NLP HW3

Chi-Yeh Chen
chiyehc

Friday 8th November, 2024

Part1: Results

There are lots of settings during training such as *hidden size, hidden layers, batch size, epochs, learning rate, and optimizers* which might influence the model performance *(accuracy)*.

As a result, I try many different combinations of settings. Here are my results.

Config	Hidden size	Layers	Batch size	Epochs	Learning rate	Optimizer	Weight Decay	Validation Accuracy
Model 1	(input size, 128)	1	32	5	0.001	1e-4	Adam	87.99%
Model 2	(input size, 128)	1	32	5	0.001	1e-4	SGD	83.47%
Model 3	(input size, 128)	1	16	5	0.001	1e-4	Adam	88.04%
Model 4	(input size, 128)	1	64	10	0.001	1e-4	Adam	88.46%
Model 5	(input size, 128), (128,64)	2	64	10	0.001	1e-4	Adam	88.60%
Model 6	(input size, 64), (64,32)	2	64	10	0.001	1e-3	Adam	88.60%
Model 7	**(input size, 64), (64,32)**	**2**	**64**	**15**	**0.001**	**1e-4**	**Adam**	**89.27%**
Model 8	(input size, 64), (64,32)	2	64	15	0.001	1e-3	SGD	83.70%

Table 1: Experiment Results with Different Parameters

Part2: Analysis and discussion

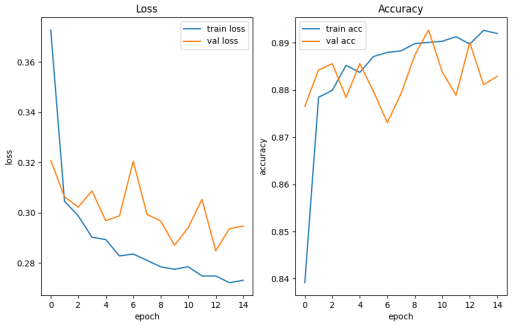
1. Loss and accuracy Analysis

- **Optimizer (Adam vs. SGD):** Models using the Adam optimizer (e.g., Model 1, 3, 4, 5, 6, 7) consistently demonstrate higher validation accuracy, with Model 7 achieving the best performance at **89.27%**. Adam is efficient and adapts learning rates per parameter, contributing to faster convergence. In contrast, models using SGD (e.g., Model 2 and Model 8) display lower validation accuracy, such as **83.70%** in Model 8, suggesting that SGD may require more fine-tuning or a different learning schedule for competitive performance.
- **Impact of Weight Decay:** Lower weight decay values (**1e-4**) lead to better model performance, as seen in Models 1, 2, 3, 4, 5, and 7. This regularization helps prevent overfitting without being overly restrictive. Higher weight decay (**1e-3**) in Models 6 and 8 shows similar or slightly lower validation accuracy, indicating it might inhibit model flexibility.
- **Impact of Hidden Size:** Models with larger hidden sizes, such as `(input size, 128)`, perform consistently well (**87-88%** validation accuracy). Models 6, 7, and 8 with smaller hidden sizes of `(input size, 64, 32)` also maintain good accuracy, with Model 7 achieving the highest at **89.27%**, suggesting that smaller architectures can generalize effectively when paired with robust optimization.
- **Impact of Layers:** Single-layer models (Models 1 to 4) perform reliably, achieving validation accuracies from **83.47% to 88.46%**. However, deeper models with two layers (Models 5, 6, 7, and 8) slightly improve performance, with Model 7 achieving the highest accuracy. This indicates that increased depth can benefit performance when combined with appropriate optimization strategies.
- **Conclusions:**

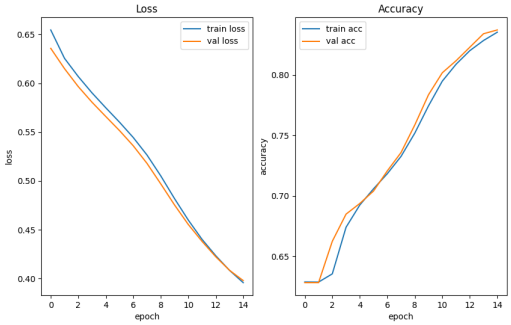
  (a) Adam Optimizer outperforms SGD, especially when combined with proper weight decay. However, from the figure below, SGD seems to be more stable but converge slower.

  (b) Weight decay of 1e-4 is preferable for better validation performance in these experiments.

  (c) Two-layer architectures (e.g., Model 7) with smaller hidden sizes can achieve high validation accuracy when optimized correctly.



(a) Model7 settings



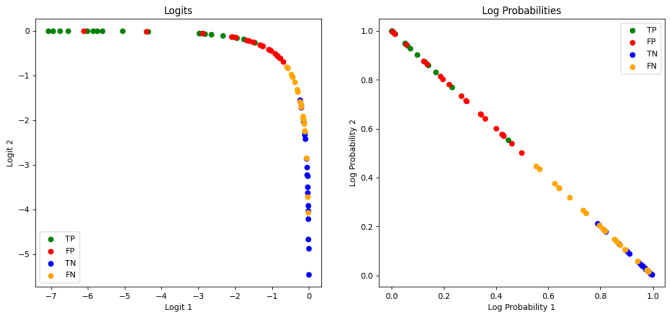
(b) Model8 settings

2. Error Analysis

- **Logits plot:** Logits are the raw, unnormalized scores produced by the model before applying a softmax function to get probabilities. In binary classification, these logits would generally be two values per sample, indicating the score for each class (e.g., [class1, class2]).
- **Log probabilities plots:** Log Probabilities are the natural logarithm of probabilities obtained by applying the softmax function to the logits. These provide a normalized measure of the model's confidence in each class.
- **Observation:**

  Observing the Logits plot, we can see a distinct clustering pattern where green (TP) and blue (TN) data points seem to spread along the x-axis and y-axis in a curved manner. This could indicate how certain data points (TP or TN) group together along axis, while red (FP) and orange (FN) might be more dispersed, spreading in curve.

  Observing the Log porbabilities plot, it seems that it spread more linearly and in the range between 0 and 1. The two parts, green (TP) and blue (TN) along the diagonal indicate successful predictions by the model, while the few points, red (FP) and orange (FN), in the middle suggest potential difficulty in accurately distinguishing between classes.



(a) logits and log probabilities

- **Example of TP, FP, FN, TN:** According to the following example, it is evident that potential problems for incorrect classification may arise from subword tokenization. In both FP and FN examples, the presence of ## in the tokenized lists indicates that certain words are split into subwords, which could hinder the model's ability to classify them correctly. On the other hand, sentences that are tokenized well do not exhibit this issue, as observed in the TP and TN cases.

  - **TP**:

```
TP Sample: the 27 most important things cats did on the internet in 2015
Tokenized: ['the', '27', 'most', 'important', 'things', 'cats', 'did', 'on', 'the', 'internet', 'in', '2015']

TP Sample: we tried animal balls, brains, udders, and tongues
Tokenized: ['we', 'tried', 'animal', 'balls', ',', 'brains', ',', 'ud', '##ders', ',', 'and', 'tongues']

TP Sample: this may be the first couple to have an official wedding hashtag
Tokenized: ['this', 'may', 'be', 'the', 'first', 'couple', 'to', 'have', 'an', 'official', 'wedding', 'hash', '##tag']

TP Sample: who would you end up with in " once upon a time " based on your zodiac sign
Tokenized: ['who', 'would', 'you', 'end', 'up', 'with', 'in', '"', 'once', 'upon', 'a', 'time', '"', 'based', 'on', 'your', 'zodiac', 'sign']

TP Sample: do you know these classic " star wars " characters
Tokenized: ['do', 'you', 'know', 'these', 'classic', '"', 'star', 'wars', '"', 'characters']
```

  - **FP**:

```
FP Sample: people are desperately waiting for nicki minaj to respond to remy ma ' s diss track
Tokenized: ['people', 'are', 'desperately', 'waiting', 'for', 'nick', '##i', 'mina', '##j', 'to', 'respond', 'to', 'remy', 'ma', ''', 's', 'di', '##ss', 'track']

FP Sample: warcraft is the critical bomb of the summer. i kinda liked it.
Tokenized: ['war', '##craft', 'is', 'the', 'critical', 'bomb', 'of', 'the', 'summer', '.', 'i', 'kinda', 'liked', 'it', '.']

FP Sample: the awards that should've been given out at the 2017 grammys
Tokenized: ['the', 'awards', 'that', 'should', "'", 've', 'been', 'given', 'out', 'at', 'the', '2017', 'grammy', '##s']

FP Sample: rick ankiel drank vodka before a start to deal with the yips
Tokenized: ['rick', 'an', '##kie', '##l', 'drank', 'vodka', 'before', 'a', 'start', 'to', 'deal', 'with', 'the', 'yi', '##ps']

FP Sample: what the rules are on flying with a gun
Tokenized: ['what', 'the', 'rules', 'are', 'on', 'flying', 'with', 'a', 'gun']
```

  - **FN**:

```
FN Sample: who is donald mcgahn?
Tokenized: ['who', 'is', 'donald', 'mc', '##ga', '##hn', '?']

FN Sample: liam hemsworth went vegan
Tokenized: ['liam', 'hem', '##sworth', 'went', 'vega', '##n']

FN Sample: french and english have messages of solidarity for the world
Tokenized: ['french', 'and', 'english', 'have', 'messages', 'of', 'solidarity', 'for', 'the', 'world']

FN Sample: buffalo bills player asks cops to shoot him during bizarre incident
Tokenized: ['buffalo', 'bills', 'player', 'asks', 'cops', 'to', 'shoot', 'him', 'during', 'bizarre', 'incident']

FN Sample: how republics end
Tokenized: ['how', 'republics', 'end']
```

  - **TN**:

```
TN Sample: screen actors guild to take strike vote
Tokenized: ['screen', 'actors', 'guild', 'to', 'take', 'strike', 'vote']

TN Sample: weening takes stage to gerardmer in photo finish
Tokenized: ['wee', '##ning', 'takes', 'stage', 'to', 'gerard', '##mer', 'in', 'photo', 'finish']

TN Sample: amazon does it again, features gandhi's picture on the flip - flops after tricolour doormats!
Tokenized: ['amazon', 'does', 'it', 'again', ',', 'features', 'gandhi', "'", 's', 'picture', 'on', 'the', 'flip', '-', 'flop', '##s', 'after', 'tri', '##col', '##our', 'door', '##mat', '##s', '!']

TN Sample: lewis hamilton wins 2009 hungarian grand prix
Tokenized: ['lewis', 'hamilton', 'wins', '2009', 'hungarian', 'grand', 'prix']

TN Sample: millionaire from california throwing hat into nyc mayoral race
Tokenized: ['millionaire', 'from', 'california', 'throwing', 'hat', 'into', 'nyc', 'mayoral', 'race']
```

Part3: Methodology

1. Loss and accuracy Analysis

Record Training process

```python
def record_metrics(epoch, train_loss, val_loss, train_acc, val_acc, filename='/content/drive/MyDrive/Colab Notebooks/hw3-handout/log/log8.csv'):
    # Check if the log file already exists
    file_exists = os.path.isfile(filename)

    # Open the file in append mode; create it if it does not exist
    with open(filename, 'a', newline='') as f:
        writer = csv.writer(f)

        # If the file does not exist, write the header row first
        if not file_exists:
            writer.writerow(['epoch', 'train_loss', 'val_loss', 'train_acc', 'val_acc'])

        # Write the current epoch's metrics to the file
        writer.writerow([epoch, train_loss, val_loss, train_acc, val_acc])
```

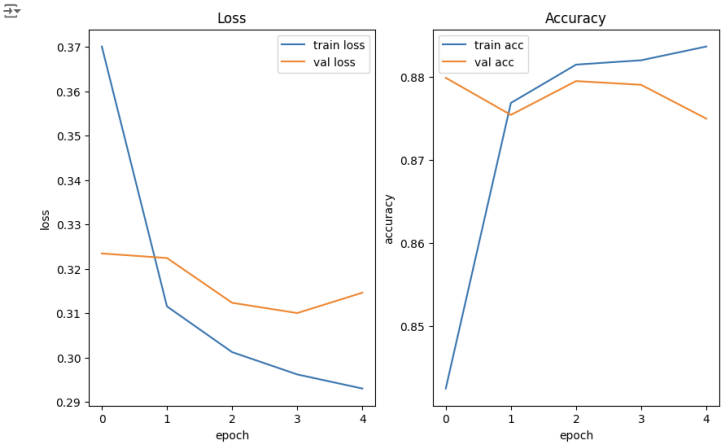
```python
def plot_log(file, sava_plot_path):
    df = pd.read_csv(file)
    plt.figure(figsize=(10,6))
    plt.subplot(1,2,1)
    plt.plot(df['epoch'], df['train_loss'], label='train loss')
    plt.plot(df['epoch'], df['val_loss'], label='val loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(loc='best')
    plt.title('Loss')
    plt.subplot(1,2,2)
    plt.plot(df['epoch'], df['train_acc'], label='train acc')
    plt.plot(df['epoch'], df['val_acc'], label='val acc')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.legend(loc='best')
    plt.title('Accuracy')
    save_plot = os.path.join(save_plot_path, (os.path.splitext(os.path.basename(file))[0]+'.png'))
    plt.savefig(save_plot)
    plt.show()
```

Setting1:

1. Epoch: 5
2. Batch size: 32
3. Learning rate: 0.001
4. Optimizer: Adam
5. Weight decay = 1e-4
6. Hidden layers: (128,num_class)

```python
save_plot_path = '/content/drive/MyDrive/Colab Notebooks/hw3-handout/log_plot'
```

```python
plot_log('/content/drive/MyDrive/Colab Notebooks/hw3-handout/log/log.csv', save_plot_path)
```



Setting2:

1. Epoch: 5
2. Batch size: 32
3. Learning rate: 0.001
4. Optimizer: SGD
5. Weight decay = 1e-4
6. Hidden layers: (128,num_class)

2. Error Analysis

- **Function of saving validation results in dictionary**

Error Analysis

```
def evaluate_val(model, dataset, batch_size, device, collate_fn=None):
    # Set the model to evaluation mode and move it to the specified device
    model = model.eval().to(device)
    dataloader = DataLoader(dataset, batch_size, shuffle=False, collate_fn=collate_fn)

    lossfn = nn.NLLLoss()
    loss_history = []
    acc_history = []

    # Dictionary to categorize results into True Positives (TP), True Negatives (TN),
    # False Positives (FP), and False Negatives (FN) for further analysis
    validation_results = {'TP': [], 'TN': [], 'FP': [], 'FN': []}

    # Disable gradient calculation for evaluation to save memory and computation
    with torch.no_grad():
      # Iterate through each batch in the dataloader
      for i, batch in enumerate(dataloader):
          # Move each tensor in the batch to the device
          batch = {k: v.to(device) for k, v in batch.items() if isinstance(v, torch.Tensor)}

          y = batch.pop('label')
          logits = model(**batch)
          loss = lossfn(logits, y)
          pred = logits.argmax(1)
          # Calculate batch accuracy and add the loss and accuracy values to history lists
          acc = (pred == y).float().mean()
          loss_history.append(loss.item())
          acc_history.append(acc.item())

          # Categorize each prediction as TP, TN, FP, or FN
          for i in range(len(y)):
              # Get the actual label and predicted label for the i-th sample in the batch
              true_label = y[i].item()
              pred_label = pred[i].item()

              # Retrieve the input IDs for further analysis (e.g., text data associated with predictions)
              input_ids = batch['input_ids'][i]  # Assumes 'input_ids' is a key in the batch

              # Categorize the predictions based on the true and predicted labels
              if pred_label == 1 and true_label == 1:
                  validation_results['TP'].append((input_ids, logits[i]))  # True Positive
              elif pred_label == 0 and true_label == 0:
                  validation_results['TN'].append((input_ids, logits[i]))  # True Negative
              elif pred_label == 1 and true_label == 0:
                  validation_results['FP'].append((input_ids, logits[i]))  # False Positive
              elif pred_label == 0 and true_label == 1:
                  validation_results['FN'].append((input_ids, logits[i]))  # False Negative

    # Return the dictionary with categorized predictions for TP, TN, FP, and FN
    return validation_results
```

- **Function of getting logits and log probabilities**

```
[24] def get_logits_and_probabilities(model, samples, device):
        # Set the model to evaluation mode (disables dropout layers, etc.)
        model.eval().to(device)

        # Tokenize the input samples and move them to the specified device (e.g., GPU)
        inputs = tokenizer(samples, return_tensors="pt", padding=True, truncation=True).to(device)
        with torch.no_grad():
          # Obtain log probabilities from the model directly (assumes model outputs log-probs)
          log_probs = model(**inputs)  # Directly get log probabilities

        # Convert log probabilities to regular probabilities for further analysis
        probabilities = log_probs.exp()  # Exponential to get back to probability space

        # Convert probabilities back to logits (log-odds ratio) for comparison
        logits = torch.log(probabilities / probabilities.sum(dim=-1, keepdim=True))  # Revert to logits

        # Detach tensors and move to CPU for numpy conversion (for easy manipulation and visualization)
        return logits.detach().cpu().numpy(), log_probs.detach().cpu().numpy()
```

- **Funtion of plotting the logits and log probabilities in 2D space**

```python
def plot_metrics(metrics_dict, save_plot_path):
    # Set up the figure with a defined size for subplots
    plt.figure(figsize=(14, 6))

    # Colors and labels for each category
    categories = {
        'TP': ('green', 'o'),  # True Positives: green circles
        'FP': ('red', 'o'),    # False Positives: red triangles
        'TN': ('blue', 'o'),   # True Negatives: blue squares
        'FN': ('orange', 'o')  # False Negatives: orange x-marks
    }

    # Plot logits (log-odds) for the two classes (subplot 1)
    plt.subplot(1, 2, 1)
    for category, (color, marker) in categories.items():
        if category in metrics_dict:
            logits = metrics_dict[category]['logits']
            plt.scatter(logits[:, 0], logits[:, 1], color=color, marker=marker, label=category)
    plt.title('Logits')
    plt.xlabel('Logit 1')
    plt.ylabel('Logit 2')
    plt.legend()  # Add a legend for distinguishing the categories

    # Plot log probabilities for the two classes (subplot 2)
    plt.subplot(1, 2, 2)
    for category, (color, marker) in categories.items():
        if category in metrics_dict:
            log_probs = metrics_dict[category]['log_probs']
            plt.scatter(log_probs[:, 0], log_probs[:, 1], color=color, marker=marker, label=category)
    plt.title('Log Probabilities')
    plt.xlabel('Log Probability 1')
    plt.ylabel('Log Probability 2')
    plt.legend()  # Add a legend for distinguishing the categories

    # Save the plot to the specified path with a descriptive name
    save_plot = os.path.join(save_plot_path, 'logits_and_log_probabilities')
    plt.savefig(save_plot)

    # Display the plot in the notebook
    plt.show()
```

- **Implementation**

```python
[37] # Initialize a BERT-based model for text classification with 2 output classes
     # 'bert-base-uncased' is a pre-trained BERT model; freeze_bert=True means the BERT layers will not be fine-tuned
     model = BertForTextClassification('bert-base-uncased', 2, freeze_bert=True)

     # Load the saved model parameters from a specified file path and set it to the CPU if needed
     # map_location=torch.device('cpu') ensures compatibility if the model was saved on a different device (e.g., GPU)
     model.load_state_dict(torch.load('/content/drive/MyDrive/Colab Notebooks/hw3-handout/model/model5.pt', map_location=torch.device('cpu')))

     # Move the model to the specified device (GPU or CPU) for evaluation
     model.to(device)

     # Generate predictions for the validation set using the trained model
     model.eval()  # Set the model to evaluation mode

     # Call the evalute_val function to generate the validation_results
     validation_results = evaluate_val(model, dataset['validation'], batch_size=64, device=device, collate_fn=tokenize)
```

```
顯示隱藏的輸出內容
```

```python
[38] print(len(validation_results))
```

```
4
```

```python
[39] for keys in validation_results.keys():
       print(f"{keys}: {len(validation_results[keys])}")
```

```
TP: 667
TN: 1281
FP: 97
FN: 146
```

```python
import random

# Randomly sample up to 20 items from each category
validation_results_20 = {
    'TP': random.sample(validation_results['TP'], min(20, len(validation_results['TP']))),
    'TN': random.sample(validation_results['TN'], min(20, len(validation_results['TN']))),
    'FP': random.sample(validation_results['FP'], min(20, len(validation_results['FP']))),
    'FN': random.sample(validation_results['FN'], min(20, len(validation_results['FN'])))
}

# Decode input IDs for each category
tp = [tokenizer.decode(items[0], skip_special_tokens=True) for items in validation_results_20['TP']]
fp = [tokenizer.decode(items[0], skip_special_tokens=True) for items in validation_results_20['FP']]
tn = [tokenizer.decode(items[0], skip_special_tokens=True) for items in validation_results_20['TN']]
fn = [tokenizer.decode(items[0], skip_special_tokens=True) for items in validation_results_20['FN']]

# Print lengths for verification
print("TP samples:", len(tp))
print("FP samples:", len(fp))
print("TN samples:", len(tn))
print("FN samples:", len(fn))

# Compute logits and probabilities for each category
logits_tp, probs_tp = get_logits_and_probabilities(model, tp, device)
logits_fp, probs_fp = get_logits_and_probabilities(model, fp, device)
logits_tn, probs_tn = get_logits_and_probabilities(model, tn, device)
logits_fn, probs_fn = get_logits_and_probabilities(model, fn, device)

# Create a metrics dictionary for plotting
metrics_dict = {
    'TP': {'logits': logits_tp, 'log_probs': probs_tp},
    'FP': {'logits': logits_fp, 'log_probs': probs_fp},
    'TN': {'logits': logits_tn, 'log_probs': probs_tn},
    'FN': {'logits': logits_fn, 'log_probs': probs_fn}
}

# Define the file path where the metric plots (logits and probabilities) will be saved
save_plot_path = '/content/drive/MyDrive/Colab Notebooks/hw3-handout/metrics'

# Generate and save plots of the logits and log probabilities for analysis
plot_metrics(metrics_dict, save_plot_path)
```

```
TP samples: 20
FP samples: 20
TN samples: 20
FN samples: 20
```