In [2]:

```
from __future__ import print_function

import numpy as np
import keras

from keras.datasets import mnist
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Activation, Input
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 1

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax', name='preds'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
```

```
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 138s 2ms/step - loss: 0.2348 - a
cc: 0.9277 - val_loss: 0.0523 - val_acc: 0.9830
Test loss: 0.05226928142679389
Test accuracy: 0.983
```

Visualizing our Dense Layers

To visualize activation over final dense layer outputs, we need to **switch the softmax activation out for linear** since gradient of our output node will depend on all the other node activations.

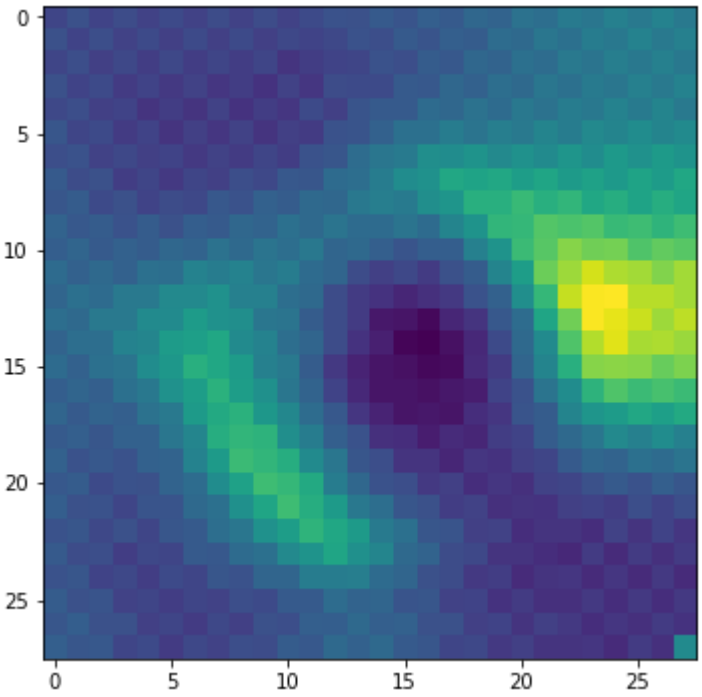
Doing this in keras is tricky, so they have provided **utils.apply_modifications** to modify network parameters and rebuild the graph.

If this swapping is not done, the results might be suboptimal. We will start by swapping out 'softmax' for 'linear' and compare what happens if we don't do this at the end.

Lets start by visualizing an input that maximizes the output of node 0. This should look like a 0.

In [3]:
```
from vis.visualization import visualize_activation
from vis.utils import utils
from keras import activations
from matplotlib import pyplot as plt
%matplotlib inline

plt.rcParams['figure.figsize'] = (18, 6)

# Utility to search for layer index by name.
# Alternatively we can specify this as -1 since it corresponds to the last layer.
layer_idx = utils.find_layer_idx(model, 'preds')

# Swap softmax with linear
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)

# This is the output node we want to maximize.
filter_idx = 0
img = visualize_activation(model, layer_idx, filter_indices=filter_idx)
plt.imshow(img[..., 0])
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x7fc23f6ac630>
```



While this sort of looks like a 0, it's not as clear as we hoped for.

The reason is because because regularization parameters needs to be tuned depending on the problem. This can be improved by:

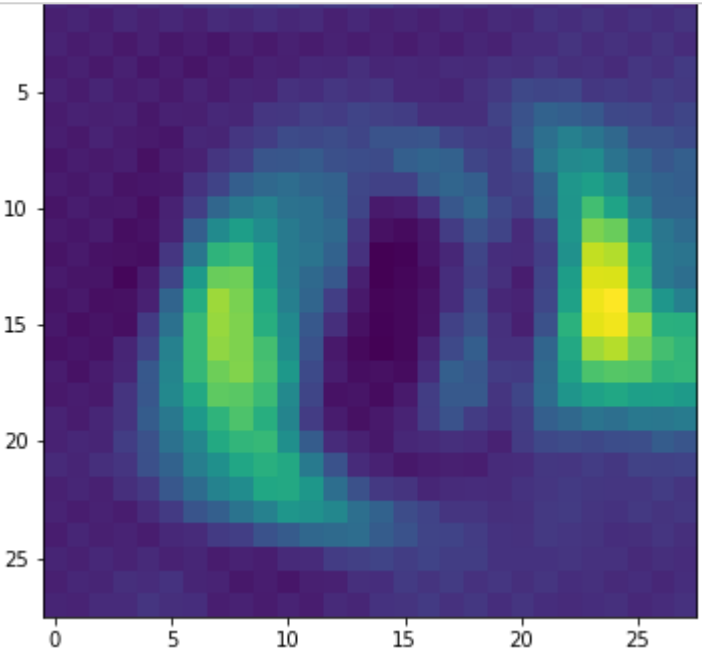
- The input to network is preprocessed to range (0, 1). We should specify input_range = (0., 1.) to constrain the input to this range.
- The regularization parameter default weights might be dominating activation maximization loss weight. One way to debug this is to use verbose=True and examine individual loss values.

Lets do these step by step and see if we can improve it.

Specifying the Input Range

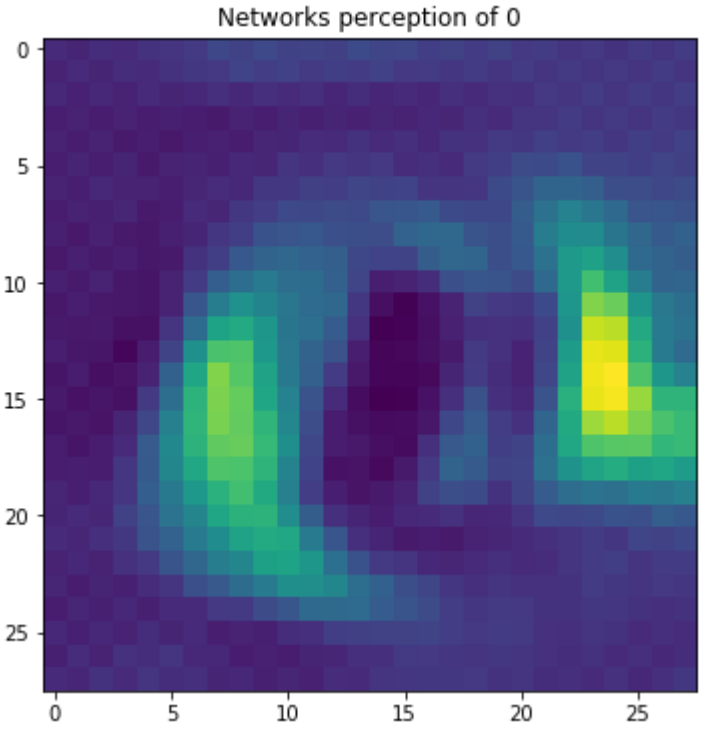
First lets explore what the individual losses look like

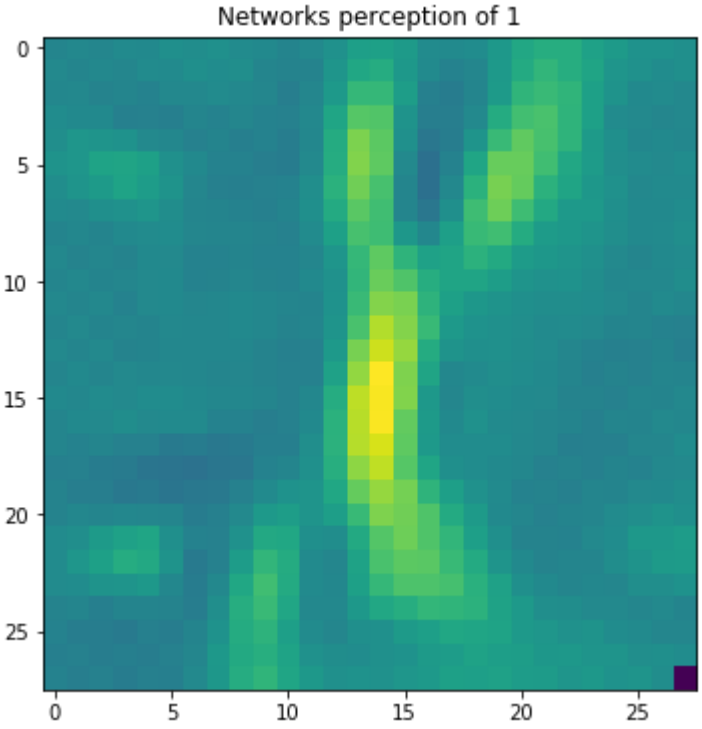
In [14]:
```
img = visualize_activation(model, layer_idx, filter_indices=filter_idx, input_rar
plt.imshow(img[..., 0])
```

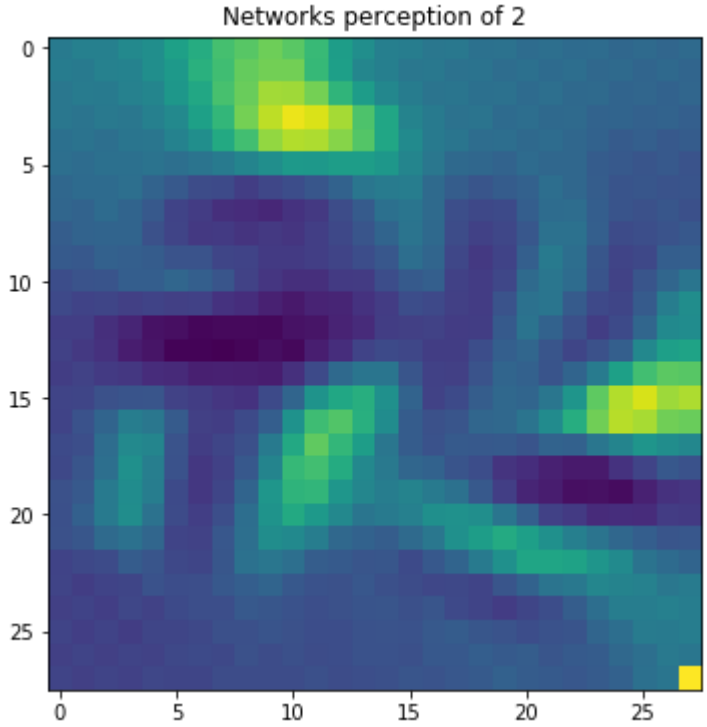


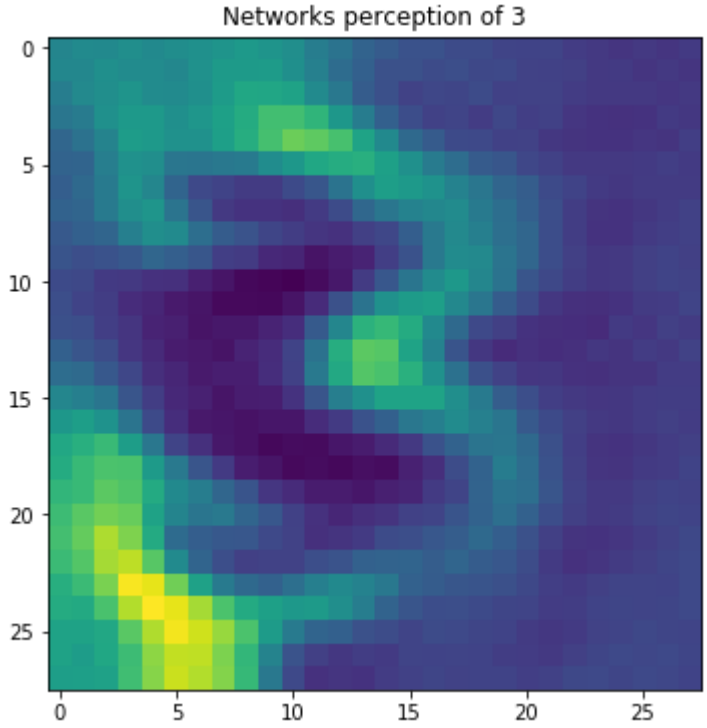
Now let's do it for all classes

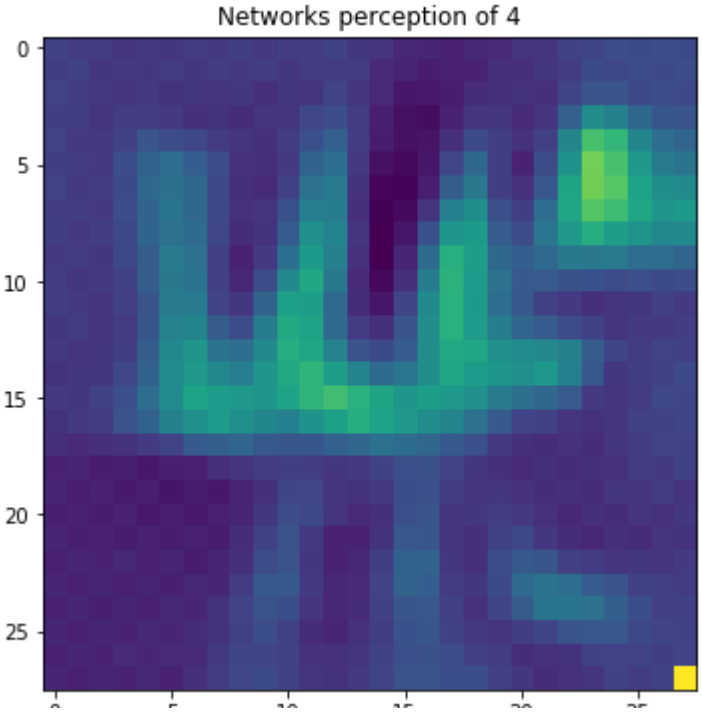
In [15]:
```
# Generating visualizations for all classes (0-9)
for output_idx in np.arange(10):
    img = visualize_activation(model, layer_idx, filter_indices=output_idx, input
    plt.figure()
    plt.title('Networks perception of {}'.format(output_idx))
    plt.imshow(img[..., 0])
```

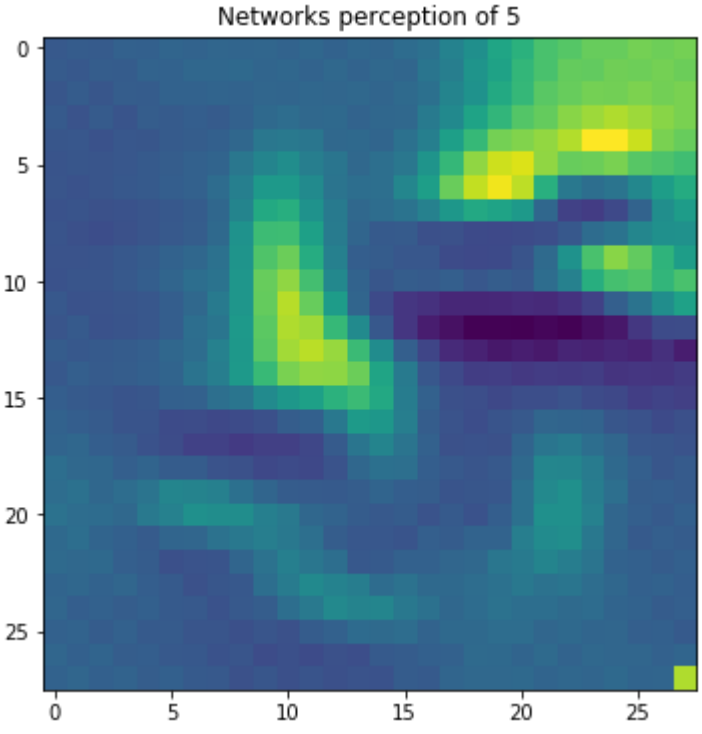


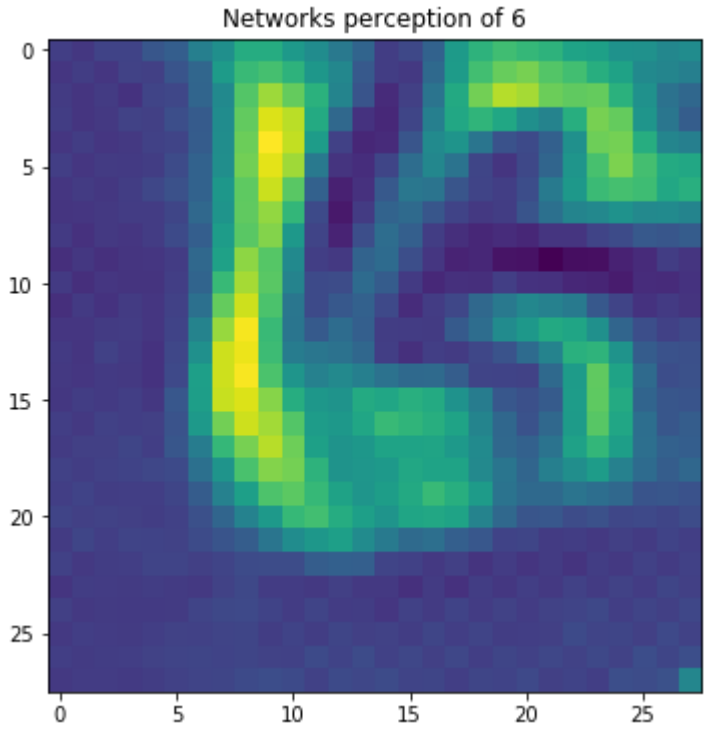


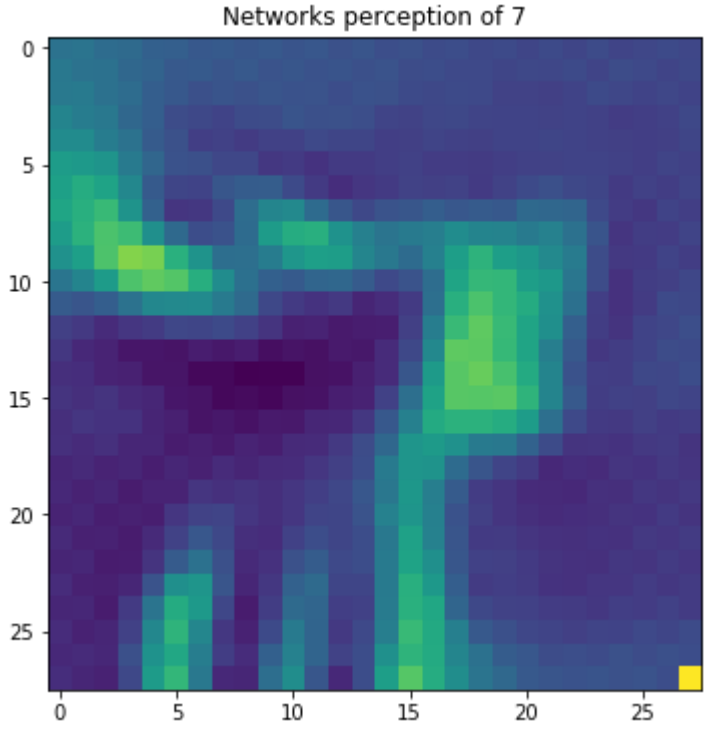

Networks perception of 2
0
5
10
15
20
25
0
5
10
15
20
25

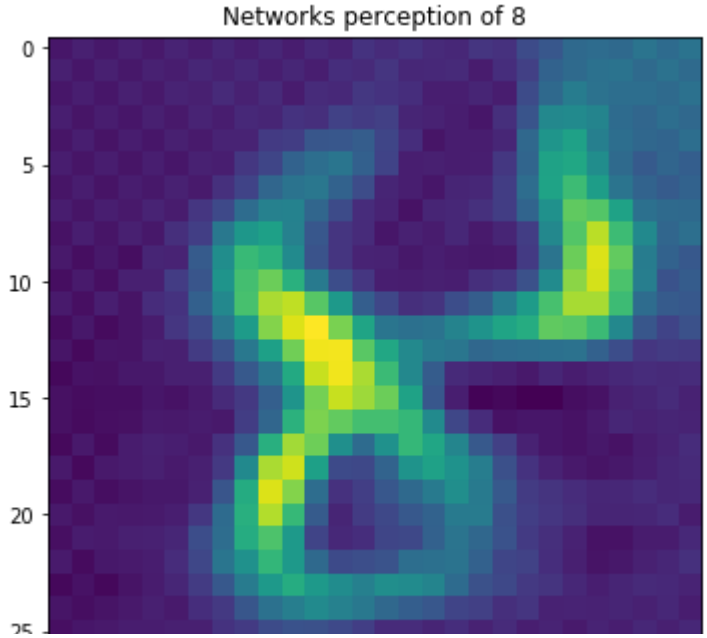

Networks perception of 3
0
5
10
15
20
25
0
5
10
15
20
25

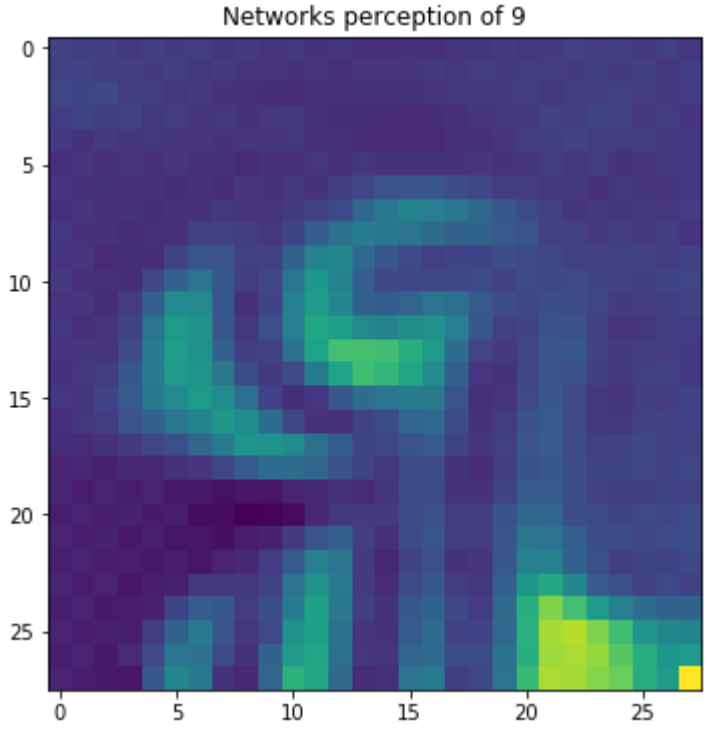

Networks perception of 4
0
5
10
15
20
25


Networks perception of 5
0
5
10
15
20
25
0
5
10
15
20
25


Networks perception of 6
0
5
10
15
20
25
0
5
10
15
20
25


Networks perception of 7
0
5
10
15
20
25
0
5
10
15
20
25


Networks perception of 8
0
5
10
15
20


Networks perception of 9
0
5
10
15
20
25
0
5
10
15
20
25