CAP5415-Computer Vision
Lecture 9-Neural Nets for Computer Vision

Dr. Ulas Bagci
bagci@ucf.edu
Typical goal of machine learning

**input**
- images/video
- audio
- text

**output**
- Label: “Motorcycle”
- Suggest tags
- Image search
- Speech recognition
- Music classification
- Speaker identification
- Web search
- Anti-spam
- Machine translation
Typical goal of machine learning

**input**
- images/video
- audio
- text

**output**
- Label: “Motorcycle”
- Suggest tags
- Image search
- ... 
- Speech recognition
- Music classification
- Speaker identification
- ... 
- Web search
- Anti-spam
- Machine translation
- ... 

*Feature engineering: most time consuming!*
Our goal in object classification

ML -> “motorcycle”
Why is this hard?

You see this:

But the camera sees this:
Pixel-based representation

Input

Raw image

Motorbikes

“Non”-Motorbikes

Learning algorithm
Pixel-based representation

Input

Raw image

Learning algorithm

Motorbikes
“Non”-Motorbikes
Pixel-based representation

Input

Raw image

Motorbikes
"Non"-Motorbikes

Learning algorithm
What we want

Input

Feature representation

E.g., Does it have Handlebars? Wheels?

Motorbikes
“Non”-Motorbikes

Raw image

Features

Wheels

Handlebars

pixel 1

pixel 2
Some feature representations

SIFT

Spin image

HoG

RIFT

Textons

GLOH
Some feature representations

Coming up with features is often difficult, time-consuming, and requires expert knowledge.
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
  – Model our systems after the brain!
Representation Learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Biologically inspired systems
  – to make the computer more robust, intelligent, and learn, ...
  – Model our systems after the brain!
    • Brain interprets imprecise information from the senses at an incredibly rapid rate!
    • It discerns a whisper in a noisy room, a face in a dimly lit alley, and hidden agenda in a political statement.
A common logic to seeing cats and cosmos

There may be a universal logic to how physicists, computers and brains tease out important features from among other irrelevant bits of data.
Types of Learning

• Supervised (inductive) learning
  – Training data includes desired outputs

• Unsupervised learning
  – Training data does not include desired outputs

• Semi-supervised learning
  – Training data includes a few desired outputs

• Reinforcement learning
  – Rewards from sequence of actions
AI, ML, and DL

**Figure 1** — Artificial intelligence (AI), machine learning and deep learning

**Artificial Intelligence** is the study of devices that perceive their environment and define a course of action that will maximize its chance of achieving a given goal.

**Machine Learning** is a subset of artificial intelligence, in which machines learn how to complete a certain task without being explicitly programmed to do so.

**Deep Learning** is a subset of machine learning in which the tasks are broken down and distributed onto machine learning algorithms that are organised in consecutive layers. Each layer builds up on the output from the previous layer. Together the layers constitute an artificial neural network that mimics the distributed approach to problem-solving carried out by neurons in a human brain.

*source www.webfoundation.org via mikequindazzi*
Where does AI go?

Three Stages of AI

Stage 1: Machine Learning
User driven big data models for machine learning.

Stage 2: Machine Intelligence
Advanced network trained to build ad-hoc models to learn from custom data.

Stage 3: Machine Consciousness
Cognitive, self-learning.
Use of Neural Networks (...& deep NN)
Image Classification

ImageNet Classification

- 1000 categories and 1.2 million training images

Semantic Segmentation
Object Detection/Tracking
Object Detection
Neural Network trained to drive a car!

Weights to output units from the hidden unit

Weights of each pixel for one hidden unit
Neurons in the Brain

- Brain is composed of **neurons**
- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately **summed**
- When the input exceeds a threshold the neuron sends an electrical spike that travels down the axon, to the next neuron(s)
Background in Neural Nets (NN)

• Neural Nets can be:
  – Biological Models
  – Artificial Models

• Desire to produce artificial systems capable of sophisticated computations similar to human brain!
Brain Analogy of NN: Brain is a remarkable Computer

- The brain is composed of a **mass of interconnected neurons**
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is transmitted is an **all-or-nothing** event (the electrical potential in the cell body of the neuron is **thresholded**)
- Whether a signal is sent, depends on the **strength of the bond** (synapse) between two neurons
Brain Analogy of NN: Brain is a remarkable Computer

- The brain is composed of a mass of interconnected neurons
  - each neuron is connected to many other neurons
- Neurons transmit signals to each other
- Whether a signal is transmitted is an all-or-nothing event (the electrical potential in the cell body of the neuron is thresholded)
- Whether a signal is sent, depends on the strength of the bond (synapse) between two neurons
Computational Implementation of the Neural Activation Function
Computational Implementation of the Neural Activation Function
Computational Implementation of the Neural Activation Function

- **Dendrites**
- **Soma (cell body)**
- **Axon**
- **Input**
- **Output**
Computational Implementation of the Neural Activation Function

Simple activation function (sum of individual inputs):

\[ n_j = \sum_i x_i w_{ij} \]

Activation value (sigmoid / logistic):

\[ y_j = \frac{1}{1 + e^{-n_j}} \]
Why Sigmoid?

- It is a standard function in NN.
- It transforms the net input value into an activation value $y$, which is then sent on to other units.

**Steps**
- Sigmoid function
- Sign function
- Step function
Background on Logit

Assumes the following functional form for $P(Y|X)$:

$$P(Y = 1|X) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

Logistic function applied to a linear function of the data

Logistic function (or Sigmoid):

$$\frac{1}{1 + \exp(-z)}$$
Background on Logit

Logistic Regression is a Linear Classifier!

Assumes the following functional form for $P(Y|X)$:

$$ P(Y = 1|X) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))} $$

Decision boundary:

$$ P(Y = 0|X) \geq \frac{0}{1} P(Y = 1|X) $$

$$ 0 \geq w_0 + \sum_i w_i X_i $$

(Linear Decision Boundary)
Perceptron

• is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.
Perceptron

• is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons.
Perceptron

\[ y = \sum_{j=1}^{d} w_j x_j + w_0 \]

- **Connection weights**
- **Bias unit**
- **Input units**
- **Output unit**

is the intercept value to make the model more general
Perceptron

$y = \sum_{j=1}^{d} w_j x_j + w_0$

is the intercept value to make the model more general

Connection weights

Bias unit

Output unit

Input units

$y = w^T x$

$w = [w_0, w_1, \ldots, w_d]^T$

$x = [x_0, x_1, \ldots, x_d]^T$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage regression or classification model
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Each target Class is coded as 0 -1

Hidden layers (derived features)

Typically $K=1$ for regression
And there is only one output Unit at the top

For classification, $K>1$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Each target Class is coded as 0 -1

Hidden layers (derived features)

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \; m = 1, \ldots, M$$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage **regression** or **classification** model

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \ m = 1, \ldots, M$$

$$T_k = \beta_{0k} + \beta_k^T Z, \ k = 1, \ldots, K$$
Perceptron-Nonlinear Statistical Models

- A NN is a two-stage \textit{regression} or \textit{classification} model

Each target Class is coded as 0 - 1

Hidden layers (derived features)

Typically $K=1$ for regression
And there is only one output
Unit at the top

For classification, $K>1$

\[
Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \ m = 1, \ldots, M
\]

\[
T_k = \beta_{0k} + \beta_k^T Z, \ k = 1, \ldots, K
\]

\[
f_k(X) = g_k(T), \ k = 1, \ldots, K
\]
Perceptron-Nonlinear Statistical Models

• A NN is a two-stage **regression** or **classification** model

For regression, output function is

$$g_k(T) = T_k$$

For classification, it is softmax func:

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

$$Z_m = \sigma(\alpha_{0_m} + \alpha_m^T X), \ m = 1, \ldots, M$$

$$T_k = \beta_{0_k} + \beta_k^T Z, \ k = 1, \ldots, K$$

$$f_k(X) = g_k(T), \ k = 1, \ldots, K$$

$$\sigma(v) = 1/(1 + e^{-v})$$
Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\begin{align*}
\{\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M\} & \quad M(p + 1) \text{ weights,} \\
\{\beta_{0k}, \beta_k; k = 1, 2, \ldots, K\} & \quad K(M + 1) \text{ weights.}
\end{align*}
\]
Fitting NNs

• The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\begin{align*}
\{\alpha_0, \alpha_m; m = 1, 2, \ldots, M\} & \text{ } M(p + 1) \text{ weights,} \\
\{\beta_0, \beta_k; k = 1, 2, \ldots, K\} & \text{ } K(M + 1) \text{ weights.}
\end{align*}
\]

• For regression, we use sum-of-squared errors as our measure of fit:

\[
R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.
\]
Fitting NNs

- The NN model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

\[
\{\alpha_{0m}, \alpha_m; \ m = 1, 2, \ldots, M\}\ \ M(p + 1) \ \text{weights}, \\
\{\beta_{0k}, \beta_k; \ k = 1, 2, \ldots, K\}\ \ K(M + 1) \ \text{weights}.
\]

- For regression, we use sum-of-squared errors as our measure of fit:

\[
R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.
\]

- For classification, we use either squared error or cross-entropy:

\[
R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i),
\]

and the corresponding classifier is

\[
G(x) = \arg\max_k f_k(x).
\]
Fitting NN: Back-propagation

- Typically we don’t want the global minimizer of $R(.)$, as this likely to be an overfit solution.
- Instead, some regularization is needed:
  - This is achieved directly through a penalty term, or indirectly by early stopping.
Fitting NN: Back-propagation

• Typically we don’t want the global minimizer of $R(.)$, as this likely to be an overfit solution.

• Instead, some regularization is needed:
  – This is achieved directly through a penalty term, or indirectly by early stopping.

• The generic approach for minimizing $R(.)$ is by gradient-descent, called back-propagation.
Fitting NN: Back-propagation

• Typically we don’t want the global minimizer of $R(.)$, as this likely to be an overfit solution.

• Instead, some regularization is needed:
  – This is achieved directly through a penalty term, or indirectly by early stopping.

• The generic approach for minimizing $R(.)$ is by gradient-descent, called back-propagation.

• Due to compositional form of the model, the gradient can be easily derived using the chain rule for differentiation. This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.
Back-propagation

\[ z_{m_i} = \sigma(\alpha_{0m} + \alpha_T^m x_i), \quad z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi}) \]

\[
R(\theta) \equiv \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,
\]
Back-propagation

\[ z_{m_i} = \sigma(\alpha_{0m} + \alpha_{m}^T x_i), \quad z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi}) \]

\[
R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,
\]

Taking the derivative of \( R \) w.r.t. weight parameters

\[
\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},
\]

\[
\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_{m}^T x_i)x_{i\ell}.
\]
Back-propagation

- Given these derivatives, a gradient descent update at the \((r+1)\)st iteration has the form

\[
\beta^{(r+1)}_{km} = \beta^{(r)}_{km} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta^{(r)}_{km}},
\]

\[
\alpha^{(r+1)}_{m\ell} = \alpha^{(r)}_{m\ell} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha^{(r)}_{m\ell}},
\]
Back-propagation

- Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

\[
\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},
\]

\[
\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},
\]

Learning rate
Back-propagation

• Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

\[
\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\
\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},
\]

Errors from the current model at the output units

\[
\frac{\partial R_i}{\partial \beta_{km}} = \delta_{kizmi}, \\
\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi}x_{il}.
\]

Errors from the current model at the hidden units

Learning rate
Back-propagation

- Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

\[
\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},
\]

\[
\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},
\]

Backpropagation equations
**Back-propagation-Delta Rule**

1. **Forward pass:** calculate $f(.)$ and obtain error rates,
2. **Backward pass:** calculate $s$ from error rates, and update gradient equations.

\[ f(x, \theta) \]
Back-propagation-Delta Rule

1. **Forward pass**: calculate $f(.)$ and obtain error rates,

2. **Backward pass**: calculate $s$ from error rates, and update gradient equations.

**If the weights are near zero**: operative part of sigmoid is roughly linear, hence NN collapses into an approximately linear model.

Often, starting values are chosen to be random values near zero; hence the model starts nearly linear and becomes nonlinear as the weights increase.
Challenges in back-propagation?

• It requires labeled training data.
• The learning time does not scale well
  – It is slow in networks with multiple hidden layers.
• It can get stuck in poor local optima.
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of R.
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of R.

• Weight decay: is analogous to ridge regression used for linear models for avoiding overfit.
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of $R$.

• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

• A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$
Avoid Overfitting

• NNs usually have too many weights, and will overfit the data at the global minimum of $R$.
• **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.
• A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$

where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$
Avoid Overfitting

- NNs usually have too many weights, and will overfit the data at the global minimum of $R$.

- **Weight decay:** is analogous to ridge regression used for linear models for avoiding overfit.

- A new functional is to be minimized:

$$R(\theta) + \lambda J(\theta)$$

where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

where $\lambda$ is tuning parameter >0, estimated by cross validation.
Neural Network - 10 Units, No Weight Decay

Training Error: 0.100
Test Error: 0.259
Bayes Error: 0.210

Neural Network - 10 Units, Weight Decay=0.02

Training Error: 0.160
Test Error: 0.223
Bayes Error: 0.210
Ex: Learning Boolean Functions

\[ y = s(x_1 + x_2 - 1.5) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What is the Boolean function that the following perceptron calculates?

\[ y = s(-x + 0.5) \]
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(-x + 0.5) \]

NOT of \( x \)
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(x_1 + x_2 - 0.5) \]
Question?

• What is the Boolean function that the following perceptron calculates?

\[ y = s(x_1 + x_2 - 0.5) \]

\( x_1 \text{ OR } x_2 \)
Key Ideas in Neural Nets

1. Learn features from data
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier
Key Ideas in Neural Nets

1. Learn features from data
2. Use differentiable functions that produce features efficiently
3. End-to-End learning: no distinction between feature extractor and classifier
Learning From Experience

Deep neural networks learn by adjusting the strengths of their connections to better convey input signals through multiple layers to neurons associated with the right general concepts.

When data is fed into a network, each artificial neuron that fires (labeled “1”) transmits signals to certain neurons in the next layer, which are likely to fire if multiple signals are received. The process filters out noise and retains only the most relevant features.
ZIP-CODE example

- Character recognition task: classification of handwritten numerals.
- It has remained a benchmark problem in the field for many years.

- Each image is 16x16 8-bit grayscale representation of handwritten digit.
5 networks used in the ZIP-CODE example

Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.
5 networks used in the ZIP-CODE example
References and Slice Credits

- Rob Fergus, Deep Learning Tutorial CVPR 2012
- A. Torralba, MIT
- Aarti Singh & Barnabas Poczos (CMU)
- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning.