Montando um modelo dinâmico para construção e utilização de redes neurais

Frederico José Ribeiro Pelogia [1]

March 18, 2020

[1] Universidade Federal de São Paulo , fredpelogia@outlook.com

Como estávamos implementando

- Arquitetura fixa da rede.
- Backpropagation modelado como feito no papel.
- Código estruturado sem orientação a objeto.
- Dificuldade: Fazer testes para diferentes organizações de rede.

O que estávamos procurando

- Procurávamos algum framework que nos permitisse montar uma estrutura de rede e receber o gradiente em relação aos parâmetros ajustáveis para refinarmos eles manualmente de diferentes maneiras.

```
1 model = keras.models.Sequential([
2     keras.layers.Dense(units=16, input_dim=30,
      activation="relu"),
3     keras.layers.Dense(units=24, activation="relu"),
4     keras.layers.Dropout(0.5),
5     keras.layers.Dense(20, activation="relu"),
6     keras.layers.Dense(24, activation="relu"),
7     keras.layers.Dense(1, activation="sigmoid"),
8 ])
```

Listing 1: Interface do Keras

Nova proposta

- Proposta: Montar uma pequena biblioteca de *Deep Learning* que tenha interface parecida com a do Keras, mas que possamos construir os otimizadores como quisermos.
- Utilizando como base o código de **Joel Grus**.
  - Joel é pesquisador no *Allen Institute for AI*.
  - escreveu o livro *Data Science from Scratch*.
  - Github: https://github.com/joelgrus/joelnet

Módulos do repositório joelnet

- Arquivo **data.py**
  - Controla a separação em "mini lotes" (*batches*).
- Arquivo **layers.py**
  - Define o comportamento genérico de uma camada e também as diferentes funções de ativação possíveis.
- Arquivo **loss.py**
  - Define as possíveis formulações da função Erro da rede.
- Arquivo **nn.py**
  - Define a Classe *NeuralNetwork*, que é a mais importante da biblioteca. Essa Classe é a portadora dos métodos responsáveis pelo Feed-Forward e pelo Backpropagation.

Módulos do repositório joelnet

- Arquivo **optim.py**
  - Define os métodos de otimização que poderão ser utilizados par ao treinamento das redes.
- Arquivo **train.py**
  - Define a função *train*, que é responsável pelo treinamento da rede neural
- Arquivo **tensor.py**
  - Define o tipo abstrato *Tensor*, que, para nossa implementação simples, nada mais é do que o *ndarray* da biblioteca Numpy.

Para utilizarmos o repositório apresentado, foram necessárias algumas modificações e a adição de algumas funcionalidades.

- Em **layers.py** foram adicionadas as funções de ativação Sigmoid e reLu, além de suas primeiras derivadas.

$$Sig(x) = \frac{1}{1 + e^{-x}}$$

$$Sig'(x) = Sig(x) \cdot (1 - Sig(x))$$

$$reLu(x) = \frac{x + \sqrt{x^2 + \epsilon}}{2}$$

$$reLu'(x) = \frac{1}{2}\left(\frac{x}{\sqrt{x^2 + \epsilon}} + 1.0\right)$$