6장. 복합 출력의 처리 방법 : 오피스31 다차원 분류 신경망

1. 개요

이번 과제는 오피스31 데이터셋을 다층 퍼셉트론 신경망에 학습하여 사진의 도메인, 품목의 종류를 예측하는 실험을 진행하였다. 이를 위해 최종 결과를 복합 출력으로 다루었으며, 성능의 개선을 위하여 아담 알고리즘을 활용하였다. 그리고 이에 대한 결과들을 살펴보았다.

2. 이론

<데이터 셋>

이번 과제에서 다루는 데이터셋은 "오피스31" 이라는 데이터 셋이다. 이는 컴퓨터 비전 분야에서 전이학습 연구용으로 구축된 표준 벤치마크 데이터 셋이다. 사무용품 이미지 4652 장으로 구성되어 있으며, 각 이미지 데이터의 구성은 다음의 2가지가 함께 레이블링 되어있다.

① 도메인 : 이미지 데이터의 사진 수집 방법에 따른 분류로 amazon, dSLR, webcam으로 구분된다.

② 품목 : 사진 속에 담긴 내용의 종류로써 총 31가지 품목이 있다. 31가지 품목의 예시로는 배낭, 자전거, 자전거용 헬멧, 파일캐비닛, 헤드폰, 키보드 등이 있다.

< 전이학습 vs. 다차원 분류 >

< 질문 2. 전이학습이란 무엇일까? 어떤 경우에 필요할까? >

① 전이학습(transfer learning) : 한 도메인에서 학습시킨 결과를 다른 도메인에 활용하여 학습 효과를 높이는 학습 기법으로 비교적 높은 정확도를 단 시간내에 달성할 수 있다. 이미지 분류 문제에서는 CNN에서도 사용되었다.

ex) 아마존 도메인에 속한 데이터들에 품목 레이블 정보가 모두 태깅되어 있어서 이를 이용해 이미 만족할 만한 성능으로 학습시킨 모델이 있다고 가정하자.
이에 반해, dSLR 도메인의 데이터들은 품목 레이블 정보가 없거나 아주 적은 양만 존재한다 하자. 이럴 경우 이미 확보된 아마존 도메인 모델의 학습 정보를 잘 활용하면 dSLR 도메인에서 좀 더 나은 성능을 얻을 수 있다. 이런 구체적인 방법을 탐구하는 연구 분야가 전이학습이다.

[전이학습이 주목받는 이유 및 필요한 경우]
수집된 데이터에 정답을 표시하는 레이블링 작업에는 많은 경우 데이터 수집 이상의 많은 인력과 시간이 들어간다. 그렇다고 해서 레이블링 없이 신경망을 학습시킬 수 있는 방법은 마땅치 않다고 볼 수 있다. 이러한 상황에서 새로운 도메인에서 만이라도 다른 도메인에서 학습된 내용을 이용하여 레이블링 작업량을 줄여줄 수 있다면 큰 보탬이 될 것이다. 이러한 이유로 전이학습 연구가 주목받고 있다.

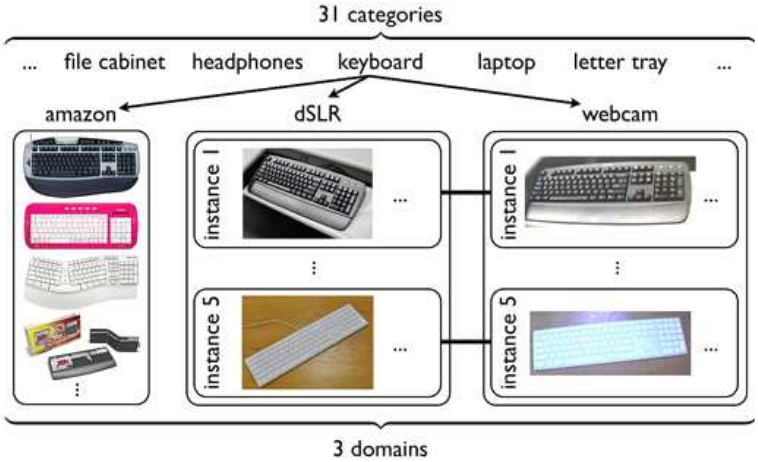
즉, 딥러닝 기반의 모델을 제대로 훈련시키기 위해서는 많은 수의 데이터가 필요한데, 충분히 큰 데이터셋을 얻는 것은 쉽지가 않다. 이러한 데이터 셋을 만들기 위해서는 많은 비용과 시간이 소모되기 때문이다. 이러한 경우에 우리는 전이학습이란 것을 사용한다. 덧붙여 전이학습이 등장한 배경 또한, 이렇게 많은 비용과 시간을 투자할 여건이 잘 구성되지 않기 때문에 이러한 현실적 어려움에 부딪혀 등장하게 되었다.

+) 실제로 전이학습을 인간의 학습에 빗대어 보면 우리는 좀 더 쉽게 이해할 수 있다. 인간의 경우 새로운 것을 학습할 시 처음부터 학습을 진행하는 것이아니라, 기존의 경험을 토대로 학습을 진행한다. 이러한 모습처럼 이전 지식을 활용하여 새로운 지식을 학습하는 것이 전이학습이라고 할 수 있다.

오피스 31 데이터셋이 주로 활용되는 연구 분야는 전이학습(transfer learning)이다. 그러나 우리는 여기에서 각 데이터 이미지의 도메인과 품목을 동시에 판별해주는 딥러닝 모델을 통해 실험을 진행할 것이다.

② 다차원 분류 : 여러 차원에서의 분류를 한꺼번에 수행하고 그에 대한 결과를 동시에 보여주는 것을 뜻한다.

이번 과제에서는 도메인과 품목을 동시에 판별해주는 이중 선택 분류라 할 수 있다. 다차원 분류 대신 차원 축소를 이용한 단순 선택 분류도 가능하다. (아마존, 배낭), (dSLR, 배낭) , ... 이런식으로 도메인과 품목을 (도메인, 품목) 순서쌍을 단위로 카테코리로 설정하면 한 선택 분류가 가능하다. 이렇게 할 경우 3가지 도메인과 31가지 품목이므로 총 93가지 경우의 순서쌍 중 1개를 고르는 선택 분류 문제가 되는 것이다. 그러나 이렇게 문제를 해결할 경우 출력이 과도하게 증가할 뿐만 아니라 도메인의 특성, 품목의 특성 들을 따로 포착하기 어려워져 학습 성과가 좋지 않게 나올 가능성이 크다. 또한 2개의 선택 분류 출력에 대한 미봉책이 될지언정 회귀 분석, 이진 판단 출력들이 섞이거나 출력 항목 수가 크게 증가하는 경우를 생각하면 근본적인 대책은 될 수 없다.



< 오피스31 다차원 분류 설명을 위한 그림 >
출처 : 파이썬 날코딩으로 알고 짜는 딥러닝 PPT

< 딥러닝에서의 복합 출력의 학습법 - PPT 참고 >
같은 퍼셉트론이라도 학습 과정에 따라 역할이 많이 달라진다. 이 때 달라지는 것은 퍼셉트론의 구조가 아닌 가중치와 편향 같은 파라미터값의 구성뿐이다. 즉, 똑같은 퍼셉트론이라도 학습을 어떻게 시키느냐에 따라 역할이 달라지는 것이다. 그러면 우리는 동일한 구조의 신경망을 다양한 용도에 사용하기 위해서는 어떻게 해야할까?

우리가 동일한 구조의 신경망을 다양한 용도에 사용하기 위해서는 후처리 과정에서의 처리 방법만 변경해주면 된다. 후처리 과정 역시 신경망에서와 마찬가지로 순전파와 역전파의 두 부분으로 구성되며, 이를 복합 출력에 맞게 바꾸어주고 신경망 회로에서는 알맞은 크기의 출력 벡터를 만들어내기만 하면 된다.

[후처리 과정의 구성]
① 순전파 : 신경망 순전파 처리 결과로부터 손실 함수 계산
② 역전파 : 신경망 출력 성분별 손실 기울기를 구해 신경망 역전파에 전달

오피스31 데이터셋에 대한 다차원 분류의 경우 신경망은 픽셀 수만큼의 입력 벡터 크기를 가지며 출력 벡터 크기는 도메인 판별에 사용될 성분 3가지와 품목 판별에 사용될 성분 31가지를 더한 34가지 성분을 가지면 된다. 즉, 출력 벡터 크기만 34로 변경하면 신경망의 내부 구조에 대해서는 변경할 게 없는 것이다. 그러면 1개의 신경망 출력을 복합 출력으로 처리하는 대신 필요한 출력별로 별도의 신경망을 구성하는 것에 대해 생각해보자.

출력 계층을 구성하는 퍼셉트론은 맡은 임무에 따라 역할이 확실히 다르기 때문에 출력 계층만 보면 신경망의 분리에 대해 큰 차이가 없다. 하지만 은닉 계층에 속한 퍼셉트론의 경우 모든 출력 계층 퍼셉트론에 영향을 주고 모든 출력 계층 퍼셉트론으로부터 역전파 피드백을 받으므로 복합 출력으로 처리하는 경우와 신경망을 따로 구성하는 경우에 큰 차이가 발생한다. 그런데 이들 출력이 서로 연관되어 있다 보니 이들 사이에 공통적인 특성이 존재할 수 도 있다.

이를 위해 각각의 출력을 위한 별도의 은닉 계층을 따로 학습시키는 것보다 모든 출력에 연결된 하나의 은닉계층이

이들의 공통 특성을 포착하도록 학습시키는 것이 계산량 절감, 성능 향상으로 이어진다. 즉, 출력별로 별도의 신경망을 구성하는 것보다는 1개의 신경망 출력을 복합 출력으로 처리하는 방법이 좀 더 바람직한 것이다.

[복합 출력 처리를 위한 신경망 내부의 처리 - 정리]
① 출력 요소별 크기를 합한 크기의 출력 벡터를 생성하도록 수정한다.
② 출력 계층 퍼셉트론 : 출력 요소 별로 역할이 분리된다.
③ 은닉 계층 퍼셉트론 : 전체 출력에 연결되어 모두에 영향을 미친다.

[복합 출력을 위한 후처리 방법]
① 순전파에서의 손실 함수 계산 방법
- 신경망 순전파 처리 결과를 $output_1$, $output_2$로 분할할 경우
  도메인 관련 정답 y_1이 신경망 출력 $output_1$에서 손실 L_1이 구해지고,
  품목 관련 정답 y_2, 신경망 출력 $output_2$에서 손실 L_2가 구해지면 전체적인 손실 값은 $L_1 + L_2$로 계산한다.
  (각 성분에 대한 손실 함수 계산에는 기존의 처리 방법이 활용 가능하다.)

② 역전파에서의 손실 함수 계산 방법
- 편미분의 특성(다른 변수의 영향을 받지 않는 점) 과 L_1, L_2는 서로에게 상수이다.
  L_1에 대한 후처리 역전파 처리 과정을 통해 $output_1$의 손실 기울기를 계산한다.
  L_2에 대한 후처리 역전파 처리 과정을 통해 $output_2$의 손실 기울기를 계산한다.
  2 가지 손실 기울기를 결합하여 신경망 역전파 처리에 전달하며, 역전파를 시작한다.
  각 성분에 대한 손실 기울기 계산에는 기존의 처리 방법을 활용 가능하다.

< 복합 출력을 위한 클래스 설계 >
지금까지 살펴본 예제는 1가지 출력만 냈다. 그러나 이제는 복합 출력이 필요하며, 이는 MlpDataset 클래스와 Dataset 클래스의 매우 간단한 확장만으로 복합 출력이 처리 가능하다. 복합 출력에서 달라지는 부분은 신경망 처리가 아니라 신경망 출력에 대한 후처리 과정, 처리 결과를 보여주는 시각화 부분과 관련된 부분들이며, 다음과 같이 확장될 수 있다.

[모델 클래스]
- MlpModel 클래스를 수정 없이 이용하는 것으로 충분하며, 신경망 처리에서 달라지는 부분은 출력 벡터 크기 뿐이다.
  (데이터 셋 객체를 통해 알아내므로 모델 코드의 수정은 불필요하다.)

[Dataset의 파생 클래스를 선언]
- 복합 출력을 위한 후처리 과정 관련 method를 재정의 해야 한다.
- 복합 출력을 위한 시각화 method를 정의해야 한다.

[AdamModel]
- MlpModel 파생 클래스

< 아담 알고리즘 (Adaptive moments Algorithm) >
아담 알고리즘은 파라미터에 적용되는 실질적인 합습률을 개별 파라미터 별로 동적으로 조절하여 경사하강법의 동작을 보완하고, 학습 품질을 높여주는 방법이다.

아담 알고리즘에서는 "모멘텀(Momentum)"이라는 개념을 도입해 처리 과정을 보완한다. 여기서의 모멘텀은 최근의 파라미터값의 변화 추세로 나타내는 정보로 생각하면 된다. 아담 알고리즘은 이러한 모멘텀 정보와 2차 모멘텀 정보를 활용하여 경사하강법의 동작을 보완한다.

[아담 알고리즘 주의할 점]
아담 알고리즘은 모멘텀 정보, 2차 모멘텀 정보가 전체적인 신경망 차원에서 관리되는 단일 값이 아니다. 이 값들은 개별 파라미터 수준에서 따로 계산되고 관리된다. 즉, 아담 알고리즘을 이용할 경우 파라미터 1개마다 모멘텀 정보와 2차 모멘텀 정보가 따라붙게 되어 파라미터 관리에 필요한 메모리 소비량이 약 3배정도로 늘어난다. 미니배치 데이터를 처리해 학습이 이루어질 때마다 파라미터 값은 물론 이들 모멘텀 정보 역시 파라미터와 마찬가지로 텐서 차원에서 계산되고 관리되기 때문에 프로그램이 생각만큼 많이 복잡하지는 않지만 전체적으로 메모리 소비와 계산량이 증가하는 것은 피할 수 없다.

학습률은 학습 결과에 매우 큰 영향을 미치는 하이퍼파라미터지만 적절한 값을 찾아내는 것은 많은 시행착오를 요구하게 된다. 아담 알고리즘은 각각의 파라미터별로 실질적인 학습률을 보정해 적용함으로써 일괄적인 학습률을 적용하여 나오는 품질 저하를 막아준다. 하지만 아담 알고리즘이 적절한 학습률을 찾는 노력을 완전히 없애주는 것은 아니다. 아담 알고리즘의 구현 내용을 살펴보면 결국 파라미터를 갱신하는 마지막 순간에 학습률이 이용된다. 아담 알고리즘은 학습률 같은 하이퍼 파라미터에값에 강건한 장점이 존재하지만, 알맞은 학습률을 찾는 노력이 완전히 없어지지는 않는다.

[아담 알고리즘 - PPT 참고]
- 모멘텀 : 관성력으로 번역이 되나 여기서는 파라미터 값의 최근 변화추세를 나타내는 정보로 생각하면 된다.
  아담 : Adaptive moments의 약자
- 아담 알고리즘
  - 모멘텀과 2차 모멘텀 개념을 도입해 경사하강법 동작을 보완한다.
  - 파리미터에 적용되는 실질적 학습률을 각각의 개별 파라미터 별로 동적으로 조절한다.
  - 경사하강법의 동작을 보완하고 학습 품질을 높여주는 방법이다.
- 역전파 처리 과정 보완
  - 경사하강법은 손실 기울기 X 학습률을 각각의 파라미터에서 감산하는 방식이다.
  - 아담 알고리즘은 손실 기울기 대신 모멘텀 정보를 이용해 보정된 값을 이용한다.
  - 적절한 학습률을 찾는 부담을 줄여주면서 학습 성능을 향상시킨다.
- 모멘텀 정보와 2차 모멘텀 정보
  - 전체 신경망 차원의 값이 아니라 파라미터 별로 따로 계산되고 관리되는 값들이다.
  - 파라미터 별로 3가지 값이 존재해야 하므로 대략 3배의 메모리가 더필요하다.
  - 텐서 수준에서 계산되고 관리된다. (프로그램은 생각보다 복잡해지지 않는다.)
  - 메모리 소비와 계산량의 증가는 불가피하다.

Algorithm 8.7 The Adam algorithm

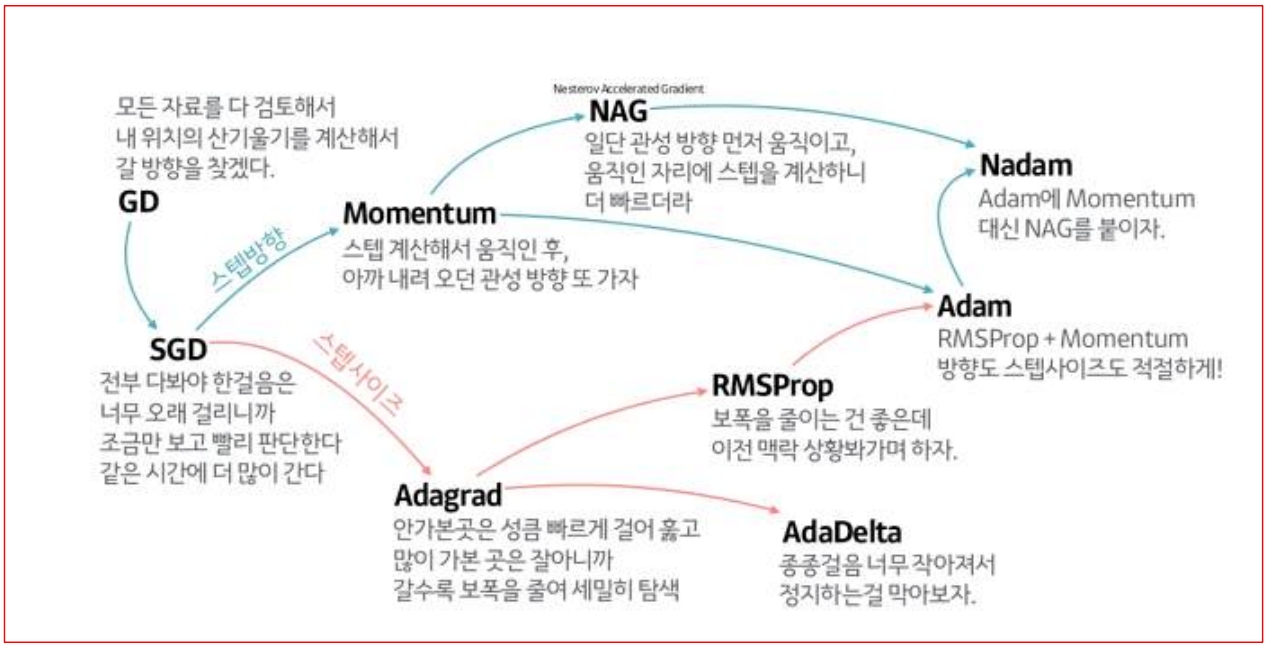
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

< 아담 알고리즘 >

< 질문 1. 아담 알고리즘 사용 시 기존의 SGD 방법을 사용할 때에 비해 어떤 변화나 장점이 발생하는가? >

위의 내용과 같이 학습률은 학습 결과에 매우 큰 영향을 미치는 하이퍼파라미터지만 적절한 값을 찾아내는 것은 많은 시행착오를 요하는 일이다. 아담 알고리즘은 파라미터별로 실질적인 학습률을 보정해 적용함으로써 일괄적인 학습률을 적용해서 나오는 품질의 저하를 막아준다. 하지만 아담 알고리즘이 적절한 학습률을 찾는 노력을 완전히 없애주는 것은 아니다. (결국 파라미터를 갱신하는 마지막 순간에 학습률 이용) 즉, 아담 알고리즘은 기존의 SGD 방법에 비해 학습률과 같은 하이퍼 파라미터 값에 강건한 장점이 있으며, 정확도와 경사에서 보폭의 크기 개선에 효과적이다.
그러나 알맞은 학습률을 찾는 노력을 완전히 없애주지는 않는다.

+) 최적화 방법에는 아담 알고리즘 이외에도 많은 방법이 존재한다. 다음의 그림을 통해 시각적으로 최적화 방법들이 어떤 방식으로 변화하였는지를 확인할 수 있다.



처음 신경망에 대한 최적화 방법으로 우리는 Gradient Descent (경사 하강법)을 사용하였다. 그러나 이 경우 한번 계산할 때 마다 전체 데이터에 대해 Loss Function을 계산해야 하므로 엄청나게 많은 계산량이 필요하다. 그래서 나온 방법이 Stochastic Gradient Descent(SDG)이다.

SDG의 경우 전체 데이터 대신 조그마한 데이터의 모음(mini-batch)에 대해 Loss function을 계산한다. 이전의 경사하강법에 비해 정확도는 떨어질 수 있으나, 훨씬 계산 속도가 빠르기 때문에 같은 시간 대비 더 많은 스텝을 진행시킬 수 있으며, 여러번 반복적으로 진행할 경우 전체 데이터를 사용했을 경우와 유사한 결과로 수렴한다.
이외에도 Adagrad, RMSProp, Momentum등이 존재하며, 이 중 아담은 Momentum, Adagrad, RMSProp의 장점들을 취해 만들어진 알고리즘이다.

3. 실험 결과 및 파라미터별 학습결과

3-1. < adam_test.ipynb >

1. < 은닉 계층 1개를 갖는 다층 퍼셉트론 >

```
In [3]:  1 om1 = AdamModel('office31_model_1', od, [10])
         2 om1.exec_all(epoch_count=20, report=10)
```

```
Model office31_model_1 train started:
    Epoch 10: cost=4.296, accuracy=0.685+0.035/0.790+0.040 (34/34 secs)
    Epoch 20: cost=4.268, accuracy=0.685+0.037/0.780+0.020 (31/65 secs)
Model office31_model_1 train ended in 65 secs:
Model office31_model_1 test report: accuracy = 0.661+0.048, (0 secs)

Model office31_model_1 Visualization
```

```
[ 도메인 추정결과 ]
추정확률분포 [65,15,20] => 추정 amazon : 정답 amazon => O
추정확률분포 [65,15,20] => 추정 amazon : 정답 amazon => O
추정확률분포 [65,15,20] => 추정 amazon : 정답 dslr => X
[ 상품 추정결과 ]
추정확률분포 [ 3, 3, 3, 3, 3, 3, 3, 3,...] => 추정 monitor : 정답 ring_binder => X
추정확률분포 [ 3, 3, 3, 3, 3, 3, 3, 3,...] => 추정 monitor : 정답 pen => X
추정확률분포 [ 3, 3, 3, 3, 3, 3, 3, 3,...] => 추정 monitor : 정답 mouse => X
```

- 크기 10의 은닉 계층 1개를 갖는 신경망으로 오피스31 데이터 셋을 학습시킨 결과, 학습 성과가 뛰어나지 않음을 확인할 수 있다. 도메인 추정 결과나 상품 추정 결과 모두 데이터에 관계 없이 동일한 확률 분포를 추정하고 그에 대한 결과를 항상 동일한 답을 선택하였다.

- 도메인 선택의 정확도가 66.1%이지만 이는 선택의 품질이 높아진 것이 아니라 모든 문제에 대해 아마존으로 대답하였기 때문이다. (테스트 데이터에 아마존 도메인의 데이터가 많이 존재)

- 상품 선택의 정확도는 4.8%의 낮은 정확도를 보이고 있다. 대부분 모니터라고 답하고 있으며, 이는 상품이 31가지임을 고려하면, 1/31 에 근접한 수준이다. (테스트 데이터 가운데 가장 많은 품목이 모니터일 가능성이 크다.)

2. < 은닉 계층 3개를 갖고 학습률을 조정한 다층 퍼셉트론 >

```
[ ]:  1 om2 = AdamModel('office31_model_2', od, [64,32,10])
      2 om2.exec_all(epoch_count=50, report=10, learning_rate=0.0001)
```

```
Model office31_model_2 train started:
    Epoch 10: cost=3.749, accuracy=0.804+0.091/0.860+0.090 (129/129 secs)
    Epoch 20: cost=3.478, accuracy=0.851+0.141/0.840+0.110 (128/257 secs)
    Epoch 30: cost=3.273, accuracy=0.868+0.192/0.840+0.090 (133/390 secs)
    Epoch 40: cost=3.123, accuracy=0.882+0.214/0.900+0.130 (135/525 secs)
    Epoch 50: cost=2.985, accuracy=0.887+0.247/0.870+0.140 (143/668 secs)
Model office31_model_2 train ended in 668 secs:
Model office31_model_2 test report: accuracy = 0.867+0.195, (1 secs)

Model office31_model_2 Visualization
```

```
[ 도메인 추정결과 ]
추정확률분포 [100, 0, 0] => 추정 amazon : 정답 amazon => 0
추정확률분포 [100, 0, 0] => 추정 amazon : 정답 amazon => 0
추정확률분포 [95, 0, 5] => 추정 amazon : 정답 amazon => 0
[ 상품 추정결과 ]
추정확률분포 [ 6, 0, 0,20, 1,11, 7, 8,...] => 추정 bookcase : 정답 tape_dispenser => X
추정확률분포 [ 0, 1, 3, 0, 0, 0, 0, 0,...] => 추정 keyboard : 정답 letter_tray => X
추정확률분포 [ 0, 5, 0, 1, 1, 3, 1, 5,...] => 추정 pen : 정답 ruler => X
```

- 3개의 은닉 계층 [64, 32, 10]을 가지도록 변화를 주고 학습률을 0.0001로 변화 시킨 후 50 epoch만큼 오피스31 데이터셋에 대해 학습한 결과 이전 보다 더 좋은 성능을 확인할 수 있었다.

- 도메인 선택의 정확도가 86.7%로 상승하였으며, 데이터에 따른 추정확률 분포또한 달라진 모습을 확인할 수 있다.

- 상품 선택의 정확도가 19.5%로 크게 상승함을 확인할 수 있다. 아직 정확도는 20%미만대이지만, 각각의 데이터에 대한 추정확률 분포도 달라지고 추정 답도 달라지는 모습을 확인할 수 있다.

3. < 아담 알고리즘을 이용한 다층 퍼셉트론 >

```
1  om3 = AdamModel('office31_model_3', od, [64,32,10])
2  om3.use_adam = True
3  om3.exec_all(epoch_count=50, report=10, learning_rate=0.0001)
```

```
Model office31_model_3 train started:
    Epoch 10: cost=3.708, accuracy=0.818+0.089/0.870+0.050 (135/135 secs)
    Epoch 20: cost=3.303, accuracy=0.867+0.166/0.860+0.150 (136/271 secs)
    Epoch 30: cost=3.096, accuracy=0.883+0.202/0.880+0.180 (140/411 secs)
    Epoch 40: cost=2.931, accuracy=0.894+0.242/0.880+0.230 (142/553 secs)
    Epoch 50: cost=2.798, accuracy=0.902+0.272/0.870+0.170 (138/691 secs)
Model office31_model_3 train ended in 691 secs:
Model office31_model_3 test report: accuracy = 0.882+0.237, (1 secs)

Model office31_model_3 Visualization
```

```
[ 도메인 추정결과 ]
추정확률분포 [99, 0, 0] => 추정 amazon : 정답 amazon => O
추정확률분포 [100, 0, 0] => 추정 amazon : 정답 amazon => O
추정확률분포 [100, 0, 0] => 추정 amazon : 정답 amazon => O
[ 상품 추정결과 ]
추정확률분포 [ 0, 1, 0, 2, 5, 2, 1,13,...] => 추정 desk_lamp : 정답 scissors => X
추정확률분포 [ 2, 2, 1, 9, 7, 2, 0,12,...] => 추정 paper_notebook : 정답 printer => X
추정확률분포 [ 0, 0, 2, 2, 0, 1, 2, 1,...] => 추정 headphones : 정답 calculator => X
```

- 3개의 은닉 계층 [64, 32, 10]을 가지도록 변화를 주고 학습률을 0.0001로 변화 시킨 후 50 epoch만큼 오피스31 데이터셋에 대해 학습시켰다. 이전 모델들과 다르게 아담 알고리즘도 적용하였다. 학습 결과 이전의 결과보다 조금 상승한 모습을 확인할 수 있었다.

- 도메인 선택의 정확도가 88.3%로 상승하였으며, 데이터에 따른 추정확률 분포 또한 달라진 모습을 확인할 수 있다.

- 상품 선택의 정확도가 23.7%로 조금 상승함을 확인할 수 있다. 각각의 데이터에 대한 추정확률 분포도 달라지고 추정 답도 달라지는 모습을 확인할 수 있다.

- 아담 알고리즘을 적용시킨 결과 확실히 성능 개선에 도움이 되었다. 무조건적으로 학습 결과가 개선되는 것은 아니지만, 대부분의 경우 기본적인 경사하강법보다 조금이라도 더 나은 결과를 얻을 수 있었다.

4. < 아담 알고리즘을 이용한 다층 퍼셉트론 >

```
om4 = AdamModel('office31_model_4', od, [64,32,10, 5])
om4.use_adam = True
om4.exec_all(epoch_count=30, report=10, learning_rate=0.005)
```

```
Model office31_model_4 train started:
    Epoch 10: cost=4.260, accuracy=0.685+0.034/0.780+0.040 (498/498 secs)
    Epoch 20: cost=4.260, accuracy=0.685+0.034/0.790+0.010 (502/1000 secs)
    Epoch 30: cost=4.260, accuracy=0.685+0.037/0.810+0.050 (1065/2065 secs)
Model office31_model_4 train ended in 2065 secs:
Model office31_model_4 test report: accuracy = 0.661+0.048, (0 secs)

Model office31_model_4 Visualization
```

```
[ 도메인 추정결과 ]
추정확률분포 [68,13,19] => 추정 amazon : 정답 webcam => X
추정확률분포 [68,13,19] => 추정 amazon : 정답 amazon => O
추정확률분포 [68,13,19] => 추정 amazon : 정답 dslr => X
[ 상품 추정결과 ]
추정확률분포 [ 3, 3, 3, 3, 2, 3, 4, 3,...] => 추정 monitor : 정답 punchers => X
추정확률분포 [ 3, 3, 3, 3, 2, 3, 4, 3,...] => 추정 monitor : 정답 pen => X
추정확률분포 [ 3, 3, 3, 3, 2, 3, 4, 3,...] => 추정 monitor : 정답 monitor => O
```

- 4개의 은닉 계층 [64, 32, 10, 5]을 가지도록 변화를 주고 학습률을 0.005로 변화 시킨 후 30 epoch만큼 오피스31 데이터셋에 대해 학습시켰다. 이전 모델과 동일하게 아담 알고리즘도 적용시켰으나, 오히려 정확도는 감소하는 모습을 확인할 수 있었다.

- 도메인 선택의 정확도가 66.1%로 감소하였으며, 데이터에 따른 추정확률 분포가 처음의 모델처럼 동일하게 나오는 것을 확인할 수 있다.

- 상품 선택의 정확도가 4.8%로 크게 감소함을 확인할 수 있다. 각각의 데이터에 대한 추정확률 분포도 동일하게 나오는 것을 확인할 수 있다.

- 무조건적으로 아담 알고리즘을 적용시킨다고 해서 좋은 결과를 내는 것은 아닌 것을 이번 실험을 통해 확인할 수 있었다.

5. < 아담 알고리즘을 이용한 다층 퍼셉트론 >

```
1 om5 = AdamModel('office31_model_5', od, [64,34,15])
2 om5.use_adam = True
3 om5.exec_all(epoch_count=30, report=10, learning_rate=0.08)
```

```
Model office31_model_5 train started:
    Epoch 10: cost=4.302, accuracy=0.685+0.040/0.780+0.020 (484/484 secs)
    Epoch 20: cost=4.300, accuracy=0.685+0.032/0.760+0.030 (487/971 secs)
    Epoch 30: cost=4.306, accuracy=0.685+0.029/0.790+0.040 (1051/2022 secs)
Model office31_model_5 train ended in 2022 secs:
Model office31_model_5 test report: accuracy = 0.661+0.054, (0 secs)

Model office31_model_5 Visualization
```



```
[ 도메인 추정결과 ]
추정확률분포 [70,12,17] => 추정 amazon : 정답 webcam => X
추정확률분포 [70,12,17] => 추정 amazon : 정답 webcam => X
추정확률분포 [70,12,17] => 추정 amazon : 정답 amazon => O
[ 상품 추정결과 ]
추정확률분포 [ 3, 3, 3, 2, 2, 4, 3, 5,...] => 추정 speaker : 정답 bike_helmet => X
추정확률분포 [ 3, 3, 3, 2, 2, 4, 3, 5,...] => 추정 speaker : 정답 calculator => X
추정확률분포 [ 3, 3, 3, 2, 2, 4, 3, 5,...] => 추정 speaker : 정답 scissors => X
```

- 3개의 은닉 계층 [64, 32, 15]을 가지도록 변화를 주고 학습률을 0.08로 변화 시킨 후 30 epoch만큼 오피스31 데이터 셋에 대해 학습시켰다. 이전 모델과 동일하게 아담 알고리즘도 적용시켰으나, 3번 모델에 비해 정확도가 현저하게 감소한 모습을 확인할 수 있다.

- 도메인 선택의 정확도가 66.1%로 이전 모델과 동일하다. 또한, 데이터에 따른 추정확률 분포가 1번 모델처럼 동일하게 나오는 것을 확인할 수 있다.

- 상품 선택의 정확도가 5.4%로 미미하게 증가함을 확인할 수 있다. 각각의 데이터에 대한 추정확률 분포도 동일하게 나오는 것을 확인할 수 있다.

- 신경망에서 적절한 하이퍼 파라미터, 은닉계층의 수 및 구성을 찾기 위해서는 엄청난 노력과 비용, 시간이 들어가야 한다는 것을 조금이나마 알 수 있는 실험이었다.