

추천시스템 KNOW 직업 추천 알고리즘+TabNet이해

대회팀 김희숙 이수연 이지호

목차

#01 contents

Tabular Data 학습과 딥러닝

#02 contents

TabNet 소개

#03 contents

KNOW 기반 직업 추천 대회



Tabular Data 학습과 딥러닝

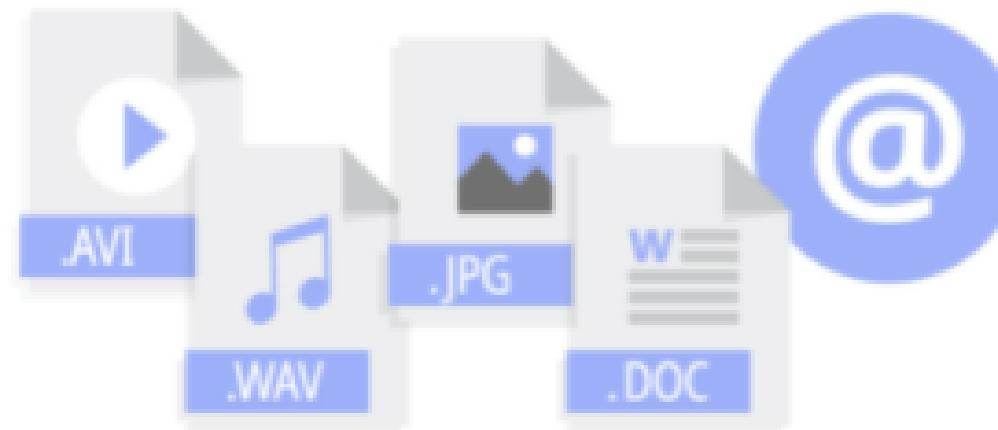


#01 Tabular Data 학습과 딥러닝

▪ Tabular Data

- Feature는 Column, Sample은 Row 방향으로 결합된 테이블 형태의 Dataset을 의미
- RDMS(관계형 데이터베이스)는 하나 이상의 테이블로 이루어져 있으며, 각 테이블은 key와 value의 관계를 통해 표현하고 싶은 대상을 추상화 함
- RDBMS의 보편화로 인해 현존하는 IT system을 통해 빈번히 경험할 수 있는 데이터 구조

Unstructured Dataset



Tabular Dataset

ID	Name	AGE	SEX
01	KIM	32	M
02	LEE	26	F
03	PARK	72	F
04	CHOI	15	M

#01 Tabular Data 학습과 딥러닝

▪ 딥러닝의 필요성

Tabular Data, 딥러닝 연구와 현장

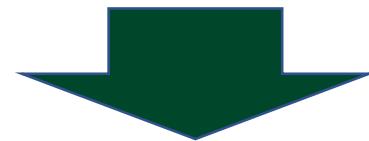
- 이미지, 음성, 언어와 같은 비정형 데이터 대비 정형 데이터에서 인상적이지 못한 성능
- 학습 비용 대비 경제성 부족 & 해석의 어려움
- 비정형 데이터를 활용한 딥러닝 연구 vs feature engineering에 집중한 현업 프로젝트

<u>Research ML</u>	<u>Production ML</u>
Offline, 불변	데이터 특징
SOTA (Accuracy, RMSE 등)	평가 방법
독창성과 높은 성능	중요 요소
모델 변경 시	학습 특성
기반 기술 확보	목표
	Online, Streaming
	모델 성능, Inference 속도, 해석
	경제성 및 안정적인 운영
	데이터 변경
	서비스 제공

#01 Tabular Data 학습과 딥러닝

▪ 딥러닝의 필요성

- 딥러닝의 점진적 학습 특성과 사전 학습 가능성은 새로운 기회가 될 수 있음
- Incremental learning: Streaming data에서 지속적인 학습이 가능함
- Pretraining: 사전학습으로 학습 시간 단축 및 적은 데이터 사용으로 성능 향상 가능
- Capacity: 복잡하고 다양한 패턴을 수용, 데이터가 누적됨에 따라 지속적인 성능 향상 기대



TabNet

TabNet 소개



#02 TabNet 소개

#1 Main Concept

#2 Encoding

#3 feature transformer & attentive transformer
* what is attention?

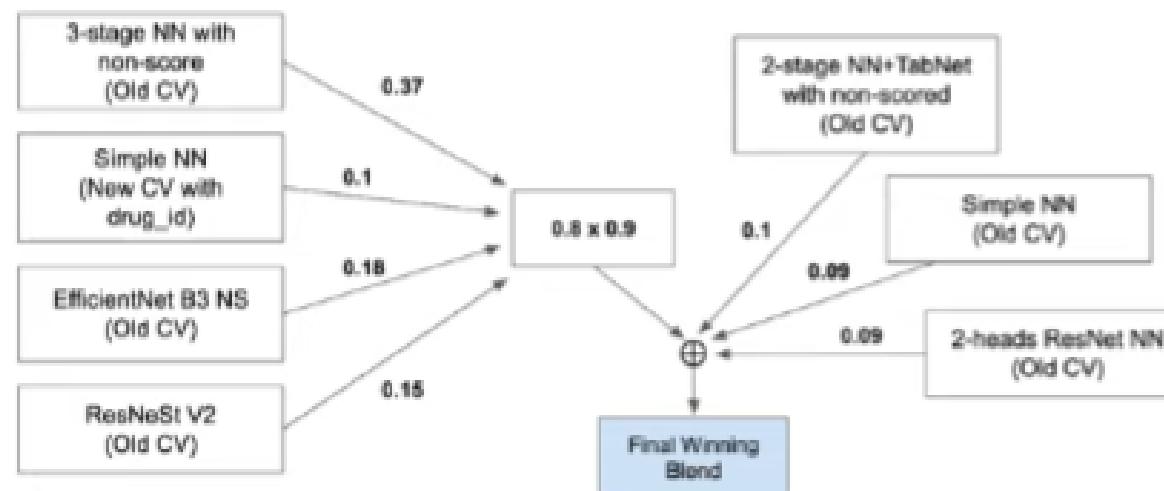
#4 Decoding(semi-supervised learning)

#02 TabNet 소개: Main Concept

■ TabNet

- 정형 데이터를 위한 딥러닝 모델
- TabNet on AI Platform: high-performance, explainable tabular learning
- 캐글 MoA 대회의 베이스라인 모델, 우승 알고리즘에도 활용
- Google ML Platform인 Vertex AI GCP 내에서 즉시 사용 가능

Winning Weighted-Average Blend for MoA



GCP User Guide for Tabular

Use a built-in algorithm training job to train a custom model with pre-built algorithms.
[Learn more](#)

① Training algorithm — ② Training data — ③ Algorithm arguments — ④ Job settings

Before you get started, make sure your algorithm follows the format required for your algorithm. [Learn more about how to prepare your data](#).

Select an algorithm:

[NEXT](#) [CANCEL](#)

Max Steps: HyperTune

Learning Rate: HyperTune

A scalar used to determine gradient step in gradient descent training. [Learn more](#)

#02 TabNet 소개: Main Concept

■ TabNet

```
import torch
import torch.nn as nn
from pytorch_tabnet.tab_model import TabNetClassifier
```

```
clf = TabNetClassifier(cat_idxs=cat_idxs,
                      cat_dims=cat_dims,
                      cat_emb_dim=10,
                      optimizer_fn=torch.optim.Adam,
                      optimizer_params=dict(lr=1e-2),
                      scheduler_params={"step_size":50,
                                        "gamma":0.9},
                      scheduler_fn=torch.optim.lr_scheduler.StepLR,
                      mask_type='sparsemax' # "sparsemax", "entmax"
                     )
```

```
clf.fit(
    X_train=X_train, y_train=y_train,
    eval_set=[(X_train, y_train), (X_valid, y_valid)],
    eval_name=['train', 'valid'],
    eval_metric=['auc'],
    max_epochs=max_epochs, patience=20,
    batch_size=1024, virtual_batch_size=128,
    num_workers=0,
    weights=1,
    drop_last=False,
)
```

- ✓ pytorch-tabnet library 제공
- ✓ feature preprocessing 없이, 원시 표 데이터를 입력하여 기존의 경사 하강 기반 최적화를 사용하여 학습

#02 TabNet 소개: TabNet 알고리즘

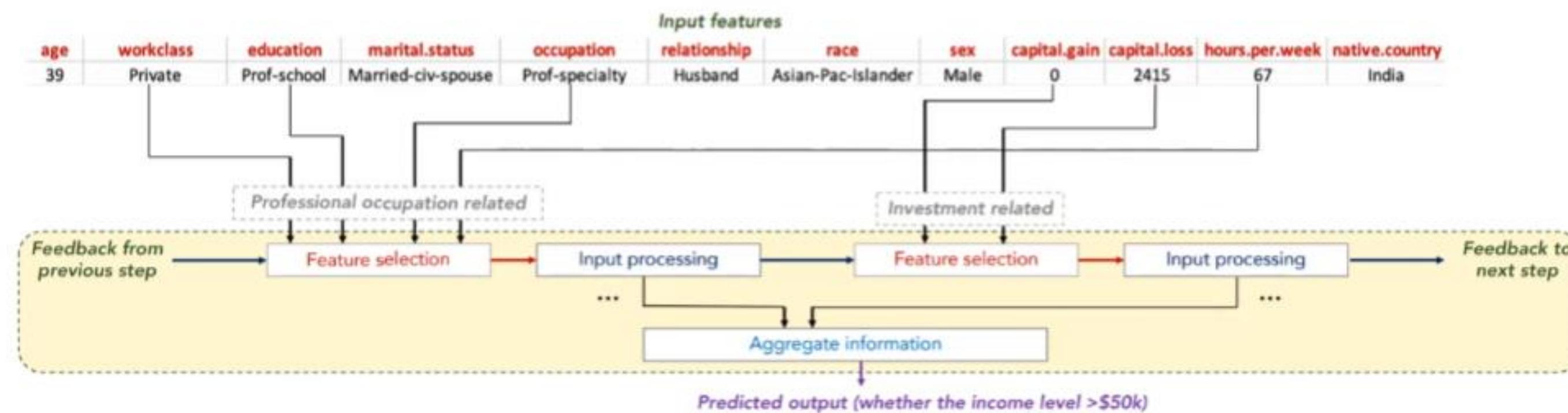
▪ Sparse Feature Selection(=Decision Blocks)

- Tree 기반 모델의 변수 선택 특징을 네트워크 구조에 반영한 딥러닝 모델
- 가공하지 않은 raw data에서 gradient를 기반한 최적화를 사용함으로써 *end-to-end 학습을 실현
- Sequential attention mechanism을 사용하여 모델의 성능과 해석 용이성을 향상

*end-to-end learning?

: 입력에서 출력까지 '파이프라인 네트워크' 없이 한 번에 처리한다는 의미로 일반적인 머신러닝과 비교했을 때, 전처리/ 변수 선택/ 모델링의 과정이 한 번에 이뤄진다는 것

Sparse Feature Selection 동작 원리

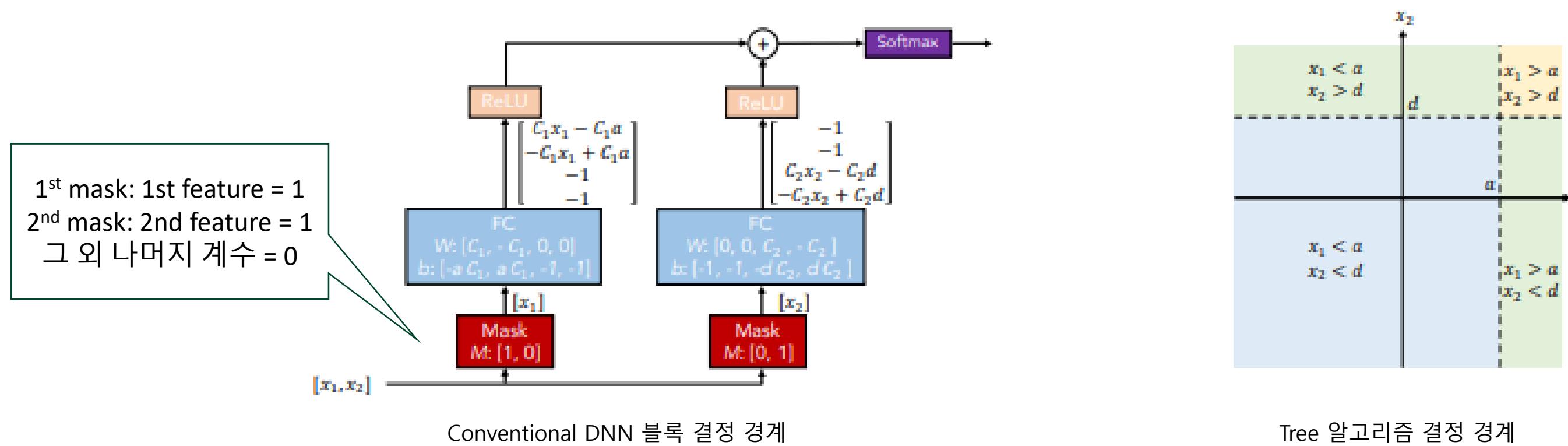


#02 TabNet 소개: TabNet 알고리즘

■ Feature selection

Conventional DNN 블록으로 Tree와 유사한 형태의 결정 경계 생성

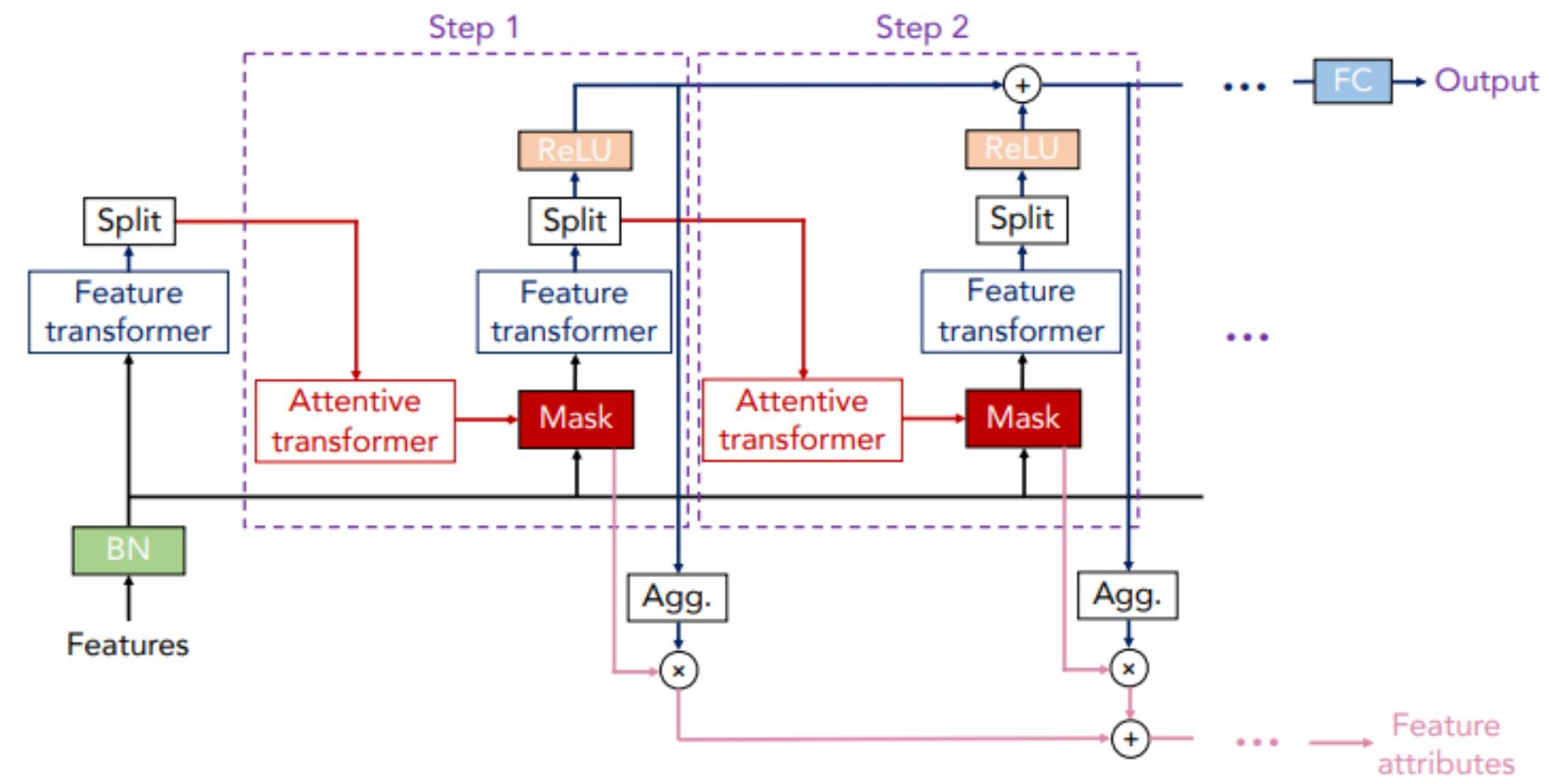
- sparse **instance-wise** feature selection learned from data
- Constructs a **sequential** multi-step architecture, where each step contributes to a portion of the decision based on the selected features
- Improves the learning capacity via nonlinear processing of the selected features



#02 TabNet 소개: TabNet 알고리즘

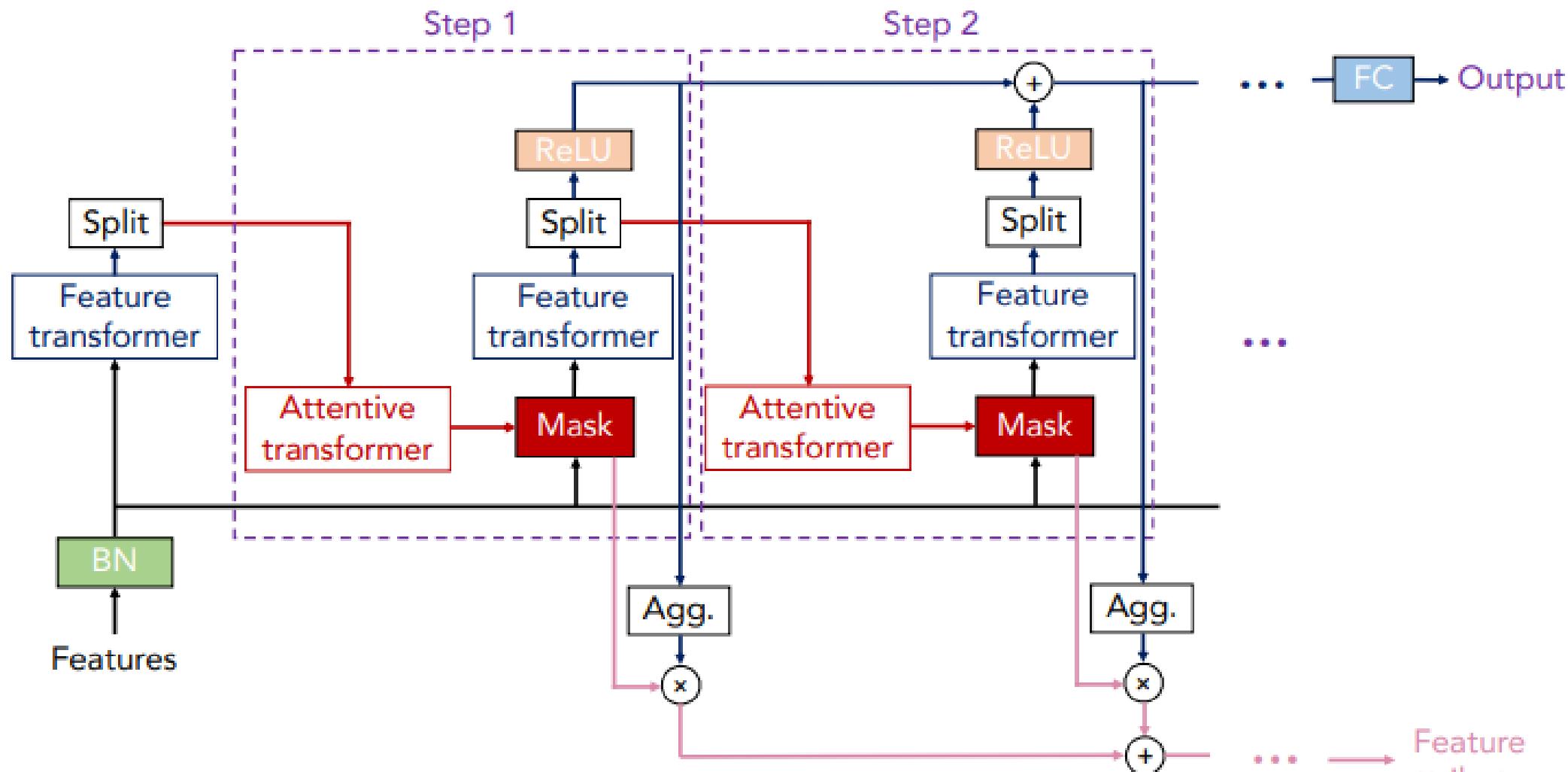
▪ Encoding

- 1)feature transformer, 2) attentive transformer, 3) feature masking으로 구성
- 이전 단계 학습 결과가 다음 단계 mask 학습에 영향을 주는 연결 구조
 - 첫 의사 결정 단계에서 부족한 부분을 다음 의사 결정 단계에서 보완하는 방식으로 트리 기반 부스팅 모델과 유사
- feature transformer와 attentive transformer 블록을 통과하여 최적 mask 학습

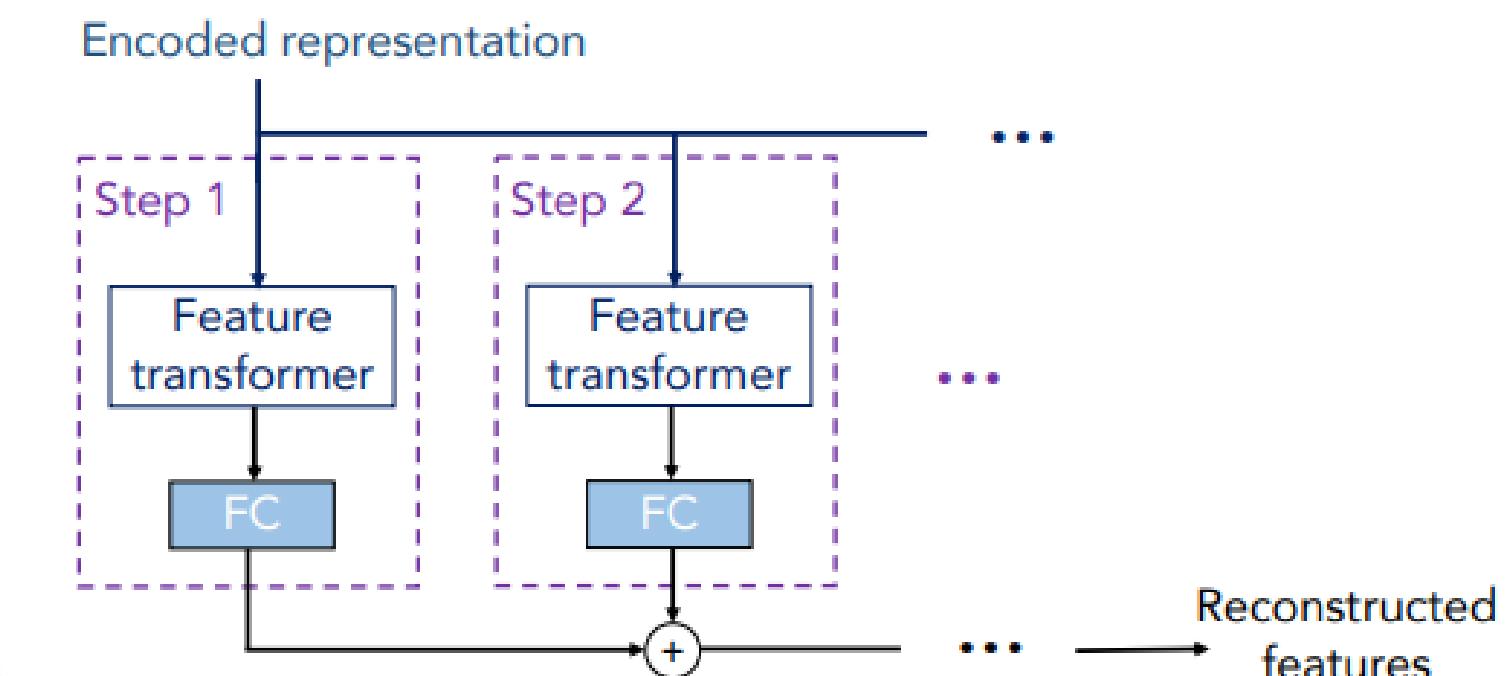


(a) TabNet encoder architecture

#02 TabNet 소개: * Attention과 Transformer



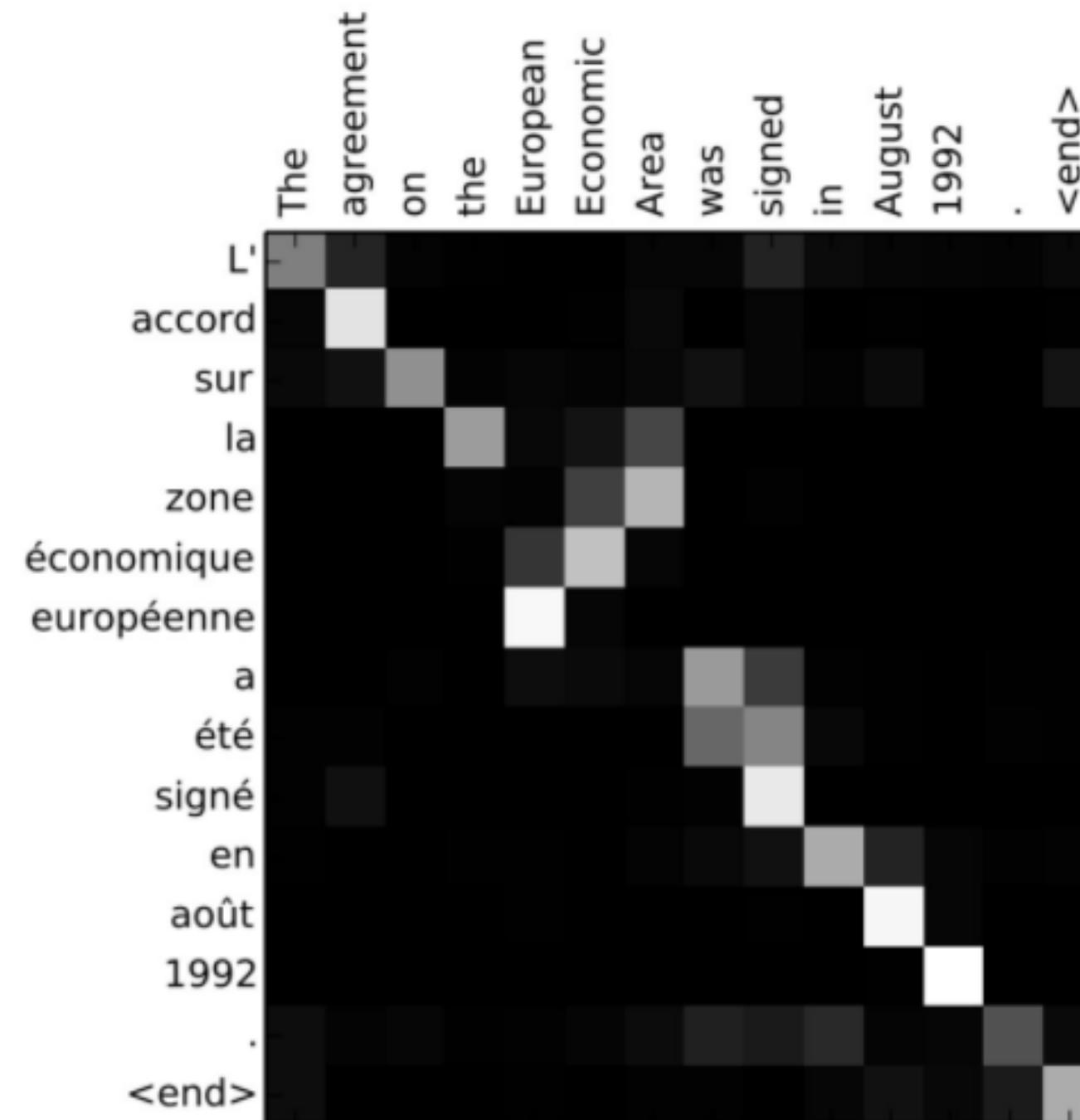
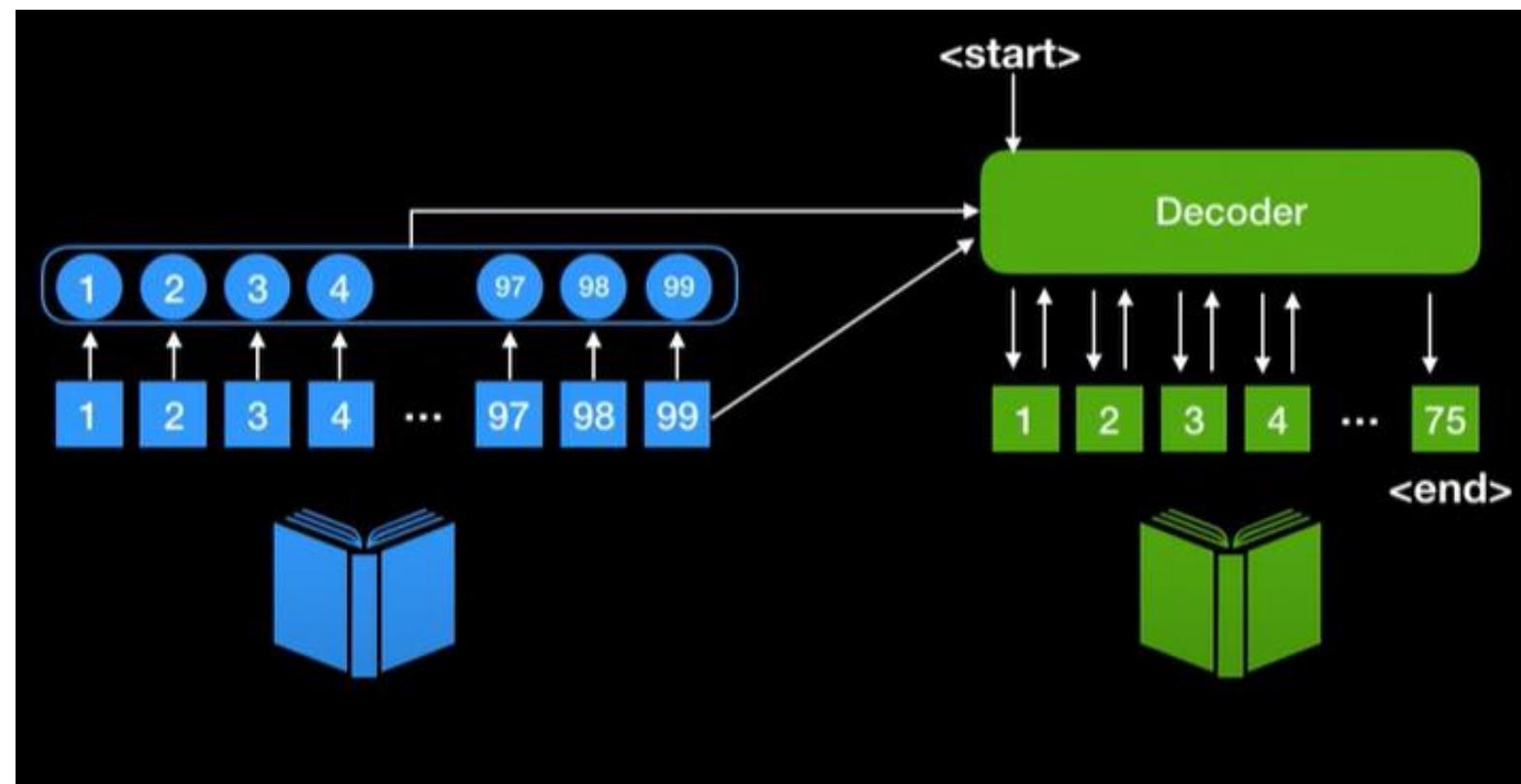
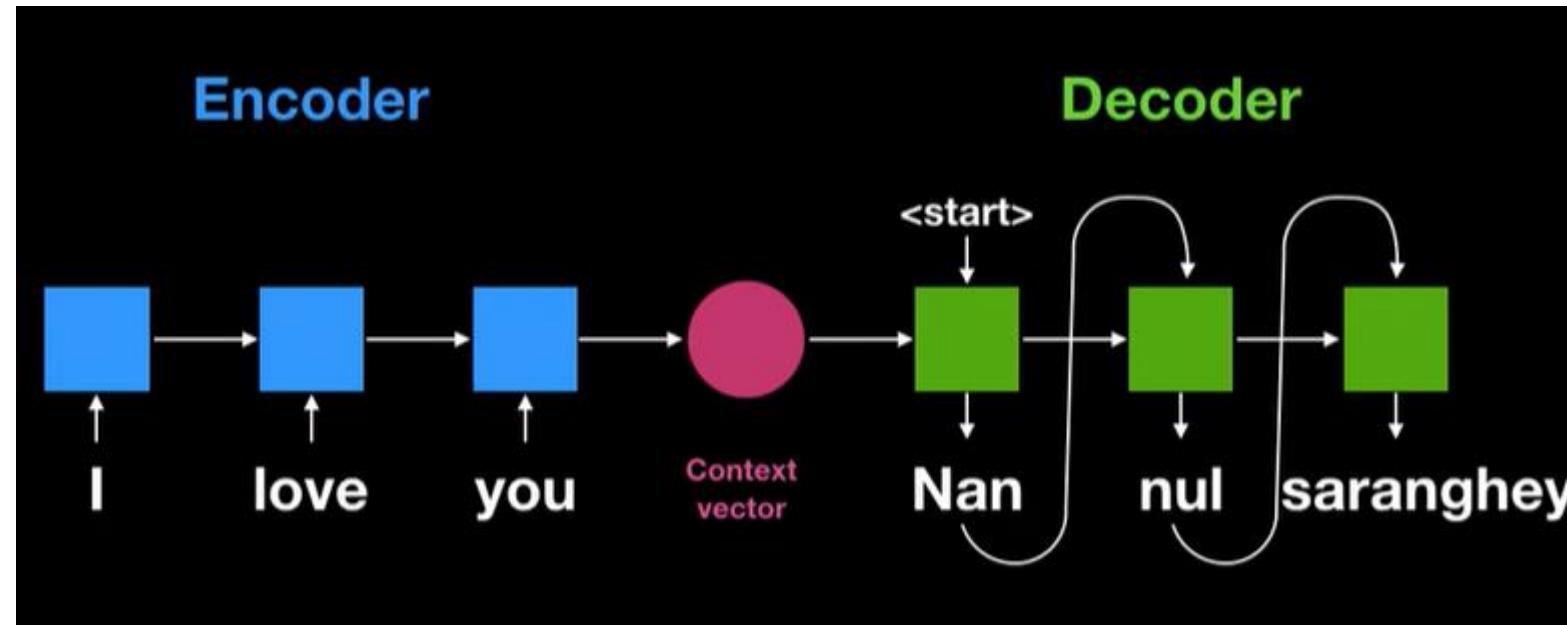
(a) TabNet encoder architecture



(b) TabNet decoder architecture

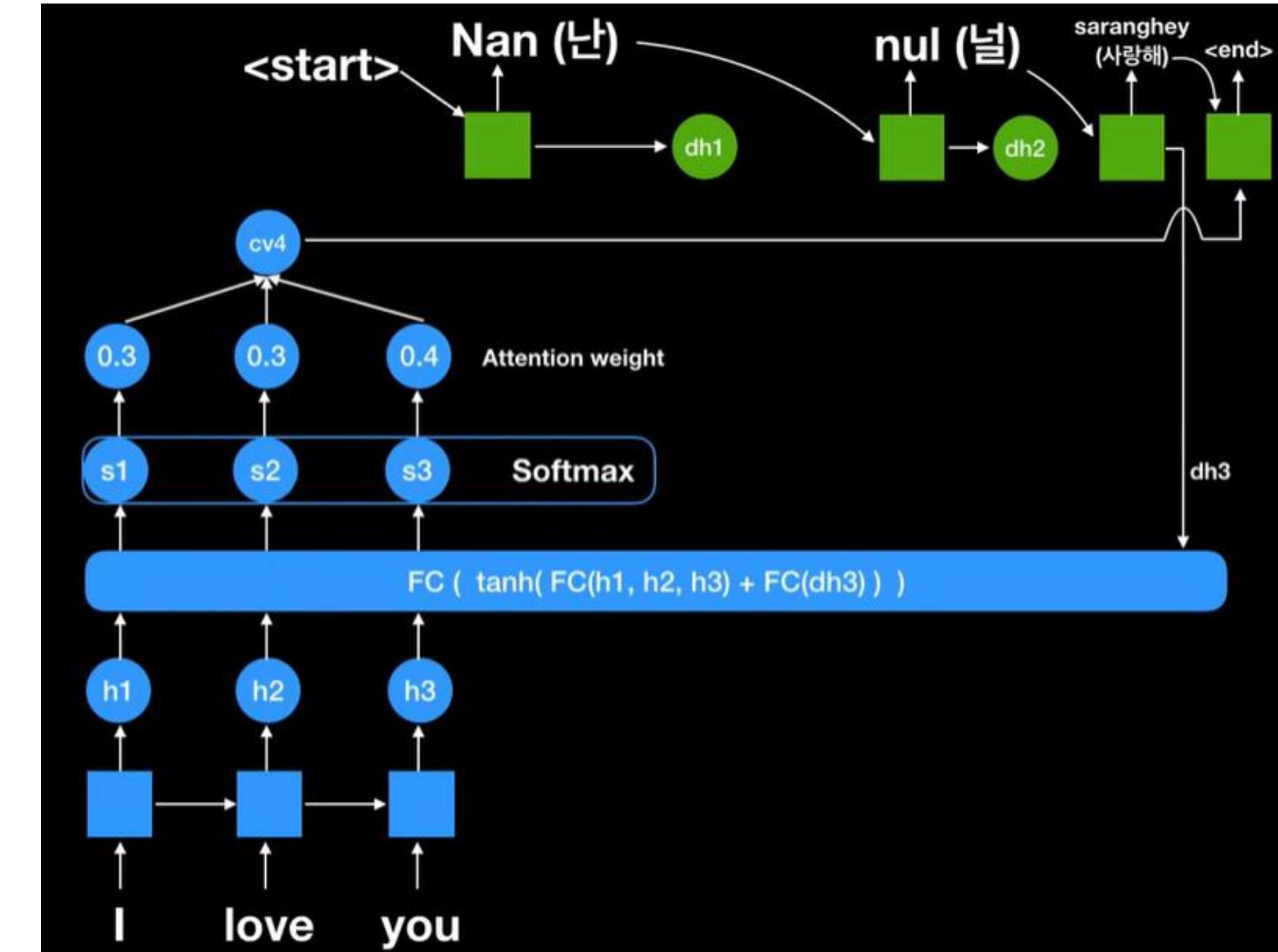
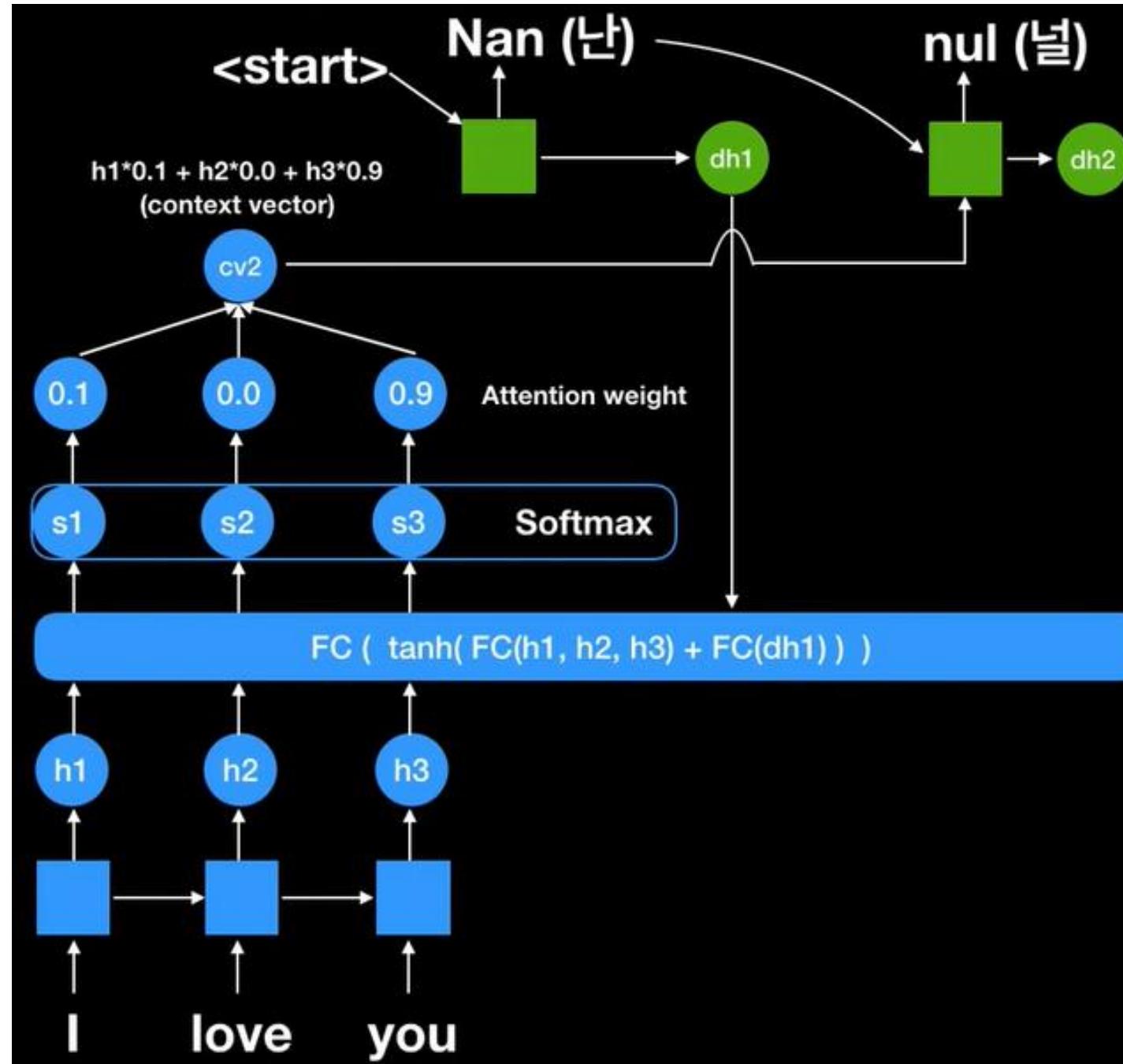
#02 TabNet 소개: * Attention과 Transformer

▪ What is Attention



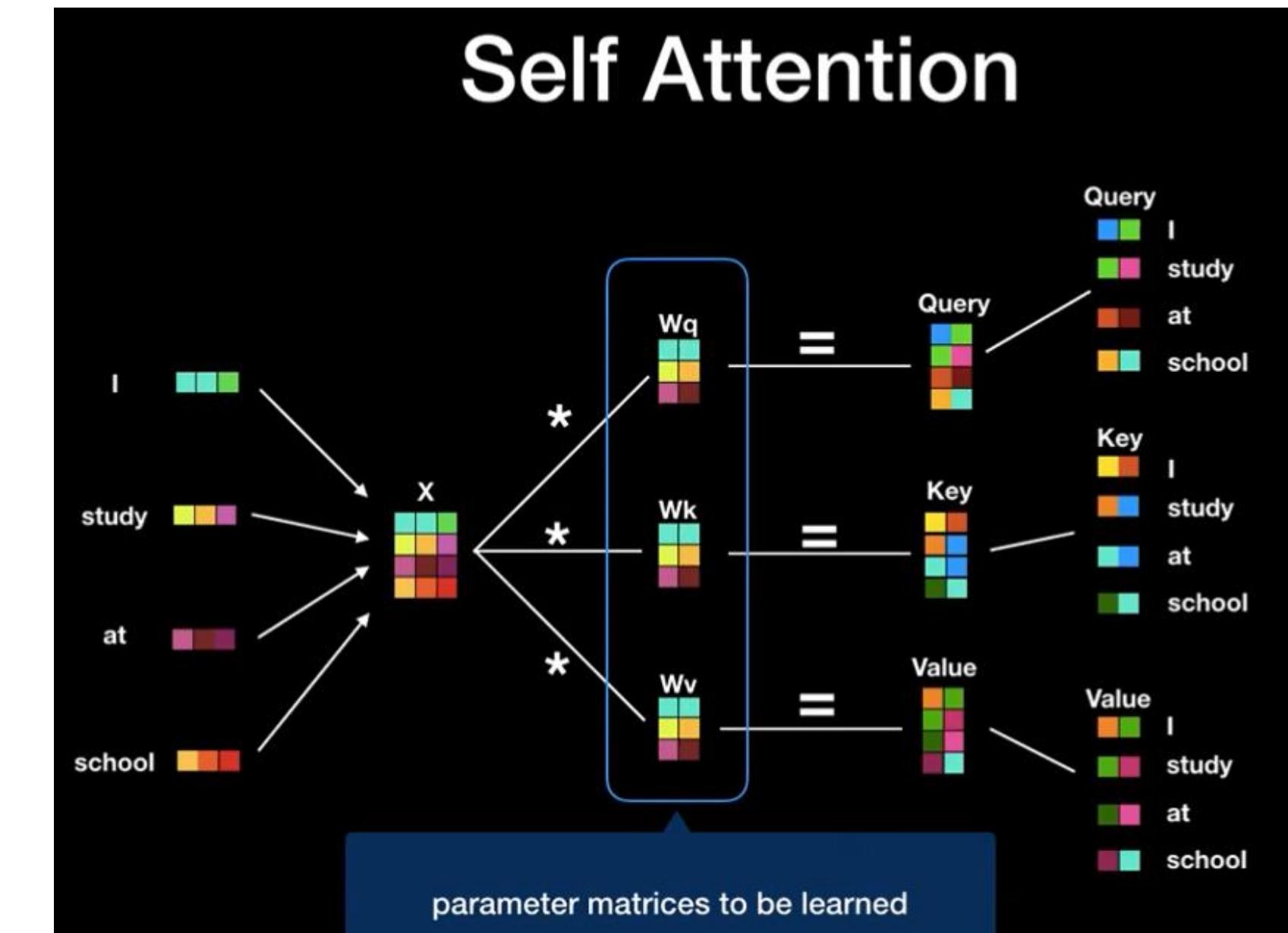
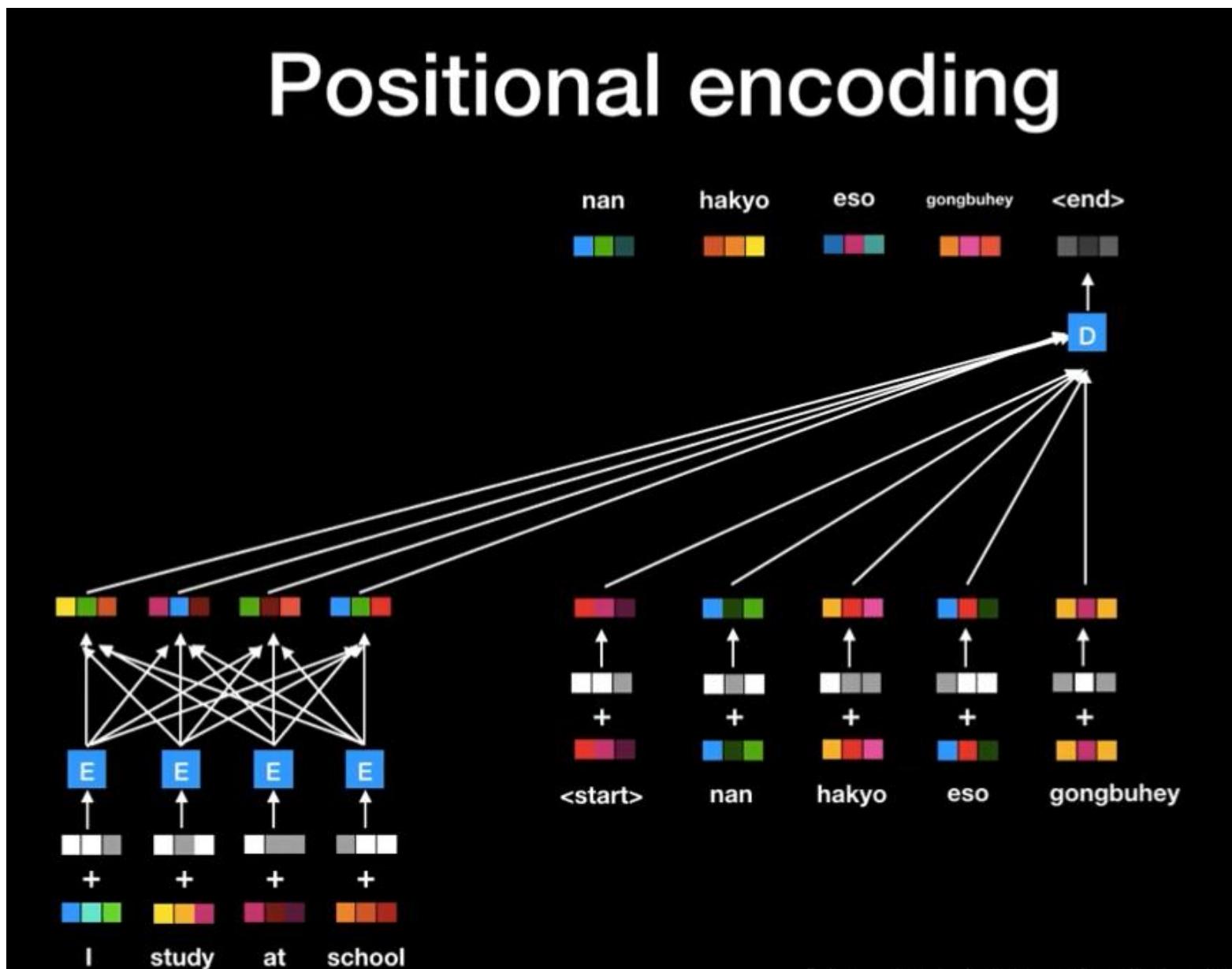
#02 TabNet 소개: * Attention과 Transformer

▪ What is Attention



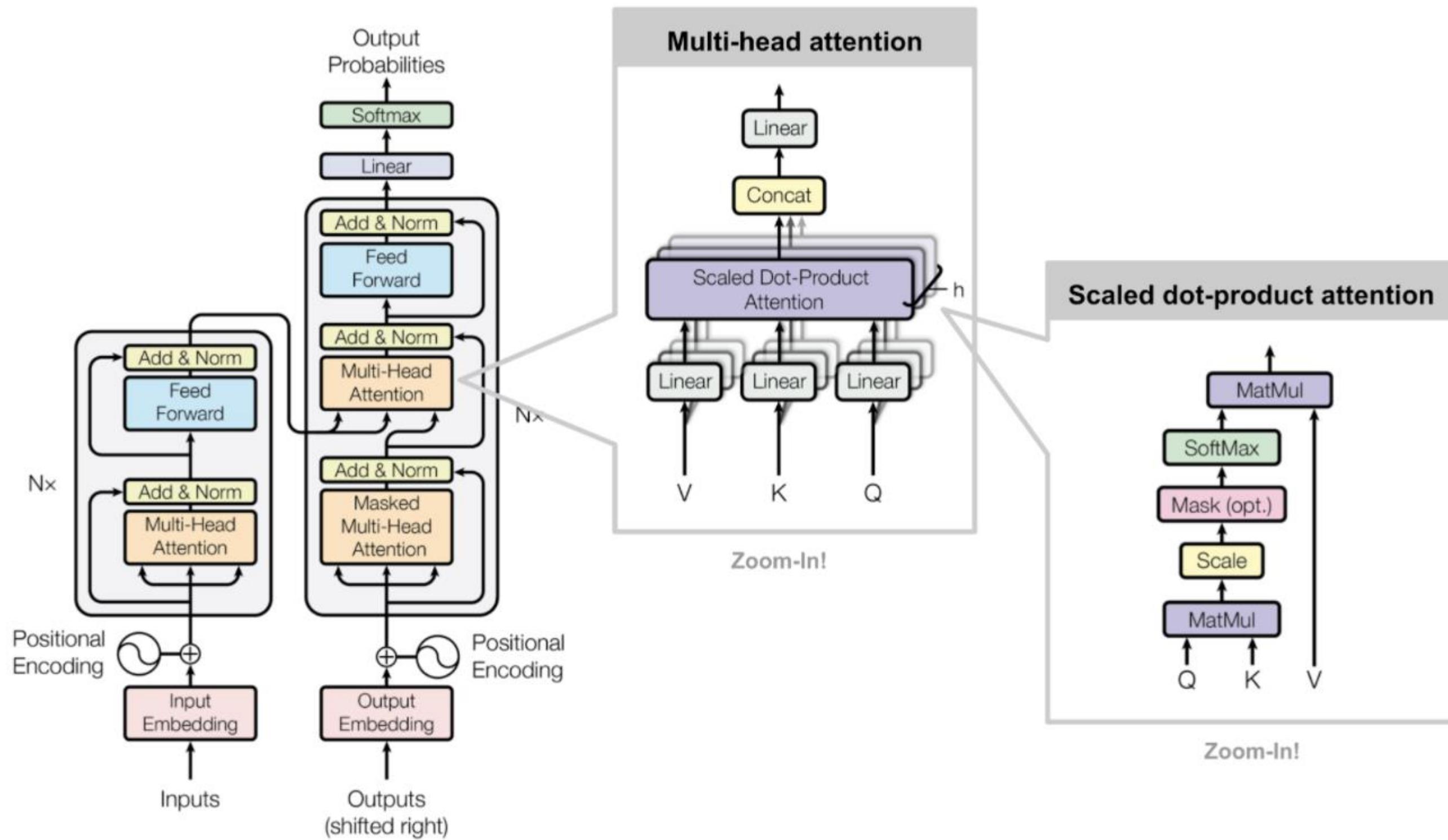
#02 TabNet 소개: * Attention과 Transformer

▪ What is Transformer



#02 TabNet 소개: * Attention과 Transformer

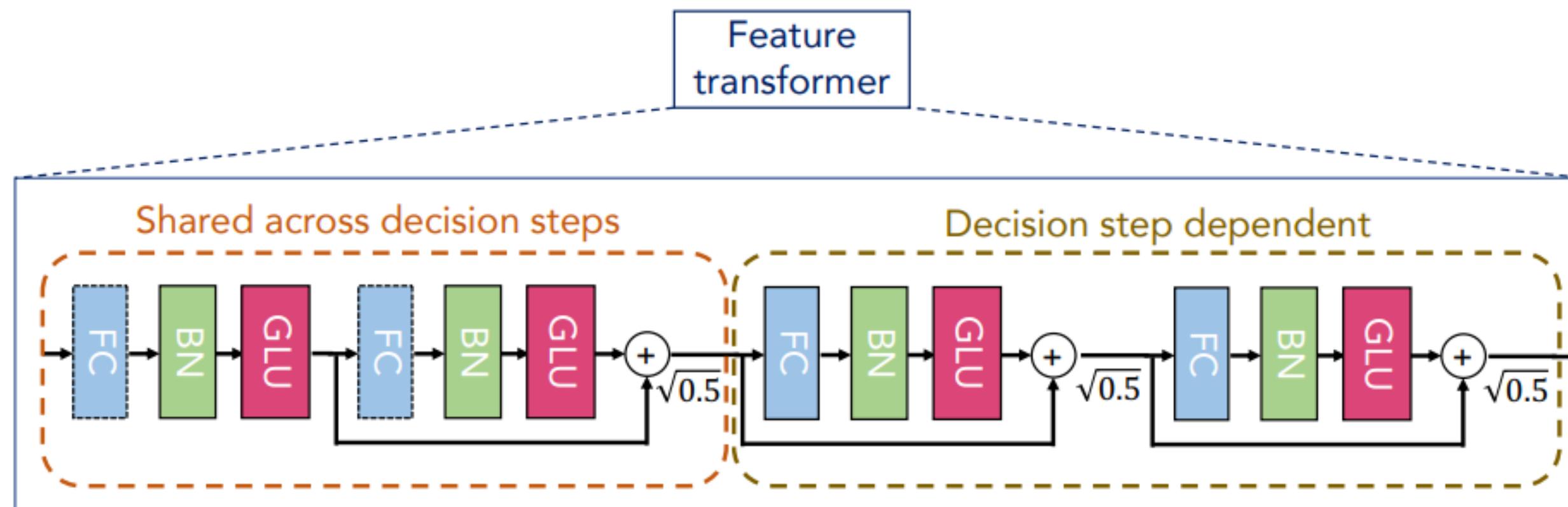
▪ What is Transformer



#02 TabNet 소개: TabNet 알고리즘

■ Feature Transformer

- 마스킹 되어 전달된 피처를 임베딩하는 기능

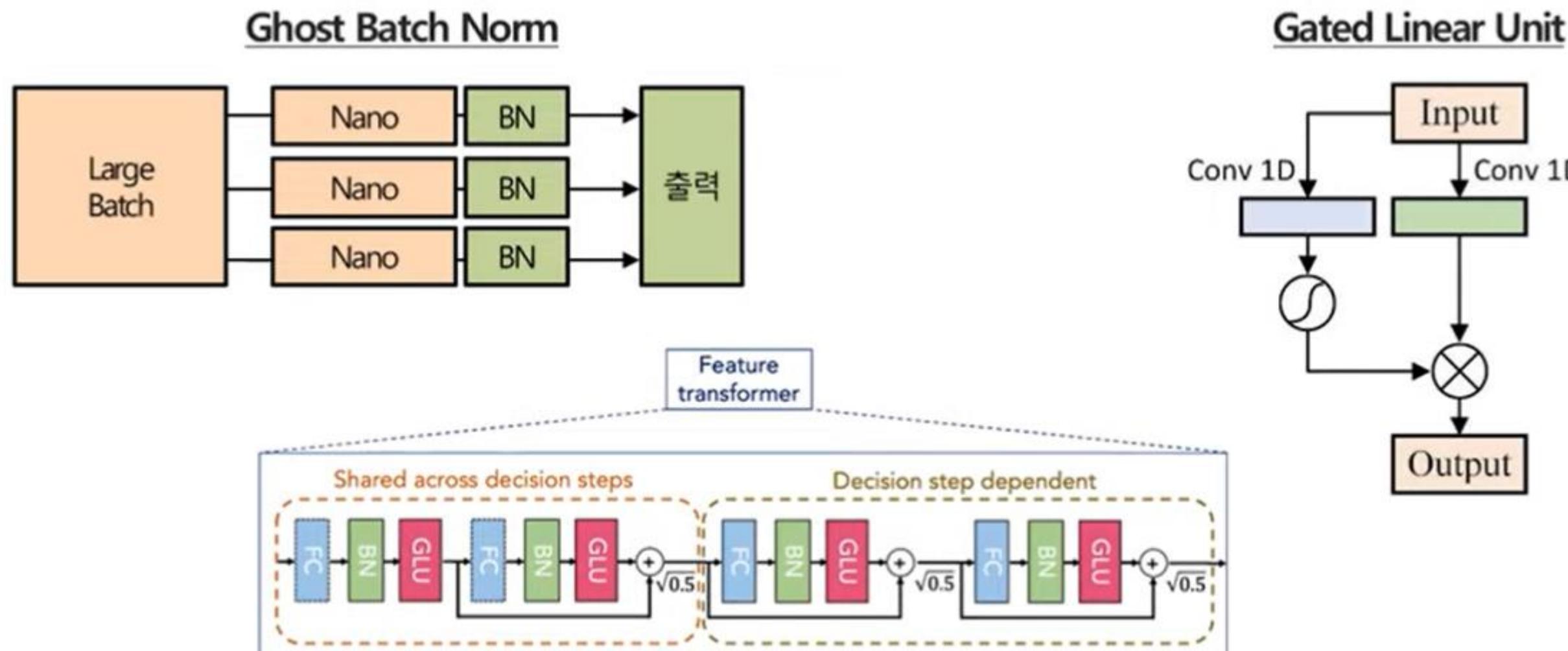


- Fully Connected Layer – Batch Normalization Layer – Gated Linear Unit이 연속되어 있는 블록구조
- Shared across decision steps: 모든 스텝에서 파라미터를 공유하는 영역, global한 성향 학습
- Decision step dependent: 각 스텝에서만 전용으로 사용되는 블록, local 특성 학습

#02 TabNet 소개: TabNet 알고리즘

■ Feature Transformer

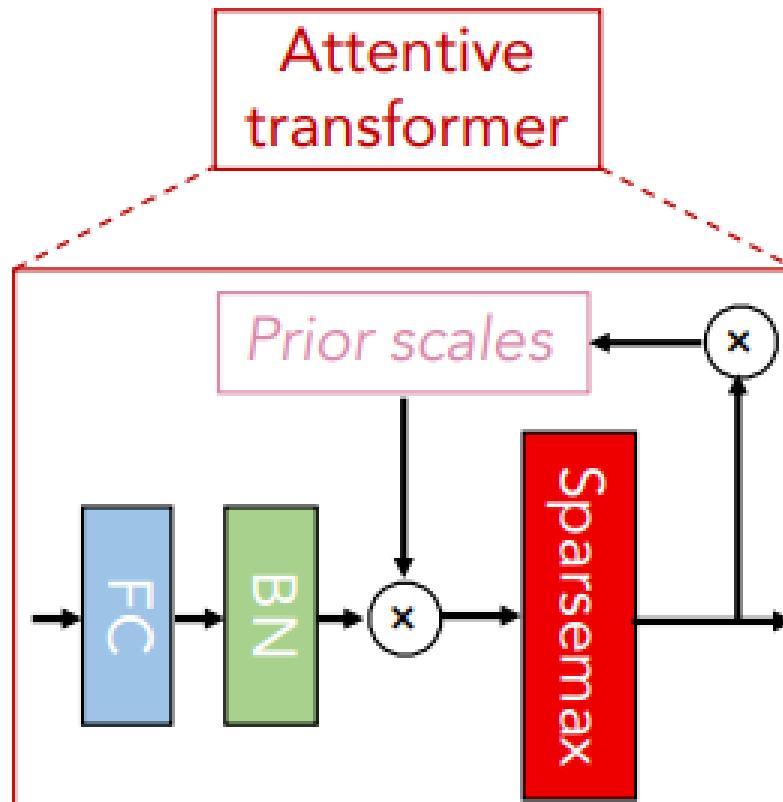
- Ghost Batch Norm(BN로 표기함)
: batch를 분할한 nano batch 사용으로 잡음 추가 → 지역 최적화 예방 → large batch size로 학습속도 향상
- Gated Linear Unit(GLU): 이전 layer에서 전달되는 정보 크기를 제어하는 역할



#02 TabNet 소개: TabNet 알고리즘

▪ Attentive Transformer

- 변수 선택 기능을 하는 블록으로 Prior Scale + Sparsemax로 구현되어 있다.



- A single layer mapping is modulated with a **prior scale** information which aggregates how much each feature has been used before the current decision step.

Prior Scales

$$P[i] = \prod_{j=1}^i (\gamma - M[j])$$

- **sparsemax** is used for normalization of the coefficients, resulting in sparse selection of the salient features.

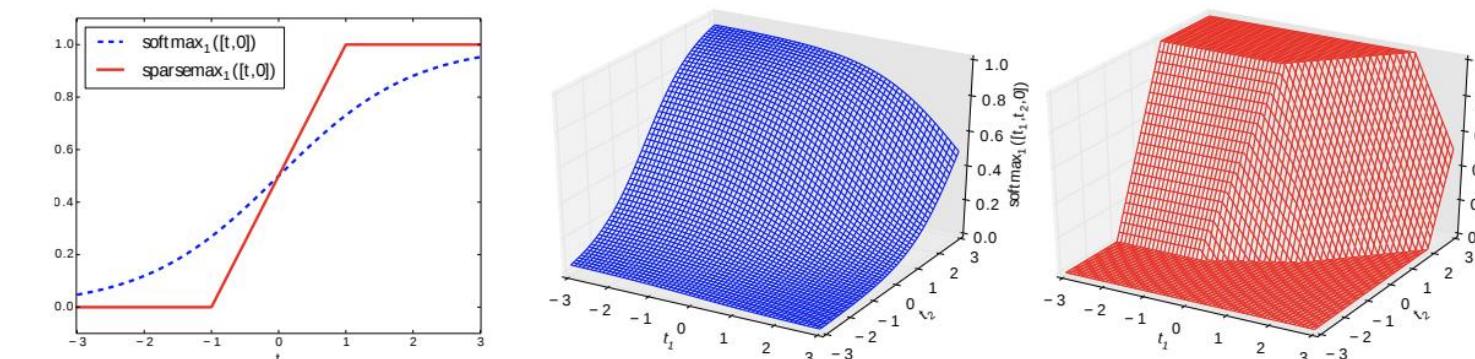


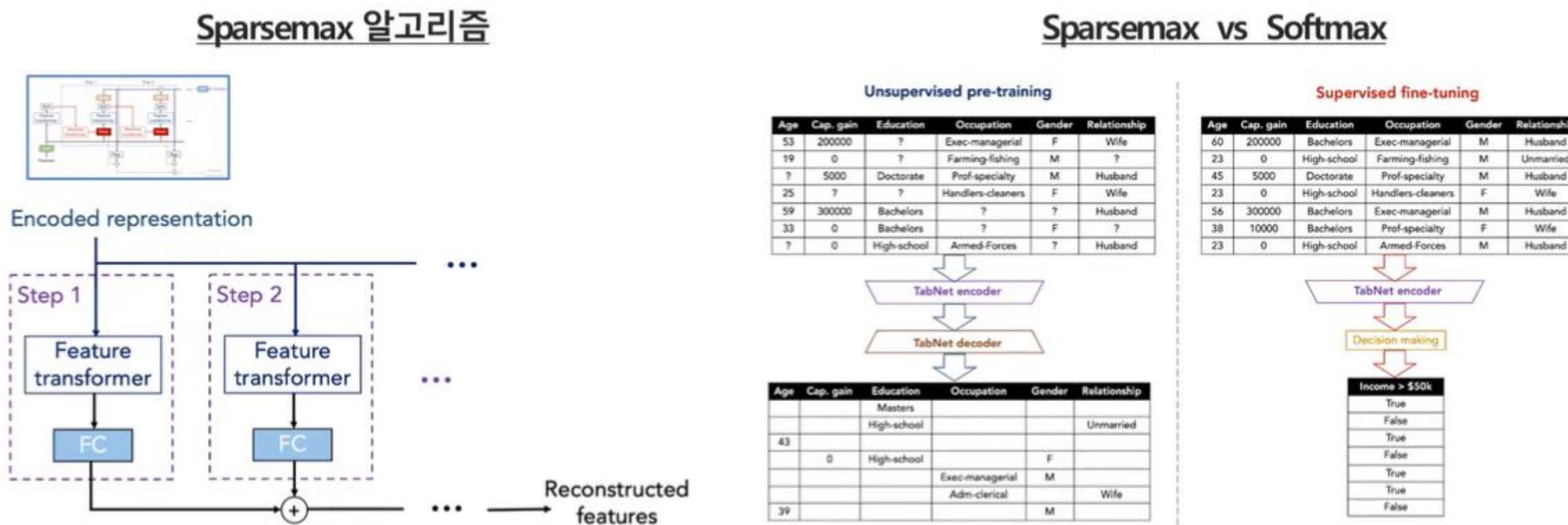
Figure 1. Comparison of softmax and sparsemax in 2D (left) and 3D (two rightmost plots).

#02 TabNet 소개: TabNet 알고리즘

■ Decoding

❖ Semi-supervised Learning

- 앞서 소개한 encoder에 decoder를 연결하면 autoencoder와 같은 자기 학습 구조를 생성할 수 있음
- 특정한 영역이 masking된 인코딩 데이터를 원본대로 복원할 수 있도록 학습
- 사전 학습을 통한 예측 성능 향상, 학습 시간 단축 및 결측치에 대한 보간 효과

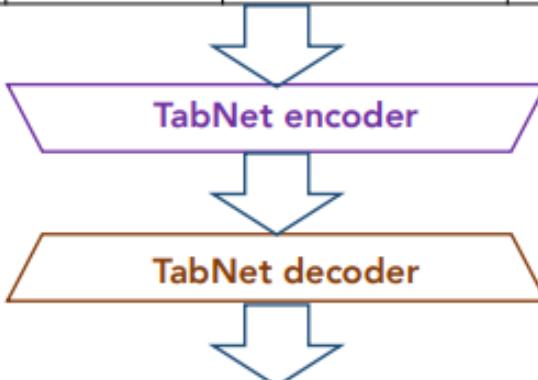


#02 TabNet 소개: TabNet 알고리즘

■ Decoding

Unsupervised pre-training

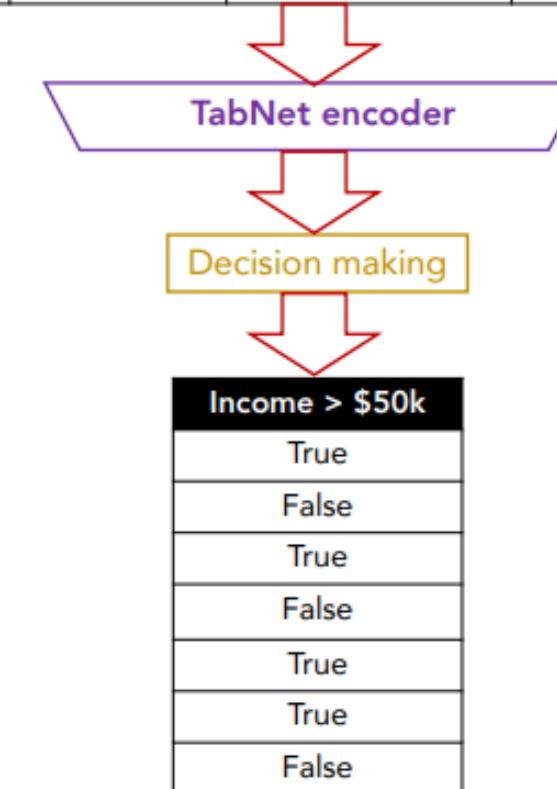
Age	Cap. gain	Education	Occupation	Gender	Relationship
53	200000	?	Exec-managerial	F	Wife
19	0	?	Farming-fishing	M	?
?	5000	Doctorate	Prof-specialty	M	Husband
25	?	?	Handlers-cleaners	F	Wife
59	300000	Bachelors	?	?	Husband
33	0	Bachelors	?	F	?
?	0	High-school	Armed-Forces	?	Husband



Age	Cap. gain	Education	Occupation	Gender	Relationship
		Masters			
		High-school			Unmarried
43					
	0	High-school		F	
			Exec-managerial	M	
			Adm-clerical		Wife
39				M	

Supervised fine-tuning

Age	Cap. gain	Education	Occupation	Gender	Relationship
60	200000	Bachelors	Exec-managerial	M	Husband
23	0	High-school	Farming-fishing	M	Unmarried
45	5000	Doctorate	Prof-specialty	M	Husband
23	0	High-school	Handlers-cleaners	F	Wife
56	300000	Bachelors	Exec-managerial	M	Husband
38	10000	Bachelors	Prof-specialty	F	Wife
23	0	High-school	Armed-Forces	M	Husband



#02 TabNet 소개: TabNet 의의

- Attentive transformer mask를 통해 활용한 변수의 중요도를 시각화 할 수 있다. 해석이 용이한 딥러닝 모델이라는 점에서 의미가 있음
- Pretraining, iterative training이라는 개념이 tabular 데이터 학습에도 적용될 수 있는 여지를 제공함
- 다수의 tabular dataset에서 우수한 성능을 보임

KNOW 기반 직업 추천 대회



#03-1 Data

KNOW 설문 데이터셋

1. 데이터 구성

- a. 일반업무활동 (2017년 데이터셋)
- b. 업무환경 및 흥미 (2018)
- c. 지식 및 성격 (2019)
- d. 업무수행능력 (2020)

2. 데이터 특징

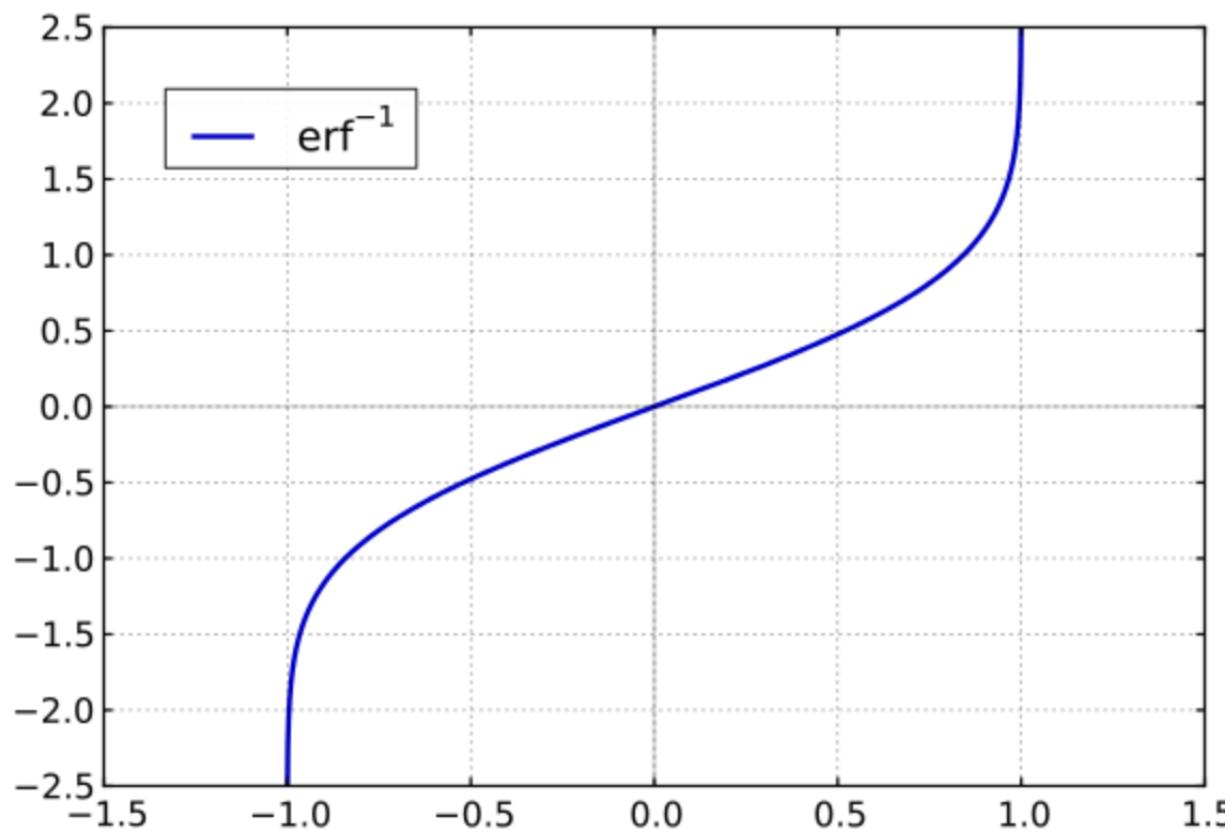
- a. 정수형, 범주형 데이터 및 텍스트 데이터로 구성
- b. Label의 경우, knwocode 변수명으로 제공, 순서에 관계없이 범주값으로 구성

순서	새변수명	새변수설명	특징
1	sq1	<성격> 성취/노력	
2	sq2	<성격> 안내	
3	sq3	<성격> 책임성과 진취성	
4	sq4	<성격> 리더십	
5	sq5	<성격> 협조	
6	sq6	<성격> 타인에대한 배려	
7	sq7	<성격> 사회성	
8	sq8	<성격> 자기통제	
9	sq9	<성격> 스트레스강내성	
10	sq10	<성격> 적응성/융통성	
11	sq11	<성격> 신뢰성	
12	sq12	<성격> 광범위	
13	sq13	<성격> 정직성	
14	sq14	<성격> 독립성	
15	sq15	<성격> 혁신	
16	sq16	<성격> 분석적 사고	
17	kq1_1	<지식> 경영 및 행정_중요도	
18	kq1_2	<지식> 경영 및 행정_수준	
19	kq2_1	<지식> 사무_중요도	
20	kq2_2	<지식> 사무_수준	
21	kq3_1	<지식> 경제와 회계_중요도	
22	kq3_2	<지식> 경제와 회계_수준	
23	kq4_1	<지식> 영업과 마케팅_중요도	
24	kq4_2	<지식> 영업과 마케팅_수준	
25	kq5_1	<지식> 고객서비스_중요도	
26	kq5_2	<지식> 고객서비스_수준	
27	kq6_1	<지식> 인사_중요도	
28	kq6_2	<지식> 인사_수준	
29	kq7_1	<지식> 상품제조 및 공정_중요도	
30	kq7_2	<지식> 상품제조 및 공정_수준	

#03-2 Preprocessing

Gauss Rank Scaler & Standard Scaler

- effective algorithm for converting numeric variable distributions to normals.
- it is based on rank transformation. The first step is to assign a spacing between -1 and 1 to the sorted features,
- then apply the inverse of error function erfinv to make it look like a Gaussian.



#03-2 Preprocessing

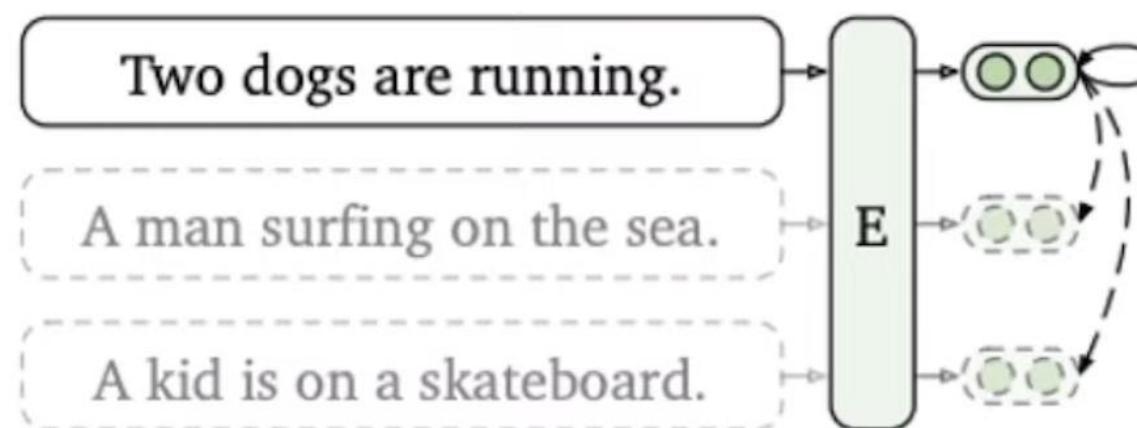
설문조사 데이터에 포함된 자연어 항목의 처리를 위해, 한국어 임베딩 모델 사용

- SimCSE

Pretrained embedding + Contrastive Learning
= SoTA embedding

- Unsupervised SimCSE

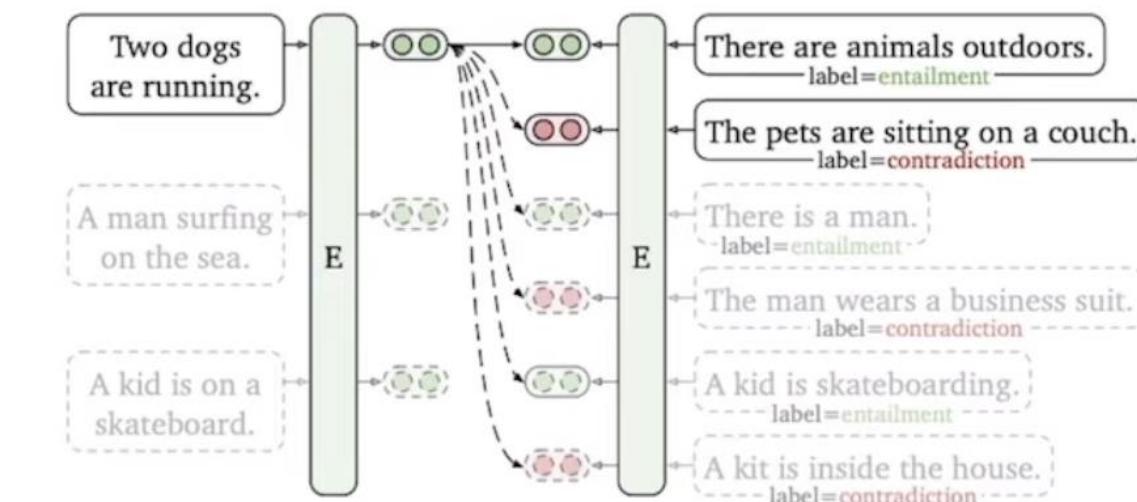
- ✓ Input : 동일한 문장을 Positive pair로 활용



- ✓ Dropout을 활용하여 minimal augmentation 적용
- ✓ Input은 동일하지만 dropout으로 상이한 hidden Representation을 생성하여 positive pair 구성

- Supervised SimCSE

- ✓ Input : 서로 다른 문장을 Positive pair로 활용



- ✓ NLI(Natural Language Inference) 데이터를 활용하여 'entailment' label을 positive pair로 구성
- ✓ 'contradiction' label은 Hard negative pair로 사용

#03-3 Pytorch Lightening

- PyTorch에 대한 High-level 인터페이스를 제공하는 오픈소스 Python 라이브러리

PyTorch

```
# model
class Net(nn.Module):
    def __init__(self):
        self.layer_1 = torch.nn.Linear(28 * 28, 128)
        self.layer_2 = torch.nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.layer_1(x)
        x = F.relu(x)
        x = self.layer_2(x)
        return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                     transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
```

PyTorch Lightning

```
# model
class Net(LightningModule):
    def __init__(self):
        self.layer_1 = torch.nn.Linear(28 * 28, 128)
        self.layer_2 = torch.nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.layer_1(x)
        x = F.relu(x)
        x = self.layer_2(x)
        return x

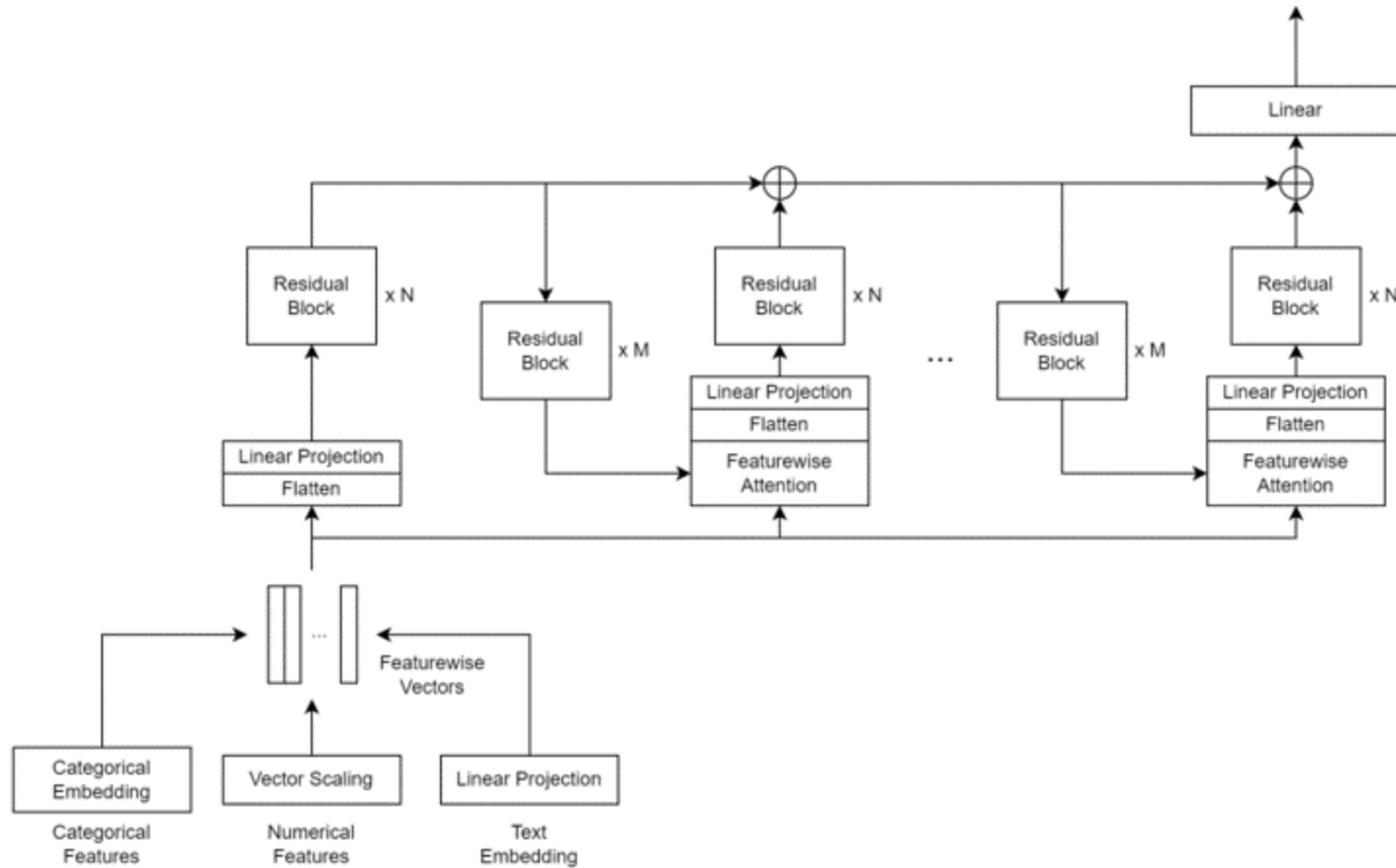
    def train_dataloader(self):
        mnist_train = MNIST(os.getcwd(), train=True, download=True,
                            transform=transforms.ToTensor())
        return DataLoader(mnist_train, batch_size=64)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
        scheduler = StepLR(optimizer, step_size=1)
        return optimizer, scheduler

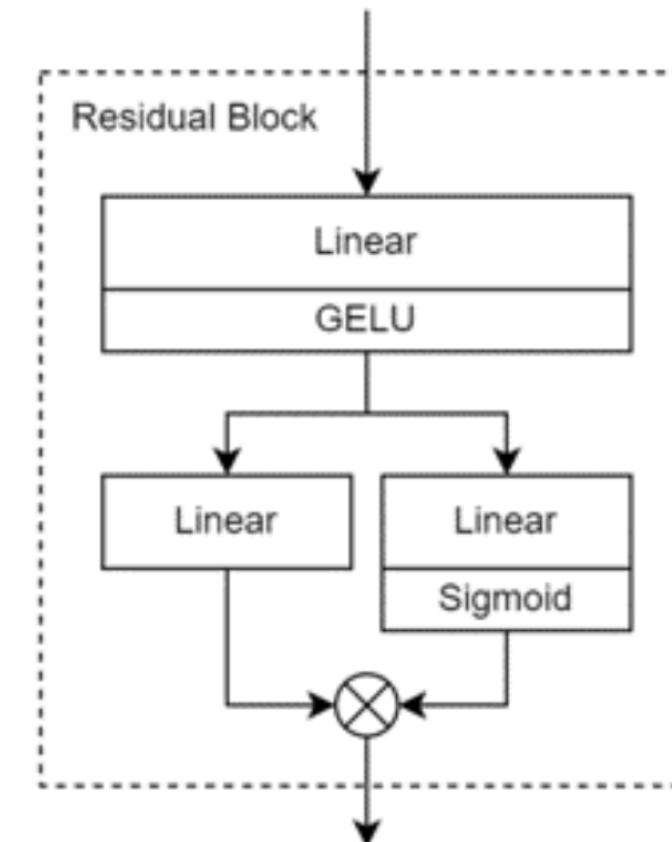
    def training_step(self, batch, batch_idx):
        data, target = batch
        output = self.forward(data)
        loss = F.nll_loss(output, target)
        return {'loss': loss}
```

#03-4 Model Architecture

- Model Architecture



- Residual Block

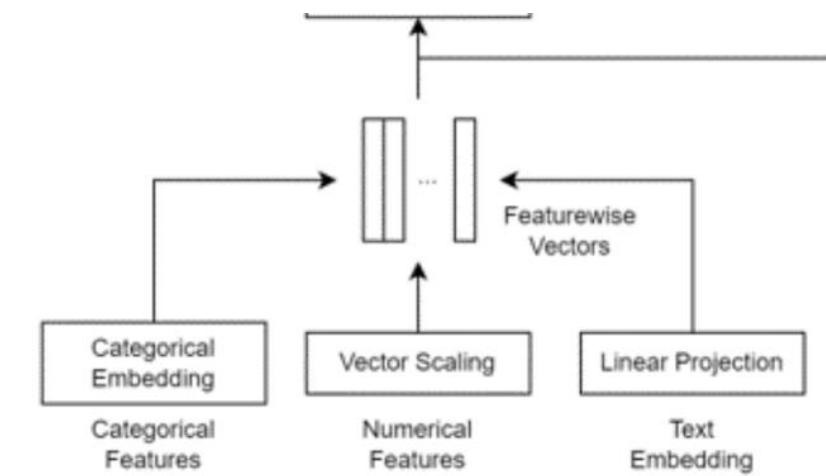


#03-4 Architecture

- SiDEmbeddings

```
class SiDEmbeddings(nn.Module):  
    def __init__(self, config: SiDConfig):  
        super().__init__()  
        self.categorical_embeddings = nn.Embedding(  
            config.num_total_categories, config.embedding_size  
        )  
        self.numerical_direction = nn.Parameter(  
            torch.rand(config.num_numerical_features, config.embedding_size)  
        )  
        self.numerical_anchor = nn.Parameter(  
            torch.rand(config.num_numerical_features, config.embedding_size)  
        )  
  
        self.text_projections = nn.ModuleList(  
            nn.Linear(config.text_input_size, config.embedding_size, bias=False)  
            for _ in range(config.num_text_features)  
        )  
  
        self.register_buffer(  
            "categorical_embedding_offsets",  
            torch.tensor([0] + config.num_categories_list[:-1]).cumsum(1),  
        )
```

```
def forward(  
    self,  
    categorical_inputs: torch.Tensor,  
    numerical_inputs: torch.Tensor,  
    text_inputs: torch.Tensor,  
) -> torch.Tensor:  
  
    categorical_inputs = categorical_inputs + self.categorical_embedding_offsets  
    categorical_embeddings = self.categorical_embeddings(categorical_inputs)  
  
    numerical_embeddings = numerical_inputs[:, :, None] * self.numerical_direction  
    numerical_embeddings = numerical_embeddings + self.numerical_anchor  
  
    stacked_weight = torch.stack(  
        [layer.weight.transpose(0, 1) for layer in self.text_projections],  
    )  
    text_embeddings = torch.einsum("btm,tmn->btn", text_inputs, stacked_weight)  
  
    return torch.cat(  
        (categorical_embeddings, numerical_embeddings, text_embeddings), dim=1  
    )
```



#03-4 Architecture

- SiDLayer

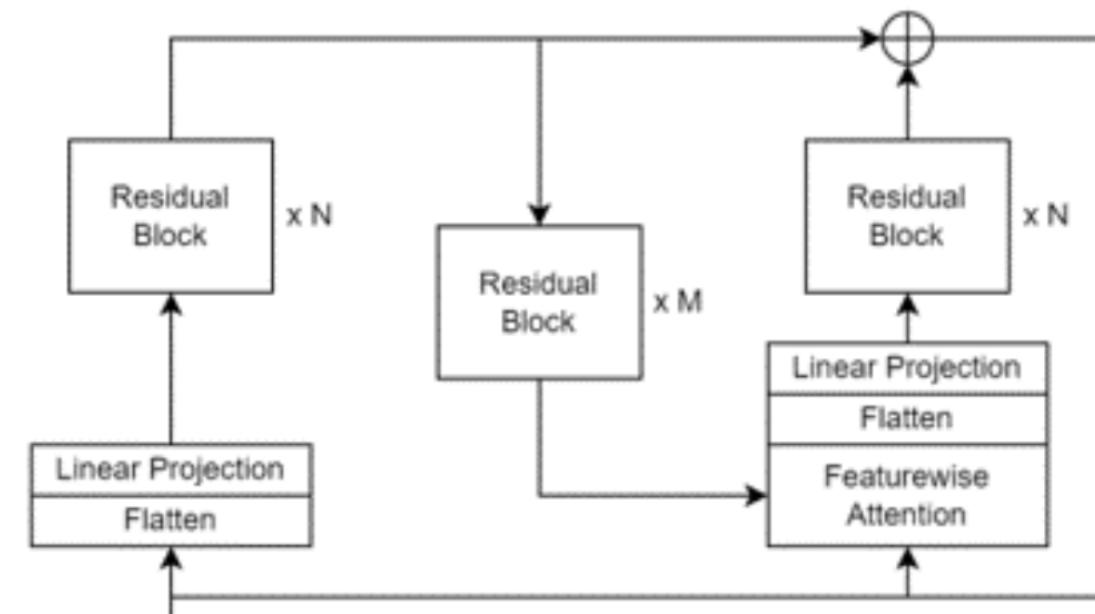
```
class SiDLayer(nn.Module):
    def __init__(self, config: SiDConfig, use_attention: bool = True):
        super().__init__()
        if use_attention:
            self.attention = nn.Sequential(
                *[SiDResidualBlock(config) for _ in range(config.num_attention_blocks)],
                nn.Linear(config.hidden_size, config.num_total_features),
                nn.Sigmoid(),
            )
        self.dropout = nn.Dropout(config.attention_dropout_prob)

        self.projection = nn.Linear(config.total_embedding_size, config.hidden_size)
        self.transform = nn.Sequential(
            *[SiDResidualBlock(config) for _ in range(config.num_transform_blocks)])
        self.droppath = StochasticDepth(config.drop_path_prob)

    def forward(
        self,
        input_embeddings: torch.Tensor,
        hidden_states: Optional[torch.Tensor] = None,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # Calculate the attention probabilities and multiply to the embeddings.
        if hasattr(self, "attention") and hidden_states is not None:
            attention_probs = self.attention(hidden_states)
            attention_probs = self.dropout(attention_probs)
            input_embeddings = input_embeddings * attention_probs[:, :, None]

        output = self.projection(input_embeddings.flatten(1))
        output = self.transform(output)

        # If `hidden_states` is not None then use residual connection.
        if hidden_states is not None:
            return hidden_states + self.droppath(output)
        return output
```



#03-4 Architecture

- SiDModel

```
class SiDModel(nn.Module):
    def __init__(self, config: SiDConfig):
        super().__init__()
        self.config = config
        self.embeddings = SiDEmbeddings(config)
        self.layers = nn.ModuleList([
            SiDLayer(config, use_attention=i > 0)
            for i in range(config.num_hidden_layers)
        ])
        self.normalization = nn.LayerNorm(config.hidden_size)
        self.init_weights()

    def init_weights(self, module: Optional[nn.Module] = None):
        if module is None:
            self.apply(self.init_weights)
        elif isinstance(module, nn.Embedding):
            nn.init.normal_(module.weight, std=self.config.embed_init_std)
        elif isinstance(module, SiDEmbeddings):
            nn.init.normal_(module.numerical_direction, std=self.config.embed_init_std)
            nn.init.normal_(module.numerical_anchor, std=self.config.embed_init_std)
        elif isinstance(module, nn.Linear):
            nn.init.kaiming_uniform_(module.weight, 5 ** 0.5)
            if module.bias is not None:
                nn.init.zeros_(module.bias)
        elif isinstance(module, nn.LayerNorm):
            nn.init.ones_(module.weight)
            nn.init.zeros_(module.bias)
```

```
    def forward(
        self,
        categorical_inputs: torch.Tensor,
        numerical_inputs: torch.Tensor,
        text_inputs: torch.Tensor,
    ) -> torch.Tensor:
        input_embeddings = self.embeddings(
            categorical_inputs,
            numerical_inputs,
            text_inputs,
        )

        hidden_states = None
        for layer in self.layers:
            hidden_states = layer(input_embeddings, hidden_states)

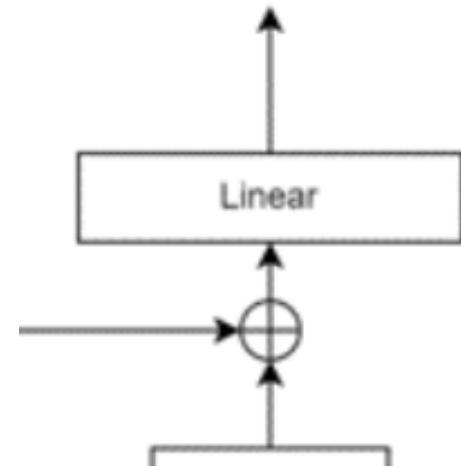
        hidden_states = self.normalization(hidden_states)
        return hidden_states
```

#03-4 Architecture

- SiDClassifier

```
class SiDClassifier(nn.Module):
    def __init__(self, config: SiDConfig):
        super().__init__()
        self.model = SiDModel(config)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

    def forward(
        self,
        categorical_inputs: torch.Tensor,
        numerical_inputs: torch.Tensor,
        text_inputs: torch.Tensor,
    ) -> torch.Tensor:
        hidden_states = self.model(categorical_inputs, numerical_inputs, text_inputs)
        logits = self.classifier(hidden_states)
        return logits
```



#03-5 Experiments

- Hyperparameter Settings

Hyper-parameter	SiD-Base
Number of Layers	18
Embedding Size	64
Hidden Size	512
Intermediate Size	1024
Number of Transform Blocks	1
Number of Attention Blocks	1
Hidden Dropout	0.5
Attention Dropout	0.5
Stochastic Depth	0.5

Table 2. Hyper-parameters of SiD model structure and its training.

Hyper-parameter	SiD-Base
Learning Rate	3e-4
Batch Size	128
Weight Decay	0.01
Gradient Clipping	1.0
Learning Rate Decay	Cosine
Minimum Learning Rate	3e-7
Epochs	100
No Bias Decay	True

- Experimental Results

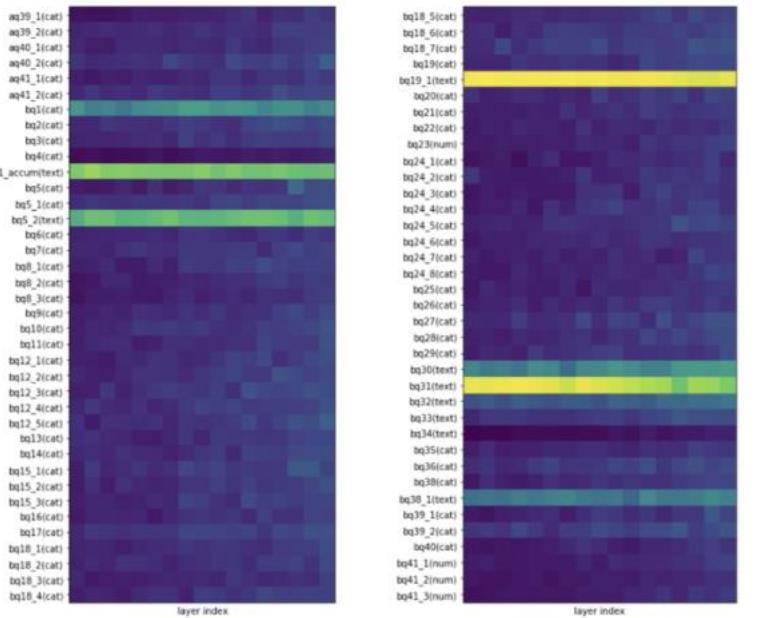
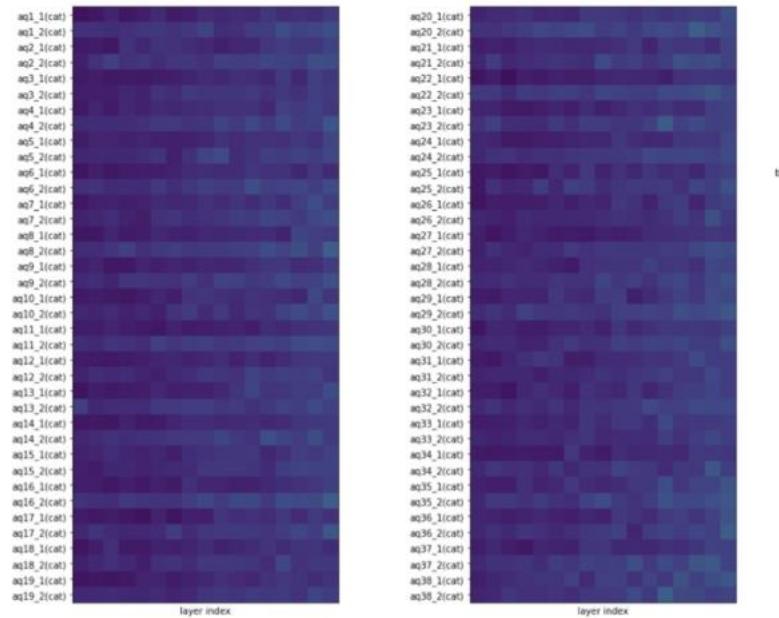
Model	CV Score	Competition Public Score	Competition Private Score
RandomForest	0.5150	–	–
LightGBM	0.5570	–	–
CatBoost	0.4945	–	–
Simple MLP (d=512, l=3)	0.7120	–	–
Simple MLP (d=512, l=18)	0.6646	–	–
SiD-Base	0.7331	0.7494	–
+ 10 Ensembles	0.7594	0.7663	–
+ 15 Ensembles	0.7628	0.7702	0.7733
+ 20 Ensembles	0.7654	0.7690	–

- Ablation Studies

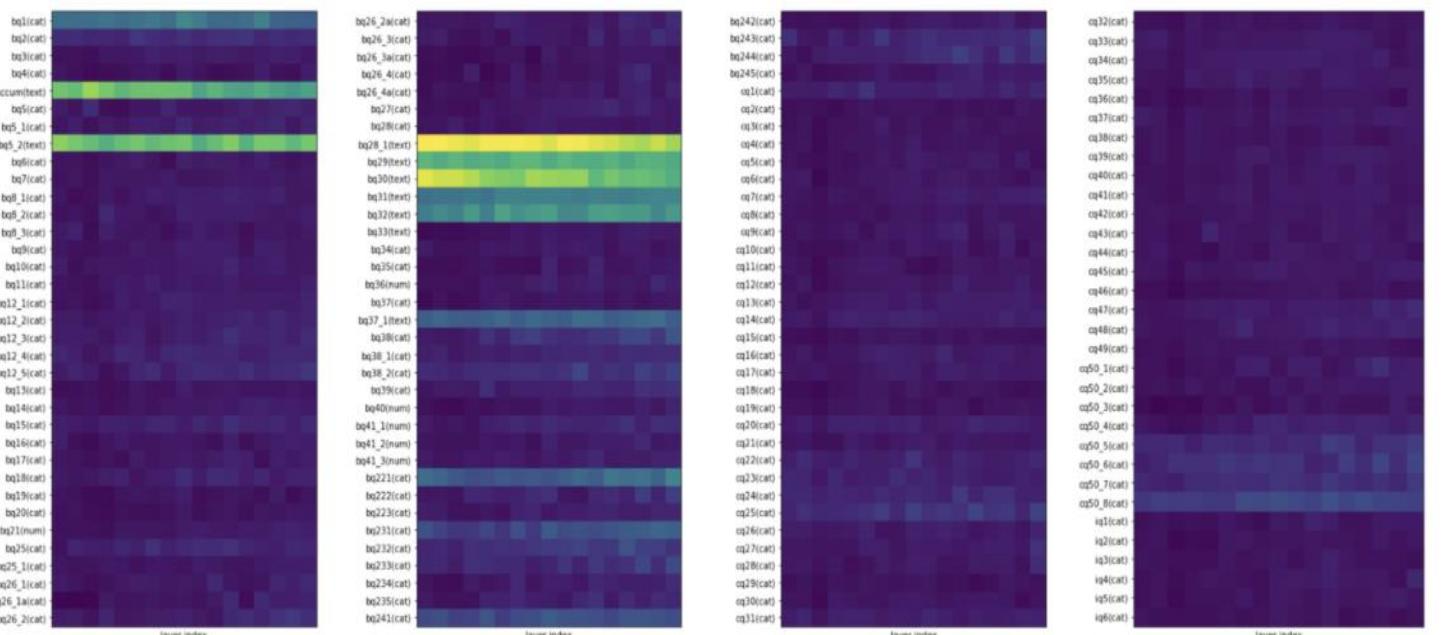
Model	CV Score	Competition Public Score
SiD-Base	0.7331	0.7494
w/o Sentence Embeddings	0.7028	0.7109
KoSimCSE Embeddings	0.7283	–
KoBART Input Embeddings	0.7210	–
KoBART + KoELECTRA Input Embeddings	0.7255	–
w/o Rank-Gauss Normalization	0.7299	–
w/o Standard Normalization	0.7328	–
No Regularization	0.5157	–
Weak Regularization	0.6810	–
20 Epochs Training	0.7092	–
200 Epochs Training	0.7335	–
1000 Epochs Training	0.7358	–

#03-6 Interpretability

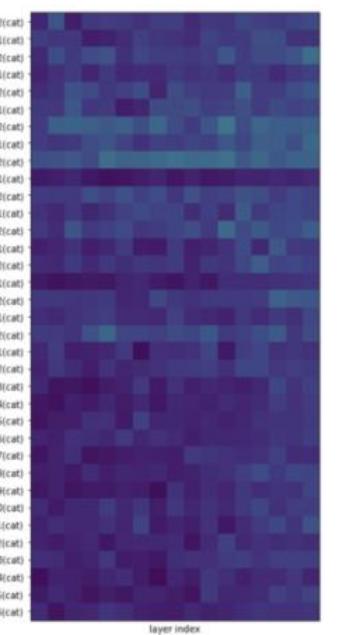
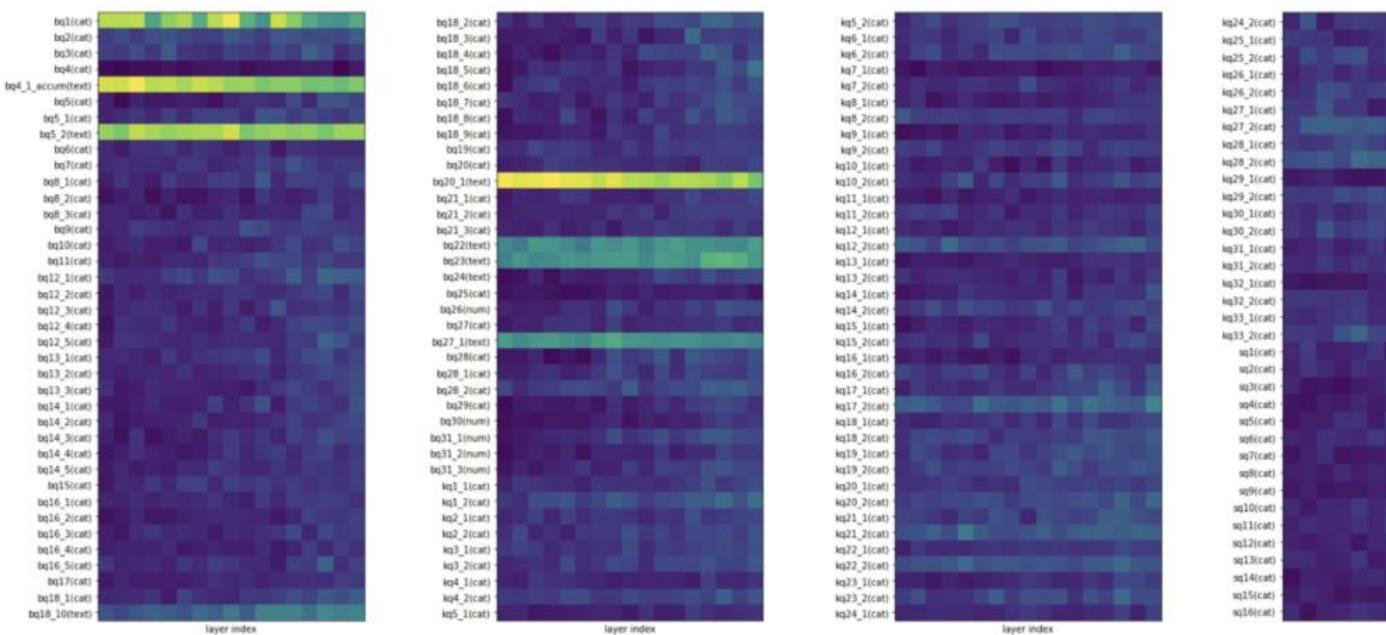
2017



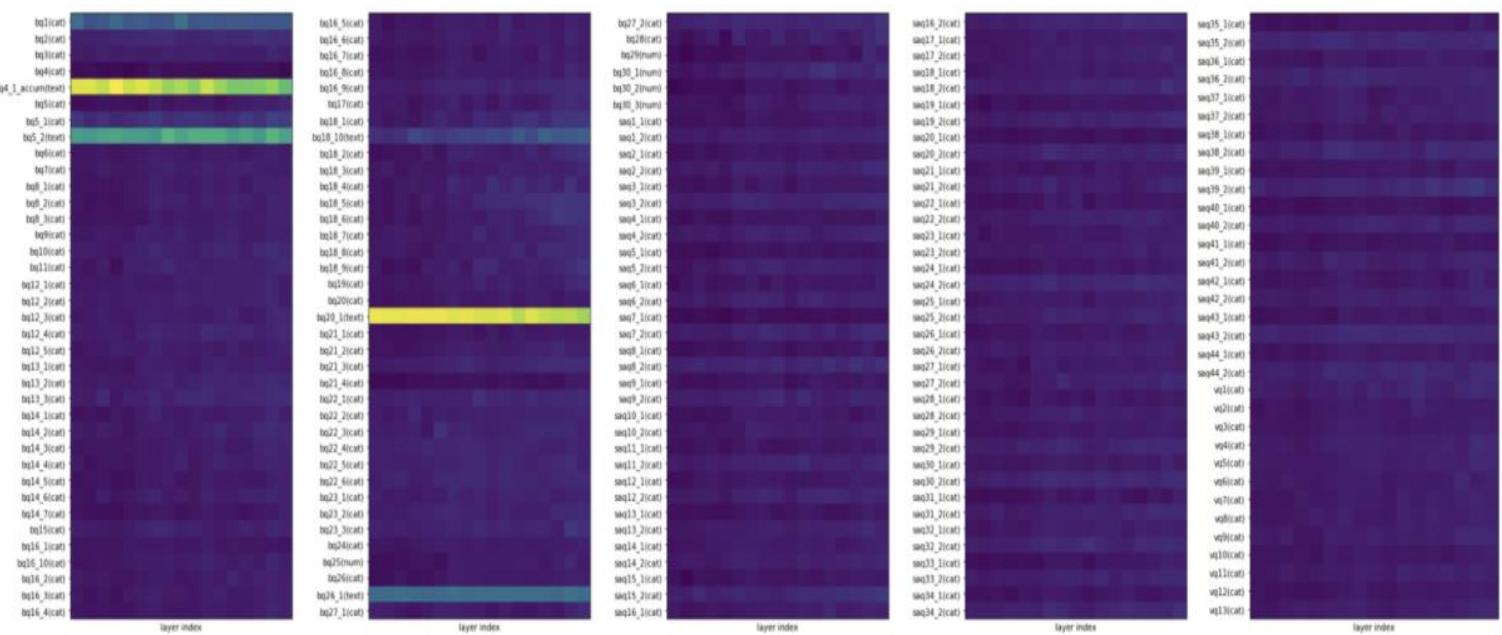
2019



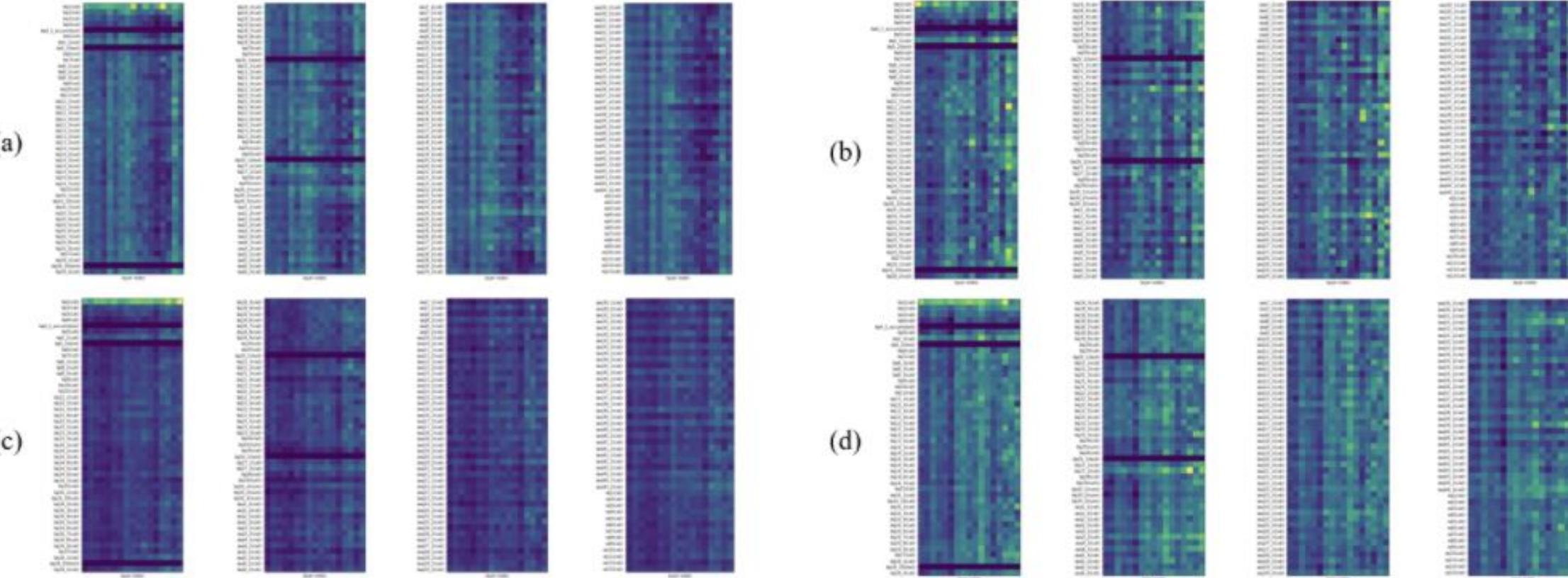
2018



2020



#03-6 Interpretability



Importance Mask

- a. 금융관리자
- b. 중고등학교 교사
- c. 가수
- d. 번역가

Name	Question Dialog
saq24_2	나. 귀하의 업무에 필요한【재정 관리】능력은 어느 수준입니까?
saq22_2	나. 귀하의 업무에 필요한【판단과 의사결정】능력은 어느 수준입니까?
saq15_2	나. 귀하의 업무에 필요한【사람 파악】능력은 어느 수준입니까?
saq26_2	나. 귀하의 업무에 필요한【인적자원 관리】능력은 어느 수준입니까?
saq16_2	나. 귀하의 업무에 필요한【행동 조정】능력은 어느 수준입니까?

(a)

Name	Question Dialog
saq38_2	나. 귀하의 업무에 필요한【정교한 동작】능력은 어느 수준입니까?
saq44_2	나. 귀하의 업무에 필요한【청력】능력은 어느 수준입니까?
saq39_2	나. 귀하의 업무에 필요한【음직임 통제】능력은 어느 수준입니까?
saq43_2	나. 귀하의 업무에 필요한【시력】능력은 어느 수준입니까?
saq16_2	나. 귀하의 업무에 필요한【행동 조정】능력은 어느 수준입니까?

(c)

Name	Question Dialog
saq24_2	나. 귀하의 업무에 필요한【재정 관리】능력은 어느 수준입니까?
saq19_2	나. 귀하의 업무에 필요한【가르치기】능력은 어느 수준입니까?
saq15_2	나. 귀하의 업무에 필요한【사람 파악】능력은 어느 수준입니까?
saq39_2	나. 귀하의 업무에 필요한【음직임 통제】능력은 어느 수준입니까?
saq12_2	나. 귀하의 업무에 필요한【학습전략】능력은 어느 수준입니까?

(b)

Name	Question Dialog
saq3_2	나. 귀하의 업무에 필요한【글 쓰기】능력은 어느 수준입니까?
saq31_2	나. 귀하의 업무에 필요한【전산】능력은 어느 수준입니까?
saq44_2	나. 귀하의 업무에 필요한【청력】능력은 어느 수준입니까?
saq7_2	나. 귀하의 업무에 필요한【창의력】능력은 어느 수준입니까?
saq24_2	나. 귀하의 업무에 필요한【재정 관리】능력은 어느 수준입니까?

(d)

THANK YOU

