Machine Learning with Deep Neural Networks

Piotr Luszczek
Terms and Acronyms

- **AI** = Artificial Intelligence
- **ML** = Machine Learning
- **DL** = Deep Learning
- **DNN** = Deep Neural Network
- **CNN** = Convolutional Neural Network
- **LSTM** = Long Short Term Memory
- **RNN** = Recurrent Neural Network
- **ResNet** = Residual Network
- **GANN** = Generative Adversarial Neural Network (also spelled GAN)

Specific types of learning
- **Adversarial learning**
  - Using negative examples
- **Transfer learning**
  - Moving pre-trained layers between models
- **Active learning**
  - “Human-assisted” learning
- **Federated learning**
  - Training data comes from federate (separate) sources
- **Representation (feature) learning**
  - Automating feature discovery
• McCulloch and Pitts
  - "A logical calculus of the ideas immanent in nervous activity"
  - 1943
  - Mathematical model of neurons as basic switching element

• Brain
  - 100 billion neurons \((10^{11})\)
  - 1000 – 10000 connections/neuron
  - Total \(10^{14}\) connections
Straight Line Fitting: Linear Regression

A = \frac{n \sum x_i y_i - (\sum x_i)(\sum y_i)}{n \sum x_i^2 - (\sum x_i)^2}

B = \frac{\sum y_i - A(\sum x_i)}{n}

y = A x + B
Logistic Regression

\[ \log \frac{p(x)}{1 - p(x)} = A x + B \]

\[ p(x) = \frac{1}{1 + e^{-(A x + B)}} \]

Issues:
- Linear classifier
- Linear separability and convergence
- Negative example: XOR

\[ \text{XOR} \]

\[ \begin{array}{cc}
O & X \\
X & O
\end{array} \]
Logistical Regression as a Binary Classifier

\[ f(\vec{x}, \vec{\theta}) \equiv \sigma(\theta^T \vec{x}) = \frac{1}{1 + \exp(-\theta^T \vec{x})} \]

\[ L(\vec{\theta}) = -\sum_{i}^{m} \{ y_i = 1 \} \log[f(x_i, \vec{\theta})] + \{ y_i = 0 \} \log[1 - f(x_i, \vec{\theta})] \]

Find \( \theta \) that minimizes objective function \( L(\theta) \):
- Predicts positives
- Does not predict negatives

Probability that \( f(\vec{x}, \vec{\theta}) \) is 1
Backpropagation Algorithm: Single Layer

Inputs $\times$ Weights $\rightarrow$ Computed Output

$\sigma\{\}$

Error Signal $\rightarrow$ Correct Output

✔
Backpropagation Algorithm: Hidden Layer

\[ w^{(k+1)} = w^{(k)} - \eta \left[ \sum_{i,j}^{m} w_{ij} (y^{(k+1)} - y^{(k)}) \right] \frac{df(z)}{dz} \]

http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.htm
Stacking Neural Layers

Input #1
Input #2
Input #3
Input #4

Hidden Layer

Output layer

Output
Example Neural Network: XOR Gate

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.96</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.97</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Finding Optimal Weights

Find $\theta$ that minimizes objective function $L(\theta)$

$$\nabla_\theta L(\tilde{\theta}) = -\sum_i^m x_i \cdot (y_i - f(x_i, \theta))$$

$$\tilde{\theta}^{(k+1)} = \tilde{\theta}^{(k)} - \eta \nabla_\theta L(\tilde{\theta})$$

- Stochastic Gradient Descent is a generic method
- Used in Adaline, Perceptron, K-Means, SVM, Lasso
- Considers empirical risk vs. expected risk
- Related to Randomized Kaczmarz
Stochastic Gradient Descent Overview

Minimize $F(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$

Initialize $x_0$

for each $j = 1, 2, ...$

randomly draw $i \leftarrow i_j$

$x^{j+1} \leftarrow x^j - \gamma \nabla f_i(x^{(j)})$

Goal: non-asymptotic bounds on $E \|x^j - \bar{x}\|$
Practical Aspects and Implementations
connections for input neuron = width · height

connections for input neuron = tile(width · height)

Local connectivity and overlap

Other tricks:
- Weight-tying
- Pooling, max-pooling
- Contrast normalization
- Rectified Linear Units (ReLu)
- Dropout to avoid overfitting
Training with Backpropagation: Points of Interest

- We need multiple layers for complicated problems
  - Convexity, separability are not required

- Problems
  - Diminishing gradients
  - Unbound weights
    - Normalize
    - Dropout of sub-threshold weights
  - Overfitting
    - Addressed with regularization
  - Sensitivity to training set
    - Who ordered your training images (on disk)?
  - Training time vs. recall time
    - Weeks, days, hours vs. millisecond
    - Few training groups
    - Almost everybody performs inference
Popular Deep Learning Models
AlexNet: Convolutional Neural Network

- Won ImageNet 2012 LSVRC
  - 60 million parameters
  - 832 million MAC ops
  - Krizhevsky et al. 2012

![Diagram of AlexNet architecture]

Task 2: Detection

- CNN
- DPM SVM1
- DPM SVM2

% Error

<table>
<thead>
<tr>
<th>Method</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>20</td>
</tr>
<tr>
<td>SIFT+FV</td>
<td>15</td>
</tr>
<tr>
<td>SVM1</td>
<td>10</td>
</tr>
<tr>
<td>SVM2</td>
<td>5</td>
</tr>
<tr>
<td>NCM</td>
<td>0</td>
</tr>
</tbody>
</table>

Task 1: Classification

% Error

<table>
<thead>
<tr>
<th>Method</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>25</td>
</tr>
<tr>
<td>SIFT+FV</td>
<td>20</td>
</tr>
<tr>
<td>SVM1</td>
<td>15</td>
</tr>
<tr>
<td>SVM2</td>
<td>10</td>
</tr>
<tr>
<td>NCM</td>
<td>5</td>
</tr>
</tbody>
</table>

http://www.slideshare.net/yuhuang/deep-learning-for-image-denoising-superresolution-27435126
GoogLeNet (2014)

Inception Module
Deep Residual Learning for Image Recognition (1512.03385)

ResNet design
- Helps with gradient stagnation.
- Resilience when deleting layers.
- Allows deeper networks.

VGG-19
- plain
  - Pool, /2
  - 3x3 conv, 64
  - 3x3 conv, 64
  - 3x3 conv, 128
  - 3x3 conv, 128
  - 3x3 conv, 256
  - 3x3 conv, 256
  - 3x3 conv, 256

- residual
  - Pool, /2
  - 3x3 conv, 64
  - 3x3 conv, 64
  - 3x3 conv, 128
  - 3x3 conv, 128
  - 3x3 conv, 256
  - 3x3 conv, 256
  - 3x3 conv, 256

ResNeXt
## Deep Networks: Summary

<table>
<thead>
<tr>
<th>Model</th>
<th>Size (MB)</th>
<th>Top-1 Accuracy</th>
<th>Top-5 Accuracy</th>
<th>Parameters</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xception</td>
<td>88</td>
<td>0.79</td>
<td>0.945</td>
<td>22910480</td>
<td>126</td>
</tr>
<tr>
<td>VGG16</td>
<td>528</td>
<td>0.713</td>
<td>0.901</td>
<td>138357544</td>
<td>23</td>
</tr>
<tr>
<td>VGG19</td>
<td>549</td>
<td>0.713</td>
<td>0.9</td>
<td>143667240</td>
<td>26</td>
</tr>
<tr>
<td>ResNet50</td>
<td>98</td>
<td>0.749</td>
<td>0.921</td>
<td>25636712</td>
<td>-</td>
</tr>
<tr>
<td>ResNet101</td>
<td>171</td>
<td>0.764</td>
<td>0.928</td>
<td>44707176</td>
<td>-</td>
</tr>
<tr>
<td>ResNet152</td>
<td>232</td>
<td>0.766</td>
<td>0.931</td>
<td>60419944</td>
<td>-</td>
</tr>
<tr>
<td>ResNet50V2</td>
<td>98</td>
<td>0.76</td>
<td>0.93</td>
<td>25613800</td>
<td>-</td>
</tr>
<tr>
<td>ResNet101V2</td>
<td>171</td>
<td>0.772</td>
<td>0.938</td>
<td>44675560</td>
<td>-</td>
</tr>
<tr>
<td>ResNet152V2</td>
<td>232</td>
<td>0.78</td>
<td>0.942</td>
<td>60380648</td>
<td>-</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>92</td>
<td>0.779</td>
<td>0.937</td>
<td>23851784</td>
<td>159</td>
</tr>
<tr>
<td>InceptionResNetV2</td>
<td>215</td>
<td>0.803</td>
<td>0.953</td>
<td>55873736</td>
<td>572</td>
</tr>
<tr>
<td>MobileNet</td>
<td>16</td>
<td>0.704</td>
<td>0.895</td>
<td>4253864</td>
<td>88</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>14</td>
<td>0.713</td>
<td>0.901</td>
<td>3538984</td>
<td>88</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>33</td>
<td>0.75</td>
<td>0.923</td>
<td>8062504</td>
<td>121</td>
</tr>
<tr>
<td>DenseNet169</td>
<td>57</td>
<td>0.762</td>
<td>0.932</td>
<td>14307880</td>
<td>169</td>
</tr>
<tr>
<td>DenseNet201</td>
<td>80</td>
<td>0.773</td>
<td>0.936</td>
<td>20242984</td>
<td>201</td>
</tr>
<tr>
<td>NASNetMobile</td>
<td>23</td>
<td>0.744</td>
<td>0.919</td>
<td>5326716</td>
<td>-</td>
</tr>
<tr>
<td>NASNetLarge</td>
<td>343</td>
<td>0.825</td>
<td>0.96</td>
<td>88949818</td>
<td>-</td>
</tr>
</tbody>
</table>

- **AlexNet**
  - 5 convolutional layers
- **VGG**
  - 16 convolutional layers
  - 19 convolutional layers
- **GoogleNet (Inception_v1)**
  - 22 convolutional layers
- **ResNet v1 and v2**
  - 50
  - 101
  - 152
Generative Adversarial Network: Overview

- **Generator** (artist)
  - Latent space
  - Noise

- **Discriminator** (critic)
  - Real samples
  - Correct?

Potential problems:
- Mode collapse
- No convergence to Nash equilibrium

Solutions:
- Wasserstein GAN
  - Use Wasserstein (earthmover) distance
DNN → Tensors → SGEMM

Input images

Filter

Output images

Storage format (for expanded data):
- NCHW (natural, naive)
- CHWN
From SGEMM to DNN with Autotuning
Deep Learning Data Sets
Popular Data Sets

- **Data Sets**
  - **MNIST**
    - Handwritten digits
  - **CIFAR10 and CIFAR100**
    - 50,000 small images (32x32)
    - 10 or 100 classes of images
  - **ImageNet**
  - **Cambridge Analytica**
    - Personality types
ImageNet Collection of Classified Images

- 15 million images
  - Tagged with concepts (synonym sets = synsets)
- Some images have
  - SIFT features
  - Bounding box annotations
- Availability
  - Links to images
  - API
  - Full download for educational purposes
- Competition
  - ImageNet Large Scale Visual Recognition Challenge
- There are precision/recall numbers for human testers
ImageNet Training

- **ResNet**
  - Deep Residual Learning

- **1 hour on 256 GPUs**
  - Accurate, Large Minibatch Stochastic Gradient Descend

- **15 minutes on 1024 GPUs**
  - [https://github.com/chainer/chainermn](https://github.com/chainer/chainermn)

- **Performance highlights**
  - MPI Allreduce()
  - Synchronous vs. Asynchronous gradient updates
  - NVLink communication primitives: NVIDIA nccl (pronounced “nickel”)
Deep Learning Software Stack
Software for Deep Learning

● Unlike HPC, Deep Learning community has low tolerance for complexity
● Some languages
  - Lua
    • Torch with Lua.JIT
  - Julia
    • Flux: zygote, capsnet
  - Python
    • PyTorch
    • TensorFlow
      - Python, C++, JXA
      - TensorFlow.js
      - TensorFlow Lite
    • Keras
    • ...
  - Jupyter
  - Vendors: cuDNN, MIOpen, MKL-DNN

● Python software stack
  - NumPy
  - SciPy
  - Matplotlib, Seaborn
  - Numba
  - Pandas
  - Scikit-learn
  - scikit-image
  - Scikit-Optimize, scikit-opt
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x)

# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker', 0.1122357),
# (u'n02504458', u'African_elephant', 0.061040461)]
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
from keras.applications.vgg19 import VGG19
from keras.preprocessing import image
from keras.applications.vgg19 import preprocess_input
from keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').output)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

block4_pool_features = model.predict(x)
Fine-tune InceptionV3 on a new set of classes

```python
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in model.layers:   layer.trainable = False
# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
from keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy')
model.fit_generator(...)
```

# at this point, the top layers are well trained and we can start fine-tuning
# convolutional layers from inception V3. We will freeze the bottom N layers
# and train the remaining top layers.

```python
# let's visualize layer names and layer indices to see how many layers
# we should freeze:
for i, layer in enumerate(base_model.layers):
    print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:   layer.trainable = False
for layer in model.layers[249:]:   layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
from keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy')
model.fit_generator(...)
```

# train the model on the new data for a few epochs
model.fit_generator(...)
from keras.applications.inception_v3 import InceptionV3
from keras.layers import Input

# this could also be the output a different Keras model or layer
input_tensor = Input(shape=(224, 224, 3))  # this assumes K.image_data_format() == 'channels_last'

model = InceptionV3(input_tensor=input_tensor, weights='imagenet', include_top=True)
Deep Learning Hardware Stack
Computational Needs According to OpenAI

Two Distinct Eras of Compute Usage in Training AI Systems

https://openai.com/blog/ai-and-compute/
Modern Hardware: Lower Precision for Deep Learning

- Hardware (company)
  - GPU Tensor Cores (NVIDIA)
  - TPU MXU (Google)
  - Zion (Facebook)
  - DaVinci (Huawei)
  - Dot-product engine (HPE)
  - Eyeriss (Amazon)
  - Wafer Scale Engine (Cerebras)
  - Nervana (Intel)
  - Deep Learning Boost (Intel AI)
  - Graph Core
  - ...

- Lower-precision benchmarks
  - Baidu
  - Dawn
  - mlperf
  - Deep500
  - ...
  - HPL-AI

60+
FP16 Hardware for DNN Training/Inference

- **AMD**
  - Radeon Instinct MI5, MI8, MI25, MI50, MI60
- **ARM**
  - NEON VFP FP16 in V8.2-A
- **Intel**
  - Cascade Lake
- **NVIDIA Pascal and Volta**
  - P100, Turing, TX1, Jetson Nano
  - Non-Tesla cards (Quadro, GeForce 11xy)
  - V100, DGX-1, DGX-2
    - Tensor core with 32-bit intermediates
- **Supercomputers**
  - TSUBAME 3 (47 Pflop/s FP16)
  - Tokyo Tech
  - Piz Daint
  - Summit +3 Eflop/s FP16
  - Sierra
- **Google**
  - TPU 1: INT8 ~30 TOPS
  - TPU 2 pod: 11.5 Pflop/s
  - TPU 3 pod: 20-90 Pflop/s
- **NVIDIA**
  - DRIVE PX 2: 24 DL TOPS
- **Intel**
  - Xeon Phi Knights Mill
  - Nervana
Modern Hardware and Floating-Point Formats

Tensor Cores

FP32

FP16

BF16

INT

FP64

FP32

FP32

HLF

HLF

HLF

MXU

128x128
# Major Floating Point Formats from IEEE 754 (2008)

<table>
<thead>
<tr>
<th>Precision</th>
<th>Width</th>
<th>Exponent bits</th>
<th>Mantissa bits</th>
<th>Epsilon</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadruple</td>
<td>128</td>
<td>15</td>
<td>112</td>
<td>$O(10^{-34})$</td>
<td>$1.2 \times 10^{4932}$</td>
</tr>
<tr>
<td>Extended</td>
<td>80</td>
<td>15</td>
<td>64</td>
<td>$O(10^{-19})$</td>
<td>$1.2 \times 10^{4932}$</td>
</tr>
<tr>
<td>Double</td>
<td>64</td>
<td>11</td>
<td>52</td>
<td>$O(10^{-16})$</td>
<td>$1.8 \times 10^{308}$</td>
</tr>
<tr>
<td>Single</td>
<td>32</td>
<td>8</td>
<td>23</td>
<td>$O(10^{-7})$</td>
<td>$3.4 \times 10^{38}$</td>
</tr>
<tr>
<td>Half*</td>
<td>16</td>
<td>5</td>
<td>10</td>
<td>$O(10^{-3})$</td>
<td>65504</td>
</tr>
<tr>
<td>BFloat</td>
<td>16</td>
<td>8</td>
<td>7</td>
<td>$O(10^{-2})$</td>
<td>$3.4 \times 10^{38}$</td>
</tr>
</tbody>
</table>

*Only storage format is specified. IEEE 2018 covers the compute rules.

- IEEE 754 2018 standard update includes 16-bit for computing.
In conclusion...
Future Work

- **Optimization**
  - SGD, momentum, ADAM, …

- **Hyper-parameter selection**
  - AutoML

- **Natural language processing**
  - BERT, Transformer, Transformer2, Megatron

- **Reinforcement learning**
  - “Stochastic branch-and-bound”
  - Deep RL, Q-learning

- **Capsule networks**
  - Not yet established, if ever