Universidad de
SanAndrés

I302 - Aprendizaje Automático y Aprendizaje Profundo

Trabajo Práctico 3: Redes Neuronales

Manuel Borrell

13 de mayo de 2025

Ingeniería en Inteligencia Artificial

1. identificación de caracteres japoneses mediante NN

Resumen

El trabajo consistió de modelar una red neuronal para distinguir entre imágenes de 28x28 con 49 clases posibles, estas imágenes son de caracteres japoneses, desarrolle la primera red neuronal que consiste de una capa de entrada de 784=28x28 dos capas ocultas de 100 y 80 neuronas respectivamente y una capa de salida de 49 neuronas, en las capas ocultas use funcion de activacion relu y en la capa de salida softmax, para calcular los gradientes use back propagation y actualizar los pesos con gradient descent, usando 0.1 de LR y 300 epochs nos dio un acc de 0.55875 en val, este modelo es m0. ya con este modelo implementado pasamos a modelar m1 el cual consiste en una red neuronal pero con diferentes mejoras al algoritmo de entrenamiento (Rate scheduling lineal (con saturación) y exponencial, Mini-batch stochastic gradient descent, Optimizador ADAM, Regularización L2 y Regularización mediante early stopping), la mejor combinación de parámetros que encontré es, learning rate=0.01, rs=, batch size=1250, adam=1, lambda reg= 0.01, patience=20, y con estos parámetros la red dio una acc en val de 0.6300. Para los siguientes modelos utilizamos la librería pytorch, para modelar m2 usamos los que encontramos en m1 pero ahora con la red armada en pytorch dando una acc en val de 0.62875 muy parecida a la anterior ya que vienen de la misma red. Para modelar m3 use un grid search para encontrar las capas que me dan mejores resultados y tras probar distintas combinaciones la que me dio mejores resultados es de [100,80] en las capas ocultas por lo que quedó igual a m2. Para modelar m4 overfitee la red, para hacer esto saque la regularización y aumente la cantidad de neuronas para que la red aprenda perfectamente el dataset de train y así overfitee, use [5000] para las capas ocultas, terminando con una acc de 1 en train y de 0.70375 en val, esto nos demuestra que el dataset de val es muy parecido al de train por lo que aunque el modelo claramente overfitee la acc de val es buena.

2. Introducción

En este trabajo nuestro problema a resolver era identificar a que caracter japones pertenecen estas imagenes de 28x28 la cual es la entrada del sistema, un vector de 784 representando cada pixel de la imagen, luego usamos una red neuronal con distintas mejoras para obtener una predicción de cuál de las 49 clases pertenece la imagen.

Primero explore el dataset y me encontré con un dataset muy limpio sin nans por lo que decidí graficar unos ejemplos (figura 1)

Figura 1: ejemplos del dataset

dividí el dataset, en 80 % development y 20 % validation, luego development lo volvi a dividir en 80 % train y 20 % validation, las dimensiones resultantes son de

Conjunto	**Shape de X_images**	**Shape de y_images**
Train	(3200, 784)	(3200,)
Val	(800, 784)	(800,)
Test	(1000, 784)	(1000,)

Cuadro 1: Tamaños de los conjuntos de datos

3. Métodos

Para lograr la predicción como dijimos en la introducción utilizamos una nn con distintas mejoras ahora vamos a explicar cada una de ellas.

3.1. Rate Scheduling Lineal y Exponencial

Primero Rate Scheduling Lineal y Exponencial, esta mejora implementa un Lr que varía según la epoch en la que esté el modelo, de esta manera se logra dar pasos grandes al principio, cuando se supone que estamos más lejos del mínimo que buscamos y mientras nos acercamos damos pasos cada vez más lentos asi podemos converger y no oscilar.

- **Lineal con saturación:** la tasa de aprendizaje disminuye linealmente hasta un mínimo $\eta_{\min}$ según la epoch t:

  $$\eta(t) = \max\left(\eta_0 - kt,\ \eta_{\min}\right)$$

  η_0 = tasa de aprendizaje inicial, k = pendiente de decaimiento.

- **Exponencial:** la tasa de aprendizaje decrece exponencialmente en cada epoch:

  $$\eta(t) = \eta_0 \cdot e^{-\lambda t}$$

  donde $\lambda > 0$ controla el decaimiento.

3.2. Mini-Batch Stochastic Gradient Descent (SGD)

Esta mejora implementa mini-batches al entrenamiento, lo que hace la mejora es que en vez de pasa todo el dataset entero y luego actualizar los pesos pasa el dataset de a mini-batches y luego actualiza los pesos, de esta manera se actualiza los pesos más de una vez por epoch, esta mejora acelera mucho el tiempo de procesamiento ya que el cálculo del gradiente escala muy rápido con el tamaño del dataset, si usamos mini-batch con size 1 es SGD.

3.3. Optimizador Adam

Esta mejora implementa dos mejoras adagrad y momentum.

El momentum acumula los gradientes pasados y ayuda a que el gradiente gane una especie de momentum ya que su paso depende de los pasos anteriores.

el adagrad crea una taza de aprendizaje para cada parámetro.

3.4. Regularización L2

Para prevenir el overfiting, se aplicó penalización L2 sobre los pesos.

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \frac{\lambda}{2}\|\theta\|_2^2$$

3.5. Early Stopping

esta mejora consiste en detener el modelo antes de tiempo cuando la loss sobre el conjunto de val dejo de mejorar hace p epochs consecutivas

$$\mathcal{L}_{\text{val}}(t) > \min_{i<t} \mathcal{L}_{\text{val}}(i) \quad \text{por } p \text{ épocas},$$

3.6. busqueda de hiperparámetros

Para buscar los hiperparámetros hice una búsqueda secuencial ya que al querer hacer un grid search terminaba dando error y el tiempo de ejecución era muy grande, asi encontre los mejores parámetros para m1, para ver las distintas configuraciones de capas ocultas itere sobre diversas configuraciones con mayor y menor cantidad de capas y neuronas por capa, para cada configuración busque maximizar la acc de val, para encontrar la configuración que más overfitea, utilice el mismo programa pero busque maximizar la acc de train.

4. Resultados

4.1. M0

Empezando por m0, este modelo es el más básico, input dim=784, hidden layers=[100, 80], output dim=49 y ninguna de las mejoras. El modelo presentó buenos resultados para ser el más básico pero si subiamos las epochs el modelo overfiteaba rápido, entrene el modelo con 300 epoch y estos fueron los resultados

Conjunto	Accuracy	Cross-Entropy Loss
Entrenamiento	0.7594	0.9795
Validación	0.5650	1.7397

Cuadro 2: Desempeño de m0 en los conjuntos de entrenamiento y validación.

para ver como cambia la acc podemos observar la (fig 2) donde se muestra acc vs epoch para ambos conjuntos, claramente podemos ver como el modelo ya estaba empezando a overfitear ya que la acc de val comenzó a bajar.

4.2. M1

Este modelo es una mejora de m1 ya que le implementamos las mejoras que describimos anteriormente, los hiperparámetros con los que entrene la NN fueron, input dim=784, hidden layers=[100, 80], output dim=49, learning rate=0.01, rs=, batch size=1250, adam=1, lambda reg= 0.01, patience=10. La única mejora que no utilize fue el rate scheduling ya que la acc en val empeoraba, el modelo fue entrenado con 500 epoch y la acc sobre val fue de

Conjunto	Accuracy	Cross-Entropy Loss
Validación	0.6088	2.1874

Cuadro 3: Desempeño de m1 en validación.

y podemos ver el gráfico de acc vs epoch (fig 3) como el early stop freno en la epoch 17 y la acc mejoró por lo que demuestra que las mejoras ayudan mucho a la convergencia del modelo.

4.3. M2

Este modelo fue implementado ya con la librería pytorch, los hiperparametros fueron los mismos que el anterior modelo, los resultado fueron

Conjunto	Accuracy	Cross-Entropy Loss
Validación	0.59625	1.572955

Cuadro 4: Desempeño de m2 en validación.

El resultado, como se esperaba, fue muy parecido al anterior y su gráfico de acc vs epoch igual.

4.4. M3

Para este modelo tuve que explorar distintas configuraciones de las capas ocultas, tras hacer un grid search llegué al resultado de que la mejor combinación es [100,80] igual que las anteriores por lo que este modelo resulta idéntico al anterior y eso lo podemos ver en los resultados.

Conjunto	Accuracy	Cross-Entropy Loss
Validación	0.58875	1.6236

Cuadro 5: Desempeño de m3 en validación.

4.5. M4

Este modelo es el modelo que hay que overfittear por lo que decidí sobre complejizar la red y de esta manera que pueda aprender por completo train y así overfittiear, saque la reg y el early stop así puede overfitear sin que nada lo frene. los resultados del modelo fueron los siguientes

Conjunto	Accuracy	Cross-Entropy Loss
Validación	0.72375	2.25841

Cuadro 6: Desempeño de m4 en validación.

el modelo dio buenos de acc pero donde podemos ver que el modelo está overfitteando es en la loss ya que dio más alto que en los anteriores modelos.

4.6. performance en test

como último probamos los modelos en el dataset de test y estos fueron sus resultados

Modelo	Accuracy
m0	0.5350
m1	0.6040
m2	0.5820
m3	0.5850
m4	0.6970

Cuadro 7: Precisión de cada modelo evaluado en test

Con esto llegamos a la conclusión que el modelo que mejor desempeño es m2 y m3 ya que estos no tuvieron una loss muy alta pero sí una buena acc, aunque los modelos más overfit teados tengan mas acc no generalizan bien por lo que pueden dar peores resultados, a raíz de esto m3 fue el modelo elegido para el punto 5.

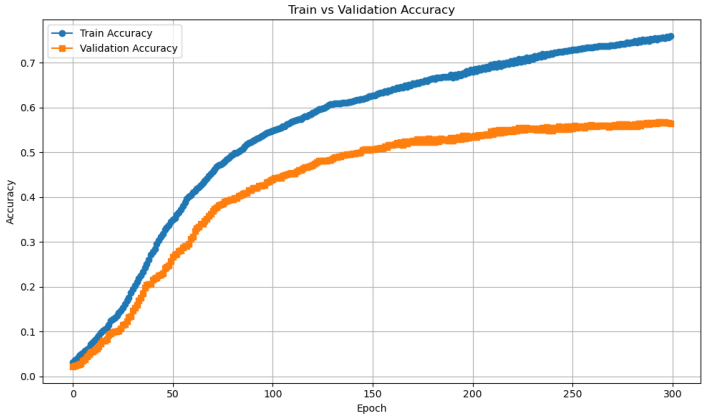

Figura 2: acc vs epoch, M0

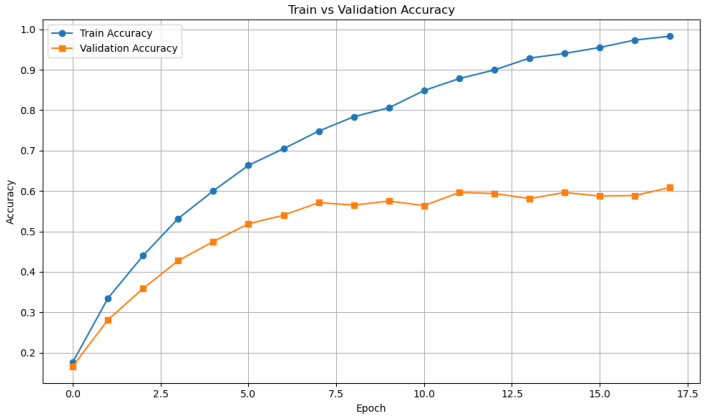

Figura 3: acc vs epoch, M1