QUANTITATIVE ISSUE

2026. 1. 2

MACRO팀

김동영, CFA Quant Analyst
dy76.kim@samsung.com

리서치센터 리포트
바로가기

퀀트 모델링 A to Z

(6) MLP(Multi-Layer Perceptron) AI 투자 모델

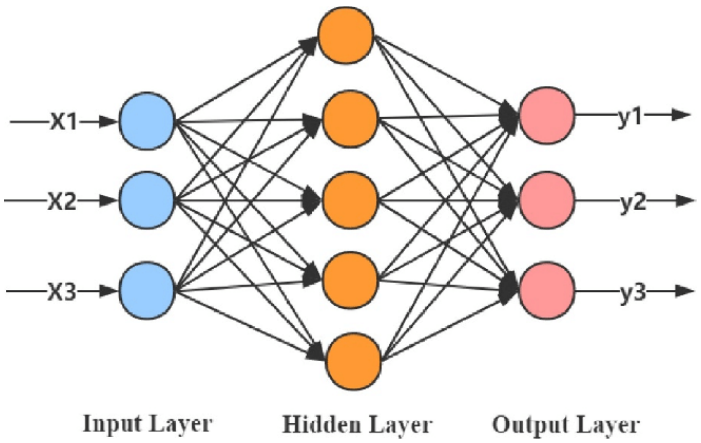
• 딥러닝의 기본 형태인 MLP 모델에 대한 설명 및 퀀트 모델 접목 방법을 정리

WHAT'S THE STORY?

ChatGPT에서 보듯이, 현재 AI 기술이 일상생활 및 각 산업 일선마다 깊숙이 침투하고 있다. 퀀트 운용에서도 Attention 혹은 NLP 알고리즘 등의 최신 AI 이론을 접목하려는 시도가 점차 늘고 있다. **본 자료는, 신경망 모델의 가장 기초인 MLP 모델을 활용하여 퀀트 투자 모델을 구축하는 방법을 예시를 통해 구체적이고 상세히 설명하고자 한다.**

우선, MLP(Multi-Layer Perceptron, 다층 퍼셉트론) 모델은 퍼셉트론을 여러 층으로 쌓은 순방향의 인공 신경망을 말한다. 다른 말로, FNN(Feedforward Neural Network, 순방향 신경망)이라고도 칭한다. 이 모델은 입력층, 여러 개의 은닉층, 출력층으로 구성된 상대적으로 간단한 형태의 신경망이다. 각 층의 연결에 사용되는 많은 파라미터, 그리고 비선형적인 활성화 함수에 의해서, MLP는 단순한 알고리즘이 풀 수 없는 복잡한 패턴의 연관 관계를 학습할 수 있다.

MLP(Multi-Layer Perceptron) 개념도



자료: ResearchGate

삼성증권

I. 모델 구상

본 리포트는, MLP 기반 투자 모델을 예제로 만들면서 MLP의 기능과 사용법을 순차적으로 설명하는 구조로 되어 있다. 여기서, 예시로 만들어 볼 퀀트 모델은 소위 "업종 선택 모델"이다.

전체 주식시장을 업종 단위로 분할해 보면, 보통 "에너지, 소재, … 유틸리티"까지로 이어지는 10개 섹터 분류가 가장 기본 형태다(글로벌 기준에서는, 부동산 섹터를 추가한 11개 섹터 분류가 일반적임). 모델의 취지는 업종 고유의 투자 정보들을 활용하여, 업종 배분 혹은 업종 로테이션 전략을 통해 코스피 수익률을 아웃퍼폼하는 전략을 만드는 것이다. 즉, 모델의 최종적인 목적은, 주가가 시장 대비 상승할 업종을 선택해 주는 딥러닝 투자 모델을 만들어서, 시장 수익률을 장기적으로 아웃퍼폼하는 것이다.

예제 모델에서, 업종 선택에 사용되는 시그널, 즉 입력 정보로는 1) 업종별 FY1 영업이익 1개월 변화율, 2) 업종별 FY2 영업이익 1개월 변화율 정보를 우선 선택했다. 과거의 여러 업종 상승 요인 분석에 따르면, 업종 지수의 주가 트리거로는 업종의 이익 모멘텀 정보(실적 전망의 변화 정보)가 상당히 유의미했던 것으로 나온다. 쉽게 말해, 실적의 컨센서스가 상향되는 업종이 사후 주가도 계속 아웃퍼폼하는 경향이 강했다고 할 수 있다. 이에 따라 해당 이익 모멘텀 정보를, 업종 선택 모델의 입력 정보로 사용했다.

"업종별 FY1 및 FY2 영업이익 1개월 변화율 정보"를 가지고 "추천 업종"을 선택하기 위해서는, 과거 둘 간의 관계의 상세 패턴을 더 세밀하게 학습해야 한다. 예를 들어, 어떤 업종은 바로 닥친 올해의 실적 변동이 더 중요할 수 있고, 어떤 업종은 미래인 내년의 실적 변동이 더 중요할 수 있다. 어떤 업종은 일정 수치까지의 실적 변화율만 주가 상승에 영향을 주고, 임계점을 넘어가면 추가 영향이 없는 방식일 수 있다. 이런 복잡한 연관 패턴에 대한 학습은 딥러닝 형태의 모델이 더 유용할 수 있다.

전체적인 투자 모델 구성 중에서 MLP 모델의 역할은, input으로 "업종별 FY1 및 FY2 영업이익 1개월 변화율 정보"를 받아서, 10개 업종의 다음 한 달간의 상대 수익률을 output으로 예측하는 부분까지의 역할이다. 포괄적인 업종 투자 실행 및 백테스팅 모델의 경우에는, 이 MLP 모델을 활용하여 10개 업종의 다음 기간 상대 수익률 예측치가 나오면, 이를 가지고 예상 수익률 상위 n개 업종에만 동일 비중 투자하는 방식의 투자 규칙을 정하고, 이를 통해 과거 백테스팅 기간의 시뮬레이션을 수행하는 역할을 하게 된다.

전체적인 모델의 디자인은 다음의 구성과 같다.

1. 관련 데이터 수집
2. 업종별 사후 수익률을 예측하는 MLP 모델의 구축 및 학습과 평가
3. MLP 모델을 사용한 업종 투자 실행 및 백테스팅 모델 구성 및 확인

여기서 사용 및 설명하는 전체 코드는, 가장 대중적인 딥러닝 개발 환경인 Python 및 Keras 딥러닝 라이브러리를 기초로 했다. 상세한 프로그래밍 환경은 다음과 같다.

```
Python 3.12.7
tensorflow 2.20.0
tensorflow.keras 3.12.0
```

데이터 정의 및 수집

구상하는 MLP 모델은, 입력 정보(시그널)로 각 업종의 현재 이익모멘텀 정보들을 받아서, 이를 가지고 다음 한 달간의 업종별 시장 초과수익률을 전부 예측하는 형태로 구상했다. 이는 multi-output regression 형태에 해당한다.

모델 구상에 맞춰, MLP 모델에서 y(종속변수 혹은 target 혹은 label) 데이터에는 10개 업종의 사후 4주간의 KOSPI 대비 상대 수익률 수치를 넣었다. 이때, 주식 시장의 환경에 따라서 코스피 및 업종의 사후 수익률 절대 수치는 위아래로 크게 변화하는 성격을 가진다. y 데이터에 절대 수익률 값을 쓴다면, 의도하지 않게 변동이 큰 데이터로 인해 모델 학습 능력이 떨어질 수 있다. 절대 수익률이 아닌 초과 수익률(KOSPI 대비 상대 수익률) 수치를 사용할 경우에는, y 데이터를 전반적으로 평균 0%에 맞추게 되므로 표준화(Standardization)의 이점이 있다. 또한 초과 수익률이 높은 업종을 찾으려는 최종 목적에도 부합한다. 따라서, y 데이터에는 10개 업종의 사후 초과 수익률 정보를 넣는 방식을 썼다. 10개 업종이므로, y 데이터의 차원은 [샘플 수, 10]이다. 실제 데이터들은 2005/12/30(금) ~ 2025/10/31(금) 동안의 1,036개 샘플을 모았다. 즉, y 데이터의 실제 차원은 [1036, 10]이다.

y 데이터의 구성

s_date	에너지	소재	산업재	경기소비재	필수소비재	건강관리	금융	IT	커뮤니케이션	유틸리티
2005-12-30	0.072	0.028	-0.042	-0.046	0.014	-0.088	-0.024	0.034	0.009	0.063
2006-01-06	0.088	0.057	-0.086	-0.051	0.046	-0.090	-0.049	0.026	0.083	0.065
2006-01-13	0.001	0.030	-0.051	-0.056	0.064	-0.051	0.000	0.009	0.077	0.030
2006-01-20	-0.044	0.038	-0.009	-0.069	0.054	0.031	0.043	-0.019	0.048	0.019
2006-01-27	0.005	0.049	0.047	-0.015	0.108	0.089	-0.028	-0.034	0.047	0.002

참고: 원 수치로 저장(예로, 사후 초과수익률이 7.2%면 0.072로 표시)
자료: Quantiwise

X(독립변수, feature) 데이터는 10개 업종에 대한 "FY1 영업이익 1개월 변화율", "FY2 영업이익 1개월 변화율" 정보를 모은 총 20종의 데이터로 되어 있다. 여기서도, 실제로는 "각 업종의 영업이익 변화율 수치"에 "KOSPI 영업이익 변화율 수치"를 뺀, "상대 변화율 정보"로 넣었다. 영업이익 변화율 정보 또한 시기별로 수치 레벨이 위아래로 크게 변동할 수 있으므로(non-stationary), 최소한의 표준화 장치를 넣은 셈이다.

X에는 총 20종의 자료가 사용되므로, X 데이터의 차원은 [샘플 수, 20]이다. 실제 데이터들은 y 데이터와 동일한 시점의 1,036개 샘플을 모았다. 최종적으로 X 데이터의 실제 차원은 [1036, 20]인 행렬로 만들었다.

X 데이터의 구성

s_date	o_y1_0	o_y1_1	o_y1_2	o_y1_3	o_y1_4	o_y1_5	o_y1_6	o_y1_7	o_y1_8	o_y1_9	o_y2_0	o_y2_1	o_y2_2	o_y2_3	o_y2_4	o_y2_5	o_y2_6	o_y2_7	o_y2_8	o_y2_9
2005-12-30	-0.7	-2.1	0.3	-0.5	-1.7	0.0	3.1	-0.7	-0.2	-2.0	1.4	-11.0	-0.3	0.8	-2.0	1.8	4.3	0.7	-1.0	8.4
2006-01-06	0.0	-0.3	0.7	0.1	-0.9	1.1	3.0	0.6	0.7	0.8	0.6	-8.1	0.2	0.7	-2.7	2.4	3.5	1.1	-0.6	9.8
2006-01-13	0.6	-9.5	-0.4	0.4	-0.5	1.0	2.4	-0.1	0.8	1.2	1.3	-7.9	1.1	1.7	-0.9	2.5	4.3	1.2	1.0	1.3
2006-01-20	1.8	-7.5	1.6	2.3	1.3	2.9	5.1	1.9	3.0	3.3	1.8	-6.6	1.9	2.7	0.1	2.9	5.8	2.8	2.7	3.2
2006-01-27	-1.3	-1.5	2.1	0.0	0.5	3.6	6.5	2.9	4.6	2.2	-0.1	-1.1	0.9	-0.4	-0.8	3.1	6.1	3.3	3.2	2.8

자료: Quantiwise

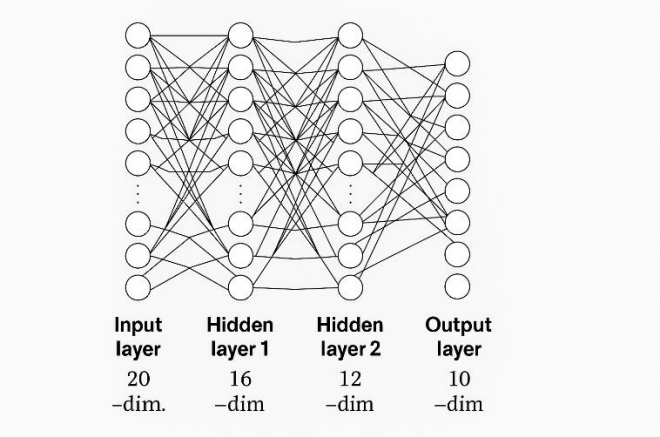
각각의 데이터는 엑셀에서 수집한 다음, Python에서 y_df, X_df란 이름의 DataFrame 자료형으로 저장한다.

II. MLP 모델 구조

기본 구조

이번 예시에서는 MLP 구조를, 20개 뉴런을 가진 "입력층"과 10개 뉴런을 가진 "출력층" 사이에 16개 뉴런을 가진 "은닉층 1"과 12개 뉴런을 가진 "은닉층 2"를 넣는 구조로 구상했다. 이 경우, 신경망의 모양은 다음처럼 된다.

구상한 MLP 모델 구조



자료: 삼성증권

즉, 입력층에 20차원(20종)의 업종 이익모멘텀 수치들이 입력되면, 은닉층 1에서는 16차원의 데이터로 변환되고, 은닉층 2에서는 12차원의 데이터로 변환된다. 최종 출력층에서는 10차원의 출력 데이터, 즉 각 업종별 사후 초과수익률 수치로 변환된다. 신경망 모델에서의 학습이란, 입력 데이터를 넣으면 출력 데이터가 그에 맞게 잘 나올 수 있도록, 각 계층 간 연결에 사용된 가중치(행렬)를 역전파 학습을 통해 열심히 최적값으로 업데이트하는 과정을 말한다.

딥러닝에 대한 가장 대표적인 라이브러리인 keras에서는, 신경망 구성을 위해서 keras.models.Sequential과 keras.layers.Input, keras.layers.Dense 클래스 등을 제공한다.

keras.models.Sequential: 층(layer)를 순차적으로 쌓아 구성하는 신경망 형태를 만드는 클래스

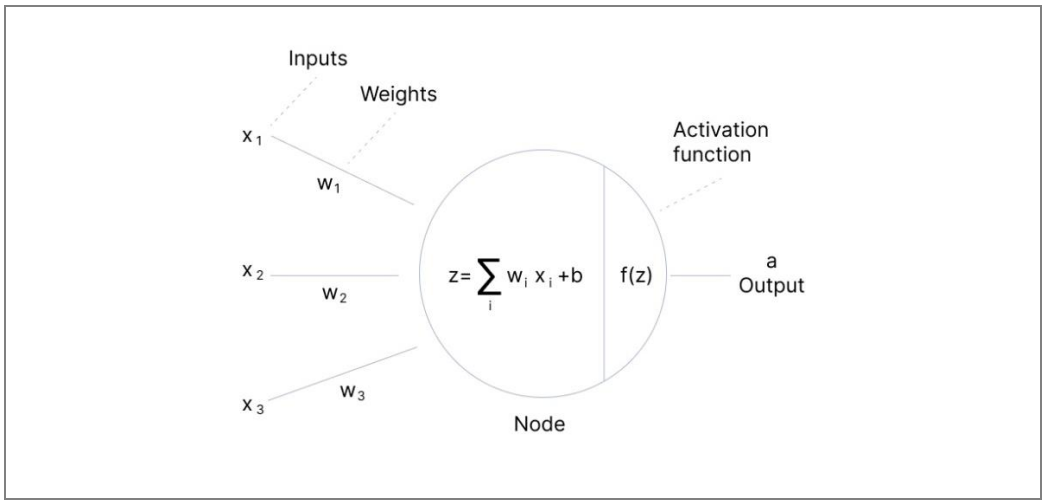
keras.layers.Dense: 완전 연결층 혹은 밀집층을 나타내는 클래스임. '이전 층의 모든 뉴런과 현재 층의 모든 뉴런이 연결되는 구조'를 가진 층을 만들 때 사용함(즉, 행렬 계산을 통해서, 두 층의 뉴런이 모두 연결되는 방식임)

keras.layers.Input: 신경망 모델에서 입력층을 정의하는 별도 함수임.

그리고 keras에서는 layer 선언 시에, 이에 대한 활성화 함수도 같이 지정해 줘야 한다.

활성화 함수:

활성화 함수는, 층 간의 수치 전달 시, 직전 층 뉴런의 가중 평균값을 입력으로 받아서 다음 층의 뉴런에 전달할 때에 마지막 단계에서 적용되는 비선형 함수를 말한다(아래 개념도 그래프에서의 f 역할). 이 활성화 함수는 항상 비선형성 특성을 부여함으로써, 신경망을 딥(deep)하게 만드는 핵심적인 역할을 하고 있다.



자료: https://velog.io/@hyunsuki/

현재 keras에서 제공하는 activation function들은 다음과 같다(물론 직접 만들어 사용하는 것도 가능하다).

```
celu, elu, exponential, gelu, glu, hard_shrink, hard_sigmoid, hard_silu,
hard_tanh, leaky_relu, linear, log_sigmoid, log_softmax, mish, relu, relu6,
selu, sigmoid, silu, softmax, soft_shrink, softplus, softsign, sparse_plus,
sparsemax, squareplus, tanh, tanh_shrink, threshold
```

이 중 많이 사용되는 것들에 relu 시리즈, sigmoid, softmax 등이 있다.

relu: 함수식은 $[ReLU(x) = max(0, x)]$다. 이 함수는 기울기 소실 문제에서 강점이 있는, 은닉층에서 거의 표준으로 사용되는 활성화 함수다.

sigmoid: 함수식은 $[\sigma(x) = 1/(1 + e^{-x})]$다. 출력 범위가 (0~1)을 가져서 확률적 해석이 가능하다. 이진 분류(binary classification)의 출력층, 그리고 다중 레이블 분류(multi-label classification)의 출력층에 사용된다. 출력층 벡터의 각 요소가 서로 독립일 때 사용한다.

softmax: 함수식은 $[softmax(x_i) = e^{x_i}/(\sum_i e^{x_i})]$다. 출력 벡터의 모든 요소가 0~1 사이의 값을 가지고 이들의 총합을 항상 1로 만들어서, 확률로 해석할 수 있도록 만들어주는 함수다. 즉, 클래스가 여러 개지만 정답은 그중 하나뿐인, 다중 분류(multiclass classification)의 출력층에 사용된다.

None(linear): regression 출력층의 경우에는 활성화 함수를 쓰지 않는 것이 보통이다. keras 상에서 linear 옵션은 None과 동일하다. (단, 회귀 모델 출력값을 특정 범위로 제한할 때는 tanh/sigmoid 등을 쓸 수도 있음.)

이번 MLP 모델에서, 우선 은닉층의 활성화 함수에는 가장 기본적인 relu를 사용한다. 출력층의 경우 이번 모델이 multi-output "regression" 형태이므로, 출력층에는 활성화 함수를 지정하지 않는다.

이상의 내용을 가지고, 모델 입출력 사이즈를 함수의 입력으로 받는, "MLP 모델을 생성하는 함수"의 초기 코드는 다음과 같다.

```
# 코드 1: 최초

from tensorflow import keras
from tensorflow.keras import layers

def build_model(input_shape, output_shape):
    """지정된 입출력 크기에 맞는 keras 모델을 생성합니다."""
    model = keras.models.Sequential([
        layers.Input(shape=(input_shape,)),
        layers.Dense(16, activation='relu'),
        layers.Dense(12, activation='relu'),
        layers.Dense(output_shape)
    ])

    return model
```

keras.models.Sequential로 신경망 형태를 만들고, 그 안의 구성은 입력층 → relu 활성화 함수를 가진 은닉층 1 → relu 활성화 함수를 가진 은닉층 2 → 활성화 함수가 없는 출력층으로 쌓는 작업을 보여주는 코드다.

드롭아웃(dropout):

신경망에서 드롭아웃이란, 과적합을 줄이기 위해서 학습 과정에서 일부 뉴런을 임의로 비활성화시키는 정규화 기법을 말한다. 대부분의 신경망 모델에서 기본적으로 사용되는 기법이다. 예를 들어, Dropout(0.3)으로 지정해 놓으면, 학습 과정에서 해당 뉴런 연결의 30%를 랜덤하게 끄면서 학습을 진행한다. 마치, 랜덤 포레스트 알고리즘에서 랜덤하게 선택된 feature subset을 사용해 랜덤한 트리를 만드는 것과 유사하게, 일부 정보를 제외하는 방식이라 할 수 있다. 이를 통해, 신경망의 과적합(overfitting)을 막는 기술이다.

keras에서는 keras.layers.Dropout 층을 선언하여 드롭아웃을 구현하는 방식을 쓰고 있다. 즉, 드롭아웃이 필요한 위치에"layers.Dropout(0.3)"의 코드를 추가하는 식이다. 이번 모델에서는 은닉층 1과 은닉층 2 사이, 은닉층 2와 출력층 사이에 드롭아웃을 적용하도록 코드를 수정했다(사후에 나올 최종 코드에서 확인).

한편, 신경망 모델의 실제 사용 전에는, 또 모델의 최적화 방식(optimizer), 손실 함수(loss), 평가 지표(metric) 들도 지정해야 한다. 이 내용들은 keras.Model.compile() 메서드 안에서 정의된다.

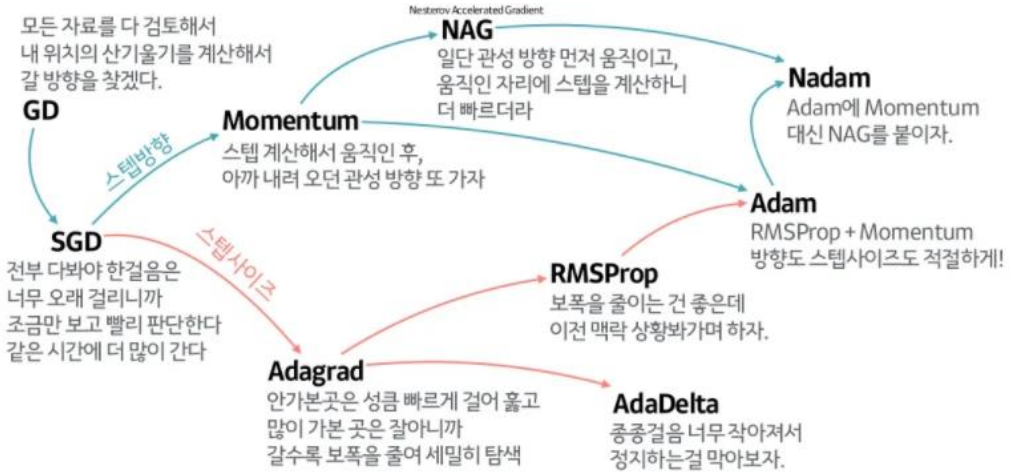
Optimizer:

Optimizer는 신경망 모델에서, 모델 학습 시 모델의 파라미터를 어떻게 업데이트할지 결정하는 세부 알고리즘을 말한다. keras에서 현재 제공하는 optimizer 종류는 다음과 같다.

```
SGD, RMSprop, Adam, AdamW, Adadelta, Adagrad, Adamax, Adafactor, Nadam, Ftrl,
Lion, Lamb, Loss Scale Optimizer, Muon
```

각 Optimizer마다 서로 다른 다양한 특성을 가지고 있으나, 많은 딥러닝 프로젝트들에서 성능 좋은 Adam을 디폴트로 사용하고 있다. Adam은 Adaptive Moment Estimation의 약자로, 적응적 학습률과 모멘텀 아이디어를 결합한 '확률적 경사하강법' 알고리즘의 하나다. 우선, 이번 모델에서도 Adam을 그대로 사용한다.

신경망 모델 Optimizer들에 대한 설명



자료: 이용호(국가인공지능전략위원회 민간위원) 님 자료. https://www.slideshare.net/slideshow/ss-79607172/79607172

손실 함수(loss function):

딥러닝에서 손실(loss)은 예측과 실제 사이의 틀린 정도를 의미하는 수치고, 손실 함수는 이 손실값을 만들어주는 함수다. 딥러닝 모델에서 학습이란, 모델의 예측값과 실제값을 손실 함수에 넣어 나온 결과인 손실값(loss)이 계속 낮아지도록 가중치를 조절하는 과정이라고 할 수 있다. 즉, 모델의 학습 방향과 속도, 최종 성능에 직접적인 영향을 미치는 핵심 기준이라 할 수 있다. 모델의 예측값과 실제값을 손실 함수에 넣어서 나온 "손실값"이 계속 낮아지면 모델의 예측 성능이 올라간다는 뜻이고, "손실값"이 계속 높아지면 모델의 예측이 많이 틀리다는 뜻이다. 이 손실값의 변화를 보면서, 역전파 학습이 신경망의 파라미터 수치를 계속 업데이트하게 된다.

현재 keras에서 제공하는 losses 및 loss function들은 다음과 같이 다양하다(물론 직접 만들어 사용하는 것도 가능하다).

```
(Probabilistic losses) BinaryCrossentropy, BinaryFocalCrossentropy,
CategoricalCrossentropy, CategoricalFocalCrossentropy,
SparseCategoricalCrossentropy, Poisson, CTC, KLDivergence,
binary_crossentropy function, categorical_crossentropy function,
sparse_categorical_crossentropy function, poisson function, ctc function,
kl_divergence function

(Regression losses) MeanSquaredError, MeanAbsoluteError,
MeanAbsolutePercentageError, MeanSquaredLogarithmicError, CosineSimilarity,
Huber, LogCosh, Tversky, Dice, mean_squared_error function,
mean_absolute_error function, mean_absolute_percentage_error function,
mean_squared_logarithmic_error function, cosine_similarity function, huber
function, log_cosh function, tversky function, dice function

(Hinge losses for "maximum-margin": SVM 알고리즘의 최대 마진 원리를 흉내 내는 손실 함
수) Hinge, SquaredHinge, CategoricalHinge, hinge function, squared_hinge
function, categorical_hinge function
```

손실 함수의 선택은, 모델의 형태 및 목적과 밀접하게 관련되어 있다. 신경망 모델 출력 형태와 그에 따른 대표적인 손실 함수의 매칭 표는 다음과 같다.

주요 손실 함수와 신경망 모델 출력 형태의 매칭 표

출력 형태	주요 손실함수	손실함수 설명	수식	값 범위
회귀(연속 값 예측)	MeanSquaredError (MSE)	예측값과 결괏값을 오차를 제곱해 평균	loss = mean(square(y_true - y_pred)) (여기서, y_true, y_pred는 벡터임)	$[0, \infty)$
	MeanAbsoluteError (MAE)	예측값과 결괏값의 절댓값 오차의 평균	loss = mean(abs(y_true - y_pred))	$[0, \infty)$
이진 분류(0/1 레이블)	BinaryCrossentropy	모델이 예측한 확률과 실제 레이블(0/1) 사이의 차이를 로그 확률 기반으로 측정	$-(y \cdot log(p) + (1-y) \cdot log(1-p))$ 이때, 결괏값 y는 0 또는 1. p는 $y = 1$에 대한 확률.	$[0, \infty)$
다중 클래스 분류	CategoricalCrossentropy	다중 클래스에서 모델 예측과 실제 레이블의 차이를 로그 확률로 측정. Binary의 확장 형태	$-\sum_{i=1}^{K} y_i log(p_i)$ 이때, 클래스는 K 개. y_i는 실제 레이블(one-hot 형태, 0 또는 1). p_i는 $y_i = 1$일 확률.	$[0, \infty)$
다중 레이블 분류	BinaryCrossentropy	각 노드(요소)별 BinaryCrossentropy 수치의 평균		$[0, \infty)$

참고: 손실값은 항상 수치가 작은 것이 좋음
자료: 삼성증권

MeanSquaredError (MSE):

연속 값을 예측하는 회귀형 모델에서는, 손실 함수로 주로 MSE나 MAE 등이 사용된다. MeanSquaredError (MSE)는 예측값 벡터와 정답값 벡터의 차이를 제곱하여 평균한 수치를 말한다. 예측값과 정답값이 정확히 일치하면, 당연히 MSE는 0이다. 둘의 수치가 다를수록 MSE는 계속 커진다.

오랜 역사를 가진 "다중 선형 회귀 분석 모델"을 신경망 모델의 관점으로 보면, 다중 선형 회귀 분석 모델은 "1) n차원의 입력층과 1차원의 출력층이 있고, 2) 은닉층이 없으며, 3) 출력층으로의 활성화 함수가 없고, 4) 손실 함수가 MSE인 선형적 신경망 모델"과 완전히 동일하다.

MeanAbsoluteError (MAE):

MeanAbsoluteError(MAE)는 MSE와 다르게, 차이의 제곱 평균이 아닌 차이의 절댓값의 평균을 사용하는 방식이다. MSE는 속도가 빠르고 안정적인 학습이 가능한 장점이 있고, MAE는 이상치가 많을 때 더 견고한 특성을 가진다. 한편, 큰 오차에서는 MAE를 썼다가 작은 오차에서는 MSE로 전환하는, huber 손실 함수 등의 개량 버전도 존재한다.

BinaryCrossentropy:

BinaryCrossentropy는 이진 분류에서 사용하는 손실 함수다. 이의 수식을 잘 보면, 실제 y가 1일 때는 $-y \cdot log(p)$ 수식만으로, y가 0일 때는 $-(1-y) \cdot log(1-p)$ 수식만으로 단순화되는 구조다. 예를 들어, 실제 y가 1일 때 예측 확률 p를 100%까지 높이면 함수 값은 0이 된다. y가 1일 때 예측 확률을 정반대인 p=0%로 낮게 잡으면, 함수 값은 ∞까지 올라간다. 실제 y가 0일 때도 비슷하게 예측을 잘하면 함수 값이 0으로 떨어지고, 틀리면 함수 값이 올라간다. 즉, 0 or 1의 분류 예측에서 맞게 분류하는 확률값을 높일수록, 이 손실 함수 값을 0으로 줄일 수 있다.

CategoricalCrossentropy:

CategoricalCrossentropy는 다중 클래스 분류에서 사용하는 손실 함수다. 위에 나온 이 함수의 수식을 보면, BinaryCrossentropy 함수를 단순히 다중 클래스 버전으로 확대한 형태일 뿐이다. 다중 클래스 상의 선택 문제(예로 개, 고양이, 소)에서 맞게 분류하는 확률을 올릴수록, 이 손실값은 0으로 줄어든다.

평가 지표(metric):

신경망 모델에서 평가 지표(metric)란, 학습된 모델의 성능을 사람이 이해하기 쉽게 측정 및 요약해 주는 숫자 지표다. 모델 학습에는 별도의 손실 함수가 실질적으로 사용되지만, 평가 시에는 더 이해하기 쉬운 전통적이고 직관적인 수치를 사용하는 것이 장점이 많다. 예를 들어, 회귀 모델에서는 손실 함수에 MSE를 쓰고, 평가 지표에 MAE를 쓰는 경우가 많다. 이는, 학습 시에는 MSE를 통해 안정적으로 진행하고, 평가 결과는 MAE를 사용해 예측값이 실제값에서 얼마나 값 차이가 나는지를 직관적으로 바로 확인하기가 쉽기 때문이다.

즉, 손실 함수와 평가 지표 모두 샘플 단위 계산을 기반으로 하지만, 손실 함수는 학습을 위한 미분 가능한 최적화 대상이고, 평가 지표는 여러 샘플의 결과를 종합해 성능을 요약하기 위한 지표라는 약간의 용도 차이가 있다. 또한 모델상에서 평가 지표를 확인할 때는, 평가 지표를 한 개가 아닌 여러 개 지표를 지정해, 다양하게 평가 결과를 확인하는 것도 가능하다.

현재 keras에서 제공하는 metric들은 다음과 같이 다양하다(물론 직접 만들어 사용하는 것도 가능하다).

```
(Base Metric class) Metric class

(Accuracy metrics) Accuracy, BinaryAccuracy, CategoricalAccuracy,
SparseCategoricalAccuracy, TopKCategoricalAccuracy,
SparseTopKCategoricalAccuracy

(Regression metrics) MeanSquaredError, RootMeanSquaredError,
MeanAbsoluteError, MeanAbsolutePercentageError, MeanSquaredLogarithmicError,
CosineSimilarity, LogCoshError, R2Score

(Classification metrics based on True/False positives & negatives) AUC,
Precision, Recall, TruePositives, TrueNegatives, FalsePositives,
FalseNegatives, PrecisionAtRecall, RecallAtPrecision,
SensitivityAtSpecificity, SpecificityAtSensitivity, F1Score, FBetaScore

(Image segmentation metrics) IoU, BinaryIoU, OneHotIoU, OneHotMeanIoU,
MeanIoU

(Hinge metrics for "maximum-margin") Hinge, SquaredHinge, CategoricalHinge

(Metric wrappers and reduction metrics) MeanMetricWrapper, Mean, Sum
```

대표적인 평가 지표에는 MAE, RMSE, BinaryAccuracy, F1Score, AUC 등을 들 수가 있다. 각 주요 평가 지표에 대한 상세 설명은 아래 표에 정리되어 있다.

평가 지표(metric)와 신경망 모델 출력 형태의 매칭 표

출력 형태	주요 평가 지표	평가 지표 설명	값 범위
회귀 (연속 값 예측)	MeanAbsoluteError (MAE)	전체 샘플에 대한 절댓값 오차 평균. 손실 함수 때와 수식은 동일함. (단 여기서, y_true, y_pred는 행렬임)	[0, ∞) (작은 게 좋음)
이진 분류 (0/1 레이블)	BinaryAccuracy	먼저, Accuracy는 [맞은 샘플 수 / 전체 샘플 수]를 말함. BinaryAccuracy는 예측값이 확률 형태일 때 threshold(0.5)를 적용해 0/1로 변환한 다음, Accuracy 값을 만들어내는 함수임	[0,1] (큰 게 좋음)
	F1Score	precision 지표(1이라고 예측한 것 중 실제로 1인 비율)과 recall 지표(실제 1중에서 모델이 1이라고 맞춘 비율)의 조화평균 값 지표. 데이터 불균형 시에 유용함	[0,1] (큰 게 좋음)
	AUC	먼저, ROC Curve는, threshold를 바꿔가며 모델의 TPR(True Positive Rate, 재현율: y축)과 FPR(False Positive Rate, 거짓 양성률: x축)의 관계를 그린 그래프임. (FPR은 낮고 TPR은 높은, "좌측 상단 점"이 최적 위치) AUC(Area Under the ROC Curve)는 ROC Curve 밑의 면적을 의미하고, 분류 모델의 전반적인 능력을 평가하는 지표임.	[0,1] (큰 게 좋음)
다중 클래스 분류	CategoricalAccuracy	이 지표는 [올바른 예측의 수 / 전체 예측 수]를 계산함. 이를 위해, 각 샘플마다 argmax(y_pred)와 argmax(y_true)를 계산하여. 두 인덱스 값이 같으면 1(예측 맞음), 다르면 0으로 계산한 뒤, 전체 평균값을 계산함. ※ argmax(argument of maximum) 함수는, 벡터에서 가장 큰 값의 인덱스를 반환하는 함수임	[0,1] (큰 게 좋음)
다중 레이블 분류	F1Score	위에서와 동일	[0,1] (큰 게 좋음)

자료: 삼성증권

중간 정리

신경망 모델의 출력 형태와 이에 따른 활성화 함수, 손실 함수, 평가 지표의 전체적인 매칭 관계를 정리하면 다음과 같다.

신경망 모델 출력 형태와 [출력층 활성화 함수, 손실 함수, 평가 지표]의 매칭 표

출력 형태	출력층 활성화 함수	손실 함수	평가 지표	신경망 모델의 예시
회귀(연속 값 예측)	없음	MSE, MAE	MAE, RMSE	10개 업종의 사후 한 달 수익률 예측 (y.shape = [, 10])
이진 분류(0/1 레이블)	sigmoid	BinaryCrossentropy	BinaryAccuracy, AUC	코스피의 사후 한 달 상승 or 하락 여부 예측 (y.shape = [, 1])
다중 클래스 분류	softmax	CategoricalCrossentropy	CategoricalAccuracy	10개 업종 중 다음 달 수익률 1위 업종 예측 (y.shape = [, 10])
다중 레이블 분류	sigmoid	BinaryCrossentropy	BinaryAccuracy, F1Score	코스피와 코스닥의 사후 한 달 상승 or 하락 여부 각각 예측 (y.shape = [, 2])

참고: 단, 회귀 모델의 출력값을 특정 범위로 제한할 때는 tanh/sigmoid 등을 쓸 수도 있음
자료: 삼성증권

모델 선언 코드

이상으로 신경망 모델 선언에 필요한 optimizer, 손실 함수, 평가 지표의 종류에 대해 알아봤다. 이번 예시의 "업종 선택 모델"은 10개 업종의 사후 한 달 수익률을 예측하는 내용의 모델이므로 회귀 모델에 해당한다. 따라서, keras.Model.compile() 메서드에서 손실 함수를 MSE, 평가 지표를 MAE로 선택했다. Optimizer에는 일반적으로 쓰이는 Adam을 선택했다.

이제 최종적으로 확정된, 신경망 모델 선언부의 코드는 다음과 같다.

```
# 코드 2: 모델 선언 최종

from tensorflow import keras
from tensorflow.keras import layers

def build_model(input_shape, output_shape):
    """지정된 입출력 크기에 맞는 keras 모델을 생성합니다."""
    model = keras.models.Sequential([
        layers.Input(shape=(input_shape,)),
        layers.Dense(16, activation='relu', kernel_initializer='he_normal', name='hidden1'),
        layers.Dropout(0.3),
        layers.Dense(12, activation='relu', kernel_initializer='he_normal', name='hidden2'),
        layers.Dropout(0.3),
        layers.Dense(output_shape, name='output_dense')
    ])

    # Adam 옵티마이저의 학습률을 지정
    optimizer = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mean_absolute_error'])
    return model
```

이 신경망 모델은 아래의 세부 세팅으로 구성되었다.

1) 층 구조는 입력층, 은닉층 1, 은닉층 2, 출력층으로 구성됨. 생성 함수에서 입출력층의 차원 수는 입력받도록 함. 실제로는 입력층 20차원 벡터, 출력층 10차원 벡터로 사용하게 됨.

2) 은닉층 1 및 은닉층 2의(로의) 활성화 함수는 relu로 설정. 이때, 랜덤한 은닉층 초기 설정 시, relu 계열에 적합한 He 초기화 방식을 사용함(kernel_initializer='he_normal').

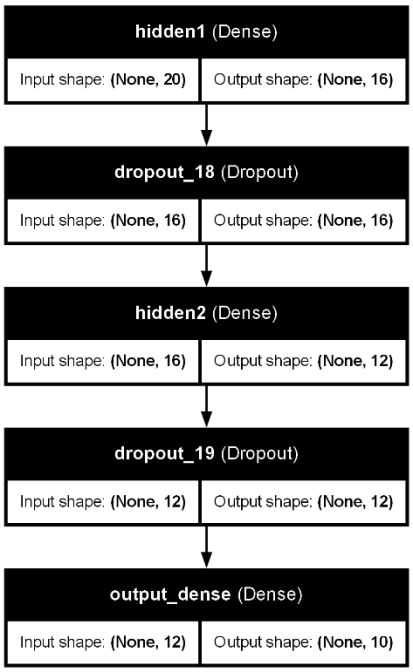
3) 출력층의 활성화 함수는 없음.

4) 입력층-은닉층 1 사이와 은닉층 1-은닉층 2 사이에 각각 30% 확률의 드롭아웃 설정

5) Optimizer는 Adam, 손실 함수는 MSE, 평가 지표는 MAE로 지정

실제 생성된 신경망 모델은 plot_model 코드를 통해서 시각화할 수도 있다. 위 모델의 시각화한 결과는 다음과 같다.

plot_model을 통한, 해당 신경망 모델 시각화



참고: input_shape=20, output_shape=10으로 지정하여 생성한 경우
자료: 삼성증권

해당 MLP 모델의 각 층의 연결에 사용된 파라미터 수(예로, ChatGPT 3.5 버전에서 1,750억 개라고 이야기하던 그 파라마터 수)는, keras.Model.summary() 메서드를 통해서 확인할 수 있다. 위 예에서 summary() 코드의 실행 결과는 다음과 같다.

```
> model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden1 (Dense)              (None, 16)                336

dropout_8 (Dropout)          (None, 16)                0

hidden2 (Dense)              (None, 12)                204

dropout_9 (Dropout)          (None, 12)                0

output_dense (Dense)         (None, 10)                130

=================================================================
Total params: 670 (2.62 KB)
Trainable params: 670 (2.62 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

참고: input_shape=20, output_shape=10으로 지정하여 생성한 경우

이 모델의 전체 파라미터 수는 336(20*16+16) + 204(16*12+12) + 130(12*10+10) = 670개가 나온다.

III. 모델 학습 및 평가

1. 학습 기본 구조

keras의 딥러닝 모델 학습 관련 API에는 compile, fit, evaluate, predict 등이 있다. 각각에 대한 짧은 설명은 다음과 같다.

1) compile: 앞서 설명한 것처럼, 모델에서 학습에 관련 세팅 설정
2) fit: train 데이터 등을 사용하여 모델 학습
3) evaluate: test(신규) 데이터를 사용해, 학습된 모델의 성능을 평가
4) predict: 입력 데이터를 받아서 출력 데이터를 예측함

딥러닝 모델의 학습을 담당하는 fit 메서드는, 보통 다음의 코드 형태를 가진다.

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    epochs=EPOCHS,
    batch_size=BATCH
)
```

우선, 딥러닝 model.fit의 학습 시에는, 학습에 사용될 train 데이터(X_train, y_train)와 함께 validation 데이터(X_valid, y_valid) 데이터가 보통 같이 들어간다.

우선 train 데이터는 신경망의 파라미터를 현실에 맞게 업데이트해 주는, 학습에 직접적으로 사용되는 데이터를 말한다. 비유하면, 선형 회귀 분석 모델이라면 회귀 계수를 계산하는 데 사용되는 전체 샘플 데이터 세트가 train 데이터에 해당하는 셈이다.

Epoch:

신경망 모델에서는 역전파 학습을 할 때, 전체 데이터를 넣어 역전파하더라도 신경망 가중치의 업데이트는 일부분만 진행이 된다(한 번의 정해진 계산 과정으로 끝나는 회귀 분석 및 일부 머신 러닝 알고리즘과는 다른 점임). 신경망 가중치의 변화가 느리게 진행되므로, 신경망이 충분히 학습되려면 여러 번의 데이터 사용이 필요하다. 즉, 반복 학습 과정이 필수적인 형태다. epoch는, 신경망의 학습 시 전체 샘플 데이터를 몇 번 반복할지를 지정하는 기준을 말한다. 즉, epoch=100이라면, 전체 샘플 데이터를 총 100번 반복해서 학습시킨다는 뜻이다. keras.Model.fit 사용 시에는, 미리 epoch 횟수를 지정하여 실행한다.

한편, 신경망 모델에서 파라미터 수, 샘플 수, epoch 수 간의 설정에 관한 완벽한 관계 공식은 따로 존재하지 않는다. epoch 횟수를 정할 때는, 여러 시행착오를 거쳐서 정하거나 아래에 나올 validation 데이터를 활용하여 적당한 횟수를 찾아가는 접근이 필요하다.

Validation 데이터:

validation 데이터는, model.fit 실행 시에 학습(파라미터 업데이트)에는 사용되지 않는 데이터다. 대신, 이 데이터는 학습 과정을 모니터링하는 역할이 1차 용도다. 모델 학습 시 각 에포크(epoch)가 끝날 때마다, 모델은 validation 데이터를 입력받아 그의 예측 결과를 확인해서, 검증 데이터 기준에서의 손실과 평가 지표(validation loss, validation metric)를 계산한다. 즉, 모델의 학습이 잘 진행되고 있는지를, 검증 데이터를 통해서 같이 체크해 가는 방식이다. model.fit()의 리턴 값인 history.history에는 각 epoch마다의 loss, metric, val_loss, val_metric 정보 변화가 담기게 된다. 이걸로 학습 곡선(learning curve)의 변화를 확인할 수 있다.

한편, validation 데이터는, 과적합(overfitting)을 막기 위한 Callback 기능 등에서도 활용된다. Callback 내용은 아래에서 상세히 설명한다.

Callback:

우선 model.fit 상에서의 Callback 기능 자체는, 학습 도중의 특정 시점(epoch의 시작 및 끝, 배치의 시작 및 끝 등)에서 자동으로 호출되는 함수를 말한다. Callback 기능 자체는, 학습하는 동안 옆에서 지켜보는 감독관이나 비서의 역할이라고 할 수 있다. 이 Callback의 주요 목적에는 1) 과적합 방지를 위한 학습 조기 중단(Early Stopping), 2) 학습률 조정(Learning Rate Scheduling), 3) 모델 저장(Checkpointing) 등 다양한 용도가 있다.

조기 중단

신경망의 학습은, 샘플 데이터를 여러 번 역전파 학습에 넣어서 신경망의 가중치를 조절하는 방식이다. 그런데, 같은 데이터를 너무 자주 학습하면, 주어진 데이터에만 100% 완벽하게 학습이 되어서 새로운 데이터에서는 성능이 급격히 떨어지는 현상인 "과적합(overfitting)"이 나타날 수 있다. 이를 막기 위한 도구로 "조기 중단(Early Stopping)" 기법이 있다. 조기 중단은, 모델의 현실 성능이 어떤지를 지켜보고 있다가, 현실 성능이 떨어질 기미가 보이면 그때 모델 학습을 조기에 중단시키는 방식이다. 즉, train 데이터를 가지고 학습하는 도중에, validation 데이터를 사용하여 모델 상황(val_loss가 일반적임)를 모니터링하고 있다가, 모니터링 지표(val_loss)의 수치 개선이 '지정한 epoch 횟수' 동안 나타나지 않으면, 즉 지정 횟수 동안 val_loss가 나빠지면, 그때 바로 학습을 중단하는 기능이다. 이를 통해 과적합을 예방할 수 있다.

학습률 조정

학습률이란, 경사 하강법에서 가중치를 업데이트할 때 얼마나 큰 폭으로 이동할지에 대한 수치다. 모델의 빠르고 안정적인 학습을 위해서, Callback 기능으로 학습률 조정을 할 수 있다. 보통의 경우, 학습 초기에는 학습률을 크게 지정해 학습을 빠르게 하고, 후반에는 학습률을 작게 지정해서 최적값을 더 정확히 찾는 방식이 사용된다. keras의 대표적인 학습률 조정 Callback 기능에는 ReduceLROnPlateau가 있는데, 이 기능은 val_loss를 기준으로, 지정한 epoch 횟수 동안 지표 개선이 없으면 학습률을 계속 여러 번 감소시키는 기능을 담당한다.

Batch size(배치 크기):

신경망 모델의 역전파 학습에서는, 한 번의 작업에서 1개의 샘플로 학습할 수도 있지만, 여러 개의 샘플 묶음으로 학습이 가능하고 보통 그렇게 한다(예시 신경망에서 1개 샘플은 1xN의 벡터이며, 전체 샘플은 MxN의 행렬 구조다. 1개 샘플 학습은 1xN의 벡터로 역전파하는 것이고, L개 샘플 묶음의 학습은 LxN의 행렬을 역전파에 넣는 방식이다). batch size를 작게 하면 모델 성능을 올릴 수 있으나 학습 속도가 느려지게 되고, batch size를 크

게 하면 속도가 빠를 수 있으나 성능이 나빠질 수 있다. keras.Model.fit에서 batch_size의 디폴트값은 32이고, 필요시에 직접 지정할 수 있다.

한편, keras.Model.fit에는 shuffle=True(디폴트)/False 옵션이 있다. 이는 학습 시에 입력되는 데이터를 랜덤하게 섞어서 쓸 건지, 순서대로 넣을 건지를 지정하는 옵션이다. 샘플 데이터들이 서로 독립인 경우에는 shuffle=True가 더 좋다. 그러나, 시계열 데이터를 사용할 때는 서로 독립의 가정이 깨지는 경우가 많으므로, shuffle 하여 학습하면 데이터 유출(data leakage)이 발생할 수 있다. 따라서, 시계열 데이터에서는 shuffle=False의 지정이 필요하다.

이상으로, 일반적인 keras 상의 신경망 학습에 사용되는 용어와 내용에 대해 알아보았다.

이번 "업종 선택 모델"에 맞게 실제로 구성한, 기본적인 모델 학습 및 평가 루틴의 코드는 다음과 같다.

```
# 코드 3: 모델 학습 및 평가 기본 코드

        X_train_raw, X_val_raw = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        # 데이터 표준화: train 데이터에만 fit하고, train/val 모두 transform
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train_raw)
        X_val = scaler.transform(X_val_raw)

        # 모델 새로 생성
        model = build_model(input_shape=input_shape, output_shape=output_shape)

        # 콜백 정의: 조기 종료 및 학습률 스케줄러
        # 검증 손실이 10번의 epoch 동안 개선되지 않으면 학습을 조기 종료합니다.
        early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True, monitor='val_loss')

        # 검증 손실이 5번의 epoch 동안 개선되지 않으면 학습률을 50%씩 감소시킵니다.
        reduce_lr_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001)

        callbacks = [early_stopping_cb, reduce_lr_cb]

        # 학습 (시계열이므로 shuffle=False)
        # epochs를 충분히 늘려주고, 조기 종료 콜백이 최적의 시점에 학습을 멈추도록 합니다.
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=200, # 충분히 큰 값으로 설정
            batch_size=32,
            callbacks=callbacks,
            shuffle=False,
            verbose=0
            )

        # 5. 해당 fold 검증 성능 기록
        val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
        print(f"val_loss: {val_loss:.4f}, val_mae: {val_mae:.4f}")
```

위 코드 중 아직 언급하지 않은 부분은 "StandardScaler"만 있다. StandardScaler 클래스는 말 그대로 feature 데이터를 평균 0, 표준편차 1로 표준화하는 작업을 한다(샘플 행렬에서 각 열(feature) 별로 표준화를 실시함). 신경망 모델 학습은 반복적인 행렬 계산을 계속 진행하는데, 이때 feature 간의(행렬의 각 열 간의) 스케일이 서로 다르면 학습이 불안정해지고 학습 효율이 떨어진다. 따라 신경망 모델에서는 X(feature) 데이터에 대해서 항상 표준화를 쓰는 것이 일반적이다(Scaler 방식에는 StandardScaler 말고 RobustScaler 등 여러 가지가 존재함). 한편, y(target) 데이터에 대해서는 표준화 작업이 필수가 아니다. 이번 예제에서도 y 변수인 "업종별 사후 초과 수익률"은 평균값이 0에 근사하고 수치 범위가 작은 편이므로, 표준화를 강제할 필요가 없다고 볼 수 있다.

위의 코드 예에서, 학습에는 train 데이터만 사용되므로, StandardScaler 작업의 fit_transform() 단계에는 train 데이터만이 사용되어야 한다. validation 데이터에 대해서는, train 데이터로 설정된 scaler(StandardScaler)의 단순 transform() 메서드를 사용해 동일한 수식으로 표준화하여야 한다.

2. TimeSeriesSplit

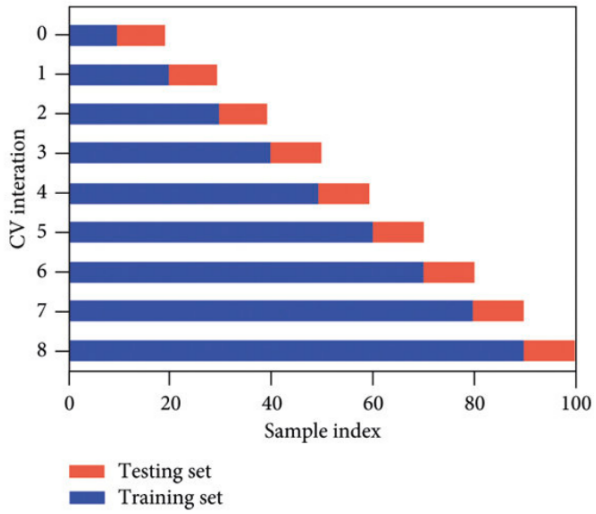
금융 시계열에 관한 딥러닝이 아닌, "일반적인 딥러닝 모델"에서는 학습 기능의 model.fit, 평가 기능의 model.evalute 작업을 한 번씩만 하는 것이 더 일반적이다. 그러나, 금융 시계열에 대한 딥러닝 모델에서는, TimeSeriesSplit을 사용하여 시계열 교차 검증을 사용하는 경우가 많다.

우선, 검증(validation)이란, 훈련이 끝난 모델에 대해 validation 데이터를 가지고 모델의 성능을 평가하고 검증하는 절차를 말한다. 교차 검증(cross validation)이란, 제한된 데이터를 여러 부분으로 나눠서 검증 작업을 반복적으로 해 안정적으로 성능을 평가하는 방식이다. 교차 검증의 가장 대표적인 형태로는 K-Fold 교차 검증이 있다. K-Fold 교차 검증은, 전체 데이터를 k개의 폴드로 나누어, k번의 반복 동안 k-1개의 폴드는 train 데이터로 나머지 1개의 폴드는 valid/test 데이터로 사용하는 검증 방식이다.

서로 독립이 아닌 "시계열 데이터"에 대한 모델일 경우에는, 교차 검증 시 보통 TimeSeriesSplit이라고하는 별도의 시계열 교차 검증 방법을 사용한다. 일반 K-Fold 교차 검증은 데이터를 섞어서 사용하기 때문에, 이를 시계열 데이터에 적용하면 미래의 데이터를 보고 과거를 예측하는 "데이터 유출(data leakage)" 문제가 발생한다. 이런 데이터 유출은 모델 성능을 과대평가하기 때문에 꼭 피해야 한다. 따라서, 시계열 데이터에 대한 교차 검증 시에는, 항상 train 데이터는 더 과거이고 valid 데이터는 train보다 미래인 구조 형태를 유지해야 한다. 이때 사용하는 기법이 TimeSeriesSplit이다(KFold/cross_validate가 일반 교차 검증, TimeSeriesSplit이 시계열 교차 검증임).

TimeSeriesSplit은, N번의 교차 검증을 할 때 전체 데이터를 N+1개 폴드로 나누고, 첫 번째는 [train: 1번 폴드, valid: 2번 폴드] 검증, 두 번째는 [train: 1번과 2번 폴드, valid: 3번 폴드] 검증의 방식으로, train 데이터를 늘려나가며 반복 검증하는 기법이다(아래의 개념도 참고). 즉, TimeSeriesSplit은 Expanding Window(확장 윈도우) 형태로, 학습 데이터를 늘리면서 모델 성능을 반복 평가하는 방식에 해당한다.

TimeSeriesSplit 개념도



자료: ResearchGate

실제로 금융 시계열 분석에서는 TimeSeriesSplit을 쓰는 것이 일반적이다. 이유는, 금융 모델은 실제 투자 환경을 얼마나 잘 모방하는가를 중요시하기 때문이다. 전통적인 백테스팅이 시간 순서로 투자 모델의 성능을 평가하는 것과 유사하게, TimeSeriesSplit은 미래 데이터 유출을 막고 시간 순서로 데이터가 증가하는 학습 환경을 만들어 시간 변화에 따라서 모델 성능이 변화하는 것을 보여주는 장점이 있어서 많이 사용된다.

한편, TimeSereisSplit 클래스에는 gap 파라미터가 있는데, 이는 엄격한 미래 데이터 유출을 막기 위해 필요한 장치이다. 보통, 금융 투자 관련 모델에서 y 데이터(target)에는, 예를 들어 사후 한 달간의 주식 수익률, 사후 일주일간의 ETF 수익률 등의 "미래 기간의 정보"를 넣게 된다. 그리고, 보통의 샘플 데이터(X와 y의 조합)는 데이터 수를 늘리기 위해, 더 작은 frequency(주기)로 데이터가 구성되는 경우가 많다. 예를 들어, 샘플 데이터의 frequency가 일별이고 이중 y 데이터의 정보가 사후 일주일 수익률이라고 하면, y_i에는 [t ~ t+7 일자]의 수익률이 들어가고 y_{i+1}에는 [t+1 ~ t+8 일자]의 수익률이 들어가게 된다. 이 경우, y_i와 y_{i+1} 간에는 서로 독립이 아니고 상관관계가 발생한다. 이 상황에서 모델 학습 시 y_i까지 학습 데이터로 사용하고, 모델 평가에 y_{i+1}를 바로 넣게 되면, 미래 데이터 유출로 인해 모델 성능의 과대 평가가 발생한다.

TimeSereisSplit의 gap 파라미터는, train 구간과 validation 구간 사이의 인접 데이터를 n개만큼 제거(purge)하는 옵션이다. 이를 통해서, 인접한 시계열 데이터 간에 상관관계가 있어서 미래 데이터 유출이 생길 수 있는 가능성을 차단하는 역할을 하므로 자주 사용되는 편이다(물론, 미래 데이터 유출 가능성이 없다면 gap을 안써도 됨).

TimeSeriesSplit을 사용하는 일반적인 코드 구문은 다음과 같다.

```
# 코드 4: TimeSeriesSplit 코드

from sklearn.model_selection import TimeSeriesSplit

X = …
y = …

tscv = TimeSeriesSplit(n_splits=5, gap=G)

all_val_losses = []

# Time Series Cross Validation (Expanding)
for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):
    print(f"\n===== Fold {fold}/5 =====")

    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    model = build_model()

    # 학습 (시계열이므로 shuffle=False)
    history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=E, shuffle=False)

    # 해당 fold 검증 성능 기록
    val_loss = model.evaluate(X_val, y_val)
    print("Fold {fold} - val_loss: ", val_loss)

    all_val_losses.append(val_loss)


# 전체 fold 성능 요약
print("mean val_loss: ", np.mean(all_val_losses))
```

TimeSeriesSplit.split(X) 메서드는 generator(for문에서 반복되는 변수를 만들어내는 객체)를 만들게 되는데 이를 for 문에 사용하면, for 문 안에서 폴드 횟수만큼의 train 인덱스 배열과 valid 인덱스 배열을 매번 만들어낸다. 이를 가지고 모델 학습 및 검증 코드를 반복 실행하여, 교차 검증을 진행하는 방식이다.

3. 학습 및 평가 모듈 전체

지금까지 살펴본 1) 모델 학습의 기본 구조와 2) TimeSeriesSplit 루틴을 결합하면, 실무적인 모델 학습 모듈이 완성된다. 최종적인 학습 및 평가 코드는 다음과 같다.

```
# 코드 5: 학습 및 평가 최종 코드

    input_shape = X.shape[1]
    output_shape = y.shape[1]

    tscv = TimeSeriesSplit(n_splits=5, gap=4)

    all_val_losses = []
    all_val_maes = []


    # Time Series Cross Validation (Expanding)
    for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):
        print(f"\n===== Fold {fold}/5 =====")
        print(f"Train index: {train_idx[0]} ~ {train_idx[-1]}")
        print(f"Valid index: {val_idx[0]} ~ {val_idx[-1]}")

        X_train_raw, X_val_raw = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        # 데이터 표준화: train 데이터에만 fit하고, train/val 모두 transform
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train_raw)
        X_val = scaler.transform(X_val_raw)

        # 매 fold마다 모델 새로 생성 (가중치 초기화)
        model = build_model(input_shape=input_shape, output_shape=output_shape)

        # 콜백 정의: 조기 종료 및 학습률 스케줄러
        # 검증 손실이 10번의 epoch 동안 개선되지 않으면 학습을 조기 종료합니다.
        early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True, monitor='val_loss')

        # 검증 손실이 5번의 epoch 동안 개선되지 않으면 학습률을 50%씩 감소시킵니다.
        reduce_lr_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001)

        callbacks = [early_stopping_cb, reduce_lr_cb]

        # 학습 (시계열이므로 shuffle=False)
        # epochs를 충분히 늘려주고, 조기 종료 콜백이 최적의 시점에 학습을 멈추도록 합니다.
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=200, # 충분히 큰 값으로 설정
            batch_size=32,
            callbacks=callbacks,
            shuffle=False,
            verbose=0  # 필요하면 1로 바꾸기
        )

        # 해당 fold 검증 성능 기록
        val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
        print(f"Fold {fold} - val_loss: {val_loss:.4f}, val_mae: {val_mae:.4f}")
        all_val_losses.append(val_loss)
```

```
    all_val_maes.append(val_mae)


# ------------------------------
# 마지막 Fold의 학습 과정 시각화
# ------------------------------
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss (Last Fold)')
plt.legend()
plt.show()

# ------------------------------
# 전체 fold 성능 요약
# ------------------------------
print("\n===== Cross-Validation Result (Expanding Window) =====")
print(f"val_loss (per fold): {[f'{v:.4f}' for v in all_val_losses]}")
print(f"val_mae  (per fold): {[f'{v:.4f}' for v in all_val_maes]}")
print(f"mean val_loss: {np.mean(all_val_losses):.4f}")
print(f"mean val_mae : {np.mean(all_val_maes):.4f}")
```

코드의 전체적인 구성인 1) 변수 초기화 및 설정 단계, 2) 교차 검증 반복문, 3) 마지막 fold 학습 과정 그래프, 4) 전체 fold 성능 평가 요약으로 나눠진다.

최초 세팅 이후, "for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):" 코드를 통해서 교차 검증을 반복할 영역이 설정된다. 이 영역 안에서, 1) test 데이터에 대한 표준화, 2) 콜백 정의, 3) 모델 학습 및 검증, 4) 검증 성능 출력의 루틴이 반복된다. 이때 사용되는 model.evaluate()을 통해서는, validation 데이터 전체에 대한 손실값의 평균(val_loss)과 평가 지표(여기서는 val_mae)가 계산된다. model.evaluate의 리턴 값은 항상, 손실값이 먼저 나오고, 그 뒤에는 compile(metrics=[])에서 지정한 metric들이 순서대로 나오게 된다.

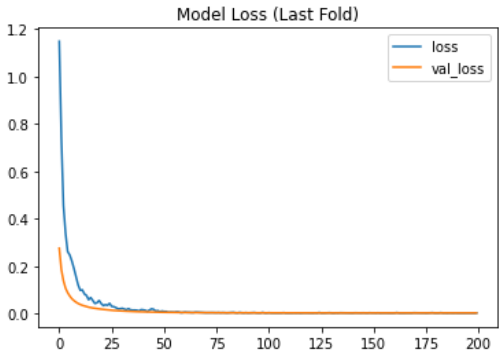
그 다음으로 "마지막 fold 학습 과정 시각화"에서는 마지막 model.fit() 결과를 담은 history 객체를 통해서, 마지막 fold 진행 당시의 학습 그래프를 나타낸다. history.history['loss']는 학습 시의 각 epoch마다 측정된 train 데이터상의 손실값(loss)를 담고 있는 리스트다. history.history['val_loss']는 학습 시의 각 epoch가 끝날 때마다 valid 데이터로 검증 평가한 손실값(val_loss)를 담고 있는 리스트다. 이들 데이터를 시각화하여, 모델 학습이 진행될 때마다 train 데이터에서의 손실값과 valid 데이터의 손실값이 잘 줄어드는지를 확인할 수 있다. history.history 딕셔너리(매핑 자료형)의 데이터 목록은, history.history.keys()로 확인 가능하다.

마지막 "전체 fold 성능 요약"에서는, 각 fold 별로 진행했던 valid loss와 valid 평가지표를 모아서, 전체 내용을 출력한다.

IV. 모델 실행 결과

전체 코드의 내용은, 모델에서 사용할 X, y 데이터를 불러오고, MLP 모델 생성 함수로 정의한 다음, 모델 학습 및 검증에 대한 시계열 교차 검증을 진행하여, 업종 투자 모델이 잘 작동하는지 확인하는 내용까지를 담고 있다. 이 코드를 실행한 결과는, 다음과 같이 출력된다.

```
# 실행 결과

===== Fold 1/5 =====
Train index: 0 ~ 171
Valid index: 176 ~ 347
Fold 1 - val_loss: 0.0121, val_mae: 0.0715

===== Fold 2/5 =====
Train index: 0 ~ 343
Valid index: 348 ~ 519
Fold 2 - val_loss: 0.0028, val_mae: 0.0399

===== Fold 3/5 =====
Train index: 0 ~ 515
Valid index: 520 ~ 691
Fold 3 - val_loss: 0.0023, val_mae: 0.0370

===== Fold 4/5 =====
Train index: 0 ~ 687
Valid index: 692 ~ 863
Fold 4 - val_loss: 0.0626, val_mae: 0.0620

===== Fold 5/5 =====
Train index: 0 ~ 859
Valid index: 864 ~ 1035
Fold 5 - val_loss: 0.0033, val_mae: 0.0429
```




```
===== Cross-Validation Result (Expanding Window) =====
val_loss (per fold): ['0.0121', '0.0028', '0.0023', '0.0626', '0.0033']
val_mae  (per fold): ['0.0715', '0.0399', '0.0370', '0.0620', '0.0429']
mean val_loss: 0.0166
mean val_mae : 0.0506
```

첫 번째 교차 검증에서는, 사용할 데이터로 train_index = 0~171, valid_index=176~347가 사용되었다. 실제 날짜상으로는, 2005/12/30 ~ 2009/4/10 기간의 데이터를 가지고 학습한 뒤, 2009/5/15 ~ 2012/8/24 기간의 검증 데이터에 대해서 모델 성능을 검증했다(y 데이터가 사후 한 달간 수익률이므로, gap 옵션을 통해서 2009/4/17~2009/5/8 구간 데이터를 제외시켰음). 이때 MLP 모델이 업종별 사후 상대 수익률을 예측하는 성능에 대해서 보자면, 평균적으로 절대 오차가 7.15% 정도로 발생했다(val_mae로 확인).

두 번째 교차 검증에서는, 데이터로 train_index = 0~343, valid_index=348~519가 사용되었다. 즉, 모델 학습에는 2005/12/30 ~ 2012/7/27 기간 데이터가 입력된 다음, 2012/8/31 ~ 2015/12/11 기간의 검증 데이터에 대해서 모델 성능을 검증했다. 2006년에서 2012년 중순까지의 패턴을 가지고, 2012년 중순에서 2015년 말까지 업종별 사후 상대 수익률을 예측하는 이 접근에서는, 절대 오차 평균이 3.99% 정도로 발생했다. 이렇게 5개의 교차 검증의 결과가 각각 출력된다.

그다음의 그래프 부분은, 마지막 폴드 단계 시의 history 정보를 사용해, 마지막 폴드에서의 각 epoch 진행에 따른 모델의 개선 여부를 시각화한 것이다. 그래프상에서 loss 선을 먼저 보면, epoch가 1부터 200까지 진행될 동안, loss 수치 즉 train 데이터를 사용한 손실값이 처음에는 가파르게 떨어지다가 이후에는 완만한 경로로 변화하는 것을 볼 수 있다. val_loss 선을 볼 때도, epoch가 1부터 200까지 진행될 동안 val_loss 값은 처음의 빠른 하락 후 이후에는 변화가 정체되는 것을 볼 수 있다.

예시로 사용한 "업종 선택 모델"의 모델 검증 결과를 보면, 모델의 예측력이 아주 우수하다고 보기는 어렵다. 실전적인 모델을 개발할 때는, 다양한 시도를 통해 모델 성능을 올리려는 노력이 지속적으로 요구된다고 하겠다. 예를 들어, 현재의 MLP 모델에서는, 업종의 미래 수익률 예측 시 업종의 올해 및 내년 영업이익 변화율 정보만 input으로 사용하는 제한된 상황이다. 업종의 미래 주가 예측에는, 순이익 컨센서스의 변화, 분기 실적 상의 서프라이즈 여부, 주가 모멘텀 보유 여부, 밸류에이션 매력도 등 다양한 정보를 추가하면 예측 가능성을 더 올릴 여지가 있다. 따라서, feature(input) 데이터를 더 늘리거나, feature에 가공 및 변형을 하는 식으로 모델 성능을 올리는 노력을 해볼 수 있다. 혹은 하이퍼 파라미터 튜닝을 통한 기존 모델 성능을 개선하는 식의 방법도 가능하겠다.

V. MLP 모델링 이후의 작업

지금까지의 내용이 MLP 모델링 부분에 해당한다. 즉, 업종별 이익모멘텀 정보를 받아서, 업종별 사후 수익률을 예측하는 모델의 개발 및 평가까지가 이에 해당한다.

전통적인 금융 투자 백테스팅 모델까지 만들기 위해서는, MLP 모델에 의한 업종별 사후 수익률을 가지고, 실제 투자할 업종을 선택하고 수익률을 계산하는 백테스팅 프로세스가 필요하다. 이는 전통적인 백테스팅 구축과 동일하다.

전체 백테스팅 프로세스:

1. 과거 특정 기간 동안에, 정해진 주기(예를 들어 1개월)로 투자 의사 결정 및 수익률을 계산하는 반복 루틴을 진행함.

2. 반복 루틴 시, t시점에서 투자 결정을 할 때는, 우선 [(학습 최초 시점) ~(t-32일)] 구간의 장기 데이터(X와 y = 업종별 이익모멘텀과 업종별 사후 한 달 수익률 정보)를 가지고, MLP 모델을 학습시킴. 사후 한 달 수익률 정보가 y에 사용되므로, 당시에 알 수 없는 [(t-31일) ~ (t-1일)]의 데이터는 제외하고(purge 혹은 gap 적용) 그 전까지의 데이터로만 학습해야 함. 또한, 앞서 모델의 시계열 교차 검증 시에도 expanding window 형태로 데이터를 계속 늘리는 방식을 썼기 때문에, 백테스팅 단계에서도 동일한 expanding window 형태의 학습 방식을 유지함.

3. 학습한 후, t시점의 업종별 이익모멘텀 input 정보를 모델에서 넣어서, 현재 기준의 업종 사후 수익률 수치를 예측해 냄(이때는 model.predict(X) 메서드를 사용함).

4. 사후 수익률 예측치가 높은 n개의 업종만 뽑아서, 이들 업종에만 동일 가중 투자하는 것으로 가정한 다음, 이 투자 포트폴리오의 사후 한 달 수익률을 산출함.

5. 반복 루틴으로 2번~4번의 단계를, 날짜를 전진하며 계속해서 진행함. 투자 결과를 계속 누적하여, 사후적인 장기 수익률 및 성과 지표들을 만들어 냄.

여기까지 진행하는 것이, 실제 투자 모델의 전체 구성이다.

물론, 표준화된 딥러닝 모델의 평가 및 검증 루틴까지만 쓰더라도, 업종 수익률 예측 모델의 예측 성능을 어느 정도는 파악할 수 있다. 하지만, 실제 해당 전략을 썼을 때, 실제 수익률이 얼마까지 나올 수 있는지 혹은 투자의 위험 요소가 어떤지는, 전통적인 백테스팅 프로세스를 별도로 추가하는 것을 통해 확실히 알 수 있다. 이런 절차와 관련해서, 최근에는 딥러닝의 모델 평가 및 검증과 투자의 백테스팅 과정을 같이 한 단계로 줄이려는 시도가 학계에서 나오고도 있다. 별도의 loss function이나 metric을 정교하게 맞춤 제작하여, 표준적인 딥러닝 학습 및 검증 단계에서 금융 투자의 백테스팅까지 일괄로 진행하는 접근법을 사용하는 논문들이 나타나는 추세다.

결론:

지금까지 계량 투자 모델 개발 시 "MLP 신경망 모델"을 접목하는 방법에 대해, 프로그래밍 상세 사용법 위주로 설명을 진행하였다. 향후에는 더 복잡하고 진화한 신경망 모델들을 퀀트 전략에 활용하는 실무 절차, 또는 계량 모델을 접목하는 새로운 아이디어 등을 추가로 전달할 계획에 있다.

VI. Appendix: MLP 모델의 전체 코드

```
# 전체 코드

# -*- coding: utf-8-sig -*-
#%%

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# ------------------------------
# 데이터 로딩 함수
# ------------------------------
def load_data(file_path='data_weekly.xlsx'):
    """데이터 파일에서 피처와 레이블을 로드합니다."""

    # feature df와 섹터 사후 상대 수익률 df
    X_df = pd.read_excel(file_path, sheet_name='X_df', skiprows=13, usecols='A:U', index_col=0)
    y_df = pd.read_excel(file_path, sheet_name='y_df', skiprows=13, usecols='A:K', index_col=0)

    return X_df, y_df


# ------------------------------
# MLP 모델 정의 함수
# ------------------------------
def build_model(input_shape, output_shape):
    """지정된 입출력 크기에 맞는 Keras 모델을 생성합니다."""
    model = keras.models.Sequential([
        layers.Input(shape=(input_shape,)),
        layers.Dense(16, activation='relu', kernel_initializer='he_normal', name='hidden1'),
        layers.Dropout(0.3),
        layers.Dense(12, activation='relu', kernel_initializer='he_normal', name='hidden2'),
        layers.Dropout(0.3),
        layers.Dense(output_shape, name='output_dense')
    ])

    # Adam 옵티마이저의 학습률을 지정
    optimizer = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mean_absolute_error'])
    return model


# ------------------------------
# MAIN
# ------------------------------
def main():
    """메인 프로세스"""
```

```
# ------------------------------
# 0. 데이터 로딩
# ------------------------------
X_df, y_df = load_data()


# ------------------------------
# 1. MLP 모델 학습 및 평가
# ------------------------------

# 준비
X = X_df.values.astype(np.float32)
y = y_df.values.astype(np.float32)

input_shape = X.shape[1]
output_shape = y.shape[1]

tscv = TimeSeriesSplit(n_splits=5, gap=4)

all_val_losses = []
all_val_maes = []

# Time Series Cross Validation (Expanding)
for fold, (train_idx, val_idx) in enumerate(tscv.split(X), start=1):
    print(f"\n===== Fold {fold}/5 =====")
    print(f"Train index: {train_idx[0]} ~ {train_idx[-1]}")
    print(f"Valid index: {val_idx[0]} ~ {val_idx[-1]}")

    X_train_raw, X_val_raw = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    # 데이터 표준화: train 데이터에만 fit하고, train/val 모두 transform
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train_raw)
    X_val = scaler.transform(X_val_raw)

    # 매 fold마다 모델 새로 생성 (가중치 초기화)
    model = build_model(input_shape=input_shape, output_shape=output_shape)

    # 콜백 정의: 조기 종료 및 학습률 스케줄러
    # 검증 손실이 10번의 epoch 동안 개선되지 않으면 학습을 조기 종료합니다.
    early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True, monitor='val_loss')

    # 검증 손실이 5번의 epoch 동안 개선되지 않으면 학습률을 50%씩 감소시킵니다.
    reduce_lr_cb = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=0.00001)

    callbacks = [early_stopping_cb, reduce_lr_cb]

    # 학습 (시계열이므로 shuffle=False)
    # epochs를 충분히 늘려주고, 조기 종료 콜백이 최적의 시점에 학습을 멈추도록 합니다.
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=200, # 충분히 큰 값으로 설정
        batch_size=32,
        callbacks=callbacks,
        shuffle=False,
        verbose=0  # 필요하면 1로 바꾸기
    )
```

```
        # 해당 fold 검증 성능 기록
        val_loss, val_mae = model.evaluate(X_val, y_val, verbose=0)
        print(f"Fold {fold} - val_loss: {val_loss:.4f}, val_mae: {val_mae:.4f}")
        all_val_losses.append(val_loss)
        all_val_maes.append(val_mae)


    # ------------------------------
    # 마지막 Fold의 학습 과정 시각화
    # ------------------------------
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.title('Model Loss (Last Fold)')
    plt.legend()
    plt.show()

    # ------------------------------
    # 전체 fold 성능 요약
    # ------------------------------
    print("\n===== Cross-Validation Result (Expanding Window) =====")
    print(f"val_loss (per fold): {[f'{v:.4f}' for v in all_val_losses]}")
    print(f"val_mae  (per fold): {[f'{v:.4f}' for v in all_val_maes]}")
    print(f"mean val_loss: {np.mean(all_val_losses):.4f}")
    print(f"mean val_mae : {np.mean(all_val_maes):.4f}")


#%%
if __name__ == '__main__':
    main()
```

삼성 All That Quant

[팩터 모델 실전 전략 및 이론]

25/10/27 "영업이익 증가율 모멘텀 팩터 - 신규 추천 전략" - High return, Low correlation의 고효율 팩터전략 신규 개발 및 제시
25/8/19 "Long-only 전략 상세 분석법 (롱 팩터 전략 시리즈 #2)" - 효율적 Long-biased 전략을 위한 프로세스와 추천 전략 제시
25/7/21 "Long-only 전략과 비대칭 알파 ((롱 팩터 전략 시리즈 #1)" - 코스피 5000 시대를 대비하는, long-biased 전략 집중 분석
24/11/20 "연구개발 팩터 투자전략" - R&D에 기반한 신규 팩터 전략을 개발하고, 이의 특징과 활용 방법 등을 제시함
24/8/8 "코스닥 투자자를 위한 퀀트 전략 Redux" – 코스닥 시장에서 변화하는 팩터 및 스타일 전략의 특징을 상세 재정리함
24/7/2 "연기금 투자자를 위한 장기 퀀트 전략" – 투자 horizon을 장기로 가져갔을 때의 팩터 투자 전략의 변화를 상세하게 비교 정리함
23/11/30 "퀀트 전략에서의 업종 특성 고려, 2편 - 업종 대비 상대 밸류에이션 팩터의 개선 전략" – 상대 밸류 팩터에서의 업종 특성 고려 전략 제시
23/9/4 "퀀트 전략에서의 업종 특성 고려, 1편 - 저P/B 팩터의 업종 조정 전략" – 한국 업종의 특성을 감안한 수정된 밸류에이션 전략 제시
23/7/26 "Data-driven Quant Model 2 - #연속 국면 #머신러닝 #부스팅 기법 #동적 팩터 모델" –부스팅 알고리즘 기반의 팩터 로테이션 모델 제시
23/5/17 "퀄리티 팩터는 정말 퀄리티 있는 전략인가" - 퀄리티 팩터가 가지는 고유 특성과 세부 전략 간의 차이점을 비교 정리
23/1/4 "포트 전략 - 롱/숏 시그널로 롱 포트폴리오 구성하기" - 롱/숏 전략을 Long-only 포트로 전환하는 효율적인 알고리즘을 신규 제시
22/11/21 "Data-driven Quant Model - 머신러닝 기법으로 시장의 파도에 올라타기" – 머신러닝 기법에 의한 데이터 기반의 팩터 로테이션 모델 제시
22/8/17 "변동성 구간에서의 주식 스타일 특징 - 낙폭 과대주 팩터 활용법" - 팩터 투자 틀 안에서 낙폭 과대주 팩터의 필요성 입증과 활용법 제시
22/5/25 "경기 국면에 따른 리스크 팩터 배분 전략" - 리스크 팩터 전반에 대한 설명과 함께 팩터와 경기 국면을 결합한 투자 전략을 제시
21/2/24 "이익 모멘텀의 차별화" – 종목 특성을 감안한, 변동성 조정 이익 모멘텀 팩터 전략을 제시
21/1/11 "주식 스타일 Fact Book - 주식 스타일 및 매크로 변수 상관성 팩트 체크" – 주식 스타일 분석 체계의 정리 및 매크로 변수 연결
19/4/15 "팩터 모델의 구조와 해설 - Factor Models for Asset Returns" – 팩터 모델 이론에 대한 상세 설명 자료
18/7/3 "Fama-French 3 팩터 모델 – 후편" - Fama & French의 3팩터 모델 1993년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
18/5/8 "Fama-French 3 팩터 모델 – 전편" – Fama & French의 3팩터 모델 1992년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
15/10/20 "가치주 투자와 금리의 연관성" – 금리와 가치/성장 스타일의 인과 분석을, Equity Duration 개념을 활용하여 국내 최초로 제시
14/11/24 "투자 Horizon에 따른 퀀트전략 변화 - Advanced Factor Book" – 1M ~ 6M의 투자 기간별로 팩터 모델 백테스팅 분석
13/4/23 "매크로 기반 Dynamic Quant 모델" – 매크로 국면 판단에 기반한 동적 팩터 투자 모델 정립
12/2/27 "스타일 로테이션의 허와 실" - 스타일 로테이션 전략의 형태로서 모멘텀 방식, 리버전(컨트래리안) 방식, Asness 방식을 각기 비교 분석

[퀀트 전략 일반]

25/11/14 "자산배분 전략의 체계" - 자산배분에 대한 기본적인 개념과 접근법을 체계적으로 정리함
25/9/2 "머신 러닝 애널리스트, 파트 1" - 주식 시장의 기업 펀더멘털 분석에서, 머신러닝 및 AI를 활용하는 접근법 제시
25/7/29 " KR-FinBERT의 활용법" – 센티먼트 분석 AI를 손쉽게 사용하는 방법을 정리함
25/5/20 "Corporate Action Book ver 2.0" - 여러 corporate action 이벤트에 대한 메커니즘 설명과 시장 반응의 최근 패턴을 정리함
25/3/20 "MSCI 지수 업데이트 ver 5.0" - MSCI 지수의 최근 이슈과 방법론 상의 변화, 정기 리뷰 로직의 요약 정리 제공
25/1/23 "MSCI 미국 지수 2월 리뷰 편출입 최종 전망" - 24년말부터 MSCI USA 종목 편출입 전망 리포트를 최초 개시함
25/1/14 "TAX 전략 2. 소득세뿐 아니라 건강보험료도 존재한다" - 개인 투자자 관점에서의 건강보험료 및 광의의 세금에 대한 설명 및 전략 자료
24/11/7 "AI와 퀀트 투자" – AI에 의한 세상의 변화와 알고리즘에 대한 이해, 퀀트 투자와의 접목법을 정리함
24/5/27 "KOSPI 지수 장기 수익률 성과 분해" – KOSPI 장기 수익률을 기업이익 증가, 주식수 변동, 밸류에이션으로 분해하여 부진 원인을 정리
24/5/8 "주당 지표(per share) 기준의 펀더멘털 접근법" – 지수 차원에서의 주당 지표 산출법과 이를 통한 시장 분석, 관련 팩터 전략 등을 정리
24/1/29 "이전 상장 이벤트 및 활용 전략" - 코스피 이전 상장 종목에 대한 주가 변화 결정 요인 상세 분석
23/7/11 "한미 간 자사주 매입의 구조적 차이" – 자사주 매입에 대한 한국과 미국 제도를 비교하고, 시가총액 정의 차이에 따른 문제점을 정리
23/4/17 "NSS(Nelson Siegel Svensson) 모델을 활용한 채권 퀀트 전략, 파트2" - NS 및 NSS 모델을 활용한 실전적인 채권 퀀트 전략 제시
22/11/7 "연방기금 선물 내재 금리의 이해" - 금리선물 등에 기반한 미래 시점별 정책금리 전망치를 확인하는 방법 및 이슈 정리
22/8/1 "주식과 채권의 상관성, 파트2 - 주식과 채권 간 상관 관계의 다이내믹스" - 주식/채권 상관관계의 변화를 경기과 물가 요인으로 설명 및 전망
22/5/17 "퀀트 모델링 A to Z - (5) 포트폴리오 밸류에이션 지표 계산법 (가중 조화평균)" - 가중 조화평균을 활용한 포트폴리오 지표 산출법 정리
22/3/14 "KOSPI 마켓 타이밍 모델" – CAPE 지표를 활용한 효과적인 주식 마켓 타이밍 모델 제시
20/10/28 "한국이 MSCI 선진시장에 들어가야 하는 이유" – MSCI 선진시장 승격의 당위성과 승격 시의 변화 등에 대해 총정리
19/11/18 "코스피 단기예측 AI 모델 - 랜덤 포레스트 기법을 활용한 머신러닝 기반 모델" – 랜덤 포레스트 기반의 시장 예측 모델 및 소스 코드 제시
17/3/8 "Volatility Drag - 레버리지 투자 수익과 변동성 손실의 관계" – 변동성 손실 개념과 레버리지 상품의 위험성에 대한 상세 설명
16/1/4 "기계학습(Machine Learning)과 투자전략 - 빅 데이터 기법: Naive Bayes Classifier의 활용" – 머신 러닝에 기반한 퀀트 전략 개발
15/7/20 "퀀트 Investment Clock, 한국편" – 성장과 물가 지표를 조합한 4개 국면 기반의 자산배분 모델 제시
14/4/24 "CAPE in Everywhere - Cyclically Adjusted P/E의 다용도 활용 분석" – 한국 증시의 CAPE 지표에 대한 본격 분석
11/5/17 "MSCI 종목편입 효과 실증분석 - 종목 편출입에 따른 최적 투자전략" – MSCI 이벤트 활용 투자 전략의 시작

Appendix. Samsung Q.Pack 메일링 자료

글로벌 주식시장 동향부터, FICC, 경제지표, 국내외 종목 및 ETF 정보까지 수록함. 이메일을 통해 정기적으로 엑셀 자료 전송

Index	Region /Country	Current Index (p)	Index change (%) (a=b+c)								Fwd EPS change (%) (b)				Fwd PER change (%) (c)				EPS Growth (% y-y)			PER (x)			
			1D	2D	1W	MTD	1M	3M	YTD	1Y	1M	3M	YTD	1Y	1M	3M	YTD	1Y	Fwd	21E	22E	Fwd	21E	22E	4yAvg
MSCI	AC World	742.26	0.0	0.0	1.3	4.6	2.5	2.7	14.9	28.0	0.2	4.1	24.6	32.9	2.3	-1.3	-7.9	-3.7	12.2	49.3	7.7	**18.5**	19.7	18.3	16.6
Index	World (DM)	3,151.16	0.0	0.1	1.3	4.8	2.5	3.4	17.1	30.2	0.6	4.7	26.0	34.1	1.9	-1.2	-7.1	-2.9	12.4	48.2	7.9	**19.5**	20.9	19.3	17.3
	N. America	4,627.53	-0.2	0.1	1.6	5.8	3.5	4.1	20.8	32.4	1.0	4.7	28.8	35.6	2.5	-0.6	-6.2	-2.3	12.3	48.0	8.4	**21.5**	23.1	21.4	18.7
	Europ																								
	EMU																								
	Pacifi																								
	Emerg																								
	EM A																								
	EM L																								
	EM E																								
	USA																								
	Fran																								
	Gerr																								
	UK																								
	Aust																								
	Japa																								
	Hon																								
	Chin																								
	India																								
	Kore																								
	Taiw																								
	Thai																								
	Indo																								
	Braz																								

Code	Name	연간실적																매출액(십억)				매출액y-y(%)		
		매출액(십억)		매출액y-y(%)		영업이익(십억)		영업이익y-y(%)		순이익(십억)		순이익y-y(%)		지배순이익(십억)		지배순이익y-y(%)								
		2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	2021E	2022E	1Q21	2Q21	3Q21	4Q21	1Q21	2Q21	3Q21
A005930	삼성전자	276,340	297,907	16.7	7.8	52,686	56,360	46.4	7.0	39,936	43,047	51.2	7.8	38,912	42,428	49.1	9.0	65,389	63,672	73,000	74,295	18.2	20.2	9.0
A000660	SK하이닉스	42,530	49,328	33.3	16.0	12,035	13,823	140.1	14.9	9,128	10,460	91.8	14.6	9,113	10,425	91.7	14.4	8,494	10,322	11,814	11,902	18.0	19.9	45.3
A035420	NAVER	6,784	8,312	27.9	22.5	1,360	1,756	11.9	29.2	15,824	1,825	1,772.6	-88.5	16,582	1,854	1,554.8	-88.8	1,499	1,664	1,727	1,900	-13.5	-12.6	1,000.3
A207940	삼성바이오로직스	1,491	1,690	28.0	13.4	490	531	67.4	8.4	396	435	64.3	9.8	396	435	64.2	9.8	261	412	411	421	25.9	34.0	49.7
A051910	LG화학	44,085	49,876	46.6	13.1	5,849	4,968	225.3	-15.1	4,439	3,476	550.5	-21.7	4,077	3,173	695.2	-22.2	9,650	11,456	11,285	11,733	35.6	74.9	50.3
A035720	카카오	5,987	7,607	44.0	27.1	786	1,147	72.3	46.0	951	1,097	448.6	15.3	902	1,029	479.4	14.1	1,258	1,352	1,644	1,733	44.9	41.9	49.4
A006400	삼성SDI	13,871	16,666	22.8	20.2	1,170	1,577	74.3	34.8	1,194	1,498	89.2	25.5	1,119	1,410	94.6	26.0	2,963	3,334	3,617	3,957	23.6	30.3	17.2
A005380	현대차	116,993	125,659	12.5	7.4	7,136	8,018	198.0	12.4	6,635	7,393	244.7	11.4	6,020	6,795	322.7	12.9	27,391	30,326	28,132	31,054	8.2	38.7	2.0
A068270	셀트리온	2,047	2,419	10.7	18.2	820	963	15.2	17.4	654	762	26.0	16.4	641	746	25.5	16.4	457	432	486	584	22.6	0.7	-11.4
A000270	기아	70,551	75,507	19.2	7.0	5,320	5,854	157.5	10.0	4,733	5,190	218.1	9.7	4,730	5,181	217.9	9.5	16,582	18,339	17,125	18,478	13.8	61.3	4.9
A323410	카카오뱅크					296	521	141.8	75.9	244	399	114.3	64.0	244	399	114.3	64.0	225	254					
A005490	POSCO	72,925	74,829	26.2	2.6	8,884	8,158	269.7	-8.2	6,603	5,877	269.3	-11.0	6,094	5,425	280.4	-11.0	16,069	18,292	20,610	19,280	10.5	33.3	44.5
A012330	현대모비스	41,047	45,462	12.1	10.8	2,228	2,706	21.8	21.4	2,616	3,102	71.3	18.6	2,599	3,081	70.0	18.5	9,816	10,285	9,900	11,004	16.5	36.5	-0.9
A259960	크래프톤	2,128	3,306	27.4	55.4	907	1,531	17.2	68.9	722	1,197	29.8	65.9	722	1,197	29.8	65.9	461	459	460	757	-9.3	21.2	31.4
A028260	삼성물산	32,740	32,926	8.4	0.6	1,407	1,448	64.2	2.9	1,923	1,796	65.7	-6.6	1,730	1,563	67.1	-9.6	7,840	8,548	8,034	8,377	12.6	18.3	2.3
A066570	LG전자	71,956	74,202	13.7	3.1	4,288	4,993	34.2	16.4	2,109	3,675	2.2	74.3	1,821	3,615	-7.5	98.5	18,809	16,117	18,785	17,986	27.7	25.6	11.0
A096770	SK이노베이션	45,612	52,196	33.5	14.4	1,938	2,132	175.5	10.0	901	1,430	142.0	58.8	878	1,401	140.4	59.5	9,240	11,120	12,270	13,120	-17.2	54.5	45.7
A017670	SK텔레콤	19,538	20,303	4.9	3.9	1,526	1,678	13.1	10.0	2,655	2,905	77.0	9.4	2,634	2,834	75.1	7.6	4,780	4,818	4,914	5,036	7.4	4.7	3.9
A105560	KB금융					6,172	6,351	33.7	2.9	4,497	4,688	28.4	4.2	4,463	4,646	29.2	4.1	17,584	12,242	16,473		-8.8	14.2	56.4
A051900	LG생활건강	8,401	9,115	7.1	8.5	1,313	1,463	7.5	11.5	892	1,005	9.7	12.7	874	984	9.6	12.6	2,037	2,021	2,139	2,203	7.4	13.4	3.3
A055550	신한지주					5,993	6,188	21.6	3.3	4,334	4,484	23.9	3.5	4,292	4,414	25.7	2.8	6,895	6,813			-15.3	11.6	
A302440	SK바이오사이언스	976	3,572	332.4	266.2	438	1,578	1,060.7	260.5	355	1,227	980.7	245.2	355	1,227	980.4	245.3	113	145	173	499	397.4	266.8	78.8
A034730	SK	94,268	100,607	15.2	6.7	4,642	5,335	2,922.4	14.9	3,106	3,609	2,964.8	16.2	1,377	1,642	627.2	19.3	22,250	23,406	23,724	25,547	-6.2	24.7	18.9
A091990	셀트리온헬스케어	1,838	2,179	12.9	18.5	331	438	-8.6	32.5	257	336	7.0	30.8	257	336	7.0	30.8	356	433	488	569	-0.2	3.1	5.2
A361610	SK아이이테크놀로	676	1,076	44.1	59.2	189	333	50.7	76.3	148	255	68.1	72.0	149	246	69.3	65.1	139	155	178	227			
A015760	한국전력	59,706	62,797	1.9	5.2	-1,631	-951	-139.9	41.7	-2,165	-1,710	-203.5	21.0	-2,230	-1,755	-212.0	21.3	15,075	13,519	16,039	15,091	-0.1	3.4	2.1
A011200	HMM	12,561	12,447	95.9	-0.9	6,315	5,669	543.9	-10.2	3,837	4,905	2,995.0	27.8	3,866	4,994	3,020.8	29.2	2,428	2,907	3,592	3,489	84.9	111.4	109.0
A003550	LG	8,303	8,612	25.2	3.7	2,797	2,957	64.3	5.7	3,054	2,606	98.3	-14.7	2,958	2,507	101.8	-15.2	2,180	1,924	2,011	2,214	37.5	28.7	2.8
A032830	삼성생명	20,270	20,905	-41.3	3.1	1,182	689	-34.0	-41.7	1,755	1,399	28.1	-20.3	1,653	1,359	30.6	-17.8	10,008	8,195	4,701	4,759	-3.5	7.3	-39.5
A086790	하나금융지주					4,393	4,649	14.5	5.8	3,259	3,416	21.4	4.8	3,200	3,357	21.4	4.9	14,011	6,908	13,195		-32.0	10.8	102.5

Compliance notice

- 본 조사자료는 당사의 저작물로서 모든 저작권은 당사에게 있습니다.
- 본 조사자료는 당사의 동의 없이 어떠한 경우에도 어떠한 형태로든 복제, 배포, 전송, 변형, 대여할 수 없습니다.
- 본 조사자료에 수록된 내용은 당사 리서치센터가 신뢰할 만한 자료 및 정보로부터 얻어진 것이나, 당사는 그 정확성이나 완전성을 보장할 수 없습니다. 따라서 어떠한 경우에도 본 자료는 고객의 주식투자의 결과에 대한 법적 책임소재에 대한 증빙자료로 사용될 수 없습니다.
- 본 자료에는 외부의 부당한 압력이나 간섭 없이 애널리스트의 의견이 정확히 반영되었습니다.

삼성증권

삼성증권주식회사
서울특별시 서초구 서초대로74길 11(삼성전자빌딩)
Tel: 02 2020 8000 / www.samsungpop.com

삼성증권 Family Center: 1588 2323

고객 불편사항 접수: 080 911 0900

Member of
Dow Jones
Sustainability Indices
Powered by the S&P Global CSA