2. 1학년 여름방학 시기에 진행한 Pre-URP 프로그램

프로그램 진행 전 주어진 사전과제는 Sung Kim 교수님의 모두를 위한 딥러닝 강좌를 보고, 딥러닝을 공부하며 배운 내용을 정리하는 것이었다.

<사전 연구 수행 보고서>

1. 딥 러닝의 구조

딥 러닝은, CNN과 RNN, DNN 등의 구조로 이루어진다. CNN은 합성곱 신경망(convolutional Neural Network)로, 최소한의 전처리를 사용하도록 설계된 다계층 퍼셉트론의 한 종류이다. 순환 신경망은 신경망 내부의 메모리 활용이 가능한데, 이런 특성으로 필기체 인식과 같은 분야에 활용되며 인식률 또한 높다. 순환 신경망은 완전 순환망, ESN, LSTM, Bi-directional RNN, CTRNN 등 굉장히 많은 종류가 있다. 순환 신경망을 훈련시키는 방법으로는 경사 하강법, Hessian FreeOptimization, Global Optimization Methods 방식이 쓰이고 있다. DNN은 심층 신경망(Deep Neural Network)이다. DNN은 입력층과 출력층 사이에 여러 개의 은닉층들로 이루어진 인공신경망이라고 할 수 있다.

2. 적용 사례

페이스북의 얼굴 인식 기능을 예로 들 수 있다. , 첫 번째는 얼굴 검출인데, 이미지를 흑백으로 바꾸어 픽셀들의 어둡기를 체크하여, 이미지가 어두워지는 방향을 나타내는 화살표를 그린다. 이를 통해 이미지를 얼굴의 기본 구조가 심플한 방법으로 표시되는 HOG표현으로 바꾼다. 이제 분리해낸 얼굴들을 위치교정하고, 투영한다. Face landmark estimation이라는 알고리즘을 사용하여 눈과 입을 최대한 가운데로 올 수 있도록 한다. 이제 얼굴을 구별하는 단계인데, Deep Convolutional Neural Network를 훈련시켜 각 얼굴에 대해 128개의 측정값을 생성하도록 훈련시킨다. 이 훈련 과정은 훈련용 얼굴 사진 적재, 동일한 얼굴의 다른 사진 적재, 전혀 다른 사람의 사진을 적재하는 과정으로 이루어진다. 여기에서 각 얼굴에 대한 128개의 측정값을 임베딩이라고 부른다. 사진과 같은 복잡한 원시 데이터를 컴퓨터가 생성한 숫자값의 목록으로 축소한다는 아이디어를 이용하는 것이다. 마지막 단계는 인코딩에서 사람의 이름을 찾는 과정인데, 가장 쉬운 단계이다. 이 과정에서는 과거에 측정해 놓은 모든 얼굴에 대해 이 얼굴의 측정값에 가장 가까운 사람이 누구인지 확인하는 것이다.

3. 훈련 방법

Gradient Descent, SGD, Adam Optimizer 등이 있다.Gradient Descent 알고리즘은 최소화문제의 경우에 가장 많이 사용되고, cost(w1,w2,w3…)등의 다양한 변수가 있는 경우에도 활용된다. Gradient Descent의 연산을 기계적으로 실행시켜서 얻어내는 변화하는 W값(H(x)= W_x라고 할 때)을 자동으로 cost(W)가 최소가 되는 W값이라고 한다. 만약 W, b, cost(W)의 함수를 3차원에 그렸을 때, W또는 b값의 초기값이 아주 조금 달랐는데 minimize된 W, b값이 많이 다르다면 Gradient descent algorithm을 적용하기에는 좋다고 볼 수 없을 것이다. Linear Regression에서의cost function은 3차원에서 Convex function으로, 볼록성이 계속 유지되어 초기값이 어디서 시작하던 반드시 동일한 지점의 최솟값을 찾아내는 것이 보장된다. 다른 cost function을 설계할 때도 반드시 Convex function인지 확인하여야 한다.

4. Hyperparameter의 의미

Hyperparameter란, 신경망 학습을 통해서 튜닝 또는 최적화 해야하는 주변수가 아니라, 학습 진도율이나 일반화 변수처럼, 사람들이 선험적 지식으로 설정을 하거나 또는 외부 모델 메커니즘을 통해 자동으로 설정이 되는 변수를 말한다.

강화학습의 기본 배경

강화 학습은 기계 학습의 한 영역으로, 행동심리학에서 영향을 받았으며 어떤 환경 안에서 정의된 에이전트가 상태를 인식하여 보상을 최대화하는 방향으로 선택을 하는 방법이다. 쉽게 표현하자면, 강화 학습은 각 상태에서의 행동에 따라 보상값을 받도록 하여, 보상값이 최대화되는 방향으로 학습이 진행되는 것이다. 보상값을 여러 번 누적시켜야 하므로 시간이 오래 걸린다. RL은 Alphago의 핵심 알고리즘으로 사용되게 되었다.

강화학습의 요소

강화학습 모델은 환경상태 집합(S), 행동 집합(A), 포상의 집합으로 구성된다. 매번 t라는 시점마다 에이전트는 상황마다의 자신의 상태(S에 포함된)와 가능한 행동A를 가지고 있다. 에이전트는 이런 상황마다 어떤 행동을 취하고, 취한 행동으로부터 환경에서 새로운 상태와 포상을 받는다. 이 포상을 통해 강화 학습의 에이전트는 포상값 R을 최대화하는 정책을 개발한다. 종료 상태가 존재하는 마르코프 결정 과정에서는 $R = r_0 + r_1 + \cdots + r_n = \sum_{t=1}^{n} r_t$이고, 그렇지 않은 마르코프 결정 과정에서는 $R = \sum_{t=1}^{n} \gamma^t r_t$가 된다.이 식에서 감마는 미래의 포상이 현재 상태에서 얼마나 가치 있는지를 표현하는 discountfactor로 0과1 사이의 값이다.

5. Markov Decision Process

마르코브 결정 과정은 의사결정 과정을 모델링하는 수학적인 틀을 제공한다. 이 때 의사결정의 결과는 의사결정자의 결정과 임의적인 값으로 주어진다. 마르코프 결정 과정은 동적 계획법과 강화 학습 등의 방법으로 푸는 넓은 범위의 최적화 문제에 유용한 도구로 활용된다. 마르코브 결정 과정은 이산 시간 확률 제어 과정이라고 할 수 있다. 어떤 시점에 마르코프 결정 과정은 상태 S에 존재하고, 의사결정자는 상태 S에서 행동 A를 취할 수 있고, 다음 시점에서 마르코프 결정 과정이 확률적으로 새로운 상태 S프라임 으로 전이된다.

마르코브 결정 과정은 마르코브 연쇄의 확장된 형태로 볼 수 있다. 마르코프 과정에서 연속적인 시간 변화를 고려하지 않고 이산적인 경우만 고려한 경우를 마르코프 연쇄라고 한다.

훈련 방법

마르코브 결정 과정에서 중요한 알고리즘의 종류로는 값 반복법과 정책 반복법이 있다.

값 반복법은 역진 귀납법이라고도 불리고, 파이함수를 사용하지 않고 필요에 따라 V(s)를 사용하여 파이(s)를 계산한다. $V_{i+1}(s) := \max \sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V_i(s'))$라는 식을 만족하게 된다. 여기서 i는 반복 차수를 의미하며,알고리즘은 i값이 0일 때부터 시작한다.

정책 반복법은 파이(s)를 계산하는 첫 번째 단계를 한 번만 수행하고 V(s)를 계산하는 두 번째 단계가 수렴할 때까지 반복된다. 이 두 단계가 계속 반복된다. 두 번째 단계가 수렴할 때까지가 아닌, 연립일차방정식으로 바꾸어 계산할 수도 있다.

6. 적용 사례

마르코프 결정 과정은 동적 계획법과 강화 학습 등의 방법으로 푸는 넓은 범위의 최적화 문제에 유용한도구로 활용되며 로봇 공학, 제어 자동화, 경제학, 제조업 등의 영역에서 폭넓게 사용되고 있다.

7. 강화학습에 적합한 임무(강화학습의 요소별), 훈련 방법

강화학습의요소로는 Q-learning, Q-learning table, Q-network, DQN 등이 있다. Deepmind는 이러한 Q-Network의 불안정하다는 특성을 개선하여 알파고를 개발하였다. Q-Net은 매번 입력되는 환경이 너무 유사하고 연관성이 많아 전체를 봐야 하는 알고리즘인 linear regression으로 제대로 학습할 수 없고, 변화시키고 싶지 않은 target 함수인 Y 또한 변한다는 것이다. 신경망 Q-Learning으로는 오목 게임을 구현할 수 있다. 양쪽 플레이어가 번갈아 수를 둘 때, 두 번째 플레이어는 돌의 상태를 반전하여 입력으로 이용한다. 그렇게 하여 각 플레이어마다 따로 학습하는 것이 아니라, 하나의 모델을 사용하도록 하였다. 게임에 이겼을 경우 1의 보상값을 받도록 하는데, 이 조건만을 이용하면 너무 학습이 느려 자기 돌과 인접한 위치일 경우 0.05의 보상값을 추가로 주었다고 한다.

강화 학습을통해서는 ANN 온라인을 훈련하는 데 사용할 수 있는 알고리즘을 찾을 수 있었다. Cart pole swing up 문제는 ANN으로 해결하고자 하는 문제인데, 강화 학습을 통해 ANN에게 보상과 처벌을 기반으로 배우게 할 수 있다.

훈련에 대한방법으로는, 미래 최대 reward에 현재 reward를 합해 업데이트만 하면 학습이 된다는 것이다. 게임판 전체의 Q를 0으로 초기화한 뒤 무한히 계속 학습을 반복하면 agent가 높은 값의 Q를 가진 길을 찾아가게 된다.

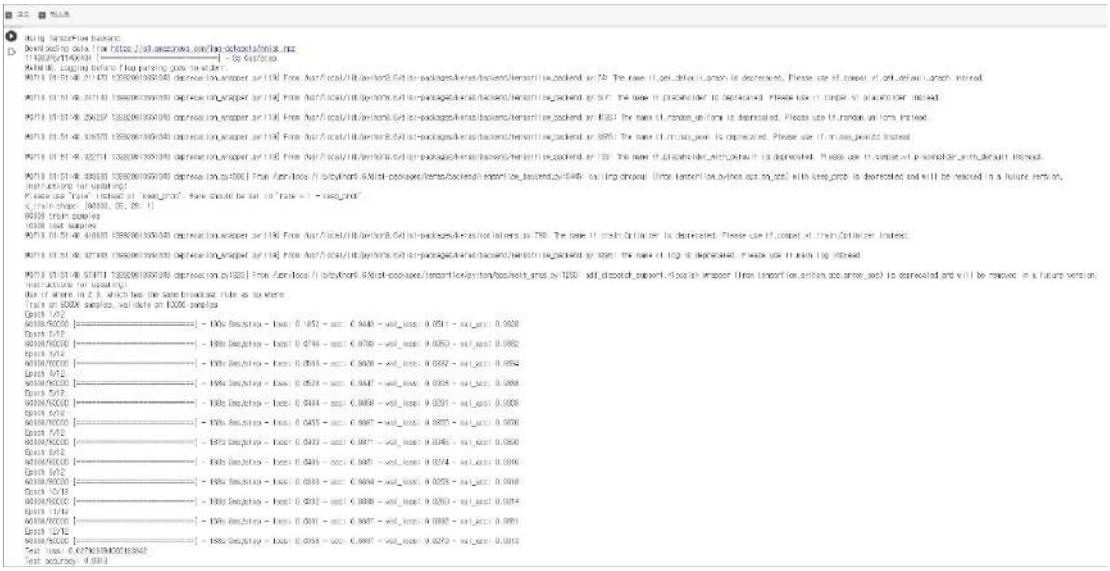
또다른 방법으로는DeepMind의 방법이 있다. 이는 non-stationary target를 해결하는데, layer수를 증가시키고 두께 또한 증가시킨다. 또, experiencereplay 방법은 agent의 action을 먼저 학습시키지 않고, 버퍼라는 것에 저장을 한다. 랜덤한 샘플들로 학습시키는 것이다. Target network를 분산시키는 마지막 방법은 여러 network에서 다른 값을 가져오고, 특정 값을 가져오는 network만 업데이트 시켜, 시간이 지나고 동일한 Q값을 복사하는 것이다.

8. 코드 실행 결과/코드 해석

다음 코드는 Simple CNN 알고리즘을 테스트하는 코드이다.

예제 코드는 딥러닝을 이용하여 mnist 데이터로 다음 데이터를 예측하는 코드이다.

코드 실행 결과



Using TensorFlowbackend.
Downloading datafrom https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434[==============================] - 0s 0us/step
WARNING: Loggingbefore flag parsing goes to stderr.
W071801:51:48.211470 139920813651840 deprecation_wrapper.py:119]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74:The name tf.get_default_graph is deprecated. Please usetf.compat.v1.get_default_graph instead.

W071801:51:48.247140 139920813651840 deprecation_wrapper.py:119]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:517:The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholderinstead.

W0718 01:51:48.256267139920813651840 deprecation_wrapper.py:119]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4138:The name tf.random_uniform is deprecated. Please use tf.random.uniform instead.

W071801:51:48.316578 139920813651840 deprecation_wrapper.py:119]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3976:The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

W071801:51:48.322711 139920813651840 deprecation_wrapper.py:119]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:133:The name tf.placeholder_with_default is deprecated. Please usetf.compat.v1.placeholder_with_default instead.

W071801:51:48.333630 139920813651840 deprecation.py:506]
From/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445:calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob isdeprecated and will be removed in a future version.
Instructions forupdating:
Please use `rate`instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
x_train shape:(60000, 28, 28, 1)
60000 trainsamples

```
10000 test samples
W071801:51:48.418620 139920813651840 deprecation_wrapper.py:119]
    From/usr/local/lib/python3.6/dist-packages/keras/optimizers.py:790: The
    nametf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizerinstead.

W071801:51:48.427948 139920813651840 deprecation_wrapper.py:119] From
    /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3295:The name
    tf.log is deprecated. Please use tf.math.log instead.

W071801:51:48.574711 139920813651840 deprecation.py:323]
    From/usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250:add_di
    spatch_support.<locals>.wrapper (fromtensorflow.python.ops.array_ops) is deprecated and
    will be removed in a futureversion.
Instructions forupdating:
Use tf.where in2.0, which has the same broadcast rule as np.where
Train on 60000samples, validate on 10000 samples
Epoch 1/12
60000/60000[==============================] - 190s 3ms/step - loss: 0.1852 - acc:
    0.9443 -val_loss: 0.0511 - val_acc: 0.9828
Epoch 2/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0746 - acc:
    0.9783 -val_loss: 0.0359 - val_acc: 0.9882
Epoch 3/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0580 - acc:
    0.9828 -val_loss: 0.0337 - val_acc: 0.9894
Epoch 4/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0528 - acc:
    0.9847 -val_loss: 0.0308 - val_acc: 0.9888
Epoch 5/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0484 - acc:
    0.9859 -val_loss: 0.0291 - val_acc: 0.9909
Epoch 6/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0455 - acc:
    0.9867 -val_loss: 0.0355 - val_acc: 0.9876
Epoch 7/12
60000/60000[==============================] - 187s 3ms/step - loss: 0.0433 - acc:
    0.9871 -val_loss: 0.0345 - val_acc: 0.9890
Epoch 8/12
60000/60000[==============================] - 189s 3ms/step - loss: 0.0405 - acc:
    0.9881 -val_loss: 0.0274 - val_acc: 0.9916
Epoch 9/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0380 - acc:
    0.9894 -val_loss: 0.0253 - val_acc: 0.9918
Epoch 10/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0382 - acc:
    0.9888 -val_loss: 0.0269 - val_acc: 0.9914
Epoch 11/12
60000/60000[==============================] - 189s 3ms/step - loss: 0.0381 - acc:
    0.9887 -val_loss: 0.0392 - val_acc: 0.9881
Epoch 12/12
60000/60000[==============================] - 188s 3ms/step - loss: 0.0358 - acc:
    0.9897 -val_loss: 0.0279 - val_acc: 0.9913
Test loss:0.027926094066183942
Test accuracy:0.9913
```

코드 해석:

패키지 가져오기

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

batch_size = 32 배치 크기

num_classes = 10 클래스 별 나눌 수

epochs = 12 반복수

입력할 이미지 크기

```
img_rows, img_cols = 28, 28
```

훈련데이터, 테스트 데이터 불러오기

```
(x_train, y_train), (x_test, y_test) =mnist.load_data()
```

케라스 텐서 데이터 형식에 맞게 설정

```
if K.image_data_format() == 'channels_first': (데이터갯수, 채널 수,행,열)
x_train = x_train.reshape(x_train.shape[0], 1,img_rows, img_cols)
x_test = x_test.reshape(x_test.shape[0], 1,img_rows, img_cols)
input_shape = (1, img_rows, img_cols)
else:
x_train = x_train.reshape(x_train.shape[0],img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0],img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
```

데이터타입변환

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

print('x_train shape:', x_train.shape) 출력

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
# convert class vectors to binary classmatrices
```

(클래스 벡터를 이진수 클래스 행렬로 변환)

```
y_train = keras.utils.to_categorical(y_train,num_classes)
y_test = keras.utils.to_categorical(y_test,num_classes)
```

```
model = Sequential()
```

케라스모델설정(은닉층, 출력층 구분)

모델추가

```
model.add(Conv2D(32, kernel_size=(3, 3),
```

activation='relu'
활성화함수 '렐루'
렐루함수는 Rectified Linear Unit이라는 함수로,
$h(x) = \begin{cases} x\,(x > 0) \\ 0\,(x \leq 0) \end{cases}$의 공식을 따른다.

input_shape=input_shape)) 컨볼루션레이어(합성곱 계층)
model.add(Conv2D(64, (3, 3),activation='relu')) 컨볼루션레이어
model.add(MaxPooling2D(pool_size=(2, 2))) 맥스풀링 이미지특징 추출하기위해사용
model.add(Dropout(0.25)) 드롭아웃처리 처리속도증가
model.add(Flatten()) 1차원으로 변환
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes,activation='softmax'))
소프트맥스 분류함수(확률로 바꾸어주는 함수)

print('Test accuracy:', score[1])

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])
모델완성 정확도 손실함수값 저장
(손실함수는 cost 함수라고도 하며,우리가 최소화하려는 목표함수임)
하이퍼파라미터값을 모델에 적용
(하이퍼파라미터는 learning rate, 또는 gamma 같은 값처럼 우리가 변화시키는 변수임)
model.fit(x_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,
validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test,verbose=0)
모델을 평가한 값을 score이란 변수에 저장
print('Test loss:', score[0]) 손실함수 정확도출력
print('Test accuracy:', score[1])

Pre-URP 연구 최종 보고서는 다음과 같이 첨부한다.

2019 과학영재 첨단연구 입문 프로그램 연구결과 보고서
TR 2019-085 (G03) p.301-316

심층 강화학습을 이용한 제어 문제 해결 탐색

A Quest for Solving Control Problem Using Deep Reinforcement Learning

책 임 지 도 자 : GIST 융합기술학제학부 류제하 교수
연 구 자 : 대전동신과학고등학교 이창섭
인천과학예술영재학교 김준서
경 산 과 학 고 등 학 교 김우진
경 기 과 학 고 등 학 교 임형민

KAIST 과학영재교육연구원
Global Institute For Talented EDucation

과학영재 첨단연구 입문 프로그램 결과보고서

G03. 심층 강화학습을 이용한 제어 문제 해결 탐색

A Quest for Solving Control Problem Using Deep Reinforcement Learning

Student : 대전동신과학고등학교 2학년 이창섭, harry020606@naver.com
인천과학예술영재학교 1학년 김준서, jsk491039@gmail.com
경산과학고등학교 2학년 김우진, kas8826@naver.com
경기과학고등학교 1학년 임형민, him030107@gmail.com

Professor : GIST 융합기술학제학부 류제하, ryu@gist.ac.kr

R. Assistant : GIST 융합기술학제학부 정대현, astra0314@gist.ac.kr

Abstract

본 연구는 로봇의 제어문제를 해결하기 위한 하나의 탐색으로서 강화학습 시 어떤 조건에서 가장 학습을 잘 하는지를 알아보기 위한 연구이다. 이 연구를 위해서 우리는 OpenAI GYM에 있는 Cartpole Simulation 환경을 이용했다. Cartpole의 DDQN 오픈소스 코드에서 기본으로 설정되어있는 Hyperparameter 8종과 함수 3종을 하나씩 우리가 임의로 정한 일정한 간격과 종류로 바꾸어 보며 각각 100회씩 실행했다. 실행 결과를 표와 그래프로 나타내고 이를 분석해 가장 좋은 점수를 받는 Hyperparameter를 찾아내는 방식으로 연구를 진행하였다. 이를 통해 우리가 얻어낸 최적의 Hyperparameter 값을 종합한 모델을 1000회 실행시키고, 오픈소스에 원래 있던 Hyperparameter를 대조군으로 하여 최근 100번 동안 시도의 평균점수가 195점 이상이 되는 횟수를 저장하고 이 횟수를 대조군과 비교했다. 이를 통해 우리의 연구가 오픈소스에 있던 기존의 Hyperparameter에 비해 더 좋은 점수를 낸다는 사실을 확인 할 수 있었다.

Key word : 강화학습, 로봇제어, Hyperparameter, Cartpole Simulation, DDQN

1. 서론

본 연구는 인공지능 강화학습을 이용하여 카트에 설치된 폴을 넘어지지 않도록 카트를 좌우로 제어하는 방법에 관한 것으로서 구체적으로는 Cartpole Simulation을 사용하여 최신의 DQN (Deep Q_Learning) 알고리즘을 적용하고 이 알고리즘의 여러 변수들을 변화시켜가면서 최적의 값들을 찾는 연구이다.

강화학습은 기계 학습의 한 영역으로, 행동심리학에서 영향을 받았으며 어떤 환경 안에서 정의된 에이전트가 상태를 인식하여 보상을 최대화하는 방향으로 행동을 선택 하는 방법이다.(Sutton & Barto, 2018) 쉽게 표현하자면, 강화학습은 각 상태에서의 행동에 따라 보상 값을 받도록 하여, 보상 값이 최대화되는 방향으로 학습이 진행되는 것이다. 보상 값을 여러 번 누적시켜야 하므로 시간이 오래 걸린다. 강화학습은 최근에 개발된 알고리즘이 아니다. 1997년의 머신러닝 교재(Sutton & Barto, 2018)에도 쓰여 있던 만큼 오래된 알고리즘이지만, 중요한 알고리즘으로 간주 받지 못하였다. 하지만 2010년대가 되고, 강화학습(Reinforcement learning)을 이용하여 Atari breakout(Minh, Kavukcuoglu, Silver & Graves, 2013)이라는 벽돌 깨기 게임을 인공지능이 플레이할 수 있게 되면서, 관심이 커지고 바둑 인공지능인 Alphago(Silver, Huang, Maddison, Guez & Sifre, 2016)의 핵심 알고리즘으로 쓰이면서 많은 주목을 받았다.

강화학습의 방법으로는 Q-learning, Q-learning table, Q-network, DQN 등이 있다.(Sutton & Barto, 2018)(Anschel, Baram & Shimkin, 2016) DeepMind는 이러한 Q-Network의 불안정하다는 특성을 개선하여 알파고를 개발하였다. Q-Net은 매번 입력되는 환경이 너무 유사하고 연관성이 많아 전체를 봐야 하는 알고리즘인 linear regression으로 제대로 학습할 수 없고, 변화시키고 싶지 않은 target 함수인 Y 또한 변한다는 것이다. 신경망 Q-Learning으로는 오목 게임을 구현할 수 있다. 양쪽 플레이어가 번갈아 수를 둘 때, 두 번째 플레이어는 돌의 상태를 반전하여 입력으로 이용한다. 그렇게 하여 플레이어마다 따로 학습하는 것이 아니라, 하나의 모델을 사용하도록 하였다. 게임에 이겼을 경우 1의 보상 값을 받도록 하는데, 이 조건만을 이용하면 너무 학습이 느려 자기 돌과 인접한 위치일 경우 0.05의 보상 값을 추가로 주었다고 한다. 강화학습을 통해서는 ANN 온라인을 훈련하는 데 사용할 수 있는 알고리즘을 찾을 수 있었다. Cart pole swing up 문제(Sutton & Barto, 2018)는 ANN으로 해결하고자 하는 문제인데, 강화학습을 통해 ANN에게 보상과 처벌을 기반으로 배우게 할 수 있다.

Q-Learning이 훈련이 되는 원리는, 미래 최대 reward에 현재 reward를 합해 업데이트하면 학습이 되는 것이다. 게임판 전체의 Q를 0으로 초기화한 뒤 무한히 계속 학습을 반복하면 agent가 높은 Q 값을 가진 길을 찾아가게 된다. 또 다른 방법으로는 DeepMind의 방법이 있다. 이는 non-stationary target을 해결하는데, layer 수를 증가시키고 노드 또한 증가시킨다. 또, experience replay 방법은 agent의 action을 먼저 학습시키지 않고, 버퍼라는 것에 저장한 후 이 저장고에서 가져온 랜덤한 샘플들로 학습시키는 것이다. Target network를 분산시키는 마지막 방법은 여러 network에서 다른 값을 가져오고, 특정 값을 가져오는 network만 업데이트시켜, 시간이 지나고 동일한 Q 값을 복사하는 것이다.

본 연구의 목표로는, Deep-Q-learning에서 Hyperparameter 변화에 따른 Cartpole 시뮬레이션 훈련 성능 변화 관찰이 있었다.

2. 연구방법(실험방법) 및 내용

본 연구를 진행한 환경은 OpenAI GYM에서 제공하는 Cartpole 라이브러리를 이용했다.(2019) 또한 신경망 구축을 위해 Keras 라이브러리를 사용하였다.(2019)

본 연구를 진행한 방법으로는, Hyperparameter 값을 바꾸어 보며 가장 최적화된 Hyperparameter값을 찾는 방법으로 진행하였다. 최적화된 Hyperparameter 값을 종합하면서 최신의 데이터 100개의 평균이 195가 넘는 에피소드를 체크해 디폴트 값과 비교한다.

본 연구의 강화학습을 진행한 방법으로는 DDQN(Hado, Silver & Guez, 2016)을 사용했는데, DQN이 agent가 현재 취할 수 있는 action 중 미래의 보상이 가장 큰 action을 neural network로 예측하는 것이다. DQN은 예측을 판 전체를 보고 하는데, 그래서 action을 하면서 그때그때 계산하는 것보다 큰 값을 예측한다. DDQN은 Double-Q-Learning + DQN의 약자로, 두 개의 모델을 번갈아 가며 경쟁적으로 학습과 예측을 시켜 정확도를 높이는 것이다.

연구를 진행할 때에 action 종류로는 좌우로 10의 힘을 가할 수 있고, state 종류로는 cart position, cart velocity, pole angle, pole velocity at tip이 있으며, 강화학습을 진행할 때에 reward를 주는 방법으로는 31프레임으로 모든 스텝에서 +1을 하고 연속된 100번의 시도에서 얻은 점수의 평균이 195 이상일 경우 성공으로 처리했다.

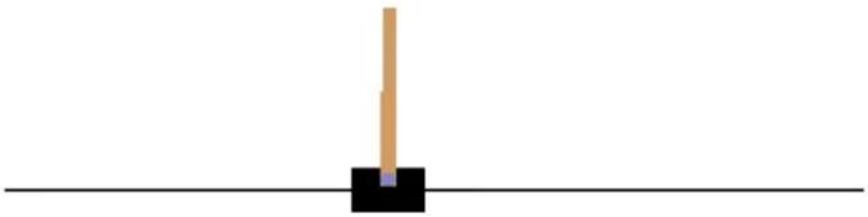


Figure 2.1. Cartpole Simulation

Hyperparameter 값들로는 gamma, epsilon, epsilon_min, epsilon_decay, the number of layer, batch_size, the number of node, learning rate, loss function, optimizer, activation function이 있다. 본 연구에서 변화시킨 Hyperparameter값과 그의 최솟값과 최댓값과 변화량은 다음과 같다.

	Hyperparameter Name	Min Value	Max Value	Step	Student	episode
1	Gamma	0.5	0.95	0.05	형민	100
2	Epsilon	0.1	1	0.2	우진	100
3	Epsilon_min	0.05	0.5	0.05	준서	100
4	Epsilon_decay	0.55	0.95	0.1	우진	100
5	The number of layer	1	4	1	창섭	100
6	Batch_size	2	64	x2	형민	100
7	The number of node	12	48	6	창섭	100
8	Learning rate	0.0005	0.1	x10	창섭	100
9	Loss function	5 functions			준서	100
10	optimizer	7 functions			준서, 형민	100
11	Activation Function	6functions			창섭	100

Table 2.1. List of hyper parameters in which each student was responsible for the study.

3. 결과 및 고찰

3.1 γ를 변경시킨 결과

γ는 강화학습에서 agent가 비효율적인 경로를 택하면 보상을 줄이는 정도에 대한 상수이다. 실험 결과 γ가 가장 클 때인 0.95일 때의 평균점수가 가장 높았고, 가장 작을 때인 0.5일 때 평균점수가 가장 낮은 것으로 기록되었다.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
Gamma_0.5	0.731751	0.030679	0.086577	0.033090	55.120000	44.376333
Gamma_0.55	0.744472	0.034672	0.071519	0.040045	75.726667	58.097951
Gamma_0.6	0.750986	0.033247	0.075533	0.041174	72.860000	52.421374
Gamma_0.65	0.745162	0.034871	0.083942	0.064952	80.433333	63.429532
Gamma_0.7	0.761117	0.032267	0.078014	0.043723	80.753333	68.255787
Gamma_0.75	0.773292	0.035904	0.093861	0.069860	81.113333	62.141563
Gamma_0.8	0.751084	0.035634	0.090977	0.053279	81.670000	72.215934
Gamma_0.85	0.770157	0.044682	0.131104	0.105755	74.156667	52.825298
Gamma_0.9	0.773685	0.046902	0.099618	0.070340	109.363333	88.138554
Gamma_0.95	0.802066	0.055306	0.138597	0.174014	147.496667	102.185566

Table 3.1. Result of γ changes.

3.2 ε을 변화시킨 결과

ε은 강화학습에서 기존 모델을 무시하고 새로운 모델을 시도해 볼 확률이다. ε-Greedy 알고리즘에서는 ε과 관련하여 세 가지 상수가 있다. ε의 초깃값, episode마다 ε이 감소하는 정도, 그리고 ε의 최솟값이다. ε의 초깃값은 가장 큰 1에 가까울수록 평균점수가 높았다. ε의 감소율은 감소 폭이 가장 작은 0.99와 0.95에서, 평균과 표준편차를 고려하면 두 경우의 효율이 가장 높았다. ε의 최솟값은 가장 작은 0.05일 때 평균점수가 가장 높았다.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
epsilon_1.0	0.778612	0.051981	0.119817	0.063019	119.050000	86.985789
Epsilon_0.9	0.777585	0.050366	0.189002	0.086592	111.843333	72.170069
Epsilon_0.8	0.798456	0.068715	0.176828	0.098381	103.323333	75.781300
epsilon_0.6	0.767573	0.059021	0.143407	0.062724	102.660000	69.553752
epsilon_0.4	0.813177	0.053539	0.312820	0.301447	95.090000	87.245412
epsilon_0.2	0.819327	0.057665	0.344840	0.273616	62.380000	67.504189
epsilon_0.0	0.900229	0.046692	1.421343	1.342371	9.120000	1.373172

Table 3.2.1. Result of ε initial value changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
epsilon_decay_0.99	0.778612	0.051981	0.119817	0.063019	119.050000	86.985789
epsilon_decay_0.95	0.802286	0.047251	0.109406	0.068076	133.56	121.689960
epsilon_decay_0.9	0.820484	0.051507	0.317273	0.188461	62.67	52.830494
epsilon_decay_0.85	0.805510	0.056113	0.169112	0.176496	112.87	92.440863
epsilon_decay_0.75	0.864223	0.057190	0.270465	0.186571	67.23	54.303748
epsilon_decay_0.65	0.853662	0.059845	0.270297	0.201109	97.93	124.815805
epsilon_decay_0.55	0.830453	0.058626	0.313720	0.305967	90.84	77.520413

Table 3.2.2. Result of ε decay rate changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
epsilon_min_0.05	0.788488	0.048596	0.134474	0.083725	129.643333	100.569724
epsilon_min_0.15	0.805698	0.063528	0.217292	0.129243	105.630000	103.206297
epsilon_min_0.2	0.826106	0.061435	0.181921	0.092349	99.200000	69.066634
epsilon_min_0.25	0.805447	0.057886	0.186540	0.060679	66.900000	44.462456
epsilon_min_0.3	0.830427	0.048879	0.225662	0.096740	69.223333	61.322536
epsilon_min_0.35	0.819767	0.068152	0.246172	0.079070	52.333333	43.847944
epsilon_min_0.4	0.803863	0.050052	0.196449	0.053446	50.973333	43.356729
epsilon_min_0.45	0.809163	0.045979	0.278367	0.124410	47.723333	38.771641
epsilon_min_0.5	0.813189	0.062591	0.254289	0.108935	38.673333	37.764268

Table 3.2.3. Result of ε minimum value changes.

3.3 신경망에 영향을 주는 상수를 변화시킨 결과

신경망의 Layer 수, 한 번에 투입할 데이터의 수 즉 Batch size, 신경망 한 Layer의 Node 수를 변화시켜 보았다. Layer의 수는 5층인 경우 평균점수가 가장 높았다. 아래 “표 3.3.1”에서 작성된 Layer의 수는 첫 번째 Layer를 제외한 Hidden Layer의 수만 센 것이다. Batch size는 32일 때 가장 평균점수가 높았다. Node의 수는 24개일 때 가장 평균점수가 높았다.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
layer_1new	0.807038	0.060173	0.286814	0.244141	90.46	82.557909
layer_2new	0.818920	0.060005	0.236090	0.211184	103.01	77.314746
layer_3new	0.759242	0.040012	0.144954	0.108510	107.73	72.089230
layer_4new	0.769768	0.056746	0.119795	0.077862	140.61	139.228438

Table 3.3.1. Result of number of layer changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
Batch_size_16	0.807411	0.059189	0.133981	0.110353	117.020000	88.760687
Batch_size_2	0.903925	0.077448	3085.628616	3976.631933	8.543333	1.433453
Batch_size_32	0.805331	0.059202	0.057808	0.087504	166.043333	147.476082
Batch_size_4	0.840136	0.092063	0.875014	1.141699	77.340000	92.163212
Batch_size_64	0.775285	0.046925	0.242609	0.114009	64.563333	58.747703
Batch_size_8	0.819753	0.078357	0.427599	0.581146	101.816667	100.962979

Table 3.3.2. Result of batch size changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
node_12_new	0.814661	0.059917	0.207998	0.150407	90.32	87.185650
node_18_new	0.832130	0.060477	0.426922	0.236584	51.36	51.035384
node_24_new	0.780848	0.041645	0.129672	0.084493	106.87	61.259882
node_30_new	0.843613	0.068860	0.281360	0.189190	88.20	103.756446
node_36_new	0.815208	0.061336	0.223438	0.193777	88.31	63.146448
node_42_new	0.835618	0.061699	0.392436	0.307736	67.16	54.818012
node_48_new	0.801991	0.048209	0.132085	0.057578	92.77	59.059268

Table 3.3.3. Result of number of node changes.

3.4 Learning Rate를 변화시킨 결과

신경망의 Learning rate는 Gradient descent algorithm에서 Loss 함수의 최솟값을 얼마의 간격으로 찾을지 정하는 상수이다. Learning Rate가 너무 작으면 제한된 시간 내에 최솟값에 다가갈 수 없고, 너무 크면 global minimum에 다가가지 못하고 발산해 버린다. Learning rate가 10-3일 때 평균점수가 가장 높았다.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
LR_1e-1	0.729657	0.040402	0.632140	0.151137	20.99	11.447703
LR_1e-2	0.776469	0.066734	0.260813	0.084821	59.25	46.911059
LR_1e-3	0.836933	0.061090	0.363349	0.313174	92.47	110.317946
LR_1e-4	0.913173	0.073137	1.898153	0.686824	9.47	3.192037
LR_5e-2	0.721561	0.039778	0.560106	0.154024	22.09	9.979073
LR_5e-3	0.780784	0.046367	0.313954	0.092486	44.35	34.309292
LR_5e-4	0.848611	0.053843	0.350963	0.263445	72.89	56.437912

Table 3.4. Result of learning rate changes.

3.5 신경망에 사용되는 함수들을 변화시킨 결과

신경망의 Loss function, Optimizer, Activation function을 변화시킨 결과이다. Loss function은 absolute error일 때, Optimizer는 Nadam(Dozat, 2016)일 때, Activation function은 ReLU(Vinod & Hinton, 2010)일 때가 가장 평균점수가 높았다. 아래 “표 3.5.1”에서 mse는 mean squared error, mae는 mean absolute error, mape는 mean absolute percentage error, msle는 mean squared logarithmic error를 의미한다.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
history_mae	0.847372	0.079866	0.803108	0.348869	26.46	27.331454
history_mape	0.846321	0.070858	6.562410	4.368073	47.42	47.286823
history_mse	0.804244	0.048988	0.712625	0.482460	152.96	73.152159
history_msle	0.798530	0.062681	0.032420	0.071997	137.79	92.748185
history_squared_error	0.808593	0.054112	0.264715	0.211058	73.47	48.365991

Table 3.5.1. Result of loss function changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
Adadelta	0.780203	0.069258	0.700990	0.099363	8.920000	1.653360
Adagrad	0.809375	0.086844	1.500182	0.686121	8.970000	2.447264
Adam	0.790213	0.047936	0.162387	0.121094	105.100000	75.940964
Adamax	0.878877	0.065925	0.626878	0.495371	117.460000	150.423364
Nadam	0.809980	0.056819	0.170658	0.218150	153.920000	149.115527
RMSprop	0.799705	0.081339	0.278527	0.211247	66.756667	69.229503
SGD	0.862381	0.066499	0.446385	0.324181	46.753333	31.576560

Table 3.5.2. Result of optimizer changes.

	Mean Accuracy	Acc Std	Mean Loss	Loss Std	Mean Score	Score Std
af_linear	0.886484	0.092762	1.281619	0.724548	10.58	6.473299
af_relu	0.793896	0.059144	0.086322	0.062721	178.84	145.358503
af_selu	0.818190	0.043959	0.149541	0.107850	114.08	63.722473
af_sigmoid	0.807502	0.064091	0.407049	0.270242	62.01	63.799921
af_softplus	0.810291	0.050347	0.201380	0.147058	120.33	92.398491
af_tanh	0.783207	0.041404	0.125216	0.057162	114.47	77.001747

Table 3.5.3. Result of activation function changes.

3.6 결과 정리 및 해석

실험에서 변화시킨 11가지의 Hyperparameter들이 각각 최고의 평균점수와 최악의 평균점수를 보였을 때의 결과를 하나의 표로 정리하였다. 점수의 표준편차 또한 고려의 대상이 되었다. 하지만 모든 모델이 초기에는 매우 낮은 점수로 시작하기 때문에, 점수가 높을수록 표준편차가 커지는 현상이 발생한다. 그래서 평균점수가 유의미하게 차이나는 경우 평균점수를 이용해 순위를 판단하였다. 평균점수로 순위를 가리는 것이 무의미할 정도로 아주 비슷한 경우에만 표준편차를 이용해 순위를 판단하였다.

Best	Hyperparameter Name	Worst
0.95	Gamma	0.5
1.0	Epsilon	0.0
0.05	Epsilon_min	0.5
0.95	Epsilon_decay	0.75
4	The number of layer	2
32	Batch_size	2
24	The number of node	18
ReLU	활성화 함수	Linear
Squared error	Loss 함수	Absolute percentage error
Nadam	Optimizer 함수	Adadelta
0.001	Learning rate	0.0001

Table 3.6. Result of every hyperparameters what we researched.

γ는 작을수록 효율적인 경로를 택할 것이라고 예상되었다. 하지만 작아질수록 평균점수가 낮아져 실험한 가장 작은 값이었던 0.5에서 최악의 결과를 냈다. 실험한 가장 큰 값이었던 0.95는 코드 작성자가 default 값으로 정해둔 값이었고, 최고의 결과를 냈다. 본 현상이 나타나는 이유는 비효율적인 경로에 너무 낮은 점수를 줌으로 인해 '비효율적이더라도 성공할 수 있는 경로'를 탐색하지 않는 경우가 나타나는 것으로 해석할 수 있다. ε은 초기에는 크고, 후기에는 작은 것이 직관적으로 합리적이다. 또한, ε이 천천히 감소하여 초기에 충분히 새로운 경로를 탐색해 볼 가능성을 주는 것이 높은 점수를 받는 방법이라고 해석된다. Layer 수와 Node 수는 너무 작으면 신경망이 얕아 합리적인 판단을 내리기 어렵다. 반면 너무 크면 신경망이 깊어 각 Node의 weight를 튜닝하기 어려워지므로 적당한 값을 찾는 게 중요하다. Cartpole 문제를 해결하는 본 DDQN에서는 Layer는 5개, Node는 24개 일 때 효과적인 것으로 해석된다. Batch size는 원래 무수히 많이 학습을 진행하면 점수에 영향을 주지 않아야 한다. 하지만 Batch size가 너무 크거나 작을 때 점수가 하락하는 현상은, 우리가 100회의 학습밖에 시행하지 않기 때문으로 예상된다. Learning rate는 상술했듯 너무 크면 발산하고 너무 작으면 적절한 시간 내에 loss 함수의 local maximum에 도달할 수 없다. 이 실험에서는 0.001이 균형 있는 적절한 값인 것으로 해석된다. Loss, Optimizer, Activation 함수들은 프로그램의 목적마다 다른 것을 사용해야 한다. 우리의 문제에서는 세 함수 모두 단순한 함수보다는 최근에 사용되기 시작한 함수들을 사용하면 더 좋은 점수를 받는 것으로 나타났다. 물론 Activation function에서는 고전적인 ReLU에서 가장 좋은 점수를 받았다.

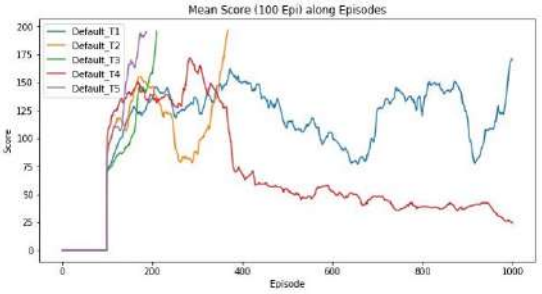
3.7 1000회 실행해 본 결과

Cartpole simulation의 규칙은 최근 100번 동안 시도의 평균점수가 195점 이상일 때 해당 모델이 성공했다

고 판단한다. 이 규칙을 따라 우리는 두 가지 모델을 1000회 실행해 보았다. 우선 오픈 소스 코드의 기본 설정으로 진행했다. 또 우리가 최적이라고 밝혀낸 Hyperparameter들을 각 자리에 대입한 모델을 실행시켰다. 이 두 모델을 편의상 순서대로 default 모델과 best 모델이라고 부르겠다. 그 결과 Cartpole simulation의 규칙에 따라 pass 되기 위한 평균 에피소드 수는 default 모델에서 256회였으며, 5회 중 2회는 1000회가 될 때까지 pass 되지 못하였다. 그리하여 종료될 때까지 필요한 평균 에피소드 수는 549.6회이다. 반면 best 모델은 5회 모두 215번의 에피소드 안에 pass 되었으며, 평균 에피소드 수는 151.8회였다. 이를 통해 우리가 Hyperparameter를 튜닝한 모델이 default 모델보다 성능이 좋은 것을 알 수 있었다.

	Final Episode	Mean Score (Last 100 Epi)	Score Std (Last 100 Epi)
Default_T1	1000	171.00	171.00
Default_T2	369	196.90	196.90
Default_T3	211	195.84	195.84
Default_T4	1000	24.41	24.41
Default_T5	188	195.02	195.02

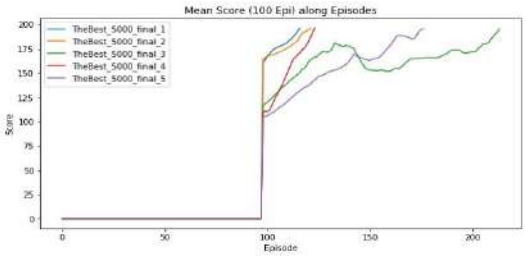
(a)



(b)

	Final Episode	Mean Score (100 Epi)	Score Std (100 Epi)
TheBest_5000_final_1	118	196.08	196.08
TheBest_5000_final_2	123	196.47	196.47
TheBest_5000_final_3	215	195.61	195.61
TheBest_5000_final_4	125	196.62	196.62
TheBest_5000_final_5	178	195.48	195.48

(c)



(d)

Figure 3.7. (a) Result of 1000 times running with pass option of default model. (b) Graph of 1000 times running with pass option of default model. (c) Result of 1000 times running with pass option of best model. (d) Graph of 1000 times running with pass option of best model.

4. 결론

제어 문제를 해결하는 방법으로는 기계공학적 접근과 컴퓨터공학적 접근이 있을 것이다. 기계공학적 접근은 물리학 법칙과 미분방정식 계산을 통해 원리를 밝혀 최적의 방법을 찾는 것이다. 컴퓨터공학적 접근은 넓은 범위에 걸친 무작위 대입과 반복 연산을 통해 경험적으로 최적인 방법을 찾는 것이다. 우리는 이 연구에서 CartPole 환경 속에서의 제어 문제를 컴퓨터공학적으로 접근하여 Hyperparameter를 넓은 범위에서 변화시키며 가장 좋은 결과를 내는 값들을 찾아내었다.

우리가 실험한 결과에서 최적의 Hyperparameter값을 정리하자면 gamma는 0.95, epsilon은 1.0,

epsilon_min은 0.05, epsilon_decay는 0.95, the number of layer는 4, batch_size는 32, node의 수로는 24개, 활성화 함수로는 ReLU함수, Loss 함수는 Squared error, optimizer 함수로는 Nadam, learning rate는 0.001로 진행하는 것이 가장 학습 효율이 높았다고 할 수 있다.

우리가 탐구한 환경에서는 이러한 Hyperparameter값들이 가장 좋았지만, 현실에서의 물리적 모델에서는 다른 결과를 보일 것으로 예상된다. 실제 물리적 모델에서는 시뮬레이션과 달리 중력가속도 값도 9.8과 미세하게 다르고, 공기 저항과 마찰도 존재한다. 제어와 측정 방식도 시뮬레이션과 현실에서는 크게 다르다. CartPole Simulation 환경에서는 좌우에 일정한 힘을 가하여 cart를 조향했지만, 현실의 cart는 벨트를 통해 모터의 회전각속도로 제어된다. CartPole Simulation 환경에서는 pole 끝의 선속도와 pole의 각도를 측정하지만 현실에서는 cart에 달린 서보모터로 pole 각도의 변화량밖에 측정하지 못한다.

앞으로 우리가 더 해야 할 일들이 많다. 첫째로, 인간을 모방하려는 시도를 계속하는 것이다. 예를 들어, 사람이 백색소음을 들으며 공부하면 집중이 잘 되는 것처럼 활성화 함수에 random한 noise를 첨가해 보자는 교수님의 아이디어가 있었지만 연구기간 동안 실현해보지 못하였다. 둘째로, 더 많은 episode에서 각 hyperparameter들이 어떤 performance를 보이는지 관찰하는 것이다. 우리 연구에서는 시간 문제로 각 hyperparameter들을 100 episode 씩 실행하면서 결과를 냈다. 만약 episode의 수가 충분히 커진다면 다른 결과가 나올 수 있다. 셋째로, 연구실에서 제작 중이었던 물리적 Cartpole 모델을 실제로 완성한 뒤 시뮬레이션 환경과 현실 환경을 비교하는 것이다. 이 두 환경은 큰 차이가 날 것이고, 특히 현실에서는 더 많은 변수를 고려해야 할 것이다.

참고문헌

keon, . . (2017,). deep-q-learning

Vinod, N. ., & Hinton, G. . (2010). Rectified linear units improve restricted boltzmann machines.. Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807-814).:

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2 ed.,). (MIT Press, Cambridge, MA:)

Silver, D. ., Huang, A. ., Maddison, C. ., Guez, A. ., & Sifre, L. . (2016). Mastering the game of Go with deep neural networks and tree search.. nature, 529(7587), 484.:

Sharma, S. . (Linear Regression).

Minh, V. ., Kavukcuoglu, K. ., Silver, D. ., & Graves, A. . (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602..

Kim, H. . (모두를 위한 머신러닝/딥러닝 강의).

Hado, H. V., Silver, D. ., & Guez, D. . (2016). Deep reinforcement learning with double q-learning.. Thirtieth AAAI conference on artificial intelligence.:

Dozat, T. . (2016). Incorporating nesterov momentum into Adam. International Conference on Learning Representations (ICLR).

Anschel, O. ., Baram, N. ., & Shimkin, N. . (2016). Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. 34th International Conference on Machine Learning-Volume 70 (pp. 176-185). JMLR. org.:

(2019,). OpenAI GYM

(2019,). Keras

[연구지도후기 – GIST 융합기술학제학부 류제하 교수]

제목 : "호기심을 충족시키기 위한 기회로써의 연구"

그동안 과학고 학생들과 여러 번의 R&E 및 유사한 프로그램을 하면서 아쉬웠던 점은 학생들이 연구를 재미가 있고 호기심을 충족시키기 위한 기회로 생각하기 보다는 스펙을 쌓기 위한 노력의 일환으로 하지 않았나 하는 것이었는 데 이번 학생들은 주제가 인공지능이라 그런지 많은 호기심과 미래에 대한 상상을 할 수 있는 기회가 된 것 같아 나름 재미가 있었다. 인공지능 중 강화학습은 최근의 심층학습의 발전과 더불어 많은 주목을 받는 기술이고 사람의 인생에서 배우는 과정과 많은 면에서 비슷하기 때문에 나 또한 많은 호기심으로 연구를 시작하고 있고 아주 많은 질문들이 생기는 분야이다. 여러 어려운 이론을 숙지하기에는 너무나 짧은 기간이라 마침 김성훈 교수의 아주 잘 짜여진 실습코드가 있어 간단한 이론의 이해를 바탕으로 간단한 물리시스템인 cartpole제어 문제를 가지고 시뮬레이션 기반으로 여러 가지를 시도하였고 마침 조교 및 학생들이 열심히 연구한 덕에 나름 의미있는 결과를 낼 수 있었다고 생각한다. 아쉽게도 실제 하드웨어가 제때 제작되지 못하여 실제 로봇시스템에의 적용은 불발되었지만 학생들이 이 분야 연구에 대한 기초 이론 및 속성에 대해 충분히 이해할 수 있는 계기는 되었다고 생각한다. 어릴 때 꿈을 가지는 기회가 되었으면 하며 미래를 스스로 개척할 수 있는 훌륭한 동량이 되기를 기원한다.

[연구지도후기 – GIST 융합기술학제학부 정대현 조교]

제목 : "이 경험이 앞으로의 길을 찾기 위해 소중히 쓰일 수 있기를"

대학 입시 준비와 일반 고등학교보다 더 힘든 학교 일정에도 불구하고 새로운 것에 대한 배움과 도전의 자세로 찾아온 학생들이었다. 과학고나 영재고에 다니고 있는, 조금 특별하다고 생각될 수 있는 방향으로 걷고 있는 학생들이지만 좋아하는 아이돌 이야기, 게임, 학교생활의 고됨 등에 대해 이야기하는 그들은 어디에나 있는 고등학생들이었다. 실험 도중 아직 부족한 코딩 실력으로 필요한 기능이나 오류를 고쳐가는 그들의 모습은 이제 막 대학원에 입학했을 때의 나를 보는 것 같았다. 그들에게 더 여러 가지 경험을 시켜주기 위해 나 이외에도 연구실 학생들이 준비해주었던 시연들에 우리의 기대보다 더욱 신기해하며 관심을 보여주어서 다행이었다. 아직 우리가 부족한 탓에 연구에 대해 더욱 깊은 경험을 시켜주지 못해 약간의 아쉬움이 남는다. 이번과 같은 경험이, 만약 연구자가 꿈이었던 학생이 있다면, 그곳으로 나아가는데 도움이 되었으면 한다.

[연구참여후기 – 대전동신과학고등학교 2학년 이창섭]

제목 : "혼자서는 1년을 공부해도 배울 수 없는 것을 2주 만에 배울 수 있었다"

Pre-URP를 시작하며 정말 하고 싶어서 새로운 것을 배울 수 있는 주제를 선택했다. 학교에서는 화학 전공으로 살고 있었으나, 예전부터 코딩도 할 줄 알았고 공학동아리 회장이기도 했다. 하지만 높은 수준의 공학 연구는 경험해보지 못했고, 특히 인공지능처럼 전문적인 분야는 해 볼 엄두도 내지 못했다.

Pre-URP 주제를 선택하던 날은 늦은 새벽이었다. 기숙사에서 몰래 노트북을 켜 놓고 꾸벅꾸벅 졸면서 주제를 고르고 신청서와 자기소개를 썼다. 졸려서 그랬는지, 인공지능이라는 주제를 용기 있게 선택할 수 있었던 것 같다. 그리고 한 3일은 화학 분야 주제를 고를 걸 하고 후회했다. 작년에 Pre-URP를 다녀온 화학 분야 선배들이 어느 연구팀이 좋더라 하고 얘기해서, 지금이라도 주제를 바꿀까 하고 고민했다. 하지만 새로운 도전 앞에서 망설이지 말아야겠다는 다짐 앞에, 주제를 바꾸지 않고 인공지능을 공부해보기로 했다.

사전 과제에서 김성훈 교수님의 '모두를 위한 딥러닝' 강의는 쉽지 않았다. 텐서플로우 코드를 읽기도 쉽지 않았다. 하지만 이 강의는 정말 큰 자산이 되었고, 요즘도 혼자 실습을 따라 해보곤 한다. 집중 연구에 가서도 그랬다. 윈도우 세대에게 Command Line Interface는 정말 낯설었다. 서버의 Docker에 접속하기 위한 리눅스 명령어도 외우지 못해 메모장에 써 두었다. 하지만 정말로 확실한 건, 혼자였다면 1년을 했어도 못 배웠을 것을, 아니 시작할 엄두도 내지 못했을 것을 2주 만에 배울 수 있는 기회였다는 것이다. 앞으로 계속 무언가를 배우는 사람이 될 텐데, 이번 Pre-URP처럼 좋은 기회가 또 주어지기를 바란다.
내년에 Pre-URP를 할 기회가 올 후배들에게도 꼭 도전하라고 강력히 추천하고 싶다. 고등학교 3년 중 어쩌면 가장 좋은 기회가 될 이 프로그램을 꼭 참여했으면 좋겠다.

[연구참여후기 - 경기과학고등학교 1학년 임형민]

제목 : "인공지능 개발의 초석을 쌓았다."

인공지능은 흥미로운 주제다. 사람의 사고를 모방하는 인공지능의 작동을 보면, 누구든지 이에 대한 자연스러운 의문이 들기 마련이다. 나는 이에 여러 의구심이 들어서, 과연 인공지능이 어떻게 작동하고, 발전해왔고, 만들 수 있는지를 공부하고 싶었다. 때마침 Pre-URP 라는 흥미로운 프로그램이 있어 신청하게 되었고, 인공지능을 활용한 주제라는 생각에 신청하게 되었다. 사전연구에서 배운 김훈의 모두를 위한 딥러닝 강의를 들으면서 딥러닝의 기초에 대해 연구해 보게 되었다. 이번에 Pre-URP 에서 배운 Cart-Pole 모델을 조정하면서 여러 인공지능을 움직이는 요소와 그 학습방법을 익히게 된 유익한 시간이었던 것 같다.

[연구참여후기 - 경산과학고등학교 2학년 김우진]

제목 : "전공 선택에 큰 영향을 끼칠 것"

과학고 2학년에 재학하며 우연히 Pre-URP를 신청할 수 있는 기회가 있어서 신청하게 되었다. 솔직히 다른 친구들은 열심히 공부하는 시간에 Pre-URP를 간다는 사실이 많이 신경 쓰였다. 하지만 GIST의 연구 시설과 연구실 조교님들이 너무 좋아서 Pre-URP를 신청하길 잘했다는 생각이 들었다.
주제 선정 시 물리 주제를 선택할까 정보 주제를 선택할까 고민을 했었는데 정보 주제를 선택하는 것도 새롭고 좋은 경험이 될 것 같아서 이 주제를 선택했다. 주제를 선택하고 친구들에게 말하니까 친구들이 너 잘하는 물리하지 왜 정보 했냐고 그랬지만 나는 주제 선정해 매우 만족했던 것 같다.
딥러닝은 말로만 들어봤지 실제로 접할 기회는 적었는데 이번 기회에 접해보며 딥러닝의 기초적인 개념과 코드를 이해 할 수 있었다. 사전 과제를 진행하면서 나는 딥러닝에 대해 아는 게 없으니까 연구에 가서 팀원들에게 짐이 될까 봐 학교 도서관에서 책도 빌려보고 여러 블로그 등을 찾아보며 열심히 공부했던 것 같다. Pre-URP가서 코드를 돌려보고 분석하고 자율 자동차의 데이터 수집을 했던 게 가장 기억에 남는 것 같다. Pre-URP전에도 컴퓨터 공학에 관심이 있었는데 이번 기회로 컴퓨터 공학에 더 많은 관심을 가지게 되었다. 앞으로 만약 컴퓨터 공학을 전공한다면 이 경험이 전공 선택에 큰 영향을 끼칠 것이라고 생각한다.
연구 이외에 GIST에서의 생활도 기억에 많이 남는데 특히 기숙사에서 팀원 준서와 즐거운 시간을 보낸 것과 매일 팀원들과 연구실 조교님과 점심 먹으러 간 것이 기억에 남는다.
앞으로도 이런 기회가 많은 친구에게 주어지면 좋겠고 오래 지속되었으면 좋겠다.

[연구참여후기 – 인천과학예술영재학교 1학년 김준서]

제목 : "인공지능에 대해 배울 기회가 생겼다"

영재학교 1학년에 재학하며 학교에서 Pre-URP 프로그램에 대한 설명을 듣고, 현장연구 학점을 대체할 수 있으면서 2주간 과학기술대학교에서 하고 싶은 연구를 할 수 있다는 말을 듣고 이 프로그램을 신청하게 되었다. 평소 인공지능에도 관심이 많고, 장래 희망이 인공지능공학자이며, 학교에서도 컴퓨터 프로그래밍 동아리에 소속되어 있는 나로써는 인공지능을 대학교에서 배운다는 것은 흥미롭게 다가왔다. 리눅스를 많이 사용하지 않는 요즈음 시대에 리눅스를 쓰며, 주피터 노트북을 통해 딥러닝을 진행하는 과정은 어렵기도 하고 흥미로웠다. 원래는 자율주행차 개발에 참여해 볼 수 있다는 것 때문에 이 주제를 신청해서 처음 연구실에 왔을 때에 연구 주제가 Cartpole Simulation으로 바뀌었다는 말을 듣고 아쉽기도 했지만, 우리가 연구실에서 이루어낸 많은 작업들로 나는 뿌듯함을 느끼며 Pre-URP를 끝마칠 수 있었다. 연구실에서는 최종 발표를 준비하며 딥러닝도 하고 많이 힘든 일도 있었지만, 기숙사에 와서는 정말 좋은 경험을 했다고 생각한다. 고등학교와는 확연히 다른 대학교 기숙사를 경험하며, 룸메이트와 좋은 추억을 쌓을 수 있었다. 매일 연수구실 조교님과 점심 먹는 것도 재미있었다. Pre-URP 프로그램을 선택한 것을 후회하지는 않을 것 같다. 꼭 기회가 된다면 후배에게 추천해 주고 싶다.