Comparision between

element-wise learning

and

vectorized learning

2016025423 박주언

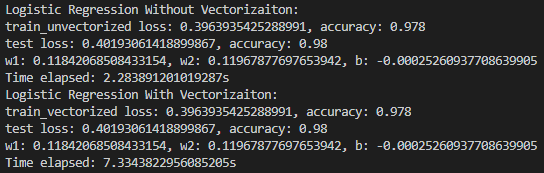
이번 과제를 수행하기 위해서 다음과 같이 두 파일을 만들었다.

* LogisticRegressionWithoutVectorization
* LogisticRegressionWithVectorization

말 그대로 전자는 vectorization없이 학습을 하는 것이고, 후자는 vectorization을 하고 학습을 수행하는 것이다.

둘 다 로직 자체는 같게 설정하였다. 같은 sigmoid 함수와 같은 loss function을 사용했으며, learning rate의 경우에도 똑같이 설정하였다. 다만, 둘을 비교하는 데에 있어서 같은 data를 가지고 비교를 해야 올바른 비교결과를 얻을 수 있기 때문에 random data를 만드는 과정에 있어서 같은 random value를 만든 후, unvectorized version과 vectorized version에 같이 넣어주는 식으로 같은 data를 부여했다.

실험에 앞서서 실험 환경은 Windows 10 64 bit, python version 3.7.3 (64bit)을 사용하였다. 일반적인 상황에서는 m=1000, n=100, learning rate alpha=0.0001로 놓고서 실험을 진행하였다.



# Time Comparison

-tested with train num = 1000, test num = 100-

|  |  |
| --- | --- |
|  | Time(s) |
| Element-wise version | 2.283891201019287s |
| Vectorized version | 7.3343822956085205s |

시간의 경우에는 예상했던 것과 다르게 Vectorized version이 더 오래 걸렸다. 여러 이유를 생각해봤지만, 우선 array의 size가 2이기 때문에 단순하게 w1, w2와 x1, x2의 dot product등 vectorized된 연산을 하는 것이 significant한 차이를 가져오지는 못했을 것이라고 생각한다.

|  |  |
| --- | --- |
| Element-wise version | Vectorized version |
| for data in datalist:      z = w1\*data['x1']+w2\*data['x2']+b      a = sigmoid(z)      da = -data['y']/a +  (1-data['y'])/(1-a)      dz = da \* a \* (1-a)      dw1 = data['x1']\*dz      dw2 = data['x2']\*dz      db = dz      batch\_dw1 += dw1 / len(datalist)      batch\_dw2 += dw2 / len(datalist)      batch\_db += db / len(datalist)  w1 -= alpha\*batch\_dw1  w2 -= alpha\*batch\_dw2  b -= alpha\*batch\_db | for Xi, Yi in zip(X, Y):      z = np.dot(W, Xi) + b      a = sigmoid(z)      da = -Yi/a + (1-Yi)/(1-a)      dz = da \* a \* (1-a)      dW = Xi\*dz      db = dz      batch\_dW = batch\_dW + dW/len(X)      batch\_db += db/len(X)  W = W - alpha\*batch\_dW  b -= alpha\*batch\_db |

실제로 코드의 차이를 보면 학습의 과정에서 크게 차이가 나는 부분은 z의 계산과 dw의 곱, 그리고 batch\_dw에서의 나눈 후 더해주는 과정이다. 시간을 잴 때 코드 전체에서 차이가 나는 부분이 이 부분이었다고만 생각을 한다면, numpy라이브러리를 썼음에도 불구하고 parallelized된 연산을 못했다거나, 변수를 copy하는 과정에서 더 오랜 시간이 걸렸을 것이라는 생각이 든다.

# Estimated unknown function parameters W & b

-tested with train num = 1000, test num = 100-

|  |  |  |
| --- | --- | --- |
|  | Unvectorized | Vectorized |
| w1 | 0.11842068508433154 | 0.11842068508433154 |
| w2 | 0.11967877697653942 | 0.11967877697653942 |
| B | -0.00025260937708639905 | -0.00025260937708639905 |

같은 data로 실험을 하다 보니 unvectorized version과 vectorized version 모두 다 같은 값의 w1, w2, 그리고 b를 가졌다. 사실 logic과 data가 같다면 이러한 결과가 나오는 것이 정상이다.

# Empirically determined (best) hyper parameter 𝜶

실험을 위해서 m=1000, n=100으로 놓고서 실험을 진행했다. 이 상태에서, learning rate **alpha**를 가지고 accuracy가 더 잘나오면 10배, 안나온다면 1/2배 하는 식으로 brute-force의 느낌으로 최적의 alpha를 찾아봤다.

|  |  |  |
| --- | --- | --- |
| Iteration | Alpha | accuracy |
| 1 | 0.0001 | 0.999 |
| 2 | 0.001 | 0.95 |
| 3 | 0.0005 | 1.0 |

원시적인 방법으로 찾은 alpha의 값으로는 0.0005가 최적의 alpha였다.

수학적으로 따지면 에서 최적의 alpha를 찾는 것이 목표일 것이고, 주어진 함수를 미분함으로써 값을 찾을 수 있을 것이다.

# Accuracy

For unvectorized version:

|  |  |  |  |
| --- | --- | --- | --- |
|  | m=10, k=100 | m=100, k=100 | m=1000, k=100 |
| With train set | 1.0 | 0.96 | 0.978 |
| With test set | 0.89 | 0.99 | 0.98 |
|  | m=100, k=10 | m=100, k=100 | m=100, k=1000 |
| With train set | 0.94 | 0.96 | 0.97 |
| With test set | 0.9 | 0.99 | 0.985 |

For vectorized version:

|  |  |  |  |
| --- | --- | --- | --- |
|  | m=10, k=100 | m=100, k=100 | m=1000, k=100 |
| With train set | 1.0 | 0.96 | 0.978 |
| With test set | 0.89 | 0.99 | 0.98 |
|  | m=100, k=10 | m=100, k=100 | m=100, k=1000 |
| With train set | 0.94 | 0.96 | 0.97 |
| With test set | 0.9 | 0.99 | 0.985 |

아무래도 같은 data를 가지고 같은 logic을 가지고 실험을 하다 보니까 같은 결과를 보였다. 사실 다른 결과가 나오는 것이 더 이상하겠지만 말이다.