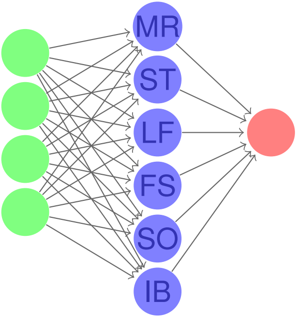
Introduction to Deep Learning and Tensorflow

Day 1



CINECA Roma - SCAI Department

Marco **R**orro **S**tefano **T**agliaventi
Luca **F**erraro **F**rancesco **S**alvadore
Sergio **O**rlandini **I**sabella **B**accarelli

Rome, 7th-9th Nov 2018

Outline

1. Intro and ML Basics
2. Introduction
3. ML ingredients
4. Tensorflow

Outline

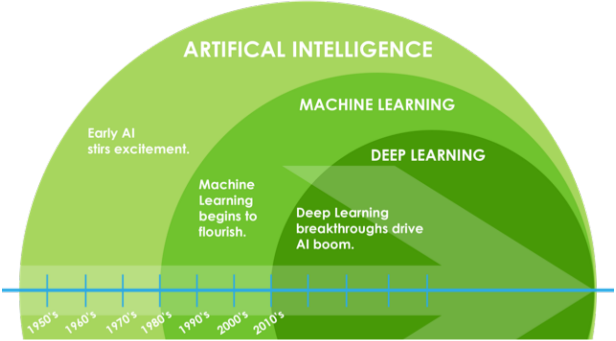
1 Intro and ML Basics

2 Introduction
- The Task
- The Performance Measure
- The Experience

3 ML ingredients

4 Tensorflow

The Big Picture


ARTIFICAL INTELLIGENCE
MACHINE LEARNING
DEEP LEARNING
Early AI stirs excitement.
Machine Learning begins to flourish.
Deep Learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Machine learning basics

Machine learning is a form of applied statistic with increased emphasis on the use of computers to statistically estimate complicated functions and decreased emphasis on proving confidence intervals around these functions.

- Machine learning is a subset of Artificial Intelligence that uses statistics to build models θ of data from data.
- Machine learning systems use frequentist statistics, i.e estimate the true fixed models from uncertain observations.
- Machine learning systems use Bayesian statistics, i.e find best model from families of models.
- **Machine learning is a matter of statistics**

Machine learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on T, as measured by P, improves with experience E.

Learning problems

Intro and ML Basics
Intro
The Task
The Performance
The Experience
ML ingredients
Data
Model
Objective f
Optimization
Tensorflow
Graph and Session
A simple example

Playing checkers

- The task **T** : playing checkers.
- The performance measure **P** : percent of games won against opponents.
- The training experience **E** : playing practice games against itself.

Recognize human faces in images

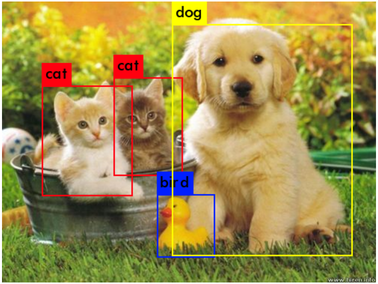
- The task **T** : automatic human face recognition.
- The performance measure **P** : percent of faces recognized.
- The training experience **E** : learn from a database of face images.

The Task

- Machine learning enable us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings.
  - fixed is referred to rule-based system where the decision process is defined at the design level of the algorithm. In such system output is deterministic with respect to the input.
  - probabilistic systems are in contrast to fixed programs.
  - Machine learning deals with probabilistic systems.
  - Time series forecasting
  - Computational linguistics
  - Game palying
  - Driving
- The task shall not be confused with the learning phase, which instead is the procedure used to achieve the task.

Tasks: **Classification** I

- In this type of task, the computer program is asked to specify which of k categories some input belongs to.
  $f : \mathbb{R}^n \to \{1, ..., k\}$.
  Map f could be a probability distribution over classes.
  A typical classification task is object recognition, where the input is an image.

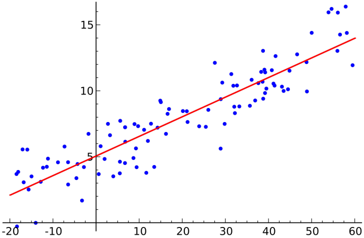


Tasks: **Classification** II

- **Classification with missing inputs:** It is possible to generalize the classification task to take into account incomplete inputs or some missing variable. Instead of learning a single function, the system can learn a probability distribution over all relevant variables, then solve the classification by marginalizing out the missing variables, i.e summing up over the unknown distributions to extract information on known ones.
- A typical use case is related to medical diagnosis where some tests could be invasive or expensive.

Tasks: **Classification** III

- **Regression:** In this type of task the program is asked to predict a numerical value given some input. $f : \mathbb{R}^n \rightarrow \mathbb{R}$ Used for example in algorithmic trading.



Tasks: **Structured output** I

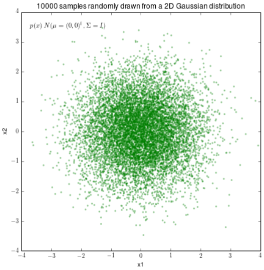
- In structured output tasks output values are related to each other. One example is parsing a natural language sentence where the output is the tree that describes its grammatical structure as for example tagging verbs, nouns, adverbs and so on.
- **Transcription:** In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe the information into discrete textual form. Examples are optical character recognition and speech recognition, where the computer program is asked to map a raw data (image pixels or waveform audio data) to a sequence of characters.
- **Machine translation:** Translate from symbols in some language to symbols in another language.

Tasks

- **Anomaly detection:** In this type of task, the computer program sifts through a set of events or objects and flags some of them as being unusual or atypical. One example is credit card fraud detection, by modeling user purchasing habits.
- **Synthesis and sampling:** In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those into the training data. For example video games can use synthesis to automatically generate textures for large objects. Another example is the speech synthesis task, where given a written sentence an audio waveform is produced containing the spoken version with the requirements that there is no single correct output for each input. A large amount of variation in the output produce a more natural and realistic system.

Tasks II

- **Denoising:** Predict a clean output from a corrupted input.
- **Density estimation:** In this kind of tasks the machine learning algorithm is asked to learn a function $p_{model}\mathbb{R}^n \to \mathbb{R}$ where $p_{model}(x)$ can be interpreted as a probability density function(if x is continuous) or a probability mass function (if x is discrete) on the space that the examples where drawn from. In density estimation tasks the algorithm needs to learn the structure of the data.



The Performance Measure,**P**

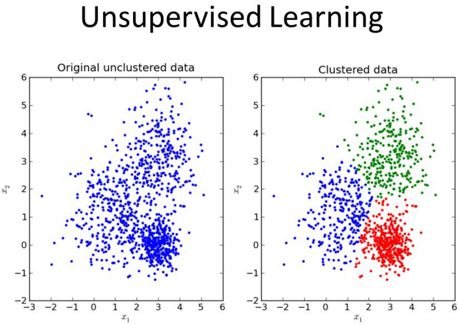
- To evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance, usually related to the task **T**.
- For classification tasks we can use the **accuracy/error rate**, i.e. the proportion of examples for which the model produces the correct/incorrect output.
- In machine learning field we are interested in evaluating the performance on data that is not seen before.
  - We evaluate these performance measures using a **test set** of data that is separate from the data used for training the machine learning system.
- It is often difficult to choose a performance measure corresponding well to the desired behavior of the system.

The Experience, E I

Two main categories:

- **Unsupervised** learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset.
    - In some contexts we wish to learn the entire probability distribution that generated a dataset
        - Implicitly : denoising, synthesis
        - Explicitly : density estimation

The Experience, **E** II

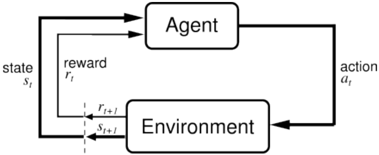


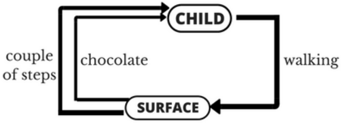
- **Supervised** learning algorithm experience a dataset containing features, but each example is also associated with a **label** or **target**
    - Regression, classification, structured output

The Experience, **E** III

- Unsupervised and supervised are not formally defined terms, many ML technologies can be used to perform both.

- **Reinforced** learning is another learning algorithm that interact with environment. Experience is not just made on a fixed dataset but there is a feedback between the algorithm and its experience.





Outline

1 Intro and ML Basics

2 Introduction

3 ML ingredients
- Data
- Model
- Objective function
- Optimization

4 Tensorflow

ML ingredients

- Data
  - The data input and output
- Model
  - The model used to generate output.
- The objective function
  - The function used to train the model
- Optimization Algorithm
  - The algorithm used to minimize loss function

The data I

- One common way of describing an input dataset is with a **design matrix**
  - A matrix containing a different example in each row.
  - Each columns of the matrix corresponds to a different feature.
- For example, if we have 101 handwritten images 28x28 representing digits, the design matrix is $X \in \mathbb{R}^{101X784}$
- A linear regression with 5 $\{y_i, x_i\}, i = 1, 2, ..., 5$ observation

$$y_i = \beta_0 + \beta_1 x_i \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

- In general if we have heterogeneous examples (not the same dimensions) an m row matrix $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ where each example vectors have different sizes.

The Central Challenge I

- The system should perform well on new, previously unseen inputs not just those on which our model was trained. **Generalization** : the ability to perform well on previously unobserved inputs.
- There are two important concepts used in machine learning: the **training error** and the **test error** that corresponds to a subdivision of the dataset.
- **Training error** : compute some error measure on the training set.
- **Test error** : the expected value of the error on a new input. Example: linear regression $\hat{y} = \mathbf{w}^T \mathbf{x}$ where $\mathbf{w} \in \mathbb{R}^n$
- $\frac{1}{m^{(train)}} ||\mathbf{X}^{(train)}\mathbf{w} - y^{(train)}||_2^2$
- $\frac{1}{m^{(test)}} ||\mathbf{X}^{(test)}\mathbf{w} - y^{(test)}||_2^2$

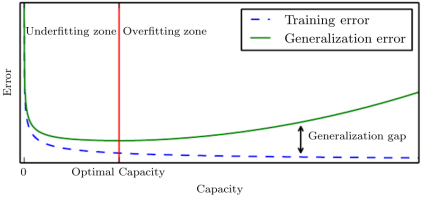
The Central Challenge II

- The factors determining how well a machine learning algorithm will perform are its ability to:

1. Make the training error small.
2. Make the gap between training and test error small. These two factors correspond to the two central challenges in machine learning: underfitting and overfitting .

The Central Challenge III

1. **Underfitting** occurs when the model is not able to obtain a sufficiently low error value on the training set. It has not enough **capacity** to represent data.
2. **Overfitting** occurs when the gap between the training error and test error is too large.



Hyperparameters

- **Hyperparameters** are settings of the machine learning algorithm that must be determined outside the learning algorithm itself.
- For example in a polynomial regression the degree of the polynomial, is a hyperparameter wich act as a capacity (the ability to fit a wide variety of functions)
- To estimate hyperparameters a **validation set** of examples is used.
- Typically the **validation set** is taken from the training set (two disjoint sets).
- A typical choice is to use 20% of the training set as validation set.

Regualrization

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

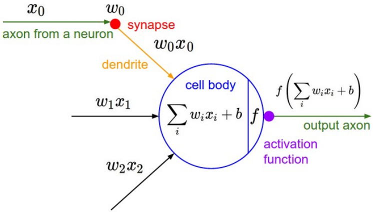
The Model

- **Parametric models** such as linear regression, learn a function described by a parameter vector whose size is finite and fixed before any data is observed.
- **Non-parametric models** have no such limitations. A practical example is the nearest neighbor regression.

$$\boxed{y = f(x; \theta)}$$

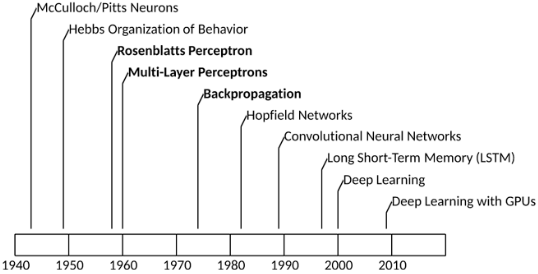
Artificial Neural Networks I

- Biologically inspired model.



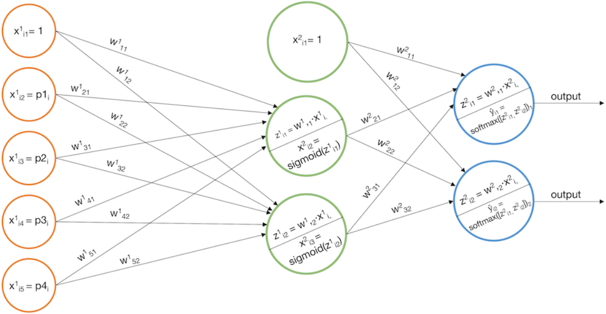
- a brief history:

Artificial Neural Networks II



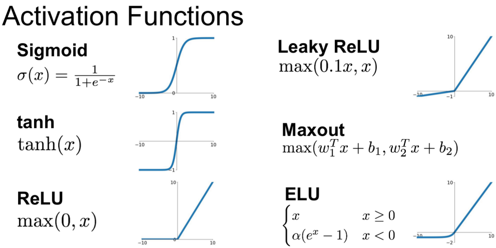
- Inputs to neuron are wheighted then summed.
- A term called **bias** independent of input is also added.
- The output of the neuron is the application of an **activation function** to the sum previously computed.
- In the perceptron model, the activation function is a threshold function.
- Units(neuron) can be combined togheter to form a network

Artificial Neural Networks III


$x^1_{i1} = 1$
$x^1_{i2} = p1_i$
$x^1_{i3} = p2_i$
$x^1_{i4} = p3_i$
$x^1_{i5} = p4_i$
w^1_{11}
w^1_{12}
w^1_{21}
w^1_{22}
w^1_{31}
w^1_{32}
w^1_{41}
w^1_{42}
w^1_{51}
w^1_{52}
$x^2_{i1} = 1$
$z^1_{i1} = w^1_{,1} \cdot x^1_{i,}$
$x^2_{i2} = \text{sigmoid}(z^1_{i1})$
$z^1_{i2} = w^1_{,2} \cdot x^1_{i,}$
$x^2_{i3} = \text{sigmoid}(z^1_{i2})$
w^2_{11}
w^2_{12}
w^2_{21}
w^2_{22}
w^2_{31}
w^2_{32}
$z^2_{i1} = w^2_{,1} \cdot x^2_{i,}$
$\hat{y}_{i1} = \text{softmax}([z^2_{i1}, z^2_{i2}])_1$
$z^2_{i2} = w^2_{,2} \cdot x^2_{i,}$
$\hat{y}_{i2} = \text{softmax}([z^2_{i1}, z^2_{i2}])_2$
output
output

- It is equivalent to function composition
- Examples of activation functions:

Artificial Neural Networks IV



- Sigmoid and tanh used in classification tasks, slow to converge
- ReLU most used, fast convergence, dead neuron issue.
- Softmax function $\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is used in categorial distribution, it represents the probability over a finite number of categories

The objective function I

- The objective function is a real values function that provides a measure of how wrong the model is in terms of its ability to estimate the relationship between inputs and outputs.
- It refers to an optimization problem, where the objectve (loss,cost) function is minimized or maximized respect to the parameters.
- An example is the mean square error function used in regression problems where:

$$J(\theta) = MSE_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} ||\hat{\boldsymbol{y}}^i(\theta) - \boldsymbol{y}^i||^2$$

  is to be minimized with respect to the parameters θ
- or the cross-entropy function used in classification problems:

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_x \boldsymbol{p}(x) \log \boldsymbol{q}(x).$$

- where $\boldsymbol{p}(x)$ is the true probability distribution.
- It can be shown that these function can be derived using a general principle.

The objective function II

- The Maximum Likelihood Estimation

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p_{model}(\mathbb{X};\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{m} p_{model}(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$$

  - where $\mathbb{X} = x^{(1)}, ..., x^{(m)}$ is a set of m examples drawn independently from the true but unknown data-generating distribution $p_{data}(\boldsymbol{x})$
  - $p_{model}(\boldsymbol{x};\boldsymbol{\theta})$ is a parametric family of probability distribution over the same space indexed by $\boldsymbol{\theta}$
  - A more convenient but equivalent optimization problem is redefining the ML taking the logarithm.

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{model}(\boldsymbol{x}^{(i)};\boldsymbol{\theta})$$

  - The ML method works if the true distribution $p_{(data)}$ lies within the model family $p_{model}(;\theta)$.
  - The $p_{(data)}$ must correspond to exactly one value of θ.

The optimization algorithm

- Optimization refers to the task of either minimizing or maximizing some function $\boldsymbol{f(x)}$ by altering $\boldsymbol{x}$.
- The first derivative of the cost function can be useful to minimize, or equivalently maximize , the cost function with respect to the parameters.
  - For example we know that $f(x - \epsilon sign(f'(x)))$ is less than $f(x)$ for small ϵ
  - **gradient descent**: we can thus reduce $f(x)$ by moving x in the opposite sign of the derivative.
- In a multi dimensional fashion partial derivatives must be computed, so $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how f changes as only the variable x_i increases at point x.
- The generalization of the **gradient descent** method is

$$\boldsymbol{x' = x - \epsilon \nabla_x f(x)}$$

- where ϵ, the **learning rate**, is a positive scalar determining the size of the step.

The backpropagation algorithm I

Intro and ML Basics

Intro
The Task
The Performance
The Experience

ML ingredients
Data
Model
Objective f
Optimization

Tensorflow
Graph and Session
A simple example

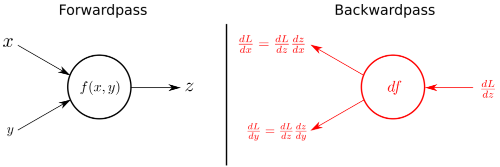
- The chain rule of calculus:
  - if y=g(x) and z=f(g(x))=f(y)

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

  - if $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{y} = g(\boldsymbol{x}) : \mathbb{R}^m \to \mathbb{R}^n$ and $z = f(\boldsymbol{y}) : \mathbb{R}^n \to \mathbb{R}$

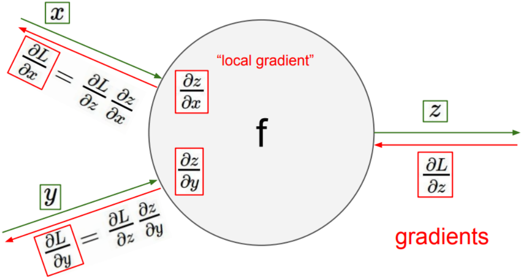
$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i} = \nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z$$

  - $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the *nxm* Jacobian matrix of *g*
- The algorithm:
  1. Forward propagate inputs example through the network.


Forwardpass
Backwardpass
x
y
$f(x,y)$
z
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
df
$\frac{dL}{dz}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$

The backpropagation algorithm II

2. Compute the changes to the loss function backward to the input using the chain rule


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

- Use the derivative of loss function to update weights of the network:

$$x' = x - \epsilon \nabla_x f(x)$$

Stochastic Gradient Descent I

- An extension of the gradient descent algorithm is the **stochastic gradient descent**
- Most used algorithm to train neural networks.
- Given X the set of all inputs and Y the corresponding observed targets, $P(Y|X,\theta)$ is the probability to predict Y given X.
- The **conditional log-likelihood** is:

$$\theta_{ML} = \underset{\theta}{\text{argmax}}\, P(Y|X;\theta)$$

- If the examples in the dataset are independet and identically distributed, $\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{m} logP(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)} : \theta)$
- An example of cost function related to negative conditional log-likelihood is: $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$ where $L(x^{(i)}, y^{(i)}, \theta) = -logp(\boldsymbol{y}|\boldsymbol{x} : \theta)$

Stochastic Gradient Descent II

- the gradient descent then become:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

Stochastic Gradient Descent III

- It can be sampled using small set of samples drawn uniformly from the training set $x^{(1)}, ..., x^{(m')}$

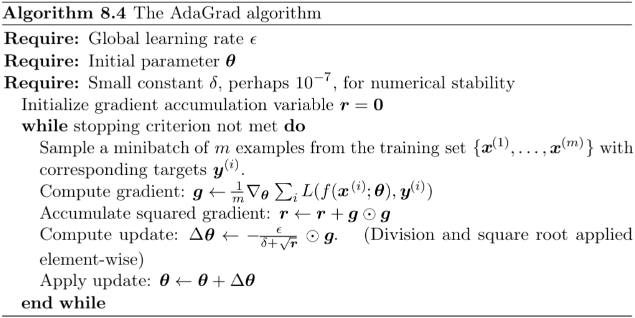
$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\theta} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

- the sample $x^{(1)}, ..., x^{(m')}$ is called **minibatch**
- Minibatch size is typically chosen small and fixed.

AdaGrad

- Adapts the learning rates of all model parameters by scaling them inversely proprortional to the square root of the sum of all the historical squared values of the gradient.
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate.
- The net effect is grater progress in mthe more gently slope directions of parameter space.



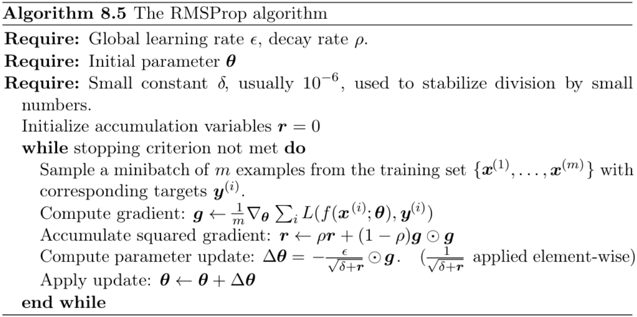
Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

- Suitable for not all deep learning models

RMSProp

- Better that AdaGrad in the nonconvex setting
- Exponentially weighted moving average.
- Decaying discard history from exterme past so that it can converge rapidly after finding a convex bowl.

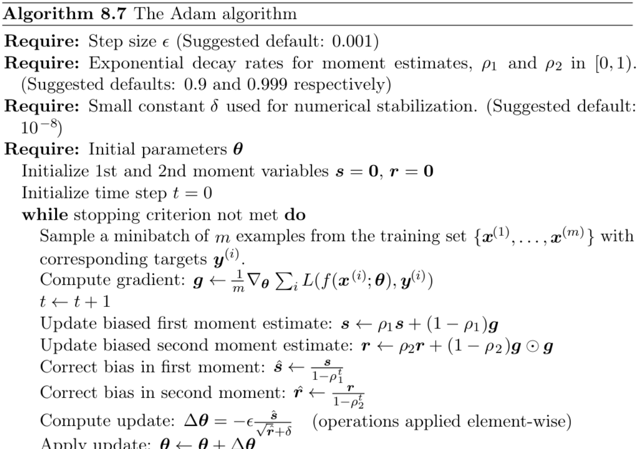


Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
  numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while
```

Adam

- Adaptive moments
- Variant of RMSProp and momentum
- As a general rule momentum appplied to gradient descent, reduce oscillating contribution.



Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
```

Outline

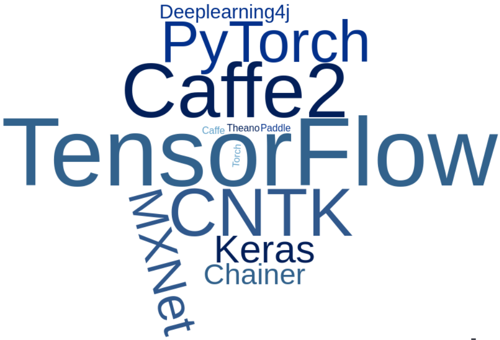
1 Intro and ML Basics

2 Introduction

3 ML ingredients

4 Tensorflow
- Graph and Session
- A simple example

Deep learning frameworks



and others

https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch

Framework desired features

We are interested in:

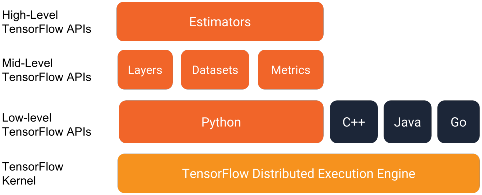
- classical machine learning algorithms
- deep learning approach (especially convolutional neural network)
- high level language (Python)
- little/no programming effort
- integration with existing pipelines
- multi-node CPU and GPU support

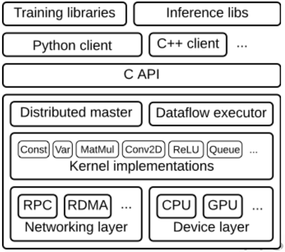
TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful data-flow graphs
- automatic differentiation capabilities
- optimization algorithms: gradient and proximal gradient based
- code portability (CPUs, GPUs, on desktop, server, or mobile computing platforms)
- Python interface is the preferred one (Java, C++ and Go also exist)
- installation through: virtualenv, pip, Docker, Anaconda, from sources
- Apache 2.0 open-source license

Tensorflow

- Tensorflow is a computational framework for building machine learning models
  - High-level, object-oriented API (tf.estimator)
  - Libraries for common model components (tf.layers/tf.losses/tf.metrics)
  - Lower-level APIs (TensorFlow)


High-Level TensorFlow APIs
Estimators
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine

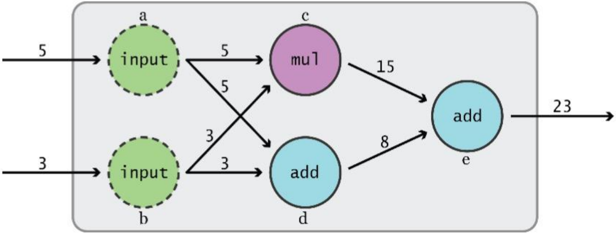


Tensorflow

- The foundation of computation in TensorFlow is the Graph object. This holds a network of nodes, each representing one operation, connected to each other as inputs and outputs
  - a graph protocol buffer
  - a runtime that executes the (distributed) graph
- What are protocol buffers?
  - Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data - think XML
    - define how you want your data to be structured
    - then you can use special generated source code to easily write and read your structured data
  - All of TensorFlow's file formats are based on Protocol Buffers
    - define data structures in text files, and the protobuf tools generate classes in C, Python, and other languages that can load, save, and access the data

Data Flow Graphs

- TensorFlow separates definition of computations from their execution
  - Phase 1: assemble a graph
  - Phase 2: use a session to execute operations in the graph.
- not true in eager mode



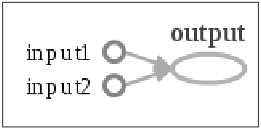
Tensor

- An n-dimensional array
- 0-d tensor: scalar (number)
- 1-d tensor: vector
- 2-d tensor: matrix
- and so on

TensorFlow: basic usage

```
import tensorflow as tf

# define graph nodes (on default graph)
input1 = tf.constant([1.0, 1.0, 1.0, 1.0])
input2 = tf.constant([2.0, 2.0, 2.0, 2.0])
output = tf.add(input1, input2)

# launch a session
sess = tf.Session()

# run graph on session and retrieve results
result = sess.run(output)

# close the session
sess.close()
```

- graph-related lines
- session-related lines
- evaluation-related lines



TensorFlow Graph Class

An instance of tf.Graph class:

- represents, as a directed graph, the dataflow of the computation we want to perform;
- graph nodes are called operations (instances of tf.Operation class);
    - not necessarily a mathematical operation, but also a variable/constant definition, . . .
- each operation involves zero or more instances of tf.Tensor class, it produces zero or more instances of same class;
- a tf.Tensor is a multi-dimensional array;
- in these examples we are using the default graph (accessible through tf.get_default_graph() function).

Session Class

An instance of tf.Session class:

- is used for running graph nodes;
- may own resources, so use Session.close() method when you are done with it;
- it can be customizable
  - you chan choose hardware where evaluation is performed;
- it can provide results of node evaluations through Session.run() method.

Tensors: variables, constants, placeholders

An instance of tf.Variable class:

- implements the mathematical concept of variable;
- maintains it's value accross different Session.run() calls;
- can be assigned a new value through the function tf.assign() (and others of type tf.assign...())
- it requires an initialization step.

Other Tensors:

- use tf.constant() function to initialize constant tensors;
- use tf.placeholder() for a tensor that will be fed later during the evaluation.

approx. minimization of a univariate functional (I)

```
 import tensorflow as tf
 import numpy as np

# define variable and placeholders
a_op = tf.placeholder( tf.float32 , shape=(1) )
b_op = tf.placeholder( tf.float32 , shape=(1) )
x_op = tf.Variable( [ 2.0 ], dtype=tf.float32 )

y_op = a_op * x_op**2  + b_op * x_op

optimizer = tf.train.AdamOptimizer(0.1)
min_step = optimizer.minimize( y_op )

init_op = tf.global_variables_initializer()
```

Define tensors: placeholders for *a* and *b* parameters, variable for *x*. Overloaded operators will call pointwise math operations: what we are actually calling is:

```
y_op = tf.add( tf.multiply( tf.pow( x_op, 2 ), a_op),
          tf.multiply(x_op, b_op ) )
```

approx. minimization of a univariate functional (II)

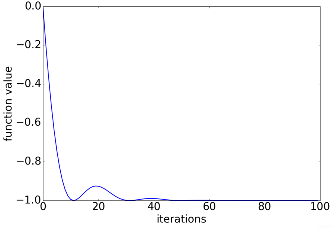
```
# launch a session
sess = tf.Session()

# initialize variables
sess.run(init_op)

a = [ 1.0]
b = [-2.0]
y = []
for ii in range( 100 ):
    _,y_p = sess.run([min_step, y_op],
                     feed_dict = { a_op: a, b_op: b } )
    y.append( y_p )

x = sess.run(x_op)

sess.close()
```

Define a session and initialize all variables of the graph. Define some Python variables: function parameters *a* and *b* and a vector *y* to hold function value at each iteration. Implement main loop: note the Python dictionary feed_dict and the return values of sess.run() method. Retrieve the estimated value of the minimum point.

approx. minimization of a univariate functional (III)

```
import matplotlib.pyplot as plt

print('solution: %f, (true: %f)'
  % (x, - np.divide(b, np.multiply( [a], 2.0) )))

plt.plot(y)
plt.xlabel('iterations')
plt.ylabel('function value')
plt.show()
```

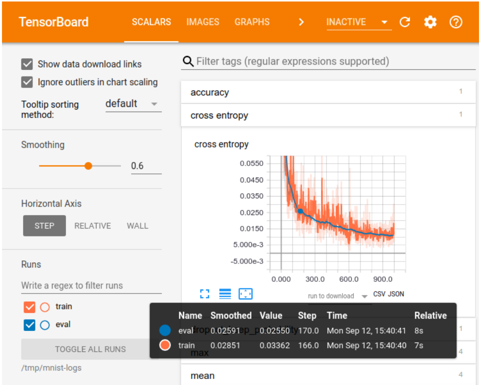
Estimated value is 1.002937, real is 1.0.



Tensorboard

- Tensorboard visualizes TensorFlow graph, quantitative metrics and other data (images)

```
 # before run.sees()
writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())

writer.close() # close the writer when you're done using it

tensorboard --logdir="./graphs"
```



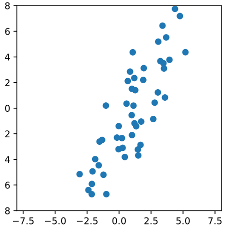
☑ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP
Runs
Write a regex to filter runs
☑ ○ train
☑ ○ eval
/tmp/mnist-logs

Filter tags (regular expressions supported)
accuracy 1
cross entropy 1
cross entropy
dropout_keep_probability 1
max 4
mean 4

	Name	Smoothed	Value	Step	Time	Relative
●	eval	0.02591	0.02550	170.0	Mon Sep 12, 15:40:41	8s
●	train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

Linear regression (I)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Set up the data with a noisy linear relationship between X and Y.
num_examples = 50
X = np.array([np.linspace(-2,4,num_examples),
              np.linspace(-6,6,num_examples)])
X += np.random.randn(2, num_examples)
x, y = X
x_with_bias = np.array([(1., a) for a in x]).astype(np.float32)

losses = []
training_steps = 50
learning_rate = 0.002
```



1. generate noisy input data
2. set slack variables and fix algorithm parameters

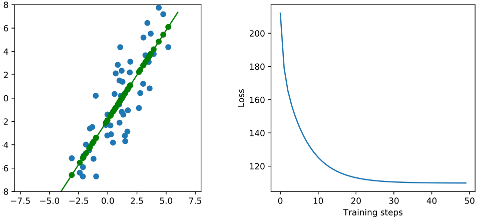
Linear regression (II)

```
# Start of graph description
# Set up all the tensors, variables, and operations.
A = tf.constant(x_with_bias)
target = tf.constant(np.transpose([y]).astype(np.float32))
weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

yhat = tf.matmul(A, weights)
yerror = tf.subtract(yhat, target)
loss = tf.nn.l2_loss(yerror)

update_weights =
  tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

sess = tf.Session()
sess.run( tf.global_variables_initializer() )
for _ in range(training_steps):
  # Repeatedly run the operations, updating variables
  sess.run( update_weights )
  losses.append( sess.run( loss ) )
```

3. define tensorflow constants and variables
4. define nodes
5. start evaluation

Linear regression (III)

```
# Training is done, get the final values
betas = sess.run( weights )
yhat = sess.run( yhat )
```



Advantages and disadvantages

TensorFlow in general:

✓ High-level, object-oriented API (*tf.estimator*)

✓ Libraries for common model components (*tf.layers*/*tf.losses*/*tf.metrics*)

✓ Lower-level APIs (TensorFlow)

✓ very simple installation/setup on supported platforms

✗ not too easy to install and setup on unsupported platform

✓ plenty of tutorials, exhaustive documentation

✓ tools for exporting (partially) trained graphs (see MetaGraph)

✓ debugger, graph flow visualization

as a Python user:

✓ “Python API is the most complete and the easiest to use”

✓ numpy interoperability

✓ Keras interoperability

Moving forward

- tensorflow.org
- TensorFlow Dev Summit 2018
- TensorFlow Dev Summit 2017
- Stanford CS 20: Tensorflow for Deep Learning Research

Useful library:

- Keras
- Horovod