Training a neural network

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

요 약

MNIST 데이터셋을 1개의 hidden layer를 가진 multi-layer perceptron으로 training을 시킨다. 데이터 전처리, Neural network 모델의 local gradient 연산 정의, softmax loss 함수의 local gradient 연산 정의, Gradient descent 알고리즘을 구현한다. 마지막으로 학습된 MLP Neural network를 이용해 테스트 데이터를 판별해 정확도를 측정한다.

1. 서론

주피터 노트북을 이용해 MNIST 데이터셋을 1개의 hidden layer를 가진 multi-layer perceptron으로 training을 시킨다. 이때 모든 연산 과정은 PyTorch 같은 별도의 머신러닝 라이브러리를 사용하지 않고 numpy만을 사용한다.

2. 수행 내용

2.1 package 불러오기 및 함수 선언

불러온 package는 numpy, matplotlib.pyplot, gzip, os와 같다. numpy는 배열 연산과 행렬 연산 등에 사용된다. matplotlib.pyplot은 그래프를 그리는 데 사용된다. gzip의 경우 파일을 압축하고 해제한다. os의 경우 디렉터리 생성, 파일 삭제 등을 할 수 있게 해준다.

선언한 utility function은 다음과 같다. “make_labels_onehot(labels, NMU_CLS)” 정수 label을 onehot 벡터로 변환한다. 벡터에서 정답 class만 1이고 나머지는 0이 된다. “VisualizeBatchData(batch_data, batch_sz)” 의 경우 batch_data를 시각화한다. “VisualizeWeights(W_arr, num_iter, num_cls)” 의 경우 weight가 들어가 있는 배열을 시각화한다.

2.2 데이터 불러오기

MNIST 데이터셋은 0부터 9 사이의 숫자를 필기한 영상 데이터셋이다. 28x28 크기를 가지며 숫자마다 같은 수의 영상을 가진다. 총 7만 장의 영상과 레이블들로 구성되며, 6만 장은 training 데이터셋과 만 장은 test 데이터셋이다. data/MNIST 폴더에서 MNIST 데이터셋을 로딩한다.

“load_mnist_data(data_dir)” 함수에서 “train_images_path = os.path.join(data_dir, 'train-images-idx3-ubyte.gz')” 가 있다. “os.path.join():” 은 경로를 구분해 주는 슬래시 또는 역 슬래시가 추가되어 파일 경로를 연결해 준다. “data_dir” 와 “train-images-idx3-ubyte.gz” 을 합쳐서 경로를 만든다. “data_dir:” 는 파일 경로인 data/MNIST/'를 저장한다.

“ with gzip.open(train_images_path, 'rb') as f: train_images = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1, 28, 28)” 에서 “with gzip.open(train_images_path, 'rb') as f:” 를 보면 “with” 는 파일을 자동으로 닫는다. gzip.open은 gzip 압축 파일을 열어준다. train_images_path 경로에 있는 gzip 압축 파일을 바이너리 읽기 모드 “'rb'” 로 연다. “as f” 는 열린 파일을 f라는 변수에 할당한다. “train_images = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1, 28, 28):” 은 “np.frombuffer(f.read(), np.uint8, offset=16)” 는 메모리에 로드된 데이터를 numpy 배열로 변환한다. “f.read()” 열린 gzip 파일의 내용을 읽는다. “np.uint8” 은 데이터 타입을 나타내며, unsigned integer 형태이다. “offset=16” 은 파일 시작에서 16바이트를 건너뛰고 데이터를 읽는다. MNIST 데이터셋에서

처음 16바이트는 헤더 파일이기 때문이다.
“reshape(-1, 28, 28)” 데이터를 numpy 배열로 변환한다.

“train_images, train_labels, test_images, test_labels = load_mnist_data(data_dir)” data_dir 경로에 있는 MNIST 데이터셋을 불러온다.

“num_imgs_tr = train_images.shape[0]” train_images 배열의 첫 번째 차원의 크기를 가져온다. 이것은 training images의 총개수를 나타낸다.

“img_sz = train_images.shape[1]: train_images” train_images 배열의 두 번째 차원의 크기를 가져온다. 이 값은 이미지의 한 변의 길이를 나타낸다.

“num_cls = 10” 클래스의 총개수는 10개라는 것을 나타낸다.

2.3 데이터 전처리

“num_view = 10” 10개의 이미지가 시각화된다.
“f, axarr = plt.subplots(1,num_view)” 는 plt.subplots 함수로 1행에 num_view 개수의 열을 가진 subplot 배열을 만든다. “for i in range(num_view): axarr[i].imshow(train_images[i,:,:],cmap='gray')” 0부터 num_view-1까지 배열에서 i번째 이미지를 gray로 표시하는 것을 반복한다.
“plt.show()” 생성된 이미지를 표시한다.

“print(train_labels[:num_view])” 처음 num_view개의 label을 출력한다.

“train_images.reshape(num_imgs_tr, -1)” train_images 배열을 training image의 총 개수인 num_imgs_tr개의 1차원 벡터로 변환한다. -1은 numpy가 크기를 자동으로 계산하게 한다.

“tr_data = tr_data / 255.0”, “te_data = te_data / 255.0” 정규화하지 않았을 때 정확도가 낮아 정규화를 진행해 주었다. 픽셀값이 8비트이기 때문에 최댓값이 255라는 것을 알고 있다. 따라서 최댓값으로 데이터를 나눠주어 데이터들이 0~1에 분포하게 한다.

“tr_mean = np.mean(tr_data)”, “tr_data = tr_data - tr_mean”, “te_data = te_data - tr_mean” 이 코드는 MNIST 데이터셋을 zero-centering 데이터 전처리를 수행한다. zero-centering이란 데이터의 평균을 구해서 모든 데이터를 평균만큼 빼줘 데이터의 평균이 0이 되게 하는 데이터 전처리를 의미한다.

2.4 Neural network 모델의 local gradient 연산 정의

MLPNeuralNetwork class는 Multi-Layer Perceptron 신경망을 나타낸다.

“__init__()” 은 신경망을 초기화할 때 필요한 매개변수들을 설정한다.

“forward(self, data)” 함수는 forward propagation 연산을 하며 마지막에 최종 점수를 출력한다. “self.data = data” 는 전달된 data를 self.data에 할당한다. 이때 data는 N*D차원이다.
“self.hidden_input = np.dot(data, self.W1) + self.b1” 이 코드에서 data와 self.W1(weight) 간의 행렬 곱을 하고 self.b1(bias)을 더한다. 이것은 hidden layer의 뉴런에 들어가는 값을 계산하며 결과는 self.hidden_input에 저장된다.
“self.hidden_output = np.maximum(0, self.hidden_input)” 에서 np.maximum 함수는 두 입력값 중 더 큰 값을 나타낸다. 즉 0과 self.hidden_input 중 더 큰 값을 self.hidden_output에 넣어주며 이때 모든 음수는 0으로 대체된다. 결국 이것은 ReLU 함수를 나타낸다. ReLU 함수를 사용함으로써 비교적 간단한 연산으로 layer를 깊게 쌓을 수 있게 해준다. “self.scores = np.dot(self.hidden_output, self.W2) + self.b2” 앞에서 처리한 것과 같은 과정을 진행한다.
“return self.scores” 최종적으로 score를 반환한다.

“backward(self, upstream_grad)” 는 backpropagation 연산을 하며 이 연산은 신경망의 예측값과 실제 값과의 차이인 loss의 gradient를 계산해 준다. “grad_W2 = np.dot(self.hidden_output.T, upstream_grad)” 는 출력층에서 hidden layer로 가는 w2에 대한 gradient를 계산 해준다. 여기서 upstream_grad는 upstream의 gradient이며 self.hidden_output는 hidden layer에서 ReLu함수의 출력값이다. 그리고 행렬 계산을 위해 transpose 해준다. “grad_b2 = np.sum(upstream_grad, axis=0)” 은 출력층에서의 bias는 upstream_grad의 합으로 나타나며 axis=0은

각 열의 값은 행을 따라 합치라는 뜻이다.
“hidden_grad = np.dot(upstream_grad, self.W2.T)” hidden_grad는 hidden 뉴런에 대한 gradient를 저장한다.
“hidden_grad[self.hidden_input <= 0] = 0” hidden layer의 input이 음수면 0을 출력하고 이것은 Relu함수의 미분값을 구현하는 데 사용한다. Relu함수는 음수의 경우 기울기가 0이고 양수면 기울기가 1이기 때문이다.

2.5 Softmax loss 함수의 local gradient 연산 정의

SoftMaxLoss class는 신경망의 loss와 gradient를 계산한다.

“compute_probs(self, data)” 는 score를 받아 softmax확률을 계산한다. “scores = self.model.forward(data)” 는 입력데이터에 대한 score를 계산한다. “scores = scores-np.max(scores, axis=1, keepdims=True)(최댓값을 빼주어 안정하게 해주는 idea는 ChatGPT를 활용하여 도출되었습니다. [1])” 는 각 데이터 포인트에 대해 최대 score를 빼줌으로써 수치상으로 안정하게 한다.
“exp_scores = np.exp(scores)” 이 것은 score에 지수 함수를 적용해 준다. “probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)” 지수 함수를 적용해 준 score를 모두 더한 후 나눠준다. 이러면 각 확률의 합이 1이 된다. “self.probs = probs” 계산 된 확률을 저장한다. “return self.prob” 저장 된 확률을 반환한다.

“compute_loss(self, data, labels)” 는 softmax를 기반으로 cross entrophy loss를 계산한다. “probs = self.compute_probs(data)” y compute_prob를 이용해 data의 확률을 계산한다.
“correct_log_probs = -np.log(probs[np.arange(data.shape[0]), labels] + epsilon)(이 코드는 ChatGPT를 활용하여 도출되었습니다. [2])” 은 cross entrophy loss를 계산해 준다. epsilon은 로그를 계산할 때 0을 대입해 주는 걸 방지하기 위해 아주 작은 값을 넣어준다. “loss = np.sum(correct_log_probs) / data.shape[0]” 은 계산된 확률을 데이터 개수로 나누어 평균 loss를 구한다.

2.5.1 compute_grad 함수

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

softmax수식

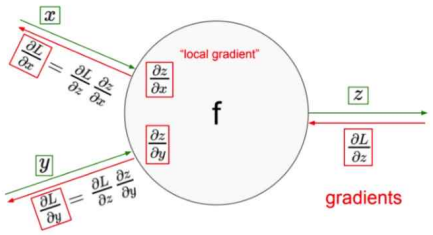
softmax 함수는 모든 확률의 합은 1로 만들어 주며 정규화한다. class 간의 확률을 상대적으로 나타내주기 때문에 가장 높은 확률을 가진 class가 예측값이 된다.

$$-\sum_k t_k^{(n)} \log o_k^{(n)}$$

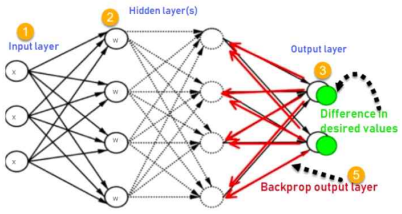
cross-entrophy loss

compute_grad 함수는 loss 함수의 기울기를 계산한다. self.probs는 softmax 함수가 나타내는 예측 확률을 의미하며 grad_z는 실제 label을 정답 class는 1이고 나머지는 0인 one-hot 레이블로 변환해서 예측 확률과의 차이를 기울기로 나타낸다. 그리고 기울기는 loss 함수에 대한 기울기이다.

“return self.model.backward(grad_z)” 는 self.model.backward(grad_z)를 사용하여 계산된 기울기를 backpropagation 한다. 이 과정에서 신경망의 weight는 loss를 줄이는 방향으로 업데이트된다. 이때 아래 그림과 같이 미분의 연쇄법칙을 이용해 gradient를 계산한다.



local gradient 계산



backprop 연산과정

2.6 Gradient descent 알고리즘 구현

“get_mini_batch()” 는 data를 한 번에 처리하는 대신 랜덤으로 추출해 처리한다. “batch_indices = np.random.choice(num_data, batch_sz)” 는 data의 총수에서 batch_sz만큼의 index를 선택한다. “batch_data = db_data[batch_indices]” , “batch_labels = db_labels[batch_indices]” 는 선택한 index에 해당하는 데이터를 추출해 각각 batch_data와 batch_labels에 할당한다.

2.7 학습된 모델을 이용한 test data classification

“te_scores = model.forward(te_data)” 은 test data 셋을 forward propagation 연산을 해서 각 class에 대한 score를 계산한다. “exp_scores = np.exp(te_scores - np.max(te_scores, axis=1, keepdims=True))” 안정성을 위해서 max score를 빼주고 softmax함수를 적용해 준다. “probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)” 지수 score를 지수 score의 합으로 나눠주어서 정규화를 해준다. axis=1은 행에 대한 연산을 의미한다. “predictions = np.argmax(probs, axis=1)” 확률이 가장 높은 index를 찾는다. “Acc = np.mean(predictions == test_labels)” 예측한 class와 실제 label을 비교하여 정확도를 계산한다.

3. 실험 결과 및 분석

learning rate의 경우 0.001이 적절하다고 생각해 lr을 제외하고 하이퍼파라미터들을 수정하면서 최적의 하이퍼파라미터를 찾았다.

하이퍼파라미터에 따른 정확도

batch_sz	num_iter	num_hid_neurons	학습시간	ACC
8	100	128	0:0.7	0.0982
128	10000	512	3:57	0.9492
128	10000	256	1:44	0.9365
256	10000	256	2:21	0.9428
256	5000	256	1:39	0.9352
512	10000	256	3:51	0.5874
256	10000	256	2:11	0.9634
512	10000	512	8:37	0.5522
256	10000	512	7:19	0.9455

표를 분석해 보면 batch size가 512인 경우 over fitting 되어서 정확도가 50%대이다. 또 hidden layer의 neurons 수가 batch size보다 학습 시간에 영향을 많이 미친다. 가장 성능이 좋은 하이퍼파라미터 값은 batch size : 256, num_iter : 10000, hidden_neurons : 256일 때이다.

4. 결론

MNIST dataset을 1개의 hidden layer를 가진 multi layer perceptron으로 학습시켰고 최고 정확도가 96% 내외를 달성했다. 1개의 hidden layer를 사용했지만 데이터셋이 비교적 단순하고 뉴런의 수가 충분히 많아서 정확도가 높게 학습하였다.

참고문헌

[1] OpenAI, ChatGPT (ver. GPT 3.5), 2023.11.10. 접속
[2] OpenAI, ChatGPT (ver. GPT 3.5), 2023.11.10. 접속
[3] 이수찬, “인공지능융합공학 강의 04_Linear Classifier” 2023
[4] 이수찬, “인공지능융합공학 강의 07_Neural Network” 2023
[5] 이수찬, “인공지능융합공학 강의 08_Backpropagation” 2023