Python+NumPy를 이용한 CNN의 지도학습 기법 구현

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

요 약

python과 numpy만을 이용해서 convolutional neural network를 구현한다. 이 때 toy data set을 만들고 학습데이터와 테스트데이터로 분리 후 convolutional neural network 연산을 정의한다. data 전처리 및 softmax loss함수의 local gradient 정의, gardeint descent 알고리즘을 구현한다. 마지막으로 학습한 모델을 이용해서 test data를 classification해준다.

1. 서론

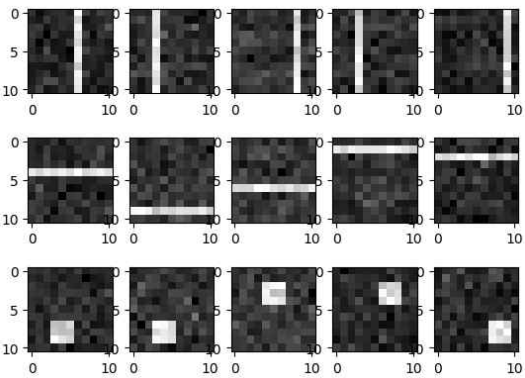
수업시간에 이론으로 배운 CNN을 python과 numpy를 이용해 구현한다. 또한 이제까지 과제에서 구현한 코드를 활용한다.

2. 수행 내용

무작위로 수직선, 수평선, 3x3이 임의로 있는 이미지를 이용해 모델을 학습하고 정확도 100%를 달성한다.

2.1 Toy dataset 생성

“img_sz = 11, num_img_per_cls = 100 num_cls = 3, num_imgs = num_img_per_cls*num_cls ” 이미지 크기가 11x11픽셀이다. class당 이미지의 수는 100개이다. class당 100개의 이미지가 있고 총 3개의 class가 있으므로 총 이미지 수는 300개이다. “db_imgs = np.zeros((num_imgs, img_sz, img_sz))” 은 이미지를 저장할 배열을 만들고 배열을 0으로 초기화한다. “ri_v = np.random.randint(img_sz-2, size=num_img_per_cls)” 무작위로 수직선을 그려준다. 0부터 image size-2 사이 중 무작위로 정수를 생성하며 size=num_img_per_cls class별로 이미지를 생성하는 걸 의미한다. “ri_h = np.random.randint(img_sz-2, size=num_img_per_cls)” 수평선을 무작위로 그려준다. “re_sz = 3, ri_rp = np.random.randint(img_sz-4, size=(num_img_per_cls,2))” 사각형의 크기를 3x3 픽셀로 정하고 사각형의 위치를 무작위로 그려준다. “for i in range(num_img_per_cls):” class별 이미지 수 만큼 반복한다. “db_imgs[i,:,ri_v[i]+1] = 1” 클래스 0의 이미지에 수직선을 그린다. “db_imgs[num_img_per_cls+i,ri_h[i]+1,:] = 1” 클래스 1의 이미지에 수평선을 그린다. “db_imgs[num_img_per_cls*2+i,ri_rp[i,0]+2:ri_rp[i,0]+2+re_sz,ri_rp[i,1]+2:ri_rp[i,1]+2+re_sz] = 1” 클래스 2의 이미지에 사각형을 그린다. “noise_w = 0.1”,“db_imgs = db_imgs + noise_w*np.random.randn(num_cls*num_img_per_cls, img_sz, img_sz)” 각 이미지에 0.1 weight의 noise를 추가해준다. “f, axarr = plt.subplots(3,5) for i in range(5):axarr[0,i].imshow(db_imgs[i,:,:],cmap='gray')axarr[1,i].imshow(db_imgs[num_img_per_cls+i,:,:],cmap='gray')axarr[2,i].imshow(db_imgs[num_img_per_cls*2+i,:,:],cmap='gray') plt.show()” 3x5 서브플롯을 생성하고 5개의 이미지를 class별로 시각화 해준다. 이 때 행별로 같은 class의 이미지가 출력된다.



위에 코드로 인해 출력된 이미지

"db_labels = np.zeros((num_cls*num_img_per_cls), dtype='int8')" 전체 이미지 수 크기의 배열을 생성하고 0으로 초기화한다. "db_labels[:num_img_per_cls] = 0" 처음부터 num_img_per_cls까지 0을 할당한다. "db_labels[num_img_per_cls:2*num_img_per_cls] = 1" num_img_per_cls부터 2*num_img_per_cls까지 1을 할당한다. "db_labels[2*num_img_per_cls:] = 2" 2*num_img_per_cls부터 끝까지 2를 할당한다. "db_data = db_imgs.copy()" 원래 이미지 데이터셋의 복사본을 만들어 저장한다. "data_mean = np.mean(db_data)" data 배열의 평균 값을 계산한다. "db_data = db_data - data_mean" 모든 이미지에서 평균값을 빼주어 이미지 데이터의 중심을 0에 맞추는 정규화를 해준다.

2.2 Toy dataset - training/test 분할

"idxs = np.arange(num_img_per_cls) np.random.shuffle(idxs) tr_ratio = 0.8" 0부터 num_img_per_cls-1까지 숫자를 포함하는 배열을 생성한다. np.random.shuffle은 idx배열을 무작위로 섞는다. 80%를 tr_data로 사용한다. "num_tr_data = int(num_img_per_cls * tr_ratio)" 학습 데이터 셋에 사용될 class의 이미지 수를 계산한다. "tr_data = np.zeros((num_tr_data * num_cls, img_sz, img_sz))" 학습 데이터를 저장할 배열을 만들어 준다. "tr_data[:num_tr_data, :] = db_data[idxs[:num_tr_data], :] tr_data[num_tr_data:2 * num_tr_data, :] = db_data[num_img_per_cls + idxs[:num_tr_data], :]tr_data[2 * num_tr_data:3 * num_tr_data, :] = db_data[2 * num_img_per_cls + idxs[:num_tr_data], :] tr_labels = np.zeros((num_tr_data * num_cls), dtype='int8') tr_labels[:num_tr_data] = 0 # 클래스 0 레이블 tr_labels[num_tr_data:2 * num_tr_data] = 1 # 클래스 1 레이블 tr_labels[2 * num_tr_data:3 * num_tr_data] = 2 # 클래스 2 레이블" 3개의 class에 대해 학습 데이터를 추출하고 tr_data 배열에 할당한다. 그 후 tr_data의 lable을 저장할 배열을 생성하고 label을 할당해준다. 이 과정을 test data에 대해서도 똑같이 해준다.

2.3 Convolutional neural network 모델의 local gradient 연산 정의

"self.K_outdim = (self.img_sz - k_sz + 1) * (self.img_sz - k_sz + 1) * k_num" 에서 self.K_outdim은 convoultion 층을 지난 후 output data의 전체 차원 수를 나타내는 변수이다. k_sz는 커널의 크기이며 k_num은 커널의 수이다. (self.img_sz - k_sz + 1)은 하나의 커널이 이미지를 통과하면서 생성되는 것의 크기를 나타낸다. 결국 self.K_outdimd은 생성된 feature들의 총 수를 나타낸다. "data = np.squeeze(data, axis=1)" 입력 데이터에서 지정된 차원이 1인 경우 제거한다. 여기서는 axis=1 차원을 제거한다. "self.data = data" data를 self.data에 저장한다. "self.batch_size = data.shape[0]" 배치 크기를 데이터의 첫 번째 차원으로 설정한다. "self.conv_output = np.zeros((self.batch_size, self.kernel_num, self.img_sz - self.kernel_sz + 1, self.img_sz - self.kernel_sz + 1))" 컨볼루션 연산의 결과를 저장할 배열을 초기화한다. "for b in range(self.batch_size):for k in range(self.kernel_num): kernel = self.K1[:, :, k] for i in range(self.img_sz - self.kernel_sz + 1):for j in range(self.img_sz - self.kernel_sz + 1):self.conv_output[b, k, i, j] =

np.sum(data[b, i:i+self.kernel_sz, j:j+self.kernel_sz] * kernel) + self.b1[k]" 각 data에 b, k, i, j에 대해 컨볼루션 연산을 한다. ([1] for문은 gpt를 이용해 도출하였습니다.) "relu_output = np.maximum(0, self.conv_output)" ReLU 활성화 함수를 사용한다. "self.fc_input = relu_output.reshape(self.batch_size, -1)" ReLU 적용 후의 출력을 1차원 벡터로 바꾼다. "self.scores = np.dot(self.fc_input, self.W2) + self.b2" data와 self.W2의 행렬 곱을 하고 self.b2을 더해 score를 계산한다. "grad_W2 = np.dot(self.fc_input.T, upstream_grad)" gradient를 계산한다. 이 gradient는 현재 레이어의 입력(self.fc_input)과의 행렬 곱을 통해 계산된다. "grad_b2 = np.sum(upstream_grad, axis=0)" bias에 대한 gradient를 계산한다. 이는 upstream gradient의 합으로 구한다. "grad_relu = np.dot(upstream_grad, self.W2.T)" ReLU 함수를 지난 후의 gradient를 계산한다. 이는 upstream gradient와 self.W2의 transpose 행렬과의 곱으로 계산한다. "grad_relu[self.fc_input <= 0] = 0" ReLU 함수의 입력이 0 이하면 gradient를 0으로 둔다. "grad_K1 = np.zeros_like(self.K1)" 커널의 gradient를 저장할 배열을 초기화한다. "grad_b1 = np.zeros_like(self.b1)" bias의 gradient를 저장할 배열을 초기화한다. "grad_conv_output = grad_relu.reshape(self.conv_output.shape)" gradient를 현재 레이어의 출력 차원에 맞게 바꿔준다. "for b in range(self.batch_size): for k in range(self.kernel_num):for i in range(self.img_sz - self.kernel_sz + 1):for j in range(self.img_sz - self.kernel_sz + 1): grad_K1[:, :, k] += self.data[b, i:i+self.kernel_sz, j:j+self.kernel_sz] * grad_conv_output[b, k, i, j] grad_b1[k] += grad_conv_output[b, k, i, j]" ([2] for문은 gpt를 이용해 도출하였습니다.)b, k, i, j에 대해 gradient를 계산한다. 컨볼루션 연산의 backpropagation으로, 입력 data와 컨볼루션 출력 gradient의 요소별 곱셈을 통해 kernal과 bias의 gradient를 업데이트합니다. "grad = {'K1': grad_K1, 'b1': grad_b1, 'W2': grad_W2, 'b2': grad_b2}" 계산된 gradient를 dictionary에 저장한다.

2.4 Softmax loss 함수 local gradient연산정의

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.

2.4.1 compute_grad 함수

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

softmax수식

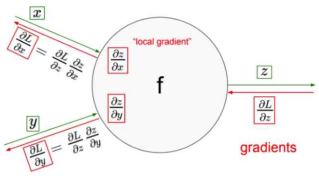
softmax 함수는 모든 확률의 합은 1로 만들어 주며 정규화한다. class 간의 확률을 상대적으로 나타내주기 때문에 가장 높은 확률을 가진 class가 예측값이 된다.

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

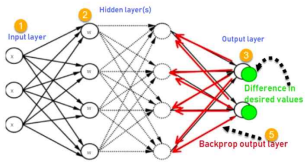
cross-entrophy loss

compute_grad 함수는 loss 함수의 gradient를 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 gradient로 나타낸다. 그리고 gradient는 loss 함수에 대한 gradient이다.

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



local gradient 계산



backprop 연산과정

2.5 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.

2.6학습된 모델을 이용한 test data classification

"if te_data.ndim == 3: te_data = te_data.reshape(te_data.shape[0], 1, te_data.shape[1], te_data.shape[2])" te_data의 차원이 3이라면 te_data에서 두 번째 차원에 흑백을 뜻하는 채널인 1을 추가한다. "te_scores = model.forward(te_data)" 은 test data 셋을 forward propagation 연산을 해서 각 class에 대한 score를 계산한다. "predicted_labels = np.argmax(te_scores, axis=1)" 은 te_scores에서 가장 높은 점수를 가진 index를 찾아서 저장해준다. axis=1은 각 행에서 가장 높은 값을 찾는다는 것을 의미한다. "correct_predictions = np.sum(predicted_labels == te_labels)" 은 predicted_labels과 te_labels이 일치할 경우의 수를 센다. "total_predictions = te_labels.shape[0]" 은 전체 예측 수는 테스트 레이블의 총 개수이다. "Acc = correct_predictions / total_predictions" 정확도는 맞게 예측한 값 나누기 전체 예측 수를 해준다.

3. 실험 결과 및 분석

dataset이 비교적 작고 단순하기에 하이퍼파라미터를 lr = 0.001, batch_sz = 128, num_iter = 200, kernel_sz = 5, kernel_num = 16로 했다. 처음에 학습횟수를 1000으로 했다가 학습시간이 오래걸리는 것을 발견하고 batch_sz와 num_iter를 작은 값부터 서서히 늘려 짧은 시간으로도 정확도 100을 달성하게 했다.

4. 결론

CNN을 이용해 toy data set을 학습시켰고 정확도 100%를 달성하였다.

참고문헌

[1] OpenAI, ChatGPT (ver. GPT 3.5), 2023.11.23. 접속
[2] OpenAI, ChatGPT (ver. GPT 3.5), 2023.11.23. 접속
[3] 이수찬, "인공지능융합공학 강의 04_Linear Classifier" 2023
[4] 이수찬, "인공지능융합공학 강의 07_Neural Network" 2023
[5] 이수찬, "인공지능융합공학 강의 08_Backpropagation" 2023