INTRODUCTION TO DEEP LEARNING WITH TENSORFLOW

Fabien Baradel
PhD Candidate

fabienbaradel.github.io
@fabienbaradel
fabien.baradel@insa-lyon.fr
DEEP LEARNING?

TENSORFLOW?
TENSORFLOW BACKGROUND

- 6 months internship at Xerox Research Center Europe:
  - « Unsupervised Domain Adaptation »
  - Image recognition & Sentimental text classification
  - Machine Learning for Services team
  - Grenoble, France

- Phd Candidate at LIRIS - INSA Lyon since October 2016:
  - « Deep Learning for human understanding: gestures, poses, activities »
  - Imagine team
  - Supervisors: Christian Wolf & Julien Mille
  - Working with videos
DEEP LEARNING
FOR HUMAN UNDERSTANDING

- Action recognition => Classification
- Sequence Learning
- Supervised Learning
DEEP LEARNING FOR HUMAN UNDERSTANDING

- Microsoft Kinect - Xbox
Microsoft Kinect v2:

- 3D joint location
- 25 joints

Video = sequence of frames
Skeleton not enough (looking at book, looking at smartphone)
SELF-DRIVING CARS

Tesla

• [https://www.youtube.com/watch?v=CxanE_W46ts](https://www.youtube.com/watch?v=CxanE_W46ts)
Virtual Machine: USB key

- Python 2.7
- Tensorflow
- Ubuntu 16.04
- Datasets

Seminar slides & Code corrections:

https://fabienbaradel.github.io
Introduction
  • Basics of Tensorflow
  • Machine Learning: analytic solution vs. gradient descent

Supervised Learning (image recognition)
  • Neural networks reminder
  • Convolution Networks
  • Going deeper with ConvNets

Unsupervised Learning
  • Autoencoder
  • Generative Adversial Network

Sequence modelling
  • RNN, LSTM
  • Word2vec

Reinforcement Learning
  • Deep Q-learning
  • Frozen Lake
INTRODUCTION
WHAT IS TENSORFLOW?

• A python library
• pip install tensorflow
• Google
• open-source
• library for numerical computation using data flow graphs
• CPU and GPU
• Research & Industry

Companies using TensorFlow

ARM
Snapchat
Quantiphi
Airbus Defence & Space
CIST
CEVA
Google
Movidius
UBER
JD.COM
Twitter
DeepMind
eBay
Xiaomi
Dropbox
• INTRODUCTION EXERCISES

• Difference between constant/variable and placeholder

• Constant = a fixed Variable

• With placeholder you need to feed data to your graph during your session

• Tensorflow workflow:
  • Draw your graph
  • Feed data
  • … and optimize
• Open « math_ops.py »
• Same thing with integer
• Mathematical operation done only using Tensorflow library (no numpy or else)
• Draw the schema of the code
ANALYTIC SOLUTION IN ML

Linear regression \( y = X\beta + \epsilon \)

Least Squares solution
\( \hat{\beta} = \text{arg min} ||X\beta - y||_2 \)

\[ \hat{\beta} = (X^TX)^{-1}X^TY \]

Could solve the same problem by solving the optimization problem using gradient descent
GRADIENT DESCENT

• Goal: minimizing an function
• Random initialization of parameters
• At time t, gradient gives the slope of the function
• Iterative process
• Updating the parameters in the positive direction of the gradient according to a learning rate
• Repeat until convergence
BATCH STOCHASTIC GRADIENT DESCENT

Linear regression  \( y = X\beta + \epsilon \)

Minimize a loss function:

\[
J(\beta) = \sum_{i=1}^{N} (X_i\beta - y_i)^2
\]

\( \hat{\beta} = \arg\min J(\beta) \)

- Initialize \( \hat{\beta}_0 \) randomly
- Choose a learning rate \( \eta \)
- for \( t \) in range(training_step):
  - Compute the loss
    \[
    J(\hat{\beta}_t) = \sum_{i=1}^{N} (X_i\hat{\beta}_t - y_i)^2
    \]
  - Update parameters
    \[
    \hat{\beta}_{t+1} = \hat{\beta}_t - \eta \nabla J(\hat{\beta}_t)
    \]

MINI-BATCH SGD

- Initialize $\hat{\beta}_0$ randomly
- Choose a learning rate $\eta$
- Choose a batch size $n$

for $t$ in range(training_step):
  - Pick a random sample $S^m_t$ from training data
  - Compute the loss function
    \[ J(\hat{\beta}_t) = \sum_{i \in S^m_t} (X_i\hat{\beta}_t - y_i)^2 \]
  - Update parameters
    \[ \hat{\beta}_{t+1} = \hat{\beta}_t - \eta \nabla J(\hat{\beta}_t) \]

SGD = stochastic gradient descent

Initialization is important!
Learning rate too!
Need a validation set to avoid overfitting

Neural nets always trained with mini-batch SGD!
• Go to the Github repo and complete the codes:
  * SGD/linear_regression_exo.py
  * SGD/binary_classif_exo.py

```python
import ipdb; ipdb.set_trace()
```

http://playground.tensorflow.org/
https://wookayin.github.io/TensorflowKR-2016-talk-debugging/
NEURAL NETWORKS
MNIST DATASET

- Handwritten digits
- 60,000 training data and 10,000 test data
- 28x28 grayscale images
- matrix of size 28x28 with value between 0 and 255
- data preprocessing = rescaling to [0,1]
MULTINOMIAL LOGISTIC REGRESSION ON MNIST: CREATE THE GRAPH

**Reasoning for Logarithm:**

The formula for computing the cross-entropy loss is:

\[
J(\hat{W}) = -y \times \log(\hat{y})
\]

**Why Log?**

[Link to explanation](http://colah.github.io/posts/2015-09-Visual-Information/)
MULTINOMIAL LOGISTIC REGRESSION ON MNIST: CREATE THE GRAPH

Images are vectorized and passed to the model. The model is parameterized by learnable parameters $\hat{W}_t$. The logits are computed and then passed through the softmax function to produce predictions. The predictions are compared to the labels to compute the cross-entropy loss $J(\hat{W}_t)$. The loss is then used to update the parameters $\hat{W}_t$ using SGD.

The loss function is given by:

$$J(\hat{W}_t) = - \sum_{i \in S_t^2} y_i \times \log(\hat{y}_i)$$

The parameters are updated as follows:

$$\hat{W}_{t+1} = \hat{W}_t - \eta \nabla J(\hat{W}_t)$$
MULTINOMIAL LOGISTIC REGRESSION ON MNIST: FEED DATA

Compute cross-entropy:

$$J(\hat{W}_t) = - \sum_{i \in S^2_t} y_i \times \log(\hat{y}_i)$$

\(\hat{W}_t\) updated by SGD:

$$\hat{W}_{t+1} = \hat{W}_t - \eta \nabla J(\hat{W}_t)$$
MULTINOMIAL LOGISTIC REGRESSION ON MNIST: FEED DATA

Compute cross-entropy:

\[ J(\hat{W}_t) = - \sum_{i \in S^2_t} y_i \times \log(\hat{y}_i) \]

\( \hat{W}_t \) updated by SGD:

\[ \hat{W}_{t+1} = \hat{W}_t - \eta \nabla J(\hat{W}_t) \]
NEURAL NETWORKS

- Minimize your error on a training set
- Find the best inference function parameters
- Difference between neuralNets an deepNets: only in the inference function

\[ \hat{y} = f(\theta, x) \]

\[ J(\theta) = \text{error}(\hat{y}, y) \text{ given } \theta \]

\[ \hat{\theta} = \arg\min \ J(\theta) \]

And train it using mini-batch SGD!
NEURAL NETWORKS IN TENSORFLOW: GENERAL GRAPH

\[ \hat{y} = f(\theta, x) \]

\[ J(\theta) = \text{error}(\hat{y}, y) \text{ given } \theta \]

\[ \hat{\theta} = \arg\min J(\theta) \]

And train it using mini-batch SGD in a Tensorflow session!
And train it using mini-batch SGD in a Tensorflow session!
And train it using mini-batch SGD in a Tensorflow session!
And train it using mini-batch SGD in a Tensorflow session!
BACKPROPAGATION

- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters

[Diagram showing the process of backpropagation]

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#l8cz02hlu
**BACKPROPAGATION**

- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters

![Diagram](https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu)
• **Forward Activation**: Predict the output
• **Compute the loss**
• **Backward Error**: And correct the parameters
BACKPROPAGATION

• **Forward Activation**: Predict the output
• **Compute the loss**
• **Backward Error**: And correct the parameters

\[
X \xrightarrow{f_\theta} \hat{y} \xrightarrow{\text{backward}} y
\]

backpropagation of the error over the network using derivative function

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu
BACKPROPAGATION

- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters

X \xrightarrow{f_{\theta}} \hat{y} \xleftarrow{error} y

backpropagation of the error over the network using derivative function

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu
EXERCISES

• Go here: https://github.com/fabienbaradel/Tensorflow-tutorials
• And do the softmax and multilayer perceptron exercises
CONVOLUTIONAL NETWORKS
« Convolutional neural networks »

- Created by Yann LeCun (90’s)
- Well-know since 2000
- Big acceleration with GPUs
- Computer vision
- NLP
- Artificial Intelligence
- Convolution & Pooling

ConvNets usually evaluated on ImageNet (5 millions images, 1000 classes)
CONVOLUTION
CONVOLUTION

• Finding information in subpart of the image
• Local spatial correlation
• Mimic the biological process
• Less parameters than fully-connected layer

Example: convolution on 5x5 matrix (1 filter=3x3 et stride=1)

Input: (5,5,1)

Ouput: (nb_filter,3,3)

http://dl.heeere.com/convolution3/
**POOLING**

- Sampling over a matrix
- Dimension reduction
- Reduce number of parameters of further layers
- No learnable parameters!

Example: pooling over a 20x20 matrix (filter=10x10 et stride=10)
CONVNETS

AlexNet (2012)

GoogleNet Inception v3 (2015)

80.1 %
93.4 %
CONVNETS

Image Model Training Time

Precision @ 1

- 50 GPUs
- 10 GPUs
- 1 GPU

Hours
EXERCISES

• Complete the exercises:
  ✴ One Conv + Max Pool
  ✴ LeNet

HAVE A LOOK TO TF.SLIM TO MAKE YOUR LIFE EASIER
WHY CONVOLUTION WORKS?
FEATURE MAPS

Layer 1: ~ Gabor filters
FEATURE MAPS

Layer 2
FEATURE MAPS

Layer 5

Convolutional layer
Local Response Normalization
Max Pooling layer
Fully Connected layer
FILTERS

https://www.youtube.com/watch?v=AgkflQ4IGaM
• Filters after first convolutional layer are generic (Gabor filters)
• Deeper you go in network and more task specific are your filters
FINE-TUNING

pretrained Inception v3 on ImageNet

\[ v(\text{image}) = \begin{pmatrix} v_1 \\ \vdots \\ v_{2048} \end{pmatrix} \]
• Wanna win $150’000? YES YOU CAN!
• Go to the Github repo and do the « Classification from DeepFeatures » exercise
• And submit your .csv in Kaggle (and cross your fingers)

WHAT IS YOUR SCORE?

... 

Want to add convolution?
Reshape your vector to a 3D matrix...
2048 = 32*32*3
AUTOENCODER
NEURAL NETWORK LEARNING

Supervised learning
- $y$ are given!

Unsupervised learning
- $y$ is no longer needed
AUTOENCODER

- Learning a compact data representation
- Encode input to smaller latent space
- Decode from the latent space to the input
- Predict input from input
- Loss function = mean square error
- $f$ and $g$ are neural networks
- SGD as usual
f and g are linear without hidden layer

=> your solution is an approximation of a PCA
GENERATIVE MODELS
GENERATIVE MOMENT MATCHING NETWORKS

\[ X \xrightarrow{f} Z \xrightarrow{g} X' \]
- \( X \): Input
- \( f \): Encoder
- \( Z \): Latent space
- \( g \): Decoder
- \( X' \): Output

\( \text{RMSE} \)
GENERATIVE MOMENT MATCHING NETWORKS

\[ Z \] is the generated latent variable, and \( q \) maps it to \( N(0,1) \), the standard normal distribution.
GENERATIVE MOMENT MATCHING NETWORKS

X \rightarrow f \rightarrow Z

Z \rightarrow \text{encoder} \rightarrow q \rightarrow N(0,1)

\text{latent} \quad \text{predicted latent}

MMD
GENERATIVE MOMENT MATCHING NETWORKS

\[ X \xrightarrow{f} Z \sim \sim \xrightarrow{g} X_{\text{GENERATED}} \]

encoder

latent

\[ f \]

\[ Z \sim \sim \]

decoder

\[ q \]

\[ N(0,1) \]
GENERATIVE ADVERSIAL NETWORKS

Intuition

Generator → fake money
GENERATIVE ADVERSIAL NETWORKS

Intuition

Generator → fake money → Discriminator

real money → Discriminator

FAKE OR REAL?

FAKE OR REAL?
GENERATIVE ADVERSIAL NETWORKS

- G and D are neural networks
- Find a G that minimizes the accuracy of the best D
- Alternate optimization of G and D

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z)))].
\]

log prob of D predicting that real-world data is genuine

log prob of D predicting that G's generated data is not genuine

GAN: EXAMPLES

- DCGAN (Deep Convolutional GAN), Radford et al., 2015/2016
GAN: EXAMPLES

Noise $\sim N(0,1)$

Generative Model
GAN: EXAMPLES

Ongoing topic…
• Complete the exercises:
  ✴ « Autoencoder_exo »
  ✴ « Conv-Deconv Autoencoder_exo »
  ✴ And GMMN if you are fast enough!
SEQUENCE MODELING
WHAT ABOUT SEQUENCE?

Image = Static (almost) Solved

Video = Sequence of images not solved at all…
WHAT ABOUT SEQUENCE?

Sequence to sequence:

Machine Translation
Imagine \( X \) as a time series: \((x_1, x_2, \ldots, x_n)\)

- \( h \) is the hidden state of the RNN
- Initialized at \((1,1,\ldots,1)\) at \( t=0 \)
- And \( h \) is modified after each timestep

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
RNN AND CLASSIFICATION

H_0

Initialize randomly

RNN
RNN AND CLASSIFICATION

X_1 → RNN → H_1 → H_0
RNN AND CLASSIFICATION

\[ X_1 \quad \rightarrow \quad \text{RNN} \quad \rightarrow \quad H_2 \]

\[ X_2 \quad \rightarrow \quad \text{RNN} \quad \rightarrow \quad H_1 \quad \rightarrow \quad H_0 \]
RNN AND CLASSIFICATION

RNN

X_1

RNN

H_0

H_1

H_N-1

H_N

X_n
RNN AND CLASSIFICATION

RNN

X_1

H_0

X_n

H_1

H_N-1

RNN

H_N

classif

Y
WORD2VEC

How to represent a word as a vector?

TF-IDF?

=> Learning word embedding

Italy = (5.12, 7.21, ..., 0.78) ∈ ℝ¹⁰⁰

Beautiful word2vec relationships:

king − man + woman = queen
Tokyo − Japan + France = Paris
best − good + strong = strongest

And of course some mistakes:

England − London + Baghdad = ?
WORD2VEC

How to represent a word as a vector?

TF-IDF?

=> Learning word embedding

Italy = (5.12, 7.21, ..., 0.78) ∈ ℝ^{100}

Beautiful word2vec relationships:

king – man + woman = queen
Tokyo – Japan + France = Paris
best – good + strong = strongest

And of course some mistakes:

England – London + Baghdad = Mosul?
WORD2VEC

How to represent a word as a vector?

TF-IDF?

=> Learning word embedding

\[ \text{Italy} = (5.12, 7.21, \ldots, 0.78) \in \mathbb{R}^{100} \]

Beautiful word2vec relationships:

- king – man + woman = queen
- Tokyo – Japan + France = Paris
- best – good + strong = strongest

And of course some mistakes:

- England – London + Baghdad = Mosul, Iraq
RNN ON MNIST

4 =
RNN ON MNIST

=
RNN ON MNIST

\[ 4 = 4 \]
RNN ON MNIST

4 = 7
RNN ON MNIST

[28,28] = sequence of 28 vectors of size 28
RNN ON MNIST

• Complete the exo «.rnn_exo »
MACHINE LEARNING

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
REINFORCEMENT LEARNING

RL = Reinforcement Learning
RL: FEW EXAMPLES

https://www.youtube.com/watch?v=V1eYniJ0Rnk
RL IN A FINITE STATE SPACE

\[
\begin{align*}
\text{state} & \quad s_t \\
\text{reward} & \quad r_t \\
\text{action} & \quad a_t \\
\text{Environment} & \quad \xrightarrow{a_t} s_{t+1} \\
\end{align*}
\]

\[s_t, s_{t+1} \in S, \quad a_t \in A(s_t), \quad t = 0, 1, 2, \ldots\]
FROZEN LAKE EXAMPLE

\[
\begin{bmatrix}
S & F & F & F \\
F & H & F & H \\
F & F & F & H \\
H & F & F & G \\
\end{bmatrix}
\]

- \(F\) = frozen surface, safe
- \(G\) = goal, where the frisbee is located
- \(S\) = starting point, safe
- \(H\) = hole, fall to your doom

- **Possible actions:**
  - Up
  - Down
  - Left
  - Right

Ice is slippery: you won’t always move in the direction you intend
The agent learns to assign values to state-action pairs

**Discounted return:**

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-1} r_T \]

where \( \gamma \in [0; 1] \) is the discount rate

**Action - value function for policy \( \pi \):**

\[ Q^\pi(s, a) = E_\pi \{ R_t | s_t = s, a_t = a \} \]

« How good an action is for the future given a certain state? »
FORMULATION OF THE Q-FUNCTION

\[ Q^\pi(s_t, a_t) = \begin{bmatrix} Q(0, \text{Up}) & Q(0, \text{Down}) & Q(0, \text{Left}) & Q(0, \text{Right}) \\ \vdots & \vdots & \vdots & \vdots \\ Q(s_t, \text{Up}) & Q(s_t, \text{Down}) & Q(s_t, \text{Left}) & Q(s_t, \text{Right}) \\ \vdots & \vdots & \vdots & \vdots \\ Q(16, \text{Up}) & Q(16, \text{Down}) & Q(16, \text{Left}) & Q(16, \text{Right}) \end{bmatrix} \]

Optimal value function unrolled recursively

\[ Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}} \left\{ r_{t+1} + \gamma \times \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right\} \]

Express state-value function by neural network with parameters \( \theta \):

\[ Q(s, a, \theta) \approx Q^\pi(s, a) \]

But what is our target vector to compute the loss function???
DEEP Q-LEARNING

Q value function

\[ Q_{\theta_t} = \begin{bmatrix} Q(0, \text{Up}, \theta_t) & Q(0, \text{Down}, \theta_t) & Q(0, \text{Left}, \theta_t) & Q(0, \text{Right}, \theta_t) \\ Q(1, \text{Up}, \theta_t) & Q(1, \text{Down}, \theta_t) & Q(1, \text{Left}, \theta_t) & Q(1, \text{Right}, \theta_t) \\ \vdots & \vdots & \vdots & \vdots \\ Q(15, \text{Up}, \theta_t) & Q(15, \text{Down}, \theta_t) & Q(15, \text{Left}, \theta_t) & Q(15, \text{Right}, \theta_t) \end{bmatrix} \]

Loss function

\[ J(\theta_t) = \sum (Q(s_t, a_t, \theta_t) - r_{t+1} + \gamma \times \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta_t))^2 \]

Train it using SGD!
NN Q-FUNCTION FOR FROZEN LAKE

state from 0 to 15

action
DEEP Q-FUNCTION FOR REAL GAMES

ConvNets ... here we go again!
• Complete the exo « q_learning_frozen_lake_exo »

Or go back to the fish classification if you want ;)}
WHAT ABOUT YOUR FIRST EXPERIENCE WITH TENSORFLOW?