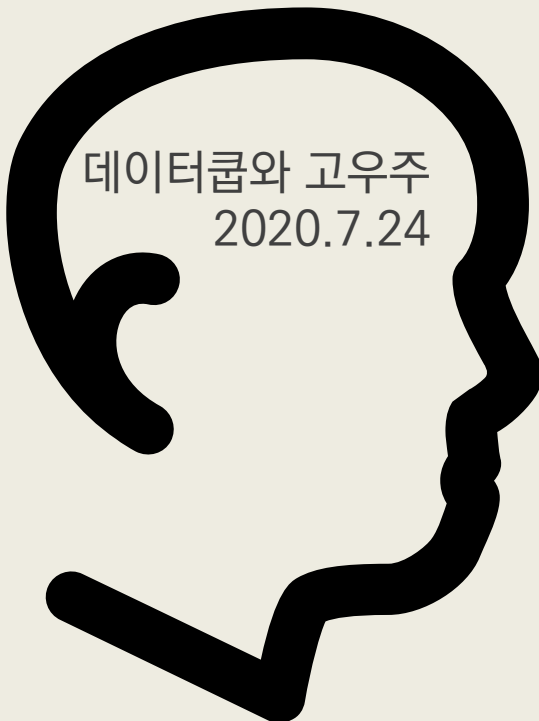


딤러닝 3일차

데이터쿵와 고우주
2020.7.24

Input Regularization / 과적합 방지



데이터쿵와 고우주
2020.7.24

Ways to scale inputs

- Linear scaling to the interval $[0, 1]$

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- Linear scaling to the interval $[-1, 1]$

$$x_i = 2 \left(\frac{x_i - \bar{x}}{x_{max} - x_{min}} \right) - 1$$

Ways to scale inputs

- Normalization or Standardization (making variable approx. std. normal)

$$x_i = \frac{x_i - \bar{x}}{\sigma}; \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

가중치 규제(Weight Regularization)

- 가중치의 분포를 균일하게 하여 높은 가중치를 갖는 손실 함수에 페널티를 명시적으로 추가
- Cost = Loss + Penealty (Penalty = $\frac{1}{2}\lambda\|w\|^2$)
- 지나치게 특정 가중치의 크기가 증가하는 것에 제약을 걸어줌으로써, 가중치 벡터들이 고르게 퍼지도록 하는 효과

$$J = \frac{1}{2n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^m w_j^2$$

- Approach to Ridge Regression (L2 Regularization: L2 Norm)

| Batch Size에 따른 Epoch, Step

Epoch 입력부터 가중치 업데이트까지 1회 학습할 때 1 epoch

Step(=Iterate) Weight와 Bias를 1회 업데이트하는 것을 1 step

Batch Size 1회 step에서 사용한 데이터의 수

ex) Batch Size가 100이라고 가정하고 Step이 50이면 약 500개의 데이터를 이용

$$s = (n * e) / b$$

n = num of sample : 전체 학습할 데이터의 개수

e = epochs: Epoch 수

b = batch size: 배치 사이즈

s = steps: Step 수

Checkpoint에서 생성 지점을 epochs, batch_size, steps로 각각 지정

Data Shuffling

- Shuffling은 주기적 움직임을 피하고 수렴을 돕기 위해 각 에포크 후에 데이터를 섞는 것
- Shuffling은 매번 데이터가 동일한 순서로 표시되지 않으며 배치가 정확히 동일한 것은 아님

```
model.fit(x_train, y_train,  
          batch_size=10,  
          validation_data=x_val,  
          epoch=20,  
          shuffle=True,  
          verbose=2)
```

Shuffle the Data!

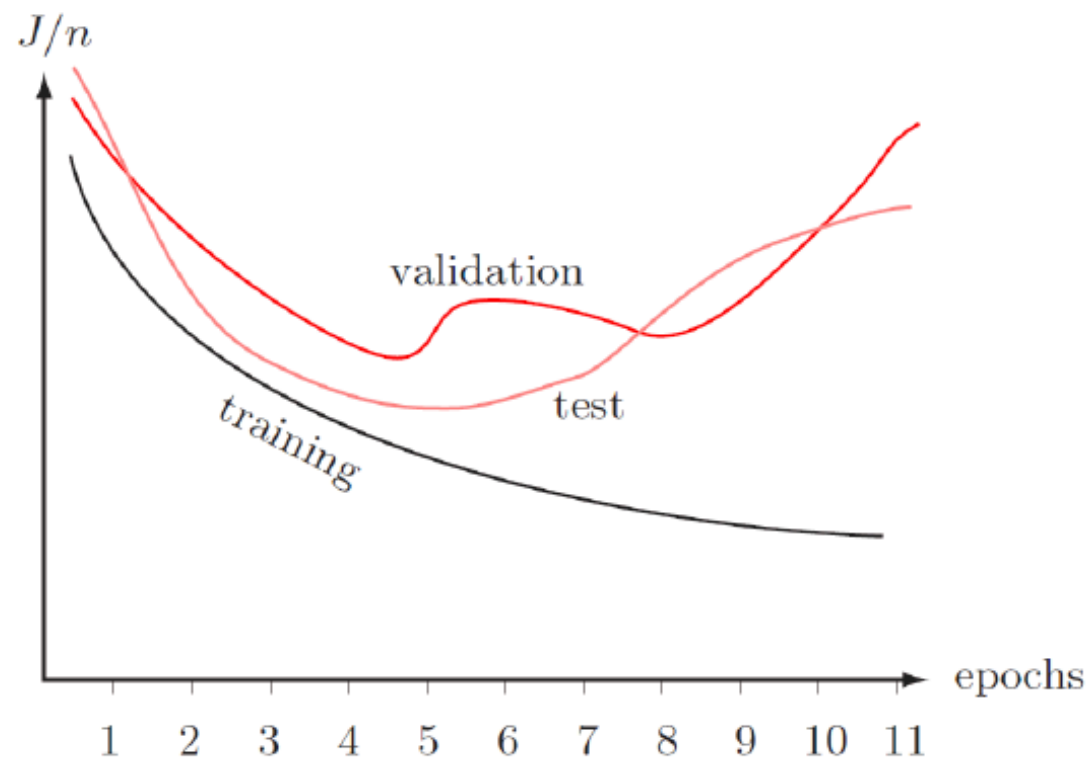


- Shuffling은 주기적 움직임을 피하고 수렴을 돕기 위해 각 에포크 후에 데이터를 섞는 것

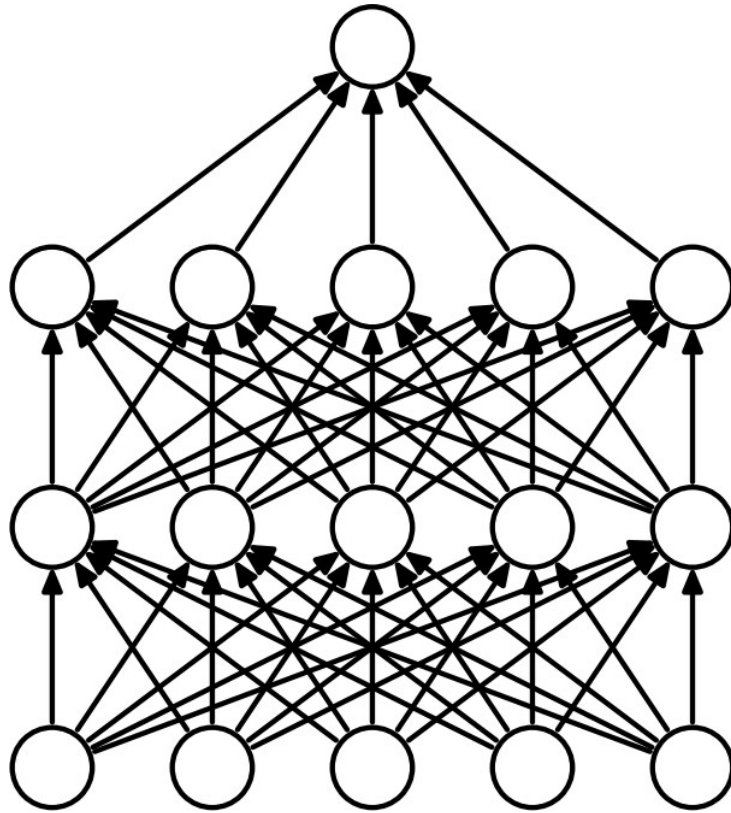
Regularizing

- 과적합(Overfitting)을 방지하기 위한 “정규화(Regularization)하는 방법

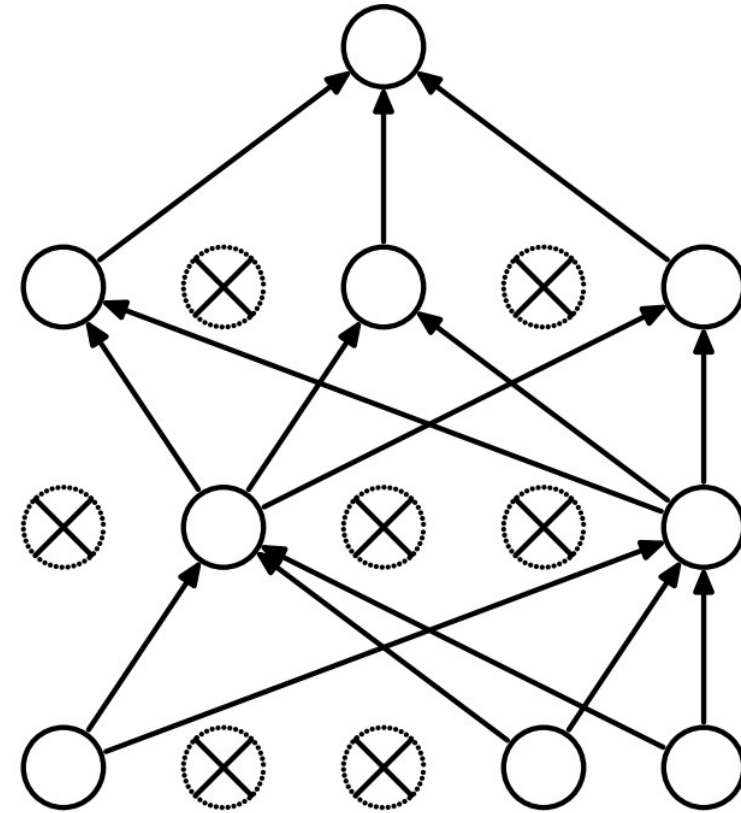
- 많은 데이터 학습
- 비용 함수의 정규화 페널티 주기 (L2 Norm)
- 드롭아웃(Dropout)
- 조기 중지(Early Stopping)
- SGD, Mini-batch Gradient Descent



Dropout—Visualization



(a) Standard Neural Net



(b) After applying dropout

Early Stopping

- $|\nabla_w E| \rightarrow 0$ or $|\Delta E| \rightarrow 0$ or $|\Delta w| \rightarrow 0$
 - Error에 대한 Gradient가 거의 0
 - 또는 Error의 변화가 거의 없는 경우
 - 또는 가중치 변화가 거의 없는 경우
- Early Stopping using validation set
- Limited number of epochs
- Epoch 수 증가에 따른 learning rate의 감소폭을 크게 설정
- 최근에는 딥러닝 학습 시 주기적으로 모델을 저장하고, 검증 데이터셋에 대한 결과를 모니터링하면서 적당한 수준에서 학습을 중지하는 방법이 제일 많이 사용

- Batch Normalization

- 딥러닝 학습을 잘 하게 만드는 방법
- Activation function 에 투입되는 값을 정규화하기 때문에, vanishing gradient 문제 에도 적합하며 exploding gradient 문제도 완화시켜준다.
- Dropout을 대체할 수 있다고 함
- Dropout보다 Batch Normalization을 사용할 때, 기대되는 학습시간 짧음

ANN 실습

데이터쿵와 고우주
2020.7.24

- Titanic Dataset – Binary Classification
- Wine Dataset – Classification
- IMDB Dataset – 평점 예측
- Boston House Price – 회귀 분석

완전 연결 네트워크

문제유형	마지막층의 활성화함수	손실 함수	평가검증
이진분류	Sigmoid	binary_crossentropy	accuracy
단일 레이블 분류	SoftMax	categorical_crossentropy	accuracy
다중 레이블 분류	Sigmoid	binary_crossentropy	accuracy
임의의 값에 대한 회귀	-	mse	rmse, mae
0과 1사이 값의 회귀	Sigmoid	mse 또는 binary_crossentropy	rmse, mae, accuracy

Optimizer

- adam과 rmsprop 주로 사용
- adam의 학습률: 0.001
- SGD, adagrad 학습률: 0.01

Tensorflow.Keras

데이터쿵와 고우주
2020.7.24

콜백(callback)

- 모델의 fit() 메서드가 호출된 때 전달되는 객체
 - 훈련하는 동안 모델은 여러 지점에서 콜백을 호출콜백은 모델의 상태와 성능에 대한 모든 정보에 접근
 - 훈련 중지, 모델 저장, 가중치 적재 또는 모델 상태 변경 등을 처리
-
- **모델 체크포인트 저장** : 훈련하는 동안 어떤 지점에서 모델의 현재 가중치를 저장
 - **조기 종료(early stopping)** : 검증 손실이 더 이상 향상되지 않을 때 훈련을 중지
 - **훈련하는 동안 하이퍼파라미터 값을 동적으로 조정.** : 옵티마이저의 학습률 등
 - **훈련과 검증 지표를 로그에 기록하거나 모델이 학습한 표현이 업데이트될 때마다 시각화**

ModelCheckpoint와 EarlyStopping

- **EarlyStopping** 콜백을 사용하면 정해진 에포크 동안 모니터링 지표가 향상되지 않을 때 훈련을 중지
- **ModelCheckpoint** 훈련하는 동안 모델을 계속 저장하며, 지금까지 가장 좋은 모델만 저장가능

fit() 메서드의 callbacks 매개변수를 사용하여 원하는 개수만큼 콜백을 모델로 전달

```
callback_list = [  
    keras.callbacks.EarlyStopping(  
        monitor='val_accuracy', # 모델의 검증 정확도 모니터링  
        patience=1, # 1 에포크보다 더 길게 향상되지 않으면 중단  
    ),  
    keras.callbacks.ModelCheckpoint(  
        filepath='my_model.h5', # *.h5로 저장  
        monitor='val_loss',  
        save_best_only=True, # 가장 좋은 모델  
    )  
]
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics[ ' accuracy'])  
model.fit(x,y, epochs=10, batch_size=32, callbacks=callback_list, validation_data=(x_val, y_val))
```

ReduceLROnPlateau 콜백

- 검증 손실이 향상되지 않을 때 학습률을 작게 만든다
- 손실 곡선이 평탄할 때 학습률을 작게 하거나 크게 하면 훈련 도중 지역 최솟값에서 효과적 중단

```
callback_list = [  
    keras.callbacks.ReduceLROnPlateau(  
        monitor='val_acc',  
        factor=0.1, # 콜백이 호출되면 학습률을 10배로 줄임  
        patience=10,  
    )  
]
```

텐서보드

- 훈련 모델의 내부에서 일어나는 모든 것을 시각적으로 모니터링
- 모델의 최종 손실 외에 더 많은 정보를 모니터링하면 모델 작동 시각화
- 모든 브라우저에서 작동
 - 훈련하는 동안 측정 지표를 시각적으로 모니터링
 - 모델 구조를 시각화
 - 활성화 출력과 그래디언트의 히스토그램
 - 3D로 임베딩 표현

코드 7-8 텐서보드 로그 파일을 위한 디렉터리 생성하기

```
mkdir my_log_dir
```

```
callbacks = [
```

```
    keras.callbacks.TensorBoard(
```

```
        log_dir='my_log_dir', # 저장될 path
```

```
        histogram_freq=1, # 1 에포크마다 활성화 출력의 히스토그램을 기록
```

```
        embeddings_freq=1) # 1 에포크마다 임베딩 데이터를 기록
```

```
]
```

```
history = model.fit(x_train, y_train, epochs=20, batch_size=128, validation_split=0.2, callbacks=callbacks)
```

```
tensorboard --logdir=my_log_dir
```

정규화(normalization)는 머신 러닝 모델에 주입되는 샘플들을 균일하게 만드는 광범위한 방법
평균을 빼고, 표준 편차로 나누어 분산을 (0,1)로 만들어준다

```
normalized_data = (data - np.mean(data, axis=...)) / np.std((data, axis=...))
```

배치 정규화

- 훈련 과정에서 사용된 배치 데이터의 평균과 분산에 대한 지수 이동 평균을 내부에서 유지(momentum 기본값 0.99)
- 배치 정규화의 주요 효과는 잔차 연결과 매우 흡사하게 그래디언트의 전파를 도와 입력과 출력의 분포가 유지로 더 깊은 네트워크를 구성

Conv2D 층

```
conv_model.add(layers.Conv2D(32, 3, activation='relu'))  
conv_model.add(layers.BatchNormalization())
```

Dense 층

```
dense_model.add(layers.Dense(32, activation='relu'))  
dense_model.add(layers.BatchNormalization())
```