DS 5220

Supervised Machine Learning and Learning Theory

Alina Oprea
Associate Professor, CCIS
Northeastern University

November 13 2019

Logistics

- Project milestone
  - Due on Nov. 18 in Gradescope
- Template
  - Problem description
  - Dataset
  - Approach and methodology
  - Remaining work
  - Team member contribution
- Last homework will be released this week

Outline

- Feed-Forward architectures
  - Multi-class classification (softmax unit)
  - Lab in Keras
- Convolutional Neural Networks
  - Convolution layer
  - Max pooling layer
  - Examples of famous architectures

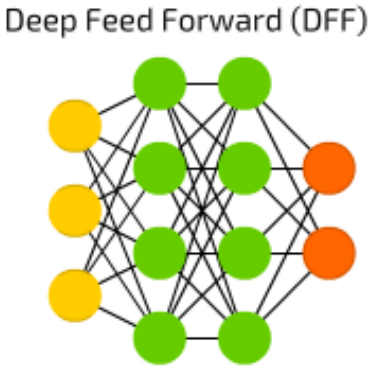
References

- Deep Learning books
  - https://www.deeplearningbook.org/
  - http://d2l.ai/
- Stanford notes on deep learning
  - http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf
- History of Deep Learning
  - https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Neural Network Architectures

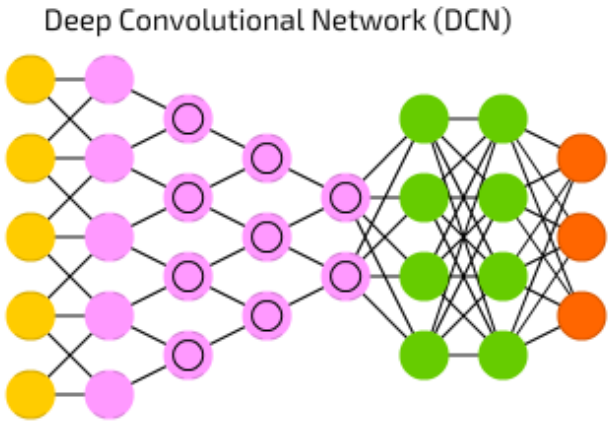
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



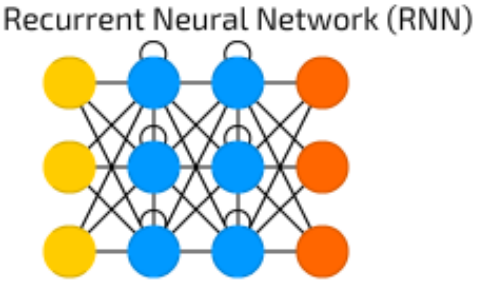
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

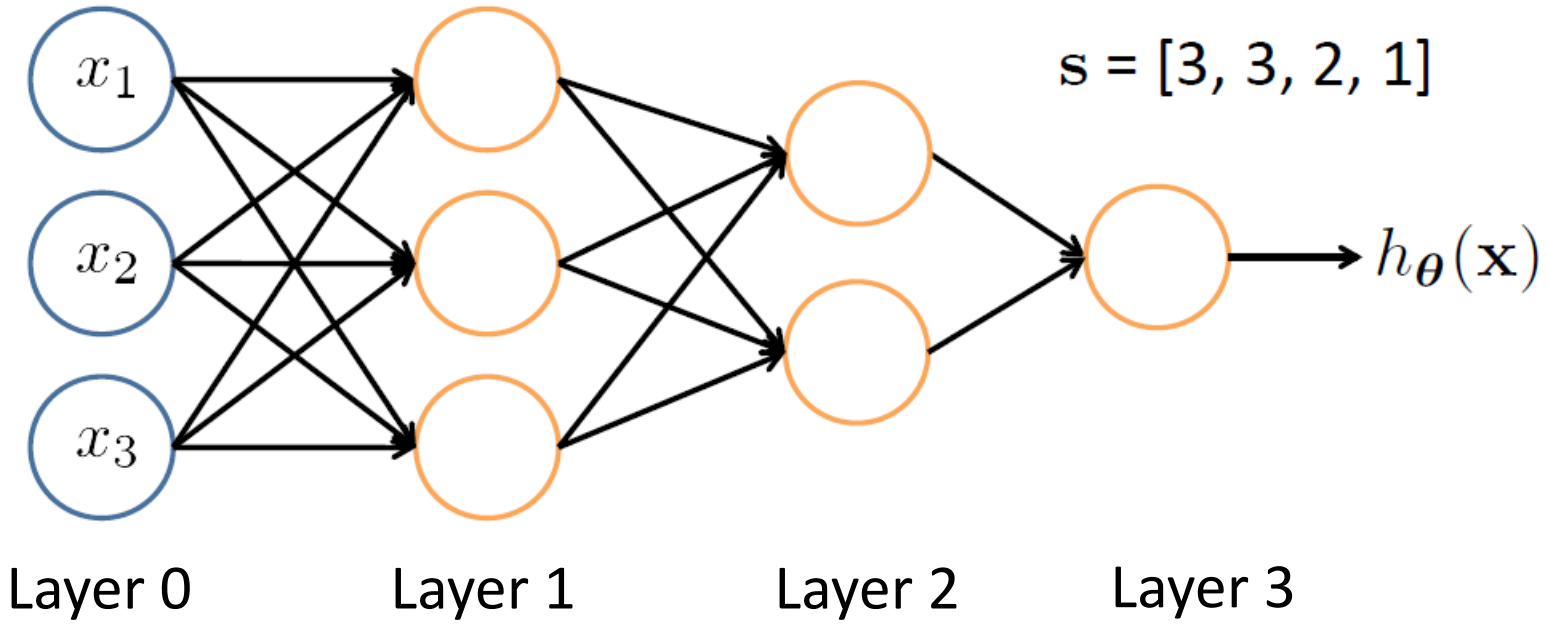


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Feed-Forward Networks

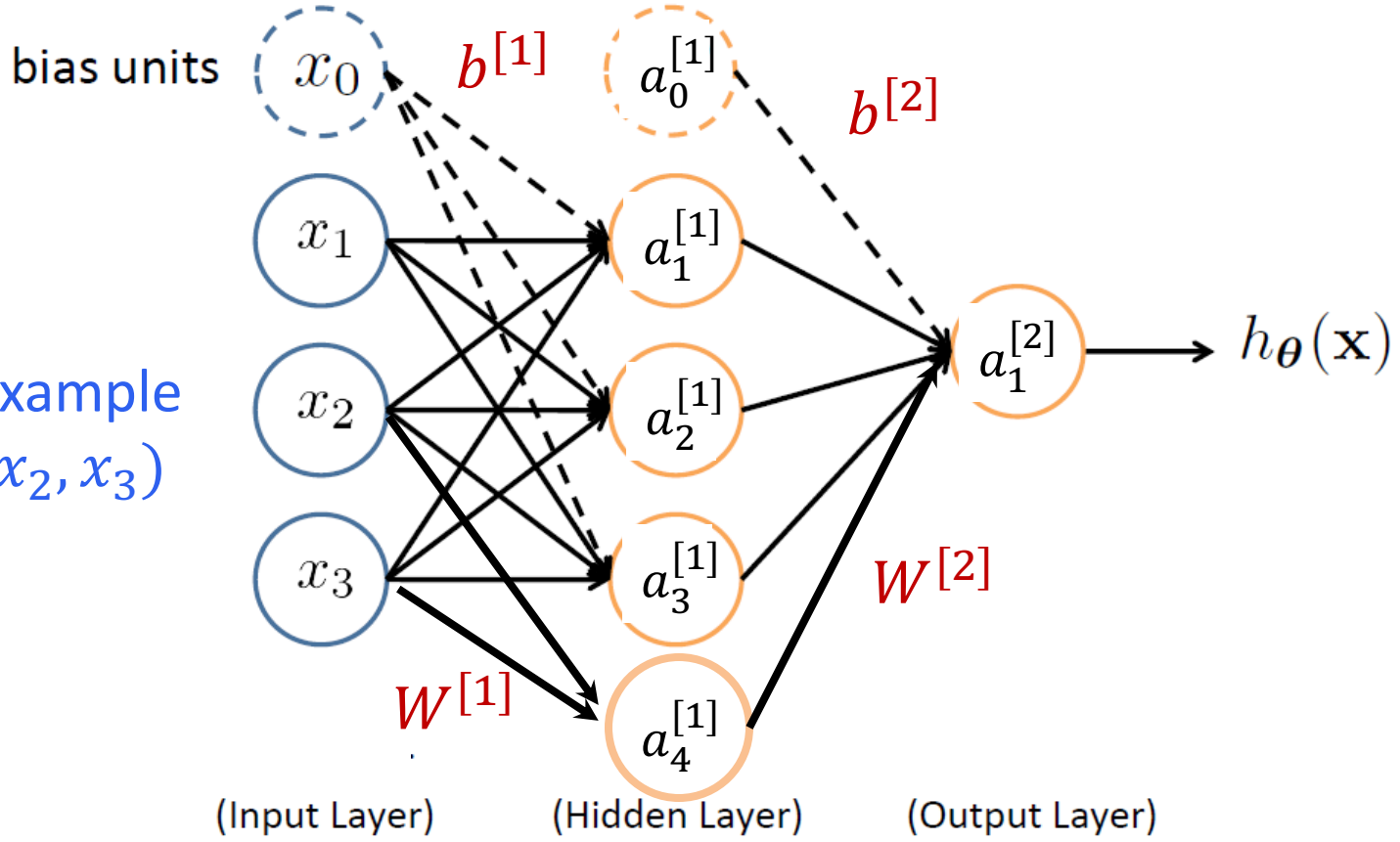


L denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1}=K$ (# classes)

Feed-Forward Neural Network



Training example
$x = (x_1, x_2, x_3)$

No cycles

$\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Vectorization

$$z_1^{[1]} = W_1^{[1]^T} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$z_4^{[1]} = W_4^{[1]^T} x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})$$

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — & W_1^{[1]^T} & — \\ — & W_2^{[1]^T} & — \\ & \vdots & \\ — & W_4^{[1]^T} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

$$z^{[1]} = W^{[1]} x + b^{[1]}$$

Linear

$$a^{[1]} = g(z^{[1]})$$

Non-Linear

Vectorization

Output layer

$$z_1^{[2]} = W_1^{[2]T} a^{[1]} + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

$$\underbrace{z^{[2]}}_{1\times1} = \underbrace{W^{[2]}}_{1\times4}\underbrace{a^{[1]}}_{4\times1} + \underbrace{b^{[2]}}_{1\times1} \quad \text{and} \quad \underbrace{a^{[2]}}_{1\times1} = g(\underbrace{z^{[2]}}_{1\times1})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

Linear

$$a^{[2]} = g(z^{[2]})$$

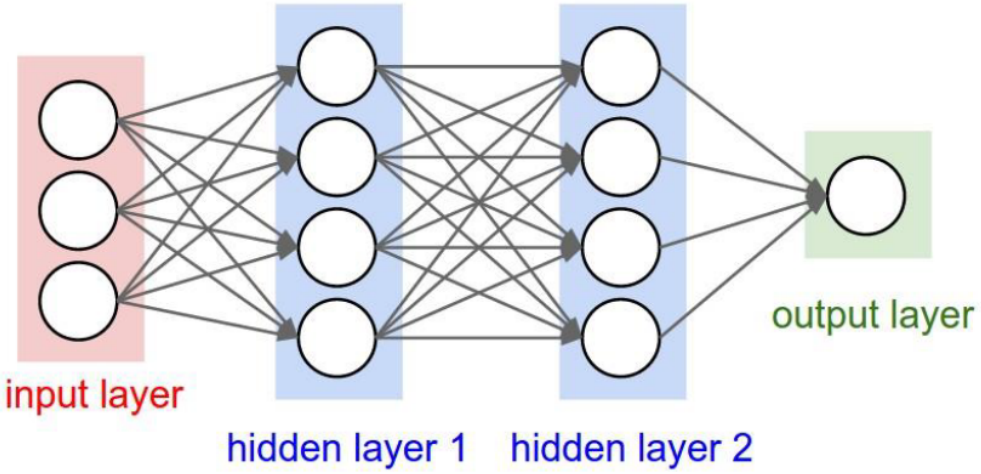
Non-Linear

Hidden Units

- Layer 1
  - First hidden unit:
    - Linear: $z_1^{[1]} = \mathrm{W}_1^{[1]\,\mathrm{T}} x + \mathrm{b}_1^{[1]}$
    - Non-linear: $a_1^{[1]} = \mathrm{g}(\mathrm{z}_1^{[1]})$
  - …
  - Fourth hidden unit:
    - Linear: $z_4^{[1]} = \mathrm{W}_4^{[1]\,\mathrm{T}} x + \mathrm{b}_4^{[1]}$
    - Non-linear: $a_4^{[1]} = \mathrm{g}(\mathrm{z}_4^{[1]})$
- Terminology
  - $a_i^{[j]}$ - Activation of unit i in layer j
  - g - Activation function
  - $W^{[j]}$ - Weight vector controlling mapping from layer j-1 to j
  - $b^{[j]}$ - Bias vector from layer j-1 to j

Forward Propagation

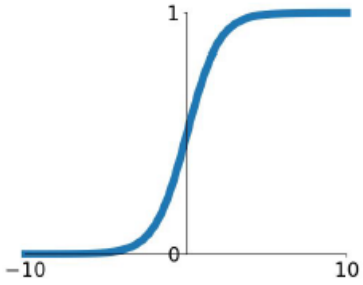
- The input neurons first receive the data features of the object. After processing the data, they send their output to the first hidden layer.
- The hidden layer processes this output and sends the results to the next hidden layer.
- This continues until the data reaches the final output layer, where the output value determines the object's classification.
- This entire process is known as Forward Propagation, or Forward prop.



Activation Functions

Sigmoid

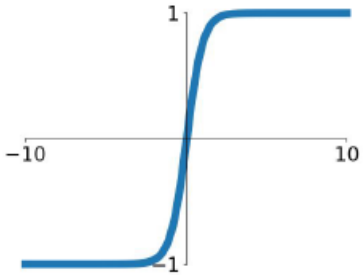
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Binary Classification

tanh

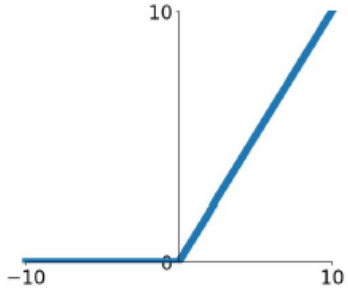
$$\tanh(x)$$



Regression

ReLU

$$\max(0, x)$$



Intermediary layers

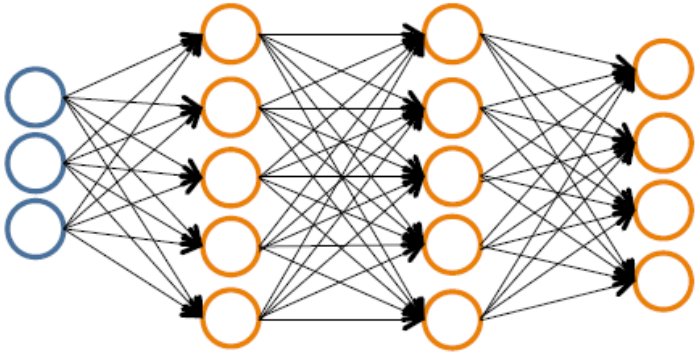
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

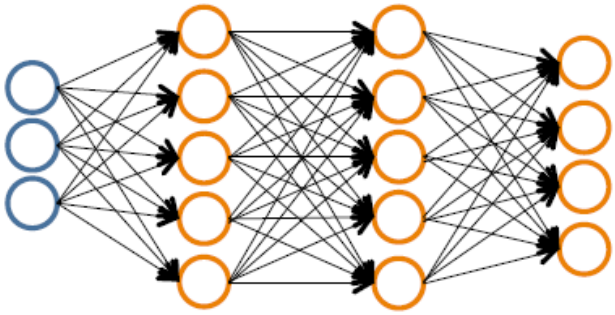
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Multiple Output Units: One-vs-Rest



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle

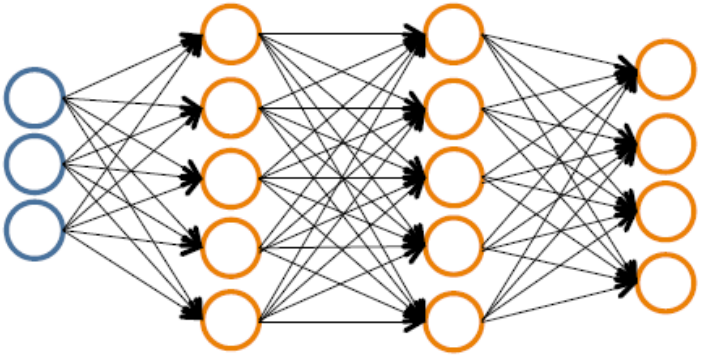
$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ when truck

---

- Given $\{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of-K representation
  - e.g., $\mathbf{y}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $\mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Based on slide by Andrew Ng

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Sigmoid

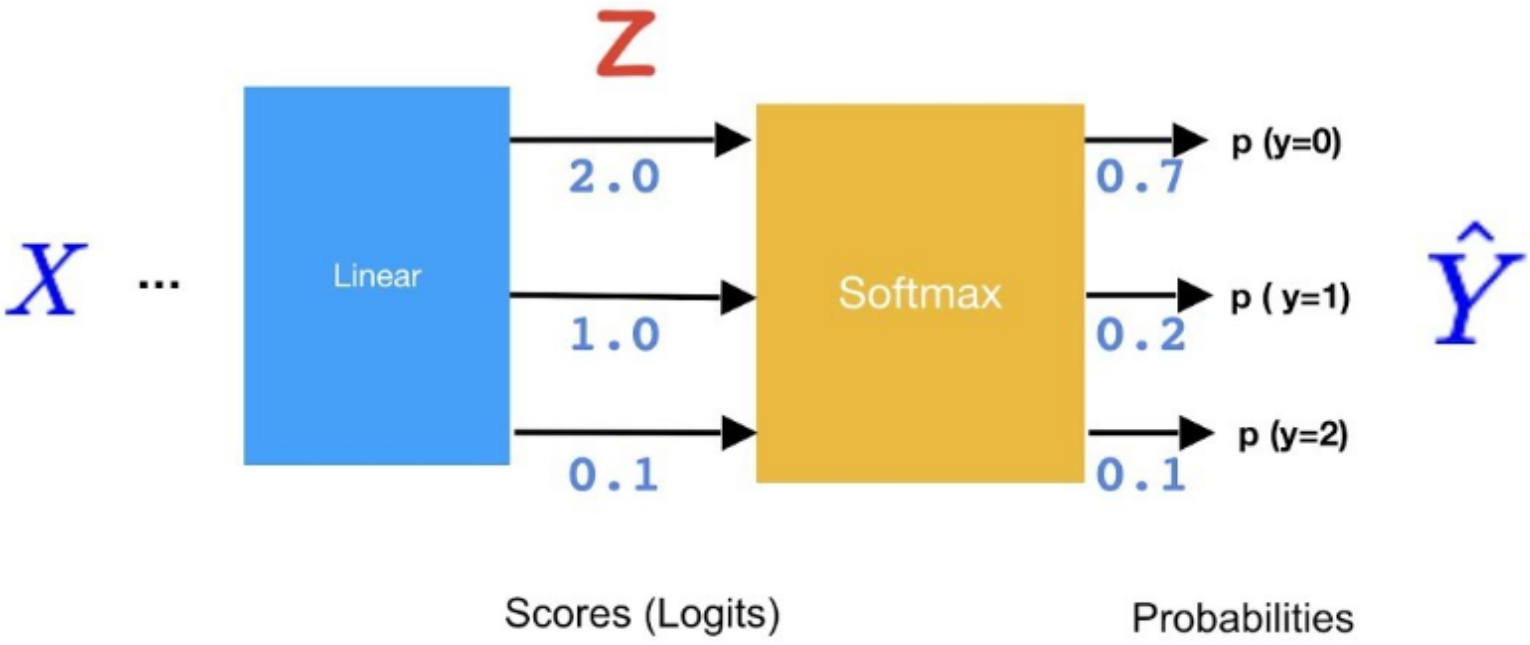
Multi-class classification (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ pedestrian, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ car, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ motorcycle, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ truck

K output units $(s_{L-1} = K)$

Softmax

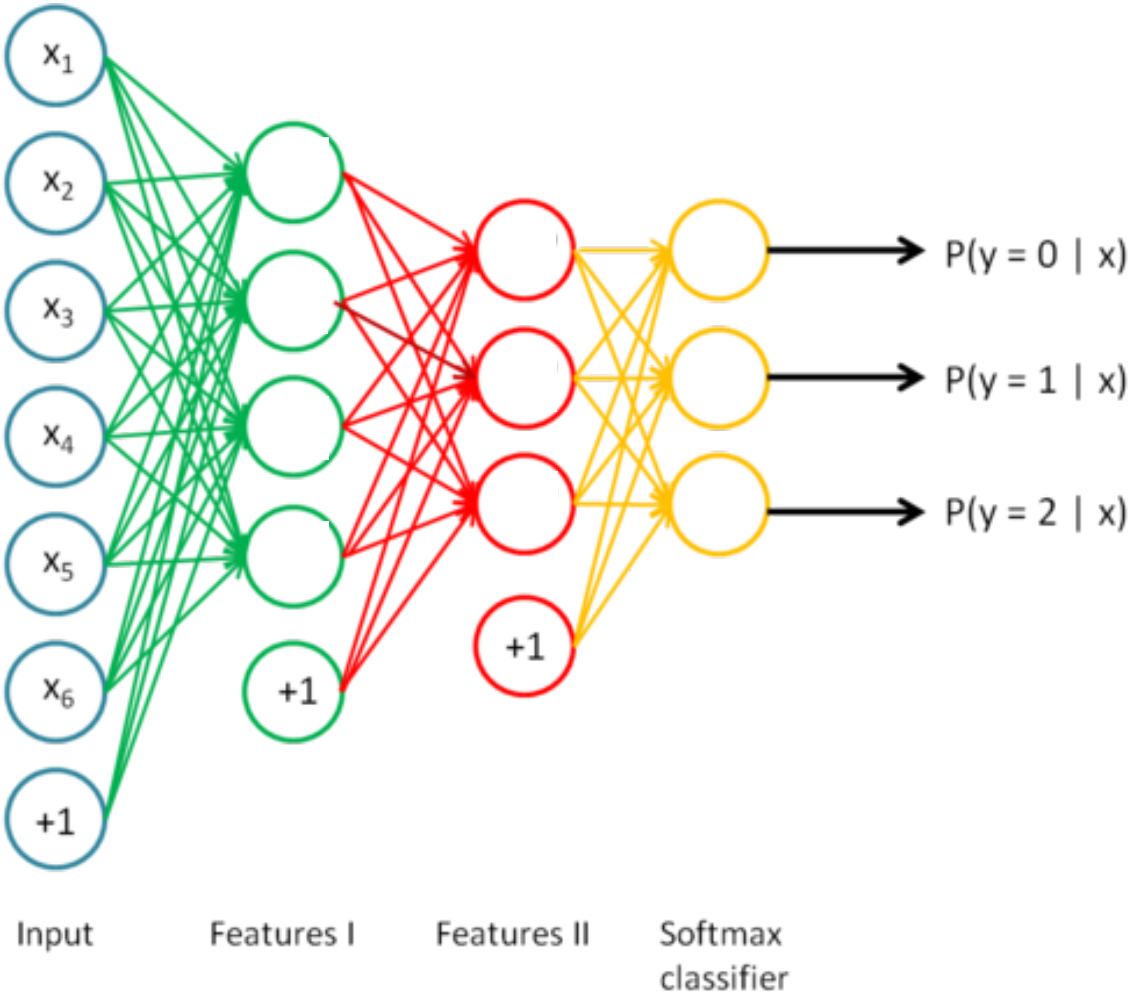
Softmax classifier



$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

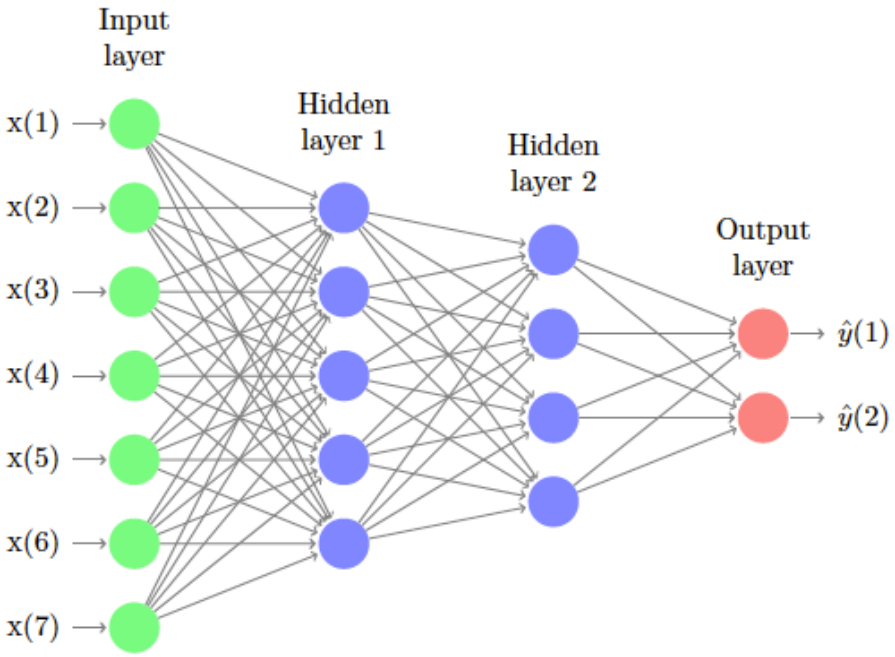
Multi-class classification



Review Feed-Forward Neural Networks

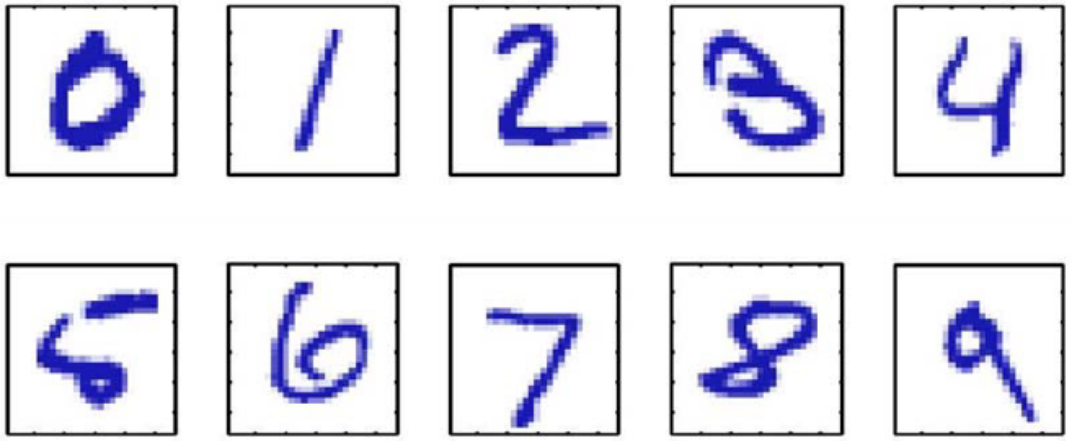
- Simplest architecture of NN
- Neurons from one layer are connected to neurons from next layer
  - Input layer has feature space dimension
  - Output layer has number of classes
  - Hidden layers use linear operations, followed by non-linear activation function
  - Multi-Layer Perceptron (MLP): fully connected layers
- Activation functions
  - Sigmoid for binary classification in last layer
  - Softmax for multi-class classification in last layer
  - ReLU for hidden layers
- Forward propagation is the computation of the network output given an input
- Back propagation is the training of a network
  - Determine weights and biases at every layer

FFNN Architectures



- Input and Output Layers are completely specified by the problem domain
- In the Hidden Layers, number of neurons in Layer i+1 is always smaller than number of neurons in Layer i

MNIST: Handwritten digit recognition



Images are 28 x 28 pixels

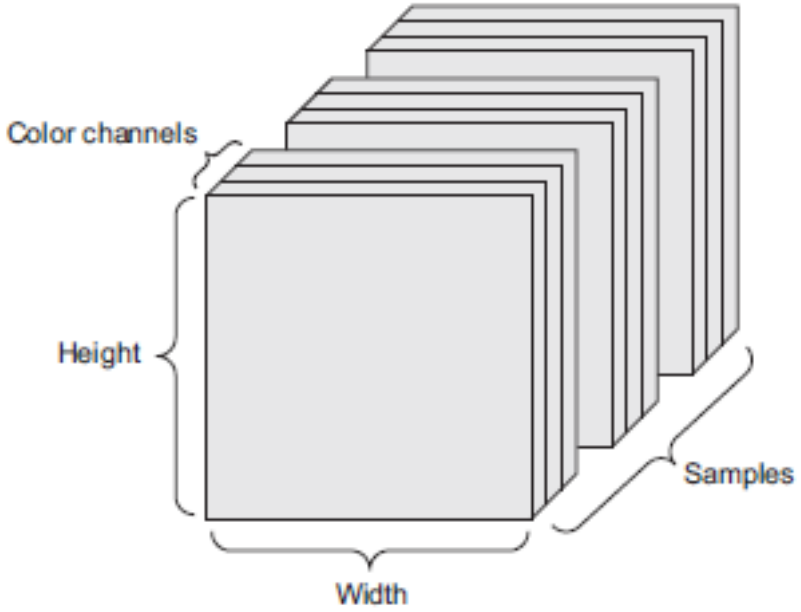
Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$
Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Predict the digit
Multi-class classifier

Image Representation

- Image is 3D “tensor”: height, width, color channel (RGB)
- Black-and-white images are 2D matrices: height, width
  - Each value is pixel intensity



Lab – Feed Forward NN

```
import time
import numpy as np
from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist

import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
```

Import modules

```
def load_data():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

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

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 784))
    X_test = np.reshape(X_test, (10000, 784))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Load MNIST data Processing

Vector representation

Neural Network Architecture

```
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(10, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

10 hidden units
ReLU activation

Output Layer
Softmax activation

Loss function

Optimizer

Feed-Forward Neural Network Architecture

- 1 Hidden Layer (“Dense” or Fully Connected)
- 10 neurons
- Output layer uses softmax activation

Train and evaluate

```
def run_network(data=None, model=None, epochs=10, batch=256):
    try:
        start_time = time.time()
        if data is None:
            X_train, X_test, y_train, y_test = load_data()
        else:
            X_train, X_test, y_train, y_test = data

        if model is None:
            model = init_model()

        print("Training model")
        history = model.fit(X_train, y_train, nb_epoch=epochs, batch_size=batch,
                            validation_data=(X_test, y_test), verbose=2)

        print("Training duration:"+format(time.time() - start_time))
        score = model.evaluate(X_test, y_test, batch_size=16)

        print("\nNetwork's test loss and accuracy:"+format(score))
        return model, history
    except KeyboardInterrupt:
        print("KeyboardInterrupt")
        return model, history
```

Training/testing results

```
2s - loss: 0.9114 - acc: 0.7649 - val_loss: 0.4499 - val_acc: 0.8819
Epoch 2/10
0s - loss: 0.3935 - acc: 0.8907 - val_loss: 0.3378 - val_acc: 0.9049
Epoch 3/10
0s - loss: 0.3296 - acc: 0.9063 - val_loss: 0.3042 - val_acc: 0.9128
Epoch 4/10
0s - loss: 0.3036 - acc: 0.9132 - val_loss: 0.2889 - val_acc: 0.9181
Epoch 5/10
0s - loss: 0.2888 - acc: 0.9189 - val_loss: 0.2874 - val_acc: 0.9185
Epoch 6/10
0s - loss: 0.2785 - acc: 0.9210 - val_loss: 0.2703 - val_acc: 0.9257
Epoch 7/10
0s - loss: 0.2705 - acc: 0.9241 - val_loss: 0.2718 - val_acc: 0.9239
Epoch 8/10
0s - loss: 0.2649 - acc: 0.9257 - val_loss: 0.2694 - val_acc: 0.9240
Epoch 9/10
0s - loss: 0.2601 - acc: 0.9264 - val_loss: 0.2616 - val_acc: 0.9261
Epoch 10/10
0s - loss: 0.2561 - acc: 0.9277 - val_loss: 0.2607 - val_acc: 0.9274
Training duration:10.31288456916809
 9840/10000 [============================>.] - ETA: 0s
Network's test loss and accuracy:[0.26067940444946291, 0.9274]
```

Epoch Output

Metrics

- Loss
- Accuracy

Reported on both training and validation

Changing Number of Neurons

```
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

500 hidden units

```
2s - loss: 0.3169 - acc: 0.9088 - val_loss: 0.1652 - val_acc: 0.9502
Epoch 2/10
0s - loss: 0.1277 - acc: 0.9626 - val_loss: 0.1071 - val_acc: 0.9679
Epoch 3/10
0s - loss: 0.0847 - acc: 0.9749 - val_loss: 0.0861 - val_acc: 0.9731
Epoch 4/10
0s - loss: 0.0607 - acc: 0.9822 - val_loss: 0.0746 - val_acc: 0.9767
Epoch 5/10
0s - loss: 0.0471 - acc: 0.9863 - val_loss: 0.0655 - val_acc: 0.9796
Epoch 6/10
0s - loss: 0.0359 - acc: 0.9895 - val_loss: 0.0636 - val_acc: 0.9813
Epoch 7/10
0s - loss: 0.0280 - acc: 0.9920 - val_loss: 0.0599 - val_acc: 0.9810
Epoch 8/10
0s - loss: 0.0223 - acc: 0.9937 - val_loss: 0.0678 - val_acc: 0.9795
Epoch 9/10
0s - loss: 0.0174 - acc: 0.9952 - val_loss: 0.0607 - val_acc: 0.9815
Epoch 10/10
0s - loss: 0.0134 - acc: 0.9964 - val_loss: 0.0672 - val_acc: 0.9806
Training duration:10.458189249038696
 9456/10000 [===========================>..] - ETA: 0s
Network's test loss and accuracy:[0.067179036217656543, 0.98060000000000003]
```

Two Layers

```
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()

    # Hidden Layer 1
    model.add(Dense(500, input_dim=784))
    model.add(Activation('relu'))

    # Hidden Layer 2
    model.add(Dense(300))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

Layer 1

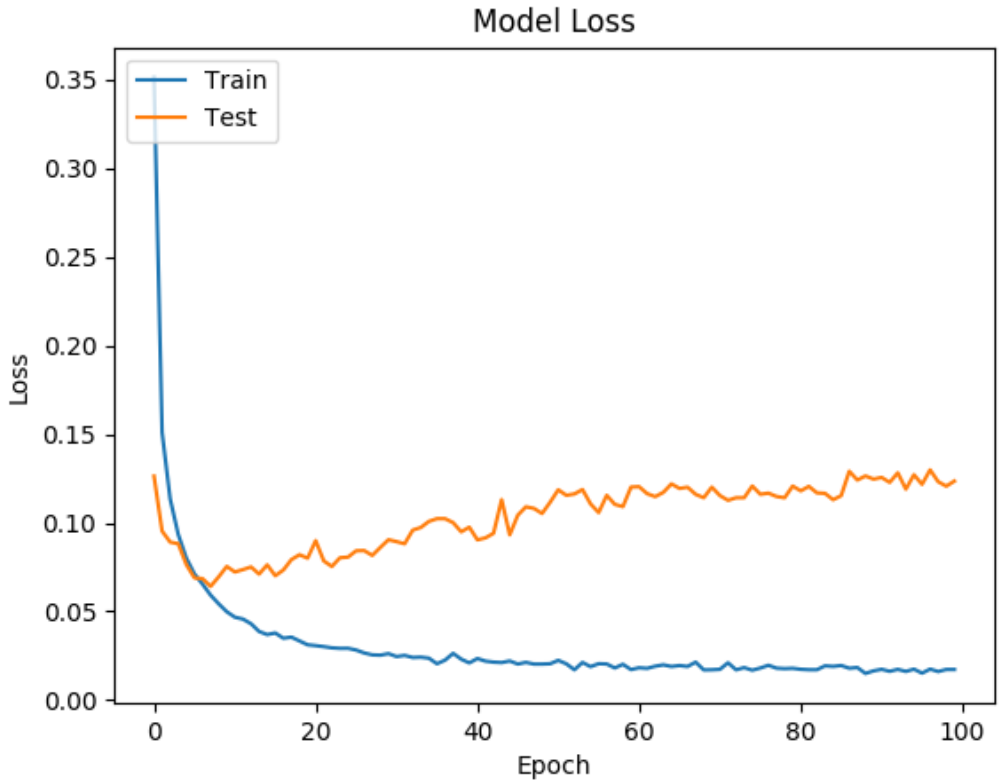
Layer 2

Output Softmax Layer

```
2s - loss: 0.2800 - acc: 0.9132 - val_loss: 0.1821 - val_acc: 0.9409
Epoch 2/10
1s - loss: 0.0974 - acc: 0.9699 - val_loss: 0.0951 - val_acc: 0.9703
Epoch 3/10
0s - loss: 0.0616 - acc: 0.9803 - val_loss: 0.0843 - val_acc: 0.9754
Epoch 4/10
0s - loss: 0.0429 - acc: 0.9862 - val_loss: 0.0670 - val_acc: 0.9809
Epoch 5/10
0s - loss: 0.0303 - acc: 0.9904 - val_loss: 0.0820 - val_acc: 0.9777
Epoch 6/10
0s - loss: 0.0233 - acc: 0.9922 - val_loss: 0.0794 - val_acc: 0.9783
Epoch 7/10
0s - loss: 0.0180 - acc: 0.9941 - val_loss: 0.0866 - val_acc: 0.9792
Epoch 8/10
0s - loss: 0.0137 - acc: 0.9956 - val_loss: 0.0788 - val_acc: 0.9814
Epoch 9/10
0s - loss: 0.0116 - acc: 0.9963 - val_loss: 0.0901 - val_acc: 0.9795
Epoch 10/10
1s - loss: 0.0100 - acc: 0.9966 - val_loss: 0.0812 - val_acc: 0.9827
Training duration:11.816290140151978
 9744/10000 [============================>.] - ETA: 0s
```

Monitor Loss

```
def plot_losses(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
    plt.savefig('output.png')
```



Model Parameters

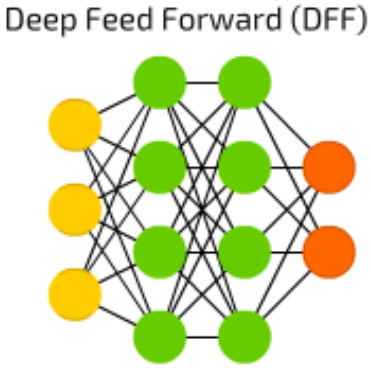
```
model.summary()
```

```
Using TensorFlow backend.
Loading data
Data loaded
Compiling Model
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               392500
_________________________________________________________________
activation_1 (Activation)    (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 300)               150300
_________________________________________________________________
activation_2 (Activation)    (None, 300)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                3010
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 545,810
Trainable params: 545,810
Non-trainable params: 0
```

Neural Network Architectures

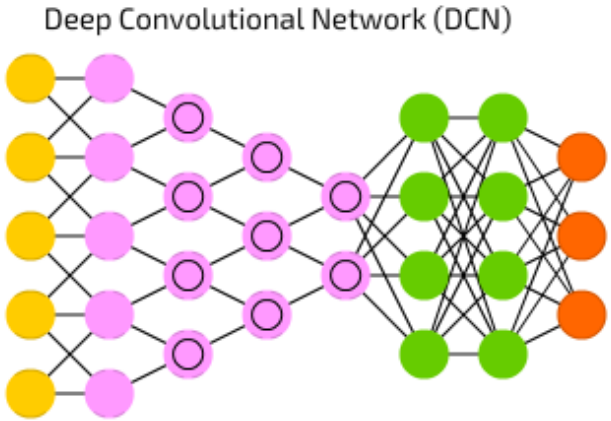
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



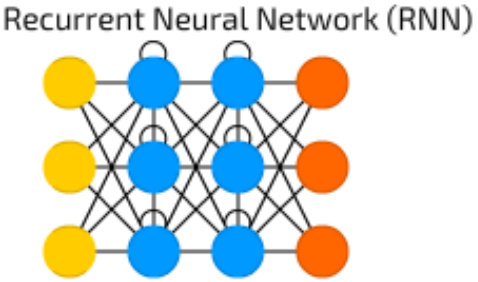
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations



Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



Convolutional Neural Networks

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

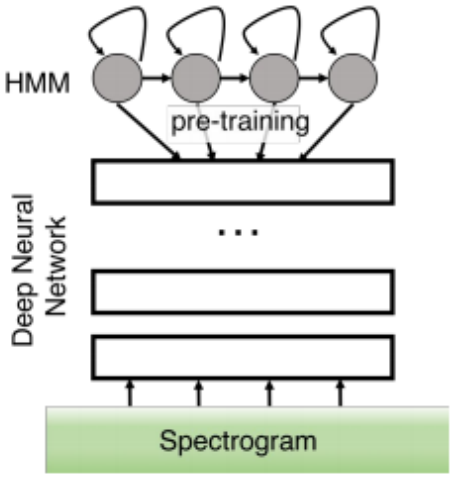
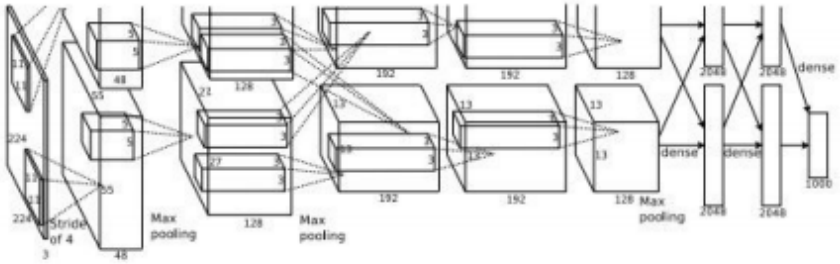
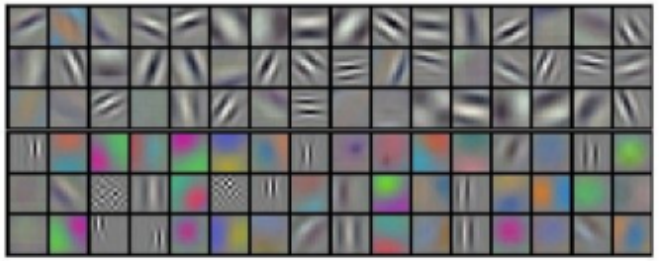


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Convolutional Nets

Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

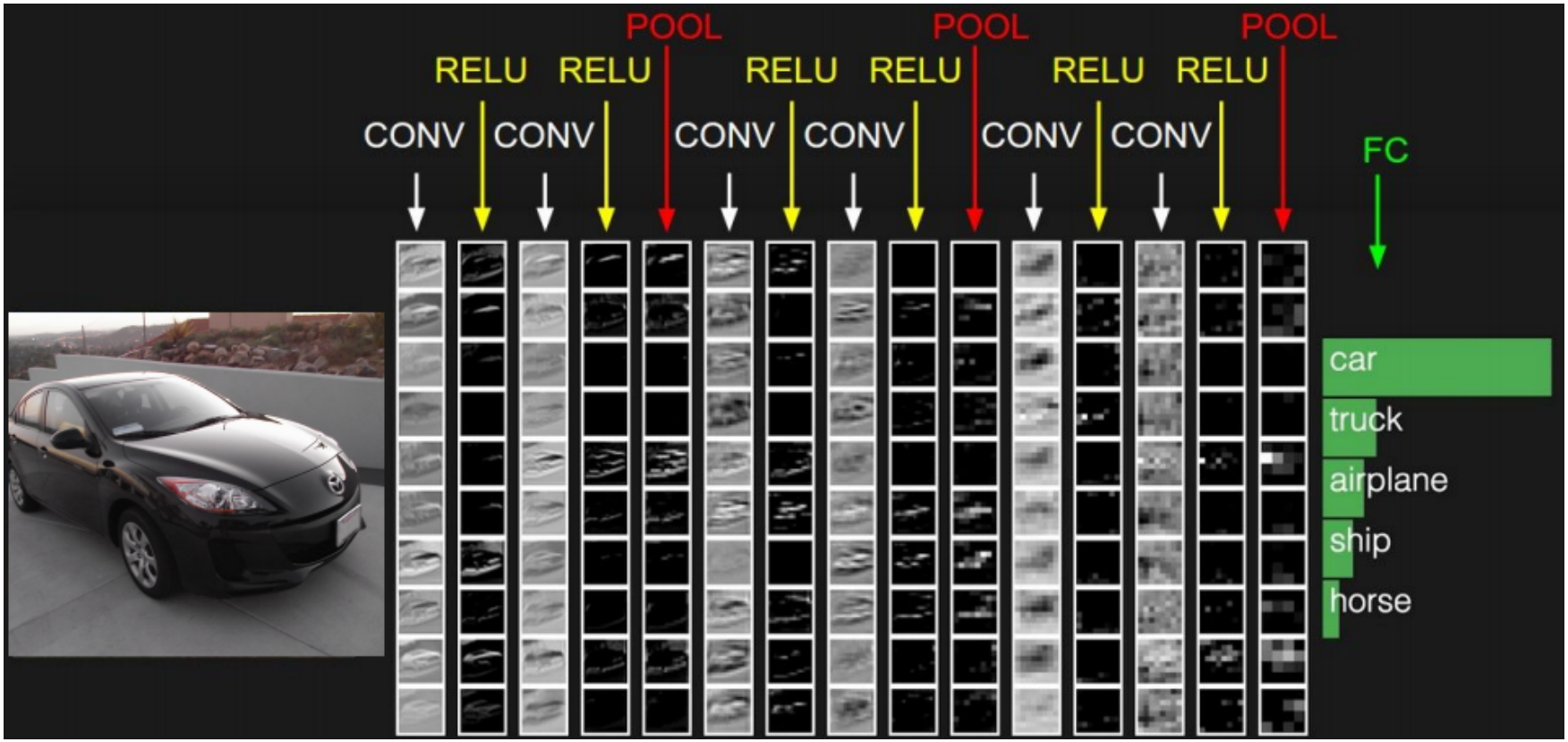
- Object recognition
- Steering angle prediction
- Assist drivers in making decisions

- Image captioning

Convolutional Nets

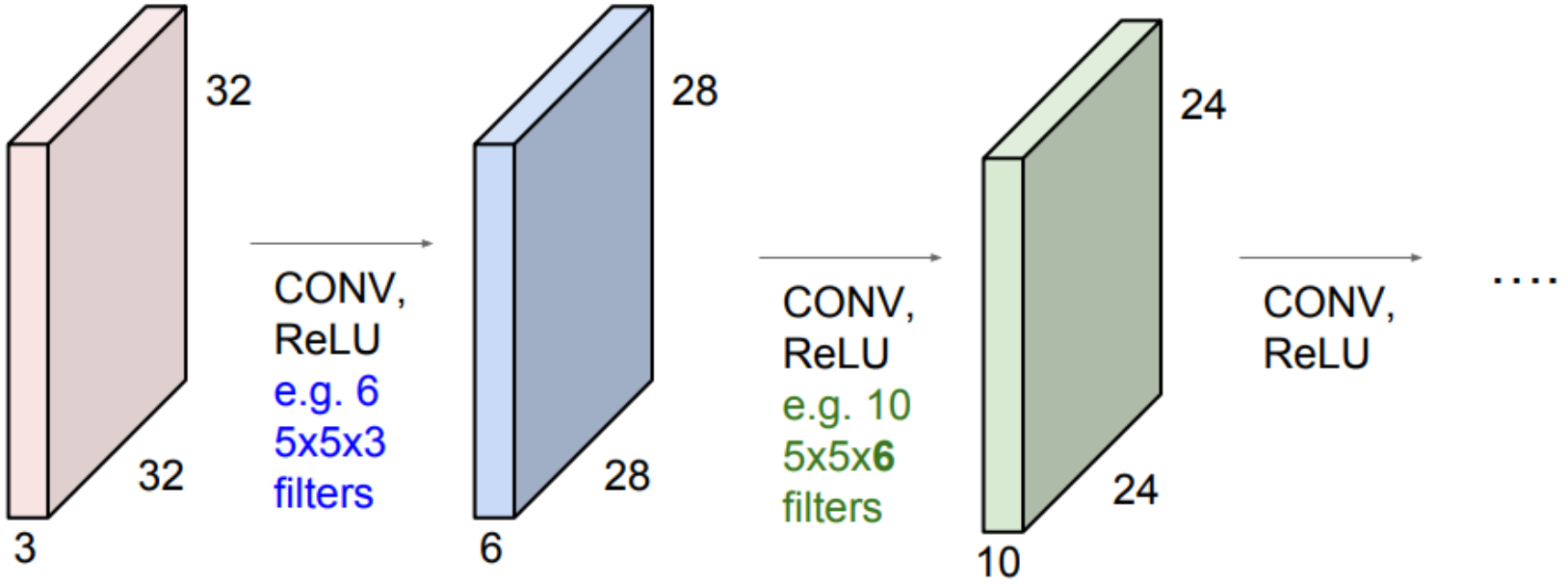
- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Convolutional Nets



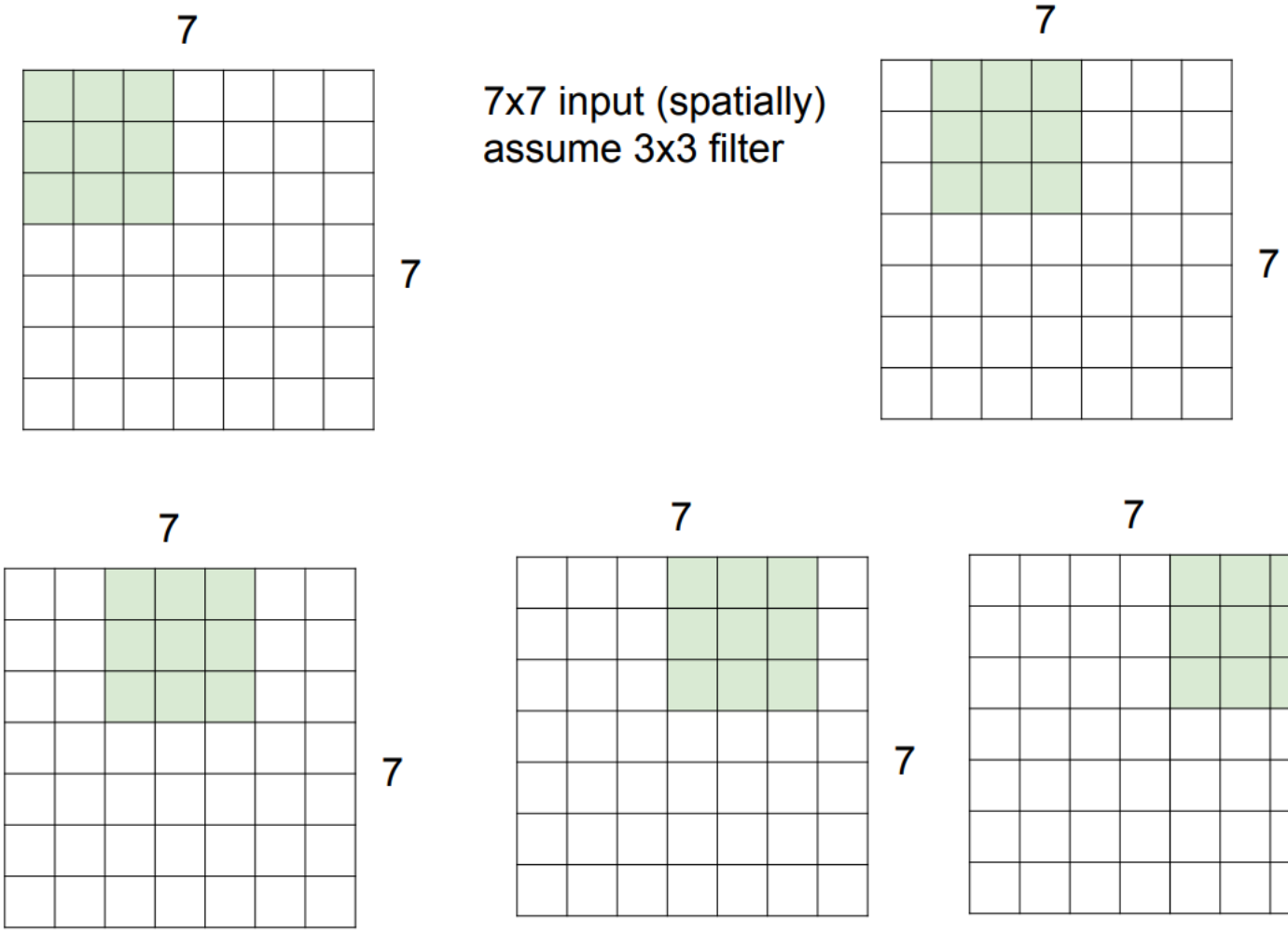
Convolutional Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

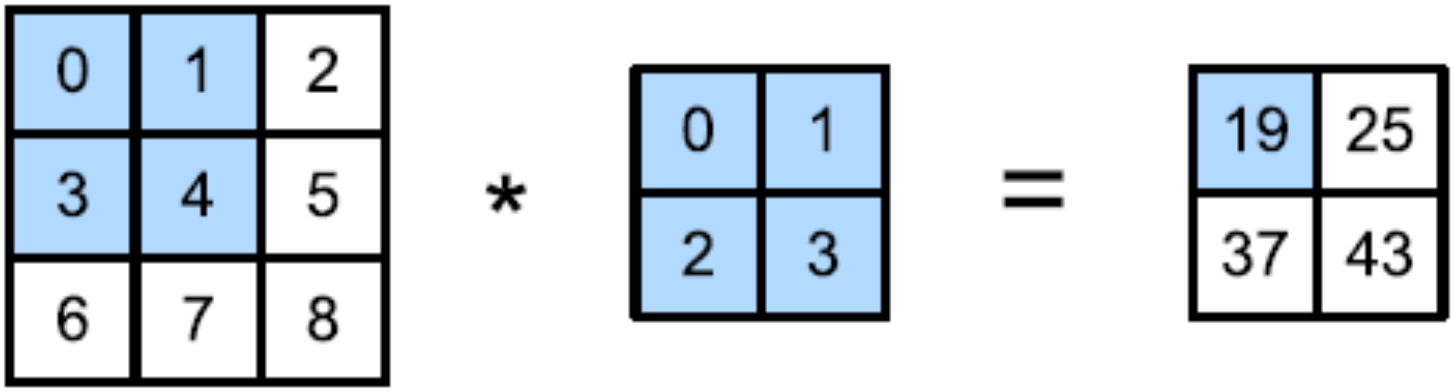


Convolutions

A closer look at spatial dimensions:



Example



Input Filter Output

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
  - Yann Lecun
- Thanks!