Neural Networks and Deep Learning: Deep Learning Resources

Nicolas Thome

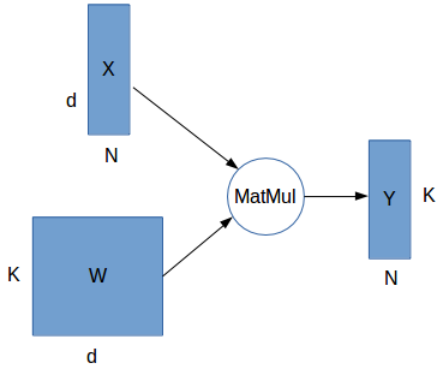
Conservatoire National des Arts et Métiers (Cnam)
Département Informatique

Deep Learning resources for the community

- Deep learning softwares / libraries features:
  - Build big computational graphs
  - Compute gradients in computational graphs
  - GPU Processing
- Libraries made available in the community:
  - MatConvNet (Oxford): easy
  - Caffe (UC Berkeley) / Caffe2 (Facebook): script
    - Good for production
  - Torch (NYU, Facebook) / PyTorch (Facebook)
  - Theano (U Montreal), TensorFlow (Google)
    - (py)Torch, Theano, TensorFlow: good for research
  - Keras (Google): wrapper on top of TensorFlow / Theano
  - Many others...

Computation Graphs on Tensors

- Tensor: multi-dimensionnal array
- Computation Graph: tensor → tensor
- Atomic computation on tensor, *e.g.* matrix multiplication, convolution
  Ex: batch matrix multiplication on vectors

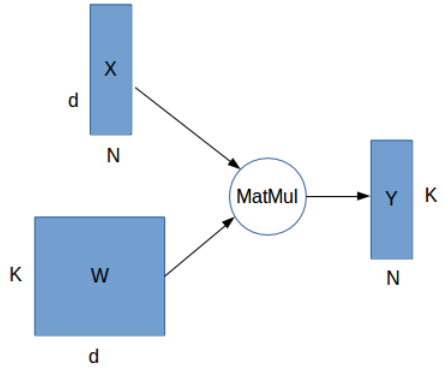


Back-prop on Tensors

- Vector representation of tensor-valued functions
  simply flattening tensor → vector
- Back-prop on tensors?
  ⇒ simply compute derivative wrt
  each flattened tensor element

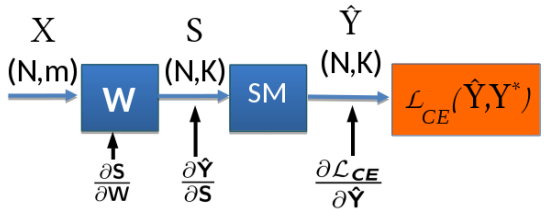
Back-prop on Tensors

- Simply compute derivative wrt flattened tensor
- **BUT:** direct chain rule application
  $\Rightarrow$ big memory & computation issues!
  Problem already with last linear layers



Back-prop on Tensors

- **Example: batch training for last linear layer**
- Data matrix $\mathbf{X}$ $(N \times m)$, label matrix $\hat{\mathbf{Y}}$, $\mathbf{Y}^*$ $(N \times K)$
- Cross-entropy loss: $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$



$\Rightarrow \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$ =? - Ex: $m = 4096$, $N = 100$, $K = 1000$ (ImageNet)

- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$: size $K \cdot d \approx 4M$ params $\approx 32MB$ memory OK

Back-prop on Tensors

$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$ =? - Ex: $m = 4096$, $N = 100$, $K = 1000$ (ImageNet)

- **Chain rule:** $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$
  - $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} = \hat{\mathbf{Y}} - \mathbf{Y}^* = \Delta$: size $K \cdot N = 100K$ params $\approx 800KB$ small
  - **BUT:** $\frac{\partial \mathbf{S}}{\partial \mathbf{W}}$ size $(K \cdot N) \cdot (K \cdot d) = 1000 \cdot 100 \cdot 1000 \cdot 4000$ $= 400G$ params $\approx 3.2TB$ huge !!
  - $\frac{\partial \mathbf{S}}{\partial \mathbf{W}}$: **far too large to fit into memory, not explicitly computable**

$$\frac{\partial \mathbf{S}}{\partial \mathbf{W}} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}_1 & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{x}_1 \\ \dots & \dots & \dots & \dots \\ \mathbf{x}_N & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{x}_N & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{x}_N \end{pmatrix}$$

Back-prop on Tensors

- $\frac{\partial \mathbf{S}}{\partial \mathbf{W}}$: **far too large to fit into memory, not explicitly computable**
- However, computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$ is tractable!
  - For a single example: $\frac{\partial \mathbf{s}}{\partial \mathbf{W}} = \begin{pmatrix} \mathbf{x} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{x} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{x} \end{pmatrix}$, $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}} = \delta = \begin{pmatrix} \delta_1 & \delta_2 & \dots & \delta_K \end{pmatrix}$

$$\Rightarrow \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \begin{pmatrix} \delta_1 & \delta_2 & \dots & \delta_K \end{pmatrix} \begin{pmatrix} \mathbf{x} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{x} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{x} \end{pmatrix} = \begin{pmatrix} \delta_1 \mathbf{x} & \delta_2 \mathbf{x} & \dots & \delta_K \mathbf{x} \end{pmatrix}$$

  - With matrix $m \times K$ reshaping:

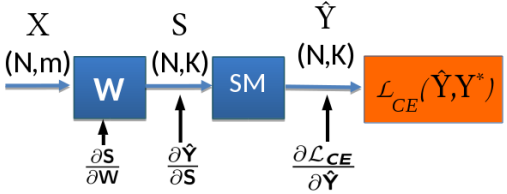
$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \begin{pmatrix} x_1 \\ x_m \\ \dots \\ x_m \end{pmatrix} \begin{pmatrix} \delta_1 & \delta_2 & \dots & \delta_K \end{pmatrix} = \mathbf{x}^T \delta$$

Back-prop on Tensors

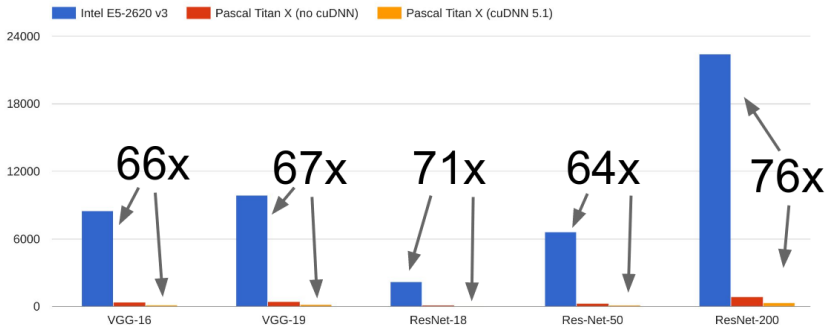
- $\frac{\partial \mathbf{s}}{\partial \mathbf{W}}$ intractable but $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{W}}$ tractable
  - **Solution:** first project $\mathbf{s}$ on δ:
    - $s_p = \mathbf{s}\ \delta^T = \sum_{k=1}^{K} s_k \delta_k = \sum_{k=1}^{K} \sum_{j=1}^{m} w_{jk} x_k \delta_k$
    - Compute gradient on $s_p \in \mathbb{R}$:

$$\frac{\partial s_p}{\partial \mathbf{W}} = \begin{pmatrix} \delta_1 \mathbf{x} & \delta_2 \mathbf{x} & \dots & \delta_K \mathbf{x} \end{pmatrix} = \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$$

  - **Each layer should be able to compute** $\frac{\partial \mathbf{S}_p}{\partial \mathbf{W}}$
    - $\frac{\partial \mathbf{S}_p}{\partial \mathbf{W}} = \frac{\partial \mathbf{S} \Delta^T}{\partial \mathbf{W}}$ tractable $(\neq \frac{\partial \mathbf{S}}{\partial \mathbf{W}})$



Computation on GPU



- Training deep ConvNets: huge speed-up with Graphical Processing Units (GPU)[1]
  - Especially convolution

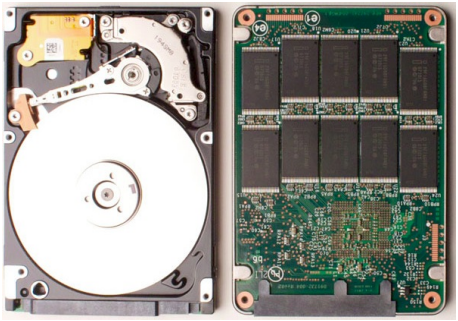
[1] data from `https://github.com/jcjohnson/cnn-benchmarks`

Computation on GPU

- Solution for GPU programming:
  - CUDA (NVIDIA):
    - C-code
    - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
  - OpenCL: ~ CUDA (not limited to NVIDIA): generally slower
- **Deep Learning libraries: transparent GPU computing**

Computation on GPU

- Big data: transfer from disk to memory ⇒ bottleneck
- Solution: use SSD instead of HDD



Deep Learning resources: Conclusion

- Very efficient implementation for computation graphs on multi-dimensional tensors
- Easy to use, GPU support
- **Do not re-invent the wheel: use them!**
- **Example with Keras**
  ⇒ following!