Deep Learning Major Report

Mukul Shingwani (B20AI023)

Paper link: [DataMUX: Data Multiplexing for Neural Networks]

- In this paper, the authors introduced `data multiplexing (DataMUX)`, a technique that enables deep neural networks to `process multiple inputs simultaneously` using a single compact representation. DataMUX demonstrates that neural networks are capable of generating accurate predictions over *mixtures* of inputs, resulting in increased inference throughput with minimal extra memory requirements.
- The approach uses two key components –
    - a ***multiplexing layer*** that performs a fixed linear transformation to each input before combining them to create a 'mixed' representation of the same size as a single input, which is then processed by the base network, and
    - a ***demultiplexing layer*** that converts the base network's output back into independent representations before producing predictions for each input

Multiplexing for Transformers (T-MUX)

The `baseline mode`l used for comparison: (B1) a 12-layer vanilla Transformer with a hidden dimension size of 768.

- The models and baselines are evaluated on two types of text classification tasks:
    - `token-level classification`
    - `sentence-level classification`.
- For token-level classification, the CoNLL-2003 `Named Entity Recognition (NER`) task is used.
- For sentence-level classification, the models are evaluated on a subset of the `General Language Understanding Evaluation (GLUE) benchmark`, including the the natural language inference tasks `MNLI` and sentence similarity task `QQP`.

Results

NOTE: In the paper the values of N were {2,5,10,20,40) and evaluation of accuracy on tasks such as MNLI, QNLI, SST2, NER, QQP were done and two mixing variants. We, have used N = {2,5,10), tasks = {MNLI, QQP, NER} and 2 muxing variants {gaussian_hadamard, random_ortho} due to limited time and computational resources (with increasing N it was getting too time costly). These results were calculated in computers on ANVISA lab and not in Colab.

- **MNLI**

Model	**Baseline**	**T-MUX(N=2)**	**T-MUX(N=5)**	**T-MUX(N=10)**
Avg. Test Accuracy (**Ortho**)	59.8%	61.5%	58.8%	56.3%
Avg. Test Accuracy (**Hadamard**)	59.8%	63.4%	60.2%	59.7%

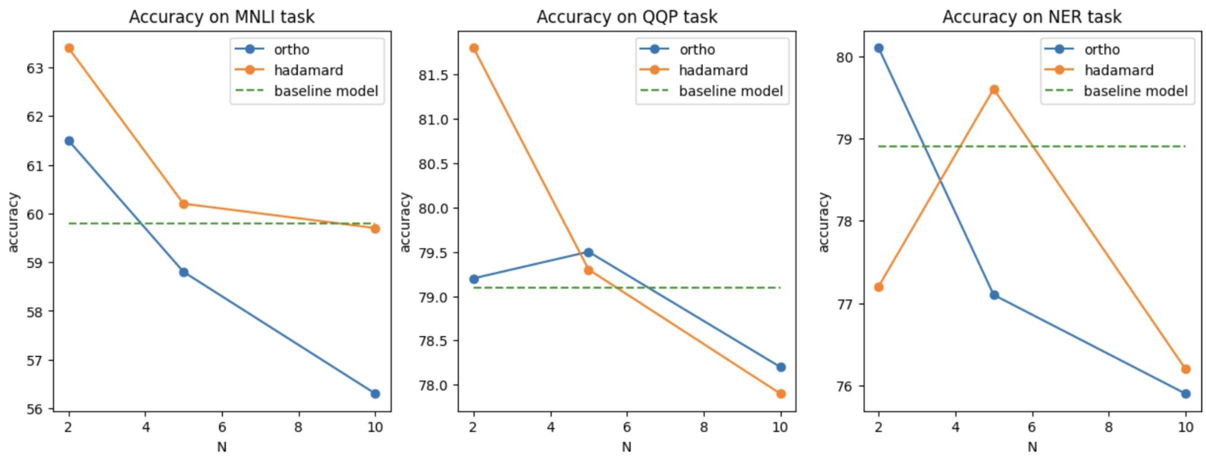
- **QQP**

Model	**Baseline**	**T-MUX(N=2)**	**T-MUX(N=5)**	**T-MUX(N=10)**
Avg. Test Accuracy (**Ortho**)	78.9%	79.2%	79.5%	78.2%

Avg. Test Accuracy (**Hadamard**)	78.9%	81.8%	79.3%	77.9%

- **NER**

Model	**Baseline**	**T-MUX(N=2)**	**T-MUX(N=5)**	**T-MUX(N=10)**
Avg. Test Accuracy (**Ortho**)	78.9%	80.1%	77.1%	75.9%
Avg. Test Accuracy (**Hadamard**)	78.9%	77.2%	79.6%	76.2%



- In **graph 1,** the gaussian_hadamard transformation works almost as good as the baseline model for this task with max drop being 1%
- In **graph 2**, Both the transformations work almost as good as the baseline model for this task a maximum of 1.75% drop in accuracy
- In **graph 3**, Both the transformations work almost as good as the baseline model for this task too, with about 3% drop in accuracy

Multiplexing for CNNs

The authors used MNIST dataset for this and following were the results

`We` have used `three types of projections` for result reproduction task and those are `Rotation`, `Gaussian`, and `non-linear`.

Now we'll see the result of each one them one by one through some plots and accuracy table (N specifies the number of inputs)

- **Using Rotation transforms**

Model	**Baseline**	**MUX(N=1)**	**MUX(N=2)**	**MUX(N=4)**	**MUX(N=8)**	**MUX(N=16)**
Avg. Test Accuracy	97.59%	97.83%	92.85%	58.95%	36.61%	23.21%

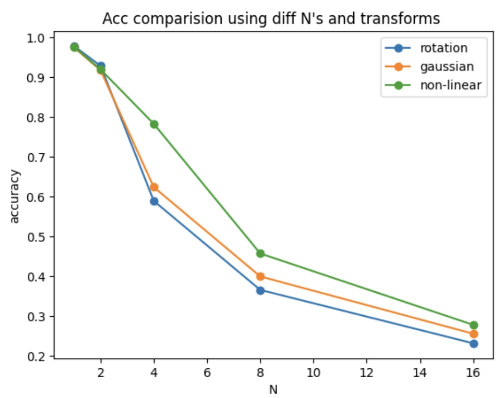
- **Using Gaussian transforms**

Model	**Baseline**	**MUX(N=1)**	**MUX(N=2)**	**MUX(N=4)**	**MUX(N=8)**	**MUX(N=16)**
Avg. Test Accuracy	97.59%	97.50%	91.74%	62.41%	39.99%	25.61%

- **Using Non-linear transforms**

Model	Baseline	MUX(N=1)	MUX(N=2)	MUX(N=4)	MUX(N=8)	MUX(N=16)
Avg. Test Accuracy	97.59%	97.67%	91.96%	78.3%	45.75%	27.84%

- So, it's quite evident that for `CNNs multiplexing doesn't perform wel`l, especially after N=2
- It is likely because the transformation destroys the property of spatial locality that CNNs rely on. The researchers therefore also explore N two-layer convolutional networks with a tanh activation as their multiplexing transformations `(CNN + Nonlinear)`. It is noted that even with this transformation, the dimensionality of the multiplexed representation still remains equal to the dimensionality of a single input.
- It is found that for $N \leq 4$, the performance is above 70%, which is significantly better than CNN+Rotation and CNN+Gaussian, but the performance drops rapidly for $N > 4$. Also, with increase in N, time also goes on increasing.
- Overall, multiplexing for CNNs seems to be more challenging than Transformers, as evidenced by the sharper performance drops with increasing N



Application 1 of DataMUX

Multi label classification was the application I undertook. The dataset contains 6 different labels(Computer Science, Physics, Mathematics, Statistics, Quantitative Biology, Quantitative Finance) to classify the research papers based on Abstract and Title.

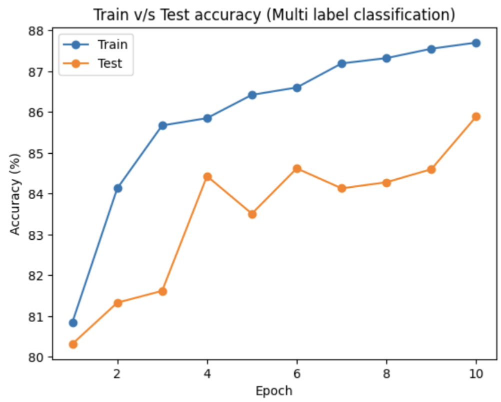
Results

For each of the labels in the dataset loss and accuracy was calculated for both training and testing loops. 1 sample result from epoch 10 is shown below along with plots of the combined accuracy for each train and test epoch.

Testing accuracy achieved was about **85%**, this was after taking average of all individual labels, so overall it did a good job.

```
Epoch Train 10: Loss CS: 0.4213 || Loss Phy: 0.3208 || Loss Math: 0.3654 || Loss Stat: 0.4281 || Loss QB: 0.1009 ||
Loss QF: 0.0523 || Loss Total: 1.6888 || Acc CS: 0.8030 || Acc Phy: 0.8737 || Acc Math: 0.8451 || Acc Stat: 0.7805 ||
Acc QB: 0.9715 || Acc QF: 0.9884 || Acc Total: 0.8770

Epoch Test 10: Loss CS: 0.4828 || Loss Phy: 0.4469 || Loss Math: 0.4488 || Loss Stat: 0.5265 || Loss QB: 0.0976 ||
Loss QF: 0.0617 || Loss Total: 2.0643 || Acc CS: 0.7675 || Acc Phy: 0.8014 || Acc Math: 0.7959 || Acc Stat: 0.7649 ||
Acc QB: 0.9762 || Acc QF: 0.9873 || Acc Total: 0.8489
```

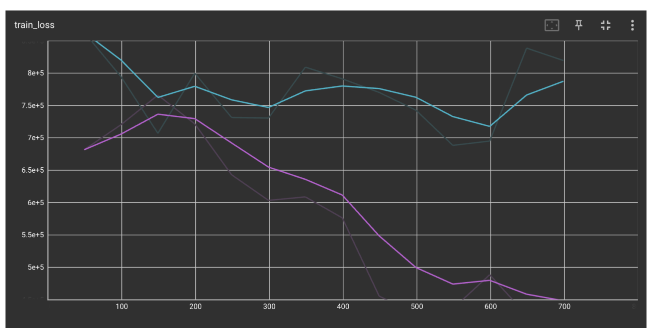


Application 2 : combining Image + Tabular data

- In this task I thought of predicting the **real estate prices** of houses through matching **image data** and **tabular information**
- compared this with just image only data, to get to know the importance of this combined approach

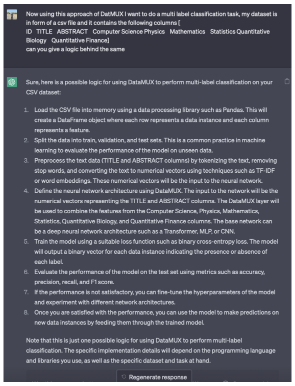
Results

- We trained the model for 10 epochs, with learning rate of 3e-6 and compared the results of `image+tabular data` with `image-only data`
- The error in prediction using `method-1(image+tabular data`) was 4,64,000, but, when only image data was taken then the error was 7,59,000 i.e using `method-2 (image only data)`
- although, they are quite large, but it's due to the fact we had limited data, but still our purpose is answered that when image+tabular data was considered the error in prediction was reduced significantly.
- Shown below is the graph of train loss in tensor board platform, here purple line is for method-1 and blue line for method-2



Using ChatGPT to solve these applications

I started of by giving chatGPT abstract of the paper and asked it to remember and understand it, and then asked it to write the code or give logic for my application but ChatGPT wasn't able to solve this, either the code it gave too many errors, or it ran out of context and gave only partial incomplete code. I tried to help it by also giving code used by the authors to see whether it could tweak it as per my requirement but it wasn't able to. But, when I asked for the logic behind the same it gave a rather good response



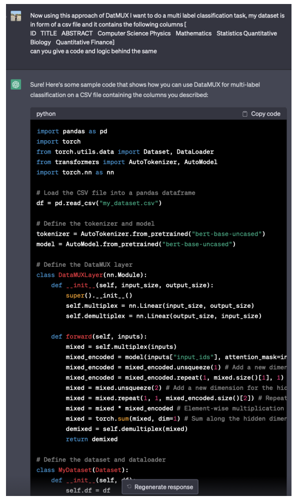
Now using this approach of DatMUX I want to do a multi label classification task, my dataset is in form of a csv file and it contains the following columns [
ID TITLE ABSTRACT Computer Science Physics Mathematics Statistics Quantitative Biology Quantitative Finance]
can you give a logic behind the same

Sure, here is a possible logic for using DataMUX to perform multi-label classification on your CSV dataset:

1. Load the CSV file into memory using a data processing library such as Pandas. This will create a DataFrame object where each row represents a data instance and each column represents a feature.
2. Split the data into train, validation, and test sets. This is a common practice in machine learning to evaluate the performance of the model on unseen data.
3. Preprocess the text data (TITLE and ABSTRACT columns) by tokenizing the text, removing stop words, and converting the text to numerical vectors using techniques such as TF-IDF or word embeddings. These numerical vectors will be the input to the neural network.
4. Define the neural network architecture using DataMUX. The input to the network will be the numerical vectors representing the TITLE and ABSTRACT columns. The DataMUX layer will be used to combine the features from the Computer Science, Physics, Mathematics, Statistics, Quantitative Biology, and Quantitative Finance columns. The base network can be a deep neural network architecture such as a Transformer, MLP, or CNN.
5. Train the model using a suitable loss function such as binary cross-entropy loss. The model will output a binary vector for each data instance indicating the presence or absence of each label.
6. Evaluate the performance of the model on the test set using metrics such as accuracy, precision, recall, and F1 score.
7. If the performance is not satisfactory, you can fine-tune the hyperparameters of the model and experiment with different network architectures.
8. Once you are satisfied with the performance, you can use the model to make predictions on new data instances by feeding them through the trained model.

Note that this is just one possible logic for using DataMUX to perform multi-label classification. The specific implementation details will depend on the programming language and libraries you use, as well as the specific dataset and task at hand.

Regenerate response



Now using this approach of DatMUX I want to do a multi label classification task, my dataset is in form of a csv file and it contains the following columns [
ID TITLE ABSTRACT Computer Science Physics Mathematics Statistics Quantitative Biology Quantitative Finance]
can you give a code and logic behind the same

Sure! Here's some sample code that shows how you can use DataMUX for multi-label classification on a CSV file containing the columns you described:

python Copy code

```python
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModel
import torch.nn as nn

# Load the CSV file into a pandas dataframe
df = pd.read_csv("my_dataset.csv")

# Define the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Define the DataMUX layer
class DataMUXLayer(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.multiplex = nn.Linear(input_size, output_size)
        self.demultiplex = nn.Linear(output_size, input_size)

    def forward(self, inputs):
        mixed = self.multiplex(inputs)
        mixed_encoded = model(inputs["input_ids"], attention_mask=ir
        mixed_encoded = mixed_encoded.unsqueeze(1) # Add a new dimer
        mixed_encoded = mixed_encoded.repeat(1, mixed.size()[1], 1)
        mixed = mixed.unsqueeze(2) # Add a new dimension for the hic
        mixed = mixed.repeat(1, 1, mixed_encoded.size()[2]) # Repeat
        mixed = mixed * mixed_encoded # Element-wise multiplication
        mixed = torch.sum(mixed, dim=1) # Sum along the hidden dimer
        demixed = self.demultiplex(mixed)
        return demixed

# Define the dataset and dataloader
class MyDataset(Dataset):
    def __init__(self,
        self.df = df
```

Regenerate response

```python
    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        inputs = tokenizer(row["TITLE"] + " " + row["ABSTRACT"], pac
        labels = torch.tensor(row[["Computer Science", "Physics", "M
        return {"inputs": inputs, "labels": labels}

batch_size = 8
dataset = MyDataset(df)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True

# Define the model
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.datamux = DataMUXLayer(768, 256)
        self.dropout = nn.Dropout(0.1)
        self.classifier = nn.Linear(256, 6)

    def forward(self, inputs):
        inputs = self.datamux(inputs)
        inputs = self.dropout(inputs)
        logits = self.classifier(inputs)
        return logits

model = MyModel()

# Define the loss function and optimizer
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    for batch in dataloader:
        inputs = batch["inputs"]
        labels = batch["labels"]
        logits = model(inputs)
        loss = loss_fn(logits, labels.float())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("Epoch {} Loss: {:.4f}".format(epoch+1,
```

In Application 2, also it wasn't able to complete the code.

I want to apply this approach of DataMUX that I just taught you to another application where I have a housing data which has column names as ['zpid', 'price', 'latitude', 'longitude', 'beds', 'baths', 'area'], the zpid is the corresponding image name, I want to combine the image with this tabular data and send it as a multi input to my model and do a prediction on price, give a code for the same along with logic in pytorch

To use DataMUX to combine image data with tabular data for a multi-input model in PyTorch, you can follow these steps:

1. Load the housing data from the CSV file using Pandas and split it into training and validation sets.
2. Load the corresponding images using PyTorch's `torchvision.datasets.ImageFolder` class, which assumes that the images are organized into subfolders by class label (in this case, the `zpid`).
3. Use PyTorch's `torch.utils.data.Dataset` and `torch.utils.data.DataLoader` classes to create separate data loaders for the tabular data and image data, respectively.
4. Define a custom PyTorch `nn.Module` for the base network, which will take in the mixed representation generated by the DataMUX layer and output a prediction for the target variable (in this case, `price`).
5. Define a custom PyTorch `nn.Module` for the DataMUX layer, which will take in the tabular data and the image data and combine them into a mixed representation that can be processed by the base network.
6. Define the loss function and optimizer for training the model.
7. Train the model using the data loaders and loss function, and evaluate its performance on the validation set.

Here is a sample code to implement the above steps:

python Copy code

```python
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset

# Load the housing data
housing_df = pd.read_csv('housing_data.csv')
train_df = housing_df.sample(frac=0.8, random_state=1)
val_df = housing_df.drop(train_df.index)

# Define the transforms to be applied to the images
image_transform = transforms.Compose([
    transforms.Resize(2
    transforms.CenterCr
```

Regenerate response

```python
])

# Load the images using ImageFolder
image_dataset = datasets.ImageFolder(root='images', transform=image_

# Define the custom dataset and dataloader for the tabular data
class TabularDataset(Dataset):
    def __init__(self, df):
        self.data = df[['latitude', 'longitude', 'beds', 'baths', 'a
        self.targets = df['price'].values

    def __getitem__(self, index):
        data = torch.FloatTensor(self.data[index])
        target = torch.FloatTensor([self.targets[index]])
        return data, target

    def __len__(self):
        return len(self.targets)

tabular_train_dataset = TabularDataset(train_df)
tabular_val_dataset = TabularDataset(val_df)
tabular_train_loader = DataLoader(tabular_train_dataset, batch_size=
tabular_val_loader = DataLoader(tabular_val_dataset, batch_size=32,

# Define the custom dataset and dataloader for the images
image_loader = DataLoader(image_dataset, batch_size=32, shuffle=Fals

# Define the base network for processing the mixed representation
class BaseNetwork(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x

# Define the DataMUX layer
class Data
```

References

1) https://arxiv.org/pdf/2202.09318v2.pdf 2) https://github.com/princeton-nlp/DataMUX/tree/main 3) https://huggingface.co/princeton-nlp 4) https://arxiv.org/pdf/2301.12900v2.pdf 5) https://www.kaggle.com/datasets/shivanandmn/multilabel-classification-dataset 6) https://rosenfelder.ai/multi-input-neural-network-pytorch/ 7) https://onedrive.live.com/?cid=595cf3061fab0fa5&id=595CF3061FAB0FA5!138981&authkey=!APNEbWW4z4-NzIQ