
Univ.AI


Ai 1

Lets Start

Start with:

The **make you dangerous** workshop.

And then, the make you super dangerous and rigorous full course.

(for those taking the full deal...)

Resources we'll use today

- your machine
- Google Colab
- binder

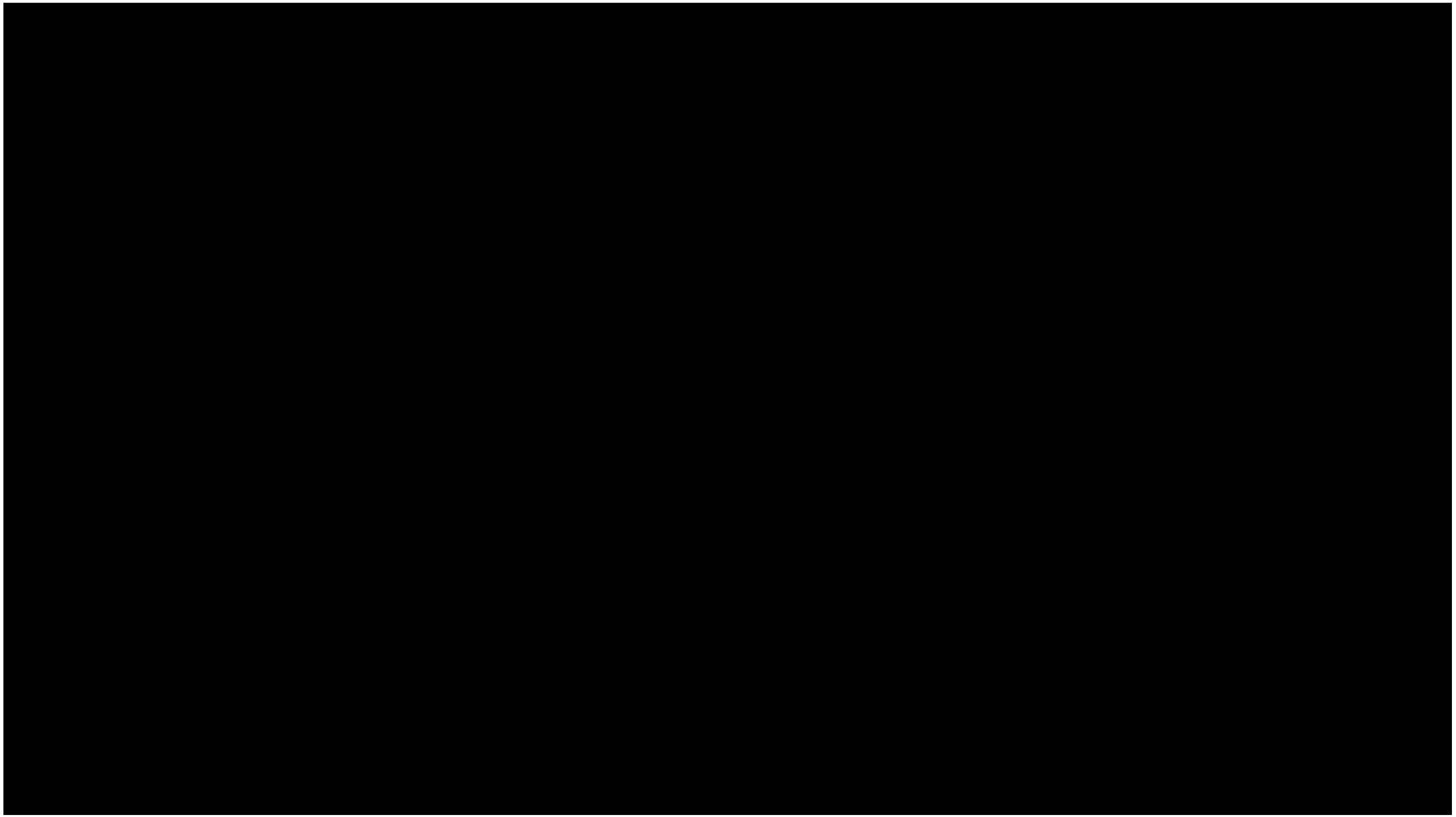
Resources for Full Dealers

(coming this week)

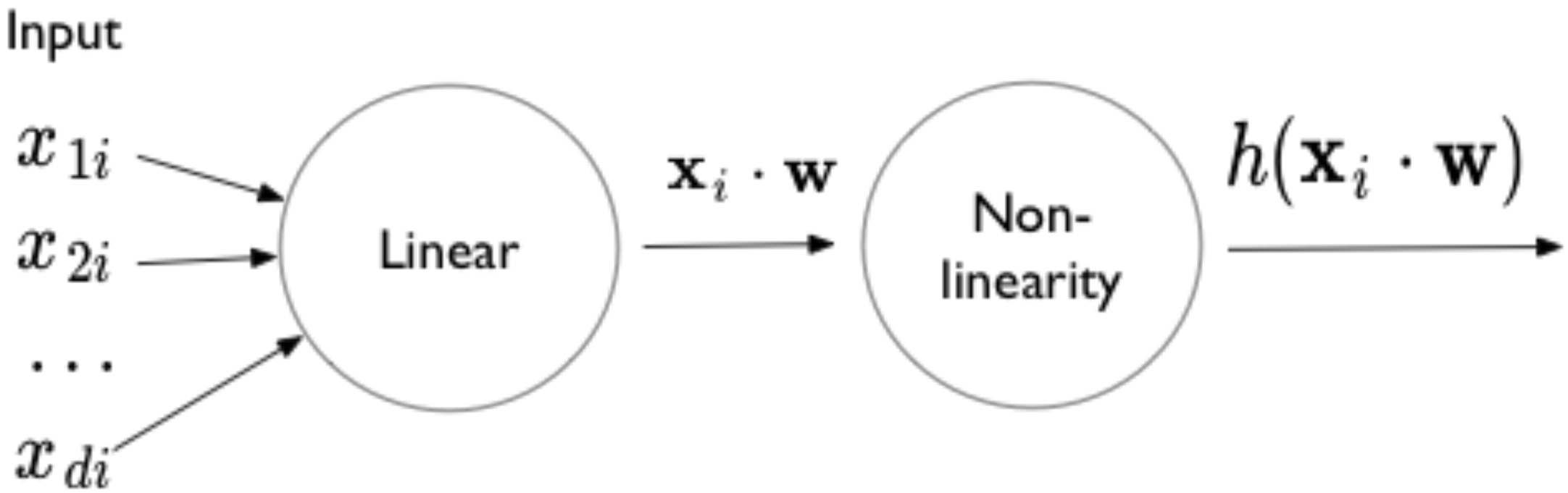
- Formation of groups for homework and fun
- Discussion Forum across college campuses
- Educational platform
- GPU based custom compute (for project)
- TA mentorship and office hours
- Professor office hours

Do not feel shy to ask anything

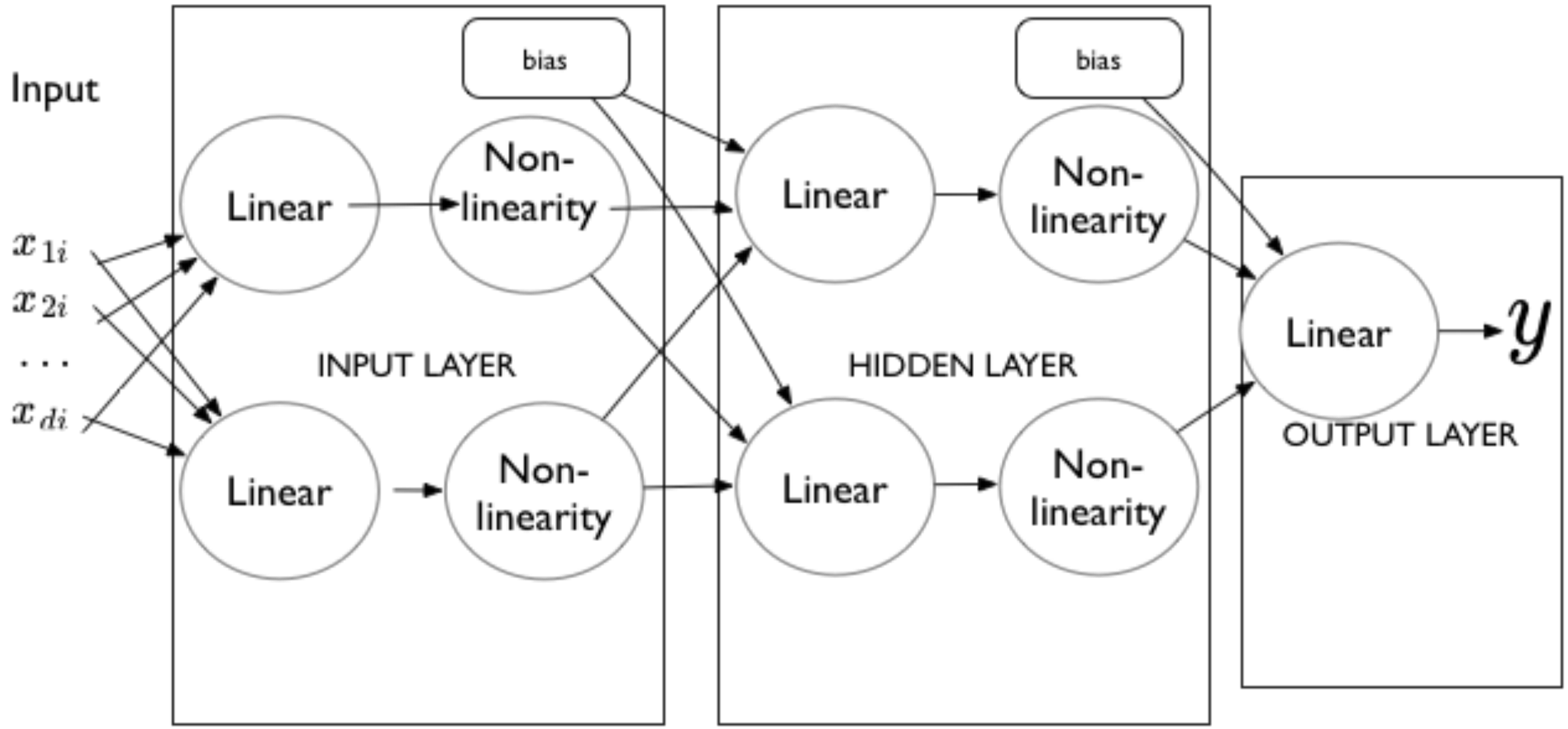
Learning a 3

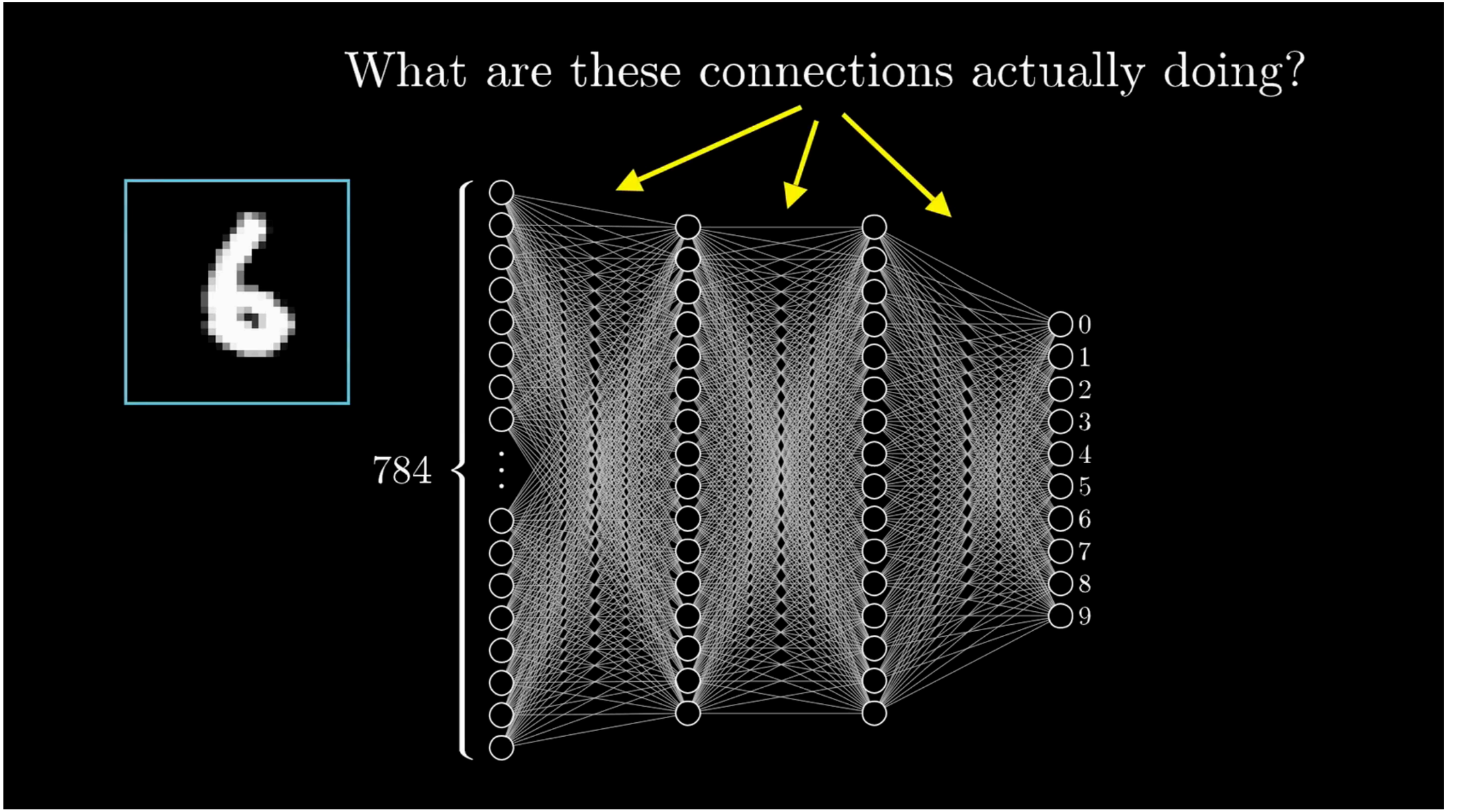


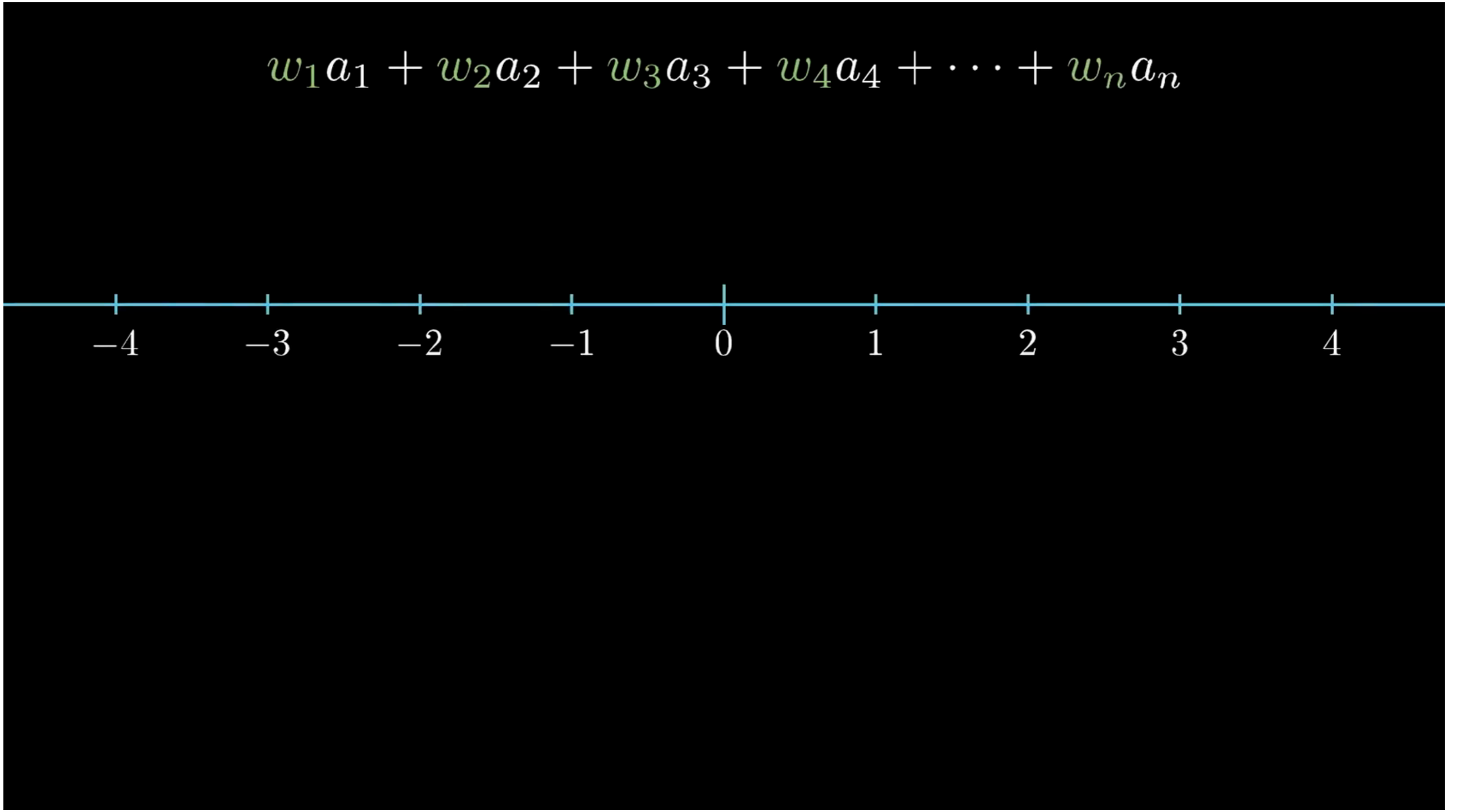
The perceptron $f((w, b) \cdot (x, 1))$



Combine Perceptrons

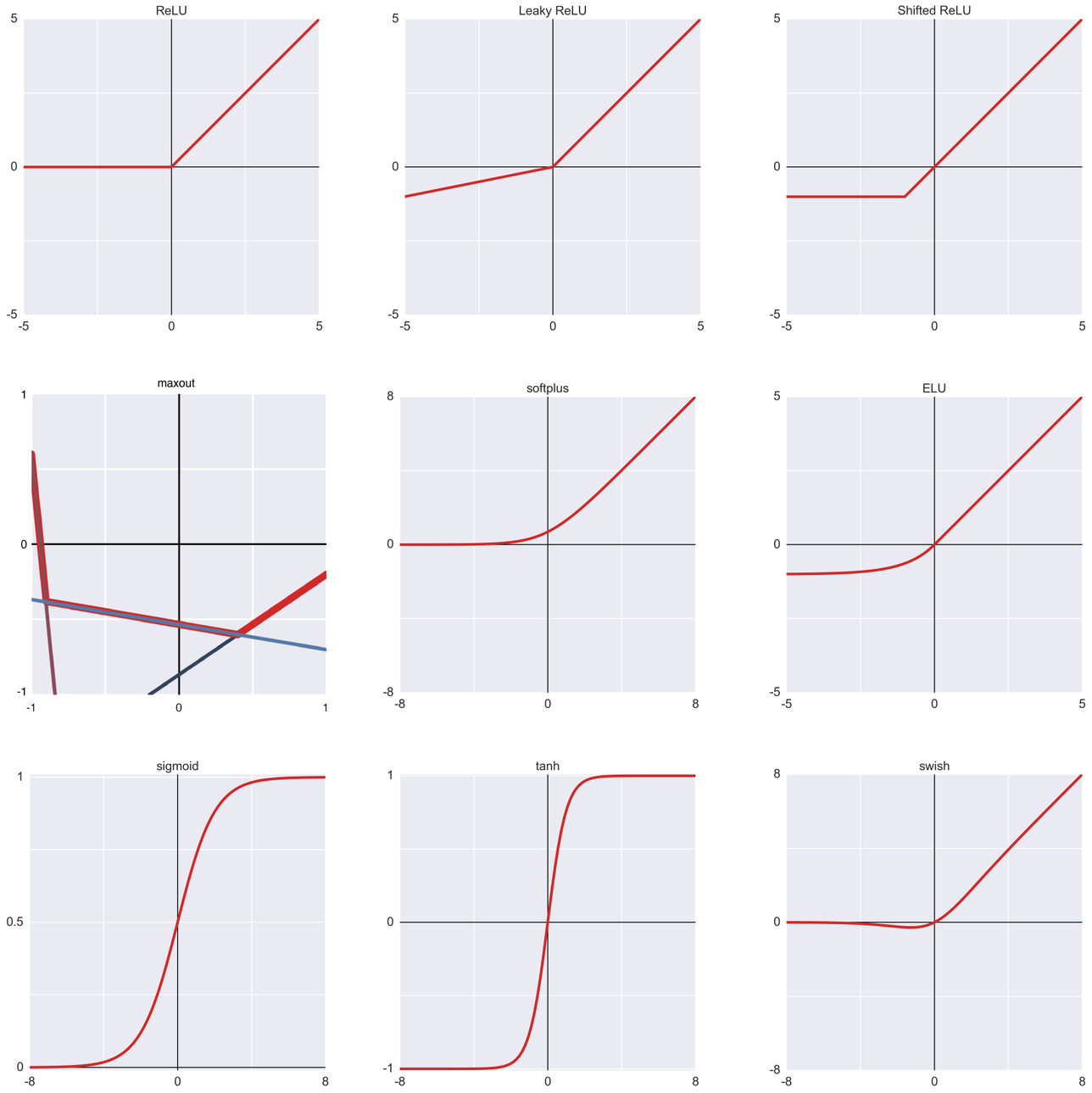


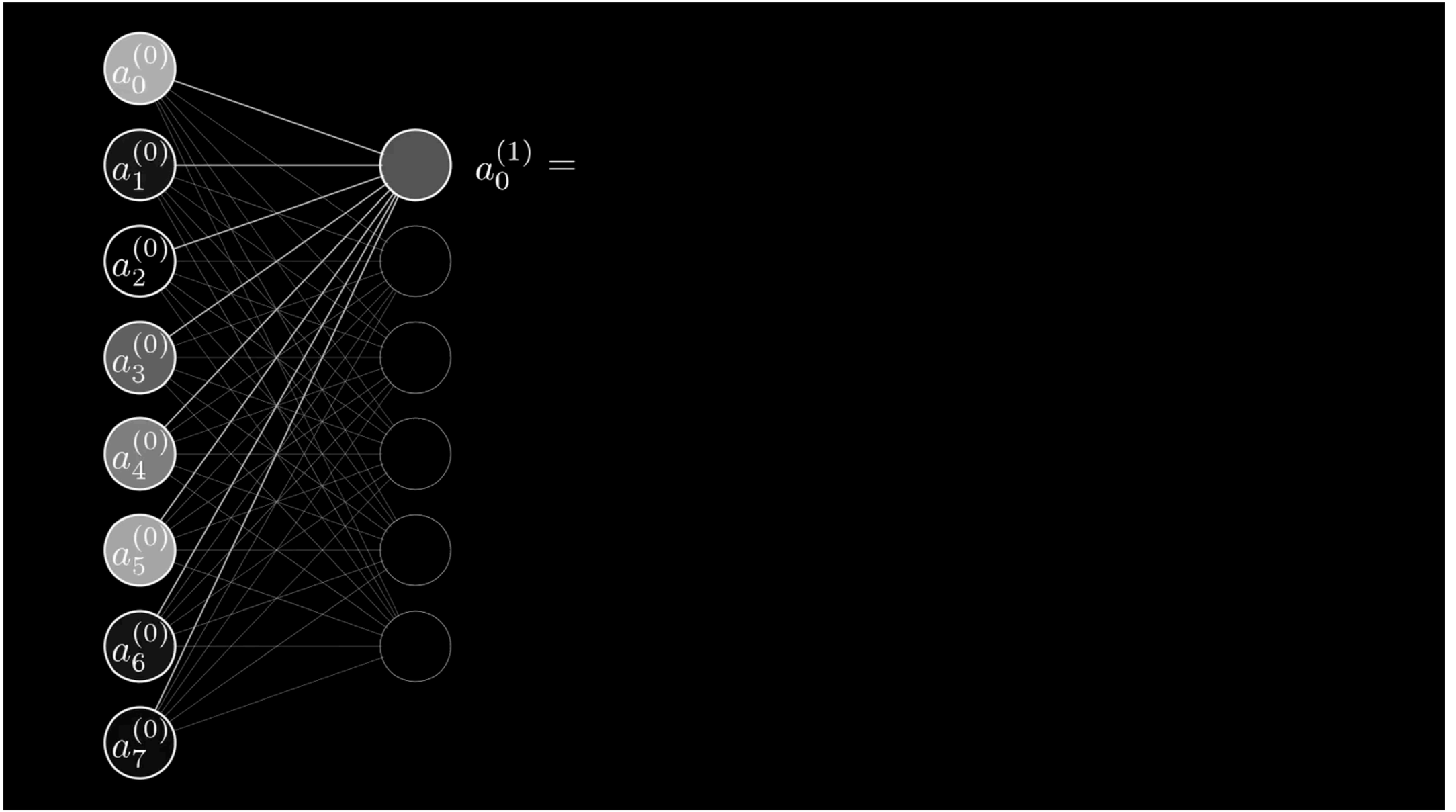

What are these connections actually doing?
784
0
1
2
3
4
5
6
7
8
9


$w_1a_1 + w_2a_2 + w_3a_3 + w_4a_4 + \cdots + w_na_n$
−4
−3
−2
−1
0
1
2
3
4

Non-Linearity

we want a non-linearity as othersie combining linear regressions just gives a big honking linear regression



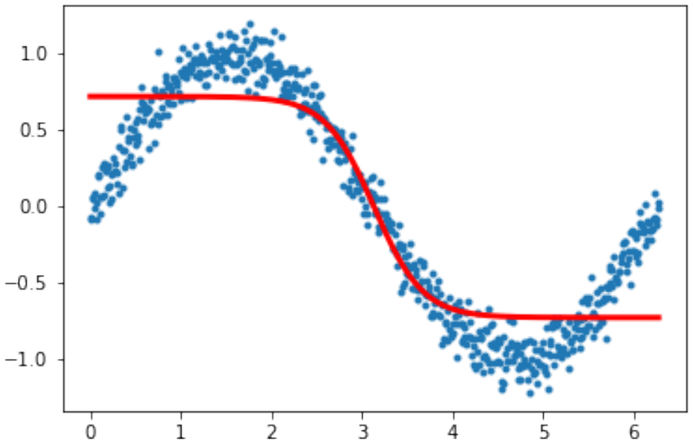

$a_0^{(0)}$
$a_1^{(0)}$
$a_2^{(0)}$
$a_3^{(0)}$
$a_4^{(0)}$
$a_5^{(0)}$
$a_6^{(0)}$
$a_7^{(0)}$
$a_0^{(1)} =$

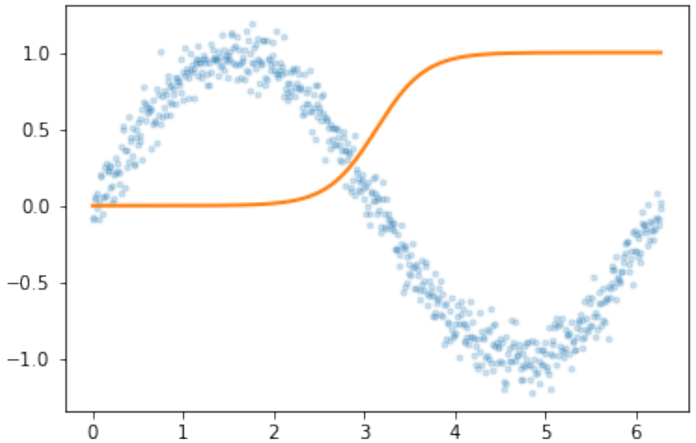
Universal Approximation: Learn a complex function

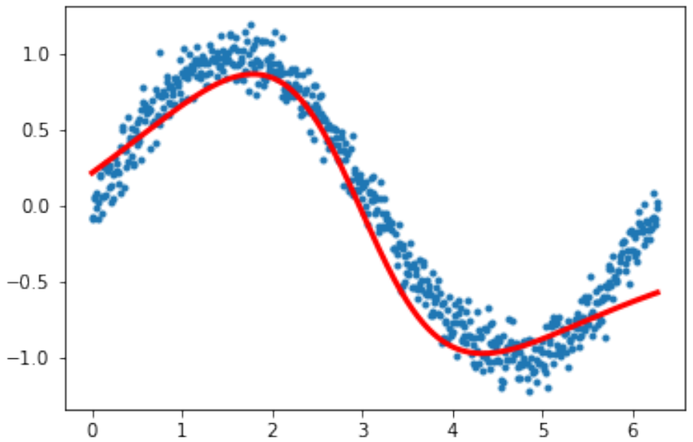
THEOREM:

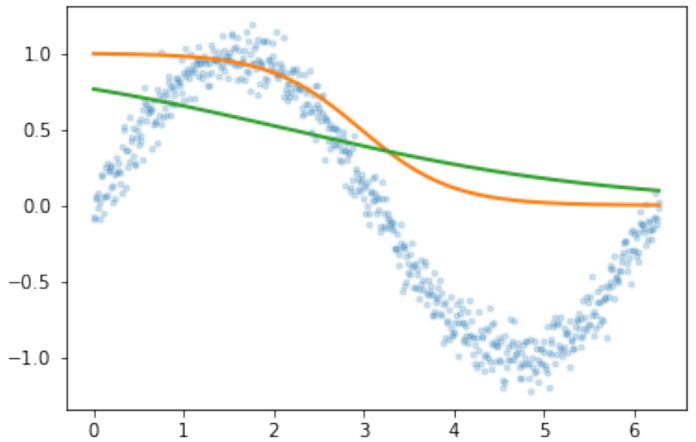
- any one hidden layer net can approximate any continuous function with finite support, with appropriate choice of nonlinearity
- but may need lots of units
- and will learn the function it thinks the data has, not what you think

One hidden, 1 vs 2 neurons

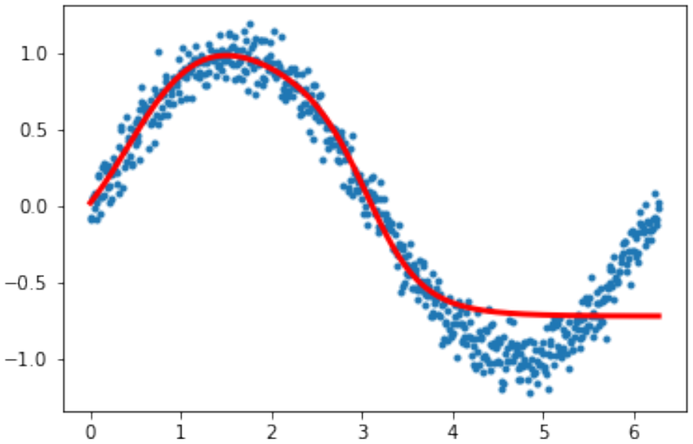


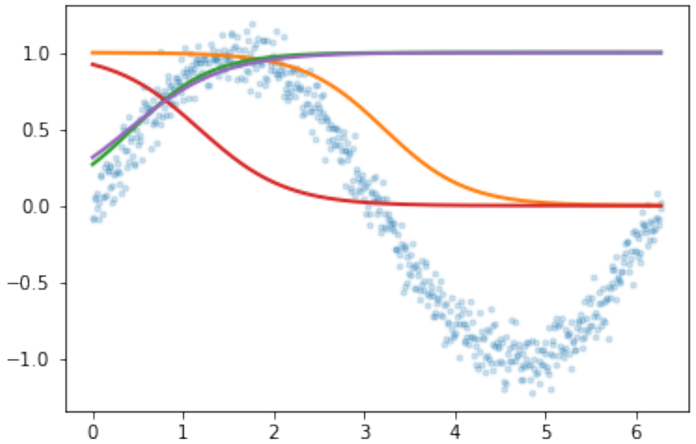


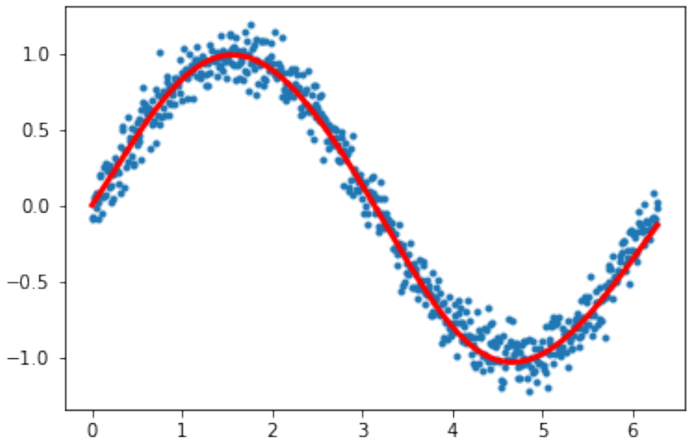


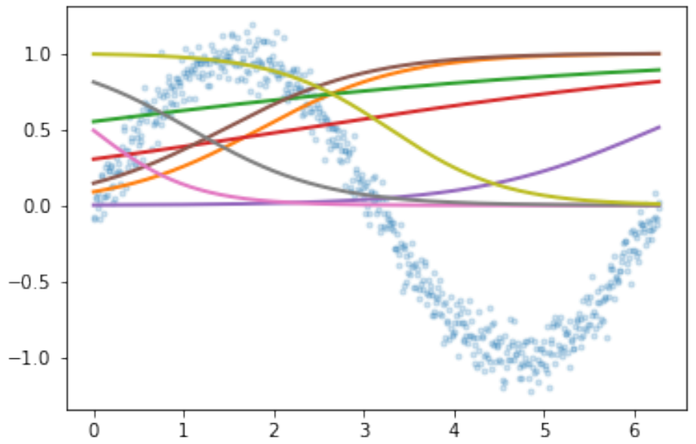


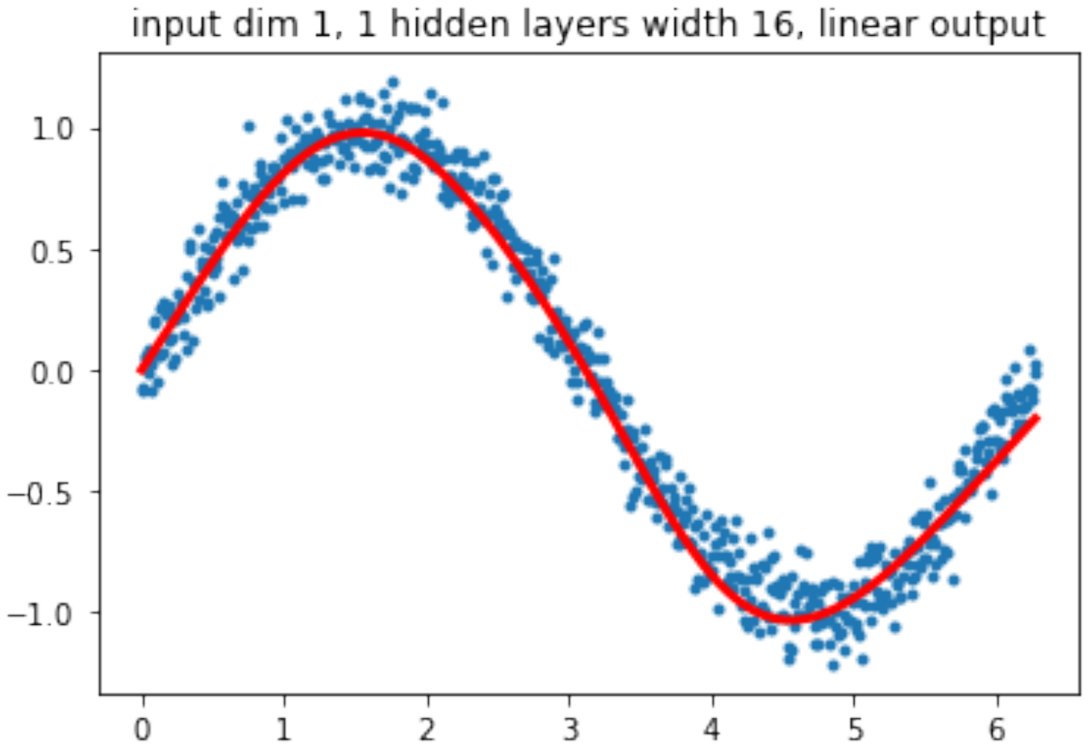
Two hidden, 4 vs 8 neurons

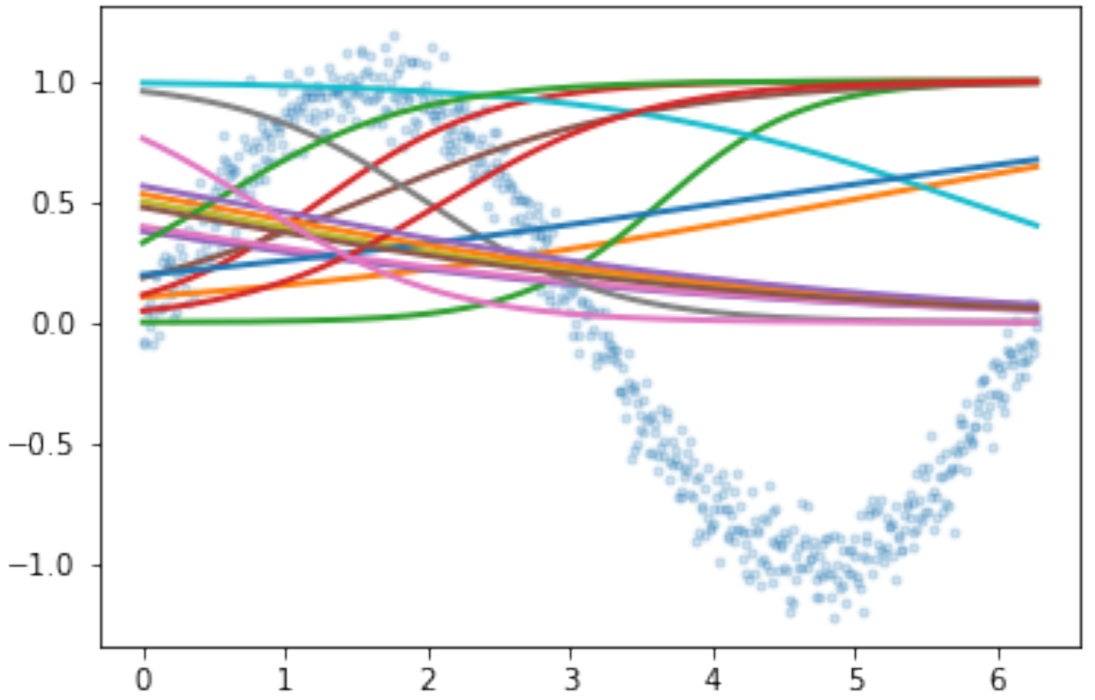




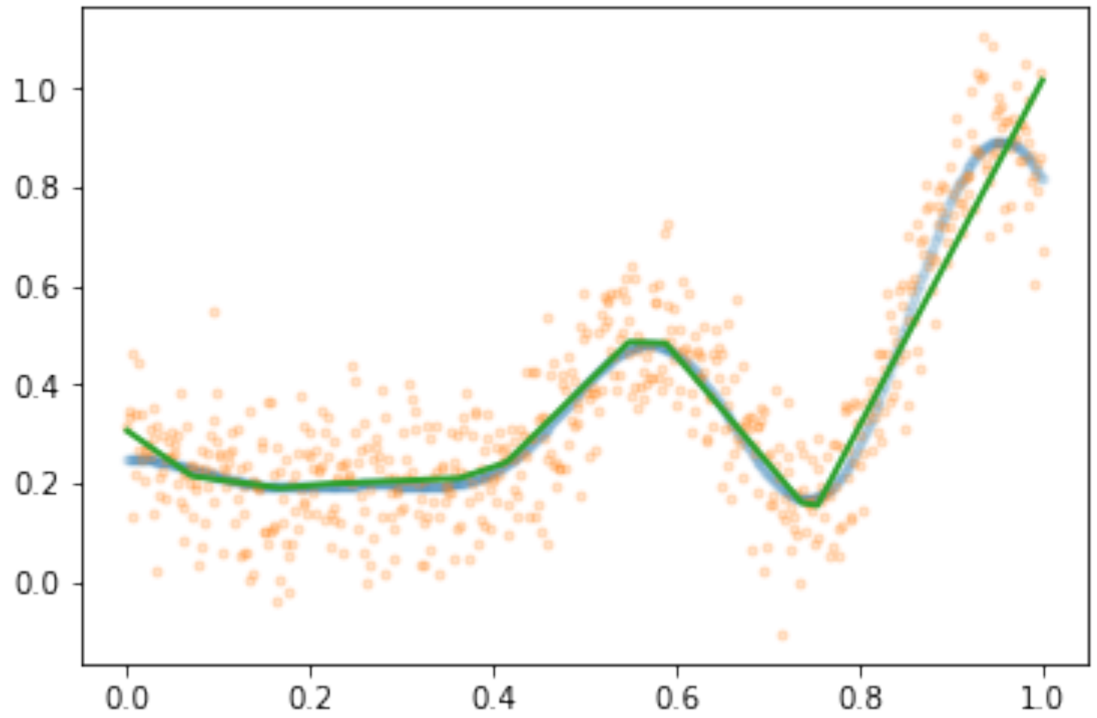


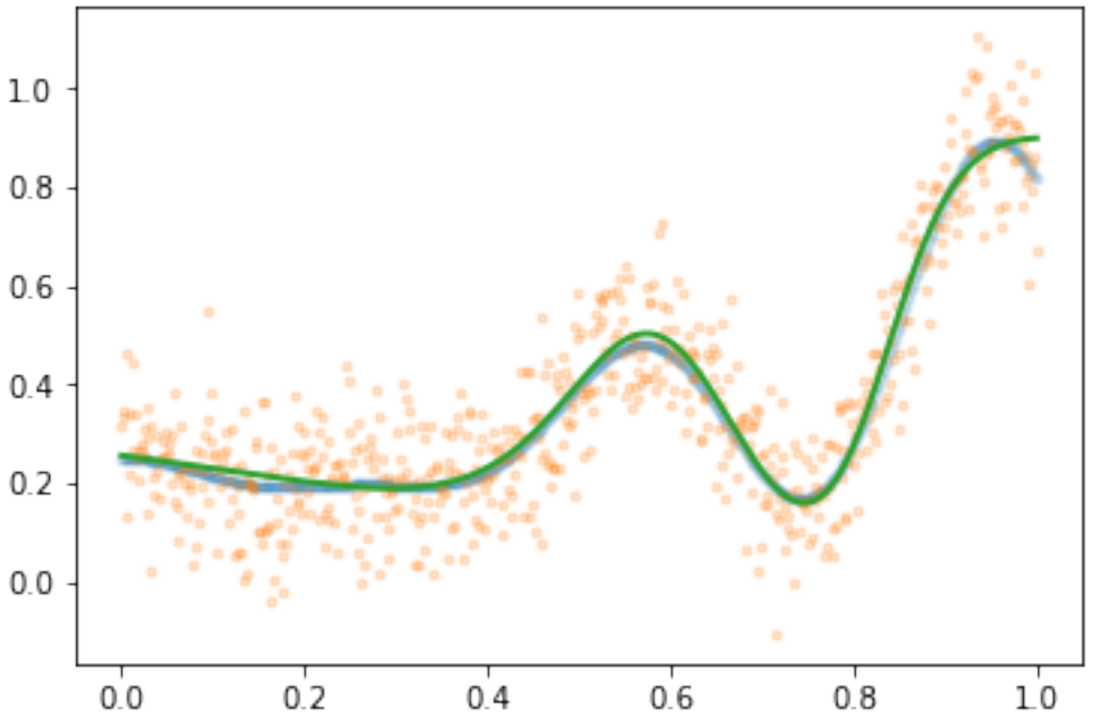



input dim 1, 1 hidden layers width 16, linear output
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6

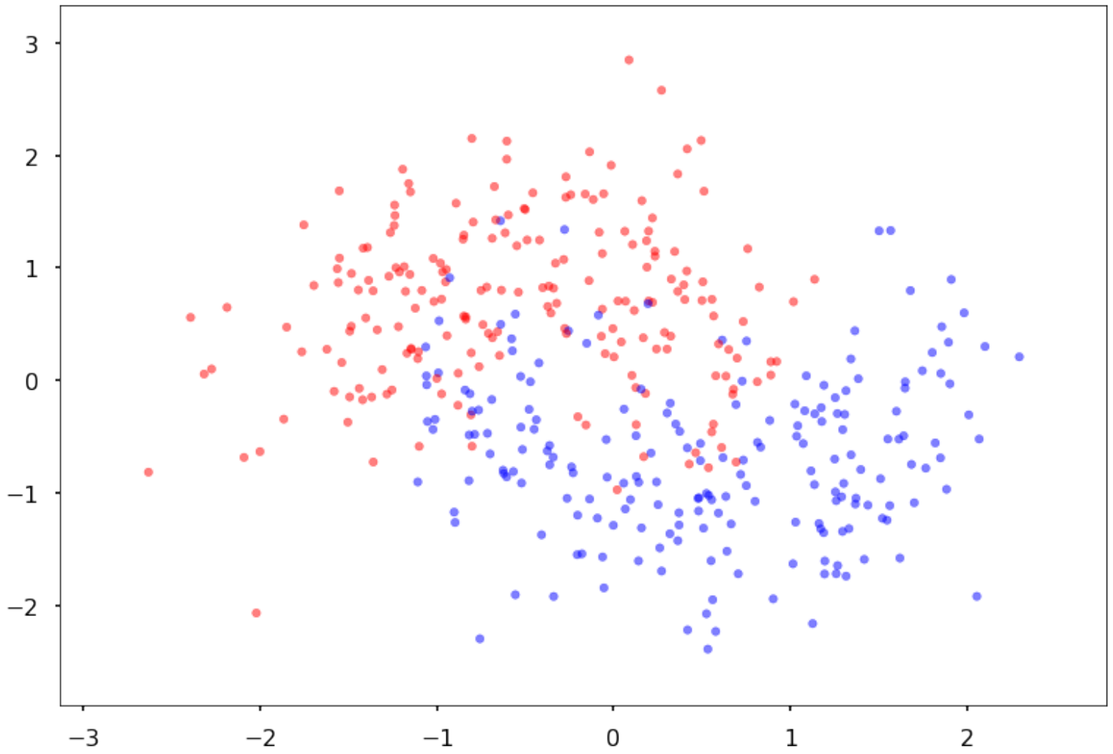

1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6

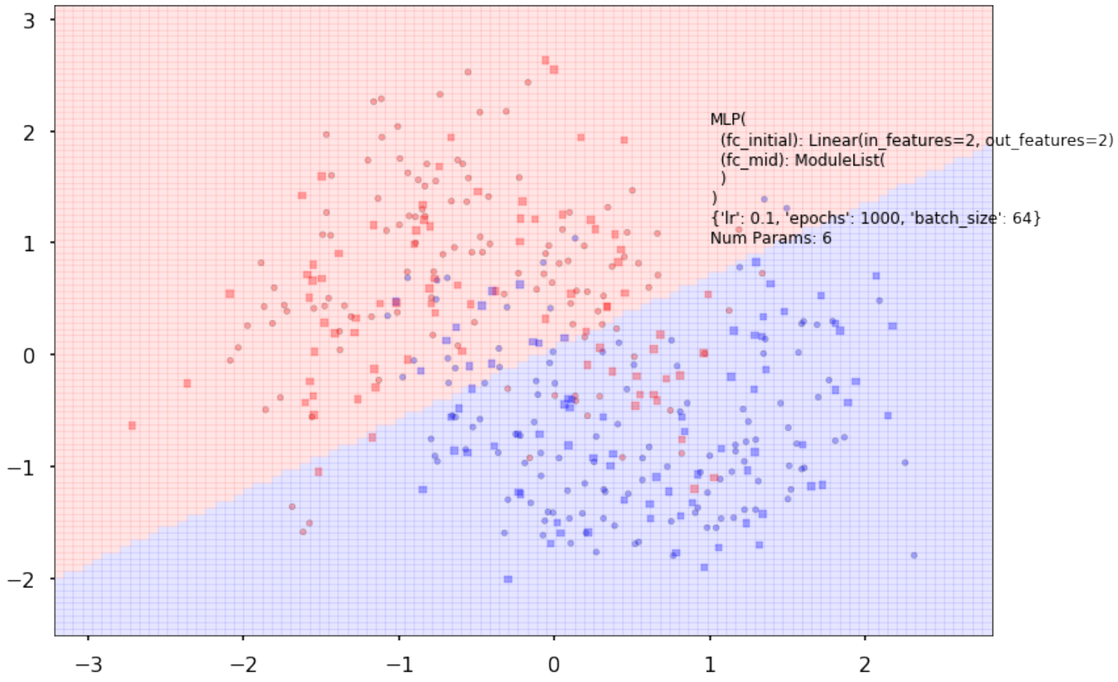
Relu (80, 1 layer) and tanh(40, 2 layer)



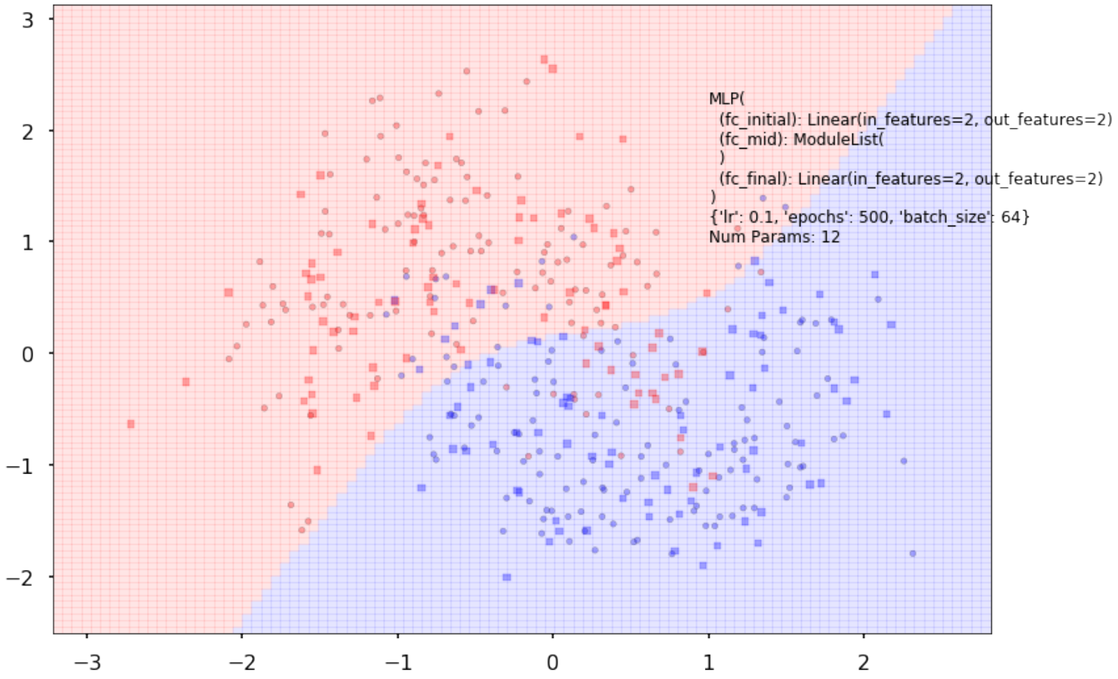


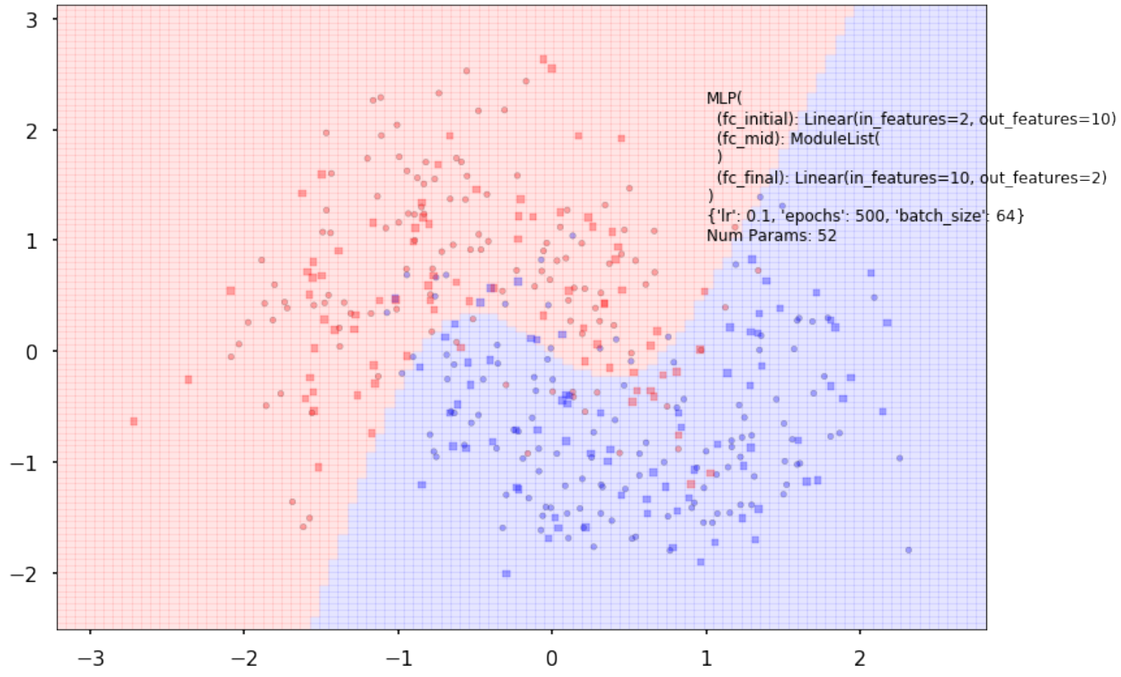
Half moon dataset (artificially GENERATED)



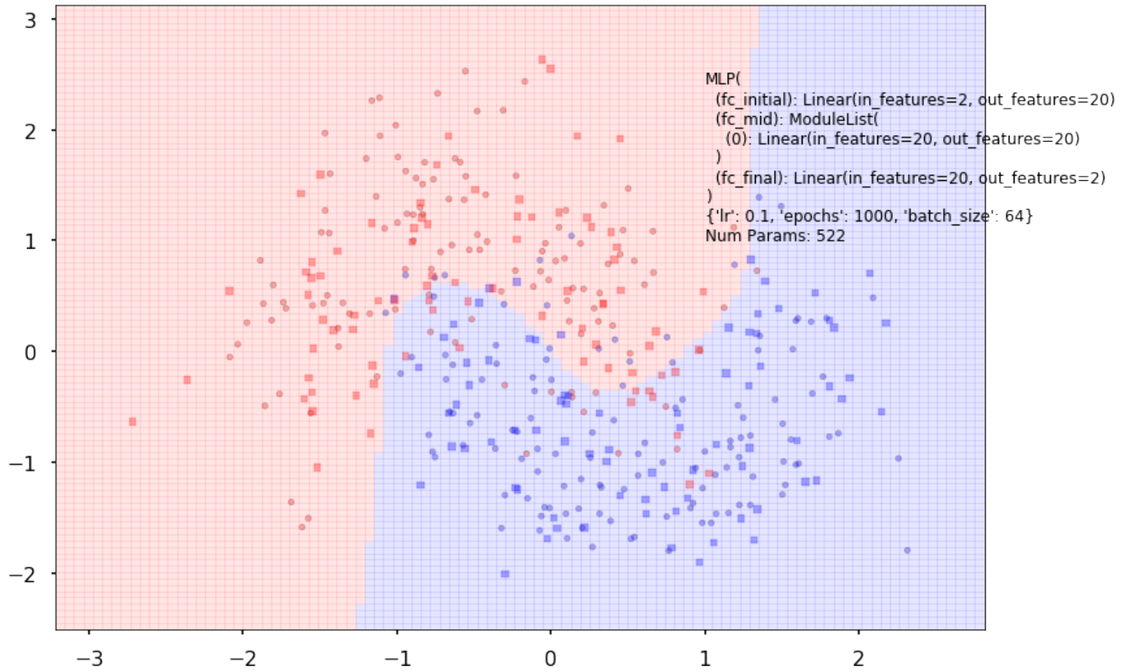


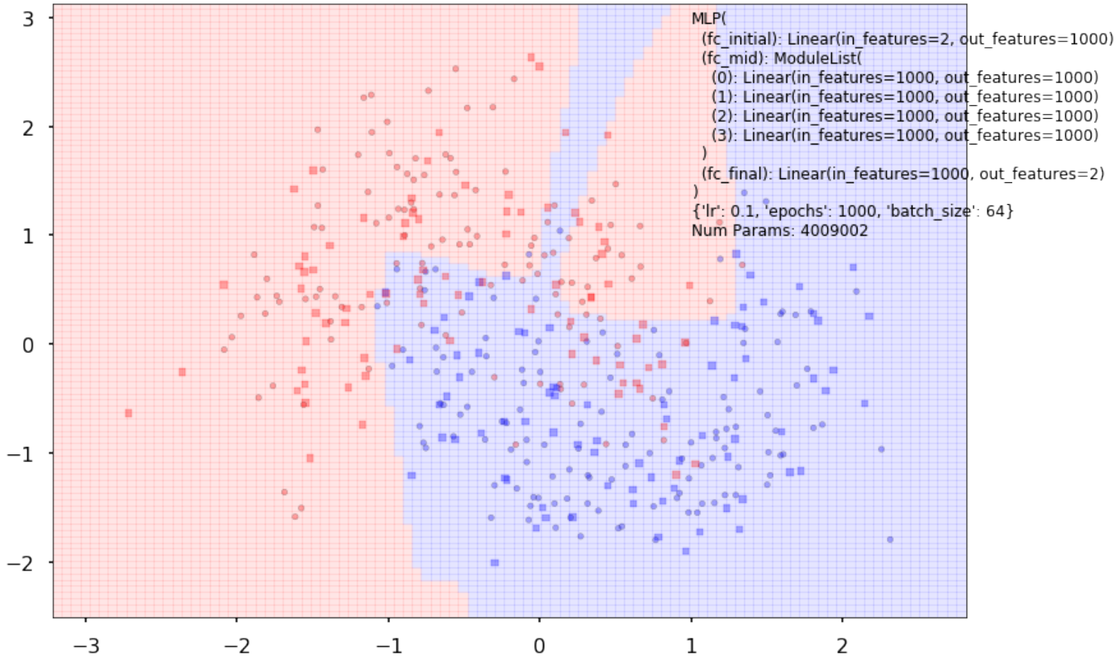
1 layer, 2 vs 10 neurons





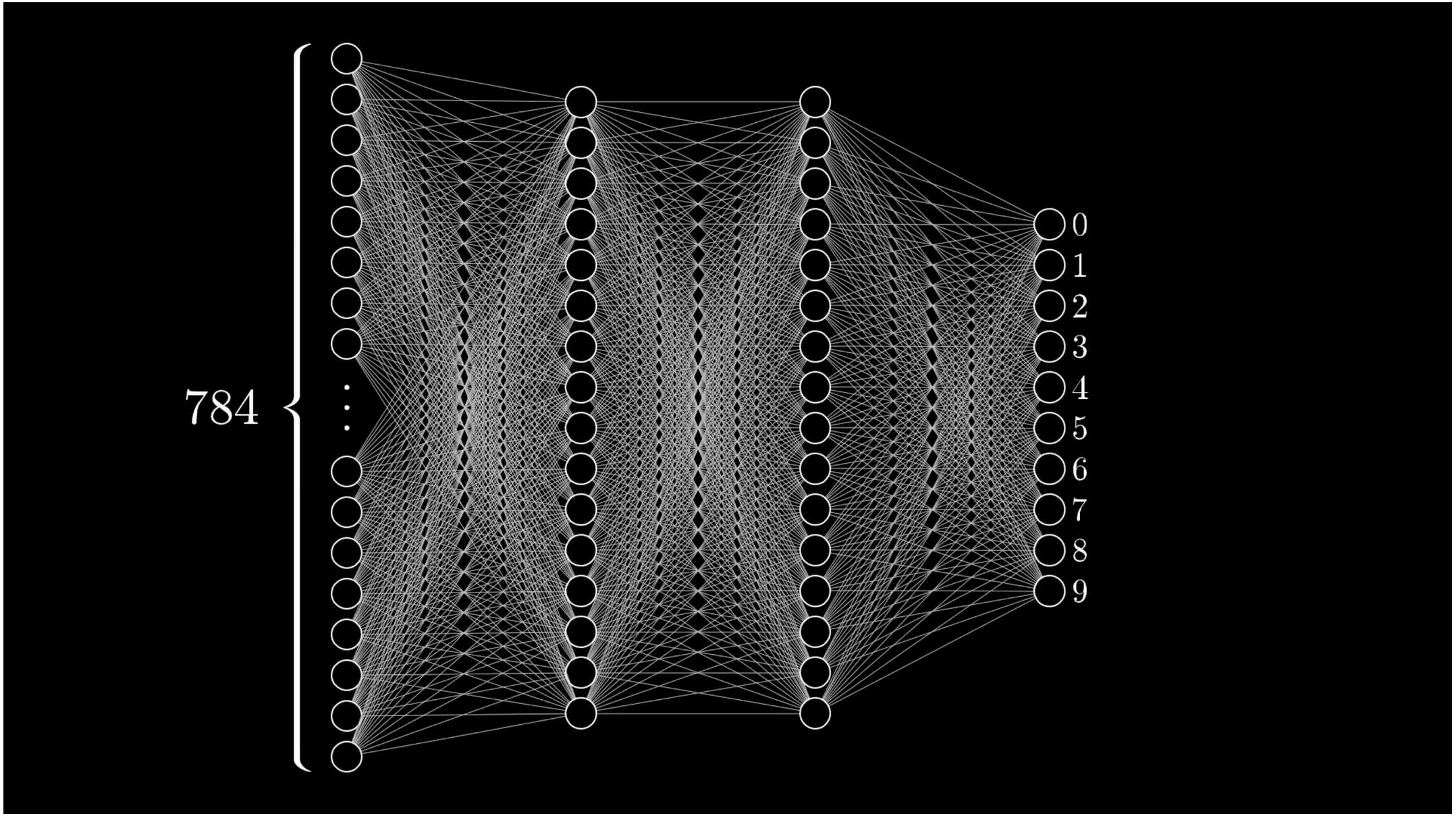
2 layers, 20 neurons vs 5 layers, 1000 neurons





How do we learn?

Univ.AI


784
0
1
2
3
4
5
6
7
8
9

Why does deep learning work?

1. Automatic differentiation

2. GPU

3. Learning Recursive Representations

Something like:

$$s(w_n . z_n + b_n) \text{ where } z_n = s(w_{n-1} . z_{n-1} + b_{n-1}) \text{ and}$$

$$s(w_{n+1} . z_{n+1} + b_{n+1}) \text{ where } z_{n+1} = s(w_n . z_n + b_n) \text{ and}$$

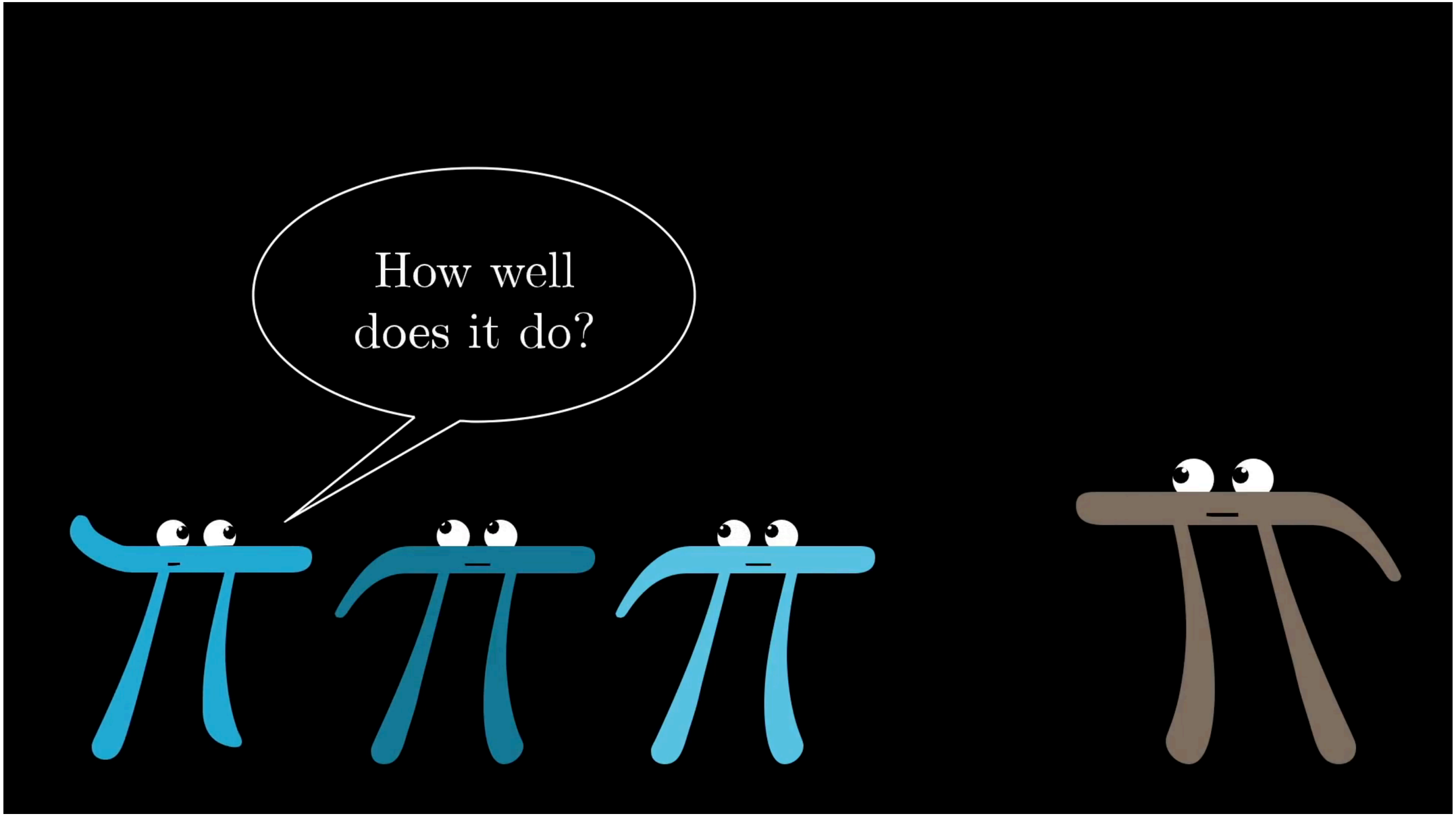
and so on.

How do we do digits?

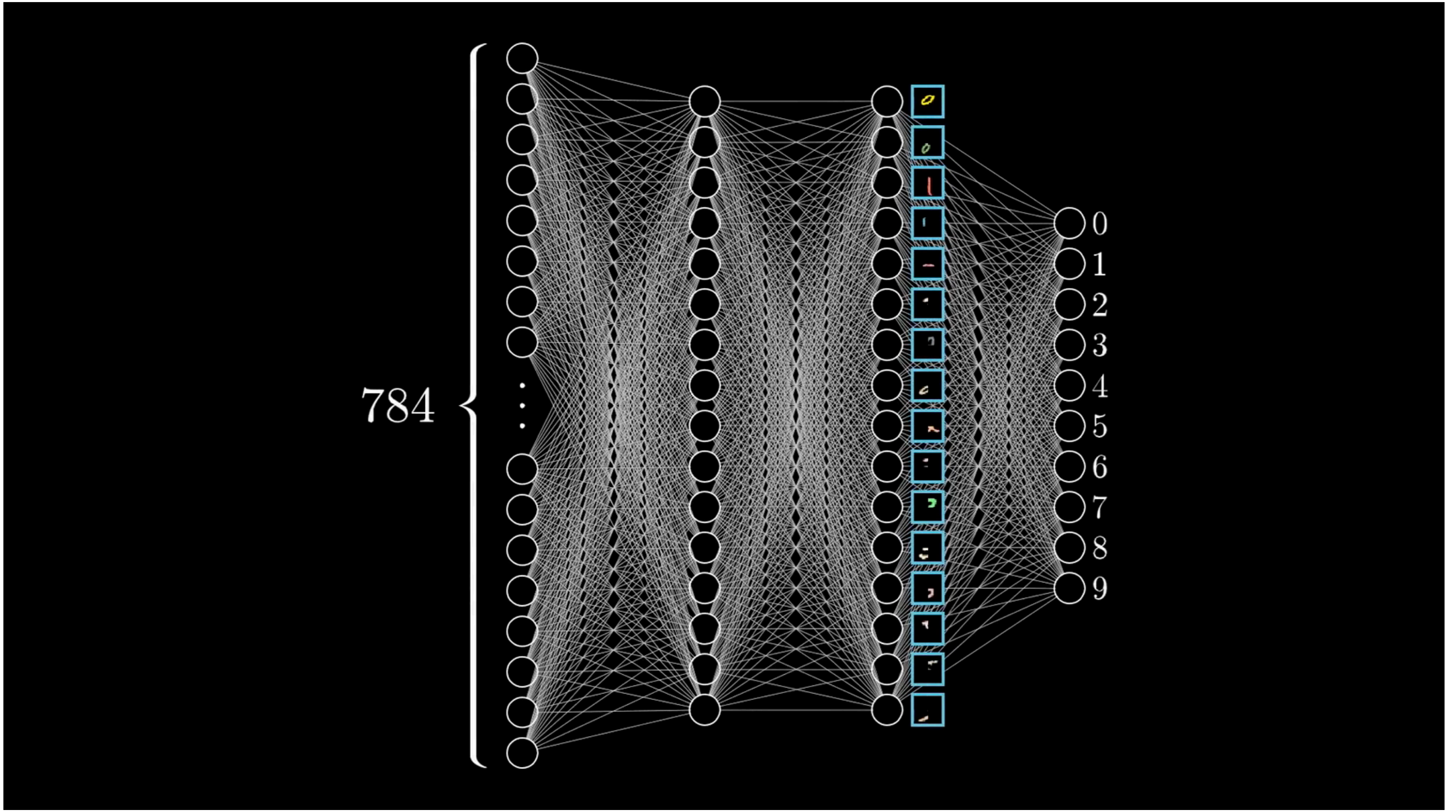
And How do we do?

Code in Keras

```
# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()
img_width = X_train.shape[1]
img_height = X_train.shape[2]

X_train = X_train.astype('float32')
X_train /= 255.
X_test = X_test.astype('float32')
X_test /= 255.

# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
labels = range(10)
num_classes = y_train.shape[1]

# create model
model = Sequential()
model.add(Flatten(input_shape=(img_width, img_height)))
model.add(Dense(config.hidden_nodes, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=config.optimizer,
              metrics=['accuracy'])
model.summary()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=config.epochs,
          callbacks=[WandbCallback(data_type="image", labels=labels)])
```

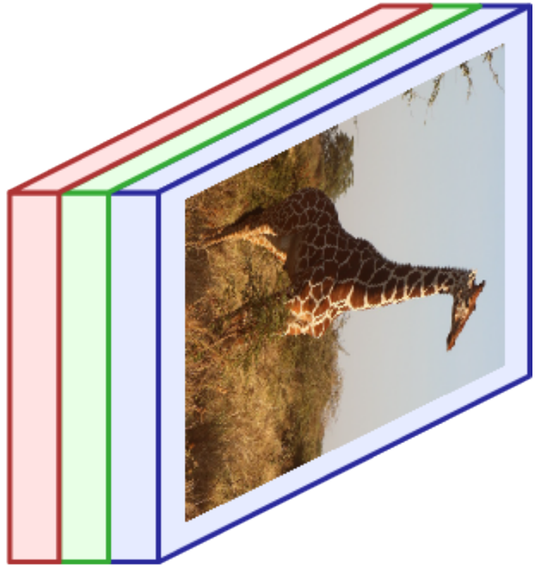

How well
does it do?

Where else can we go?


784
0
1
2
3
4
5
6
7
8
9

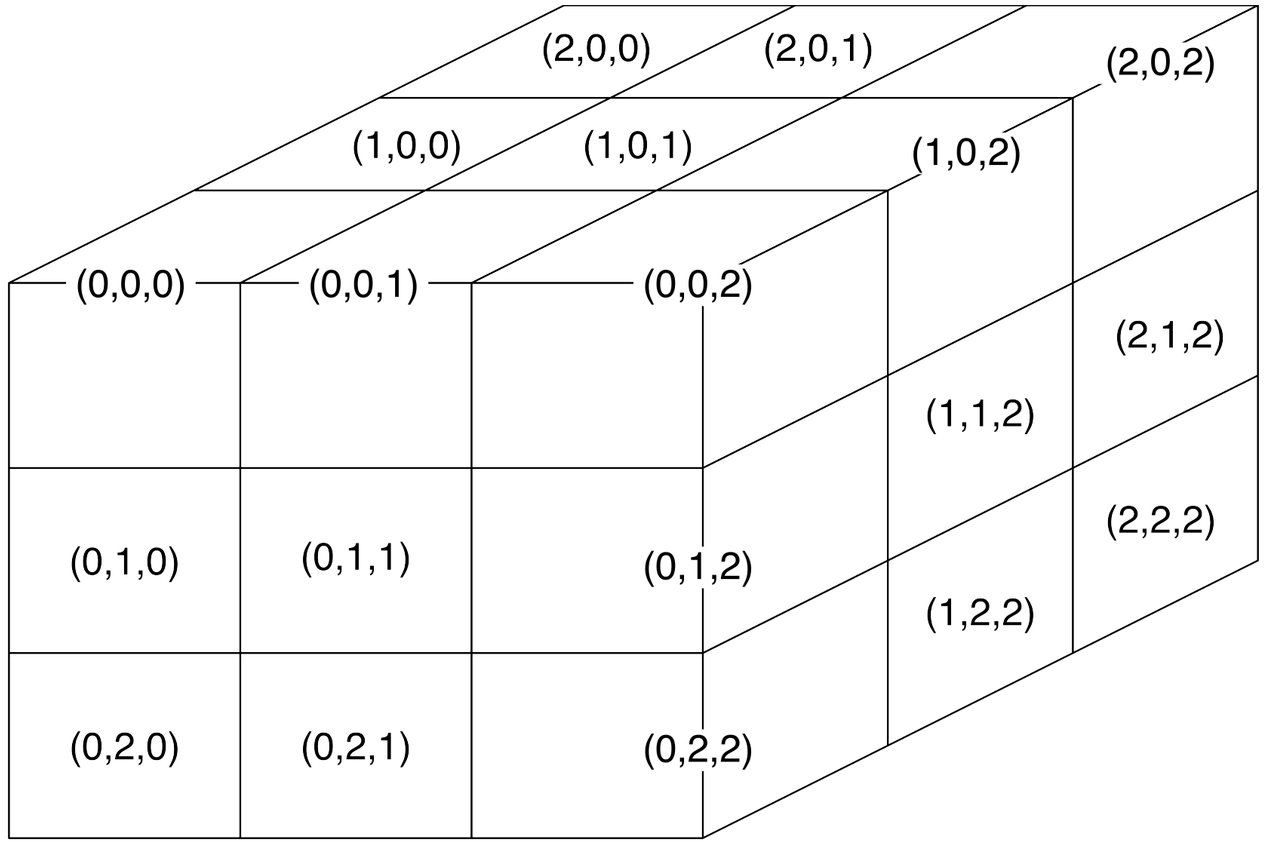
Images

(a)

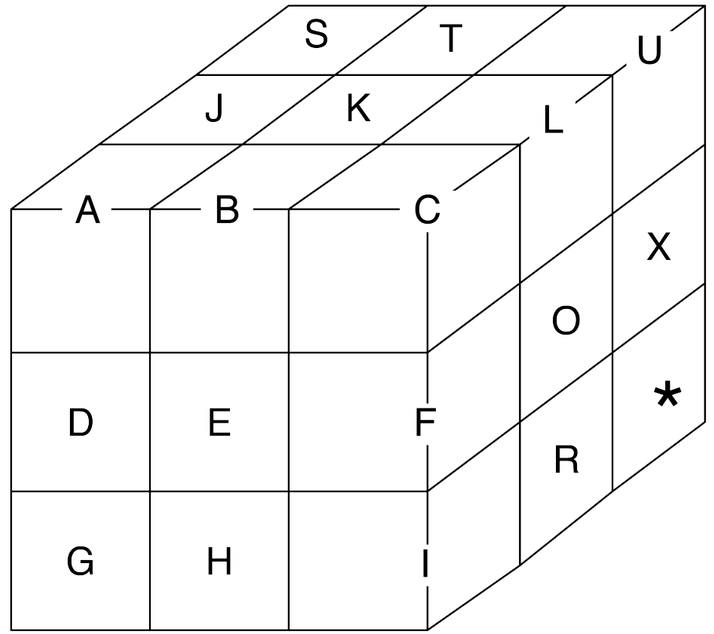


(b)

Channels first arrangement

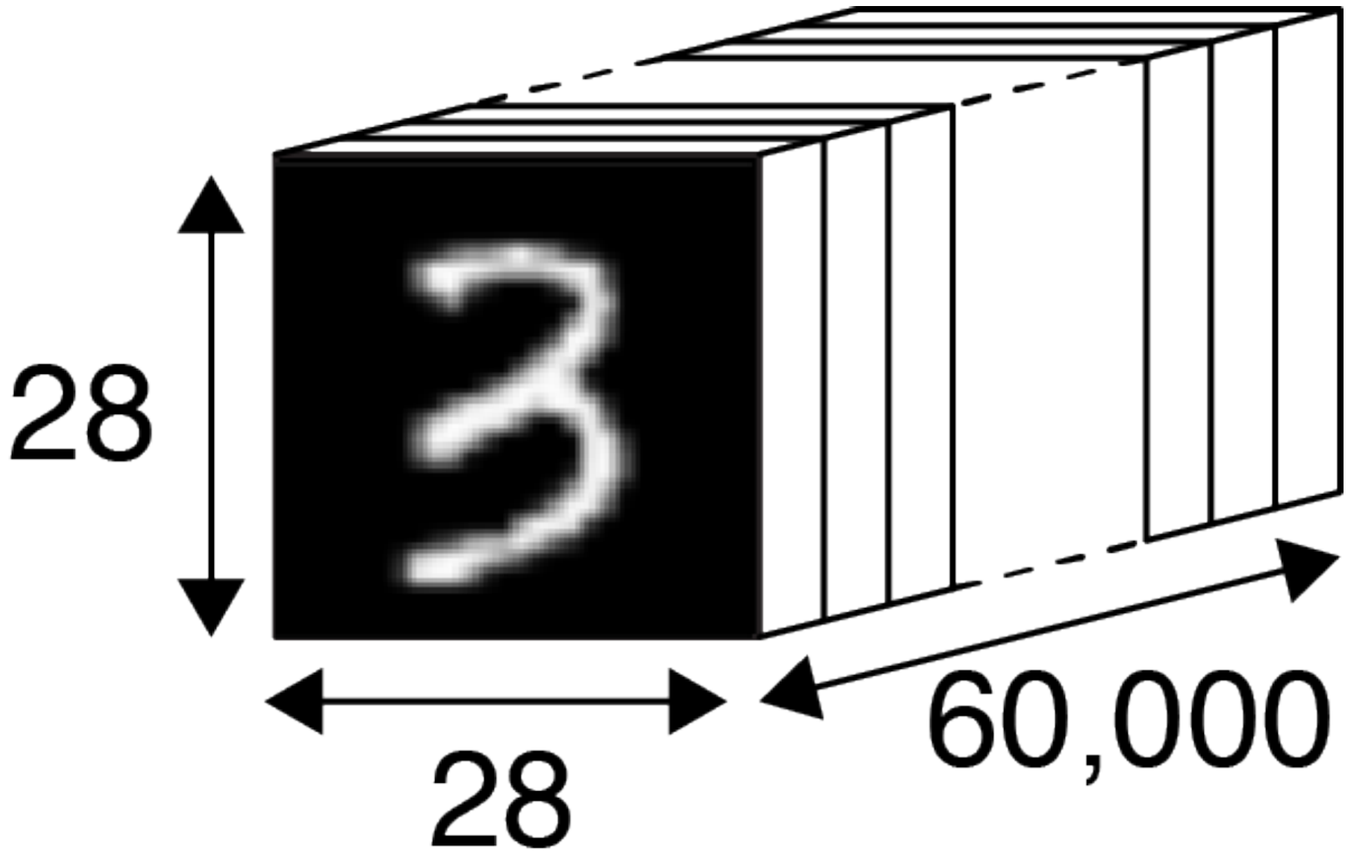


(a)

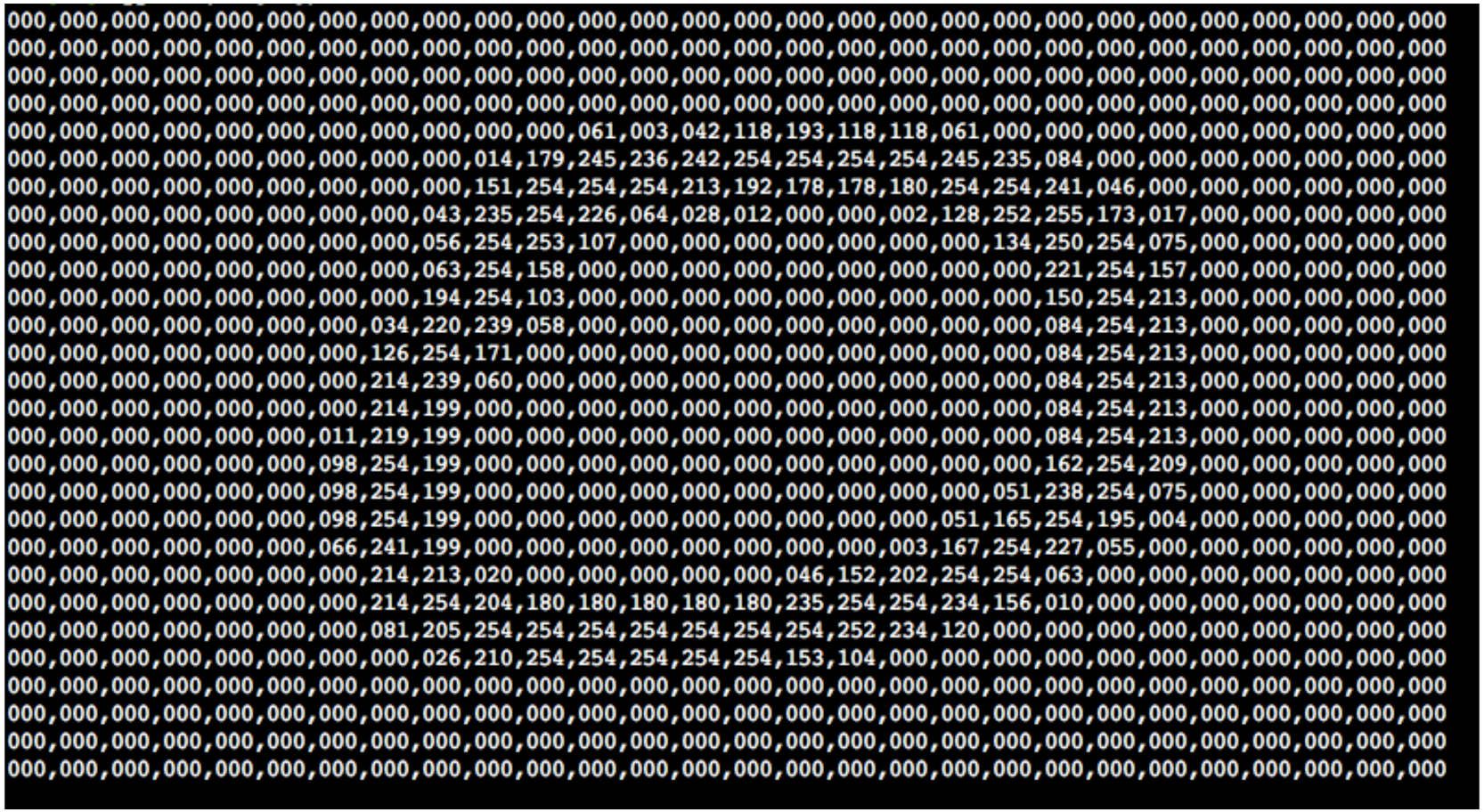


(b)

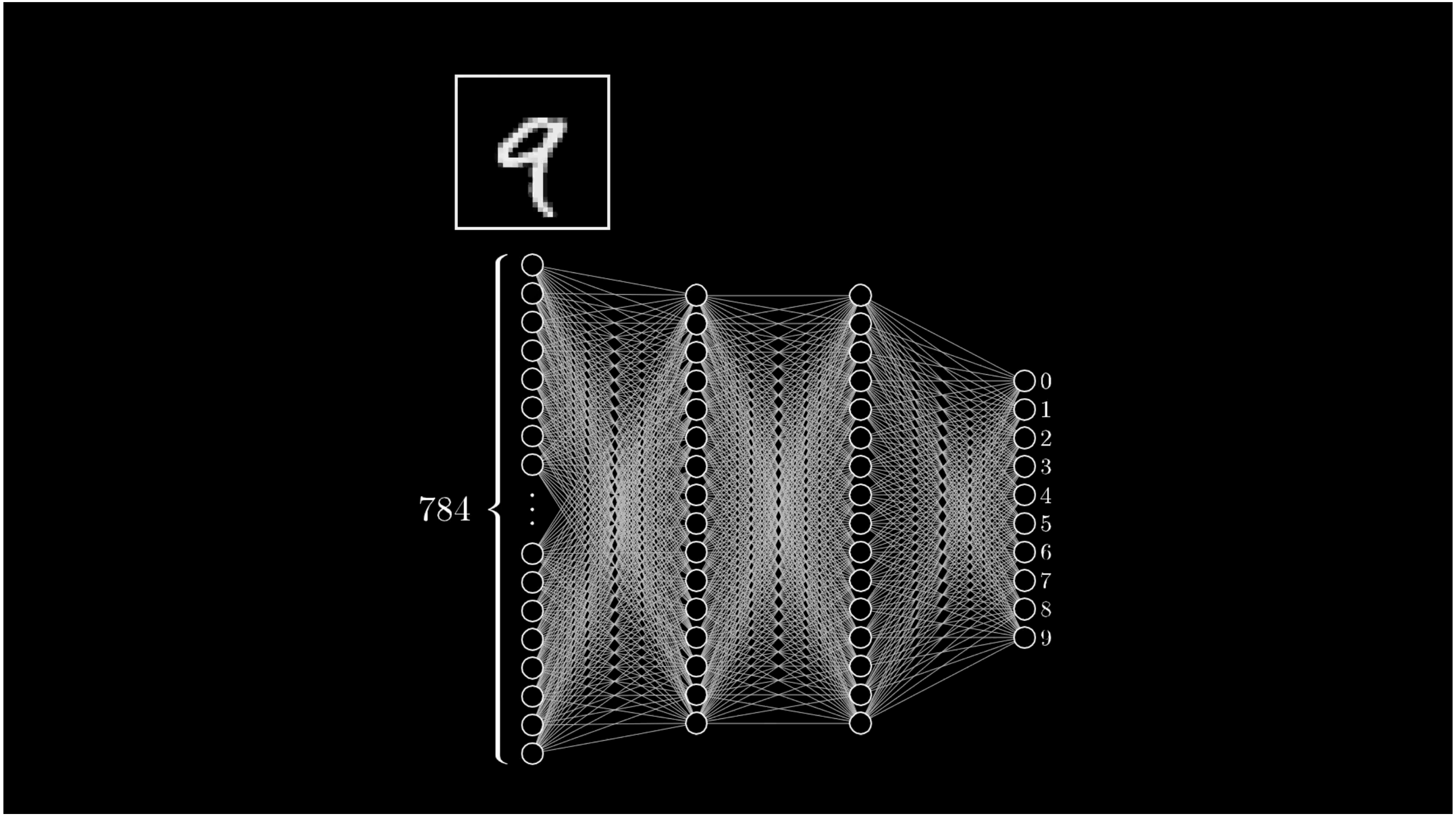
What about multiple images?



And a single image?



MLP's dont actually work how we want them to!

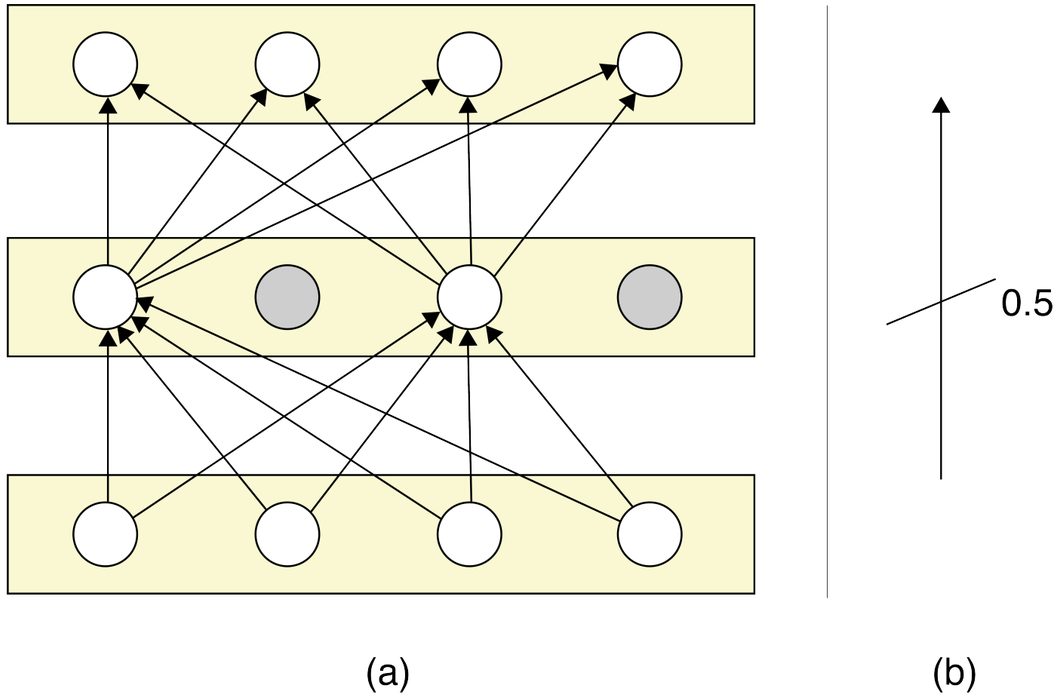

784
0
1
2
3
4
5
6
7
8
9

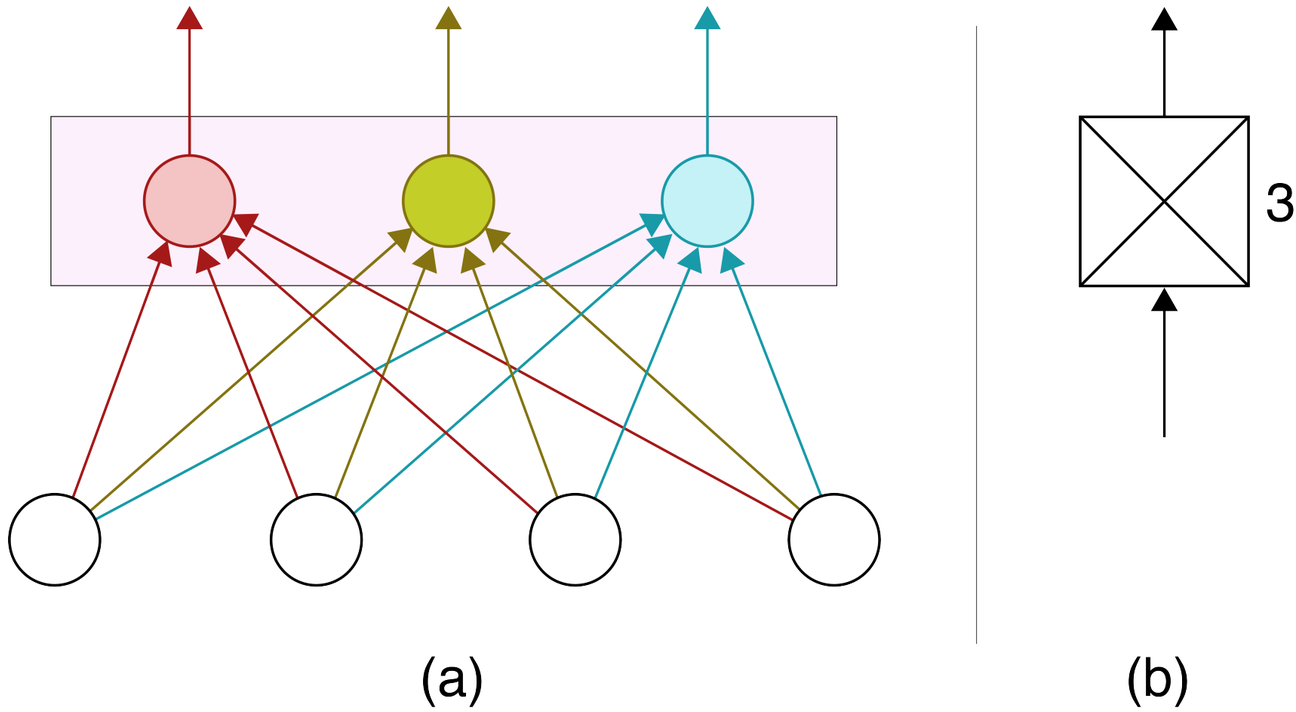
Convolutional Networks

- pay attention to the spatial locality of images
- this is done through the use of "filters"
- thus the representations learnt are spatial and bear a mapping to reality
- and are hierarchical..later layers learn features composed from the previous layers
- perhaps even approximating what the visual cortex does.

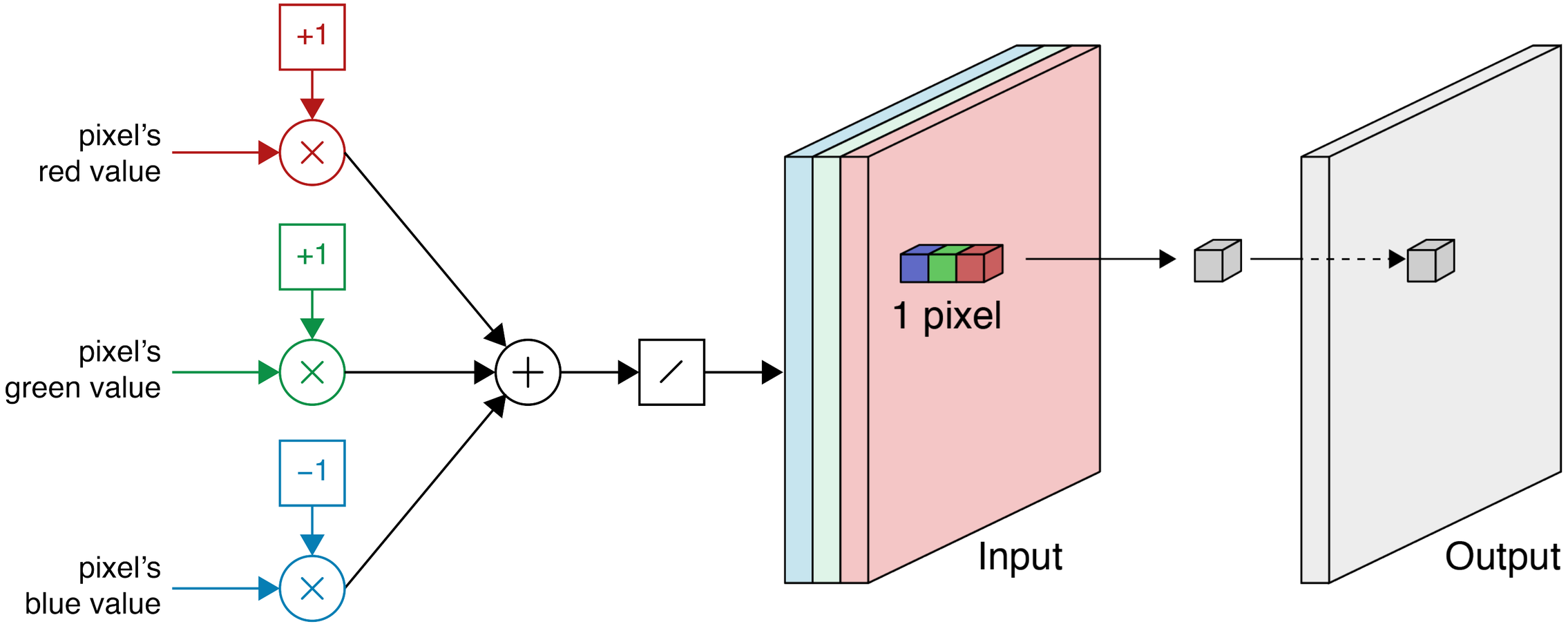
Convolutional Components

Fully Connected layers, 1-1 layers, regularization layers like dropout



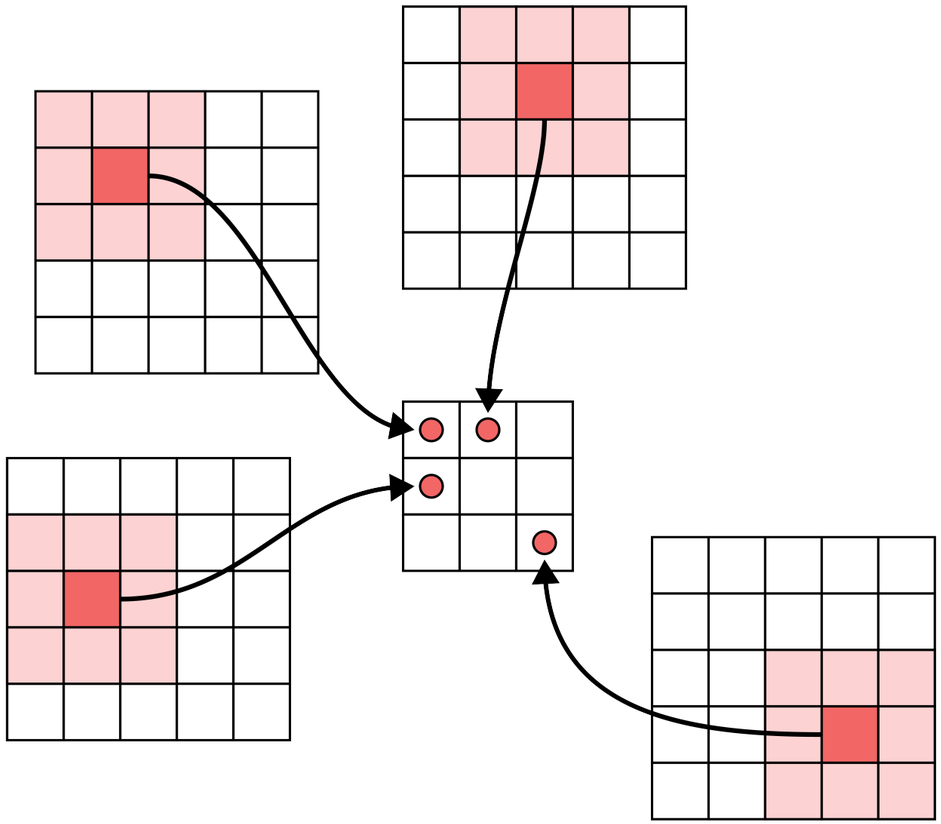


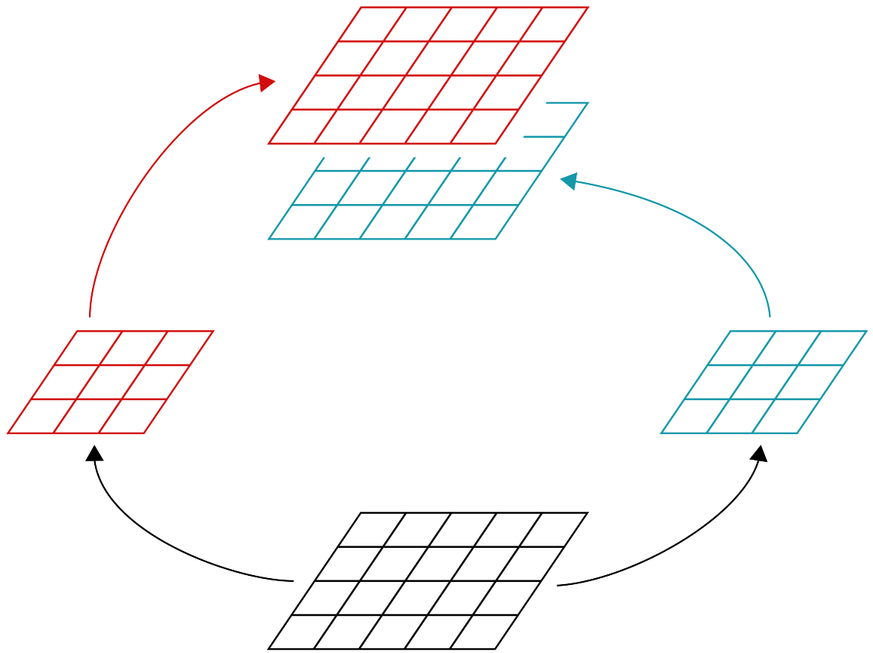
The idea of a filter: detecting yellow



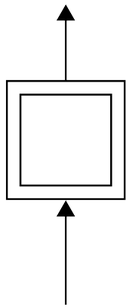
Convolution Layer

$$f((w, b) \cdot (x, 1))$$



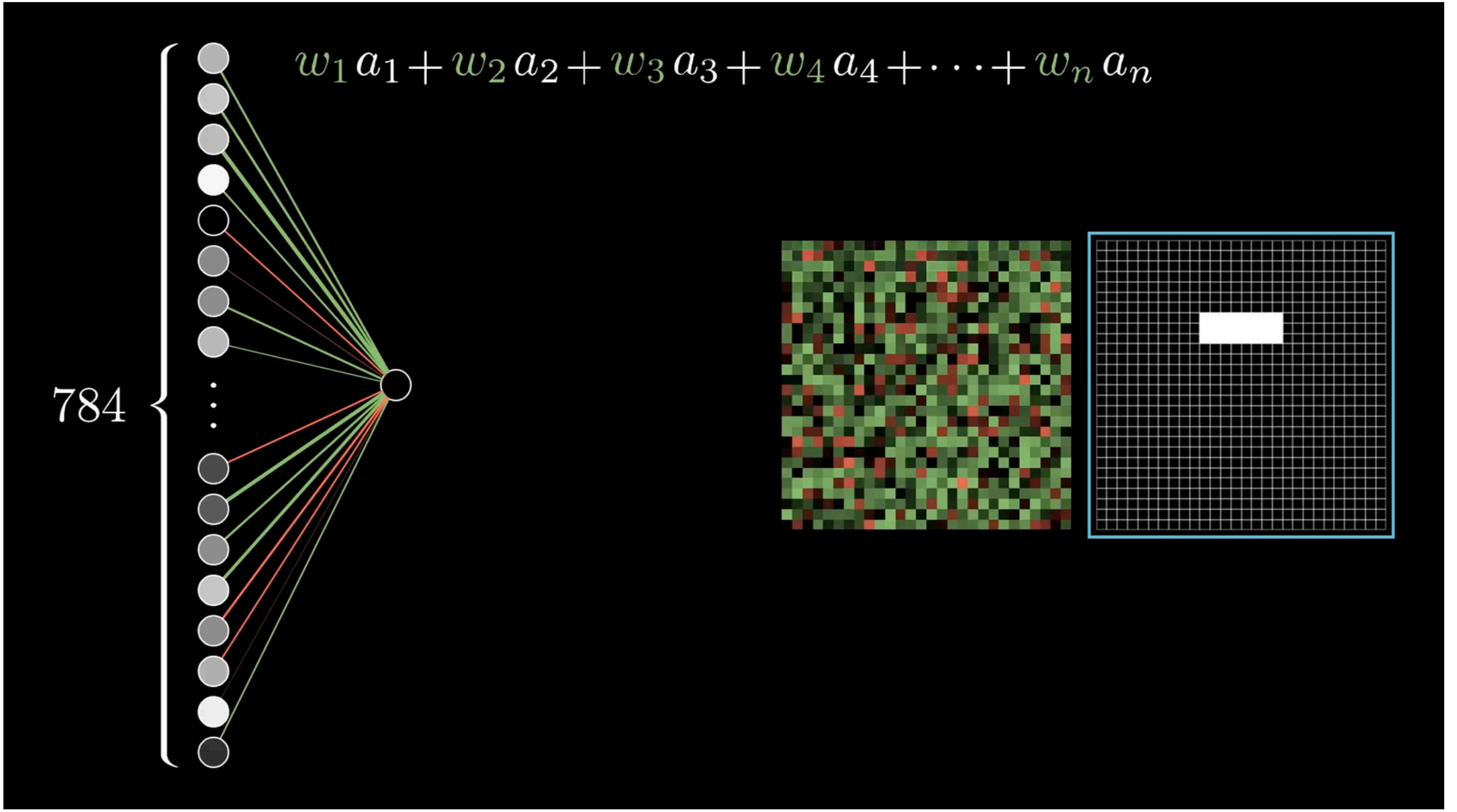


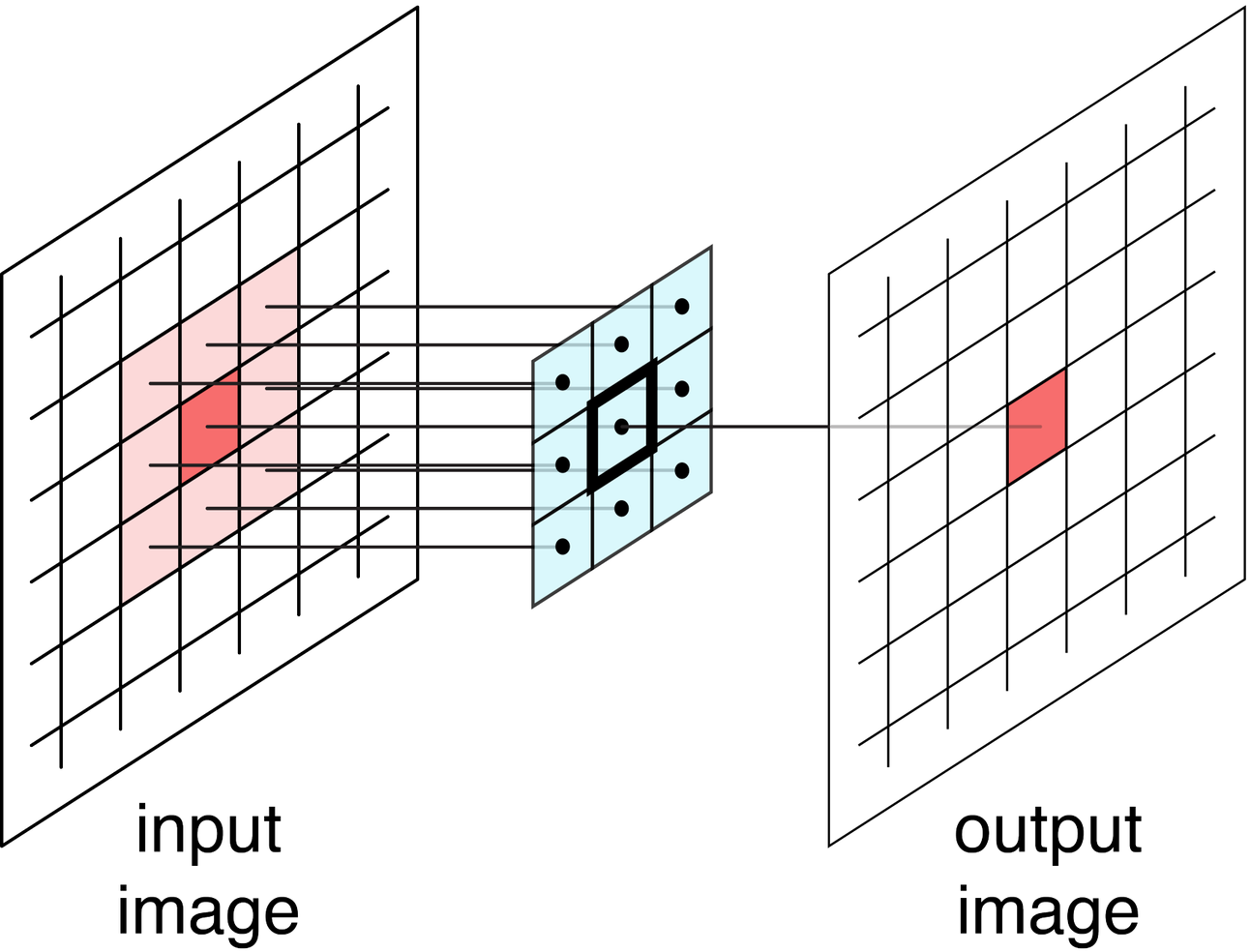
(a)



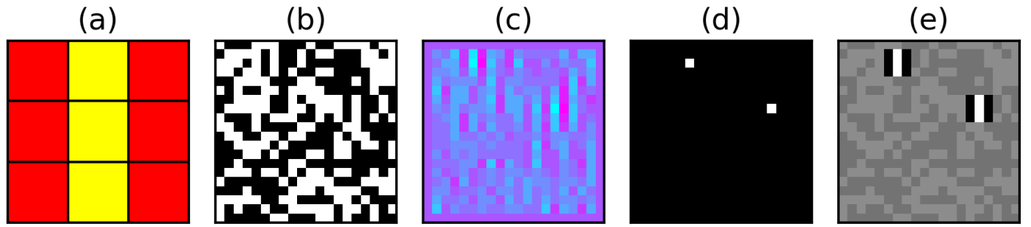
(b)

Convolution looks for patterns


$w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + \cdots + w_n a_n$
784

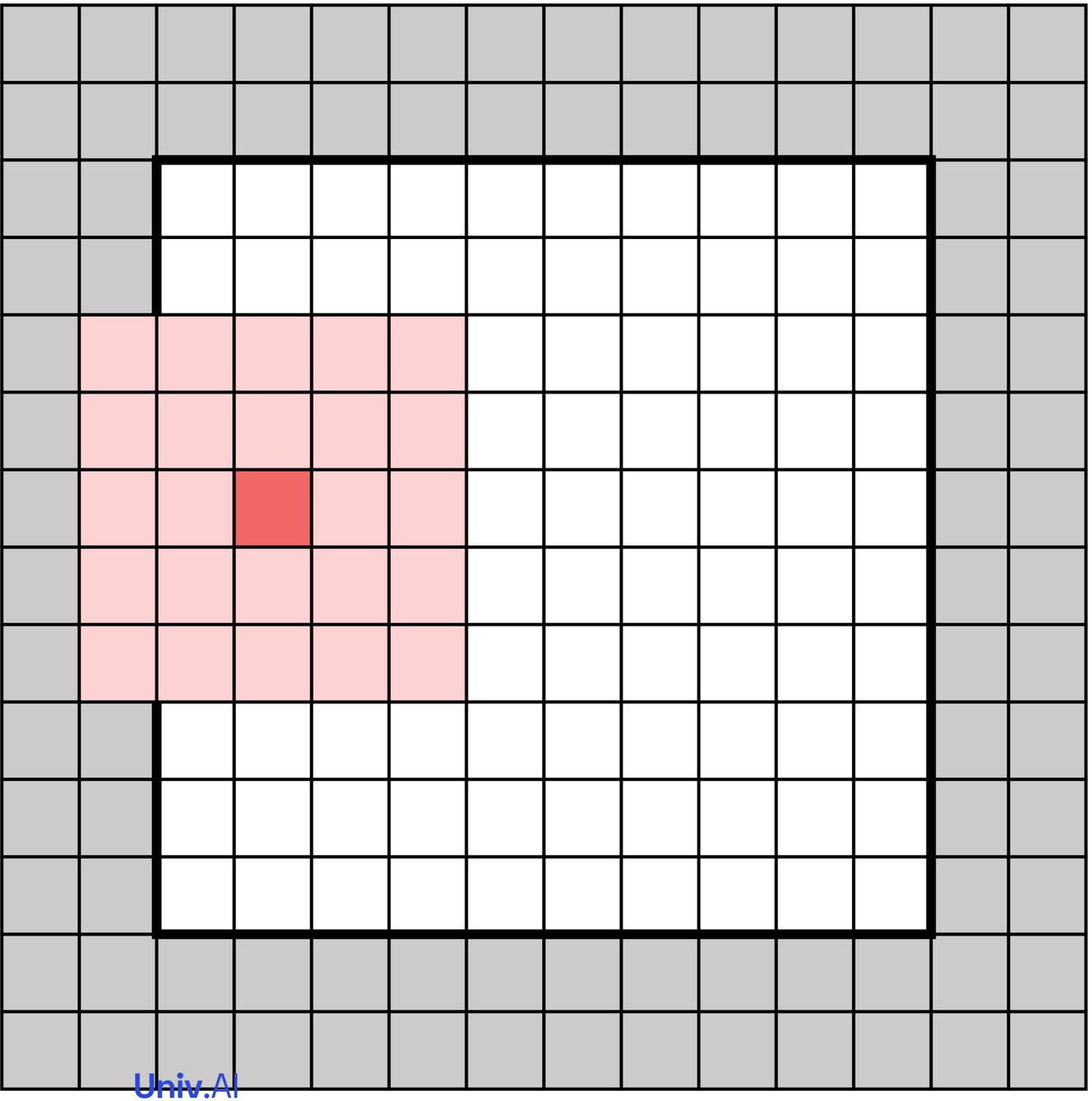


Movethe filter over the original image and produce a new one

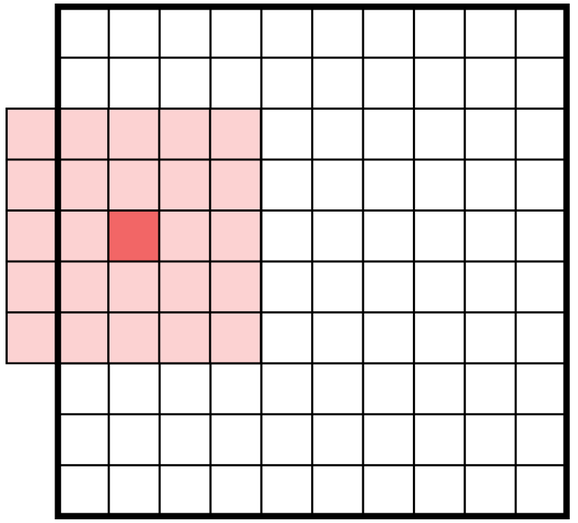


-1	1	-1	×	1	1	0	=	-1	1	0	⇒	– 3 + 1 = -2		
-1	1	-1		0	0	0		0	0	0				
-1	1	-1		1	0	1		-1	0	-1				

-1	1	-1	×	0	1	0	=	0	1	0	⇒	3		
-1	1	-1		0	1	0		0	1	0				
-1	1	-1		0	1	0		0	1	0				

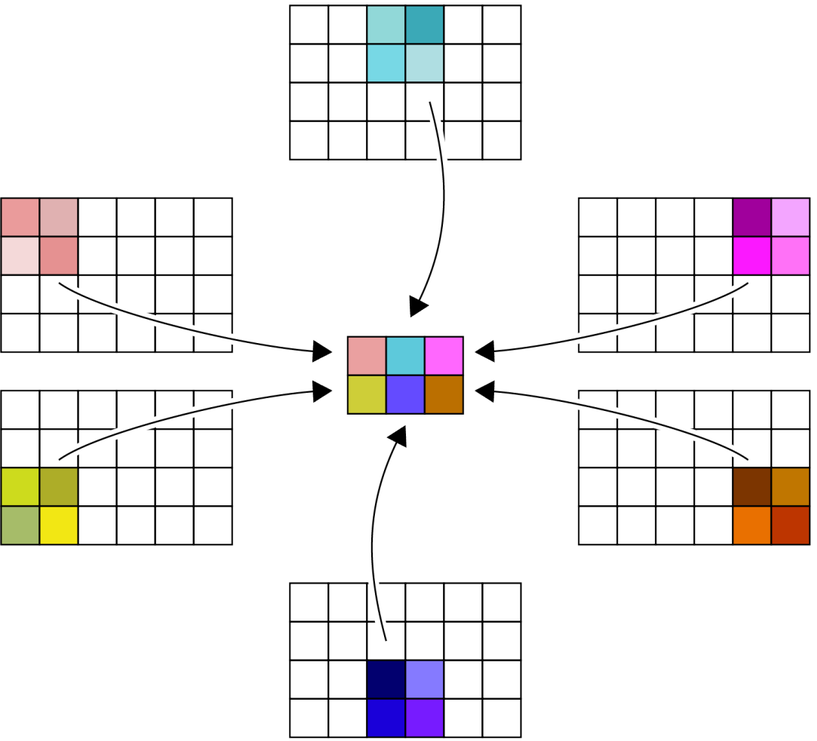


Padding

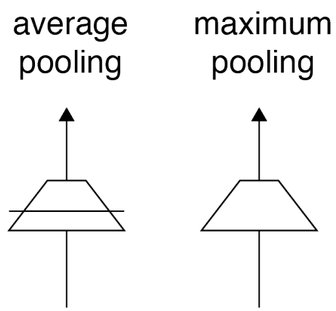


- Image size decrease by a convolution is called a "valid" convolution.
- Keeping the same size by 0-padding is called a "same" convolution

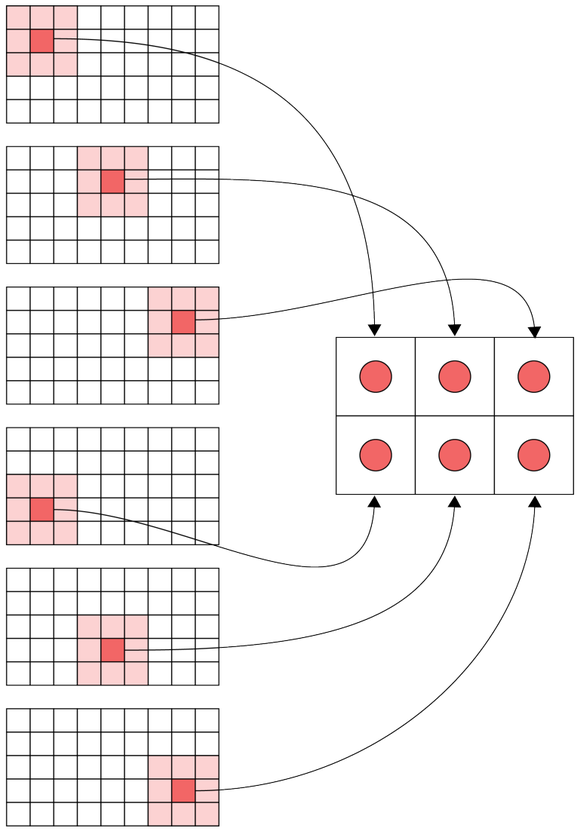
Downsampling: pooling, striding



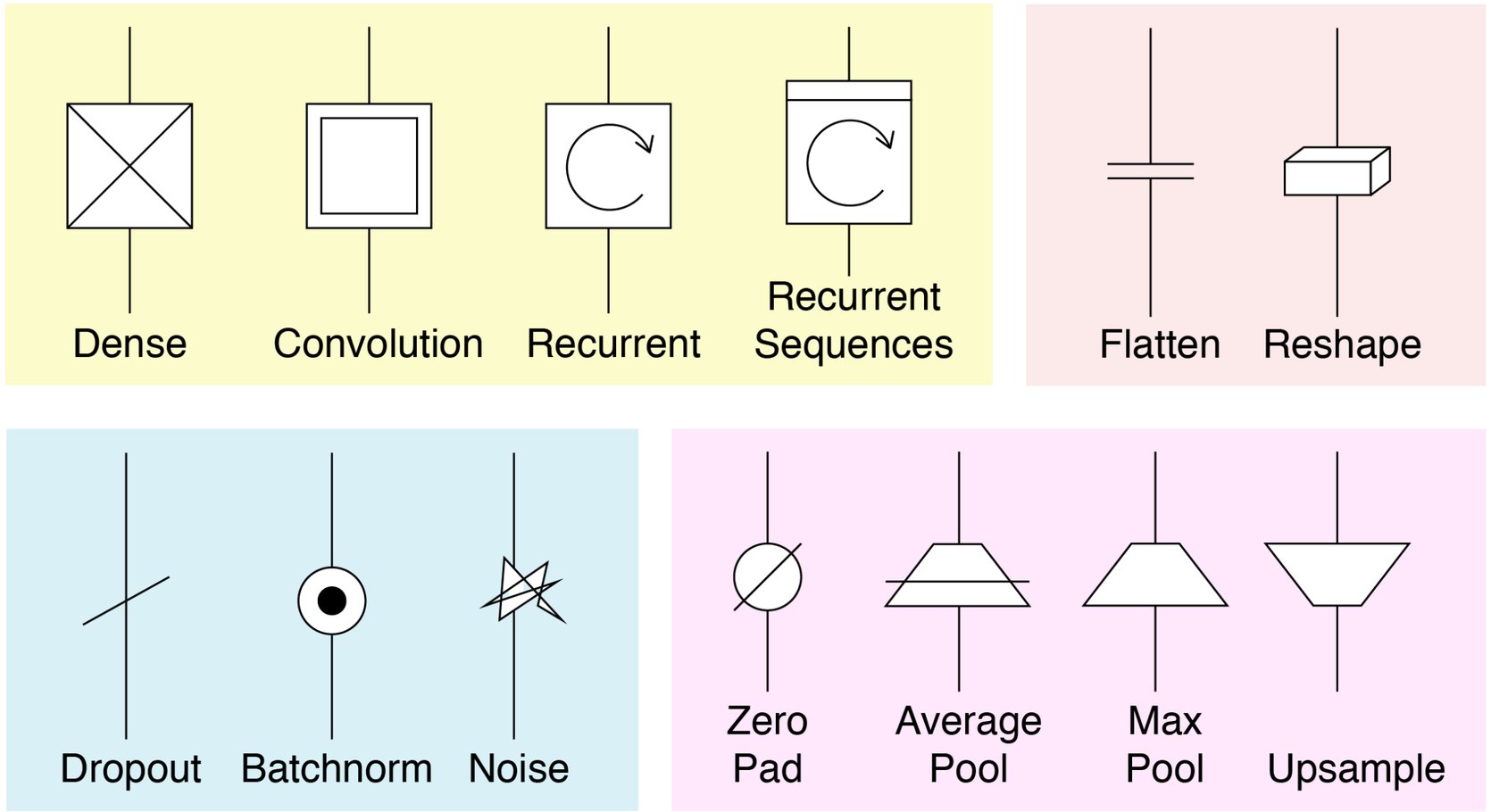
(a)



(b)

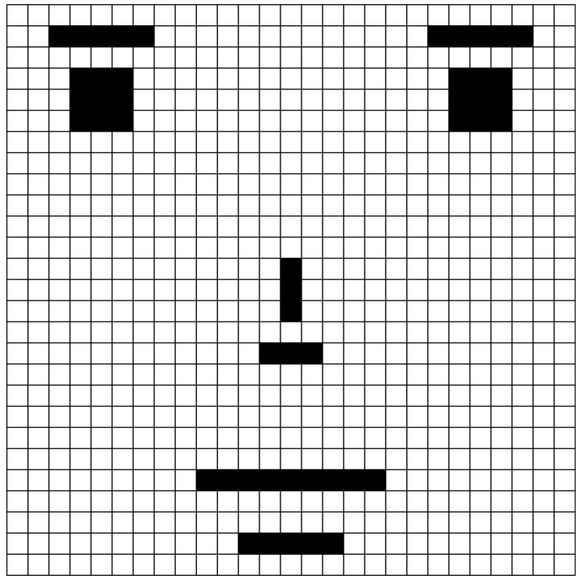


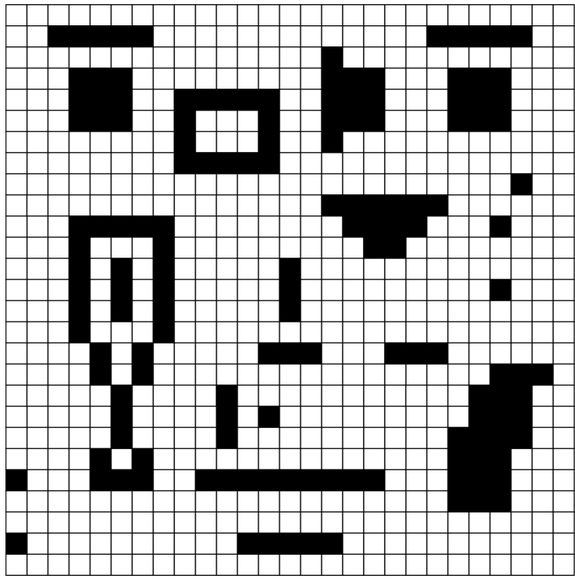
Layer types schematic

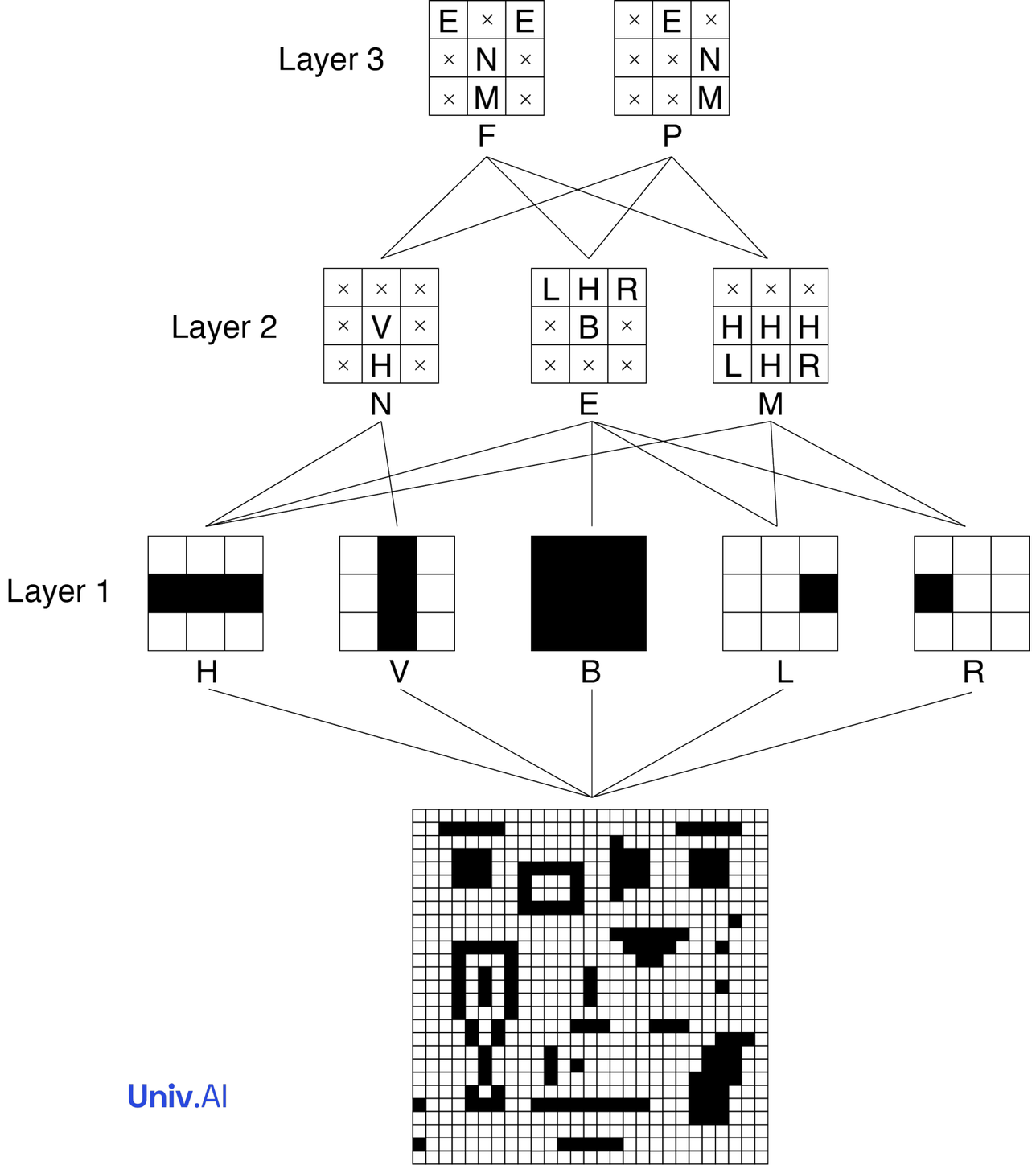


Hierarchical Filters

- do we then need to know every pattern we can find? NO! We learn the weights.
- now we do this hierarchically, with each filter at the next layer
- we hope to learn representations made up from smaller scale representations and so on
- here is an example: find the LHS face in the RHS image...



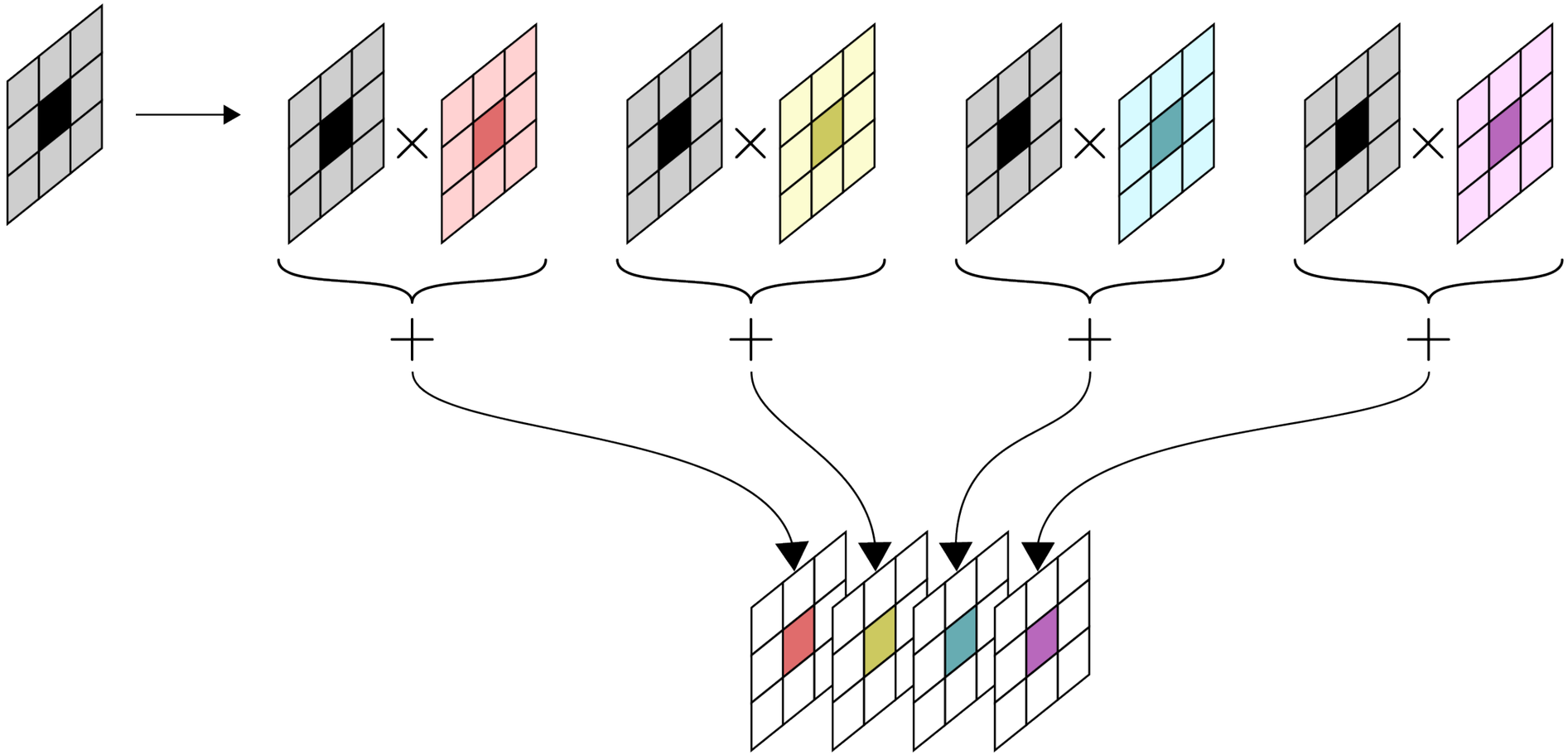




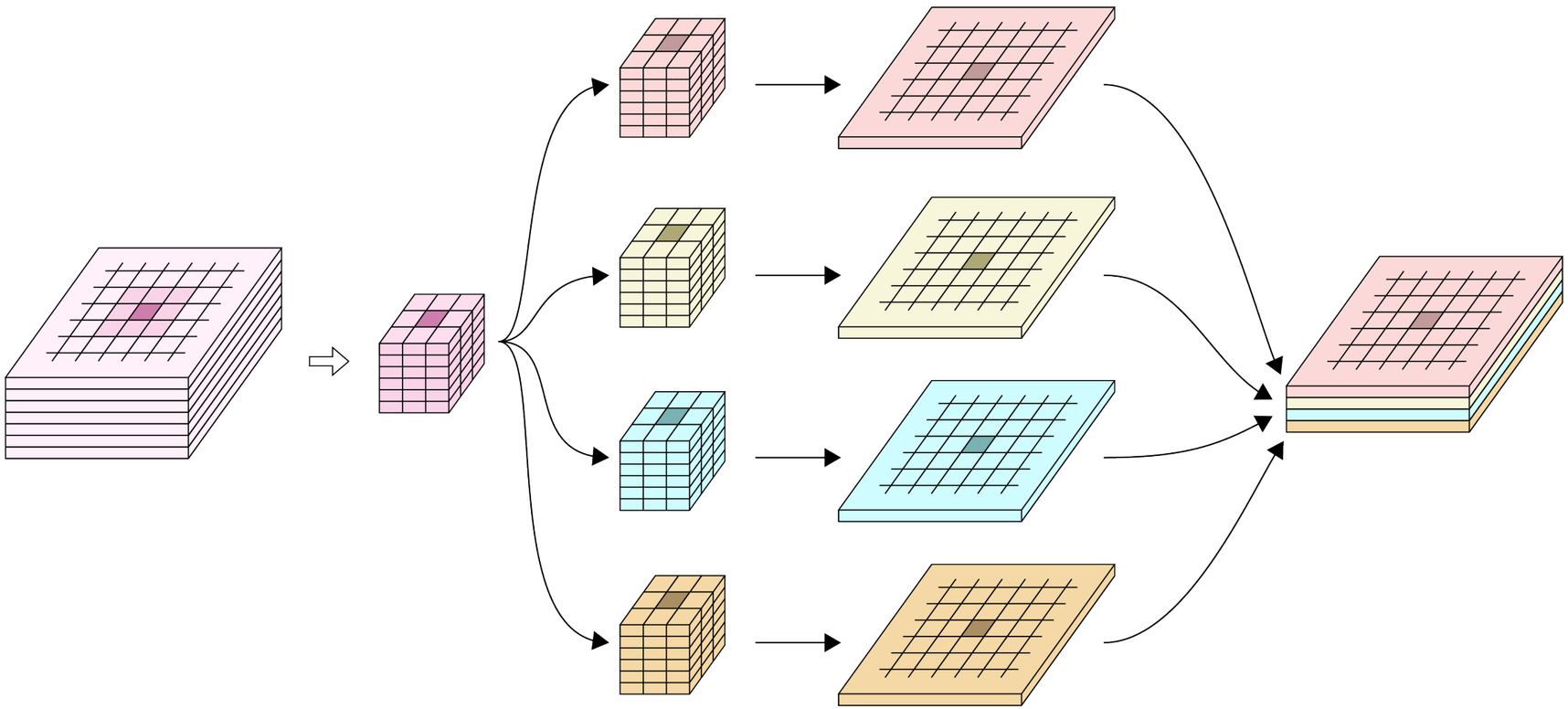
Strategy

- Move layer 1 filters around
- max pool 27x27 to 9x9
- x means dont care about value
- now apply second level filter to 9x9 image
- max pool again to 3x3 image
- apply level 3 filters and see if we activate

How Channels work: each color a different feature



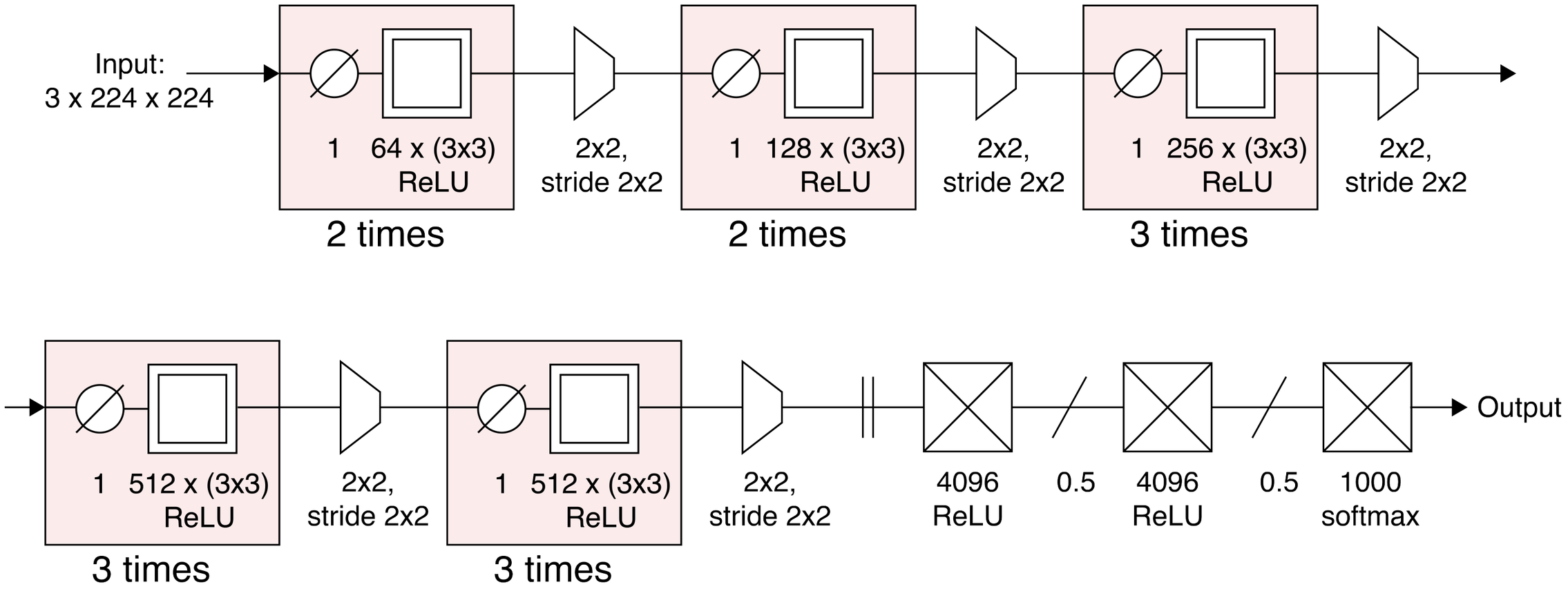
Channel Arithmetic



input is (say) 26x26x6, so filters MUST have 6 channels and we have 4 new featuremaps: `Conv2D(4,(3,3))`

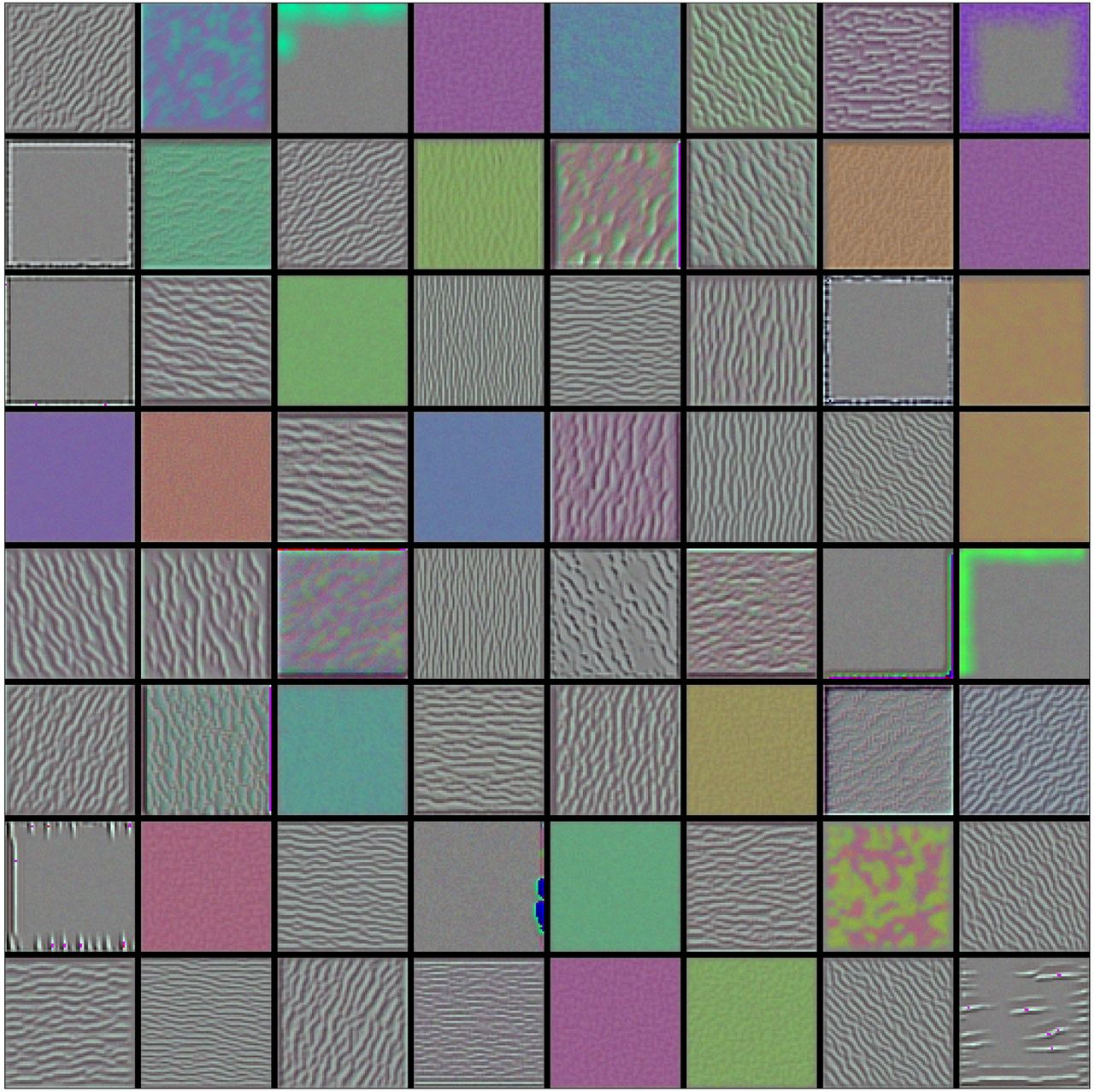
```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28
input_shape = (28, 28)

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'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

VGG16

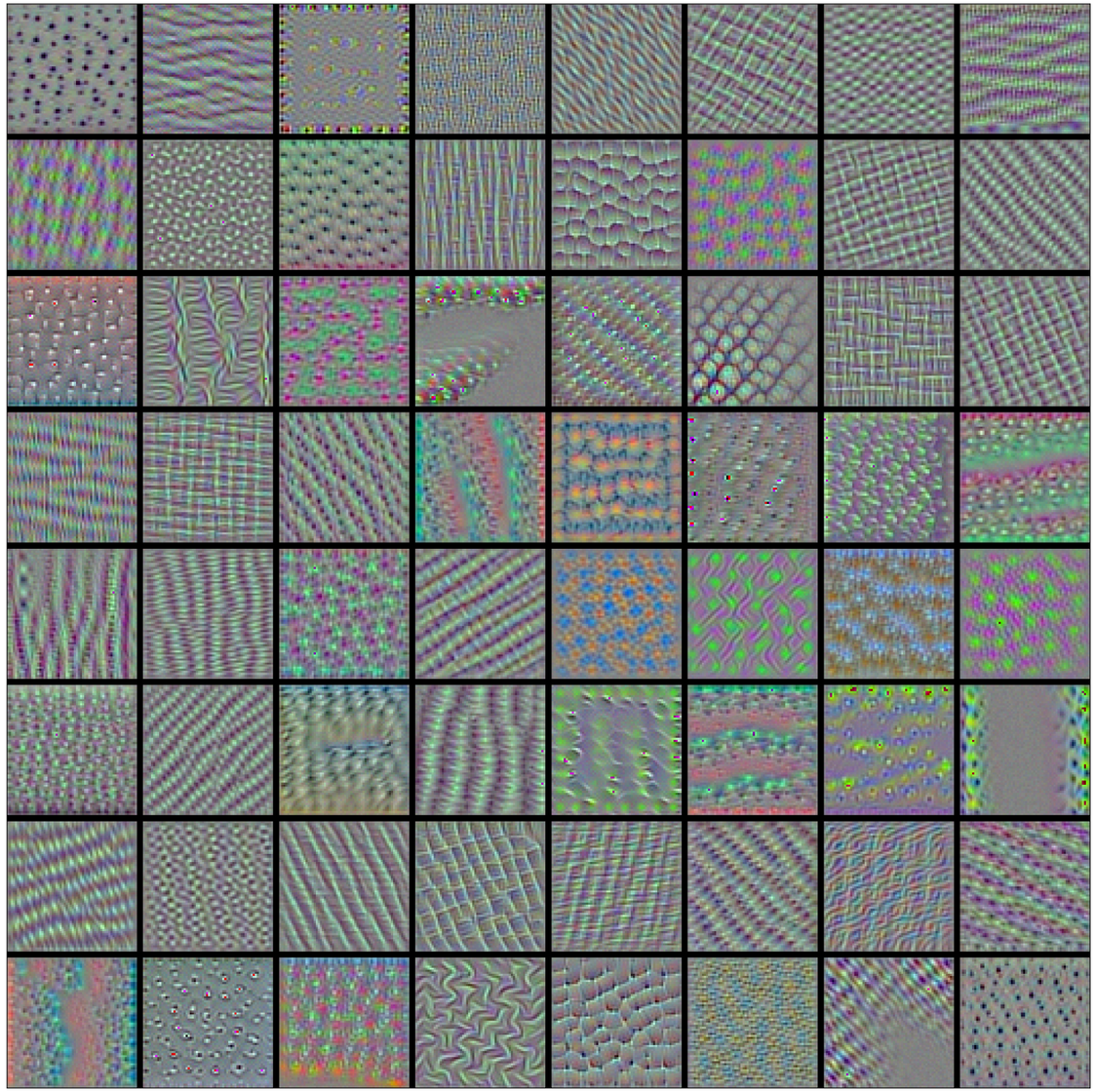

Input:
3 x 224 x 224
1 64 x (3x3)
ReLU
2 times
2x2,
stride 2x2
1 128 x (3x3)
ReLU
2 times
2x2,
stride 2x2
1 256 x (3x3)
ReLU
3 times
2x2,
stride 2x2
1 512 x (3x3)
ReLU
3 times
2x2,
stride 2x2
1 512 x (3x3)
ReLU
3 times
2x2,
stride 2x2
4096
ReLU
0.5
4096
ReLU
0.5
1000
softmax
Output

How VGG16 learns

Filters from VGG16 layer block1_conv2



Filters from VGG16 layer block3_conv1



Filters from VGG16 layer block4_conv1