Guatemala 19 de Julio de 2024
Daniel Valdez - 21240
Emilio Solano - 21212

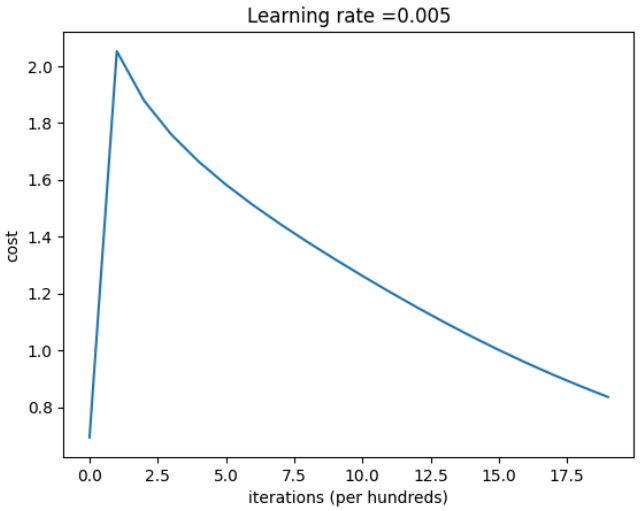
Deep Learning, sección 10

Laboratorio 1

Parte 1

1. ¿Qué se podría hacer para mejorar el rendimiento de esta red?
    - Existen diversas variantes para la mejora de la red neuronal de esta parte. Como primera opción, se puede experimentar con diferentes tasas de aprendizaje, tamaños de lote o números de épocas para encontrar la mejor combinación. Por otro lado, también se podría intentar incrementar el número de capas o de neuronas en cada una de ellas, así como implementar técnicas de regularización (L2 o Dropout), optimización (AdaGrad, Adam, etc.) o inclusive buscar más datos para aumentar el set de entrenamiento y así tener una mayor diversidad de datos.

2. Interprete la gráfica de arriba



    - La siguiente gráfica muestra la evolución de la función de pérdida, la cual mide el error del modelo, en función del número de iteraciones que se dan durante el entrenamiento, representadas en cifras divididas por cien.
    Se puede apreciar que la función de costo aumenta inicialmente, lo cual puede considerarse poco común, pero que aun así se puede justificar en la inicialización de los pesos o en el ajuste inicial del modelo.

Seguidamente, este mismo costo empieza a decaer constantemente, señal que el modelo está aprendiendo y mejorando en cuanto a rendimiento.

A pesar de contar con un indicador de aprendizaje continuo, sería de mucha utilidad contar con otras métricas como la precisión o exactitud en un conjunto de validación para así tener mejores visualizaciones en cuanto al rendimiento del modelo.

Parte 2

3. ¿En qué consiste ```optim.SGD```?
    - Consta de una implementación del optimizador de descenso de gradiente estocástico en PyTorch. Su utilidad surge de la necesidad de optimizar los parámetros del modelo utilizando gradientes calculados usando únicamente un subconjunto de datos, en vez de el lote completo. Al usar torch.optim.SGD, éste actualiza los pesos del modelo en dirección negativa del gradiente de la función de pérdida con respecto a los parámetros que presenta el modelo (PyTorch, 2023).

4. ¿En qué consiste ```nn.NLLLoss```?
    - Se puede describir como una función de pérdida de PyTorch, la cual es mayormente conocida como pérdida logarítmica negativa y se utiliza para tareas de clasificación en donde las salidas del modelo son logaritmos de probabilidades. Suele utilizarse más con capas de activación LogSoftmax y mide la discrepancia entre la distribución de probabilidad de las predicciones que da el modelo y etiquetas verdaderas (PyTorch, 2023).

5. ¿Qué podría hacer para mejorar la red neuronal, y si no hay mejoras, por qué?
    - De primeras, se podría realizar una búsqueda de hiperparámetros para encontrar la mejor combinación posible. Como se mencionó en la primera parte, también se podría aumentar la cantidad de capas y neuronas, incrementar la cantidad de datos de entrenamiento o emplear técnicas como L2 o Dropout para evitar el sobreajuste.

Referencias

PyTorch. (2023). *NLLLoss — PyTorch 2.3 documentation*. PyTorch. Retrieved July 18, 2024, from https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html

PyTorch. (2023). *SGD — PyTorch 2.3 documentation*. PyTorch. Retrieved July 18, 2024, from https://pytorch.org/docs/stable/generated/torch.optim.SGD.html