

심층신경망 성능향상



가중치 초기화

- 가중치 초기화 기법은 뉴런 포화가 일어날 가능성을 감소시키고, 딥러닝 성능을 크게 끌어 올립니다.
- 기본적으로 텐서플로는 모델을 만들 때 합리적인 값으로 w 와 b 를 초기화 합니다.
- 그리고, 문제에 맞도록 기본값을 바꿀 수 있습니다.

Bias 초기화

```
[3] b_init = Zeros()
```

가중치 초기화 - 표준 정규 분포

```
[4] w_init = RandomNormal(stddev=1.0)
```

가중치 초기화 - 세이버이어 글로럿 분포

```
[5] # w_init = glorot_normal()
```

가중치 초기화 - 세이버이어 글로럿 균등 분포

```
[6] # w_init = glorot_uniform()
```

가중치 초기화

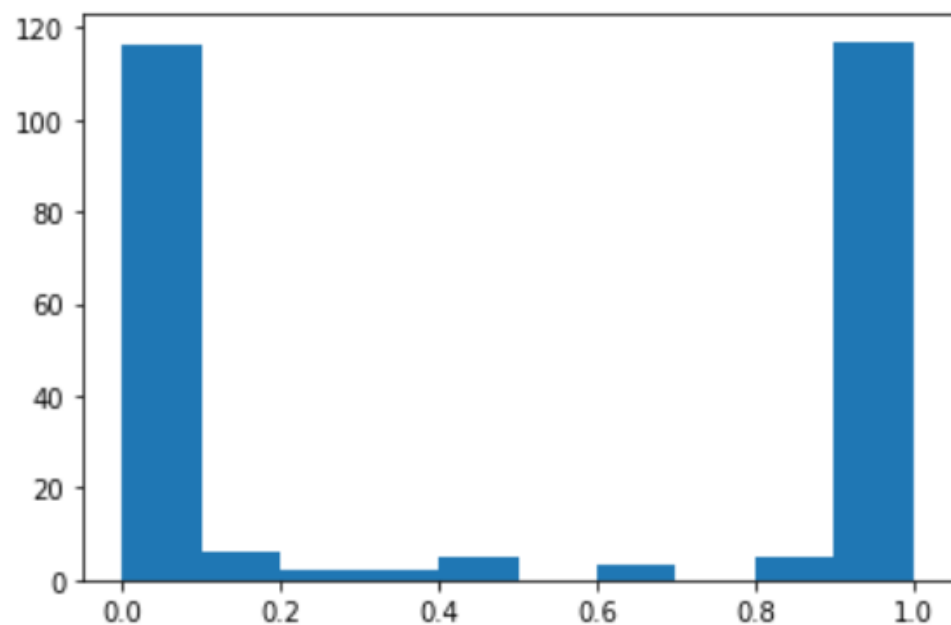
신경망 모델

```
[7] model = Sequential()  
    model.add(Dense(n_dense,  
                    input_dim=n_input,  
                    kernel_initializer=w_init,  
                    bias_initializer=b_init))  
  
    # model.add(Activation('sigmoid'))  
    # model.add(Activation('tanh'))  
    model.add(Activation('relu'))
```

가중치 초기화

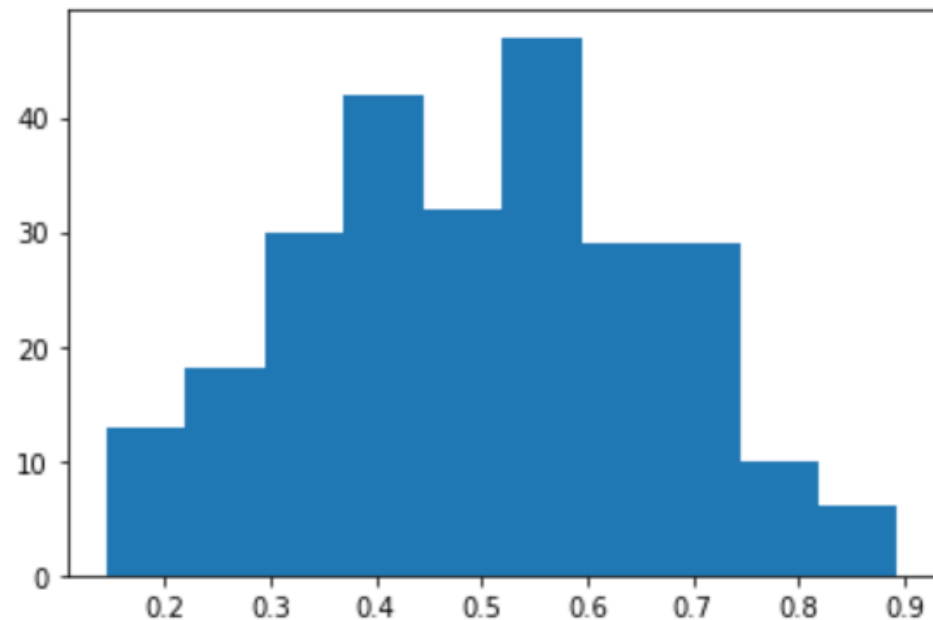
가중치 초기화 - 표준 정규 분포

```
[4] w_init = RandomNormal(stddev=1.0)
```



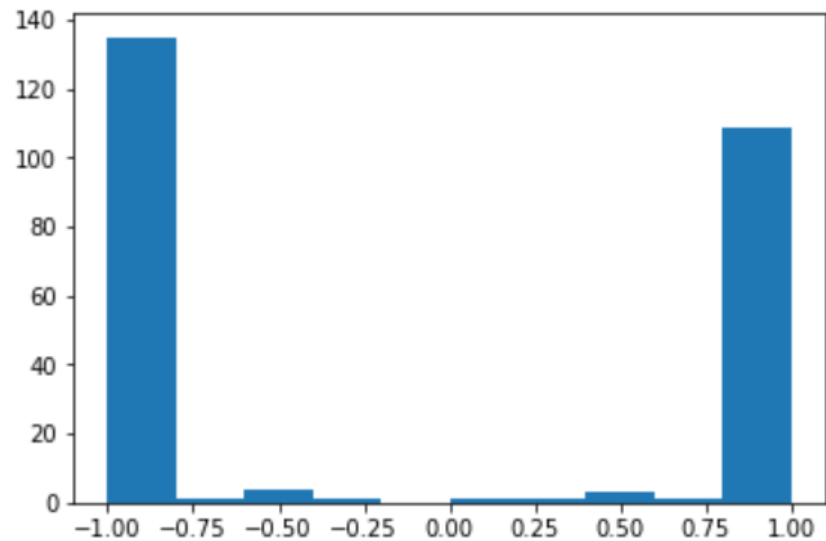
가중치 초기화 - 세이버어 글로럿 분포

```
[5] w_init = glorot_normal()
```

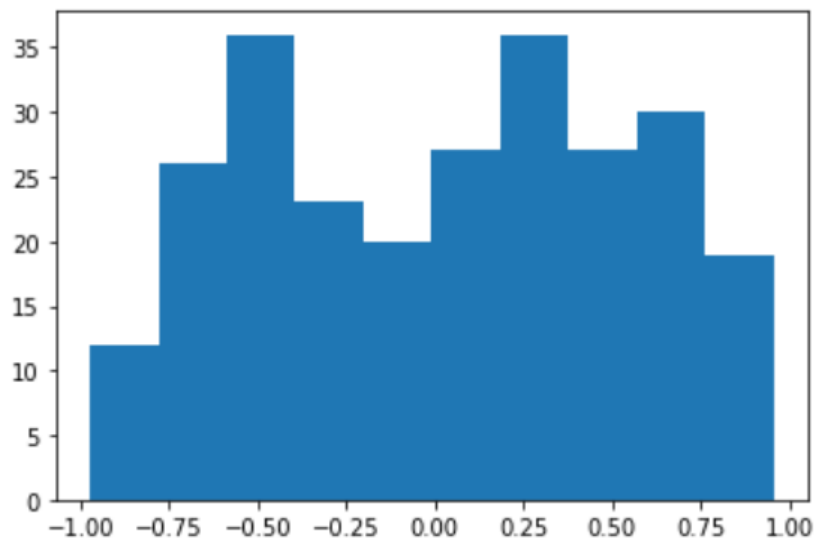


가중치 초기화

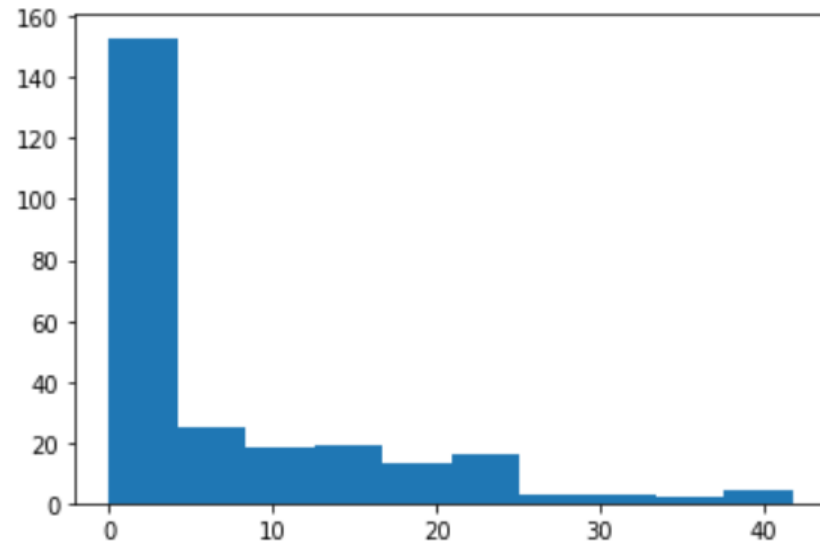
tanh + 표준 정규분포 초기화



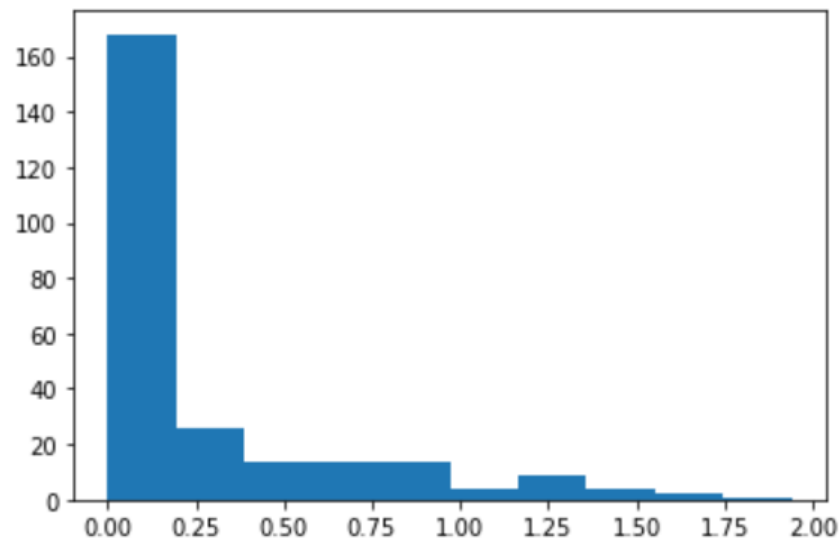
tanh + 글로렛 초기화



ReLU+ 표준 정규분포 초기화

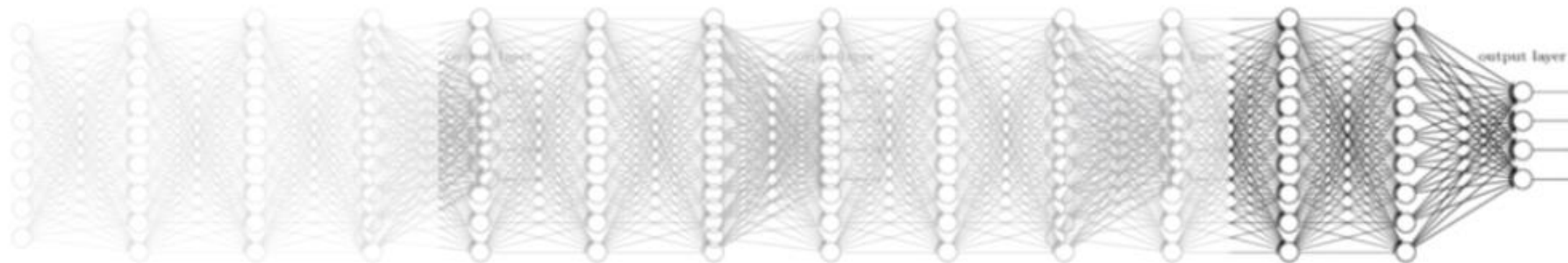


ReLU+ 글로렛 초기화



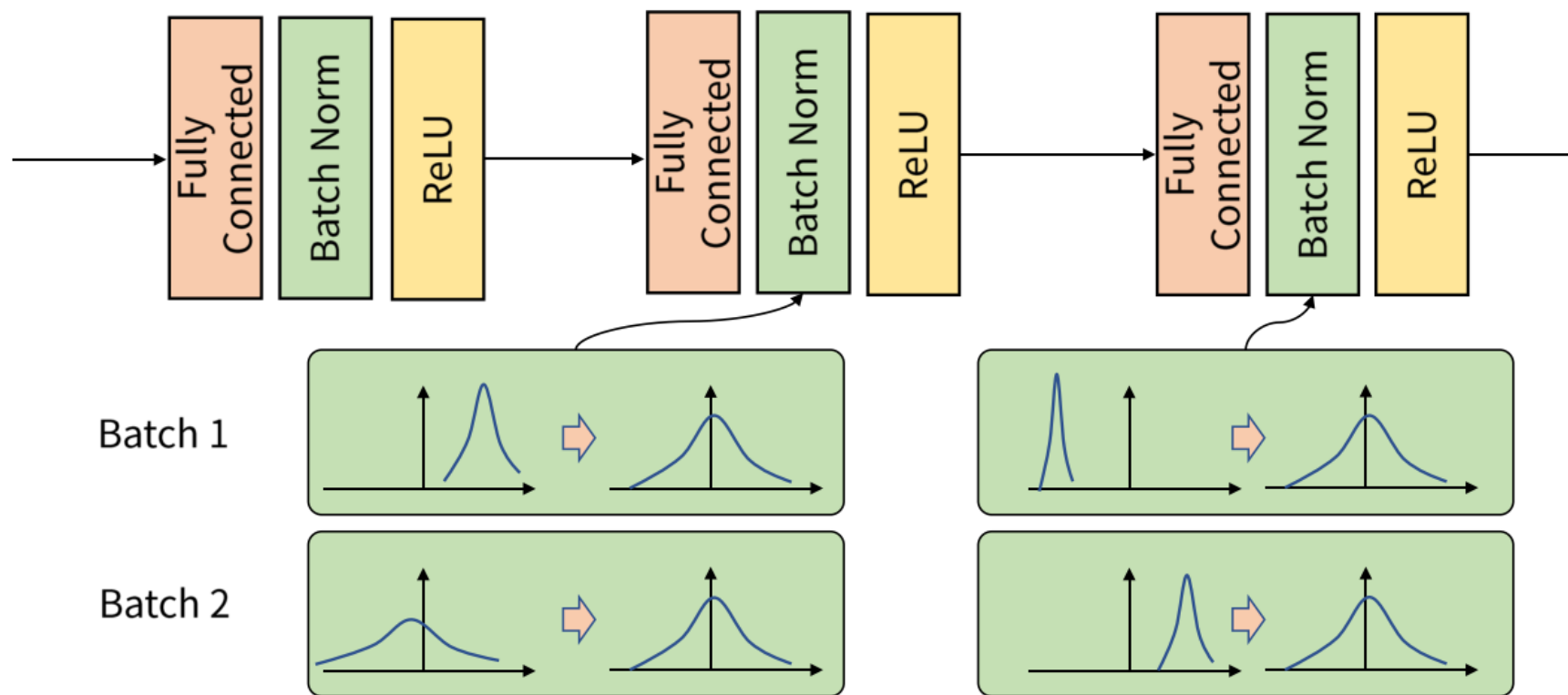
그레이디언트 소실 (Gradient Vanishing)

- 역전파가 출력층에서 입력층으로 작동하면서 손실이 최소화 되도록 신경망의 파라미터를 조정합니다.
- 각 파라미터는 손실에 대한 그레이디언트에 비례하여 조정 됩니다.
- 마지막 은닉층에서 첫 번째 은닉층으로 갈수록 손실에 비례한 파라미터의 그레이디언트가 소멸됩니다.
- 많은 은닉층을 신경망을 추가하면, 그레이디언트 소실 문제 때문에 멀리 떨어진 은닉층이 일정량 만큼 학습할 수 없습니다.
- X에서 y로 근사하는 신경망의 전체 능력이 훼손 됩니다.



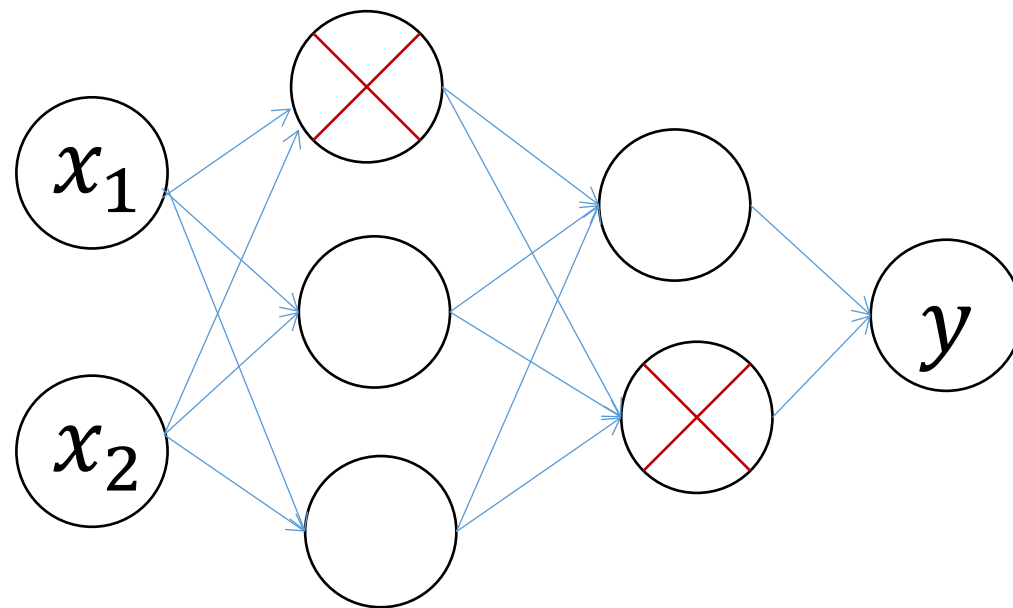
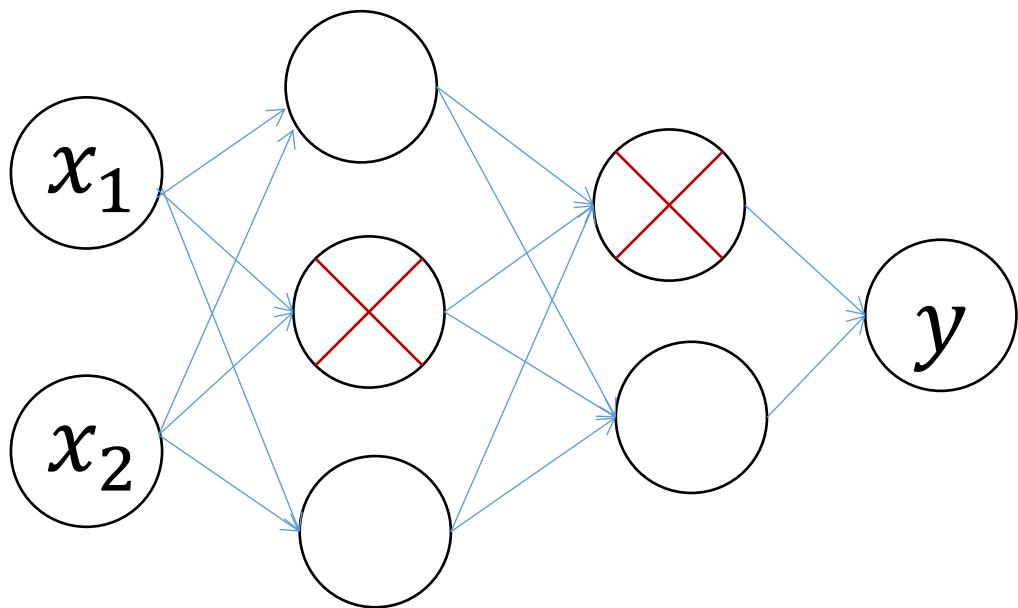
배치 정규화(Batch Normalization)

- 신경망이 훈련하는 동안 층의 파라미터 분포는 점진적으로 이동합니다.
- 이를 내부 공변량 변화(Internal Covariate Shift)라고 합니다.
- 배치 정규화를 통해 값의 분포를 평균이 0이고 표준 편차가 1인 분포로 다시 조정됩니다.
- 배치 정규화는 층간 독립적 학습, 더 높은 학습률 선택, 일반화가 잘 되게 하는 규제와 같은 긍정적인 효과가 있습니다.



과대적합(Overfitting) 방지 - Dropout

- 훈련 손실은 계속 내려가지만 검증 손실이 상승하는 상황을 과대적합(Overfitting) 이라고 합니다.
- 머신러닝 분야에서는 과대적합을 줄이기 위해 L1규제(Lasso), L2규제(Ridge)를 사용합니다.
- 딥러닝에서는 신경망에 특화된 드롭아웃 기법을 사용합니다.
- 드롭아웃(Dropout)은 각 층에 있는 랜덤한 일부 뉴런이 훈련하는 동안 없는 것처럼 동작하는 것입니다.
- 드롭아웃은 훈련 데이터셋의 일부 특성이 과도하게 신경망의 특정 정방향 계산을 주도하지 못하도록 만듭니다. 데이터의 특정 특성에 과도하게 의존하지 않게 됩니다.



과대적합(Overfitting) 방지 - Data Augmentation

- 훈련 데이터셋이 많을수록 모델의 일반화 성능이 높아집니다.
- Image Augmentation은 딥러닝 모델을 훈련하기 위해 새로운 이미지를 생성하는 프로세스입니다.
- 이미지 회전, 블러 처리, 이미지 이동, 노이즈 추가 등의 여러가지 변환으로 훈련 데이터를 생성합니다.



고급 옵티마이저

■ Momentum

- local minimum 에 빠지게 않도록 이전 gradient들을 계산에 포함해서 현재 파라미터를 업데이트 합니다.
- 이전 gradient들을 모두 동일한 비율로 포함시키지는 않고 비율을 감소시켜 줍니다.

■ Adagrad

- 가중치 기울기의 제곱을 통해 학습률을 점차 줄여나가면서 학습이 진행될수록 세밀한 학습이 가능하도록 하는 Optimizer입니다.

■ RMSProp

- RMSProp은 Hyper Parameter ρ 를 추가하여 누적 가중치 기울기는 $(0,1)$ 사이의 값이므로 시간이 지날 수록 점차 작아지고, 최신 가중치 기울기를 더욱 반영하여 AdamGrad에서 발생하는 $G(t)$ 값이 무한히 커져 minimum과 같은 극점 근처에서 학습 속도가 느려지고, local minimum에 수렴하는 문제를 해결하였다.

■ Adam (Adaptive moment estimation)

- Adam은 Momentum과 RMSProp을 병합한 Optimizer입니다.
- RMSProp의 특징인 gradient의 제곱을 지수 평균한 값을 사용하며 Momentum의 특징으로 gradient를 제공하지 않은 값을 사용하여 지수 평균을 구하고 수식에 활용합니다.

MNIST 분류기 소스코드 - DNN

```
[31] model = Sequential()

model.add(Dense(64, activation='relu', input_shape=(784,)))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))
```

TensorBoard 로깅 디렉토리 설정

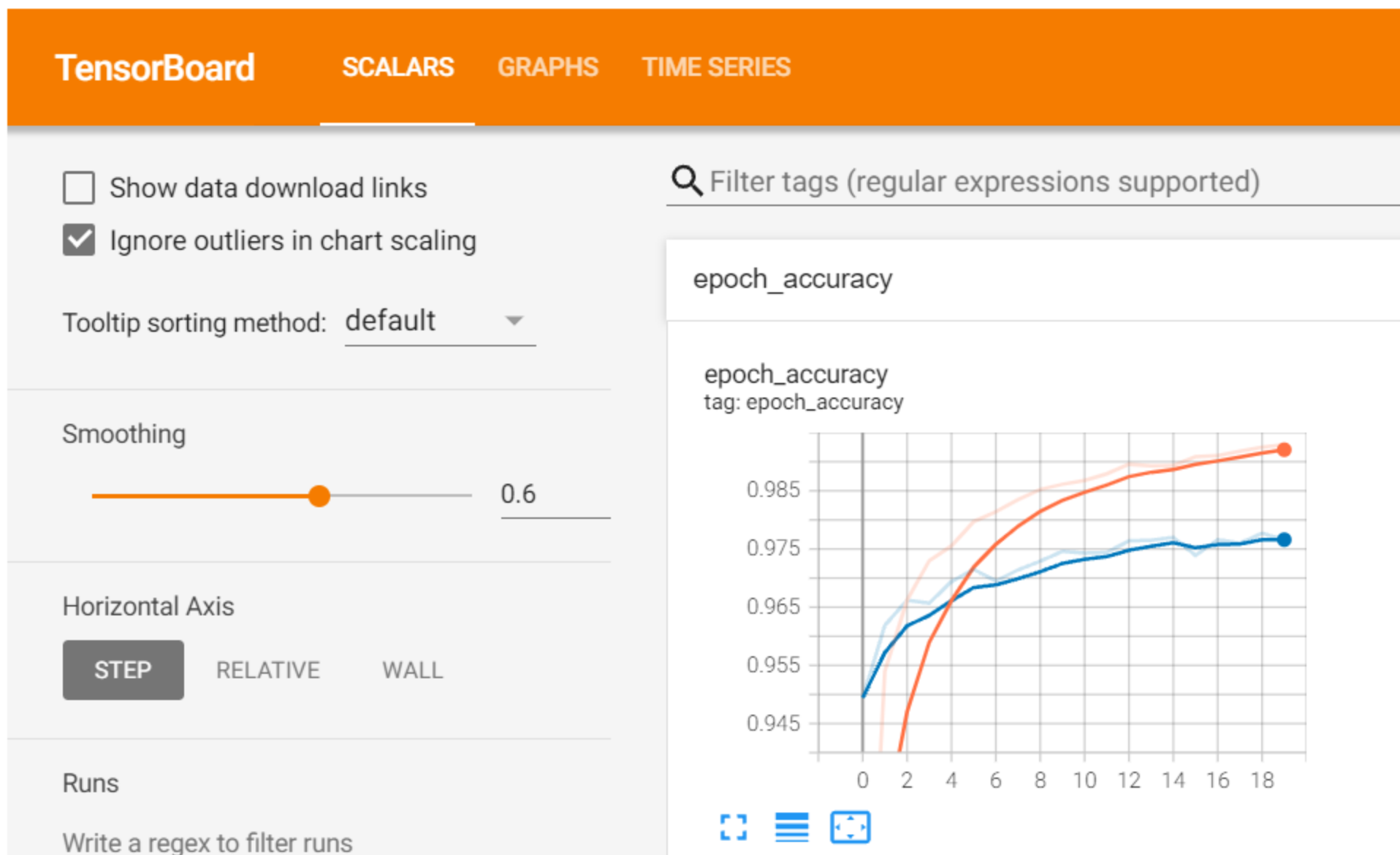
```
[34] tensorboard = TensorBoard('logs/deep-net')
```

모델 훈련

```
[35] model.fit(X_train, y_train,  
               batch_size=128,  
               epochs=20, verbose=1,  
               validation_data=(X_valid, y_valid),  
               callbacks=[tensorboard])
```

```
[36] %load_ext tensorboard
```

```
[37] %tensorboard --logdir logs/deep-net
```



MNIST 분류모델 구현 실습 - DNN



`mnist_dnn_tensorboard_ann`

Thank you