Training a linear classifier

이름 : 임효석
국민대학교 전자공학부
imhyoseok@kookmin.ac.kr

요 약

영상 toy dataset을 생성 후 이미지에 난수 노이즈 배열을 더해준다. 생성된 toy dataset은 training data와 test data로 나눠줍니다. linearClassifier와 softmax loss function를 정의하고 gradient descent 알고리즘을 구현해준다. 마지막으로 test data를 사용하여 학습된 모델의 성능을 평가하고, classification accuracy를 출력해 모델 성능을 평가해줍니다. 이미지 data는 두 가지 class(흑/백, 백/흑)로 분류됩니다.

1. 서론

주피터 노트북을 이용해 data를 생성하고 training data와 test data로 분리한다. softmax loss function을 정의하고 gradient descent 알고리즘을 구현해 linearClassifier를 학습 후 두 가지 class 이미지를 구별 후 정확도를 판별하는 코드를 구현한다.

linearClassifier는 linear boundary를 가지며 이 boundary는 공간을 두 개의 “half space”로 분할한다. 즉 기하학적 관점에서 생각한다면 linear boundary로 구역을 나누고 구역별로 class를 분류 하는 것이며 이 때 class를 잘 구별해주는 최적의 linear boundary를 정해주는 것이 학습이다.

2. 수행 내용

2.1 Toy dataset 생성

“db_imgs[num_img_per_cls:,:,:mid_idx]=1” db_imgs는 이미지 data를 저장하는 3차원 numpy 배열이다. 첫 번째 차원은 이미지의 인덱스이며, 각 이미지를 나타낸다. 두 번째 차원은 이미지의 행을 나타낸다. 세 번째의 차원은 이미지의 열을 나타낸다. “num_img_per_cls:, 이 부분은 첫 번째 차원(이미지 인덱스)을 슬라이스해서 num_img_per_cls 이상의 이미지를 선택한다. 즉 class 1에 해당하는 이미지를 선택한다. “,:,” 이 부분은 두 번째 차원을 슬라이스한다. :는 해당 차원의 모든 요소를 선택한다. “,:mid_idx” 세 번째 차원을 슬라이스한다. mid_idx 이전의 열을 선택하며 이미지의 왼쪽 절반을 선택해 1을 대입해준다. 배열의 값이 1인 경우 흰색이 나타나고 배열의 값이 0인 경우 흑색이 나타난다. 즉 처음에 적은 코드의 경우 왼쪽 절반이 흰색으로 설정된다. 다만 출력값의 경우 랜덤 노이즈를 더해줘서 완전한 흰색과 흑색은 아니다.

db_data = db_imgs.reshape(num_imgs, -1) 3차원 배열을 1차원 벡터로 바꿔준다.

2.2 Toy dataset - training/test 분할

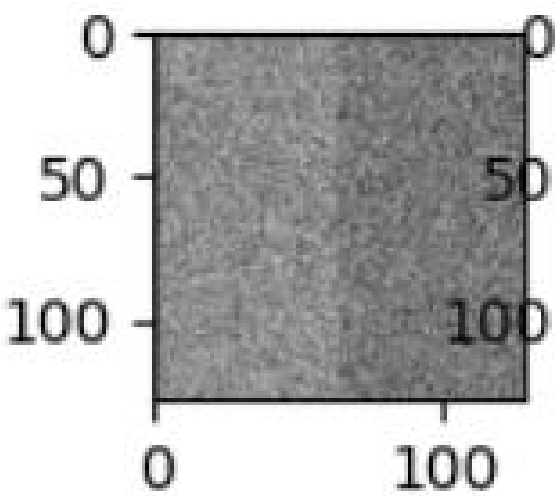
“tr_labels[:num_tr_data]=0” 처음 num_tr_data개의 원소를 0으로 설정한다. 첫 번째 class(Class 0)에 속하는 training data의 lable을 0으로 표시한 것이다.

“tr_labels[num_tr_data:2*num_tr_data]=1” num_tr_data 개의 원소를 1로 설정한다. 두 번째 class(Class 1)에 속하는 training data의 레이블을 1로 표시한 것이다.

“te_data[:num_te_data, :]=db_data[num_img_per_cls+idxs[num_tr_data:num_img_per_cls], :]” class 0에 속하는 test data를 할당한다. test data 중에서 처음 num_te_data개의 데이터를 선택하고, db_data 배열에서 해당 데이터의 인덱스(idxs[num_tr_data:num_img_per_cls])를 사용하여 가져와서 te_data에 할당해 준다.

“te_data[num_te_data:2 * num_te_data,:]= db_data[num_img_per_cls+idxs[num_tr_data:num_img_per_cls],:] class1에 속하는 test data를

할당한다. test data 중에서 num_te_data개의 데이터를 선택하고, db_data 배열에서 해당 데이터의 인덱스(num_img_per_cls + idxs[num_tr_data:num_img_per_cls])를 사용하여 가져와서 te_data에 할당한다.



생성한 Toy data 예시

2.3 Linear classifier 모델의 local gradient 연산 정의

def forward(self, data)의 함수 내부 코드

"self.data = data" 입력 data를 class 내부의 self.data 변수에 저장한다. 저장한 변수는 backpropagation 및 gradient 계산에 필요한 data로 사용된다.

"self.scores=np.dot(data, self.W) + self.b" 입력 data와 weight matrix(self.W)와 bias(self.b)를 사용하여 선형 연산을 수행하고, 그 결과를 self.scores에 저장한다. 이것은 각 class에 대한 score를 나타낸다.

def backward(self, upstream_grad)의 함수 내부 코드

"grad_W = np.dot(self.data.T, upstream_grad)" weight martirx W에 대한 local gradient인 grad_W를 계산한다. 이 gradient는 입력 data인 self.data의 transpose matrix와 upstream_grad의 행렬 곱으로 계산된다. upstream_grad는 backpropagation로부터 전달된 gradient이므로, weight matrix W에 대한 gradient를 계산하는 데 사용된다.(upstream_grad는 backpropagation로부터 전달된 gradient이다.)

"grad_b = np.sum(upstream_grad, axis=0) bias b에 대한 local gradient인 grad_b를 계산한다. 이 gradient는 upstream_grad의 열 방향으로 합산된 값이다. upstream_grad는 행렬이고, axis=0는 열 방향으로 더하라는 것을 의미한다.

2.4 Softmax loss 함수의 local gradient 연산 정의

def compute_probs(self, data): 의 함수 내부 코드

"exp_scores = np.exp(scores - np.max(scores, axis=1, keepdims=True))" softmax function을 계산하기 위해 각 score를 지수 함수로 변환한다. np.exp() 함수를 사용하여 score를 지수한다. 이 때 "np.max(scores, axis=1, keepdims=True)"를 사용하여 각 행의 최대 score를 빼주는 정규화 과정을 수행한다. 이렇게 함으로써 수치 안정성을 높이고, softmax function의 지수 계산에서 오버플로우를 방지한다.

"self.probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)" 지수화된 score를 사용하여 softmax function을 계산하고, 각 class에 속할 확률을 self.probs에 저장한다. soft amx function은 결국 지수화된 score를 각 data point에 대한 합으로 나누어 각 class에 속할 확률을 얻는 함수이다.

def compute_loss(self, data, labels)의 함수 내부 코드

"labels = np.array(labels)" classlable을 배열로 변환한다. labels는 주어진 data point에 대한 실제 class lable을 포함하는 배열이다.

"correct_class_probs = self.probs[np.arange(num_data), labels]:" 모델이 예측한 class 확률 중, 실제 class에 해당하는 확률을 선택한다. self.probs는 Softmax fucntion을 통해 얻은 각 클래스에 속할 확률을 담고 있는 배열이다. "np.arange(num_data)"는 0부터 num_data-1까지의 배열을 생성하며 이것은 data point의 인덱스를 나타낸다. labels 배열은 각 data point에 대한 실제 클래스 레이블을 포함하고 있다. 이 코드는 각 data point에 대한 실제 class에 대한 예측 확률을 correct_class_probs에 저장한다.

"neg_log_probs = -np.log(correct_class_probs)" 확률을 이용하여 음의 로그 확률을 계산한다. 이것은 cross entrophy loss의 일부이다. np.log 함수를 사용하여 각 클래스에 대한 음의 로그 확률을 구하고, -를 사용하여 음의 로그 확률을 양으로 변환한다.

"self.loss = np.sum(neg_log_probs)/num_data" 모든 data point에 대한 음의 로그 확률을 더한 후, data point 수로 나누어 평균 cross entrophy loss를 계산합니다. self.loss에 이 loss 값을 저장한다.

2.4.1 SoftMaxLoss 함수의 이론 정리

"grad_z = z - make_labels_onehot(self.labels,self.num_cls)" gradient를 계산하는 식이다. softmax 확률 z와 실제 class lable을 one-hot 인코딩으로 나타낸 벡터 간의 차이를 나타내며 class별 gradient가 계산된다. 이것은 cross-entropy-loss function에 대한 gradient와 일치한다.

cross-entropy loss function의 gradient는 각 class에 대한 softmax 확률과 실제 class의 label간의 차이로 계산되며 식은 다음과 같다.

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

cross-entrophy loss

여기서 i는 정답 class의 인덱스이다. y_i는 정답 class를 나타내며, f_{yi}는 정답 클래스에 대한 점수이다. f_j는 class j에 대한 점수를 나타낸다. softmax functoin을 통해 class 점수가 확률로 변환되어 로그 안에 들어간다.

처음 코드에서 z는 softmax함수로 얻은 class별 확률이다. "make_labels_onehot(self.labels, self.num_cls)" 는 실제 class lable (self.labels)을 one-hot encoding 형식으로 변환한 벡터이다. 이 벡터는 정답 class를 나타내며 정답 class에 대한 원소는 1이고 아닌 class에 대한 원소는 0이다.

결국 gradient 벡터는 각 class에 대한 확률 z와 정답 class를 나타내는 원-핫 벡터 간의 차이를 계산하며, 이것은 loss function을 최소화하는 방향을 가리킨다. 이 gradient를 사용하여 모델의 weight 및 bias를 업데이트하고, 모델을 training 하는 데 사용된다.

2.5 Gradient descent 알고리즘 구현

"indices = np.random.choice(num_data, batch_sz, replace=False)" num_data에서 무작위 인덱스를 batch_sz 개수만큼 선택한다. replace=False를 해줌으로써 이미 선택한 데이터는 다시 선택하지 않는다.

"batch_data = db_data[indices]" 전체 data에서 무작위로 데이터 선택한다.

"batch_labels = db_labels[indices]" 전체 label에서 무작위로 데이터 선택한다.

2.6 test data classification

"te_scores = model.forward(te_data)" 모델에 test data를 입력하여 예측 score를 계산해 준다.

"te_predictions = np.argmax(te_scores, axis=1)" 각 test data point에 대한 예측 값을 계산한다. np.argmax 함수를 사용하여 각 test data point에 대한 가장 높은 score를 갖는 class를 선택해준다.

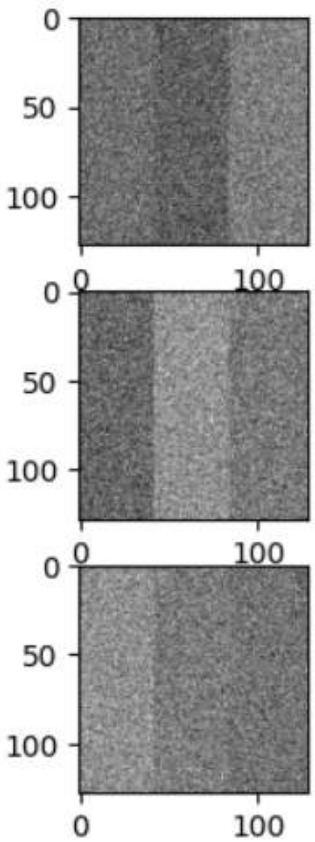
argmax 함수를 사용할 때 axis = 1는 각 행에 대한 최대 값의 인덱스를 반환한다. 즉 행렬에서 각 행에 대해 최대 값을 찾는 것을 나타낸다.

"correct_predictions = np.sum(te_predictions == te_labels)" 예측된 lable (te_predictions)과 실제 lable (te_labels)을 비교하여 정확하게 예측한 개수를 계산한다.

"Acc = correct_predictions / len(te_labels)" 정확도를 계산한다. 정확도는 정확하게 예측한 개수를 전체 test data point 수로 나눈 것이다.

2.7 더 어려운 3 class toy 데이터 생성하여 위 과정 반복 후 결과 정리

함수에 의해 생성된 가중치(Weight)의 시각화 이미지가 출력된다.



시각화된 weight

```
Classification accuracy= 1.0
```

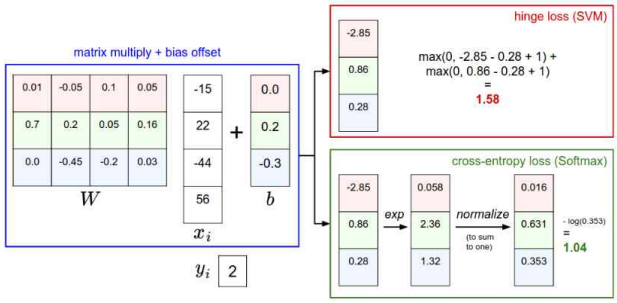
출력된 정확도

3. 실험 결과 및 분석

linear classifier를 이용해서 흑/백, 백/흑 각각 100개의 image, 총합 200개의 Dataset 중 40개를 training data로 학습을 시키고 160개를 test data로 이용해 정확도를 판별했고 그 결과 정확도가 100%이다. 2개의 class를 분류하는 코드를 바탕으로 백/흑/백, 흑/백/흑, 백/흑/흑 총 3개의 class를 만들어 분류하는 코드를 작성하였고 역시 정확도 100%를 달성했다.

우리가 사용한 분류 방식은 Softmax classifier이고 다른 방법으로는 SVM이 있다. SVM과 Softmax classifier 간의 차이점은 Softmax classifier는 주로 다중 class 분류에 사용되며 SVM은 주로 이진 분류에 사용된다. 또 다른 점으로는 vector f 내에서 score를 구하는 방식에 있다. SVM은 data point에 대한 score를 class score로 구한다. 이것은 각 class에 대한 score를 나타내며 이러한 점수는 data point가 해당 class에 얼마나 가까운지를 측정하는 것이다. 반면 Softmax classifier는 score를 각 class에 대한 로그 확률로 구한다. 각 class에 대한 확률을 나타내며 이 확률은 data point가 해당 class에 속할 확률을 나타내는 것이다. 아래 예시에서 SVM의 최종 loss는 1.58이고, Softmax classifier의 최종 loss는 1.04이다.

둘 중 어느 것이 좋다기보다 데이터의 특성과 상황을 고려해 적절히 사용한다.



Softmax와 SVM classifier 간의 차이

4. 결론

Python code를 이용해 data를 생성 후 data를 training data와 test data로 나눈다. 그 후 Linear classifier model의 local gradient 연산과 softmax loss 함수의 local gradient 연산을 정의해준다. 마지막으로 gradient descent 알고리즘으로 linear classifier를 학습하고 학습된 linear classifier의 정확도를 구해준다. 이 경우 data가 2개의 class로 분류되고 비교적 단순하기에 정확도는 100%이다.

참고문헌

[1] stanford CS231n 강의 노트, URL: https://cs231n.github.io/linear-classify/#softmax-classifier

[2] 이수찬, “인공지능융합공학 강의 03_Linear Classifier” 2023