데이터 전처리, 특성 공학, 특성 학습

학습 내용

- 신경망을 위한 데이터 전처리

01 데이터 전처리의 목적

- 주어진 원본 데이터를 신경망에 적용하기 쉽도록 만드는 것.
  - 벡터화(vectorization), 정규화(normalization), 누락된 값 다루기, 특성 추출

벡터화

- 신경망에서 모든 입력과 타깃은 부동 소수 데이터로 이루어진 텐서여야 한다.
- 사운드, 이미지, 텍스트 등 처리해야 할 것이 무엇이든지 먼저 텐서로 변환해야 한다.
  - 이 단계를 데이터 벡터화(data vectorization)이라고 한다.
  - 이전에 나온 2개의 텍스트 분류 예에서 텍스트를 (단어 시퀀스를 의미하는) 정수 리스트로 변환

값 정규화

- 입력 데이터를 float32타입으로 변환하고 255로 나눠서 최종으로 0~1 사이의 부동 소수 값으로 만듬.
- 일반적으로 비교적 큰 값(예를 들어 네트워크의 가중치 초깃값보다 훨씬 큰 여러 자리수를 가진 정수)이나 균일하지 않은 데이터를 신경망에 주입하는 것은 위험하다.
  - 이렇게 되면 업데이트할 그래디언트가 커져 네트워크가 수렴하는 것을 방해한다.
- 신경망을 쉽게 학습시켜려면 데이터가 다음 특징을 가져야 한다.
  - 작은 값을 취합니다. 대부분의 값이 0~1 사이여야 한다.
  - 균일해야 한다. 즉 모든 특성이 대체로 비슷한 범위를 가져야 한다.
- 기타 도움이 되는 것.
  - 각 특성별로 평균이 0이 되도록 정규화
  - 각 특성별로 표준 편차가 1이 되도록 정규화

02. 과대적합 및 일반화

- **과대적합** : 훈련 데이터에 최적화되고, 일반화 성능이 떨어지는 현상
- 최적화(optimization)는 가능한 훈련 데이터에서 최고의 성능을 얻으려고 모델을 조정하는 과정.
- 일반화(generalization)는 훈련된 모델이 이전에 본 적이 없는 데이터에서 얼마나 잘 수행되는지를 의미

일반화 성능을 향상시키는 방법.

- 01 더 많은 훈련 데이터를 모으기 - 가장 좋은 방법
- 02 과대적합을 피하는 처리 과정 - 규제(regularization)

A. 네트워크 크기 축소

- 과대적합을 막는 가장 단순한 방법은 모델의 크기, 즉 모델에 있는 학습 파라미터의 수를 줄이는 것.

B. 가중치 규제 추가

- 간단한 모델이 복잡한 모델보다 덜 과대적합될 가능성이 높다.
- 네트워크의 복잡도에 제한을 두어 가중치가 작은 값을 가지도록 강제하는 것.
  - 가중치 규제(weight regularization)이라한다.
  - L1규제 : 가중치의 절대값에 비례하는 비용이 추가(가중치의 L1노름(norm))
    - 불필요한 가중치의 수치를 0으로 적용.
  - L2규제 : 가중치의 제곱에 비례하는 비용이 추가(가중치의 L2노름(norm))
    - L2규제는 신경망에서 가중치 감쇠(weight decay)라고 부른다. 가중치 감쇠는 수학적으로 L2규제와 동일

In [1]:

```
from keras.datasets import imdb
import numpy as np

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

def vectorize_sequences(sequences, dimension=10000):
    # 크기가 (len(sequences), dimension))이고 모든 원소가 0인 행렬을 만듭니다
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # results[i]에서 특정 인덱스의 위치를 1로 만듭니다
    return results
```

```
<__array_function__ internals>:5: VisibleDeprecationWarning: Creating an ndarray fro
m ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays w
ith different lengths or shapes) is deprecated. If you meant to do this, you must sp
ecify 'dtype=object' when creating the ndarray
c:₩users₩toto₩anaconda3₩envs₩tf2x₩lib₩site-packages₩tensorflow₩python₩keras₩datasets
₩imdb.py:159: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequ
ences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'
when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
c:₩users₩toto₩anaconda3₩envs₩tf2x₩lib₩site-packages₩tensorflow₩python₩keras₩datasets
₩imdb.py:160: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequ
ences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object'
when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

훈련데이터와 테스트 데이터를 벡터로 변환

In [2]:

```
# 훈련 데이터를 벡터로 변환합니다
X_train = vectorize_sequences(train_data)
# 테스트 데이터를 벡터로 변환합니다
X_test = vectorize_sequences(test_data)
```

In [3]:

```
# 레이블을 벡터로 변환합니다
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [4]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(25000, 10000) (25000, 10000) (25000,) (25000,)
```

적용 예

In [5]:

```
from keras import models
from keras import layers
```

In [6]:

```
original_model = models.Sequential()
original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
original_model.add(layers.Dense(16, activation='relu'))
original_model.add(layers.Dense(1, activation='sigmoid'))

original_model.compile(optimizer='rmsprop',
                       loss='binary_crossentropy',
                       metrics=['acc'])
```

In [7]:

```
original_hist = original_model.fit(X_train, y_train,
                                   epochs=20,
                                   batch_size=512,
                                   validation_data=(X_test, y_test))
```

```
Epoch 1/20
49/49 [==============================] - 6s 98ms/step - loss: 0.5801 - acc: 0.7308 -
val_loss: 0.3648 - val_acc: 0.8754
Epoch 2/20
49/49 [==============================] - 1s 28ms/step - loss: 0.2959 - acc: 0.9063 -
val_loss: 0.2929 - val_acc: 0.8870
Epoch 3/20
49/49 [==============================] - 1s 29ms/step - loss: 0.2153 - acc: 0.9257 -
val_loss: 0.2790 - val_acc: 0.8883
Epoch 4/20
49/49 [==============================] - 1s 28ms/step - loss: 0.1695 - acc: 0.9430 -
val_loss: 0.2897 - val_acc: 0.8841
Epoch 5/20
49/49 [==============================] - 1s 28ms/step - loss: 0.1418 - acc: 0.9530 -
val_loss: 0.3096 - val_acc: 0.8789
Epoch 6/20
49/49 [==============================] - 1s 27ms/step - loss: 0.1224 - acc: 0.9610 -
val_loss: 0.3244 - val_acc: 0.8768
Epoch 7/20
49/49 [==============================] - 1s 27ms/step - loss: 0.1037 - acc: 0.9660 -
val_loss: 0.3484 - val_acc: 0.8743
Epoch 8/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0887 - acc: 0.9736 -
val_loss: 0.3695 - val_acc: 0.8723
Epoch 9/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0719 - acc: 0.9794 -
val_loss: 0.4047 - val_acc: 0.8668
Epoch 10/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0604 - acc: 0.9838 -
val_loss: 0.4524 - val_acc: 0.8587
Epoch 11/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0490 - acc: 0.9882 -
val_loss: 0.4586 - val_acc: 0.8634
Epoch 12/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0391 - acc: 0.9905 -
val_loss: 0.4963 - val_acc: 0.8603
Epoch 13/20
49/49 [==============================] - 1s 27ms/step - loss: 0.0329 - acc: 0.9928 -
val_loss: 0.6106 - val_acc: 0.8442
Epoch 14/20
49/49 [==============================] - 1s 28ms/step - loss: 0.0285 - acc: 0.9930 -
val_loss: 0.5977 - val_acc: 0.8510
Epoch 15/20
49/49 [==============================] - 1s 27ms/step - loss: 0.0201 - acc: 0.9962 -
val_loss: 0.6059 - val_acc: 0.8569
Epoch 16/20
49/49 [==============================] - 1s 27ms/step - loss: 0.0158 - acc: 0.9969 -
val_loss: 0.6431 - val_acc: 0.8551
Epoch 17/20
49/49 [==============================] - 1s 24ms/step - loss: 0.0105 - acc: 0.9984 -
val_loss: 0.7203 - val_acc: 0.8526
Epoch 18/20
49/49 [==============================] - 2s 36ms/step - loss: 0.0100 - acc: 0.9976 -
```

```
val_loss: 0.7456 - val_acc: 0.8524
Epoch 19/20
49/49 [==============================] - 1s 31ms/step - loss: 0.0066 - acc: 0.9989 -
val_loss: 0.7843 - val_acc: 0.8522
Epoch 20/20
49/49 [==============================] - 1s 30ms/step - loss: 0.0047 - acc: 0.9995 -
val_loss: 0.8262 - val_acc: 0.8526
```

In [23]:

```
original_val_loss = original_hist.history['val_loss']
```

In [32]:

```
from keras import regularizers

l2_model = models.Sequential()
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu', input_shape=(10000,)))
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu'))
l2_model.add(layers.Dense(1, activation='sigmoid'))
```

- l2(0.001)는 가중치 행렬의 모든 원소를 제곱하고 0.001을 곱하여 네트워크의 전체 손실에 더해진다는 의미입니다.
- 이 페널티 항은 훈련할 때만 추가

In [33]:

```
l2_model.compile(optimizer='rmsprop',
                 loss='binary_crossentropy',
                 metrics=['acc'])
```

In [34]:

```
l2_model_hist = l2_model.fit(X_train, y_train,
                             epochs=20,
                             batch_size=512,
                             validation_data=(X_test, y_test))
```

```
Epoch 1/20
49/49 [==============================] - 4s 54ms/step - loss: 0.5663 - acc: 0.7534 -
val_loss: 0.4059 - val_acc: 0.8522
Epoch 2/20
49/49 [==============================] - 1s 30ms/step - loss: 0.3117 - acc: 0.9081 -
val_loss: 0.3338 - val_acc: 0.8885
Epoch 3/20
49/49 [==============================] - 1s 30ms/step - loss: 0.2650 - acc: 0.9243 -
val_loss: 0.3633 - val_acc: 0.8736
Epoch 4/20
49/49 [==============================] - 2s 31ms/step - loss: 0.2418 - acc: 0.9325 -
val_loss: 0.3537 - val_acc: 0.8807
Epoch 5/20
49/49 [==============================] - 1s 31ms/step - loss: 0.2249 - acc: 0.9407 -
val_loss: 0.3971 - val_acc: 0.8632
Epoch 6/20
49/49 [==============================] - 2s 31ms/step - loss: 0.2240 - acc: 0.9428 -
val_loss: 0.3554 - val_acc: 0.8812
Epoch 7/20
49/49 [==============================] - 2s 31ms/step - loss: 0.2118 - acc: 0.9468 -
val_loss: 0.3636 - val_acc: 0.8791
Epoch 8/20
49/49 [==============================] - 1s 30ms/step - loss: 0.2113 - acc: 0.9463 -
val_loss: 0.3708 - val_acc: 0.8786
Epoch 9/20
49/49 [==============================] - 2s 31ms/step - loss: 0.2054 - acc: 0.9481 -
val_loss: 0.4088 - val_acc: 0.8664
Epoch 10/20
49/49 [==============================] - 1s 31ms/step - loss: 0.2053 - acc: 0.9477 -
val_loss: 0.3881 - val_acc: 0.8737
Epoch 11/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1962 - acc: 0.9500 -
val_loss: 0.3869 - val_acc: 0.8739
Epoch 12/20
49/49 [==============================] - 2s 31ms/step - loss: 0.1963 - acc: 0.9537 -
val_loss: 0.4286 - val_acc: 0.8623
Epoch 13/20
49/49 [==============================] - 2s 31ms/step - loss: 0.2041 - acc: 0.9451 -
val_loss: 0.4040 - val_acc: 0.8716
Epoch 14/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1890 - acc: 0.9536 -
val_loss: 0.4080 - val_acc: 0.8696
Epoch 15/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1820 - acc: 0.9567 -
val_loss: 0.5049 - val_acc: 0.8429
Epoch 16/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1920 - acc: 0.9513 -
val_loss: 0.4084 - val_acc: 0.8710
Epoch 17/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1863 - acc: 0.9538 -
val_loss: 0.4722 - val_acc: 0.8512
Epoch 18/20
49/49 [==============================] - 2s 31ms/step - loss: 0.1840 - acc: 0.9547 -
```

```
val_loss: 0.4184 - val_acc: 0.8689
Epoch 19/20
49/49 [==============================] - 2s 31ms/step - loss: 0.1891 - acc: 0.9521 -
val_loss: 0.4260 - val_acc: 0.8666
Epoch 20/20
49/49 [==============================] - 1s 31ms/step - loss: 0.1826 - acc: 0.9546 -
val_loss: 0.4209 - val_acc: 0.8685
```

In [35]:

```
import matplotlib.pyplot as plt
```

In [36]:

```
epochs = range(1, 21)
```

In [37]:

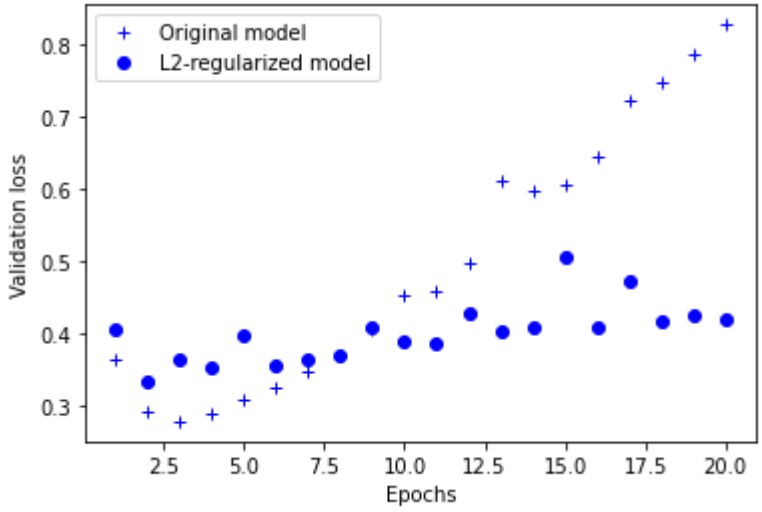
```
original_val_loss
```

Out[37]:

```
[0.3647705614566803,
 0.29290249943733215,
 0.278982013463974,
 0.289738267660141,
 0.30962374806404114,
 0.3244265913963318,
 0.34837329387664795,
 0.36947593092918396,
 0.40472090244293213,
 0.452373206615448,
 0.45863077044487,
 0.49632343649864197,
 0.610617458820343,
 0.5977329611778259,
 0.6058666110038757,
 0.6430662274360657,
 0.7203129529953003,
 0.745570719242096,
 0.7842505574226379,
 0.826181173324585]
```

In [38]:

```
l2_model_val_loss = l2_model_hist.history['val_loss']

plt.plot(epochs, original_val_loss, 'b+', label='Original model')
plt.plot(epochs, l2_model_val_loss, 'bo', label='L2-regularized model')
plt.xlabel('Epochs')
plt.ylabel('Validation loss')
plt.legend()

plt.show()
```



두 모델이 동일한 파라미터 수를 가지고 있더라도 L2규제를 사용한 모델(점)이 기본 모델보다 훨씬 더 과적합을 잘 견디고 있다

콜랩 환경 실행 시간

실습 2-4

- 기존 모델에 %%time을 붙여서 해 보고, 시간확인 후, GPU모델로 변경후, 다시 시간을 확인해 보자.
- CPU 버전 :
  - ori model time : 36.6 s
  - l2 모델 time : 37.4 s
- GPU 버전 :
  - ori model time : 24.9 s
  - l2 모델 time : 23.4 s
- TPU 버전 :
  - ori model time : 40.1 s
  - l2 모델 time : 39.3 s

실습 2-4(추가) EarlyStopping

In [39]:

```
from keras import regularizers

l2_model = models.Sequential()
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu', input_shape=(10000,)))
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu'))
l2_model.add(layers.Dense(1, activation='sigmoid'))


l2_model.compile(optimizer='rmsprop',
                 loss='binary_crossentropy',
                 metrics=['acc'])
```

In [40]:

```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience = 20) # 조기종료 콜백함수 정의
```

In [41]:

```
l2_model_hist_add = l2_model.fit(X_train, y_train,
                                 epochs=200,
                                 batch_size=512,
                                 validation_data=(X_test, y_test),
                                 callbacks=[early_stopping])
```

```
Epoch 1/200
49/49 [==============================] - 6s 61ms/step - loss: 0.5868 - acc: 0.7373 -
val_loss: 0.3718 - val_acc: 0.8805
Epoch 2/200
49/49 [==============================] - 1s 30ms/step - loss: 0.3168 - acc: 0.9057 -
val_loss: 0.3334 - val_acc: 0.8876
Epoch 3/200
49/49 [==============================] - 2s 31ms/step - loss: 0.2633 - acc: 0.9223 -
val_loss: 0.3262 - val_acc: 0.8883
Epoch 4/200
49/49 [==============================] - 1s 31ms/step - loss: 0.2410 - acc: 0.9321 -
val_loss: 0.3600 - val_acc: 0.8735
Epoch 5/200
49/49 [==============================] - 2s 31ms/step - loss: 0.2280 - acc: 0.9376 -
val_loss: 0.3392 - val_acc: 0.8857
Epoch 6/200
49/49 [==============================] - 1s 30ms/step - loss: 0.2206 - acc: 0.9407 -
val_loss: 0.3511 - val_acc: 0.8803
Epoch 7/200
49/49 [==============================] - 1s 30ms/step - loss: 0.2096 - acc: 0.9453 -
val_loss: 0.3539 - val_acc: 0.8804
Epoch 8/200
49/49 [==============================] - 2s 31ms/step - loss: 0.2089 - acc: 0.9453 -
val_loss: 0.3679 - val_acc: 0.8762
Epoch 9/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1985 - acc: 0.9514 -
val_loss: 0.3902 - val_acc: 0.8685
Epoch 10/200
49/49 [==============================] - 1s 31ms/step - loss: 0.1990 - acc: 0.9484 -
val_loss: 0.3742 - val_acc: 0.8742
Epoch 11/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1893 - acc: 0.9540 -
val_loss: 0.3901 - val_acc: 0.8724
Epoch 12/200
49/49 [==============================] - 1s 31ms/step - loss: 0.1992 - acc: 0.9481 -
val_loss: 0.3889 - val_acc: 0.8726
Epoch 13/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1886 - acc: 0.9545 -
val_loss: 0.4279 - val_acc: 0.8624
Epoch 14/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1876 - acc: 0.9547 -
val_loss: 0.3928 - val_acc: 0.8724
Epoch 15/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1815 - acc: 0.9572 -
val_loss: 0.4013 - val_acc: 0.8713
Epoch 16/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1838 - acc: 0.9569 -
val_loss: 0.4344 - val_acc: 0.8644
Epoch 17/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1816 - acc: 0.9561 -
val_loss: 0.4112 - val_acc: 0.8700
Epoch 18/200
```

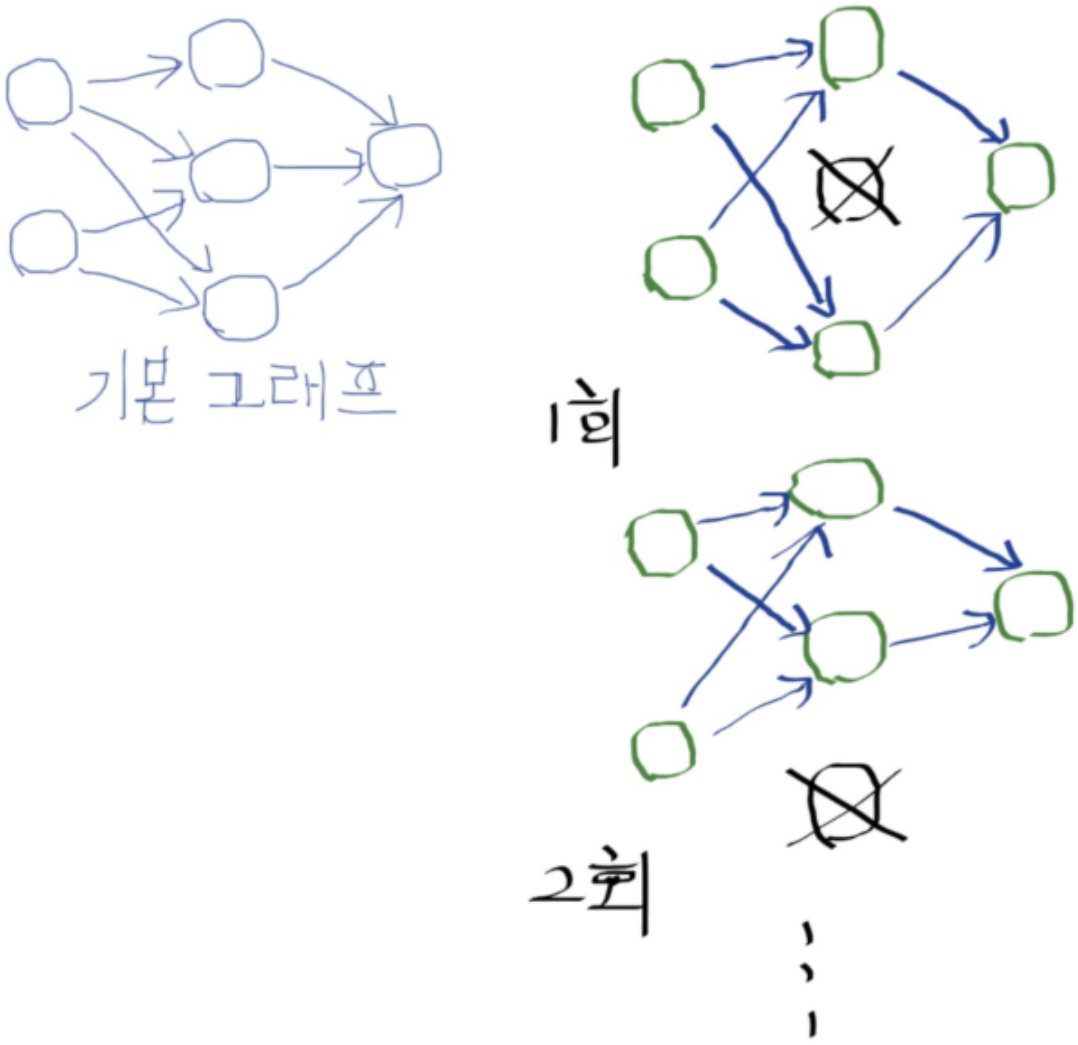
```
49/49 [==============================] - 1s 30ms/step - loss: 0.1678 - acc: 0.9625 -
val_loss: 0.4106 - val_acc: 0.8698
Epoch 19/200
49/49 [==============================] - 1s 29ms/step - loss: 0.1739 - acc: 0.9585 -
val_loss: 0.4251 - val_acc: 0.8654
Epoch 20/200
49/49 [==============================] - 2s 31ms/step - loss: 0.1621 - acc: 0.9655 -
val_loss: 0.4170 - val_acc: 0.8673
Epoch 21/200
49/49 [==============================] - 1s 29ms/step - loss: 0.1615 - acc: 0.9651 -
val_loss: 0.4282 - val_acc: 0.8648
Epoch 22/200
49/49 [==============================] - 1s 29ms/step - loss: 0.1614 - acc: 0.9650 -
val_loss: 0.4235 - val_acc: 0.8678
Epoch 23/200
49/49 [==============================] - 1s 30ms/step - loss: 0.1590 - acc: 0.9666 -
val_loss: 0.4682 - val_acc: 0.8607
```

C. 드롭아웃 추가

- 드롭 아웃은 토론토 대학의 제프린 힌튼과 그의 학생들이 개발.
- 신경망을 위해 사용되는 규제 기법 중에서 가장 효과적이고 널리 사용되는 방법중 하나.
- 네트워크의 층에 드롭아웃을 적용하면 훈련하는 동안 무작위로 층의 일부 출력 특성을 제외

In [42]:

```
from IPython.display import display, Image
display(Image(filename="img/dropout1.png"))
```



In [43]:

```
dpt_model = models.Sequential()
dpt_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(16, activation='relu'))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(1, activation='sigmoid'))

dpt_model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['acc'])
```

In [48]:

```
epochs = range(1, 21)
```

In [49]:

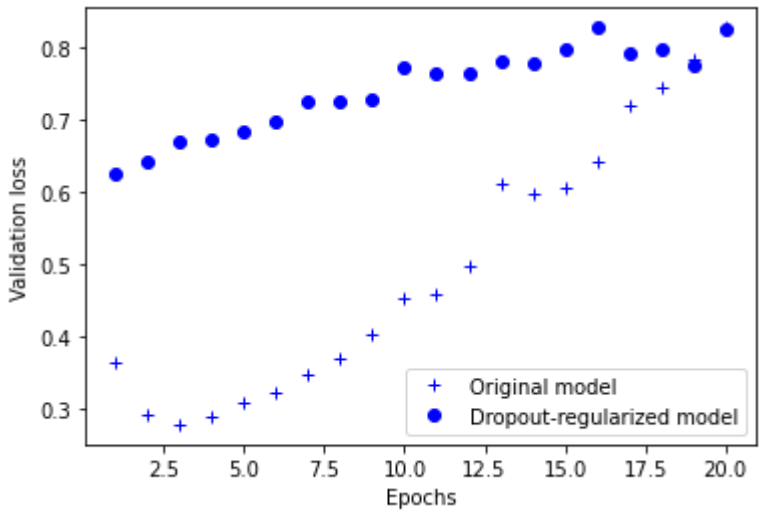
```
dpt_model_hist = dpt_model.fit(X_train, y_train,
                               epochs=20,
                               batch_size=512,
                               validation_data=(X_test, y_test))
```

```
Epoch 1/20
49/49 [==============================] - 3s 55ms/step - loss: 0.1023 - acc: 0.9699 -
val_loss: 0.6252 - val_acc: 0.8681
Epoch 2/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0988 - acc: 0.9702 -
val_loss: 0.6424 - val_acc: 0.8717
Epoch 3/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0924 - acc: 0.9706 -
val_loss: 0.6692 - val_acc: 0.8718
Epoch 4/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0939 - acc: 0.9714 -
val_loss: 0.6708 - val_acc: 0.8688
Epoch 5/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0978 - acc: 0.9703 -
val_loss: 0.6826 - val_acc: 0.8678
Epoch 6/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0951 - acc: 0.9718 -
val_loss: 0.6970 - val_acc: 0.8679
Epoch 7/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0983 - acc: 0.9718 -
val_loss: 0.7239 - val_acc: 0.8699
Epoch 8/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0899 - acc: 0.9724 -
val_loss: 0.7261 - val_acc: 0.8666
Epoch 9/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0977 - acc: 0.9719 -
val_loss: 0.7266 - val_acc: 0.8665
Epoch 10/20
49/49 [==============================] - 2s 33ms/step - loss: 0.0950 - acc: 0.9722 -
val_loss: 0.7718 - val_acc: 0.8706
Epoch 11/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0967 - acc: 0.9703 -
val_loss: 0.7624 - val_acc: 0.8670
Epoch 12/20
49/49 [==============================] - 2s 32ms/step - loss: 0.1034 - acc: 0.9723 -
val_loss: 0.7630 - val_acc: 0.8650
Epoch 13/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0905 - acc: 0.9720 -
val_loss: 0.7795 - val_acc: 0.8655
Epoch 14/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0953 - acc: 0.9713 -
val_loss: 0.7762 - val_acc: 0.8638
Epoch 15/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0960 - acc: 0.9715 -
val_loss: 0.7964 - val_acc: 0.8648
Epoch 16/20
49/49 [==============================] - 2s 31ms/step - loss: 0.1000 - acc: 0.9729 -
val_loss: 0.8267 - val_acc: 0.8655
Epoch 17/20
49/49 [==============================] - 2s 32ms/step - loss: 0.0942 - acc: 0.9728 -
val_loss: 0.7924 - val_acc: 0.8593
Epoch 18/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0953 - acc: 0.9736 -
```

```
val_loss: 0.7974 - val_acc: 0.8624
Epoch 19/20
49/49 [==============================] - 2s 32ms/step - loss: 0.1011 - acc: 0.9701 -
val_loss: 0.7757 - val_acc: 0.8569
Epoch 20/20
49/49 [==============================] - 2s 31ms/step - loss: 0.0946 - acc: 0.9726 -
val_loss: 0.8251 - val_acc: 0.8636
```

In [50]:

```
dpt_model_val_loss = dpt_model_hist.history['val_loss']

plt.plot(epochs, original_val_loss, 'b+', label='Original model')
plt.plot(epochs, dpt_model_val_loss, 'bo', label='Dropout-regularized model')
plt.xlabel('Epochs')
plt.ylabel('Validation loss')
plt.legend()

plt.show()
```

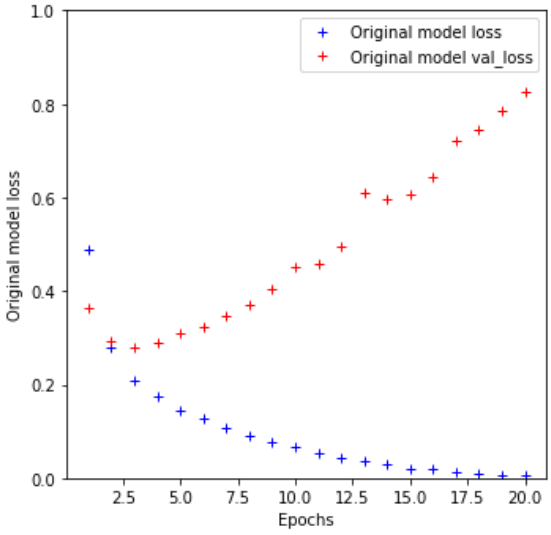


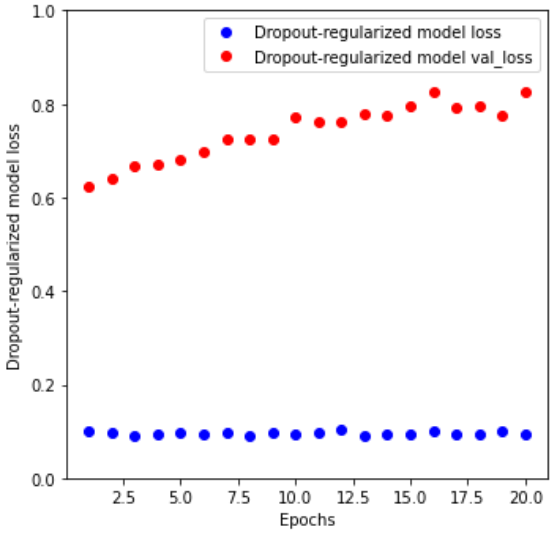
In [51]:

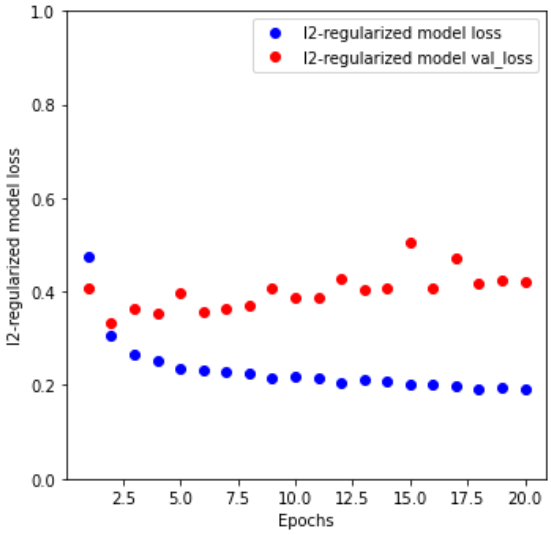
```
ori_model_tr_loss = original_hist.history['loss']
dpt_model_tr_loss = dpt_model_hist.history['loss']
l2_model_tr_loss = l2_model_hist.history['loss']
```

In [52]:

```
plt.figure(figsize=(12,12))

plt.subplot(2,2,1)
plt.plot(epochs, ori_model_tr_loss, 'b+', label='Original model loss')
plt.plot(epochs, original_val_loss, 'r+', label='Original model val_loss')
plt.xlabel('Epochs')
plt.ylabel('Original model loss')
plt.ylim(0,1)
plt.legend()

plt.subplot(2,2,2)
plt.plot(epochs, dpt_model_tr_loss, 'bo', label='Dropout-regularized model loss')
plt.plot(epochs, dpt_model_val_loss, 'ro', label='Dropout-regularized model val_loss')
plt.xlabel('Epochs')
plt.ylabel('Dropout-regularized model loss')
plt.ylim(0,1)
plt.legend()

plt.subplot(2,2,3)
plt.plot(epochs, l2_model_tr_loss, 'bo', label='l2-regularized model loss')
plt.plot(epochs, l2_model_val_loss, 'ro', label='l2-regularized model val_loss')
plt.xlabel('Epochs')
plt.ylabel('l2-regularized model loss')
plt.ylim(0,1)
plt.legend()

plt.show()
```







기본 네트워크 보다 성능이 확실히 향상

03 Summary

과대적합을 방지하기 위해 가장 널리 사용되는 방법.

- 훈련 데이터를 더 모은다.
- 네트워크 용량을 감소시킨다.
- 가중치 규제를 추가합니다.
- 드롭 아웃을 추가합니다.