Fastai Lesson 11 Review

TWiML Fastai Meetup
Sat. 13 Apr 2019
Joseph Catanzarite

Agenda

- **Lesson 11 review 9:00 – 10:00**
- **Mini-presentation 10:00 – 10:15**
- **Open mic / chat 10:15 – 10:30**

Topics covered in Lesson #11

Notebooks covered in Lesson 11

07a_lsuv.ipynb
08_data_block.ipynb
09_optimizers.ipynb
09b_learner.ipynb
09c_add_progress_bar.ipynb
10_transforms.ipynb

Layerwise Sequential Unit Variance (LSUV)

Notebook: 07a_lsuv.ipynb
Paper: [All You Need is a Good Init, by Dmytro Mishkin, Jiri Matas (2016)]

Algorithm 1 Layer-sequential unit-variance orthogonal initialization. L – convolution or full-connected layer, W_L - its weights, B_L - its output blob., Tol_{var} - variance tolerance, T_i – current trial, T_{max} – max number of trials.

Pre-initialize network with orthonormal matrices as in [Saxe et al. (2014)]
for each layer L **do**
  while $|Var(B_L) - 1.0| \geq Tol_{var}$ and $(T_i < T_{max})$ **do**
    do Forward pass with a mini-batch
    calculate $Var(B_L)$
    $W_L = W_L / \sqrt{Var(B_L)}$
  end while
end for

- LSUV is "data-driven weights initialization" that works on all kinds of layers: dropout, pooling, etc.
- Designed for SOTA (state of the art) performance on thin and deep convolutional neural networks
- Essentially LSUV is "batch normalization of layer outputs done only once before the start of training".
- Less computational overhead than full batch normalization
- They showed that the batch normalization layer is best done **after** the nonlinear activation function in most cases.
- Achieves SOTA performance on MNIST data

Data Block API Foundations

Notebook: 08_data_block.ipynb

Papers:
Systematic Evaluation of CNN advances on the ImageNet, by Dmytro Mishkin, Nikolay Segievsky, Jiri Matas (2005)
Bag of Tricks for Image Classification with Convolutional Neural Network, by Tong He, et al. (2018)

- Smaller image sizes allow accelerated training, enabling systematic experimental study of network architectures for ImageNet
- Subsets of ImageNet, 160 x 213 x 3 images, 10 of 1000 classes
    - Imagenette (easy)
    - Imagewoof (hard, similar dog breeds)
- In this notebook, we use Imagenette
- 13,394 image files
- Too large to keep in memory
- We will read in one image at a time
- To do this, we build a Data Block API

Data Block API Foundations

- get_files() fast function for reading images
- ListContainer, ItemList, ImageList classes enable construction of data set
- grandparent_splitter to build train and valid sets
- SplitData() class
- Processor() and CategoryProcessor() classes
- ProcessedItemList()
- LabeledData() class
- Transform ResizeFIxed(128) resizes the images to 128x128
- databunchify()
- On a Windows machine, put num_workers = 0, so you can't exploit multi-core CPU
- "It really is 3x3 kernels everywhere"
- 1-cycle scheduling with cosine annealing gets 72.6% accuracy on Imagenette

This gives us the full summary on how to grab our data and put it in a `DataBunch`:

```
path = datasets.untar_data(datasets.URLs.IMAGENETTE_160)
tfms = [make_rgb, ResizeFixed(128), to_byte_tensor, to_float_tensor]

il = ImageList.from_files(path, tfms=tfms)
sd = SplitData.split_by_func(il, partial(grandparent_splitter, valid_name='val'))
ll = label_by_func(sd, parent_labeler)
data = ll.to_databunch(bs, c_in=3, c_out=10, num_workers=4)
```

Model

Optimizer tweaks

Notebook: 09_optimizer.ipynb

Papers:

L2 Regularization versus Batch and Weight Normalization, by Twaan van Laarhoven (2017)

Three Mechanisms of Weight Decay Regularization, Guodong Zhang, Chaoqi Wang, Bowen Xu, Roger Grosse (2018)

*L2 regularization does not have a regularizing effect when used with normalization

* L2 regularization does something, but nobody really understands what

- Continue with imagenette data set
- Class Optimizer
- param_groups (used to be called layer_groups) list of lists
- [[hyperparams for all but the last 2 layers] , [hyperparams for last 2 layers]]
- Hyperparameters for each parameter group are stored in a dictionary
- stepper sgd_step() performs an SGD step
- Class ParamScheduler
- weight_decay() and l2_reg()
- Add momentum,
- Class StatefulOptimizer() keeps track of state
- Class Stat() keeps statistics
- Class AverageGrad(Stat)
- momentum_step()

Optimizer tweaks

Notebook: 09_optimzer.ipynb
Paper:
[Reducing BERT Pre-Training Time from 3 Days to 76 Minutes, by Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, Cho-Jui Hsieh (2019)]

- Debiasing momentum and squared momentum
- ADAM is dampened debiased momentum divided by dampened debiased squared momentum
- Epsilon goes **outside** the square root:

```
#export
def adam_step(p, lr, mom, mom_damp, step, sqr_mom, sqr_damp, grad_avg, sqr_avg, eps, **kwargs):
    debias1 = debias(mom,     mom_damp, step)
    debias2 = debias(sqr_mom, sqr_damp, step)
    p.data.addcdiv_(-lr / debias1, grad_avg, (sqr_avg/debias2).sqrt() + eps)
    return p
adam_step._defaults = dict(eps=1e-5)
```

LAMB

It's then super easy to implement a new optimizer. This is LAMB from a very recent paper:

$$
\begin{aligned}
g_t^l &= \nabla L(w_{t-1}^l, x_t) \\
m_t^l &= \beta_1 m_{t-1}^l + (1-\beta_1) g_t^l \\
v_t^l &= \beta_2 v_{t-1}^l + (1-\beta_2) g_t^l \odot g_t^l \\
m_t^l &= m_t^l / (1-\beta_1^t) \\
v_t^l &= v_t^l / (1-\beta_2^t) \\
r_1 &= \|w_{t-1}^l\|_2 \\
s_t^l &= \frac{m_t^l}{\sqrt{v_t^l + \epsilon}} + \lambda w_{t-1}^l \\
r_2 &= \|s_t^l\|_2 \\
\eta^l &= \eta * r_1 / r_2 \\
w_t^l &= w_t^{l-1} - \eta_l * s_t^l
\end{aligned}
$$

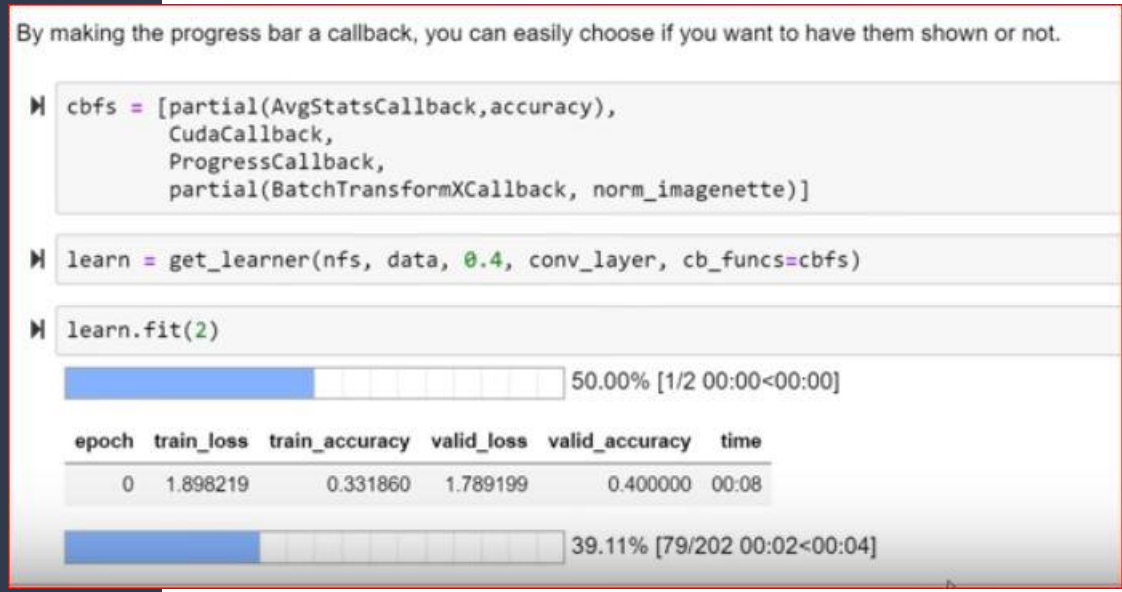
```
def lamb_step(p, lr, mom, mom_damp, step, sqr_mom, sqr_damp, grad_avg, sqr_avg, eps, wd, **kwargs):
    debias1 = debias(mom,     mom_damp, step)
    debias2 = debias(sqr_mom, sqr_damp, step)
    r1 = p.data.pow(2).mean().sqrt()
    step = (grad_avg/debias1) / ((sqr_avg/debias2).sqrt()+eps) + wd*p.data
    r2 = step.pow(2).mean().sqrt()
    p.data.add_(-lr * min(r1/r2,10), step)
    return p
lamb_step._defaults = dict(eps=1e-6, wd=0.)
```

Callbacks supercharged

Notebook: 09b_learner.ipynb

- Class Learner is refactored and now replaces class Runner.

Adding progress bars to learner (Sylvain Gugger)



Notebook: 09c_add_progress_bar.ipynb

Make sure to update fastprogress first:

```
conda install -c fastai
   fastprogress
```

Data augmentation

Notebook: 10_transforms.ipynb

- PIL Python Image Library (pillow)
- Understand the augmented data: look at the data if it's images, listen to it if it's audio, read it if it's text,
- Resize, Random flip
- Random crop & resize, Transpose
- Calculating with bytes is very, very fast
- Do transformation on bytes, then convert to tensor.
- Most important augmentation used in ImageNet is RandomResizedCrop
- Perspective warping: new feature
- Pytorch has module for solving system of linear equations
- Noisy labels: pick 8% to 100% of the pixels, so you often lose the target label
- Affine transforms, rotations, interpolation

Appendix

Things Jeremy says to do (Part 2) Lesson 11