

Klasyfikacja punktów w R^2

April 2024

1 Wprowadzenie

X - wektor wejściowy W^1, b^1 - wagi i wyrazy wolne warstwy wejściowej do ukrytej W^2, b^2 - wagi i wyrazy wolne warstwy ukrytej do wyjściowej $Z^1 = XW^1 + b^1$ - wartości wejściowe funkcji aktywacji warstwy ukrytej $A^1 = f(Z^1)$ - wartości wyjściowe (po aktywacji) warstwy ukrytej $Z^2 = A^1W^2 + b^2$ - wartości wejściowe funkcji aktywacji warstwy wyjściowej $A^2 = g(Z^2)$ - wartości wyjściowe (po aktywacji) warstwy wyjściowej Y - zakodowany wektor poprawnych klas

Algorytm propagacji wstecznej (backpropagation, backprop) to metoda estymacji gradientu wykorzystywana do uczenia sieci neuronowych. Estymowane wartości są wykorzystywane do aktualizacji wag sieci.

W przypadku sieci z jedną warstwą ukrytą (oraz po jednej wejściowej i wyjściowej), używając funkcji aktywacji f^i dla i -tej warstwy, możemy opisać działanie naszej sieci (propagację wprzód, *forward pass*):

$$\bar{Y} = f^2(W^2 f^1(W^1 X))$$

gdzie $\bar{Y}$ jest wynikiem obliczeń sieci. Naszym celem jest (automatyczny) dobór wartości macierzy wag W^i (oraz wyrazów wolnych b^i , które minimalizują jakąś funkcję $L(Y, \bar{Y})$ (funkcję straty, *loss function*). Dobór oraz wymyślanie funkcji straty odpowiedniej dla danego problemu może być bardzo interesującym zajęciem w Królestwie Bajtocji. Jedną ze strategii automatycznego doboru wartości macierzy wag jest oszacowanie ich wpływu na funkcję straty L oraz odpowiednie ich aktualizowanie:

$$\begin{aligned} W^i &\rightarrow W^i - \alpha \frac{\partial L}{\partial W^i} \\ b^i &\rightarrow b^i - \alpha \frac{\partial L}{\partial b^i} \end{aligned}$$

gdzie α określa tempo uczenia (*learning rate*).

2 Zadanie

Zadanie polega na zaimplementowaniu klasy **Vanilla** z wykorzystaniem biblioteki numpy, oraz wykorzystaniu zaimplementowanej sieci do przeprowadzenia

pewnej analizy.

2.1 Implementacja

Klasa powinna zawierać następujące metody i pola:

1. **Konstruktor** przyjmujący parametry (`input_size`, `hidden_size`, `output_size`, `learning_rate`).
2. **forward**, obliczająca przejście wprzód (forward pass)
3. **predict**, przyjmującą wektor wartości wejściowych i zwracającą wektor klas
4. **compute_loss**, obliczającą wartość funkcji straty
5. **backprop**, wykonującą aktualizację wag
6. **train**, przyjmującą wartości wejściowe oraz odpowiadające im klasy, liczbę epok oraz rozmiar porcji danych (`batch_size`) `X_train`, `y_train`, `epochs`, `batch_size`)
7. **loss_track**, w którym przechowywane będą wartości funkcji straty obliczane podczas uczenia

Dodatkowo potrzebne będą funkcje **sigmoid**, **sigmoid_derivative**, **softmax**, **train_test_split** oraz **one_hot_encode**. Przykładowe wykorzystanie:

```
X = np.random.random((20000, 2)) # punkty z  $R^2$ 
y = ((X[:,0] - 0.5)**2 + (X[:,1] - 0.5)**2) < 0.33**2 # maska okręgu
X_train, X_test, y_train, y_test = train_test_split(X, y)
nn = Vanilla(2, 2, 2)
nn.train(X_train, y_train) # uczenie
preds = nn.predict(X_test) # predykcja
```

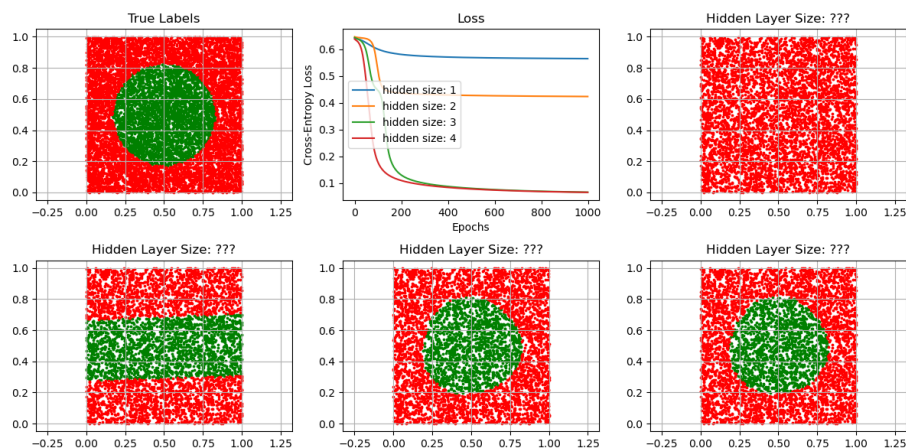
2.2 Analiza i wskazówki

Sieć należy wykorzystać do określenia minimalnego rozmiaru warstwy ukrytej potrzebnego do poprawnej klasyfikacji punktów w R^2 na zbiorze testowym. Ze stałą uczenia można eksperymentować, ja zacząłem od wartości 1, podobnie ze zmiennymi `batch_size` (100) oraz `epochs` (1000). Liczbę epok zazwyczaj dobieramy, analizując funkcję straty (np. wizualnie oceniając czy ma szansę maleć, jeśli będziemy dłużej trenować).

Ja w swojej implementacji wykorzystałem następującą funkcję straty:

```
def compute_loss(self, y_true, y_pred):  
    """Compute the cross-entropy loss."""  
    m = y_true.shape[0]  
    log_likelihood = -np.log(y_pred[range(m), y_true.argmax(axis=1)]  
        + 1e-9) # Added epsilon for numerical stability  
    loss = np.sum(log_likelihood) / m  
    return loss
```

W ramach rozwiązania, zobaczyć wykorzystanie sieci oraz funkcje straty dla kilku instancji klasy Vanilla, oto przykładowy wynik:



Rysunek 1: Przykładowy wynik projektu

Na pytania chętnie odpowiem mailowo: a.krupinski-ptaszek@uw.edu.pl
Powodzenia i przyjemnej zabawy.