Deep learning for natural language processing
A short primer on deep learning

Benoit Favre <benoit.favre@univ-mrs.fr>

Aix-Marseille Université, LIF/CNRS

20 Feb 2017

Deep learning for Natural Language Processing

- Day 1
  - Class: intro to natural language processing
  - Class: quick primer on deep learning
  - Tutorial: neural networks with Keras
- Day 2
  - Class: word embeddings
  - Tutorial: word embeddings
- Day 3
  - Class: convolutional neural networks, recurrent neural networks
  - Tutorial: sentiment analysis
- Day 4
  - Class: advanced neural network architectures
  - Tutorial: language modeling
- Day 5
  - Tutorial: Image and text representations
  - Test

Mathematical notations

Just to be make sure we share the same vocabulary

- x can be a scalar, vector, matrix or tensor (n-dimensional array)
  - An "axis" of x is one of the dimensions of x
  - The "shape" of x is the size of the axes of x
  - $x_{i,j,k}$ is the element of index i, j, k in the 3 first dimensions
- $f(x)$ is a function on x, it returns a same-shape mathematical object
- $xy = x \cdot y = dot(x, y)$ is the matrix-to-matrix multiplication
  - if $r = xy$, then $r_{i,j} = \sum_k x_{i,k} \times y_{k,j}$
- $x \odot y$ is the elementwise multiplication
- $tanh(x)$ applies the $tanh$ function to all elements of x and returns the result
- σ is the sigmoid function, $|x|$ is the absolute value, $max(x)$ is the largest element...
- $\sum x$ is the sum of elements in x, $\prod x$ is the product of elements in x
- $\frac{\partial f}{\partial \theta}$ is the partial derivative of f with respect to parameter θ

What is machine learning?

- Objective
  - Train a computer to simulate what humans do
  - Give examples to a computer and teach it to do the same
- Actual way of doing machine learning
  - Adjust parameters of a function so that it generates an output that looks like some data
  - Minimize a loss function between the output of the function and some true data
  - Actual minimization target: perform well on new data (empirical risk)

A formalization

- Formalism
  - $x \in \mathbb{R}^k$ is an observation, a vector of real numbers
  - $y \in \mathbb{R}^m$ is a class label among m possible labels
  - $X, Y = \left\{(x^{(i)}, y^{(i)})\right\}_{i \in [1..n]}$ is training data
  - $f_\theta(\cdot)$ is a function parametrized by θ
  - $L(\cdot, \cdot)$ is a loss function
- Inference
  - Predict a label by passing the observation through a neural network

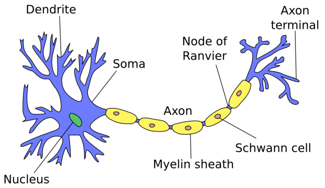
$$y = f_\theta(x)$$

- Training
  - Find the parameter vector that minimizes the loss of predictions versus truth on a training corpus

$$\theta^\star = \operatorname*{argmin}_\theta \sum_{(x,y) \in T} L(f_\theta(x), y)$$

Neural networks

- A biological neuron
  - Inputs: dendrite
  - Output: axon
  - Processing unit: nucleus



Source: http://www.marekrei.com/blog/wp-content/uploads/2014/01/neuron.png

- One formal neuron
  - $output = \text{activation}(\text{weighted sum}(inputs) + \text{bias})$
- A layer of neurons
  - f is an activation function
  - Process multiple neurons in parallel
  - Implement as matrix-vector multiplication

$$y = f(Wx + b)$$

- A multilayer perceptron

$$y = f_3(W_3 f_2(W_2 f_1(W_1 x + b_1) + b_2) + b_3)$$
$$y = NN_\theta(x), qquad\theta = (W_1, b_1, W_2, b_2, W_3, b_3)$$

Encoding inputs and outputs

- Input x
  - Vector of real values
- Output y
  - Binary problem: 1 value, can be 0 or 1 (or -1 and 1 depending on activation function)
  - Regression problem: 1 real value
  - Multiclass problem
    - One-hot encoding
    - Example: class 3 among $6 \rightarrow (0, 0, 1, 0, 0, 0)$

Non linearity

- Activation function
  - If f is identity, composition of linear applications is still linear
  - Need non linearity ($tanh$, σ, ...)
  - For instance, 1 hidden-layer MLP

$$NN_\theta(x) = \sigma(W_2 z(x) + b_2)$$
$$z(x) = \sigma(W_1 x + b_1)$$

- Non linearity
  - Neural network can approximate any[1] continuous function [Cybenko'89, Hornik'91, ...]
- Deep neural networks
  - A composition of many non-linear functions
  - Faster to compute and better expressive power than very large shallow network
  - Used to be hard to train

[1] http://neuralnetworksanddeeplearning.com/chap4.html

Loss

- Loss suffered by wrongfully predicting the class of an example

$$L(X,Y) = \frac{1}{n}\sum_{i=1}^{n} l(y^{(i)}, NN_\theta(x))$$

- Well-known losses
  - y_t is the true label, y_p is the predicted label

$$\begin{aligned}
l_{\text{mae}}(y_t, y_p) &= |y_t - y_p| && \text{absolute loss} \\
l_{\text{mse}}(y_t, y_p) &= (y_t - y_p)^2 && \text{mean square error} \\
l_{\text{ce}}(y_t, y_p) &= y_t \text{ln} y_p + (1 - y_t)\text{ln}(1 - y_p) && \text{cross entropy} \\
l_{\text{hinge}}(y_t, y_p) &= \max(0, 1 - y_t y_p) && \text{hinge loss}
\end{aligned}$$

- The most common loss for classification
  - Cross entropy

Training as loss minimization

- As a loss minimization problem

$$\theta^{\times} = \operatorname*{argmin}_{\theta} L(X, Y)$$

- So 1-hidden layer MLP with cross entropy loss

$$\theta^{\times} = \operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} y_t \mathrm{ln} y_p + (1 - y_t) \mathrm{ln}(1 - y_p)$$

$$y_p =$$

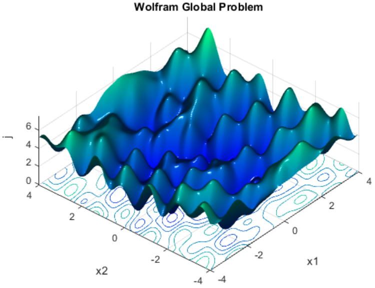
- We have a multilayer perceptron with two hidden layers

$$y_p = NN_{\theta}(x) = \sigma(W_2 z(x) + b_2)$$
$$z(x) = \sigma(W_1 x + b_1)$$

- $\rightarrow$ Need to minimize a non linear, non convex function

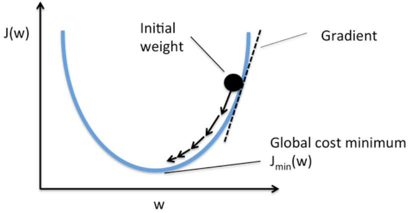
Function minimization

- Non convext $\rightarrow$ local minima



Source: https://www.inverseproblem.co.nz/OPTI/Images/plot_ex2nlpb.png

- Gradient descent



Source: https://qph.ec.quoracdn.net/main-qimg-1ec77cdbb354c3b9d439fbe436dc5d4f

Gradient descent

- Start with random θ
- Compute gradient of loss with respect to θ

$$\nabla L(Y, X) = \left(\frac{\partial L(X,Y)}{\partial \theta_1}, \dots \frac{\partial L(X,Y)}{\partial \theta_n} \right)$$

- Make a step towards the direction of the gradient

$$\theta^{(t+1)} = \theta^{(t)} + \lambda \nabla L(X, Y)$$

- λ is a small value called *learning rate*

Chain rule

- Differentiation of function composition
  - Remember calculus class

$$g \circ f(x) = g(f(x))$$
$$\frac{\partial(g \circ f)}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$$

- So if you have function compositions, you can compute their derivative with respect to a parameter by multiplying a series of factors

$$\frac{\partial(f_1 \circ \dots \circ f_n)}{\partial \theta} = \frac{\partial f_1}{\partial f_2} \dots \frac{\partial f_{n-1}}{\partial f_n}\frac{\partial f_n}{\partial \theta}$$

Example for MLP

- Multilayer perceptron with one hidden layer (z_2)

$$
\begin{aligned}
L(X,Y) &= \frac{1}{n}\sum_{i=1}^{n} l_{\text{ce}}(y^{(i)}, NN_\theta(x^{(i)})) \\
NN_\theta(x) &= z_1(x) = \sigma(W_2 z_2(x) + b_2) \\
z_2(x) &= \sigma(W_1 x + b_1) \\
\theta &= (W_1, b_1, W_2, b_2)
\end{aligned}
$$

- So we need to compute

$$
\begin{aligned}
\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial l_{\text{ce}}} \frac{\partial l_{\text{ce}}}{\partial z_1} \frac{\partial z_1}{\partial W_2} \\
\frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial l_{\text{ce}}} \frac{\partial l_{\text{ce}}}{\partial z_1} \frac{\partial z_1}{\partial b_2} \\
\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial l_{\text{ce}}} \frac{\partial l_{\text{ce}}}{\partial z_1} \frac{\partial z_1}{\partial z_2} \frac{\partial z_2}{\partial W_1} \\
\frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial l_{\text{ce}}} \frac{\partial l_{\text{ce}}}{\partial z_1} \frac{\partial z_1}{\partial z_2} \frac{\partial z_2}{\partial b_1}
\end{aligned}
$$

- A lot of the computation is redundant

Back propagation

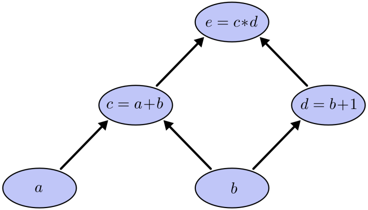
- A lot of computations are shared
  - No need to recompute them
  - Similar to dynamic programming
- Information propagates back through the network
  - We call it "back-propagation"

Training a neural network

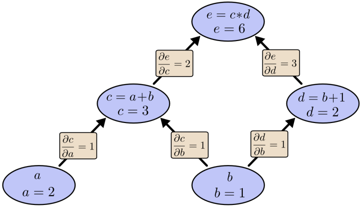
1. $\theta_0 = \text{random}$
2. while not converged
   1. forward: $L_{\theta_t}(X, Y)$
      - Predict y_p
      - Compute loss
   2. backward: $\nabla L_{\theta_t}(X, Y)$
      - Compute partial derivatives
   3. update $\theta_{t+1} = \theta_t + \lambda \nabla L_{\theta_t}(X, Y)$

Computational Graphs

- Represent operations in $L(X, Y)$ as a graph
  - Every operation, not just high-level functions



Source: http://colah.github.io

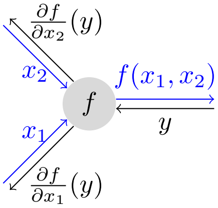


- More details: `http://outlace.com/Computational-Graph/`

Building blocks for neural networks

- Can build a neural network like lego
  - Each block has inputs, parameters and outputs
  - Examples
    - Logarithm: forward: $y = ln(x)$, backward: $\frac{\partial ln}{\partial x}(y) = 1/y$
    - Linear: forward: $y = f_{W,b}(x) = W \cdot x + b$
      backward: $\frac{\partial f}{\partial x}(y) = y^T \cdot x$, $\frac{\partial f}{\partial W}(y) = y \cdot W$, $\frac{\partial f}{\partial b}(y) = y$
    - Sum, product: ...

- Provides auto-differentiation
  - A key component of modern deep learning toolkits



Stochastic optimization

- Stochastic gradient descent (SGD)
  - Look at one example at a time
  - Update parameters every time
  - Learning rate λ
- Many optimization techniques have been proposed
  - Sometimes we should make larger steps: adaptive λ
    - $\lambda \leftarrow \lambda/2$ when loss stops decreasing on validation set
  - Add inertia to skip through local minima
  - Adagrad, Adadelta, Adam, NAdam, RMSprop...
  - The key is that fancier algorithms use more memory
    - But they can converge faster
- Regularization
  - Prevent model from fitting too well to the data
  - Penalize loss by magnitude of parameter vector ($loss + ||\theta||$)
  - Dropout: randomly disable
  - Mini-batches
    - Averages SGD updates over a set of examples
    - Much faster because computations are parallel

Deep learning toolkits

- Low level toolkits
  - Tensorflow: `https://www.tensorflow.org`
  - Theano: `http://deeplearning.net/software/theano`
  - Torch: `http://torch.ch`
  - mxnet: `http://mxnet.io`
- High level frameworks
  - Keras: `http://keras.io`
  - Tflearn: `http://tflearn.org`
  - Lasagne: `https://lasagne.readthedocs.io`
- Some can do both
  - Chainer: `http://chainer.org`
  - Pytorch: `http://pytorch.org`

What they provide

- Low level toolkits
  - Can "implement paper from the equations"
  - Static or dynamic computation graph compilation and optimization
  - Hardware acceleration (CUDA, BLAS...)
  - But lots of house keeping
- High level frameworks
  - Generally built on top of low level toolkits
  - Implementation of most basic layers, losses, etc.
  - Your favourite model in 10 lines
  - Data processing pipeline
  - Harder to customize
- At some point, you will need to jump from high-level to low-level

Comparison

Preferred Networks

Framework Comparison: Basic information*

Viewpoint	Torch.nn**	Theano***	Caffe	autograd (NumPy, Torch)	Chainer	MXNet	Tensor-Flow
GitHub stars	4,719	3,457	9,590	N: 654 T: 554	1,295	3,316	20,981
Started from	2002	2008	2013	2015	2015	2015	2015
Open issues/PRs	97/26	525/105	407/204	N: 9/0 T: 3/1	95/25	271/18	330/33
Main developers	Facebook, Twitter, Google, etc.	Université de Montréal	BVLC (U.C. Berkeley)	N: HIPS (Harvard Univ.) T: Twitter	Preferred Networks	DMLC	Google
Core languages	C/Lua	C/Python	C++	Python/Lua	Python	C++	C++/Python
Supported languages	Lua	Python	C++/Python MATLAB	Python/Lua	Python	C++/Python R/Julia/Go etc.	C++/Python

* Data was taken on Apr. 12, 2016
** Includes statistics of Torch7
*** There are many frameworks on top of Theano, though we omit them due to the space constraints

PAKDD2016 DLIF Tutorial 20

Graphical Processing Units

- Most toolkits can take advantage of hardware acceleration
  - Graphical Processing Units
    - GPGPU $\rightarrow$ accelerate matrix product
    - Take advantage of highly parallel operations
  - x10-x100 acceleration
    - Things that would take weeks to compute, can be done in days
    - The limiting factor is often data transfer from and to GPU
- NVIDIA
  - Currently the best (only?) option
  - High-end gamer cards: cheaper but limited
    - Gforce GTX 1080 ($800)
    - Titan X ($1,200)
  - Professional cards
    - Can run 24/7 for years, passive cooling
    - K40/K80: previous generation cards ($3.5k)
    - P100: current generation ($6k)
    - DGX-1: datacenter with 8 P100 ($129k)
  - Renting: best way to scale
    - Amazon AWS EC2 P2 ($1-$15 per hour)

Information sources

- The Deep learning landscape is moving fast
  - Conferences: NIPS, ICML,ICLR...
  - Need to read scientific papers from arxiv
  - Plenty of reading lists on the web
    - https://github.com/ChristosChristofidis/awesome-deep-learning
    - https://github.com/kjw0612/awesome-rnn
    - https://github.com/kjw0612/awesome-deep-vision
    - https://github.com/keon/awesome-nlp
- Where to get news from
  - Twitter http://twitter.com/DL_ML_Loop/lists/deep-learning-loop
  - Reddit https://www.reddit.com/r/MachineLearning/
  - HackerNews http://www.datatau.com/

Keras: short presentation

- Keras is an abstraction over Theano and Tensorflow
  - Advice: follow the tutorial at `https://keras.io/`

```
from keras.models import Sequential
from keras.layers import Dense, Activation

# build and compile the model
model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# assumes you have loaded data in X_train and Y_train
model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

# get the classes predicted by the model
proba = model.predict_classes(X_test, batch_size=32)
```

Conclusion

- Deep learning is loosely modeled after the brain
  - Neural network is a parametrisable function composition
  - Learns a non-linear function of its input
  - Back-propagation of the error
    - Chain rule
    - Computation graph
  - Loss minimization
- Many toolkits available today
  - High-level programming language
  - Automatic differentiation
  - Accelerated with GPU