Machine learning, a bird’s eye view

Kipton Barros
Los Alamos National Lab.

2\textsuperscript{nd} ML in Solid Earth Geoscience
March 18, 2019

Main Collaborators:
Claudia Hulbert
Bertrand Rouet-Leduc
Nick Lubbers

LA-UR-19-22370
Forms of machine learning

Traditional Stats.
Confidence intervals, hypothesis testing, probabilistic models…

Data mining (1990s?)
Google web search (1998)
Netflix Prize (2006)
10% improvement 2009

Statistical/computational learning theory
Theorems from statistics and functional analysis

Artificial intelligence (1950s--?)
Artificial neurons, McCullough & Pitts (1943)
Turing test (1950)
Chinook checkers (champion in 1994) (solved in 2007)
Deep Blue chess (1996)
AlphaGo (2016)
“AI winter” … AI spring?

Machine learning
“Algorithms that find structure in big datasets, using empirical models and regularizations.” (?)
Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5).

(Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

In the left panel of Figure 4 we qualitatively assess what the network has learned by computing its top-5 predictions on eight test images. Notice that even off-center objects, such as the mite in the top-left, can be recognized by the net. Most of the top-5 labels appear reasonable. For example, only other types of cat are considered plausible labels for the leopard. In some cases (grille, cherry) there is genuine ambiguity about the intended focus of the photograph.

Another way to probe the network’s visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer. If two images produce feature activation vectors with a small Euclidean separation, we can say that the higher levels of the neural network consider them to be similar. Figure 4 shows five images from the test set and the six images from the training set that are most similar to each of them according to this measure. Notice that at the pixel level, the retrieved training images are generally not close in L2 to the query images in the first column. For example, the retrieved dogs and elephants appear in a variety of poses. We present the results for many more test images in the supplementary material.

Computing similarity by using Euclidean distance between two 4096-dimensional, real-valued vectors is inefficient, but it could be made efficient by training an auto-encoder to compress these vectors to short binary codes. This should produce a much better image retrieval method than applying auto-encoders to the raw pixels [14], which does not make use of image labels and hence has a tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar.

7 Discussion

Our results show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network’s performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. Ultimately we would like to use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.
Transfer learning

Handwriting synthesis

Human input

Generated output

Training samples

New styles


Trained using a large database of professional games.
Subsequent learning through self-play.

AlphaZero, 2017
Learning to play video games

Google’s DeepMind AI beats 49 Atari games, exceeds human performance

Training technique: Positive reinforcement learning

ML for physics, interesting ideas

**Chemo-informatics**

Links between chemical structures and activity, molecular finger-printing

**Materials informatics**

Design of new functional materials


**Statistical physics**

MD / DEM potentials
Coarse grained molecular dynamics
Effective models for fluids
Microstructure / phase field modeling

**Geophysics**

Earthquake early warning
Seismic inversion
Flow in fractured media
...

Links to other topics:

- Effective models for fluids
- Microstructure / phase field modeling
Types of Machine Learning

- *Unsupervised Learning*: Learn structure of unlabeled data.

- *Supervised Learning*: Learn the map between inputs and outputs.

- *Reinforcement Learning*: Learn to perform tasks using a reward scheme.

...
Unsupervised learning

Clustering

Manifold learning

Anomaly detection
Supervised learning

Labeled dataset
\[ D = \{ (x_1, y_1), (x_2, y_2), \ldots \} \]

Goal: Learn map
\[ x \rightarrow y = f(x) \]

60,000 handwritten digits (MNIST data)

Labels
0
1
2
3
4
5
6
7
8
9
Puzzle -- no free lunch

Y. S. Abu-Mostafa’s online class “Learning from Data” (edX CS1156x)
Step by step: Linear regression

Source:
\[ y_i = 4 + 0.6x_i + \epsilon_i \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal: Build model
\[ \hat{y}(x) = ??? \]

![Data points and linear regression line](image)
Step by step: Linear regression

Source:
\[ y_i = 4 + 0.6x_i + \epsilon_i \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal: Build model
\[ \hat{y}(x) = ?? \]

Step 1: Split data (80/20)

![Graph showing training and testing data sets](image)
Step by step: Linear regression

Source:
\[ y_i = 4 + 0.6x_i + \epsilon_i \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal: Build model
\[ \hat{y}(x) = \text{???} \]

Step 1: Split data (80/20)

Step 2: Define model
\[ \hat{y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \]

and cost function to optimize
\[ \mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 \]
Step by step: Linear regression

Source:
\[ y_i = 4 + 0.6x_i + \epsilon_i \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal: Build model
\[ \hat{y}(x) = \text{???} \]

Step 1: Split data (80/20)

Step 2: Define model
\[ \hat{y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \]

and cost function to optimize
\[ \mathcal{L} = \sum_{i} (y_i - \hat{y}_i)^2 \]

Step 3: Optimize on training data
\[ \hat{\beta} = (X^T X)^{-1} X^T y \]

\( X_{ij} \) is the \( j^{\text{th}} \) feature of point \( x_i \)
Step by step: Linear regression

**Step 1:** Split data (80/20)

**Step 2:** Define model

\[ \hat{y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x \]

and cost function to optimize

\[ \mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 \]

**Step 3:** Optimize on training data

\[ \hat{\beta} = (X^T X)^{-1} X^T y \]

**Step 4:** Measure performance on test data

\[ R^2_{\text{test}} = 0.84 \]

\[ R^2_{\text{train}} = 0.66 \]

**Performance metric**

\[ R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)}{\sum_i (y_i - \bar{y}_i)} = \begin{cases} 1 & \text{if perfect} \\ 0 & \text{if unpredictive} \end{cases} \]
Linear regression, take 2

Source:  
\[ y_i = 1 + \sin(x_i + 1) + \epsilon_i/2 \]  

Noise term:  
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]  

Goal:  
\[ \hat{y}(x) = ??? \]
Linear regression, take 2

Source:
\[ y_i = 1 + \sin(x_i + 1) + \epsilon_i/2 \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal:
\[ \hat{y}(x) = ??? \]

Step 1: Split data (80/20)
Linear regression, take 2

Source:
\[ y_i = 1 + \sin(x_i + 1) + \epsilon_i/2 \]

Noise term:
\[ \epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 1) \]

Goal:
\[ \hat{y}(x) = \ ? \ ? \ ? \]

Step 1: Split data (80/20)

Step 2: Define feature space
\[ f_i = x^i \in \{1, x, x^2, \ldots, x^N\}, \]
for linear model
\[ \hat{y}(x) = \sum_{i=0}^{N} \beta_i f_i(x), \]
and sum-of-squares cost function
\[ \mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 \]
Linear regression, take 2

Step 1: Split data (80/20)

Step 2: Define model

\[ \hat{y}(x) = \sum_{i=0}^{N} \beta_i f_i(x) \]

\[ f_i = x^i \]

Step 3: Optimize on training data

\[ \hat{\beta} = (X^T X)^{-1} X^T y \]

\[ X_{ij} = f_j(x_i) \]

… how to evaluate various cutoff \( N \)?
Linear regression, take 2

Step 1: Split data (80/20)

Step 2: Define model

Step 3: Optimize on training data

Step 4: Measure $R^2$ scores on validation data

<table>
<thead>
<tr>
<th>$N$</th>
<th>Train</th>
<th>Valid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.27</td>
<td>-0.07</td>
</tr>
<tr>
<td>10</td>
<td>0.73</td>
<td>0.59</td>
</tr>
<tr>
<td>20</td>
<td>0.79</td>
<td>0.71</td>
</tr>
<tr>
<td>40</td>
<td>0.83</td>
<td>0.42</td>
</tr>
</tbody>
</table>
Linear regression, take 2

Step 1: Split data (80/20)

Step 2: Define model

Step 3: Optimize on training data

Step 4: Select polynomial order $N$ with validation data

---

**Final scores**

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Validation</th>
<th>Actual error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.79</td>
<td>0.71</td>
<td>???</td>
</tr>
</tbody>
</table>
A recap

Proper *regularization* of model is context dependent.

https://shapeofdata.wordpress.com/2013/03/26/general-regression-and-over-fitting/
• **Step 1**: Randomly split data in *Training* and *Testing* sets

• **Step 2**: Optimize model from *training* data

• **Step 3**: Estimate *generalization error* on *testing* data

---

**Model Capacity**

http://www.deeplearningbook.org 2016
Hyperparameters

- Selected before training
- Often control model capacity (e.g. forcing smoothness)
- For example: order of polynomial fitting
Data split into *Training*, *Validation*, and *Testing* sets. Validation data is insulated from training. Testing data is insulated from entire training process.
Some interesting ML algorithms

Non-parametric kernel methods

- k-Nearest neighbors
- Support vector machines
- Gaussian process regression

Random forest

- Collection of decision trees

Neural networks

- Deep convolutional nets
- …
Kernel methods

Define kernel $K(x_1, x_2)$ to measure similarity between $x_1$ and $x_2$.

$k$-NN algorithm: Majority vote of $k$-nearest neighbors.

Effectively interpolates nearby data.

Model grows automatically with new data.

Other methods: Support Vector Machines, Kernel Ridge Regression, …
Linear (ridge) regression

\[ \hat{\beta} = (XX^T + \lambda)^{-1}Xy \]

\[ = X^T(X^TX + \lambda)^{-1}y \]

---

**Points**

\[ x_i, x_j \]

---

**Features**

\[ f(x_i) = [f_1(x_i) \ldots f_N(x_i)] \]

\[ f(x_j) = [f_1(x_j) \ldots f_N(x_j)] \]

---

**Similarities**

\[ (X^TX)_{i,j} = f_i \cdot f_j \]

---

**Kernel trick:** bypass features to directly define

\[ (X^TX)_{ij} = K(x_i, x_j) \]

Arbitrary kernel \( K \) completely defines model

\[ \hat{y}(x) = \sum_i \alpha_i K(x_i, x) \]
## Linear regression, take 3
(Gaussian process version)

Step 1: Split data (80/20)

Step 2: Define kernel
\[ K(x_1, x_2) = c_1 e^{-c_2 |x_1 - x_2|^2} + c_3 \delta_{i,j} \]

Step 3: Optimize on training data.
GP automatically handles hyperparameters \( c_1, c_2, c_3 \)!

Step 4: Evaluate performance on testing data.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Actual error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R^2 )</td>
<td>0.75</td>
<td>0.71</td>
</tr>
</tbody>
</table>
Linear regression, take 3

By the way, this is super easy in scikit-learn:

```python
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import WhiteKernel, RBF

gp_kernel = 1.0 * RBF(1) + WhiteKernel(1)
gpr = GaussianProcessRegressor(gp_kernel)

regr = gpr.fit(train[0,:), train[1,:])
print("Kernel params ", regr.kernel_)
print("Training score %f" % regr.score(train[0,:], train[1,:]))
print("Test score %f" % regr.score(test[0,:], test[1,:]))
```

Kernel params  1.41**2 * RBF(length_scale=0.891) + WhiteKernel(noise_level=0.285)
Training score 0.746590
Test score 0.708480
Where can’t we use ML?

How Artificial Intelligence Is Changing Science

By DAN FALK
March 11, 2019

The latest AI algorithms are probing the evolution of galaxies, calculating quantum wave functions, discovering new chemical compounds and more. Is there anything that scientists do that can’t be automated?

https://www.quantamagazine.org/how-artificial-intelligence-is-changing-science-20190311/
Predicting Lab-quakes

Claudia H. Bertrand R.-L. Nick L.

Media buzz

https://www.scientificamerican.com/article/can-artificial-intelligence-predict-earthquakes/


• Central “loader” plate pushed down at constant velocity
• Normal force applied on side plates
• Glass beads (“gouge”) between plates

• Force (“shear stress”) on driving block

• Acoustic emission
Precursor activity

- Impulsive precursors follow Gutenberg–Richter (power law) decay
- Rate of precursors grows exponentially before characteristic event (lab-quake)
Goal: Predict time until next failure (stress drop) from \textit{local} window.
Features

- “Now” prediction, window size ~1/10 cycle
- Extract from acoustic emissions

- Centered moments: variance, skew, kurtosis…
- Amplitude maximum, minimum, extreme quantiles
- Counts over and under various thresholds
- Time correlation measures — power spectrum, autocorrelation…
Decision Trees

• Recursive splitting of training data

• Splits maximize difference between the two branches of the training data

• Leaves predict sample average of training data

Random forest

• Average over many decision trees

Eg. Survival odds of passengers of the Titanic
Predictions

Training $R^2 = 0.91$
Testing $R^2 = 0.89$

- Predictions
- Experimental data
Physics of failure

Stress

Strain

Physics of distant failure

Physics of imminent failure

Tremor-like signals

Impulsive & tremor-like signals

(amplitude x10)
ML, a bird’s eye view:

• *Good data* is crucial (even more important than algorithms).

• *Training* is simply optimization of a *cost function*.

• The essence of machine learning is empirical tuning of model complexity (hyperparameter selection) using validation data.

• Keep *test data* separate from training/validation data!

• [scikit-learn.org](http://scikit-learn.org) is a great place to start. *Gaussian Process* and *Random Forest* methods are particularly easy.

• Neural networks (next talk) are amazingly powerful with large datasets, but take a lot more fiddling.