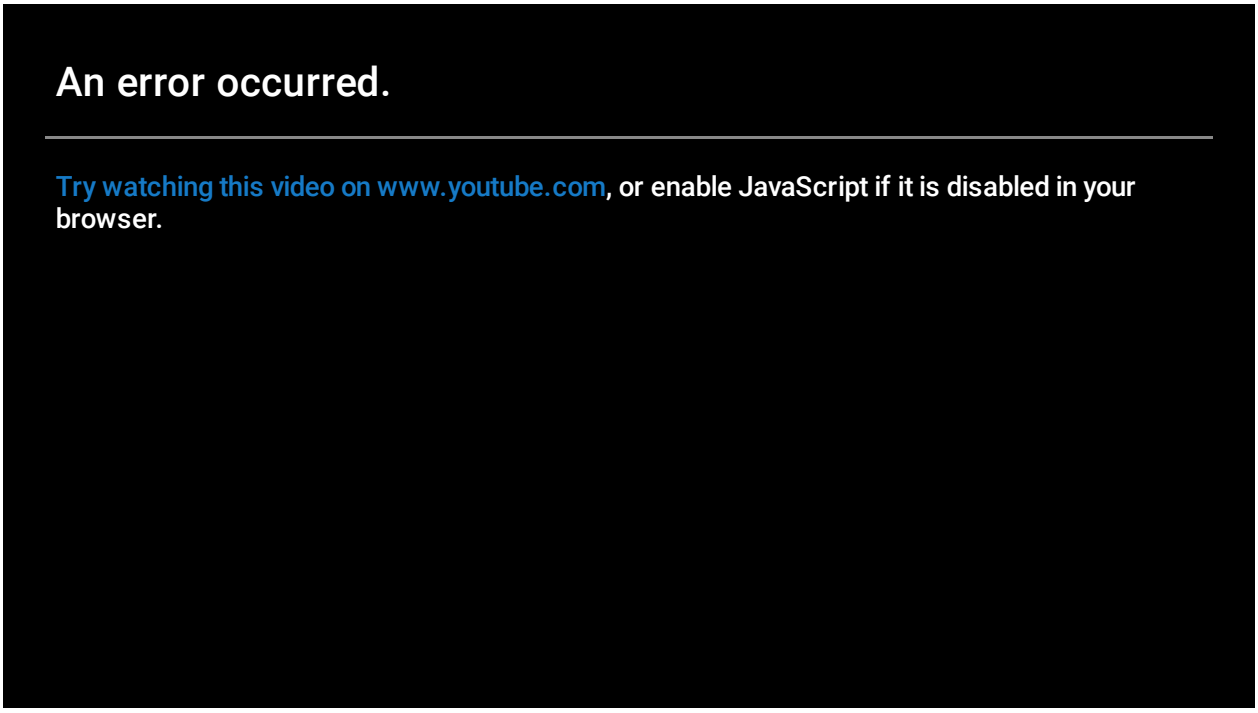
Run PYTHON

```python
from numpy import array
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding

# generate a sequence from a language model
def generate_seq(model, tokenizer, max_length, seed_text, n_words):
    in_text = seed_text
    # generate a fixed number of words
    for _ in range(n_words):
```

Code Source: https://machinelearningmastery.com/develop-word-based-neural-language-models-python-keras/

Some videos



Intel AI. (2016, July 14). *Recurrent Neural Networks* [online video]. Retrieved from https://www.youtube.com/watch?v=Ukgii7Yd_cU&feature=emb_title.

The below links provide tutorials on the application of LSTMs for text classification and advances in

LSTMs.

1. https://keras.io/examples/nlp/text_classification_from_scratch/
2. https://keras.io/examples/nlp/bidirectional_lstm_imdb/
3. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

References

1. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf
2. Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780. Retrieved from https://www.mitpressjournals.org/doi/pdfplus/10.1162/neco.1997.9.8.1735.
3. Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.https://www.researchgate.net/profile/Felix_Gers/publication/12292425_Learning_to_Forget_Continual_Prediction_with_LSTM/links/5759414608ae9a9c954e84c5/Learning-to-Forget-Continual-Prediction-with-LSTM.pdf
4. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222-2232. https://nichenjie.com/mdres/posts/2018/lstm/1503.04069.pdf
5. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. https://arxiv.org/pdf/1409.0473.pdf
6. Zhang, L., Wang, S., & Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1253. https://arxiv.org/ftp/arxiv/papers/1801/1801.07883.pdf

LSTM for time series prediction

LSTM for time series given below.

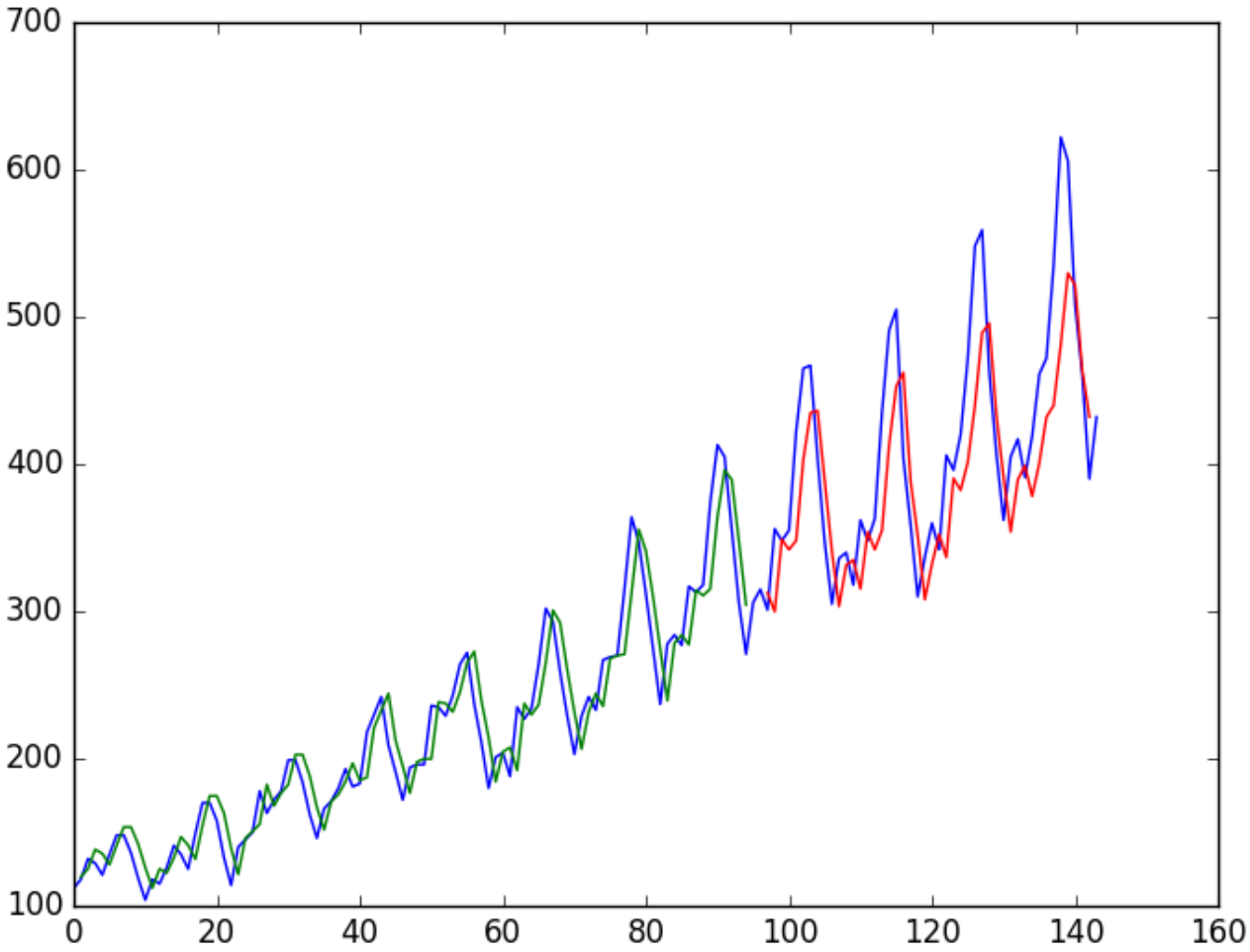


Figure and Data Source: https://github.com/jbrownlee/Datasets/blob/master/airline-passengers.csv
https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Note we need to first process the data and do embedding (windowing as done in Week 4) with code below. The function takes two arguments: the **dataset** and the **look_back** which is the number of previous time steps (window) to use as input variables to predict the next time period which is one step ahead prediction.

PYTHON

```python
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

Code Source: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Convolutional Neural Networks

Convolutional neural network (**CNN**, or **ConvNet**) is a type of deep learning model for machine learning and computer vision tasks such as image and video recognition, recommender systems, natural language processing, and financial time series. Below we see some key examples of computer vision tasks.

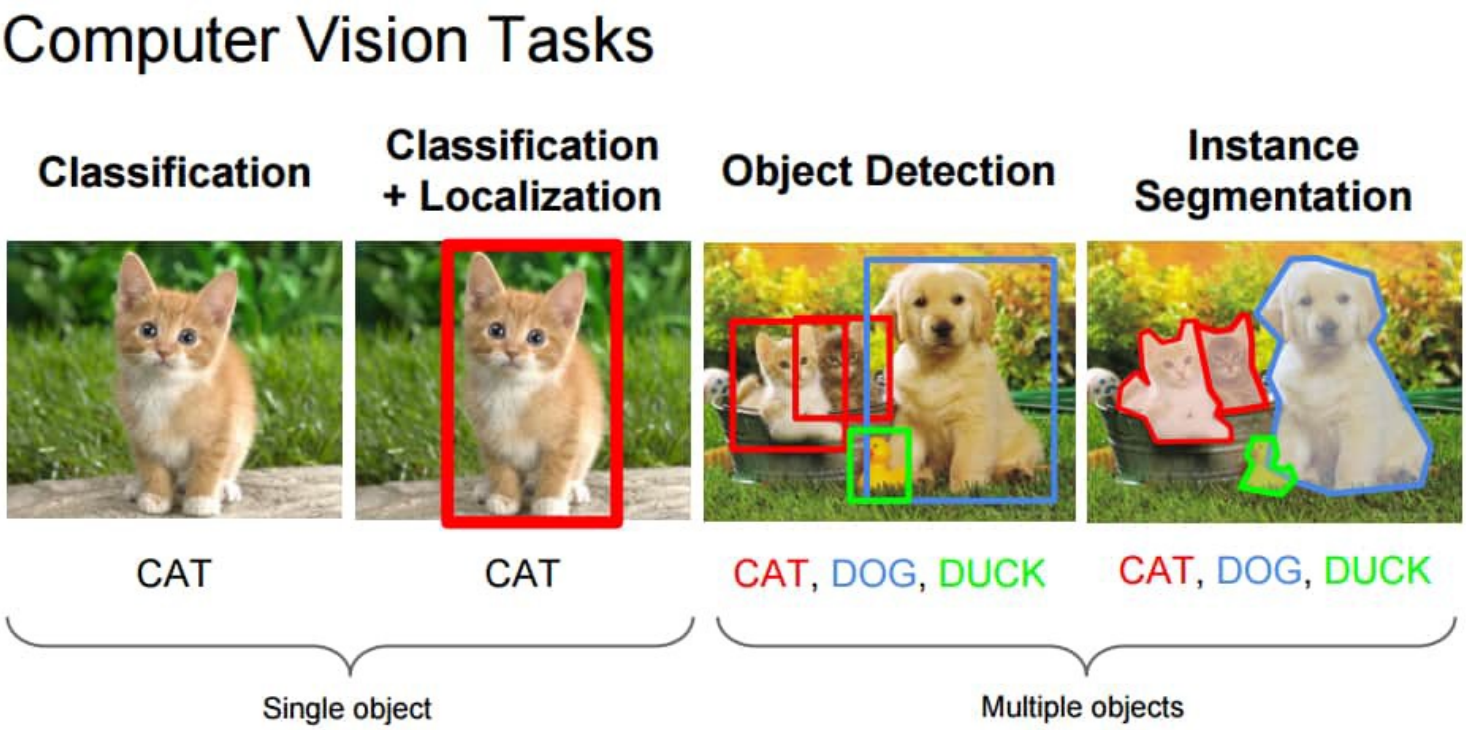


Figure Source: Example of computer vision tasks: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

CNN could be seen as a deep feedforward neural network that features automatic feature extraction from image or multi-media datasets.

The below figure highlights the convolutional layers of a CNN and how subsampling is done between which helps in automatic feature extraction from image datasets. This feature makes CNN a powerful method and has gained a lot of attention in the industry. Note the convolution and pooling layers.

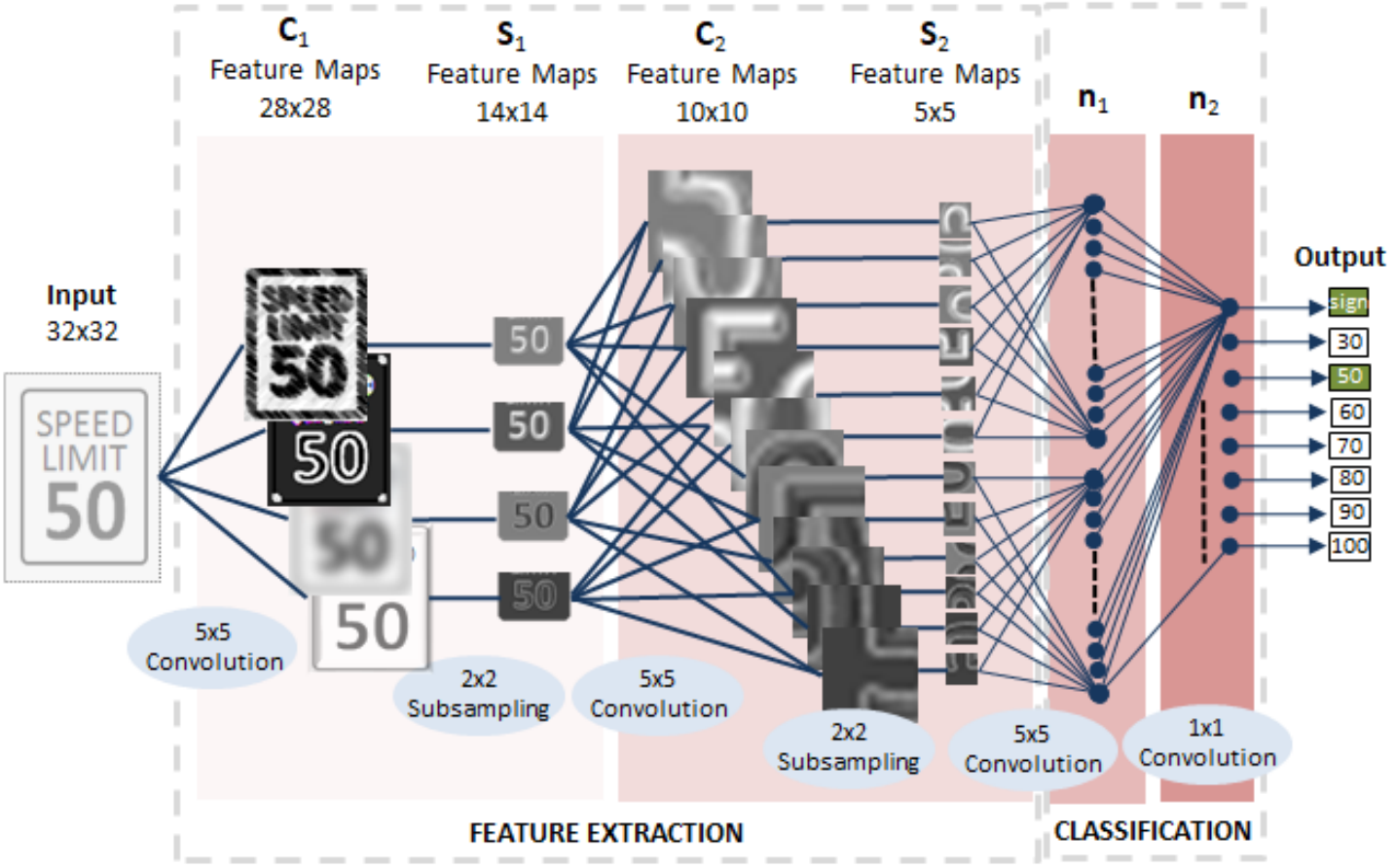


Figure: CNN example. Adapted from "AI in Automotive: Practical deep learning" by Texas Instruments, 2018 Retrieved from https://e2e.ti.com/blogs_/b/behind_the_wheel/archive/2018/02/08/ai-in-automotive-practical-deep-learning.

Note the convolution and pooling layers and how the raw image is processed.

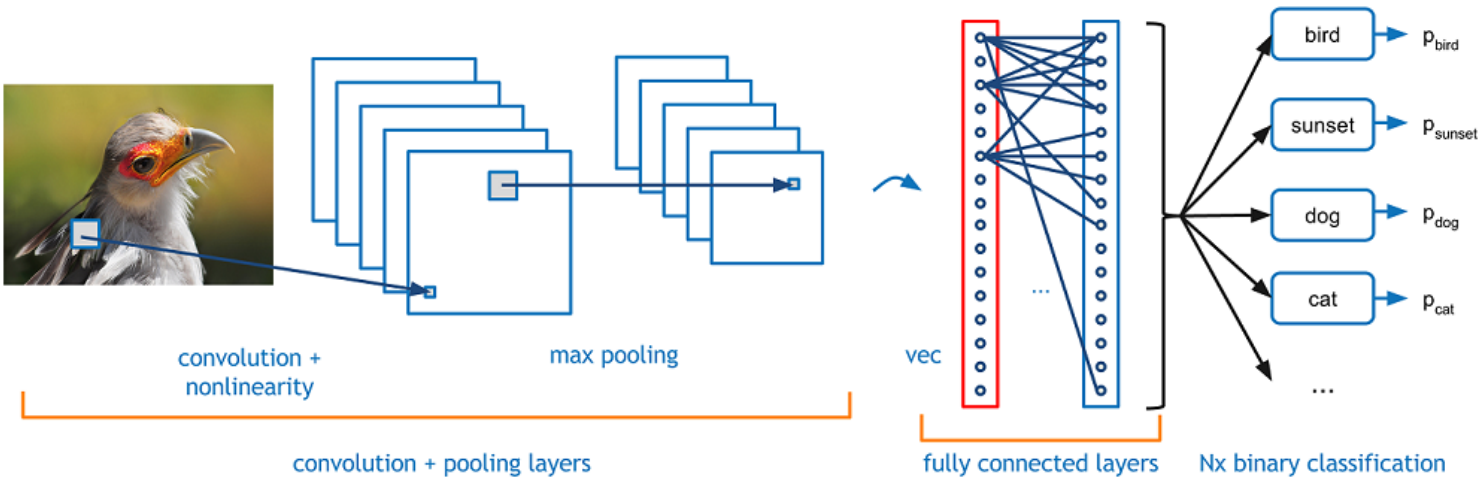
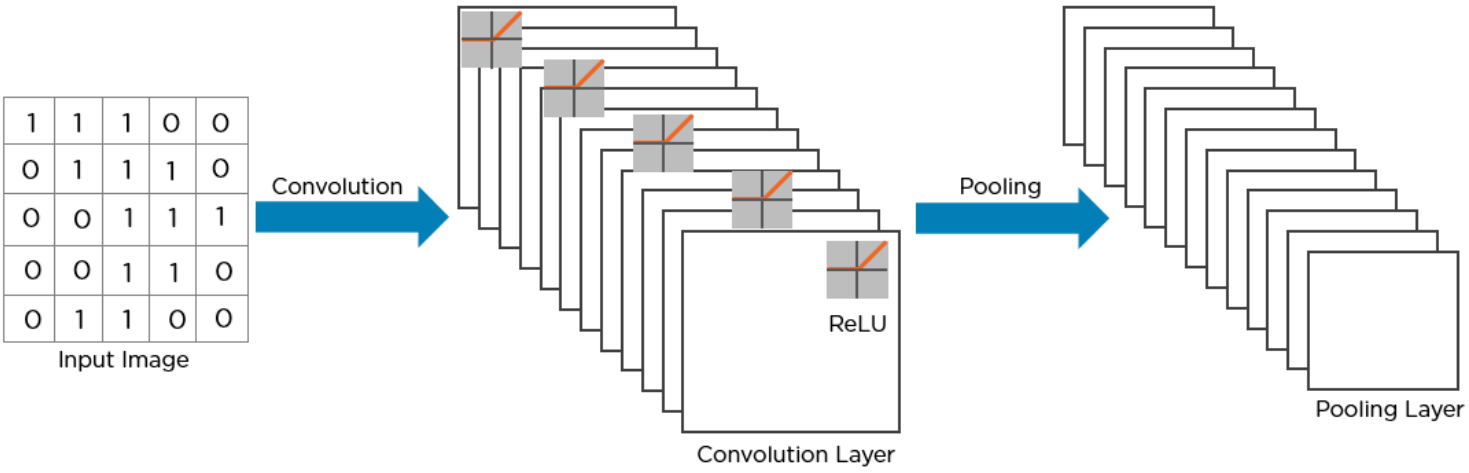


Figure Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Moving into finer details, we see how multiple convolutional layers are used for a single input (which

is a matrix) .



https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

Below is an example of a CNN processing data with convolution and pooling layers.

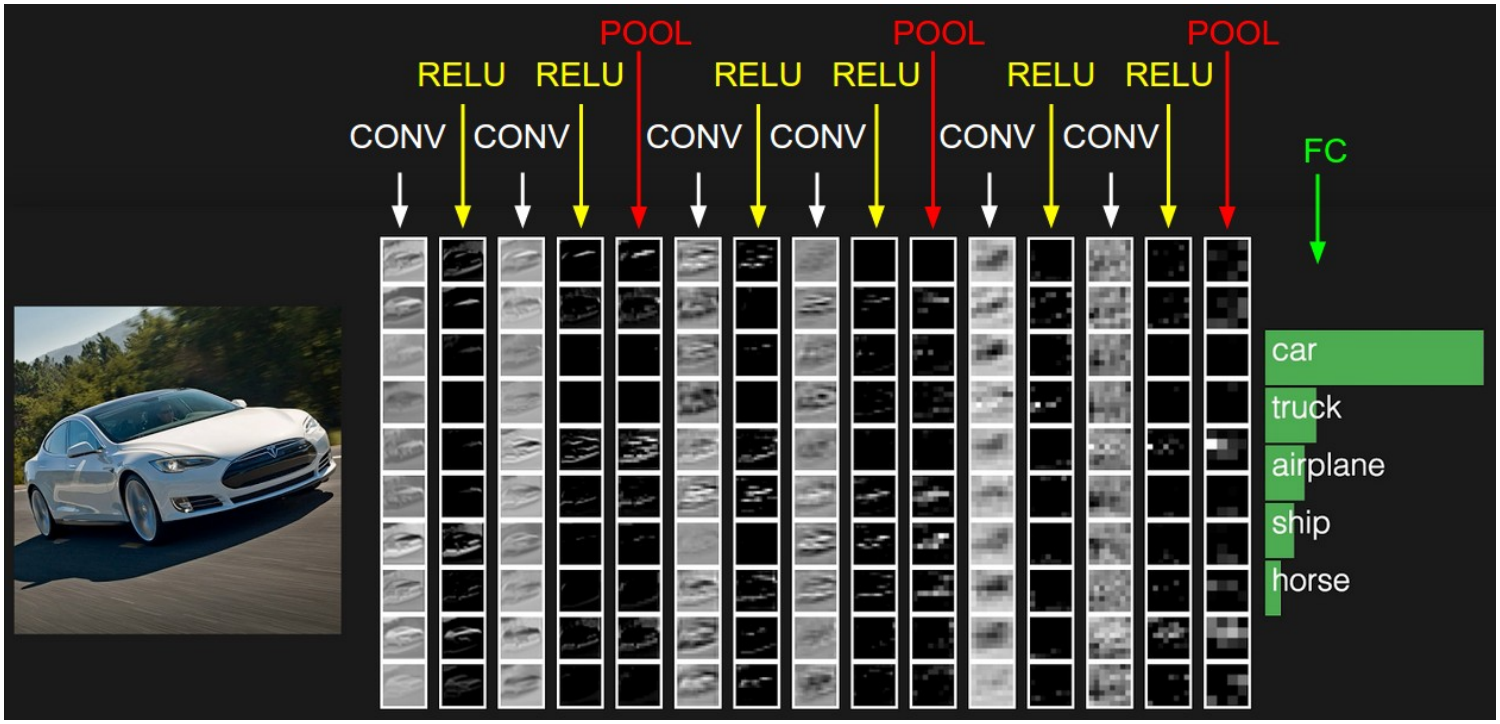


Figure Source: https://cs231n.github.io/convolutional-networks/

A RGB channel representing a photo is composed of three images where each image can store discrete pixels with conventional brightness intensities between 0 and 255.

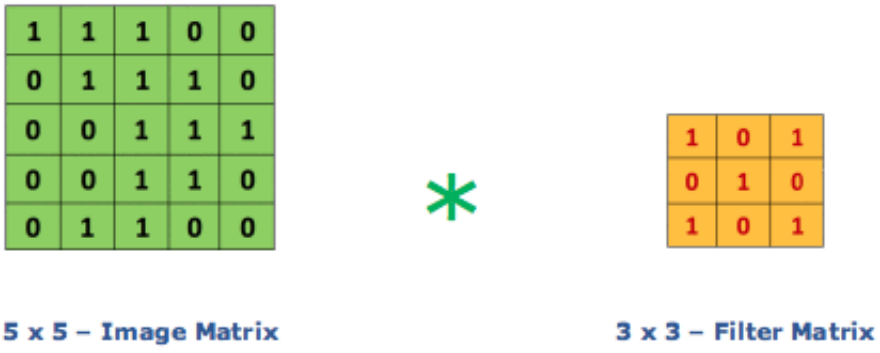
More info: https://en.wikipedia.org/wiki/Channel_(digital_image)

What is a convolution?

Given an input matrix ***A*** and a weight matrix ***K*** called a kernel, a convolution process will output a new matrix ***B***. The first element in *B* will be computed by

1. Taking the first *CxC* submatrix of *A*.
2. Multiplying each of its elements by its corresponding weight in *K*.
3. Adding all the products.

Step 2 and 3 are equivalent to flattening both *A*'s submatrix and *K*, and computing the dot product of the resulting vectors. The Figure below shows how we slide K to the right to get the next element, repeating this process for each of *A*'s rows.



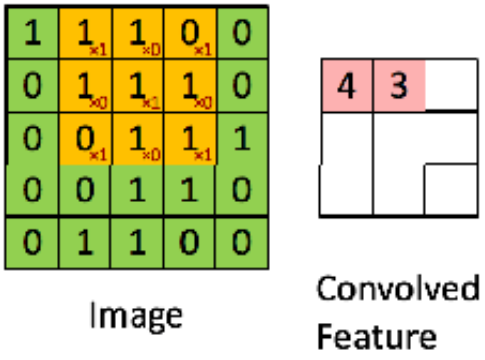
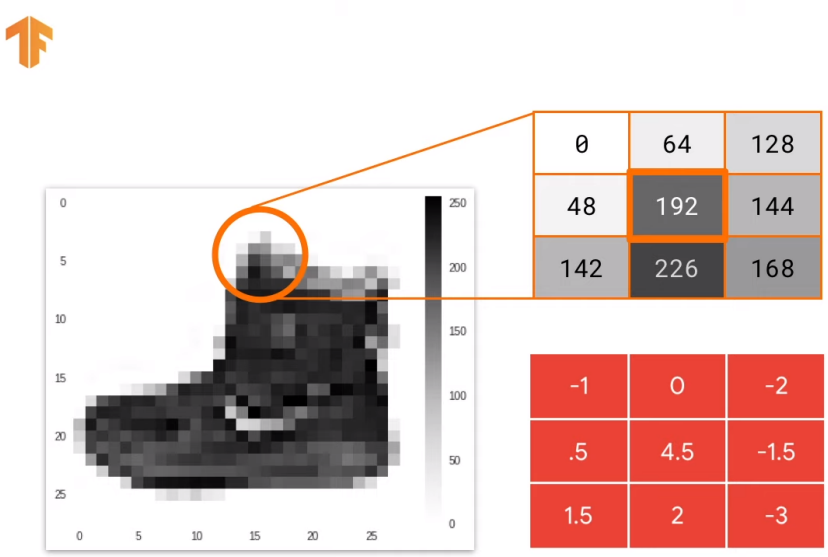


Figure Source. CNN: https://www.kdnuggets.com/2019/07/convolutional-neural-networks-python-tutorial-

tensorflow-keras.html

Below we see an example of how a filter is used.



0	64	128
48	192	144
142	226	168

Current pixel value is 192

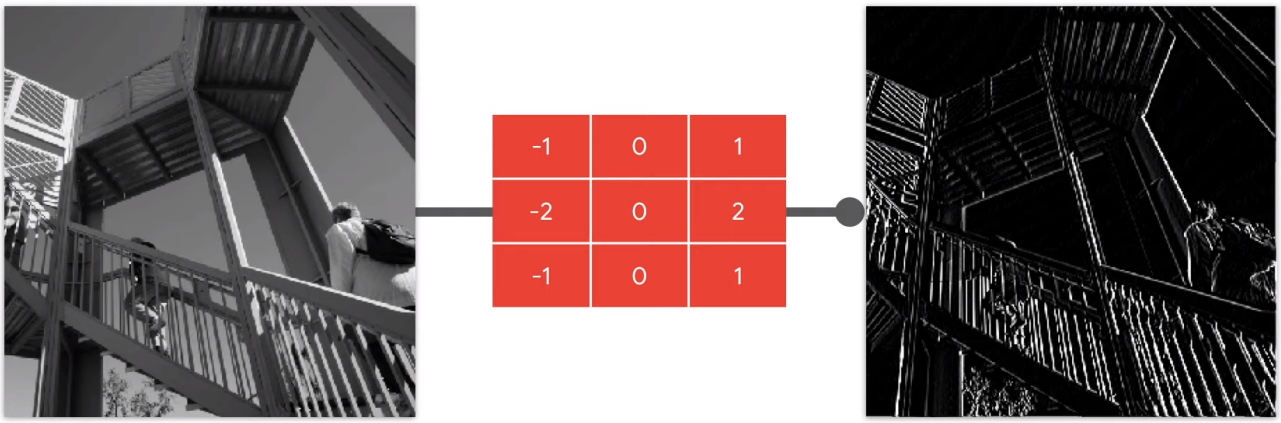
Consider neighbor values

-1	0	-2
.5	4.5	-1.5
1.5	2	-3

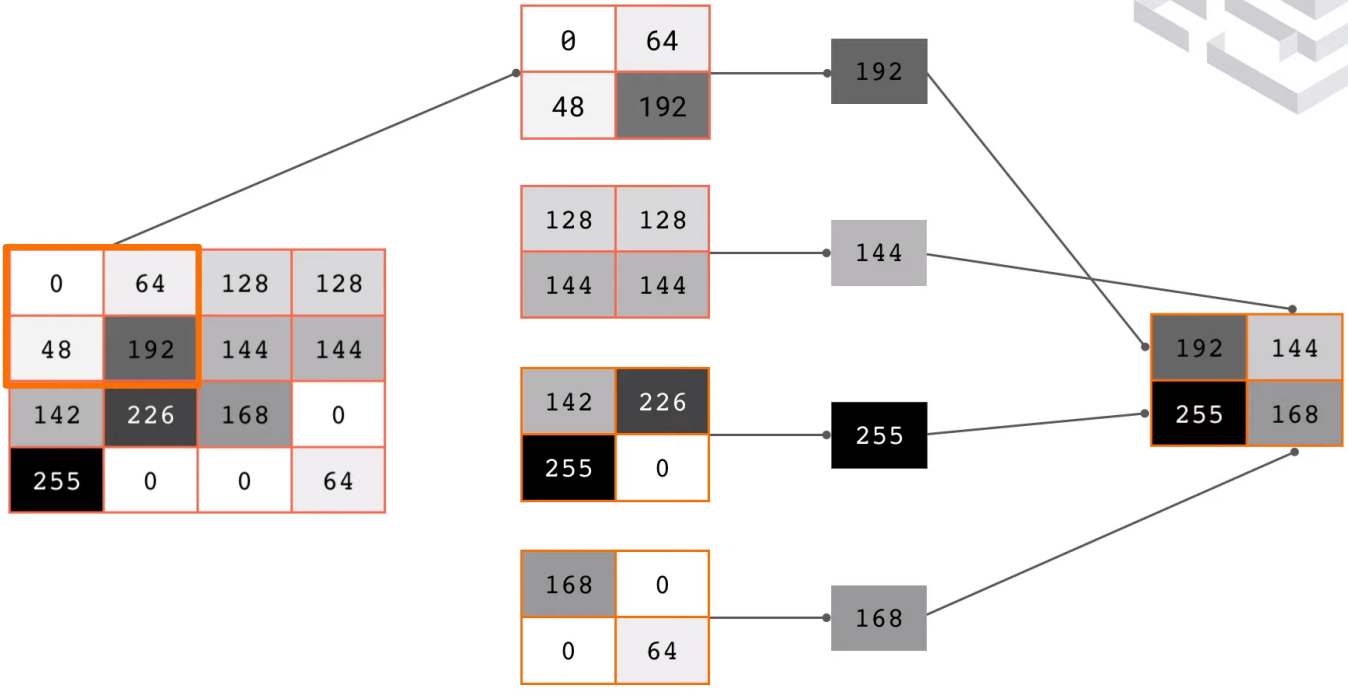
Filter definition

```
CURRENT_PIXEL_VALUE = 192
NEW_PIXEL_VALUE = (-1 * 0) + (0 * 64) + (-2 * 128) +
                  (.5 * 48) + (4.5 * 192) + (-1.5 * 144) +
                  (1.5 * 42) + (2 * 226) + (-3 * 168)
```

Below see how filters make a difference in actual images.



Next, we demonstrate the idea of max pooling and how it improves an image.



TensorFlow (2019, September 11). *Introducing convolutional neural networks (ML Zero to Hero - Part 3 and 4)* [online Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=x_VrgWTKkiM.

Below we see the case of Max pooling which further reduces the matrix.

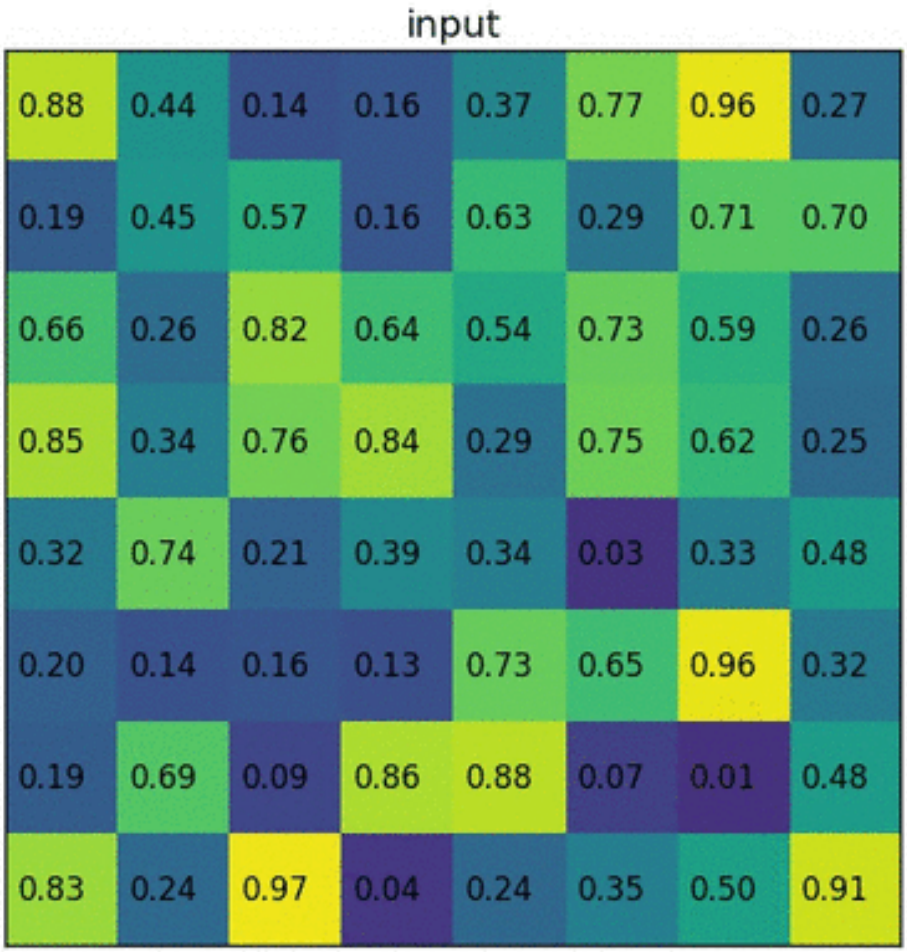


Figure Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

In keras, we can see these terms implemented.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

TensorFlow (2019, September 11). *Introducing convolutional neural networks (ML Zero to Hero - Part 3 and 4)* [online Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=x_VrgWTKkiM.

Below we see another demonstration of the entire operation.

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	2	1	2	0
0	1	0	1	0	2	0
0	1	0	2	0	2	0
0	0	2	0	0	2	0
0	0	0	1	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	2	2	0
0	1	0	0	2	0	0
0	1	0	2	0	0	0
0	2	1	0	1	2	0
0	1	0	1	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	2	0	1	0	2	0
0	0	2	1	0	1	0
0	0	0	1	1	1	0
0	0	2	0	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
1	1	1
0	1	1

w0[:,:,1]

1	1	0
-1	1	-1
-1	1	0

w0[:,:,2]

-1	1	0
0	1	0
1	0	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	-1	-1
0	1	-1
0	0	-1

w1[:,:,1]

-1	1	1
0	0	1
1	-1	0

w1[:,:,2]

0	-1	1
1	1	-1
1	1	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

6	5	4
11	6	14
4	8	9

o[:,:,1]

1	4	7
-7	6	-2
-1	2	2

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	2	1	2	0
0	1	0	1	0	2	0
0	1	0	2	0	2	0
0	0	2	0	0	2	0
0	0	0	1	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	2	2	0
0	1	0	0	2	0	0
0	1	0	2	0	0	0
0	2	1	0	1	2	0
0	1	0	1	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	2	0	1	0	2	0
0	0	2	1	0	1	0
0	0	0	1	1	1	0
0	0	2	0	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
1	1	1
0	1	1

w0[:,:,1]

1	1	0
-1	1	-1
-1	1	0

w0[:,:,2]

-1	1	0
0	1	0
1	0	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	-1	-1
0	1	-1
0	0	-1

w1[:,:,1]

-1	1	1
0	0	1
1	-1	0

w1[:,:,2]

0	-1	1
1	1	-1
1	1	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

6	5	4
11	6	14
4	8	9

o[:,:,1]

1	4	7
-7	6	-2
-1	2	2

toggle movement

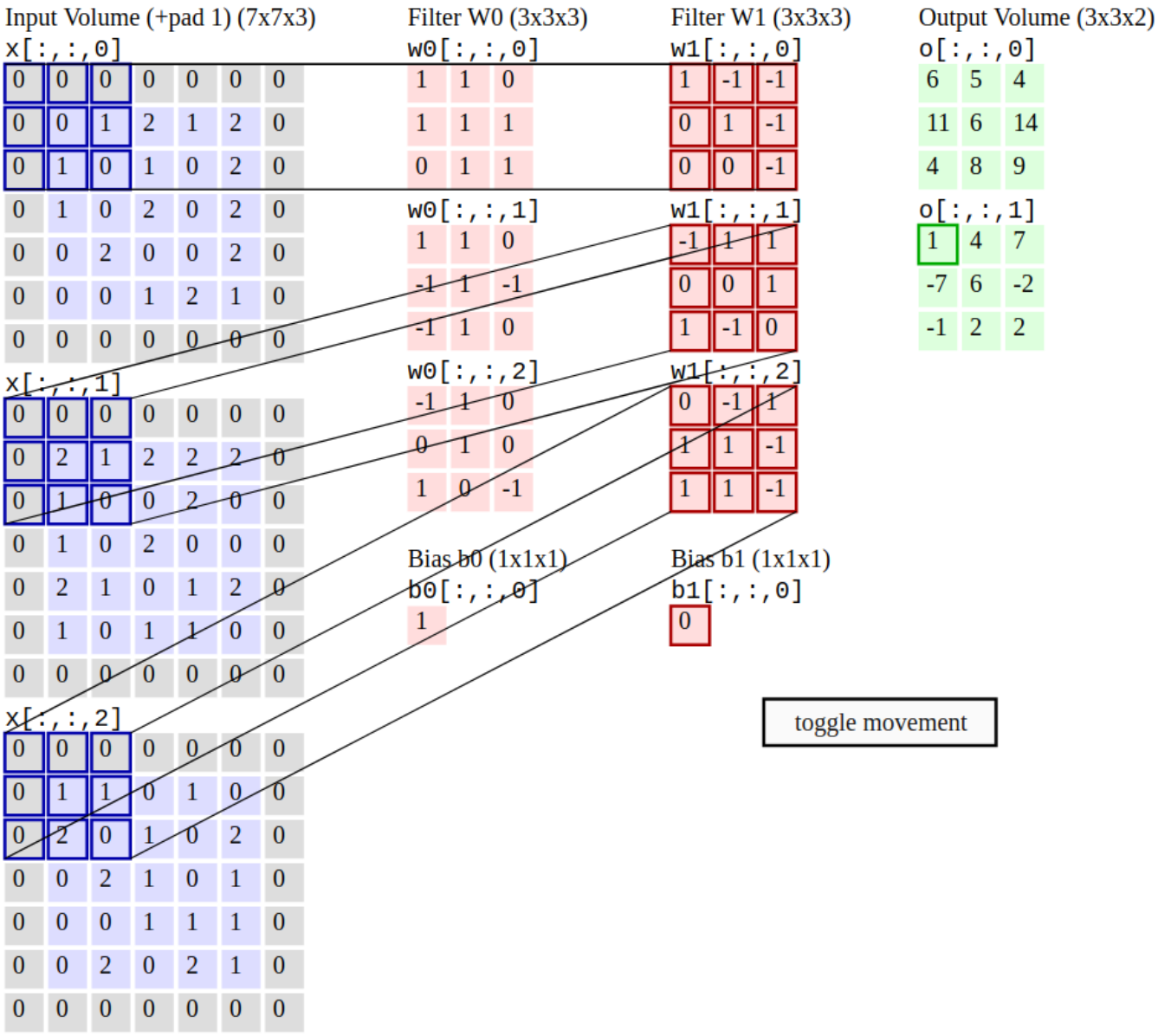


Figure. Screenshots taken from convolution animation: https://cs231n.github.io/convolutional-networks/

Live CNN Demo

We next see a live demo where the CNN trains in a browser and you can see the errors reducing with animation of the outputs. he demo features the CIFAR-10 dataset (object detection dataset) with Javascript.

Training Stats

pause

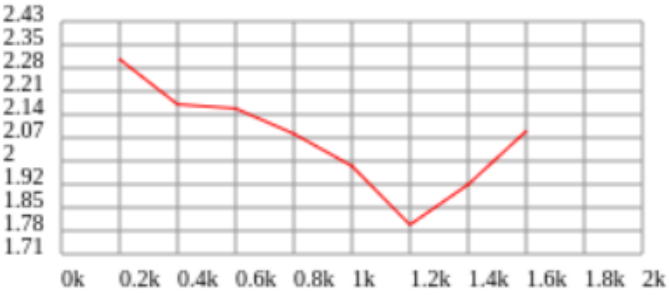
Forward time per example: 11ms
Backprop time per example: 19ms
Classification loss: 2.0248
L2 Weight decay loss: 0.00127
Training accuracy: 0.28
Validation accuracy: 0.31
Examples seen: 1789

Learning rate: 0.01 change
Momentum: 0.9 change
Batch size: 4 change
Weight decay: 0.0001 change

save network snapshot as JSON
init network from JSON snapshot

load a pretrained network (achieves ~80% accuracy)

Loss:



clear graph

Instantiate a Network and Trainer

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:32, out_sy:32, out_depth:3});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:4, l2_decay:0.0001});
```

change network

Network Visualization

input (32x32x3)
max activation: 0.49607, min: -0.5
max gradient: 0.02656, min: -0.05282

Activations:

relu (8x8x20)
max activation: 1.77512, min: 0
max gradient: 0.20934, min: -0.25226
Activations:
Activation Gradients:

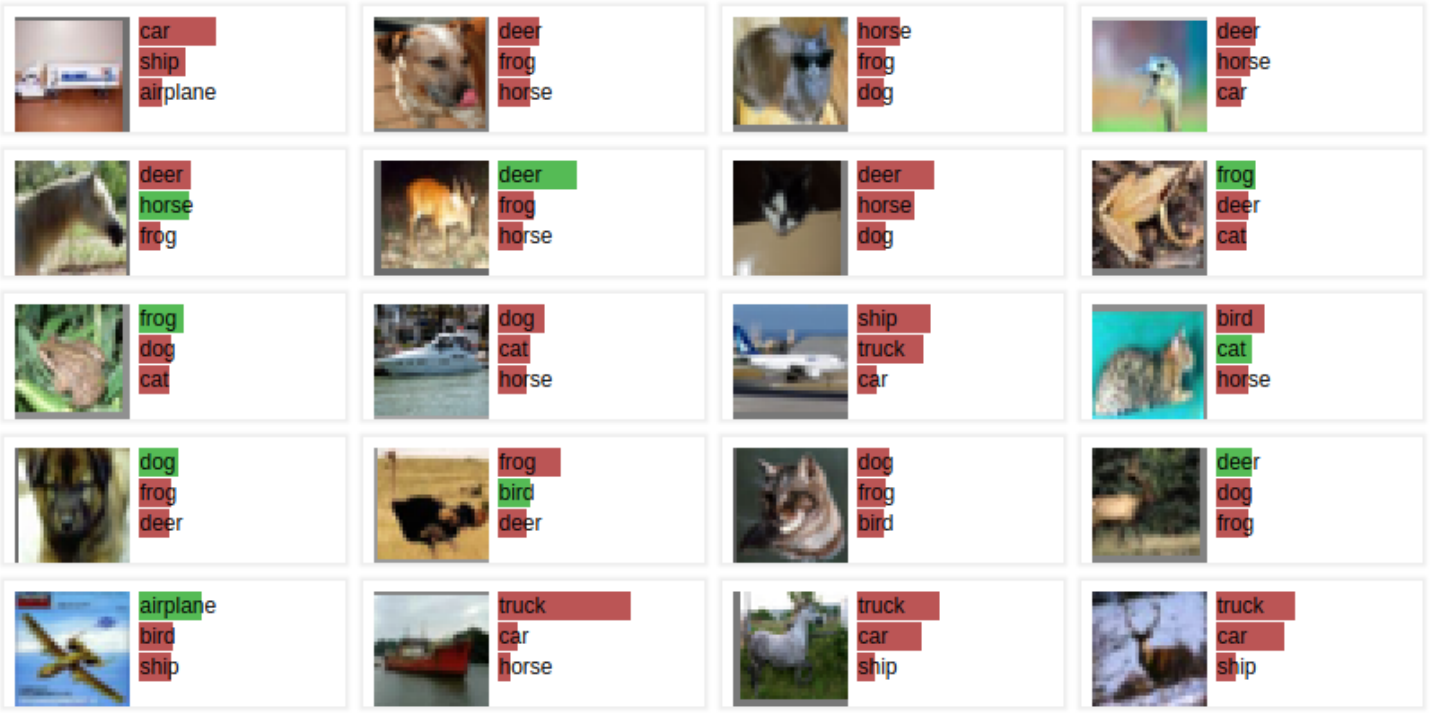
pool (4x4x20)
pooling size 2x2, stride 2
max activation: 1.77512, min: 0
max gradient: 0.20934, min: -0.25226
Activations:
Activation Gradients:

fc (1x1x10)
max activation: -2.138, min: -4.13538
max gradient: 0.26779, min: -0.9001
parameters: 10x320+10 = 3210
Activations:
Activation Gradients:

softmax (1x1x10)
max activation: 0.26779, min: 0.03633
max gradient: 0, min: 0
Activations:

Example predictions on Test set

test accuracy based on last 200 test images: 0.21428571428571427



CNN Live Demo: " The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags." https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

We note that there are many combinations of different parameters in Convolution and pooling layers and they all affect the results. Since there are benchmark datasets, many researchers share their

training CNN model weights online which are further used for comparison and improvement. Below link gives an overview of different CNN architectures and their model performance for known datasets. This helps researchers and engineers develop a better model.

CNN model benchmarks: https://github.com/soumith/convnet-benchmarks

Next we see code example using Keras and Python for MNIST character recognition dataset that we have seen in earlier lessons. Note that due to Ed memory limitations, you cant run this but you can try in your own computer. Try adding and removing a different set of convolutional layers and see the difference in performance.

Run PYTHON

```
import keras
from keras import backend as K
from keras.datasets import mnist
from keras.models import Sequential
from keras.models import model_from_yaml
from keras.utils import to_categorical
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten


# A "class" for each digit from 0-9
num_classes = 10
# How many times to run through the data while training
num_epochs = 1
# Input (image) dimensions
```

Source: https://github.com/HackBinghamtonArchives/CNN-Demo

Next, we see the same problem with Keras in R.

```r
library(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

#The x data is a 3-d array (images,width,height) of grayscale values.
#To prepare the data for training we convert the 3-d arrays into matrice
#by reshaping width and height into a single dimension (28x28 images
#are flattened into length 784 vectors). Then, we convert the grayscale
#values from integers ranging between 0 to 255 into floating point value
# ranging between 0 and 1:

```

Code Source: https://tensorflow.rstudio.com/guide/keras/

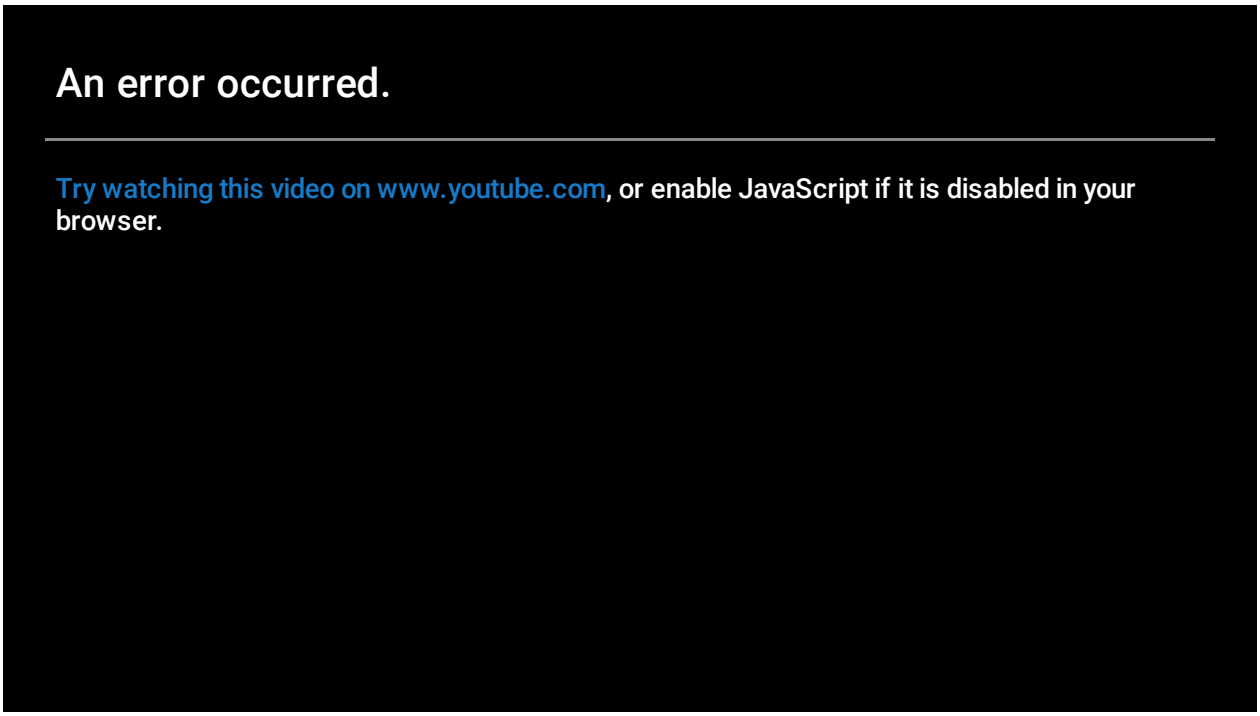
CNN for Time Series

We note that CNNs can also be used for time series forecasting. We do not have enough time to cover it, but the tutorial below gives more information with code examples.

https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/

https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/

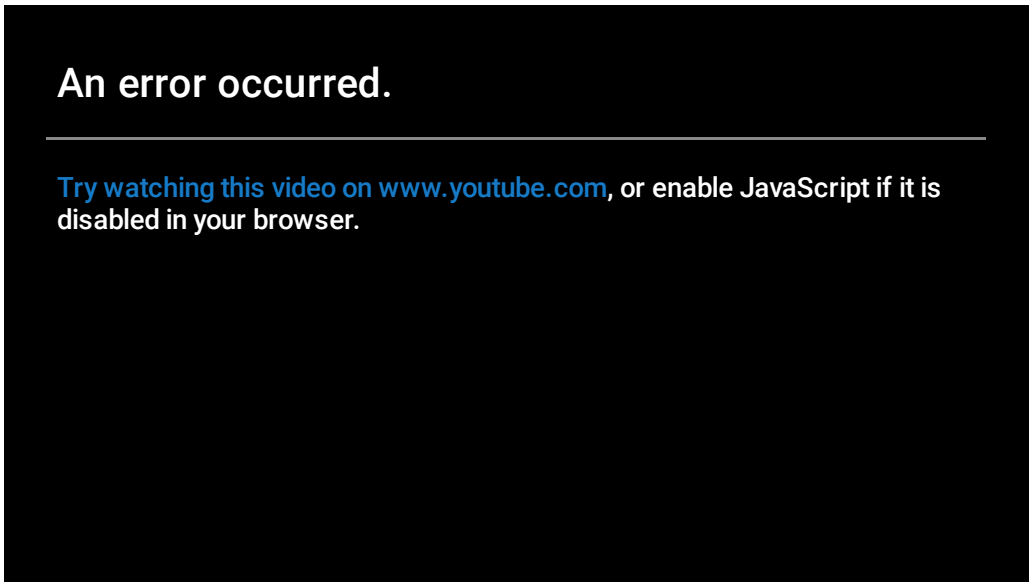
Introducing convolutional neural networks



TensorFlow (2019, September 11). *Introducing convolutional neural networks (ML Zero to Hero - Part 3)* [online Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=x_VrgWTKkiM.

Watch the following video on machine learning for web applications using TensorFlow.

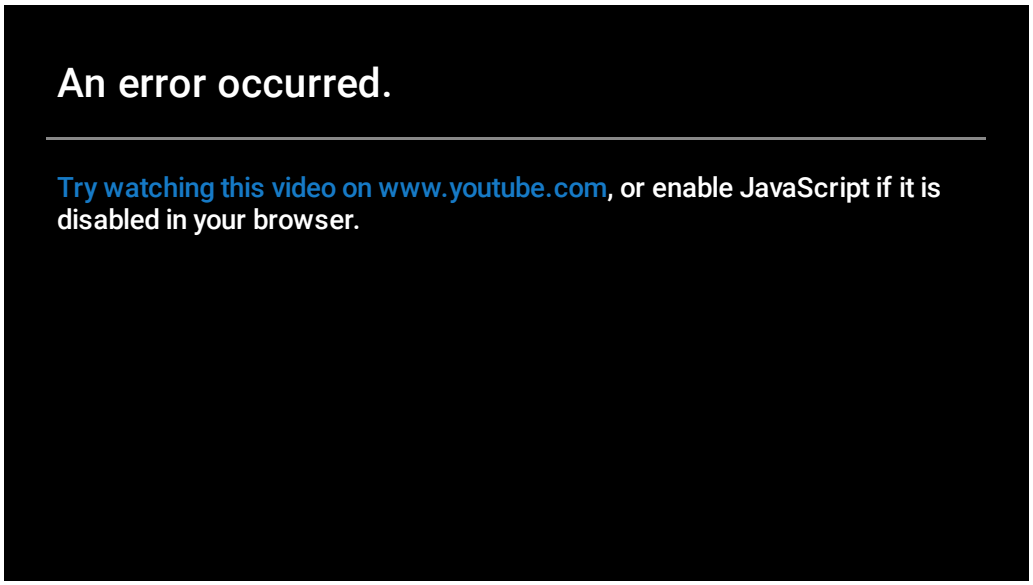
Machine learning magic for your web application



Google Chrome Developers (2019, November 25). *Machine Learning magic for your web application with TensorFlow.js (Chrome Dev Summit 2019)* [online video]. YouTube. Retrieved from https://www.youtube.com/watch?v=FDUfaYsFQrc&feature=emb_title.

Watch the following video to learn more about deep learning and neural networks with JavaScript.

Learn TensorFlow.js - deep learning and neural networks with JavaScript



freeCodeCamp.org (2018, November 13). *Learn TensorFlow.js - Deep Learning and Neural Networks with JavaScript* [online video]. Retrieved from https://www.youtube.com/watch?v=EoYfa6mYOG4&feature=emb_title.

Finally, here are some code examples using Keras that you may find useful. https://www.programcreek.com/python/example/104283/keras.optimizers.RMSprop

References

1. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541-551. http://www.ics.uci.edu/~welling/teaching/273ASpring09/lecun-89e.pdf
2. LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995. http://www.iro.umontreal.ca/~lisa/pointeurs/handbook-convo.pdf
3. Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, *8*(1), 98-113. https://ieeexplore.ieee.org/document/554195
4. Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*. https://arxiv.org/pdf/1404.2188.pdf
5. Liang, M., & Hu, X. (2015). Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3367-3375).http://openaccess.thecvf.com/content_cvpr_2015/papers/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.pdf
6. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems* (pp. 649-657).

https://www.researchgate.net/profile/Xiang_Zhang20/publication/281607724_Character-level_Convolutional_Networks_for_Text_Classification/links/56097c5e08ae1396914a1af7/Character-level-Convolutional-Networks-for-Text-Classification.pdf

7. Varior, R. R., Haloi, M., & Wang, G. (2016, October). Gated siamese convolutional neural network architecture for human re-identification. In *European conference on computer vision* (pp. 791-808). Springer, Cham. https://arxiv.org/pdf/1607.08378
8. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90. https://dl.acm.org/doi/pdf/10.1145/3065386

Exercise 1

Either using R or Python via Keras, use LSTM model and apply to one-step and five-step ahead prediction for Sunspot and Lazer datasets given in Week 4 lessons. Use D=5 and T=1 for windowing via Taken's theorem.

Report:

1. RMSE of train and test datasets
2. Prediction plot and residual errors for each dataset.
3. Compare with FNN-Adam and discuss your results.

Use LSTM to develop language model as shown in example.

Use CNN to train the MNIST dataset as shown in example.

Recommender systems

Recommender systems are very widely used applications of machine learning. The basic idea is relatively simple, but the way in which it works is a bit different from other machine learning methods we discussed so far. We consider the matrix operations used for recommender systems using matrix factorization.

1. Recommender systems are a form of **personalisation**, for example: "person who liked *x* may also like *y*"
2. The basic idea is related to instance-based learning. It uses a similarity function.
3. Other forms of learning may be used to model user choices based on content.
4. Netflix Prize transformed the field-methods based matrix decomposition to find **latent factors**.
5. If the data is enough, the current methods use deep learning to find latent factors.

A framework for recommendation

Example: movie rating matrix, where each entry represents user *c* rating movie *s*.

We are given a utility function to represent the "value" for a user *c* receiving a recommendation for some item *s*:

	K-PAX	Life of Brian	Memento	Notorious
Alice	4	3	2	4
Bob	∅	4	5	5
Cindy	2	2	4	∅
David	3	∅	5	2

$$\text{utility} \qquad u: \ c \times s \ \mapsto \ R$$

The recommendation problem can be formalised as:

$$\forall c \in C, \text{choose} \qquad s'_c = \arg\max_{s \in S} u(c, s)$$

This is learning in the sense of requiring **extrapolation** to predict the unknown values of the utility function.

Alternatively, it can be viewed as **matrix completion**, i.e., fill in the missing ratings for users and

movies.

Content-based recommendation

User *c* is recommended items *s* that are similar *to* past choices. The idea comes from **information retrieval** and requires a profile of the content, or description of items

$$u(c, s) = score(ContentBasedProfile(c), Content(s))$$

E.g.,

$$u(c, s) = \cos(\vec{w}_c, \vec{w}_s) = \frac{\vec{w}_c \cdot \vec{w}_s}{\|\vec{w}_c\| \times \|\vec{w}_s\|}$$

where $\vec{w}_c$ and $\vec{w}_s$ are vectors of the content terms summarising *c*'s choices, and the most relevant terms describing the content of *s*, respectively.

Advantages:

- well-understood techniques from information retrieval
- an extract latent features from text analysis

Disadvantages:

- may not have content, or maybe limited or sparse
- over-specialisation: recommendations given for known types only.

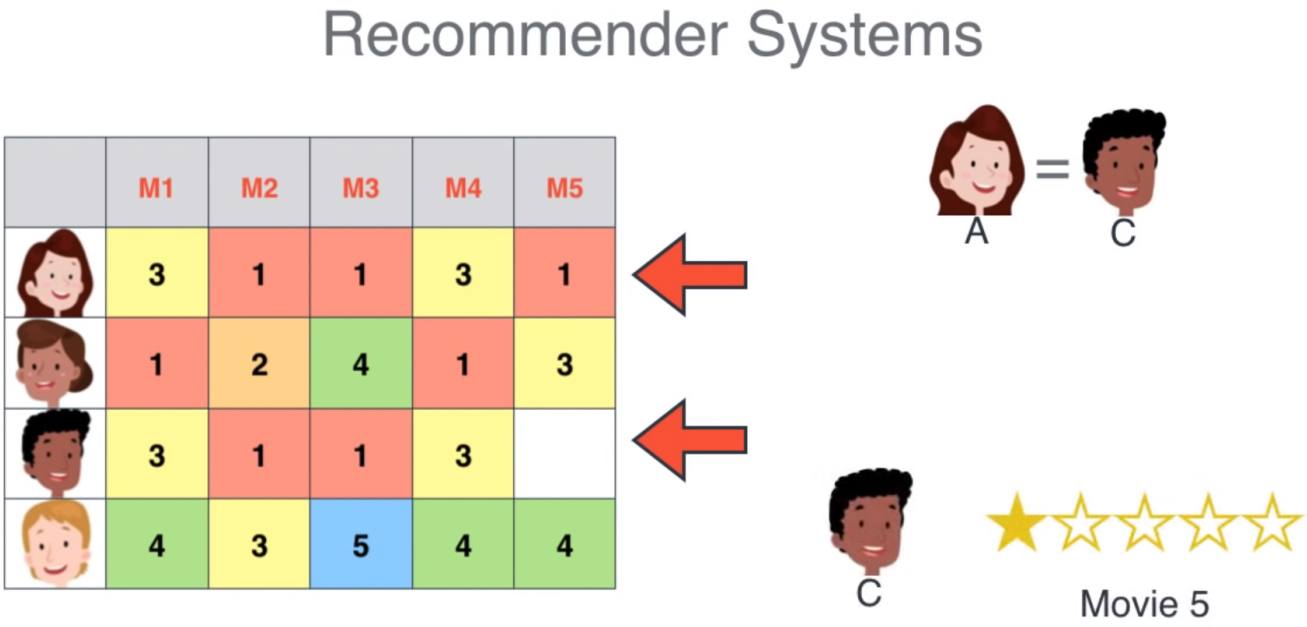


Figure Source. How Recommender Systems Work (Netflix/Amazon): https://www.youtube.com/watch?v=n3RKsY2H-NE&feature=emb_logo

Collaborative-based recommendation

User *c* is recommended items that users with similar tastes have chosen. This is also known as **collaborative filtering** (CF) and is similar to Amazon-style recommender systems. This is behind the message "Customers Who Bought This Item Also Bought ..." that is familiar to online shoppers. There are two main methods: **memory-based**, and **model-based** CF.

Memory-based CF

Predict unknown rating $r_{c,s}$ of user *c* for item *s* by aggregating the ratings of *N* users *c'* most similar to *c* who have previously rated *s*:

$$r_{c,s} = k \sum_{c' \in C} \mathrm{sim}(c, c') \times r_{c',s}$$

where

- *k* is just a normalising factor, and
- the similarity function $\mathrm{sim}()$ can be correlation, cosine distance, etc. on the vector of items rated (or bought) by users.

Alternatively, **item-based** similarity can be used (Amazon).

Model-based CF

Memory-based CF is like a nearest-neighbour method. A big problem with this approach is sparsity – to address this, often try to find a low-rank approximation to the matrix (i.e., finding smaller user-feature and movie-feature matrices) using a form of stochastic gradient descent. Alternatively, you can use other machine learning methods to build a model to predict directly the unknown rating $r_{c,s}$ from examples in the database. E.g., Naive Bayes-type approaches. This is called model-based CF.

Advantages of CF:

- works well in practice
- does not require content (descriptions).

Disadvantages of CF:

- new user problem: must do some rating to get recommendations
- new item problem: must be rated to be used in recommendations
- **grey sheep**: insufficiently individual!

- **black sheep**: too individual!

Hybrid recommender systems

Key idea: combine model-based and memory-based approaches. For example:

- **cold-start problem**: use model to predict before user activity
- **sparsity problem**: use model to predict missing values.

However, a challenge is that learning the models may be difficult or expensive.

Matrix factorization

Matrix factorization enables to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

Matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities which can be applied for building models such as recommender systems.

Looking for structure in the rating matrix.

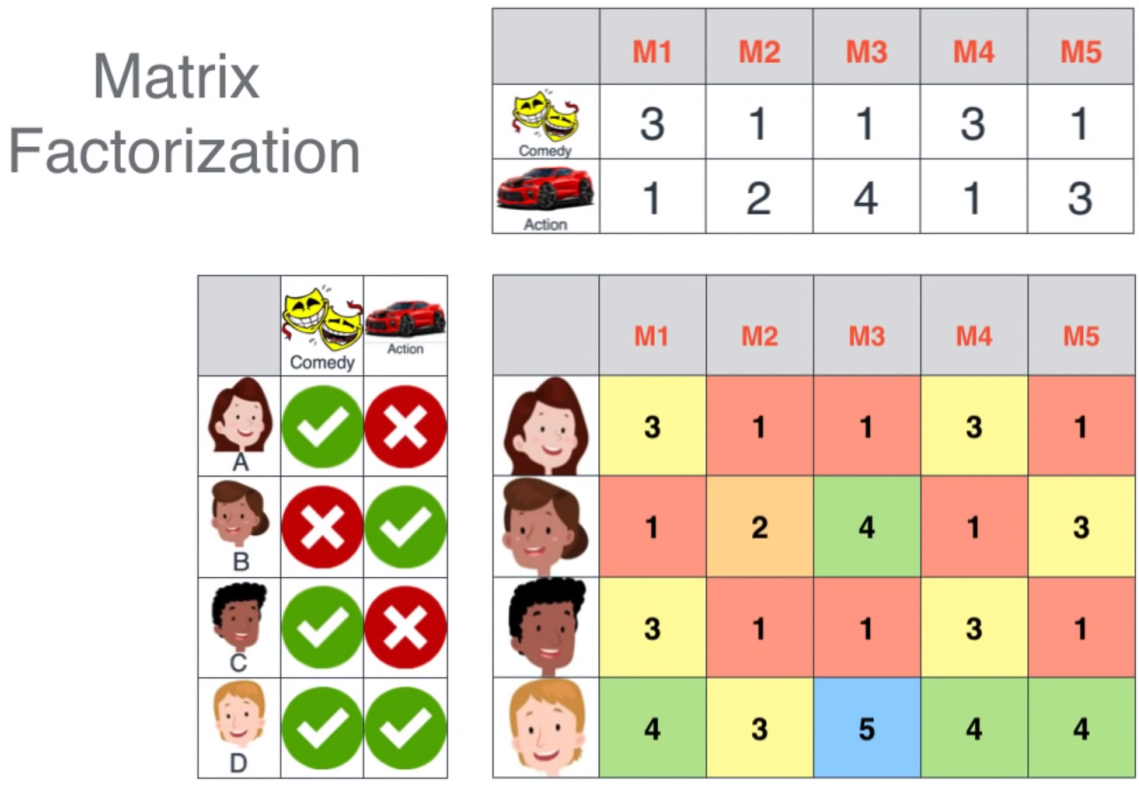


Figure. How does Netflix recommend movies? Matrix Factorization: https://www.youtube.com/watch?v=ZspR5PZemcs&feature=emb_logo

In a matrix form, we have another example below with a different example for rating. Imagine these represent ratings by six different people (in rows), on a scale of 0 to 3, of four different movies; e. g. The Shawshank Redemption, The Usual Suspects, The Godfather, and The Big Lebowski (in columns, from left to right). The Godfather seems to be the most popular of the four with an average rating of 1.5, and The Shawshank Redemption is the least appreciated with an average rating of 0.5.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix}$$

We can **factorise** the matrix and see what is revealed.

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The original matrix on the left-hand side is equal to the product of 3 new matrices on the right-hand side.

- The right-most matrix associates movies (in columns) with genres (in rows): The Shawshank Redemption and The Usual Suspects belong to two different genres, say drama and crime, The Godfather belongs to both, and The Big Lebowski is a crime film and also introduces a new genre (say comedy).
- The tall, 6 x 3 left-most matrix then expresses people's preferences in terms of genres.
- Finally, the middle 3 x 3 square matrix states that the crime genre is twice as important as the other two genres in terms of determining people's preferences.

Below we further describe the process of adjusting values in the given matrices using gradient descent.

We note that a learning rate and regularisation parameter is used via gradient descent.

Given a set of U users and set of D items. Let R of size |U|×|D| be the matrix that contains all the ratings that the users have assigned to the items.

We would like to discover K latent features. Our task is to find two matrics P (of size |U|×|K|) and Q (of size |D|×|K|) such that their product approximates R as given below.

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

	M1	M2	M3	M4	M5
F1	1.2	3.1	0.3	2.5	0.2
F2	2.4	1.5	4.4	0.4	1.1

	F1	F2
A	0.2	0.5
B	0.3	0.4
C	0.7	0.8
D	0.4	0.5

	M1	M2	M3	M4	M5
	1.44	1.37	2.26	0.7	0.59
	1.32	1.53	1.85	0.91	0.5
	2.76	3.37	3.73	2.07	1.02
	1.68	1.99	2.32	1.2	0.63

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

Figure. How does Netflix recommend movies? Matrix Factorization: https://www.youtube.com/watch?v=ZspR5PZemcs&feature=emb_logo

Each row of P represents the strength of the associations between a user and the features, and each row of Q represents the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we compute the dot product of their vectors.

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{K} p_{ik} q_{kj}$$

Next, we have to find a way to obtain P and Q. We first initialize the two matrices with some values, and then calculate how different their product is to M, and then try to minimize this difference iteratively using gradient descent.

The error between the estimated and the real rating is computed for each user-item pair.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

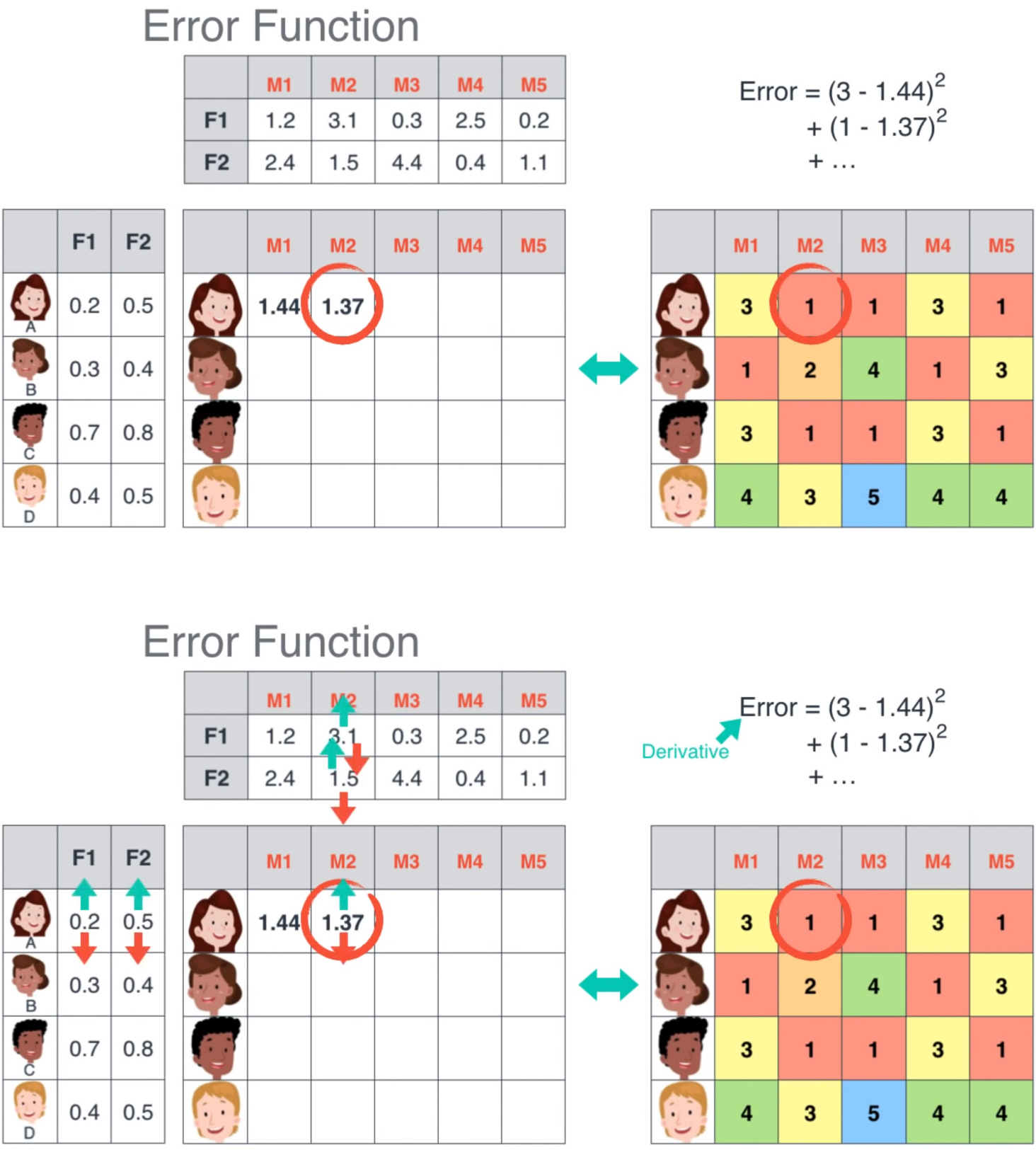


Figure. How does Netflix recommend movies? Matrix Factorization: https://www.youtube.com/watch?v=ZspR5PZemcs&feature=emb_logo

We consider the squared error because the estimated rating can be either higher or lower than the real rating. We use gradient descent to minimize the error and therefore, we differentiate the above equation with respect to these two variables separately as shown below.

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

Once we get the gradient, we can update the values with the following (as done in case of neural networks).

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

where α is the learning rate chosen by user. We can also add regularisation (similar to case of neural networks) to avoid overfitting.

Takács, G., Pilászy, I., Németh, B., & Tikk, D. (2008, October). Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems* (pp. 267-274).

We see the python code below.

Run PYTHON

```
# Created by Albert Au Yeung (2010)
#
# An implementation of matrix factorization
import numpy


##############################################################################
"""
@INPUT:
    R     : a matrix to be factorized, dimension N x M
    P     : an initial matrix of dimension N x K
    Q     : an initial matrix of dimension M x K
    K     : the number of latent features
```

Code Source: hhttps://albertauyeung.github.io/2017/04/23/python-matrix-factorization.html

Deep Learning for Recommender Systems

Now that we have seen how matrix factorization for recommender system works, let's see how deep learning can be used to enhance it further.

Neural collaborative filtering (NCF) implements a recommender system by replacing the inner product with a neural architecture that can learn an arbitrary function from data.

In NCF modelling, a multi-layer perceptron to learn the user-item interaction function and extensive experiments on two real-world datasets showed significant improvements over the state-of-the-art methods. Consider the framework below.

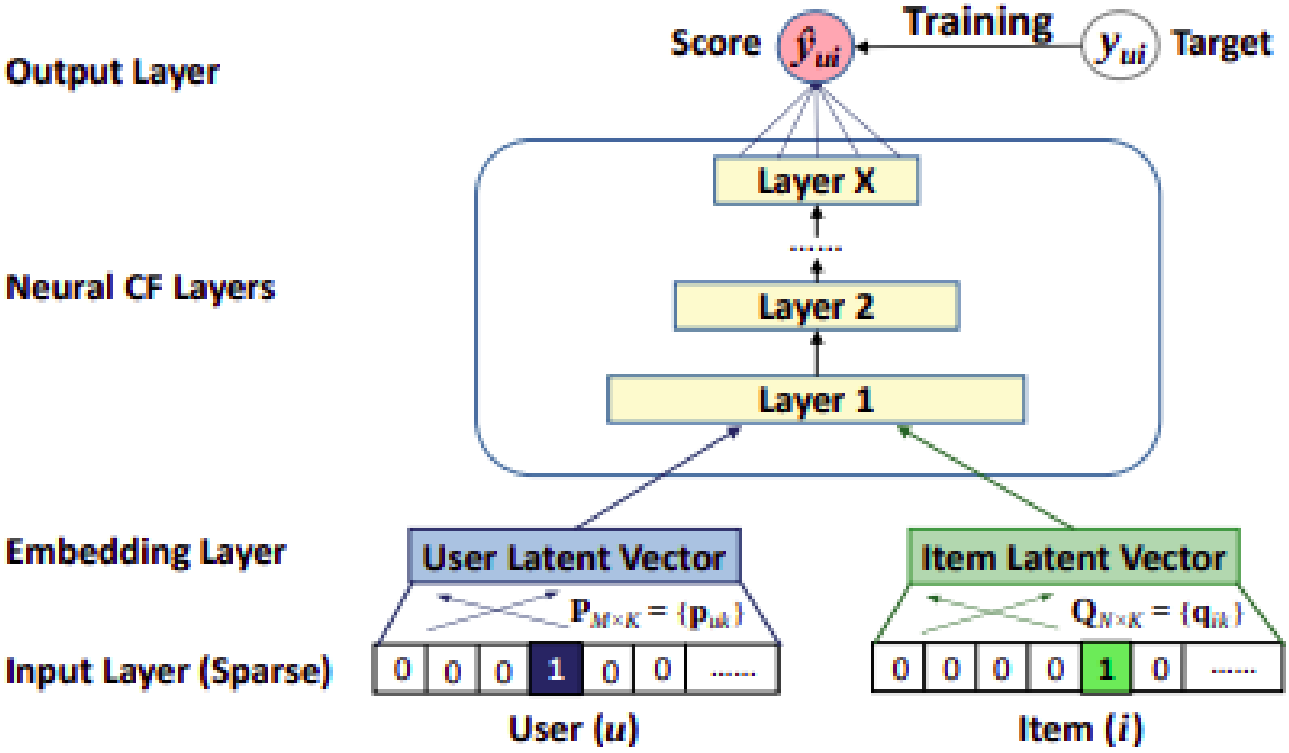


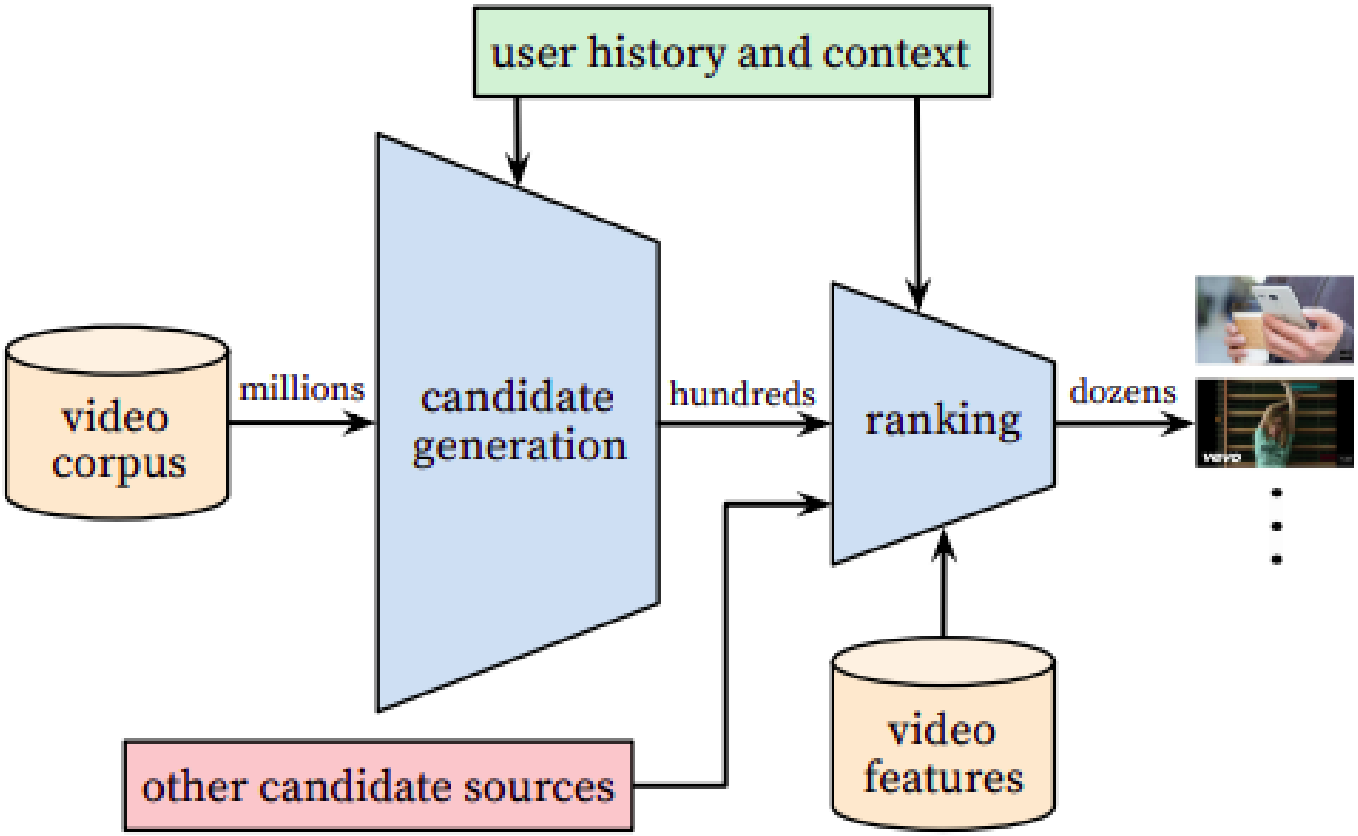
Figure 2: Neural collaborative filtering framework

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173-182)
http://cseweb.ucsd.edu/classes/fa17/cse291-b/reading/p173.pdf

Let's consider the case of using deep learning for Youtube recommender system. There will be mainly three challenges:

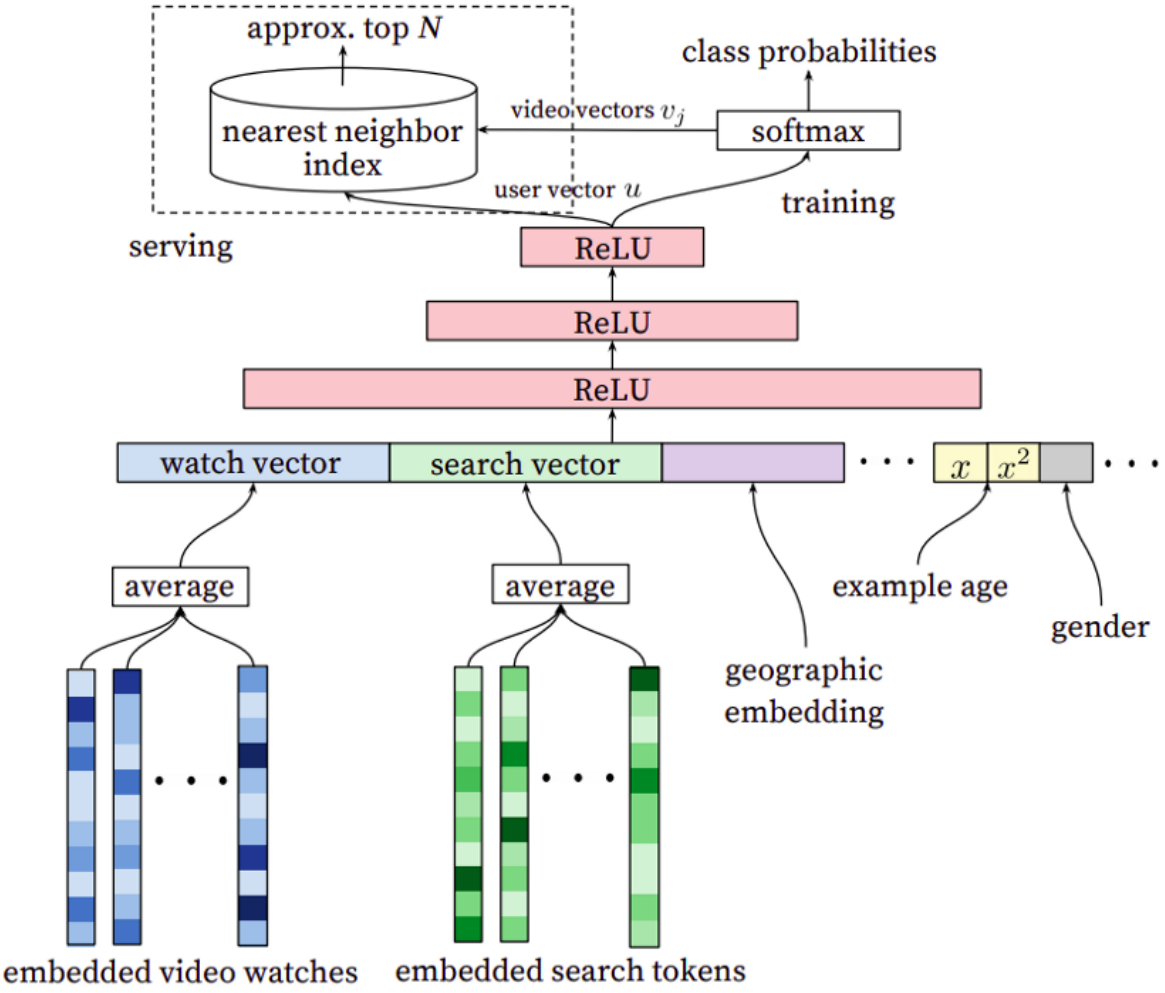
- Youtube has a massive user base and a huge corpus and hence massive data.
- The corpus of youtube is very dynamic, as every second, hour of videos are uploaded
- There are sparsity and external factors that create noisy implicit feedback.

Below we see a framework showing the key components of Youtube.



The system comprises of two neural networks, where one is for candidate generation and another designated for ranking. The candidate generator uses collaborative filtering and is responsible for taking in the users watch history as input and give a small subset of videos as recommendations from youtube's huge corpus of videos. Collaborative filtering considers history and demographics to decide the similarities between users. The ranking system selects top N items by assigning scores to each video according to the desired objective function using the set of features describing the video and user.

Youtube works based on an implicit feedback mechanism; however, given the huge number of videos and users, explicit feedback becomes very sparse in nature. Therefore, implicit feedback is used to train the model. If a user completes watching a video, it is regarded as positive feedback and this way the feedbacks are collected which results in a large amount of feedback which can be used to train model as shown below.

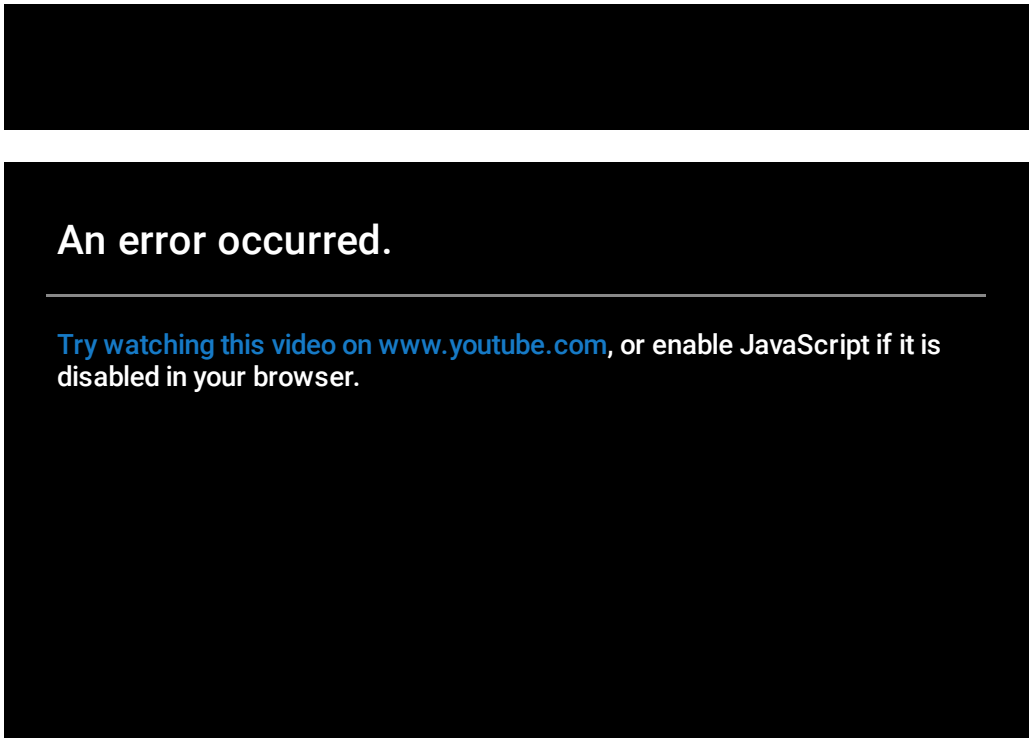


Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 191-198). https://dl.acm.org/doi/pdf/10.1145/2959100.2959190

Some videos

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.



References

1. Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*(3), 56-58. https://dl.acm.org/doi/pdf/10.1145/245108.245121?casa_token=zi6LxDR6CqEAAAAA:v8J1H1NqQU30KLygrZEp3cy99mzQRbfQC0PEr-6ymU8liC6A-wPQW2OOYEImrUvIdsZSg5T60yn1
2. Schafer, J. B., Konstan, J., & Riedl, J. (1999, November). Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce* (pp. 158-166). https://dl.acm.org/doi/pdf/10.1145/336992.337035?casa_token=aDvdD6KIQyYAAAAA:T-nTXEo9G30SIlhO0kLHfNf5cO3NjjTYoaUvzZ_FRtsKMGf2Vat3ar4T8Ajhmwst0TNEiKbhp0kZ
3. Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 5-53.
4. Takács, G., Pilászy, I., Németh, B., & Tikk, D. (2008, October). Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems* (pp. 267-274).
5. Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, *46*, 109-132. https://www.sciencedirect.com/science/article/pii/S0950705113001044?casa_token=RFkFxoOMC-0AAAAA:sinjI6TbaJFzYXqnH3ia90UOsa-JYmkxNOl7AiLXXC2K20BmCXPT2FgT8x-EqM-aYBgz4yZV
6. Cheng, H. T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., ... & Anil, R. (2016, September). Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems* (pp. 7-10). https://dl.acm.org/doi/pdf/10.1145/2988450.2988454?casa_token=nTLBXvnqATgAAAAA:L8pi_gx7aCn2xnSKVaE06VcXoQqIEnLsolp3GeGUc3EKv2n-v-

Z_ALJLZy4NGss7gm_ZUQWBOneY

7. Xue, H. J., Dai, X., Zhang, J., Huang, S., & Chen, J. (2017, August). Deep Matrix Factorization Models for Recommender Systems. In *IJCAI* (Vol. 17, pp. 3203-3209).https://www.ijcai.org/Proceedings/2017/0447.pdf
8. Mu, R. (2018). A survey of recommender systems based on deep learning. *IEEE Access*, *6*, 69009-69022.https://ieeexplore.ieee.org/iel7/6287639/8274985/08529185.pdf
9. Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 191-198). https://dl.acm.org/doi/pdf/10.1145/2959100.2959190
10. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173-182) http://cseweb.ucsd.edu/classes/fa17/cse291-b/reading/p173.pdf

Ethics in AI and Data Science

In this lesson, we discuss ethical aspects of machine learning, data science and artificial intelligence. We refer to them collectively as intelligent systems.

What comes to your mind with the term ethics?

Figure. Ethics in Life and Business: https://www.scu.edu/mobi/resources--tools/blog-posts/ethics-in-life-and-business/ethics-in-life-and-business.html

Creating An Ethical Framework

Define your core values early on

Integrate ethics into your hiring process

Lead by example

Create a culture of openness and welcome dissent

Craft values everyone can own

Learn from immediate peers or distant models

Remember that institutional character is fragile

Know your limits as a leader

Establish an independent board

Figure. A Startup's Guide to Business Ethics and Social Responsibility: https://www.embroker.com/blog/business-ethics-and-social-responsibility/

Bias, fairness, accountability, and transparency

Bias: it usually refers to the bias in collecting data, or in predicting outcomes.

Fairness: It is the opposite of bias in a positive sense.

Accountability: If an intelligent system makes predictions that affect people, there should be some accountability.

Transparency: Is the ML system a 'black-box'? Is it opaque when answering 'why' and making predictions? Can it explain (to some degree) the basis on which decisions were made?

Some questions

Consider a model that has a test on some attribute *A* (e.g., in a decision tree). The following questions need to be asked about the model using the test on *A*:

- is it legal to use a test of *A*?
- is it transparent as to why the test is being used?
- is there a record of how data on *A* was obtained?
- is data collected on *A* likely to be from a biased sampling method?
- is it fair to use a test on *A* for a given sub-group?

Example of the legality of a test

In some jurisdictions (US, Europe), applications for financial loans **cannot** use tests based on gender, ethnic background, etc. It is not entirely clear, but a model cannot use tests that are statistically dependent on such criteria.

Example of data provenance requirements

Among many rules affecting how personal data can be collected and used, the EU's General Data Protection Regulation (GDPR) requires that a person must be given a copy of the data on them used in a system, and provided with meaningful information about the logic involved in that system.

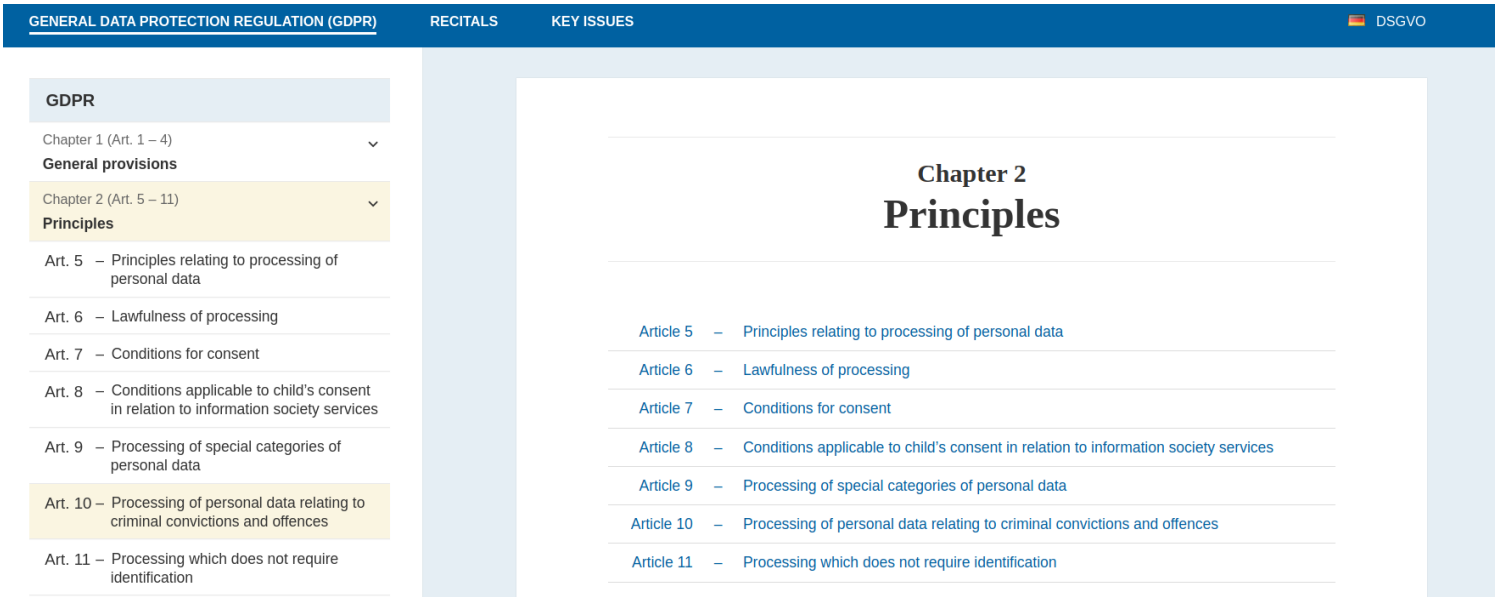


Figure. European Union General Data Protection Regulation (GDPR): https://gdpr-info.eu/

Example of a fairness test

Is the probability distribution of outcome classes the same for all sub-groups conditioned on the values of some attribute or feature *A*? If the answer to this question is **no**, then the model may be unfair, discriminatory or biased in some way.

Please note, in general, there is no single test for fairness.

Read the below research article on fairness:

Hutchinson, B., & Mitchell, M. (2019, January). 50 years of test (un) fairness: Lessons for machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (pp. 49-58): http://www.m-mitchell.com/papers/History_of_Fairness-arxiv.pdf

Interpretability = Explainability = Understanding what the model is doing

Some learning algorithms (such as decision and regression trees, or rule learners) are inherently designed for interpretability. Others are designed with no such considerations. Even 'interpretable' models can become very hard to understand, e.g., if trees are above a certain level of complexity, or are in an ensemble. Many methods have been developed to try and interpret 'black-box' models.

Variable (feature) importance:

In determining variable feature importance, ask yourself which features in the model seem to have the most important role in making the predictions. What are the features that turn up most frequently in the ensemble when computed over the training set, e.g., in a Random Forest?

There are techniques available with many implementations, one of them is explained below:

Issue of Trust

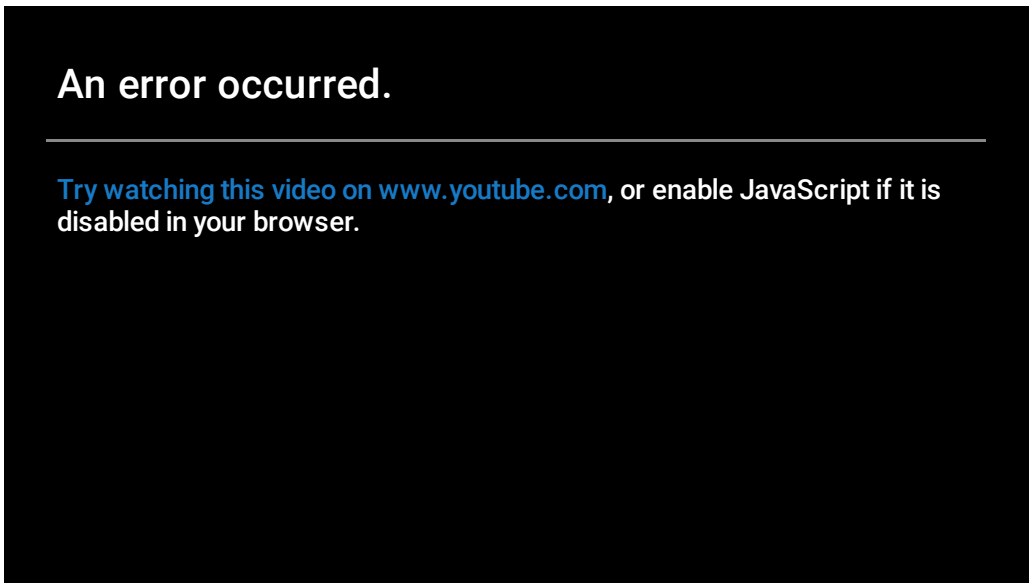
Local Interpretable Model-agnostic Explanations (LIME) is a method that when given a test example to be predicted and a set of 'interpretable' features explains predictions by generating synthetic examples similar to the test example. It fits a local regression or decision tree to these examples using the interpretable features and can be applied to ensemble learning and deep networks.

"Many of the state of the art machine learning models are functionally black boxes, as it is nearly impossible to get a feeling for its inner workings. This brings us to a question of trust: do I trust that a certain prediction from the model is correct? Or do I even trust that the model is making reasonable predictions in general? While the stakes are low in a Go game, they are much higher if a computer is replacing my doctor, or deciding if I am a suspect of terrorism (Person of Interest, anyone?). Perhaps more commonly, if a company is replacing some system with one based on machine learning, it has to trust that the machine learning model will behave reasonably well."

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 4: Explaining an image classification prediction made by Google's Inception neural network. The top 3 classes predicted are "Electric Guitar" ($p = 0.32$), "Acoustic guitar" ($p = 0.24$) and "Labrador" ($p = 0.21$)

Source and Figure. LIME - Local Interpretable Model-Agnostic Explanations: https://homes.cs.washington.edu/~marcotcr/blog/lime/



Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). " Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1135-1144). https://arxiv.org/pdf/1602.04938v1.pdf

Global + Local interpretability

- global: compute SHAP values for each feature in the model to quantify its contribution to the output prediction
- local: for an individual instance, how important was each feature in contributing to the classification?
- packages to compute this efficiently for tree-based models and ensembles.

SHAP (SHapley Additive exPlanations) is a game theory approach to explain the output of any machine learning model by breaking down a prediction to show the impact of each feature.

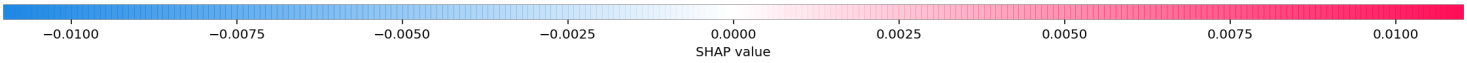
Where could you use this?

- A model says a bank shouldn't loan someone money, and the bank is legally required to explain the basis for each loan rejection
- A healthcare provider wants to identify what factors are driving each patient's risk of some disease so they can directly address those risk factors with targeted health interventions

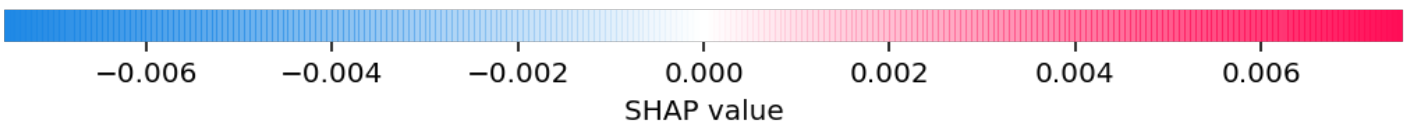
You'll use SHAP Values to explain individual predictions in this lesson. In the next lesson, you'll see how these can be aggregated into powerful model-level insights.

Note that none of these explainability methods imply causality!

Below are some examples of attempts to explain how the decision has been reached by SHAP python based library.


−0.0100
−0.0075
−0.0050
−0.0025
0.0000
0.0025
0.0050
0.0075
0.0100
SHAP value


dowitcher
red-backed_sandpiper
meerkat
mongoose


−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
SHAP value

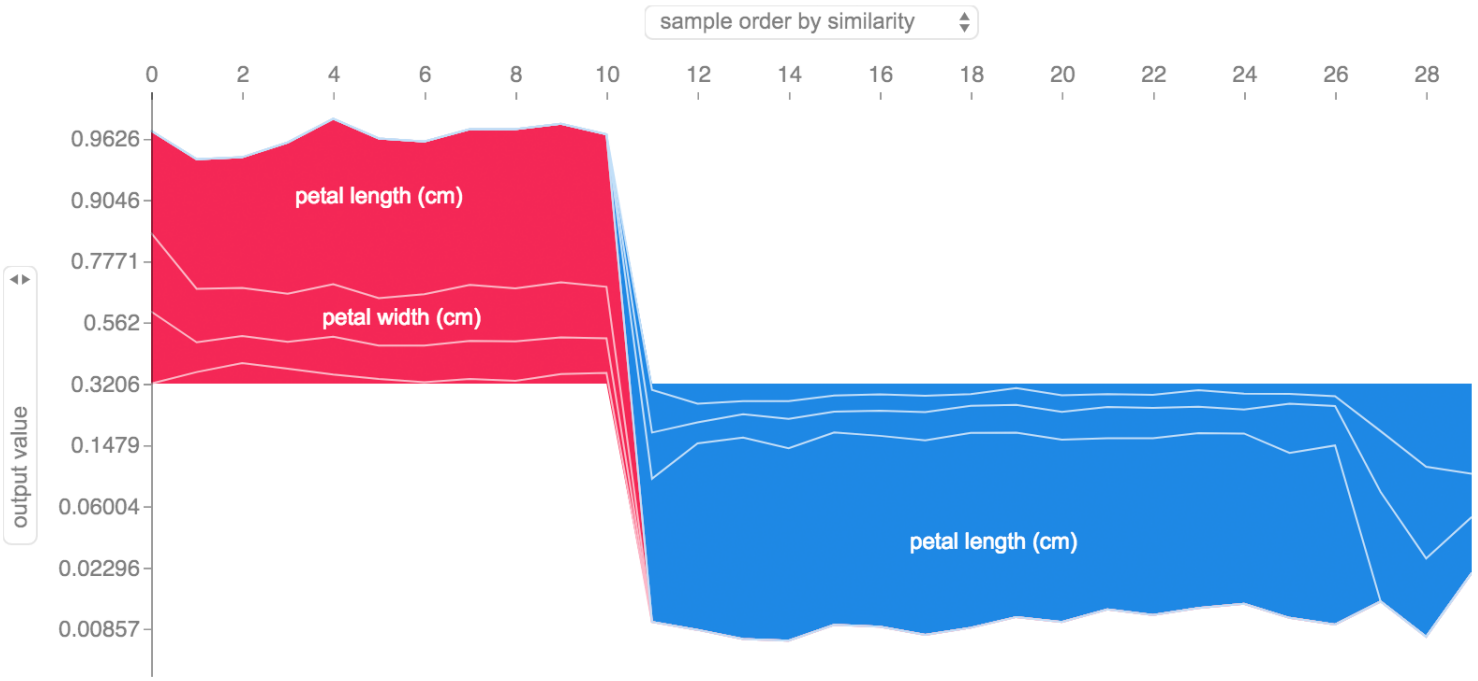


Figure: shap: https://github.com/slundberg/shap

The python shap library usage is below.

PYTHON

```python
import shap # Not in Ed (try in own pc)
import numpy as np

# select a set of background examples to take an expectation over
background = x_train[np.random.choice(x_train.shape[0], 100, replace=Fal

# explain predictions of the model on four images
e = shap.DeepExplainer(model, background)
# ...or pass tensors directly
# e = shap.DeepExplainer((model.layers[0].input, model.layers[-1].output
shap_values = e.shap_values(x_test[1:5])

# plot the feature attributions
shap.image_plot(shap_values, -x_test[1:5])
```

Code: https://github.com/slundberg/shap

The paper is given below.

Štrumbelj, E., & Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, *41*(3), 647-665.

Interpretability relates to fairness

In finance, systems need to justify a credit-granting decision to the customer.

If your credit card limit was 5% lower, would you obtain the mortgage?

Provably beneficial Artificial Intelligence

Instead of focusing solely on preventing the negative aspects, here is a perspective on ensuring artificial intelligence does not harm humans.

the robot can be maximally useful and avoid potentially disastrous mistakes. Yet acting optimally in coffee acquisition while leaving the robot as a passive observer may not be the best way to achieve value alignment. Instead, the human should perhaps explain the steps in coffee preparation and show the robot where the backup coffee supplies are kept and what to do if the coffee pot is left on the heating plate too long, while the robot might ask what the button with the puffy steam symbol is for and try its hand at coffee making with guidance from the human, even if the first results are undrinkable. None of these things fit in with the standard IRL framework.

By extending IRL to incorporate both human and robot as agents, it becomes possible to formulate and solve a value alignment problem as a cooperative and interactive reward maximization process (Hadfield-Menell et al., 2017a). More precisely, a *cooperative inverse reinforcement learning* (CIRL) problem is a two-player game of partial information, in which the human knows the reward function[2] while the robot does not; but the robot's payoff is exactly the human's actual reward. (Thus, CIRL instantiates all three principles given above.) Optimal solutions to this game maximize human reward and may naturally generate active instruction by the human and active learning by the robot.

Within the CIRL framework, one can formulate and solve the off-switch problem—that is, the problem of preventing a robot from disabling its own off-switch. (With this, Turing may rest easier.) A robot that is designed to solve the CIRL problem is sure it wants to maximize human values, but also sure it does not know exactly what those are. Now, the robot actually *benefits* from being switched off, because it understands that the human will press the off-switch to prevent the robot from doing something counter to human values. Thus, the robot has a positive incentive to preserve the off-switch, and this incentive derives directly from its uncertainty about human values. Furthermore, it is possible to show that in some cases the robot is *provably beneficial*, that is, the expected reward for the human is higher when the CIRL-solving robot is available, regardless of what the human's actual reward function is (Hadfield-Menell et al., 2017b).

The off-switch example suggests some templates for controllable agent designs and provides at least one case of a provably beneficial system. The overall approach has some resemblance to mechanism design problems in economics,

Russell, S. (2017). Provably beneficial artificial intelligence. *Exponential Life, The Next Step:* https://people.eecs.berkeley.edu/~russell/papers/russell-bbvabook17-pbai.pdf

Safety and Fairness in systems using Artificial Intelligence

We refer to a set of safety and fairness points given by the following except in a paper

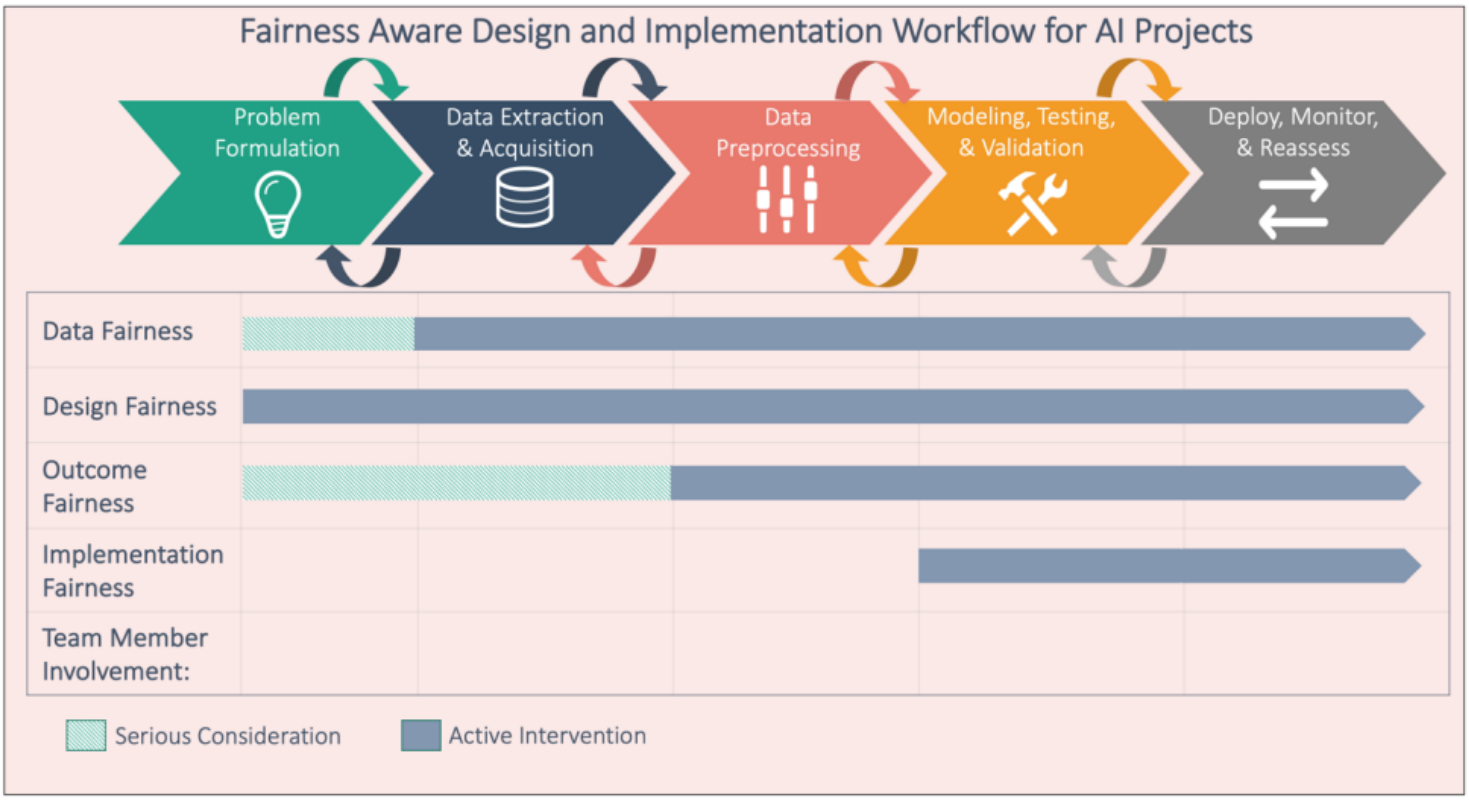
Principle of Discriminatory Non-Harm: The designers and users of AI systems, which process social or demographic data pertaining to features of human subjects, societal patterns, or cultural formations, should prioritise the mitigation of bias and the exclusion of discriminatory influences on the outputs and implementations of their models. Prioritising discriminatory non-harm implies that the designers and users of AI systems ensure that the decisions and behaviours of their models do not generate discriminatory or inequitable impacts on affected individuals and communities. This entails that these designers and users ensure that the AI systems they are developing and deploying:

1. Are trained and tested on properly representative, relevant, accurate, and generalisable datasets (**Data Fairness**)
2. Have model architectures that do not include target variables, features, processes, or analytical structures (correlations, interactions, and inferences) which are unreasonable, morally objectionable, or unjustifiable (**Design Fairness**)
3. Do not have discriminatory or inequitable impacts on the lives of the people they affect (**Outcome Fairness)**
4. Are deployed by users sufficiently trained to implement them responsibly and without bias (**Implementation Fairness**)

Leslie, D. (2019). Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*. https://www.turing.ac.uk/sites/default/files/2019-

Leslie, D. (2019). Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*. https://www.turing.ac.uk/sites/default/files/2019-

Leslie, D. (2019). Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*. https://www.turing.ac.uk/sites/default/files/2019-



Leslie, D. (2019). Understanding artificial intelligence ethics and s*afety. arXiv preprint arXiv:1906.05684.* *https://www.turing.ac.uk/sites/default/files/2019-*

Beijing AI Principles

- *"Do Good: AI should be designed and developed to promote the progress of society and human civilization, to promote the sustainable development of nature and society, to benefit all mankind and the environment, and to enhance the well-being of society and ecology.*
- *For Humanity: The R&D of AI should serve humanity and conform to human values as well as the overall interests of mankind. Human privacy, dignity, freedom, autonomy, and rights should be sufficiently respected. AI should not be used to against, utilize or harm human beings.*
- *Be Responsible: Researchers and developers of AI should have sufficient considerations for the potential ethical, legal, and social impacts and risks brought in by their products and take concrete actions to reduce and avoid them.*
- *Control Risks: Continuous efforts should be made to improve the maturity, robustness, reliability, and controllability of AI systems, so as to ensure the security for the data, the safety and security for the AI system itself, and the safety for the external environment where the AI system deploys.*
- *Be Ethical: AI R&D should take ethical design approaches to make the system trustworthy. This may include, but not limited to: making the system as fair as possible, reducing possible discrimination and biases, improving its transparency, explainability, and predictability, and making the system more traceable, auditable and accountable.*
- *Be Diverse and Inclusive:The development of AI should reflect diversity and inclusiveness, and be designed to benefit as many people as possible, especially those who would otherwise be easily*

neglected or underrepresented in AI applications.

- *Open and Share:It is encouraged to establish AI open platforms to avoid data/platform monopolies, to share the benefits of AI development to the greatest extent, and to promote equal development opportunities for different regions and industries."*

Source: https://www.baai.ac.cn/news/beijing-ai-principles-en.html?fbclid=IwAR2HtIRKJxxy9Q1Y953H-2pMHl_bIr8pcsIxho93BtZY-FPH39vV9v9B2eY

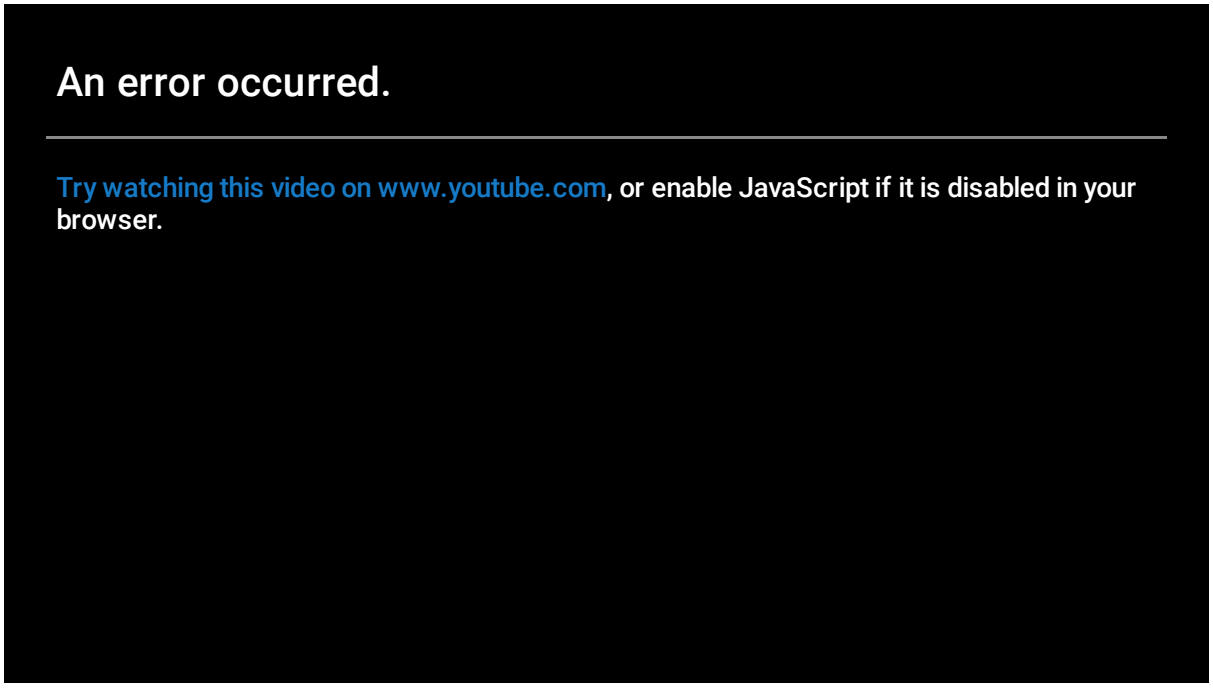
More information

1. https://deepmind.com/safety-and-ethics
2. https://plato.stanford.edu/entries/ethics-ai/
3. https://en.wikipedia.org/wiki/Machine_ethics

These videos will help you to understand the future of machine learning and artificial intelligence in the industry. It is important to understand that artificial intelligence does not only have technical issues but there are social and ethical challenges as well. These videos will help you analyse these challenges.

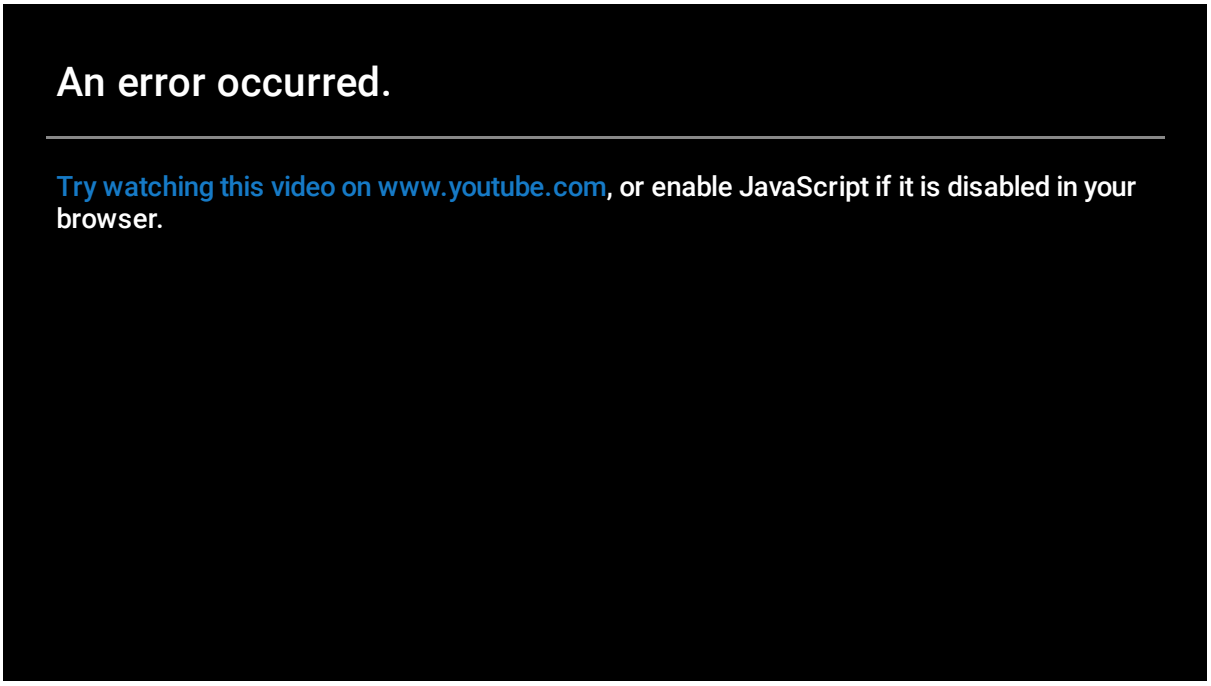
Watch the below videos that give an overview of artificial intelligence and ethical issues related to AI.

What is Artificial Intelligence? In 5 minutes



Ramesh, R. (2017, August 13). *What is Artificial Intelligence? In 5 minutes.* [online video]. Retrieved from: https://www.youtube.com/watch?v=2ePf9rue1Ao.

The ethical dilemma we face on AI and autonomous tech



TEDx Talks (2017, May 11). *The ethical dilemma we face on AI and autonomous tech | Christine Fox | TEDxMidAtlantic* [online video]. Retrieved from https://www.youtube.com/watch?v=3oE88_6jAwc&feature=emb_title.

References

1. Moor, J. H. (2006). The nature, importance, and difficulty of machine ethics. *IEEE intelligent systems*, *21*(4), 18-21. https://ieeexplore.ieee.org/document/1667948
2. Goodall, N. J. (2014). Machine ethics and automated vehicles. In *Road vehicle automation* (pp. 93-102). Springer, Cham. https://www.researchgate.net/profile/Noah_Goodall/publication/300567119_Machine_Ethics_and_Automated_Vehicles/links/57d02ff308ae5f03b489083f.pdf
3. Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., ... & Chatila, R. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82-115.
4. Leslie, D. (2019). Understanding artificial intelligence ethics and safety. *arXiv preprint arXiv:1906.05684*. https://www.turing.ac.uk/sites/default/files/2019-06/understanding_artificial_intelligence_ethics_and_safety.pdf
5. Etienne, H. (2020). When AI Ethics Goes Astray: A Case Study of Autonomous Vehicles. *Social Science Computer Review*, 0894439320906508. https://journals.sagepub.com/doi/pdf/10.1177/0894439320906508
6. Štrumbelj, E., & Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, *41*(3), 647-665.

Exercise 2

Part 1

Think about the following questions on ethical issues around machine learning. Are you able to address them?

- What are the main areas of ethical concern and responsibility in ML?
- Why transparency in ML is important and what are the techniques to address it?
- What are the potential legal issues around applications of ML?

Part 2

1. Consider the case of face recognition used by the Australian police force. What ethical issues can arise
2. Discuss how you would ensure that a deep learning model is explainable?
3. Discuss privacy and ethical concerns in recording an online tutorial discussion. Would you be comfortable if it's shared in youtube or social media later?
4. Consider the following papers.

: Estudillo, A. J. (2020). Self-reported face recognition abilities for own and other-race faces. https://www.frontiersin.org/articles/10.3389/fpsyg.2020.00208/full

Why face-recognition technology has a bias problem https://www.cbsnews.com/news/facial-recognition-systems-racism-protests-police-bias/

Yucer, S., Akçay, S., Al-Moubayed, N., & Breckon, T. P. (2020). Exploring Racial Bias within Face Recognition via per-subject Adversarially-Enabled Data Augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 18-19). https://openaccess.thecvf.com/content_CVPRW_2020/papers/w1/Yucer_Exploring_Racial_Bias_Within_Face_Recognition_via_Per-Subject_Adversarially-Enabled_Data_CVPRW_2020_paper.pdf

Discuss what sort of biases with training data is there for face recognition tasks taking ethnicity into account?