# Deep Learning

Review of Fundamental Learners
Review of ANNs
Tensorflow(Keras API, Low Level)
Autoencoders
CNN
GRU
LSTM
DBN
GAN
and more...

Dom Huh

## Review of Fundamental Learners

[Linear, Ridge, LASSO, ElasticNet, Logistic] Regression

K-Nearest Neighbors (KNN)

Support Vector Machines (SVM)

[K-Means, Mean Shift] Clustering

Naive Bayes

Decision Trees

[Bagging, Boosting, Stacking] Ensemble Learning

## Review of ANNs

Network structure

[Input Layer, Hidden Layers (Perceptrons), Activation Layers, Output Layer]

Propagation

Epoch

Forward propagation [Weights, Biases, Activation]

LossFunction [Cross-Entropy, Gradient Descent / RMSProp / AdaGrad / AdaDelta / ADAM]

Backpropagation [Adjusting weights and/or biases]

Momentum: https://www.youtube.com/watch?v=k8fTYJPd3_I

RMSProp: https://www.youtube.com/watch?v=_e-LFe_igno

ADAM: https://www.youtube.com/watch?v=JXQT_vxqwIs

# Tensorflow (Keras API)

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential()
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(256, kernel_regularizer=tf.keras.regularizers.l1(0.01),
bias_regularizer=tf.keras.regularizers.l2(0.01))
model.add(layers.Dense(256, kernel_initializer='orthogonal',
bias_initializer=tf.keras.initializers.constant(2.0))
                ...
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer=tf.train.AdamOptimizer(0.01), loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(train_data, train_labels, epochs=100, batch_size=32)

model.save('file.h5')
model = tf.keras.models.load_model('file.h5')

model.evaluate(test_data, test_labels, batch_size=32)
result = model.predict(predict_data, batch_size=32)
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers
https://keras.io/losses/
https://keras.io/metrics/
https://keras.io/initializers/

# Tensorflow (Low Level)

```
import tensorflow as tf
from sklearn.model_selection import train_test_split

learning_rate = 0.01
epochs = 1000
layer_1_nodes = 256
layer_2_nodes = 256

X_train, X_test, Y_train, Y_test = train_test_split( dataset.data, dataset.target, test_size=0.2)
size_inputs = X_train.shape[1]
size_output = len(np.unique(Y_train, axis=0))

with tf.variable_scope('input'):
      X = tf.placeholder(tf.float32, shape= (None, size_inputs))
with tf.variable_scope('layer_1'):
      weights = tf.get_variable('weights1', shape=[size_inputs, layer_1_nodes], initializer =
tf.contrib.layers.xavier_initializer())
      biases = tf.get_variable('bias1', shape=[layer_1_nodes], initializer = tf.zeros_initializer())
      layer_1_output = tf.nn.relu(tf.matmul(X, weights) + biases)
with tf.variable_scope('layer_2'):
      weights = tf.get_variable('weights2', shape=[layer_1_nodes, layer_2_nodes], initializer =
tf.contrib.layers.xavier_initializer())
      biases = tf.get_variable('bias2', shape=[layer_2_nodes], initializer = tf.zeros_initializer())
      layer_2_output = tf.nn.relu(tf.matmul(layer_1_output, weights) + biases)
```

https://www.tensorflow.org/api_docs/python/tf/Tensor
https://www.tensorflow.org/api_docs/python/tf/Operation
https://www.tensorflow.org/api_docs/python/tf/placeholder
https://github.com/tensorflow/tensorflow/blob/r1.12/tensorflow/contrib/layers/python/layers/initializers.
py

# Tensorflow (Low Level)

```
with tf.variable_scope('output'):
      weights = tf.get_variable('weights3', shape=[layer_2_nodes, num_output], initializer =
tf.contrib.layers.xavier_initializer())
      biases = tf.get_variable('bias3', shape=[num_output], initializer = tf.zeros_initializer())
      prediction = tf.matmul(layer_2_output, weights) + biases

with tf.variable_scope('cost'):
      Y = tf.placeholder(tf.float32, shape = (None, num_output))
      cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = Y, logits = prediction))
with tf.variable_scope('train'):
      optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
with tf.variable_scope('accuracy'):
      correct_prediction = tf.equal(tf.argmax(Y, axis =1), tf.argmax(prediction, axis =1) )
      accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.variable_scope("logging"):
      tf.summary.scalar('current_cost', cost)
      tf.summary.scalar('current_accuacy', accuracy)
      summary = tf.summary.merge_all()
```

https://www.tensorflow.org/api_docs/python/tf/variable_scope
https://www.tensorflow.org/api_docs/python/tf/losses
https://www.tensorflow.org/api_docs/python/tf/math/reduce_mean
https://www.tensorflow.org/versions/r1.9/api_guides/python/summary
https://www.tensorflow.org/api_docs/python/tf/get_variable
https://www.tensorflow.org/api_docs/python/tf/train
https://www.tensorflow.org/api_docs/python/tf/math/argmax

# Tensorflow (Low Level)

```
with tf.Session() as session:
     session.run(tf.global_variables_initializer())

     for epoch in range(epochs):
          session.run(optimizer, feed_dict={X:X_train, Y:Y_train})

          if epoch %10 == 0:
               training_cost, training_summary = session.run([cost, summary], feed_dict={X: X_train, Y:
          Y_train})
               testing_cost, testing_summary = session.run([cost, summary], feed_dict={X: X_test, Y:
          Y_test})
               train_accuracy = session.run(accuracy, feed_dict={X: X_train, Y: Y_train})
               test_accuracy = session.run(accuracy, feed_dict={X: X_test, Y: Y_test})
               print(epoch, training_cost, testing_cost, train_accuracy, test_accuracy )

print("Done")

final_train_accuracy = session.run(accuracy, feed_dict={X: X_train, Y: Y_train})
final_test_accuracy = session.run(accuracy, feed_dict={X: X_test, Y: Y_test})

print("Final Training Accuracy: {}".format(final_train_accuracy))
print("Final Testing Accuracy: {}".format(final_test_accuracy))
```

https://www.tensorflow.org/versions/r1.3/api_docs/java/reference/org/tensorflow/Session
https://www.tensorflow.org/api_docs/python/tf/initializers/global_variables
https://www.tensorflow.org/api_docs/python/tf/train/Saver

## Tensorflow Resources

Learn more from following resources:

https://www.tensorflow.org/guide/datasets

https://www.youtube.com/watch?v=vq2nnJ4g6N0&t=2277s

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist

## Autoencoders

Feeding in an input and expecting that input as the output. Essential for unsupervised learning.
Essentially, developing a model that can output a compressed version of the input data with the lowest amount of loss in data.

Dimensionality Reduction (PCA, Neural Gas)

Encoder:
    Fuses the parameters together to create a compressed version but with fewer parameters
Decoder:
    Reconstructs the compressed version

https://github.com/Tathagatd96/Deep-Autoencoder-using-Tensorflow/blob/master/Stacked_AutoEncoder.py
https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Variantional Autoencoder: http://kvfrans.com/variational-autoencoders-explained/

# CNN

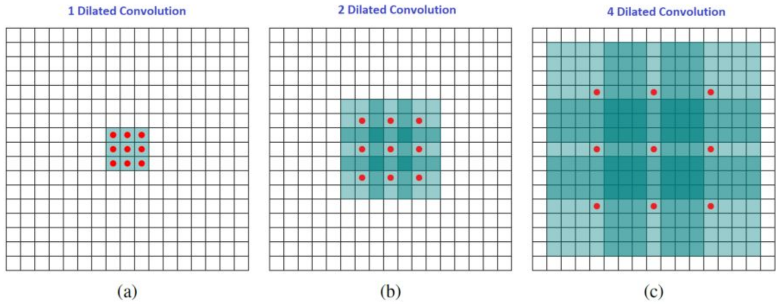
Convolution Layer

Conv layer scans and filters (developed through training) through the given data in small batches.

Hyperparameters:

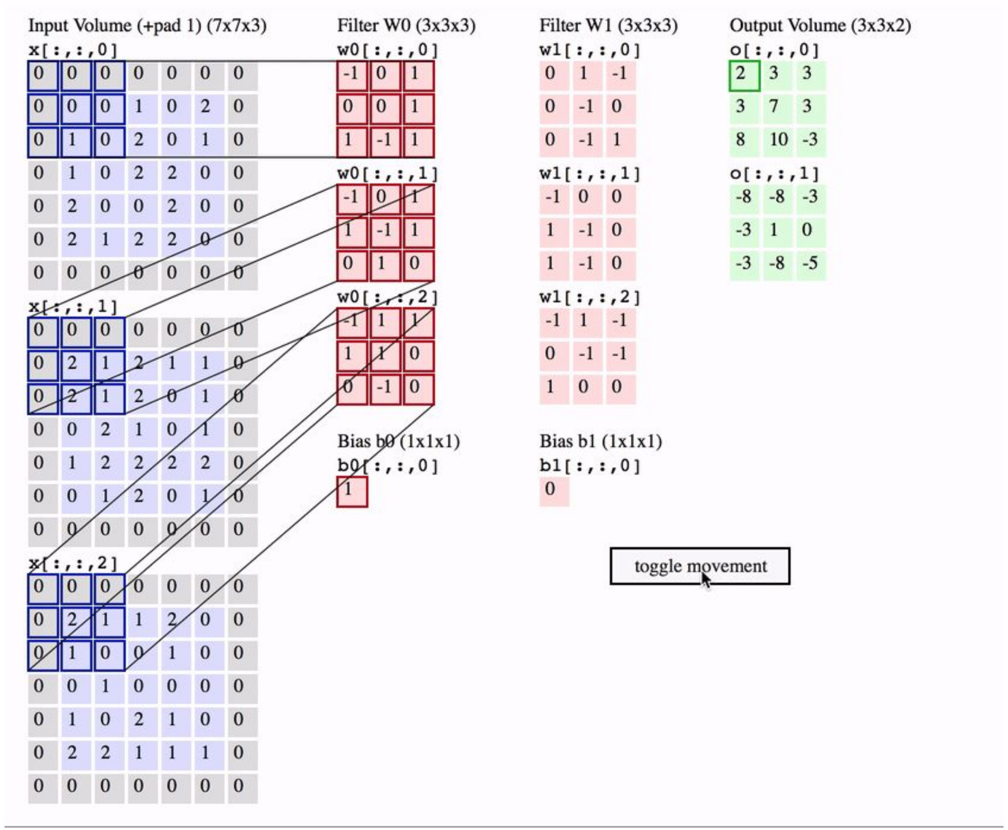
- Depth - How many filters?
- Stride- How many pixels/units away is the next stride?
- Zero-padding - Pads with zero (Useful for spatial control)
  - Same vs. Valid (drop)

Dilation -



Separable, Deconv(transpose) layers : https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
Visualization: https://www.youtube.com/watch?v=f0t-OCG79-U

# Conv Layer Visualization



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 2 | 0 |
| 0 | 1 | 0 | 2 | 0 | 1 | 0 |
| 0 | 1 | 0 | 2 | 2 | 0 | 0 |
| 0 | 2 | 0 | 0 | 2 | 0 | 0 |
| 0 | 2 | 1 | 2 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

x[:,:,1]

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 1 | 2 | 1 | 1 | 0 |
| 0 | 2 | 1 | 2 | 0 | 1 | 0 |
| 0 | 0 | 2 | 1 | 0 | 1 | 0 |
| 0 | 1 | 2 | 2 | 2 | 2 | 0 |
| 0 | 0 | 1 | 2 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

x[:,:,2]

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 1 | 1 | 2 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 2 | 1 | 0 | 0 |
| 0 | 2 | 2 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter W0 (3x3x3)

w0[:,:,0]

| | | |
|---|---|---|
| -1 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | -1 | 1 |

w0[:,:,1]

| | | |
|---|---|---|
| -1 | 0 | 1 |
| 1 | -1 | 1 |
| 0 | 1 | 0 |

w0[:,:,2]

| | | |
|---|---|---|
| -1 | 1 | 1 |
| 1 | 1 | 0 |
| 0 | -1 | 0 |

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

| | | |
|---|---|---|
| 0 | 1 | -1 |
| 0 | -1 | 0 |
| 0 | -1 | 1 |

w1[:,:,1]

| | | |
|---|---|---|
| -1 | 0 | 0 |
| 1 | -1 | 0 |
| 1 | -1 | 0 |

w1[:,:,2]

| | | |
|---|---|---|
| -1 | 1 | -1 |
| 0 | -1 | -1 |
| 1 | 0 | 0 |

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

| | | |
|---|---|---|
| 2 | 3 | 3 |
| 3 | 7 | 3 |
| 8 | 10 | -3 |

o[:,:,1]

| | | |
|---|---|---|
| -8 | -8 | -3 |
| -3 | 1 | 0 |
| -3 | -8 | -5 |

## CNN

Pooling Layer

- Pooling layer down-samples the input, compressing the data based on specified operation
- Hyperparameters:
  - Size : How large of a window?
  - Stride: How many pixels/units away is the next stride?
  - Type: Max, Average

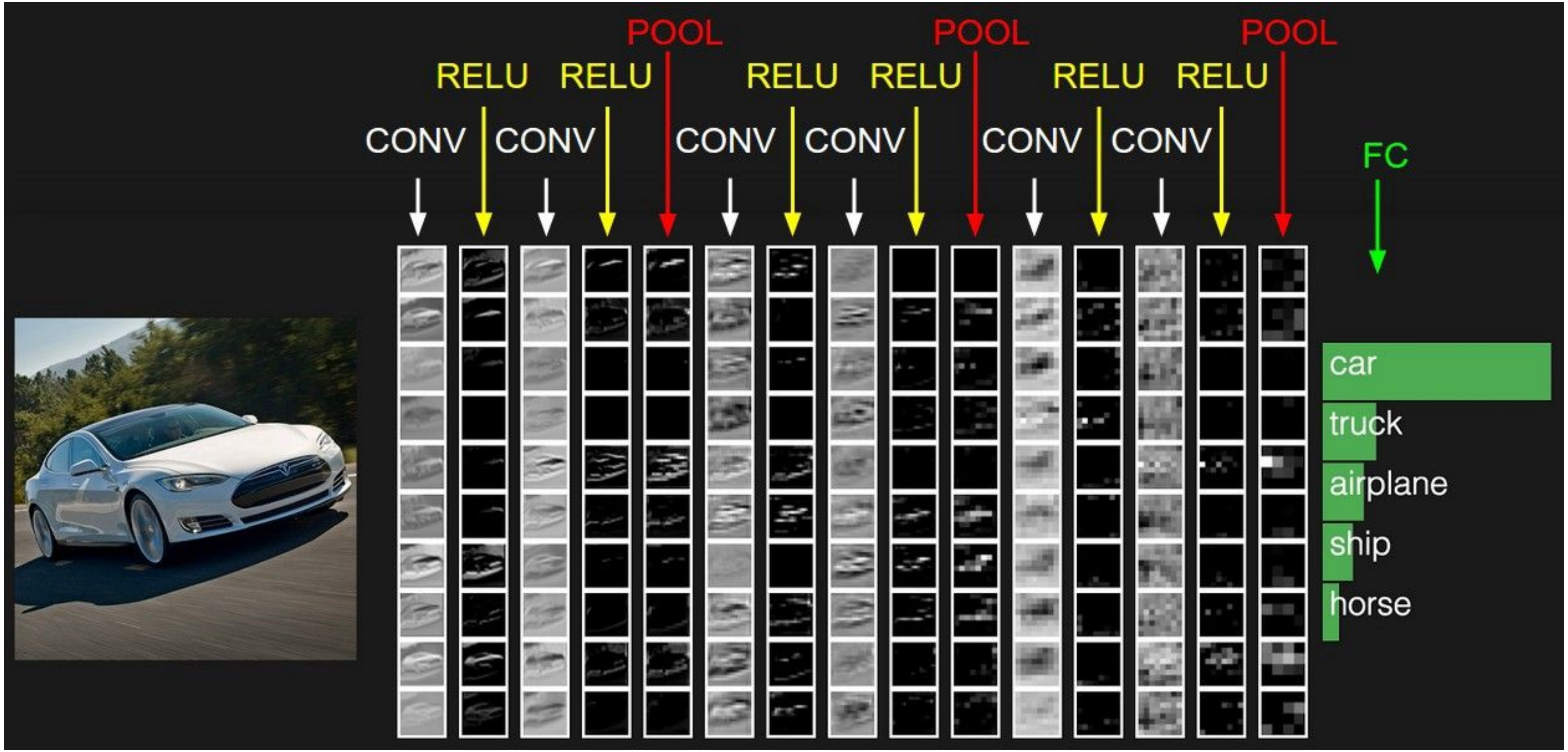
Max pooling - Maximum of that value within the pool window

General/average pooling - Average of pool window elements

Used to reduce computational time and complexity by reducing the number of parameters. But loses positional information about the different objects inside the image. (N)

https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling2D

# Basic architecture of CNN



## CNN

Video:

https://www.youtube.com/watch?v=bNb2fEVKeEo

https://www.youtube.com/watch?v=FmpDIaiMIeA&t=911s

Link:

http://cs231n.github.io/convolutional-networks/

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## Examples of CNNs ( http://cs231n.github.io/convolutional-networks/ )

**LeNet**. The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.

**AlexNet**. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

**ZF Net**. The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

**GoogLeNet**. The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much. There are also several followup versions to the GoogLeNet, most recently Inception-v4.

**VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

**ResNet**. Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special *skip connections* and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016). In particular, also see more recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016).

## Review of RNN

Simple RNN [Elman, Jordan, Hopsfield] networks

Includes memory either from output or hidden layers

Memory(Hidden State) Unit/Layer

Where memory is feed/accessed/managed

Training with time variable(Truncated backpropagation through time):
http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Werbos.backprop.pdf

Batch Normalization: https://arxiv.org/pdf/1502.03167.pdf
https://www.youtube.com/watch?v=nUUqwaxLnWs

Vanishing gradient problem: https://www.youtube.com/watch?v=qhXZsFVxGKo
https://www.tensorflow.org/api_docs/python/tf/keras/layers/RNN

RNN: https://arxiv.org/pdf/1506.00019.pdf

## Review of RNN

VIDEO:

https://www.youtube.com/watch?v=LHXXI4-IEns

## GRU (Gated Recurrent Unit)

Able to retain memory from a long time ago while training without washing it through time or remove information which is irrelevant to the prediction

With this memory functionality, how does it is used?

Update gate: Determine how much of the past information needs to be passed along to the future.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

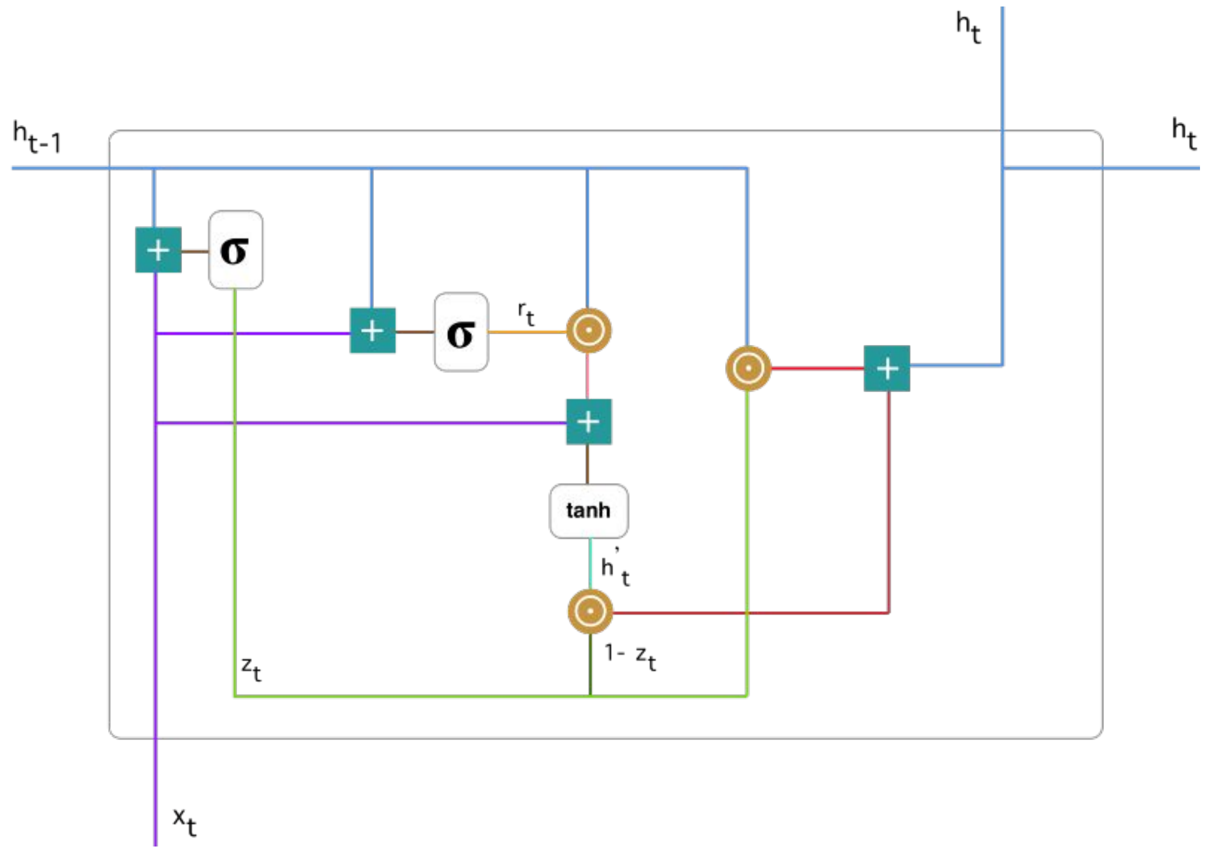
Reset gate: Decide how much of the past information to forget

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU

# GRU



## LSTM (Long Short Term Memory)

Use weight matrix to find the following:

Input gate: Decides whether to write to  cell (sigmoid)

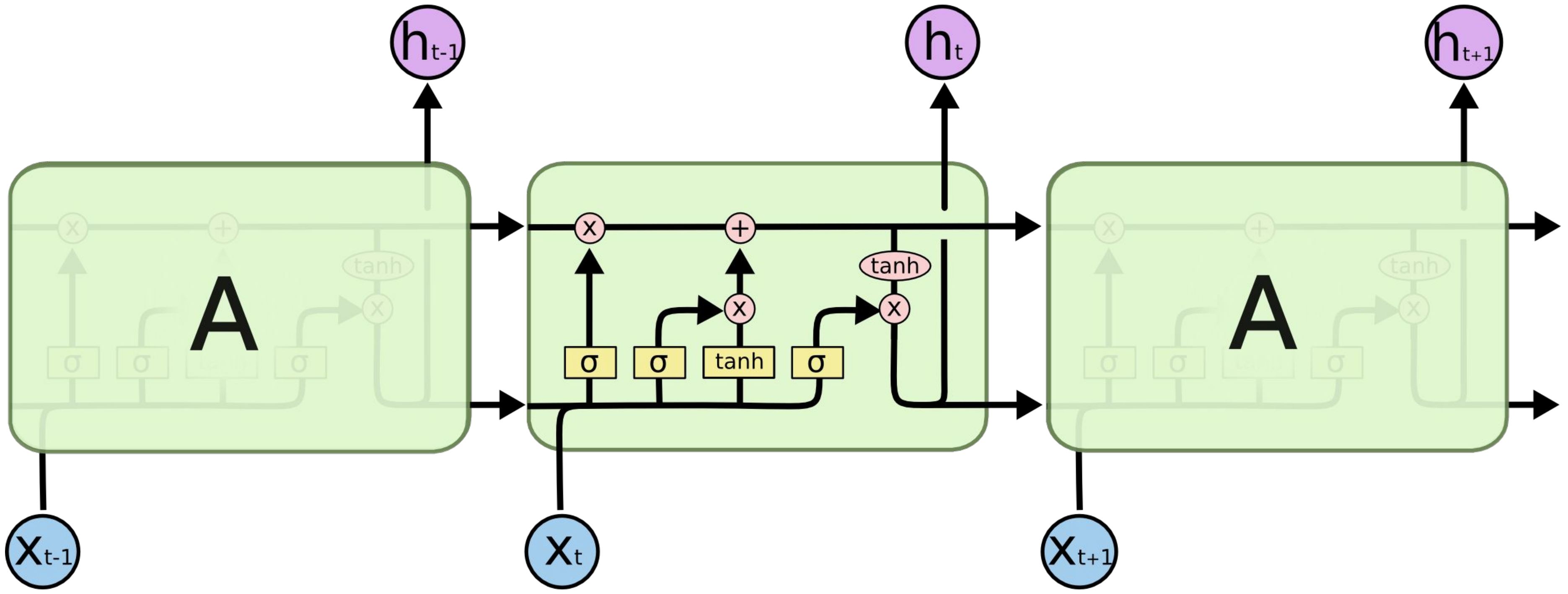
Forget gate: Decides whether to erase cell (sigmoid)

Output gate: Decides how much to reveal cell (sigmoid)

Tanh gate: Decides how much to write to cell (tanh)

https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

# LSTM



# GRU, LSTM

VIDEO:

https://www.youtube.com/watch?v=8HyCNIVRbSU

## BRNN, NTM

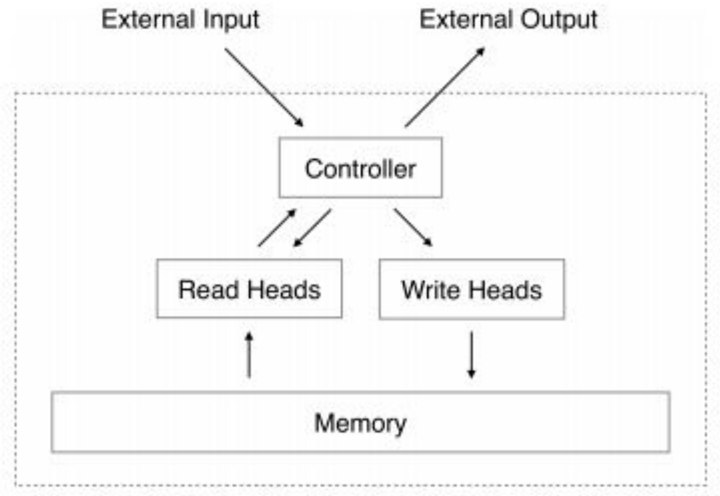
Bidirectional recurrent neural networks: https://www.youtube.com/watch?v=uRFegQXnY54

[Noncausal] acyclic graph approach that takes into account future values

Neural Turing machines: https://arxiv.org/pdf/1410.5401.pdf

Takes input and output and learn algorithms that map from one to the other.

https://arxiv.org/pdf/1506.00019.pdf



## RBM, HMM

Restricted Boltzmann Machines: Same as autoencoders except consists of two layers (visible, hidden) and uses a stochastic approach. Therefore defines the probability distribution of the input and models the hidden layer. The reconstruction is then passed back to the visible layer. (Issue: Vanishing Gradient)

RBM: https://www.youtube.com/watch?v=it_PXVIMyWg

Hidden Markov Machines: Similar to Naive Bayes, but in a sequential approach. Diverges from RNN since it is a generative model, whereas RNN are often discriminative, HMM are far simpler structurally, and HMM are rooted in the Markovian idea that present states are dependant on the surrounding states in terms of time.

HMM: https://www.youtube.com/watch?v=9dp4whVQv5s

# DBN

Deep Belief Network: Stacking RBMs but act as mixed undirected and directed acyclic graph. Structurally similar to MLP with layers near the visible layer acting as sigmoid belief networks with RBM stacked at the top. Generative model.

Greedy Layer-wise Training: Pre-train each layer one at a time (unsupervised) then fine-tune with supervised learning.

Wake-sleep: Train all the layers at a time

DBN:
**** https://www.mitpressjournals.org/doi/pdf/10.1162/neco.2006.18.7.1527 ****
https://www.youtube.com/watch?v=GJdWESd543Y
https://www.youtube.com/watch?v=wFY0P2miQSw

DBM:
http://proceedings.mlr.press/v5/salakhutdinov09a/salakhutdinov09a.pdf

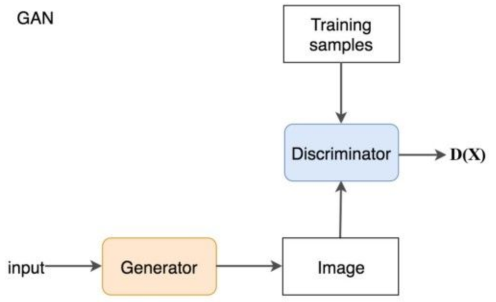
# GAN

Generative Adversarial Networks: Consists of two agents. The generative agent passes its "fake" data into the discriminative agent with intentions to "fool" the discriminative agent. With training, both agents will become very accurate. (Counterfeiters vs Police)

https://arxiv.org/pdf/1406.2661.pdf

https://arxiv.org/pdf/1701.00160.pdf

https://www.youtube.com/watch?v=CIfsB_EYsVI


GAN
Training samples
Discriminator
D(X)
input
Generator
Image