Deep Neural Network for Classification of Breast Cancer Type based on Cellular Nuclei Features

Tirthajyoti Sarkar, Ph.D.
Sr. Principal Engineer, ON Semiconductor, Sunnyvale, CA
Email: tirthajyoti@gmail.com

Term paper submitted in the Stanford Continuing Education SCI-52 class:
Artificial Intelligence: An Introduction to Neural Networks and Deep Learning

Abstract: In this paper, we discuss the use of a densely-connected 2-layer(hidden) neural network for the classification task of breast cancer malignancy type from the Wisconsin cancer research center's dataset. In this dataset, features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Basic exploratory data analysis and visualization are done followed by building 2-layer neural network (using Keras framework in Python with a TensorFlow backend). Suitable classification metrics are computed from the test results and impact of some hyperparameters the quality of classification task is demonstrated through simulation.

Keywords: Deep learning, Cancer, Classification, Neural network, scaling, Confusion matrix, F1 score.

Introduction

Andrew Ng said *"Artificial Intelligence (AI) is the new electricity"*. If that is the case, then undoubtedly, Deep Neural Networks (DNN) are the new oil for the modern engine of this electricity generator. The basic idea—that software can approximately emulate the collective behavior and firing patterns of neocortex's large array of neurons in an artificial "neural network"—is decades old, and it has led to as many disappointments as breakthroughs. But because of improvements in mathematical formulas and increasingly powerful computers, computer scientists can now model many more layers of virtual neurons than ever before.

Inventors have long dreamed of creating machines that think. This desire dates back to at least the time of ancient Greece. When programmable computers were first conceived, people wondered whether such machines might become intelligent, over a hundred years before one was even built. Today, AI

is a thriving field with many practical applications and active research topics. We look to intelligent software to automate routine labor, understand speech or images, make diagnoses in medicine and support basic scientific research.

Deep learning (DL) has, in recent years, accelerated AI research and innovation, to a break taking pace. In some way, it is not a brand new technology - at its core it is a resurgent form of the old ***multi-layer perceptrons***, as advanced by the likes of **Marvin Minsky, John McCarthy, Frank Rosenblatt, Alexey Ivakhnenko, V.G. Lapa,** in 1950s and 60s. However, new algorithms, advanced graphics-processing-unit (GPU), innovative data engineering, and massive trove of Big Data, have enabled Deep Learning to emerge as the leader among all machine learning techniques that have the potential of creating a truly *intelligent machine* in near future.

In this article, we show how to model and use a simple 2-hidden-layer neural network to predict the class of breast cancer from the measurements of tumor images and their descriptive statistics. We use the famous Wisconsin Cancer dataset for this purpose which can be downloaded from University of California, Irvine Machine Learning repository. The model shows high *accuracy, precision, recall*, and *F1 score* when the data is properly preprocessed and the model is trained for sufficient number of epochs.

The Dataset: Features and exploratory analysis

This particular data set was created by imaging breast cancer fluid nuclei from patients in a simple outpatient procedure. First, a sample of fluid is taken from patient's breast. This procedure involves using a small-gauge needle to take the fluid, known as fine-needle-aspirate (FNA), directly from a breast lump or mass, the lump having been previously detected by self-examination or mammography. The fluid from the FNA is placed on a glass slide and sainted to highlight the nuclei of the constituent cells. An image from the FNA is transferred to a workstation by a video camera mounted on a microscope. An in-house program called Xcyt, uses curve-fitting techniques to determine the boundaries and geometric properties of the nuclei. Ten features are computer for each nucleus:

area, radius, perimeter, symmetry, number and size of concavities, fractal dimension of the boundary, compactness, smoothness (local variation of radial segments), and texture (variance of gray level inside the boundary).

The mean value, extreme value, and standard error of each of these cellular features are also computed resulting in 30 real-valued features for each image. In the public dataset, 569 such images are available. The data set can be downloaded from UCI ML repo. However, this data set is also readily available from Python's Scikit-Learn machine learning library utility tools and we use one simple command to import the dataset into our program.

Import necessary Python packages and methods¶

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import binarize
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Import the data set using scikit-learn's utility function

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

Create a Pandas DataFrame from the raw data¶

```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
```

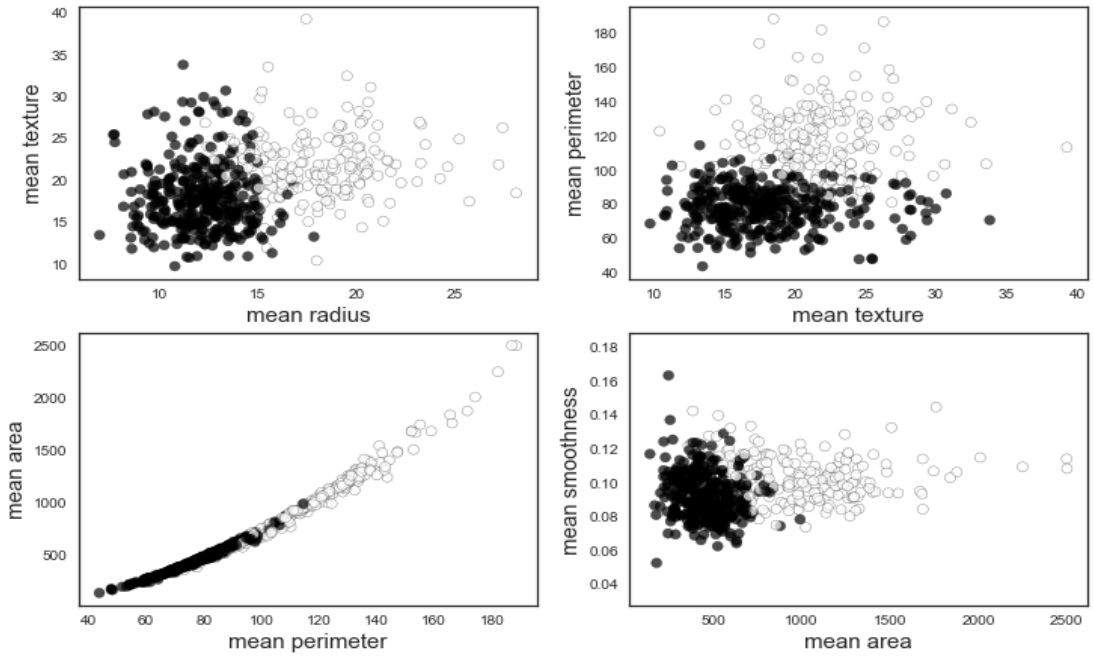
Append the 'target' i.e. response variable to the Data Frame and show few examples¶

```
df['Cancer'] = pd.DataFrame(cancer['target'])
df.head(5).transpose()[:10]
```

	0	1	2	3	4
mean radius	17.99000	20.57000	19.69000	11.42000	20.29000
mean texture	10.38000	17.77000	21.25000	20.38000	14.34000
mean perimeter	122.80000	132.90000	130.00000	77.58000	135.10000
mean area	1001.00000	1326.00000	1203.00000	386.10000	1297.00000
mean smoothness	0.11840	0.08474	0.10960	0.14250	0.10030
mean compactness	0.27760	0.07864	0.15990	0.28390	0.13280
mean concavity	0.30010	0.08690	0.19740	0.24140	0.19800
mean concave points	0.14710	0.07017	0.12790	0.10520	0.10430
mean symmetry	0.24190	0.18120	0.20690	0.25970	0.18090
mean fractal dimension	0.07871	0.05667	0.05999	0.09744	0.05883

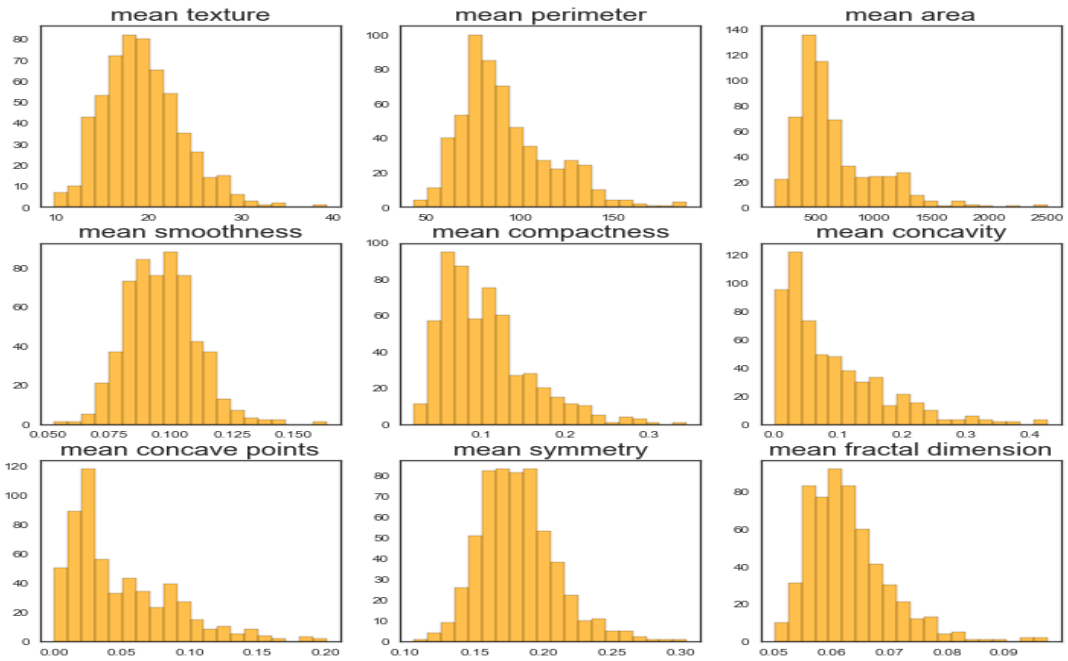
Visualizations - *is there linear separability*?

Exploratory analysis with visualizations is always a good place to start any machine learning task. We create some pairwise scatter plots with cancer type (benign or malignant) marked by color. Dark color points denote malignant cancer cases and light color ones denote benign cases. It is noted that the **points are not very well separated or linearly separated by these measurement features** i.e. simple linear classifier will not be able to diagnose malignant and benign cases with high accuracy based on the image measurements.



Distributions of the features

We plot the distributions/histograms of some of the features. Most of them are either left- or right-skewed.



The Dataset: Preprocessing

In statistics and machine learning we usually split our data into to subsets: **training data** and **testing data** (and sometimes to three: **train, validate and test**), and fit our model on the train data, in order to make predictions on the test data. This is a mandatory step to avoid overfitting our data i.e. model fits to noise in the data rather than to the hidden pattern. **Test data should never be seen by the model except at the time of final test.** Validation set can be used to fine tune the hyperparameters of the model.

We choose a test fraction of 0.3 i.e. 30% and equally divide the rest of the data into traing set and validation set. We use Scikit-Learn's `train_test_split` function to create this random split.

We create multi-dimensional vectors (Python `Numpy` objects) from the Dataset to feed properly to the neural network model later - `feature` and `response`vectors. We also scale this data using Scikit-Learn's `preprocessing.scale()` function. This scales the data by dividing each data point by the range of that series (difference between the max and min).

Create features and response vectors

```
X=df.drop(['Cancer'],axis=1)
y=df['Cancer']
```

Preprocessing - scaling the data

```
X=preprocessing.scale(X)
```

Train/Test/Validation set split

```
test_frac=0.3
val_frac=0.5
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_f
rac)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_s
ize=val_frac)
input_dim=X.shape[1]
```

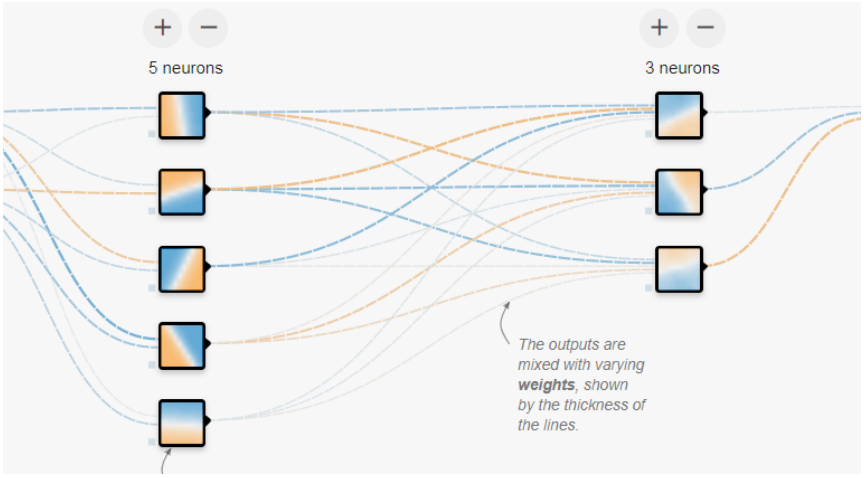
Classification with the Neural Network

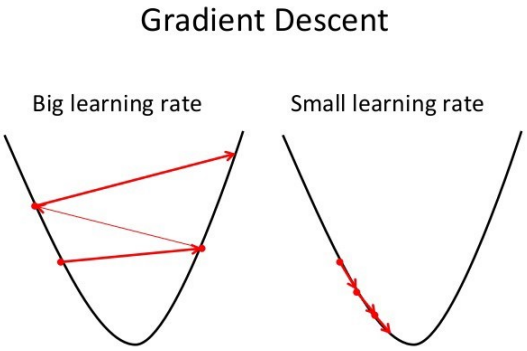
We use Keras framework to model our neural network. Keras is an open source neural network modeling library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

```
import keras
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

Model Hyperparameters

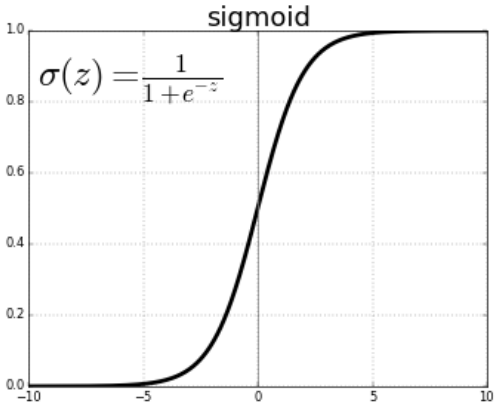
Hyperparameters are the variables which determines the network structure (e.g: **number of layers** and **number of hidden units**) and the variables which determine how the network is trained (e.g. **learning rate**). Hyperparameters are set before training (before optimizing the weights and bias). For the first pass, we choose 5 neurons in the first layer and 3 neurons in the second layer.

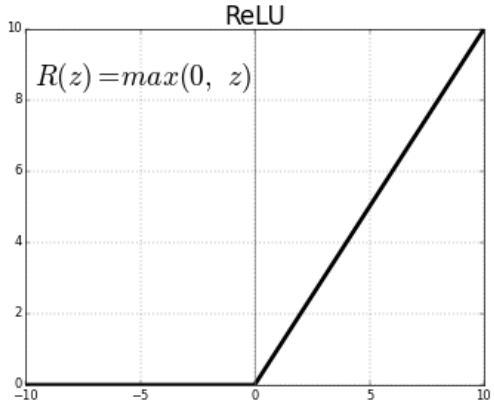




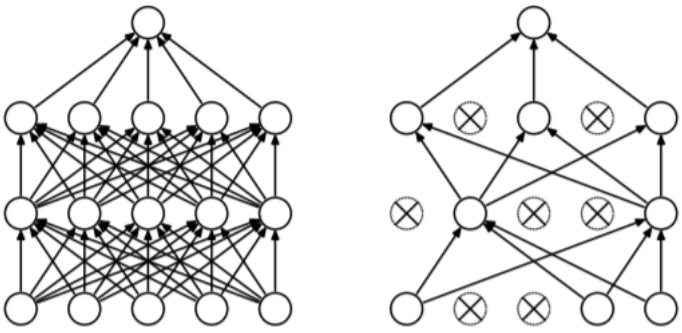
The **learning rate** defines how quickly a network updates its parameters. Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge. Usually a decaying Learning rate is preferred.

Activation functions are used to introduce nonlinearity to models, which allows deep learning models to learn nonlinear prediction boundaries. Generally, the rectified linear activation (or ***ReLU***) function is the most popular. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored. ***Sigmoid*** is used in the output layer while making binary predictions. ***Softmax*** is used in the output layer while making multi-class predictions.





Dropout is a regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power. Basically, during back-propagation, the model drops/eliminates randomly a certain percentage of connections - the percentage being equal to the dropout rate. Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.



(a) Standard Neural Net (b) After applying dropout.

Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Model architecture¶

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 5)                 155
_________________________________________________________________
dropout_3 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 18
_________________________________________________________________
dropout_4 (Dropout)          (None, 3)                 0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
```

Optimizer and Error Function

The standard gradient descent algorithm updates the parameters ΘΘ of the objective J(Θ)J(Θ) as,

$$\Theta=\Theta-\alpha\nabla_{\Theta}E[J(\Theta)]\Theta=\Theta-\alpha\nabla\Theta E[J(\Theta)]$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent (SGD)simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by,

$$\Theta=\Theta-\alpha\nabla_{\Theta}J(\Theta;x_{(i)},y_{(i)})]\Theta=\Theta-\alpha\nabla\Theta J(\Theta;x(i),y(i))]$$

with a pair $(x_{(i)},y_{(i)})(x(i),y(i))$ from the training set.

Generally, each parameter update in SGD is computed w.r.t a few training examples or a **minibatch** as opposed to a single example. The reason for this is twofold: first this **reduces the variance in the parameter update** and can lead to more stable convergence, second this allows the computation to **take advantage of highly optimized matrix operations** that should be used in a well vectorized computation of the cost and gradient.

We use **binary cross-entropy error function** for this learning task. It is given by,

$$L(y,p)=-(y.\log(p)+(1-y).\log(1-p))$$

Epochs and Batch size

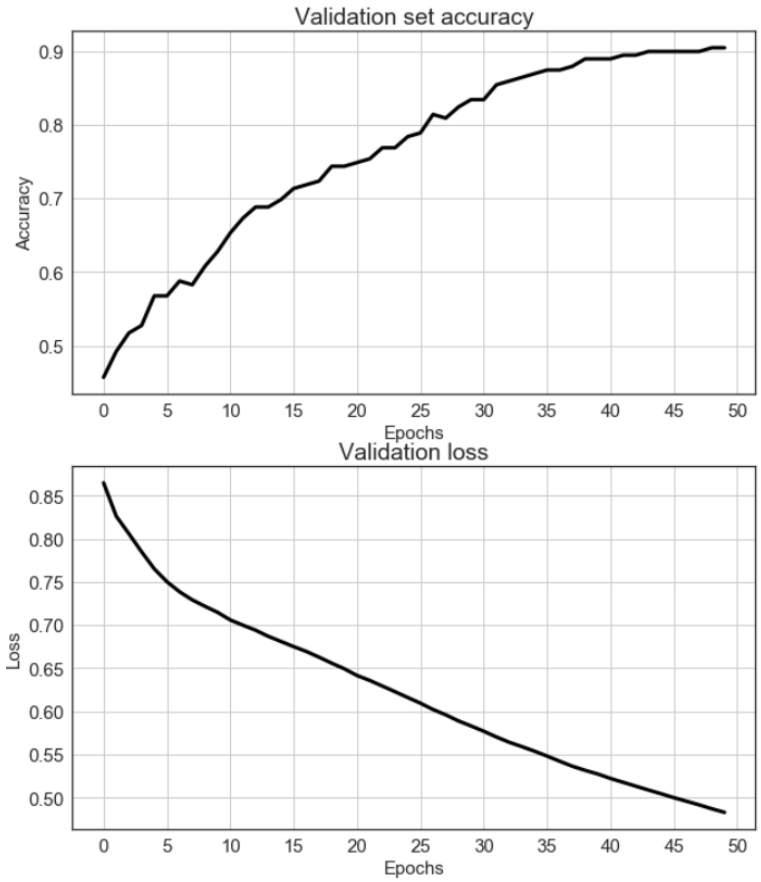
Number of epochs is the number of times the whole training data is shown to the network while training. Basically, it is one forward pass through the network to calculate the outputs (or output probabilities) and one back-propagation to adjust the weights based on the errors and their propagataed derivatives. Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting). We initially run the network for **50 epochs**.

Mini batch size is the number of sub samples given to the network after which parameter update happens. A common choice for batch size is 16 or 32, but since we are not dealing with many data here, we choose **batch size of 5**.

Running the model i.e. fitting training data

```
hist = model.fit(X_train, y_train, batch_size=batch_size, epochs=num_epoch
s,validation_data=(X_val, y_val), verbose=0, shuffle=False)
```

Plotting the validation set loss and accuracy



Checking the quality of the model

Confusion Matrix

A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making. We use `confusion_matrix` method from `sklearn.metrics` class of the Scikit-Learn package in this paper. We first use the `hist.model.predict`function to predict the breast cancer types (classes 0 or 1) by passing along the test data set `X_test`. This output predictions are real valued probability numbers as the output of the sigmoid activation function of our output layer. Thereafter, we use `binarize` method from `sklearn.preprocessing` class of Scikit-Learn to convert those probability values to binary outputs - `y_pred`. Finally, we compare the Ground Truth i.e. `y_test` with `y_pred` to calculate *True Positive, False Positive, True Negative*, and *False Negative* and construct the Confusion Matrix.

• **True Positives (TP) -** These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g. if actual class value indicates that this passenger survived and predicted class tells you the same thing.

• **True Negatives (TN) -** These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g. if actual class says this passenger did not survive and predicted class tells you the same thing.

• **False Positives (FP)** – When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

• **False Negatives (FN)** – When actual class is yes but predicted class in no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

	Predicted: YES	**Predicted: NO**
Actual: YES	51	4
Actual: NO	1	115

Classification report with *F1 score*

Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. Precision can be thought of as a measure of a classifiers exactness.

Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity. Recall can be thought of as a measure of a classifiers completeness.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. Put another way, the F1 score conveys the balance between the precision and the recall.

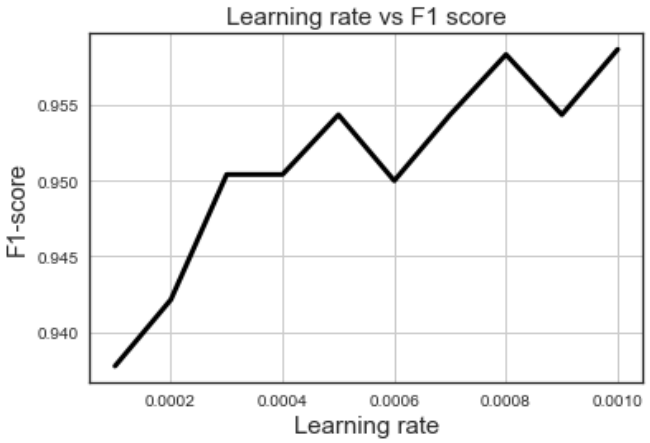
	precision	recall	f1-score	support
0	0.98	0.93	0.95	55
1	0.97	0.99	0.98	116
avg / total	0.97	0.97	0.97	171

Impact of the network hyperparameters on the classification quality

We further studied the impact of Hyperparameters of the neural network model on the F1 score. Learning rate and number of neurons (simultaneously in both layers) were varied over a wide range and the classification quality was measured. As expected, a slight increase of F1 score was observed with learning rate and a large change in F1 score was seen when number of neurons was increased from 1 to 15.

Code to compute and plot F1 score as a function of learning rate¶

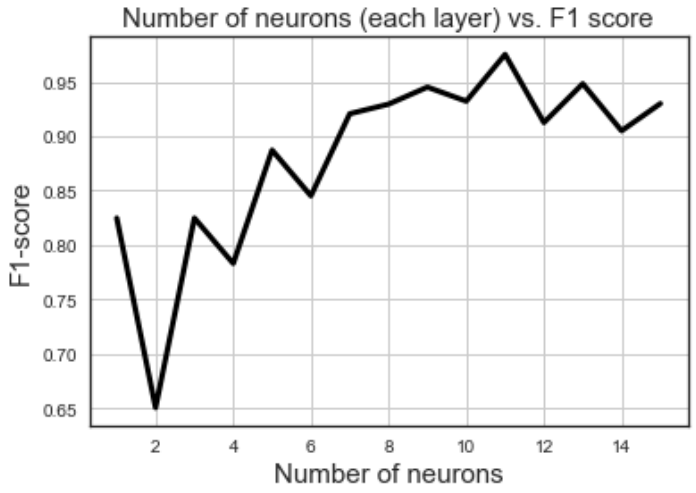
```
from sklearn.metrics import f1_score
lr_lst= [1e-4*i for i in range(1,11)]
f1_score_lst=[]
num_epochs=10
for lr in lr_lst:
    optimizer=keras.optimizers.sgd(lr=lr)
    model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=
['accuracy'])
    hist = model.fit(X_train, y_train, batch_size=batch_size, epochs=num_e
pochs,validation_data=(X_val, y_val),
              verbose=0, shuffle=False)
    predictions = hist.model.predict(X_test)
    predictions=binarize(predictions,0.5)
    f1score=f1_score(y_test,predictions)
    f1_score_lst.append(f1score)

plt.plot(lr_lst,f1_score_lst,color='k',lw=3)
plt.title("Learning rate vs F1 score",fontsize=15)
plt.grid(True)
plt.xlabel("Learning rate",fontsize=15)
plt.ylabel("F1-score",fontsize=15)
plt.show()
```



Code to compute and plot F1 score as a function of number of neurons in the hidden layers

```
neuron_lst= [i for i in range(1,16)]
f1_score_lst=[]
```

```
dropout_prob=0.25
learning_rate=1e-3
activation_func='relu'
num_epochs=20
batch_size=4

for neuron in neuron_lst:
    model = Sequential()
    model.add(Dense(neuron,input_shape=(input_dim,),activation=activation_
func))
    model.add(Dropout(dropout_prob))
    model.add(Dense(neuron,activation=activation_func))
    model.add(Dropout(dropout_prob))
    model.add(Dense(1,activation='sigmoid'))

    optimizer=keras.optimizers.sgd(lr=learning_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=
['accuracy'])

    hist = model.fit(X_train, y_train, batch_size=batch_size, epochs=num_e
pochs,
                     validation_data=(X_val, y_val),verbose=0, shuffle=Fal
se)

    predictions = hist.model.predict(X_test)
    predictions=binarize(predictions,0.5)
    f1score=f1_score(y_test,predictions)
    f1_score_lst.append(f1score)

plt.plot(neuron_lst,f1_score_lst,color='k',lw=3)
plt.title("Number of neurons (each layer) vs. F1 score",fontsize=15)
plt.grid(True)
plt.xlabel("Number of neurons",fontsize=15)
plt.ylabel("F1-score",fontsize=15)
#plt.xticks([i for i in range(0,num_epochs+1,5)],fontsize=15)
plt.show()
```



Summary and Conclusion

In this article, we discussed the basics of a densely-connected neural network and applied the model to the task of classification of breast cancer type (benign or malignant) based on ten geometric features (and their summary statistics) extracted from the images of cellular nuclei. In general, the model shows good accuracy, precision/recall and F1 score. We further studied the impact of some hyperparameters such as learning rate and number of neurons on the classification quality (F1 score). F1 score exhibited a positive trend with these hyperparameters.

Although conventional neural networks (CNN) have shown near-human level (or better) performance in recent years for direct image classification tasks, and they have been applied quite successfully for medical diagnostic applications (e.g. skin cancer detection), classification/diagnosis based on simple geometric features, extracted from cellular images, remain an interesting area of study. A simple 2-layer neural network could classify breast cancer type with high degree of accuracy based on these features and their summary statistics. This approach does not need nearly as much computing power as demanded by large and complex CNNs. Therefore, it can be asserted that a machine learning pipeline consisting of a front-end geometric feature extractor followed by a small neural network remains a viable alternative to much more complex CNN models for medical diagnostic applications.

References

1. Breast Cancer Wisconsin (Diagnostic) Data Set (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29)
2. O.L. Mangasarian, W.N. Street, and W.H. Wolberg. "Breast cancer diagnosis and prognosis via linear programming", *Operations Research*, 43(4), pages 570-577, 1995.
3. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, "Deep Learning Book", 2016
4. Pranoy Radhakrishnan, "What are Hyperparameters? and How to tune the Hyperparameters in a Deep Neural Network?", *Towards Data Science*, Web link: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a
5. Sagar Sharma, "Activation Functions: Neural Networks", *Towards Data Science*, Web link: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6