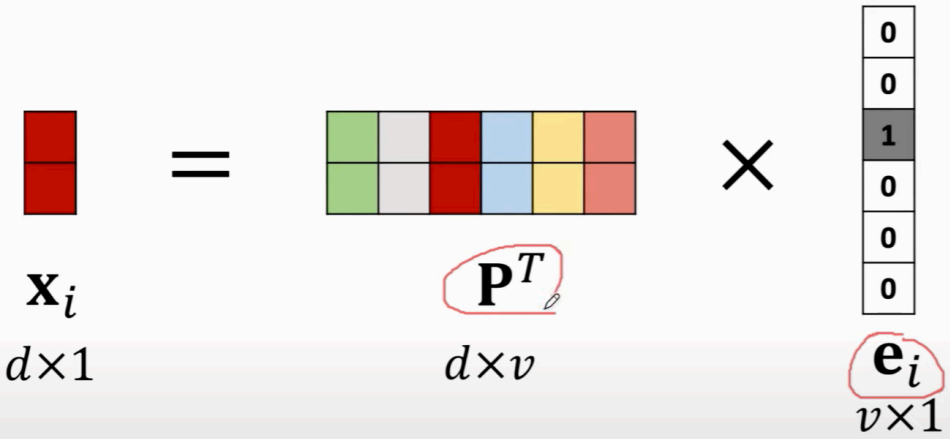
Word Embedding

- Second, map the one-hot vectors to low-dimensional vectors by



$\mathbf{x}_i$ $=$ $\mathbf{P}^T$ $\times$ $\mathbf{e}_i$

$d \times 1$ $\quad$ $d \times v$ $\quad$ $v \times 1$

- $\mathbf{P}$ is parameter matrix which can be learned from training data.
- $\mathbf{e}_i$ is the one-hot vector of the i-th word in dictionary.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

embedding_dim = 8

model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
```

$v = 10K$ $d = 8$ $= 20$

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 8)             80000
```

word_num

embedding_dim

= vocabulary ×embedding_dim

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding

embedding_dim = 8

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 8)             80000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 161
=================================================================
Total params: 80,161
Trainable params: 80,161
Non-trainable params: 0
```

```
from keras import optimizers

epochs = 50

model.compile(optimizer=optimizers.RMSprop(lr=0.0001),
              loss='binary_crossentropy', metrics=['acc'])
```

```
from keras import optimizers

epochs = 50

model.compile(optimizer=optimizers.RMSprop(lr=0.0001),
              loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs,
                    batch_size=32, validation_data=(x_valid, y_valid))
```

- The training set is randomly split to a training set and a validation set.
- 80% for training and 20% for validation.

- x_train: $20{,}000 \times 20$ matrix
- X_valid: $5{,}000 \times 20$ matrix

Performance on test set

```
loss_and_acc = model.evaluate(x_test, labels_test)
print('loss = ' + str(loss_and_acc[0]))
print('acc = ' + str(loss_and_acc[1]))
```

```
25000/25000 [==============================] - 0s 18us/step
loss = 0.5025235502243042
acc = 0.74928
```

- About 75% accuracy on the test set.
- Not bad, because we use only the last 20 words in each movie review. (`word_num=20`)

Logistic Regression for Sentiment Analysis