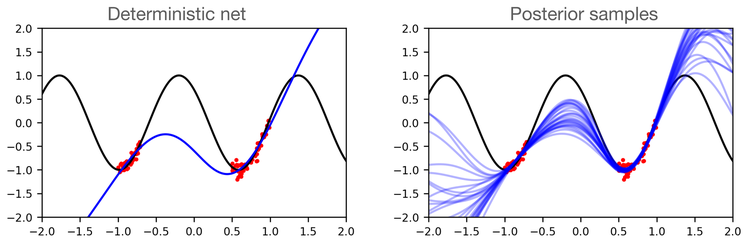
TyXe : Pyro-Based Bayesian Neural Networks for Pytorch Users in 5 Lines of Code

(3) University College London, UK
Hippolyt Ritter (1,3) , Theofanis Karaletsos (1,2) (1) (2)
(at submission) Uber AI Labs, San Francisco, California
Facebook, Menlo Park, California

Motivation

Neural networks fit a single function to the data, but many are similarly plausible.



Various libraries simplify the workflow for a deterministic network, but training BNNs is often cumbersome. Packages typically provide implementations for specific combinations of prior, inference method and layer type, leaving users with few options for the probabilistic model definition and making it hard to plug in existing network architectures.

We introduce **TyXe** (*Greek: chance*), a Pyro- and Pytorch-based libary that facilitates constructing and training Bayesian neural networks for Pytorch practitioners.

```
net = nn.Sequential(nn.Linear(1, 50), nn.Tanh(), nn.Linear(50, 1))
obs = tyxe.observation_models.HomoskedasticGaussian(scale=0.1)
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1))
guide_factory = tyxe.guides.ParameterwiseDiagonalNormal
bnn = tyxe.SupervisedBNN(net, prior, obs, guide_factory)
```

We cleanly separate prior, likelihood, inference and neural architecture, enabling a flexible workflow that leverages Pytorch and Pyro to independently iterate over any of these components.

Key features

Architectures
- Manually defined nn.Modules, e.g. fully connected networks
- Architectures from torchvision.models (e.g. Resnet or wide Resnet)

Weight priors
- IID prior (supports Pyro distributions, e.g. Normal or mixtures of Normals)
- Layerwise (allows for different variances depending on the layer size)
- Flexible choice for which layers to perform inference on, e.g. last-layer only

Likelihoods
- Binary
- Categorical
- Homoskedastic Gaussian (known or unknown variance)
- Heteroskedastic Gaussian

Inference
- Variational posteriors through pyro.infer.autoguides
- HMC and NUTS through pyro.infer.mcmc
- Flexible factorized variational Gaussian posterior that allows for local reparameterization, limiting the variance and only training means or variances

More to come…

Code: https://github.com/karalets/tyxe

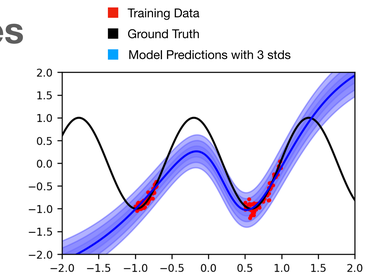
BNN-specific inference features

High-level interface for training...

```
optim = pyro.optim.Adam({"lr": 1e-3})
with tyxe.poutine.local_reparameterization():
    bnn.fit(data_loader, optim, num_epochs)
```

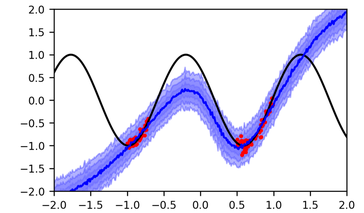
...and test time prediction.

```
pred_params = bnn.predict(x_test, num_samples)
```



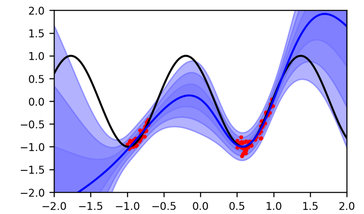
Sampling logic (local reparameterization) can be modified through a context manager both at train and test time. No coupling of prior, inference and sampling in a layer class.

```
with tyxe.poutine.local_reparameterization():
    pred_params = bnn.predict(x_test, num_samples)
```



Unified interface for variational inference and MCMC.

```
kernel = pyro.infer.mcmc.HMC
bnn = tyxe.MCMC_BNN(net, prior,
    obs_model, kernel)
bnn.fit(data_loader, num_samples,
    warmup_steps=num_steps)
pred_params = bnn.predict(x_test,
    num_test_samples)
```



Application example: Bayesian Resnet

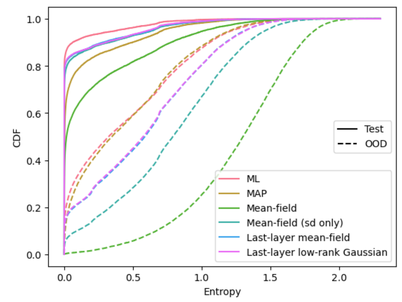
We can pick a pre-existing architecture from Pytorch:

```
net = torchvision.models.resnet18()
```

Desiderata:
- We want to make the model Bayesian
- We want to iterate rapidly over choices
- We want to examine what that has bought us

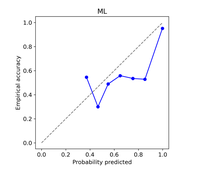
TyXe & Pyro facilitate:
- Different priors
- Different variational posteriors
- Different Inference Methods
- Last-layer methods
- Refining pre-trained (i.e. MLE models)

Experiments CIFAR10: Application to SVHN-OOD detection

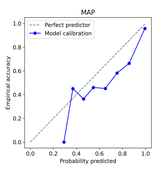


We examine model robustness to capturing OOD samples with a case study applied to SVHN data. TyXe models improve over MLE here (MF wins).

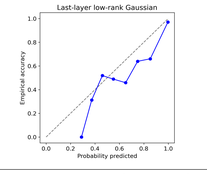
Experiments CIFAR10: Impact on test-calibration



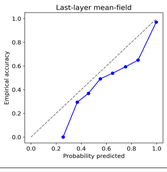
Pre-trained Network (MLE Baseline)



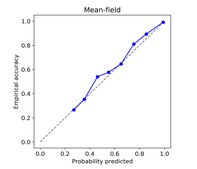

```
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1))
guide = autoguides.AutoDelta
```




```
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1),
    expose_modules=[net.fc])
guide = autoguides.AutoLowRankMultivariateNormal
```

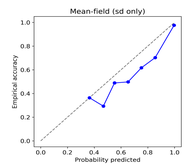



```
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1),
    expose_modules=[net.fc])
guide = tyxe.guides.ParameterwiseDiagonalNormal
```




```
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1))
guide = tyxe.guides.ParameterwiseDiagonalNormal
```

MF wins!




```
prior = tyxe.priors.IIDPrior(dist.Normal(0, 1),
    expose_modules=[net.fc])
guide = partial(
    tyxe.guides.ParameterwiseDiagonalNormal,
    train_loc=False)
```

Keep trained means, train 'uncertainty' bits.