ESS2222

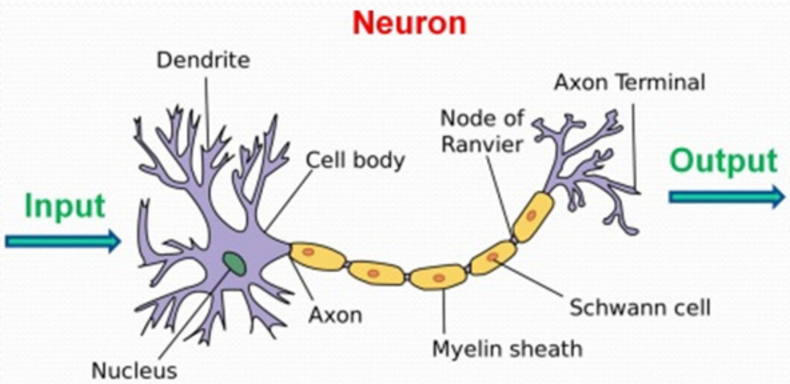
Lecture 7 – Deep learning and Convolutional Networks

Hosein Shahnas

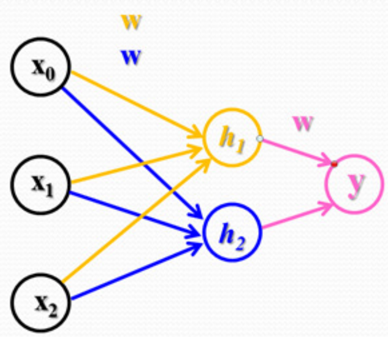
University of Toronto, Department of Earth Sciences,

Outline

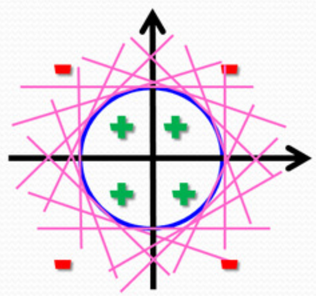
- ❑ **Deep Learning**
- ❑ **Convolutional Neural Networks**
- ❑ **Recurrent Neural Networks**



Review of Lecture 6

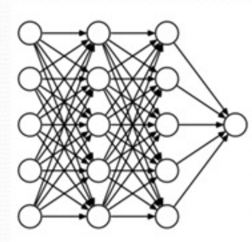


h_1 *Or* h_2 *Nand* y And

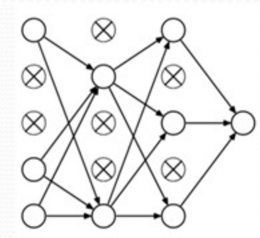


16-Perceptrons

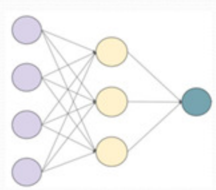
Dropout

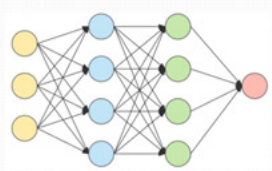


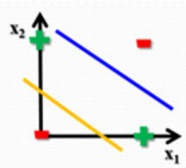
Standard neural network

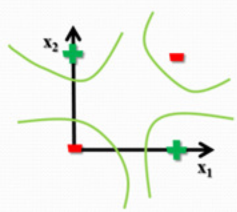


Neural network with dropout

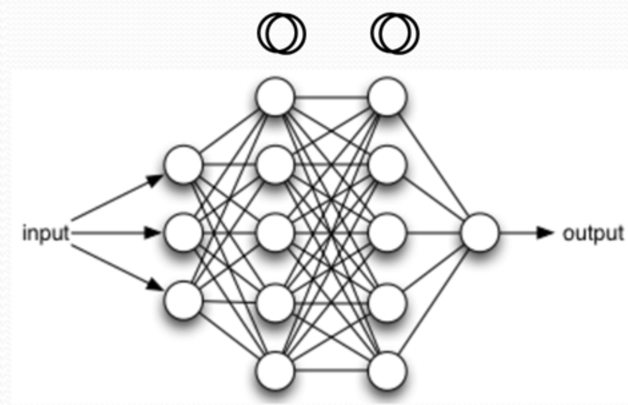








Neural Networks



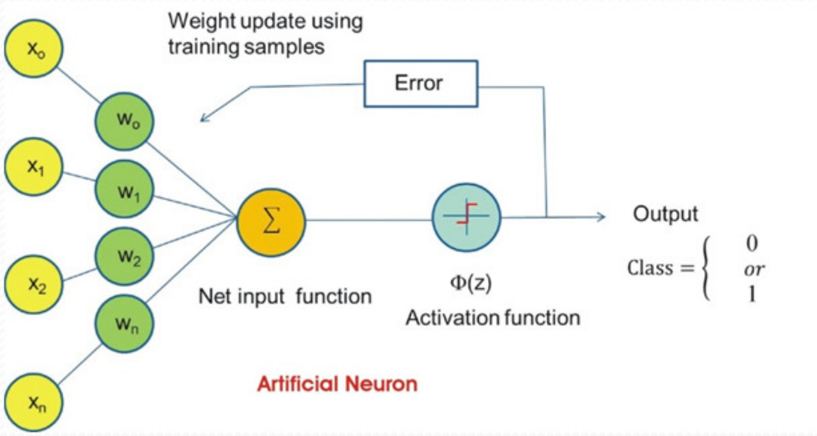
Back Propogation

$$\frac{\partial e(w)}{\partial \boldsymbol{w}_{ij}^{(l)}} = x_i^{(l-1)} \boldsymbol{\delta_j}^{(l)}$$

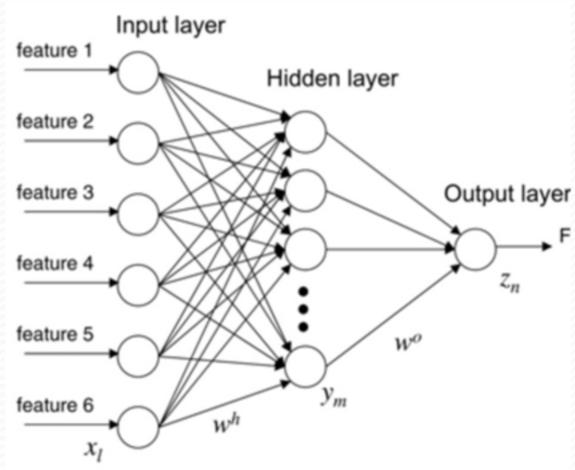
$$\delta_j^{(l-1)} = (1 - (x_i^{(l-1)})^2) \quad \sum_j^{d(l)} \delta_i^{(l)} \quad w_{ij}^{(l)}$$

Deep Learning

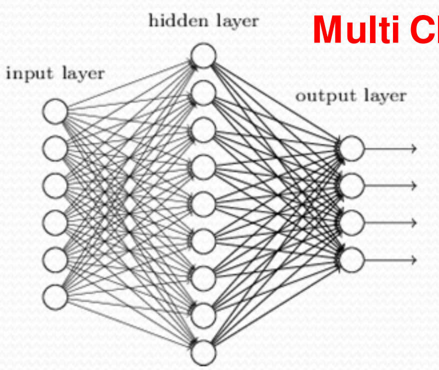
Single Neuron
Binary Class



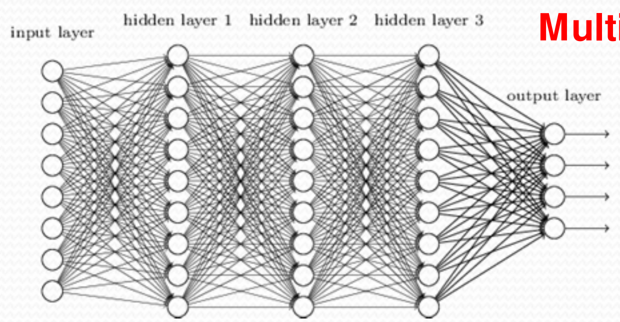
Shallow Learning
Binary Class



Shallow Learning
Multi Class

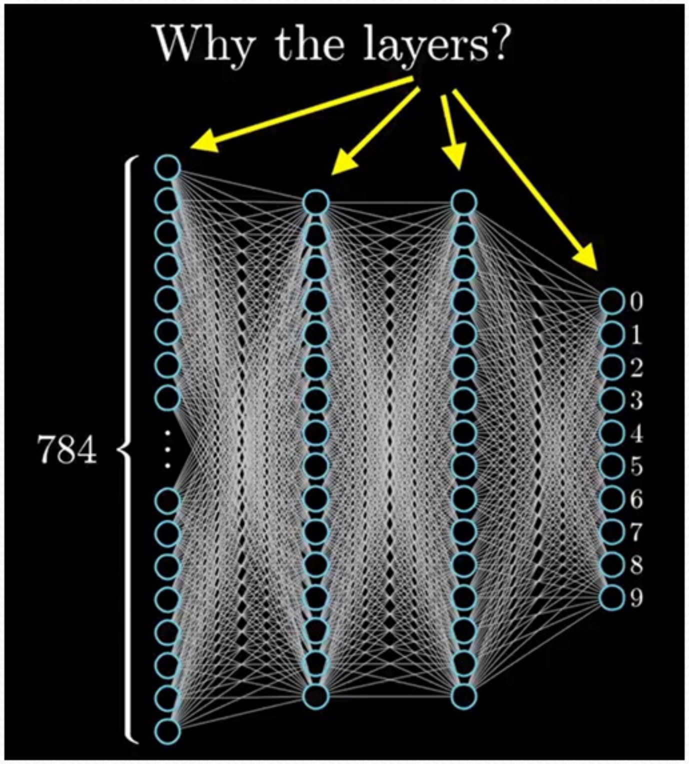


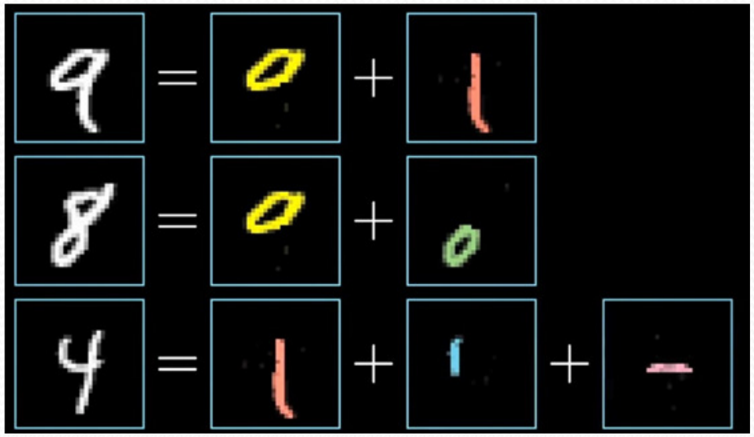
Multi layer
Multi Class

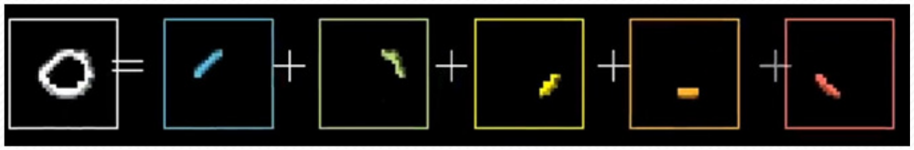


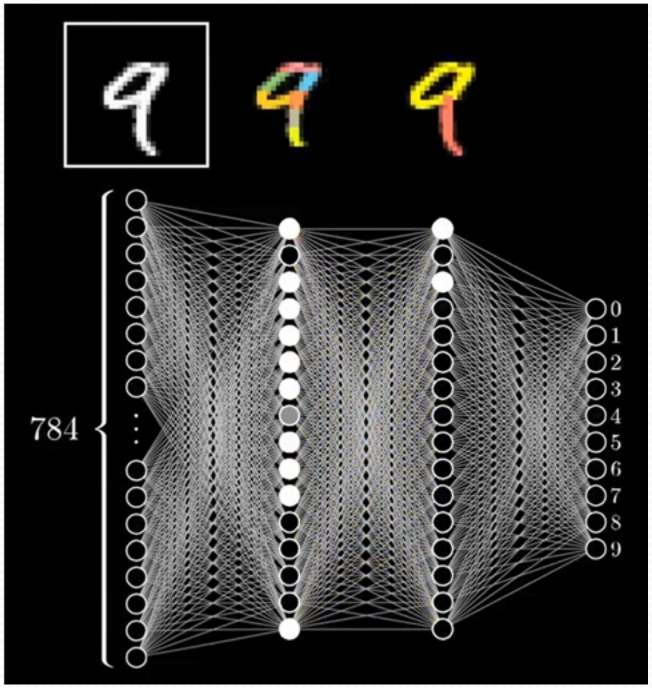
Deep neural Network

Deep Learning
Why more layers?

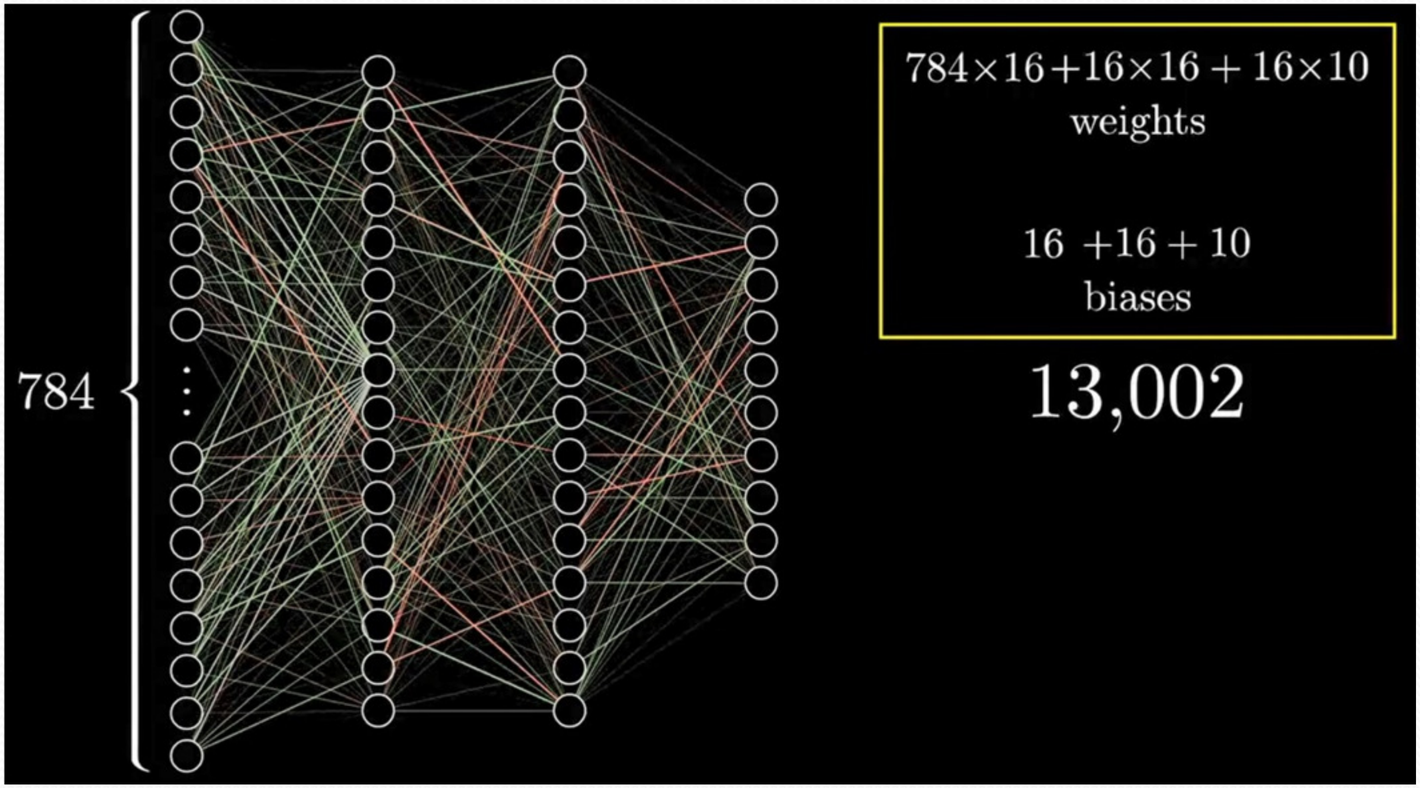






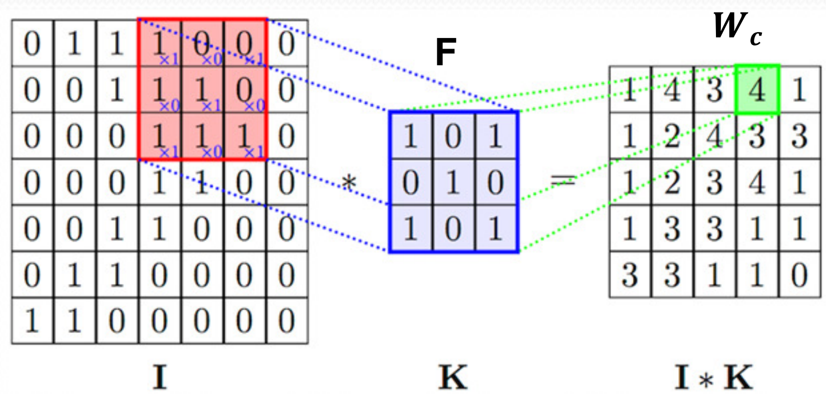


Deep Learning
Large Number of parameters



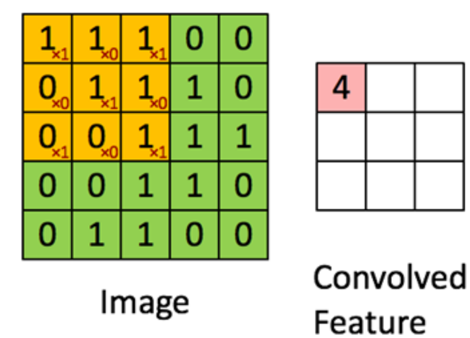
Convolutional Networks (CNN)

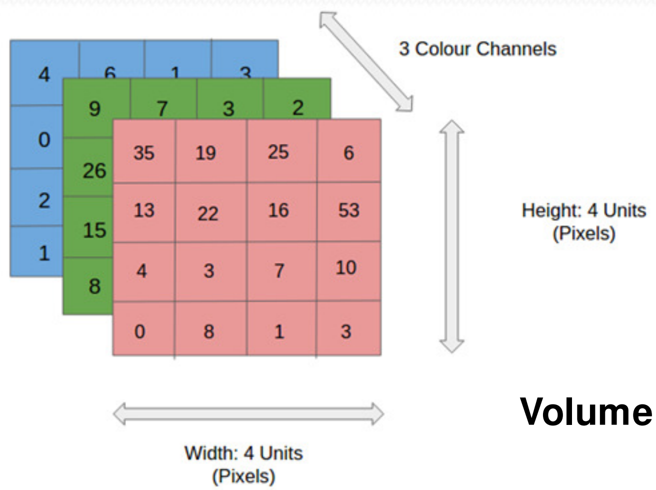
W



$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

I: Image
K: 'Filter' or 'Kernel' or 'Feature Detector'
I*K: Convolved Feature (activation map)





Volume image

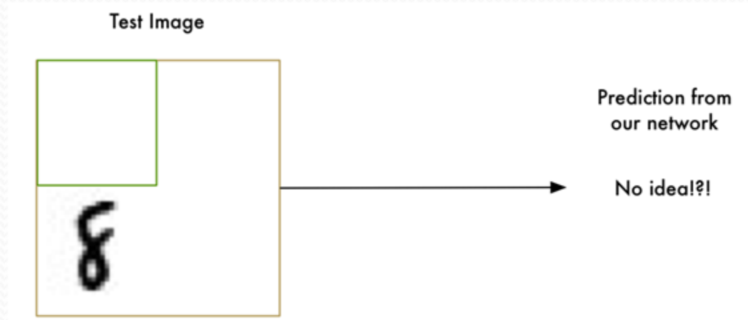
$$W_c = \frac{W - F_W}{S_W} + 1$$

The size of the convolved field (I*K)

S: stride

Ex.: W_c = (7-3)/1+1 = 5

Convolutional Networks (CNN)
Filters

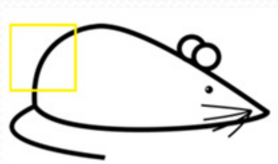


-1	-1	-1
1	1	1
0	0	0

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

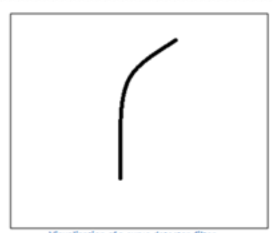
0	0	0
1	1	1
-1	-1	-1

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



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

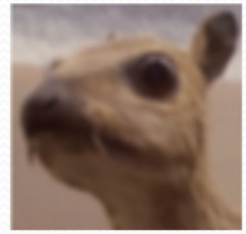


Visualization of a curve detector filter

https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

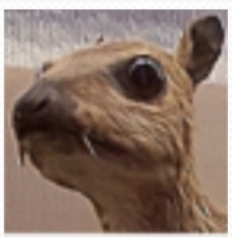
Convolutional Networks (CNN)
Filters

Identity $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

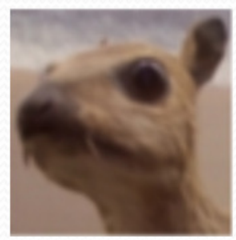


Box blur $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Edge detection $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

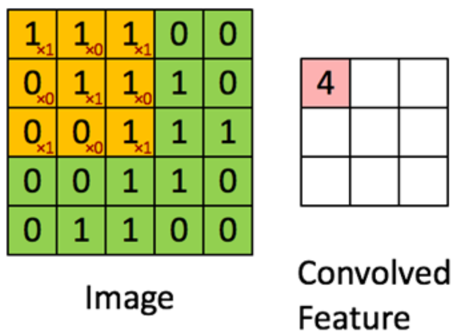


Sharpen $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$



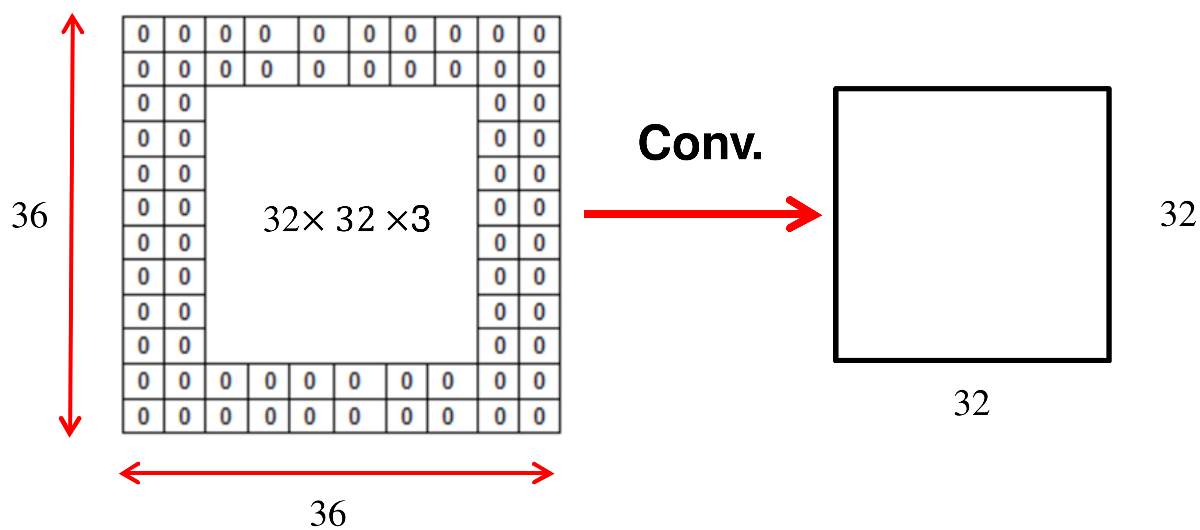
Gaussian blur $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

Stride and Padding

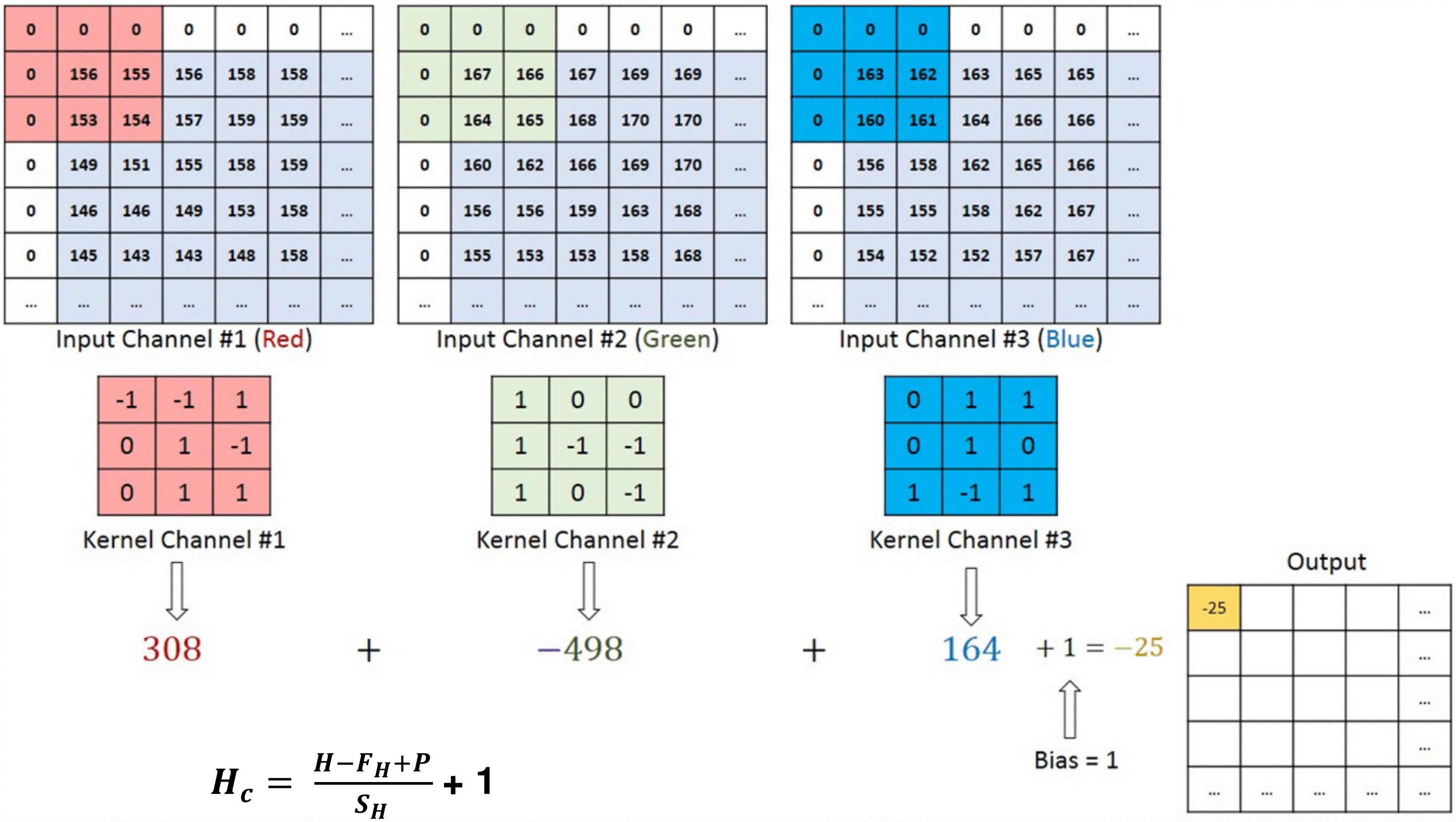


In convolving an image:
1) The outputs shrink
2) The information on corners of the image is lost

This can be prevented by padding.



Stride and Padding
Color Image

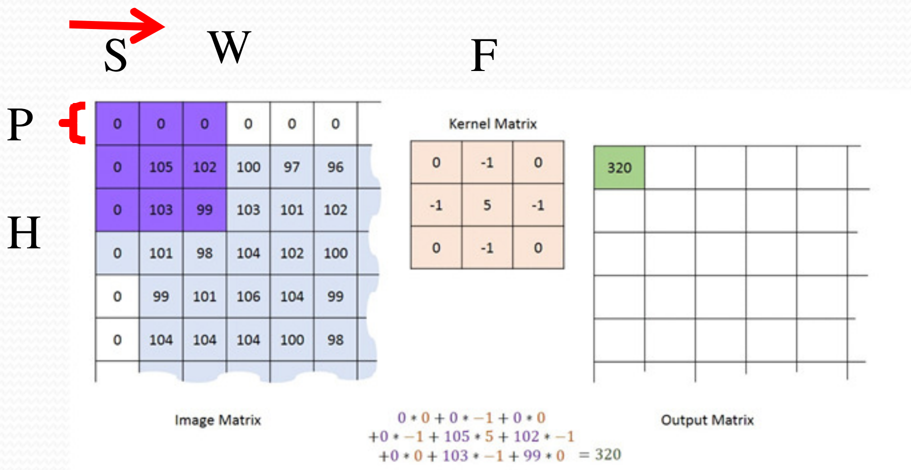


$$H_c = \frac{H - F_H + P}{S_H} + 1$$

$$W_c = \frac{W - F_W + P}{S_W} + 1$$

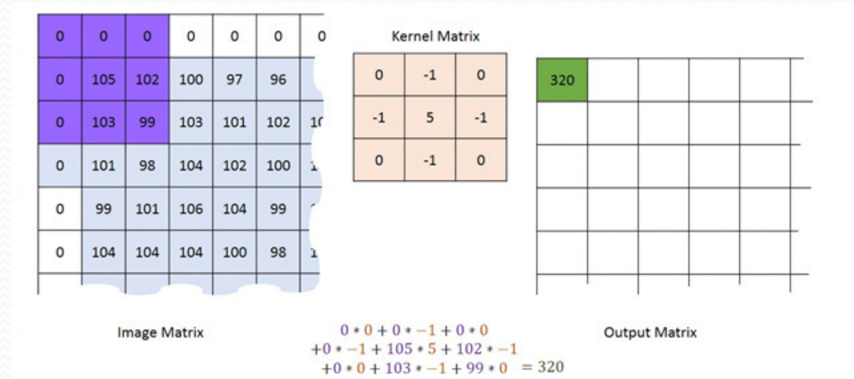
http://machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html

Stride and Padding



Convolution with horizontal and vertical strides = 1

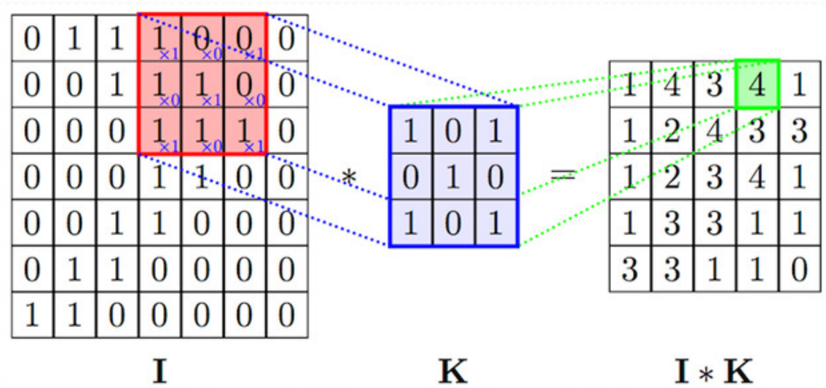
Stride = 1

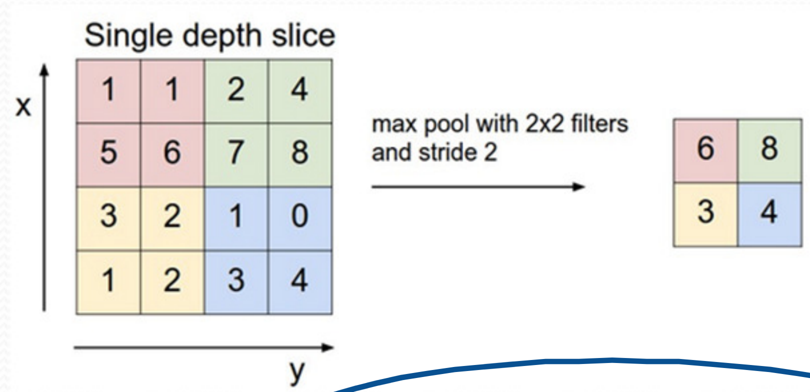


Convolution with horizontal and vertical strides = 2

Stride = 2

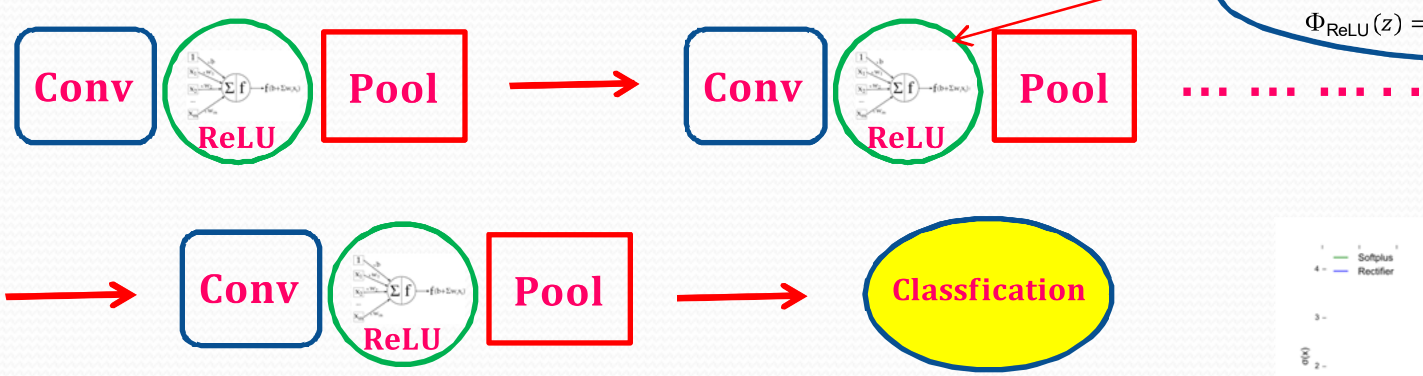
Convolutional Networks
Max Pooling/Downsampling with CNNs

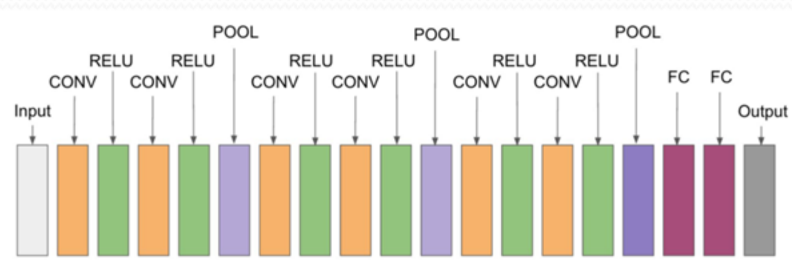


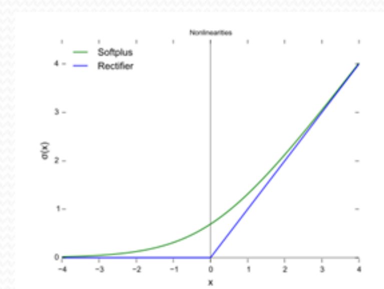


$\mathbf{Z} = \mathbf{X.W} = \sum_1^n w_i x_i = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

$\Phi_{\text{ReLU}}(z) = max(0, z)$



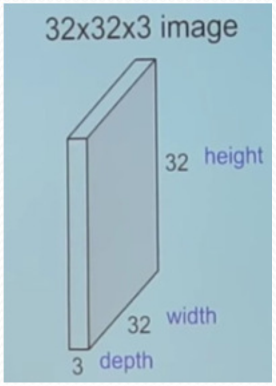


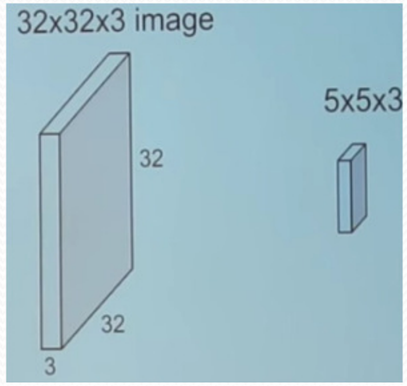


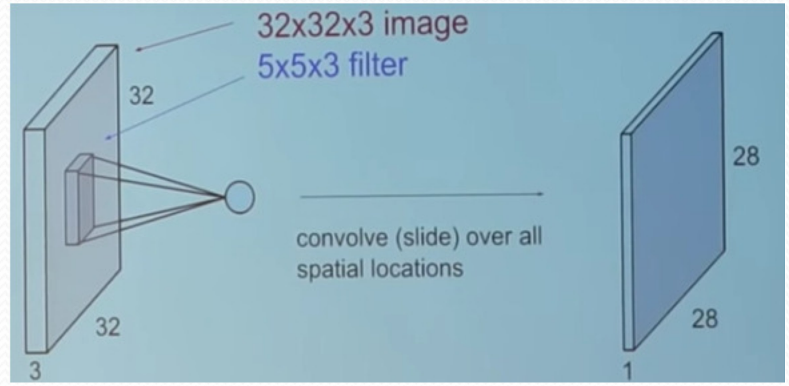
$\Phi_{\text{ReLU}}(z) = max(0, z)$
Rectified Linear Unit (ReLU)
$\Phi_{\text{SReLU}}(z) = log(1+exp(z))$
Analytic approx.

Convolutional Networks (CNN)

W W_c

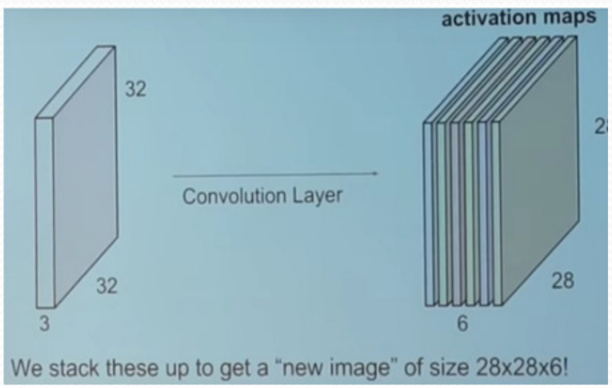






RGB image

Convolving

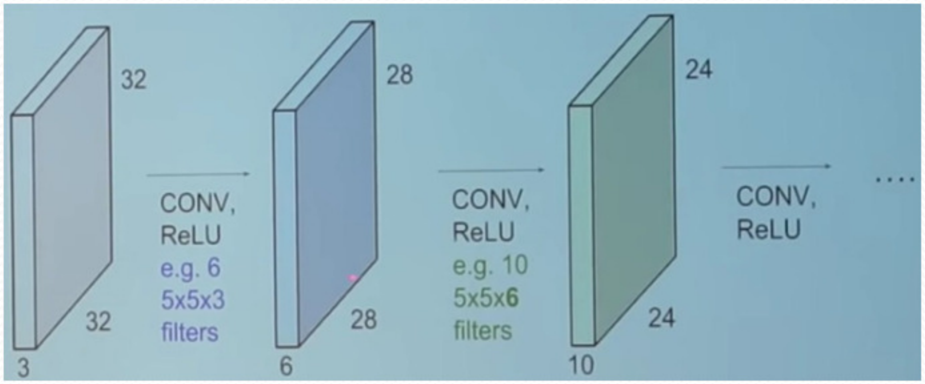


W = 32
W_c = 28

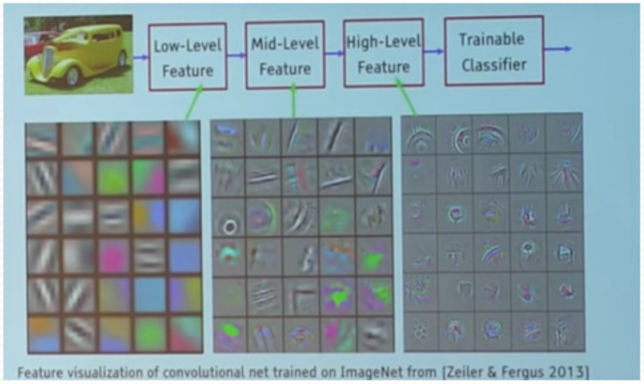
Six independent filters

https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

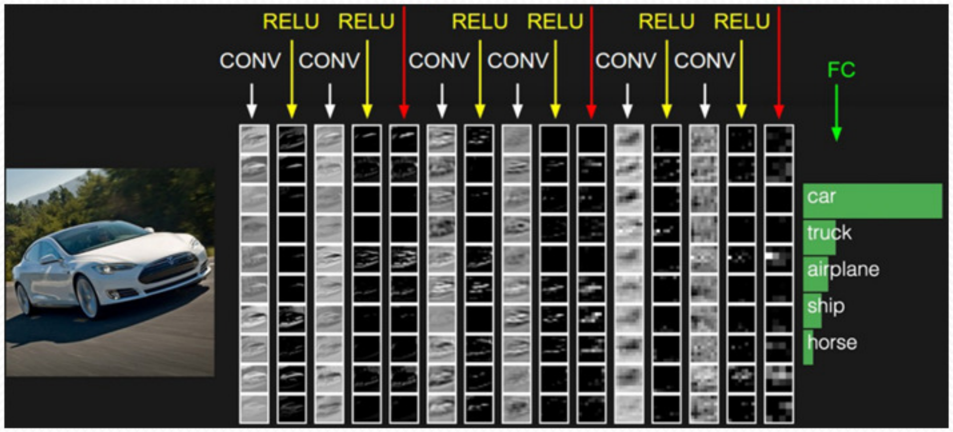
Convolutional Networks (CNN)



Sequence of CNN layers

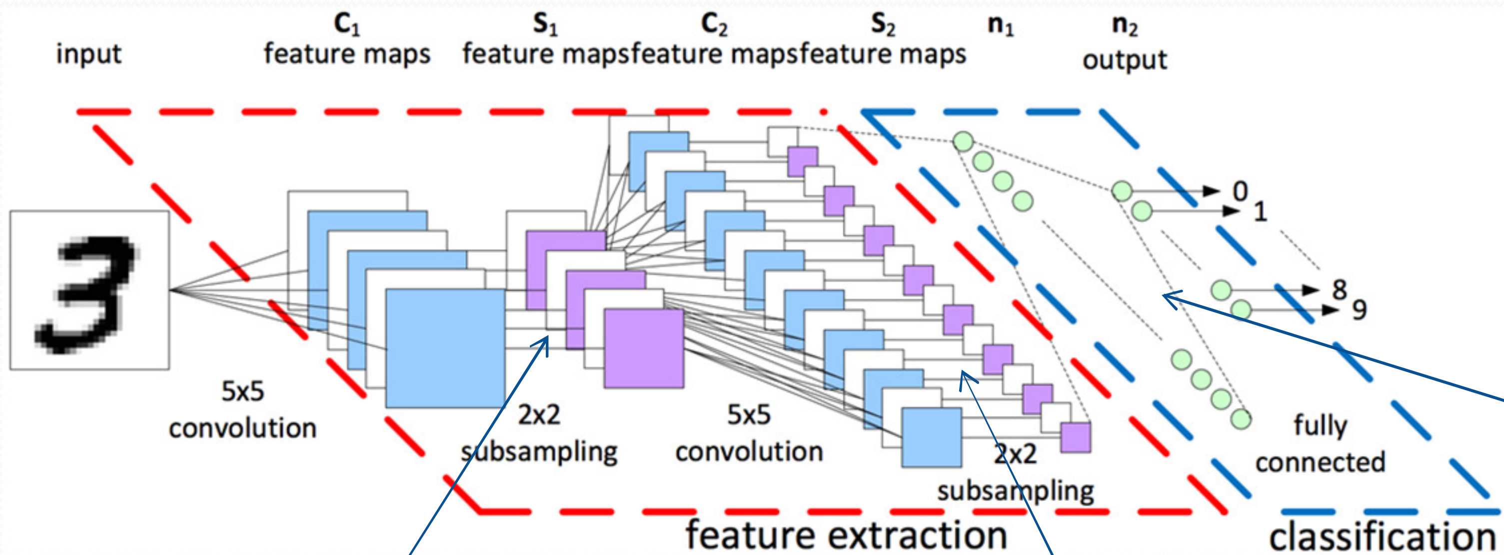


Feature visualization

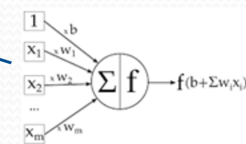


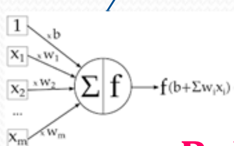
CNN Architecture

Convolutional Networks

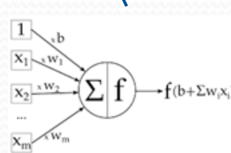


Softmax

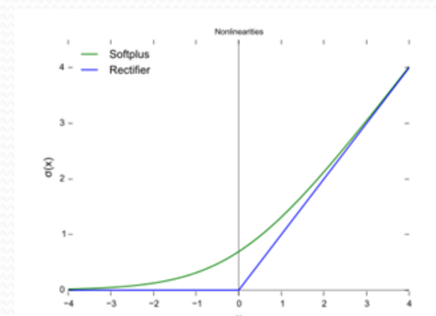




ReLU



ReLU



Softmax function

$$\Phi_{\text{Softmax}}(z_j) = \frac{e^{-zj}}{\sum_m^k e^{-z_m}}$$

k: Number of classes

$\Phi_{\text{ReLU}}(z) = max(0,z)$

Rectified Linear Unit (ReLU)

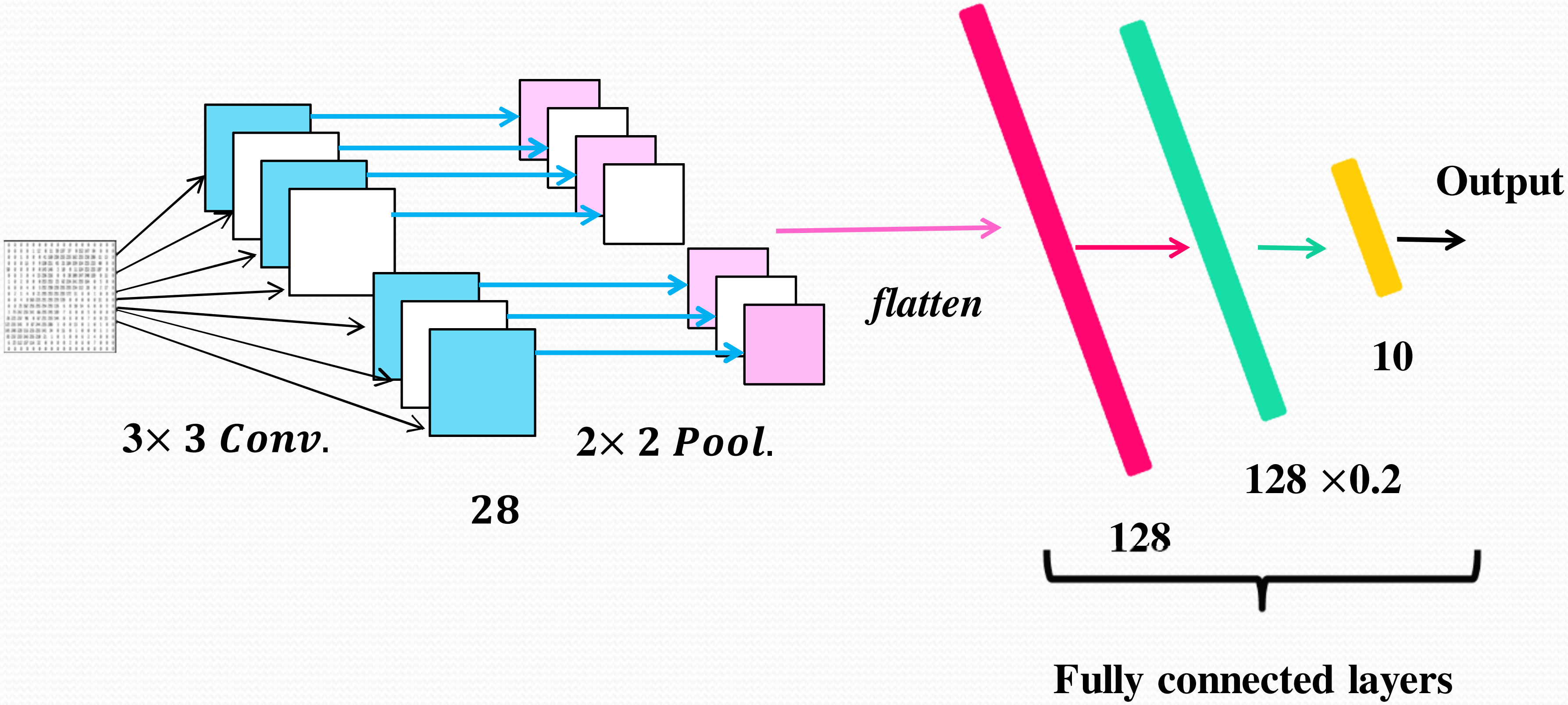
$\Phi_{\text{SReLU}}(z)$= *log(1+exp(z))*

Analytic approx.

https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

Example 1: MNIST Database - Handwritten digits

A simple CNN deep learning model for handwritten digits recognition using Keras,



A Simple CNN Model

```
#https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d
import tensorflow as tf
import sys
#========================================================= import data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()  # Keras integrated into core Tensorflow
print('x_train.shape = ', x_train.shape)
print('y_train.shape = ', y_train.shape)
#========================================================= import data

#========================================================= sample image
import matplotlib.pyplot as plt
#%matplotlib inline # Only use this if using iPython
image_index = 7777 # You may select anything up to 60,000
print('y_train = ', y_train[image_index]) # The label is 8
plt.imshow(x_train[image_index], cmap='Greys')
print('Sample image')
#========================================================= sample image

#========================================================= reshape the image for keras
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)      # grayscale image
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)         # grayscale image
print('x_train.shape[0] = ', x_train.shape[0])
print('x_train.shape2 = ', x_train.shape)
print('x_test.shape2 = ', x_test.shape)

input_shape = (28, 28, 1)

#========================================================= reshape the image for keras

```

A Simple CNN Model

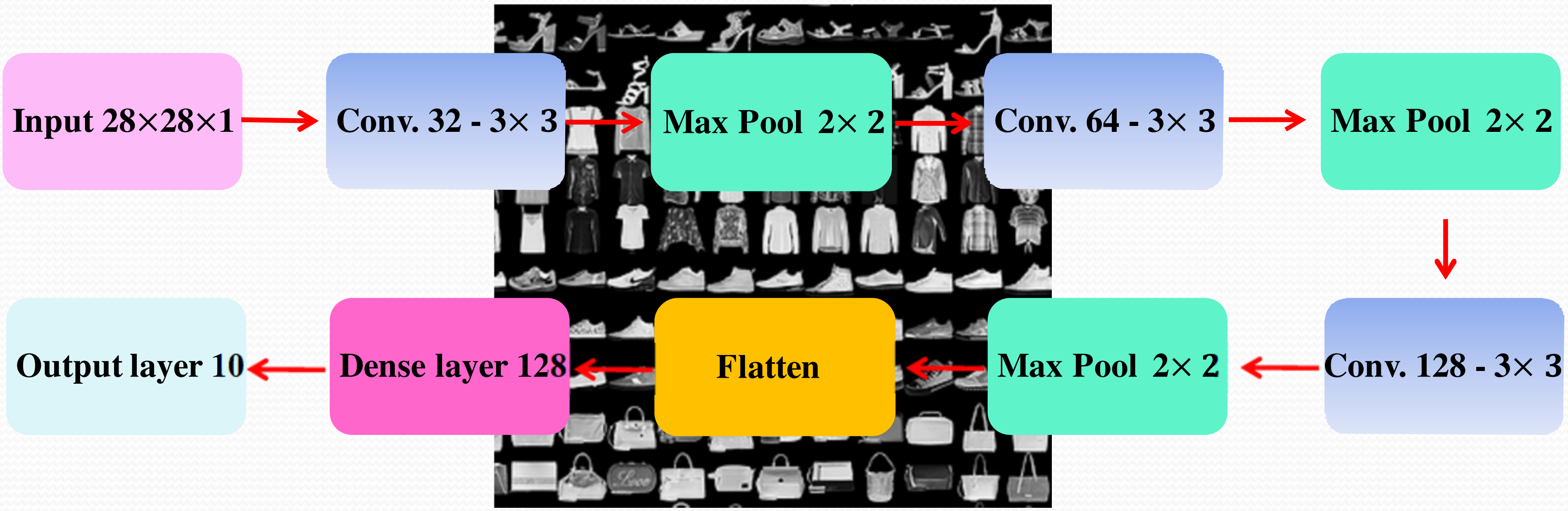
```
#========================================================== change the pixel-values to floating poit
# Making sure that the values are float so that we can get decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
print('x_test[0] = ', x_test[0])
#========================================================== change the pixel-values to floating poit

#========================================================== normalize to (0.0-1.0)
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
print('after normalization: x_test[0] = ', x_test[0])
#========================================================== normalize to (0.0-1.0)

#========================================================== model
# Importing the required Keras modules containing model and layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))    # set the number of filters, the size of kernel,
model.add(MaxPooling2D(pool_size=(2, 2)))                            # imput shape, pool (size and kind)
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))                         # set the number of the nodes just before the output
model.add(Dropout(0.2))                                              # set the amount of droupout
model.add(Dense(10,activation=tf.nn.softmax))                        #set the number of nodes for output

#========================================================== model

#========================================================== complie the model and train using training samples

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)              # use 10 epochs for training
#========================================================== complie the model and train using training samples
```

A Simple CNN Model

```python

#======================================================== use the test samples for evaluation
print('---------------------------------------------------test')
print()
test_eval = model.evaluate(x_test, y_test)
print('test_eval', test_eval)
print()
print('---------------------------------------------------test')
#======================================================== use the test samples for evaluation

#======================================================== plot a sample and find the prediction
img_rows = 28
img_cols = 28
image_index = 4444
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, img_rows, img_cols, 1))
print('---------------------------------------------------- predit a sample')
print()
print('prediction = ', pred.argmax())
print()
print('---------------------------------------------------- predit a sample')
#======================================================== plot a sample and find the prediction
sys.exit()
```

Example 2: Fashion-MNIST Database

CNN deep learning model using Keras,



A Simple CNN Model

```python
# https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
import sys
#========================================================== import data (Fashion-MNIST)
#Load the Data
from keras.datasets import fashion_mnist
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data()
#========================================================== import data (Fashion-MNIST)

#========================================================== check data
print('============================================1')
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
#%matplotlib inline
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)

# Find the unique numbers from the train labels
classes = np.unique(train_Y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)

# ploat
plt.figure(figsize=[15,15])

# Display the first image in training data
plt.subplot(121)
n = 0
plt.imshow(train_X[n,:,:], cmap='gray')
plt.title("Class : {}".format(train_Y[n]))

# Display the first image in testing data
plt.subplot(122)
n = 5
plt.imshow(test_X[n,:,:], cmap='gray')
plt.title("Class : {}".format(test_Y[n]))
print('============================================1')
#========================================================== check data

```

A Simple CNN Model

```python

#========================================================= preprocess data
print('============================================2')
print('train_X.shape = ', train_X.shape)
print('test_X.shape = ', test_X.shape)
train_X = train_X.reshape(-1, 28,28, 1)       # reshape
test_X = test_X.reshape(-1, 28,28, 1)         # reshape

print('train_X.shape = ', train_X.shape)
print('test_X.shape = ', test_X.shape)


train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.

# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
n = 0
print('Original label:', train_Y[n])
print('After conversion to one-hot:', train_Y_one_hot[n])
n = 7
print('Original label:', train_Y[n])
print('After conversion to one-hot:', train_Y_one_hot[n])
print('============================================2')
#========================================================= preprocess data

#========================================================= test-train
print('============================================3')
print('train_X.shape = ', train_X.shape)
print('train_Y_one_hot.shape = ', train_Y_one_hot.shape)
print()
from sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot, test_size=0.2, random_state=13)

```

A Simple CNN Model

```

print('train_X.shape = ', train_X.shape)
print('valid_X.shape = ', valid_X.shape)
print('train_label.shape = ', train_label.shape)
print('valid_label.shape = ', valid_label.shape)
print('============================================3')
#========================================================== test-train

#========================================================== CNN model
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU


batch_size = 64
epochs = 20
num_classes = 10

#Neural Network Architecture
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(28,28,1),padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
#fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
#========================================================== CNN model
```

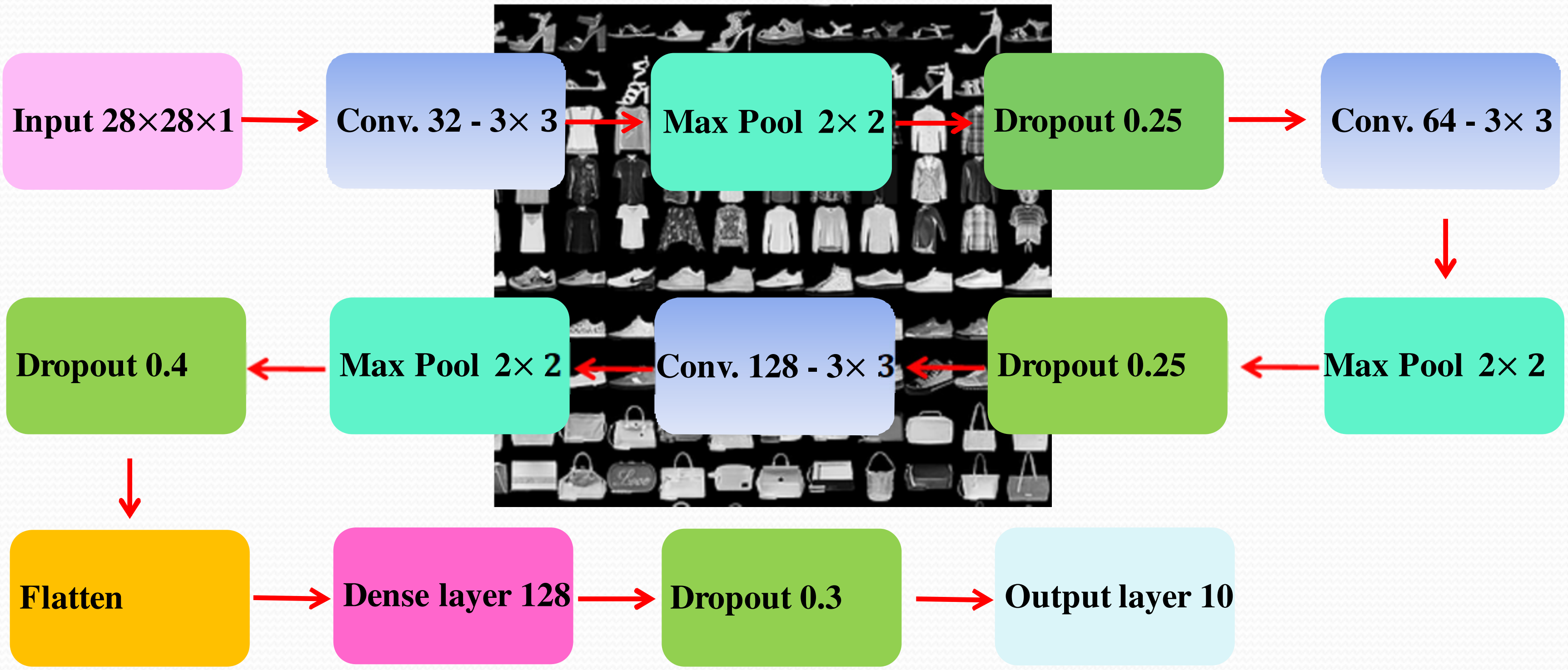
A Simple CNN Model

```python

#========================================================== compile the model

fashion_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
fashion_model.summary()
#========================================================== compile the model

#========================================================== train the model
fashion_train = fashion_model.fit(train_X, train_label, batch_size=batch_size,epochs=epochs,verbose=1,
                                  validation_data=(valid_X, valid_label))
#========================================================== train the model

#========================================================== evaluate the model
test_eval = fashion_model.evaluate(test_X, test_Y_one_hot, verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
#========================================================== evaluate the model

#========================================================== ploat the evaluation results
accuracy = fashion_train.history['acc']
val_accuracy = fashion_train.history['val_acc']
loss = fashion_train.history['loss']
val_loss = fashion_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
#========================================================== ploat the evaluation results
```

A Simple CNN Model

Example 3: Fashion-MNIST Database

CNN deep learning model with dropout using Keras,

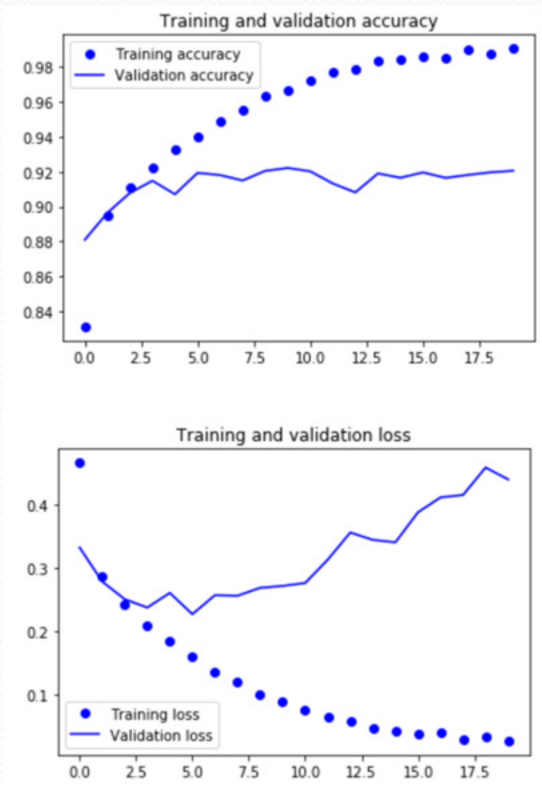


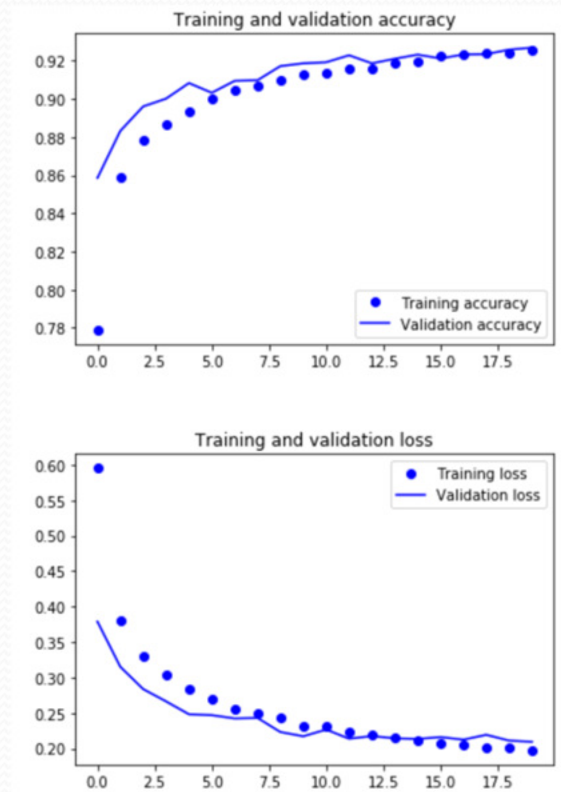
A Simple CNN Model

```python
#========================================================== CNN model with dropout
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU

# Adding Dropout into the Network
batch_size = 64
epochs = 20
num_classes = 10

#Neural Network Architecture
fashion_model = Sequential()
fashion_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(28,28,1)))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4))
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))
fashion_model.add(Dense(num_classes, activation='softmax'))
#========================================================== CNN model with dropout
```

Overcome Overfitting with Dropout

Fashion MNIST Data





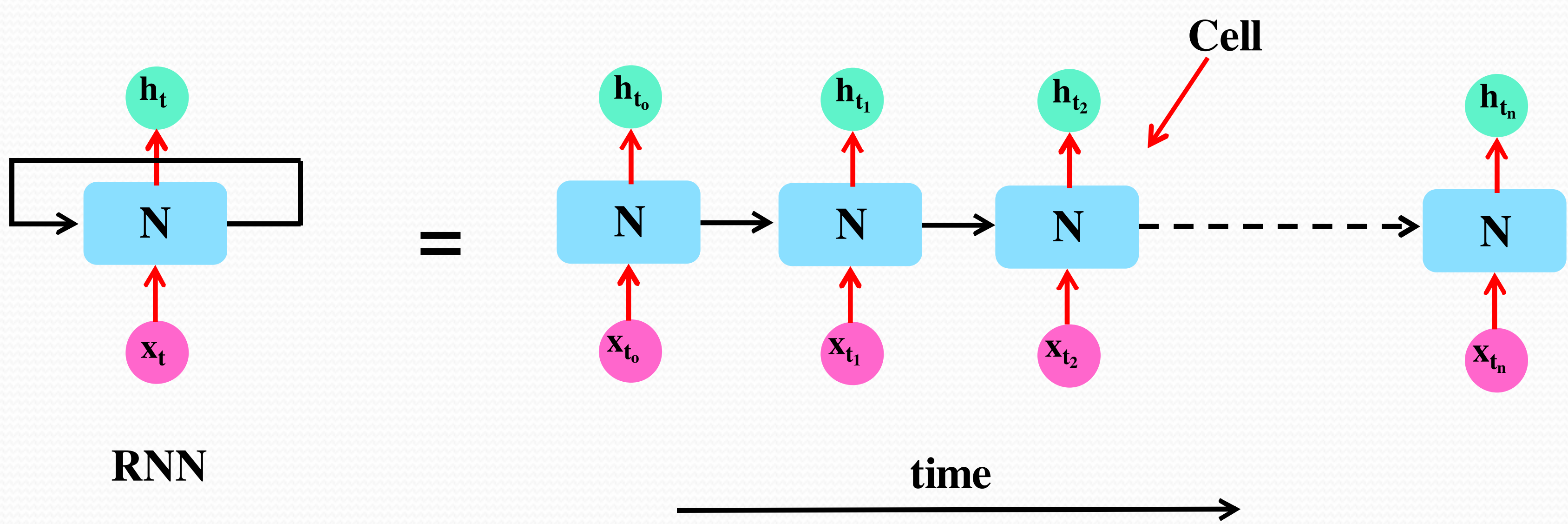
Without dropout **With dropout**

https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

Recurrent Neural Networks (RNN)

RNN:

Unlike the regular neural networks in which the samples are assumed to be time independent, being inputted to the network as a whole, the recurrent neural networks, take their inputs from temporally distributed samples. They can be thought as multiple copies of the same network, each passing a message to a successor. These networks have loops in them, allowing time-dependent information to persist.



Recurrent Neural Networks (RNN)

Example 1: **Classification of events happening at every frame in a movie. RNN can use its reasoning about the previous events in the film to inform later ones.**

Example 2: **Earthquakes and their relevance to the previous earthquakes.**

Extension: **The application of recurrent neural networks is not limited the time dependent samples. The application can be extended to the other spaces (other than time) as can be seen from the following example.**

Example 3: **MNIST- digit recognition using RNN**
In this example each 28×28-pixel image of handwritten digits, instead of being flattened to a 784-dimensional array as input for a regular neural network, is assumed as 28 one-dimensional images. Each row of pixels of the image is assumed to be a single 1D-image , and the next row another image, following the previous 1D-image. Each row of the image is then sent to one cell of the recurrent neural network in sequence.
The following code trains a model using RNN algorithm. Note that each row has some information to the next row which is not preserved in the regular neural networks.

MNIST- Digit Recognition - RNN

```python
#https://github.com/CSCfi/machine-learning-scripts/blob/master/notebooks/keras-mnist-rnn.ipynb
#===================================================================
#%matplotlib inline
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.layers.recurrent import SimpleRNN, LSTM, GRU
from keras.utils import np_utils
from keras import backend as K

from distutils.version import LooseVersion as LV
from keras import __version__

from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

print('Using Keras version:', __version__, 'backend:', K.backend())
assert(LV(__version__) >= LV("2.0.0"))
import sys
import os
import tensorflow as tf
#===================================================================

#=================================================================== import data
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
nb_classes = 10
img_rows, img_cols = 28, 28
my_batch_size = 128
epochs = 3

nb_units = 50 # Number of hidden units to use
nb_units = 128 # Number of hidden units to use

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

MNIST- Digit Recognition - RNN

```
X_train /= 255
X_test /= 255

# one-hot encoding:
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

print()
print('MNIST data loaded: train:',len(X_train),'test:',len(X_test))
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('Y_train:', Y_train.shape)
#====================================================================== import data

#====================================================================== RNN model
model = Sequential()
# Recurrent layers supported: SimpleRNN, LSTM, GRU:
model.add(SimpleRNN(nb_units,
                    input_shape=(img_rows, img_cols)))

# To stack multiple RNN layers, all RNN layers except the last one need
# to have "return_sequences=True".  An example of using two RNN layers:
#model.add(SimpleRNN(16,
#                    input_shape=(img_rows, img_cols),
#                    return_sequences=True))
#model.add(SimpleRNN(32))

model.add(Dense(units=nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print(model.summary())
#====================================================================== RNN model
```

MNIST- Digit Recognition - RNN

```
#====================================================================
SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
#====================================================================

#==================================================================== train
history = model.fit(X_train,
                    Y_train,
                    epochs=epochs,
                    batch_size=my_batch_size,
                    verbose=2)
#==================================================================== train

#==================================================================== plot loss & accuracy
plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('accuracy');
#==================================================================== plot loss & accuracy

#==================================================================== scores
scores = model.evaluate(X_test, Y_test, verbose=2)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
#==================================================================== scores

#==================================================================== ploting function
def show_failures(predictions, trueclass=None, predictedclass=None, maxtoshow=10):
    rounded = np.argmax(predictions, axis=1)
    errors = rounded!=y_test
    print('Showing max', maxtoshow, 'first failures. '
          'The predicted class is shown first and the correct class in parenthesis.')
```

MNIST- Digit Recognition - RNN

```
    ii = 0
    plt.figure(figsize=(maxtoshow, 1))
    for i in range(X_test.shape[0]):
        if ii>=maxtoshow:
            break
        if errors[i]:
            if trueclass is not None and y_test[i] != trueclass:
                continue
            if predictedclass is not None and predictions[i] != predictedclass:
                continue
            plt.subplot(1, maxtoshow, ii+1)
            plt.axis('off')
            plt.imshow(X_test[i,:,:], cmap="gray")
            plt.title("%d (%d)" % (rounded[i], y_test[i]))
            ii = ii + 1
#====================================================================== ploting function

#=========================================================== preditions for the test samples

predictions = model.predict(X_test)
print('predictions.shape = ', predictions.shape)
#=========================================================== preditions for the test samples

#=========================================================== accuracy
save_path = os.path.dirname(os.path.abspath(__file__))
print(save_path)
y_pred = model.predict(X_test)
name_of_file = 'Pred_Obs'
completeName = os.path.join(save_path, name_of_file+".dat")
file1 = open(completeName, "w")

acc = 0.0
print(y_test[5])
print(y_pred[5])
print('y_test.shape = ', y_test.shape)
print('y_Pred.shape = ', y_pred.shape)
```

MNIST- Digit Recognition - RNN

```python

class_labels_pred = np.argmax(y_pred, axis=1)
print('=======class_labels_pred=========', class_labels_pred)
print('=======class_labels_pred.shape=========', class_labels_pred.shape)

for j in range (y_test.size):
    Y_o = y_test[j]
    Y_p = class_labels_pred[j]
    file1.write("%6.3f  %6.3f  " % (Y_p, Y_o))
    if (Y_p==Y_o):
        acc = acc + 1.0
    file1.write(" \n " )
file1.close();

print('acc = ', acc/y_test.size)
#========================================================== accuracy

#=====================================================================
'''
The first 10 test digits the RNN classified to a wrong class.
'''
show_failures(predictions)
#=====================================================================

#=====================================================================
'''
Failures in which the true class is "6".
'''
show_failures(predictions, trueclass=6)
#=====================================================================
sys.exit()
#=====================================================================
```

Optimizers are a crucial part of the neural networks

Batch Gradient Descent (BGD)

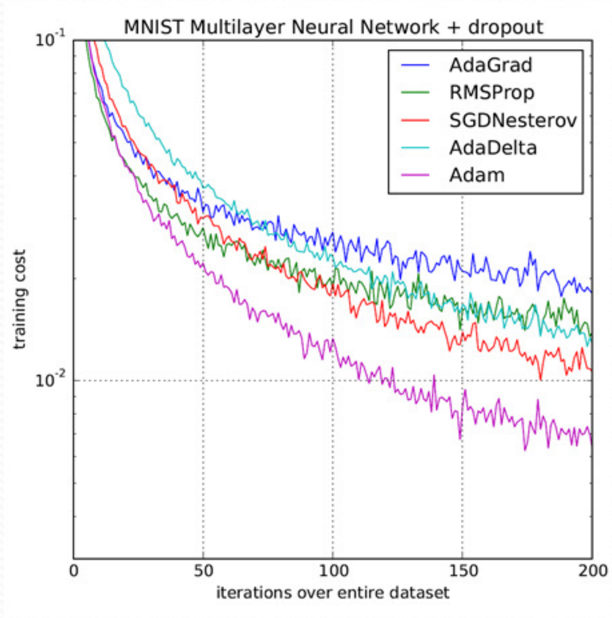
$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^i - \phi(z^i)\right) x^i{}_j$$

Mini-Batch Gradient Descent (BGD)

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_{i=1}^{k \ll n} \left(y^i - \phi(z^i)\right) x^i{}_j$$

Stochastic Gradient Descent (SGD)

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \left(y^i - \phi(z^i)\right) x^i{}_j$$



i: sample, j: feature

Optimization

AdaGrad

AdaGrad (adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate (adaptively tuned per parameter).

$$w_j = w_j - \frac{\eta}{\sqrt{G_{jj}}} g_j, \qquad G_{jj} = \sum_{\tau=1}^{t} g^2{}_{\tau j}, \qquad g_\tau = \nabla j_i\,(w)$$

RMSProp optimization algorithm

RMSProp (Root Mean Square Propagation) optimization algorithm (Kingma & Ba, 2015) is an update to the RMSProp optimizer.

$$J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$ **sample objective function**

$$\mathbf{w} = \mathbf{w} + \Delta w,$$

$$s^{t+1}{}_{dw} = \beta\, s^{t}{}_{dw} + (1-\beta)\left(\frac{\partial j^t}{\partial w}\right)^2, \qquad \widehat{s_{dw}} = \frac{s^{t+1}{}_{dw}}{1-\beta^{t+1}}$$

$$w^{t+1} = w^t - \eta \frac{\frac{\partial j^t}{\partial w}}{\sqrt{\widehat{s_{dw}}} + \epsilon}$$

t: time step, i: sample, j: feature

Optimization

Adam optimization algorithm

Adam (Adaptive moment estimation) optimization algorithm (Kingma & Ba, 2015) is an update to the RMSProp optimizer.

1) **Computationally efficient**
2) **Little memory requirements**
3) **Well suited for large data**

$$v^{t+1}{}_{dw} = \beta_1 v^{t}{}_{dw} + (1-\beta_1)\frac{\partial j^t}{\partial w}, \qquad \widehat{v_{dw}} = \frac{v^{t+1}{}_{dw}}{1-\beta^{t+1}{}_1}$$

$$s^{t+1}{}_{dw} = \beta_2 s^{t}{}_{dw} + (1-\beta_2)\left(\frac{\partial j^t}{\partial w}\right)^2, \qquad \widehat{s_{dw}} = \frac{s^{t+1}{}_{dw}}{1-\beta^{t+1}{}_2}$$

$$w^{t+1} = w^t - \eta\frac{\widehat{v_{dw}}}{\sqrt{\widehat{s_{dw}}+\epsilon}}$$

Optimization

Momentum (Modified SGD)

Stochastic gradient descent with momentum remembers the update Δw at each iteration, and determines the next update as a linear combination of the gradient and the previous update.

$$\Delta w^t = \alpha \Delta w^{t-1} - \eta \, \nabla J(w)^{\mathrm{i}}$$
$$w^t = w^{t-1} - (\alpha \Delta w - \eta \, \nabla J(w)^{\mathrm{i}})$$

SGD

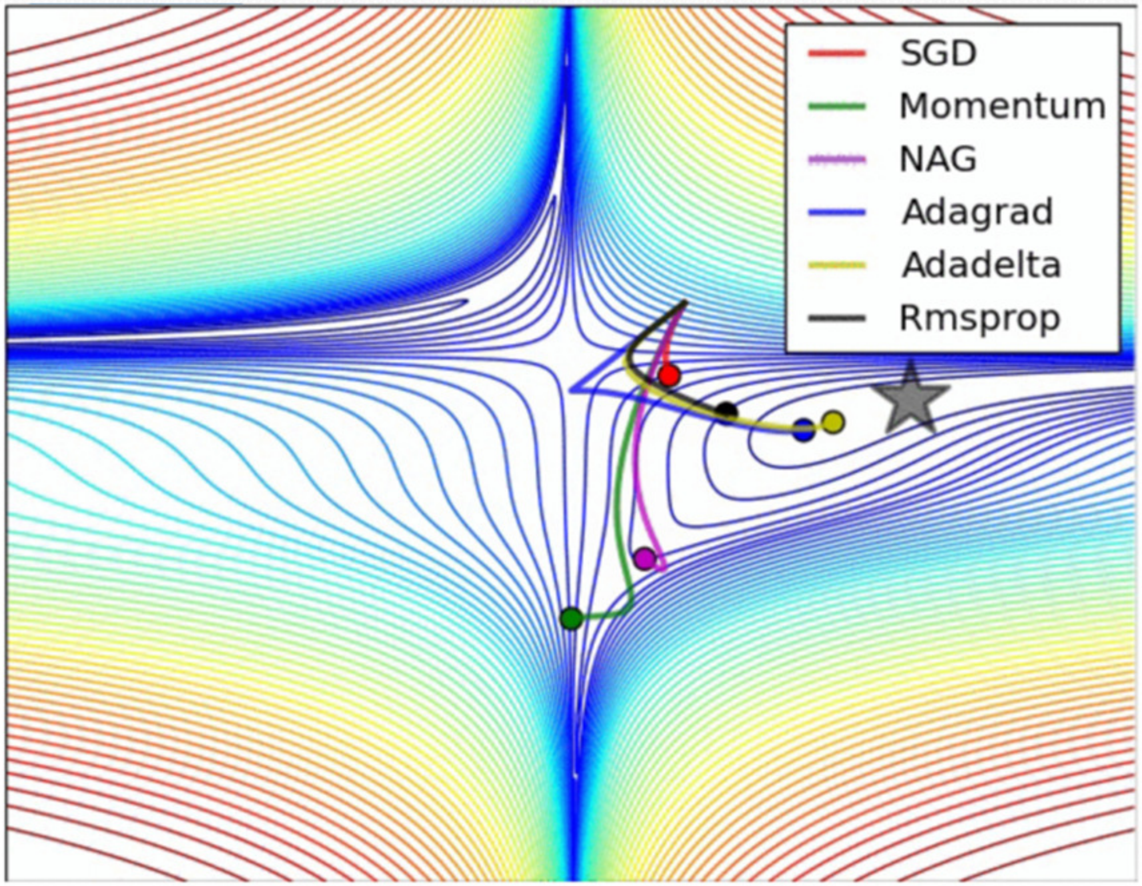
Momentum

Implicit Stochastic Gradient Descent (ISGD)

SGD is generally sensitive to learning rate η. Fast convergence requires large learning rates but this may induce numerical instability. The problem can be largely solved by considering implicit updates whereby the stochastic gradient is evaluated at the next iterate rather than the current one.

$$w^{new} = w^{old} + \Delta w^{new}$$

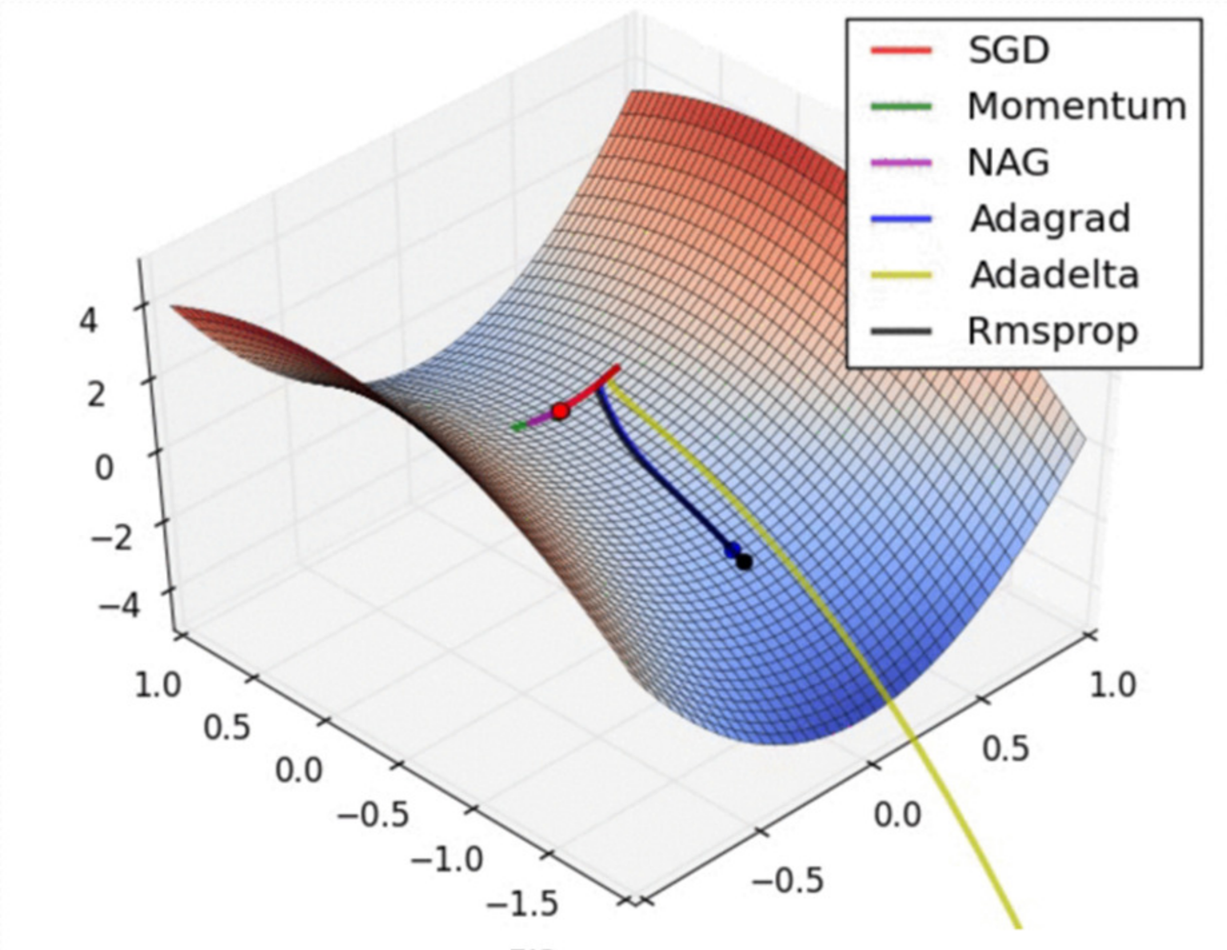
Optimization



https://stackoverflow.com/questions/36162180/gradient-descent-vs-adagrad-vs-momentum-in-tensorflow

Saddle Point Problem

$$\frac{\partial j}{\partial w} = 0$$



https://stackoverflow.com/questions/36162180/gradient-descent-vs-adagrad-vs-momentum-in-tensorflow