Clasificando Fashion-MNIST con PyTorch

Molina Franco*
Facultad de Matemática, Astronomía, Física y Computación,
Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina
(Dated: November 20, 2024)

En este trabajo exploramos el uso de redes neuronales para la clasificación de imágenes en el conjunto de datos Fashion-MNIST, utilizando la biblioteca PyTorch. Donde evaluaremos el desempeño de diferentes arquitecturas y estrategias de optimización.

INTRODUCCIÓN

Fashion-MNIST es un conjunto de datos compuesto por 70,000 imágenes en escala de grises de 28x28 píxeles, divididas en 10 categorías de prendas de vestir. Las imágenes están etiquetadas como camisetas, pantalones, abrigos, entre otros. Este dataset es una alternativa más compleja al clásico MNIST de dígitos manuscritos, lo que lo hace ideal para probar modelos más avanzados en clasificación.

El objetivo de este trabajo es implementar una red neuronal en PyTorch para clasificar estas imágenes, comparando diferentes configuraciones del modelo. Nos enfocamos en el impacto de parámetros como el learning-rate, el optimizador, el valor de dropout, el número de neuronas en las capas intermedias, el número de épocas de entrenamiento y el tamaño de los lotes.

Todas estas configuraciones se compararán evaluando del error promedio (*Cross Entropy Loss*) y la precisión a lo largo de las épocas en entrenamiento y validación.

TEORÍA

Función de Activación ReLU

La función de activación *ReLU* (*Rectified Linear Unit*) es ampliamente utilizada en redes neuronales debido a su simplicidad y efectividad. Transforma las entradas negativas en cero y deja las positivas sin cambios, introduciendo no linealidad en el modelo. Esto permite que la red aprenda relaciones complejas en los datos mientras mantiene eficiencia computacional. Además, *ReLU* ayuda a mitigar el problema del gradiente desaparecido, lo que facilita el entrenamiento de redes profundas.

Dropout

El *dropout* es una técnica de regularización que apaga aleatoriamente un porcentaje de las neuronas durante el entrenamiento. Esto previene que el modelo dependa excesivamente de ciertas neuronas específicas, promoviendo una representación más robusta de los datos y mejorando la capacidad de generalización. En este trabajo, utilizamos un *dropout* del 20%.

Optimizador Stochastic Gradient Descent (SGD)

El algoritmo *Stochastic Gradient Descent* (SGD) es una técnica eficiente para minimizar la función de pérdida. En lugar de usar todo el conjunto de datos para calcular el gradiente, utiliza *mini-batches*, reduciendo el costo computacional y facilitando actualizaciones más frecuentes. La tasa de aprendizaje ($\eta = 10^{-3}$) controla el tamaño de los pasos dados hacia el mínimo de la función de pérdida.

Optimizador Adam

El optimizador **Adam** (Adaptive Moment Estimation) combina las ventajas de *SGD* con técnicas de adaptación automática del learning-rate. Calcula promedios móviles del gradiente y su magnitud al cuadrado, lo que permite realizar ajustes dinámicos en cada iteración. Esto hace que Adam sea particularmente efectivo en problemas donde el gradiente es ruidoso o los hiperparámetros requieren ajustes más finos. En comparación con *SGD*, Adam converge más rápido y es menos sensible a la elección de la tasa de aprendizaje inicial.

Datos: Fashion-MNIST

El conjunto de datos **Fashion-MNIST** consiste en 70,000 imágenes de 28x28 píxeles divididas en 10 clases (por ejemplo, camisetas, pantalones, zapatos, etc.). Para preparar los datos:

- Las imágenes fueron *normalizadas*, es decir, sus valores se escalaron al rango $[0, 1]$ para facilitar el entrenamiento.
- Se desordenaron (*shuffled*) antes de dividirlas en lotes, asegurando que los patrones del conjunto no introdujeran sesgos.

CONFIGURACIÓN INICIAL DEL MODELO

A continuación, describimos las características principales de la red neuronal utilizada como punto de partida:

- **Arquitectura:** Modelo *feedforward* con una capa de entrada, dos capas intermedias de 128 y 64 neuronas con activación *ReLU*, un *dropout* de 0.2 y una capa de salida con 10 neuronas.
- **Optimización:** Entrenado utilizando el algoritmo *Stochastic Gradient Descent* (SGD) con una tasa de aprendizaje (η) de 10^{-3}.
- **Datos:** El conjunto de datos Fashion-MNIST fue dividido en 60,000 imágenes para entrenamiento y 10,000 para prueba. Las imágenes se normalizaron (valores entre 0 y 1) y se desordenaron para mejorar el aprendizaje.

Este modelo servirá como punto de partida para explorar el impacto de la modificación de parámetros en el desempeño de la red neuronal.

Cabe aclarar que en los grafico visualizaremos el error y la precision, en entrenamiento como en validacion y ademas se agrega un resultado erroneo estudiado en el curso referido a una medicion incorrecta realizada mientras se entrena el modelo. La dejamos planteada para si el lector decide comparar y llega a esos resultados, darse cuenta que dicha curva es erronea y tendra que repasar su codigo.

RESULTADOS

En esta sección, presentamos los resultados obtenidos al evaluar las diferentes configuraciones del modelo. Se comparan las métricas de error (*Cross Entropy Loss*) y precisión tanto en entrenamiento como en validación. Los gráficos presentados destacan las tendencias de las métricas a lo largo de las épocas.

Caso Base

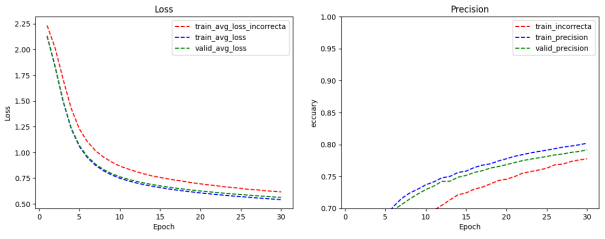

FIG. 1. Evolución del error y la precisión en el caso base.

La configuración inicial mostró una disminución significativa del error en las primeras 5 épocas, con un comportamiento asintótico posterior. La precisión superó el umbral de 0.7 en la época 5, estabilizándose en valores cercanos a 0.8 hacia el final. Este comportamiento representa una buena línea base para futuras comparaciones.

Caso Adam

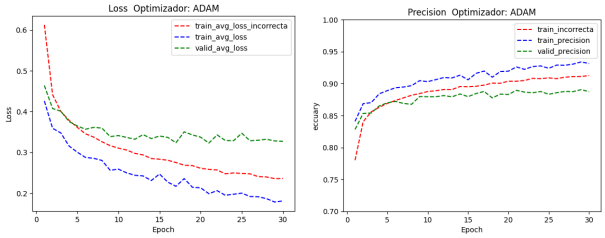

FIG. 2. Resultados al utilizar el optimizador Adam.

El uso del optimizador Adam produjo una convergencia más rápida. La precisión alcanzó valores entre 0.85 y 0.95 a partir de la quinta época, y el error se redujo considerablemente, con valores por debajo de 0.2 en entrenamiento. Esto confirma que Adam es más eficiente para este problema que SGD.

Caso Dropout

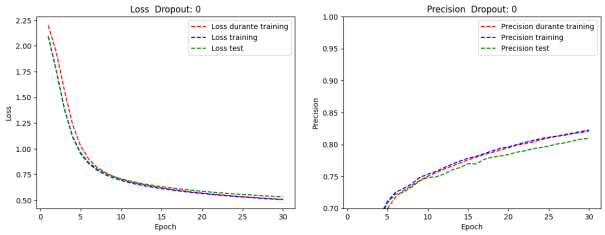

FIG. 3. Resultados al eliminar la regularización por dropout.

Al eliminar el *dropout*, se observó un leve aumento en la precisión en comparación con el caso base, aunque el impacto en el error fue marginal. Esto sugiere que en problemas de corta duración, la ausencia de *dropout* no afecta significativamente la capacidad del modelo para generalizar.

Caso Lotes

Aumentar el tamaño de los lotes produjo un aprendizaje más lento. El error se mantuvo alto (superior a 1.6) y la precisión no superó 0.6 en las primeras 30 épocas.

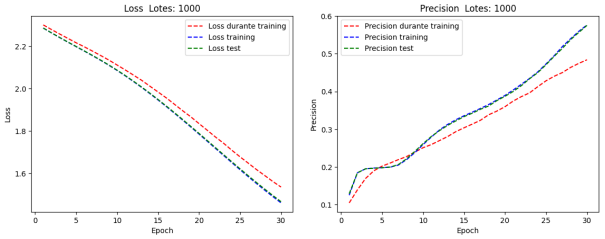

FIG. 4. Resultados al incrementar el tamaño de los lotes a 1000.

Este resultado demuestra que lotes más grandes limitan la capacidad del modelo para aprender rápidamente en pocas épocas.

Caso Learning Rate

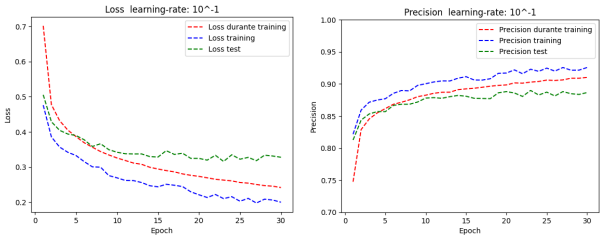

FIG. 5. Resultados con tasa de aprendizaje aumentada ($\eta = 10^{-1}$).

Un aumento en la tasa de aprendizaje permitió al modelo alcanzar valores de error más bajos y precisiones más altas en menos épocas. Sin embargo, los gráficos presentan una oscilación considerable, indicando posibles problemas de estabilidad que podrían mitigarse con técnicas de regularización.

Caso Número de Neuronas

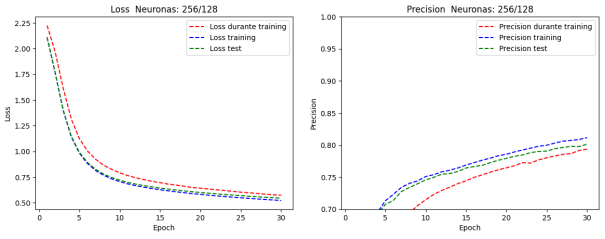

FIG. 6. Resultados al incrementar el número de neuronas intermedias.

Incrementar el número de neuronas en las capas intermedias produjo una mejora marginal en la precisión, mientras que el error se mantuvo similar al caso base. Este cambio tiene un impacto limitado en problemas simples o de corta duración.

DISCUSIÓN

A partir de los experimentos realizados, es evidente que ciertos ajustes tienen un impacto significativo en el rendimiento del modelo, mientras que otros presentan resultados marginales.

El uso del optimizador Adam y el aumento de la tasa de aprendizaje destacan como las modificaciones más beneficiosas, permitiendo convergencias más rápidas y mayores precisiones. Sin embargo, estas estrategias deben manejarse con cuidado para evitar problemas de estabilidad. Por otro lado, la eliminación del *dropout* y el incremento en el tamaño de los lotes no mostraron ventajas claras en este contexto, aunque podrían ser útiles en escenarios diferentes.

Para el modelo final, se propone combinar las características más exitosas: utilizando el optimizador Adam, con un *learning rate* inicial más alto ($\eta = 10^{-1}$) y una arquitectura ajustada con capas intermedias densas, además de reintroducir el *dropout* con valores moderados (por ejemplo, 0.2) para mejorar la regularización.

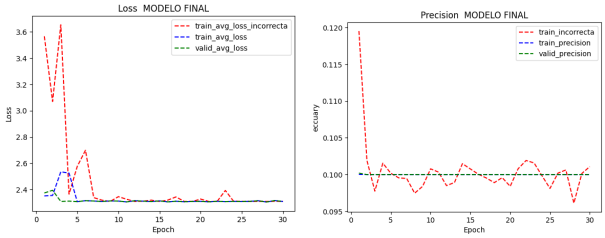

FIG. 7. Resultados del experimento final 1.

Como vemos la configuración propuesta fue un fracaso. Como Adam adapta el learning rate para cada parámetro en función de la primera y segunda derivada del gradiente. Un learning rate inicial alto resulta en actualizaciones excesivas en las primeras iteraciones.

Por otro lado, SGD, realiza actualizaciones uniformes. Por lo que nuestras prubeas iniciales funcionaron como tal por estar basadas en este optimizador.

Proponemos repetir el experimento pero manteniendo SGD para ver los resultados finales.

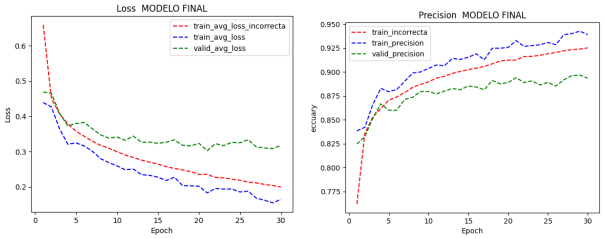

FIG. 8. Resultados del experimento final 2.

El modelo final, entrenado con el optimizador SGD, se consolida como el mejor experimento realizado. Su capacidad para alcanzar una alta precisión con una baja pérdida, tanto en entrenamiento como en validación, lo convierte en una solución ideal para el problema planteado.

Además, se observa que logra niveles de error extremadamente bajos, por debajo de 0.15, y una precisión más alta que en cualquier experimento previo, alcanzando valores cercanos a 0.95. Concluimos que este es el mejor modelo obtenido hasta ahora, aunque seguramente podría seguir mejorándose con ajustes adicionales.

CONCLUSIONES

Al final, nos dimos cuenta de que nuestra idea inicial de mejorar por separado distintas partes de la red y luego juntarlas en un solo modelo no funcionó tan bien como esperábamos. Por ejemplo, combinar Adam con un learning rate alto dio resultados desastrosos, a pesar de que por separado cada uno había mostrado mejoras. Esto nos enseñó algo clave: cuando se trata de optimizar una red neuronal, los cambios deben hacerse de forma progresiva, paso a paso, y no todos al mismo tiempo sin verificar cómo interactúan entre sí.

Este enfoque progresivo no solo evita sorpresas desagradables, sino que también nos permite entender mejor cómo afecta cada ajuste al modelo en conjunto. Si simplemente asumimos que juntar lo mejor de cada configuración siempre va a funcionar, podríamos terminar empeorando el rendimiento, como nos pasó aquí. Esto se debe a que los diferentes componentes (optimizador, learning rate, regularización, arquitectura, etc.) interactúan de formas complejas y a veces inesperadas.

Por otro lado, también vimos que encontrar configuraciones adecuadas para nuestro problema específico tiene un gran impacto. Ajustando bien los hiperparámetros, logramos resultados con una precisión muy alta y una tasa de error insignificante, y además en tiempos bastante cortos. Esto demuestra que, aunque pueda parecer un proceso de prueba y error, con paciencia y un enfoque estructurado es posible llegar a una red que cumpla perfectamente con nuestras necesidades.

En resumen, optimizar redes neuronales no es solo cuestión de probar configuraciones individuales, sino de construirlas de forma cuidadosa y progresiva. Al final, aunque es un camino iterativo, los resultados valen la pena porque nos llevan a modelos potentes y bien adaptados al problema.

* franco.molina13@mi.unc.edu.ar

[1] *Guia 12 - Clasificando Fashion-MNIST con PyTorch* (Catedra de Redes Neuronales).