

Deep Learning Topics: Special Connections, and Keras Model API

Paul Rodriguez, PhD
(SDSC)
08/10/2023

Outline

- **Part I**

Gate connection idea

Skip and Residual connections

Keras Model API

Exercise MNIST Autoencoding

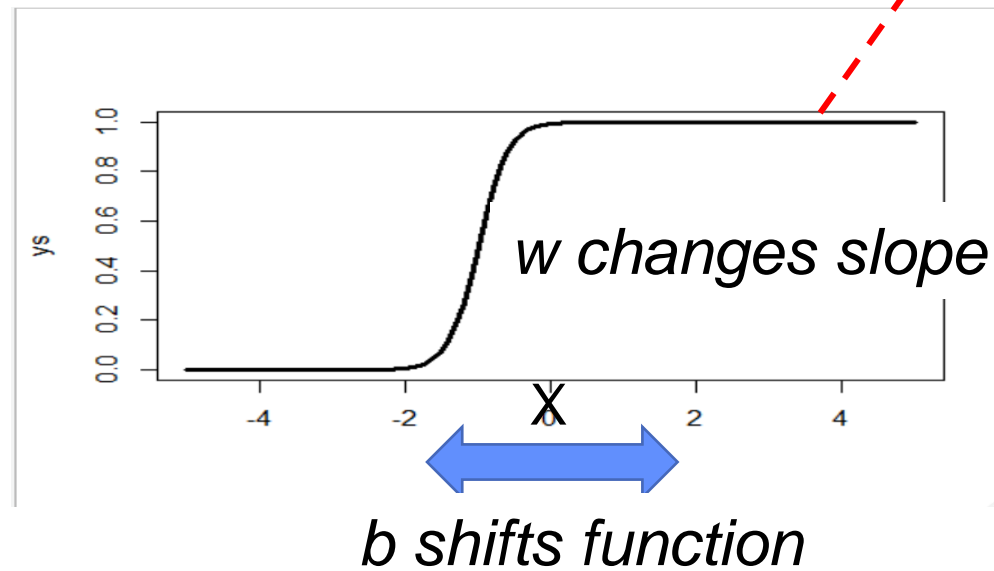
- **Part II**

Attention Head and Transformers,

Keras-NLP

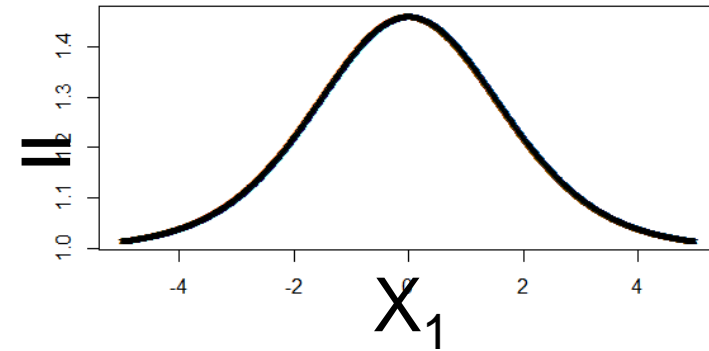
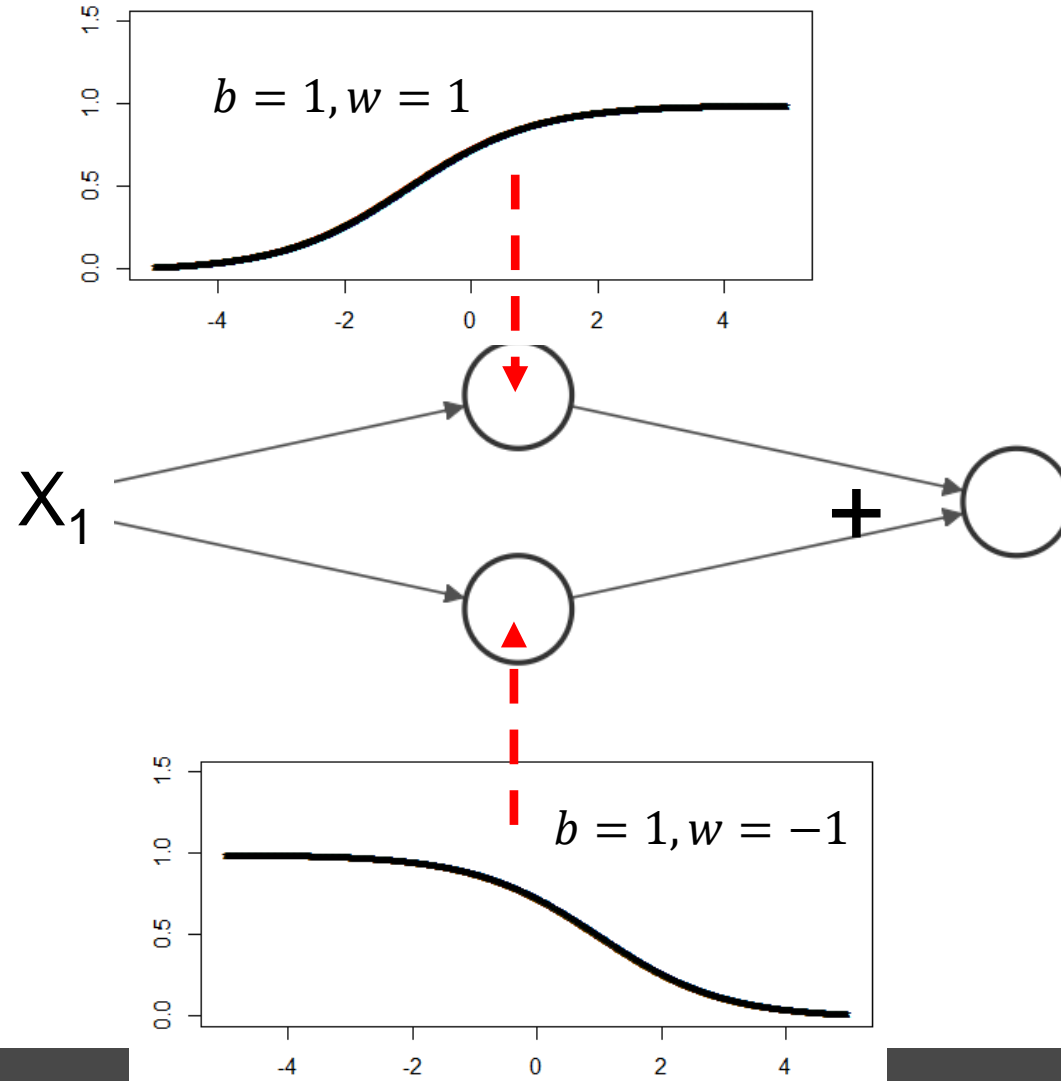
Recall: the logistic unit

$$f(x) = \frac{1}{1 + \exp(-(b + wx))} \iff \begin{array}{c} b \\ \swarrow \\ x \xrightarrow{w} \bigcirc \rightarrow \text{output value} \end{array}$$

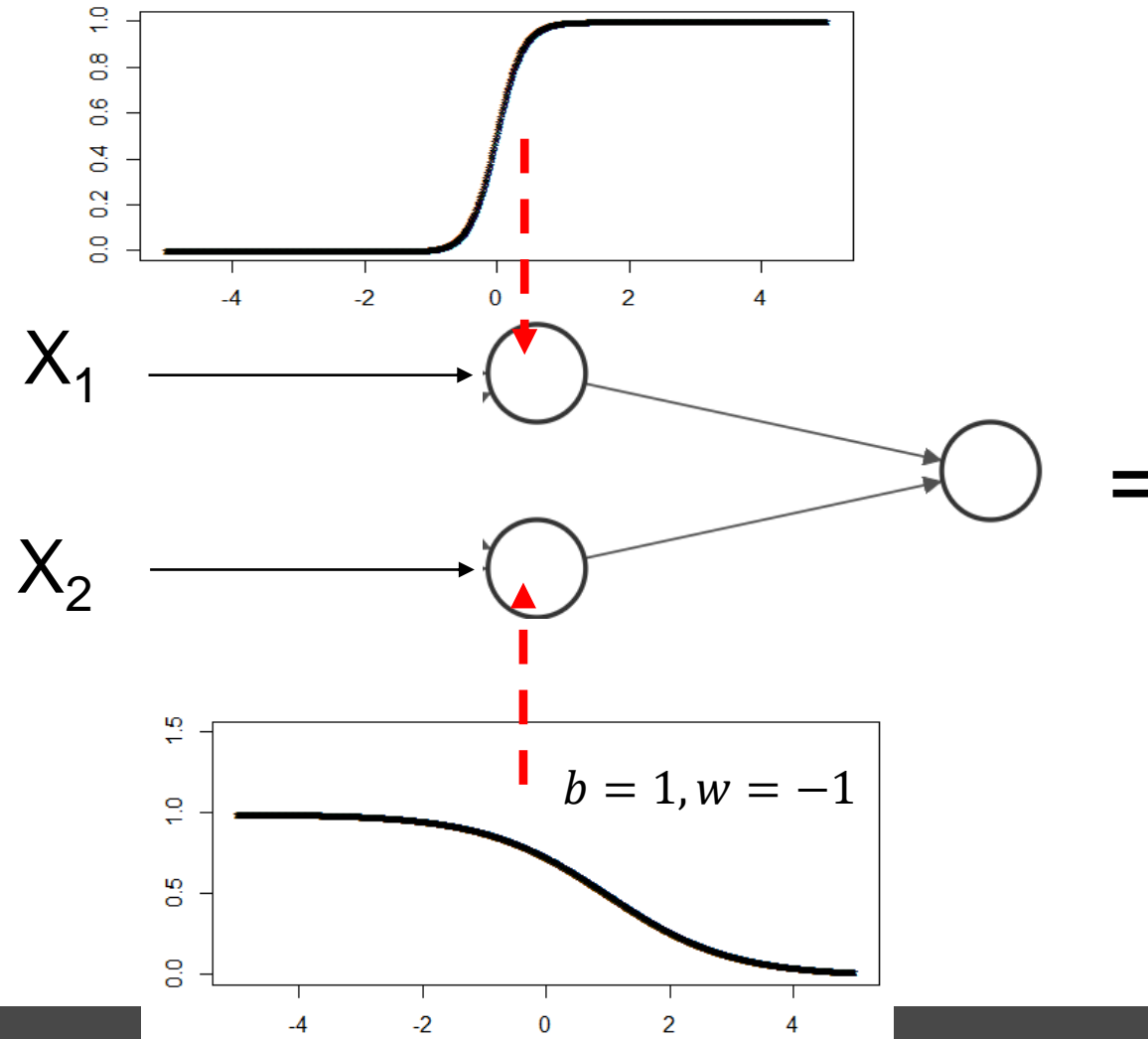


Example: 1 input into 2 logistic units with these activations

If you add these 2 units into a final output unit what would the output function look like?

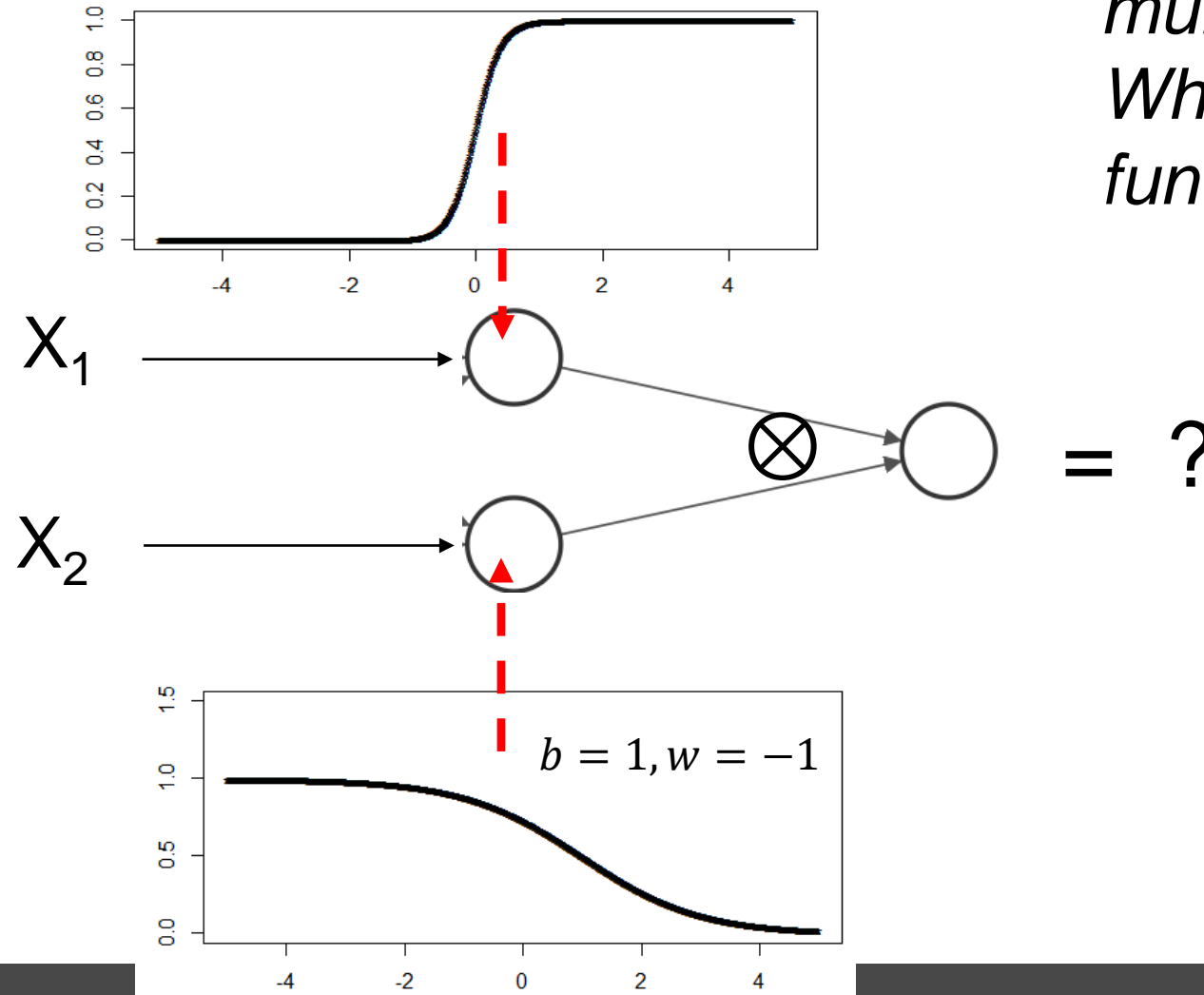


Example: 2 input into 2 logistic units with these activations



Example: 2 input into 2 logistic units with these activations

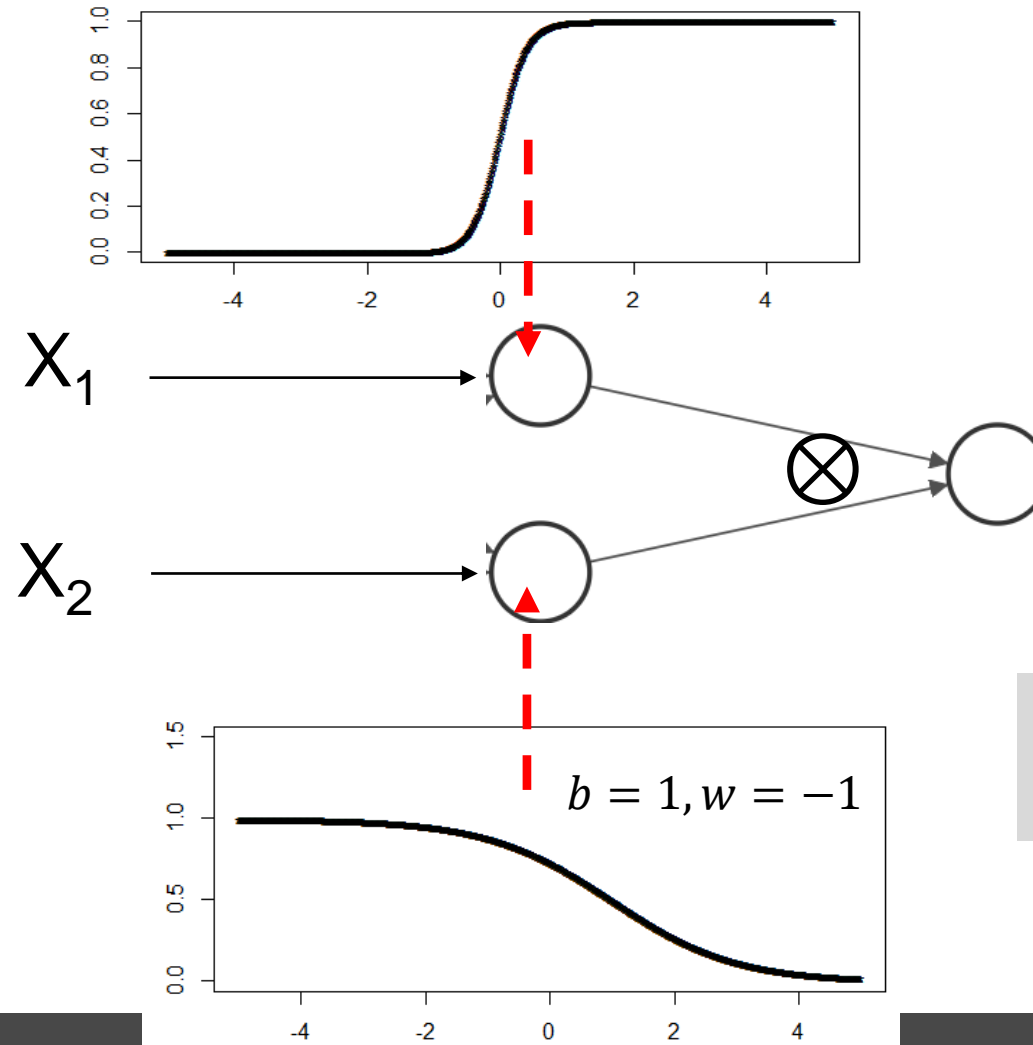
*What if you multiply these?
What is the output function doing?*



Example: 2 input into 2 logistic units with these activations

What if you multiply these?

For linear activation, what is the output function doing ?

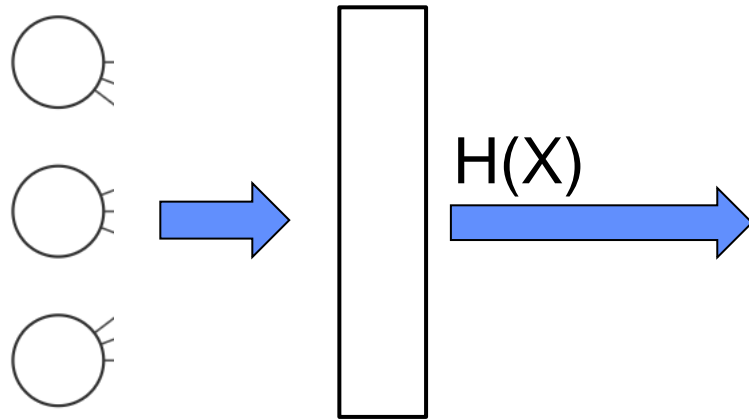


$$= \begin{cases} 0 & \text{if } X_1 < 0 \\ h(X_2) & \text{if } X_1 > 0 \end{cases}$$

X_1 "gates" X_2 activation

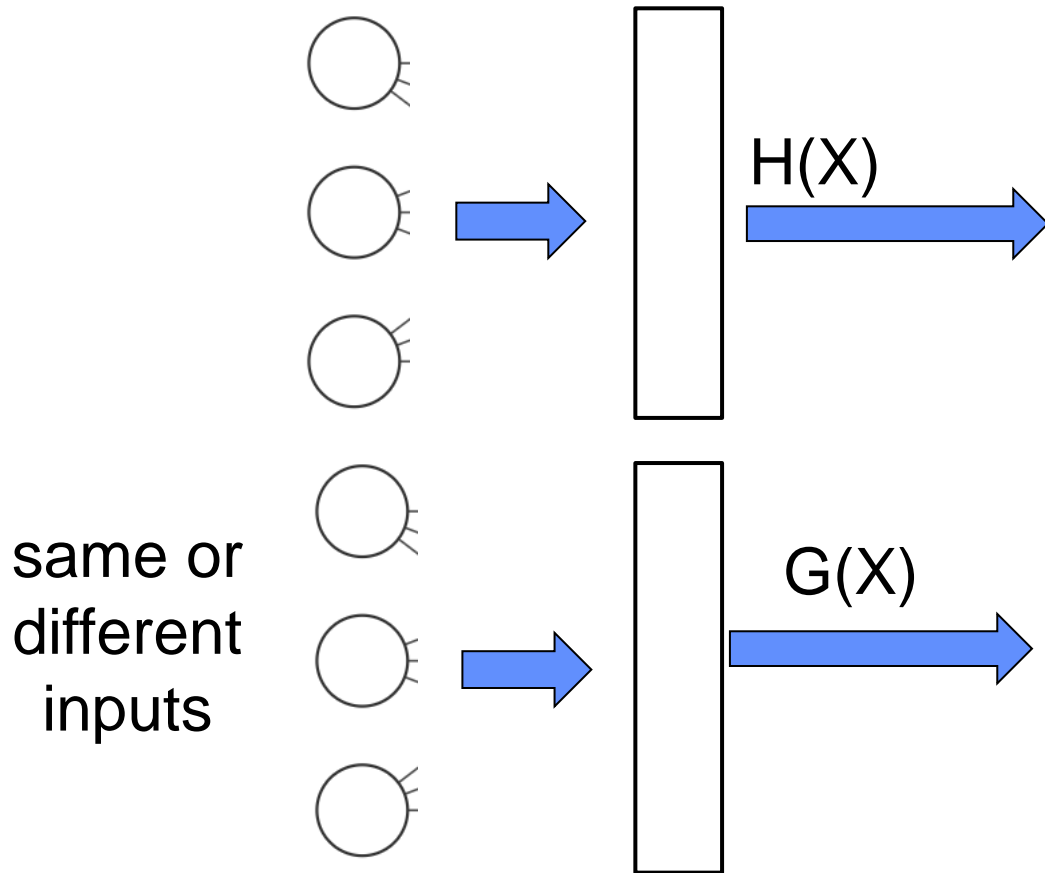
Drawing one feed forward

Input layer Hidden layer

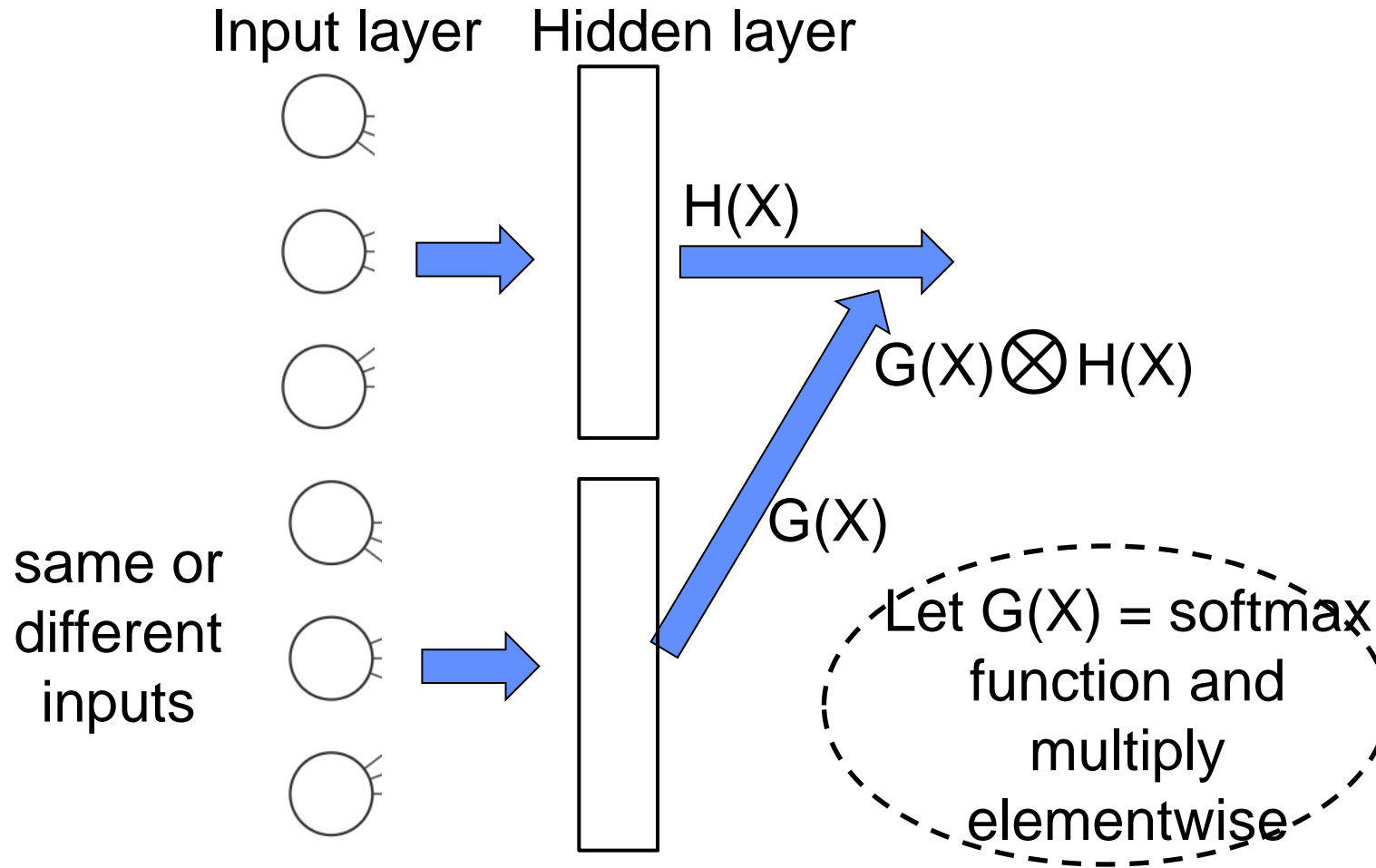


Drawing the gate from two sets of hidden units

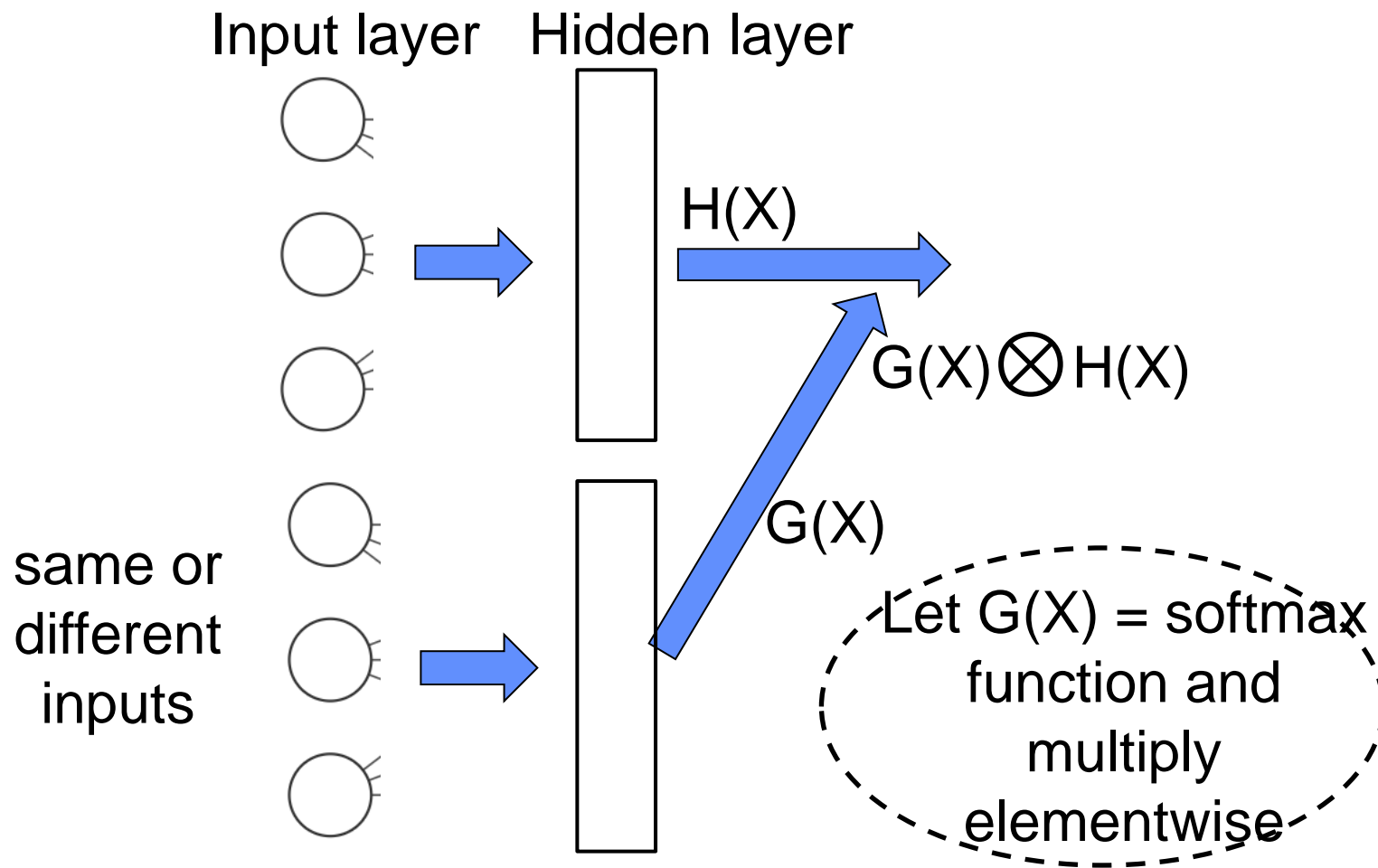
Input layer Hidden layer



Use softmax for $G(X)$ to get gating weights



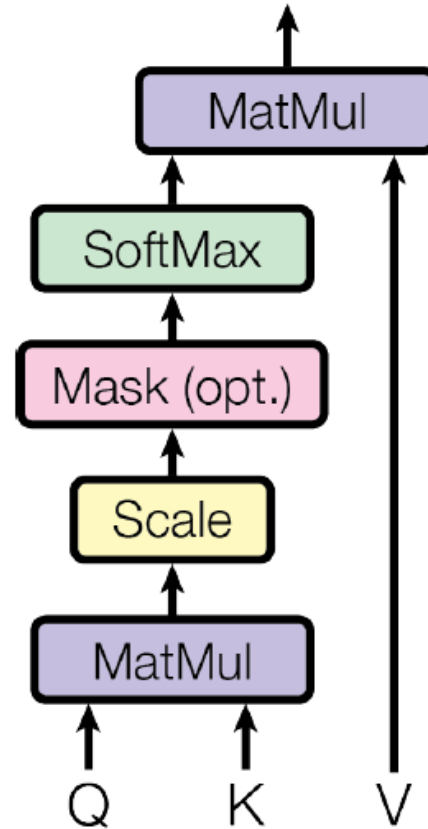
Use softmax for $G(X)$ to get gating weights



**Recall: softmax
normalizes outputs
into probability
weights**

Scaled Dot-Product Attention (very rough summary)

“Attention” mechanism in language transformers use a softmax gate



The gate is applied to possible Values (V) for decoding

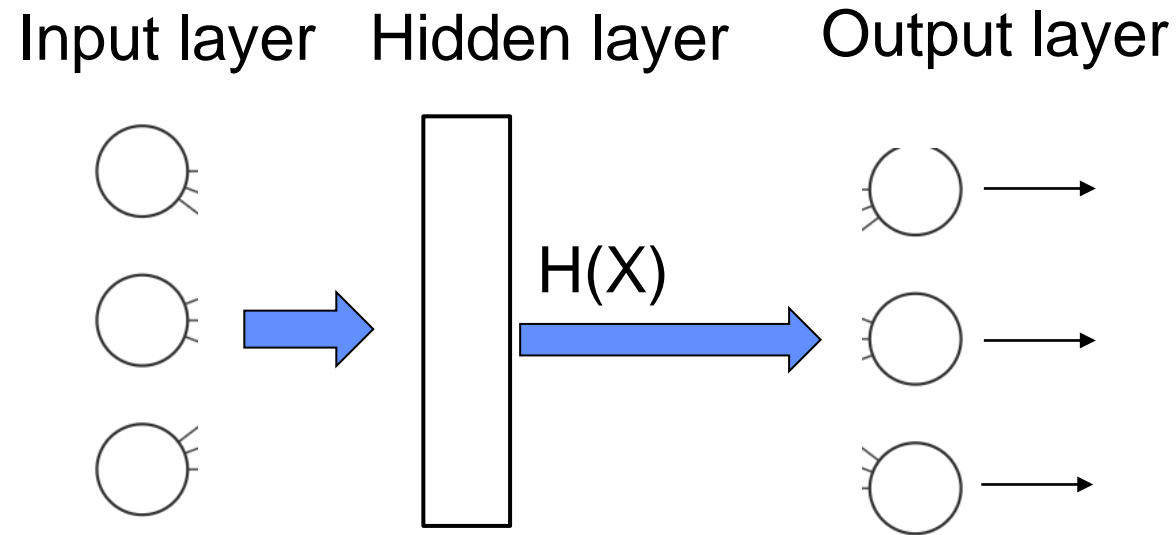
input transformations

Vaswani, et al. 2017
Attention Is All You Need (for Transformers)

Outline

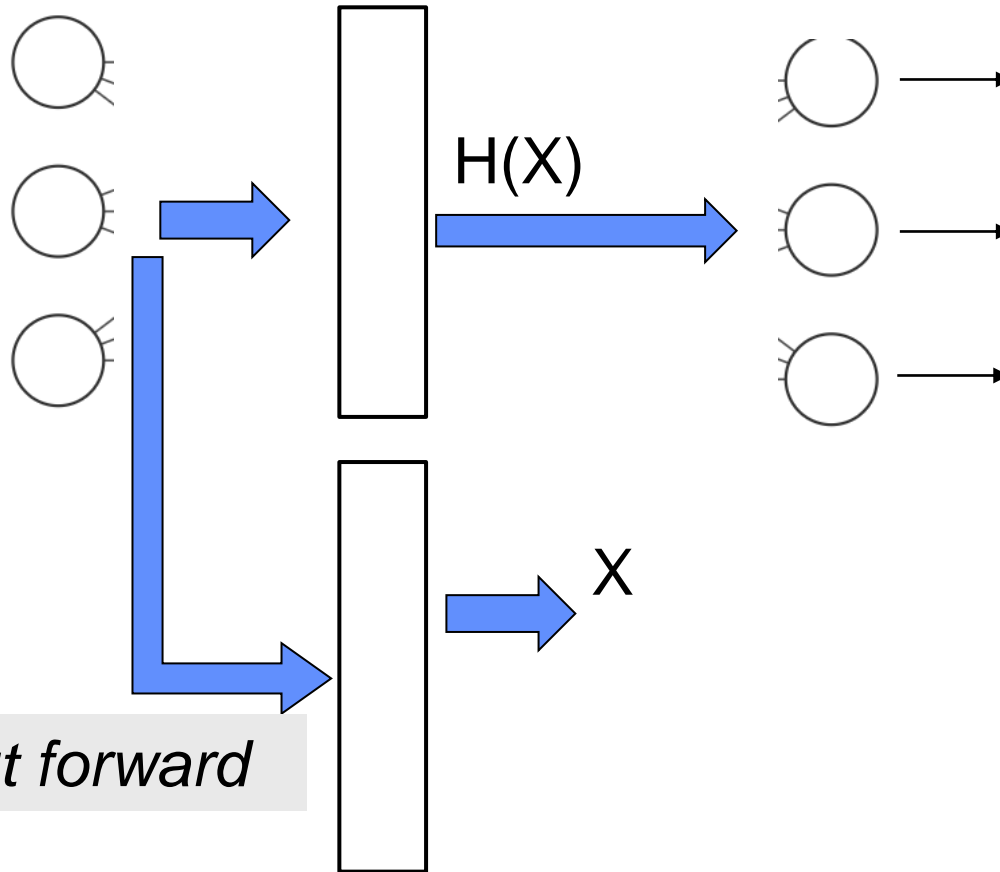
- Gate connection idea
- Skip and Residual
- Keras Model API
- Exercise MNIST Autoencoding

Recall the Multilayer Perceptron (MLP)



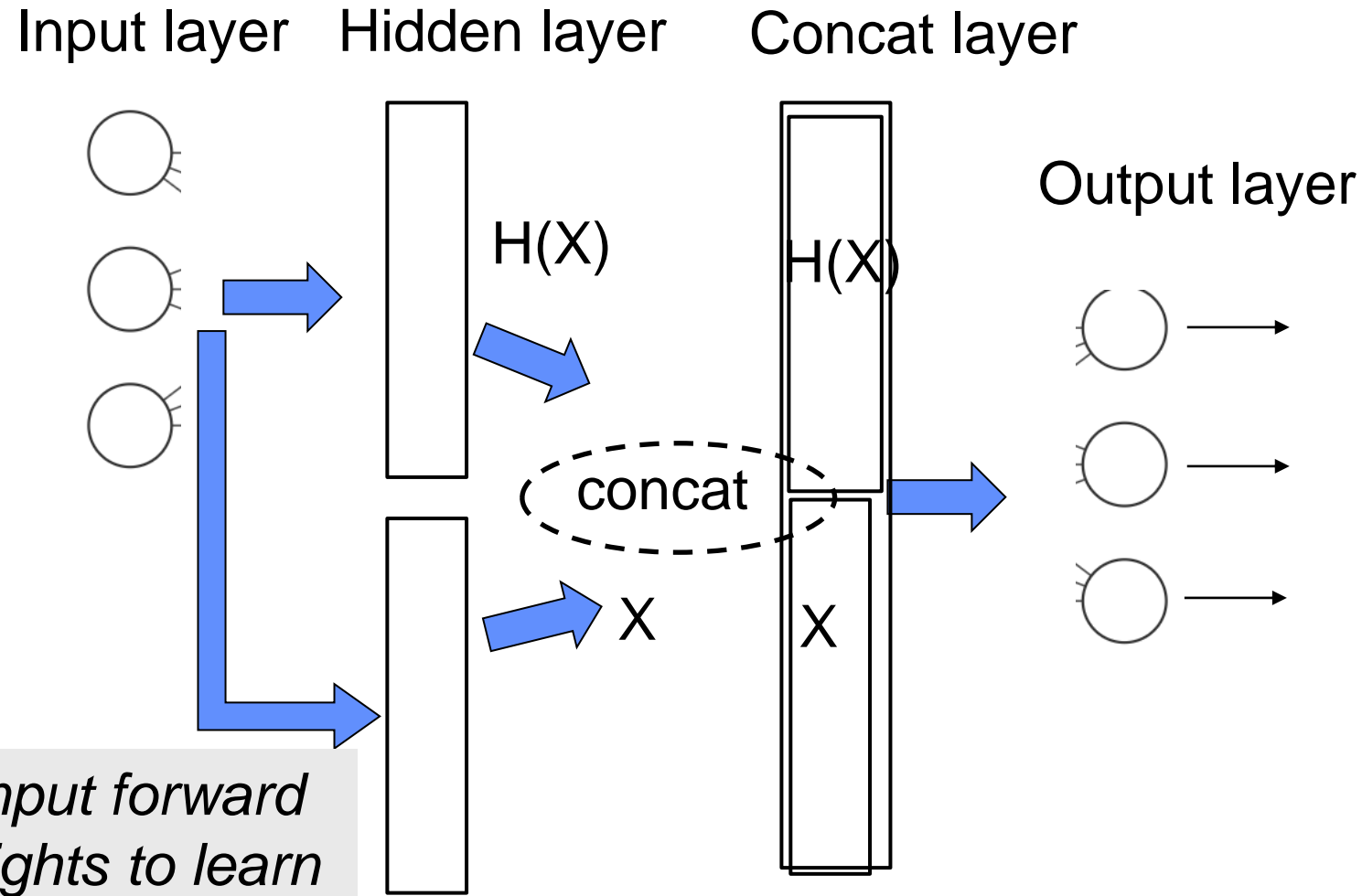
To help the MLP learn directly from input carry input forward

Input layer Hidden layer Output layer

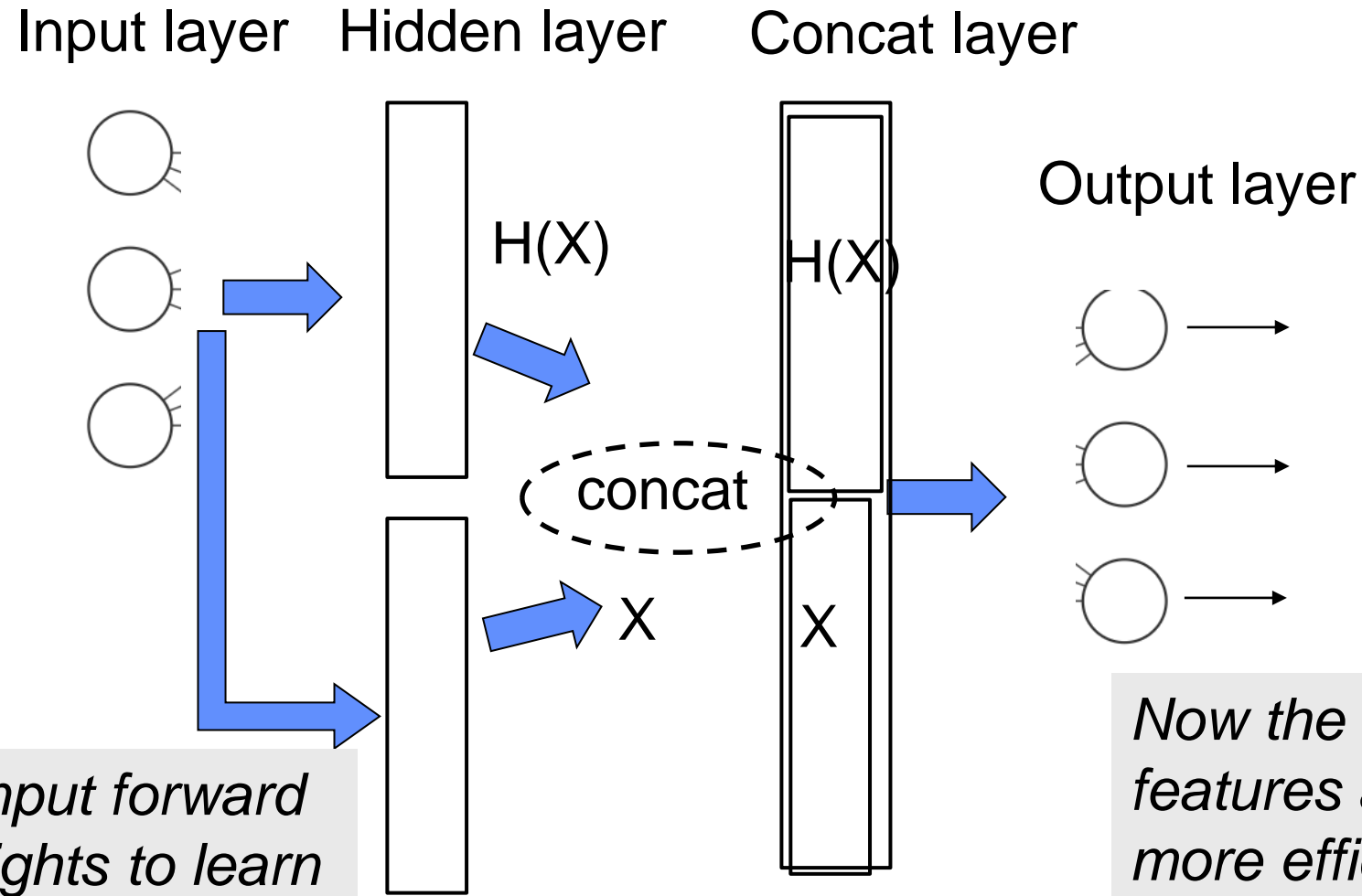


carry input forward

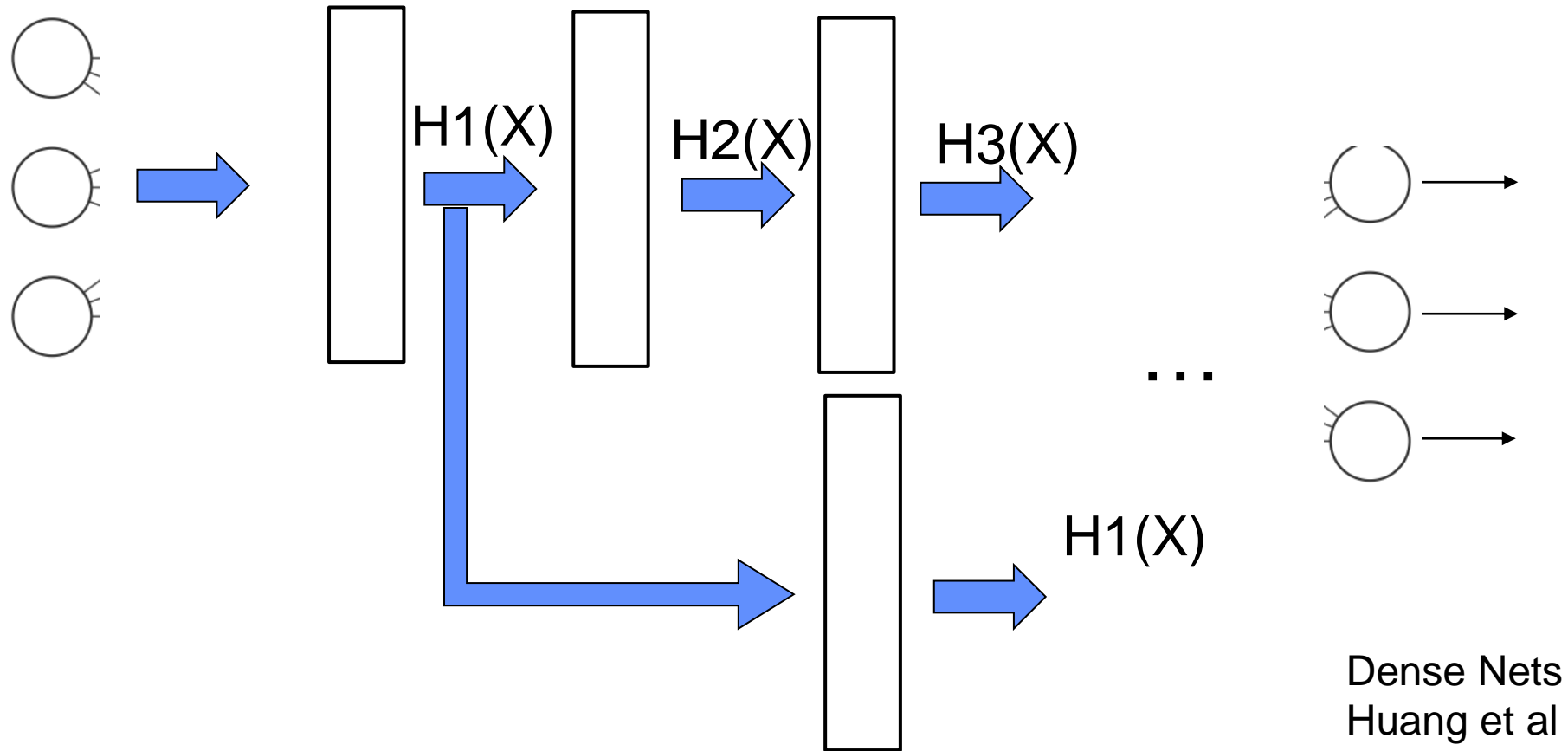
Concatenate input with hidden units into new layer



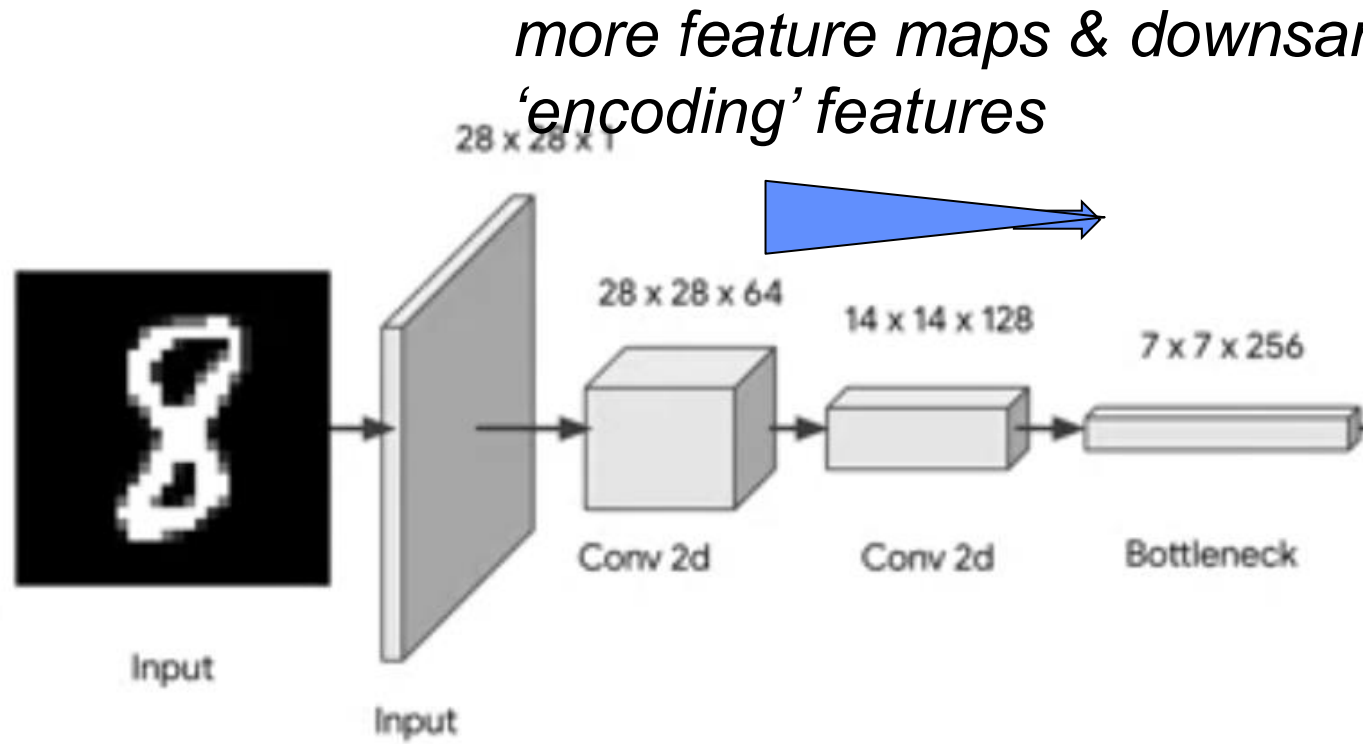
Concatenate input with hidden units into new layer



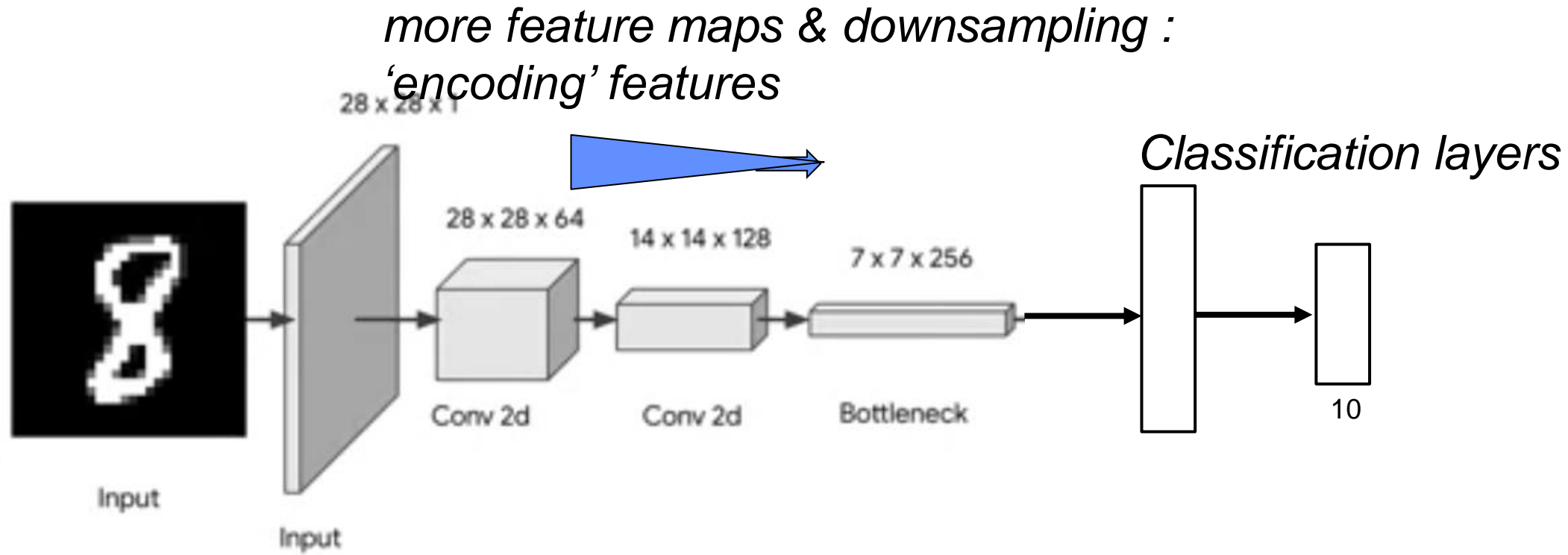
Can be done for any (or all) previous layer and *skip* any number of layers



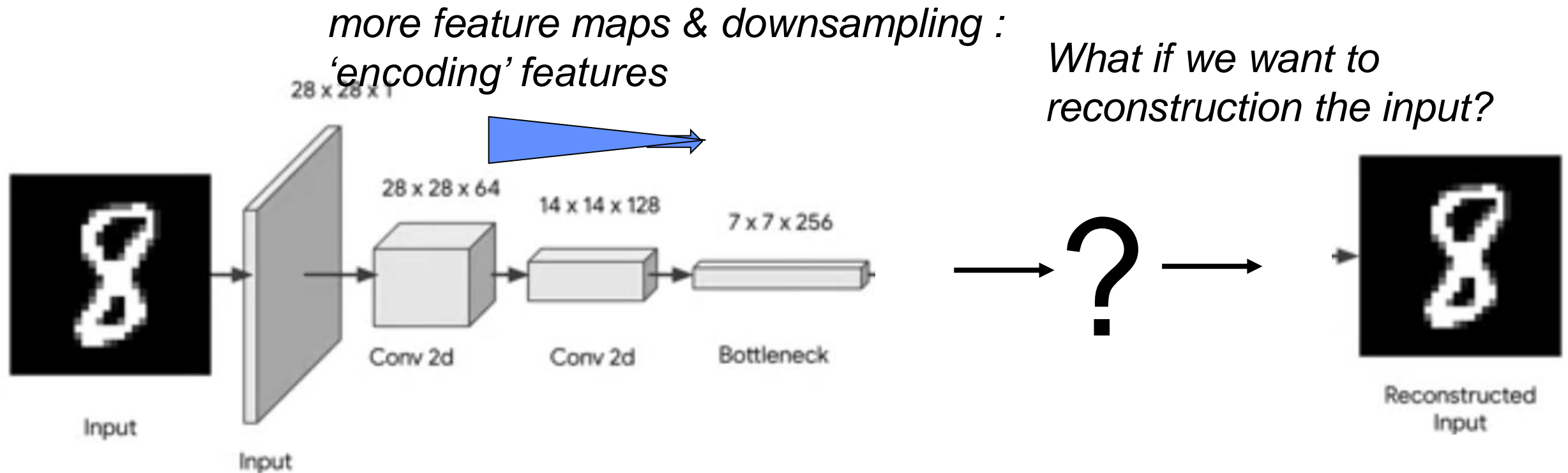
Recall: CNN architecture for MNIST classification



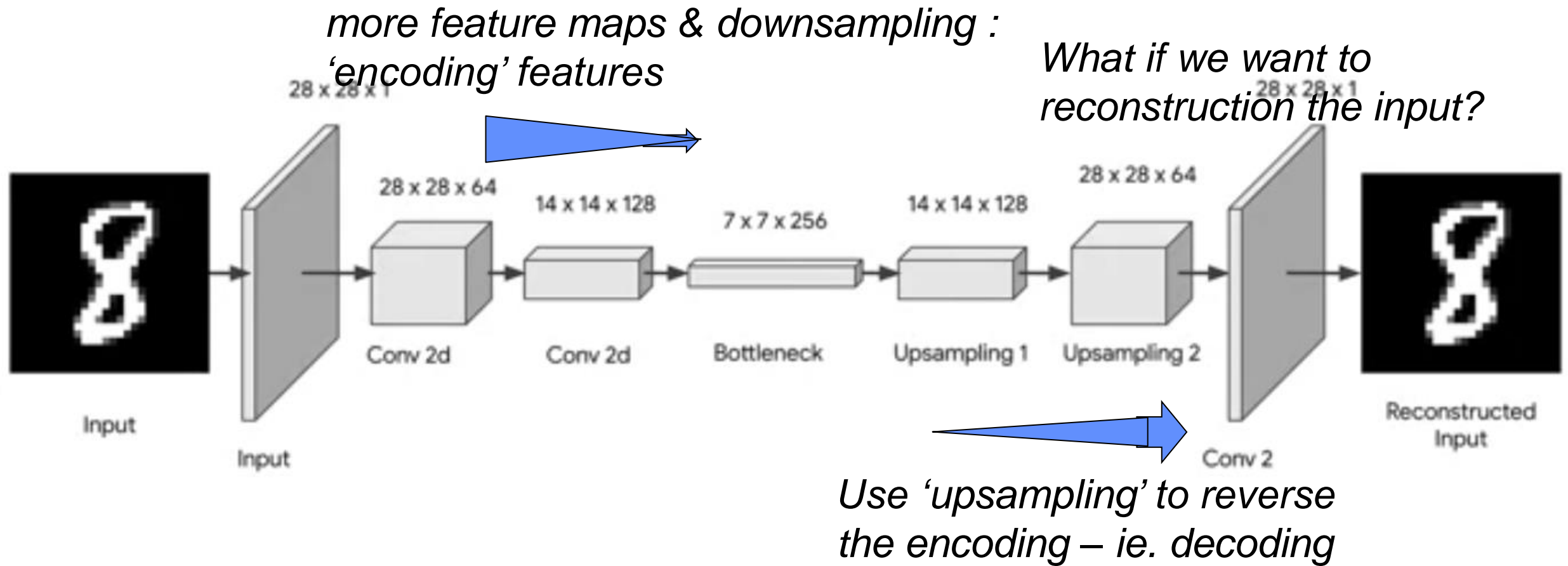
Consider: CNN architecture for MNIST classification



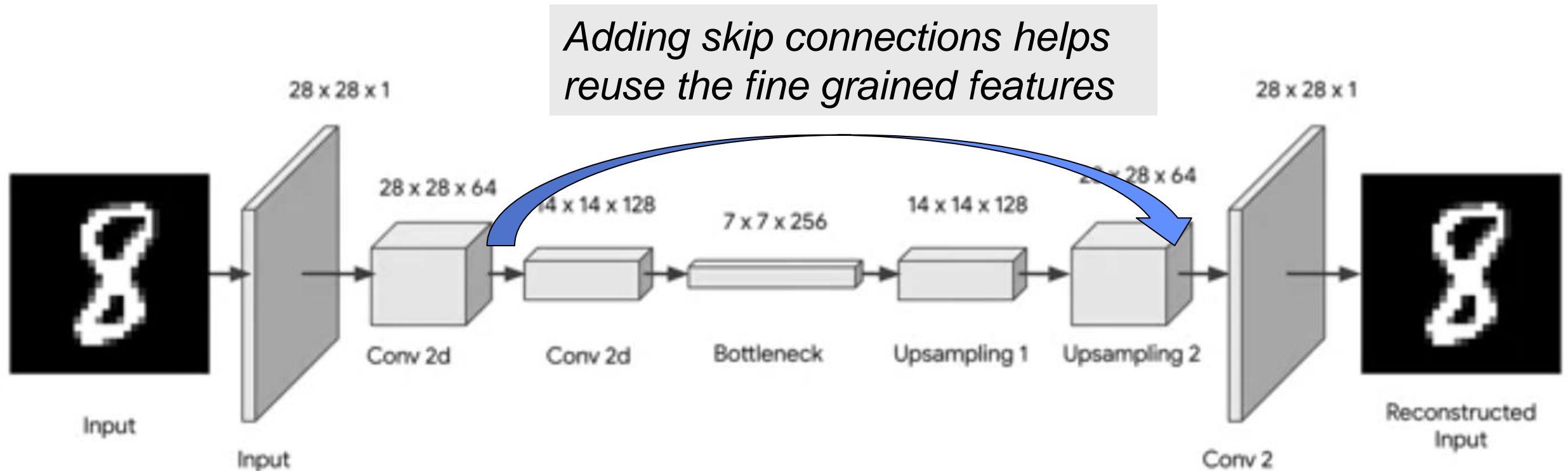
A CNN architecture for MNIST autoencoding



A CNN architecture for MNIST autoencoding

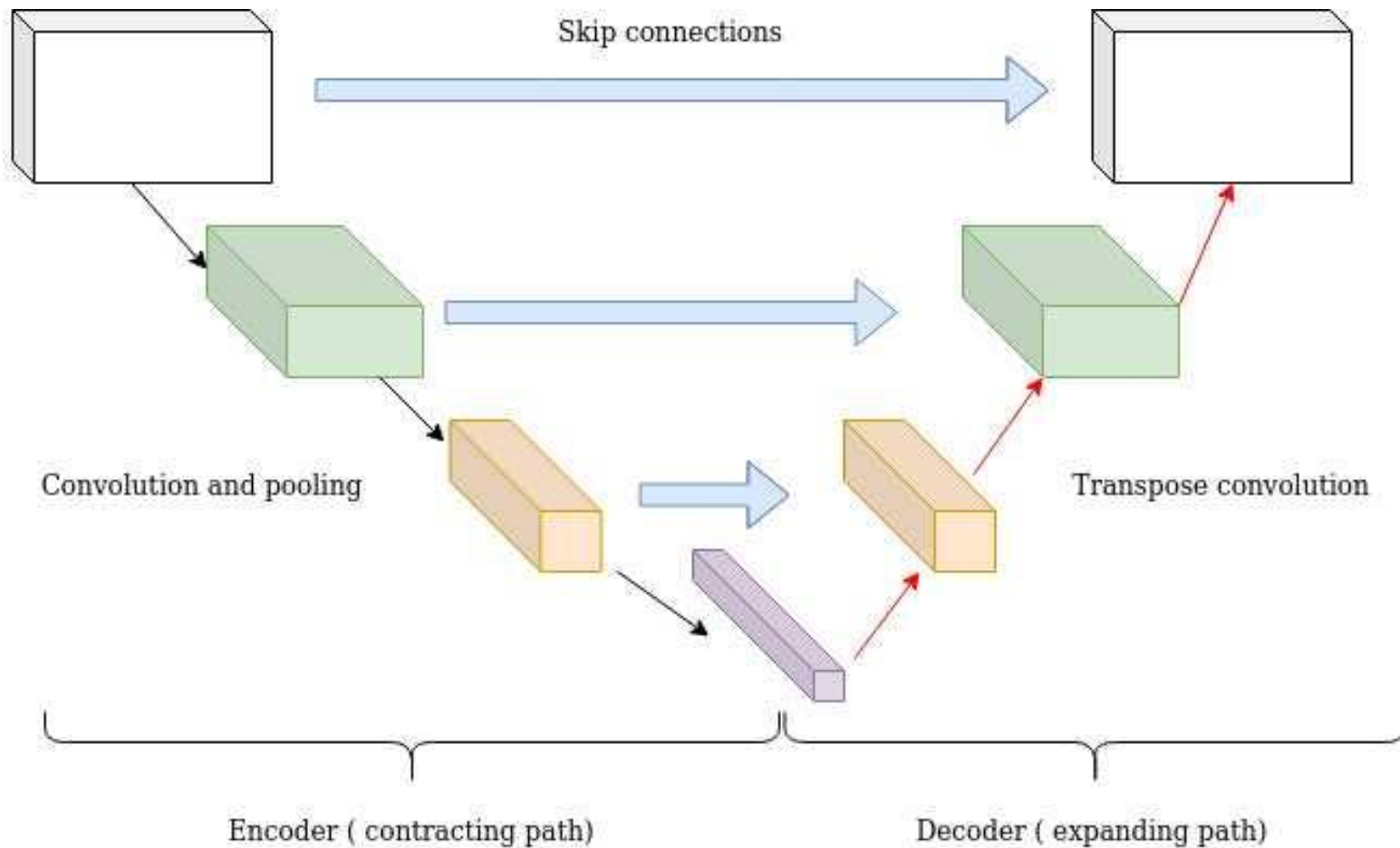


A CNN architecture for MNIST autoencoding



NOTE the 28x28 encoded maps have to be skipped ahead to where the 28x28 decoding maps are – which axis is concatenated?

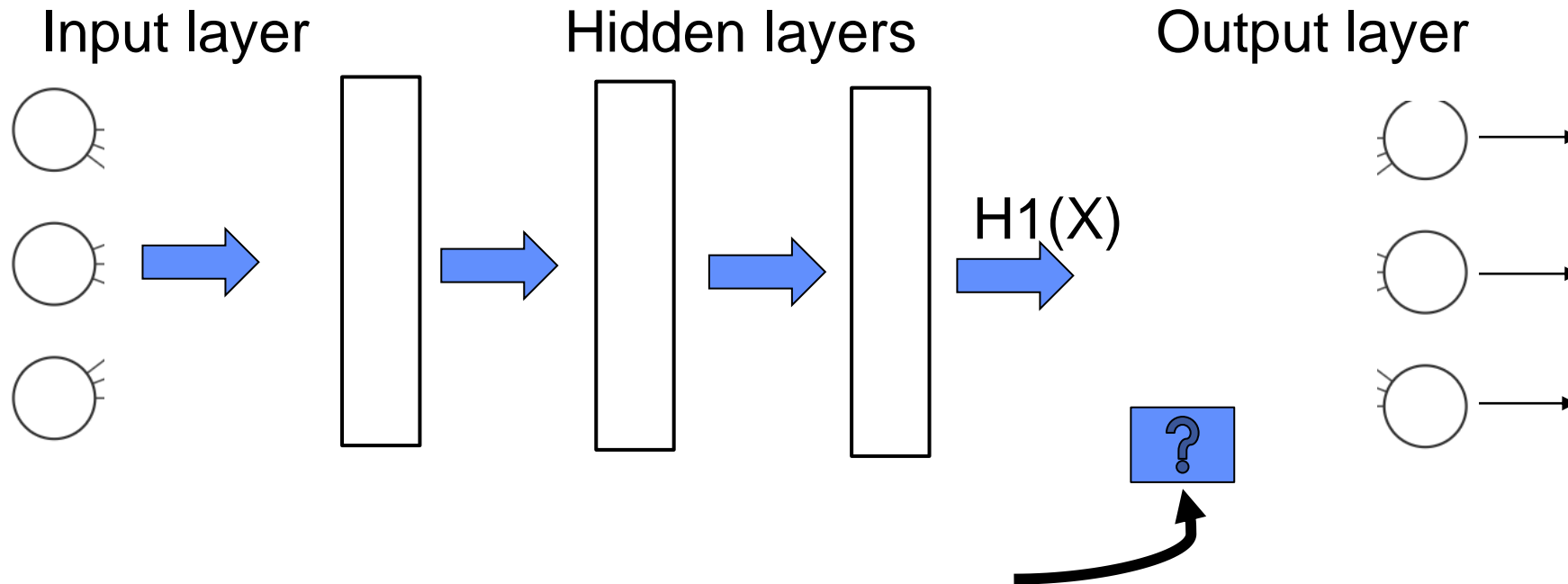
Image Encoder-Decoder is a “UNET” architecture



Outline

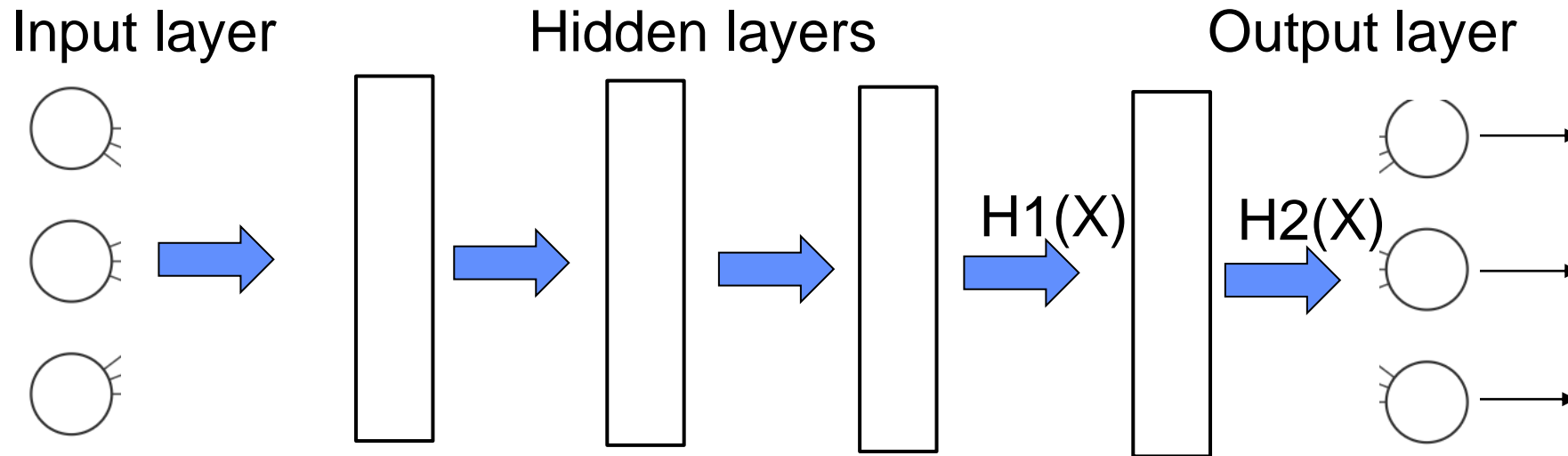
- Gate connection idea
- Skip and Residual
- Keras Model API
- Exercise MNIST Autoencoding

Consider: Can we keep adding deep layers?



Given some deep network,
should I add another layer?
What should a new layer learn?

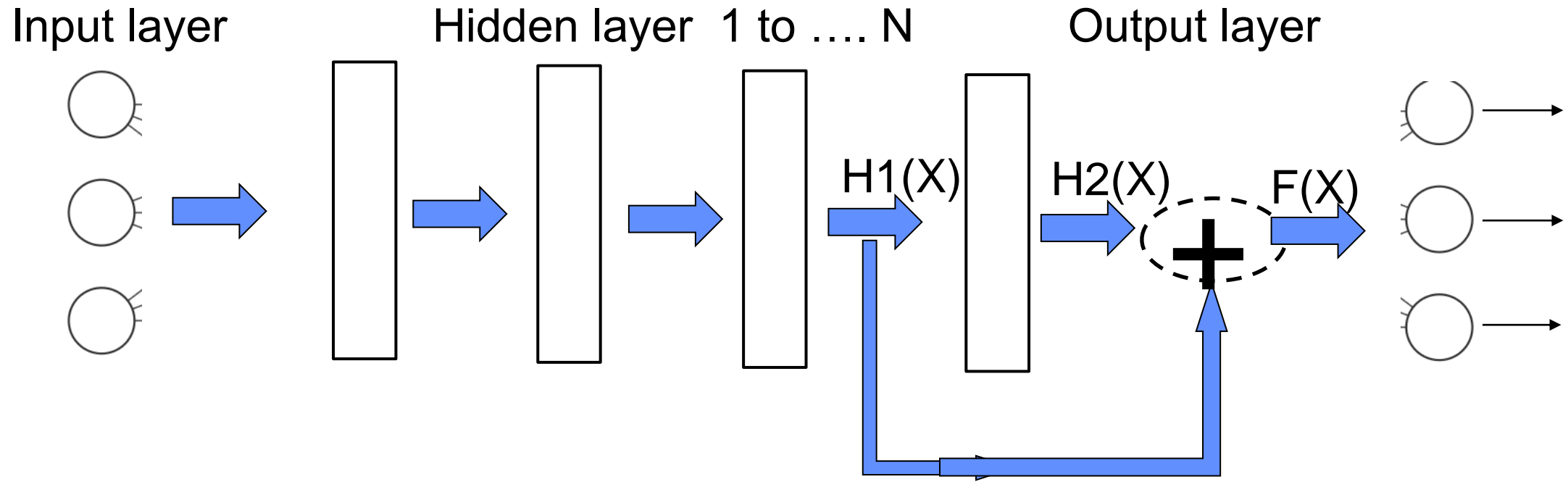
Consider: Can we keep adding deep layers?



If $H1(X)$ is good then this new layer could be unnecessary,

eg $H2(X)$ should be just $H1(X)$

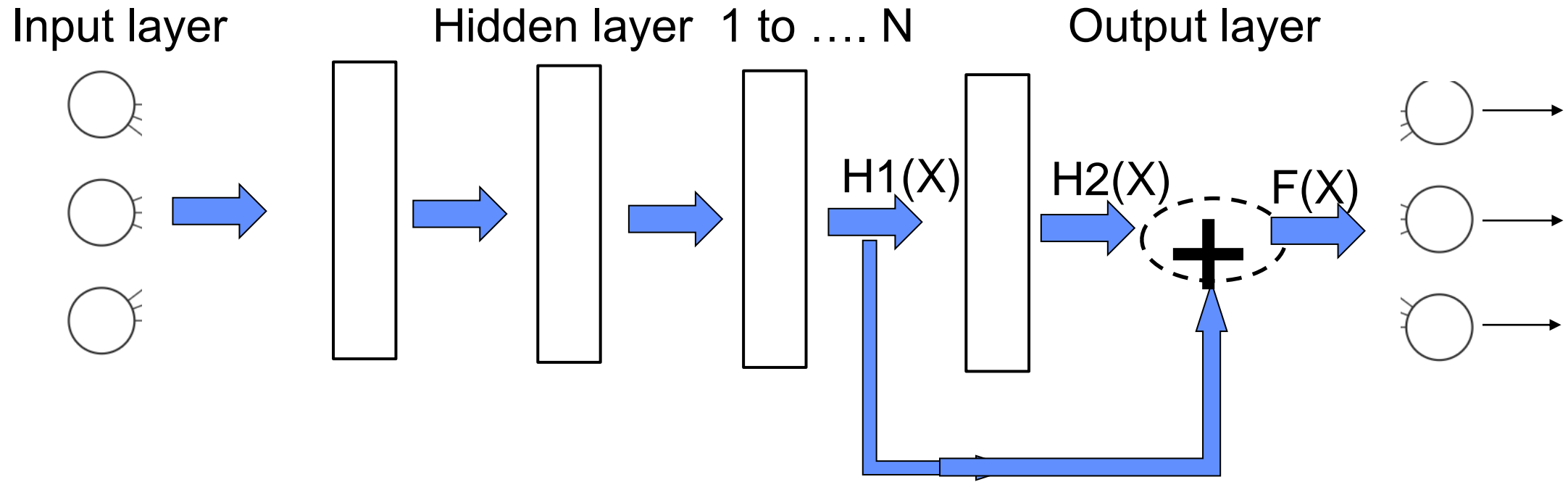
Skip with addition makes a 'residual' connection



Make it easy for next layer to learn nothing –

e.g. use $F(X)=H2(X)+H1(X)$ so that $H2(X)=F(X)-H1(X)$.

Skip with addition makes a 'residual' connection

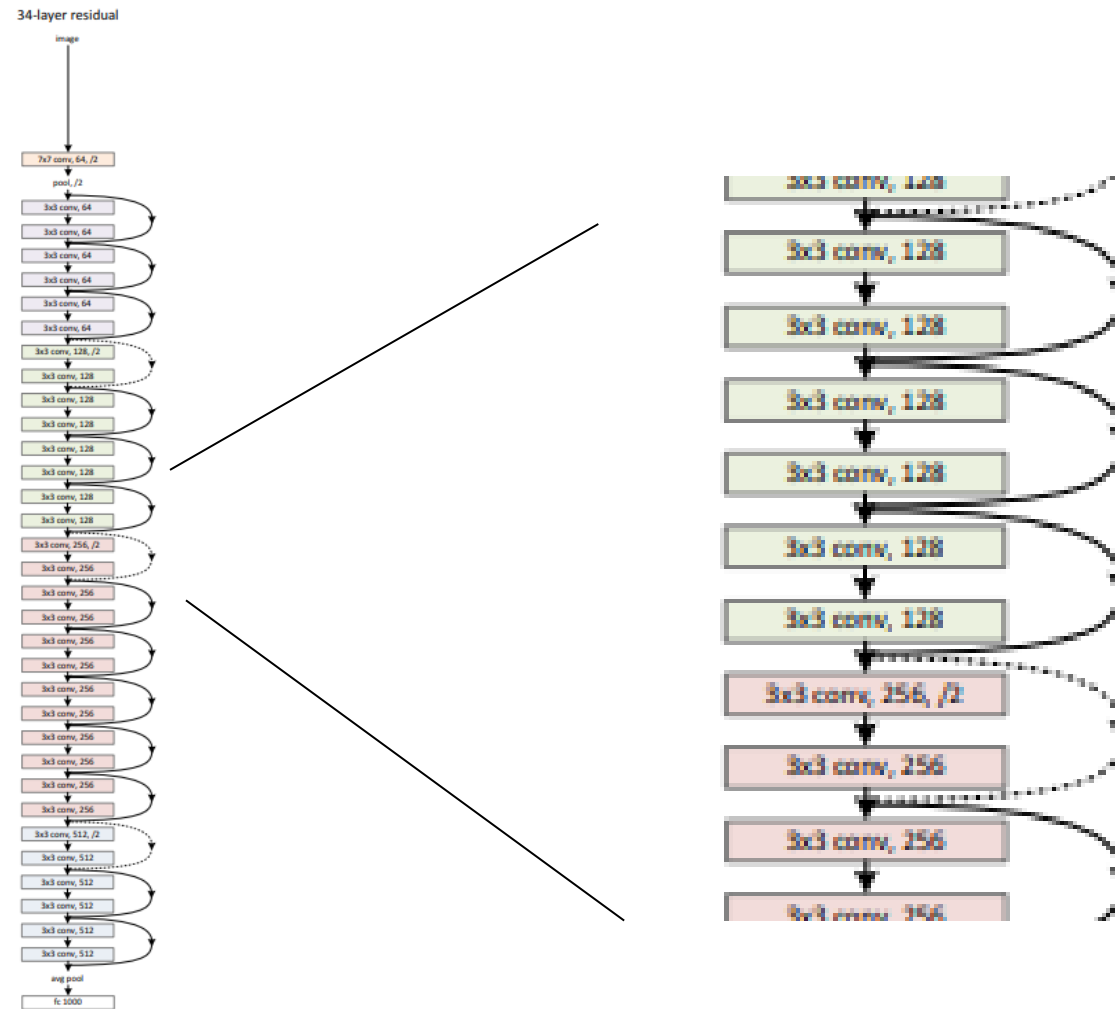


Make it easy for next layer to learn nothing –

e.g. use $F(X)=H2(X)+H1(X)$ so that $H2(X)=F(X)-H1(X)$

this is a 'residual'

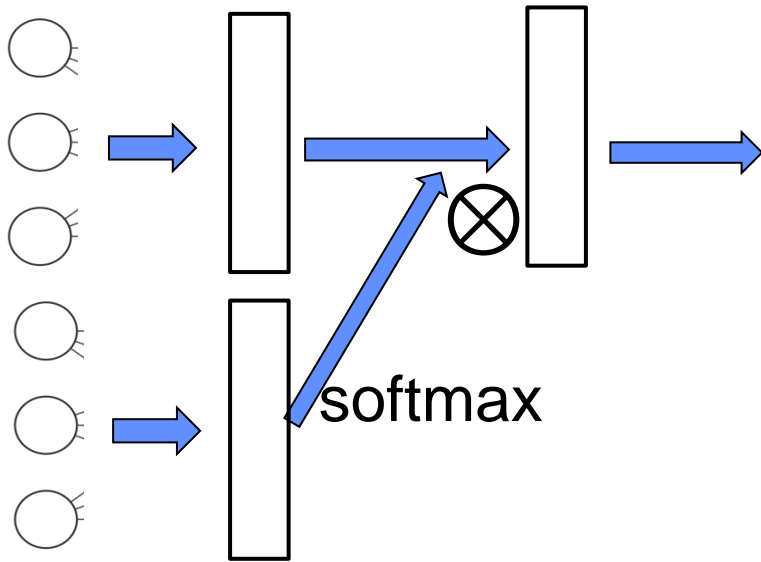
“Resnet” residual connections help deeper learning



*Deep Residual Learning,
He et.al, 2015*

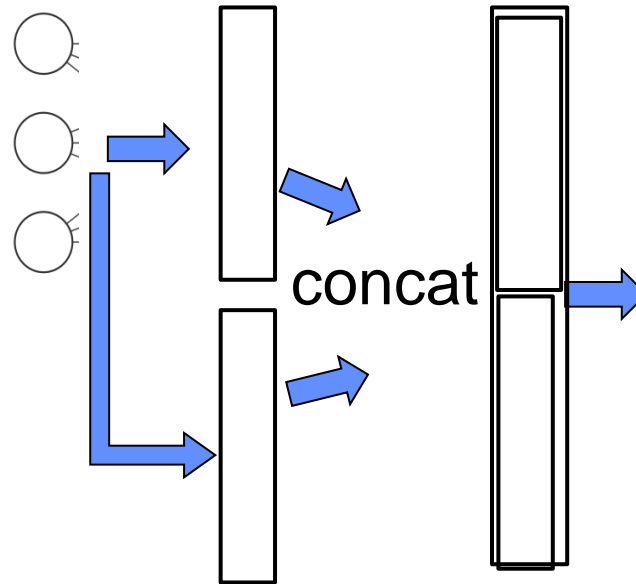
Summary: useful connections for architectures, and the intuitions

Softmax for gating



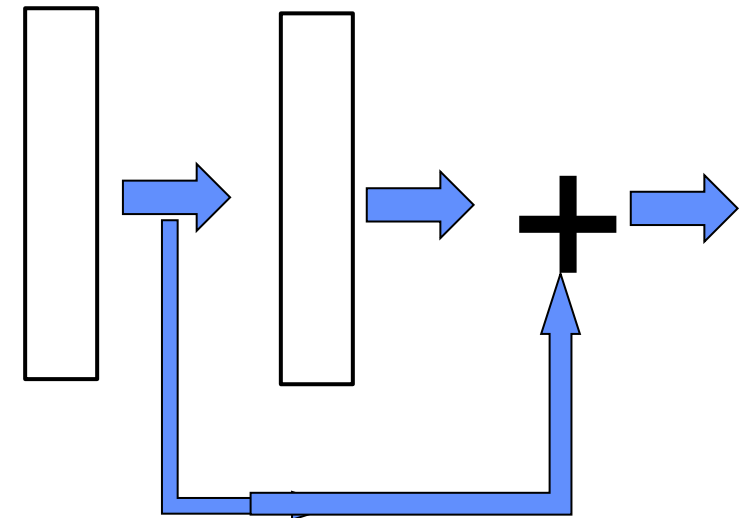
Recurrent nets,
language transformer nets

Skip connections
for feature reuse



UNET, also
feedforward nets..

Residual connections
help deeper learning



Resnet, large image
classification

Outline

- Gate connection idea
- Skip and Residual
- **Keras Model API**
- Exercise MNIST Autoencoding

Keras: Sequential API VS Functional API

```
#specify the neural network model and Learning parameters  
my_model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(32,activation='relu'),  
    tf.keras.layers.Dense(10,activation='softmax')])  
my_model.summary()
```

*A sequence of layers:
the inputs are assumed
to be in order*

Keras: Sequential API VS Functional API

```
#specify the neural network model and learning parameters
my_model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')])
my_model.summary()
```

*A sequence of layers:
the inputs are assumed
to be in order*

```
#specify the neural network model and learning parameters
inputs = tf.keras.layers.Input(shape=(28, 28, 1,))
inputs_flattened = tf.keras.layers.Flatten()(inputs)
hidden_layer = tf.keras.layers.Dense(32,activation='relu')(inputs_flattened)
output_layer = tf.keras.layers.Dense(10,activation='softmax')(hidden_layer)
```

*A sequence of functions:
Input layer(s) are specified*

Keras: Sequential API VS Functional API

#specify the neural network model and learning parameters

```
my_model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])
my_model.summary()
```

*A sequence of layers:
the inputs are assumed
to be in order*

#specify the neural network model and learning parameters

```
inputs = tf.keras.layers.Input(shape=(28, 28, 1,))
inputs_flattened = tf.keras.layers.Flatten()(inputs)
hidden_layer = tf.keras.layers.Dense(32, activation='relu')(inputs_flattened)
output_layer = tf.keras.layers.Dense(10, activation='softmax')(hidden_layer)

my_model = tf.keras.Model(inputs, output_layer)
my_model.summary()
```

*A sequence of functions:
Input layer(s) are specified*

*The Model() function figures out the full
path(s) to connect the input(s) to output(s)*

Outline

- Gate connection idea
- Skip and Residual
- Keras Model API
- **Exercise MNIST Autoencoding**

Exercise

- **MNIST autoencoder, reconstruct digits from noisy inputs**
- **Add skip connections with concatenation**

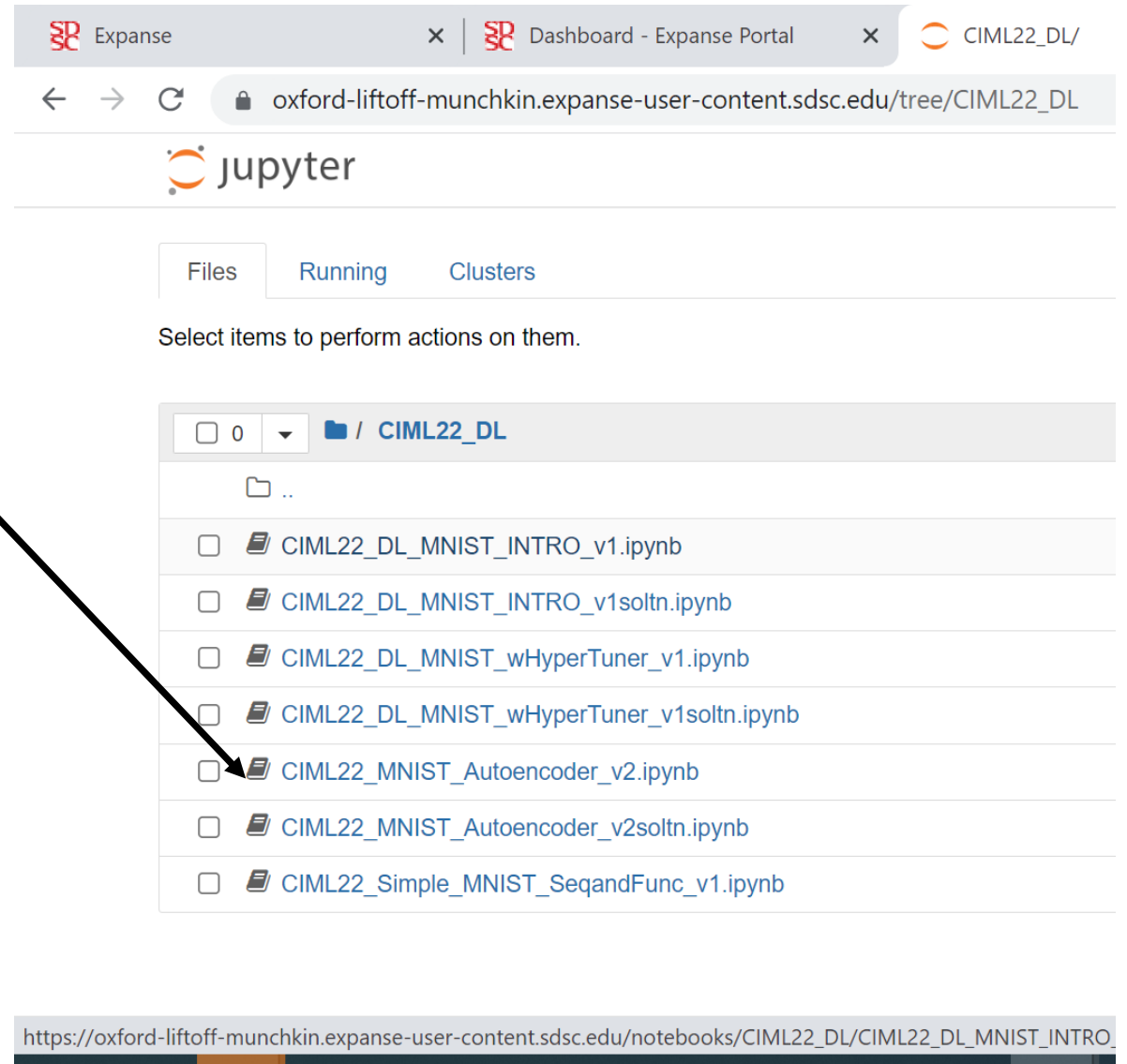
Note: make sure you see how the outputs from encoding layers are matched up to inputs for decoding layers!

14x14 encoding feature maps should be concatenated with 14x14 decoding maps








- **Review reconstruction outputs to see improvements**

In jupyter notebook session
open the
MNIST_Autoencoder
notebook

Follow instructions in the
notebook



The screenshot shows a web browser with three tabs: 'Expance', 'Dashboard - Expance Portal', and 'CIML22_DL/'. The address bar shows the URL 'oxford-liftoff-munchkin.expance-user-content.sdsc.edu/tree/CIML22_DL'. The JupyterLab interface has tabs for 'Files', 'Running', and 'Clusters'. Below these is a message: 'Select items to perform actions on them.' The file browser shows a directory structure with a folder icon and '..'. Below that is a list of files:

- ☐  CIML22_DL_MNIST_INTRO_v1.ipynb
- ☐  CIML22_DL_MNIST_INTRO_v1soltn.ipynb
- ☐  CIML22_DL_MNIST_wHyperTuner_v1.ipynb
- ☐  CIML22_DL_MNIST_wHyperTuner_v1soltn.ipynb
- ☐  CIML22_MNIST_Autoencoder_v2.ipynb
- ☐  CIML22_MNIST_Autoencoder_v2soltn.ipynb
- ☐  CIML22_Simple_MNIST_SeqandFunc_v1.ipynb

An arrow points from the text 'MNIST_Autoencoder notebook' to the file 'CIML22_MNIST_Autoencoder_v2.ipynb'. At the bottom of the screenshot, a URL is visible: 'https://oxford-liftoff-munchkin.expance-user-content.sdsc.edu/notebooks/CIML22_DL/CIML22_DL_MNIST_INTRO_'.

```

def encoder(inputs):
    '''Defines the encoder with two Conv2D and max pooling layers.'''
    conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(inputs)
    #padding same produces same output size
    max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1) #max pooling does the downsampling

    conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(max_pool_1)
    max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)

    return max_pool_2, conv_1, conv_2

```

```

def decoder(inputs, enc_conv1, enc_conv2):
    '''Defines the decoder path to upsample back to the original image size.'''
    #Notice that padding = same keeps the output same size as input

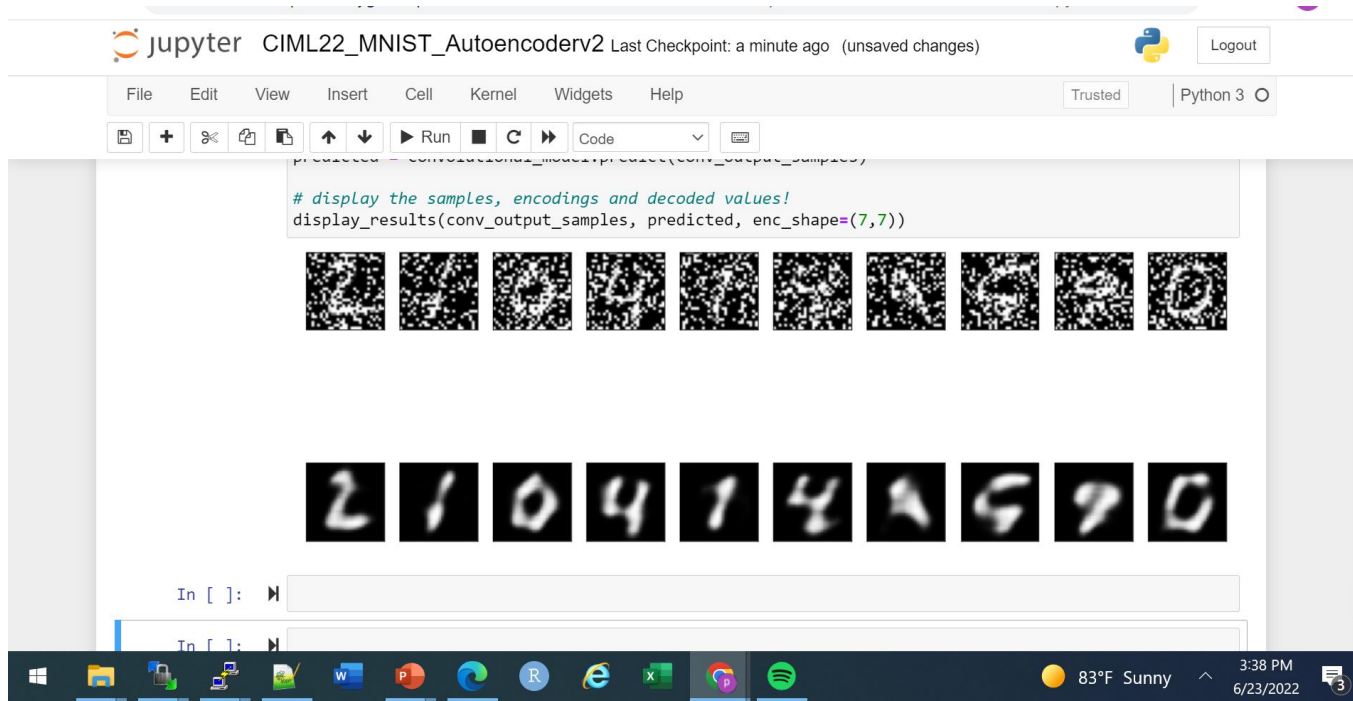
    conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(inputs)
    up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

    #another option is transpose
    # up_sample_1 = tf.keras.layers.Conv2DTranspose(128, kernel_size=(2,2), strides=(2,2))(conv_1)
    # in a transpose convolutional layer,

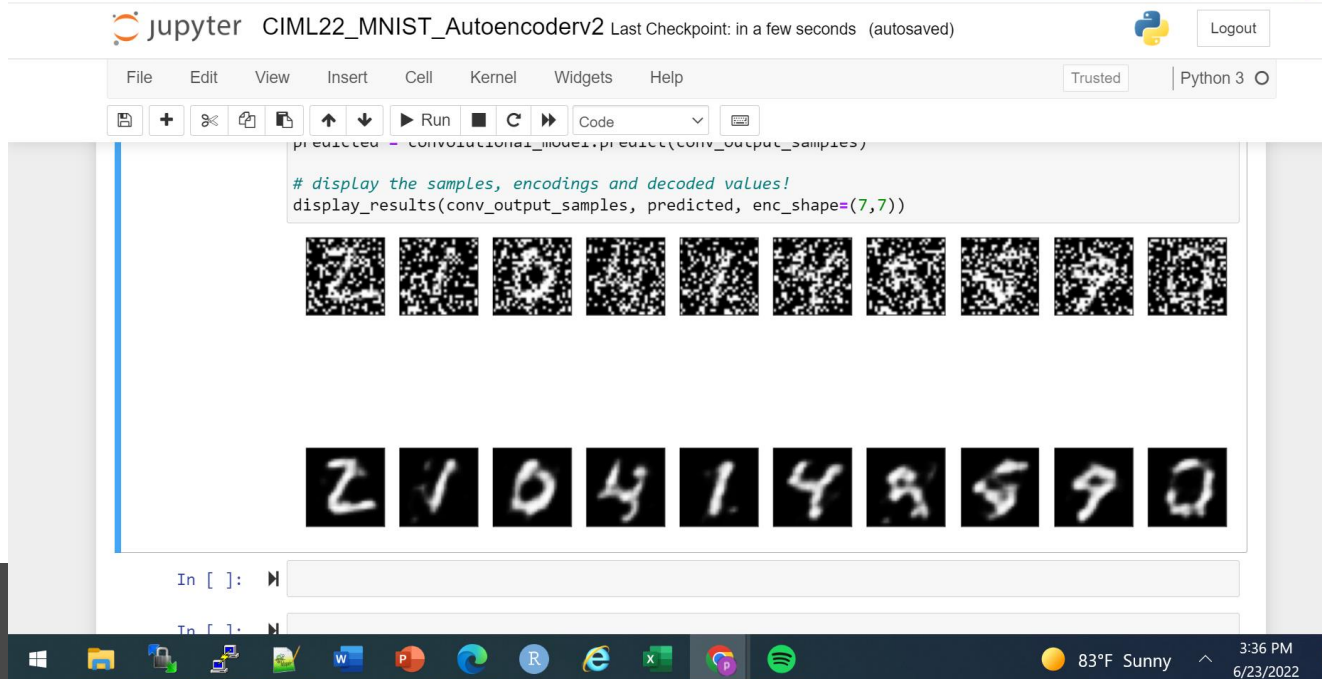
    # ----->>>> before the conv_2 line add a
    # ----->>>> tf.keras.layers.concatenate statement to combine enc_conv2 with decoding u
    # conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(up_sample_1)

    skip_concat_1 = tf.keras.layers.concatenate([up_sample_1, enc_conv2])
    conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(skip_concat_1)
    # ----->>>> and change the input into conv_2

```



With out skip
20 epochs
Loss 0.1664



With skip,
20 epochs loss 0.14

Are the numbers a little bit
more reconstructed?