Alzheimer's Predictions

Alzheimer's is still uncureable and many people suffer because of its effects on themselves and their loved ones. In this analysis I will be utilizing a data set from Kaggle in order to predict Alzheimer's based on a multitude of factors. Predicting Alzheimer's allows people to prepare themselves for the worst case scenario through screening and such for early detection.

Imports

```
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
```

Reading in our dataset

Note: Since the dataset provided did not include a GitHub, I downloaded the .csv file and will use the following code to upload the file by selecting it from my local machine.

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  alzheimers_data = pd.read_csv(fn)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving alzheimers_prediction_dataset.csv to alzheimers_prediction_dataset.csv
```

```
# Preview the data
alzheimers_data.head()
```

	Country	Age	Gender	Education Level	BMI	Physical Activity Level	Smoking Status	Alcohol Consumption	Diabetes	Hypertension	...	Dietary Habits	Air Pollution Exposure	Employment Status
0	Spain	90	Male	1	33.0	Medium	Never	Occasionally	No	No	...	Healthy	High	Retired
1	Argentina	72	Male	7	29.9	Medium	Former	Never	No	No	...	Healthy	Medium	Unemployed
2	South Africa	86	Female	19	22.9	High	Current	Occasionally	No	Yes	...	Average	Medium	Employed
3	China	53	Male	17	31.2	Low	Never	Regularly	Yes	No	...	Healthy	Medium	Retired

Creating the training and test data

```
alzheimers_data_encoded = pd.get_dummies(alzheimers_data, dtype = int) # Because our data has many qualitative variables, they need to be or


# Since we only need either "Alzheimer's Diagnosis_No" or "Alzheimer's Diagnosis_Yes"... I will be dropping "Alzheimer's Diagnosis_No" from

alzheimers_data_encoded = alzheimers_data_encoded.drop("Alzheimer’s Diagnosis_No", axis = 1)


# Now I will separate the explanatory variables with what we actually want to predict i.e. the alzheimer's diagnosis

alzheimer_expvar = alzheimers_data_encoded.drop("Alzheimer’s Diagnosis_Yes", axis = 1)
diagnosis = alzheimers_data_encoded['Alzheimer’s Diagnosis_Yes']
```

```
alzheimer_train_expvar, alzheimer_test_expvar, diagnosis_train, diagnosis_test = train_test_split(alzheimer_expvar, diagnosis, test_size=.2,
alzheimer_train_expvar
```

	Age	Education Level	BMI	Cognitive Test Score	Country_Argentina	Country_Australia	Country_Brazil	Country_Canada	Country_China	Country_Fran
2968	50	4	19.6	87	0	0	0	0	0	
70623	64	19	27.1	41	0	0	0	1	0	
37340	84	1	25.7	33	0	0	0	0	0	
55130	87	16	20.3	52	0	0	0	0	0	
17358	77	16	33.0	91	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
17904	64	6	30.8	41	0	0	0	0	0	
37597	76	6	27.7	51	0	0	0	0	0	
10201	82	0	25.6	72	0	0	0	0	0	
9372	69	12	27.4	76	0	1	0	0	0	
50496	70	14	33.0	36	0	1	0	0	0	

59426 rows × 74 columns

Making the Neural Network

To start, I will create a basic neural network that has one layer with one hidden unit. Then I will be slowly improving the model by:

1. Adding hidden layers
2. Varying the number of hidden units
3. Changing the optimizer
4. Changing the learning rate
5. Changing the number of epochs

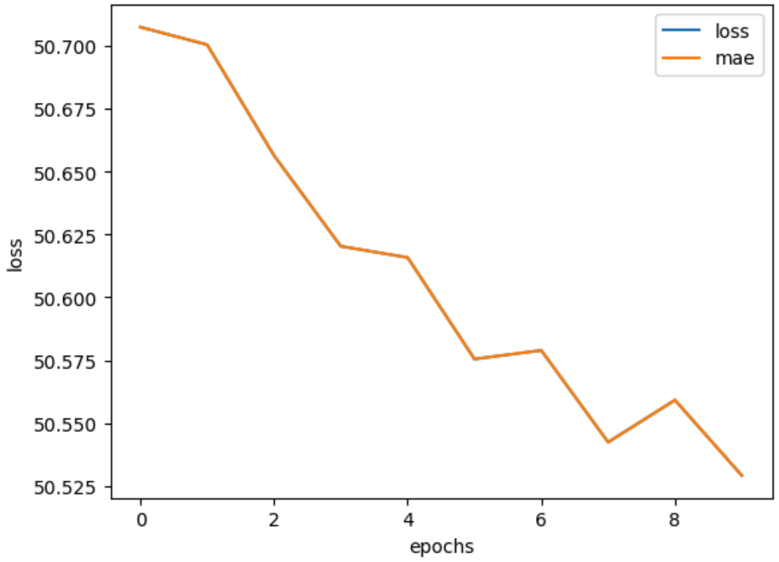
```
# Set seed for reproducibility
tf.random.set_seed(10)

# 1. Creating the model
alzheimer_model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
alzheimer_model.compile(loss = tf.keras.losses.mae,
                        optimizer = tf.keras.optimizers.SGD(),
                        metrics = ["mae"])

# 3. Fit the model
history = alzheimer_model.fit(alzheimer_train_expvar, diagnosis_train, epochs=10, verbose=0) # each step takes a while to run so only 10 epc
```

```
# Evaluating with test data
alzheimer_model.evaluate(alzheimer_test_expvar, diagnosis_test)
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 54.1333 - mae: 54.1333
[54.397281646728516, 54.397281646728516]
```

```
# Plotting the loss
pd.DataFrame(history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
Text(0.5, 0, 'epochs')
```



Improving the Model

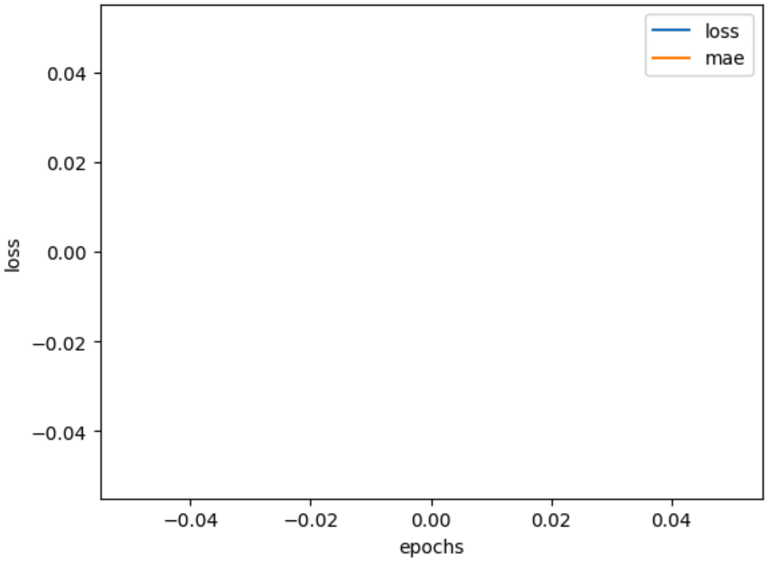
- `Model 1` will have 2 additional hidden layers with 50 nodes each
- `Model 2` will be the same as Model 1 but run 50 epochs
- `Model 3` will run 50 epochs with no additional layers
- `Model 4` will have 2 additional hidden layers with 100 nodes each, run for 100 epochs, and use the Adam optimizer
- `Model 5` will have 5 additional hidden layers with 50 nodes each, run for 100 epochs, and use the Adam optimizer

```
# Model 1

# Set random seed
tf.random.set_seed(10)

# 1. Creating the model
model_1 = tf.keras.Sequential([
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
model_1.compile(loss = tf.keras.losses.mae,
                optimizer = tf.keras.optimizers.SGD(),
                metrics = ["mae"])

# 3. Fitting the model
model_1_history = model_1.fit(alzheimer_train_expvar, diagnosis_train, epochs = 10, verbose = 0)


# Evaluating with test data
model_1.evaluate(alzheimer_test_expvar, diagnosis_test)
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: nan - mae: nan
[nan, nan]
```

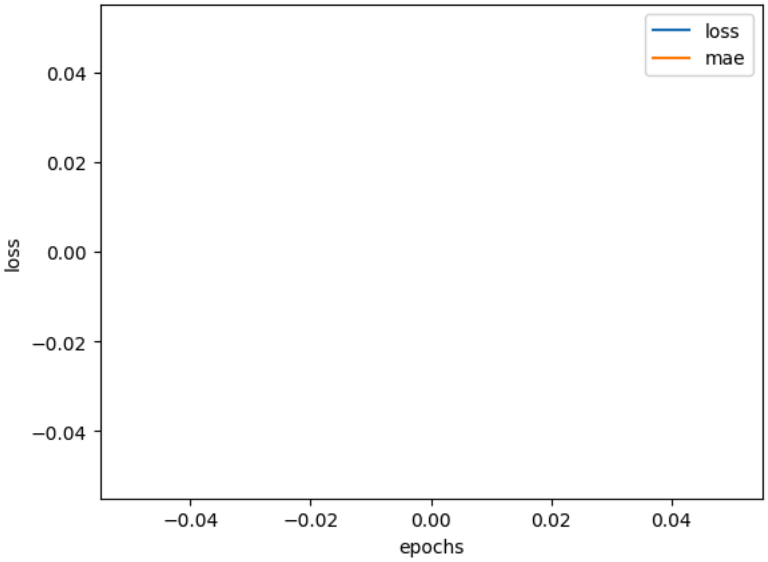
```
# Plotting the loss
pd.DataFrame(model_1_history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
Text(0.5, 0, 'epochs')
```



```
# Model 2

# Set random seed
tf.random.set_seed(10)

# 1. Creating the model
model_2  = tf.keras.Sequential([
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
model_2.compile(loss = tf.keras.losses.mae,
                optimizer = tf.keras.optimizers.SGD(),
                metrics = ["mae"])

# 3. Fitting the model
model_2_history = model_2.fit(alzheimer_train_expvar, diagnosis_train, epochs = 50, verbose = 0)
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 14.8008 - mae: 14.8008
Text(0.5, 0, 'epochs')
```



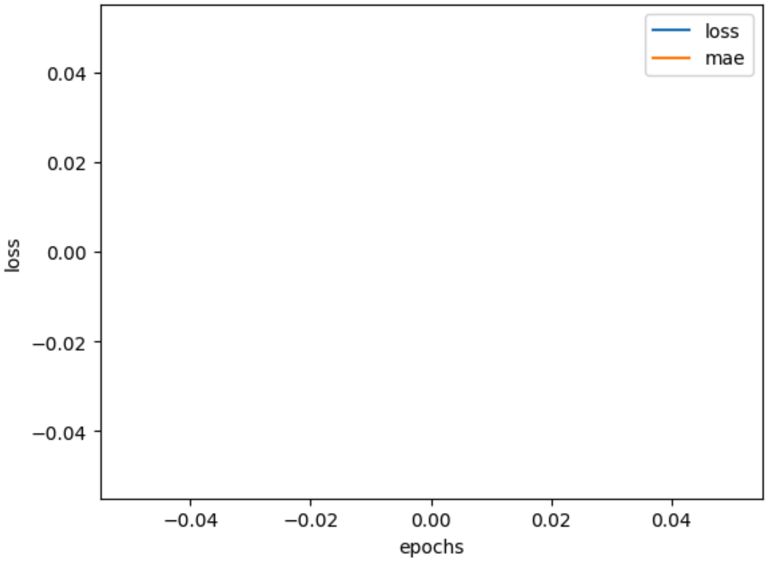
```
# Evaluating with test data
model_2.evaluate(alzheimer_test_expvar, diagnosis_test)
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 14.8008 - mae: 14.8008
```

```
[14.833300590515137, 14.833300590515137]
```

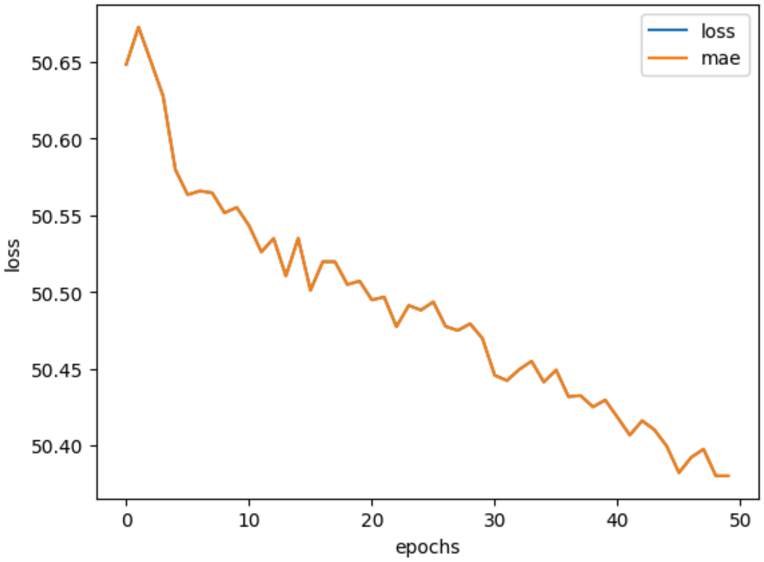
```
# Plotting the loss
pd.DataFrame(model_2_history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
Text(0.5, 0, 'epochs')
```



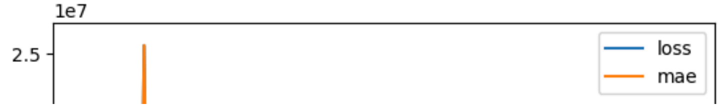
```
# Model 3

# Set random seed
tf.random.set_seed(10)

# 1. Creating the model
model_3 = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
model_3.compile(loss = tf.keras.losses.mae,
                optimizer = tf.keras.optimizers.SGD(),
                metrics = ["mae"])

# 3. Fitting the model
model_3_history = model_3.fit(alzheimer_train_expvar, diagnosis_train, epochs  = 50, verbose = 0)

# Evaluating with test data
model_3.evaluate(alzheimer_test_expvar, diagnosis_test)

# Plotting the loss
pd.DataFrame(model_3_history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 57.9513 - mae: 57.9513
Text(0.5, 0, 'epochs')
```



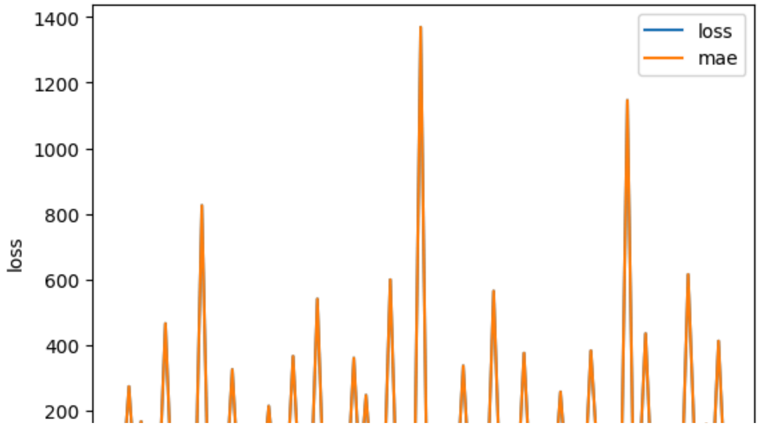
```
# Model 4

# Set random seed
tf.random.set_seed(10)

# 1. Creating the model
model_4 = tf.keras.Sequential([
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(50),
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
model_4.compile(loss = tf.keras.losses.mae,
                optimizer = tf.keras.optimizers.Adam(learning_rate=.1), # Starting with a learning rate of .1, might be changed later if thi
                metrics = ["mae"])

# 3. Fitting the model
model_4_history = model_4.fit(alzheimer_train_expvar, diagnosis_train, epochs = 100, verbose = 0)

# Evaluating with test data
model_4.evaluate(alzheimer_test_expvar, diagnosis_test)

# Plotting the loss
pd.DataFrame(model_4_history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 1363.9346 - mae: 1363.9346
Text(0.5, 0, 'epochs')
```



```
# Model 5

# Set random seed
tf.random.set_seed(10)

# 1. Creating the model
model_5 = tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(1)
])

# 2. Compiling the model
model_5.compile(loss = tf.keras.losses.mae,
                optimizer = tf.keras.optimizers.Adam(learning_rate=.1), # Starting with a learning rate of .1, might be changed later if thi
                metrics = ["mae"])

# 3. Fitting the model
model_5_history = model_5.fit(alzheimer_train_expvar, diagnosis_train, epochs = 100, verbose = 0)

# Evaluating with test data
model_5.evaluate(alzheimer_test_expvar, diagnosis_test)

# Plotting the loss
pd.DataFrame(model_5_history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")
```

```
465/465 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.7750 - mae: 0.7750
Text(0.5, 0, 'epochs')
```


loss
loss
mae