MAC05921 – Deep Learning – Relatório Tarefa 2

Giovani Tavares
giovanitavares@usp.br (10788620)

University of Sao Paulo — September 6, 2025

Este relatório contem a descrição do experimento realizado na **Tarefa 2**. Foram realizados os passos *a, b, c* e *f* da seção **Sugestão de explorações adicionais**.

1 Etapa 1: Rede Neural Convencional (NN)

1.1 Preparação dos Dados

Foi utilizado o conjunto de dados **FashionMNIST** (80% para treinamento e 20% para validação, sendo o conjunto de testes carregado separadamente). Para cada amostra de imagem, duas transformações foram aplicadas: as imagens foram transformadas em tensores e normalizadas para o intervalo $[-1, 1]$.

1.2 Definição do Modelo

Uma rede neural totalmente conectada (MLP) foi definida com a seguinte arquitetura:

- **Camada de Entrada**: Achatamento (`nn.Flatten`) da imagem 28×28.
- **Camada Oculta**: Uma camada densa (`nn.Linear`) seguida por uma função de ativação `ReLU`.
- **Regularização**: `Dropout` é aplicado para mitigar o overfitting.
- **Camada de Saída**: Uma camada densa final que produz as 10 pontuações de classe.

1.3 Otimização de Hiperparâmetros

Uma busca aleatória (**Random Search**) foi realizada em 10 tentativas para encontrar a melhor combinação de hiperparâmetros que maximiza a acurácia no conjunto de validação. Os hiperparâmetros ajustados foram os seguintes:

- `hidden_size`: tamanho da camada oculta.
- `dropout`: porcentagem de dropout.
- `weight_decay`: regularização L2.
- `batch_size`: tamanho do lote de dados.
- `lr`: taxa de aprendizado.
- `optimizer`: otimizador (AdamW ou SGD).

Cada combinação foi limitada por uma estratégia de **early stopping** em ele foi interrompido quando a acurácia no conjunto de validação não melhorou significativamente por 5 épocas seguidas ou mais.

1.4 Treinamento com Conjuntos Desbalanceados

O modelo com a melhor combinação de hiperparâmetros foi então treinado em cinco conjuntos de dados de treinamento com diferentes níveis de desbalanceamento de classe. A classe 0 foi definida como a classe minoritária, e sua representação no conjunto de treinamento foi reduzida para frações de 1.0 (balanceado), 0.7, 0.5, 0.3 e 0.1.

2 Etapa 2: Rede Neural Convolucional (CNN)

2.1 Arquitetura da CNN

Uma **Rede Neural Convolucional** foi definida para comparação. A arquitetura foi composta por:

1. Uma primeira camada convolucional (`nn.Conv2d`).
2. Uma função de ativação `ReLU`.
3. Uma segunda camada convolucional.
4. Uma função de ativação `ReLU`.
5. **Adaptive Average Pooling** para redimensionar a saída para 1×1.
6. Achatamento.
7. Uma camada linear final (`nn.Linear`) para a classificação.

O treinamento foi feito utilizando-se a mesma taxa de aprendizagem, *batch size*, *weight decay* e otimizador da *NN* para garantir melhor comparabilidade.

2.2 Otimização dos Parâmetros de Treinamento

O número de parâmetros treináveis da NN com a melhor configuração foi calculado como 407.050. O objetivo foi encontrar combinações de canais de convolução (C) e tamanho do kernel (K) para a CNN de modo que o número de parâmetros treináveis foi aproximadamente o mesmo da NN. A equação para o número de parâmetros da CNN é:

$$\text{NumberOfTrainableParameters(CNN)} = (K^2)C^2 + (K^2 + 12)C + 10$$

Essa equação foi resolvida para C para diferentes valores de K (de 3 a 11 com passo de 2) para encontrar as combinações ideais, sendo a melhor encontrada equivalente a um *kernel* de tamanho 11 e $C = 64$.

3 Resultados

Observou-se que a **CNN** teve melhores resultados de acurácia no conjuntos de testes do que a **NN** para todos os níveis de desbalanceamento, apesar de a diferença de acurácia ter sido de apenas *2%* em todos os níveis com exceção do caso em que a classe minoritária foi reduzida para apenas *10%* da quantidade no conjunto original, quando a **CNN** foi *6%* mais acurada.

A **CNN** manteve acurácia constante para conjuntos de treinamento balanceados e desbalanceados (*91%*), enquanto a **NN** só viu sua acurácia cair a partir de quando a classe minoritária foi reduzida para apenas *30%* da quantidade no conjunto original.

Ambos modelos demonstraram queda na *performance* com a adição do ruído do tipo sal e pimenta no conjunto de testes, sendo que a acurácia da **NN** treinada no conjunto balanceado caiu de *89%* para *83%*. Já para a **CNN**, a queda também foi de *6%*: foi de *91%* para *85%*.

4 Implementação

A implementação das classes **NeuralNet** (**NN**) e **ConvNeuralNet** (**CNN**) foram feitas consultando-se a documentação oficial do **PyTorch**, tal quel a implentação da função **train_model** que contém o *loop* de treinamento. Códigos fora de funções foram implementados sem consultas externas, enquando as outras funções (*plot* de gráficos, criação dos conjuntos de treinamento desbalanceados e adição de ruído nas imagens) foram implementadas utilizando-se uma ferramenta de IA generativa. Todas as células do *notebook* contendo explicações também foram criadas sem consultas externas.