Bayesian Deep Active Learning

Computer vision and NLP

Sébastien Loustau

10/04/2023

Outlines

1. Bayesian Deep Learning
2. Active Learning for Computer Vision
3. Active Learning for NLP

Outlines

1. Bayesian Deep Learning
2. Active Learning for Computer Vision
3. Active Learning for NLP

Convex optimization

Gradient descent can be written as:

$$x_{t+1} := \arg\min_{x \in X} \left\{ \eta \nabla f(x_t) \cdot x + \frac{\|x - x_t\|^2}{2} \right\}.$$

Convex optimization

Gradient descent can be written as:

$$x_{t+1} := \arg\min_{x \in X} \left\{ \eta \nabla f(x_t) \cdot x + \frac{\|x - x_t\|^2}{2} \right\}.$$

Mirror descent solves:

$$x_{t+1} := \arg\min_{x \in \mathcal{P}} \left\{ \eta \nabla f(x_t) \cdot x + \mathcal{B}_{\Phi}(x, x_t) \right\}.$$

Convex optimization

Gradient descent can be written as:

$$x_{t+1} := \arg\min_{x \in X} \left\{ \eta \nabla f(x_t) \cdot x + \frac{\|x - x_t\|^2}{2} \right\}.$$

Mirror descent solves:

$$x_{t+1} := \arg\min_{x \in \mathcal{P}} \left\{ \eta \nabla f(x_t) \cdot x + \mathcal{B}_{\Phi}(x, x_t) \right\}.$$

Main idea change the search space X to a particular set of probability distribution $\mathcal{P}$.

Bayesian Learning

Given a dataset $\{(X_i, Y_i),\ i = 1, \ldots, n\}$, a set of candidate models $\{g_\theta,\ \theta \in \Theta\}$ and a loss function $\ell(\cdot, \cdot)$:

Frequentist paradigm

$$\min_{\theta \in \Theta} \left\{ \sum_{i=1}^{n} \ell(Y_i, g_\theta(X_i)) + \alpha \, \mathrm{pen}(\theta), \right\}$$

where $\mathrm{pen}(\cdot)$ avoids overfitting.

Bayesian paradigm

$$\min_{\rho \in \mathcal{P}(\Theta)} \left\{ \mathbb{E}_{\theta \sim \rho} \sum_{i=1}^{n} \ell(Y_i, g_\theta(X_i)) + \alpha \, \mathcal{K}(\rho, \pi) \right\},$$

where π is a prior distribution and $\mathcal{K}(\cdot, \cdot)$ is the KL divergence.

The Bayesian Learning Rule [6]

Approximate the solution by solving :

$$\min_{q \in \mathcal{Q}} \left\{ \mathbb{E}_{\theta \sim q} \sum_{i=1}^{n} \ell(Y_i, g_\theta(X_i)) + \alpha \, \mathcal{K}(q, \pi) \right\},$$

where $\mathcal{Q}$ is a particular family of distribution.

The Bayesian Learning Rule [6]

Approximate the solution by solving :

$$\min_{q\in\mathcal{Q}} \left\{ \mathbb{E}_{\theta\sim q} \sum_{i=1}^{n} \ell(Y_i, g_\theta(X_i)) + \alpha\, \mathcal{K}(q,\pi) \right\},$$

where $\mathcal{Q}$ is a particular family of distribution.

For exp-families $\mathcal{Q} = \{q(\theta) = \exp\left(\langle\lambda, T(\theta)\rangle\right\}$, we have the natural gradient VI algorithm available:

$$\lambda_{t+1} = \lambda_t - \rho \nabla_\mu \mathbb{E}_q \left[\bar{\ell}(\theta) - \mathcal{H}(q) \right],$$

where $\bar{\ell}(\cdot) = \sum_{i=1}^{n} \ell(Y_i, f_\cdot(X_i))$ and $\mu = \mathbb{E}_{\theta\sim q} T(\theta)$.

The Bayesian Learning Rule [6]

Approximate the solution by solving :

$$\min_{q \in \mathcal{Q}} \left\{ \mathbb{E}_{\theta \sim q} \sum_{i=1}^{n} \ell(Y_i, g_\theta(X_i)) + \alpha\, \mathcal{K}(q, \pi) \right\},$$

where $\mathcal{Q}$ is a particular family of distribution.

We can recover **standard algorithm**:

- $\mathcal{Q} = \mathcal{N}(m, I_p)$ GD-like algorithm,
- $\mathcal{Q} = \mathcal{N}(m, \Sigma^2)$ Newton-like algorithm,
- $\mathcal{Q} = \mathcal{B}(p)$ STE estimator.

Sketch of proof : GD (order 1)

$q(\theta) = \mathcal{N}(m, I_p)$
$\lambda = m$
$\mu = m$
$\mathcal{H}(q) = \log \frac{2\pi}{2}$

Sketch of proof : GD (order 1)

$q(\theta) = \mathcal{N}(m, I_p)$
$\lambda = m$
$\mu = m$
$\mathcal{H}(q) = \log \frac{2\pi}{2}$

$$\theta_{t+1} = \theta_t - \rho \nabla_\theta \bar{\ell}(\theta_t)$$

$$\lambda_{t+1} = \lambda_t - \rho \nabla_\mu \mathbb{E}_q \left[\bar{\ell}(\theta) - \mathcal{H}(q) \right]$$

Sketch of proof : GD (order 1)

$q(\theta) = \mathcal{N}(m, I_p)$
$\lambda = m$
$\mu = m$
$\mathcal{H}(q) = \log \frac{2\pi}{2}$

$$\theta_{t+1} = \theta_t - \rho \nabla_\theta \bar{\ell}(\theta_t)$$

$$m_{t+1} = m_t - \rho \nabla_m \mathbb{E}_q \bar{\ell}(\theta)$$
$$\lambda_{t+1} = \lambda_t - \rho \nabla_\mu \mathbb{E}_q \left[\bar{\ell}(\theta) - \mathcal{H}(q) \right]$$

Sketch of proof : GD (order 1)

$q(\theta) = \mathcal{N}(m, I_p)$
$\lambda = m$
$\mu = m$
$\mathcal{H}(q) = \log \frac{2\pi}{2}$

$$
\begin{aligned}
\theta_{t+1} &= \theta_t - \rho \nabla_\theta \bar{\ell}(\theta_t) \\
\text{Smoothing } \mathbb{E}_q \bar{\ell}(\theta) &\approx \bar{\ell}(m) \\
m_{t+1} &= m_t - \rho \nabla_m \mathbb{E}_q \bar{\ell}(\theta) \\
\lambda_{t+1} &= \lambda_t - \rho \nabla_\mu \mathbb{E}_q \left[\bar{\ell}(\theta) - \mathcal{H}(q) \right]
\end{aligned}
$$

Variational Online Gauss-Newton (order 2)

When $\mathcal{Q} = \mathcal{N}(m, \Sigma^2)$ with diagonal covariance Σ, we lead to VOGN ([4]):

$$\mu_{t+1} = \mu_t - \alpha_t \frac{\nabla_\theta \ell(Y_i, g_{\theta_t}(X_i)) + \tilde{\delta}\mu_t}{s_{t+1} + \tilde{\delta}},$$

where $\tilde{\delta} = \tau\delta/M$ and:

$$s_{t+1} = (1 - \tau\beta_t)s_t + \beta_t \sum_{i \in \mathrm{mb}_t} \nabla_\theta \ell(Y_i, g_{\theta_t}(X_i))^2.$$

Variational Online Gauss-Newton (order 2)

When $\mathcal{Q} = \mathcal{N}(m, \Sigma^2)$ with diagonal covariance Σ, we lead to VOGN ([4]):

$$\mu_{t+1} = \mu_t - \alpha_t \frac{\nabla_\theta \ell(Y_i, g_{\theta_t}(X_i)) + \tilde{\delta}\mu_t}{s_{t+1} + \tilde{\delta}},$$

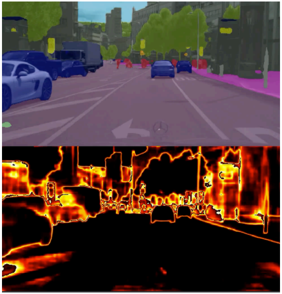
where $\tilde{\delta} = \tau\delta/M$ and:

$$s_{t+1} = (1 - \tau\beta_t)s_t + \beta_t \sum_{i \in \mathrm{mb}_t} \nabla_\theta \ell(Y_i, g_{\theta_t}(X_i))^2.$$

RMSPROP or ADAM equivalence :

$$\theta_{t+1} = \theta_t - \alpha_t \frac{\frac{1}{M}\sum_{i \in \mathrm{mb_t}} \nabla_\theta \ell(Y_i, g_{\theta_t}(X_i)) + \delta\theta_t}{s_{t+1} + \tilde{\delta}}$$

History of Bayesian Neural Networks



- 1995 [5] for MCMC origin,
- 2011 [3] for VI approach to TIMIT speech dataset,
- 2016 [2] uses MC-dropout as Bayesian approximation,
- 2017 [1] uses confidence intervals for adversarial attacks,
- 2018 [4] **scales to Imagenet**

Pytorch library

We use pytorch-sso an open-source library for second-order optimization and Bayesian inference developped Approx-Bayes Team of the Riken Institute.

```
import torch
+import torchsso

train_loader = torch.utils.data.DataLoader(train_dataset)
model = MLP()

-optimizer = torch.optim.Adam(model.parameters())
+optimizer = torchsso.optim.VOGN(model, dataset_size=len(train_loader.dataset))

for data, target in train_loader:

    def closure():
        optimizer.zero_grad()
        output = model(data)
        loss = F.binary_cross_entropy_with_logits(output, target)
        loss.backward()
        return loss, output

    loss, output = optimizer.step(closure)
```

Outlines

1. Bayesian Deep Learning
2. Active Learning for Computer Vision
3. Active Learning for NLP

The problem

We want to use this optimizer for active learning on $\mathrm{CIFAR}\ 10$:

1. Start an initial training with a subsample of the training set,
2. Select a second subsample with Bayesian-like incertainty,
3. Continue training with this second subsample,
4. Validate over the test set.

We want to compare this routine with standard non-Bayesian approaches.

Uncertainty metrics

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

Uncertainty metrics

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

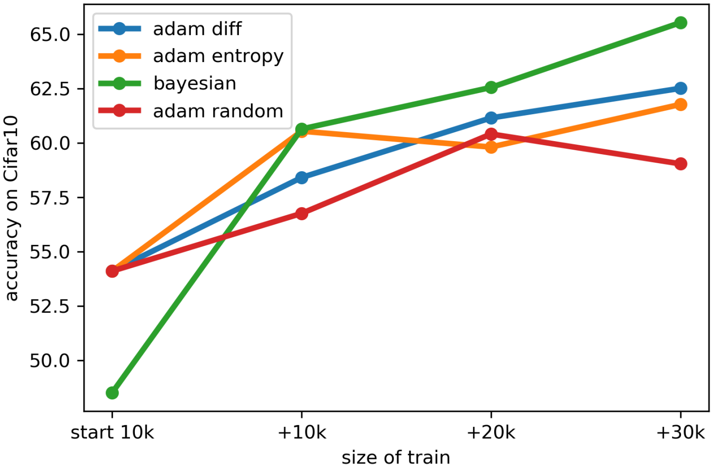
Bayesian measures

For a Bayesian solution $q_T(\cdot) \sim \mathcal{N}(m_T, \Sigma_T)$, sample $N = 10$ outputs and compute:

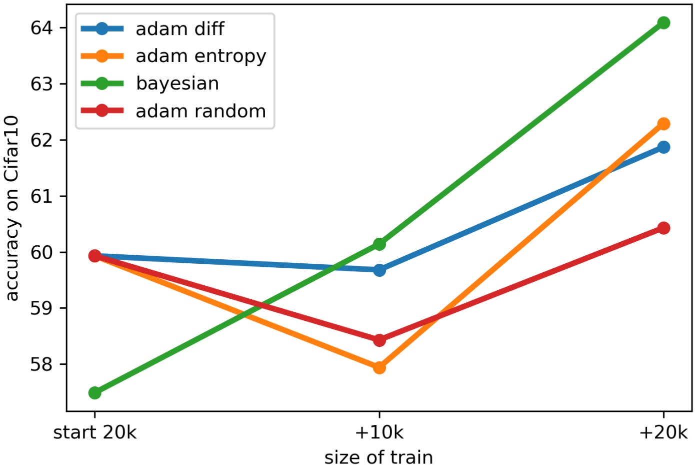
$$x \mapsto \mathcal{H}_{q_T}(y^{(u)}(x)),$$

where $y^{(u)}(\cdot)$ is the distribution of the top-u outputs over the $N = 10$ realizations.

Results


adam diff
adam entropy
bayesian
adam random
accuracy on Cifar10
65.0
62.5
60.0
57.5
55.0
52.5
50.0
start 10k
+10k
+20k
+30k
size of train

Results


adam diff
adam entropy
bayesian
adam random
64
63
62
61
60
59
58
accuracy on Cifar10
start 20k
+10k
+20k
size of train

Outlines

1. Bayesian Deep Learning
2. Active Learning for Computer Vision
3. Active Learning for NLP

The problem

Objective We want to reproduce these results with NLP models.

- Task: Multi-label text classification (MTC, XMTC)
- Dataset la-derniere-bibliotheque.org
- Dataset philoml.org
- Validation metrics top1, top5 and 'recall'.

Benchmark

LDB dataset	Camembert[1]	Flaubert[2]
top5	0.98	0.97
top1	0.78	0.77
recall	0.8	0.8

PHI dataset	Camembert	Flaubert
top5	0.91	-
top1	0.66	-
recall	0.71	-

[1] `CamembertForSequenceClassification`50 epochs, lr=10^{-4}

[2] `FlaubertForSequenceClassification` with 50 epochs, lr=10^{-5}

Bayesian NLP

We test active learning by training a FC layer with VOGN in the following pipeline:

- Start an initial training with a subsample of the training set in two steps:
  - train Tranformers embeddings with 5 epochs,
  - train a FC layer with VOGN.
- Select a second subsample with Bayesian-like incertainty,
- Continue training with this second subsample,
- Validate over the test set.

Uncertainty metrics for MTC

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

Uncertainty metrics for MTC

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

WARNING

Uncertainty metrics for MTC

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

WARNING The uncertainty depends on the number of tags!

Uncertainty metrics for MTC

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

WARNING The uncertainty depends on the number of tags!

Bayesian measures

For a Bayesian solution $q_T(\cdot) \sim \mathcal{N}(m_T, \Sigma_T)$, sample $N = 10$ outputs and compute:

$$x \mapsto \mathcal{H}_{q_T}(y^{(u)}(x)),$$

where $y^{(u)}(\cdot)$ is the distribution of the top-u outputs over the $N = 10$ realizations.

Uncertainty metrics for MTC

Non-Bayesian measures

For a deterministic network g_{θ_T}, compute $x \mapsto \mathcal{U}(g_{\theta_T}(x))$, where:

- $\mathcal{U}(y) = \mathcal{H}(y)$ for entropy uncertainty,
- $\mathcal{U}(y) = 1 - (y^{(1)} - y^{(2)})$ for difference uncertainty.

WARNING The uncertainty depends on the number of tags!

Bayesian measures

For a Bayesian solution $q_T(\cdot) \sim \mathcal{N}(m_T, \Sigma_T)$, sample $N = 10$ outputs and compute:

$$x \mapsto \mathcal{H}_{q_T}(y^{(u)}(x)),$$

where $y^{(u)}(\cdot)$ is the distribution of the top-u outputs over the $N = 10$ realizations.

The randomness of the Bayesian procedure does the job !

References I

John Bradshaw, Alexander G. de G. Matthews, and Zoubin Ghahramani.
Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks, 2017.

Yarin Gal and Zoubin Ghahramani.
Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.

Alex Graves.
Practical variational inference for neural networks.
In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

References II

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava.
Fast and scalable bayesian deep learning by weight-perturbation in adam, 2018.

Redford M. Neal.
Bayesian learning for neural networks, 1995.

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, Rio Yokota, and Mohammad Emtiyaz Khan.
Practical deep learning with bayesian principles, 2019.