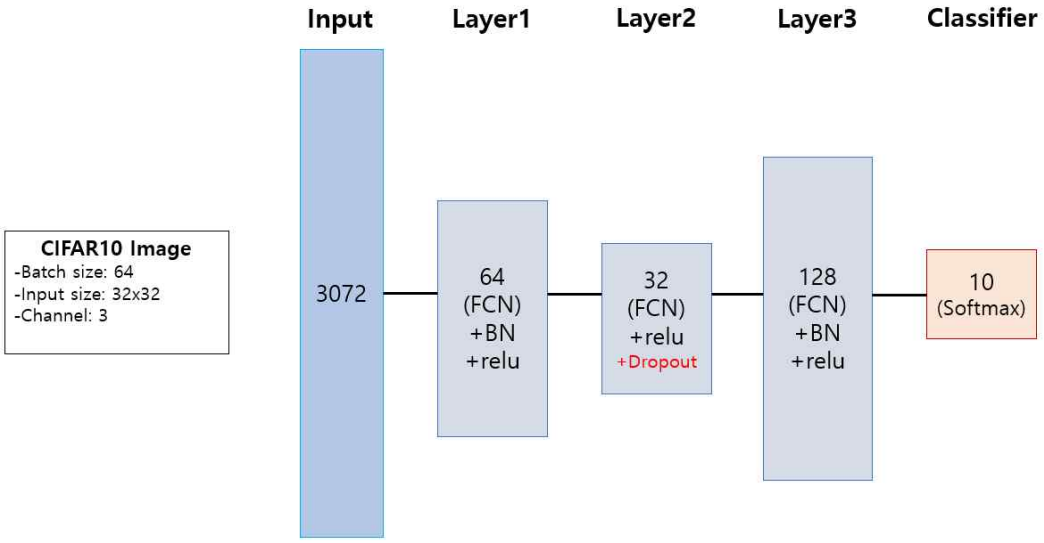
1. 아래 그림에 해당하는 Fully Connected Layer를 구현하시오



[참고사항]

1. 해당 문제 의도는 높은 성능이 아닌 제시한 모델을 정확히 구현하는 것에 있습니다. 해당 모델로 기록할 수 있는 성능보다 과도하게 낮은 성능이 나오거나 추가적인 테크닉을 사용하여 과도하게 높은 성능이 나오면 감점 사항이 됩니다.
2. Layer1과 Layer3은 Batch Norm, Layer2는 Dropout으로 구성되어있습니다.
3. Dropout의 p값은 0.5입니다.
4. BN(Batch Norm)과 Dropout의 연산순서는 그림에 나타난 순서와 같습니다.

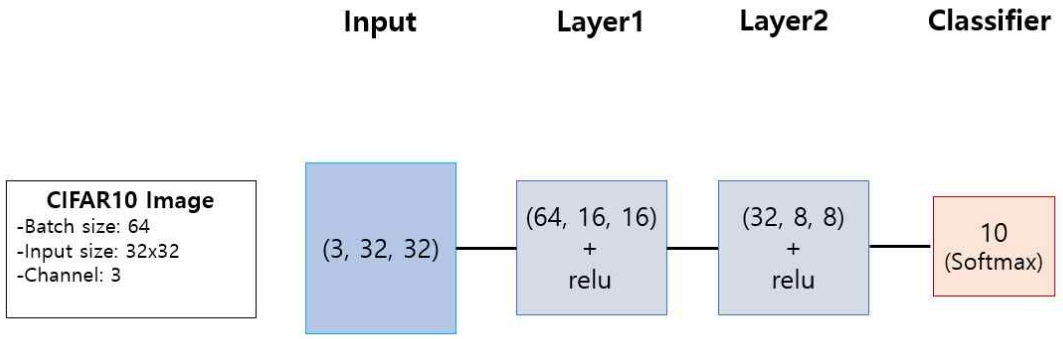
- Layer1, 3에서는 FC이후 BN이 수행되며, BN 이후 relu가 수행됩니다.
- Layer2는 FC 이후 relu가 수행되며, relu 이후 Dropout(p=0.5)이 수행됩니다.

5. 각 Layer와 Classifier 내부 숫자는 Node 숫자를 의미합니다.
6. CIFAR10 한 장의 이미지 shape는 (3, 32, 32)이며, 본 네트워크에 넣기 위해 3072로 reshape을 하셔야 합니다. (3072=3*32*32)
7. 코드에서 구현이 명시된 MyModel class 이외 부분의 코드 수정 시 감점 사항이 됩니다.
8. 기존

[파일 제출]

-Kernel->Restart & Run All을 실행하여 모든 결과값을 출력한 후, File->Download as HTML형태로 제출해주세요

2. 아래 그림에 해당하는 CNN 모델을 구현하시오



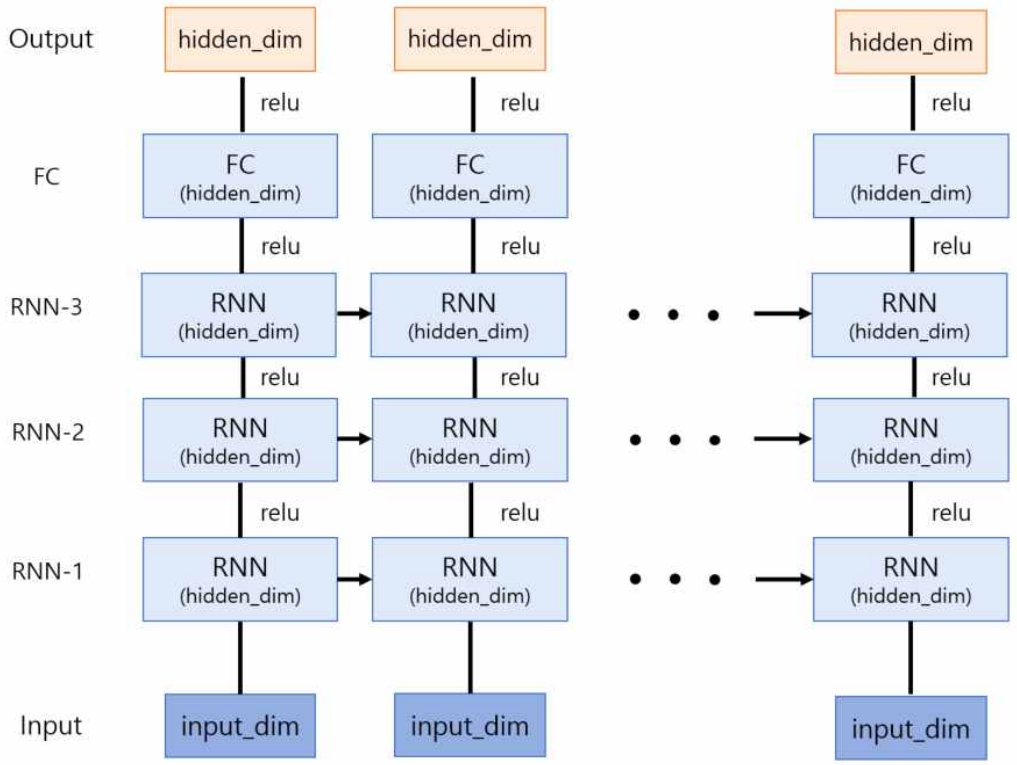
[참고사항]

1. 해당 문제 의도는 높은 성능이 아닌 제시한 모델을 정확히 구현하는 것에 있습니다. 해당 모델로 기록할 수 있는 성능보다 과도하게 낮은 성능이 나오거나 추가적인 테크닉을 사용하여 과도하게 높은 성능이 나오면 감점 사항이 됩니다.
2. Layer1, Layer2은 3x3 필터를 갖는 Convolution 연산을 활용해야 합니다.
(Feature map 수, 가로길이, 세로길이)를 의미합니다.
3. Conv연산 후 크기 유지를 위해 stride와 padding은 각각 1을 가지고, 이후 stride가 2인 Max pooling을 통해 이미지 크기를 절반으로 줄입니다.
4. Layer2와 Classifier 사이의 연결은 Layer2의 relu연산 이후,
(32, 8, 8) 벡터를 2048짜리 벡터로 reshape하고, nn.Linear(2048, 10) 연산을 활용하여 Classifier로 연결하시면 됩니다. nn.Linear(2048, 10)연산 이후 softmax를 통해 분류를 진행합니다.
5. 하나의 CIFAR10 이미지 shape은 (3, 32, 32)입니다.
6. 코드에서 구현이 명시된 MyModel class 이외 부분의 코드 수정시 감점 사항이 됩니다.

[파일 제출]

-Kernel->Restart & Run All을 실행하여 모든 결과값을 출력한 후, File->Download as HTML형태로 제출해주세요

3. 아래 그림에 해당하는 RNN 모델을 구현하시오



[참고사항]

1. 해당 문제 의도는 높은 성능이 아닌 제시한 모델을 정확히 구현하는 것에 있습니다. 해당 모델로 기록할 수 있는 성능보다 과도하게 낮은 성능이 나오거나 추가적인 테크닉을 사용하여 과도하게 높은 성능이 나오면 감점 사항이 됩니다.
2. 문제의 Input으로는 제시된 문장들을 학습하는 Next Character Prediction 문제가 주어지며, 총 sequence_length는 10을 활용할 예정입니다. 그러나, 모든 Input은 전처리 후 들어오므로, 전처리 대신 해당 모델 구조 구현에 집중하시면 됩니다.
3. RNN은 그림과 같이 총 3 layer를 갖고 있으며, 모든 RNN Layer와 Fully Connected Layer 연산 이후는 relu함수가 사용됩니다.

[파일 제출]

-Kernel->Restart & Run All을 실행하여 모든 결과값을 출력한 후, File->Download as HTML형태로 제출해주세요

4. X를 통해 Y를 예측하는 간단한 Linear Regression 학습(y = ax + b)을 진행한 후, 학습 결과를 [결과 예시]처럼 출력하고(그래프 출력은 기본적으로 제공됨), 해당 모델의 최종 학습 epoch 횟수(num_epochs회)와 학습된 모델 파라미터를 저장하시오.

[참고사항]

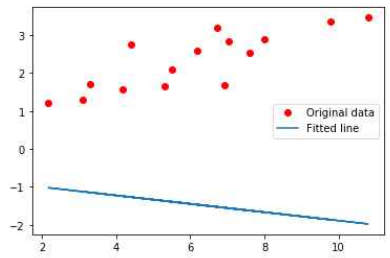
1. 해당 문제 의도는 높은 성능이 아닌 제시한 모델을 정확히 구현하는 것에 있습니다. 해당 모델로 기록할 수 있는 성능보다 과도하게 낮은 성능이 나오거나 추가적인 테크닉을 사용하여 과도하게 높은 성능이 나오면 감점 사항이 됩니다.
2. 코드 내부의 Hyperparameter 값을 그대로 활용하셔서 학습을 진행해주셔야 합니다.
3. Loss Function은 MSE Loss를, Optimizer는 Adam Optimizer를 사용하시면 됩니다. Adam Optimizer의 learning rate는 Hyperparameter를 사용하셔야하며, learning rate이외 beta값이나 Epsilon값은 Adam Optimizer default 변수를 사용하시면 됩니다.
3. 코드 출력은 다음 코드를 활용하여, 총, 5 Epoch주기로 학습된 결과를 출력해주시면 됩니다.

```
print ('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.item()))
```

4. [결과 예시]는 출력 예시일뿐, 실제 제공한 하이퍼파라미터값을 통해 학습을 진행한 경우 Loss와 그래프값은 다르게 출력됩니다.
5. 'model.ckpt' 파일에는 최종적으로 학습된 epoch와 모델 파라미터가 저장돼야 합니다. 각각의 키값은 'epoch'와 'net'으로 하시면 됩니다.
6. 마지막 isCorrectlySaved()가 "Correctly Saved"라는 문구가 뜨면 정상적으로 저장이 된 것입니다.

[결과 예시]

```
Epoch [5/60], Loss: 39.4624
Epoch [10/60], Loss: 36.2194
Epoch [15/60], Loss: 33.2459
Epoch [20/60], Loss: 30.5196
Epoch [25/60], Loss: 28.0198
Epoch [30/60], Loss: 25.7278
Epoch [35/60], Loss: 23.6263
Epoch [40/60], Loss: 21.6995
Epoch [45/60], Loss: 19.9328
Epoch [50/60], Loss: 18.3129
Epoch [55/60], Loss: 16.8276
Epoch [60/60], Loss: 15.4658
```



[파일 제출]

-Kernel->Restart & Run All을 실행하여 모든 결과값을 출력한 후, File->Download as HTML형태로 저장한 후, 해당 HTML파일과 'model.ckpt'파일을 zip으로 묶은 후 제출해주세요