بسم الله الرحمن الرحیم

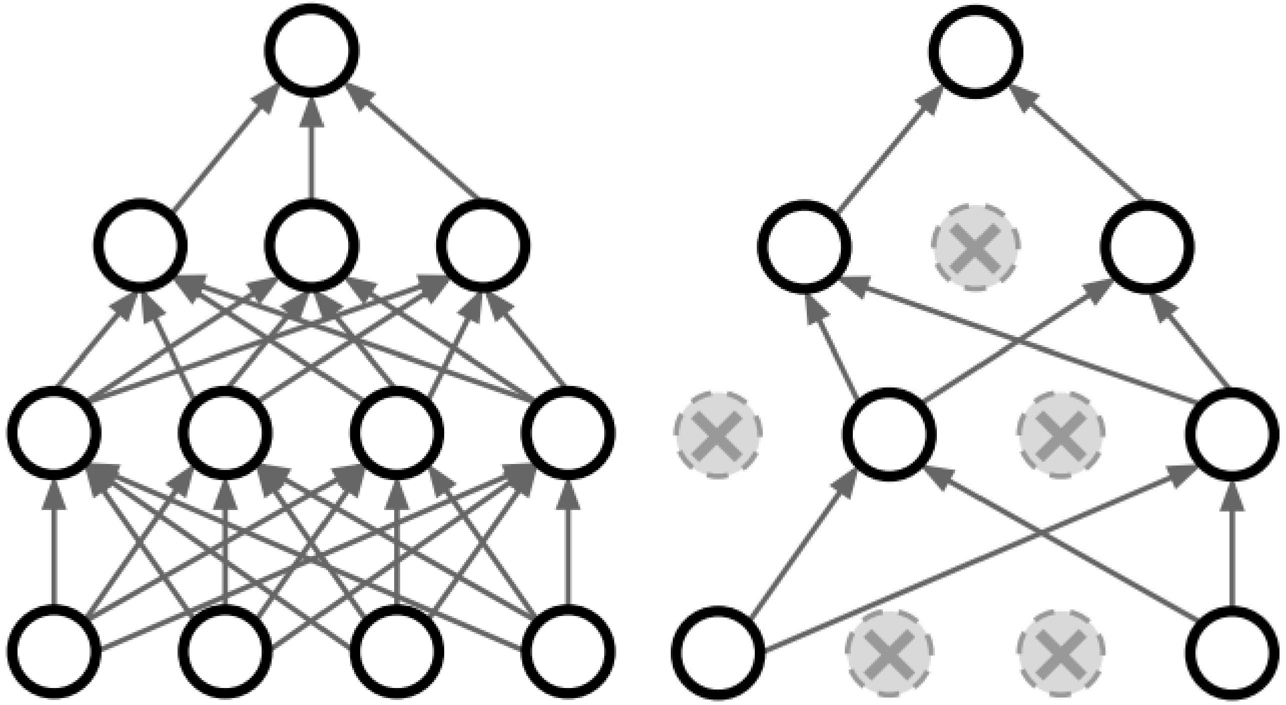
Deep Learning

Mohammad Reza Mohammadi

2021

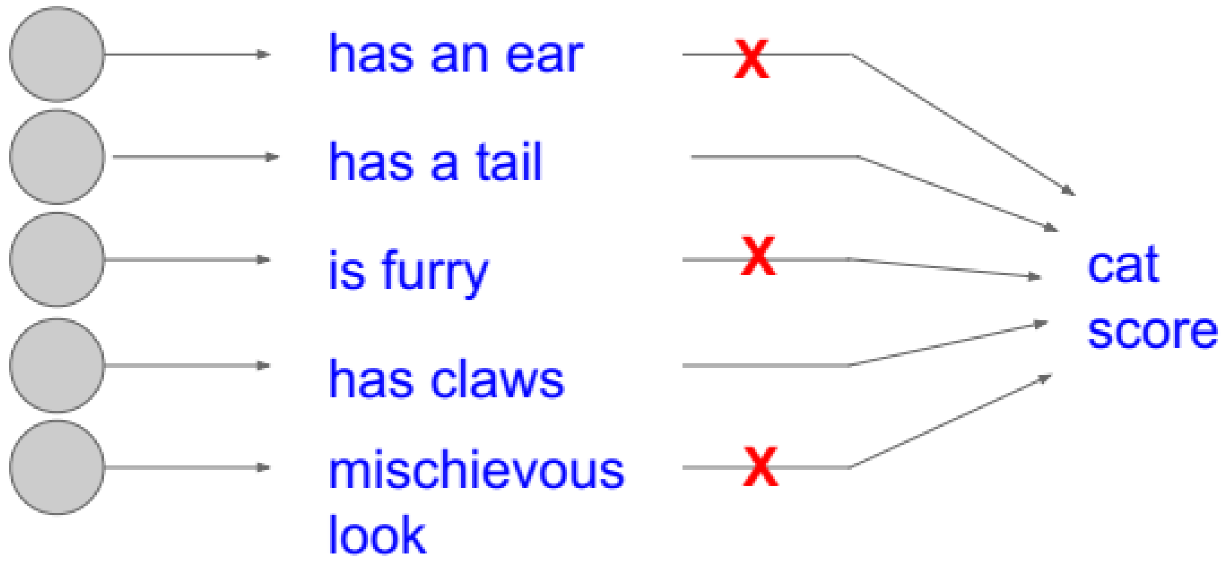
Dropout

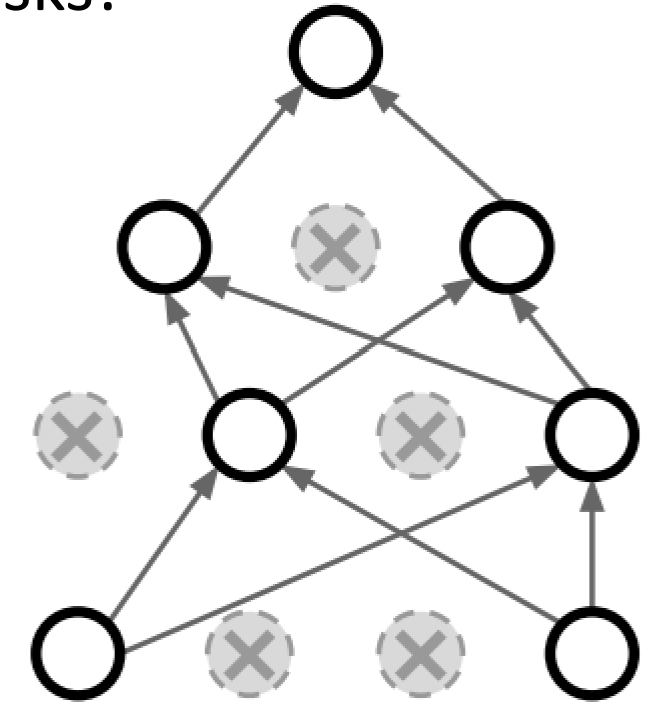
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter; 0.5 is common



Dropout

- Dropout is training a large ensemble of models (that share parameters)
- Each binary mask is one model
- An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!



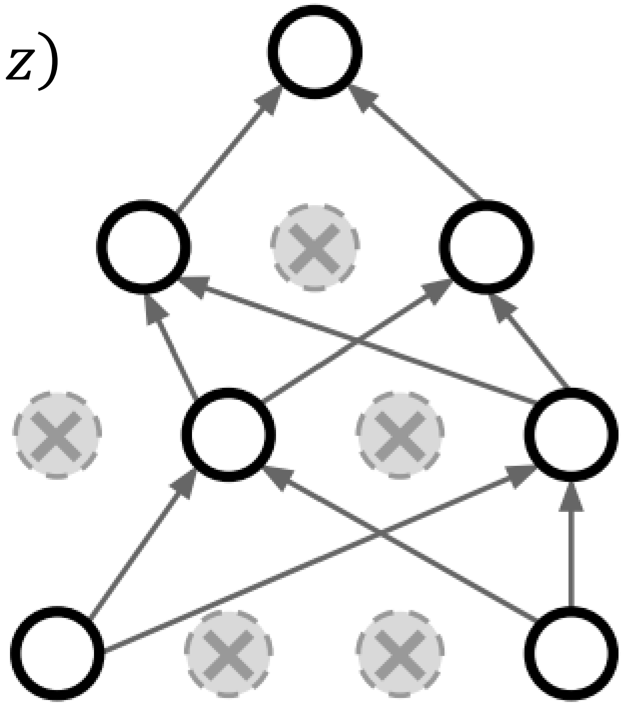


Dropout: Test time

- Dropout makes our output random!
- Want to “average out” the randomness at test-time

$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

- But this summation is very time-consuming

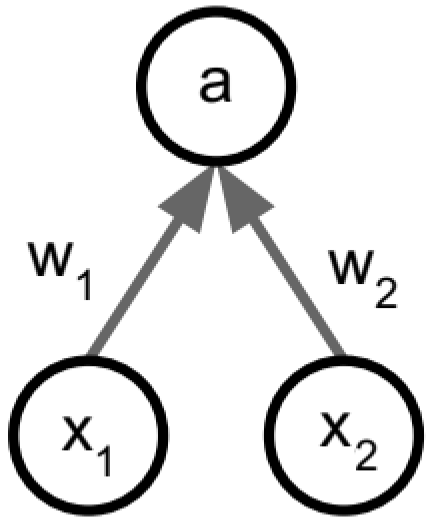


Dropout: Test time

- Want to approximate the expected value
- Consider a single neuron

$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

$$E[a] = p^2(w_1x_1 + w_2x_2) + p(1-p)(w_1x_1 + w_2 0)$$
$$+ (1-p)p(w_1 0 + w_2x_2) + (1-p)^2(w_1 0 + w_2 0)$$
$$= p(w_1x_1 + w_2x_2)$$



- At test time, multiply by dropout probability

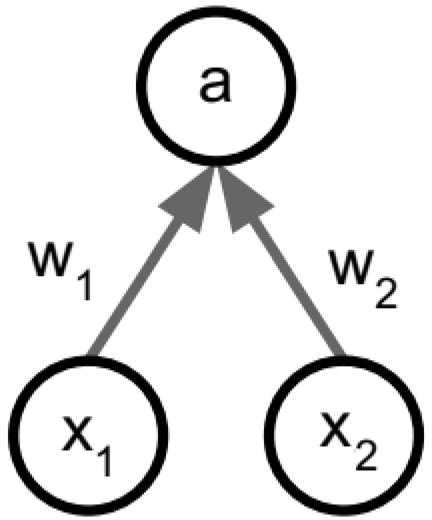
Dropout: Test time

- Want to approximate the expected value
- Consider a single neuron

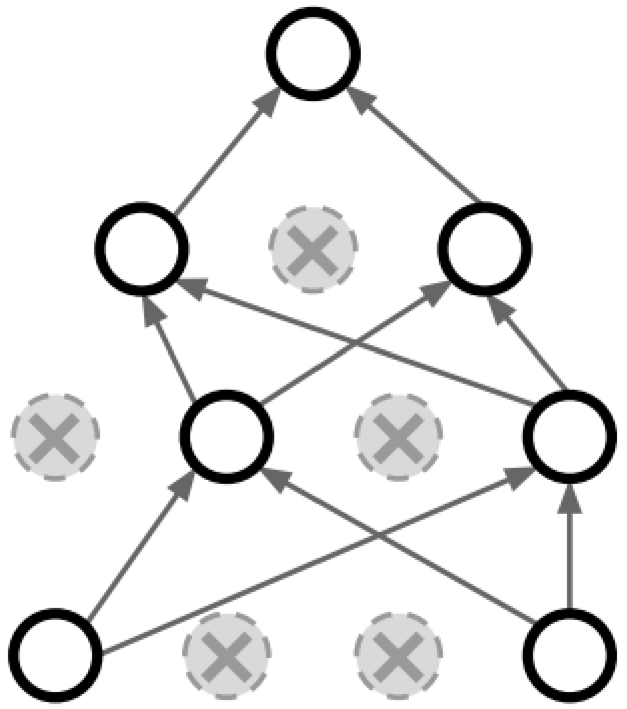
$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

$$E[a] = p^2(w_1x_1 + w_2x_2) + p(1-p)(w_1x_1 + w_20)$$
$$+ (1-p)p(w_10 + w_2x_2) + (1-p)^2(w_10 + w_20)$$
$$= p(w_1x_1 + w_2x_2)$$

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```



Dropout

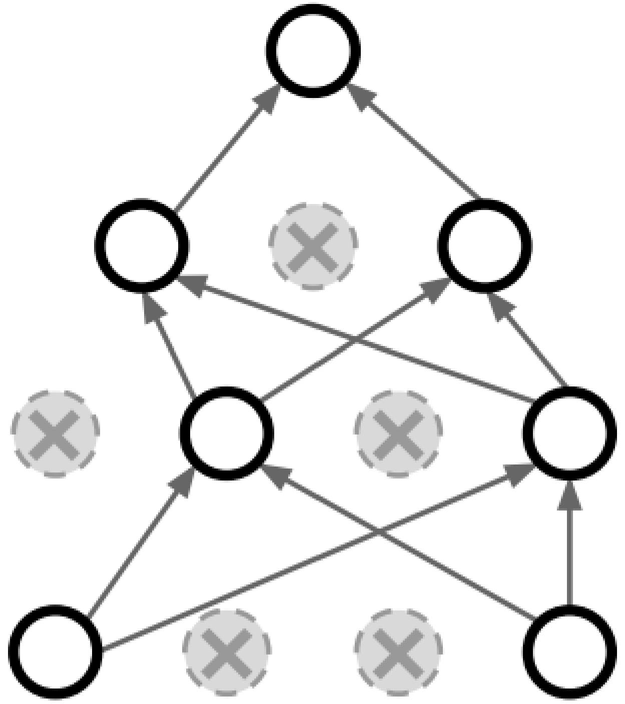


```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) # first dropout mask.
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) # second dropout mask.
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # scale the activations
  out = np.dot(W3, H2) + b3
```

drop in forward pas

scale at test time

Inverted Dropout



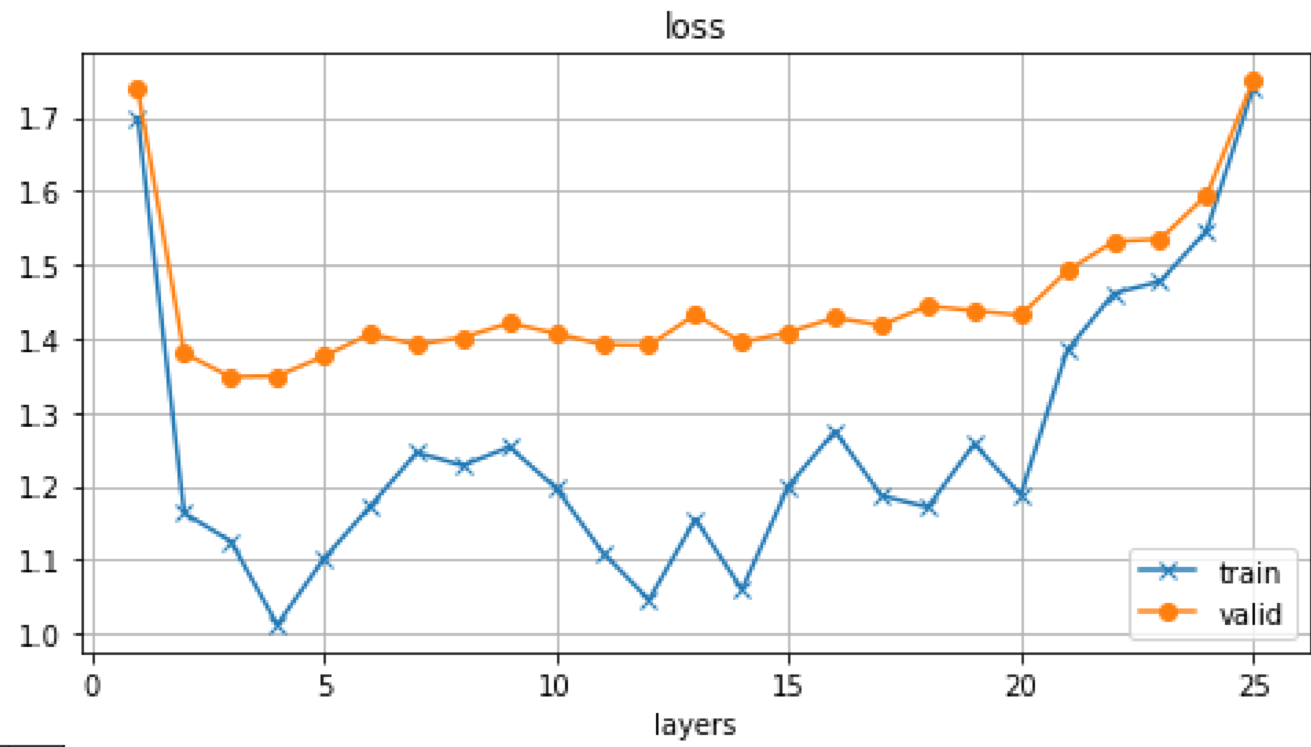
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

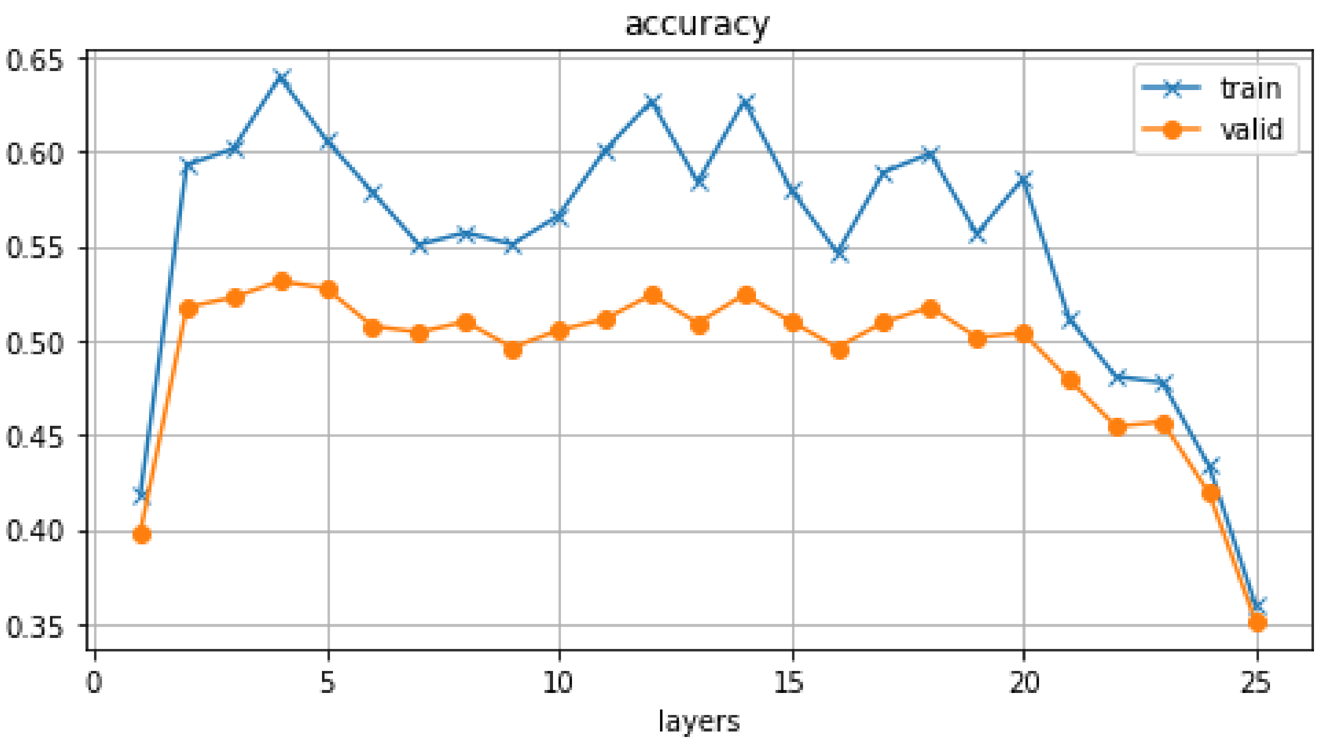
drop in forward pass

test time is unchanged

Convolutional Networks

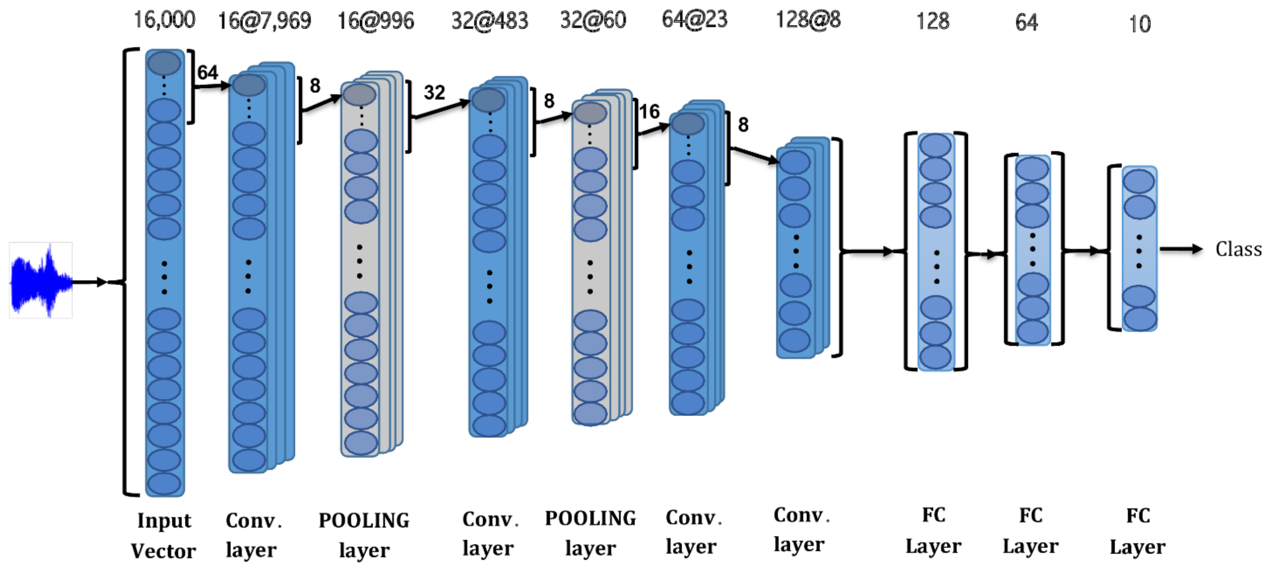
```python
early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss",
                                               min_delta=0,
                                               patience=5,
                                               restore_best_weights=True)

for idx, num_layers in enumerate(range(25)):
    # define model
    model = keras.Sequential()
    model.add(keras.layers.Input(shape=x_train[0].shape))
    model.add(keras.layers.Flatten())
    for l in range(num_layers):
        model.add(keras.layers.Dense(units=512, activation='elu'))
    model.add(keras.layers.Dense(units=num_classes, activation='softmax'))

    # compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    # train model
    history = model.fit(x_train, y_train,
                        batch_size=256,
                        epochs=100,
                        validation_data=(x_test, y_test),
                        callbacks=[early_stopping])

    #
    train_loss[idx], train_acc[idx] = model.evaluate(x_train, y_train, verbose=0)
    valid_loss[idx], valid_acc[idx] = model.evaluate(x_test, y_test, verbose=0)
```

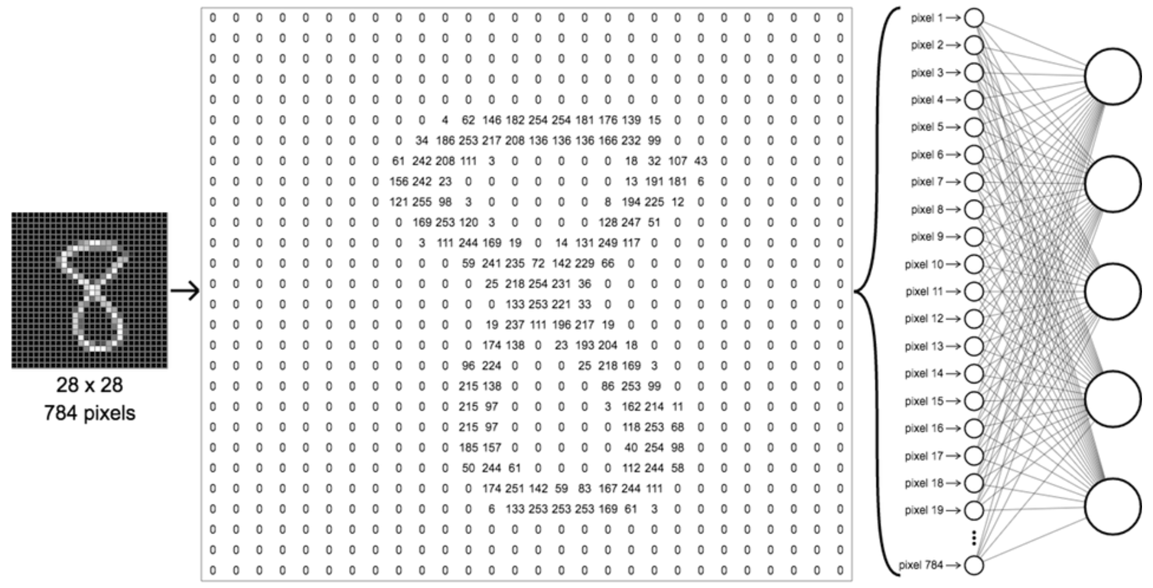

loss
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
5
10
15
20
25
layers
train
valid


accuracy
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
5
10
15
20
25
layers
train
valid

Convolutional Networks

- CNNs, are a specialized kind of NN for processing data that has a known, grid-like topology
- Convolution is a specialized kind of linear operation

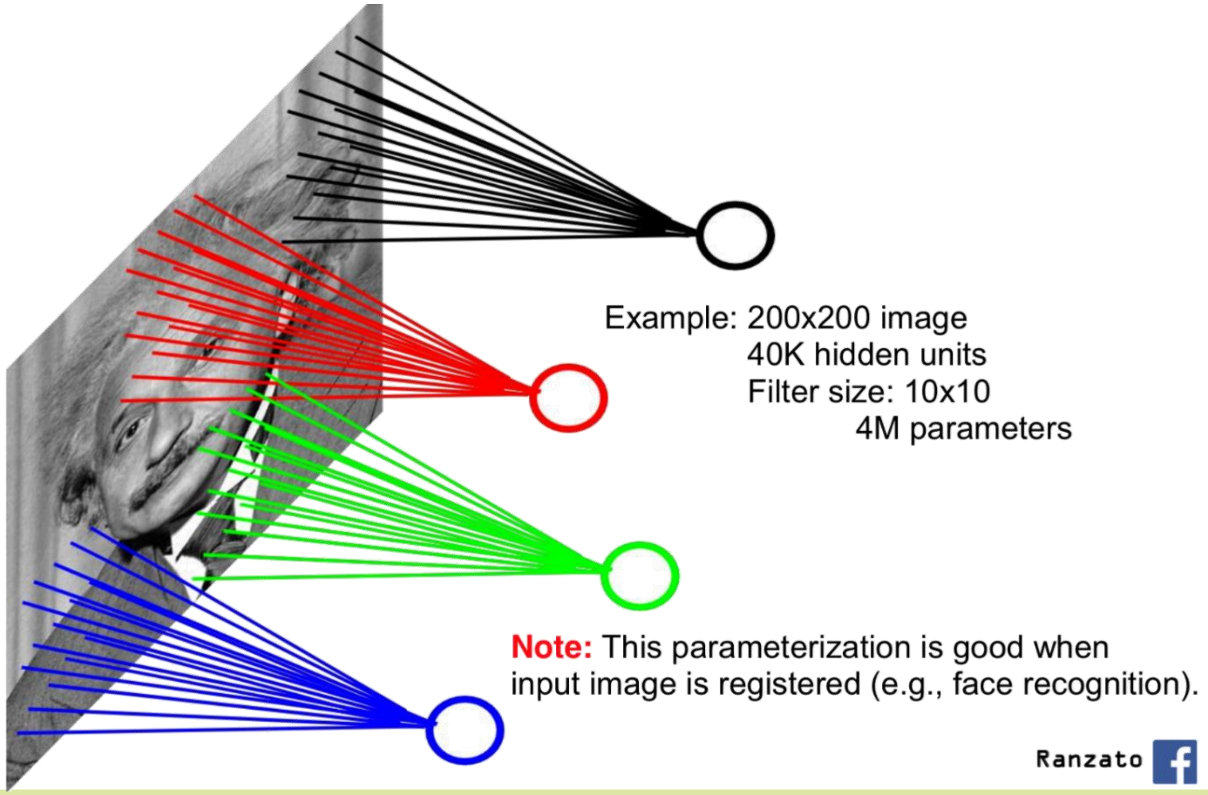




0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Locally Connected Layers

- Many features that the human eye can easily detect are local features
- We can detect edges, textures, and even shapes using pixel intensities in a small region of an image
- If we wanted to detect a feature, we can use the same detector on the bottom-left corner of an image and on the top right of the image
- We can reuse the same weights everywhere else in the image
    - **weight sharing**

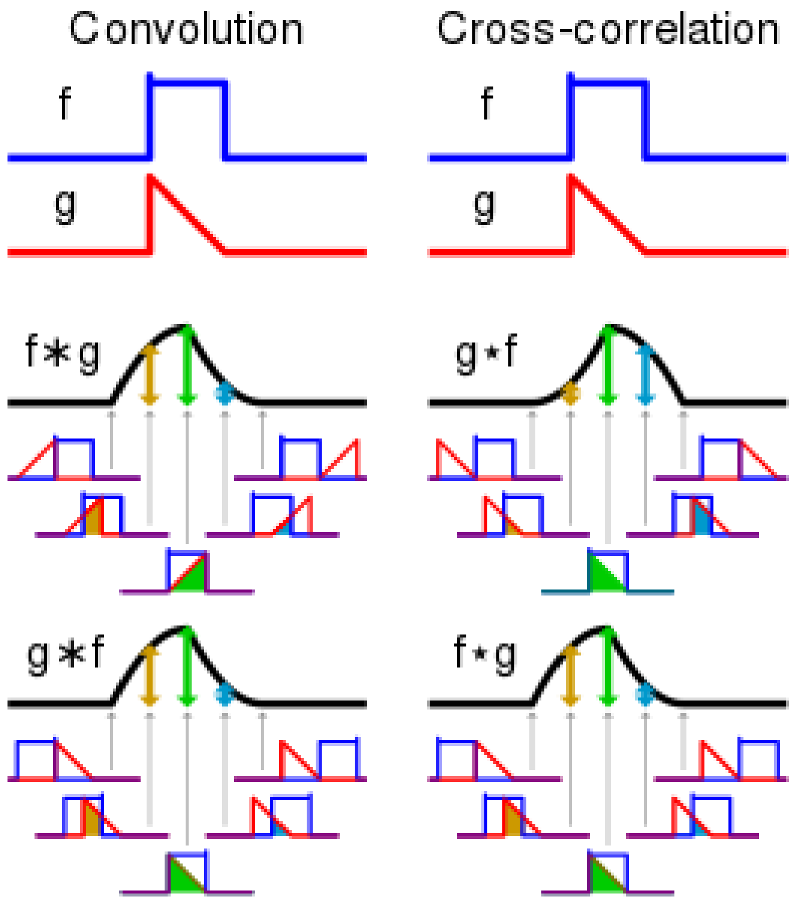

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
Note: This parameterization is good when input image is registered (e.g., face recognition).
Ranzato

Convolution vs Correlation

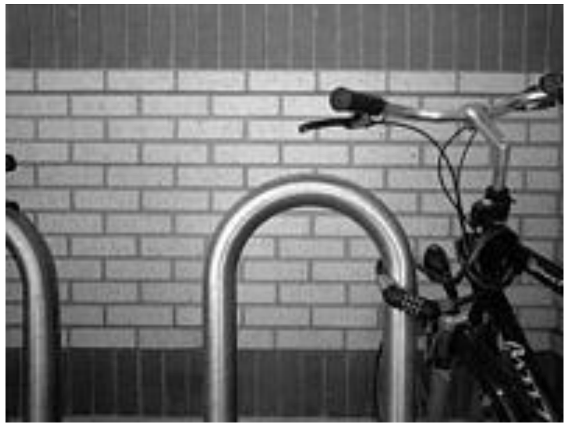
- Many ML libraries implement cross-correlation but call it convolution!
- The learning algorithm will learn the appropriate values of the kernel in the appropriate place

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

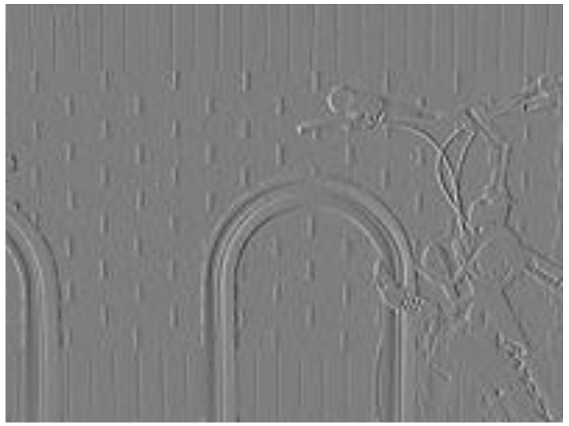


Convolution



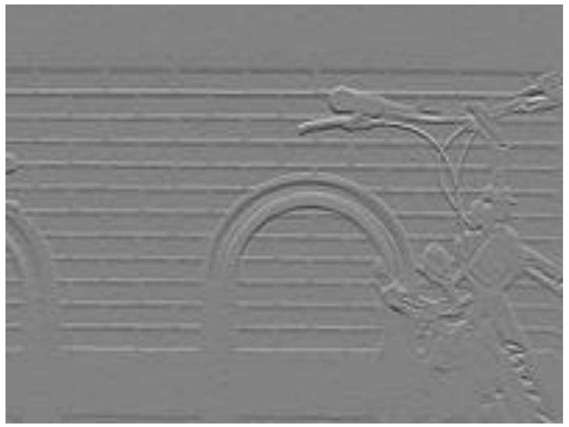
G_y

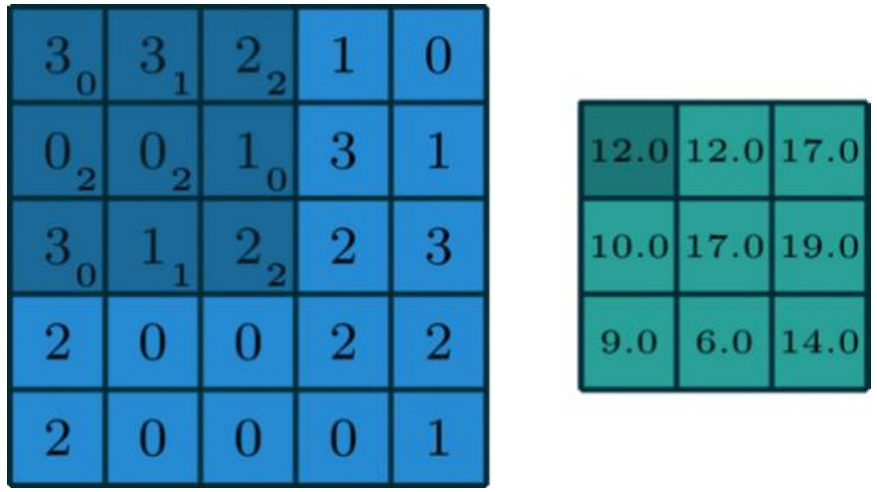
+1	0	-1
+2	0	-2
+1	0	-1



G_x

+1	+2	+1
0	0	0
-1	-2	-1



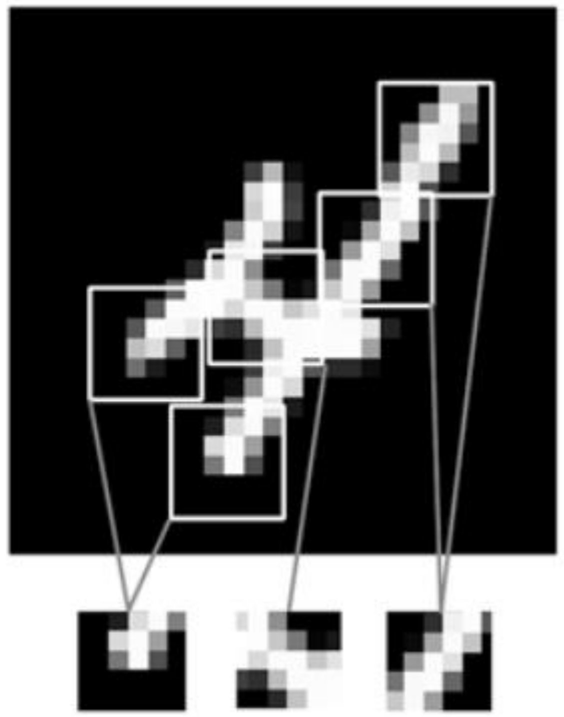


3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

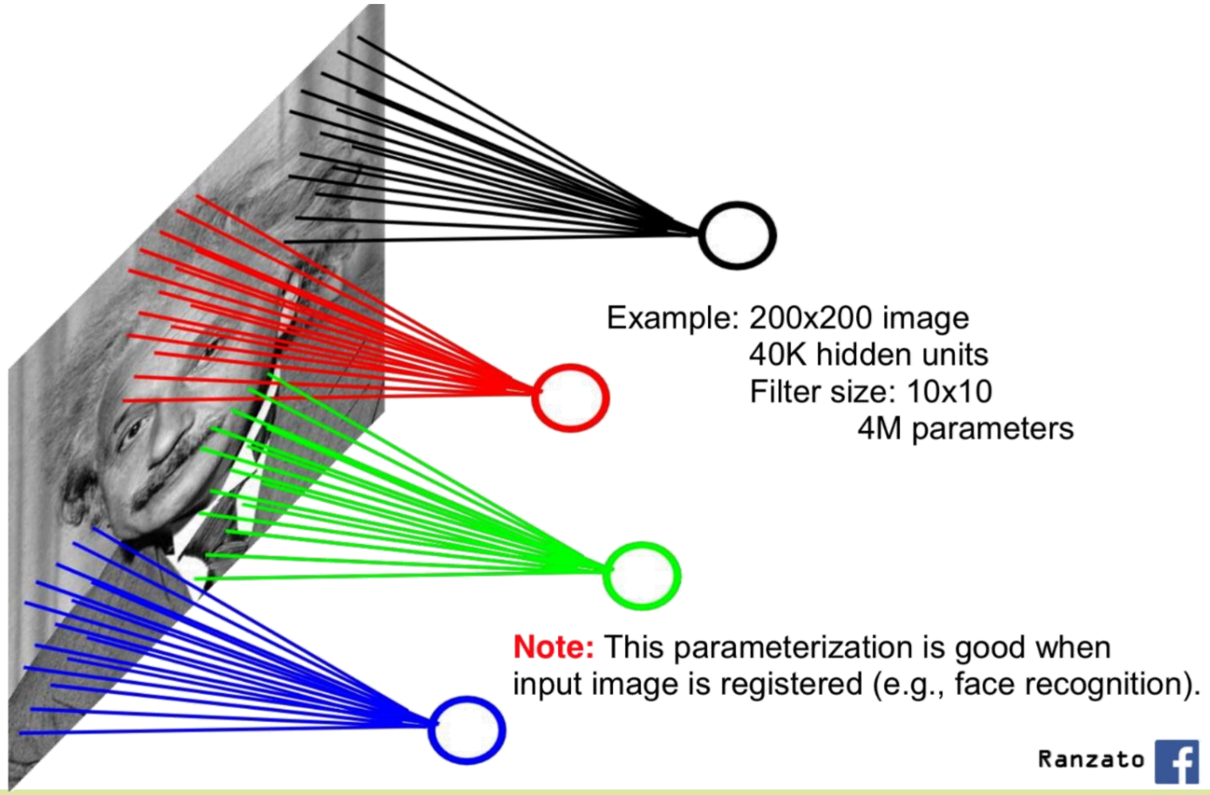
Dense layer vs Convolution layer

- Dense layers learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels)
- Convolution layers learn local patterns



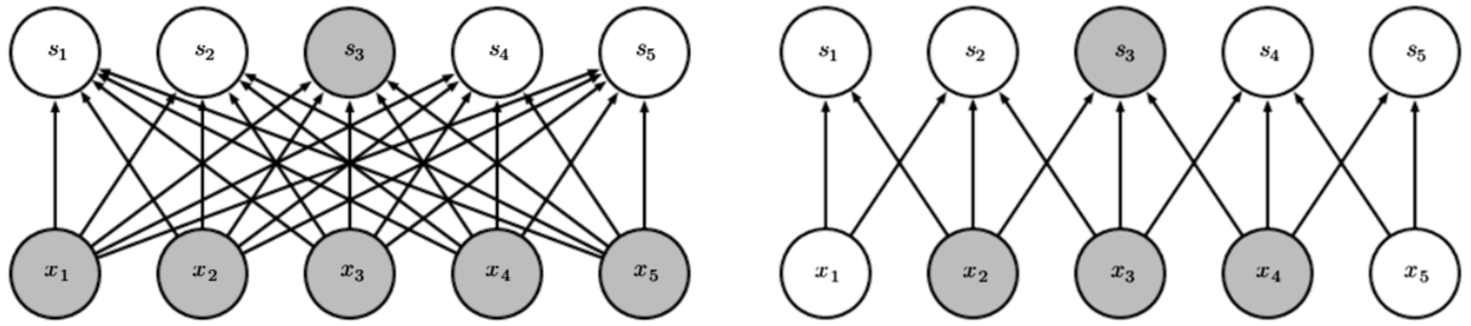
Convolution

- Sparse interactions
- Parameter sharing
- Equivariant representations
- Working with inputs of variable size


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
Note: This parameterization is good when input image is registered (e.g., face recognition).
Ranzato

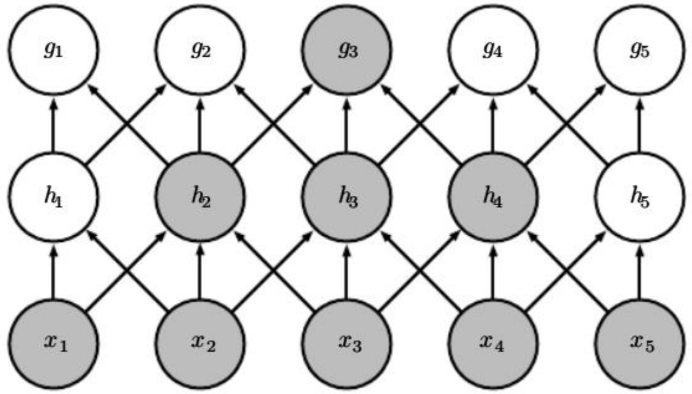
Sparse interactions

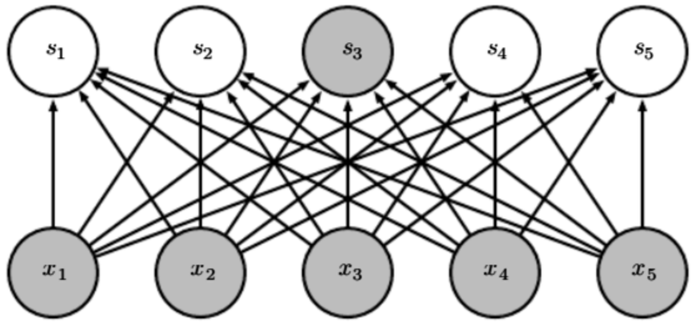
- In traditional NN layers, every output unit interacts with every input unit
- Convolutional networks, typically have sparse interactions
- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels
  - We need to store fewer parameters

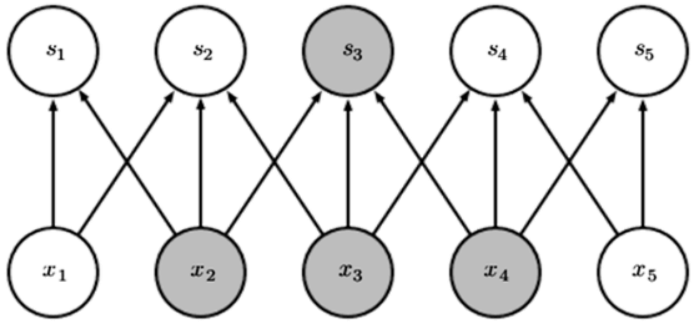


Sparse interactions

- In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input
- This allows the network to efficiently describe complicated interactions between many variables by constructing such sparse interactions
- The receptive field of the units in the deeper layers can be very large

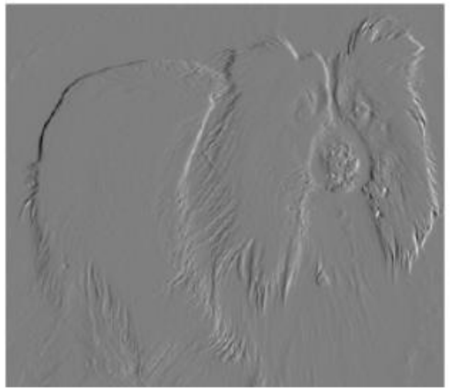
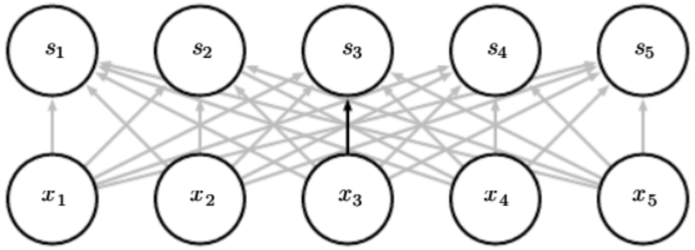
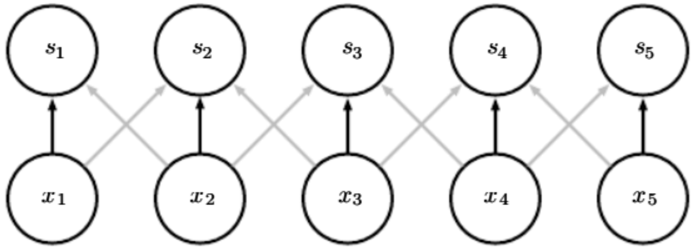

g_1 g_2 g_3 g_4 g_5
h_1 h_2 h_3 h_4 h_5
x_1 x_2 x_3 x_4 x_5





Parameter sharing

- Parameter sharing refers to using the same parameter for more than one function in a model
- In a convolutional neural net, each member of the kernel is used at every position of the input
- The parameter sharing means that rather than learning a separate set of parameters for every location, we learn only one set



s_1
s_2
s_3
s_4
s_5
x_1
x_2
x_3
x_4
x_5

s_1
s_2
s_3
s_4
s_5
x_1
x_2
x_3
x_4
x_5