Investigation of Various Neural Network Designs on the Titanic Dataset

By: Olly Love, Nathan Singer, David Kelly

The Problem

- Investigating multiple neural network designs, a comparative study of different machine learning approaches
- Including a new architecture, the tabular autoencoder, as part of the investigation.
- Comparing performance of each model on the Kaggle Titanic dataset, predicting the survival of passengers: https://www.kaggle.com/datasets/yasserh/titanic-dataset

Models

- Linear Classifier
- Logistic Regression
- MLP (1D, 2D, and 4D with a residual network)
- Tabular Autoencoder
- Trained, evaluated, and saved with a training loop

Linear Classifier

```
class LinearClassifier(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.num_features = num_features
        self.linear = nn.Linear(num_features, 2)

    def forward(self, x):
        return self.linear(x)
```

Linear Classifier - Training

```
n_samples, n_features = train_dataset.tensors[0].shape
n_epochs = 100
```

```
def lin_train(model: nn.Module, loss: nn.Module, optimizer: Optimizer, dataloader: DataLoader, epochs: int):
    for epoch in range(epochs):
        for (x,target) in dataloader:
            pred = model(x)
            l = loss(pred, target)

            optimizer.zero_grad()
            l.backward()
            optimizer.step()

        if epoch % 10 == 0:
            [w,b] = model.parameters()
            print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')
```

Linear Classifier - Evaluation

```
# Training linear
lin = LinearClassifier(n_features).to(device)
loss = nn.CrossEntropyLoss().to(device)

# May want to adjust learning rate
optimizer = SGD(lin.parameters(), lr=0.001)
# May want less epochs
lin_train(lin, loss, optimizer, train_dataloader, n_epochs)
torch.save(lin, "models/lin_model.mdl")
```

```
epoch 220: w = 0.258, loss = 0.03939191, total_loss = 99.5446
epoch 230: w = 0.263, loss = 0.03853292, total_loss = 99.4039
epoch 240: w = 0.268, loss = 0.03774219, total_loss = 99.2733
epoch 250: w = 0.272, loss = 0.03701298, total_loss = 99.1516
...
epoch 970: w = 0.457, loss = 0.02777275, total_loss = 96.4977
epoch 980: w = 0.458, loss = 0.02778550, total_loss = 96.4869
epoch 990: w = 0.459, loss = 0.02779860, total_loss = 96.4764
epoch 1000: w = 0.461, loss = 0.02781193, total_loss = 96.4661
```

Logistic Regression

```
class LogisticRegression(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.num_features = num_features
        # Makes logit
        self.linear = nn.Linear(num_features, 1)

    def forward(self, x):
        return torch.sigmoid(self.linear(x))
```

Logistic Regression - Training

```
def log_train(model: nn.Module, loss: nn.Module, optimizer: Optimizer, dataloader: DataLoader, epochs: int):
    for epoch in range(epochs):
        for x, target in dataloader:
            target = target.float().unsqueeze(1)
            pred = model(x)
            l = loss(pred, target)

            optimizer.zero_grad()
            l.backward()
            optimizer.step()


        if epoch % 10 == 0:
            [w,b] = model.parameters()
            print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')
```

Logistic Regression - Evaluation

```
# Training Logistic
log = LogisticRegression(n_features).to(device)
loss = nn.BCELoss().to(device)

# May want to adjust learning rate
optimizer = SGD(log.parameters(), lr=0.001)
log_train(log,loss,optimizer,train_dataloader,n_epochs)
torch.save(log, "models/log_model.mdl")
```

```
epoch 10: w = -0.090, loss = 0.21280563, total_loss = 116.4880
epoch 20: w = -0.181, loss = 0.19022161, total_loss = 110.5007
epoch 30: w = -0.220, loss = 0.17497328, total_loss = 106.8198
epoch 40: w = -0.242, loss = 0.16306059, total_loss = 103.9601
epoch 50: w = -0.258, loss = 0.15303203, total_loss = 101.6191
epoch 60: w = -0.271, loss = 0.14434722, total_loss = 99.6729
epoch 70: w = -0.283, loss = 0.13674350, total_loss = 98.0404
epoch 80: w = -0.295, loss = 0.13004620, total_loss = 96.6603
epoch 90: w = -0.306, loss = 0.12411778, total_loss = 95.4854
epoch 100: w = -0.317, loss = 0.11884392, total_loss = 94.4788
```

One-D MLP

```
class MLPClassifier(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.num_features = num_features
        self.linear1 = nn.Linear(num_features, 100)
        self.act1 = nn.ReLU()
        self.output = nn.Linear(100, 2)

    def forward(self, x):
        x = self.linear1(x)
        x = self.act1(x)
        x = self.output(x)
        return x
```

MLP Training (For All MLP Models)

```python
def mlp_train(model: nn.Module, loss: nn.Module, optimizer: Optimizer, dataloader: DataLoader, epochs: int):
    for epoch in range(epochs):
        for x, target in dataloader:
            pred = model(x)
            l = loss(pred, target)

            optimizer.zero_grad()
            l.backward()
            optimizer.step()

        if epoch % 10 == 0:
            w = model.linear1.weight
            print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')
```

MLP One-D Evaluation

```
# Training MLP

loss = nn.CrossEntropyLoss().to(device)

# 1 layer
mlp = MLPClassifier(n_features).to(device)
optimizer = Adam(mlp.parameters(), lr=0.001)
mlp_train(mlp,loss,optimizer,train_dataloader,n_epochs)

torch.save(mlp, "models/mlp_onelayer_model.mdl")
```

```
epoch 10: w = -0.127, loss = 0.11859322, total_loss = 82.2418
epoch 20: w = -0.147, loss = 0.08677053, total_loss = 78.1386
epoch 30: w = -0.158, loss = 0.08098578, total_loss = 75.4810
epoch 40: w = -0.168, loss = 0.07231505, total_loss = 77.6081
epoch 50: w = -0.178, loss = 0.07026675, total_loss = 74.0961
epoch 60: w = -0.191, loss = 0.06030358, total_loss = 71.9251
epoch 70: w = -0.199, loss = 0.05834802, total_loss = 68.8279
epoch 80: w = -0.201, loss = 0.05212197, total_loss = 68.7105
epoch 90: w = -0.200, loss = 0.04799564, total_loss = 65.6423
epoch 100: w = -0.202, loss = 0.04922508, total_loss = 64.8338
```

Two-D MLP

```
class MLP2DClassifier(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.num_features = num_features
        self.linear1 = nn.Linear(num_features, 100)
        self.act1 = nn.ReLU()
        self.linear2 = nn.Linear(100,50)
        self.output = nn.Linear(50,2)
    def forward(self, x):
        x = self.linear1(x)
        x = self.act1(x)
        x = self.linear2(x)
        x = self.output(x)
        return x
```

Two-D MLP Evaluation

```
# 2 layers
mlp2 = MLP2DClassifier(n_features).to(device)
optimizer = Adam(mlp2.parameters(), lr=0.001)
mlp_train(mlp2,loss,optimizer,train_dataloader,n_epochs)

torch.save(mlp2, "models/mlp_twolayer_model.mdl")
```

```
epoch 10: w = 0.047, loss = 0.09897653, total_loss = 80.8777
epoch 20: w = 0.058, loss = 0.07110681, total_loss = 77.1249
epoch 30: w = 0.077, loss = 0.05992853, total_loss = 75.2195
epoch 40: w = 0.101, loss = 0.05771663, total_loss = 73.3120
epoch 50: w = 0.111, loss = 0.05571553, total_loss = 71.0313
epoch 60: w = 0.125, loss = 0.05121632, total_loss = 69.1391
epoch 70: w = 0.122, loss = 0.04961312, total_loss = 68.1810
epoch 80: w = 0.131, loss = 0.04892419, total_loss = 65.8632
epoch 90: w = 0.138, loss = 0.04627061, total_loss = 64.6637
epoch 100: w = 0.145, loss = 0.04573051, total_loss = 63.1262
```

Four-D MLP With Residual Network

```python
class MLP4DResidualClassifier(nn.Module):
    def __init__(self, num_features):
        super().__init__()

        self.act = nn.ReLU()
        self.num_features = num_features

        # Main layers
        self.linear1 = nn.Linear(num_features, 128)
        self.linear2 = nn.Linear(128, 64)
        self.linear3 = nn.Linear(64, 32)
        self.linear4 = nn.Linear(32, 16)

        # Skip connections
        self.skip1 = nn.Linear(num_features, 128)
        self.skip2 = nn.Linear(128, 64)
        self.skip3 = nn.Linear(64, 32)
        self.skip4 = nn.Linear(32, 16)

        self.output = nn.Linear(16,2)
```

```python
def forward(self, x):
    s1 = self.skip1(x)
    x = self.act(self.linear1(x) + s1)

    s2 = self.skip2(x)
    x = self.act(self.linear2(x) + s2)

    s3 = self.skip3(x)
    x = self.act(self.linear3(x) + s3)

    s4 = self.skip4(x)
    x = self.act(self.linear4(x) + s4)

    return self.output(x)
```

Four-D MLP Evaluation

```
# 4 layers residual
mlp4 = MLP4DResidualClassifier(n_features).to(device)
optimizer = Adam(mlp4.parameters(), lr=0.001)
mlp_train(mlp4,loss,optimizer,train_dataloader,n_epochs)

torch.save(mlp4, "models/mlp_fourlayer_model.mdl")
```

```
epoch 10: w = 0.181, loss = 0.09603711, total_loss = 81.6603
epoch 20: w = 0.181, loss = 0.08851510, total_loss = 75.3010
epoch 30: w = 0.181, loss = 0.08842213, total_loss = 71.9302
epoch 40: w = 0.181, loss = 0.10740285, total_loss = 68.4317
epoch 50: w = 0.181, loss = 0.07169863, total_loss = 67.1694
epoch 60: w = 0.181, loss = 0.04554076, total_loss = 64.4520
epoch 70: w = 0.181, loss = 0.03830257, total_loss = 63.8072
epoch 80: w = 0.181, loss = 0.04722589, total_loss = 62.9862
epoch 90: w = 0.181, loss = 0.03841064, total_loss = 67.9770
epoch 100: w = 0.181, loss = 0.03018275, total_loss = 60.5111
```

Tabular Autoencoder

```python
class TabularAutoencoder(nn.Module):
    def __init__(self, input_dim, latent_dim=8):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 16),
            nn.ReLU(),
            nn.Linear(16, latent_dim),
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 16),
            nn.ReLU(),
            nn.Linear(16, input_dim)
        )

    def forward(self, x):
        z = self.encoder(x)
        x_hat = self.decoder(z)
        return x_hat
```

Tabular Autoencoder Training

```
def ta_train(model: nn.Module, loss: nn.Module, optimizer: Optimizer, dataloader: DataLoader, epochs: int):
    for epoch in range(epochs):
        for x, _ in dataloader:
            x_prime = x[0]

            pred = model(x)
            l = loss(pred, x_prime)

            optimizer.zero_grad()
            l.backward()
            optimizer.step()

        if epoch % 10 == 0:
            print(f'epoch {epoch + 1}: loss = {l:.8f}')
```

Tabular Autoencoder Evaluation

```
ta = TabularAutoencoder(input_dim = n_features, latent_dim = 8).to(device)
optimizer = Adam(ta.parameters(), lr=1e-3)
loss = nn.MSELoss().to(device)
ta_train(ta, loss, optimizer, train_dataloader, n_epochs)

torch.save(mlp2, "models/tae_model.mdl")
```

```
epoch 10: loss = 30.11790276, total_loss = 19016.2615
epoch 20: loss = 29.80267715, total_loss = 18565.0683
epoch 30: loss = 28.63719940, total_loss = 18291.9598
epoch 40: loss = 25.97786140, total_loss = 18054.6605
epoch 50: loss = 24.59546280, total_loss = 17911.9967
epoch 60: loss = 23.84953690, total_loss = 17823.4341
epoch 70: loss = 23.19057846, total_loss = 17732.9634
epoch 80: loss = 22.73792458, total_loss = 17660.4718
epoch 90: loss = 22.81102562, total_loss = 17601.6074
epoch 100: loss = 22.60476494, total_loss = 17579.1379
```

Comparing Performance Through Loss

Final Total Loss	Linear Classifier	Logistic Regression	MLP-1D	MLP-2D	MLP-4D
100 Epochs	104.1929	94.4788	64.8338	63.1262	60.5111
500 Epochs	100.3295	89.8364	56.0969	53.5090	46.3157
1000 Epochs	96.4661	85.1941	43.3600	43.8918	32.1203

Deployment - Titanic Guesser

- Game implementing our MLP 4D model
- User reads the story of a Titanic passenger and must predict their survival
- The ML model predicts the correct answer, comparing with the users guess to check their score
- Bonus: User can create a passenger, this data is put through the model to predict there survival
- Tkinter GUI was used for the development of the game

Deployment - Titanic Guesser - Model

```python
def ml_evaluate(self):
    #model: MLPClassifier = torch.load("models/mlp_onelayer_model.mdl", weights_only=False)
    model: MLP4DResidualClassifier = torch.load("models/mlp_fourlayer_model.mdl", weights_only = False)
    model.eval()
    x = torch.tensor([self.current_features], dtype=torch.float32)
    with torch.no_grad():
        x = x.to("cpu")
        logits = model(x)
        preds = torch.argmax(logits, dim=1)
        # Gives value within preds tensor
        # print(preds.item())
        return preds.item()

def change_page(self):
    self.clear_frame(self.page_container)
    self.current_page_index += 1
    self.pages[self.current_page_index]()
```

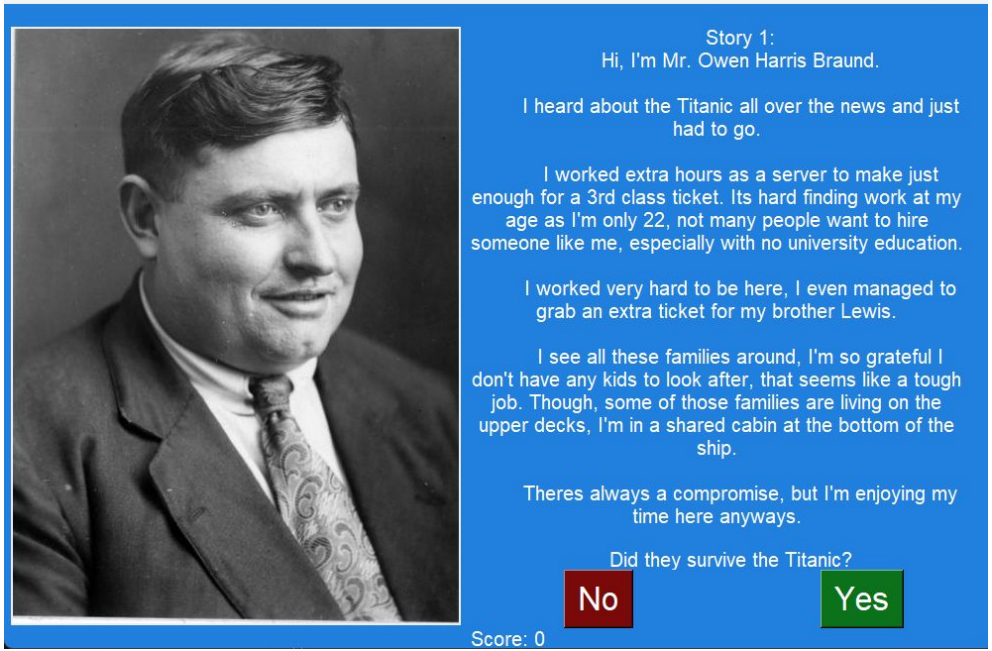
Deployment - Titanic Guesser - Model

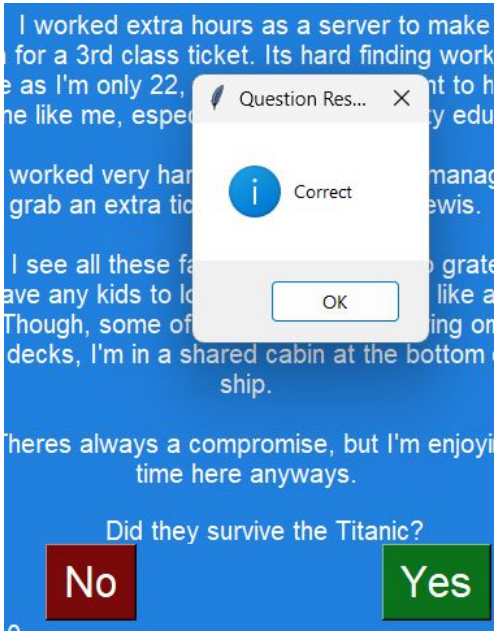
```
# When user clicks yes or no button, check with correct
# answer from model
def score(self):
    correct = self.ml_evaluate()
    if self.no_button == 1 and correct == 0:
        self.total_score += 1
        self.no_button = 0
        showinfo("Question Response", "Correct")
    elif self.no_button == 1 and correct == 1:
        self.no_button = 0
        showinfo("Question Response", "Incorrect")
    elif self.yes_button == 1 and correct == 1:
        self.total_score += 1
        self.yes_button = 0
        showinfo("Question Response", "Correct")
    elif self.yes_button == 1 and correct == 0:
        self.yes_button = 0
        showinfo("Question Response", "Incorrect")
```

Deployment - Titanic Guesser - Ex. Page Layout

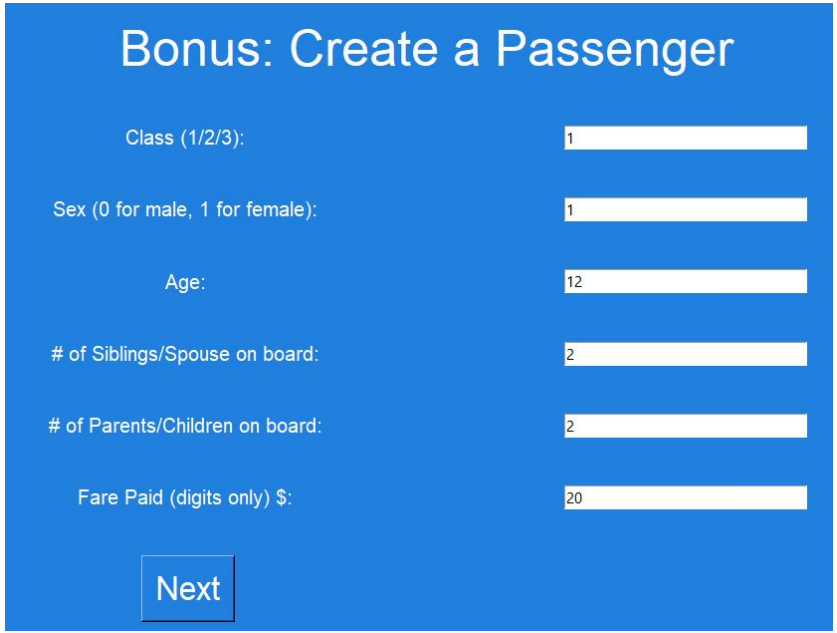
```
# Survived - PassengerId = 86
self.current_features = [3,1,33,3,0,15.85,0,0,0,0,0,0,0,0,1,0,0,1]
story = """
Story 5:
Well hey there, my names Mrs. Karl Alfred.

My husband thought it would be nice to surprise the family with Titanic tickets, so here we are.

Our quarters are cramped, but its a fabulous ship with tons to do.
If only the kid didn't keep running away, but, at least were getting good exercise.
"""

self.portrait = ImageTk.PhotoImage(Image.open("imgs/titanicmom.jpg").resize((450, 600), Image.Resampling.LANCZOS))
image_label = tk.Label(self.page_container, image=self.portrait)
image_label.grid(row=0,column=0,rowspan=4,sticky="w")
story_txt = tk.Label(self.page_container,text=story, font=("Helvetica", 12),wraplength=500,fg="white",bg="#217fdd",anchor
story_txt.grid(row=0, column=1,columnspan=2,sticky="w")
question_txt = tk.Label(self.page_container,text="Did they survive the Titanic?", font=("Helvetica", 12),fg="white",bg="#
question_txt.grid(row=1, column=1,columnspan=2,sticky="n")

no_btn = tk.Button(self.page_container, text="No", font=("Helvetica", 20), fg="white", bg="#7A0A0A",command=self.no_btn_c
no_btn.grid(row=2,column=1,sticky="s")
yes_btn = tk.Button(self.page_container, text="Yes", font=("Helvetica", 20), fg="white", bg="#0b711a",command=self.yes_bt
yes_btn.grid(row=2,column=2,sticky="s")

score_txt = tk.Label(self.page_container,text="Score: " + str(self.total_score), font=("Helvetica", 12),fg="white",bg="#2
score_txt.grid(row=3, column=1,sticky="w")
```

Deployment - Titanic Guesser - Gameplay





Deployment - Titanic Guesser - Gameplay



Conclusion

- MLP 4D with a residual network performed the best on the Titanic dataset
- As a result, we chose to deploy this into the Titanic Guesser to evaluate a users choice, model accuracy is very important
- TA performed terribly, we concluded that its not the correct model for our dataset

The End!