Optimization of Deep Learning for Pl@ntnet: work more and stock less

Alena Shilova,
with Guillaume Aupy, Olivier Beaumont, Lionel Eyraud-Dubois, Julien Herrmann and Alexis Joly

January 24, 2019

Introduction

- Funded by Inria IPL on convergence between Big Data, HPC and Learning
- IPL gathers researchers from these 3 communities
- HPC for Deep Learning in the context of Pl@ntNet. Work with Alexis Joly (Inria Montpellier), co-supervisor of the PhD
- They have parallel training algorithms already
- Pl@ntNet is complex and big (in terms on nb of species and memory)
- Parallelism and Scheduling for training will be used to go faster and go larger (better model, more species, better accuracy)

Outline

Pl@ntNet

DL Training: Forward and backward propagations, a computational perspective

Where to find parallelism ?

Scheduling re-computations to use less memory

Pl@ntNet

An innovative citizen science platform making use of machine learning to help people identify plants through their mobile phone

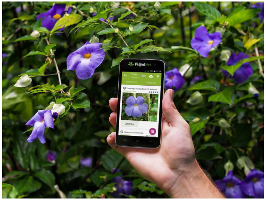
Application

Professional usage

- Agriculture & Agri-food industry
- Education & animation
- Professional botanists, consulting, expertise
- Merchants
- Natural area management
- Tourism

Research Projects

- Invasive species distribution models
- Pl@ntHealth: automated plant epidemiology



Statistics

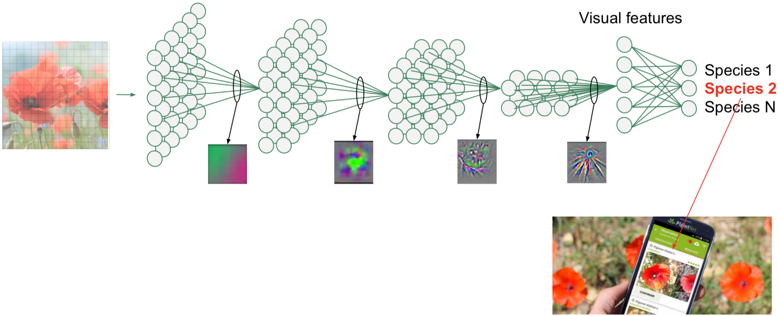
- More than 8M downloads
- Between 60k - 100K users / day
- 11 languages
- 17K species (illustrated by 1M revised images)
- 22 projects & micro-projects
- 35M raw plant images / 55M users sessions
- 12K followers on social networks

In 2018 : 3,352,788 users in 235 countries

Target

- recognize 300K species
- requires richer database and
- more sophisticated models

Pl@ntNet technology

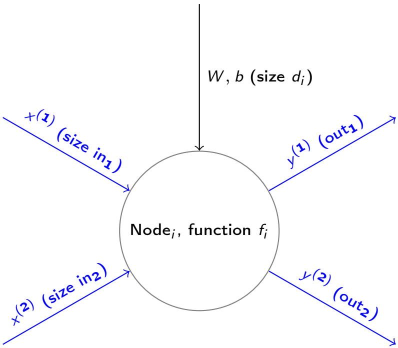


Outline

DL training phase: computational DAG

DNN: a DAG (googlenet)

Node of the DAG



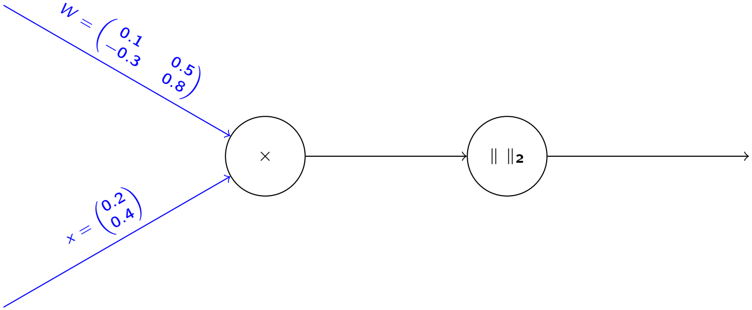
for instance, $f_i = RELU(Wx + b)$

DL: supervised learning process

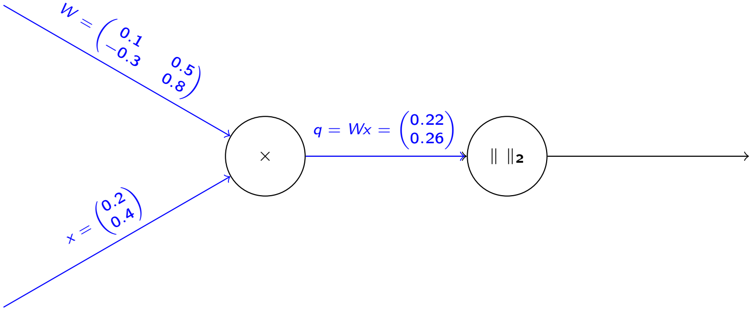
Principe

- Start with an example (x, y)
- Evaluate x using the DAG (as for a classical DAG)
- Evaluate the loss at the end
- Do the backward propagation of the gradient to evaluate the sensitivity of loss to input parameters (almost as for a classical DAG)
- Update the weights ? speed (and more generally optimization theory issues) is out of scope

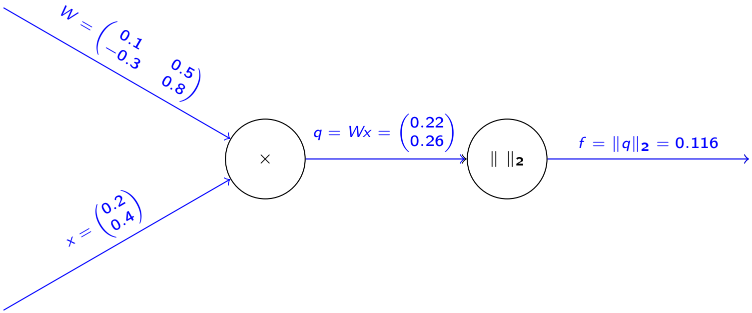
Forward propagation example: $f = \|Wx\|_2$



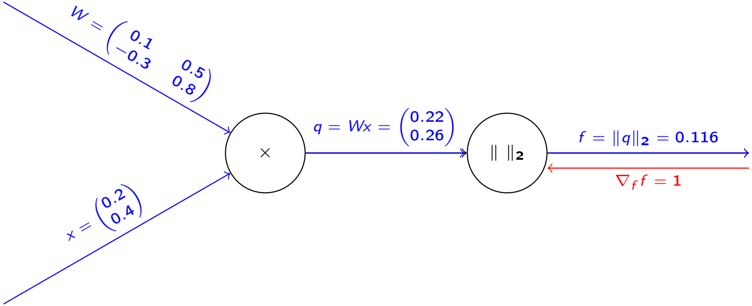
Forward propagation example: $f = \|Wx\|_2$



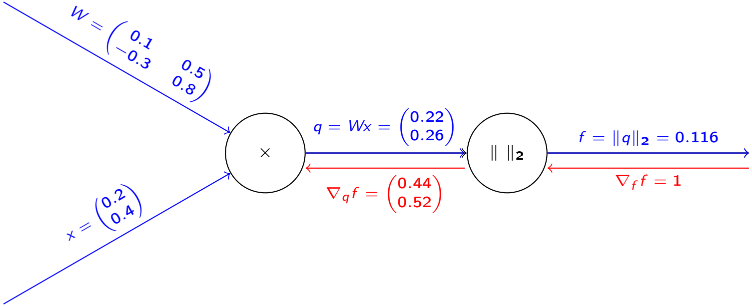
Forward propagation example: $f = \|Wx\|_2$



Backward propagation example $f = \|Wx\|_2$

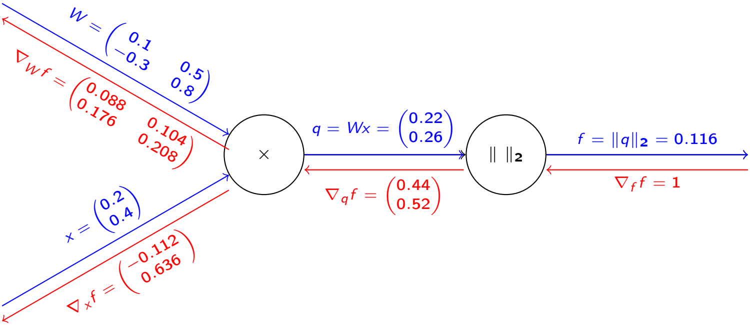


Backward propagation example $f = \|Wx\|_2$



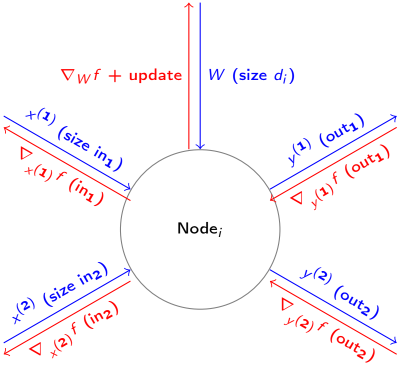
- $f = q_1^2 + q_2^2 \longrightarrow \nabla_q f = 2q$

Backward propagation example $f = \|Wx\|_2$



- $f = q_1^2 + q_2^2 \longrightarrow \nabla_q f = 2q$
- $q = Wx \longrightarrow \frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial q_1}\frac{\partial q_1}{\partial x_i} + \frac{\partial f}{\partial q_2}\frac{\partial q_2}{\partial x_i} = 2q_1 W_{1,1} + 2q_2 W_{2,1}$ et $\nabla_x f = 2W^T q = W^T \nabla_q f$
- $q = Wx \longrightarrow \frac{\partial f}{\partial W_{i,j}} = \frac{\partial f}{\partial q_1}\frac{\partial q_1}{\partial W_{i,j}} + \frac{\partial f}{\partial q_2}\frac{\partial q_2}{\partial W_{i,j}} = q_i x_j$ et $\nabla_W f = 2qx^T = \nabla_q f x^T$

Distributed DL: forward propagation and backward propagation



- $\dfrac{\partial f}{\partial x_i^{(1)}} = \dfrac{\partial f}{\partial y^{(1)}} \dfrac{\partial y^{(1)}}{\partial x_i^{(1)}} + \dfrac{\partial f}{\partial y^{(2)}} \dfrac{\partial y^{(2)}}{\partial x_i^{(1)}}$ $\qquad \dfrac{\partial f}{\partial x_i^{(2)}} = \dfrac{\partial f}{\partial y^{(1)}} \dfrac{\partial y^{(1)}}{\partial x_i^{(2)}} + \dfrac{\partial f}{\partial y^{(2)}} \dfrac{\partial y^{(2)}}{\partial x_i^{(2)}}$
- $\dfrac{\partial f}{\partial W_i} = \dfrac{\partial f}{\partial y^{(1)}} \dfrac{\partial y^{(1)}}{\partial W_i} + \dfrac{\partial f}{\partial y^{(2)}} \dfrac{\partial y^{(2)}}{\partial W_i}$

DL: forward propagation and backward propagation

Important issues with respect to memory usage

- keep W and update it (depends on the layer type)
- receive $x^{(1)}$ et $x^{(2)}$ and keep them until backward propagation
- compute and send $y^{(1)}$ et $y^{(2)}$
- receive $\nabla_{y^{(1)}} f$ and $\nabla_{y^{(2)}} f$ (same size as $y^{(1)}$ et $y^{(2)}$)
- compute and send $\nabla_{x^{(1)}} f$ and $\nabla_{x^{(2)}} f$ (same size $x^{(1)}$ et $x^{(2)}$)
- compute $\nabla_W f$ and update W
- the overall DAG is: forward + loss + backward + extra dependencies:
  - $(x^{(1)}, x^{(2)})$ needed during the backpropagation

Outline

How to find parallelism (I) ? hyper parameters tuning

- There are many hyper parameters to determine...

How to find parallelism (I) ? hyper parameters tuning

- There are many hyper parameters to determine... and in many cases no clear algorithm to do it beyond brute force
- Parallel algorithm
  - try several hyper parameters sets

How to find parallelism (I) ? hyper parameters tuning

- There are many hyper parameters to determine... and in many cases no clear algorithm to do it beyond brute force
- Parallel algorithm
  - try several hyper parameters sets
  - choose the most promising ones

How to find parallelism (I) ? hyper parameters tuning

- There are many hyper parameters to determine... and in many cases no clear algorithm to do it beyond brute force
- Parallel algorithm
  - try several hyper parameters sets
  - choose the most promising ones
  - possibly reallocate resources

How to find parallelism (I) ? hyper parameters tuning

- There are many hyper parameters to determine... and in many cases no clear algorithm to do it beyond brute force
- Parallel algorithm
  - try several hyper parameters sets
  - choose the most promising ones
  - possibly reallocate resources
- Easy way to achieve good parallel scalability
- at least at the beginning, i.e. before hyper parameters are determined.

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.
- Data parallelism (the one used for Pl@ntnet at the moment)

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.
- Data parallelism (the one used for Pl@ntnet at the moment)
  - perform several mini-batches in parallel

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.
- Data parallelism (the one used for Pl@ntnet at the moment)
  - perform several mini-batches in parallel
  - compute $\nabla_W f$ for each mini-batch in parallel

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.
- Data parallelism (the one used for Pl@ntnet at the moment)
  - perform several mini-batches in parallel
  - compute $\nabla_W f$ for each mini-batch in parallel
  - sum the different $\nabla_W f$ using **MPI_Reduce** like algorithm.

How to find parallelism (II) ? data parallelism

- In practice, use of mini-batches
  - aggregate several (x, y) pairs
  - transform vectors into matrices
  - to keep GPUs happy
- In practice, does not affect convergence if mini-batches are small enough.
- Data parallelism (the one used for Pl@ntnet at the moment)
  - perform several mini-batches in parallel
  - compute $\nabla_W f$ for each mini-batch in parallel
  - sum the different $\nabla_W f$ using **MPI_Reduce** like algorithm.
- first drawback:
  - requires to communicate all Ws
  - and is equivalent to use a large mini-batch size
  - and thus can generate convergence issues (increase the number of epochs)

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism
  - but do not help to solve memory issues
  - since the exact same model has to be stored on each node

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism
  - but do not help to solve memory issues
  - since the exact same model has to be stored on each node
- In the context of Pl@ntnet, we need to consider
  - larger models (parametrized models)
  - larger batch sizes on each GPU
- Potential solutions
  1. work more, stock less (the end of this talk)

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism
  - but do not help to solve memory issues
  - since the exact same model has to be stored on each node
- In the context of Pl@ntnet, we need to consider
  - larger models (parametrized models)
  - larger batch sizes on each GPU
- Potential solutions
  1. work more, stock less (the end of this talk)
     - optimal checkpointing strategies
     - to do extra computations to save memory

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism
  - but do not help to solve memory issues
  - since the exact same model has to be stored on each node
- In the context of Pl@ntnet, we need to consider
  - larger models (parametrized models)
  - larger batch sizes on each GPU
- Potential solutions
  1. work more, stock less (the end of this talk)
     - optimal checkpointing strategies
     - to do extra computations to save memory
  2. communicate more, store less (the end of this thesis)

What are the limitations of above approaches in the context of Pl@ntnet ?

- hyper parameter tuning and data parallelism
  - enable to increase parallelism
  - but do not help to solve memory issues
  - since the exact same model has to be stored on each node
- In the context of Pl@ntnet, we need to consider
  - larger models (parametrized models)
  - larger batch sizes on each GPU
- Potential solutions
  1. work more, stock less (the end of this talk)
     - optimal checkpointing strategies
     - to do extra computations to save memory
  2. communicate more, store less (the end of this thesis)
     - use model parallelism
     - split the model across several nodes
     - communicate forward and backward activations between nodes

Outline

Single Adjoint Chain Computation problem

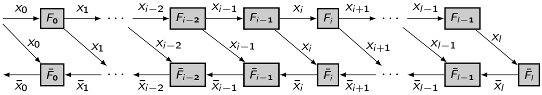


Figure: The data dependencies in the AC chain.

$$
\begin{aligned}
\mathrm{Opt}_0(l,1) &= \frac{l(l+1)}{2}u_f + (l+1)u_b \\
\mathrm{Opt}_0(1,c) &= u_f + 2u_b \\
\mathrm{Opt}_0(l,c_m) &= \min_{1\le i\le l-1}\{iu_f + \mathrm{Opt}_0(l-i,c_m-1) + \mathrm{Opt}_0(i-1,c_m)\}
\end{aligned}
$$

Multiple Adjoint Chain Computation problem

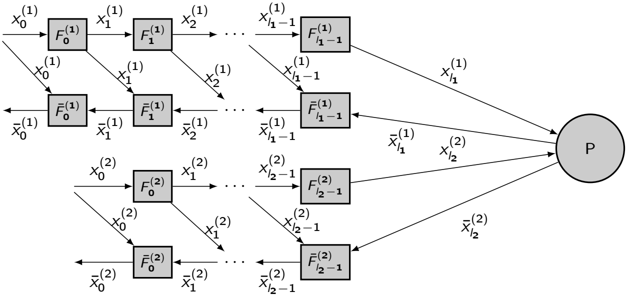


Figure: The data dependencies in the multiple adjoint chain with two branches.

- **A much more complicated Dynamic Programming solves above problem**
- Generalization to trees, series parallel and DAGs are needed (but will be hard)

Conclusion

- DL training phase and Parallelism
  - Memory issues are crucial for Pl@ntNet
  - Scalability is not difficult to achieve
- At the moment, we concentrate on the single node case
  - We implemented optimal checkpointing strategy for homogeneous chains in PyTorch
- Can be combined with model parallelism to further save memory