Fundamentals of Deep (Artificial) Neural Networks (DNN)

Greg Tsagkatakis
CSD - UOC
ICS - FORTH
Accelerated growth

1. The accelerating pace of change...

- Agricultural Revolution: 6,000 years
- Industrial Revolution: 120 years
- Light-bulb: 90 years
- Moon landing: 22 years
- World Wide Web: 9 years
- Human genome sequenced: Current year

2. ...and exponential growth in computing power...

- Computer technology, shown here climbing dramatically by powers of 10, is now progressing more each hour than it did in its entire first 90 years

- **COMPUTER RANKINGS**
  - By calculations per second per $1,000

- **Analytical engine**
  - Never fully built, Charles Babbage’s invention was designed to solve computational and logical problems

- **Colossus**
  - The electronic computer, with 1,500 vacuum tubes, helped the British crack German codes during WW II

- **UNIVAC I**
  - The first commercially marketed computer, used to tabulate the U.S. Census, occupied 943 cu. ft.

- **Apple II**
  - At a price of $1,298, the compact machine was one of the first massively popular personal computers

3. ...will lead to the Singularity

- **Power Mac G4**
  - The first personal computer to deliver more than 1 billion floating-point operations per second
Brief history of DL

1958: Perceptron
1969: Perceptron criticized
1974: Backpropagation
1998: Convolution Neural Networks for Handwritten Recognition
2006: Restricted Boltzmann Machine
2012: Google Brain Project on 16k Cores
2012: AlexNet wins ImageNet

awkward silence (Al Winter)
1995: SVM reigns
2012: ImageNet
Why Today?

Lots of Data

Why deep learning

How do data science techniques scale with amount of data?
Why Today?

Lots of Data
Deeper Learning
Why Today?

Lots of Data
Deep Learning
More Power

https://www.slothparadise.com/what-is-cloud-computing/
Apps: Gaming

History of Game AI
By: Andrey Kurenkov

1956: Dartmouth Conference - the birth of AI
1958: Bernstein's Chess AI - first fully functional chess AI developed
1962: Checkers AI Wins - Samuel's program wins game against person
1968: Zobrist's AI - first Go AI, beats human amateur
1974: Kaissa - first world computer chess champion
1982: TD-Gammon - RL and neural net based backgammon AI shown
1986: Backprop - multi-layer neural net approach widely known
1989: CNN - convolutional nets first demonstrated
1992: Monte Carlo Go - first research on Go with stochastic search
1994: CHINOOK - checkers AI draws with world champion
1997: Deep Blue - IBM chess AI beats world champion
1998: AlphaGo - Deep Learning+MCST Go AI beats top human
2006: MCTS Go - French researchers advance Go AI with MCTS
2008: Crazy Stone - MCTS Go AI beats 4 dan player
2012: Zen19 - MCTS based Go AI reaches 5-dan rank
2014: DeepMind - Google buys deep-RL AI company for $400M
Key components of ANN

- Architecture (input/hidden/output layers)
Key components of ANN

- Architecture (input/hidden/output layers)
- Weights
Key components of ANN

- Architecture (input/hidden/output layers)
- Weights
- Activations

LINEAR

LOGISTIC / SIGMOIDAL / TANH

RECTIFIED LINEAR (ReLU)
Perceptron: an early attempt

Activation function

\[ \hat{f}(x) = \sigma(w \cdot x + b) \]

\[ \sigma(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases} \]

Need to tune \( w \) and \( b \)
A neuron is of the form $\sigma(w \cdot x + b)$ where $\sigma$ is an activation function.

We just added a neuron layer!

We just introduced non-linearity!
A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

Sasen Cain (@spectralradius)
Training & Testing

Training: determine weights
- Supervised: labeled training examples
- Unsupervised: no labels available
- Reinforcement: examples associated with rewards

Testing (Inference): apply weights to new examples
Training DNN

1. Get batch of data
2. Forward through the network -> estimate loss
3. Backpropagate error
4. Update weights based on gradient
BackPropagation

Chain Rule in Gradient Descent: Invented in 1969 by Bryson and Ho

Defining a loss/cost function

Assume a function

\[ J(x, y; \theta) = \frac{1}{2} \sum (y - f(x; \theta))^2 \]

\[ f(x; \theta) = w^T x + b , \quad \theta = \{w, b\} \]

Types of Loss function

- Hinge \( J(x, y) = max\{0, 1 - xy\} \)
- Exponential \( J(x, y) = exp(-xy) \)
- Logistic \( J(x, y) = log_2(1 + exp(-xy)) \)
Gradient Descent

Minimize function $J$ w.r.t. parameters $\theta$

$$\theta^* = \theta - n \times \nabla J(y, x; \theta)$$

- **Gradient**
  \[
  \nabla J(x) = \left( \frac{\partial J(x)}{\partial x_1}, \frac{\partial J(x)}{\partial x_2}, \ldots, \frac{\partial J(x)}{\partial x_n} \right)
  \]

- **Chain rule**

**New weights** → $\theta^*$ → Gradient → $\theta$ → **Old weights**

**Learning rate**

---

17
Given: \( y = g(u) \) and \( u = h(x) \).

**Chain Rule:**

\[
\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k
\]
BackProp

Chain rule:

- Single variable

\[
\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}.
\]

- Multiple variables

\[
\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.
\]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

E.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}
\]
Visualization
Training Characteristics

Over-fitting

Under-fitting
Supervised Learning
Supervised Learning

Exploiting prior knowledge
- Expert users
- Crowdsourcing
- Other instruments

Data
Labels

Model
Prediction

Spiral
Elliptical
?
State-of-the-art (before Deep Learning)

Support Vector Machines
- Binary classification
State-of-the-art (before Deep Learning)

Support Vector Machines
- Binary classification
- Kernels $\leftrightarrow$ non-linearities
State-of-the-art (before Deep Learning)

Support Vector Machines
- Binary classification
- Kernels ↔ non-linearities

Random Forests
- Multi-class classification
State-of-the-art (before Deep Learning)

Support Vector Machines
- Binary classification
- Kernels <-> non-linearities

Random Forests
- Multi-class classification

Markov Chains/Fields
- Temporal data
State-of-the-art (since 2015)

Deep Learning (DL)

Convolutional Neural Networks (CNN) <-> Images

Recurrent Neural Networks (RNN) <-> Audio
Convolutional Neural Networks

(Convolution + Subsampling) + () ... + Fully Connected
Convolutional Layers

$32 \times 32 \times 1$ Image

$5 \times 5 \times 1$ filter

K filters

$28 \times 28 \times K$ activation map

$\sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n)K(m, n)$

$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n)K(-m, -n)$
Convolutional Layers

Characteristics
- Hierarchical features
- Location invariance

Parameters
- Number of filters \((32, 64, \ldots)\)
- Filter size \((3 \times 3, 5 \times 5)\)
- Stride \((1)\)
- Padding \((2, 4)\)
Subsampling (pooling) Layers

<-> downsampling

- Scale invariance

Parameters
- Type
- Filter Size
- Stride
Activation Layer

Introduction of non-linearity
- Brain: thresholding -> spike trains

Identity (Linear)
- $identity(x) = x$

Sigmoid
- $sigmoid(x) = \frac{1}{1 + e^{-x}}$

Tanh (Hyperbolic Tangent)
- $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Gaussian
- $gaussian(x) = e^{-x^2/2^2}$
Activation Layer

ReLU: $x = \max(0, x)$

- Simplifies backprop
- Makes learning faster
- Avoids saturation issues
- ~ non-negativity constraint

(Note: The brain)
Fully Connected Layers

Full connections to all activations in previous layer

Typically at the end

Can be replaced by conv
LeNet [1998]

[LeCun et al., 1998]
AlexNet [2012]

Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, [ImageNet ILSVRC challenge](http://vision03.csail.mit.edu/cnn_art/data/single_layer.png) in 2012

http://vision03.csail.mit.edu/cnn_art/data/single_layer.png
### VGGnet

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11 weight</td>
<td>11 weight</td>
<td>13 weight</td>
<td>16 weight</td>
<td>16 weight</td>
<td>19 weight</td>
</tr>
<tr>
<td></td>
<td>layers</td>
<td>layers</td>
<td>layers</td>
<td>layers</td>
<td>layers</td>
<td>layers</td>
</tr>
</tbody>
</table>

_D: VGG16_  
_E: VGG19_  
All filters are 3x3  

More layers smaller filters
Inception (GoogLeNet, 2014)

Inception module with dimensionality reduction
Residuals

\[ F(x) \]

\[ H(x) = F(x) + x \]
ResNet, 2015

Training protocols

Fully Supervised
- Random initialization of weights
- Train in supervised mode (example + label)

Unsupervised pre-training + standard classifier
- Train each layer unsupervised
- Train a supervised classifier (SVM) on top

Unsupervised pre-training + supervised fine-tuning
- Train each layer unsupervised
- Add a supervised layer
Batch Normalization

**Input:** Values of $x$ over a mini-batch: $B = \{ x_1...m \}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{ y_i = \text{BN}_{\gamma, \beta}(x_i) \}$

\[ \mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean} \]

\[ \sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance} \]

\[ \hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalize} \]

\[ y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{// scale and shift} \]

(a) Without BN (b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]
Transfer Learning

 Pixels  Layer 1  Layer 2  Layer L

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

\[\ldots\]  

\[\ldots\]  

\[\ldots\]  

\[\ldots\]  

\[\text{elephant}\]  

\[\ldots\]
Transfer Learning

![Diagram of a neural network with input images of an elephant and a brain scan, outputting classifications of "elephant" and "Healthy." The network consists of layers transforming pixel inputs into final layer outputs.](image)

The diagram illustrates how different layers of a neural network transform input data (pixels from an elephant and a brain scan) through layers 1 to L, leading to final classifications of "elephant" and "Healthy."
Layer Transfer - Image

Source: 500 classes from ImageNet
Target: another 500 classes from ImageNet

Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, “How transferable are features in deep neural networks?”, NIPS, 2014
ImageNET

• ~14 million labeled images, 20k classes
• Images gathered from Internet
• Human labels via Amazon MTurk
• ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes

www.image-net.org/challenges/LSVRC/
## Summary: ILSVRC 2012-2015

<table>
<thead>
<tr>
<th>Team</th>
<th>Year</th>
<th>Place</th>
<th>Error (top-5)</th>
<th>External data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(AlexNet, 7 layers)</td>
<td>2012</td>
<td>-</td>
<td>16.4%</td>
<td>no</td>
</tr>
<tr>
<td>SuperVision</td>
<td>2012</td>
<td>1st</td>
<td>15.3%</td>
<td>ImageNet 22k</td>
</tr>
<tr>
<td>Clarifai – NYU (7 layers)</td>
<td>2013</td>
<td>-</td>
<td>11.7%</td>
<td>no</td>
</tr>
<tr>
<td>Clarifai</td>
<td>2013</td>
<td>1st</td>
<td>11.2%</td>
<td>ImageNet 22k</td>
</tr>
<tr>
<td>VGG – Oxford (16 layers)</td>
<td>2014</td>
<td>2nd</td>
<td>7.32%</td>
<td>no</td>
</tr>
<tr>
<td>GoogLeNet (19 layers)</td>
<td>2014</td>
<td>1st</td>
<td>6.67%</td>
<td>no</td>
</tr>
<tr>
<td>ResNet (152 layers)</td>
<td>2015</td>
<td>1st</td>
<td>3.57%</td>
<td></td>
</tr>
<tr>
<td>Human expert*</td>
<td></td>
<td></td>
<td>5.1%</td>
<td></td>
</tr>
</tbody>
</table>

Skin cancer detection

- Basal cell carcinomas
  - Epidermal benign
  - Epidermal malignant
  - Melanocytic benign
  - Melanocytic malignant
- Squamous cell carcinomas
- Melanomas
- Nevi
- Seborrheic keratoses
CNN & FMRI
Different types of mapping

- **Image classification**
- **Sentiment analysis**
- **Synced sequence (video classification)**
- **Machine translation**
- **Image captioning**
Recurrent Neural Networks

Motivation

- Feed forward networks accept a fixed-sized vector as input and produce a fixed-sized vector as output
- fixed amount of computational steps
- recurrent nets allow us to operate over sequences of vectors

Use cases

- Video
- Audio
- Text
RNN Architecture

\[ o(t) \]

Output

\[ w e i g h t s \ V \]

Hidden Units

\[ w e i g h t s \ U \]

Inputs

\[ x(t) \]

Delay

\[ s(t) \]

\[ s(t - 1) \]

\[ w e i g h t s \ W \]
Unfolding RNNs

- Each node represents a layer of network units at a single time step.
- The same weights are reused at every time step.
Unsupervised Learning
Agenda

- Autoencoders
- Sparse coding
- Generative Adversarial Networks
Autoencoders

Unsupervised feature learning

Network is trained to output the input (learn identify function).

\[ J = \frac{1}{m} \sum_{i=1}^{m} \| \hat{x} - x \|_2 \]

Encoder

\[ f(x) = h = z(W_1 x + b_1) \]

Decoder

\[ g(f(x)) = \hat{x} = z(W_2 h + b_2) \]
Regularized Autoencoders

Sparse neuron activation

$$J_{sparse} = \sum \| \hat{x} - x \|^2_2 + \beta \sum KL(p, \hat{p})$$

Contractive auto-encoders

$$J_{contractive} = \sum \| \hat{x} - x \|^2_2 + \beta \left\| \frac{\partial f(x)}{\partial x} \right\|^F$$

Denoising auto-encoders

Convolutional AE

$$f(x) = h = z(W_1 * x + b_1)$$
Stacked Autoencoders

Extended AE with multiple layers of hidden units

Challenges of Backpropagation

Efficient training

- Normalization of input

Unsupervised pre-training

- Greedy layer-wise training
- Fine-tune w.r.t criterion

Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning, 2009
Layer 1

Layer 2

Layer 3

\[ h_\theta(x) \]
New representation for input.
Train parameters so that $h_\theta(x) \approx a$, subject to $b_i$'s being sparse.
Train parameters so that $h_\theta(x) \approx a$, subject to $b_i$'s being sparse.
Train parameters so that \( h_\theta(x) \approx a \), subject to \( b_i \)'s being sparse.
New representation for input.

\[
\begin{bmatrix}
  b_1 \\
  b_2 \\
  b_3 
\end{bmatrix}
\]
\[ h_\theta(x) \approx b \]
Use \([c_1, c_3, c_3]\) as representation to feed to learning algorithm.

New representation for input.
TensorFlow

Deep learning library, open-sourced by Google (11/2015)

TensorFlow provides primitives for
  ◦ defining functions on tensors
  ◦ automatically computing their derivatives

What is a tensor

What is a computational graph

Material from lecture by Bharath Ramsundar, March 2018, Stanford
Introduction to Keras

Official high-level API of TensorFlow
- Python
- 250K developers

Same front-end <-> Different back-ends
- TensorFlow (Google)
- CNTK (Microsoft)
- MXNet (Apache)
- Theano (RIP)

Hardware
- GPU (Nvidia)
- CPU (Intel/AMD)
- TPU (Google)

Companies: Netflix, Uber, Google, Nvidia...

Material from lecture by Francois Chollet, 2018, Stanford
Keras models

Installation

- Anaconda -> Tensorflow -> Keras

Build-in

- Conv1D, Conv2D, Conv3D...
- MaxPooling1D, MaxPooling2D, MaxPooling3D...
- Dense, Activation, RNN...

The Sequential Model

- Very simple
- Single-input, Single-output, sequential layer stacks

The functional API

- Mix & Match
- Multi-input, multi-output, arbitrary static graph topologies
Sequential

```python
>>> from keras.models import Sequential

>>> model = Sequential()

>>> from keras.layers import Dense

>>> model.add(Dense(units=64, activation='relu', input_dim=100))

>>> model.add(Dense(units=10, activation='softmax'))

>>> model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

>>> model.fit(x_train, y_train, epochs=5, batch_size=32)

>>> loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

>>> classes = model.predict(x_test)
```
Functional

```python
>>> from keras.layers import Input, Dense
>>> from keras.models import Model
>>> inputs = Input(shape=(784,))
>>> x = Dense(64, activation='relu')(inputs)
>>> x = Dense(64, activation='relu')(x)
>>> predictions = Dense(10, activation='softmax')(x)
>>> model = Model(inputs=inputs, outputs=predictions)
>>> model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
>>> model.fit(data, labels)
```
References


