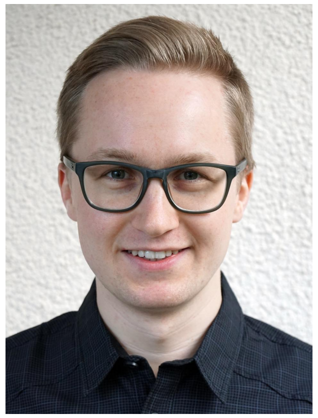
Thirty-seventh International Conference on Machine Learning (ICML 2020)

Training Binary Neural Networks Using the Bayesian Learning Rule



Xiangming Meng
(RIKEN AIP)
Presenter

Roman Bachmann
(EPFL)

Mohammad Emtiyaz Khan
(RIKEN AIP)

Binary Neural Networks (BiNN)

- BiNN: Neural Networks with binary weights
- Much faster and much smaller [1,2]
- Difficult to optimize in theory (discrete optimization)
- But easy in practice: Just use SGD with "Straight-through estimator (STE)"!
- It is mysterious as to why this works [3]
- Are there any principled approaches to explain this?

1. Courbariaux et al., Training deep neural networks with binary weights during propagations. NeurIPS 2015.
2. Courbariaux et al., . Binarized neural networks.… arXiv:1602.02830, 2016.
3. Yin, P. et al., Understanding straight-through estimator in training activation quantized neural nets. arXiv, 2019.

Our Contribution: Training BiNN using Bayes

- We show that by using the Bayesian Learning Rule [1,2] (natural-gradient variational inference), we can justify such previous approaches

  - Main point: optimize the parameter of a Bernoulli distribution (a continuous optimization problem)

- The Bayesian approach gives us an estimate of uncertainty which can be used for continual learning [3]

1. Khan, M. E. and Rue, H. Learning-algorithms from bayesian principles. ArXiv. 2019.
2. Khan, M. E. and Lin, W. Conjugate-computation variational inference. AISTATS, 2017
3. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. PNAS, 114(13):3521–3526, 2017.

Training BiNN is a Discrete Optimization problem!

Binary weights

Output

Input

$$\min_{\boldsymbol{w}\in\{-1,1\}^W} \sum_{i\in\mathcal{D}} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))$$

Loss

Neural Network

Training BiNN is a Discrete Optimization problem!

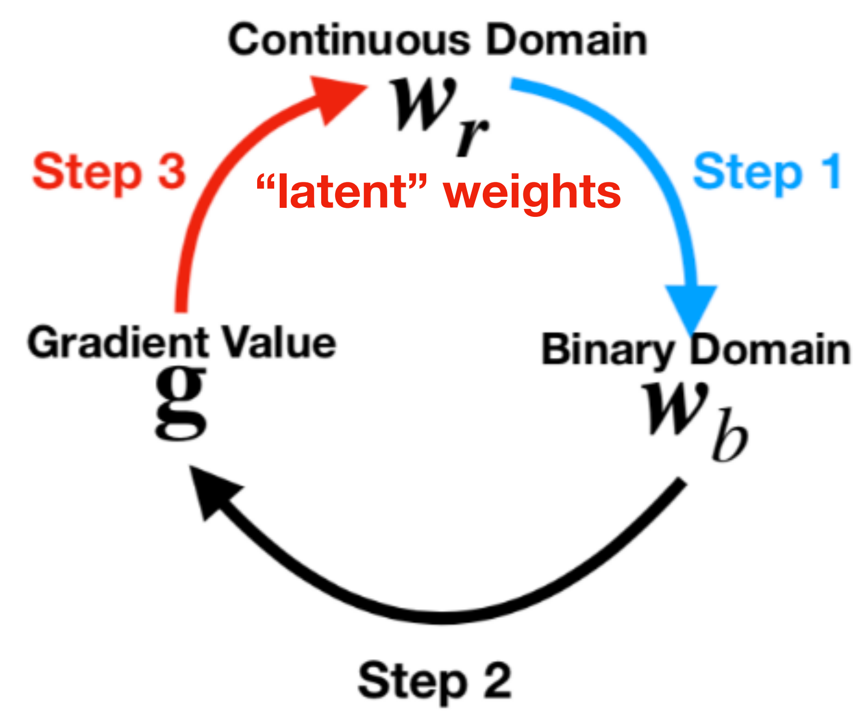
Binary weights

Output Input

$$\min_{\boldsymbol{w}\in\{-1,1\}^W} \sum_{i\in\mathcal{D}} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))$$

Loss Neural Network

- Easy in practice: SGD with "Straight-through estimator (STE)" [1]



1. Bengio et al. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432, 2013.
2. Helwegen et al. Latent weights do not exist: Rethinking binarized neural network optimization. arXiv preprint arXiv:1906.02107, 2019.
3. Yin, P. et al., Understanding straight-through estimator in training activation quantized neural nets. arXiv, 2019.

Training BiNN is a Discrete Optimization problem!

Binary weights

Output Input

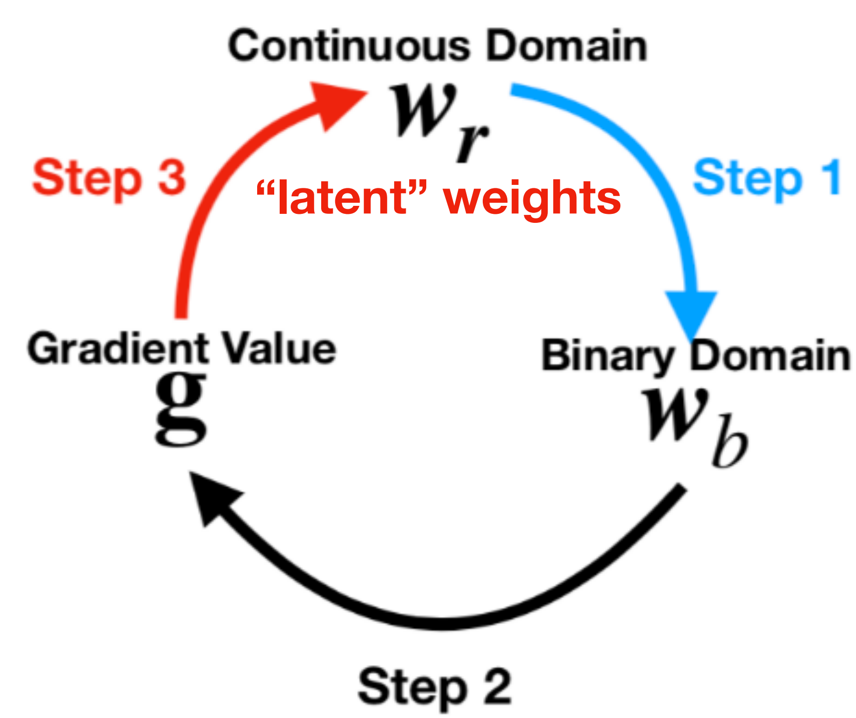
$$\min_{\boldsymbol{w} \in \{-1,1\}^W} \sum_{i \in \mathcal{D}} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))$$

Loss Neural Network

- Easy in practice: SGD with "Straight-through estimator (STE)" [1]

- Helwegen et al. [2] argued "latent" weights are not weights but "Inertia"

  → Binary Optimizer (Bop)



1. Bengio et al. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432, 2013.
2. Helwegen et al. Latent weights do not exist: Rethinking binarized neural network optimization. arXiv preprint arXiv:1906.02107, 2019.
3. Yin, P. et al., Understanding straight-through estimator in training activation quantized neural nets. arXiv, 2019.

Training BiNN is a Discrete Optimization problem!

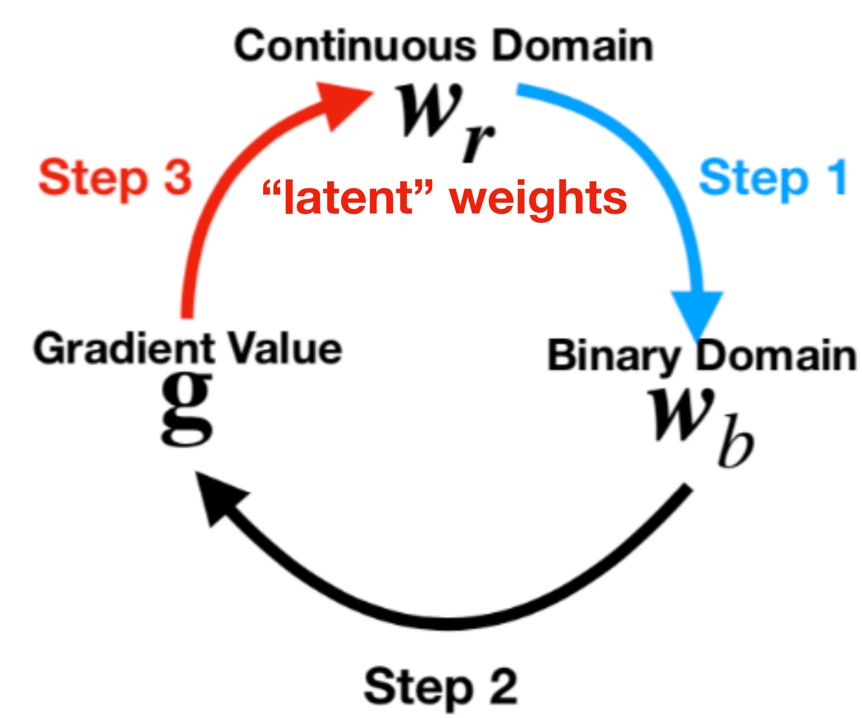
Binary weights

Output Input

$$\min_{\boldsymbol{w}\in\{-1,1\}^W} \sum_{i\in\mathcal{D}} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))$$

Loss Neural Network

- Easy in practice: SGD with "Straight-through estimator (STE)" [1]
- Helwegen et al. [2] argued "latent" weights are not weights but "Inertia"
  - → Binary Optimizer (Bop)
- Open question: Why does this work?[3]



1. Bengio et al. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432, 2013.
2. Helwegen et al. Latent weights do not exist: Rethinking binarized neural network optimization. arXiv preprint arXiv:1906.02107, 2019.
3. Yin, P. et al., Understanding straight-through estimator in training activation quantized neural nets. arXiv, 2019.

BayesBiNN

- Main point: optimize the parameters of Bernoulli distribution (a continuous optimization problem)
- Problem reformulation: Optimize distribution over weights[1,2]

Loss

$$\min_{q(\boldsymbol{w})} \quad \mathbb{E}_{q(\boldsymbol{w})}\left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))\right] + \mathbb{D}_{KL}[q(\boldsymbol{w}) \,\|\, p(\boldsymbol{w})]$$

Posterior approximation over weights

KL Divergence

Prior Distribution

1. Zellner, A. Optimal information processing and Bayes's theorem. The American Statistician, 42(4):278–280, 1988.
2. Bissiri et al.. A general framework for updating belief distributions. Journal of the Royal Statistical Society, 78(5):1103–1130, 2016.

BayesBiNN

- Main point: optimize the parameters of Bernoulli distribution (a continuous optimization problem)
- Problem reformulation: Optimize distribution over weights[1,2]

$$\min_{q(\boldsymbol{w})} \quad \underbrace{\mathbb{E}_{q(\boldsymbol{w})}}_{\text{Posterior approximation over weights}} \left[\sum_{i=1}^{N} \overbrace{\ell}^{\text{Loss}}(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] + \underbrace{\mathbb{D}_{KL}}_{\text{KL Divergence}}[q(\boldsymbol{w}) \,\|\, \underbrace{p(\boldsymbol{w})}_{\text{Prior Distribution}}]$$

- $q(\boldsymbol{w})$ is chosen to be mean-field Bernoulli distribution

$$q(\boldsymbol{w}) = \prod_{i=1}^{D} \underbrace{p_i}_{\text{Probability of } w_i = +1}^{\frac{1+w_i}{2}} (1 - p_i)^{\frac{1-w_i}{2}} \qquad w_i \in \{-1, +1\}$$

$$\Leftrightarrow$$

$$q(\boldsymbol{w}) = \prod_{i=1}^{D} \exp\left[\lambda_i \phi(w_i) - A(\lambda_i) \right]$$

Natural parameters: $\lambda_i := \frac{1}{2} \log \frac{p_i}{1 - p_i}$

1. Zellner, A. Optimal information processing and Bayes's theorem. The American Statistician, 42(4):278–280, 1988.
2. Bissiri et al.. A general framework for updating belief distributions. Journal of the Royal Statistical Society, 78(5):1103–1130, 2016.

BayesBiNN

- The Bayesian learning rule[1] (natural-gradient variational inference)

Learning rate

$$\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha \left\{ \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] - \boldsymbol{\lambda}_0 \right\}$$

Natural parameter of $q(\boldsymbol{w})$

Expectation parameter of $q(\boldsymbol{w})$

Natural parameter of $p(\boldsymbol{w})$

1. Khan, M. E. and Rue, H. Learning-algorithms from bayesian principles. 2019.
2. Maddison, et al., The concrete distribution: A continuous relaxation of discrete random variables. arXiv:1611.00712, 2016.
3. Jiang et al. Categorical repa-rameterization with gumbel-softmax. arXiv:1611.01144, 2016.

BayesBiNN

- The Bayesian learning rule[1] (natural-gradient variational inference)

Learning rate

$$\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha \left\{ \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{w})} \left[\sum_{i=1}^{N} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \right] - \boldsymbol{\lambda}_0 \right\}$$

How to compute?

Natural parameter of $q(\boldsymbol{w})$

Expectation parameter of $q(\boldsymbol{w})$

Natural parameter of $p(\boldsymbol{w})$

1. Khan, M. E. and Rue, H. Learning-algorithms from bayesian principles. 2019.
2. Maddison, et al., The concrete distribution: A continuous relaxation of discrete random variables. arXiv:1611.00712, 2016.
3. Jiang et al. Categorical repa-rameterization with gumbel-softmax. arXiv:1611.01144, 2016.

BayesBiNN

- The Bayesian learning rule[1] (natural-gradient variational inference)

Learning rate

$$\boldsymbol{\lambda} \leftarrow (1-\alpha)\boldsymbol{\lambda} - \alpha\left\{\nabla_{\boldsymbol{\mu}}\mathbb{E}_{q(\boldsymbol{w})}\left[\sum_{i=1}^{N}\ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))\right] - \boldsymbol{\lambda}_0\right\}$$

How to compute?

Natural parameter of $q(\boldsymbol{w})$

Expectation parameter of $q(\boldsymbol{w})$

Natural parameter of $p(\boldsymbol{w})$

- Using the Gumbel Softmax trick[2,3], we can approximate the natural gradient by using the mini-batch gradient

$$\nabla_{\boldsymbol{\mu}}\mathbb{E}_{q(\boldsymbol{w})}\left[\sum_{i=1}^{N}\ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i))\right] \approx \mathbf{s}\odot\mathbf{g}$$

Minibatch Gradient, easy to compute!

Scale vector

$$\mathbf{g} := \frac{1}{m}\sum_{i\in\mathcal{M}}\nabla_{\boldsymbol{w}_b}\ell(y_i, f_{\boldsymbol{w}_b}(\boldsymbol{x}_i)) \qquad \mathbf{s} := \frac{N(1-\boldsymbol{w}_b^2)}{\tau(1-\tanh(\boldsymbol{\lambda})^2)}$$

1. Khan, M. E. and Rue, H. Learning-algorithms from bayesian principles. 2019.
2. Maddison, et al., The concrete distribution: A continuous relaxation of discrete random variables. arXiv:1611.00712, 2016.
3. Jiang et al. Categorical repa-rameterization with gumbel-softmax. arXiv:1611.01144, 2016.

BayesBiNN Justifies Some Previous Methods

STE	Our BayesBiNN method	Bop
$\boldsymbol{w}_b \leftarrow \mathrm{sign}(\boldsymbol{w}_r)$ $\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$ $\boldsymbol{w}_r \leftarrow \boldsymbol{w}_r - \alpha\mathbf{g}$	$\boldsymbol{w}_b \leftarrow \tanh\left((\boldsymbol{w}_r + \boldsymbol{\delta})/\tau\right)$ $\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$ $\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha\mathbf{s}\odot\mathbf{g}$	$\boldsymbol{w}_b \leftarrow \mathrm{hyst}(\boldsymbol{w}_r, \boldsymbol{w}_b, \gamma)$ $\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b}\ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$ $\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha\mathbf{g}$

Note that $\boldsymbol{w}_r$ in BayesBiNN corresponds to $\boldsymbol{\lambda}$

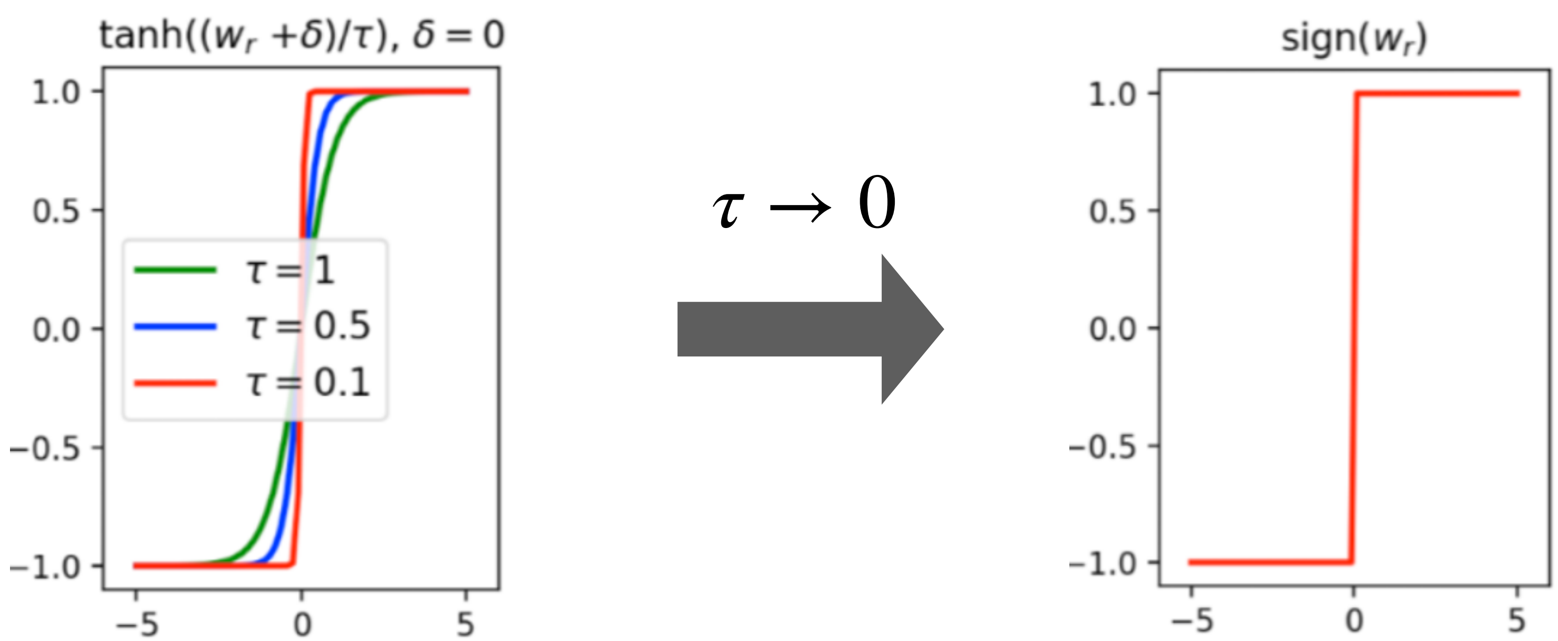
- Main point 1: STE works as a special case of BayesBiNN as $\tau \to 0$

BayesBiNN Justifies Some Previous Methods

STE	Our BayesBiNN method	Bop
$\boxed{\boldsymbol{w}_b \leftarrow \text{sign}(\boldsymbol{w}_r)}$	$\boxed{\boldsymbol{w}_b \leftarrow \tanh\left((\boldsymbol{w}_r + \boldsymbol{\delta})/\tau\right)}$	$\boldsymbol{w}_b \leftarrow \text{hyst}(\boldsymbol{w}_r, \boldsymbol{w}_b, \gamma)$
$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$
$\boldsymbol{w}_r \leftarrow \boldsymbol{w}_r - \alpha \mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{s} \odot \mathbf{g}$	$\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{g}$

Note that $\boldsymbol{w}_r$ in BayesBiNN corresponds to $\boldsymbol{\lambda}$

- Main point 1: STE works as a special case of BayesBiNN as $\tau \to 0$

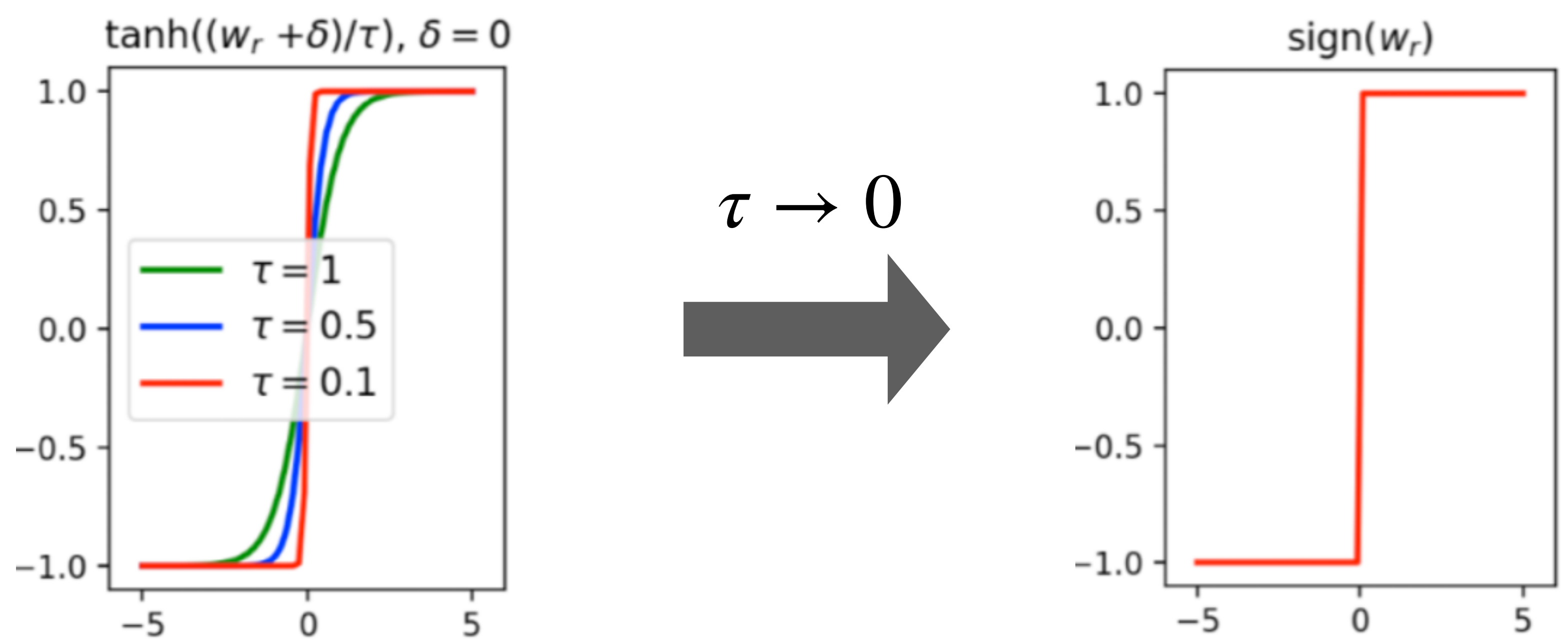


BayesBiNN Justifies Some Previous Methods

STE	Our BayesBiNN method	Bop
$\boxed{\boldsymbol{w}_b \leftarrow \mathrm{sign}(\boldsymbol{w}_r)}$	$\boxed{\boldsymbol{w}_b \leftarrow \tanh\left((\boldsymbol{w}_r + \boldsymbol{\delta})/\tau\right)}$	$\boldsymbol{w}_b \leftarrow \mathrm{hyst}(\boldsymbol{w}_r, \boldsymbol{w}_b, \gamma)$
$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$	$\mathbf{g} \leftarrow \nabla_{\boldsymbol{w}_b} \ell(y, f_{\boldsymbol{w}_b}(\boldsymbol{x}))$
$\boldsymbol{w}_r \leftarrow \boldsymbol{w}_r - \alpha \mathbf{g}$	$\boxed{\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{s} \odot \mathbf{g}}$	$\boxed{\boldsymbol{w}_r \leftarrow (1-\alpha)\boldsymbol{w}_r - \alpha \mathbf{g}}$

Note that $\boldsymbol{w}_r$ in BayesBiNN corresponds to $\boldsymbol{\lambda}$

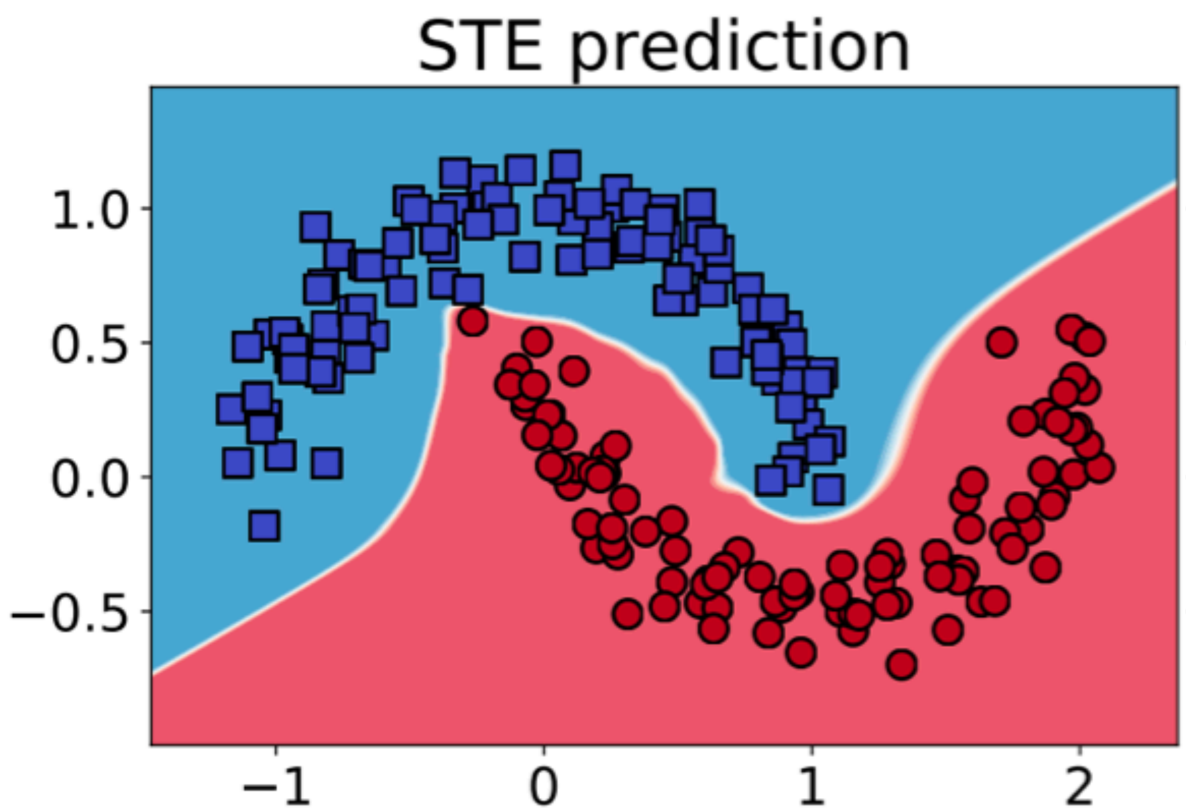
- Main point 1: STE works as a special case of BayesBiNN as $\tau \to 0$

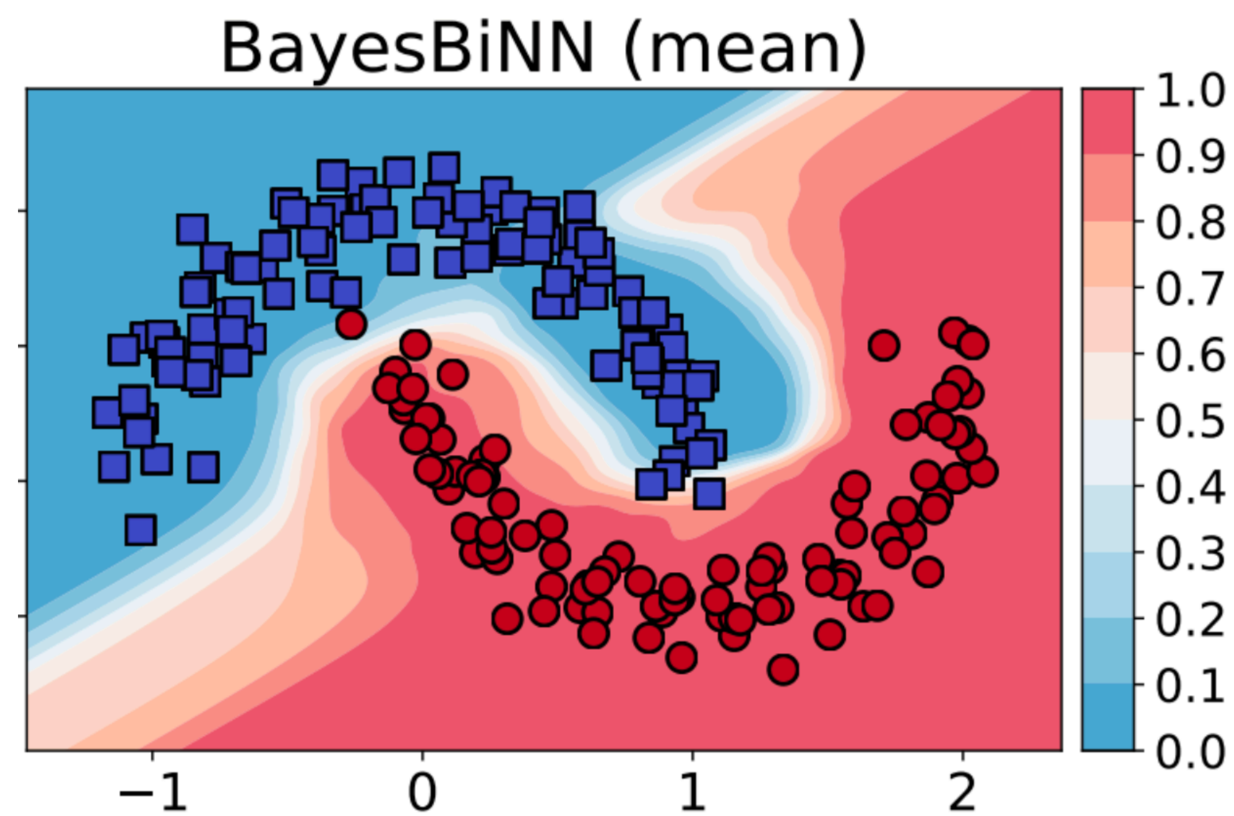


- Main point 2: Justify the “exponential average” used in Bop

Uncertainty Estimation

- Main point: BayesBiNN obtains uncertainty estimates around the classification boundaries





Classification on two moons dataset

$\hat{p}_k \leftarrow \frac{1}{C}\sum_{c=1}^{C} p\left(y=k \mid \boldsymbol{x}, \boldsymbol{w}^{(c)}\right), C=10 \quad \boldsymbol{w}^{(c)} \sim q(\boldsymbol{w})$

- STE finds a deterministic boundary

- **Open-source Code Available** : https://github.com/team-approx-bayes/BayesBiNN

BayesBiNN $\approx$ STE

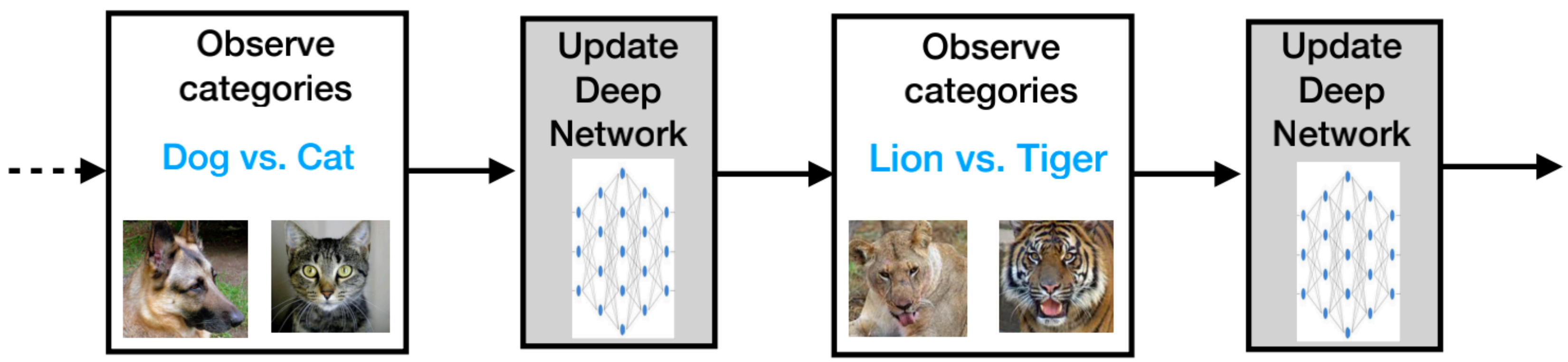
- **Open-source Code Available** : https://github.com/team-approx-bayes/BayesBiNN

Table 2. Results of different optimizers trained on MNIST, CIFAR-10 and CIFAR-100 (Averaged over 5 runs).

Dataset	Optimizer	Train Accuracy	Validation Accuracy	Test Accuracy
MNIST	STE Adam	99.78 ± 0.10 %	99.02 ± 0.11 %	$\mathbf{98.85} \pm 0.09$ %
	Bop	99.23 ± 0.04 %	98.55 ± 0.05 %	98.47 ± 0.02 %
	PMF		99.06 ± 0.01 %	98.80 ± 0.06 %
	BayesBiNN (mode)	99.85 ± 0.05 %	99.02 ± 0.13 %	$\mathbf{98.86} \pm 0.05$ %
	BayesBiNN (mean)	99.85 ± 0.05 %	99.02 ± 0.13 %	$\mathbf{98.86} \pm 0.05$ %
	Full-precision	99.96 ± 0.02 %	99.15 ± 0.14 %	99.01 ± 0.06 %
CIFAR-10	STE Adam	99.99 ± 0.01 %	94.25 ± 0.42 %	$\mathbf{93.55} \pm 0.15$ %
	Bop	99.79 ± 0.03 %	93.49 ± 0.17 %	93.00 ± 0.11 %
	PMF		91.87 ± 0.10 %	91.43 ± 0.14 %
	BayesBiNN (mode)	99.96 ± 0.01 %	94.23 ± 0.41 %	$\mathbf{93.72} \pm 0.16$ %
	BayesBiNN (mean)	99.96 ± 0.01 %	94.23 ± 0.41 %	$\mathbf{93.72} \pm 0.15$ %
	Full-precision	100.00 ± 0.00 %	94.54 ± 0.29 %	93.90 ± 0.17 %
CIFAR-100	STE Adam	99.06 ± 0.15 %	74.09 ± 0.15 %	72.89 ± 0.21 %
	Bop	90.09 ± 0.57 %	69.97 ± 0.29 %	69.58 ± 0.15 %
	PMF		69.86 ± 0.08 %	70.45 ± 0.25 %
	BayesBiNN (mode)	98.02 ± 0.18 %	74.76 ± 0.41 %	$\mathbf{73.68} \pm 0.31$ %
	BayesBiNN (mean)	98.02 ± 0.18 %	74.76 ± 0.41 %	$\mathbf{73.65} \pm 0.41$ %
	Full-precision	99.89 ± 0.02 %	75.89 ± 0.41 %	74.83 ± 0.26 %

Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN enables continual learning (CL) for BiNN using the intrinsic KL divergence as regularization
- CL: Sequentially learning new tasks without forgetting old ones[1]



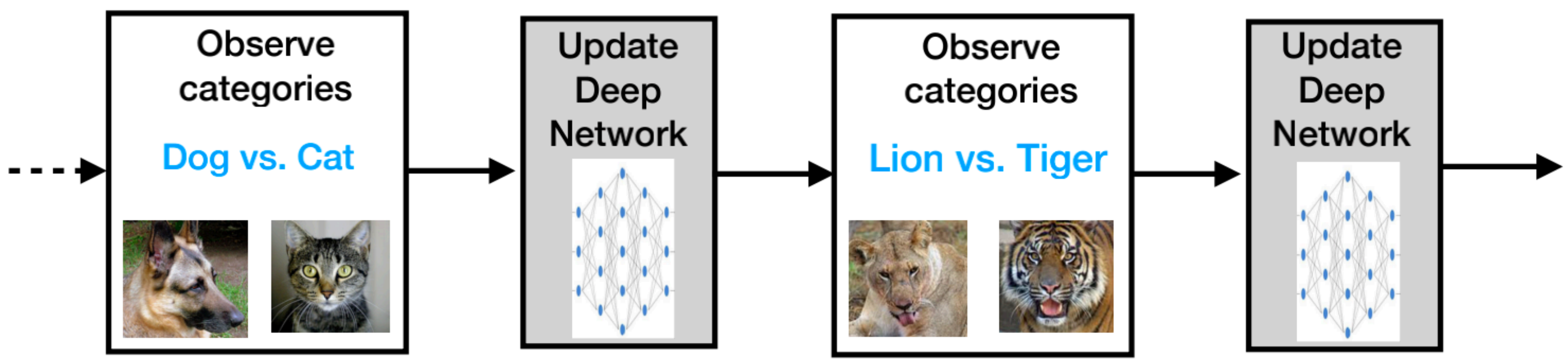
Overcoming catastrophic forgetting

Common Method:
Regularizing weights

- But, it is unclear how to regularize binary weights of BiNN using STE/Bop

Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN enables continual learning (CL) for BiNN using the intrinsic KL divergence as regularization
- CL: Sequentially learning new tasks without forgetting old ones[1]



Overcoming catastrophic forgetting

Common Method:
Regularizing weights

- But, it is unclear how to regularize binary weights of BiNN using STE/Bop
- In BayesBiNN, there is one natural solution using KL divergence

Independent Learning

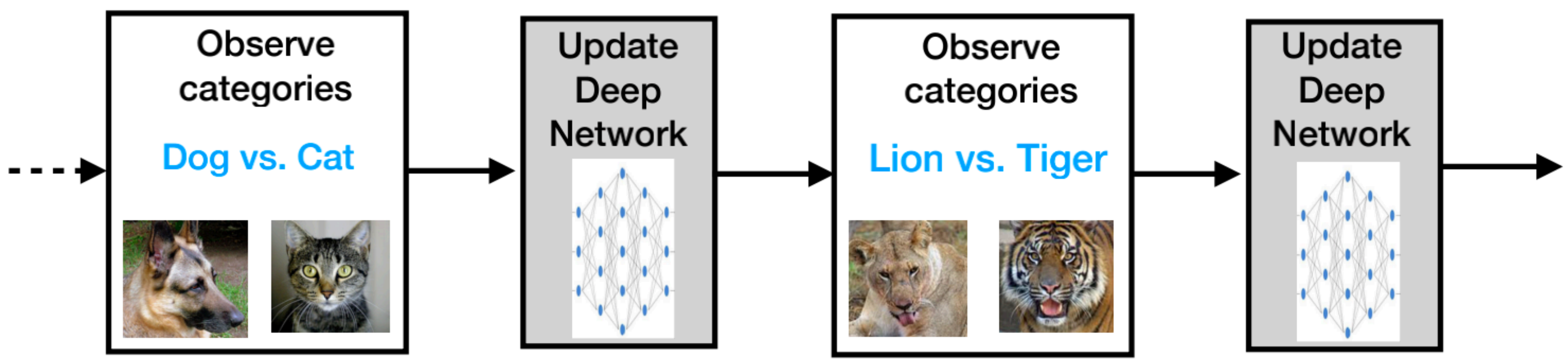
$$\min_{q_t(\mathbf{w})} \mathbb{E}_{q_t(\mathbf{w})}\left[\sum_{i\in D^t} \ell(y_i^t, f_{\mathbf{w}}(\mathbf{x}_i^t))\right] + \mathbb{D}_{KL}\left(q_t(\mathbf{w})\,||\,p(\mathbf{w})\right)$$

Prior Distribution (uniform)

1. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. PANS, 114(13):3521–3526, 2017.

Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN enables continual learning (CL) for BiNN using the intrinsic KL divergence as regularization
- CL: Sequentially learning new tasks without forgetting old ones[1]



Overcoming catastrophic forgetting

Common Method:
Regularizing weights

- But, it is unclear how to regularize binary weights of BiNN using STE/Bop
- In BayesBiNN, there is one natural solution using KL divergence

Continual Learning

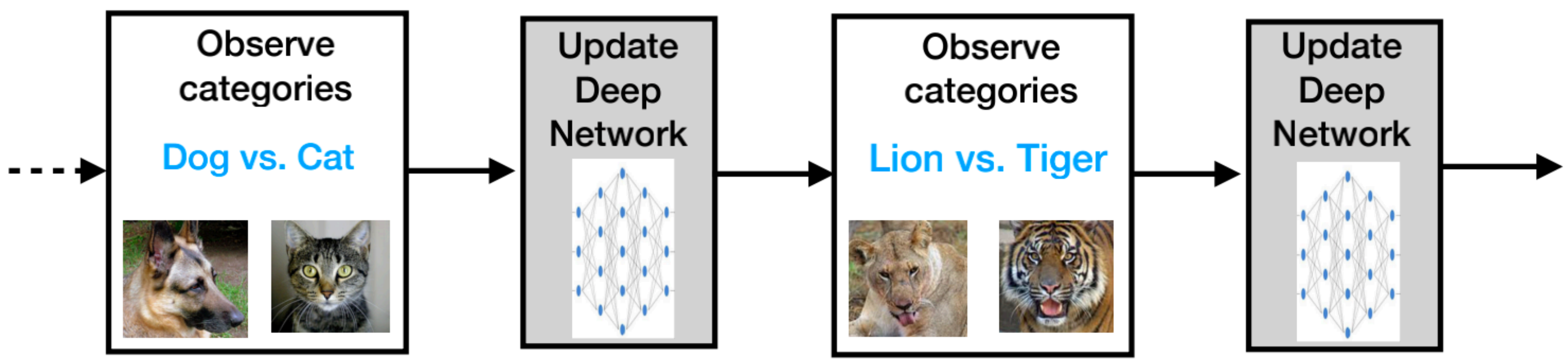
$$\min_{q_t(\boldsymbol{w})} \mathbb{E}_{q_t(\boldsymbol{w})}\left[\sum_{i \in D^t} \ell(y_i^t, f_{\boldsymbol{w}}(\boldsymbol{x}_i^t))\right] + \mathbb{D}_{KL}\left(q_t(\boldsymbol{w}) \,||\, q_{t-1}(\boldsymbol{w})\right)$$

posterior approximation after task $t-1$

1. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. PANS, 114(13):3521–3526, 2017.

Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN enables continual learning (CL) for BiNN using the intrinsic KL divergence as regularization
- CL: Sequentially learning new tasks without forgetting old ones[1]



Overcoming catastrophic forgetting

Common Method:
Regularizing weights

- But, it is unclear how to regularize binary weights of BiNN using STE/Bop
- In BayesBiNN, there is one natural solution using KL divergence

Continual Learning

$$\min_{q_t(\mathbf{w})} \mathbb{E}_{q_t(\mathbf{w})}\left[\sum_{i\in D^t} \ell(y_i^t, f_{\mathbf{w}}(\mathbf{x}_i^t))\right] + \mathbb{D}_{KL}\left(q_t(\mathbf{w})\,||\,q_{t-1}(\mathbf{w})\right)$$

posterior approximation after task $t-1$

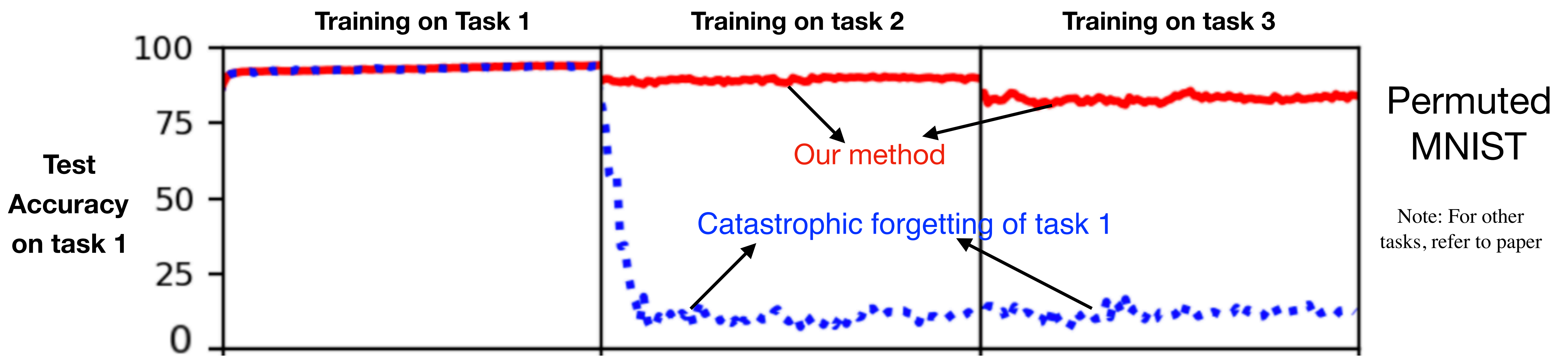
$$\lambda_t \leftarrow (1-\alpha)\lambda_t - \alpha\left[\mathbf{s}\odot\mathbf{g} - \lambda_{t-1}\right]$$

Learned natural parameter after task $t-1$

1. Kirkpatrick, J. et al. Overcoming catastrophic forgetting in neural networks. PANS, 114(13):3521–3526, 2017.

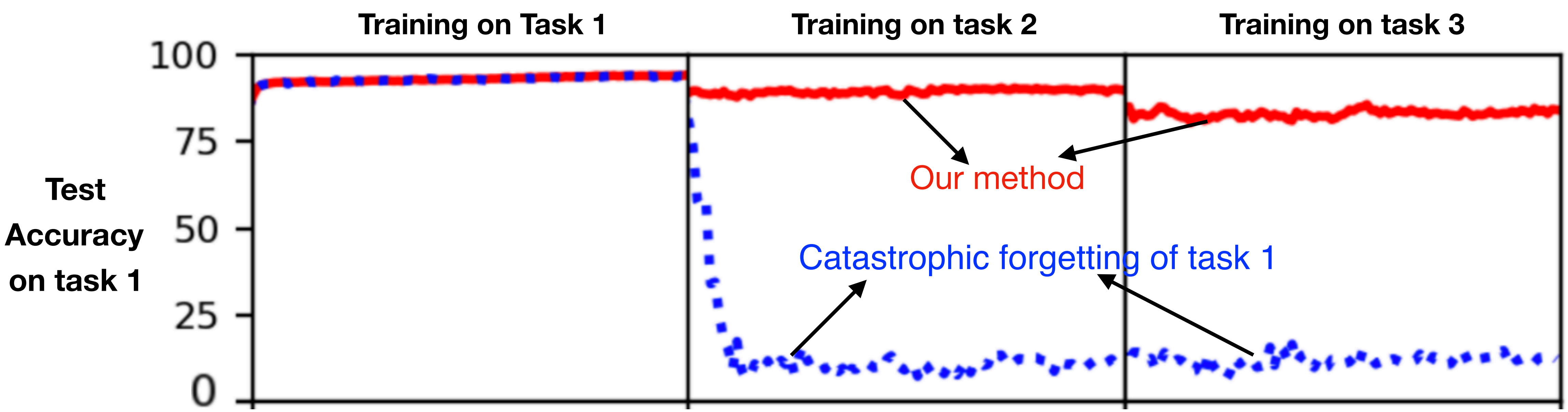
Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN avoids the catastrophic forgetting problem



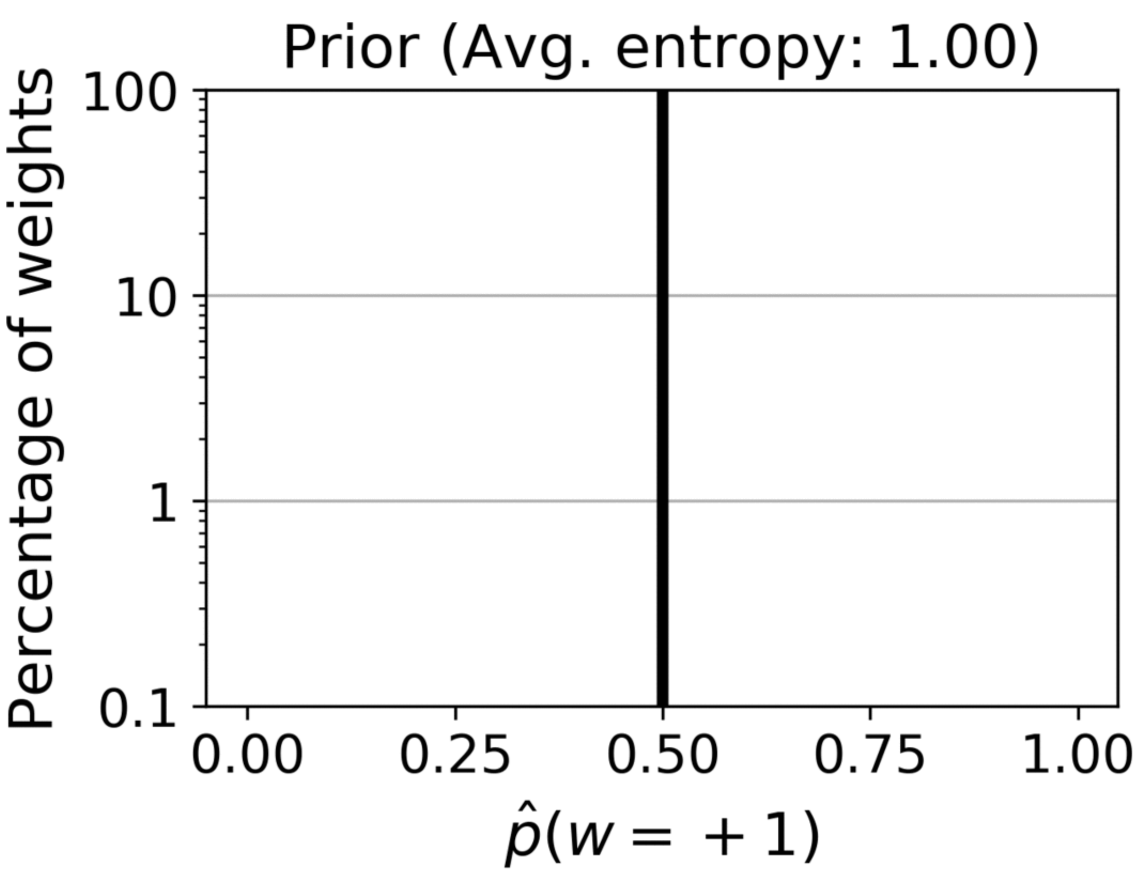
Uncertainty Provided by BayesBiNN Enables Continual Learning

- Main point: BayesBiNN avoids the catastrophic forgetting problem



Permuted MNIST

Note: For other tasks, refer to paper



- As the number of tasks increases, the distribution over binary weights become more and more deterministic

- Open-source Code Available : https://github.com/team-approx-bayes/BayesBiNN

Summary

- BiNN: Neural Networks with binary weights
- Much faster and much smaller but difficult to optimize
- Gradient based methods work well but not well understood
- We proposed a principled approach to train BiNN using the Bayesian Learning Rule, which can justify some previous approaches
- The Bayesian approach also gives us estimate of uncertainty which can be used for continual learning

Thank you!
Q&A