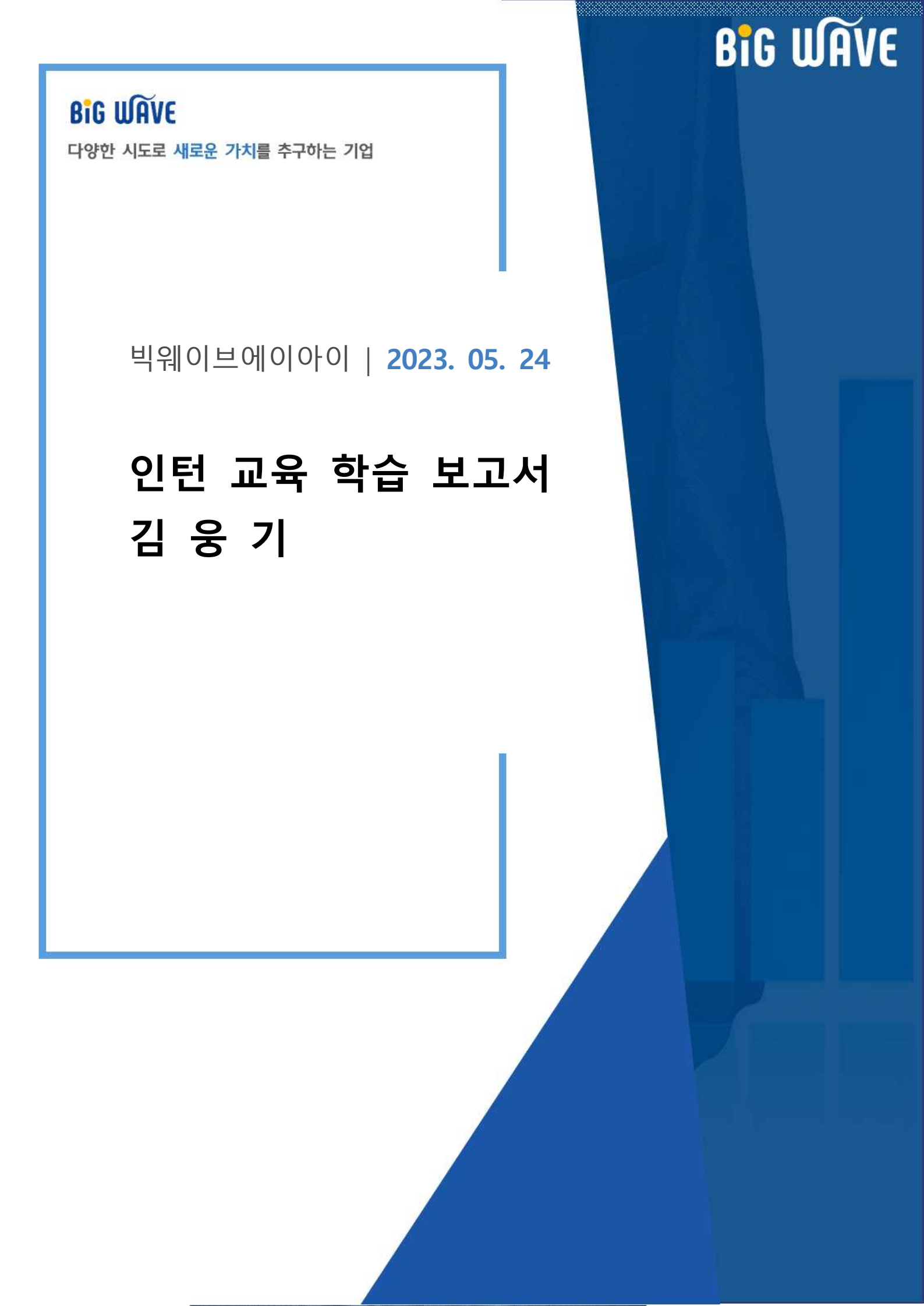
BIG WAVE

다양한 시도로 새로운 가치를 추구하는 기업

빅웨이브에이아이 | **2023. 05. 24**

인턴 교육 학습 보고서

김 웅 기



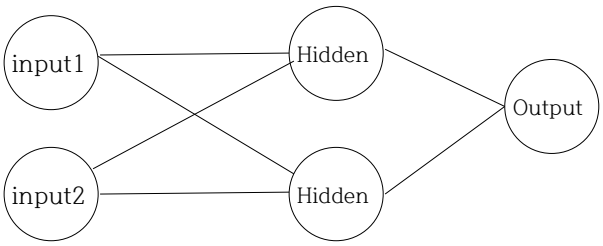
Ⅰ. 목차

01 대학원 합격 확률 예측

1.1 딥러닝이란?

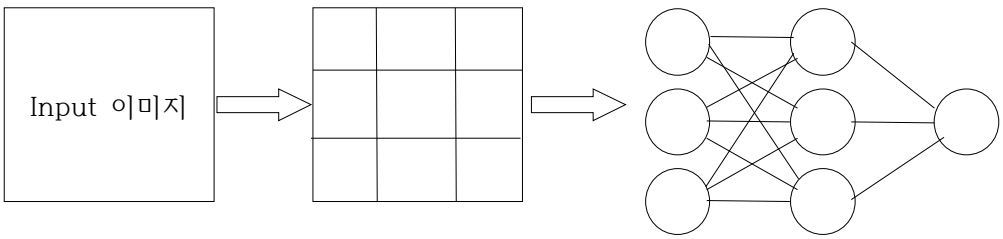
Deep Learning

머신러닝에서 Input과 Output 사이에 Hidden Layer를 추가한 것



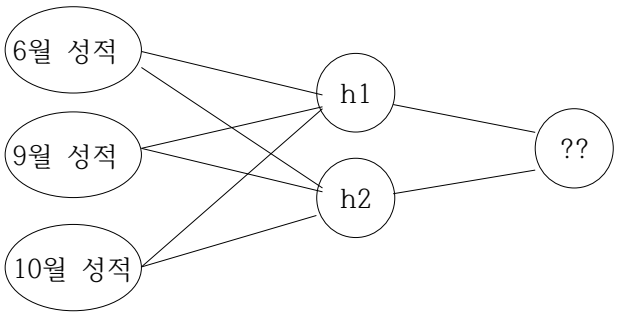
- Hidden Layer : 기계가 생각하는 공간
- Hidden Layer를 거쳐 Output으로 가는 과정에서 가중치(w)를 계산하여 예측에 반영
- Hidden Layer에서 Feature Extraction(특성추출)을 하여 '~일 것이다.'로 예측하게 된다.

1.1.1 이미지 데이터의 딥러닝 구조



- Input 이미지를 잘게 쪼갠 후 행렬형태로 변환하여 딥러닝 진행

1.1.2 딥러닝 계산 과정



- w : 가중치
- h : Hidden Layer 계산값
- ?? : 예측값
- h1 = 6월 * w1 + 9월 * w2 + 10월 * w3
- h2 = 6월 * w4 + 9월 * w5 + 10월 * w6
- ?? = h1 * w7 + h2 * w8

1.2 Activation Function(활성화함수)

Hidden Layer에 들어가는 Activation Function 종류

- Sigmoid : 분류 문제에 사용되며, 특정 기준 이하면 0, 이상이면 1을 출력하는 활성화 함수. 기울기 손실 문제가 발생할 수 있다는 문제점이 있다.
- Rectified Linear Units(ReLU) : 값이 음수면 0으로 출력하고 양수면 값을 그대로 출력하는 활성화 함수. 기울기 손실이 발생하지 않지만 값이 음수일 경우 모조리 0으로 변환되기에 가중치 업데이트가 이루어지지 않을 수 있다.
- Hyperbolic Tangent : -1~1 사이의 값을 출력하며, 평균이 0인 활성화 함수. Sigmoid에서 발생하는 기울기 소실 증상이 비교적 적게 발생한다.

- 활성함수 없이 예측을 할 경우 선형적이고 단순한 예측이 되며, 활성함수를 포함하여 예측을 할 경우에는 비선형적이고 복잡한 예측이 가능하여 다양한 상황에 적용가능하다는 장점이 있다.
- 활성화 함수는 은닉층에 사용하지만 결과를 0~1 사이로 표현하고 싶으면 출력층에 사용할 수도 있다.

1.3 Optimizer

・손실을 최소화하는 Loss Function을 찾고 최적화하는 파라미터

Optimizer의 종류

・Momentum : 가속도를 유지하여 최적화
・AdaGrad : 자주 변하지 않는 w는 크게, 자주 변하는 w는 작게하여 최적화
・RMSProp : AdaGrad에 제곱을 취한 최적화 방법
・AdaDelta : AdaGrad에서 a가 너무 작게되어 학습이 안되는 현상을 방지하는 최적화 방법
・Adam : RMSProp과 Momentum을 결합한 최적화 방법

1.4 Tensorflow로 모델 구축하기

1. 모델 만들기
・tf.keras.models.Sequential([레이어1, 레이어2, 레이어3])
・tf.keras.layers.Dense(노드수) : 노드수에 기준은 없으나 보통 2의 제곱수를 기입한다.
・tf.keras.layers.Dense(노드수, activation= ‘’)
마지막 레이어 : 출력하고자 하는 값 개수만큼 노드를 지정하며, 활성화 함수를 지정해줘야한다.
2. 모델 컴파일하기
・model.compile(optimizer = ‘’ , loss = ‘’ , metrics = [ ])
3. model 학습하기
・model.fit(X, Y, epochs =)
4. 예측
・model.predict([])

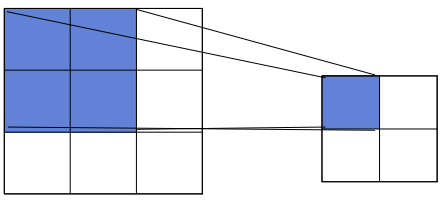
※ 모델 성능을 높이는 방법
・완벽한 데이터 전처리, 파라미터 튜닝, 레이어 추가, 삭제, epoch 횟수 조정 등

02 이미지 학습과 CNN

・뉴럴 네트워크에 넣을 수 있는건 무조건 숫자 형태여야 한다.
・이미지의 Pixel 데이터는 숫자로 표현이 가능하며, 색상은 RGB로 표현이 가능하다.
・실제값과 예측값의 비교로 총 손실값을 계산하여 모델의 성능을 파악할 수 있다.
・model 구축 시 확률 예측 문제일 경우
1. 마지막 레이어 노드 수를 카테고리 개수만큼 지정
2. Cross Entropy Loss Function 함수 사용
・행렬을 1차원으로 압축시켜주는 레이어 : tf.keras.layers.Flatten
・Flatten의 문제점 : 1차원으로 만들기에 2차원 데이터일 때 학습한 가중치들이 다른 2차원 이미지에 적용할 수 없게 됨 -> 응용력이 없는 AI
이에 대한 해결책으로 나온 것이 Convolution Layer
・Convolution Layer 과정
1. 이미지에서 중요한 정보를 추려서(필터링(압축)) 복사본을 여러 장 만든다.
2. 그곳엔 이미지의 중요한 Feature 특성이 담겨져 있다.(이미지 특성이 각각 다르게 강조되게)
3. 1, 2를 통해 특성추출(Feature Extraction)

2.1 Convolution Layer

2.1.1 Feature map 만들기



<Layer1> <Layer2>

・단순 Convolution의 문제점 : Feature의 위치 -> 해결책 : Pooling Layer

Pooling Layer
• Max Pooling(최대값만 추림) : Layer1에서 색칠된 부분의 feature 중 최대값을 Layer2에 전달하는 방법.
• Average Pooling(평균값으로 추림) : Layer1에서 색칠된 부분의 feature의 평균을 Layer2에 전달하는 방법.
장점 : translation invariance(이미지 위치의 영향이 없음)

• Convolution + Pooling Layer를 도입하면 특징 추출 + 특징을 가운데로 모아주는 효과를 얻을 수 있다.
• Convolution Neural Network의 일반적인 구성법
Input Image -> Filters -> Convolition Layer -> Pooling -> Flattening

2.1.2 model에 convolution layer 추가하기

• tf.keras.layers.Conv2D(복사할 이미지 수, (크기, 크기), Padding = "이미지 크기", activation = '', input_size = (데이터 하나의 shape)
• tf.keras.layers.Maxpooling2D((풀링사이즈, 풀링사이즈))
• 레이어 순서 : Conv -> Pooling -> Flatten -> Dense -> 출력
• Conv와 Pooling은 여러 번 가능하다.
• 모델 평가하기 : model.evaluate(testX, testY)

※ overfitting 줄이기
1. epoch 1회 끝날때마다 채점(model.fit(validation_data=())
2. Dense layer 추가
3. Conv + Pooling 추가
4. Dropout(윗 레이어의 노드 일부 삭제)

2.2 Tensorflow 사용하기

2.2.1 Tensorflow로 전처리하기

・tf.keras.preprocessing.image_dataset_from_directory(경로, 이미지 사이즈(,), batch_size = 데이터 수)

・이미지 나누기 (label에 따라)

import shutil

shutil.copyfile(어떤 파일을, 어떤 경로로)

・validationset 만들기

tf.keras.preprocessing.image_dataset_from_directory(경로, 이미지 사이즈(,), batch_size = 데이터 수, subset = ‘validation’, validation_split = 0~1사이 값, seed =)

2.2.2 model 저장하기

・model 저장항목

레이어 설정

loss 함수 종류

optimizer 종류

훈련 후의 w값(가중치)

・model 저장 방법

1. 전체 모델 저장
   ・model.save(‘폴더/모델명’)
   ・tf.keras.models.load_model(‘폴터/모델명’)
2. w(가중치)값만 저장
   ・epoch 마다 checkpoint 저장 가능
   ・콜백함수=tf.keras.callbacks.ModelCheckpoint(filepath= ‘파일명’, save_weight_only = True, save_freq = ‘epoch’)
   ・model.fit(callbacks=[콜백함수])
   ・filepath 이름 옆에 {epoch}을 넣어주면 epoch마다 저장. 없으면 덮어쓰기 이기 때문에 하나의 파일만 생성됨
   ・ModelCheckpoint(monitor=’ val_acc’, mode= ‘max’) : val_acc가 가장 높은 것 저장

・불러오기 : 모델을 만들고 compile 후 model.load_weights(‘’)

2.2.3 Image Augmentation(이미지 증강)

・증강된 데이터 사본을 생성하는 방법.
・원본 이미지 + 증강된 이미지 -> 모델 학습
・layers.experimental.preprocessing.RandomFlip : 이미지 뒤집기
・layers.experimental.preprocessing.RandomRotation : 이미지 위치 랜덤지정
・layers.experimental.preprocessing.RandomZoom : 이미지 확대
-> epoch = 10 : 증강을 하지 않았을 경우 같은 이미지 10번 epoch, 증강을 했을 경우 같은 이미지의 다른 버전 10번 epoch

・문제점 : 원본 데이터가 적거나 너무 많을 경우 효과가 없다, 시간이 오래 걸린다.

2.3 전이학습

・전이학습 : 다른 사람이 구축한 모델을 가져와 쓰는 것
・model을 다운받을 수 있는 url을 얻어 다운받아 사용
・전이학습방법(inception_model=InceptionV3(input_shape=(150,150,3), include_top=False, weights=None))
1. 만든 사람이 정한 input_shape을 데이터에 맞게 변경
2. include_top : 마지막 레이어도 로드할 것인지 결정
3. weights : 가중치를 가져올 것인지 결정
(weights 파일 불러오기 : model.load_weights(파일명))
・Summary()를 보면 Connected to는 API로 레이어를 만들면 생김. 마지막 레이어가 없는 것을 확인할 수 있다.
4. 마지막에 Dense레이어 추가 후 학습. (마지막 레이어 윗부분은 학습시키면 안됨)

03 Sequence 학습과 RNN

3.1 단어예측 Dense 레이어로 학습

- 단어 -> 정수 형태로 (원 핫 인코딩)
- 일반적인 Dense 레이어에 집어넣을땐 단어들의 순서 정보는 사라진다.
- 해결책 : Simple RNN(Recurent)레이어

3.1.1 RNN

- Simple RNN (예측할 때 이전 예측 결과 일부를 씀)

Input -> tanh(h1) -> 예측1

Input -> tanh(h2 + h1의 일부) -> 예측2

Input -> tanh(h3 + h2의 일부) -> 예측3(우리가 원하는 것)

- RNN은 Sequence데이터 학습에 좋음(단어, 음성, 가격예측)
- 자료의 순서 & 자료 간 연결성이 있을 때 도입하면 좋은 성과를 보인다.
- RNN 적용 예시

1. Input vector -> Sequence 예측
2. Sequence 입력 -> Vector 예측
3. Sequence 입력 -> Sequence예측(seq to vec to seq = encoder to decoder)
   - encoder : 한국어 sequence를 이상한 행렬로 압축
   - decoder : 행렬에서 정보를 뽑아서 영어 sequence로 뽑음

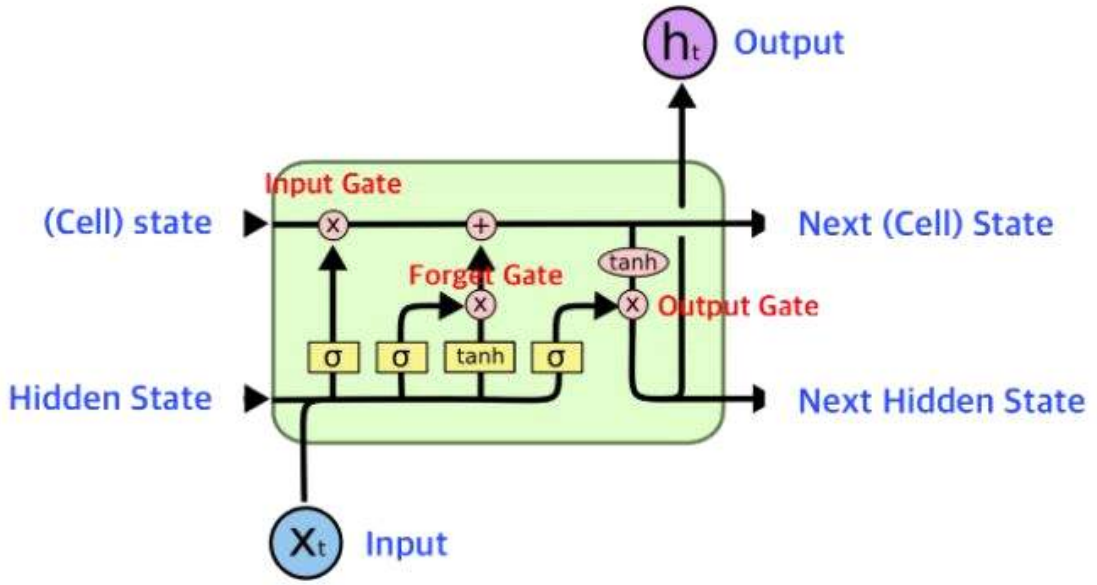
- Simple RNN의 문제 : Diminishing Gradient

앞쪽 단어의 중요도는 낮아진다. 중요 단어가 앞에 있으면 정확도 하락을 야기한다.

- 해결책 : LSTM(Long Short Term Memory)

3.2 LSTM

• RNN에 장기기억층(cell state)를 추가한 모델
• 장기기억층에 이전 단어 혹은 데이터의 가중치를 보존해두고 모델을 학습하는 방식



출처 : https://wikidocs.net/152773

• tensorflow에서 LSTM 만들기
tf.keras.layers.LSTM(노드수, input_shape=(,))

3.2.1 작곡 AI 만들기

• 문자를 숫자로 바꾸는 방법
1. Bag of words(단어 주머니) 만들기 (출현하는 모든 단어 모음)
2. 문자 -> 숫자 변환기 만들기
3. 원본 데이터 숫자로 바꾸기(변환기 사용)
• 모델 만들기
tf.keras.layers.LSTM(노드수, input_shape=(,)),
tf.keras.layers.Dense(), ...,
• 마지막 레이어
tf.keras.layers.Dense(예측 가지수, activation= ‘’)
• 모델 예측값 뽑기
1. 저장해놓은 모델 파일 불러오기
2. 문자 집어넣어서 model.predict() 하기

04 다양한 CSV 데이터 다루기

・tensorflow에서 feature_columns만들기

・numeric_column : 정수화

feature_columns.append(tf.feature_column.numeric_column(''))

・bucketized_column : 구간화

tf.feature_column.bucketized_column(Age, boundaries=[])

・indecator_column : 멀티 핫 인코딩

cat = tf.feature_column.categorical_column_with_vocabulary_list('' ,칼럼에 들어가 있는 유니크한 문자리스트)

tf.feature_column.indicator_column(cat)

・ embedding_column : 범주형 데이터를 저차원으로 임베딩(구간분할 후 특성교차 적용)

vocab = data[''].unique()

cat = tf.feature_column.categorical_column_with_vocabulary_list('', vocab)

tf.feature_column.embedding_column(cat, dimension=)

05 Generative Adversarial Net

・GAN(Generative Adversarial Net) : 적대적 생성 신경망
・모델 2개를 경쟁시켜 새로운 데이터 생성
(Generator 모델(가짜 데이터 생성) + Discriminator 모델(생성 데이터 검증))
・Discriminator가 진짜인지 가짜인지 구분못하면 Generator를 꺼낸다.
・모델 만들기
1. 초반 레이어 구성
2. Conv2DTranspose 적용을 위해 이전 레이어의 아웃풋이 이미지 모양을 하도록 Reshape
3. Covariate shift(학습데이터의 분포가 테스트 데이터의 분포와 다른 상황) 문제해결을 위해 BatchNormalization() 적용
4. 레이어의 인풋을 균일한 분포를 가지게 한번 전처리 해줌(Conv 레이어와 같이 사용)
5. 이미지 모양, 컬러를 출력할 수 있게 마지막 레이어 디자인

기본적인 GAN 학습 Step
・Discriminator 학습
1. 실제 이미지와 가짜 이미지 준비
2. 구분 잘하게 학습(model.train_on_batch(이미지,정답))
・Generator 학습
1. 랜덤 숫자 행렬 100개 준비
2. 1로 라벨링해서 집어넣기
・모델 합치기
Sequential([Generator모델, Discriminator모델])

・랜덤 숫자 x100 -> Generator -> (Generated 이미지, 정답1)추출 -> Discriminator -> 진짜 / 가짜 구분 -> 총 Loss 계산(최소화 시켜야 한다.)
・Generator는 Loss를 최소화할 수 있게 진짜같은 이미지를 만들어야한다.

5.1 Discriminator

・모델 구축하기(강의에서 사용한 코드)

```
discriminator = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(필터값,(커널사이즈(필터)),strides=(풀링후의크기),padding='same', input_shape=[64,64,1]),
  tf.keras.layers.LeakyReLU(alpha=0.2),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Conv2D(64, (3,3), strides=(2, 2), padding='same'),
  tf.keras.layers.LeakyReLU(alpha=0.2),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

※ 일반적인 ReLU가 아닌 LeakyReLU를 사용하는 이유

・ReLU는 음수를 0으로 치환하기 때문에 가중치 업데이트가 이루어지지 않을 수 있다.

・LeakyReLU는 0이하의 값에 아주 작은 값을 곱해서 이를 방지한다.

5.2 Generator

• 모델 구축하기(강의에서 사용한 코드)

```
generator = tf.keras.models.Sequential([
  tf.keras.layers.Dense(4 * 4 * 256, input_shape=(100,) ),
  tf.keras.layers.Reshape((4, 4, 256)),
  tf.keras.layers.Conv2DTranspose(256, 3, strides=2, padding='same'),
  tf.keras.layers.LeakyReLU(alpha=0.2),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2DTranspose(128, 3, strides=2, padding='same'),
  tf.keras.layers.LeakyReLU(alpha=0.2),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2DTranspose(64, 3, strides=2, padding='same'),
  tf.keras.layers.LeakyReLU(alpha=0.2),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2DTranspose(1,3,strides=2,padding='same',
activation='sigmoid')
])
```

• 예측하기

```
랜덤숫자 = np.random.uniform(-1,1,size=(150,100))
가짜사진들 = generator.predict(랜덤숫자)
마킹한정답0 = np.zeros(shape =(150,1))
```

5.2.1 Conv2DTranspose

• Convolution은 이미지를 중요한 부분만 추출하고 싶을 때 사용한다.

• Transposed Convolution은 행렬을 Transpose한 다음 컨볼루션을 진행한다.

• 효과

  • 원본 이미지의 사이즈를 키우면 빈 공간이 생긴다. 여기에 임의의 숫자를 컴퓨터가 채우게 된다.

  • 여기에 임의의 숫자가 아니라 학습된 숫자를 끼워넣을수록 생성하고자 하는 이미지의 형상과 가까워진다.

• stride 옵션 : 커널을 몇 번 건너뛰면서 적용해줄지 결정(얼마나 이미지를 키울 것인지)

• kernel 옵션 : 일반 Conv kernel과 같음

・아웃풋 사이즈 공식

output size = (input size -1) x stride -2 x padding + (kernel size -1) + 1

5.3 Train_on_batch() 사용하기

1. 랜덤 숫자를 Generator에 집어넣어 가짜 이미지 생성
2. 진짜 이미지를 데이터 셋에서 가져옴
3. Discriminator모델에 train_on_batch()함수를 써서 1번은 0으로, 2번은 1로 라벨링 한 다음 학습
4. 랜덤 숫자 noise를 1로 라벨링한(진짜 이미지로 라벨링 한) 데이터 셋 만들기
5. 그 데이터 셋으로 GAN 전체를 train_on_batch() 함수로 학습
6. 이미지 개수만큼 위 과정을 반복하면 epoch 1회

・코드

1. Discriminator (진짜 데이터 학습)

   loss1 = discriminator.train_on_batch(진짜사진들, 마킹한정답1)

   loss2 = discriminator.train_on_batch(가짜사진들, 마킹한정답0)

2. GAN (생성된 데이터 학습)

   loss3 = GAN.train_on_batch(랜덤숫자, 마킹한정답1)

주의점 : 진짜데이터로 먼저 학습하면 편향 발생 가능 -> 섞어서 학습하면 편향 방지