INTRODUCTION TO DEEP LEARNING

Alessandro Rigazzi, PhD
Cray EMEA Research Lab / AI Team

arigazzi@cray.com

Introduction

© 2018 Cray Inc.

Course overview

- **Tuesday**
  - Introduction to AI and DL
    - What is (and what is not) AI?
    - A bit of history
    - Deep Learning ingredients
    - Supervised Learning 101
  - Toolkits, frameworks, SDKs
  - Hands-on sessions
    - Keras
    - Multi-layer perceptrons
    - Convolutional Neural Networks

- **Bonus content**
  - Cray in Deep Learning
    - Cray Graph Engine
    - CosmoFlow
  - Success stories
    - Training a question answering Neural Network

© 2018 Cray Inc.

Course overview

- **Wednesday (subject to change)**
  - Other types of networks
  - Benchmarks, performances, figures of merit
  - Parallelism
  - Ethics, accountability, explainability
- **Bonus content**
  - Cray in Deep Learning
    - Cray Graph Engine
    - CosmoFlow
  - Success stories
    - Training a question answering Neural Network

© 2018 Cray Inc.

Me, myself & Cray

© 2018 Cray Inc.

Who I am

- My name is Alessandro
  - But let's face it, it is a very long and misspelling-prone name
    - That's why I go by the name of **AL**
- Based in Lugano, Switzerland
  - 5 minute bus-ride from CSCS
  - 1 minute walk from where you had the social dinner
- Computational Scientist turned to Deep Learning and AI
  - Interest in large-scale training, workflow optimization, containers
  - But still interested in solvers, scientific visualization, HPC, MPI, …

© 2018 Cray Inc.

What is Cray?

- Cray – the Supercomputing company
- Cray research was founded in 1972 by Seymour Cray
- First supercomputer was presented to the public in 1976
- More than 1300 employees worldwide
- Specialized in large, fast, reliable supercomputers
  - Like Piz Daint!

© 2018 Cray Inc.

WHAT IS (AND WHAT IS NOT) AI?

AI – a simple definition from Wikipedia

- Artificial intelligence (AI), sometimes called machine intelligence, is **intelligence demonstrated by machines**, in contrast to the natural intelligence displayed by humans and other animals. In computer science AI research is defined as the **study of "intelligent agents"**: **any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals**.
- Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".
- The scope of AI is disputed: as machines become increasingly capable, **tasks considered as requiring "intelligence" are often removed from the definition** […].
- Modern machine capabilities generally classified as AI include successfully understanding human speech, competing at the highest level in strategic game systems (such as chess and Go), autonomously operating cars, and intelligent routing in content delivery networks and military simulations.

© 2018 Cray Inc.

Analytics vs Machine Learning vs Data Mining

- **Analytics**
  - Data analysis refers to reviewing data from past events for patterns.
  - Predictive analytics is making assumptions and testing based on past data to predict future what-ifs.

- **Machine Learning**
  - AI ML analyzes data, makes assumptions, learns and provides predictions at a scale and depth of detail impossible for individual human analysts.

- **Data Mining**
  - Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on known properties learned from the training data, data mining focuses on the discovery of (previously) unknown properties in the data (this is the analysis step of knowledge discovery in databases).

Sources: https://www.forbes.com/ and Wikipedia

© 2018 Cray Inc.

So… what is ML?

- A collection of techniques, algorithms, and methods which can extrapolate information from data
  - The output must be the consequence of some **learnable parameters** for which the model has been **trained**

- Some classes
  - Supervised and semi-supervised learning
  - Unsupervised learning
  - Reinforcement learning

- Some techniques
  - Support Vector Machines
  - Decision Trees, Random Forests
  - Artificial Neural Networks
  - …

© 2018 Cray Inc.

What is **not** AI?

- Programs which compute a result from a non-learnable set of instructions
  - Computational Fluid Dynamics
  - Continuum Mechanics
  - Classic Numerical Weather Prediction
  - …

- What is in the gray area?
  - Linear or polynomial regression
    - Nobody would have called it AI some years ago
    - But it learns some parameters and can label a sample which it has not seen before… is this not intelligence?

© 2018 Cray Inc.

AI TAXONOMY

© 2018 Cray Inc.

Two families

- Supervised Learning
  - The AI agent learns starting from what we teach it
  - Examples
    - Reward
      - How did the agent perform in a maze?
    - Labeled data
      - Is there a dog in this picture?
- **This is what we will focus on!**

- Unsupervised Learning
  - The AI agent learns by itself
  - Examples
    - Data Clustering
      - How many classes are needed to cluster the data?
    - Autoencoding
      - Learn a model which can be used to represent the data with lower dimensionality
- Sometimes used before supervision

© 2018 Cray Inc.

Many algorithms

- Support Vector Machines
- Radial Basis Functions
- Bayesian Learning
- Mixture Models
- Reinforcement Learning
- Restricted Boltzmann Machines
- Deep Belief Networks
- Self-Organizing Maps
- Random Forests
- **Artificial Neural Networks**
- Recommender systems
- …

© 2018 Cray Inc.

A bit of history

© 2018 Cray Inc.

ML Timeline

Source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

© 2018 Cray Inc.

A neuron

https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html

© 2018 Cray Inc.

Spike Response Model

https://icwww.epfl.ch/~gerstner/SPNM/node27.html

© 2018 Cray Inc.

The perceptron

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

© 2018 Cray Inc.

The perceptron

- Input: vector $\mathbf{x} = \{x_\mathrm{i}\}$
- Parameters: weights $\mathbf{w} = \{w_\mathrm{i}\}$
- Compute output: $o = \phi(w_0 + \sum_{i=1}^{n} x_i w_i)$
  - Where $\phi(y) = \begin{cases} 1, & if\ y > threshold \\ 0, & otherwise \end{cases}$
- Thus, the output is a binary classification of the input
- To train it, we feed a set of labeled inputs to it
  - A labeled input is an input for which we already know the correct category (0 or 1)
  - This is also called desired output d

© 2018 Cray Inc.

Training a perceptron

1. Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value. In the example below, we use 0.
2. For each example j in our training set D, perform the following steps over the input $\mathbf{x}_j$ and desired output d_j:

   a. Calculate the actual output:

   $$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j]$$
   $$= f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$$

   b. Update the weights:

   $w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}$, for all features $0 \leq i \leq n$, r is the learning rate.

© 2018 Cray Inc.

Training a perceptron – visualization

© 2018 Cray Inc.

A perceptron's failure: the XOR function (1969)

- A perceptron cannot be trained to correctly solve the XOR function
- **And it cannot be trained to solve non-linearly separable problems**

- **Welcome to the first AI winter**

© 2018 Cray Inc.

A perceptron cannot choose the best solution

© 2018 Cray Inc.

ML Timeline

Source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

© 2018 Cray Inc.

Hidden layer and the Multi-Layered Perceptron

- Stacking more than one perceptron we can approximate any function
- The step function can be replace by different non-linear functions, called **activation functions**
  - For many years scientists only used saturating functions mapping input to [-1, 1] or [0, 1]
  - Now many more possibilities

© 2018 Cray Inc.

Minimization problem

- Typical in many fields
  - Minimization of energy to solve differential equations
  - Maximization of profit
  - Travel time inverse problem in geophysics
- In Machine Learning
  - Minimize the error by following negative gradient
  - Intuitively: step-wise steepest descent

© 2018 Cray Inc.

Minimization problem – a more realistic view

- Often difficult to find global minimum
  - Optimizers can get stuck in local minima
  - Saddle points introduce instability

- In Machine Learning
  - Many techniques have been developed to make the error surface (or loss landscape) smooth and regular and to avoid local minima

© 2018 Cray Inc.

Minimization problem – the truth

Li et al. (2018): Visualizing the Loss Landscape of Neural Nets

© 2018 Cray Inc.

Back-propagation of the error

- The perceptron update function only gives us a way of modifying the output layer of a network
  - Intuitively it gives us the gradient for the current set of weights
  - Good for perceptrons, because they only have one layer
  - What about MLPs?

- Geoffrey Hinton in 1986 publishes a paper about how to “back-propagate” the error
  - Key fact: a stack of perceptrons is nothing else than a composition of functions
    - We know how to find the gradient for composite functions, i.e. we know the **chain rule**

© 2018 Cray Inc.

Feedforward, backprop, and steepest descent

- Feedforward: we classify a sample $\mathbf{x}$ using our weight matrix $\mathbf{W} = \{w_\mathrm{i}\}$
- We compute the **loss of the prediction** $\mathbf{L}(\mathbf{x}, \mathbf{W})$
- We compute the **gradient of the error function w.r.t. all weights** $\frac{\partial \mathbf{L}}{\partial w_\mathrm{i}}$ for that sample, by applying the chain rule
- We move every weight by a small step along the opposite direction

© 2018 Cray Inc.

Backprop in a nutshell

https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

© 2018 Cray Inc.

Convolutional Neural Networks appear

- LeCun 1998

- Weights define sliding filter which is applied on the image
  - Like a convolution, but you switch some signs…

- Two strong features
  - Exploits spatial proximity of pixels
  - Re-uses the same weights in different areas of the image

© 2018 Cray Inc.

Second AI winter

- Convolutional Networks are among the most used now
- The most used algorithms to train them are variations of SGD
- So… why was there no NN boom in 1998?
- The answer is: lack of computational power!
  - Backprop seemed to be too demanding
  - Training a network with more than 5 layers was an overkill
- This led to the second AI winter

© 2018 Cray Inc.

ML Timeline

Source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

© 2018 Cray Inc.

And… deep we go!

- The term *Deep Learning* was coined by Geoffrey Hinton
  - He was focusing on Restricted Boltzmann Machines and Deep Belief Nets
  - With such models, one could train small portions of a network, then stack another layer on top of it, re-train, and so on
    - This was less computationally demanding

- The real breakthrough: AlexNet, 2012 [Krizhevsky, Sutskever, Hinton]
  - CNN architecture, winner of the ILSVRC challenge (ImageNet)
    - Classification error of 16%, best error before was 26%
    - Trained on two **GPUs**

© 2018 Cray Inc.

When will the next AI winter be?

- Possibly never

- AI and DL are now much more an applied field than they used to be
  - Research is not (only) theoretical anymore

- People will realize the limitations of DL
  - As it happens for any technology
  - But DL and AI have now many branches, it is hard to think they will all die at the same time

© 2018 Cray Inc.

Deep Learning Building Blocks

© 2018 Cray Inc.

The main ingredients

- Any Machine Learning method needs four parts
  1. The data
     - E.g. sentences in different languages
  2. The model
     - E.g. a neural network
  3. A measure of fitness
     - E.g. an error (or loss) function
  4. An algorithm to learn parameters
     - E.g. gradient descent and back-propagation

© 2018 Cray Inc.

The data

- Data is the key point to any good algorithm
- Data has to be prepared
  - Cleaned
  - Whitened
  - **Labeled**
  - Stored in an efficient way
- Data must be representative of the problem we are trying to solve
- We usually divide the data we have in *training* data, *validation* data, and *test* data
  - We use train data to adjust parameters
  - We use validation data to decide which technique gave the best results
  - Finally, we measure the real accuracy on test data
  - If you only train with one algorithm, you do not need test data

© 2018 Cray Inc.

The data – an example

- Images
  - Usually stored in RGB format
  - Maybe compressed, varying size

- Data preparation
  - Whitening: compute average values, scale
  - Enhance: maybe RGB is not the right space, use PCA
  - Store the data in the right format
    - GPUs like de-interleaved (channels first, or NCWH)
    - CPUs like interleaved (channels last, or NWHC)

© 2018 Cray Inc.

The model

- A model is simply a parametrized function mapping input to output
  - The parameters can be **learned**
- A model should have an optimal set of parameters
  - Those yielding the best results on the test set
- A model can work as a black box (unfortunately)
- The model's complexity should reflect the problem's complexity

© 2018 Cray Inc.

The model – an example

© 2018 Cray Inc.

The measure of fitness

- It must always be possible to assess the model's accuracy or fitness

- The fitness function should in principle be differentiable
  - Otherwise one must find an alternative definition for the parameter gradients

- The minimum (or maximum) of the fitness function should coincide with the optimal value of the parameters
  - But different fitness functions will lead to different parameters!

© 2018 Cray Inc.

The measure of fitness – an example

- Mean Squared Error (MSE):
- We have a set containing data belonging to c classes
- The components of the output vector $\mathbf{o} = o_i$ represent the probability that the input belongs to the ith class
- We can define the mean square error as:

$$MSE(\mathbf{x}) = \frac{1}{2c}\sum_{c}(d_i - o_i)^2$$

- Not really used in practice, but easy to compute gradient:

$$\nabla MSE(\mathbf{x}) = \mathbf{d} - \mathbf{o}$$

- Usually cross-entropy loss is more convenient

© 2018 Cray Inc.

Categorical cross-entropy loss function

- The most used loss function for classifiers
  - Turns NN output into probabilities
  - Computes entropy

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

- In many toolkits, one has to specify that the output layer uses softmax as activation function, and then train the network using CE as loss
  - This will be the case for Keras

from: https://gombru.github.io/2018/05/23/cross_entropy_loss/

© 2018 Cray Inc.

The algorithm

- An algorithm's purpose is to modify the parameters of the model so that they reach the best possible configuration
  - This corresponds to the fitness function's minimum

- An algorithm must be reproducible
  - Even if it can have stochastic parts, those should be controllable
    - E.g. by means of a random number generator with a predefined seed

- The performance of an algorithm is usually measured
  - In terms of time to convergence
  - In terms of absolute accuracy of the prediction

© 2018 Cray Inc.

The algorithm - an example

- Until now, we always talked about feeding the network one sample at the time and then compute the corresponding gradient
  - This is actually called **Stochastic Gradient Descent**
  - It is computationally inefficient, because modern processors (GPUs too) can take advantage of large matrix operations
- One could in principle ”vectorize” all the samples, i.e. stack them as a tensor, and pass them to the network
  - The result would be a stack of gradients, which we would then average to compute the next step to take
  - This would be called **Gradient Descent** or **Batch Gradient Descent**
- In reality this has two drawbacks
  - Often the xPU available memory is not enough
  - It leads to overfitting (more on this later) of the training data

© 2018 Cray Inc.

The algorithm – an example (2)

- Since both SGD and GD are not good choices, we need to find a trade-off
- We group the data in **mini-batches**, which are subsets of the whole training set
- We iterate:
  - Feed the network with one mini-batch
  - Compute the feedforward pass
  - Compute the average gradients
  - Update the weights
- Once we have used all samples, we said that we trained the network for **one epoch**
  - The number of iterations per epoch is the total size of the training set divided by the (mini)batch size
  - We can train for as many epochs as we like, but the network will “stop learning” after a while
- This variant should be called Mini-batch Stochastic Gradient Descent, but…
  - Everybody call this **STOCHASTIC GRADIENT DESCENT**

© 2018 Cray Inc.

NEURAL NETWORKS – A SURVEY

BYON – Build Your Own Network

- One might be tempted to start from scratch, stacking layer after layer
  - Surely not the best way to start in practice
  - Tons of tutorials available, where simple networks can be found
- Two other possibilities:
  - Fine-tune existing networks
    - Often a network trained on a different dataset (pre-trained) can be adapted to specific needs
- Modify existing architectures
  - If the researcher knows something about the used data, or about the behavior of the network for a specific case, some parts can be changed
- For all possible approaches, it is better to know the basic elements of the networks, i.e. the layers

© 2018 Cray Inc.

Most Valuable Layers

© 2018 Cray Inc.

Most Valuable Layers – Fully Connected

- Also known as dense layer
- Simplest form of layer
- Takes an input, in form of a vector, multiplies it with a matrix containing the weights, and applies a non-linear function (*activation function*) to the result
- Useful if there is no spatial structure in the input or in the output
  - E.g. if the output is the predicted label
  - In fact, most object recognition networks employ a FC layer (or a sequence of FC layers) at the end

© 2018 Cray Inc.

Most Valuable Layers – Fully Connected

- What are the disadvantages of Fully Connected Layers?
- They require a large number of parameters
- Remember, the weights are organized as a matrix, and they are multiplied by the output of the previous layer (which are organized as a vector)
- Thus, the total number of weights is:
  - (#inputs+1)*(#outputs)
    - Why ***+1***? This is called bias! We add a component to each vector, and we fix it to 1.
    - That breaks linearity of matrix-vector multiplication, which would lead to indefiniteness of the result!

© 2018 Cray Inc.

Most Valuable Layers – Convolutional

- Possibly the best layer when input structure is important
  - 1D: signals
  - 2D: images
  - 3D: full MRIs
- It applies a set of small filters to the input, using a sliding-window technique
  - Closely related to mathematical convolution, but has some inverted ordering
- Each filter is stacked with the other ones, and the output of the layer is a stack of the resulting convolutions
  - Usually called channels, as RGB
- Convolutional layers can be used transposed to obtain “de-convolution”

© 2018 Cray Inc.

Convolutional Neural Network

Left: A regular 3-layer Neural Network. *Right*: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

http://cs231n.github.io/convolutional-networks/

© 2018 Cray Inc.

Most Valuable Layers – Convolutional example

http://cs231n.github.io/convolutional-networks/

© 2018 Cray Inc.

Most Valuable Layers – Convolutional example

http://cs231n.github.io/convolutional-networks/

© 2018 Cray Inc.

Most Valuable Layers – Convolutional weights

© 2018 Cray Inc.

Most Valuable Layers – Convolutional variants

- A 2D convolutional layer is defined by many parameters
- Volume size as WxHxD (D is depth, number of input channels)
  - Common: 3x3xD, 5x5xD
  - Less common, but in use: 7x7xD, 11x11xD
  - Interesting: 1x1xD
- Stride (distance between two applications of the filter)
  - Common: 1, 2
  - Less common: 4…
- Padding (how many zeroes to add at the boundary)
  - Common: enough to obtain the desired output size…
- Dilation
  - Interval between two different weights of the filter
  - Common: 0, but 1 can be used to substitute pooling (see next slide)

© 2018 Cray Inc.

A quick formula you might need to use…

- When applying convolutions, it might be sometimes tricky to understand what the size of the output will be
- Let's look at what happens for a 1D input (all dimensions work the same way)
- We can compute the spatial size of the output volume as a function of
  - The input volume size (W)
  - The receptive field size of the Conv Layer neurons (F)
  - The stride with which they are applied (S)
  - The amount of zero padding used (P) on the border
- You can convince yourself that the correct formula for calculating how many neurons "fit" is given by $\mathbf{(W-F+2P)/S+1}$.
- For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output. With stride 2 we would get a 3x3 output.

http://cs231n.github.io/convolutional-networks/

© 2018 Cray Inc.

Most Valuable Layers – Pooling

- When we want to reduce the intermediate size of the data, we add a pooling layer
- Can be seen as a sort of convolutional layer
  - But can use non-linear functions of the input, e.g. max
- Typically two types:
  - max-pool
    - Useful to eliminate parts of a channel which are not active (noise reduction)
  - avg-pool
    - Keeps information, but tends to diffuse it
- Drawbacks: often computationally inefficient, better use convolutions

© 2018 Cray Inc.

Most Valuable Layers – Pooling example

Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

https://cs231n.github.io/convolutional-networks/#pool

© 2018 Cray Inc.

Most Valuable Layers – Dropout

- Multiply the input with a random matrix of 1s and 0s

- Effect:
  - Some weights will not influence the output
  - This prevents complex co-adaptations on the input data

- In other words:
  - Since the network cannot see all neurons at the same time, it will learn simpler and more robust connections
  - It reduces overfitting
  - It can also be used to generate probability distributions of the output and quantify uncertainty

© 2018 Cray Inc.

Most Valuable Layers – Dropout example for UQ

http://www.computervisionblog.com/2016/06/making-deep-networks-probabilistic-via.html

© 2018 Cray Inc.

Most Valuable Layers – Batch Normalization

- A layer to normalize (or whiten) the input

- It consists of two steps:
  - Normalization: Subtract mean value of batch input for that layer and divide by standard deviation
  - Rescaling: affine transformation with two learnable parameters

(a) Vanilla Network

(b) Vanilla Network + Single BatchNorm Layer

Santurkar et al. (2018): How Does Batch Normalization Help Optimization?

© 2018 Cray Inc.

Most Valuable Layers – Batch Normalization?

- Initially, Batch Normalization was proposed to contrast internal covariate shift
  - If a class is too represented in a batch, this could lead to less useful gradients
- Recent studies (NeurIPS 2018) show that this seems not to be the real effect of this layer
- Still poorly understood, we know that it smoothens the loss function landscape

© 2018 Cray Inc.

Most Valuable Layers – Residual Blocks

- Residual Blocks are multi-layer blocks, containing **skip connections**
  - The output of one layer can go to the next layer and ”skip” it at the same time, going to the layer after

- This has a large influence on gradient back-propagation
  - Signal is stronger and gradients do not vanish

© 2018 Cray Inc.

Most Valuable Layers – Residual Blocks Effect

Loss function without skip connections

Loss function with skip connections

© 2018 Cray Inc.

Not all networks are born equal

© 2018 Cray Inc.

Multi-Layered Perceptron

- Also called Fully-connected Neural Network or Feedforward Neural Network (FNN)
- Simplest form of Neural Network
- Many drawbacks (overfitting, slow convergence, vanishing gradients)
- Every layer acts like a linear transformation of the input, followed by a non-linear function
  - When used as a classifier, this is equivalent to continuously deform a manifold to make it separable

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – a visualization

- Simple network
  - Input layer
  - FC layer with 6 neurons
    - tanh activation function
  - FC layer with 2 neurons
    - tanh activation function
  - FC layer with 2 neuron

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

© 2018 Cray Inc.

Multi-Layered Perceptron – another visualization

© 2018 Cray Inc.

Multi-Layered Perceptron – another visualization

© 2018 Cray Inc.

Convolutional Neural Networks

- Possibly the most successful types of networks
- Uses sequences of convolutional layers
  - Can be interleaved with pooling operations or fully-connected layers
- Train faster than MLPs
- Can be used for 2D, 3D or higher dimensions (though 2D are the most common)
- Used for image recognition, object detection, sound analysis…
- Usually maximize GPU vs CPU speedup

© 2018 Cray Inc.

Convolutional Neural Network

Left: A regular 3-layer Neural Network. *Right*: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

http://cs231n.github.io/convolutional-networks/

© 2018 Cray Inc.

The data – an example

- Images
  - Usually stored in RGB format
  - Maybe compressed, varying size

- Data preparation
  - Whitening: compute average values, scale
  - Enhance: maybe RGB is not the right space, use PCA
  - Store the data in the right format
    - GPUs like de-interleaved (channels first, or NCHW)
    - CPUs like interleaved (channels last, or NHWC)

© 2018 Cray Inc.

Convolutional Neural Networks: LeNet5

- LeCun 1998

- Weights define sliding filter which is applied on the image
  - Like a convolution, but you switch some signs…

- Two strong features
  - Exploits spatial proximity of pixels
  - Re-uses the same weights in different areas of the image

© 2018 Cray Inc.

Convolutional Neural Network – an example

- AlexNet

© 2018 Cray Inc.

Some examples of CNNs

© 2018 Cray Inc.

Encoder-Decoder

https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.26841

© 2018 Cray Inc.

Generative Adversarial Networks

- GANs were introduced in 2014 by Goodfellow et al.
- The idea is to train a network to generate samples which are indistinguishable from real ones (from the training set)
  - The input is a random noise sample (latent space)
- Another network is trained at the same time to distinguish between real and fake samples

© 2018 Cray Inc.

Generative Adversarial Networks – Applications

Paganini et al.: CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks

Edmond de Belamy (Obvious)

© 2018 Cray Inc.

TRAINING THE NETWORK

Preparing the data

- The first rule is: the more data, the better
- The second rule is: there is no such thing as too much data
- Data has to be subdivided into training, validation, and test sets
  - For datasets which are not very large, 70%, 20%, 10% is a good split
    - If test set is not needed, usually 80% training, 20% validation is good
  - For larger datasets, it is best to have very large training set
    - For example, ImageNet, ~1.2 M images in the training set, ~50k validation
  - All subsets should be statistically representative (~same distribution of cases, classes, labels…)
- Preprocess data when possible
  - Cleaning, whitening
  - Uncompressed storage
  - Parallel storage when possible

© 2018 Cray Inc.

Choosing the network

- If possible, start from a well-known network
  - Something used to solve a problem similar to what you are tackling
  - Many pre-trained models are available
    - Fine-tuning techniques can be used

- Modify the network according to observations
  - What layers are more active?
  - Does convergence stop too early?

- Go large!
  - If you have a powerful system, look for very large networks

© 2018 Cray Inc.

Defining the metric

- Some loss functions are better than others for specific problems
  - Mean Square Error or Mean Absolute Error
    - Good for regressions, extrapolation
  - Cross-entropy loss
    - Good for classification with discrete labels

- Define the figure of merit
  - Usually something easier to understand
  - Top-1 accuracy or error
  - Confusion matrix

© 2018 Cray Inc.

Overfitting and underfitting – a tradeoff

- As in every approximation problem, the choice of the number of parameters is crucial
- Over-parametrization, overfitting
  - The set of learned parameters (weights) give good results for the training set, but bad for the validation or test sets
- Under-parametrization, underfitting
  - With few parameters, the model does not learn and performs bad in all cases
- The right number of parameters is often difficult to estimate in advance
  - There are techniques to adapt it at training time

© 2018 Cray Inc.

Overfitting and underfitting – an example

© 2018 Cray Inc.

Overfitting in deep learning

- During training, we can understand if our model is overfitting the training set

- A large difference between the train and the validation loss is an indicator of overfitting
  - Especially when the validation loss stops converging earlier than the training one!

https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model

© 2018 Cray Inc.

Useful hyper-parameters

- Until now we introduced parameters
  - They determine the output
  - They can be learned
  - They are part of the model

- There is another set of parameters, called hyper-parameters, which have another role
  - They can define the structure of the model
    - E.g. number of layers, convolution size…
  - They can define the training algorithm
    - E.g. learning rate, mini-batch size, weight initialization

© 2018 Cray Inc.

Another class of hyper-parameters: regularizers

- Some hyperparameters can be used to reduce overfitting, i.e. to **regularize the model**
  - Weight decay, or L2-penalty regularization
    - introduces a penalty term which is proportional to the square of the weight
    - basically adds a penalty factor to the gradient
    - avoids large weights
  - L1-penalty regularization
    - introduces a penalty term which is proportional to the absolute value of the weight
    - reduces small weights to zero, thus induces sparsity
    - avoids overfitting, but also selects most relevant features

© 2018 Cray Inc.

Hyper-parameter schedules

- Hyper-parameters can vary during training
- E.g. learning rate:
  - Can be set to initial large value, then reduced after fixed number of iterations or lack of convergence
  - With large batches, we could set it initially to a very small value to avoid gradient explosion, and linearly increase it (warmup phase)
  - Cyclic learning rates (warm-up and cool-down cycles) seem to show some good results, but it is unclear why
- New methods suggest to vary
  - mini-batch size
  - weight decay

© 2018 Cray Inc.

New regularization techniques: pruning

- Recently, neural networks have been growing to several millions of parameters
- Their memory footprint can be a problem for edge devices
- Pruning is the process of removing parts of the network
  - Can be single weights
  - Can be entire filters
  - Can be entire layers

© 2018 Cray Inc.

Automatic Hyper-Parameter Optimization

- Hyper-parameters are critical for reaching the best possible performances
- But how can one determine the best set of Hyper-parameters?
- E.g. we decide to statically set learning rate, weight decay, and warmup epochs
  - three continuous hyper-parameters
  - grid search?
    - what is the right lattice size/increment?
    - even using only 10 possible values for each HP, we need to train 1000 times
- Now what if we want to find a schedule, i.e. an optimal setting for each epoch?
  - Grid search infeasible, we need an automatic way of exploring the HP space
  - **crayai.hpo: Automatic Hyper-Parameter Optimization library!**

© 2018 Cray Inc.

Stochastic Gradient Descent

- (Mini-batch) Stochastic Gradient Descent
  - is naïve
  - in many fields, it has been replaced by more sophisticated techniques
    - E.g. conjugate gradient, BFGS
  - is sensitive to learning rate, batch size, and -basically- all HPs
  - is still used
- Different variants, different ways of computing the step, which are more or less stable, depending on the case
- Some variants are a bit more complex and require more computation time
  - But efficient implementations can reduce this time to a negligible fraction of the total training time

© 2018 Cray Inc.

Stochastic Gradient Descent variations

Method	Formula
Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla \ell(w^{(t)}, z) \quad = w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$
Nesterov Momentum [Nesterov 1983]	$w^{(t+1)} = w^{(t)} + v_t; \quad v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla \ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [Duchi et al. 2011]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t}} + \varepsilon}; \quad A_{i,t} = \sum_{\tau=0}^{t} \left(\nabla w_i^{(t)}\right)^2$
RMSProp [Hinton 2012]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t}} + \varepsilon}; \quad A'_{i,t} = \beta \cdot A'_{t-1} + (1-\beta)\left(\nabla w_i^{(t)}\right)^2$
Adam [Kingma and Ba 2015]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)}} + \varepsilon}; \quad M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1-\beta_m)\left(\nabla w_i^{(t)}\right)^m}{1-\beta_m^t}$

© 2018 Cray Inc.

Activation functions

- Matrix-vector multiplications are linear by definition

- Non-linear transformations are needed
  - To keep the input in a desired range
  - To obtain better (bounded) gradients during backprop

- This is why, in each layer, we apply an **activation function** to the result of the linear transformation

© 2018 Cray Inc.

Activation functions – a very quick survey

Sigmoid

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

tanh

$$\tanh(x)$$

ReLU

$$\max(0, x)$$

Leaky ReLU

$$\max(0.1x, x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

© 2018 Cray Inc.

HANDS-ON SESSION

MLPs in Keras

KERAS – A PRIMER

Keras: Deep Learning for Humans

- Deep Learning Framework for Python
  - High-level API
  - Developed by François Chollet, Google
- Runs on top of other deep learning frameworks
- Supported backends
  - TensorFlow, MXNet (sort of), TypeScript, JavaScript, CNTK, Theano, PlaidML, Scala, CoreML,…
- Ease of use
- Minimalist, highly modular, neural networks library
- Rapid prototyping
- Learn Keras
  - Docs: http://keras.io/
  - Tutorials:
    - https://keras.io/getting-started/sequential-model-guide/
    - https://keras.io/getting-started/functional-api-guide/
  - Examples: https://github.com/keras-team/keras/tree/master/examples

© 2018 Cray Inc.

Two APIs

- Sequential model
  - Sequential definition of all layers belonging to a network, mapping of input to output is automatic
  - Basically:
    - The **Sequential** class is a container, which can be filled with layers
    - It is mandatory to define the input size of the first layer only
    - For all other layers, output size (defined through the first positional argument, or with the units argument) is all you need
    - Once you added the last layer, you have to *compile* the network. The **compile** function simply tells Keras that tensors will have to flow through the layers contained in the sequential model. Keras will turn this structure into one that is suitable for its backend
    - Once the model is compiled, you can train it using the **fit** function
  - **This is what we will use in all notebooks… it is easier done that said, I promise**

- Functional API
  - Manual definition of data movement and transformation
    - More control
    - More code to write

© 2018 Cray Inc.

First: prepare your environment!

- The data used today (and possibly partially tomorrow) can be obtained through a simple script

© 2018 Cray Inc.

Some hints before we start

- We will use Keras in Jupyter Lab
- The first two notebooks are about Multi-Layer Perceptrons
- This means that you will build Neural Networks which only use fully-connected layers
- In Keras, such layers are called **Dense**
  - Refer to the Keras documentation (https://keras.io) every time you want a quick fact or detail, but…
  - **I am here to help! Raise your hand and I will help you! Remember, your doubts may be shared by other students, and my answers could be useful for them too!**
- OK, time to start the first notebook!

© 2018 Cray Inc.

HANDS-ON SESSION

CNNs in Keras

Time for some convolutions!

- Notebooks 3 and 4 are about CNNs
- They should be exhaustive in the definition of the layers, but who knows!
  - I am here!

- Just remember one thing, in Keras, you cannot feed a 2D output to a Dense layer, you will need to put a **Flatten** layer in between!

© 2018 Cray Inc.

Most Valuable Layers

2nd round!

© 2018 Cray Inc.

Most Valuable Layers – Recurrent

- Recurrent layers use their own output as input
  - Introduce history or time dependency in NNs!

- The only way to efficiently train them is to **unroll** them

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

© 2018 Cray Inc.

Most Valuable Layers – Recurrent

- Recurrent blocks suffer from two problems:
  - Long-term dependencies do not work well
    - Difficult to connect two distant parts of the input
  - Magnitude of the signal can get amplified at each recurrent connection
    - At every time iteration, the gradient can either **v**anish or **e**xplode (VEG problem)
  - Very hard to train them

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

© 2018 Cray Inc.

Most Valuable Layers – Long Short-Term Memory

- LSTM layers are improved versions of the recurrent layers
  - They can learn long-term dependencies
  - They can forget
- They have an internal state and a structure which is composed of four actual layers
  - Layers labelled with σ are gates which can block or let information flow

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

© 2018 Cray Inc.

Most Valuable Layers – LSTM forget gate

- LSTMs follow two paths
  - They update their internal state
  - They give an output based on the internal state and the input

- A gate layer σ decides if we should forget an old part of the internal state
  - Something which has to be replaced by new information

© 2018 Cray Inc.

Most Valuable Layers – LSTM new state

- Once that the layer decided what to forget, it computes
  - what has to replace it, i_t, based on the input and the old state
  - what has to be used to replace it, the candidate value $\widetilde{C}_t$

- The new state C_t can be computed based on the new information

© 2018 Cray Inc.

Most Valuable Layers – LSTM Output

- Based on the new state and the input, the layer can produce a result
  - This is the output
  - The same value is also passed to the next iteration

- Why is this so important?
  - Many translation algorithms and voice interpreters are based on small variations of this layer

© 2018 Cray Inc.

Not all networks are born equal

2nd round!

© 2018 Cray Inc.

Recurrent Neural Networks and LSTMs

- Networks based only on Recurrent layers are called RNNs
  - Not very popular because of VEG problem
- Usually LSTMs are the choice for
  - Natural Language Processing
  - Sequence To Sequence
- Not very hardware-friendly
  - Inherently serial
  - Input varies in length
- New: encoder-decoder convolutional networks with attention mechanism
  - Similar to classic CNNs, but the network can focus on parts of the input

© 2018 Cray Inc.

Deep Learning Use Cases

Consumer	Retail	Energy	Financial	Health	Industrial	Autonomous Driving
•Search •Face/Object Detection •Image Segmentation •Speech Understanding •NLP •Text to Speech	•Person and Object Detection •Image Segmentation •Scene Analytics •Support •Marketing •Supply Chain •Security	•Oil and Gas Exploration •Smart Grid •Operational Improvement •Conservation	•Algorithmic Trading •Fraud Detection •Personal Finance •Risk Mitigation •Security	•Enhanced Diagnostics •3D Medical Imaging •Drug Discovery •Sensory Aids	•Factory Automation •Predictive Maintenance •Precision Agriculture •Field Automation	•Pedestrian, Vehicle, and Object Detection and Classification •Ego Motion •Sensor Fusion •Environment Modeling
Topologies: •ResNet •SSD •LSTM •Attention •SparseNN •FCN	**Topologies:** •ResNet •SSD •FCN •RNN	**Topologies:** •Deep Reinforcement Learning •LSTM	**Topologies:** •Deep Reinforcement Learning •LSTM	**Topologies:** •ResNet •SSD •FCN	**Topologies:** •ResNet •SSD •Deep Reinforcement Learning	**Topologies:** •Deep Reinforcement Learning •LSTM •SSD

Life after training

© 2018 Cray Inc.

Deep Learning in Production

Example of an end-to-end workflow

© 2018 Cray Inc.

TensorFlow

- Original developer
  - Google Brain
- Pros:
  - most popular framework
  - tons of tutorials
- Cons:
  - Graph structure adds overhead
  - TensorFlow runtime engine
  - Accessing the C core is hard
- Notes:
  - This is usually the first toolkit used by scientists
  - Google Brain has private copy of repository

© 2018 Cray Inc.

Caffe

- Original developer:
  - Yangqing Jia @ The Berkeley Vision and Learning Center (BVLC)
  - Now also: Berkeley AI Research (BAIR)
- Pros:
  - Fast
  - Good for computer vision
  - Easy to read
- Cons:
  - Many dependencies
  - Python API is incomplete
- Notes:
  - Still very popular

© 2018 Cray Inc.

Caffe2

- Original developer:
  - Yangqing Jia @ Facebook Open Source
- Pros:
  - Fast
  - C core easily accessible
  - Focused on distributed training from scratch
- Cons:
  - Not really really alive
- Notes:
  - More on this later

© 2018 Cray Inc.

Torch

- Original developers:
  - R. Collobert, K. Kavukcuoglu, C. Farabet, S. Chintala
- Pros:
  - Fast
  - Modular
- Cons:
  - Lua interface
- Notes:
  - Rarely used

© 2018 Cray Inc.

PyTorch

- Original developers:
  - R. Collobert, K. Kavukcuoglu, C. Farabet, S. Chintala
  - Now: Facebok Open Source
- Pros:
  - Built from Torch core, not simply a Python API
  - Fast and modular
  - C core easy to access
  - Easy to debug
- Cons:
  - Small community
  - Kind of beta stage
- Notes:
  - Caffe2 has been merged into PyTorch

© 2018 Cray Inc.

MXNet

- Original developer:
  - DMLC – Distributed (Deep) Machine Learning Community
- Pros:
  - Fast
  - C core easily accessible, can also build “C++ executables”
- Cons:
  - Small community
- Notes:
  - Adopted by Amazon
  - Apache Incubator project

© 2018 Cray Inc.

CNTK

- Original developer:
  - Microsoft Research
- Pros:
  - Good for RNNs
  - Native MPI support
- Cons:
  - Not easy to read

© 2018 Cray Inc.

Chainer

- Original developer:
  - Preferred Networks (PFN), Japan
- Pros:
  - Super-fast
  - MPI based
- Cons:
  - Very hot in Japan, not so much anywhere else
- Notes:
  - It’s one of the most recent toolkits and it is gaining momentum

© 2018 Cray Inc.

The DL arms race

© 2018 Cray Inc.

The DL arms race without TensorFlow

© 2018 Cray Inc.

cuDNN

- Original developer:
  - Nvidia
- HW:
  - Nvidia GPUs
- Notes:
  - Leverages CUDA
  - Updated regularly
  - Closed source

© 2018 Cray Inc.

MKL-DNN

- Original developer:
  - Intel
- HW:
  - Intel CPUs
- Notes:
  - Open source
  - Leverages MKL
    - But can work with any BLAS library

© 2018 Cray Inc.

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

© 2018 Cray Inc.