Neural Network Optimisation

2019-04-18, CERN

Andrey Ustyuzhanin
NRU HSE
YSDA
ICL
Outline

[PyTorch quick intro]

I. Hyper parameter optimisation

II. Network architecture search

III. Bayesian NN perspective

IV. Outlook
Hyper parameter optimisation
Classic Machine Learning approach

There is always set of parameters, that define $F$, so we can repeat $\text{argmax}$ once again...
Scikit-Learn optimization classes

class sklearn.model_selection.GridSearchCV(estimator, param_grid, 
scoring=None, fit_params=None, n_jobs=1, iid=True, cv=None, 
pre_dispatch='2*n_jobs')

Curse of dimensionality makes this inefficient in higher dimensional problems.
Scikit-Learn optimization classes

class sklearn.model_selection.RandomizedSearchCV(estimator, param_distributions, n_iter=10, scoring=None, fit_params=None, n_jobs=1, iid=True, refit=True, cv=None)
Towards Bayesian optimisation

Filled contours plot of loss function $L(\gamma, C)$

- Low values of $L$ here, should probably not focus in this area
- High values of $L$ here, should probably focus in this area
Why ‘Bayesian’ search?

Operates with sample distributions
Bayesian inference
Iteratively updates posterior distribution given prior and observations
$f$ is a black box for which no closed form is known (nor its gradients);
$f$ is expensive to evaluate;
and evaluations of $y = f(X)$ may be noisy.
Bayesian optimization cycle

Define family of $f(x)$, approximated by a generative model, defined by $\Theta$ and some prior for it

Estimate probability of finding maximum of $f$ at $x : u(x)$

Update $\Theta$ for generative model conditional prob.

Measure $f$ at $\text{argmax}(u(x))$
Types of generative models

- Gaussian process Regression
- Random Forest Regression
- GBDT Regression
- NN Regression
Gaussian Process (GP)

Like a Gaussian defines distribution of variables (tensors), GP defines distribution of functions:

For every $x$, defines mean $\mu(x)$ and variance $\sigma(x)$

http://www.tmpl.fi/gp/

Determined by ($\Theta$): $X$, covariance matrix + kernel
Next best candidate x selection

\[ EI(\theta) = \mathbb{E}_{\theta}[\max\{0, f_M(\theta) - f_M(\hat{\theta})\}] \]
Next best candidate $x$ selection

$$EI(\theta) = \begin{cases} 
(\mu(\theta) - f(\hat{\theta})) \Phi(Z) + \sigma(\theta) \phi(Z), & \sigma(\theta) > 0 \\
0, & \sigma(\theta) = 0 
\end{cases},$$

$$Z = \frac{\mu(\theta) - f(\hat{\theta})}{\sigma(\theta)}$$
Next best candidate $x$ selection
Bayesian optimization cycle

Define family of $f(x)$, approximated by a generative model, defined by $\Theta$ and some prior for it

Estimate probability of finding maximum of $f$ at $x : u(x)$

Update $\Theta$ for generative model conditional prob.

Measure $f$ at $\text{argmax}(u(x))$
<Bayesian optimization demo>
Hyper parameter search is immediate step towards meta-learning
Usually no gradients are available
Should account for stochasticity

**Choose an appropriate scale for your hyperparameters**: For parameters like a learning rate, or regularization term, it makes more sense to sample on the log-uniform domain, instead of the uniform domain

**Choose the kernel of the GP carefully**: each kernel implicitly assumes different properties on the loss function, in terms of differentiability and periodicity

**Libraries**: scikit-optimize, Hyperopt, SMAC, ...
Network Architecture
Search
Basic idea

Search for network topology as a graph of nodes connected by operations.

For simplicity the whole network is split into sub-graphs (cells) connected to each other.

Connection between nodes and cells are parametrized (relaxed) by softmax operation.

Liu, Hanxiao and Simonyan, Karen and Yang, Yiming, DARTS: Differentiable Architecture Search, 2018
Structure of a cell[i]

0-th node is the output of cell[i-2], 1-st node is the output of cell[i-1]

All other nodes are connected to all of its predecessors through operations like convolution, pooling, identity, zero:

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)})$$

Softmax of all possible operations:

$$\bar{o}^{(i,j)}(x) = \frac{\exp(\alpha_{o}^{(i,j)})}{\sum_{o'\in\mathcal{O}} \sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

Nodes are parametrized by $\mathbf{w}$, connections are parametrized by $\alpha$
Cell optimisation procedure

\[
\begin{align*}
\min_{\alpha} & \quad \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\
\text{s.t.} & \quad w^*(\alpha) = \arg\min_{w} \mathcal{L}_{\text{train}}(w, \alpha)
\end{align*}
\]

**Algorithm 1: DARTS – Differentiable Architecture Search**

Create a mixed operation \(\bar{\sigma}^{(i,j)}\) parametrized by \(\alpha^{(i,j)}\) for each edge \((i, j)\)

while not converged do

1. Update weights \(w\) by descending \(\nabla_w \mathcal{L}_{\text{train}}(w, \alpha)\)
2. Update architecture \(\alpha\) by descending \(\nabla_\alpha \mathcal{L}_{\text{val}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)\)

Replace \(\bar{\sigma}^{(i,j)}\) with \(o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \alpha^{(i,j)}\) for each edge \((i, j)\)
Network structure

Initialise array of cells according to ‘genotype’ (specific cell structure trained before)

Connect each cell to two previous neighbors

Output of the final cell is the output of the network

Learn through backprop
Discussion

Trains reasonable amount of time ~ 1 day on GTX 1080
Works both for CNNs and RNNs
No rigorous guarantees, but:

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10. Results marked with † were obtained by training the corresponding architectures using our setup.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Test Error (%)</th>
<th>Params (M)</th>
<th>Search Cost (GPU days)</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet-BC (Huang et al., 2017)</td>
<td>3.46</td>
<td>25.6</td>
<td>–</td>
<td>manual</td>
</tr>
<tr>
<td>NASNet-A + cutout (Zoph et al., 2017)</td>
<td>2.65</td>
<td>3.3</td>
<td>1800</td>
<td>RL</td>
</tr>
<tr>
<td>NASNet-A + cutout (Zoph et al., 2017)†</td>
<td>2.83</td>
<td>3.1</td>
<td>3150</td>
<td>RL</td>
</tr>
<tr>
<td>AmoebaNet-A + cutout (Real et al., 2018)</td>
<td>3.34 ± 0.06</td>
<td>3.2</td>
<td>3150</td>
<td>evolution</td>
</tr>
<tr>
<td>AmoebaNet-A + cutout (Real et al., 2018)†</td>
<td>3.12</td>
<td>3.1</td>
<td>3150</td>
<td>evolution</td>
</tr>
<tr>
<td>AmoebaNet-B + cutout (Real et al., 2018)</td>
<td>2.55 ± 0.05</td>
<td>2.8</td>
<td>3150</td>
<td>evolution</td>
</tr>
<tr>
<td>Hierarchical Evo (Liu et al., 2017b)</td>
<td>3.75 ± 0.12</td>
<td>15.7</td>
<td>300</td>
<td>evolution</td>
</tr>
<tr>
<td>PNAS (Liu et al., 2017a)</td>
<td>3.41 ± 0.09</td>
<td>3.2</td>
<td>225</td>
<td>SMBO</td>
</tr>
<tr>
<td>ENAS + cutout (Pham et al., 2018b)</td>
<td>2.89</td>
<td>4.6</td>
<td>0.5</td>
<td>RL</td>
</tr>
<tr>
<td>Random + cutout</td>
<td>3.49</td>
<td>3.1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DARTS (first order) + cutout</td>
<td>2.94</td>
<td>2.9</td>
<td>1.5</td>
<td>gradient-based</td>
</tr>
<tr>
<td>DARTS (second order) + cutout</td>
<td>2.83 ± 0.06</td>
<td>3.4</td>
<td>4</td>
<td>gradient-based</td>
</tr>
</tbody>
</table>
Play on your own

Original repository: https://github.com/quark0/darts (supports pytorch 0.4)

Updated repository: https://github.com/dragen1860/DARTS-PyTorch

git clone <> ; cd DARTS-PyTorch
nvidia-smi # check if you have GPU onboard
mkdir exp
python train_search.py
<wait for genotype to evolve, add it to genotype.py>
python visualize.py NAME_OF_YOUR_GENOTYPE
python train.py —genotype NAME_OF_YOUR_GENOTYPE

Andrey Ustyuzhanin
Other approaches

Reinforcement learning
Neural Architecture Search with Reinforcement Learning (Zoph and Le, 2016)
NASNet (Zoph et al., 2017)
ENAS (Pham et al., 2018)

Evolutionary algorithm
Hierarchical Evo (Liu et al., 2017)
AmoebaNet (Real et al., 2018)

Sequential model-based optimisation (SMBO)
PNAS (Liu et al., 2017)

Bayesian optimisation
Auto-Keras (Jin et al., 2018)
NASBOT (Kandasamy et al. 2018)

Gradient-based optimisation
SNAS (Xie et al., 2018)
DARTS (Liu et al., 2018)
Other approaches

https://www.fast.ai/2018/07/16/auto-ml2/
Bayesian Perspective on Deep Learning

\[ P(w|X,Y) = \frac{P(Y|X,w)P(w)}{P(Y|X)} \]
What if instead of training a network via Maximum Likelihood (loss-function minimization), we could define a neural net as a tensor sampled from some distribution that we can get from the data?

\[ p(\theta|X) = \frac{\prod_{i=1}^{n} p(x_i|\theta)p(\theta)}{\int \prod_{i=1}^{n} p(x_i|\theta)p(\theta)d\theta} \]

Encodes uncertainty of the tensor in terms of distribution and expressed via data using Bayesian rule (inference).
Bayesian Machine Learning

- Suppose we’re given training data \((X, T)\) and a probabilistic classifier \(p(t|x, W)\)
- Define reasonable prior over the weights \(p(W)\)
- Training stage:

\[
p(W|X, T) = \frac{p(T|X, W)p(W)}{\int p(T|X, W)p(W)\,dW}
\]

- Test stage:

\[
p(t^*|x^*, X, T) = \int p(t^*|x^*, W)p(W|X, T)\,dW
\]

- Bayesian learning results in an ensemble of classifiers
Variational Trick

- Approximate posterior with a simpler distribution from a restricted parametric family

\[ p(W|X, T) \approx q(W|\phi) = \arg \min_{\phi} KL(q(W|\phi)\|p(W|X, T)) \]

- It can be shown that

\[ \arg \min_{\phi} KL(q(W|\phi)\|p(W|X, T)) = \arg \max_{\phi} \int q(W|\phi) \log \frac{p(T|X, W)p(W)}{q(W|\phi)} dW \]

- The last expression is usually denoted as \( \mathcal{L}(\phi) \) and has special name evidence lower bound (ELBO)
ELBO properties

\[ \mathcal{L}(\phi) = \int q(W|\phi) \log \frac{p(T|X,W)p(W)}{q(W|\phi)} dW \rightarrow \max_{\phi} \]

has several nice properties

- We may compute its stochastic gradient by performing **mini-batching** and removing integral with its MC estimate
- We do not overfit - the richer is parametric family the closer we are to the true posterior
- We may rewrite ELBO as follows

\[ \mathcal{L}(\phi) = \int q(W|\phi) \log p(T|X,W) dW - KL(q(W|\phi)||p(W)) \]

- The second term prevents \( q(W|\phi) \) from collapsing to maximum likelihood point
Bayesian Neural Network representation

\[ w_{ij} \sim \mathcal{N}(\theta_{ij}, \sigma^2_{ij}) \]

Gaussian mean field approximation
 Dropout reinvented

- In 2015 Kingma, Salimans and Welling decided to understand the nature of dropout
- They assumed that gaussian dropout corresponds to Bayesian procedure that optimizes ELBO using SGD with $q(W|\theta, \alpha) = \mathcal{N}(W|\theta, \alpha \theta^2)$

$$\int \mathcal{N}(W|\theta, \alpha \theta^2) \log p(T|X, W) dW - KL(\mathcal{N}(W|\theta, \alpha \theta^2)||p(W)) \rightarrow \max_\theta$$

- The first term corresponds to the criterion that is really optimized during dropout training... BUT there is no $KL$-term!
Variational Dropout

\[ p(W) \propto \frac{1}{|W|} \]

- KL-term does not depend on \( \theta \) but still depends on \( \alpha \)

\[ \mathcal{L}(\theta, \alpha) = \text{DataTerm}(\theta, \alpha) + KL(\alpha) \rightarrow \max_{\theta} \]

- Why not trying to optimize ELBO both w.r.t. \( \theta \) and \( \alpha \)?

\[ \mathcal{L}(\theta, \alpha) = \text{DataTerm}(\theta, \alpha) + KL(\alpha) \rightarrow \max_{\theta,\alpha} \]
Sparse Variational Dropout

- Now we may extend the variational family even further and assign individual dropout rates $\alpha_{ij}$ per each weight

$$
q(W|\theta, \alpha) = \prod_{i,j} \mathcal{N}(w_{ij}|\theta_{ij}, \alpha_{ij}\theta_{ij}^2)
$$

- It can be shown that if $\alpha_{ij} \to +\infty$ then $\theta_{ij} = O\left(\frac{1}{\alpha_{ij}}\right)$ i.e.

$$
\lim_{\alpha_{ij} \to +\infty} q(w_{ij}|\theta_{ij},\alpha_{ij}) = \delta(0)
$$

- Incredibly efficient way for removing the redundancy of current deep architectures

- Up to 99.9% of the weights in the layer become irrelevant

Visualisation

Epoch: 0  Compression ratio: 1x  Accuracy: 8.4
## Comparison

### Fully Connected network: LeNet-300-100
### Convolutional network: Lenet-5-Caffe

| Network          | Method   | Error % | Sparsity per Layer % | $|W_{\neq 0}|$ |
|------------------|----------|---------|-----------------------|---------|
| Original         |          | 1.64    |                       | 1       |
| Pruning          |          | 1.59    | 92.0 – 91.0 – 74.0    | 12      |
| LeNet-300-100 DNS|          | 1.99    | 98.2 – 98.2 – 94.5    | 56      |
| SWS              |          | 1.94    |                       | 23      |
| (ours) Sparse VD |          | 1.92    | 98.9 – 97.2 – 62.0    | 68      |
|                  |          | 0.80    |                       |         |
| Pruning          |          | 0.77    | 34 – 88 – 92.0 – 81   | 12      |
| LeNet-5-Caffe DNS|          | 0.91    | 86 – 97 – 99.3 – 96   | 111     |
| SWS              |          | 0.97    |                       | 200     |
| (ours) Sparse VD |          | 0.75    | 67 – 98 – 99.8 – 95   | 280     |
More details

https://www.youtube.com/watch?v=TD2PF6TZcx0
https://www.youtube.com/playlist?list=PLe5rNUydzV9Q01vWCP9BV7NhJG3j7mz62
Discussion

Bayesian framework is extremely powerful and extends ML tools
Impossible to overtrain!
Uncertainty estimation
Scalable algorithms for approximate Bayesian inference are already available
Quite breathtaking development!

http://cs.hse.ru/lambda/en
anaderiRu@twitter
austyuzhanin@hse.ru
Other interesting stuff

› Ternary quantization https://github.com/TropComplique/trained-ternary-quantization/tree/master/ttq_densenet_small

› Review talk by Tim Genewein


› Information Bottleneck: https://github.com/ravidziv/IDNNs

› Google vizier (https://github.com/tobegit3hub/advisor)

› https://github.com/ray-project/ray


› The Lottery Ticket Hypothesis... https://arxiv.org/abs/1803.03635