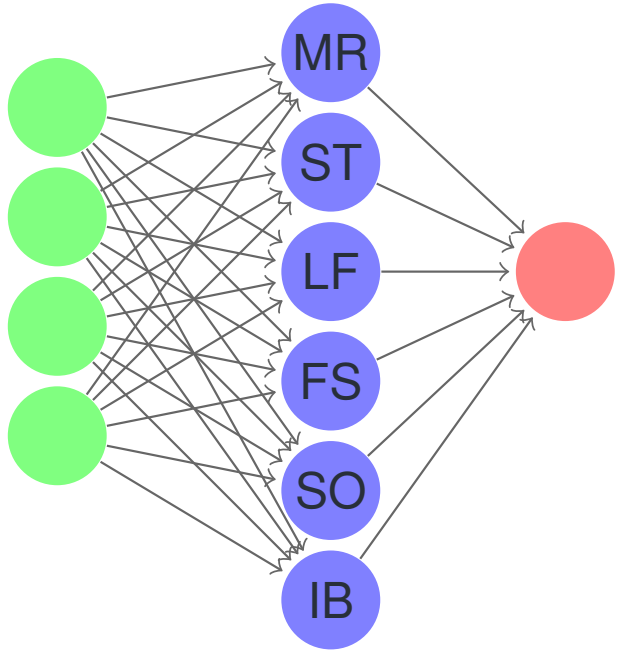
CINECA SCAI
SuperComputing Applications and Innovation

Introduction to Deep Learning and Tensorflow

Day 3



CINECA Roma - SCAI Department

Marco **R**orro
Luca **F**erraro
Sergio **O**rlandini
Stefano **T**agliaventi
Francesco **S**alvadore
Isabella **B**accarelli

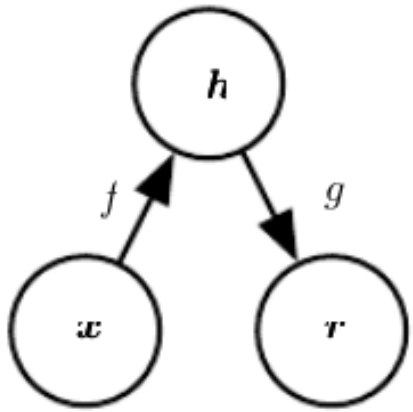
Rome, 7th-9th Nov 2018

CINECA

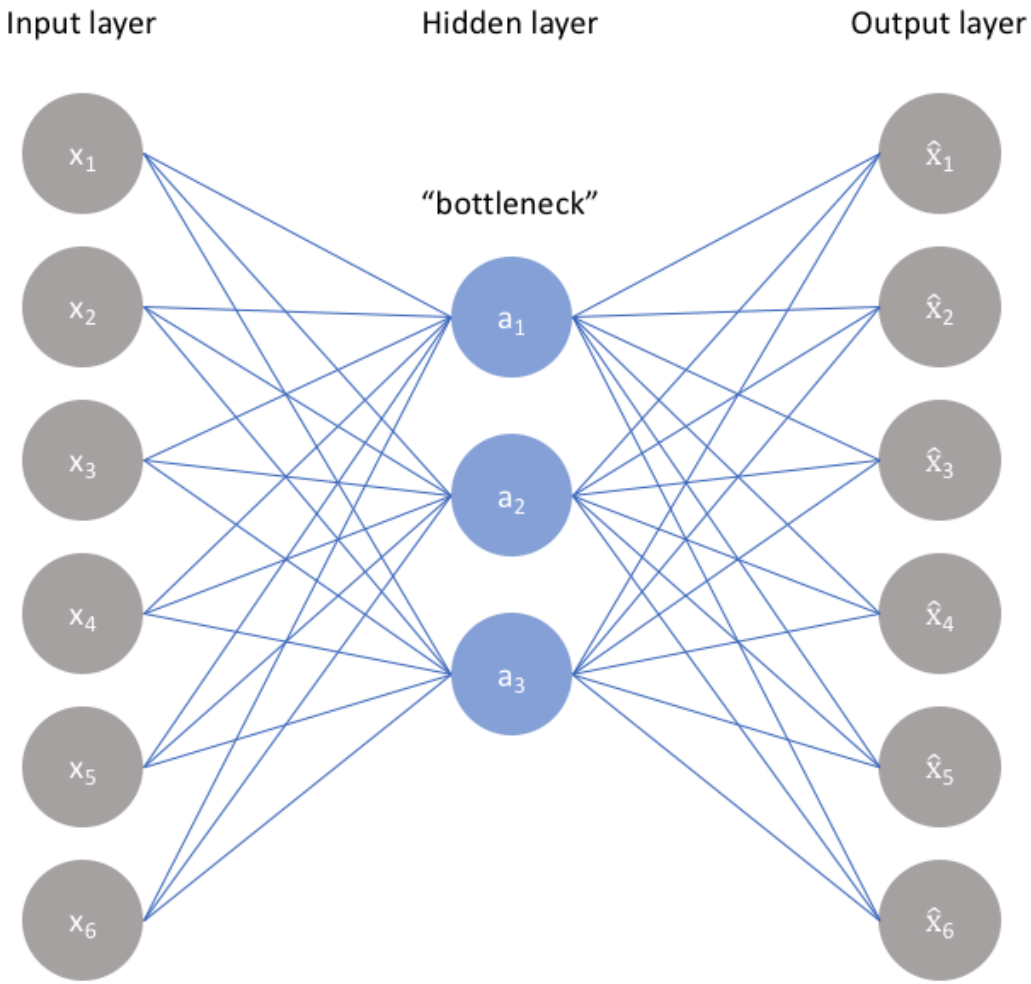
Outline

Autoencoders

- A basic **autoencoder** is a neural network that is trained to attempt to copy its input to its output.
  - as such it is usually considered an *unsupervised learning* algorithm
- Internally, it has a hidden layer **h** that describes a **code** (*internal representation*) used to represent the input.
- The network may be viewed as consisting of two parts
  - an encoder function $h = f(x)$
  - a decoder that produces a *reconstruction* $r = g(h)$.



Autoencoders sketch



AE applications

- If an autoencoder succeeds in simply learning to set $g(f(x)) = x$ everywhere, it does not seem especially useful
- However, (different types of) autoencoders have several applications:
  - compression
  - dimensionality reduction (to be used to feed other networks)
  - feature extraction (possibly before feeding to other networks)
  - unsupervised pre-training
  - noise reduction
  - anomaly detection

Undercomplete AE

- If the *h* is constrained to have a smaller dimension than *x* – *undercomplete autoencoder* – the autoencoder can only produce an approximate reconstruction of *x*
- ...often we are not interested in the output of the decoder but in the the *code* output of the encoding part *h*
- In such case the *code h*
  - is a compressed version of *x*
  - can capture the most salient features of the training data
- The learning process tries to minimize the loss function which penalizes the reconstruction for being dissimilar from the input

$$L(x, g(f(x)))$$

- Nonlinear autoencoders somehow learn a nonlinear generalization of PCA

Regularized AE

- If the dimensionality of the *code* is too high – equal to input or larger than input (*overcomplete*) – it can completely store the input information (even using linear encoder and decoder) but may fail to learn anything useful
- Rather than limiting the model capacity by keeping the *code* size small, *regularized autoencoders* use a loss function that encourages the model to have other properties besides the ability to copy its input to its output
  - sparsity of the representation
  - smallness of the derivative of the representation
  - robustness to noise
  - robustness to missing inputs
- A regularized autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution

Sparse AE

- A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the *code* layer h, in addition to the reconstruction error:

$$L(x, g(f(x))) + \Omega(h)$$

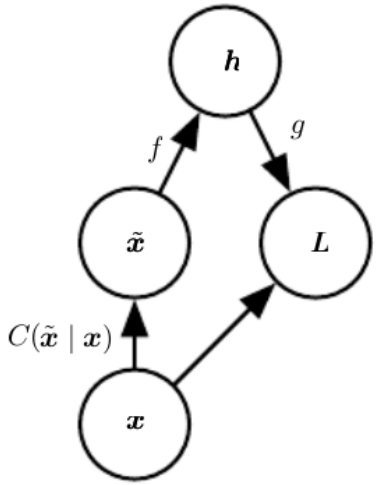
  where typically we have $h = f(x)$, the encoder output.
- Sparse autoencoders are typically used to learn features for another task such as classification.
- The penalty $\Omega(h)$ can be seen as a regularizer term added to a feedforward network whose primary task is to copy the input to the output (unsupervised learning objective) and possibly also perform some supervised task (with a supervised learning objective) that depends on these sparse features.
- Unlike other regularizers such as weight decay, there is not a straightforward Bayesian interpretation to this regularizer.
- One way to achieve actual zeros in h for sparse (and denoising) autoencoders is to use rectified linear units to produce the code layer. With a prior that actually pushes the representations to zero (like the absolute value penalty), one can thus indirectly control the average number of zeros in the representation.

Denoising AE

- Rather than adding a penalty Ω to the cost function, we can obtain an autoencoder that learns something useful by changing the reconstruction error term of the cost function.
- A denoising autoencoder or DAE instead minimizes

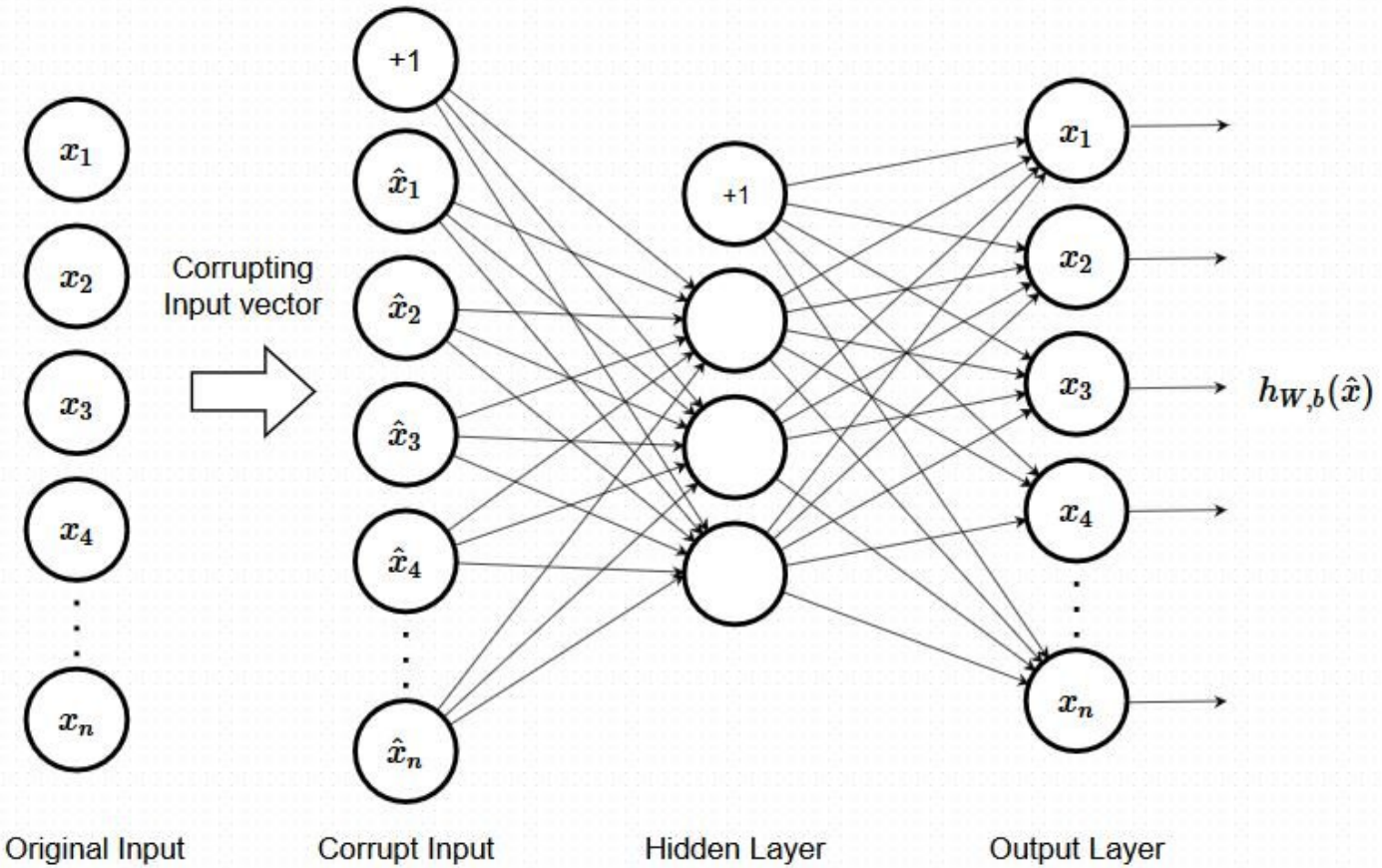
$$L(x, g(f(\tilde{x})))$$

  where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise.
- Denoising autoencoders must therefore undo this corruption rather than simply copying their input.



Denoising AE sketch

- Not only denoising autoencoders are used to remove noise: in order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, we train the autoencoder to reconstruct the input from an artificially corrupted version of it.



Deep or shallow?

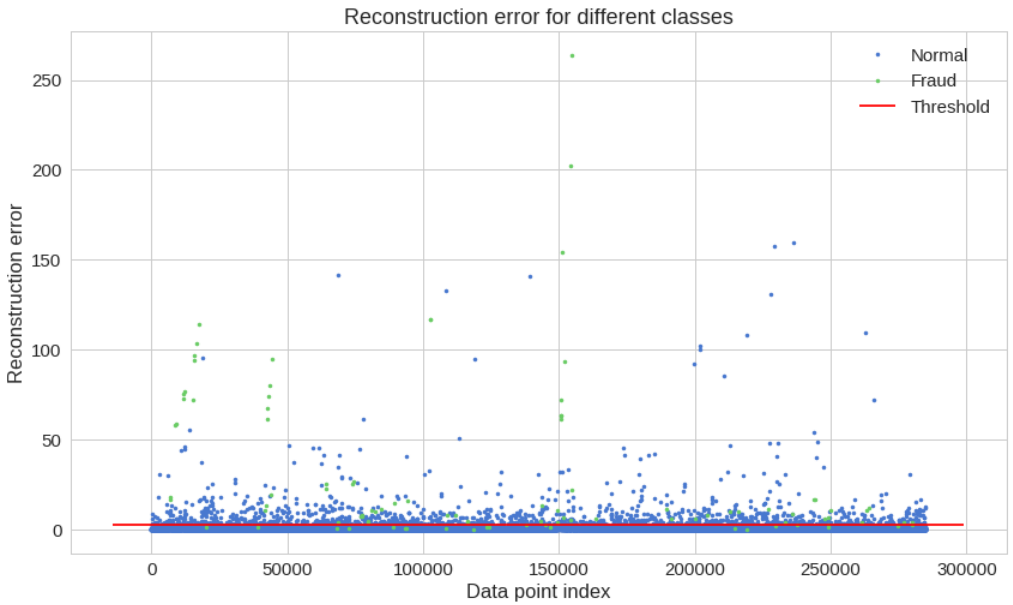
- Autoencoders are just feedforward networks. The same loss functions and output unit types that can be used for traditional feedforward networks are also used for autoencoders.
- Autoencoders are often trained with only a single layer encoder and a single layer decoder. However, this is not a requirement. In fact, using deep encoders and decoders offers many advantages.
- Depth can exponentially reduce the computational cost of representing some functions.
- Depth can also exponentially decrease the amount of training data needed to learn some functions.
- Experimentally, deep autoencoders yield much better compression than corresponding shallow or linear autoencoders
- A common strategy for training a deep autoencoder is to greedily pretrain the deep architecture by training a stack of shallow autoencoders, so we often encounter shallow autoencoders, even when the ultimate goal is to train a deep autoencoder.

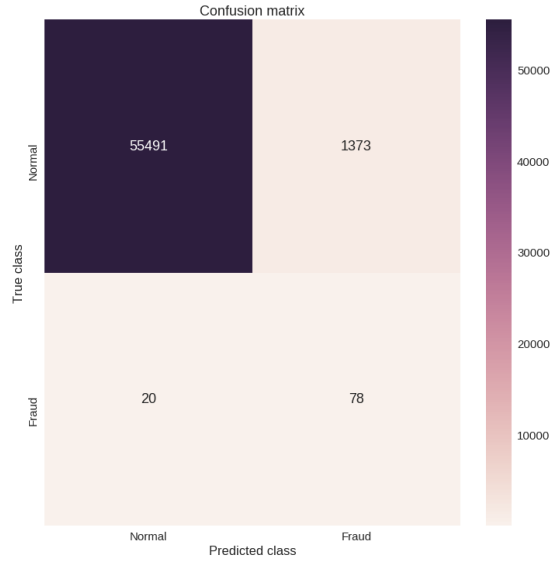
Anomaly detection using AE - Part I

- Example dataset: 280000 instances of credit card use and for each transaction the classification fraudulent/legit
- The goal: for a new transaction predict if fraudulent or not
- Some critical issues:
  - as a classification problem there is lack of large fraudulent training set
  - marking every transaction as non-fraud would lead to > 99% accuracy (we need a different metrics)
  - no single parameter analysis helps in finding out the fraud transactions
- Idea: use AE training the model on the normal (non-fraud) transactions
- Once the model is trained, in order to predict whether or not a new/unseen transaction is normal or fraudulent, we calculate the reconstruction error

Anomaly detection using AE - Part II

- If the error is larger than a predefined threshold, we will mark it as a fraud
  - depending on the threshold we will have different quality of results
  - the so called *confusion matrix* helps summarizing the quality of results



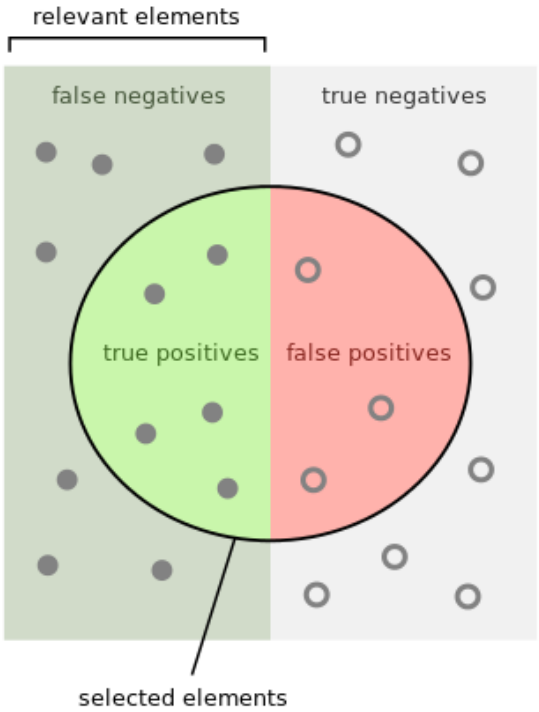


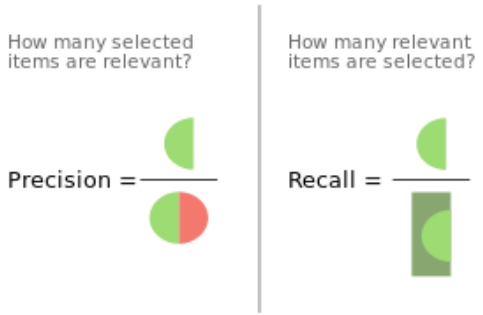
Accuracy vs Precision vs Recall

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}$$

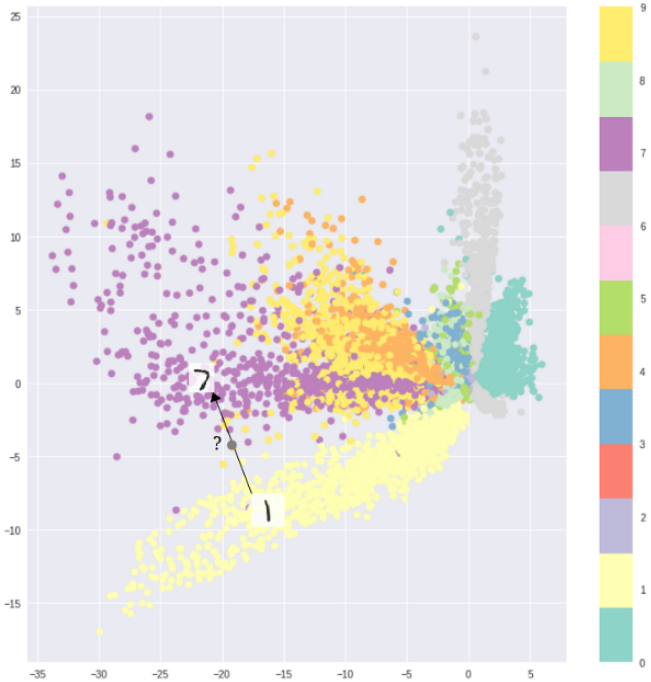




- Accuracy alone doesn't tell the full story when you're working with a class-imbalanced data set
  - assuming trivial *always predominant* model gives high accuracy, but it is useless

Variational autoencoders (ideas) - 1

- Variational Autoencoders (VAEs) are powerful *generative* models now having applications as diverse as
  - generate a random, new output, that looks similar to the training data (e.g. generating fake human faces, producing purely synthetic music)
  - alter, or explore variations on data you already have, and not just in a random way either, but in a desired, specific direction
- The fundamental problem with autoencoders, for generation, is that the latent space where their encoded vectors lie, may not be continuous, or allow easy interpolation.

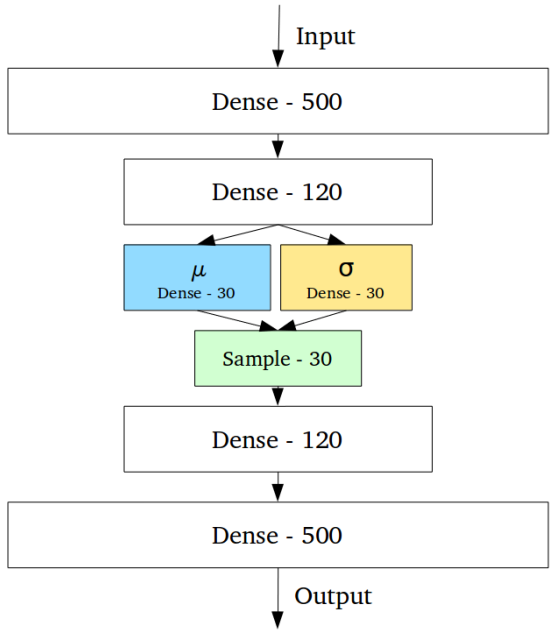


Variational autoencoders (ideas) - 2

- In a generative model, you want to randomly sample from the latent space, or generate variations on an input image, from a continuous latent space.
  - if the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will most probably generate an unrealistic output
- In VAEs their latent spaces are, by design, continuous, allowing easy random sampling and interpolation.
- Strategy: make encoder not output a *code* of size n but two vectors of size n: a vector of means, μ, and another vector of standard deviations, σ.
- μ and σ are mean and standard deviation of the $i - th$ random variable

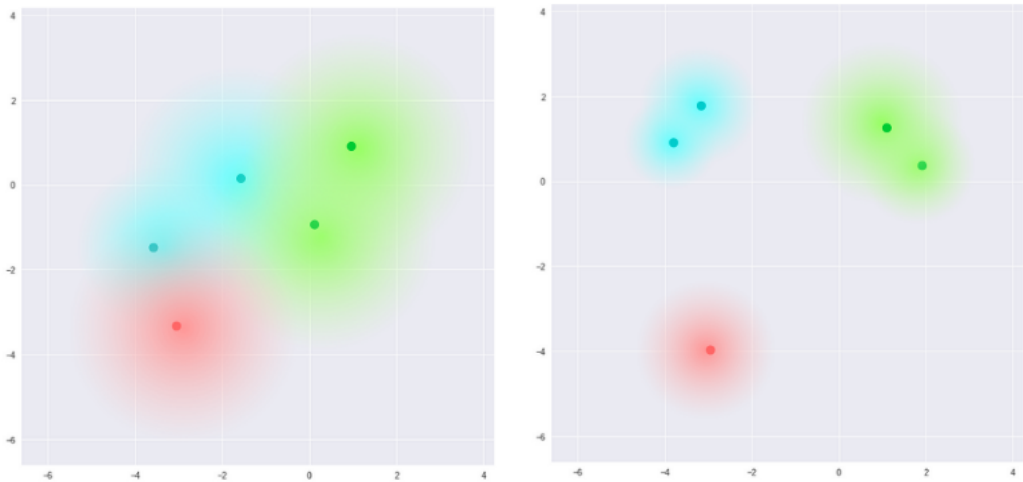
Variational autoencoders (ideas) - 3

- This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling
  - this allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.
- Ideally the target is having encodings which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of new samples



Variational autoencoders (ideas) - 4

- This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling
  - this allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.
- Ideally we want encodings, all of which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of new samples



What we require

What we may inadvertently end up with

Variational autoencoders (ideas) - 5

- We introduce the Kullback-Leibler divergence (D_{KL}, KL divergence) into the loss function. D_{KL} between two probability distributions measures how much they diverge from each other.
  - D_{KL} can be interpreted as the expected number of extra bits per message needed to encode events drawn from true distribution p, if using an optimal code for distribution q rather than p.

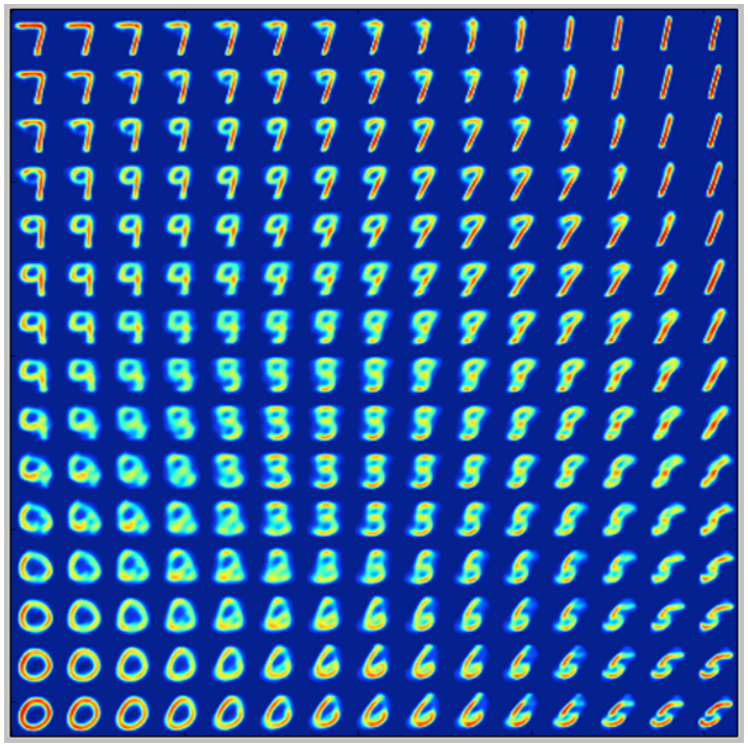
$$D_{KL}(p \parallel q) = \underbrace{H(p,q)}_{\text{cross-entropy}} - \underbrace{H(p)}_{\text{entropy}}$$

- In VAEs, the minimized $D_{KL}(\mathcal{N}(\mu,\sigma) \parallel \mathcal{N}(0,1))$ refers to the distributions of the points in the latent space (sampled according to $\mathcal{N}(\mu,\sigma)$) and the target distribution taken as $\mathcal{N}(0,1)$ in order to get a "good" latent space.
  - A set of points clustered apart and away from the origin is penalized according to this loss.
  - Comparing two gaussian distributions through is D_{KL} is trivial because of a closed analytical form
- Keeping only KL divergence loss leads to not clusterized gaussian latent space (left)
- Re-adding a standard reconstruction loss (e.g. cross-entropy) results into the wanted result (right)

Variational autoencoders (ideas) - 6

Variational autoencoders (ideas) - 7

- Because VAE is a generative model, we can also use it to generate new digits scanning the latent plane and sampling latent points at regular intervals generating the corresponding digit for each of these points
- This results into a visualization of the latent manifold that “generates” the MNIST digits.



Variational autoencoders (ideas) - 8

- When randomly generating (sampling a vector from the same prior distribution of the encoded vectors, $\mathcal{N}(0, 1)$), the decoder will successfully decode it.
  - when interpolating, there are no sudden gaps between clusters, but a smooth mix of features a decoder can understand.
- new sample halfway between two inputs (classical and inputs)? Average the μ of the starting samples
- add specific feature to input (glasses to face)? Use the vector of difference between the code of an image with glasses and an image without glasses

Outline

1 Autoencoders
2 Best practices
3 Large-Scale Deep Learning
4 DAVIDE
5 Jupyter on DAVIDE
6 GAN
Objective
Algorithm
7 References

Machine learning way of doing

- Many choices to do: algorithm, metrics, data, model, regularizations, optimization, debugging, ...
- Knowing a wide variety of machine learning techniques and being good at different kinds of math seems the most significant skills but...
- ...actually one can usually do much better with a correct application of a commonplace algorithm than by sloppily applying an obscure algorithm.
- Practical design process
  - determine your goals (metrics and target value) according to the application
  - prepare a first end-to-end pipeline
  - instrument to detect bottlenecks (causes of overfitting and underfitting)
  - make incremental changes: gather new data, adjust hyperparameters, change algorithms

Performance metrics

- For most applications it impossible to achieve absolute zero error
  - your input features may not contain complete information about the output variable
  - the system might be intrinsically stochastic
  - your training data amoung is limited: data collection (and cleaning) is expensive
- For realistic cases, the target level of performance must ensure the application to be safe, cost-effective or appealing
- Which metrics? When accuracy is not suitable, precision and recall can help
  - unfortunately they usually have opposite behaviors when changing evaluation parameters (e.g thresholds)
  - combined metrics exist (e.g. F-score = $\frac{2pr}{p+r}$)
  - when trying to use the model to predict results, for some inputs we can decide to return no outputs and let the human decide the results
  - the corresponding metrics is called *coverage*
- Selecting metrics and target value is crucial to start

Baseline choice

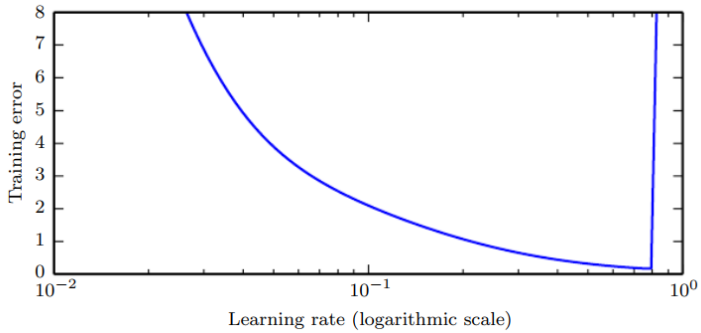
- First: try to guess if deep learning is what you really need, e.g. there is no other (simple) algorithm performing well for your case
- If your problem is an “AI-complete” problem – object recognitions, speech recognition, machine translation – the deep learning model is potentially an adequate choice
- Some rules of the thumb
  - Supervised learning with fixed-size vectors as input: feedforward network with fully connected layers
  - If input is an image: convolutional network
  - Activation function: start with ReLU, Leaky ReLUs, ...
  - If input or output is a sequence use a gated recurrent net (LSTM or GRU based)
  - Optimization algorithm: SGD with momentum with a decaying learning rate is a good start
  - Adam or Adadelta good choices too
  - Batch normalization is a good way if optimization is needed
  - If data are not enough (let’s say hundred or thousands) include regularization
  - Early stopping is standard almost everywhere
  - Dropout is a good regularizer (but batch normalization can be enough)
  - If your task is similar to another task already studied, use that algorithm and possibly start with an existing trained model

Better algorithm or more data?

- If performance on training data is not good, it is not time to gather more data
  - Instead the model is probably oversimplified, try to add neurons
  - ... or improve the learning algorithm (hyperparameters)
- The problem can be also poor quality data: too noisy, or not having all features to predict the output
- Some known behaviours may help in knowing how much data you really need to get your goals
  - If test data performance is much worse than training data one, gathering more data is a good option
- If gathering more data is impossible or too expensive, consider *data augmentation*
  - data augmentation means artificially increasing the number of data points: number of images, objects, rows, ...
  - creation of altered copies of each instance within a training dataset
  - e.g. for images: different orientation, location, scale, brightness, shear
  - CNNs can robustly classify objects even after these corrections

Selecting hyperparameters

- Evaluate the choice according to capacity, computational cost, convergence
- The learning rate (probably the most important one)
  - increasing it leads to smaller training error
  - but above an optimal value there is a sharp rise that can lead to optimization failure



- Important hyperparameters:
  - Number of hidden units
  - Convolutional kernel width
  - Weight decay coefficient
  - Dropout rate
  - Batch size
- Automatic hyperparameter optimization algorithms exist but they still have new hyperparameters
- Otherwise common practices are doing *grid search* or *random search*

Outline

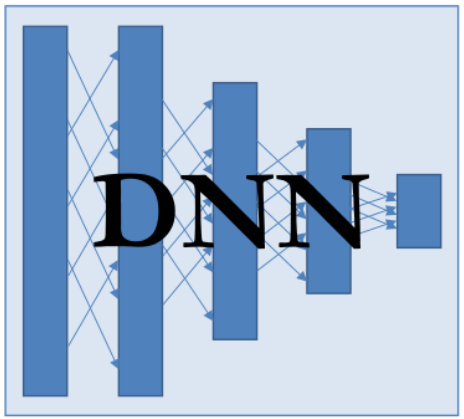
Large-Scale Deep Learning

- "Serious" AI applications require large scale neural network implementation
- ... the number of neurons must be **large**.
- Since the start of the AI era **the size of the neural networks** has hugely increased (exponentially grown for the past three decades) allowing improvement in neural network's accuracy and complexity of tasks they can solve
- ... but artificial neural networks are still only as large as the nervous systems of insects.
- Deep Learning requires **high performance hardware and software infrastrucure**.

ML hardware

What does a modern machine learning pipeline look like?

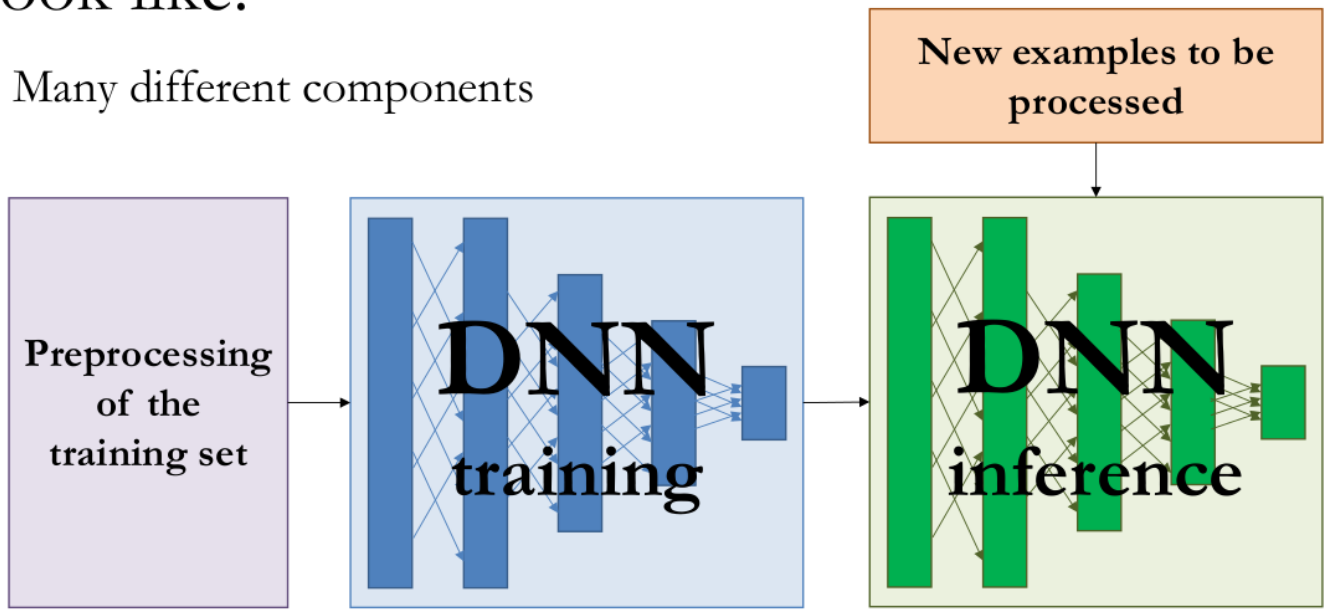
- Not just a neural network:



ML hardware

What does a modern machine learning pipeline look like?

- Many different components



ML hardware

- Hardware can help everywhere in the pipeline
  - both adapting existing hardware architectures
  - and developing new ones
- What possible improvements?
  - Lower latency inference
  - Higher throughput training
  - Lower power cost
- How?
  - speed up the basic building blocks of ML computations: matrix-matrix multiply, convolution
  - add data/memory paths specialized to machine learning workloads (e.g.: having a local cache to store network weights)
  - create application-specific functional units

Fast CPU Implementations

- CPUs of many machines networked together
- careful implementation for specific CPU families can yield large improvements
- for instance, using fixed-point arithmetic rather than floating-point
- ... but new models of CPU have different performance features: floating-point arithmetic can be faster too
- or for instance optimizing data structures to avoid cache misses and using vector instructions
- (see TensorFlow on Marconi Knights Landing partition)

GPU Implementations

- Most modern NN implementations are based on GPUs
- GPUs: specialized hardware components originally developed for graphics applications
- Namely: need to perform many operations in parallel (matrix multiplications and divisions on many vertices to convert 3D coordinates of vertices into 2D on-screen coordinates, and then many computations at each pixel to determine the color of each pixel).
- Fairly simple computations (no much branching with respect to typical CPU workloads, and entirely independent computations), involving the processing of massive buffers of memory containing bitmaps describing the texture (color pattern) of each object to be rendered.
- Consequence: GPUs designed to have a high degree of parallelism and high memory bandwidth at the cost of having a lower clock speed and less branching capability (w.r.t. traditional CPUs)

GPU Implementations

- The performance characteristics needed for good video gaming systems turn out to be beneficial for neural networks as well:
  - NNs usually involve large buffers (of parameters, activation values, and gradient values, each of which must be completely updated during every step of training) falling outside the cache of a traditional computer, so the memory bandwidth becomes the rate limiting factor.
  - NN training algorithms typically do not involve much branching or control (appropriate to GPU hardware)
  - NNs can be divided into multiple individual "neurons" that can be processed independently one for the other neurons in the same layer (hence, highly benefit from the parallelism provided by the GPUs)
- NOTE THAT: GPU hardware was originally so specialized that it could only be used for graphics tasks (GPUs were configurable but not programmable). The popularity of GPUs for NN training exploded after the advent of general purpose GPUs (GP-GPUs), capable to execute arbitrary code (not just rendering subroutines). NVIDIA CUDA programming language provided a way to write an arbitrary code in a C-like language (and Fortran-like, see PGI and IBM XL compilers extentions).

GPU Implementations

- **CUDA programming model** +
- **massive parallelism** +
- **high memory bandwidth** =

make GP-GPUs an ideal platform for NN programming, rapidly adopted by deep learning researchers. But:

- but writing eficient code for GP-GPUs remains a difficult task
- rely on existing libraries of high performance operations like convolution and matrix multiplication, and specify new models or algorithms in terms of calls to those libraries of operations
- Pylearn2 (Goodfellow et al, 2013): specifies all of its ML alhorithms in terms of calls to Theano and cuda-convnet (Theano can run on CPU and GPU, no need to change the calls to Theano)
- TensorFlow, Torch

GPU vs CPU Implementations

- CPU is a general purpose processor
  - Modern CPUs spend most of their area on deep caches
  - This makes the CPU a great choice for applications with random or non-uniform memory accesses
- GPU is optimized for
  - more compute intensive workloads
  - streaming memory models

Do machine learning applications look more like this?

- GPUs have higher memory bandwidths than CPUs
  - e.g. new NVIDIA Tesla V100 has a claimed 900 GB/s memory bandwidth
  - Wheras Intel Xeon E7 has only about 100 GB/s memory bandwidth
- but:
  - GPU memory bandwidth is the bandwidth to GPU memory
  - e.g. on a PCIE2, bandwidth is only 32 GB/s for a GPU (Intel systems). NVLink is quite better.

GPU competitors

- More compute-intensive CPUs (like Intel's Phi line: promise same level of compute performance and better handling of sparsity)
- Low-power devices (like mobile-device-targeted chips, configurable hardware like FPGAs and CGRAs)
- Accelerators that speed up matrix-matrix multiply (like Google's TPU)

But: will all DL computation become dense matrix-matrix multiply? new models are being developed every day -> possible new highly specialized hardware

Large-Scale Distributed Implementations

Distribute the workload of training and inference across many machines (one is not enough!).

- **model parallelism**: multiple machines work on a single datapoint, with each machine running a different part of the model (both for inference and training)
- **data parallelism**: distributing **inference** is simple: each input example can be run by a separate machine. Data parallelism during **training** is harder

Large-Scale Distributed Implementations

- Data parallelism during **training** is harder:
  - increase the size of the minibatch used for a single SGD step -> but usually less than linear returns in terms of optimization performance
  - better to allow multiple machines to compute multiple gradient descent steps in parallel -> but the standard definition of gradient descent is a completely sequential algorithm (the gradient at step t is a function of the parameters produced by step t-1).
- This can be solved using asynchronous stocastic gradient descent (cores share the memory with the parameters; each core reads parameters without a lock, computes a gradient, and increments the parameters without a lock. Some of the cores overwrite each other's progress, but the increased rate of production of steps causes the learning process to be faster overall).
- multi-machine implementation of lock-free approach to gradient descent, where the parameter are managed by a parameter server rather than stored in shared memory. Distributed asynchronous gradient descent is an important strategy for training large deep networks.

Outline

D.A.V.I.D.E. Supercomputer

- **D**evelopment of an **A**dded **V**alue **I**nfrastructure **D**esigned in **E**urope
- Derived from the IBM POWER8 System S822LC (codename Minsky).
- 2 IBM POWER8 NVlink and 4 NVIDIA Tesla P100 HSXM2 with the intra node communication layout optimized for best performance.
- While the original design of the Minsky server is air cooled, its implementation for DAVIDE uses direct liquid cooling for CPUs and GPUs.
- Each compute node has a peak performance of 22 TFLOPS and a power consumption of less than 2kW.
- 2 Infiniband switch with 36 QFSP (100Gb/s) EDR ports

Total number of nodes	45 (compute) + 2 (login)
Form factor	2U
SoC	2xPOWER8 NVlink
GPU	4xNVIDIA Tesla P100 HSMX2
Network	2xIB EDR, 1x 1GbE
Cooling	SoC and GPU with direct hot water
Max performance Xnode	22 TFlops
Storage	1xSSD SATA, 1x NVMe

D.A.V.I.D.E. Supercomputer

E4 COMPUTER ENGINEERING

THE COMPANY

PCP PHASE III – D.A.V.I.D.E. SUPERCOMPUTER

OCP form-factor compute node

4x NVIDIA Tesla P100 HSMX2

2 x IBM POWER8 with NVLink

Liquid cooling components

2xIB EDR

PRACE

D.A.V.I.D.E. Supercomputer

E4 COMPUTER ENGINEERING

PRACE PCP
PHASE III
D.A.V.I.D.E.
SUPERCOMPUTER
(**D**evelopment of an
Added
Value
Infrastructure
Designed in
Europe)

PRACE

CINECA

GPUs on DAVIDE

Tesla P100 HSXM2 (Pascal):

- 5.3 TFLOPS of double precision floating point (FP64) performance
- 10.6 TFLOPS of single precision (FP32) performance
- 21.2 TFLOPS of half-precision (FP16) performance

`deviceQuery:`

CUDA Driver Version / Runtime Version	9.2 / 9.2
CUDA Capability Major/Minor version number:	6.0
Total amount of global memory:	16281 MBytes (17071734784 bytes)
(56) Multiprocessors, (64) CUDA Cores/MP:	3584 CUDA Cores
GPU Max Clock rate:	1481 MHz (1.48 GHz)
Memory Clock rate:	715 Mhz
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
Max dimension size of a thread block (x,y,z):	(1024, 1024, 64)
Max dimension size of a grid size (x,y,z):	(2147483647, 65535, 65535)

GPUs on DAVIDE

NVlink bus:

- NVIDIA new High-Speed Signaling interconnect (NVHS).
- NVHS transmits data over a differential pair running at up to 20 Gb/sec.
- Eight of these differential connections form a Sub-Link that sends data in one direction, and two sub-links-one for each direction-form a Link that connects two processors (GPU-to-GPU or GPU-to-CPU).
- A single Link supports up to 40 GB/sec of bidirectional bandwidth between the endpoints.
- Multiple Links can be combined to form Gangs for even higher-bandwidth connectivity between processors.
- The NVLink implementation in Tesla P100 supports up to four Links, enabling ganged configurations with aggregate maximum bidirectional bandwidth of 160 GB/sec.

Production environment on DAVIDE: module

- Module environment: the installed (non-system) softwares are accessible by loading the corresponding module (a simple set of variables)
- The software modules are collected in different profiles and organized by functional category (compilers, libraries, tools, applications, data, ...).
- What are the available modules (in the default profile/base)?

```
> module av
```

- How do I look for a specific module (e.g., tensorflow)?

```
> module av tensorflow
------------ /cineca/prod/opt/modulefiles/profiles ------------
------------ /cineca/prod/opt/modulefiles/base/data ------------
------------ /cineca/prod/opt/modulefiles/base/environment ------------
------------ /cineca/prod/opt/modulefiles/base/libraries ------------
tensorflow/1.10.1--python--3.6.5 tensorflow/1.9.0--python--3.6.5
------------ /cineca/prod/opt/modulefiles/base/compilers ------------
------------ /cineca/prod/opt/modulefiles/base/tools ------------
------------ /cineca/prod/opt/modulefiles/base/applications ------------
```

- How do I load a specific module?

```
> module load \textcolor{red}{autoload} tensorflow/1.10.1--python--3.6.5
```

- How do I unload a specific module?

```
> module unload tensorflow/1.10.1--python--2.6.5
```

- How do I purge the environment (unload all loaded modules)?

```
> module purge
```

Production environment on DAVIDE: module

- What does the loading of a module mean?

```
> module show tensorflow/1.10.1–python–3.6.5
———————————————————————————————————————
/cineca/prod/opt/modulefiles/base/libraries/tensorflow/1.10.1–python–3.6.5:
prereq python/3.6.5
prereq cuda/9.2.88
prereq nccl/2.3.4–cuda–9.2.88
prereq cudnn/7.1.4–cuda–9.2.88
prereq gnu/6.4.0
prereq openmpi/3.1.0–gnu–6.4.0
prereq szip/2.1.1–gnu–6.4.0
prereq hdf5/1.10.2–gnu–6.4.0
conflict tensorflow
setenv TENSORFLOW_HOME /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5
setenv TENSORFLOW_LIB /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/lib
setenv TENSORFLOW_INC /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/include
setenv TENSORFLOW_INCLUDE /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/include
prepend-path PATH /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/bin :
prepend-path LIBPATH /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/lib :
prepend-path PYTHONPATH /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/lib/python3.6/site-packages :
prepend-path LD_LIBRARY_PATH /cineca/prod/opt/libraries/tensorflow/1.10.1/python–3.6.5/lib :
module-whatis An open source machine learning framework for everyone
———————————————————————————————————————
```

Production environment on DAVIDE: slurm

- SLURM ("Simple Linux Utility for Resource Management") Workload Manager:
  - allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time, so they can perform their work.
  - provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes.
  - arbitrates contention for resources by managing the queue of pending jobs.
- Batch/interactive jobs:
  - write a batch script and submit it to the queue:

```
> sbatch script.sh
```

  - request the resources and start a shell on the master node:

```
> srun <options> --pty bash
```

  - request the resources and start a shell on the login node:

```
> salloc <options>
```

Production environment on DAVIDE: slurm

- A batch script has three main sections:
  1. shell interpreter invocation
  2. SLURM directives (#SBATCH ....)
  3. the script body (setting of variables, launch of the application etc.)

```
#!/bin/bash

#SBATCH --nodes=1                       # 1 node
#SBATCH --ntasks-per-node=8             # 36 tasks per node
#SBATCH --time=1:00:00                  # time limits: 1 hour
#SBATCH --gres=gpu:2                    # requested GPUs
#SBATCH --account=<account_no>          # account name
#SBATCH --partition=<partition_name> # partition name
#SBATCH --qos=<qos_name>                # quality of service

srun ./my_application
```

- Interactive jobs:

```
> srun -N 1 -n 8 -t 1:00:00 --gres=gpu:2 -A <> -p <> --pty bash
# -> shell on the compute node

> salloc -N 1 -n 8 -t 1:00:00 --gres=gpu:2 -A <> -p <>
# -> shell on the login node
```

- Check the status of jobs:

```
> squeue      # squeue -u $USERNAME
```

Outline

Setup Jupyter on D.A.V.I.D.E. cluster I

- **Strategy**: launch the server from a DAVIDE computing node and connect to it from the local browser using ssh tunnels
- **Detailed steps**: open three terminals **A**, **B**, **C**

Terminal A

A-1 Connect to DAVIDE using your credentials (usernames from a08tra21 to a08tra48)

```
ssh -X a08tra21@login.davide.cineca.it
```

A-2 Submit a job in interactive mode specifying the number of GPUs (1 to 4)

```
srun -N 1 -A train_dlrn2018 --ntasks-per-node=4 --gres=gpu:tesla:1 --pty /bin/bash
```

A-3 Start your notebook server using jupyter

```
module load autoload tensorflow
module load jupyter
unset XDG_RUNTIME_DIR
jupyter notebook --port=9921 --no-browser
```

Setup Jupyter on D.A.V.I.D.E. cluster II

Terminal B

B-1 Connect to DAVIDE using your credentials (usernames from a08tra21 to a08tra48)

```
ssh -X a08tra21@login.davide.cineca.it
```

B-2 Forward local 9921 port to port 9921 to the compute node allocated in step **A-2**

```
ssh -L 9921:localhost:9921 davide42
```

Terminal C

C-1 Bind the local 9921 port to port 9921 on login.davide.cineca.it

```
ssh -L 9921:localhost:9921 a08tra21@login.davide.cineca.it
```

C-2 In your local browser open the url returned by step **A-3**: now you should be connected to the notebook server running on the allocated DAVIDE compute node

Outline

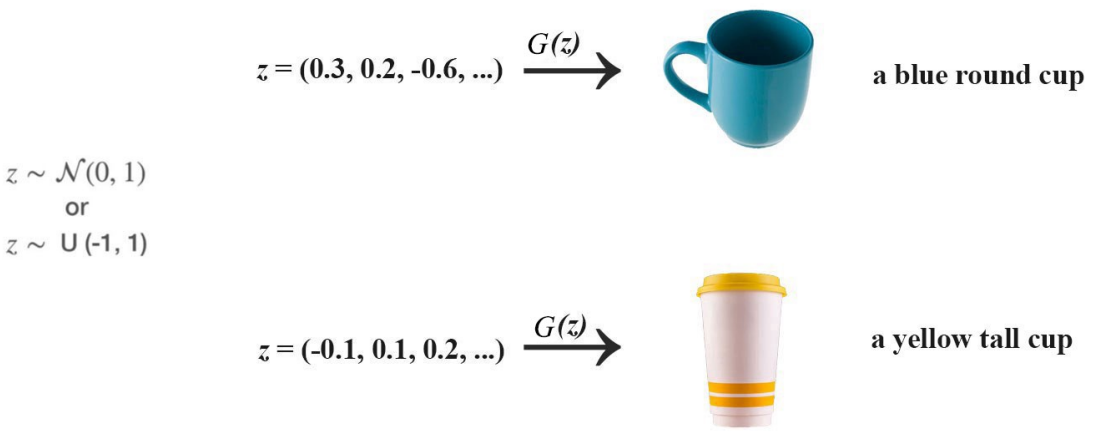
Generative Adversarial Network I

- First introduced by Goodfellow et al. in 2014
- *"Generative Adversarial Networks is the **most interesting idea** in the last ten years in machine learning"*

Yann LeCunn, Director, Facebook AI

Generative Adversarial Network II

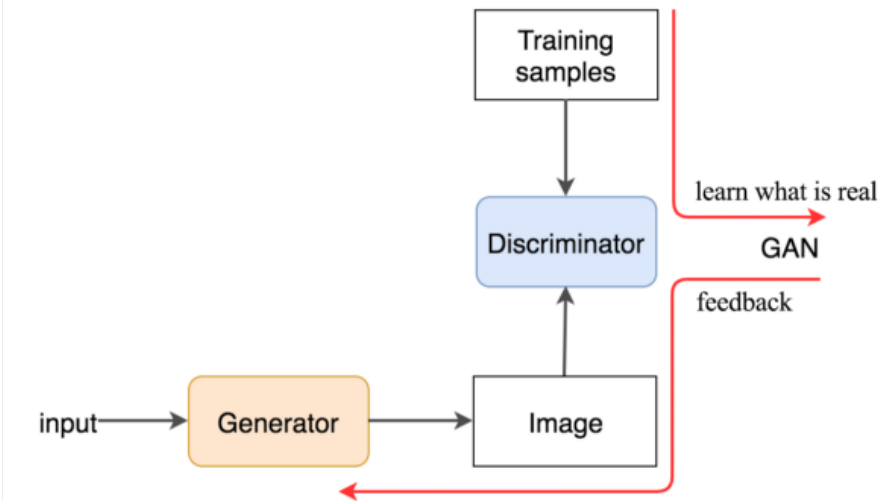
- Is about to creating; generating data from scratch, typically images.
- An advanced topic in deep networks.
- It is composed by two components: the **generator** and the **discriminator**
- The generator $\boldsymbol{x} = \boldsymbol{G}(z)$ is fed with random noise $\boldsymbol{z}$ to produce meaningful data
- $\boldsymbol{z}$ is typically distributed uniformly or according to a gaussian distribution.



- A key concept of the generator is the space of $\boldsymbol{z}$ that is related to the features of the data produced.
- The semantic of the z space , of which dimensions are an hyperparameter, is inaccessible.
- The meaning of $\boldsymbol{z}$ is determined by the training process.

Generative Adversarial Network III

- The discriminator $D(\boldsymbol{x})$ is the block that is fed either with the data produced by the generator and data from a training dataset.
- Its role is to distinguish if the input data is good or is a fake, i.e. comes from training samples or generated by the discriminator.
- Its output is the probability to have good data as input.
- Discriminator information are used to train the generator.

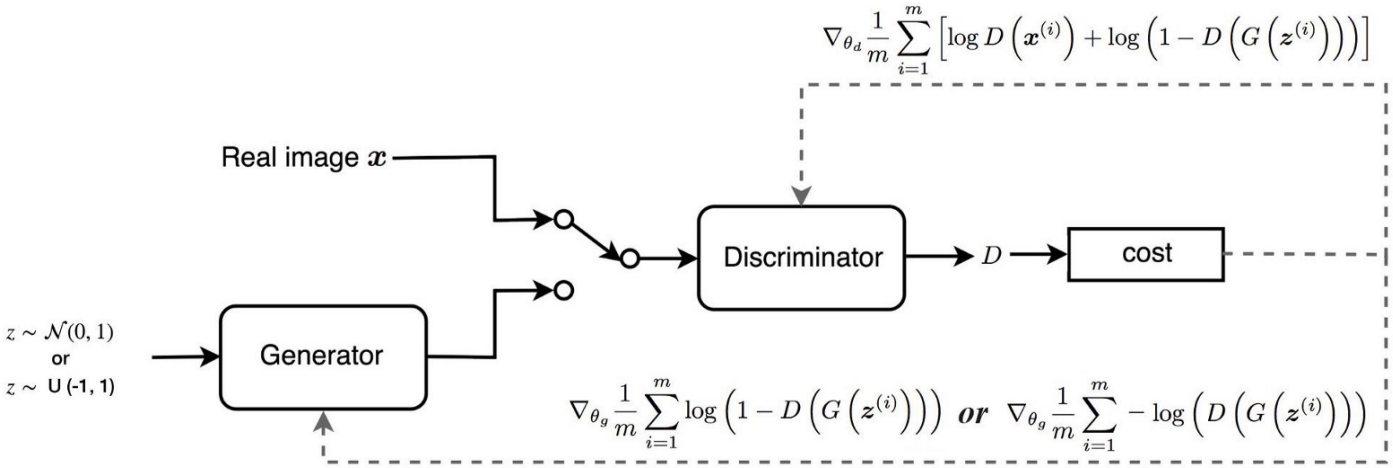


- The discriminator too is to be trained in order to distinguish real vs fake data.

GAN Objective I

- Cross-entropy is used as loss estimator.
- A minimax non-cooperative game with objective function:

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



GAN Objective II

- The discriminator usually wins early against the generator, $log(1 - D(G(\boldsymbol{z}))) \rightarrow 0$ and so the gradient and the learning speed of the generator.
- To overcome this: $\nabla_{\theta_g} log(1 - D(G(\boldsymbol{z}))) \rightarrow \nabla_{\theta_g} log - D(G(\boldsymbol{z}))$

GAN Algorithm I

Minibatch SGD applied to GAN:

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN issues I

- Known issues:
  - Non-convergence: oscillation of parameters
    - Cost functions may not converge using gradient descent in a minimax game.
  - Model collapse: the collapse of the generator produces a limit variety of outputs.
    - If generator converges too early w.r.t the discriminator.
  - Diminished gradient: the discriminator gets too successful that the generator gradient vanishes and learns nothing
  - Hyperparameters sensitivity

GAN issues II

- Lots of GAN flavor exists: DCGAN (Deep Convolutional), SRGAN (Super Resolution), Adversarial Autoencoders, CycleGAN,...

Figure 7: Generated samples

GAN issues III

- Try it out.
- Run notebook gan.ipynb on DAVIDE.

Outline

1 Autoencoders
2 Best practices
3 Large-Scale Deep Learning
4 DAVIDE
5 Jupyter on DAVIDE
6 GAN
Objective
Algorithm
7 References

Further Reading I

- On machine learning and deep learning:
  - **Deep Learning.**
    Ian Goodfellow; Yoshua Bengio; Aaron Courville.
    The MIT Press, 2016.
    Freely available at `https://www.deeplearningbook.org/`
  - **Deep Learning with Python.**
    François Chollet
    Manning Publications Company, 2017.
  - **Hands-On Machine Learning With Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems**
    Aurelien Geron
    O'Reilly, 2017
  - **Learning Deep Architectures for AI**
    Yoshua Bengio Foundations and Trends in Machine Learning, Vol. 2: No. 1, pp 1-127. 2009
  - **Deep Learning**
    Yann LeCun, Yoshua Bengio, Geoffrey Hinton
    Nature 521,436-444, 2015
  - **https://www.deeplearning.ai/**
  - **http://cs231n.stanford.edu/syllabus.html**

Further Reading II

- On GAN:
  - **Generative Adversarial Networks .**
    Ian J. Goodfellow et al.
    Advances in neural information processing systems, pages 2672-2680, 2014.