
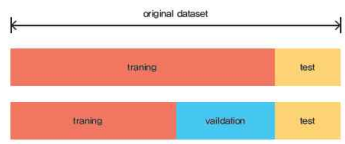
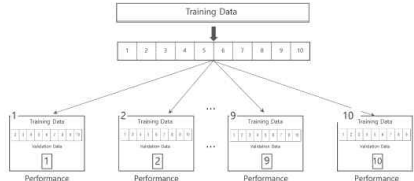
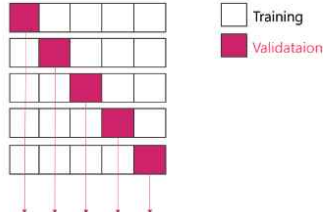
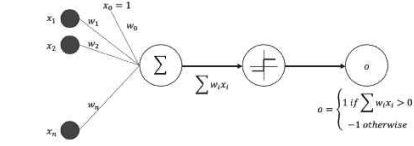
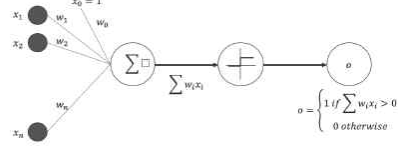
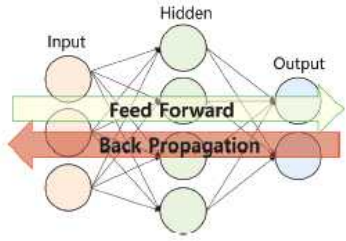
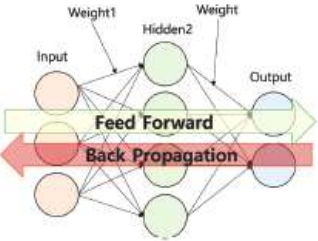


파이썬 딥러닝 파이토치 정오표(초판 1쇄 발행 2020년 10월 8일)

페이지	책	수정 내용
11 4라인	y축은 Top1	y축은 <b>Top-1</b>
21 코드라인3	# tensor([[[[1., 2.], # [3., 4.]], # [[5., 6.], # [7., 8.]])	# tensor([[[[10., 12.], # [14., 16.]], # [[18., 20.], # [22., 24.]])
23 코드라인2	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
25 9라인	한 결과는 (1, 100)이며	한 결과는 ( <b>64</b> , 100)이며
25 코드라인2	for t in range(1, 501)	for t in range(1, 501):
25 아래 5라인	t 값이 1부터 501까지	t 값이 1부터 <b>500</b> 까지
26 12라인	즉, y_pred-y.pow(2)는	즉, (y_pred-y).pow(2)는
41 수식	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
48	 [그림 2-17] 데이터의 분할	 [그림 2-17] 데이터의 분할
49	 [그림 2-18] Cross Validation의 개념	 [그림 2-18] Cross Validation의 개념
50	 [그림 2-19] 퍼셉트론	 [그림 2-19] 퍼셉트론
50 아래 9라인	이상이 아니면 -1을 출력	이상이 아니면 <b>0</b> 을 출력
50	$w_i \leftarrow w_i + \Delta w_i$ 여기서 - $\Delta w_i = \eta(t - o)x_i$ - $t$ = 실제 값 - $o$ = 예측 값	$w_i \leftarrow w_i + \Delta w_i$ Where - $\Delta w_i = \eta(t - o)x_i$ - $t$ = 실제 값 - $o$ = 예측 값 - $\eta$ = 학습률

52	 <p>[그림 2-23] MLP의 학습</p>	 <p>[그림 2-23] MLP의 학습</p>
53	<p>Feed Forward</p> <p>Input -&gt; Hidden -&gt; Output -&gt; Error -&gt; Hidden</p> <p>Back Propagation</p> <p>[그림 2-24] MLP의 학습 순서</p>	<p>Feed Forward</p> <p>Input -&gt; Hidden -&gt; Output -&gt; Error -&gt; Hidden</p> <p>Back Propagation</p> <p>[그림 2-24] MLP의 학습 순서</p>
55 아래 6라인	실제 True 값을 $t_j$ 로 표현하고	실제 True 값을 $t_j$ 로 표현하고
56	$\therefore \frac{\partial \epsilon_k}{\partial w_{hj}^k} = \frac{\partial \epsilon_k}{\partial \sigma(y_j^k)} \frac{\partial \sigma(y_j^k)}{\partial y_j^k} \frac{\partial y_j^k}{\partial w_{hj}^k}$ $-(t_j^k - \sigma(y_j^k)) = e_j^k \quad \sigma(y_j^k) (1 - \sigma(y_j^k)) \quad \sigma(z_h^k)$ $\frac{\partial \epsilon_k}{\partial w_{hj}^k} = -e_j^k \sigma(y_j^k)' \sigma(z_h^k)$	$\therefore \frac{\partial \epsilon_k}{\partial w_{hj}^k} = \frac{\partial \epsilon_k}{\partial \sigma(y_j^k)} \frac{\partial \sigma(y_j^k)}{\partial y_j^k} \frac{\partial y_j^k}{\partial w_{hj}^k}$ $-(t_j^k - \sigma(y_j^k)) = -e_j^k \quad \sigma(y_j^k) (1 - \sigma(y_j^k)) \quad \sigma(z_h^k)$ <p>위 식을 정리하면 다음과 같이 정리할 수 있습니다.</p> $\frac{\partial \epsilon_k}{\partial w_{hj}^k} = -e_j^k \sigma(y_j^k)' \sigma(z_h^k)$
60 코드라인2	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
63 코드라인4	for i in range(10)	for i in range(10):
63 아래코드	class Net(nn.Module) # (1) def __init__(self) # (2) def forward(self, x) # (7)	class Net(nn.Module): # (1) def __init__(self): # (2) def forward(self, x): # (7)
65 아래7라인	(1) '6. MLP 모델 설계하기'의 (6)에서	(1) '6. MLP 모델 설계하기'의 (1)에서
66 코드	def train(model, train_loader, optimizer, log_interval)	def train(model, train_loader, optimizer, log_interval):
67 코드	for batch_idx, (image, label) in enumerate(train_loader) # (2) def evaluate(model, test_loader)	for batch_idx, (image, label) in enumerate(train_loader): # (2) def evaluate(model, test_loader):
68 코드라인2	with torch.no_grad( ) # (4)	with torch.no_grad( ) : # (4)
69 아래코드 아래라인5	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):
70 코드5번 라인3	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
70 코드6번 라인1,2,7	for i in range(10)	for i in range(10):
71 코드8번 라인1,3	class Net(nn.Module) def __init__(self) def forward(self, x)	class Net(nn.Module): def __init__(self): def forward(self, x):
71 코드9번 라인1,5	def train(model, train_loader, optimizer, log_interval) for batch_idx, (image, label) in enumerate(train_loader)	def train(model, train_loader, optimizer, log_interval): for batch_idx, (image, label) in enumerate(train_loader):
	def evaluate(model, test_loader) with torch.no_grad( )	def evaluate(model, test_loader): with torch.no_grad( ):

71 코드10번 라인1	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):																												
73 13라인	[그림 2-34]는 Hidden Layer가	[그림 2-35]는 Hidden Layer가																												
74	<div> <div>표 2-1 분류 모형(Classification Model)에서 사용하는 성능 지표</div> <table> <tr> <th rowspan="2">실제 클래스</th><th colspan="2">예측한 클래스</th></tr> <tr> <th>정상</th><th>불량</th></tr> <tr> <th>클래스=(정상, 불량)</th><td></td><td></td></tr> <tr> <td>정상</td><td>TN(True Negative)</td><td>FP(False Positive)</td></tr> <tr> <td>불량</td><td>FN(False Negative)</td><td>TP(True Positive)</td></tr> </table> </div>	실제 클래스	예측한 클래스		정상	불량	클래스=(정상, 불량)			정상	TN(True Negative)	FP(False Positive)	불량	FN(False Negative)	TP(True Positive)	<div> <div>표 2-1 분류 모형(Classification Model)에서 사용하는 성능 지표</div> <table> <tr> <th rowspan="2">실제 클래스</th><th colspan="2">예측한 클래스</th></tr> <tr> <th>정상</th><th>불량</th></tr> <tr> <th>클래스=(정상, 불량)</th><td></td><td></td></tr> <tr> <td>정상</td><td>TP(True Positive)</td><td>FN(False Negative)</td></tr> <tr> <td>불량</td><td>FP(False Positive)</td><td>TN(True Negative)</td></tr> </table> </div>	실제 클래스	예측한 클래스		정상	불량	클래스=(정상, 불량)			정상	TP(True Positive)	FN(False Negative)	불량	FP(False Positive)	TN(True Negative)
실제 클래스	예측한 클래스																													
	정상	불량																												
클래스=(정상, 불량)																														
정상	TN(True Negative)	FP(False Positive)																												
불량	FN(False Negative)	TP(True Positive)																												
실제 클래스	예측한 클래스																													
	정상	불량																												
클래스=(정상, 불량)																														
정상	TP(True Positive)	FN(False Negative)																												
불량	FP(False Positive)	TN(True Negative)																												
75	$\text{정밀도(Precision)} = \frac{\text{올게 분류된 불량 데이터의 수}}{\text{불량으로 예측한 데이터}} = \frac{TP}{FP+TP}$ $\text{재현율(Recall)} = \frac{\text{올게 분류된 불량 데이터의 수}}{\text{실제 불량 데이터의 수}} = \frac{TP}{FN+TP}$ $\text{특이도(Specificity)} = \frac{\text{올게 분류된 정상 데이터의 수}}{\text{실제 정상 데이터의 수}} = \frac{TN}{TN+FP}$	$\text{정밀도(Precision)} = \frac{\text{정상으로 올바르게 예측된 데이터의 수}}{\text{정상으로 예측된 데이터의 수}} = \frac{TP}{FP+TP}$ $\text{재현율(Recall)} = \frac{\text{정상으로 올바르게 예측된 데이터의 수}}{\text{실제로 정상인 데이터의 수}} = \frac{TP}{FN+TP}$ $\text{특이도(Specificity)} = \frac{\text{불량으로 예측된 데이터의 수}}{\text{실제 불량인 데이터의 수}} = \frac{TN}{TN+FP}$																												
78 아래6라인	첫째, 신경망의 단점으로 지적돼왔던 과적합과 Gradient Vanishing을 완화시킬 수 있는 알고리즘이 발전한 것과 타 알고리즘 대비 학습 시간이 매우 오래 걸리는 문제가 있었는데 Graphics Processing Unit(GPU)을 신경망의 연산에 사용할 수 있게 되면서 이를 해결하게 된 것입니다.	첫 번째로, 신경망의 단점으로 지적되어 왔던 과적합과 Gradient Vanishing을 완화시킬 수 있는 알고리즘이 효과를 보였다는 점입니다. 두 번째로, 신경망은 타 알고리즘 대비 학습 시간이 매우 오래 걸리는 문제가 있었는데 Graphics Processing Unit(GPU)을 신경망의 연산에 사용할 수 있게 되면서 학습 속도를 높일 수 있게 되었다는 점입니다.																												
82 코드라인 2,3,8	<pre>class Net(nn.Module)     def __init__(self)      def forward(self, x)</pre>	<pre>class Net(nn.Module):     def __init__(self):      def forward(self, x):</pre>																												
82 아래 5라인	이 예제에서는 30%의 노드들은	이 예제에서는 50%의 노드들은																												
83 코드라인 2,3,9	<pre>class Net(nn.Module)     def __init__(self)      def forward(self, x)</pre>	<pre>class Net(nn.Module):     def __init__(self):      def forward(self, x):</pre>																												
83 아래 1라인	Validation Accuracy를 결과를	Test Accuracy를 결과를 비교한																												
82 코드라인 2,3,9	<pre>class Net(nn.Module)     def __init__(self)      def forward(self, x)</pre>	<pre>class Net(nn.Module):     def __init__(self):      def forward(self, x):</pre>																												
89 수식아래	[그림 3-7]을 보면	[그림 3-9]를 보면																												
90 6라인	코드에 따라 activation functoin 이전에	코드에 따라 Activation Fuction 이전에																												
90 코드라인 2,3,9	<pre>class Net(nn.Module)     def __init__(self)      def forward(self, x)</pre>	<pre>class Net(nn.Module):     def __init__(self):      def forward(self, x):</pre>																												
91 3라인	512차원으로	256차원으로																												
91 코드위3라인	Validation Loss, Validation Accuracy	Test Loss는 감소하며, Test Accuracy 기법으로 Xavier Initialization, LeCun Initialization 그리고 He Initialization 등이 있습니다.																												
92 12라인	기법을 간략하게 소개하겠습니다.																													

92	<p>• <b>LeCun Initialization</b>: LeCun이라는 Convolutional Neural 네트워크의 창시자의 이름에서 따온 기법으로, LeCun Normal Initialization과 LeCun Uniform Initialization이 있습니다. 각각 초기 분포가 다음과 같은 분포를 따르도록 weight를 초기화하는 것입니다.</p> <p>~ LeCun Normal Initialization</p> $W \sim N(0, Var(W))$ $Var(W) = \frac{1}{n_{in}}$ <p>여기서 <math>n_{in}</math>: 이전 Layer의 노드 수</p> <p>~ LeCun Uniform Initialization</p> <p>• <b>He Initialization</b>: Xavier Initialization은 ReLU 함수를 사용할 때 비효율적이라는 것을 보이는 데, 이를 보완한 초기화 기법이 He Initialization입니다.</p> $W \sim U(-\sqrt{\frac{1}{n_{in}}}, +\sqrt{\frac{1}{n_{in}}})$	<p>• <b>LeCun Initialization</b>: LeCun이라는 CNN 창시자의 이름에서 따온 기법으로, LeCun Normal Initialization과 LeCun Uniform Initialization이 있습니다. 각각 초기 분포가 다음과 같은 분포를 따르도록 weight를 초기화하는 것입니다.</p> <p>~ LeCun Normal Initialization</p> $W \sim N(0, Var(W))$ $Var(W) = \frac{1}{n_{in}}$ <p>여기서 <math>n_{in}</math>: 이전 Layer의 노드 수</p> <p>~ LeCun Uniform Initialization</p> $W \sim U(-\sqrt{\frac{1}{n_{in}}}, +\sqrt{\frac{1}{n_{in}}})$ <p>• <b>Xavier Initialization</b>: 이전 Layer의 노드 수에 따라 가중치를 결정짓는 LeCun Initialization과 비슷하게 이전 Layer의 노드 수와 다음 Layer의 노드 수에 따라 가중치를 결정 짓습니다.</p> <p>~ Xavier Normal Initialization</p> $W \sim N(0, Var(W))$ $Var(W) = \frac{1}{n_{in} + n_{out}}$ <p><math>n_{in}</math>: 이전 Layer의 노드 수, <math>n_{out}</math>: 다음 Layer의 노드 수</p> <p>~ Xavier Uniform Initialization</p> $W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, +\sqrt{\frac{6}{n_{in} + n_{out}}})$ <p>• <b>He Initialization</b>: Xavier Initialization은 ReLU 함수를 사용할 때 비효율적이라는 것을 보이는 데, 이를 보완한 초기화 기법이 He Initialization입니다.</p>
93 코드	def weight_init(m) # (2) if isinstance(m, nn.Linear) # (3)	def weight_init(m): # (2) if isinstance(m, nn.Linear): # (3)
95	$\theta = \theta - \eta \nabla_{\theta} J(\theta)$	$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} J(\theta)$
95	$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta)$ $\theta = \theta - v_t$	$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta)$ $\theta_{t+1} \leftarrow \theta_t - v_t$
96	$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$ $\theta = \theta - v_t$	$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$ $\theta_{t+1} = \theta_t - v_t$
96	$G_t = G_{t-1} - (\nabla_{\theta} J(\theta_t))^2$ $\theta = \theta - v_t$	$G_t = G_{t-1} - (\nabla_{\theta} J(\theta_t))^2$ $\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_{\theta} J(\theta_t)$
96 7라인	진행될수록 부분이 RMSProp는 G가	진행될수록 $G_{-t}$ 부분이 RMSProp는 $G_{-t}$ 가
96	$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$ $\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \nabla_{\theta} J(\theta_t)$	$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$ $\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G + \epsilon}} \nabla_{\theta} J(\theta_t)$
96	$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$ $\Delta_{\theta} = \frac{\sqrt{s + \epsilon}}{\sqrt{G + \epsilon}} \nabla_{\theta} J(\theta_t)$ $\theta = \theta - \Delta_{\theta}$ $s = \gamma s + (1 - \gamma) \Delta_{\theta}^2$	$G = \gamma G + (1 - \gamma)(\nabla_{\theta} J(\theta_t))^2$ $\Delta_{\theta} = \frac{\sqrt{s + \epsilon}}{\sqrt{G + \epsilon}} \nabla_{\theta} J(\theta_t)$ $\theta_{t+1} \leftarrow \theta_t - \Delta_{\theta}$ $s = \gamma s + (1 - \gamma) \Delta_{\theta}^2$
103 코드라인2	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
104 아래 3, 5 라인	MNIST	FashionMNIST
105 4라인	해당 데이터를 인터넷상에서 다운로드해 이용할 것인지를 지정합니다.	FashionMNIST 데이터셋은 의류, 가방, 신발 등 10가지 종류로 구성된 이미지 데 이터셋입니다.
105 5라인	■ transform	■ transform
105 12라인	다운로드한 MNIST 데이터셋을	다운로드한 FashionMNIST 데이터셋을
106 코드라인4	for i in range(10)	for i in range(10):
107 코드	class AE(nn.Module) # (1) def __init__(self) # (2) def forward(self, x) # (16)	class AE(nn.Module): # (1) def __init__(self): # (2) def forward(self, x): # (16)

108 13번	Output의 크기를 '256'으로	Output의 크기를 ' <b>512'로</b>
109 코드아래 1라인	설계하기'의 (6)에서 정의한	설계하기'의 (1)에서 정의한
109 코드8번 라인 1,3	def train(model, train_loader, optimizer, log_interval) for batch_idx,(image, _) in enumerate(train_loader) #(2)	def train(model, train_loader, optimizer, log_interval): for batch_idx,(image, _) in enumerate(train_loader): #(2)
111 코드라인 1, 6	def evaluate(model, test_loader) with torch.no_grad( ) #(5)	def evaluate(model, test_loader): with torch.no_grad( ): #(5)
112 13번	Mini-Batch 개수만큼	<b>데이터</b> 개수만큼
112 코드라인 2, 7, 12	for Epoch in range(1, EPOCHS + 1) for i in range(10) for i in range(10)	for Epoch in range(1, EPOCHS + 1): for i in range(10): for i in range(10):
114 코드2번 라인1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
114 코드5번 라인3	for i in range(10)	for i in range(10):
115 코드6번 라인 1,2,16	class AE(nn.Module) def __init__(self) def forward(self, x)	class AE(nn.Module): def __init__(self): def forward(self, x):
115 코드8번 라인 1, 3	def train(model, train_loader, optimizer, log_interval) for batch_idx,(image, _) in enumerate(train_loader)	def train(model, train_loader, optimizer, log_interval): for batch_idx,(image, _) in enumerate(train_loader):
116 코드9번 라인 1, 6	def evaluate(model, test_loader) with torch.no_grad( )	def evaluate(model, test_loader): with torch.no_grad( ):
116 코드10번 라인 1,6,11	for Epoch in range(1, EPOCHS + 1) for i in range(10) for i in range(10)	for Epoch in range(1, EPOCHS + 1): for i in range(10): for i in range(10):
121 아래 2라인	선행 <a href="http://item.gmarket.co.kr/Item?goodscode=1686873084">http://item.gmarket.co.kr/ Item?goodscode=1686873084</a> 결합	<b>선행 결합</b>
124 아래 5라인	픽셀 값을 뽑으면 'Max Pooling', 평균 픽 셀 값을 뽑으면 'Average Pooling'이라 합니다.	픽셀 중 최대값을 추출하는 것을 'Max Pooling', 사각형 안의 픽셀 값의 평균을 계산하여 추출하는 것을 'Average Pooling'이라고 합니다.
125 6라인	CNN의 구조는 다음과 같습니다.	CNN의 구조는 <b>Convolution - Pooling - Convolution - Pooling - Fully Connected Layer</b> 와 같은 구조를 지닙니다.
129 코드2번 라인 1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
132 코드5번 라인 3	for i in range(10)	for i in range(10):
132 코드6번	class Net(nn.Module) #(1) def __init__(self) #(2) def forward(self, x) #(7)	class Net(nn.Module): #(1) def __init__(self): #(2) def forward(self, x): #(7)
134 2번	(2) Back Propagation을 통해 파라미터를 업데이트할 때 이용하는 옵티마이저를 정의 합니다. 이 예제에서는 Adam 알고리즘을 이용하며, 파라미터를 업데이트할 때 반영될 학습률을 0.001로 설정합니다. 보통 Adam을 기본 옵션으로 설정하며 학습률은 분석가 취향에 따라 다르게 설정하기도 합니다.	
135 코드8번 라인 1, 3	def train(model, train_loader, optimizer, log_interval) for batch_idx,(image, label) in enumerate(train_loader) #(2)	def train(model, train_loader, optimizer, log_interval): for batch_idx,(image, label) in enumerate(train_loader): #(2)

136 코드9번 라인 1, 5	def evaluate(model, test_loader) with torch.no_grad( ) # (4)	def evaluate(model, test_loader): with torch.no_grad( ): # (4)
137 코드라인1	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):
136 코드2번 라인 1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
136 코드5번 라인 3	for i in range(10)	for i in range(10):
138 코드2번 라인 1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
138 코드5번 라인 3	for i in range(10)	for i in range(10):
139 코드6번 라인 1,2,7	class Net(nn.Module) def __init__(self) def forward(self, x)	class Net(nn.Module): def __init__(self): def forward(self, x):
140 코드8번 라인 1, 3	def train(model, train_loader, optimizer, log_interval) for batch_idx,(image, label) in enumerate(train_loader)	def train(model, train_loader, optimizer, log_interval): for batch_idx,(image, label) in enumerate(train_loader):
140 코드9번 라인 1, 5	def evaluate(model, test_loader) with torch.no_grad( )	def evaluate(model, test_loader): with torch.no_grad( ):
140 코드10번 라인 1	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):
144 코드6번 라인 1, 2	class Net(nn.Module) # (1) def __init__(self) # (2)	class Net(nn.Module): # (1) def __init__(self):# (2)
142 코드	def forward(self, x) # (20)	def forward(self, x): # (20)
147 4라인	[그림 4-12]를 보면 왼쪽	[그림 4-14]를 보면 왼쪽
147 아래 4라인	[그림 4-13]의 예시를	[그림 4-15]의 예시를
148 5라인	[그림 4-14]의 두 번째	[그림 4-16]의 두 번째
148 아래 6라인	[그림 4-14]의 세 번째	[그림 4-16]의 세 번째
152 5라인	[그림 4-15]와 같습니다.	[그림 4-17]과 같습니다.
153 아래 2라인	[그림 4-18]처럼	[그림 4-20]처럼
155 1라인	[그림 4-20]을 확대한 부분이	[그림 4-22]를 확대한 부분이
156 7라인	[그림 4-22]처럼 이전	[그림 4-24]처럼 이전
158 코드	class BasicBlock(nn.Module) # (1) def __init__(self, in_planes, planes, stride = 1) # (2) def forward(self, x) # (24) class ResNet(nn.Module) # (30) def __init__(self, num_classes = 10) # (31)	class BasicBlock(nn.Module): # (1) def __init__(self, in_planes, planes, stride = 1): # (2) def forward(self, x): # (24) class ResNet(nn.Module): # (30) def __init__(self, num_classes = 10): # (31)
159 코드	def _make_layer(self, planes, num_blocks, stride) # (44) def forward(self, x) # (51) out = F.avg_pool2d(out, 8s) # (56)	def _make_layer(self, planes, num_blocks, stride): # (44) def forward(self, x): # (51) out = F.avg_pool2d(out, 8) # (56)
163	(43) 세 번째 레이어의 입력 값이 64이며, 최종 출력 값은 10개의 클래스를 표현하기 위해 One-Hot Encoding으로 표현된 벡터 값과 Loss를 계산해야 하므로 출력 값의 크기를 10으로 설정합니다.	
163~165	(44)~(58)	(45)~(59)
169 코드	for Epoch in range(1, EPOCHS + 1) # (6)	for Epoch in range(1, EPOCHS + 1): # (6)
175 코드2번 라인1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
176	transforms.C[Enter]Crop(224), # (7)	transforms.CenterCrop(224), # (7)

177	(7) 해당 이미지 중앙을 기준으로 224 * 224 크기로 이미지를 잘라내어 사이즈를 변경하는 것을 의미합니다. (8) 해당 이미지를 256 * 256 크기로 사이즈를 변경합니다.	
178 5번코드 라인 3	for i in range(10)	for i in range(10):
179 6번코드 라인 1	def train(model, train_loader, optimizer, log_interval)	def train(model, train_loader, optimizer, log_interval):
180 7번코드 라인 1, 5	def evaluate(model, test_loader) with torch.no_grad( )	def evaluate(model, test_loader): with torch.no_grad( ):
182 3번	resnet34 모델	resnet18 모델
182 10번코드	for Epoch in range(1, EPOCHS + 1) # (1)	for Epoch in range(1, EPOCHS + 1): # (1)
184 (1) 위 코드 추가	<pre> ''' 11. IMAGENET 데이터로 미리 학습된 ResNet18 모델을 불러온 후 개미, 벌 이미지 데이터에 맞게 Fine Tuning 해보기 ''' model = models.resnet18(pretrained = True)           #(1) num_fts = model.fc.in_features                       #(2) model.fc = nn.Linear(num_fts, 2)                     #(3) model = model.cuda()                                #(4)  optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001) #(5) EPOCHS = 10   #(6) for epoch in range(1, EPOCHS + 1):                   #(7)     train(model, dataloaders["train"], optimizer, log_interval = 5) #(8)     valid_loss, valid_accuracy = evaluate(model, dataloaders["val"]) #(9)     print("\n[EPOCH: {}], \tTest Loss: {:.4f}, \tTest Accuracy: {:.2f}%\n".format(epoch, valid_loss, valid_accuracy)) </pre>	
184 2번	resnet34 모델	resnet18 모델
186 2번코드 라인1	if torch.cuda.is_available( )	if torch.cuda.is_available( ):
187 3번코드 라인9	transforms.C[Enter]Crop(224),	transforms.CenterCrop(224),
187 5번코드 라인3	for i in range(10)	for i in range(10):
187 6번코드 라인 1, 3	def train(model, train_loader, optimizer, log_interval) for batch_idx,(image, label) in enumerate(train_loader)	def train(model, train_loader, optimizer, log_interval): for batch_idx,(image, label) in enumerate(train_loader):
188 7번코드 라인 1, 5	def evaluate(model, test_loader) with torch.no_grad( )	def evaluate(model, test_loader): with torch.no_grad( ):
188 10번코드 라인 1	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):
189 라인 2	for Epoch in range(1, EPOCHS + 1)	for Epoch in range(1, EPOCHS + 1):
189 아래 8라인	optimizer~ 끝까지 삭제	
202 라인 1	def indexed_sentence(sentence)	def indexed_sentence(sentence):
203 두 번째 코드 라인 5	def indexed_sentence_unk(sentence)	def indexed_sentence_unk(sentence):
212 첫 번째 코드 라인 3,5,7,10,21	def get_stats(vocab) for word, freq in vocab.items( ) for i in range(len(symbols)-1) def merge_vocab(pair, v_in) for i in range(num_merges) # 4번 과정	def get_stats(vocab): for word, freq in vocab.items( ): for i in range(len(symbols)-1): def merge_vocab(pair, v_in): for i in range(num_merges): # 4번 과정
214 라인 7	for i in range(num_merges)	for i in range(num_merges):



216 첫 번째 코 드라인 4	for n in range(5)	for n in range(5):
217	(1) tokenize Module에 문장을	(1) <b>tokenizer.tokenize 함수에</b> 문장을
229 라인 2	def PreProcessingText(input_sentence)	def PreProcessingText(input_sentence):
230 두 번째 코 드라인 4,12	for idx,(k, v) in enumerate(TEXT.vocab.stoi.items( )) for idx,(k, v) in enumerate(LABEL.vocab.stoi.items( ))	for idx,(k, v) in enumerate(TEXT.vocab.stoi.items( )): for idx,(k, v) in enumerate(LABEL.vocab.stoi.items( )):
244 라인5,6	class SentenceClassification(nn.Module) def __init__(self, **model_config)	class SentenceClassification(nn.Module): def __init__(self, **model_config):
244 아래 라인5	def forward(self, x)	def forward(self, x):
247 라인 9	def binary_accuracy(preds, y)	def binary_accuracy(preds, y):
257 코드라인 7,8,13	class SentenceClassification(nn.Module) def __init__(self, **model_config) def forward(self, x)	class SentenceClassification(nn.Module): def __init__(self, **model_config): def forward(self, x):
260 세 번째 코 드라인 1	def PreProcessingText(input_sentence)	def PreProcessingText(input_sentence):
262 세 번째 코 드라인 1,2	class SentenceClassification(nn.Module) def __init__(self, **model_config)	class SentenceClassification(nn.Module): def __init__(self, **model_config):
263 코드라인16	def forward(self, x)	def forward(self, x):
264 라인 1,6,26,31	def train(model, iterator, optimizer, loss_fn, idx_Epoch, **model_params) for idx, batch in enumerate(iterator) def evaluate(model, iterator, loss_fn) with torch.no_grad( )	def train(model, iterator, optimizer, loss_fn, idx_Epoch, **model_params): for idx, batch in enumerate(iterator): def evaluate(model, iterator, loss_fn): with torch.no_grad( ):
265 코드라인18	for Epoch in range(N_EPOCH)	for Epoch in range(N_EPOCH):
266 세 번째 코 드라인 5	def predict_sentiment(model, sentence)	def predict_sentiment(model, sentence):
267 두 번째 코 드라인 5	def new_tokenizer(sentence)	def new_tokenizer(sentence):
267 세 번째 코 드라인 1,11	def PreProcessingText(input_sentence) def PreProc(list_sentence)	def PreProcessingText(input_sentence): def PreProc(list_sentence):
268 두 번째 코 드 라인 4,5,10	class SentenceClassification(nn.Module) def __init__(self, **model_config) def forward(self, x)	class SentenceClassification(nn.Module): def __init__(self, **model_config): def forward(self, x):
269 첫 번째 코 드라인 5	def count_parameters(model)	def count_parameters(model):
269 두 번째 코 드라인 7	for Epoch in range(N_EPOCH)	for Epoch in range(N_EPOCH):
274 아래 2라인	의 입장에서서는 $D(x)$ 가 1이고(진짜 데이터 를 1로 구분) 가 0일 때	<b><math>D</math>의 입장에서서는 <math>D(x)</math>가 1이고(진짜 데이터를 1로 구분) <math>D(G(z))</math>가 0일 때</b>
91 아래 6라인	[그림 7-4]를 이용해 확인할 수	<b>[그림 6-21] 자료를 통해 확인할 수</b>