AMMI – Introduction to Deep Learning

1.2. Current applications and success

François Fleuret
https://fleuret.org/ammi-2018/
Sat Oct 6 18:43:49 CAT 2018
Object detection and segmentation

(Pinheiro et al., 2016)
Human pose estimation

(Wei et al., 2016)
Image generation

(Radford et al., 2015)
Reinforcement learning

Self-trained, plays 49 games at human level.

(Mnih et al., 2015)
Strategy games

March 2016, 4-1 against a 9-dan professional without handicap.

(Silver et al., 2016)
The reason Boeing are doing this is to cram more seats in to make their plane more competitive with our products,” said Kevin Keniston, head of passenger comfort at Europe’s Airbus.

When asked about this, an official of the American administration replied: “The United States is not conducting electronic surveillance aimed at offices of the World Bank and IMF in Washington.”

(Wu et al., 2016)
Auto-captioning

Figure 5. A selection of evaluation results, grouped by human rating.

4.3.7 Analysis of Embeddings
In order to represent the previous word $S_{t-1}$ as input to the decoding LSTM producing $S_t$, we use word embedding vectors \[22\], which have the advantage of being independent of the size of the dictionary (contrary to a simpler one-hot-encoding approach). Furthermore, these word embeddings can be jointly trained with the rest of the model. It is remarkable to see how the learned representations have captured some semantic from the statistics of the language.

Table 4.3.7 shows, for a few example words, the nearest other words found in the learned embedding space.

Note how some of the relationships learned by the model will help the vision component. Indeed, having "horse", "pony", and "donkey" close to each other will encourage the CNN to extract features that are relevant to horse-looking animals. We hypothesize that, in the extreme case where we see very few examples of a class (e.g., "unicorn"), its proximity to other word embeddings (e.g., "horse") should provide a lot more information that would be completely lost with more traditional bag-of-words based approaches.

5. Conclusion
We have presented NIC, an end-to-end neural network system that can automatically view an image and generate a reasonable description in plain English. NIC is based on a convolution neural network that encodes an image into a compact representation, followed by a recurrent neural network that generates a corresponding sentence. The model is trained to maximize the likelihood of the sentence given the image. Experiments on several datasets show the robustness of NIC in terms of qualitative results (the generated sentences are very reasonable) and quantitative evaluations, using either ranking metrics or BLEU, a metric used in machine translation to evaluate the quality of generated sentences. It is clear from these experiments that, as the size of the available datasets for image description increases, so will the performance of approaches like NIC. Furthermore, it will be interesting to see how one can use unsupervised data, both from images alone and text alone, to improve image description approaches.

(Vinyals et al., 2015)
Question answering

I: Jane went to the hallway.
I: Mary walked to the bathroom.
I: Sandra went to the garden.
I: Daniel went back to the garden.
I: Sandra took the milk there.
Q: Where is the milk?
A: garden

I: It started boring, but then it got interesting.
Q: What’s the sentiment?
A: positive

(Kumar et al., 2015)
Why does it work now?
The success of deep learning is multi-factorial:

- Five decades of research in machine learning,
- CPUs/GPUs/storage developed for other purposes,
- lots of data from “the internet”,
- tools and culture of collaborative and reproducible science,
- resources and efforts from large corporations.
Five decades of research in ML provided

- a taxonomy of ML concepts (classification, generative models, clustering, kernels, linear embeddings, etc.),
- a sound statistical formalization (Bayesian estimation, PAC),
- a clear picture of fundamental issues (bias/variance dilemma, VC dimension, generalization bounds, etc.),
- a good understanding of optimization issues,
- efficient large-scale algorithms.
From a practical perspective, deep learning

- lessens the need for a deep mathematical grasp,
- makes the design of large learning architectures a system/software development task,
- allows to leverage modern hardware (clusters of GPUs),
- does not plateau when using more data,
- makes large trained networks a commodity.
Francois Fleuret

(Wikipedia “FLOPS”)

<table>
<thead>
<tr>
<th>TFlops ($10^{12}$)</th>
<th>Price</th>
<th>GFlops per $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i7-6700K</td>
<td>0.2</td>
<td>$344</td>
</tr>
<tr>
<td>AMD Radeon R-7 240</td>
<td>0.5</td>
<td>$55</td>
</tr>
<tr>
<td>NVIDIA GTX 750 Ti</td>
<td>1.3</td>
<td>$105</td>
</tr>
<tr>
<td>AMD RX 480</td>
<td>5.2</td>
<td>$239</td>
</tr>
<tr>
<td>NVIDIA GTX 1080</td>
<td>8.9</td>
<td>$699</td>
</tr>
</tbody>
</table>
The typical cost of a 4Tb hard disk is $120 (Dec 2016).
Figure 1: Top-1 vs. network. Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share colour, for example ResNet are all variations of pink.

Figure 2: Top-1 vs. operations, size $\propto$ parameters. Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from $5 \times 10^6$ to $155 \times 10^6$ params.

Figure 3: Inference time vs. batch size. These two charts show inference time across different batch sizes with a linear and logarithmic ordinate respectively and logarithmic abscissa. Missing data points are due to lack of enough system memory required to process bigger batches.

3.2 Inference Time
Figure 3 reports inference time per image on each architecture, as a function of image batch size (from 1 to 64). We notice that VGG processes one image in more than half second, making it a less likely contender in real-time applications on a NVIDIA TX1. AlexNet shows a speed up of roughly $15 \times$ going from batch of 1 to 64 images, due to weak optimisation of its fully connected layers. It is a very surprising finding, that will be further discussed in the next subsection.

3.3 Power
Power measurements are complicated by the high frequency swings in current consumption, which required high sampling current read-out to avoid aliasing. In this work, we used a 200 MHz digital oscilloscope with a current probe, as reported in section 2. Other measuring instruments, such as an AC power strip with 2 Hz sampling rate, or a GPIB controlled DC power supply with 12 Hz sampling rate, did not provide enough bandwidth to properly conduct power measurements.
<table>
<thead>
<tr>
<th>Data-set</th>
<th>Year</th>
<th>Nb. images</th>
<th>Resolution</th>
<th>Nb. classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1998</td>
<td>$6.0 \times 10^4$</td>
<td>$28 \times 28$</td>
<td>10</td>
</tr>
<tr>
<td>NORB</td>
<td>2004</td>
<td>$4.8 \times 10^4$</td>
<td>$96 \times 96$</td>
<td>5</td>
</tr>
<tr>
<td>Caltech 101</td>
<td>2003</td>
<td>$9.1 \times 10^3$</td>
<td>$\approx 300 \times 200$</td>
<td>101</td>
</tr>
<tr>
<td>Caltech 256</td>
<td>2007</td>
<td>$3.0 \times 10^4$</td>
<td>$\approx 640 \times 480$</td>
<td>256</td>
</tr>
<tr>
<td>LFW</td>
<td>2007</td>
<td>$1.3 \times 10^4$</td>
<td>$250 \times 250$</td>
<td>–</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>2009</td>
<td>$6.0 \times 10^4$</td>
<td>$32 \times 32$</td>
<td>10</td>
</tr>
<tr>
<td>PASCAL VOC</td>
<td>2012</td>
<td>$2.1 \times 10^4$</td>
<td>$\approx 500 \times 400$</td>
<td>20</td>
</tr>
<tr>
<td>MS-COCO</td>
<td>2015</td>
<td>$2.0 \times 10^5$</td>
<td>$\approx 640 \times 480$</td>
<td>91</td>
</tr>
<tr>
<td>ImageNet</td>
<td>2016</td>
<td>$14.2 \times 10^6$</td>
<td>$\approx 500 \times 400$</td>
<td>21,841</td>
</tr>
<tr>
<td>Cityscape</td>
<td>2016</td>
<td>$25 \times 10^3$</td>
<td>$2,000 \times 1000$</td>
<td>30</td>
</tr>
</tbody>
</table>
“Quantity has a Quality All Its Own.”

(Thomas A. Callaghan Jr.)
Implementing a deep network, PyTorch
Deep-learning development is usually done in a framework:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Language(s)</th>
<th>License</th>
<th>Main backer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyTorch</td>
<td>Python</td>
<td>BSD</td>
<td>Facebook</td>
</tr>
<tr>
<td>Caffe2</td>
<td>C++, Python</td>
<td>Apache</td>
<td>Facebook</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Python, C++</td>
<td>Apache</td>
<td>Google</td>
</tr>
<tr>
<td>MXNet</td>
<td>Python, C++, R, Scala</td>
<td>Apache</td>
<td>Amazon</td>
</tr>
<tr>
<td>CNTK</td>
<td>Python, C++</td>
<td>MIT</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Torch</td>
<td>Lua</td>
<td>BSD</td>
<td>Facebook</td>
</tr>
<tr>
<td>Theano</td>
<td>Python</td>
<td>BSD</td>
<td>U. of Montreal</td>
</tr>
<tr>
<td>Caffe</td>
<td>C++</td>
<td>BSD 2 clauses</td>
<td>U. of CA, Berkeley</td>
</tr>
</tbody>
</table>

A fast, low-level, compiled backend to access computation devices, combined with a slow, high-level, interpreted language.
We will use the PyTorch framework for our experiments.

PyTorch

http://pytorch.org

"PyTorch is a python package that provides two high-level features:

- Tensor computation (like numpy) with strong GPU acceleration
- Deep Neural Networks built on a tape-based autograd system

You can reuse your favorite python packages such as numpy, scipy and Cython to extend PyTorch when needed."
MNIST data-set

28 × 28 grayscale images, 60k train samples, 10k test samples.

(leCun et al., 1998)
```python
def model:
    nn.Sequential(
    nn.Conv2d(1, 32, 5), nn.MaxPool2d(3), nn.ReLU(),
    nn.Conv2d(32, 64, 5), nn.MaxPool2d(2), nn.ReLU(),
    Flattener(),
    nn.Linear(256, 200), nn.ReLU(),
    nn.Linear(200, 10)
)

nb_epochs, batch_size = 10, 100
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)

model.cuda()
criterion.cuda()
train_input, train_target = train_input.cuda(), train_target.cuda()

mu, std = train_input.mean(), train_input.std()
train_input.sub_(mu).div_(std)

for e in range(nb_epochs):
    for input, target in zip(train_input.split(batch_size),
                              train_target.split(batch_size)):
        output = model(input)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

≈7s on a GTX1080, ≈1% test error
```
The end
References


