

Laboratorio 1: Implementación y Optimización de una Red Neuronal Feed-Forward con PyTorch

Objetivo General:


El objetivo de este laboratorio es que el estudiante implemente una red neuronal feed-forward (FFNN) en PyTorch, aplicando técnicas avanzadas de optimización y regularización para mejorar el rendimiento en una tarea de clasificación binaria.

Instrucciones:

1. Construcción de la Red (5 puntos):

- El estudiante debe implementar una red neuronal feed-forward con dos capas ocultas en PyTorch.
- Es necesario aplicar funciones de activación, como ReLU o tanh en las capas ocultas, y sigmoid en la capa de salida.
- La red deberá configurarse para trabajar adecuadamente con la entropía cruzada como función de pérdida.
- **Ejemplo de código:**

python

 Copy code

```
import torch.nn as nn

class SimpleFFNN(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(SimpleFFNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, output_size)

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

2. Optimización e Inicialización (10 puntos):

- El estudiante debe experimentar con diferentes algoritmos de optimización, como **SGD**, **SGD con Momentum**, **SGD con Nesterov Momentum**, **RMSProp** y **Adam**. Cada optimizador debe ser probado y comparado para analizar cómo impacta en la convergencia y el rendimiento del modelo.
- Se debe aplicar **normalización de los datos de entrada** antes de ser procesados por la red para garantizar una distribución uniforme y mejorar la estabilidad del entrenamiento.
- Se espera que el estudiante explore distintas estrategias de inicialización de pesos, incluyendo:
 - **Inicialización de Xavier** (o Glorot): Ideal para redes con funciones de activación simétricas como tanh o sigmoid, y mejora la propagación de los gradientes en redes profundas.
 - **Inicialización de He**: Recomendada para redes con funciones de activación como ReLU, ya que ayuda a manejar el problema del desvanecimiento de gradientes.
- El estudiante debe documentar las observaciones sobre cómo la inicialización de los pesos y la normalización de los datos afectan la estabilidad del entrenamiento, la rapidez de la convergencia y el rendimiento final del modelo.
- **Sugerencia de análisis**: Comparar los resultados obtenidos con cada optimizador e inicialización utilizando métricas como la pérdida de entrenamiento y precisión, además de visualizar las curvas de aprendizaje para cada configuración.

3. Regularización (10 puntos):

- El estudiante debe aplicar diversas técnicas de regularización en la red para mejorar su capacidad de generalización:
 - **Dropout**: Implementar Dropout en las capas ocultas para reducir el sobreajuste y mejorar la robustez del modelo.
 - **Batch Normalization**: Utilizar Batch Normalization para estabilizar y acelerar el proceso de entrenamiento.

- **Regularización L2 (penalización de peso):** Implementar la regularización L2 en la función de pérdida para prevenir que los pesos de la red crezcan demasiado, promoviendo modelos más simples y generalizables.
- Se espera que el estudiante implemente un modelo **ensemble** usando técnicas como **bootstrap** para combinar varias redes entrenadas con diferentes subconjuntos de datos. Esto ayudará a mejorar la estabilidad y precisión de las predicciones, reduciendo la varianza del modelo.
- Se recomienda utilizar **early stopping** para detener el entrenamiento cuando la pérdida de validación deje de mejorar, evitando el sobreentrenamiento.
- El estudiante deberá explorar el impacto de estas técnicas en el rendimiento del modelo, analizando cómo afectan la precisión, la estabilidad y la capacidad de generalización. Es importante documentar los cambios en los hiperparámetros de Dropout, Batch Normalization y L2, así como los resultados obtenidos con el modelo ensemble.

4. Entrenamiento y Evaluación (15 puntos):

- El estudiante debe entrenar la red en tres datasets diferentes:
 1. **RandomDataSet** (incluido en el notebook entregado): Este dataset básico permite verificar que la red esté correctamente implementada y que los procesos de optimización y regularización funcionen.
 2. **Dataset de clasificación binaria:** Un conjunto de datos alternativo que permita evaluar la red en una tarea de clasificación binaria real (por ejemplo, el dataset de detección de spam o el dataset de clasificación de cáncer de mama).
 3. **MNIST:** Un conjunto de datos más desafiante que contiene imágenes de dígitos escritos a mano, útil para evaluar el modelo en una tarea de clasificación multclasificación.
- Deberá dividir cada dataset en conjuntos de entrenamiento, validación y prueba (Test-Dev-Train), evaluando así la generalización de la red.
- Durante el entrenamiento, se requiere la creación de gráficos que muestren la evolución de la pérdida y la precisión a lo largo de las

épocas, permitiendo visualizar los efectos de las técnicas de optimización, regularización y ensemble aplicadas.

- **Métricas de referencia:** Se recomienda aspirar a una precisión de al menos 85% en el conjunto de prueba en MNIST para evaluar la efectividad del modelo.

5. Reporte Final (20 puntos):

- El estudiante debe elaborar un informe detallado que incluya los siguientes elementos:
 - **Descripción de la arquitectura:** Explicación de la estructura de la red, las técnicas de regularización aplicadas y la configuración del ensemble.
 - **Resultados gráficos y análisis:** Gráficos de pérdida y precisión que muestren el rendimiento de la red a lo largo del entrenamiento. Cada gráfico debe incluir un análisis sobre cómo las técnicas utilizadas (optimización, regularización, ensemble y early stopping) influyeron en la convergencia y estabilidad del modelo. Se recomienda utilizar tablas que muestren los resultados clave de cada configuración.
 - **Comparación de configuraciones:** Evaluación de los resultados obtenidos al aplicar diferentes algoritmos de optimización, técnicas de regularización y el modelo ensemble. El estudiante debe discutir el impacto de cada técnica en términos de precisión, estabilidad y capacidad de generalización.
 - **Conclusiones:** Resumen de las principales observaciones, indicando las configuraciones y técnicas que mejoraron el rendimiento del modelo, y posibles recomendaciones para futuros ajustes.

6. Desafíos Opcionales (No Puntuales):

- Implementar técnicas de ensemble avanzadas (como bagging o boosting).
- Utilizar un dataset más desafiante, como CIFAR-10, y analizar los resultados.

- Explorar el uso de **early stopping avanzado** con paciencia para mejorar la generalización.

7. **Checklist de Entrega:**

- Código funcional y bien documentado en un notebook de Jupyter (.ipynb).
- Gráficos de pérdida y precisión.
- Comparación de configuraciones de optimización, regularización y ensemble.
- Reporte en PDF con análisis detallado y conclusiones.

8. **Recursos Adicionales Sugeridos:**

- Videos o artículos sobre optimización avanzada en PyTorch y aplicaciones de ensemble.
- Documentación de PyTorch sobre optimizadores y técnicas de regularización.

(AYUDA) Problemas Comunes y Referencias:

1. Desbordamiento de Memoria en GPU:

- **Problema:** Durante el entrenamiento en GPU, puede producirse un desbordamiento de memoria, especialmente en datasets grandes.
- **Solución:** Reducir el tamaño del batch o utilizar `torch.cuda.empty_cache()` para liberar memoria.
- **Referencia:** PyTorch Forum - Clearing CUDA Memory

2. Problemas de Convergencia:

- **Problema:** La pérdida no disminuye o fluctúa de manera irregular, lo que puede indicar problemas de convergencia.
- **Solución:** Ajustar la tasa de aprendizaje, probar diferentes optimizadores o revisar la implementación de la red.
- **Referencia:** [Goodfellow, I., Bengio, Y., & Courville, A. \(2016\). Deep Learning. MIT Press - Capítulo 8.2](#)

3. Overfitting en el Modelo:

- **Problema:** La precisión en el conjunto de entrenamiento es alta, pero la precisión en el conjunto de prueba es baja, indicando sobreajuste.
- **Solución:** Utilizar técnicas de regularización, como Dropout o Early Stopping, y aumentar el tamaño del dataset si es posible.
- **Referencia:** Zhang, Y., & Wallace, B. (2015). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research

4. Desvanecimiento o Explosión de Gradientes:

- **Problema:** Los gradientes se vuelven extremadamente pequeños o grandes, lo que afecta el proceso de backpropagation.
- **Solución:** Utilizar Batch Normalization, Clipping de gradientes o ajustar la arquitectura de la red para estabilizar el entrenamiento.
- **Referencia:** [Pascanu, R., Mikolov, T., & Bengio, Y. \(2013\). On the difficulty of training Recurrent Neural Networks. Proceedings of ICML](#)

5. Problemas de Generalización entre RandomDataSet y Dataset Alternativo:

- **Problema:** El modelo obtiene buenos resultados en el RandomDataSet, pero tiene un rendimiento bajo en el dataset alternativo.
- **Solución:** Asegurarse de que la red esté suficientemente compleja para capturar las características del dataset alternativo, ajustar la regularización, y utilizar ensemble y early stopping para mejorar la capacidad de generalización.
- **Referencia:** [Goodfellow, I., Bengio, Y., & Courville, A. \(2016\). Deep Learning. MIT Press - Capítulo 7.5](#)