Neural Networks

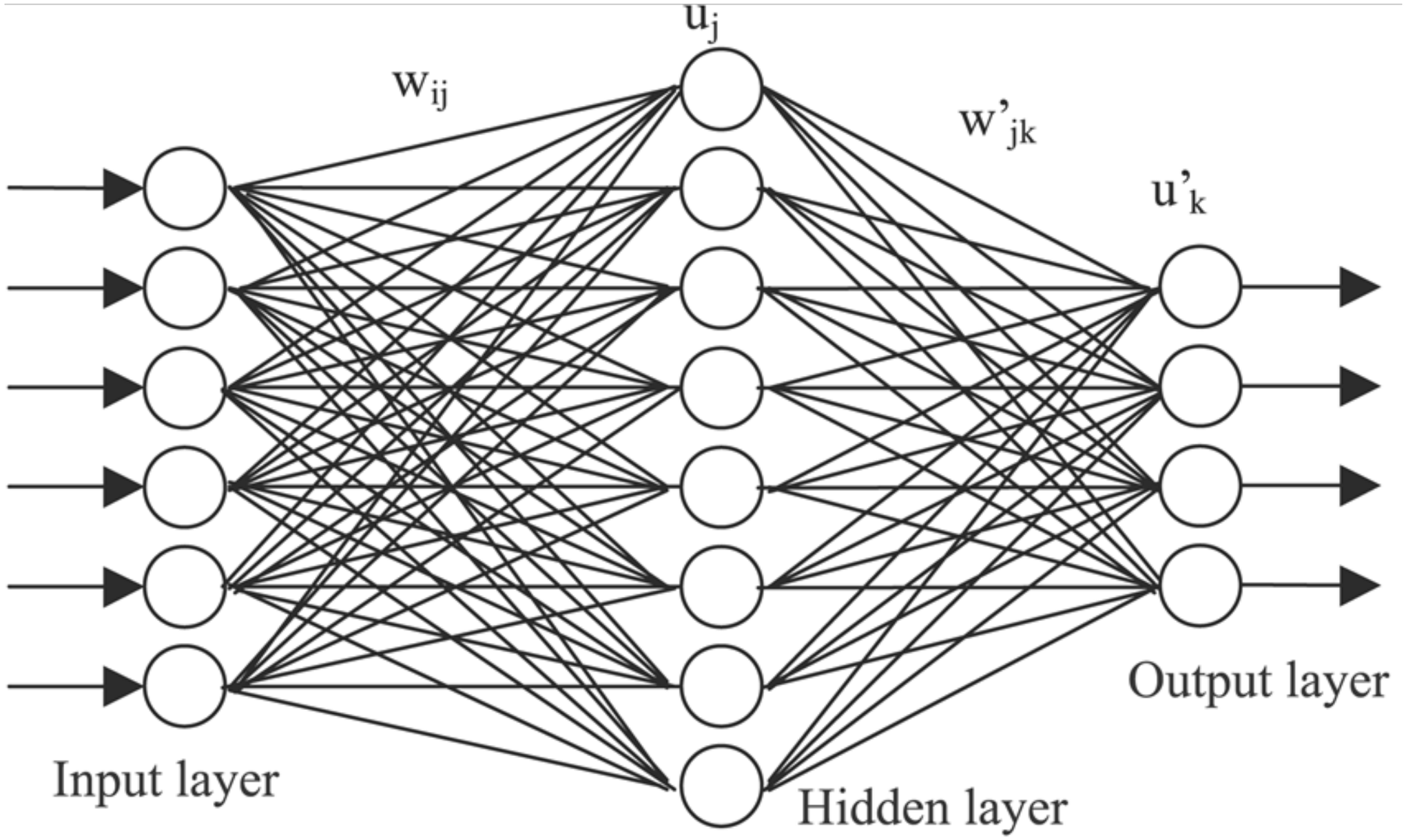
Feed-forward Neural Network

Feed-forward Neural Network

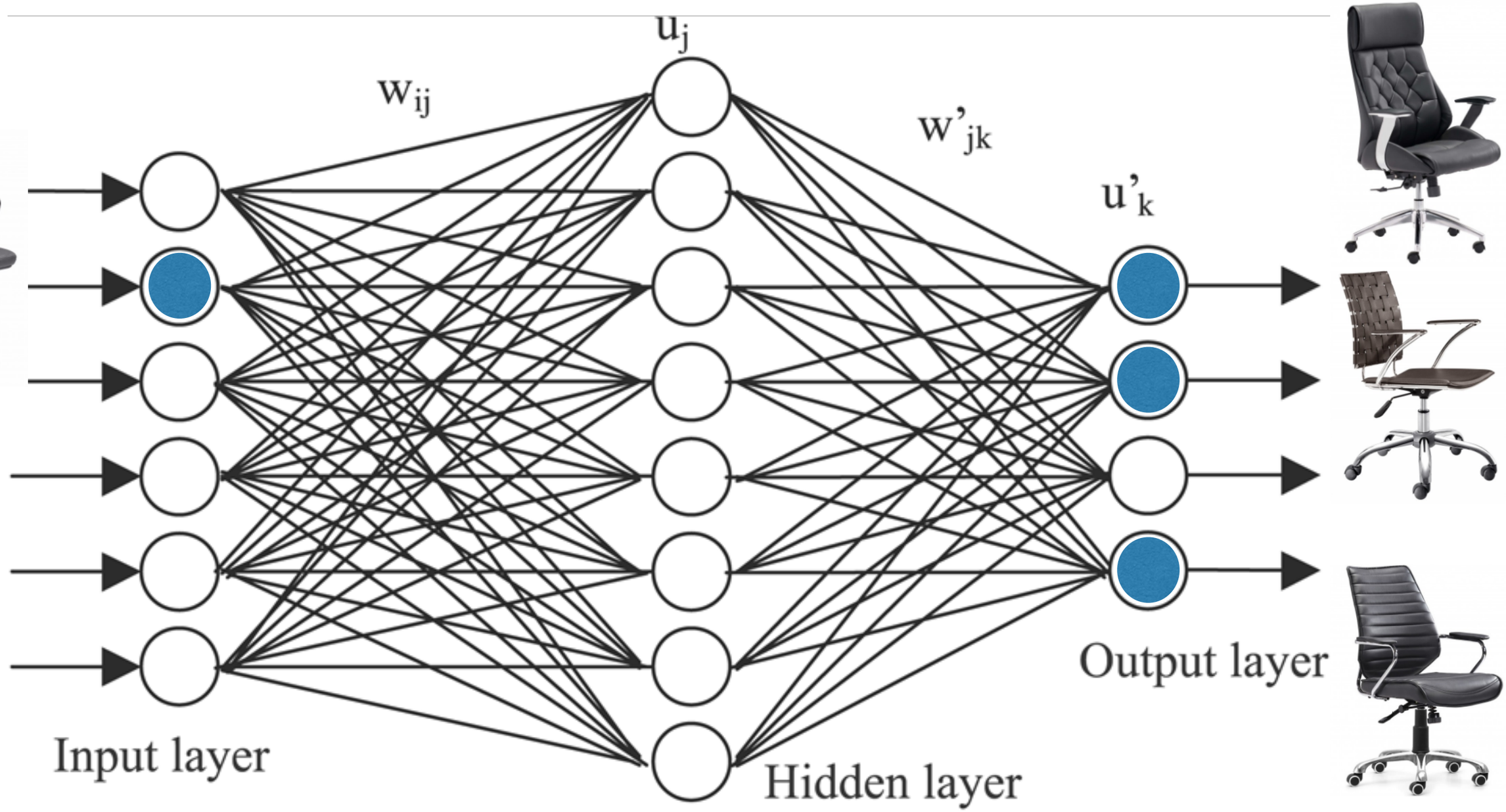
HIGH FASHION HOME

- High Fashion Home browse data
- Divided into product groups per customer
- Subdivided at browsing pauses > 30 min

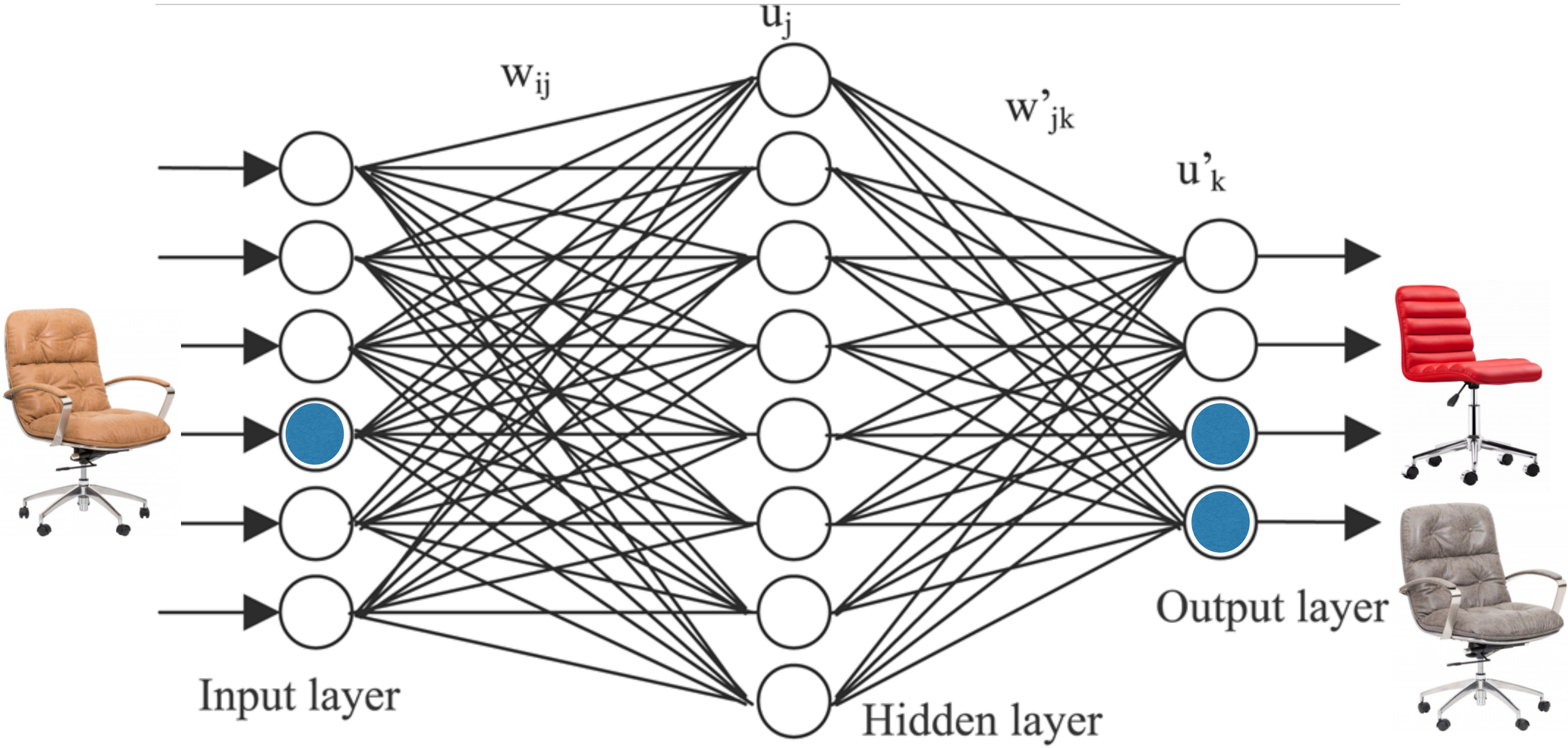
Feed-forward Neural Network



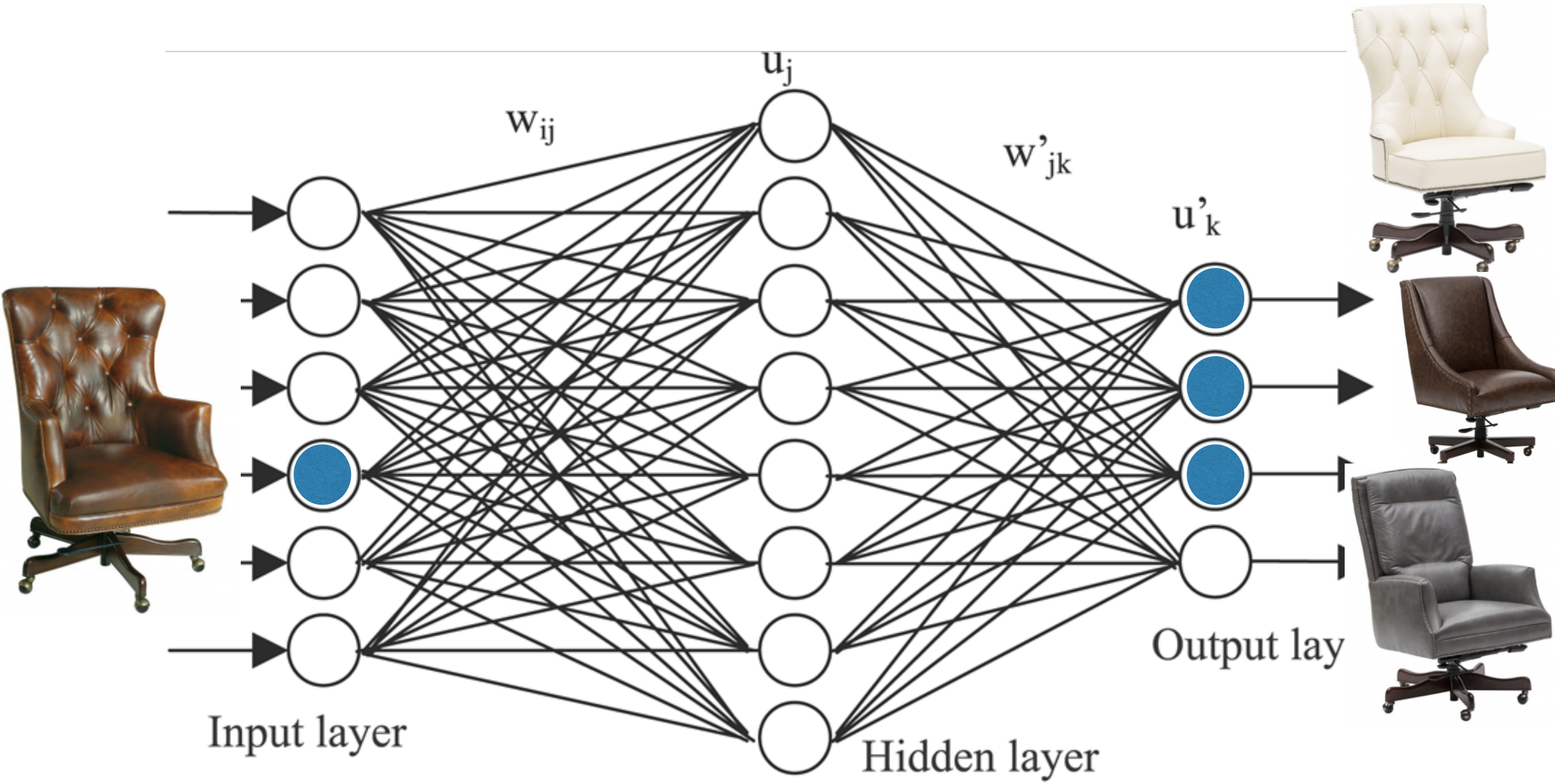
Feed-forward Neural Network



Feed-forward Neural Network



Feed-forward Neural Network




python
Keras

Keras

- machine learning python package
- API interface for other libraries:
  — Tensorflow, CNTK, Theano


python™
Keras
using
TensorFlow

python™

Keras

TensorFlow

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
```

```
model.add(Dense(2 * numProducts,
                input_dim=numProducts,
                activation='relu'))

model.add(Dense(numProducts,
                activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

python™

Keras

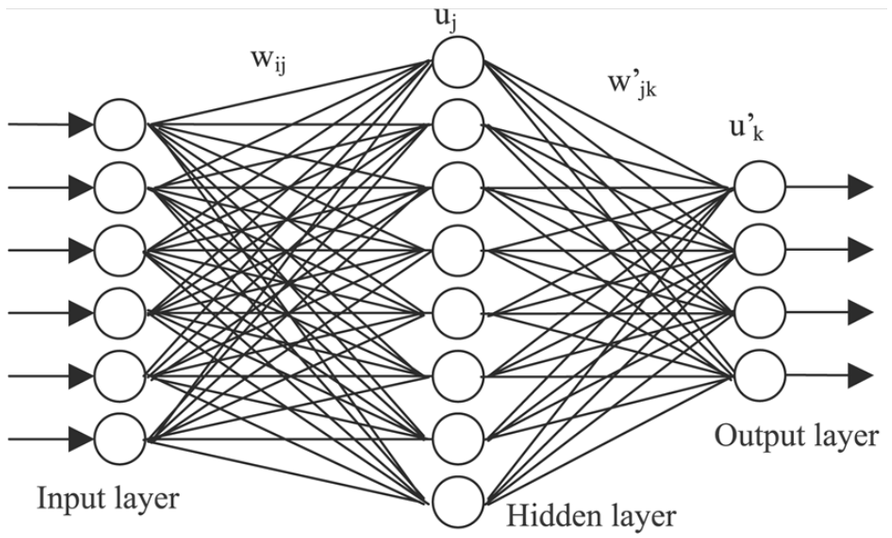
TensorFlow

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
```

```
model.add(Dense(2 * numProducts,
                input_dim=numProducts,
                activation='relu'))

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




```
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Keras

python™

TensorFlow

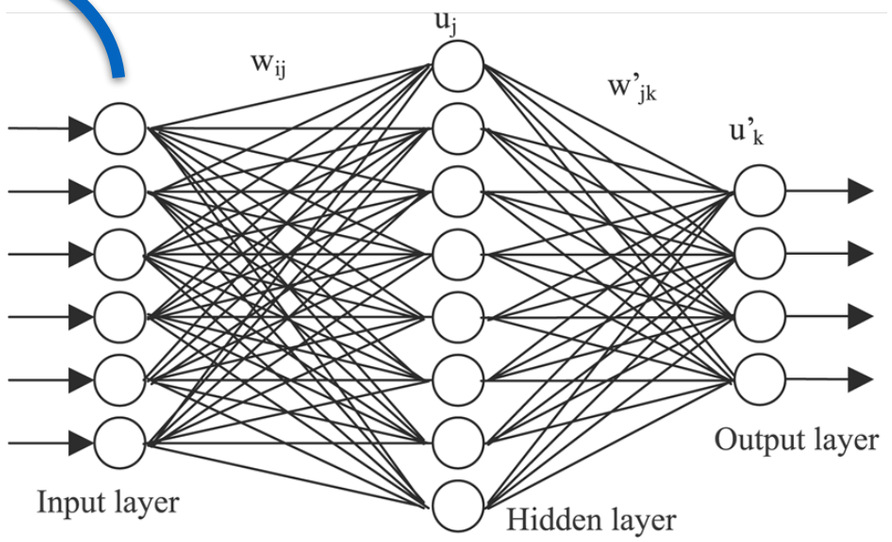
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(2 * numProducts,
                input_dim=numProducts,
                activation='relu'))

model.add(Dense(numProducts,
                activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```



python™

Keras

TensorFlow

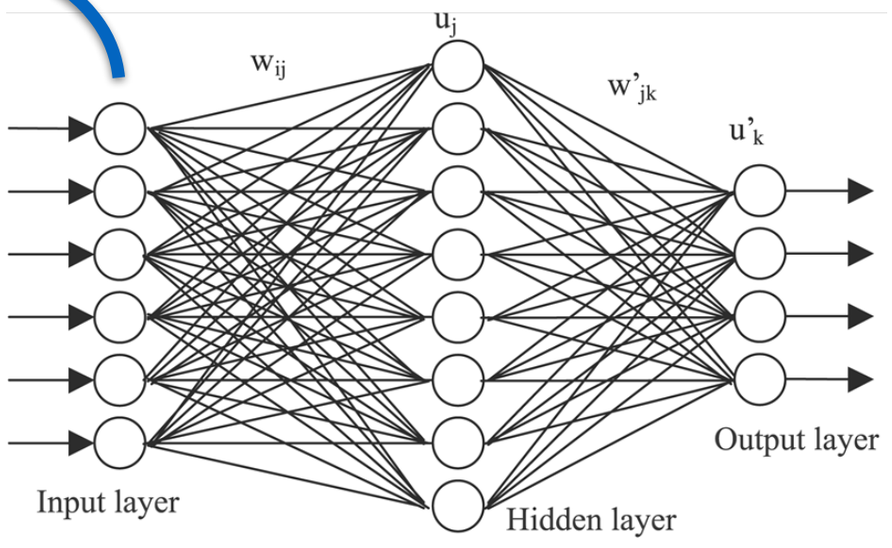
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(2 * numProducts,
                input_dim=numProducts,
                activation='relu'))

model.add(Dense(numProducts,
                activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```



python™

Keras

TensorFlow

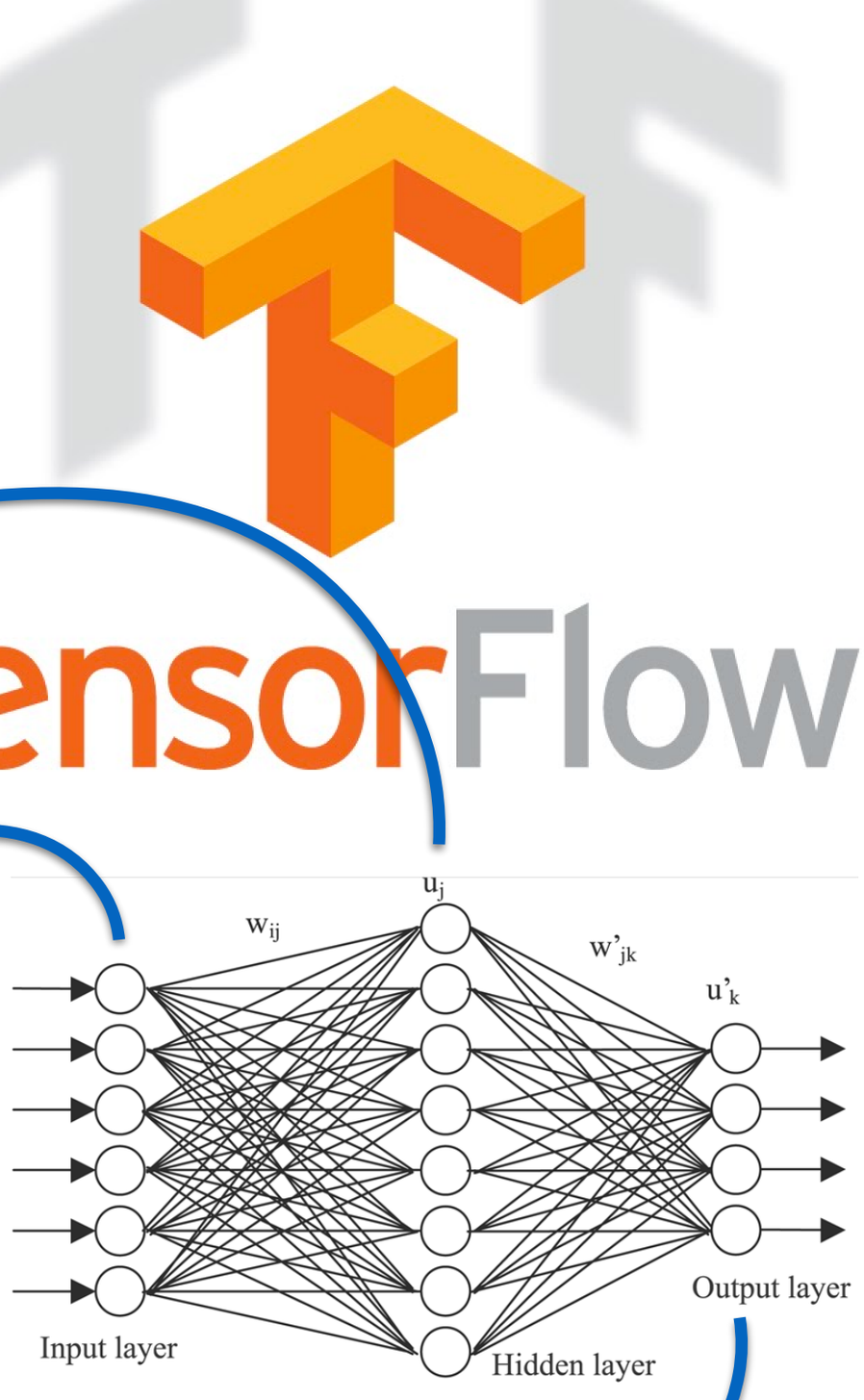
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(2 * numProducts,
                input_dim=numProducts,
                activation='relu'))

model.add(Dense(numProducts,
                activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```




```
model.fit(X, y, epochs=3, batch_size=2000)
```


```
model.fit(X, y, epochs=3, batch_size=2000)
```

epoch

- one complete pass through all training data

batch size

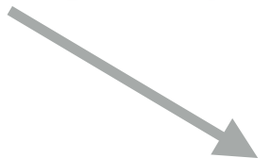
- amount of training data to consider before updating model weights
- higher batch size gives better results, but takes more RAM

Keras

TensorFlow

python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```



Keras

TensorFlow

python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

training samples

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

training samples

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

y

training samples

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

training samples

X

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

y

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	0	0	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
0	1	0	0	1	1	0
1	1	1	0	0	1	1
0	1	1	1	1	0	1
1	1	1	1	1	0	1
1	1	1	0	0	1	0
1	1	0	1	1	0	1

Keras TensorFlow python™

```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

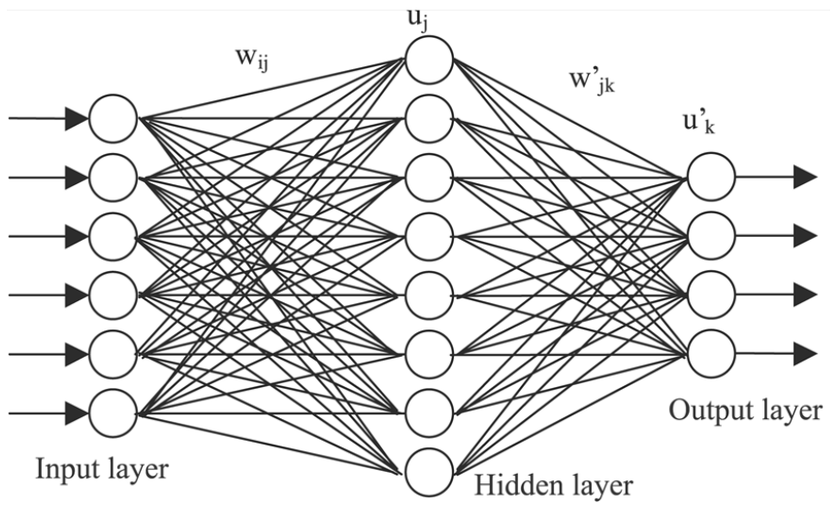
training samples

y

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	0	0	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
0	1	0	0	1	1	0
1	1	1	0	0	1	1
0	1	1	1	1	0	1
1	1	1	1	1	0	1
1	1	1	0	0	1	0
1	1	0	1	1	0	1

Keras

TensorFlow




```
model.fit(X, y, epochs=3, batch_size=2000)
```

X

y

training samples

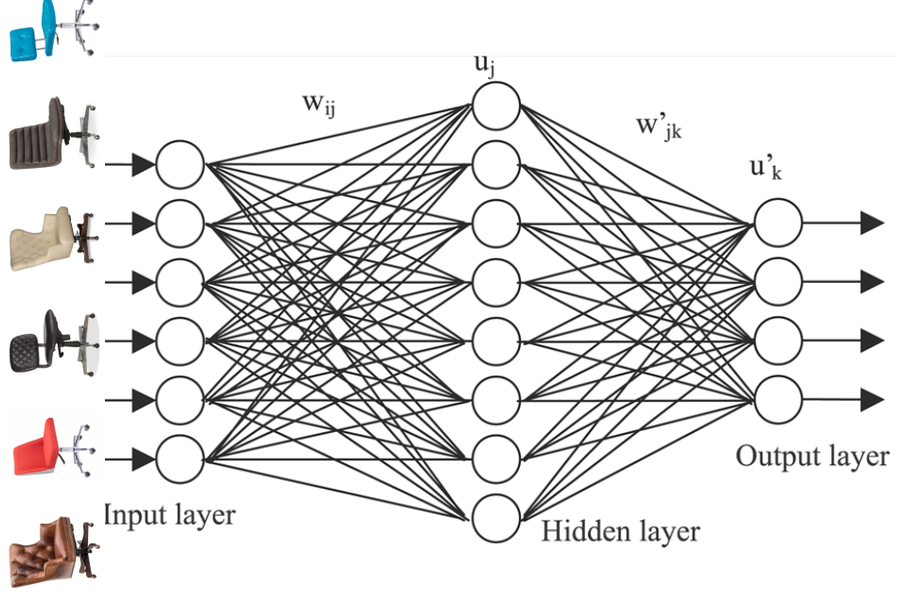
0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	0	0	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
0	1	0	0	1	1	0
1	1	1	0	0	1	1
0	1	1	1	1	0	1
1	1	1	1	1	0	1
1	1	1	0	0	1	0
1	1	0	1	1	0	1

Keras

TensorFlow

```
model.fit(X, y, epochs=3, batch_size=2000)
```



X

y

training samples

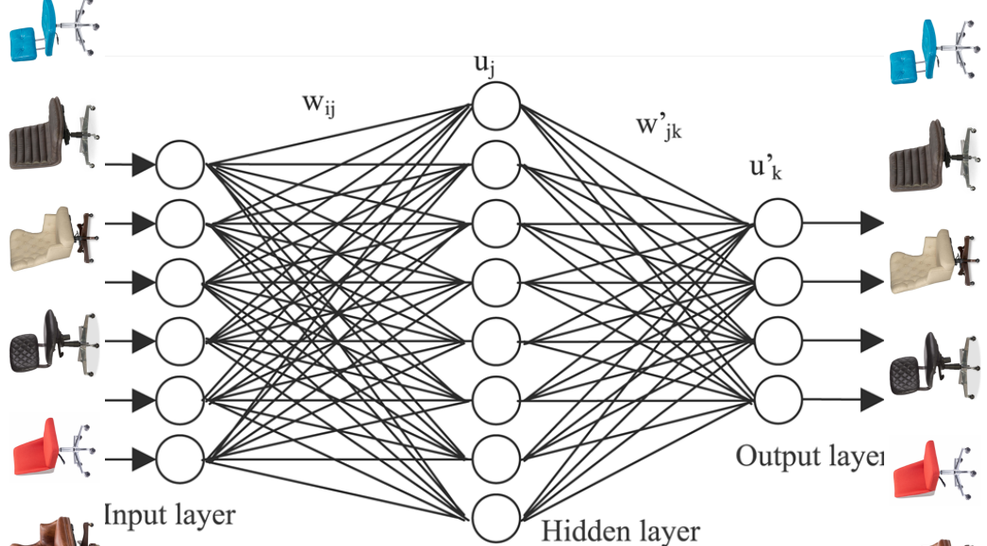
0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	0	0	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
0	1	0	0	1	1	0
1	1	1	0	0	1	1
0	1	1	1	1	0	1
1	1	1	1	1	0	1
1	1	1	0	0	1	0
1	1	0	1	1	0	1

Keras

TensorFlow

```
model.fit(X, y, epochs=3, batch_size=2000)
```



X

0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	1	0	0
0	0	1	0	0	0	0
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

y

0	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	0	0	0	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
0	1	0	0	1	1	0
1	1	1	0	0	1	1
0	1	1	1	1	0	1
1	1	1	1	1	0	1
1	1	1	0	0	1	0
1	1	0	1	1	0	1

training samples

should have used: **Word Embeddings** GloVe word2vec

Co-occurrence Matrix

	man	woman	queen
man	0	3	1
woman	3	0	5
queen	1	5	0

should have used: **Word Embeddings** GloVe word2vec

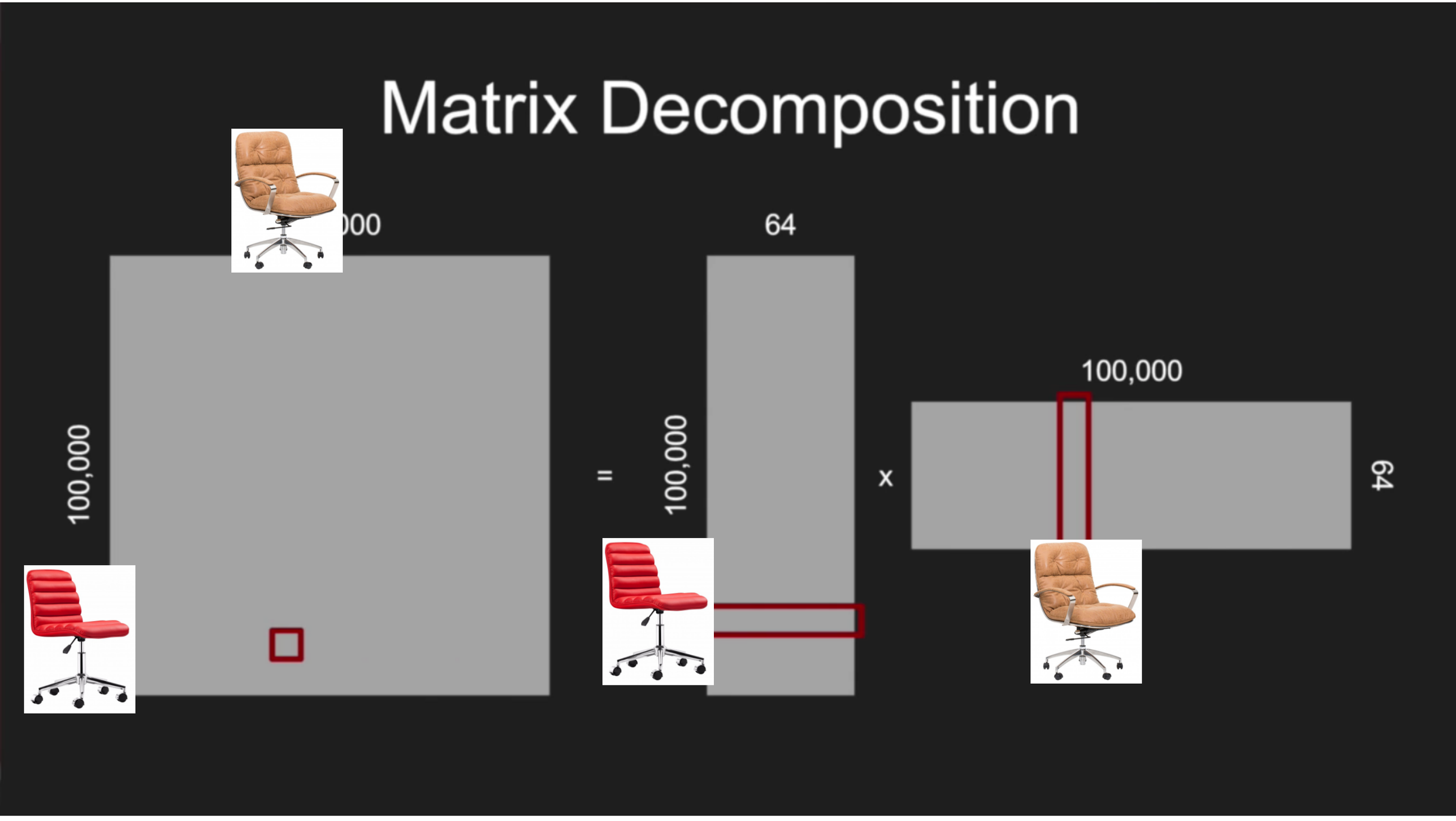
Co-occurrence Matrix






	0	3	1
	3	0	5
	1	5	0

should have used: **Word Embeddings** GloVe word2vec



should have used: **Word Embeddings** GloVe word2vec

gets you

[0.23, 0.65, 0.12, 0.84, 0.34]

instead of

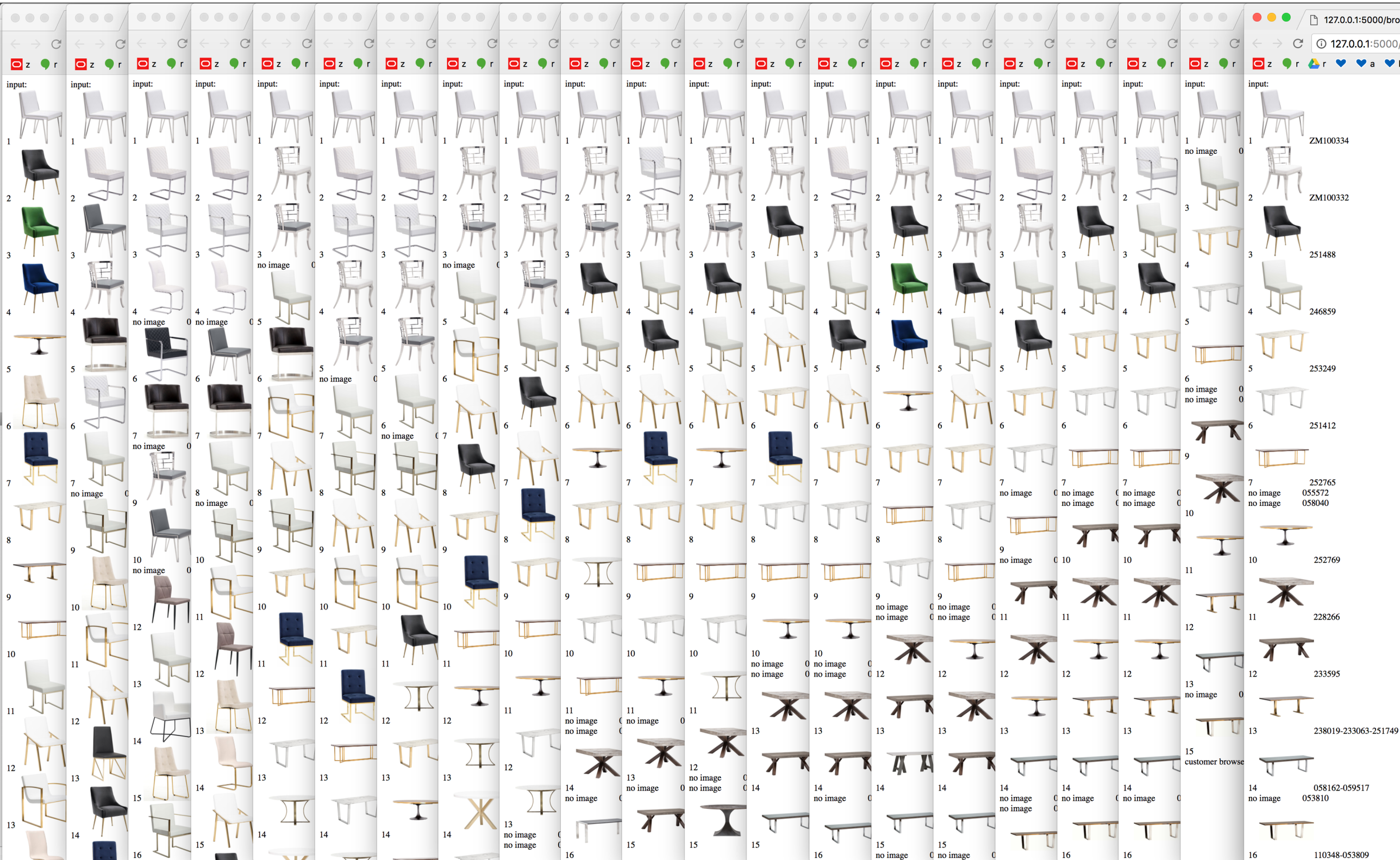
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, … 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0**]**

(for example)

epoch = 1 complete pass through all training data

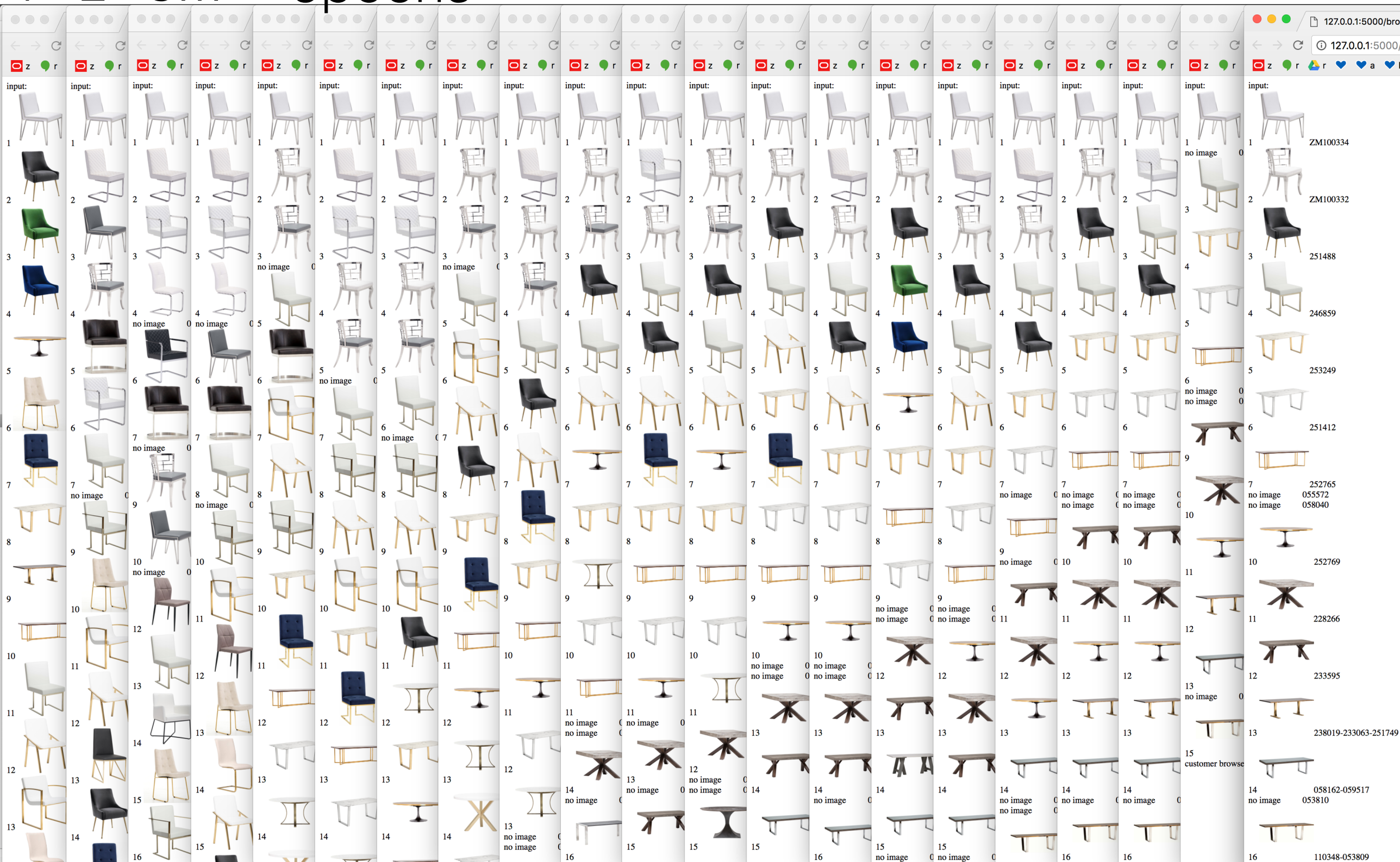
more is better?

epoch = 1 complete pass through all training data



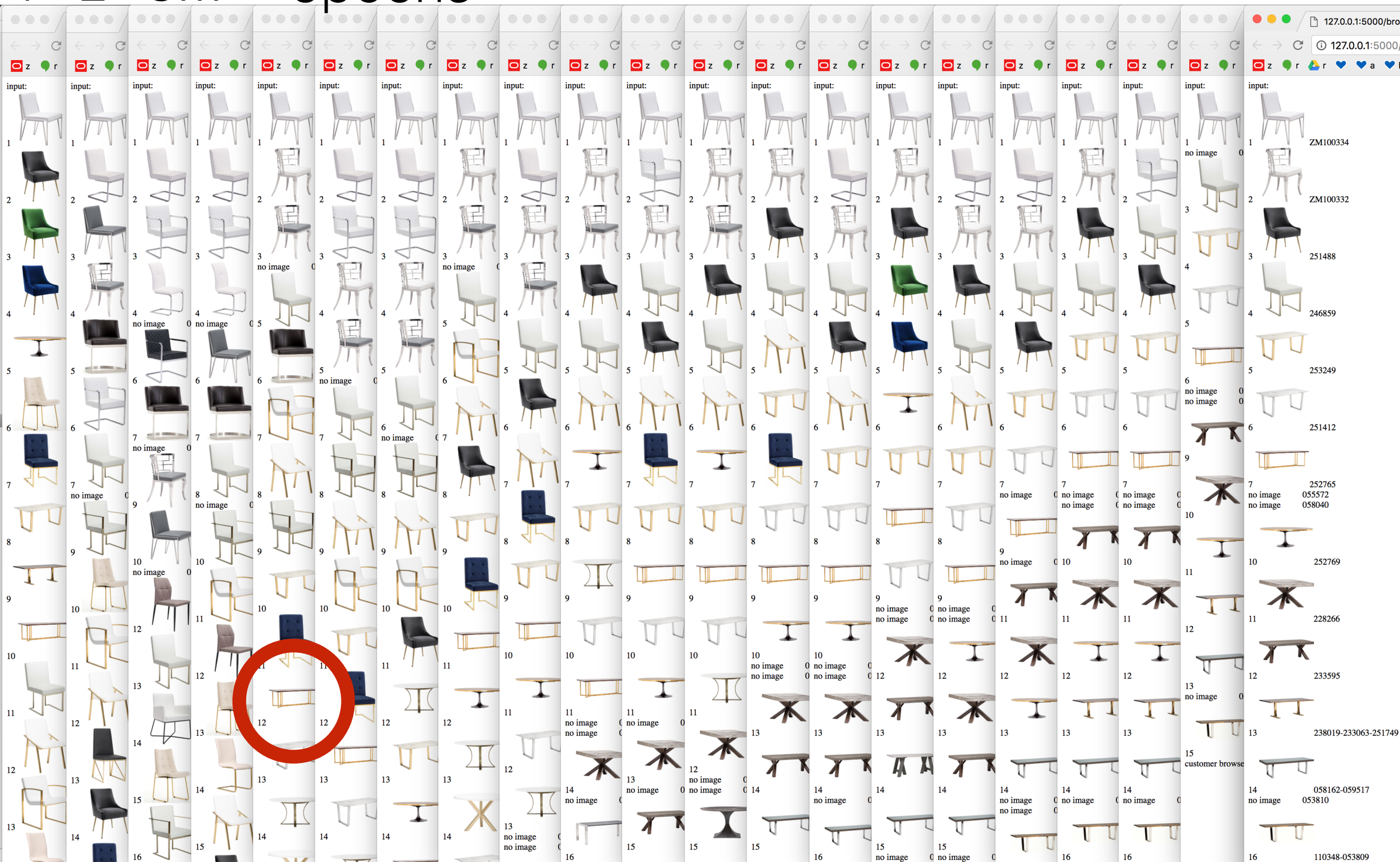
epoch = 1 complete pass through all training data

1 2 3... epochs



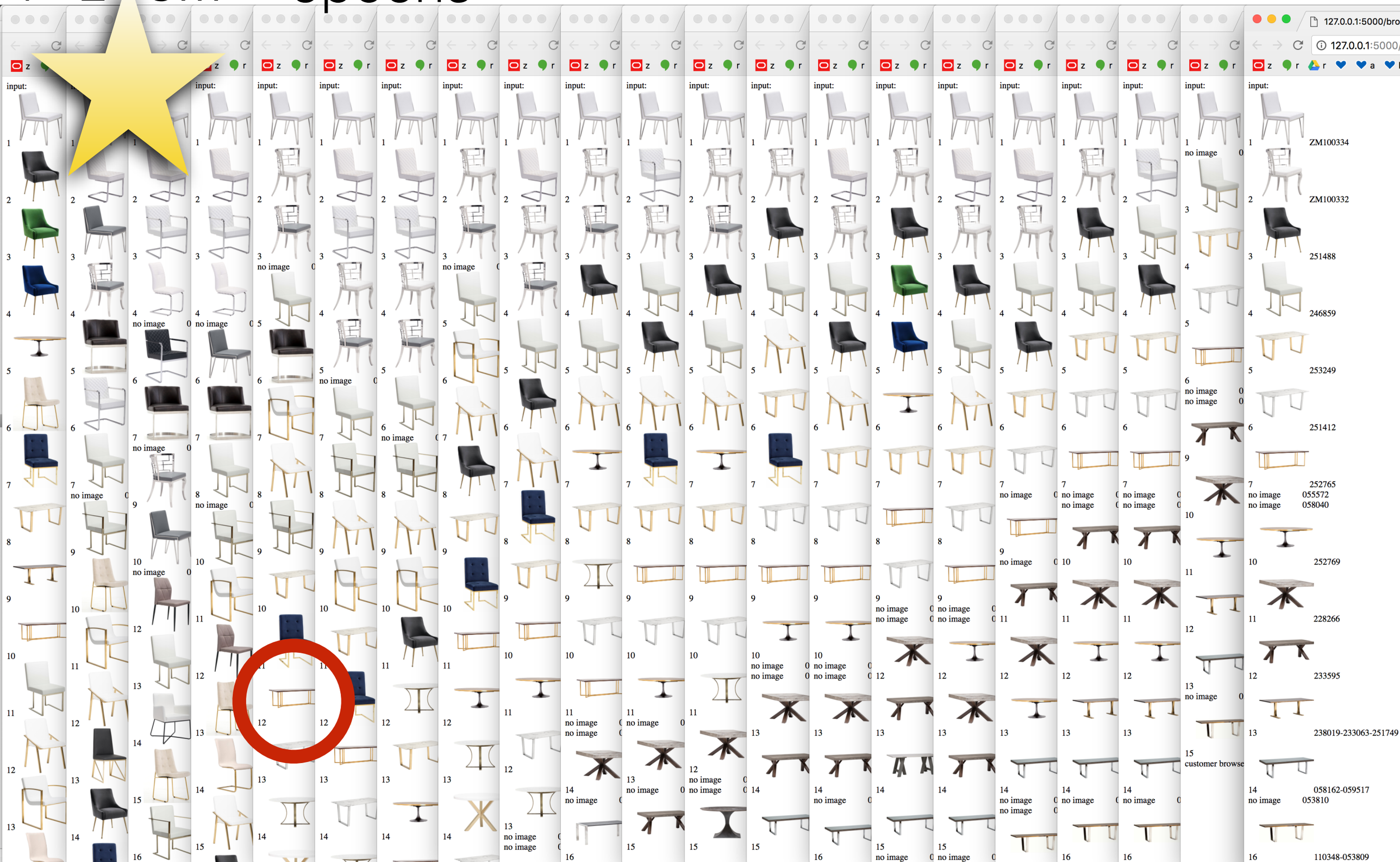
epoch = 1 complete pass through all training data

1 2 3... epochs



epoch = 1 complete pass through all training data

1 2 3... epochs



neural network model file: **1.2 GB**

(not feasible for production)

maybe using product embedding would help?

Scala

Vue.js

python™

elasticsearch

Solr

thoughts

- Easier than React
- Official routing
- Official state management
- Templates (easier to read/manage than JSX)
- full CSS allowed per component