Statistical Machine Learning

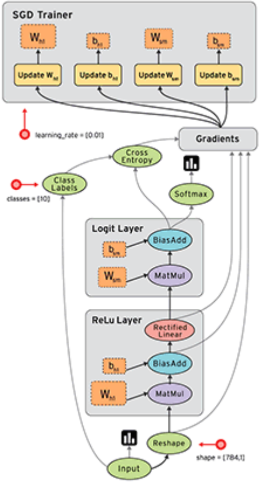
Modern NN Libraries

Workshop 6

August 30, 2019

Static Graphs

- Static Graphs: Represent computation as graph structure.
  - Nodes represent operations (e.g. matrix multiplication).
  - Tensors represent edges and flow between nodes.
- User defines computational graph beforehand, to be executed at later stage.
- This is *metaprogramming* - write a program which represents a program.
- Allows optimization of tensor computation on `C++` backend, distribution over multiple devices.
- Tradeoff between algorithm flexibility with optimization of performance during execution.



https://www.tensorflow.org/guide/graphs

Dynamic Graphs

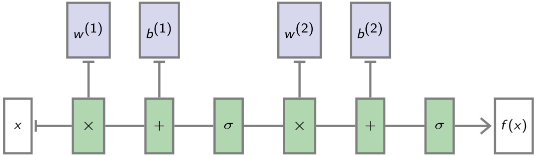
- Necessary computation graph metadata is generated automatically each time an operation is executed.
- Why is this competitive performance-wise? Tensors generally stay on GPU/TPU. Graph metadata is lightweight and cheap to generate.
- Multiple advantages:
  - Allows different computational architecture for each training example/batch - more flexible algorithms, especially for dynamically sized data.
  - Control flow (`if`, `while`) can be implemented in host language.

(Py)Torch

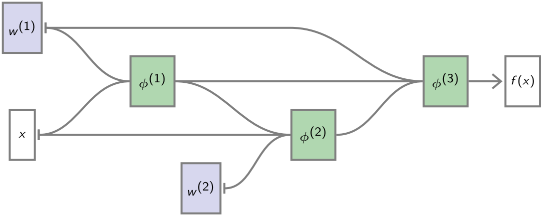
PyTorch

- Python library built on top of `C++` backend of computational library Torch.
- Designed for efficient tensor operations on CPU/GPU.
- In-built automatic differentiation routines (Autograd) allow users to take the gradients of their model with respect to learnable parameters for gradient-based learning.

Autograd



- Any Tensor operation can be represented as a Directed Acyclic Graph (DAG).
  - Nodes are operators.
  - Edges are tensors.



François Fleuret - Autumn 2019 Lecture slides.

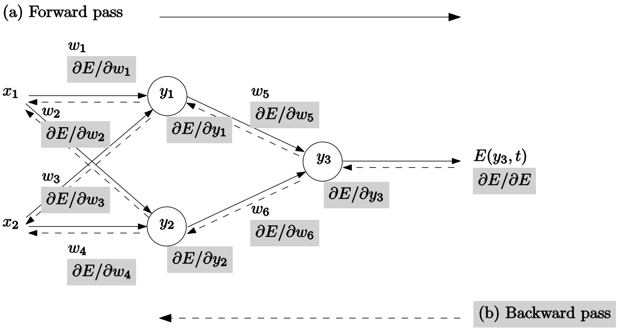
Autograd

- Let the result
$$(u_1, \ldots, u_m) = \Phi(v_1, \ldots v_n)$$
- Torch will automatically generate the DAG to compute the gradient of any scalar element u_k of the result to any involved tensor.
- Useful to minimize loss:
  - Model $\Phi(\mathbf{x}; \mathbf{w})$ accepts input tensors $\mathbf{x}$ and performs arbitrary operations parameterized by $\mathbf{w}$.
  - Minimize some (scalar) loss $E\left(\Phi(\mathbf{x}; \mathbf{w})\right)$
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E\left(\Phi(\mathbf{x}; \mathbf{w})\right)$$

Autograd

- Forward pass: Input tensors x fed forward through model with weights w to generate activations y and final error E.
- Backward pass computes gradient of error with respect to parameters w using algorithm based on chain rule, used to update w using SGD.

$$\nabla_{\mathbf{w}} E = \left(\frac{\partial E}{\partial w_1}, \dots \frac{\partial W}{\partial w_n} \right)$$



Automatic Differentiation in Machine Learning: a Survey

Hands-on

1. Download `worksheet-xx.ipynb` from the LMS.
2. Move the worksheet to a working directory `$WORKDIR`
3. `cd $WORKDIR`
4. Start → Anaconda3 (64-bit) → Anaconda Prompt
5. Enter the following at the prompt:
   `conda install pytorch torchvision cpuonly -c pytorch`
6. Launch Jupyter
   `jupyter notebook`
7. The Jupyter UI should open in a web browser.
8. Click on `worksheet-xx.ipynb` to get started.

You can work on the notebooks at home if you install the Anaconda3 distribution on your machine.