Lecture: Deep Learning

Juan Carlos Niebles and Ranjay Krishna
Stanford Vision and Learning Lab
Announcements

• Exam next week
  – Hewlett Teaching Center, Room 200
  – December 10, 3:30 to 6:30pm
  – Practice exam + solutions released on piazza

• HW8
  – due tomorrow

• Extra credit
  – Writing lecture notes (especially for this week's lectures)
  – Explanations on how to write and submit notes is available on the webpage
CS 131 Roadmap

Pixels
- Convolutions
- Edges
- Descriptors

Segments
- Resizing
- Segmentation
- Clustering

Images
- Recognition
- Detection
- Ontology

Videos
- Motion
- Tracking

Web
- Neural networks
  Convolutional neural networks
Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks
Today's agenda

• Perceptron
  • Linear classifier
  • Loss function
  • Gradient descent and backpropagation
  • Neural networks
1950s Age of the Perceptron
1957 The Perceptron (Rosenblatt)
1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network
1986 Back propagation (Hinton)
1990s Age of the Graphical Model
2000s Age of the Support Vector Machine

2010s Age of the Deep Network
deep learning = known algorithms + computing power + big data
Perceptron

\[ x_1, x_2, x_3, \ldots, x_N \rightarrow \text{weights} \rightarrow \Sigma \rightarrow \text{sum} \rightarrow f \rightarrow y \rightarrow \text{output} \]

\[ w_1, w_2, w_3, \ldots, w_N \]

\[ \text{sign function} \]
Aside: Inspiration from Biology

Neural nets/perceptrons are loosely inspired by biology.

But they are NOT how the brain works, or even how neurons work.
Perceptron: for image classification

\[ x_1, x_2, x_3, \ldots, x_N \] 

weights: \( w_1, w_2, w_3, \ldots, w_N \)

sum

sign function

\[ f \]

\[ y \] dog
Recall: image classification pipeline

Training Images

Training Labels

Image Features

Training

Learned Classifier

Image Features

Learned Classifier

Prediction

Test Image

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
Recall: we can featurize images into a vector

- Raw pixels
- Raw pixels + (x,y)
- PCA
- LDA
- BoW
- BoW + spatial pyramids
Recall: we can featurize images into a vector

- Raw pixels
- Raw pixels + (x,y)
- PCA
- LDA
- BoW
- BoW + spatial pyramids

Image Vector

\[ x_1, x_2, x_3, \ldots, x_{n} \]
Perceptron: for image classification
Perceptron: simplified view with one perceptron

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ \vdots \]
\[ x_N \]

\[ \text{dog} \]
Perceptron: simplified view with two perceptrons
Linear classifier as a set of perceptrons
Today's agenda

• Perceptron
• Linear classifier
• Loss function
• Gradient descent and backpropagation
• Neural networks
Linear classifier: input layer

\[ x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_N \]

- `cat`
- `dog`
- `bird`
Linear classifier: output layer

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ \ldots \]
\[ x_N \]

‘input’ layer

\[ \text{cat} \]
\[ \text{dog} \]
\[ \text{bird} \]

‘output’ layer
Linear classifier: mathematical formulation

\[ f(x, W) \]

10 numbers giving class scores

\[ W \]

Weights or Parameters
Linear classifier: mathematical formulation

\[ f(x, W) = Wx \]

\[ x = 3072 \times 1 \]

\[ W = ? \]

10 numbers giving class scores

(32x32x3)
3072 dimensional vector

Weights or Parameters
Linear classifier: mathematical formulation

\[ f(x, W) = Wx \]
\[ x = 3072 \times 1 \]
\[ W = 10 \times 3072 \]

Weights or Parameters

10 numbers giving class scores

(32x32x3) 3072 dimensional vector
Linear classifier: function visualized

\[
\text{dog Weight Vector} \times \text{Image Vector} = \text{“score” of the image being an dog}
\]
Linear classifier: function visualized

\[
\text{Image Vector} \times \text{cat Weight Vector} = \text{“score” of the image being an cat}
\]
Linear classifier: function visualized

Image Vector \times \text{bird Weight Vector} = \text{"probability" of the image being a bird}
Linear classifier: function visualized

Truck Weight Vector \times \text{Image Vector} = \text{“score” of the image being a truck}
Linear classifier: function visualized

\[ X \times \text{Weight Matrix} = \text{Image Vector} \]
Linear classifier: bias vector
Linear classifier: size

\[
\begin{align*}
10 \times 3072 & \quad \text{Weight Matrix} \\
3072 \times 1 & \quad \text{Image Vector} \\
10 \times 1 & \quad \text{Bias Vector}
\end{align*}
\]
Linear classifier: Making a classification

\[
\text{argmax}
\]

dog

234
94
-104
22
1
27.3
-13
-123
99
-82
Interpreting the weights

• Assume our weights are trained on the CIFAR 10 dataset with raw pixels:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck
Interpreting the weights as templates

Let us look at each row of the weight matrix
Interpreting the weights as templates

We can reshape the vector back in to the shape of an image
Let's visualize what the templates look like

We can reshape the row back to the shape of an image
Interpreting the weights geometrically

• Assume the image vectors are in 2D space to make it easier to visualize.

Plot created using Wolfram Cloud
Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks
Training linear classifiers

We need to learn how to pick the weights in the first place.

Formally, we need to find $w$ such that

$$\min_w \text{Loss}(y, \hat{y})$$

where $y$ is the true label, $\hat{y}$ is the model's predicted label.

All we have to do is define a loss function!
Given training data:

\[ y = wx \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10.1</td>
</tr>
<tr>
<td>2</td>
<td>1.9</td>
</tr>
<tr>
<td>3.5</td>
<td>3.4</td>
</tr>
<tr>
<td>1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

What do you think is a good approximation weight parameter for this data point?
Given training data:

\[ y = wx \]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10.1</td>
<td>2</td>
<td>20.1</td>
</tr>
<tr>
<td>2</td>
<td>1.9</td>
<td>4</td>
<td>13.1</td>
</tr>
<tr>
<td>3.5</td>
<td>3.4</td>
<td>5</td>
<td>2.4</td>
</tr>
<tr>
<td>1</td>
<td>1.1</td>
<td>0.6</td>
<td>0.1</td>
</tr>
</tbody>
</table>

What about this one?
Properties of a loss function

Given several training examples:
\[ \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \]
and a perceptron:
\[ \hat{y} = wx \]
where \( x_i \) is image and \( y_i \) is (integer) label
(0 for dog, 1 for cat, etc.)

A loss function \( L_i(y_i, \hat{y}_i) \) tells us how good our current classifier
- When the classifier predicts correctly \( (y_i = \hat{y}) \), the loss should be low
- When the classifier makes mistakes \( (y_i \neq \hat{y}) \), the loss should be high
Properties of a loss function

Given several training examples:

\[ \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \]

and a perceptron:

\[ \hat{y} = wx \]

where \( x_i \) is image and \( y_i \) is (integer) label
(0 for dog, 1 for cat, etc.)

Loss over the entire dataset is an average of loss over examples:

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i(y_i, \hat{y}_i) \]
How do we choose $L_i$?

YOU get to chose the loss function!
(some are better than others depending on what you want to do)
Squared Error (L2)
(a popular loss function) ((why?))

Not robust to outliers

\[ L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \]
L1 loss

\[ L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i| \]
L1 Loss
\[ L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i| \]

L2 Loss
\[ L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \]

Zero-One Loss
\[ L_i(y_i, \hat{y}_i) = 1||y_i \neq \hat{y}_i|| \]

Hinge Loss
\[ L_i(y_i, \hat{y}_i) = \max(0, 1 - y_i \hat{y}_i) \]
Softmax Classifier (Multinomial Logistic Regression)

- It allows us to treat the outputs of a model as probabilities for each class.
- Common way of measuring distance between probability distributions is Kullback–Leibler (KL) divergence.
  \[ D_{KL} = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]
- where \( P \) is the ground truth distribution
- and \( Q \) is the model's output score distribution
Softmax Classifier (Multinomial Logistic Regression)

- KL divergence.

\[ D_{KL} = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]

- In our case, \( P \) is only non-zero for the correct class.

- For example, consider the case where we only have 3 classes:
Softmax Classifier (Multinomial Logistic Regression)

• KL divergence.

\[ D_{KL} = \sum_y P(y) \log \frac{P(y)}{Q(y)} \]

\[ = - \log Q(y) \text{ when } y = \text{dog} \]

\[ = - \log \text{Prob}[f(x_i, W) = y_i] \]
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = -\log \text{Prob}[f(x_i, W) = y_i] \]

Remember our linear classifier:

\[ \hat{y} = wx \]

there are no limits on the output space. Meaning that the model can generate outputs > 1 or < 0.
**Softmax Classifier** (Multinomial Logistic Regression)

\[ L_i = - \log Prob[f(x_i, W) == y_i] \]

We need a mechanism to convert or normalize the outputs into probability ranges [0,1].

Solution: **SOFTMAX**: \[ Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}} \]
**Softmax Classifier** (Multinomial Logistic Regression)

\[ L_i = - \log Prob[f(x_i, W) == y_i] \]

We need a mechanism to convert or normalize the outputs into probability ranges \([0,1]\).

Solution: SOFTMAX: \( Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}} \)
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \text{Prob}[f(x_i, W) = y_i] \]

In this case, what is the loss:

\[ L_i = ?? \]
**Softmax Classifier** (Multinomial Logistic Regression)

\[ L_i = - \log \text{Prob}[f(x_i, W) == y_i] \]

In this case, what is the loss:

\[ L_i = - \log(0.13) = 2.04 \]
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \text{Prob}[f(x_i, W) = y_i] \]

What is the minimum and maximum values that the loss can be?
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = -\log \text{Prob}[f(x_i, W) == y_i] \]

At initialization, all the weights will be random. In this case, we can assume that the outputs will have the same probability. In this case, what will the initial loss be?

Model outputs: 
\[
\begin{bmatrix}
3.2 \\ 5.1 \\ -1.7
\end{bmatrix}
\]

Probabilities: 
\[
\begin{bmatrix}
24.5 \\ 164 \\ 0.18
\end{bmatrix}
\]

Correct outputs: 
\[
\begin{bmatrix}
1 \\ 0 \\ 0
\end{bmatrix}
\]
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = -\log P\text{rob}[f(x_i, W) == y_i] \]

At initialization, all the weights will be random. In this case, we can assume that the outputs will have the same probability. In this case, what will the initial loss be?

\[ L_i = -\log \left( \frac{1}{C} \right) = \log(C) = \log(10) = 2.03 \]
Today's agenda

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks
Gradient descent visualized: Minimizing loss
Gradient descent visualized: Minimizing loss

\[ L \] \\
\[ w \]
Gradient descent visualized: Minimizing loss
Gradient descent visualized: Minimizing loss
Gradient descent visualized: Minimizing loss

\[ L \]

\[ w \]
Gradient descent visualized: Minimizing loss
Gradient descent visualized: Minimizing loss

$\begin{align*}
L & \Downarrow \\
w \quad & \quad \\
\Downarrow & \\
\end{align*}$
Gradient descent visualized: Minimizing loss
Gradient Descent Pseudocode

for _ in {0,...,num_epochs}:
    \[ L = 0 \]
    for \( x_i, y_i \) in data:
        \[ \hat{y}_i = f(x_i, W) \]
        \[ L += L_i(y_i, \hat{y}_i) \]
        \[ \frac{dL}{dW} = ??? \]
    \[ W := W - \alpha \frac{dL}{dW} \]
Partial derivative of loss to update weights

Given training data point \((x, y)\), the linear classifier formula is: \(\hat{y} = Wx\)

Let's assume that the correct label is class \(k\), implying \(y = k\)

\[
\text{Loss} = L(\hat{y}, y) = -\log \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}
\]

\[
= -\hat{y}_k + \log \sum_j e^{\hat{y}_j}
\]

Calculating the loss \(\frac{dL}{dW}\) is hard mathematically. But we can use the chain rule to make it simpler:

\[
\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}
\]
Partial derivative of loss to update weights

Given training data point \((x, y)\), the linear classifier formula is: \(\hat{y} = Wx\)

Let's assume that the correct label is class \(k\), implying \(y = k\)

\[
\text{Loss} = -\hat{y}_k + \log \sum_j e^{\hat{y}_j}
\]

Now, we want to update the weights \(W\) by calculating the direction in which to change the weights to reduce the loss:

\[
\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}
\]

We know that: \(\frac{d\hat{y}}{dW} = x\) but what about \(\frac{dL}{d\hat{y}}\)?
Partial derivative of loss to update weights

\[ L = -\hat{y}_k + \log \sum_j e^{\hat{y}_j} \]

To calculate \( \frac{dL}{d\hat{y}} \), we need to consider two cases:

Case 1:
\[ \frac{dL}{d\hat{y}_k} = -1 + \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}} \]

Case 2:
\[ \frac{dL}{d\hat{y}_{1\neq k}} = \frac{e^{\hat{y}_1}}{\sum_j e^{\hat{y}_j}} \]
Partial derivative of loss to update weights

Putting it all together:

\[
\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}
\]

\[
\frac{dL}{dW} = \begin{bmatrix}
 e^\hat{y}_0 \\
\sum_j e^\hat{y}_j \\
\vdots \\
\vdots \\
\vdots \\
 e^\hat{y}_k \\
\sum_j e^\hat{y}_j
\end{bmatrix} x
\]

\[
\frac{dL}{dW} = -1 + \frac{e^\hat{y}_k}{\sum_j e^\hat{y}_j} \\
\frac{e^\hat{y}_3071}{\sum_j e^\hat{y}_j}
\]
Gradient Descent Pseudocode

for _ in \{0, ..., num\_epochs\}:
    \( L = 0 \)
    for \( x_i, y_i \) in data:
        \( \hat{y}_i = f(x_i, W) \)
        \( L += L_i(y_i, \hat{y}_i) \)
        \( \frac{dL}{dw} = We \ know \ how \ to \ calculate \ this \ now! \)

\( W := W - \alpha \frac{dL}{dw} \)
Backprop – another way of computing gradients

\[ \hat{y} = Wx \]

\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
Backprop - another way of computing gradients

Key Insight:
- visualize the computation as a graph
- Compute the forward pass to calculate the loss.
- Compute all gradients for each computation backwards

\[
\hat{y} = Wx \\
L = \text{Loss}(\hat{y}, y) \\
\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}
\]
Backprop example in 1D:

\[ W = 1 \]

\[ x = 2 \]

\[ \frac{dL}{dW} = ? \]

\[ \frac{dL}{d\hat{y}} = 1.2 \]

\( \hat{y} = 2 \)

\[ y = 0 \]

We know that:

\[
\frac{dL}{dW} = \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dW}
\]

\[
\frac{dL}{d\hat{y}} = \frac{dL}{d\hat{y}} 
\]

\[
= \frac{dL}{d\hat{y}} x
\]

\[
= 1.2 x
\]

\[
= 1.2 \times 2
\]

\[
= 2.4
\]
Interpreting the weights **geometrically**

- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: **dog**.
- Initialize the weights randomly

- What is the bias vector here?
Interpreting the weights geometrically

- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two data points
Interpreting the weights geometrically

• Assume the image vectors are in 2D space to make it easier to visualize.
• Let's start with one class: dog.
• Initialize the weights randomly
• Now let's add two data points
• Update the weights
Interpreting the weights geometrically

- Assume the image vectors are in 2D space to make it easier to visualize.
- Let's start with one class: dog.
- Initialize the weights randomly
- Now let's add two more data points
- Update the weights
Interpreting the weights geometrically

• Assume the image vectors are in 2D space to make it easier to visualize.
• Let's start with one class: dog.
• Initialize the weights randomly
• Now let's add two more data points
• Update the weights
Interpreting the weights geometrically

Plot created using Wolfram Cloud
Recall: image classification pipeline

- **Training Images**
  - airplane
  - automobile
  - bird
  - cat
  - deer
  - dog
  - frog
  - horse
  - ship
  - truck

- **Test Image**
  - Puppy

- **Learning**
  - Image Features
  - Training
  - Learned Classifier
  - Prediction

- **Labels**
Recall: we can **featurize** images into a vector

- Raw pixels
- Raw pixels + (x,y)
- PCA
- LDA
- BoW
- BoW + spatial pyramids
Features sometimes might not be linearly separable

\[ f(x, y) = (r(x, y), \theta(x, y)) \]
Remember our linear classifier.
Let's change the features by adding another layer

\[ x_1 \quad W_1 \quad x_2 \quad W_2 \quad x_3 \quad \vdots \quad x_N \]

\[ W \text{ for weights} \]

\[ \text{cat} \quad \text{dog} \quad \text{bird} \]
2-layer network: mathematical formula

• linear classifier:
  \[ y = W x \]

• 2-layer network:
  \[ y = W_2 \max(0, W_1 x) \]

• 3-layer network:
  \[ y = W_3 \max(0, W_2 \max(0, W_1 x)) \]

Choosing the number of layers is a new hyperparameter!
Today's agenda

• Perceptron
• Linear classifier
• Loss function
• Gradient descent and backpropagation
• Neural networks
2-layer network: mathematical formula

- linear classifier:
  \[ y = Wx \]

- 2-layer network:
  \[ y = W_2 \max(0, W_1 x) \]

We know the size of \( x = 1 \times 3072 \) and \( y = 10 \times 1 \)
So what are \( W_1 \) and \( W_2 \)

We know that they must be: \( W_1 = h \times 3072 \) and \( W_2 = 10 \times h \)
\( h \) is a new hyperparameter!
2-layer network: mathematical formula

• linear classifier:
  \[ y = Wx \]

• 2-layer network:
  \[ y = W_2 \max(0, W_1 x) \]

Why is the \( \max(0, \_\) necessary? Let's see what happens when we remove it:

\[ y = W_2 W_1 x = Wx \]

where

\[ W = W_2 W_1 \]
Activation function

The non-linear max function allows models to learn more complex transformations for features.

Choosing the right activation function is another new hyperparameter!

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \]
Recall: image classification pipeline

Training Images

<table>
<thead>
<tr>
<th>airplane</th>
<th>automobile</th>
<th>bird</th>
<th>cat</th>
<th>deer</th>
<th>dog</th>
<th>frog</th>
<th>horse</th>
<th>ship</th>
<th>truck</th>
</tr>
</thead>
</table>

Test Image

Image Features -> Training -> Learned Classifier

Prediction
2-layer neural network performance

• ~40% accuracy on CIFAR-10 test
  – Best class: Truck (~60%)
  – Worst class: Horse (~16%)

• Check out the model at: https://tinyurl.com/cifar10
Next Time…

Backpropagation with deep neural networks

Convolutional neural networks

Designing architectures