GSoC Project Proposal
Add Variational Inference Interface to PyMC4

Description

Variational Inference is a powerful algorithm that turns the task of computing the posterior(p(z|x)) into an optimization problem. This project is about implementing two inference algorithms Mean Field ADVI and Full Rank ADVI based on ADVI paper in PyMC4. Mean Field ADVI posits a Spherical Gaussian family and Full Rank ADVI posits a Multivariate Gaussian family to minimize KL divergence. The implementation will use tf and tfp libraries.

Key Challenges

1. theano.clone equivalent for Tf2: For drawing samples from posterior, we need to replace some nodes of the graph with variational distributions. PyMC3 achieves this by using theano.clone. Tensorflow v1 implements some similar functions in the contrib module - tf.contrib.copy_graph . But the tf.contrib has been deprecated in TF2. Support of symbolic graph manipulations has also been removed.
    a. tf.keras.backend can be used for symbolic graphs.
    b. tf-nightly provides tf.raw_ops.Copy for recursive deep copying of tensors. Maybe it is possible to implement a feature similar to theano.clone by looking upon implementations of both copy_graph(in tf.v1) and Copy(in tf.nightly).
2. Provide an interface for flattened view of variables. PyMC3 does this by using theano.ravel(). Maybe it can be accomplished by tf.reshape().
3. Design an interface to account for minibatches.
4. Figure out a way to optimize ELBO. tf.keras.optimizers is an option or provide an iterative optimization algorithm.
5. Handle the initialization of means and stds for MeanField and Full Rank ADVI.
6. Add a Progress Bar using tf.keras.utils.Progbar.
7. Provide a reusable interface to implement more inference algorithms (SVGD, ASVGD).

Project Timeline

- Pre GSoC Period: March 29 - March 31
  Figure out the solutions of Challenges described above. I will make design less and less abstract as the time progresses and submit the proposal.

- Student Application and Review Application Period: March 31 - May 4
  I will get myself more familiar with the codebase of PyMC3, PyMC4 and begin implementing a few utility functions.

- Community Bonding Period: May 4 - May 31
  I will finalize implementation design for both algorithms by seeking reviews from mentors and community and also referring to OPVI and ADVI paper.

- Week 1 - 2: Jun 1 - June 14
  The actual coding begins. By this time, I will have a fair idea of interface design. I will implement utility functions (theano.clone and flatten). For every functionality, I will write test cases.

- Week 3: June 14 - June 21
  During this interval, I will begin coding base Approximation class keeping in mind the utility functions. I will try to complete implementation of the base classes upto this week so that week 4 can be utilized for dealing with optimizers.

- Week 4: June 21 - June 28
  The best part before Phase 1 evaluation is to explore `tf.keras.optimizers`. Having set up variational gradient tensor, tf.keras provides a wide range of optimizers to choose from, to maximize ELBO. Good options - `optimizers.Adagrad`, `optimizers.Adamax`. Also I will create a progress report for phase 1 evaluations.

- Evaluation Phase 1: June 29 - July 3
  At this time, my focus will be on writing visual illustrations/notebooks for use cases of these base classes. I also wish to add a progress bar using `tf.keras.utils.Progbar` to check convergence of ELBO during this interval or before phase 2 evaluations.

- Week 5 - 6: July 4 - July 18
  Having implemented base classes, I will write Mean Field Approximation class and pm.fit function to make inference. Meanwhile, I will also draft a design for FullRank ADVI to save time in later weeks.

- Week 7: July 19 - July 26
  I am not sure if I will be able to complete the implementation of Mean Field and fit function. So, this week can be kept as a buffer to complete any pending tasks. I will also look for handling inference for mini batches. Designing progress reports.

- Evaluation Phase 2: July 27 - July 31
  Again, writing notebooks/blogs to explain good use cases.

- Week 8 - 9: August 1 - August 14
  Implementation of FullRank Approximation class. Alter the pm.fit function to also account for FullRank ADVI option. Handle mini batches in FullRank ADVI. Tests in Pytest. Writing illustrations/blogs.

- Week 10: August 15 - August 23
  This final week can be used to complete any pending tasks and to summarize the ups and downs while working on the project. I will share my entire experience of GSoC through blogs.

- Final Submission of Code: August 24 - August 31

- Bonus Section:
  If I will be able to complete tasks in time, I will look for implementing SVGD and ASVGD algorithms. I also want to add support of Local Random Variables as mentioned in the Auto-Encoding Variational Bayes paper.

- Post GSoC Period: September onwards
  I will work on implementing Normalizing Flows in PyMC4 using tf and tfp libraries.

Contributing to PyMC4

1. Pull Request #220 (Merged): Add AutoRegressive distribution -
   This PR added Auto Regressive class by wrapping `sts.AutoRegressive` Model. The main task was to call the `make_state_space_model` method with suitable arguments to capture the underlying `tfd.LinearGaussianStateSpaceModel`. It took a lot of debugging to make this AR class compatible with PyMC4.

2. Pull Request #215 (Merged): Add transform for UnitContinuous Distribution -
   This PR added sigmoid transform to Unit Continuous Distribution. To make the default transform compatible with PyMC4, I also added a Sigmoid transform that used `tfb.Sigmoid` bijector.

3. Pull Request #212 (Merged): Update design_guide notebook -
   This small PR fixes typos and variable names in `pymc4_design_guide.ipynb`.

4. Issue #211 (Closed): Installation issues -
   I encountered installation issues while setting up the working environment using pip. So, I created the issue and Luciano Paz helped me out with other ways of installing PyMC4.

5. Pull Request #777 (Merged): Fix code examples in tfp.mcmc -
   This small PR fixes code examples in tfp.mcmc (backend for PyMC4). Changed files are mcmc/sample.py and mcmc/dual_averaging_step_size_adaptation.py.

I will continue contributing to PyMC4 throughout the selection process and beyond.

Personal Projects

- SendToS3 - This python project sends backup files to AWS S3 bucket using Boto3. Searching for files is done by regex and results of logs are sent to email using smtplib.

- Osint-Spy - This Python project performs Osint scan on email, domain, ip, organization, etc. This information can be used by Data Miners or Penetration Testers in order to find deep information about their target.

- Turbofan Degradation - Implemented Deep learning based Encoder-Decoder model for analysing the turbofan degradation dataset provided by NASA.

- Neural Network from Scratch - Implemented a deep neural network from scratch in numpy with custom hyperparameters.

Basic/Contact Information

- Time Zone - UTC+05:30
- Github Handle - Sayam753
- Contact details -
  - Gmail - sayamkumar049@gmail.com
  - Yahoo mail - sayamkumar753@yahoo.in
  - LinkedIn - https://www.linkedin.com/in/sayam049/
  - Mobile - +91 9815247310
- Resume - Google drive link
- Personal Website to post all blogs/notebooks - https://codingpaths.com

Personal Info

I am Sayam Kumar from Indian Institute of Information Technology Sri City, India. I am a second year Undergraduate pursuing a Bachelor's in Computer Science Engineering. I mostly code in Python. I got interested in Machine Learning as it involves Maths and helps in solving daily life problems. I have done some MOOCs to get myself familiar with it last year. Now I am preparing to participate in Kaggle competitions.

Why did I choose this project?

I see PyMC as a perfect blend of Maths and Probability. That’s why I enjoy contributing to it. I am interested in working on Variational Inference project to expand my knowledge in Machine Learning and Bayesian Statistics. This will also help me in learning and implementing state of the art papers on Variational Inference in PyMC4. Also this is my first time participating in GSoC.

Commitments

As I have no other projects/internships planned for summers, I can spend 40~50 hours per week or more if required working on the project. My Fall semester will start from early August. So only August month of GSoC timeline overlaps with my college timings. But I am sure that I will be working on the project alongside, as the workload is less in the beginning of the semester.

With my continuous efforts to learn and know more about Bayesian Statistics, I believe I will be able to complete the project in time. Along the way, I will design progress reports and extensive documentation of the implementation of various classes. This will help in submitting reports to mentor and Google at evaluation time.

Interface Design

In this section, I have written a draft design for both algorithms. I believe it can change dramatically over the course of GSoC. I will keep updating it before the submission deadline as I find out solutions to proposed challenges above.

```python
# Base class for Approximation
class Approximation:

    # Defining parameters
    def __init__(self, model, size, random_seed=None):
        self.model = model  # handle the initialization of means/stds
        self.size = size
        self.random_seed = random_seed

    @property
    def mean(self):
        return self.mu

    @property
    def std(self):
        return self.std

    # For Naive Monte Carlo
    def random(self):
        g = tf.random.Generator.from_seed(self.random_seed)
        n = g.normal(shape=some_shape)
        return self.std*n + self.mean
```

```
class MeanField(Approximation):

    def __init__(self, model, size, random_seed):
        super().__init__(model, size, random_seed)

    def cov(self):
        sq = tf.math.square(self.std)
        return tf.linalg.diag_part(sq)

class FullRank(Approximation):

    def __init__(self, model, size, random_seed):
        super().__init__(model, size, random_seed)

    def L(self):
        n = self.size
        entries = n*(n+1)//2
        L = np.zeros([n, n], dtype=int)
        L[np.tril_indices(n)] = np.arange(entries)
        L[np.tril_indices(n)[::-1]] = np.arange(entries)
        return L

    def cov(self):
        L = self.L
        return tf.linalg.matmul(L, tf.transpose(L))
```

The fit function serves as an abstraction to infer using Mean Field or Full Rank ADVI. Full Rank ADVI is computationally expensive as it tries to capture the correlations between various parameters. The interface design for accounting mini-batches, is yet to be decided.

```python
def fit(model, n=10000, random_seed=None, method='MeanFieldADVI'):

    # Transform the model into an unconstrained space
    _, state = pm.evaluate_model_transformed(model)
    logpt = state.collect_log_prob()

    # Collect the free random variables
    untransformed = state.untransformed_values
    free_RVs = untransformed.update(state.transformed_values)

    # Not sure about the use of local random variables
    size = 0
    for name, dist in free_RVs.items():
        size += int(np.prod(dist.event_shape))

    approx = None
    if method == "MeanFieldADVI":
        approx = MeanField(model, size, random_seed)
    else:
        approx = FullRank(model, size, random_seed)

    # Create variational gradient tensor
    q = approx.random()
    elbo = q + tf.reduce_sum(approx.std) +
           0.5*size*(1 + tf.math.log(2.0*np.pi))

    # Set up optimizer

    # Draw samples from variational posterior

    # TODO: Plot the trace using ArviZ
```

References and Useful Papers

- Automatic Differentiation Variational Inference . Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2016).
- Automatic Variational Inference in Stan. Kucukelbir, A., Ranganath, R., Gelman, A., & Blei, D. (2015).
- Operator Variational Inference. Rajesh Ranganath, Jaan Altosaar, Dustin Tran, David M. Blei (2016).
- Auto-Encoding Variational Bayes. Kingma, D. P., & Welling, M. (2014).
- Weight Uncertainty in Neural Network. Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015).