Image classification
Image classification

• Given an image, produce a label
• Label can be:
  • 0/1 or yes/no: *Binary classification*
  • one-of-k: *Multiclass classification*
  • 0/1 for each of k concepts: *Multilabel classification*
MNIST

- 2D
- 10 classes
- 6000 examples per class
Caltech 101

- 101 classes
- 10 classes
- 30 examples per class
- Strong category-specific biases
- Clean images

MNIST

1990’s

2004
PASCAL VOC

- 20 classes
- ~500 examples per class
- Clutter, occlusion, natural scenes

MNIST 1990’s
Caltech 101 2004
2007-2012
ImageNet

- 1000 classes
- ~1000 examples per class
- Mix of cluttered and clean images
Why is recognition hard?

- Pose/articulation
- Scale
- Clutter/occlusion
- Lighting
Learning

• Key idea: teach computer visual concepts by providing examples

\[ S = \{(x_i, y_i) \sim \mathcal{D}, i = 1, \ldots, n\} \]
Example

• Binary classifier “Dog” or “not Dog”
• Labels: \{0, 1\}
• Training set

\{(\text{Dog image}), 1), (\text{Dog image}), 1), (\text{Bird image}), 0) , ... \}
Learning

- Key idea: teach computer visual concepts by *providing examples*

\[ S = \{(x_i, y_i) \sim D, i = 1, \ldots, n\} \]

- Want to be able to estimate label \( y \) for *new images* \( x \)
  - Want to give score \( s(y, x) \) for each possible label \( y \), then pick highest scoring
  - Want to estimate \( y(x) \)
  - Want to estimate \( P(y|x) \), then pick most likely
Choosing a model class

• Will estimate a probability $P(y \mid x)$

• Any function that takes $x$ as input and outputs probability distribution
  • $h : \mathcal{X} \rightarrow \mathcal{C}^{\mid \mathcal{Y} \mid}$ where $\mathcal{C}^d$ is a probability distribution over $d$ classes
  • Very large set of possibilities for $h$

• Constrain choice: Choose a family of possible functions $H$
  • Hypothesis class
Hypothesis class I: Classical models

• Choose h to be a linear classifier over some feature space

• First extract features: $z = \phi(x)$
  • $\phi$ is a fixed, hand-crafted function that converts images into features useful for recognition: $\phi: \mathcal{X} \to \mathbb{R}^d$

• Next multiply by a weight matrix to produce class scores: $s = Wz$
  • $W$ is unknown a priori

• Next normalize scores to a probability
  • $P(y = k | x) \propto e^{s_k}$
  • “Softmax”
Hypothesis class I: Classical models

- \( h(x; W) = \text{softmax}(W \phi(x)) \)
- For different settings of \( W \), get different hypotheses
- Hypothesis class \( H = \{ h(\cdot; W); W \in \mathbb{R}^{|Y| \times d} \} \)
- \( W \) are parameters: index hypotheses in hypothesis class
Choice of feature extractor?

- SIFT, HOG, GIST, BOW....
- The rest of the pipeline is very simple: linear function + softmax
- So heavy lifting must be done by feature extractor
- But how do we design feature extractor?
SIFT

- SIFT itself a series of simple, fixed steps
- Make some of them parametric?
Hypothesis class 2: Multilayer perceptrons

• Key idea: build complex functions by composing *many* simple functions
General recipe

• Fix hypothesis class
  • \( h_w(x) = \text{softmax} \left( f_3 \left( f_2 \left( g \left( f_1(x, w_1) \right), w_2 \right), w_3 \right) \right) \)
  • \( h_w(x) = \text{softmax} \left( W \phi(x) \right) \)

• Define loss function
  • \( L(h_w(x_i), y_i) = -\log p_{y_i}(x_i) \)

• Minimize average (or total) loss on the training set
  \[
  \min_w \frac{1}{n} \sum_{i=1}^{n} L(h_w(x_i), y_i)
  \]

• How do we minimize?
• Why should this work?
Training: Choosing the best hypothesis

• Need to minimize an objective function.
• In general, optimization problem.
• If $L$ is differentiable and $h$ is differentiable: can do gradient descent

$$\min_w \frac{1}{n} \sum_{i=1}^{n} L(h_w(x_i), y_i)$$
Training = Optimization

• Simple solution: \textit{gradient descent}

\[
\min_w f(w)
\]

\[
w^{(t+1)} = w^{(t)} - \alpha \nabla_w f(w^{(t)})
\]
Stochastic gradient descent

\[ f(w) = \frac{1}{n} \sum_i L(h_w(x_i), y_i) \]

\[ \nabla_w f(w) = \frac{1}{n} \sum_i \nabla_w L(h_w(x_i), y_i) \]

Objective function

Gradient

Gradient = average of per example gradients

\[ \nabla_w f(w) \approx \nabla_w L(h_w(x_i), y_i) \]

Stochastic gradient descent using single examples

\[ \nabla_w f(w) \approx \frac{1}{|B|} \sum_{k=1}^{\left|B\right|} \nabla_w L(h_w(x_{i_k}), y_{i_k}) \]

Stochastic gradient descent using minibatch
Stochastic gradient descent

- Randomly sample small subset of examples
- Compute gradient on small subset
  - *Unbiased estimate of true gradient*
- Take step along estimated gradient
Computing derivatives

\[ \nabla_w f(w) \approx \nabla_w L(h_w(x_i), y_i) \]

- How do we compute gradient?
- Composition of functions: use chain rule

\[
\begin{align*}
    z_1 &= f_1(x, w_1) & g_1 &= \frac{\partial l}{\partial z_1} = g_2 \frac{\partial z_2}{\partial z_1} \\
    z_2 &= f_2(z_1, w_2) & g_2 &= \frac{\partial l}{\partial z_2} = g_3 \frac{\partial z_3}{\partial z_2} \\
    z_3 &= f_3(z_2, w_3) & g_3 &= \frac{\partial l}{\partial z_3} \\
    l &= L(z_3, y) & \frac{\partial l}{\partial w_1} &= g_1 \frac{\partial z_1}{\partial w_1} \\
    & & \frac{\partial l}{\partial w_2} &= g_2 \frac{\partial z_2}{\partial w_2} \\
    & & \frac{\partial l}{\partial w_3} &= g_3 \frac{\partial z_3}{\partial w_3}
\end{align*}
\]
The gradient of convnets

Backpropagation
Risk

• Given:
  • Distribution $\mathcal{D}$
  • A hypothesis $h \in H$
  • Loss function $L$

• We are interested in Expected Risk:

$$R(h) = \mathbb{E}_{(x,y) \sim \mathcal{D}} L(h(x), y)$$

• Given training set $S$, and a particular hypothesis $h$, Empirical Risk:

$$\hat{R}(S, h) = \frac{1}{|S|} \sum_{(x,y) \in S} L(h(x), y)$$
Risk

\[ R(h) = \mathbb{E}_{(x,y) \sim \mathcal{D}} L(h(x), y) \quad \hat{R}(S, h) = \frac{1}{|S|} \sum_{(x, y) \in S} L(h(x), y) \]

- By central limit theorem,
  \[ \mathbb{E}_{S \sim \mathcal{D}^n} \hat{R}(S, h) = R(h) \]

- Variance proportional to 1/n

- For randomly chosen \( h \), empirical risk is an unbiased estimator of expected risk
Risk

• Empirical risk unbiased estimate of expected risk
• Want to minimize expected risk
• Idea: Minimize empirical risk instead
• This is the **Empirical Risk Minimization Principle**

\[
R(h) = \mathbb{E}_{(x,y) \sim D} L(h(x), y) \quad \hat{R}(S, h) = \frac{1}{|S|} \sum_{(x,y) \in S} L(h(x), y)
\]

\[
h^* = \arg \min_{h \in H} \hat{R}(S, h)
\]
Generalization

\[ R(h) = \mathbb{E}_{(x,y) \sim D} L(h(x), y) \]

\[ \hat{R}(S, h) = \frac{1}{|S|} \sum_{(x,y) \in S} L(h(x), y) \]

\[ R(h) = \hat{R}(S, h) + (R(h) - \hat{R}(S, h)) \]

Training error

Generalization error
Overfitting

- We are minimizing training error
- Empirical risk of chosen hypothesis *no longer* unbiased estimate:
  - We chose hypothesis based on S
  - Might have chosen h for which S is a special case
- Overfitting:
  - Minimize training error, but generalization error *increases*
Controlling generalization error

• Variance of empirical risk inversely proportional to size of $S$
  • Choose very large $S$!

• Larger the hypothesis class $H$, Higher the chance of hitting bad hypotheses with low training error and high generalization error
  • Choose small $H$!

• For many models, can bound generalization error using some property of parameters
  • Regularize during optimization!
  • Eg. L2 regularization
Controlling generalization error

• How do we know we are overfitting?
  • Use a *held-out* “validation set”
  • To be an unbiased sample, must be completely *unseen*
Putting it all together

• Want model with least expected risk = expected loss
• But expected risk hard to evaluate
• Empirical Risk Minimization: minimize empirical risk in training set
• Might end up picking special case: overfitting
• Avoid overfitting by:
  • Constructing large training sets
  • Reducing size of model class
  • Regularization
Putting it all together

• Collect training set and validation set
• Pick hypothesis class
• Pick loss function
• Minimize empirical risk (+ regularization)
• Measure performance on held-out validation set
• Profit!
Loss functions and hypothesis classes

<table>
<thead>
<tr>
<th>Loss function</th>
<th>Problem</th>
<th>Range of $h$</th>
<th>$\mathcal{Y}$</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log loss</td>
<td>Binary Classification</td>
<td>$\mathbb{R}$</td>
<td>${0, 1}$</td>
<td>$\log(1 + e^{-yh(x)})$</td>
</tr>
<tr>
<td>Negative log likelihood</td>
<td>Multiclass classification</td>
<td>$[0, 1]^k$</td>
<td>${1, \ldots, k}$</td>
<td>$- \log h_y(x)$</td>
</tr>
<tr>
<td>Hinge loss</td>
<td>Binary Classification</td>
<td>$\mathbb{R}$</td>
<td>${0, 1}$</td>
<td>$\max(0, 1 - yh(x))$</td>
</tr>
<tr>
<td>MSE</td>
<td>Regression</td>
<td>$\mathbb{R}$</td>
<td>$\mathbb{R}$</td>
<td>$(y - h(x))^2$</td>
</tr>
</tbody>
</table>
Multilayer perceptrons

• Key idea: build complex functions by composing simple functions

\[ f(x) = Wx \]

\[ g(x) = \text{max}(x,0) \]

\[ f(x) = Wx \]

\[ g(x) = \text{max}(x,0) \]

\[ f(x) = Wx \]
Multilayer perceptrons

• Key idea: build complex functions by composing simple functions
• Caveat: simple functions must include non-linearities
• $W(U(Vx)) = (WUV)x$
Reducing capacity
Reducing capacity
Idea 1: local connectivity

• Inputs and outputs are *feature maps*
• Pixels only related to nearby pixels
Idea 2: Translation invariance

- Pixels only related to nearby pixels
Local connectivity + translation invariance = \textit{convolution}

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>0.1</td>
<td>3.6</td>
</tr>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = convolution

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>0.1</td>
<td>3.6</td>
</tr>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = convolution

<table>
<thead>
<tr>
<th>5.4</th>
<th>0.1</th>
<th>3.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Convolution as a primitive

Convolution
Invariance to distortions
Invariance to distortions

Image gradients → Keypoint descriptor
Invariance to distortions: Pooling
Invariance to distortions: Subsampling
Convolution subsampling convolution
Convolution subsampling convolution

• Convolution in earlier steps detects more local patterns less resilient to distortion
• Convolution in later steps detects more global patterns more resilient to distortion
• Subsampling allows capture of larger, more invariant patterns