Neural networks for vision

Nicolas Le Roux
nicolas.le.roux@gmail.com
Outline

- Linear classifiers
- Combining linear classifiers
- Learning a neural network
- Convolutional neural networks
- The power of sloppiness
I'm here for you, I already know that stuff
It's better to look silly than to stay so
Ask questions if you don't understand!
Find linear function \((hyperplane)\) to separate positive and negative examples

\[
x_i \text{ positive: } x_i \cdot w + b \geq 0
\]
\[
x_i \text{ negative: } x_i \cdot w + b < 0
\]

Which hyperplane is best?
Support vector machines

- Find hyperplane that maximizes the margin between the positive and negative examples

\[ \begin{align*}
    \mathbf{x}_i \text{ positive } (y_i = 1): & \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1 \\
    \mathbf{x}_i \text{ negative } (y_i = -1): & \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \\
\end{align*} \]

For support vectors, \[ \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1 \]

Distance between point and hyperplane:
\[ \frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{|| \mathbf{w} ||} \]

Therefore, the margin is \[ \frac{2}{|| \mathbf{w} ||} \]

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998
Linear classifiers

The perceptron (Rosenblatt'57)

What the perceptron can learn, it will learn using a simple weight update rule.
What does w look like?

- Gabor filters
  - Edge detectors
  - Various angles
  - Various frequencies
Some problems are not linear

- Can we learn them using a combination of linear filters?
- "Features" are more and more complex!
Modeling non-linear functions
A multilayer neural network

- Linear classifier at the end!
- Unrestricted hidden layer
Link between NNs and SVMs

- A neural network is a linear classifier in a new space.
- It is a universal approximator.
- It is an SVM whose kernel can be (sometimes badly) learnt!
- Wait... Learnt?
Learning a neural network

- A neural network is just a function (which one?)
- If it has a gradient, we can do gradient descent.
- Does it?
A sigmoid is a smoothed version of the threshold.
Any function can be learned by a 3-layer network with enough hidden units
**Gradient-based supervised learning**

- Parametric prediction function: $f(x, w) \rightarrow y$

- Learning: Minimize

\[
E = \sum_i L(y_i, f(x_i, w))
\]

- Recognition: $y = f(x, w)$

How can we minimize $E$? ..Gradient descent..
Gradient-based supervised learning II

• Difference between stochastic and batch.
Backpropagation

- The gradient with respect to one layer depends on the gradient with respect to the layer above.
- We can "backpropagate" the gradient to the layers below.
The vertical face-finding part of Rowley, Baluja and Kanade's system

Architecture of the complete system: they use another neural net to estimate orientation of the face, then rectify it. They search over scales to find bigger/smaller faces.

Advantages of MLP

- Can learn anything
- Extremely fast at test time (computing the answer for a new datapoint)
- Complete control over the power of the network (by controlling the hidden layers sizes).
Problems with MLP

- Highly non-convex → many local minima
- Upper layers much harder to train than lower layers
- Can learn anything → needs tons of examples to be good (make some awesome Tennis analogy here).
Take-home messages

- Neural networks can learn anything
- But it is HARD!
- If you wish to use them, be smart (or ask someone who knows)!
- If you have a huge dataset, they CAN be awesome!
Convolutional NNets

- An image was just a huge vector
- Can we make more assumptions?
- Filters are mostly LOCAL!
Basic idea

- Instead of computing features over the entire image, compute it over small patches.
- Repeat for every patch.
- "Pool" features to get local invariance.
[Hubel & Wiesel 1962]:

- **simple cells** detect local features
- **complex cells** “pool” the outputs of simple cells within a retinotopic neighborhood.

---

**An Old Idea for Local Shift Invariance**

- Convolutions
- Retinotopic Feature Maps
  - “Simple cells”
  - “Complex cells”
  - pooling subsampling
**CNN architecture**

**Convolutional net for handwriting recognition** (400,000 synapses)
- **Convolutional layers** (simple cells): all units in a feature plane share the same weights
- **Pooling/subsampling layers** (complex cells): for invariance to small distortions.
- **Supervised gradient-descent learning using back-propagation**
- The entire network is trained end-to-end. All the layers are trained simultaneously.
Face detection - pose estimation
Face detection
Applying a ConvNet on Sliding Windows is Very Cheap!

Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales. Convolutional nets can replicated over large images very cheaply. The network is applied to multiple scales spaced by 1.5.
Building a Detector/Recognizer: Replicated Convolutional Nets

Computational cost for replicated convolutional net:
- 96x96 -> 4.6 million multiply-accumulate operations
- 120x120 -> 8.3 million multiply-accumulate operations
- 240x240 -> 47.5 million multiply-accumulate operations
- 480x480 -> 232 million multiply-accumulate operations

Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:
- 96x96 -> 4.6 million multiply-accumulate operations
- 120x120 -> 42.0 million multiply-accumulate operations
- 240x240 -> 788.0 million multiply-accumulate operations
- 480x480 -> 5,083 million multiply-accumulate operations
Generic Object Detection and Recognition with Invariance to Pose and Illumination

50 toys belonging to 5 categories: animal, human figure, airplane, truck, car
10 instance per category: 5 instances used for training, 5 instances for testing
Raw dataset: 972 stereo pair of each object instance. 48,600 image pairs total.

For each instance:
18 azimuths
0 to 350 degrees every 20 degrees
9 elevations
30 to 70 degrees from horizontal every 5 degrees
6 illuminations
on/off combinations of 4 lights
2 cameras (stereo)
7.5 cm apart
40 cm from the object

Training instances
Test instances
Data Collection, Sample Generation

Image capture setup

Objects are painted green so that:
- all features other than shape are removed
- objects can be segmented, transformed, and composited onto various backgrounds

Original image
Object mask

Shadow factor
Composite image
Textured and Cluttered Datasets
90,857 free parameters, 3,901,162 connections.

The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).

The entire network is trained end-to-end (all the layers are trained simultaneously).

A gradient-based algorithm is used to minimize a supervised loss function.
Local features are extracted everywhere. Averaging/subsampling layer builds robustness to variations in feature locations. Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....
Normalized-Uniform Set: Error Rates

- **Linear Classifier on raw stereo images:** 30.2% error.
- **K-Nearest-Neighbors on raw stereo images:** 18.4% error.
- **K-Nearest-Neighbors on PCA-95:** 16.6% error.
- **Pairwise SVM on 96x96 stereo images:** 11.6% error.
- **Pairwise SVM on 95 Principal Components:** 13.3% error.
- **Convolutional Net on 96x96 stereo images:** 5.8% error.

![Training instances](image1.png)

![Test instances](image2.png)
Jittered-Cluttered Dataset:

- **291,600** stereo pairs for training, **58,320** for testing
- Objects are jittered: position, scale, in-plane rotation, contrast, brightness, backgrounds, distractor objects,...
- Input dimension: 98x98x2 (approx 18,000)
Experiment 2: Jittered-Cluttered Dataset

- 291,600 training samples, 58,320 test samples
- SVM with Gaussian kernel: 43.3% error
- Convolutional Net with binocular input: 7.8% error
- Convolutional Net + SVM on top: 5.9% error
- Convolutional Net with monocular input: 20.8% error
- Smaller mono net (DEMO): 26.0% error

Dataset available from http://www.cs.nyu.edu/~yann
What's wrong with K-NN and SVMs?

K-NN and SVM with Gaussian kernels are based on matching global templates. Both are “shallow” architectures. There is no way to learn invariant recognition tasks with such naive architectures (unless we use an impractically large number of templates).

The number of necessary templates grows exponentially with the number of dimensions of variations. Global templates are in trouble when the variations include: category, instance shape, configuration (for articulated object), position, azimuth, elevation, scale, illumination, texture, albedo, in-plane rotation, background luminance, background texture, background clutter, .....
Examples (Monocular Mode)
Examples (Monocular Mode)
Examples (Monocular Mode)
Examples (Monocular Mode)

Zoom = 1.0, Threshold = -1.0, filter on

animal [0.1]
car [0.3]
Examples (Monocular Mode)
Examples (Monocular Mode)

Zoom= 0.7, Threshold= -1.8, filter on

- plane [1.1]
- plane [-0.8]
- animal [-0.6]
- truck [-0.3]
Convolutional nets can be trained to perform a wide variety of visual tasks. Global supervised gradient descent can produce parsimonious architectures.

**BUT:** they require lots of labeled training samples
- 60,000 samples for handwriting
- 120,000 samples for face detection
- 25,000 to 350,000 for object recognition

Since low-level features tend to be non task specific, we should be able to learn them unsupervised.

Hinton has shown that layer-by-layer unsupervised “pre-training” can be used to initialize “deep” architectures

[Hinton & Shalakhutdinov, Science 2006]

Can we use this idea to reduce the number of necessary labeled examples.
Learning fast

- Common point of neural networks: need many examples
- → we need to be able to use these examples fast
Objectives and Essential Remarks

- Baseline large-scale learning algorithm
  
  Randomly discarding data is the simplest way to handle large datasets.

  - What are the statistical benefits of processing more data?
  - What is the computational cost of processing more data?

- We need a theory that joins Statistics and Computation!
  
  - 1967: Vapnik’s theory does not discuss computation.
  - 1981: Valiant’s learnability excludes exponential time algorithms, but (i) polynomial time can be too slow, (ii) few actual results.
  - We propose a simple analysis of approximate optimization...
Learning Algorithms: Standard Framework

- Assumption: examples are drawn independently from an unknown probability distribution $P(x, y)$ that represents the rules of Nature.

- Expected Risk: $E(f) = \int \ell(f(x), y) dP(x, y)$.

- Empirical Risk: $E_n(f) = \frac{1}{n} \sum \ell(f(x_i), y_i)$.

- We would like $f^*$ that minimizes $E(f)$ among all functions.

- In general $f^* \notin \mathcal{F}$.

- The best we can have is $f^*_\mathcal{F} \in \mathcal{F}$ that minimizes $E(f)$ inside $\mathcal{F}$.

- But $P(x, y)$ is unknown by definition.

- Instead we compute $f_n \in \mathcal{F}$ that minimizes $E_n(f)$. Vapnik-Chervonenkis theory tells us when this can work.
Learning with Approximate Optimization

Computing $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ is often costly.

Since we already make lots of approximations, why should we compute $f_n$ exactly?

Let's assume our optimizer returns $\tilde{f}_n$ such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

For instance, one could stop an iterative optimization algorithm long before its convergence.
Decomposition of the Error (i)

\[ E(\tilde{f}_n) - E(f^*) = E(f^*_\mathcal{F}) - E(f^*) + E(f_n) - E(f^*_\mathcal{F}) + E(\tilde{f}_n) - E(f_n) \]

Approximation error
Estimation error
Optimization error

Problem:
Choose \( \mathcal{F}, n, \) and \( \rho \) to make this as small as possible,

subject to budget constraints \( \left\{ \begin{array}{l} \text{maximal number of examples } n \\ \text{maximal computing time } T \end{array} \right\} \)
Decomposition of the Error (ii)

Approximation error bound:  
– decreases when $\mathcal{F}$ gets larger. \hspace{1cm} (Approximation theory)

Estimation error bound:  
– decreases when $n$ gets larger.  
– increases when $\mathcal{F}$ gets larger. \hspace{1cm} (Vapnik-Chervonenkis theory)

Optimization error bound:  
– increases with $\rho$. \hspace{1cm} (Vapnik-Chervonenkis theory plus tricks)

Computing time $T$:  
– decreases with $\rho$  
– increases with $n$  
– increases with $\mathcal{F}$ \hspace{1cm} (Algorithm dependent)
Small-scale vs. Large-scale Learning

We can give rigorous definitions.

- **Definition 1:**
  We have a small-scale learning problem when the active budget constraint is the number of examples $n$.

- **Definition 2:**
  We have a large-scale learning problem when the active budget constraint is the computing time $T$. 
Small-scale Learning

The active budget constraint is the number of examples.

- To reduce the estimation error, take $n$ as large as the budget allows.
- To reduce the optimization error to zero, take $\rho = 0$.
- We need to adjust the size of $\mathcal{F}$.

See Structural Risk Minimization (Vapnik 74) and later works.
Large-scale Learning

The active budget constraint is the computing time.

- More complicated tradeoffs.
  The computing time depends on the three variables: $F$, $n$, and $\rho$.

- Example.
  If we choose $\rho$ small, we decrease the optimization error. But we must also decrease $F$ and/or $n$ with adverse effects on the estimation and approximation errors.

- The exact tradeoff depends on the optimization algorithm.

- We can compare optimization algorithms rigorously.
Executive Summary

- Good optimization algorithm (superlinear).
  \( \rho \) decreases faster than \( \exp(-T) \)

- Mediocre optimization algorithm (linear).
  \( \rho \) decreases like \( \exp(-T) \)

- Extraordinary poor optimization algorithm
  \( \rho \) decreases like \( 1/T \)
Case Study

Simple parametric setup

- $\mathcal{F}$ is fixed.
- Functions $f_w(x)$ linearly parametrized by $w \in \mathbb{R}^d$.

Comparing four iterative optimization algorithms for $E_n(f)$

1. Gradient descent.
2. Second order gradient descent (Newton).
4. Stochastic second order gradient descent.
Gradient Descent (GD)

Iterate

\[ w_{t+1} \leftarrow w_t - \eta \frac{\partial E_n(f_{w_t})}{\partial w} \]

Gradient J

Best speed achieved with fixed learning rate \( \eta = \frac{1}{\lambda_{\text{max}}} \).
(e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\hat{f}<em>n) - E(f</em>{f_{\alpha}}^*) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD</td>
<td>( \mathcal{O}(nd) )</td>
<td>( \mathcal{O}(\kappa \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}(nd\kappa \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}\left( \frac{d^2 \kappa}{\varepsilon^2 \alpha \log^2 \frac{1}{\varepsilon}} \right) )</td>
</tr>
</tbody>
</table>

– In the last column, \( n \) and \( \rho \) are chosen to reach \( \varepsilon \) as fast as possible.
– Solve for \( \varepsilon \) to find the best error rate achievable in a given time.
– Remark: abuses of the \( \mathcal{O}() \) notation
Second Order Gradient Descent (2GD)

Iterate
\[ w_{t+1} \leftarrow w_t - H^{-1} \frac{\partial E_n(f_{w_t})}{\partial w} \]

We assume \( H^{-1} \) is known in advance.
Superlinear optimization speed (e.g., Dennis & Schnabel, 1983)

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach ( \rho )</th>
<th>Time to reach accuracy ( \rho )</th>
<th>Time to reach ( E(\tilde{f}<em>n) - E(f^*</em>\mathcal{F}) &lt; \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2GD</td>
<td>( \mathcal{O}(d(d + n)) )</td>
<td>( \mathcal{O}(\log \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}(d(d + n) \log \log \frac{1}{\rho}) )</td>
<td>( \mathcal{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon}\right) )</td>
</tr>
</tbody>
</table>

- Optimization speed is much faster.
- Learning speed only saves the condition number \( \kappa \).
**Stochastic Gradient Descent (SGD)**

Iterate
- Draw random example \((x_t, y_t)\).
- \(w_{t+1} \leftarrow w_t - \eta \frac{\partial \ell(f_w(x_t), y_t)}{\partial w}\)

Best decreasing gain schedule with \(\eta = \frac{1}{\lambda_{\text{min}}}\).
(see Murata, 1998; Bottou & LeCun, 2004)

<table>
<thead>
<tr>
<th></th>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\hat{f}_n) - E(f^*_x) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SGD</strong></td>
<td>(O(d))</td>
<td>(\frac{\nu k}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(O\left(\frac{d\nu k}{\rho}\right))</td>
<td>(O\left(\frac{d\nu k}{\varepsilon}\right))</td>
</tr>
</tbody>
</table>

With \(1 \leq k \leq \kappa^2\)

- Optimization speed is *catastrophic*.
- Learning speed does not depend on the statistical estimation rate \(\alpha\).
- Learning speed depends on condition number \(\kappa\) but *scales very well*. 
Second order Stochastic Descent (2SGD)

Iterate
- Draw random example \((x_t, y_t)\).
- \(w_{t+1} \leftarrow w_t - \frac{1}{t} H^{-1} \frac{\partial \ell(f_{w_t}(x_t), y_t)}{\partial w}\)

Replace scalar gain \(\frac{\eta}{t}\) by matrix \(\frac{1}{t} H^{-1}\).

<table>
<thead>
<tr>
<th>Cost per iteration</th>
<th>Iterations to reach (\rho)</th>
<th>Time to reach accuracy (\rho)</th>
<th>Time to reach (E(\tilde{f}<em>n) - E(f</em>*^\ast) &lt; \varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{O}(d^2))</td>
<td>(\frac{\nu}{\rho} + o\left(\frac{1}{\rho}\right))</td>
<td>(\mathcal{O}\left(\frac{d^2 \nu}{\rho}\right))</td>
<td>(\mathcal{O}\left(\frac{d^2 \nu}{\varepsilon}\right))</td>
</tr>
</tbody>
</table>

- Each iteration is \(d\) times more expensive.
- The number of iterations is reduced by \(\kappa^2\) (or less.)
- Second order only changes the constant factors.
Benchmarking SGD in Simple Problems

- The theory suggests that SGD is very competitive.
  - Many people associate SGD with trouble.

- SGD historically associated with back-propagation.
  - Multilayer networks are very hard problems (nonlinear, nonconvex)
  - What is difficult, SGD or MLP?

- Try **PLAIN SGD** on simple learning problems.
  - Support Vector Machines
  - Conditional Random Fields

These simple programs are very short.

See also (Shalev-Schwartz et al., 2007; Vishwanathan et al., 2006)
Text Categorization with SVMs

- **Dataset**
  - Reuters RCV1 document corpus.
  - 781,265 training examples, 23,149 testing examples.
  - 47,152 TF-IDF features.

- **Task**
  - Recognizing documents of category CCAT.
  - Minimize $E_n = \frac{1}{n} \sum_i \left( \frac{\lambda}{2} w^2 + \ell(w x_i + b, y_i) \right)$.
  - Update $w \leftarrow w - \eta_t \nabla (w_t, x_t, y_t) = w - \eta_t \left( \lambda w + \frac{\partial \ell(w x_t + b, y_t)}{\partial w} \right)$

Same setup as (Shalev-Schwartz et al., 2007) but plain SGD.
Text Categorization with SVMs

• Results: Linear SVM
  \[ \ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001 \]

<table>
<thead>
<tr>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMLight</td>
<td>23,642 secs</td>
<td>0.2275</td>
</tr>
<tr>
<td>SVMPerf</td>
<td>66 secs</td>
<td>0.2278</td>
</tr>
<tr>
<td>SGD</td>
<td>1.4 secs</td>
<td>0.2275</td>
</tr>
</tbody>
</table>

• Results: Log-Loss Classifier
  \[ \ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y})) \quad \lambda = 0.00001 \]

<table>
<thead>
<tr>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibLinear (\varepsilon = 0.01)</td>
<td>30 secs</td>
<td>0.18907</td>
</tr>
<tr>
<td>LibLinear (\varepsilon = 0.001)</td>
<td>44 secs</td>
<td>0.18890</td>
</tr>
<tr>
<td>SGD</td>
<td>2.3 secs</td>
<td>0.18893</td>
</tr>
</tbody>
</table>
SGD for Real Life Applications

A Check Reader
Examples are pairs (image, amount).
Problem with strong structure:
  - Field segmentation
  - Character segmentation
  - Character recognition
  - Syntactical interpretation.

- Define differentiable modules.
- Pretrain modules with hand-labelled data.
- Define global cost function (e.g., CRF).
- Train with SGD for a few weeks.

Industrially deployed in 1996. Ran billions of checks over 10 years.