

Stochastic variational inference and Bayesian neural networks

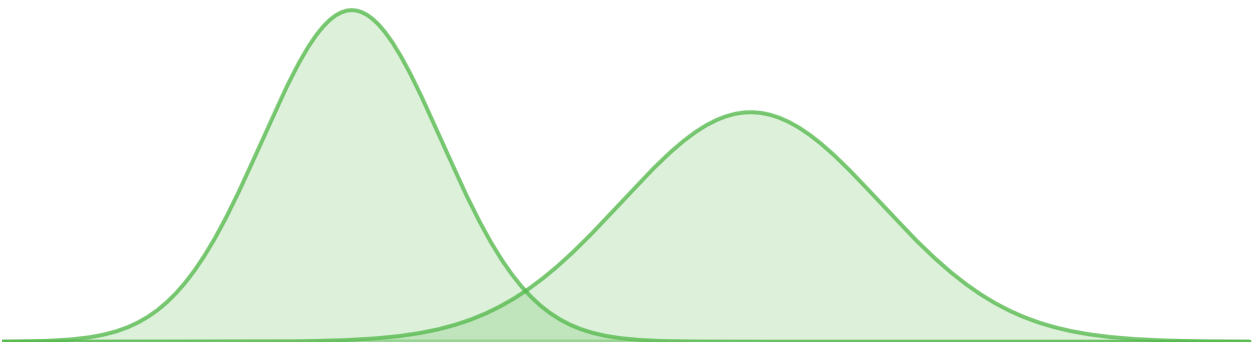
Nadezhda Chirkova

Higher School of Economics, Samsung-HSE Laboratory
Moscow, Russia



NATIONAL RESEARCH
UNIVERSITY

SAMSUNG
Research



$p(\mathbf{B}|\mathbf{A})$ **yesgroup.ru**

Plan

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- First example: binary dropout
- Second example: Bayesian sparsification

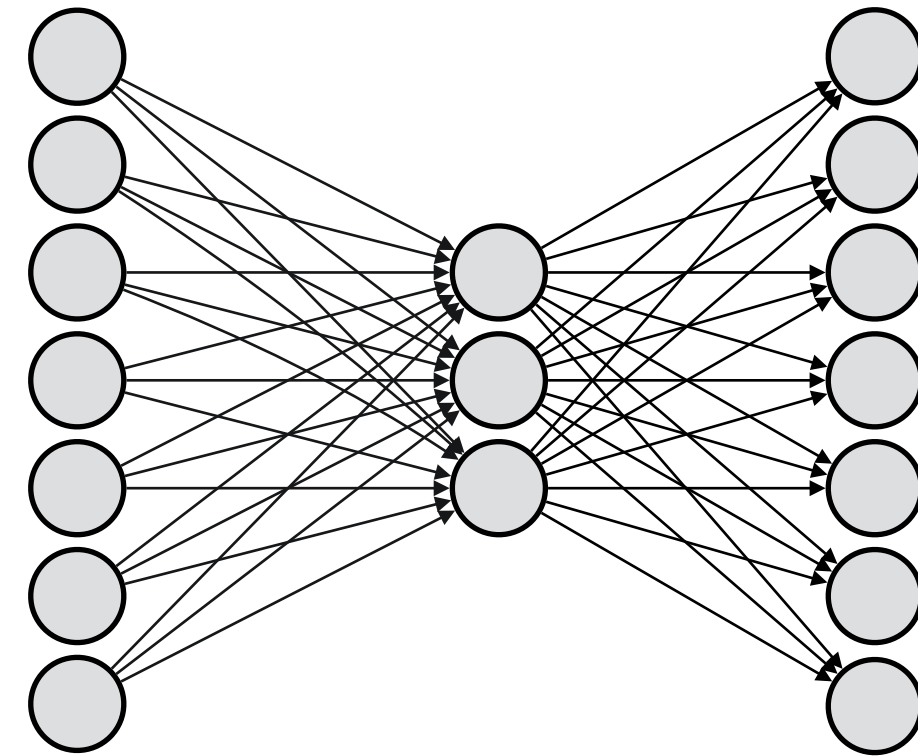
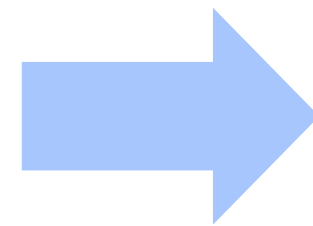
Plan

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- First example: binary dropout
- Second example: Bayesian sparsification

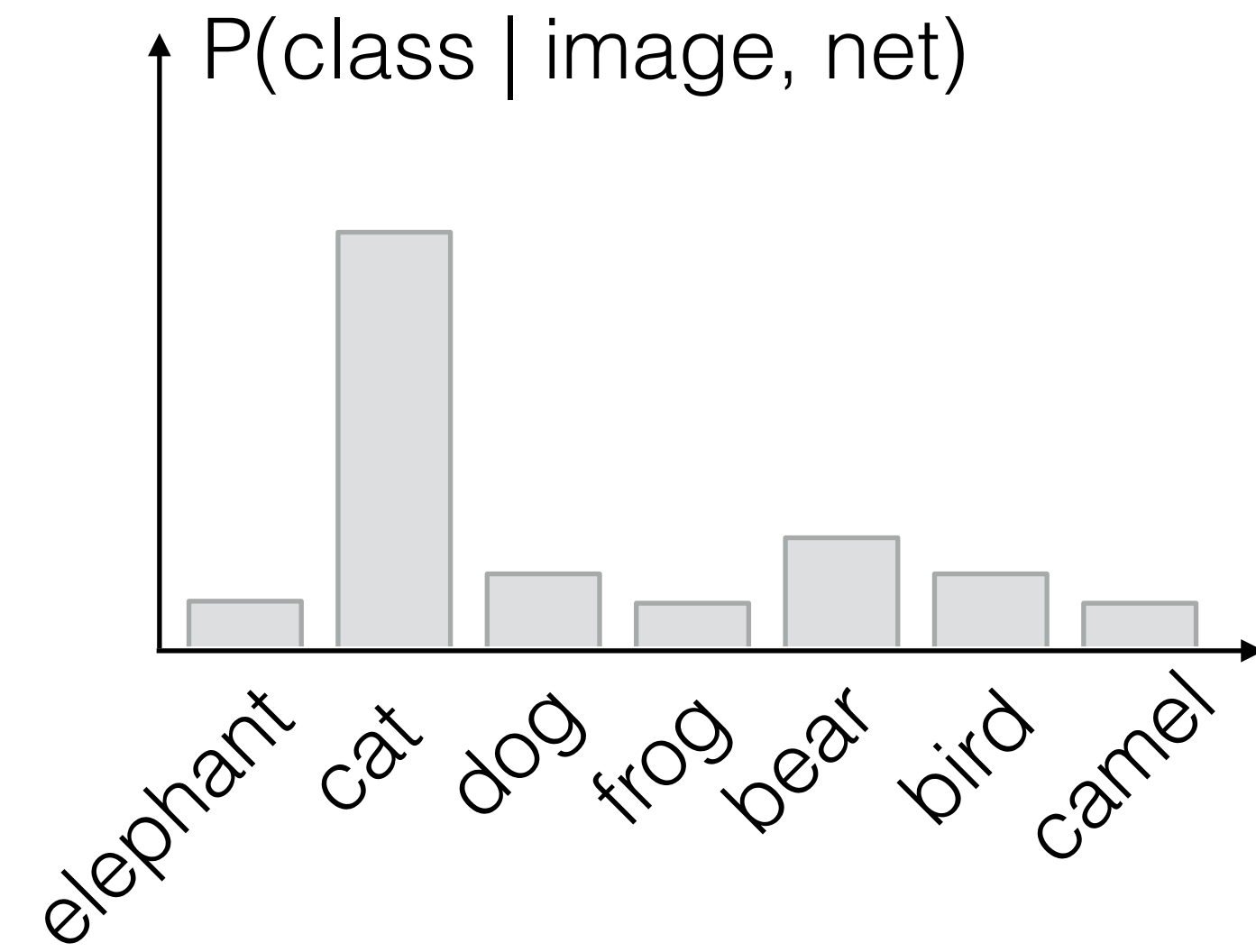
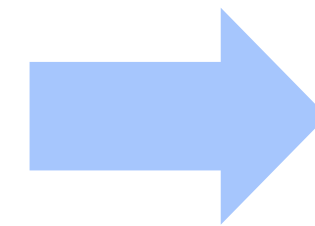
Neural networks



input x



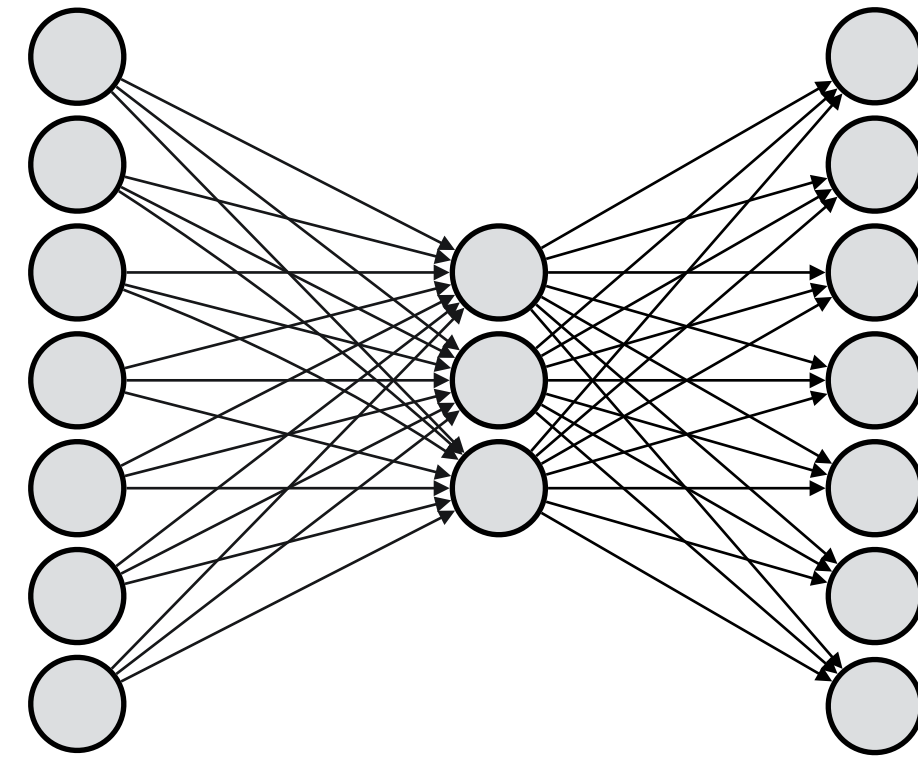
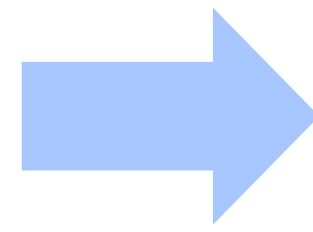
neural network
with weights w



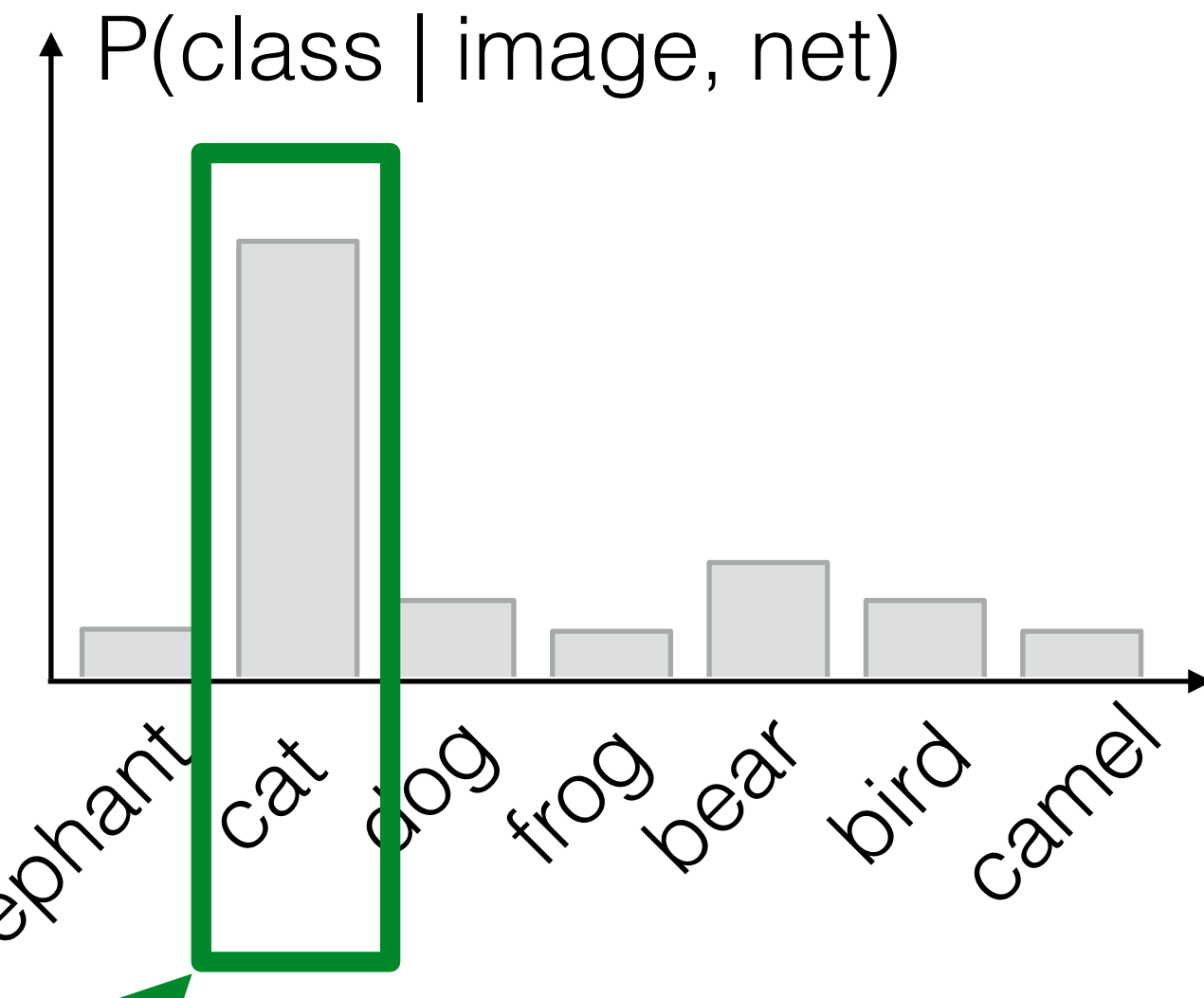
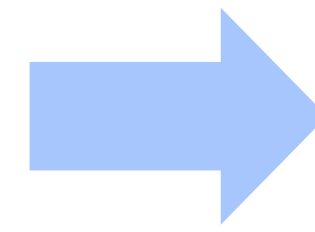
Neural networks



input x



neural network
with weights w

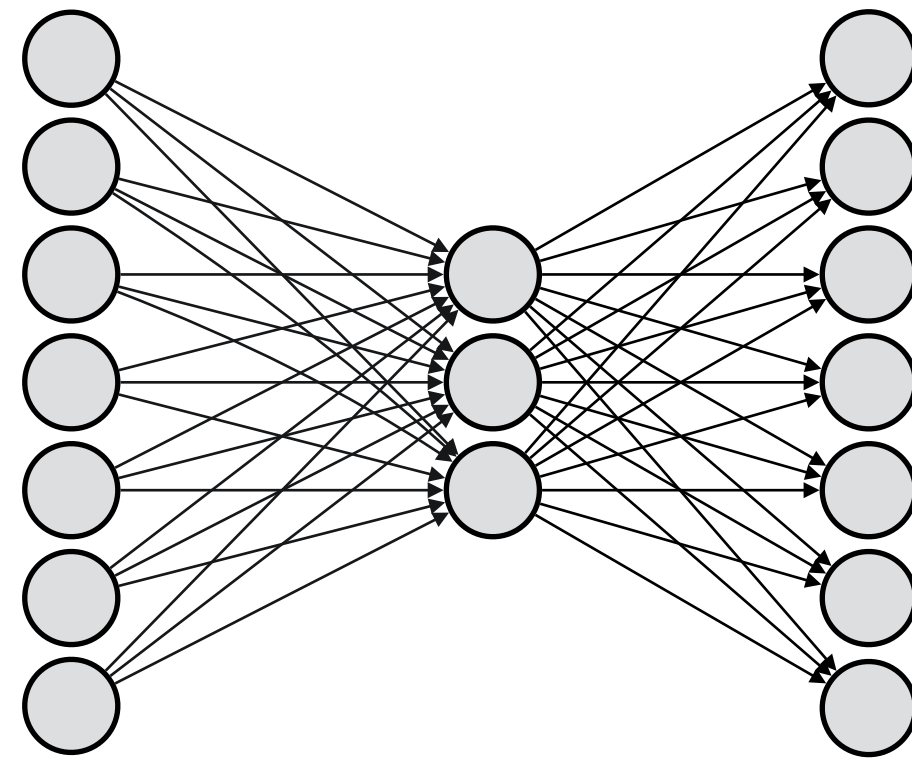
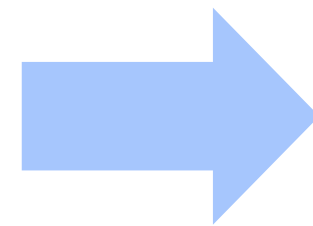


$$p(y = \text{"cat"} | x, w)$$

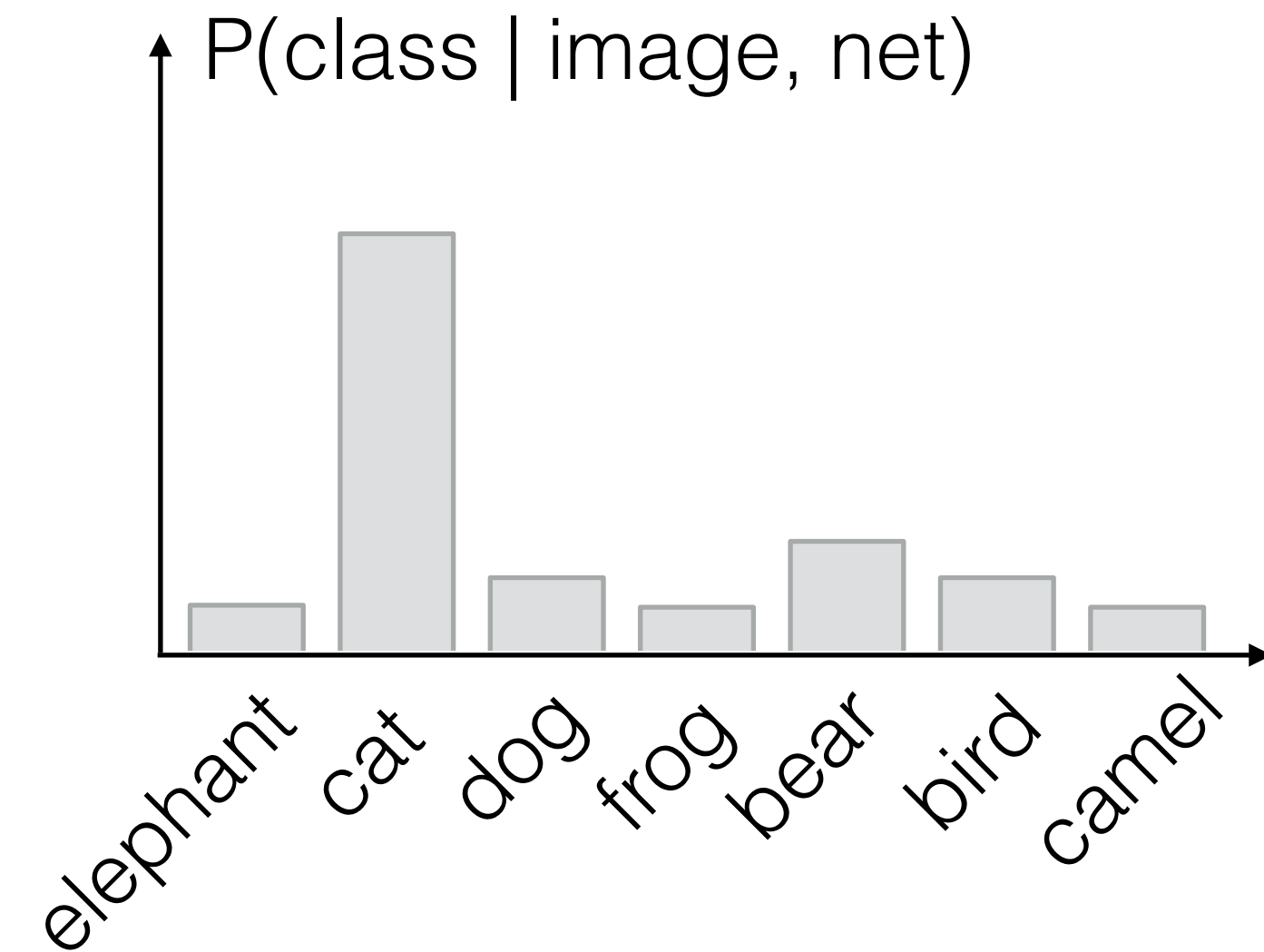
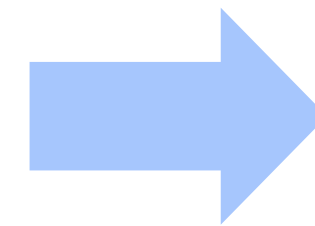
Neural networks



input x



neural network
with weights w



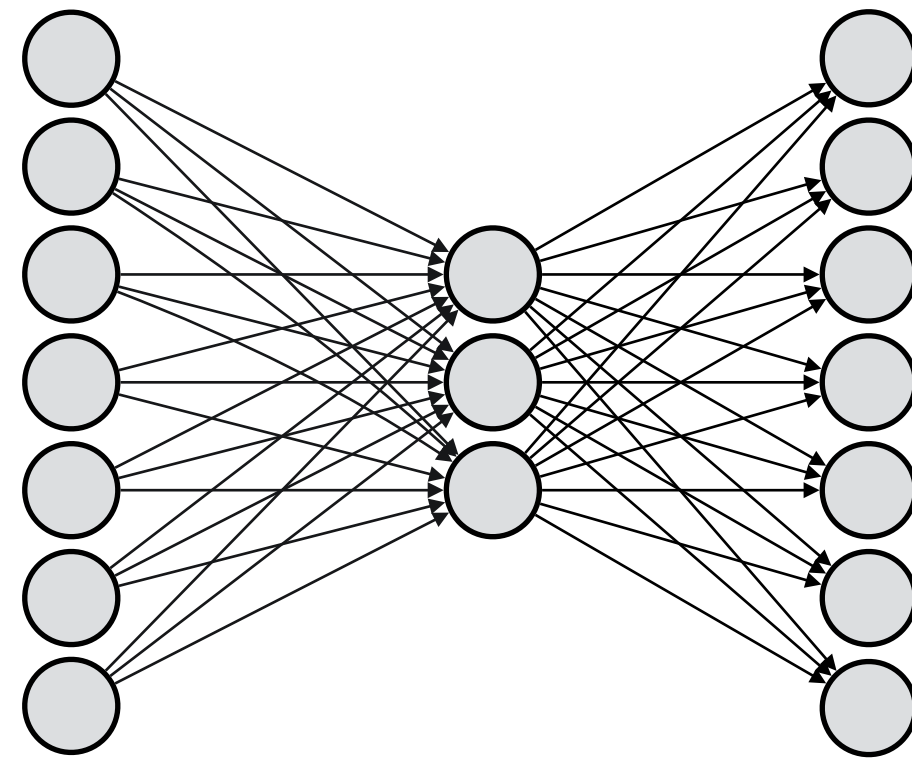
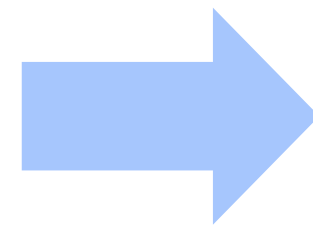
Training — optimization
over weights w
using stochastic
gradient descend:

$$-DataLoss(X, Y, w) \rightarrow \max_w$$

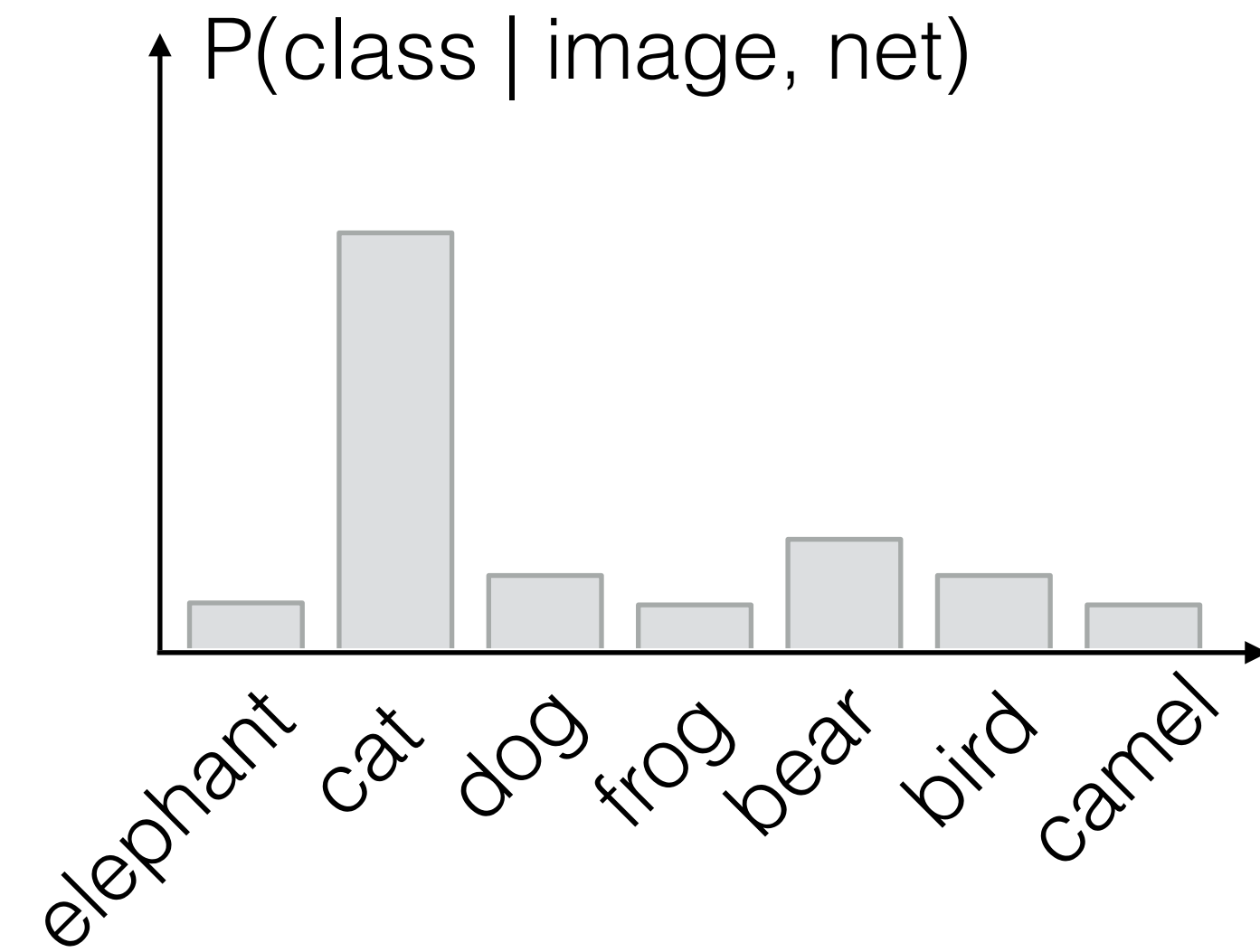
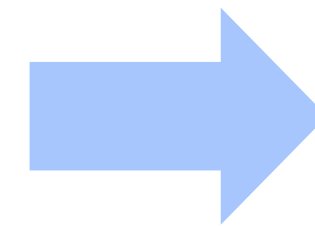
Neural networks



input x



neural network
with weights w



Training — optimization
over weights w
using stochastic
gradient descend:

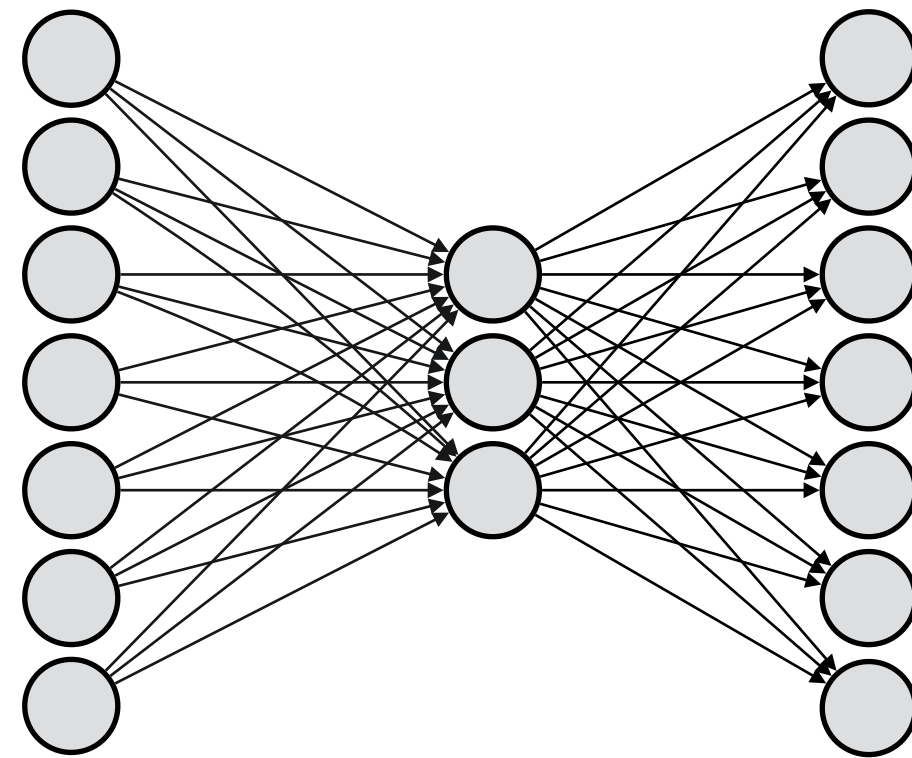
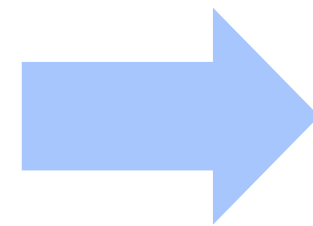
$$\sum_{i=1}^N \log p(y^i | x^i, w) \rightarrow \max_w$$

sum over objects (images) probability of true class

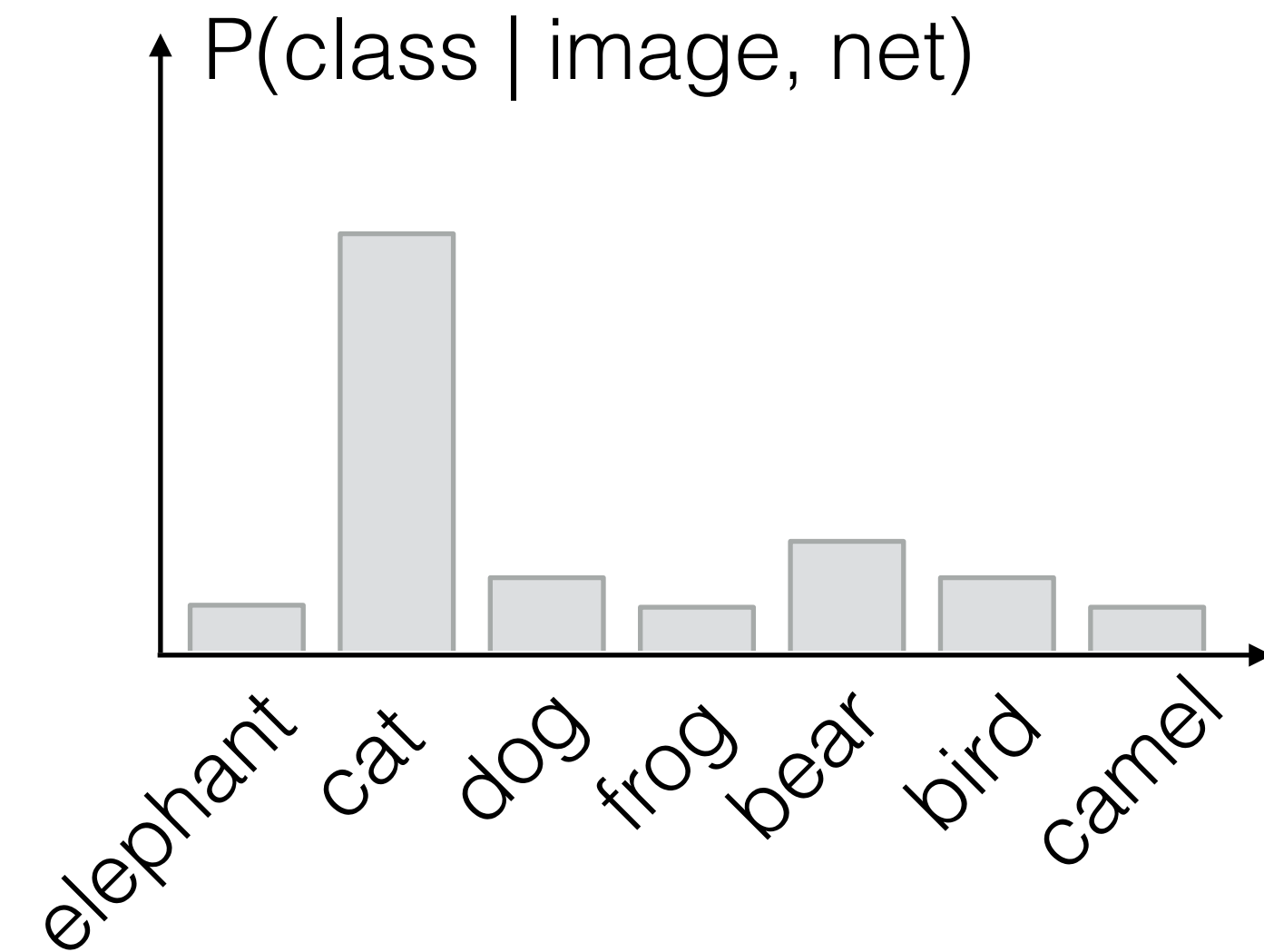
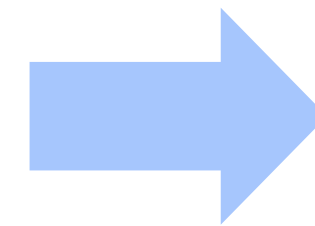
Neural networks



input x



neural network
with weights w



Training — optimization
over weights w
using stochastic
gradient descend:

$$w^{new} = w^{old} + \eta \frac{\partial}{\partial w} \sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, w^{old})$$

$$i_j \sim \text{Unif}(1, \dots, N)$$

m — mini-batch size η — learning rate

Regularization by noise

- Traditional regularization: add some penalty for model complexity
 - L_2 , L_1 - regularization, max norm constraint

$$Objective = DataLoss(X, Y, w) + Regularizer(w)$$

- More recent approaches: regularization by noise
 - Data augmentation, dropout, gradient noise

$$Objective = \mathbb{E}_{p(\Omega)} DataLoss(X, Y, w, \Omega)$$

Bayesian framework provides a principled approach to training with noise!

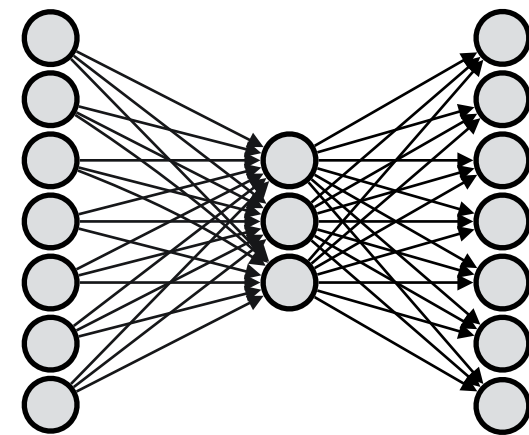
Bayesian neural networks

Deterministic neural network

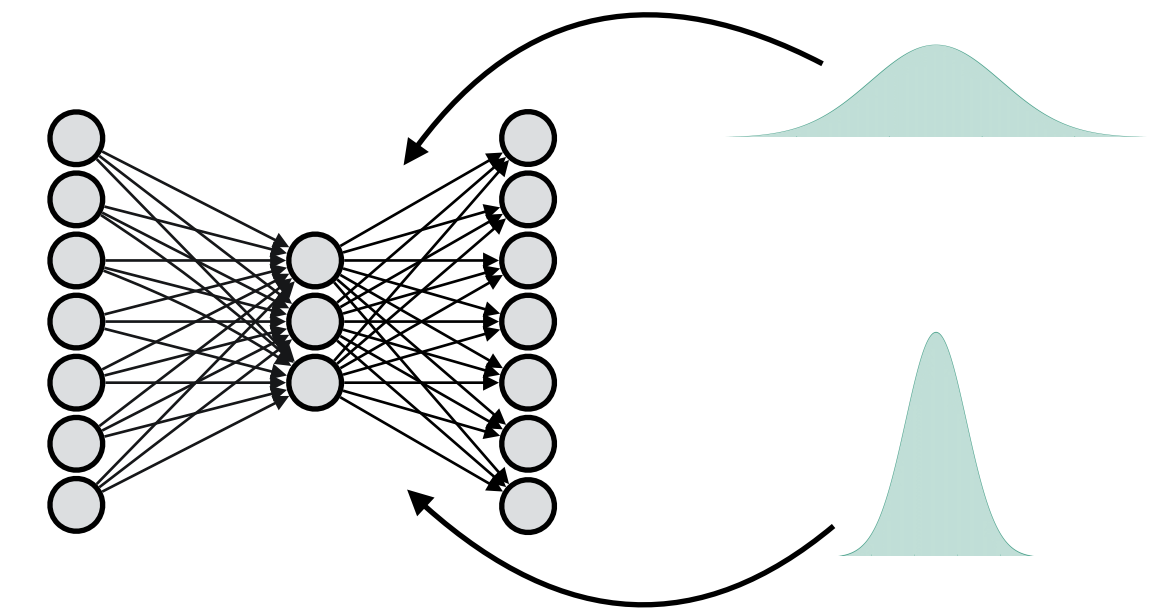
Bayesian neural network

Weights:

deterministic weights



stochastic weights



Training:

Data
 X, Y



point estimate
 w

Prior
 $p(w)$



Data
 X, Y



Posterior
 $p(w|X, Y)$

Prediction
on new
object x_* :

$$p(y_* | x_*, w)$$

$$\mathbb{E}_{p(w|X, Y)} p(y_* | x_*, w)$$

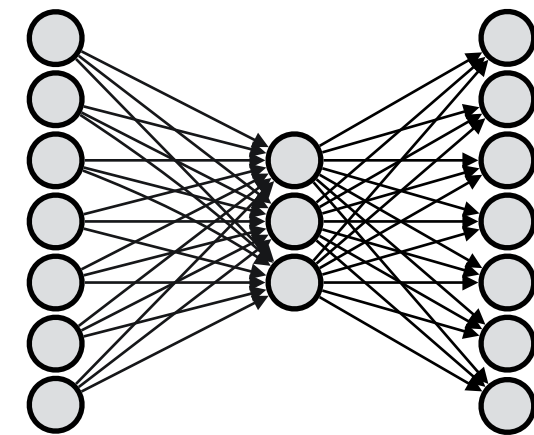
Bayesian neural networks

Deterministic neural network

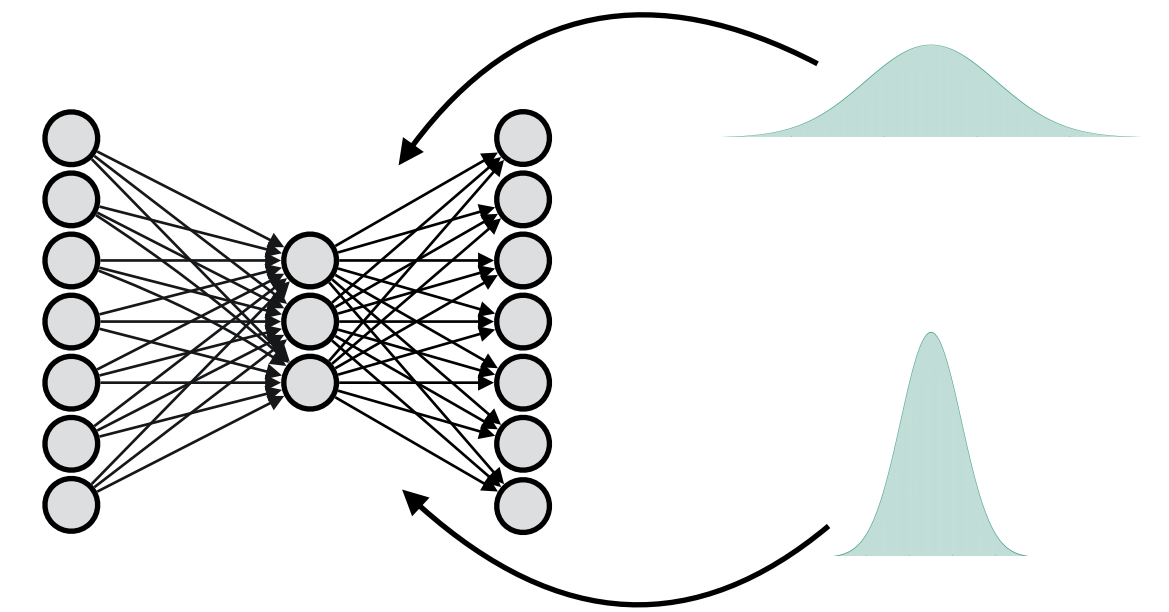
Bayesian neural network

Weights:

deterministic weights



stochastic weights



Training:

Data
 X, Y

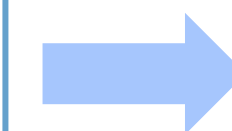


point estimate
 w

Prior
 $p(w)$



Data
 X, Y



Posterior
 $p(w|X, Y)$

Prediction
on new
object x_* :

$$p(y_* | x_*, w)$$

$$\mathbb{E}_{p(w|X, Y)} p(y_* | x_*, w)$$

BNN as an ensemble of neural networks

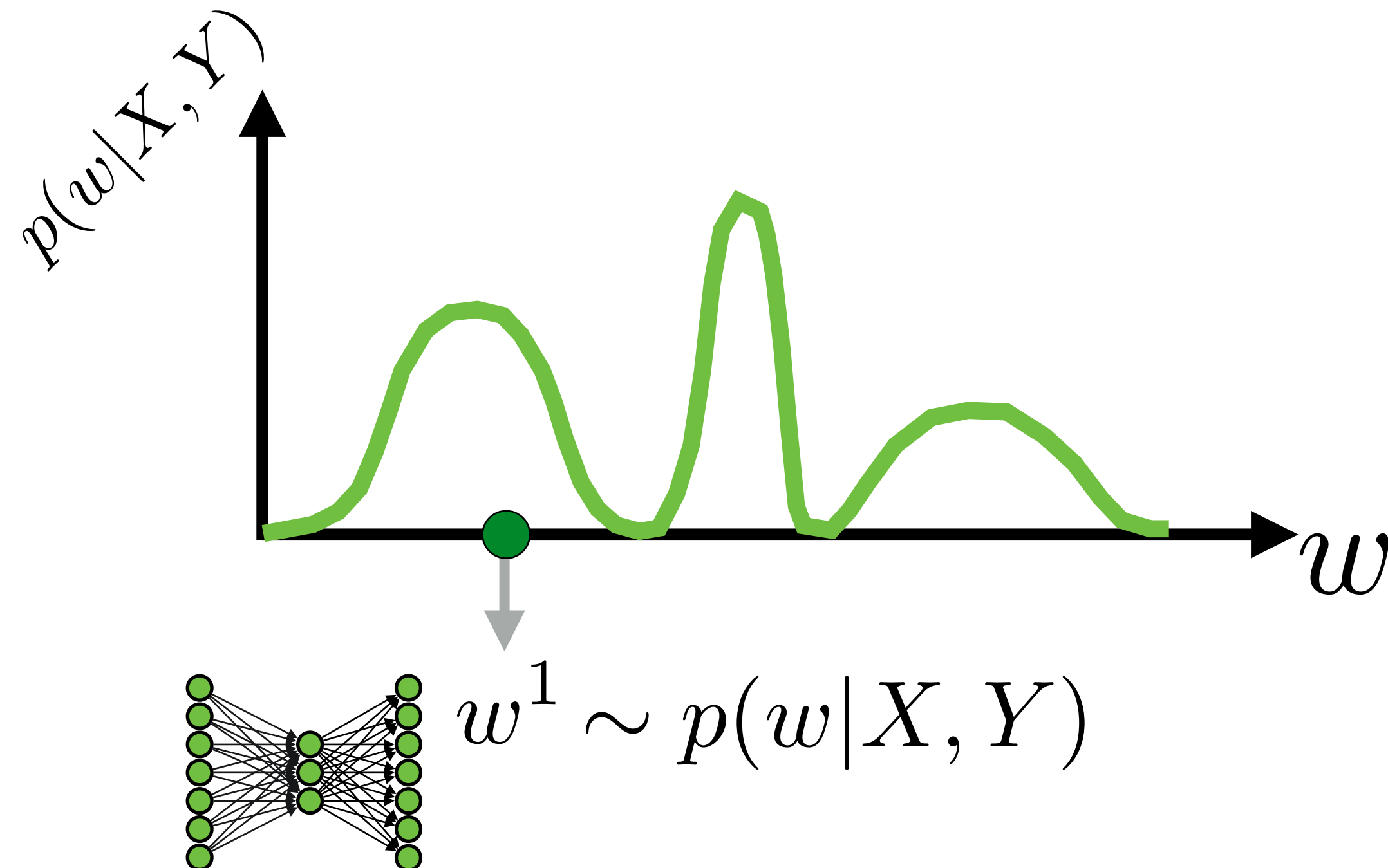
Prediction on a new object x_* :

$$\mathbb{E}_{p(w|X,Y)} p(y_* | x_*, w) \approx \frac{1}{K} \sum_{k=1}^K p(y_* | x_*, w^k), \quad w^k \sim p(w|X, Y)$$

BNN as an ensemble of neural networks

Prediction on a new object x_* :

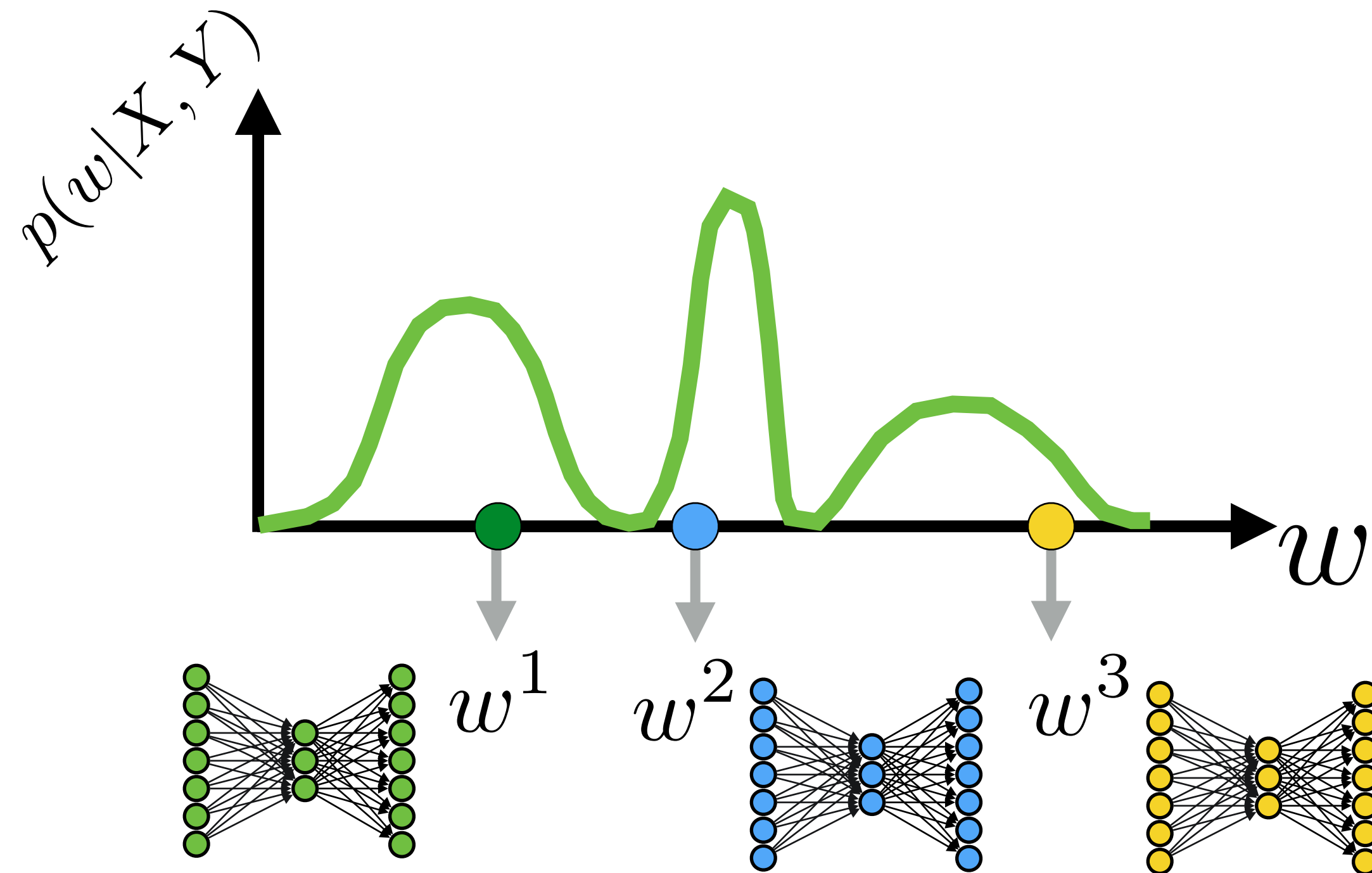
$$\mathbb{E}_{p(w|X, Y)} p(y_* | x_*, w) \approx \frac{1}{K} \sum_{k=1}^K p(y_* | x_*, w^k), \quad w^k \sim p(w|X, Y)$$



BNN as an ensemble of neural networks

Prediction on a new object x_* :

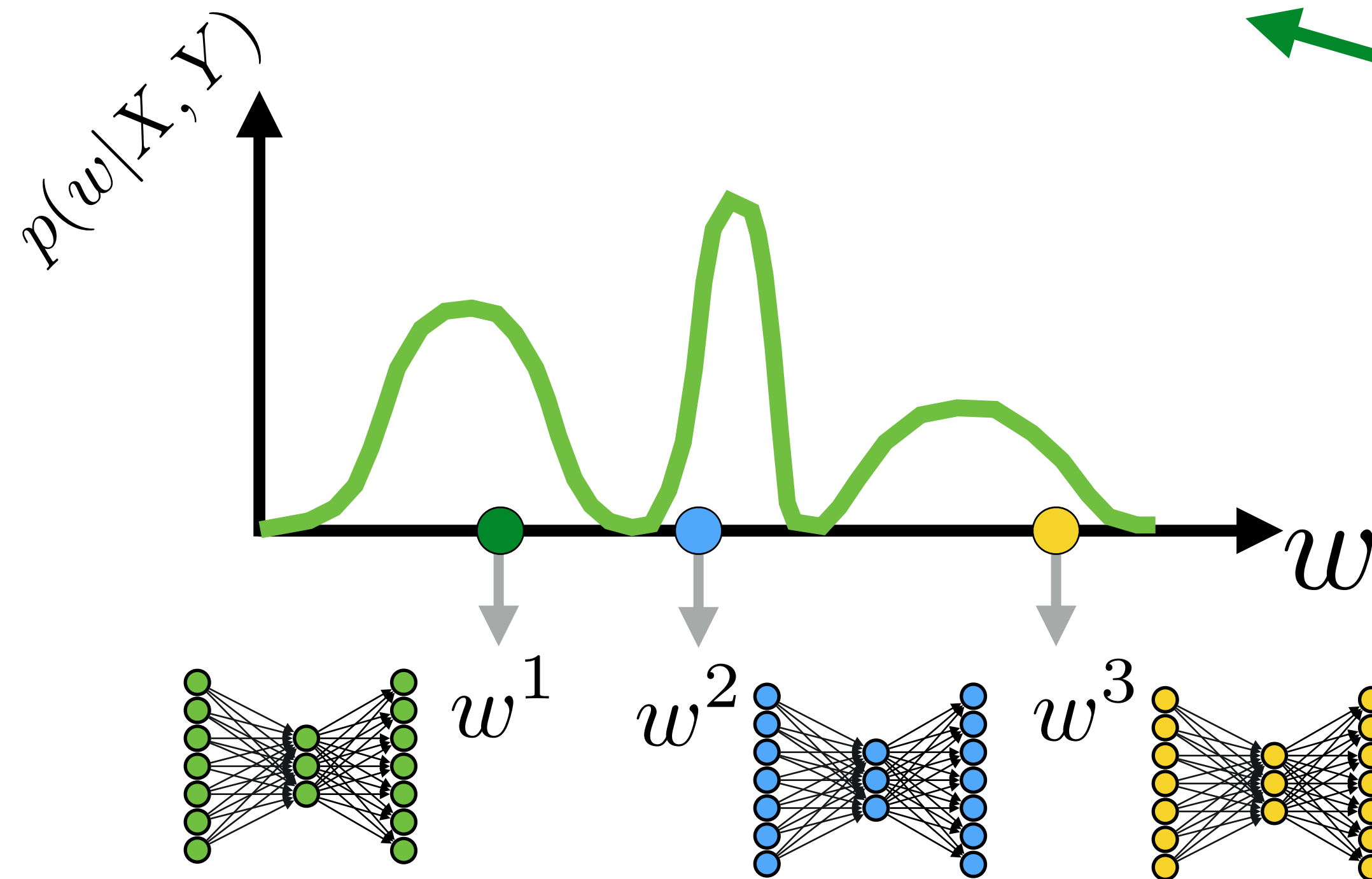
$$\mathbb{E}_{p(w|X,Y)} p(y_* | x_*, w) \approx \frac{1}{K} \sum_{k=1}^K p(y_* | x_*, w^k), \quad w^k \sim p(w|X, Y)$$



BNN as an ensemble of neural networks

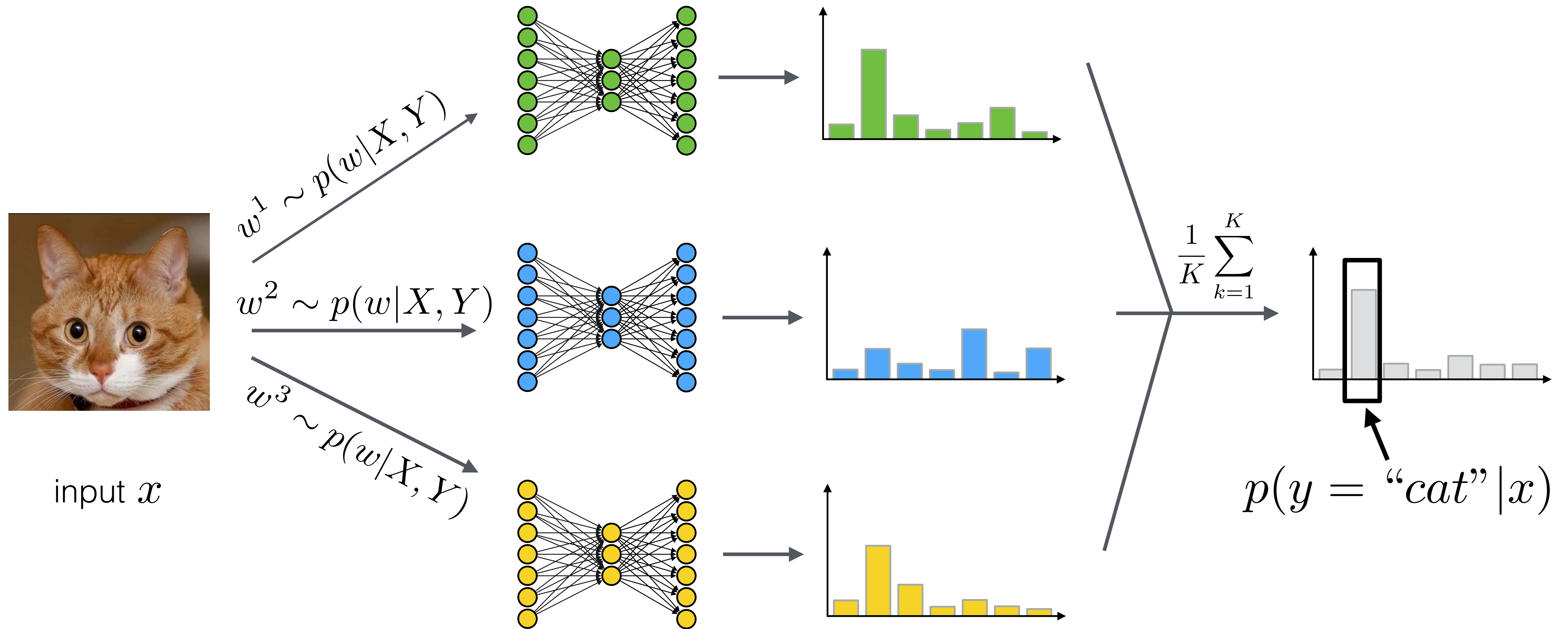
Prediction on a new object x_* :

$$\mathbb{E}_{p(w|X,Y)} p(y_* | x_*, w) \approx \frac{1}{K} \sum_{k=1}^K p(y_* | x_*, w^k), \quad w^k \sim p(w|X, Y)$$



average net's output
across several weight
samples

BNN as an ensemble of neural networks



BNN as an ensemble of neural networks

Prediction on a new object x_* :

$$\mathbb{E}_{p(w|X,Y)} p(y_* | x_*, w) \approx \frac{1}{K} \sum_{k=1}^K p(y_* | x_*, w^k), \quad w^k \sim p(w|X, Y)$$

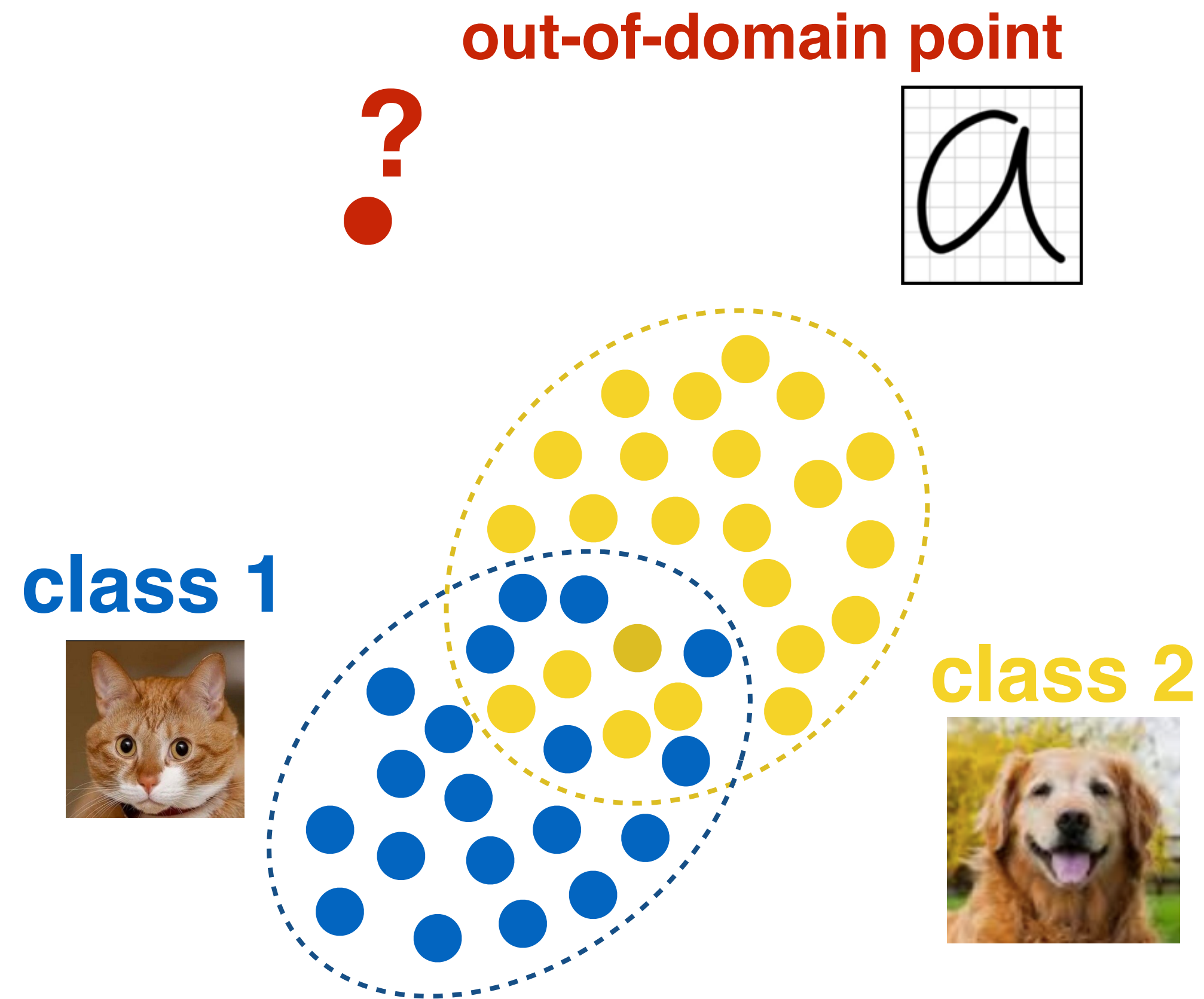
- Higher quality (models compensate each other's errors)
- Better uncertainty estimation

Why go Bayesian?

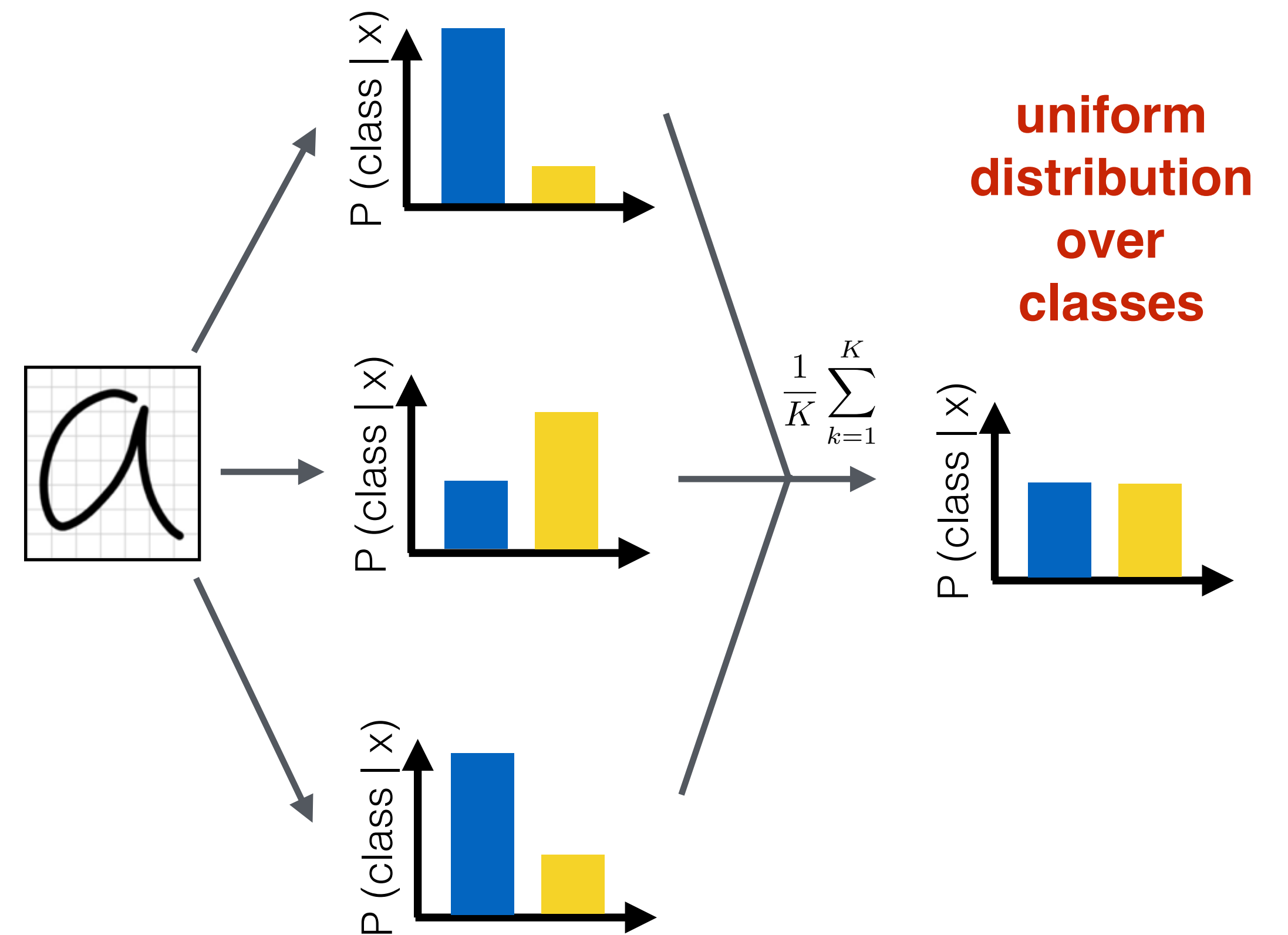
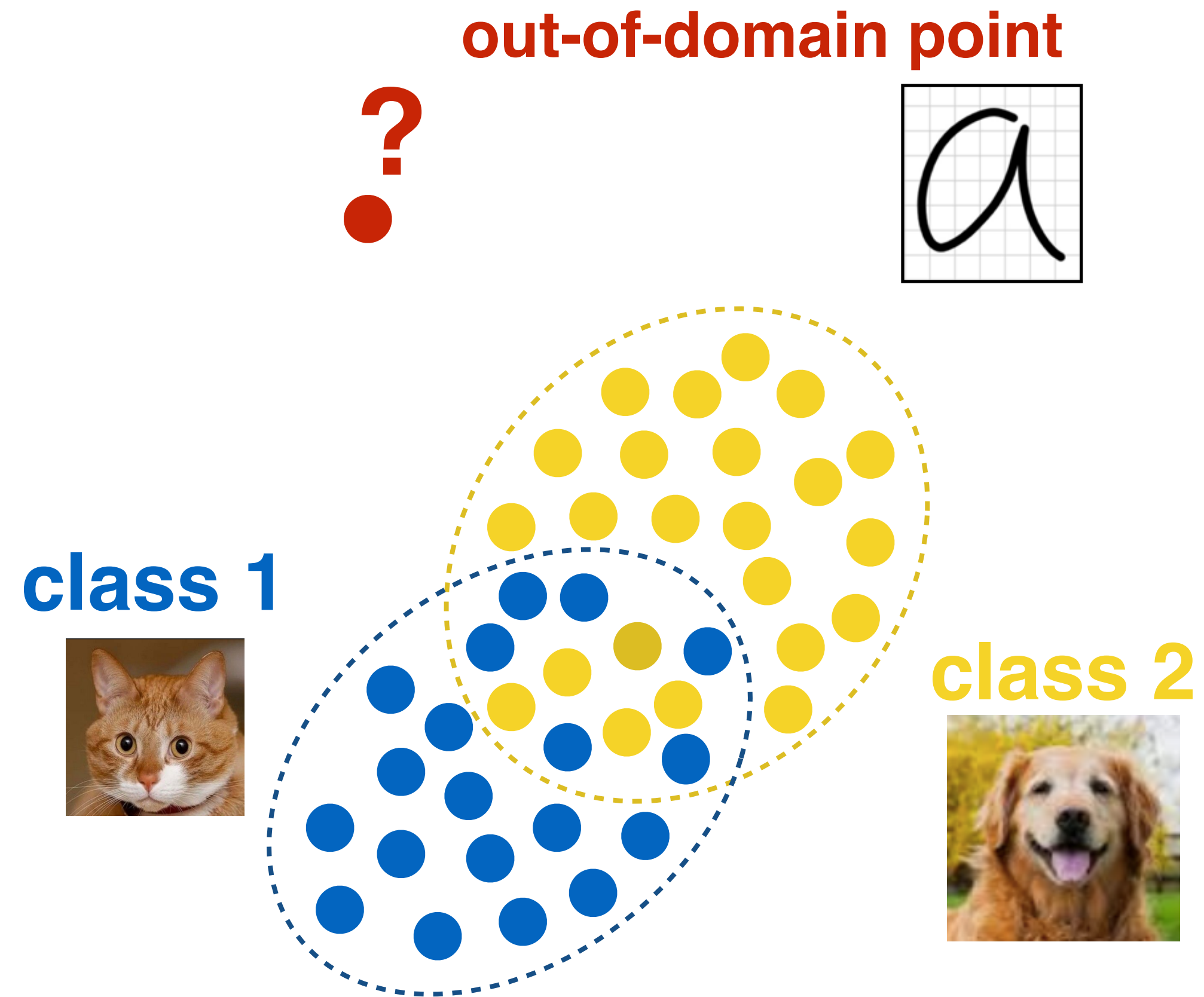
A principled framework with many useful applications

- Regularization
- Ensembling
- Uncertainty estimation
- On-line / continual learning
- Automatic hyperparameter choice
- Different prior \Rightarrow different properties of the network

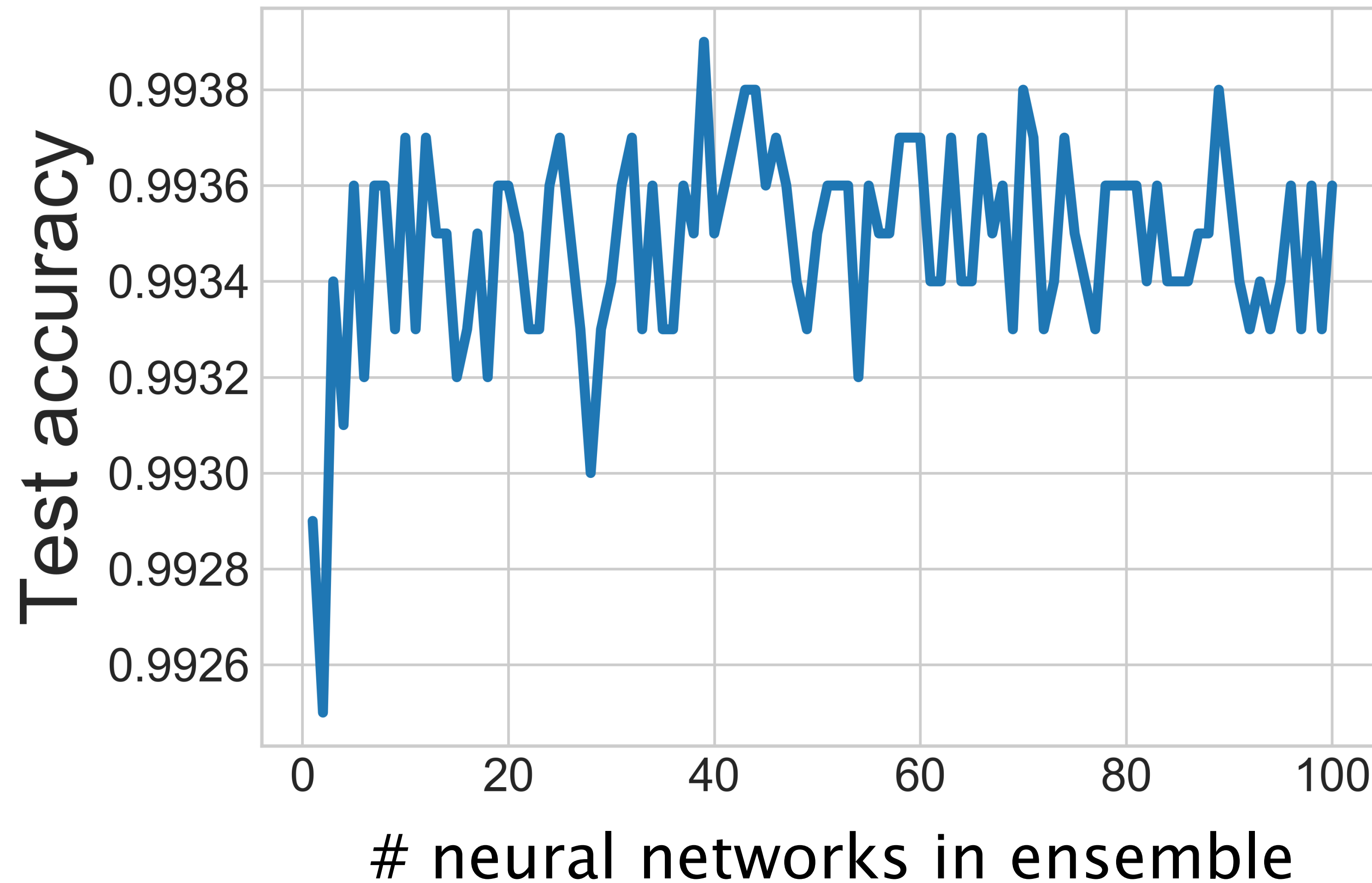
Uncertainty estimation



Uncertainty estimation

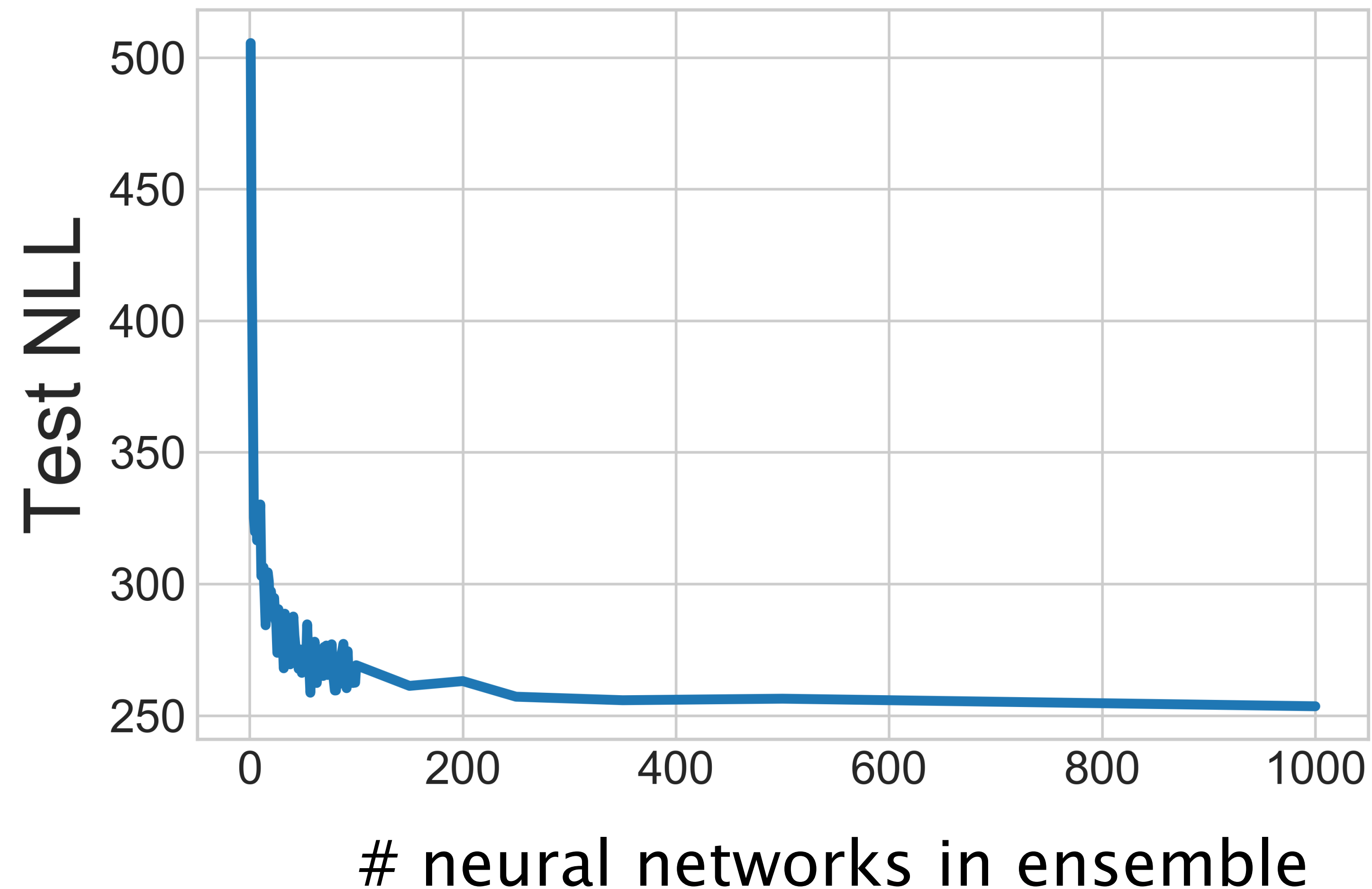


Ensembling: accuracy



Accuracy quickly saturates

Ensembling: uncertainty estimation



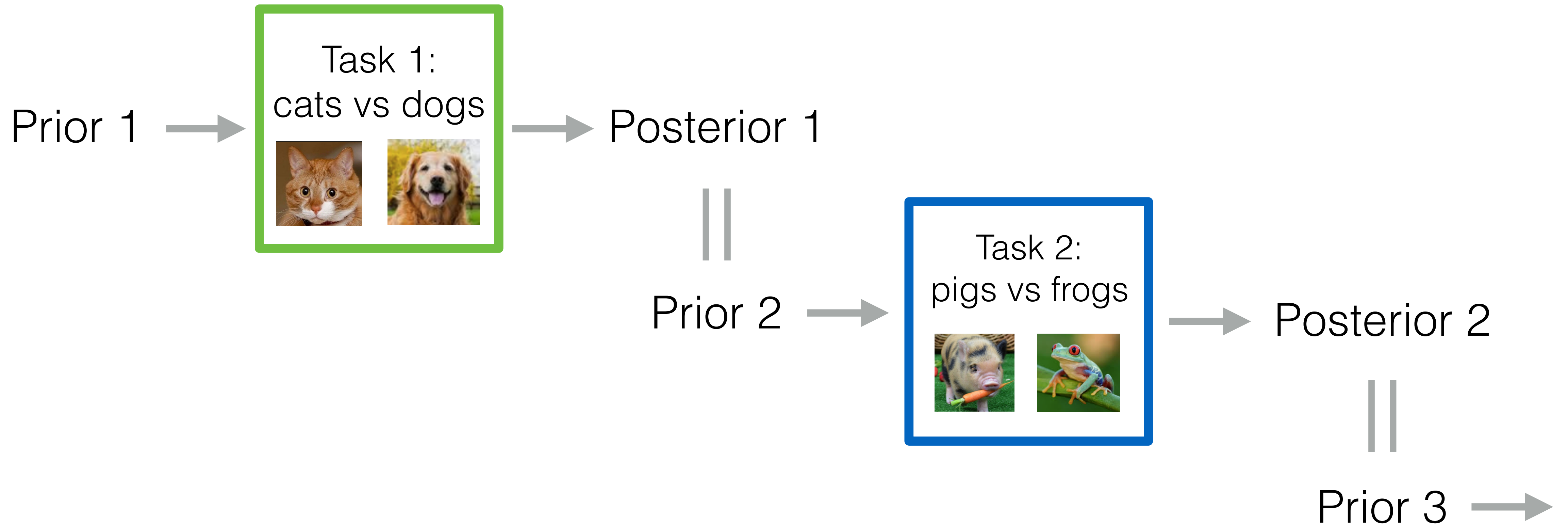
But the negative log—likelihood keeps improving! This is a measure of “**uncertainty**”

Why go Bayesian?

A principled framework with many useful applications

- Regularization ✓
- Ensembling ✓
- Uncertainty estimation ✓
- On-line / continual learning
- Automatic hyperparameter choice
- Different prior \Rightarrow different properties of the network

On-line / continual learning

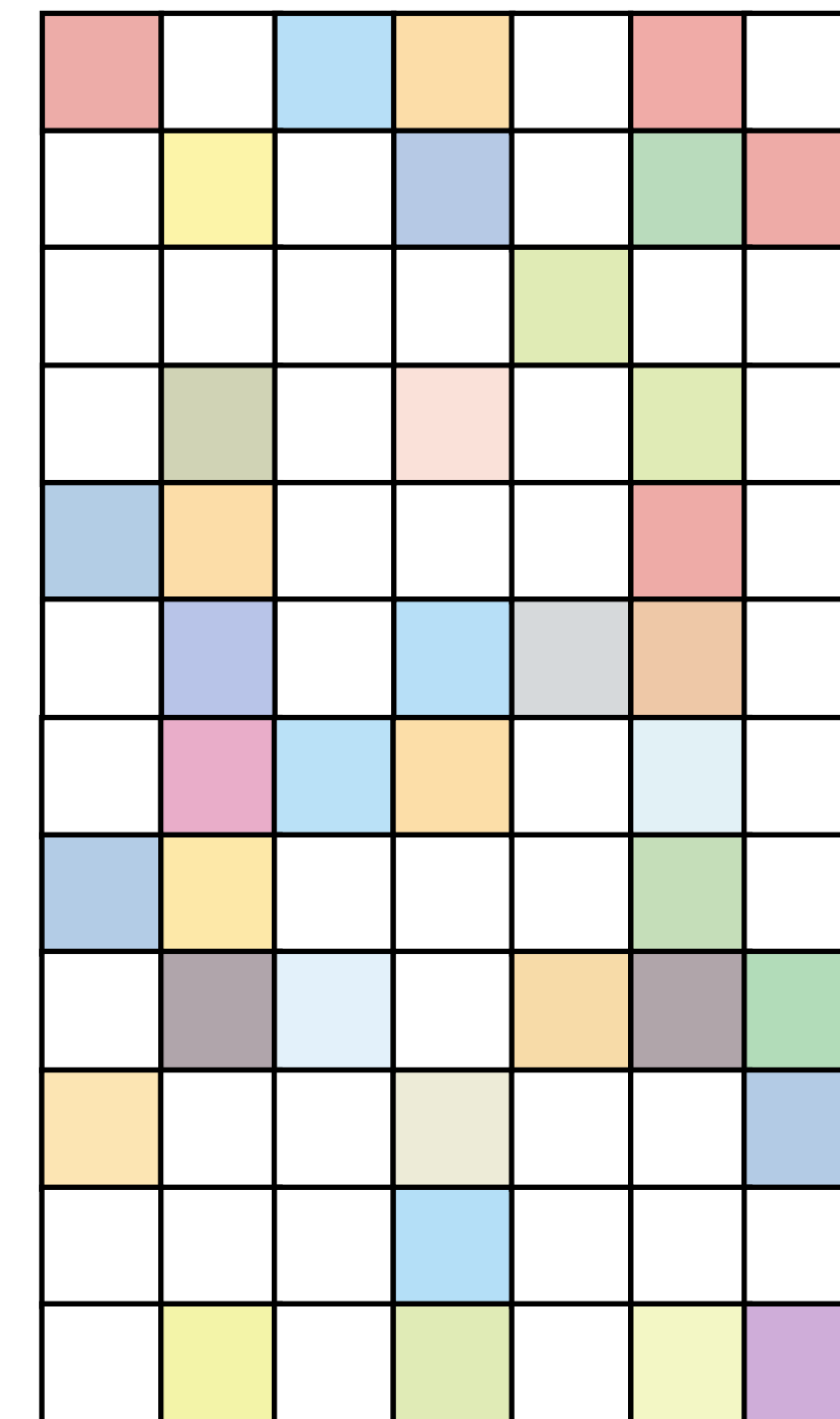
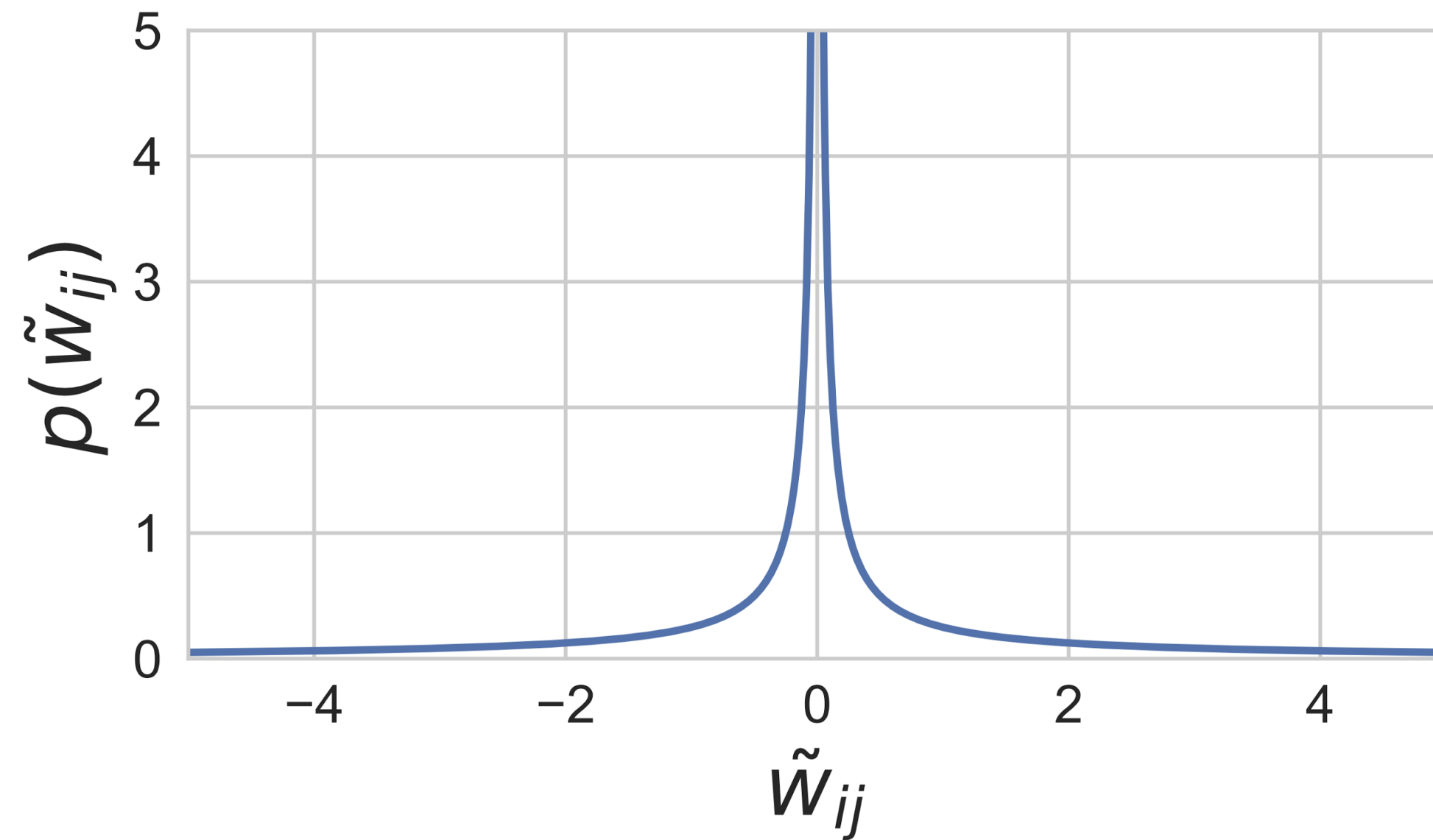


Prior can encode our desirable model properties

Prior concentrated at zero



A lot of zero weights



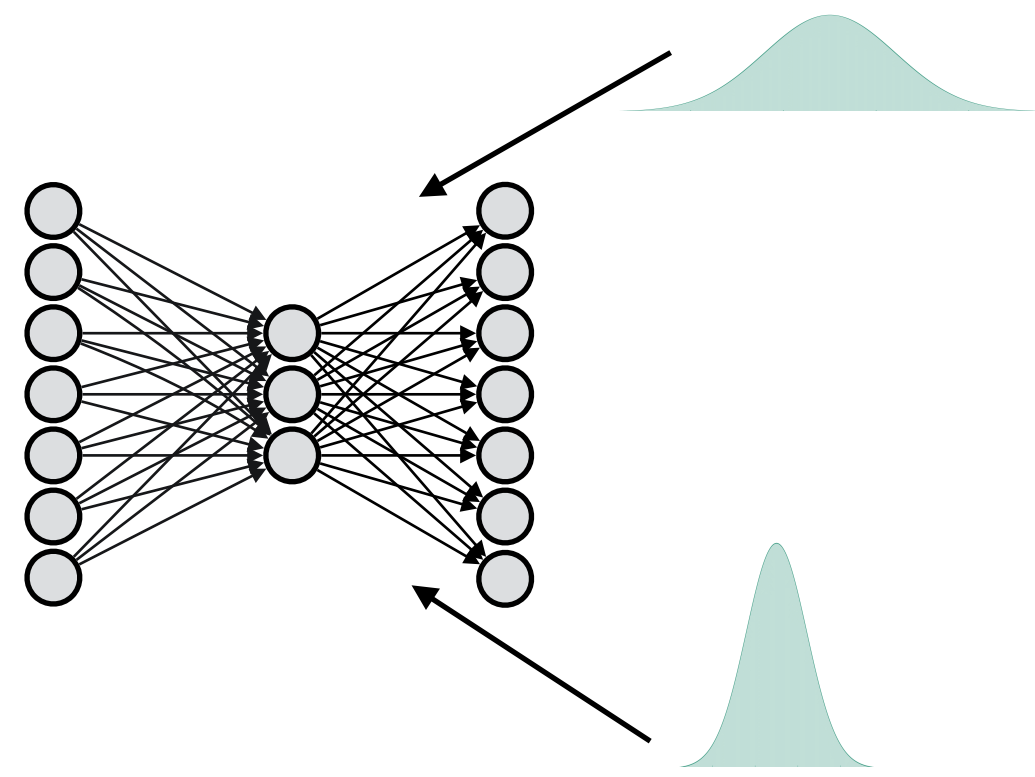
Weight matrix W

Plan

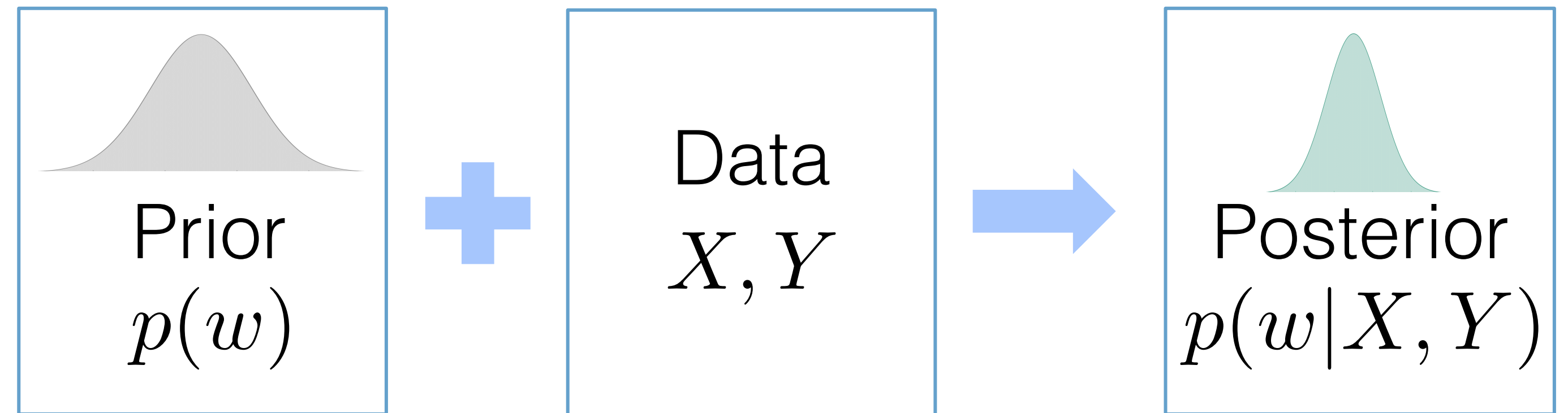
- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- First example: binary dropout
- Second example: Bayesian sparsification

Training Bayesian neural networks

Stochastic weights:



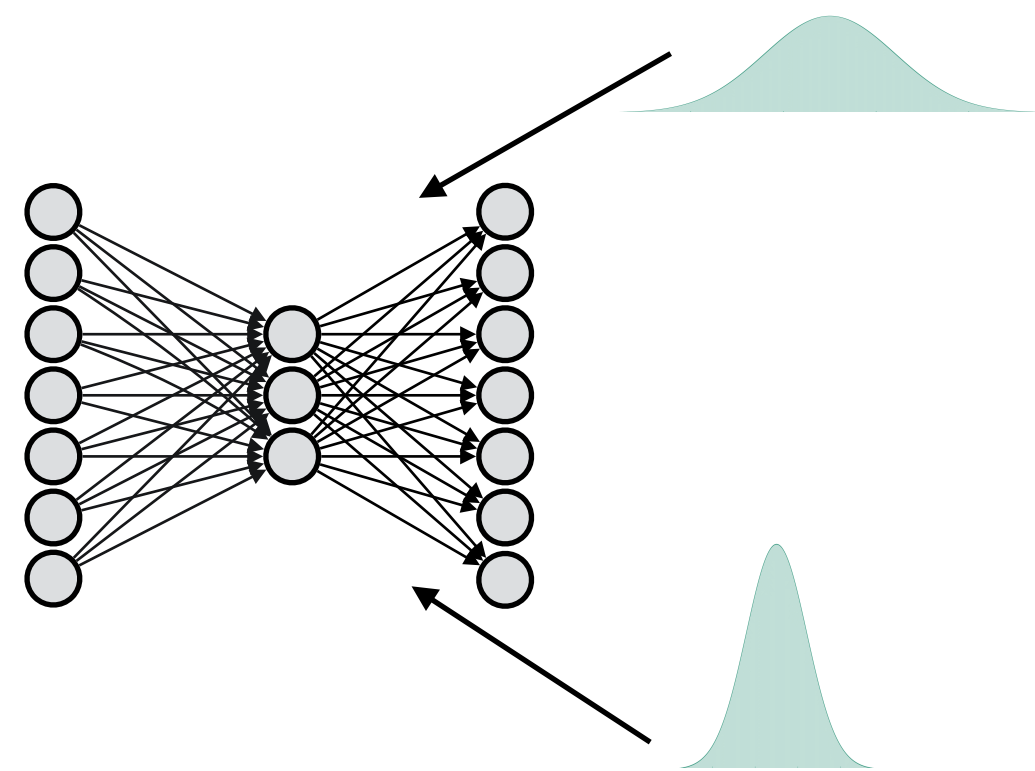
Bayesian Inference:



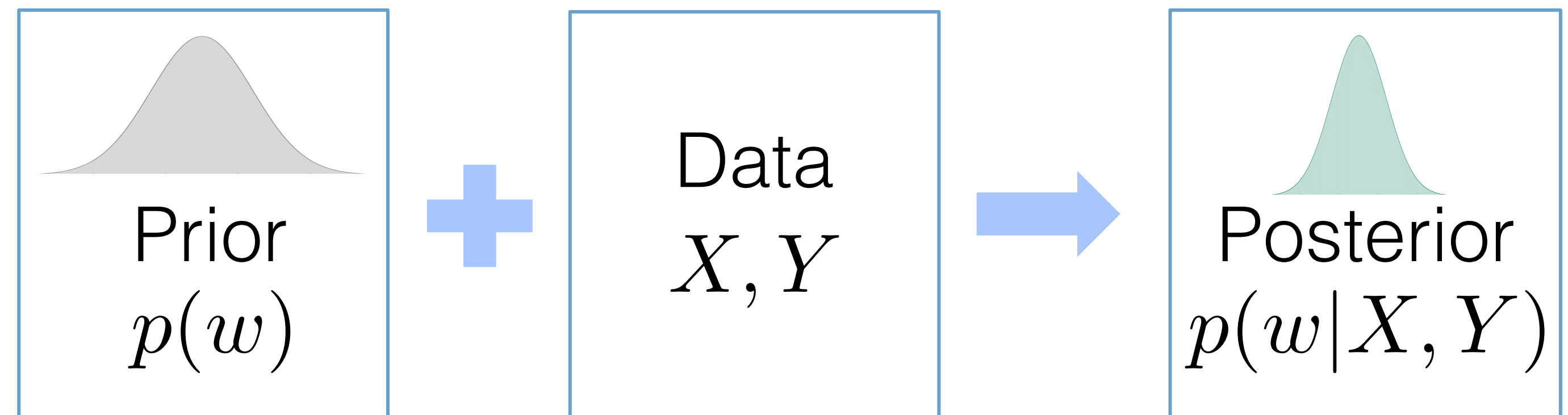
$$p(w|X, Y) = \frac{p(Y|X, w)p(w)}{\int p(Y|X, \tilde{w})p(\tilde{w})d\tilde{w}}$$

Training Bayesian neural networks

Stochastic weights:



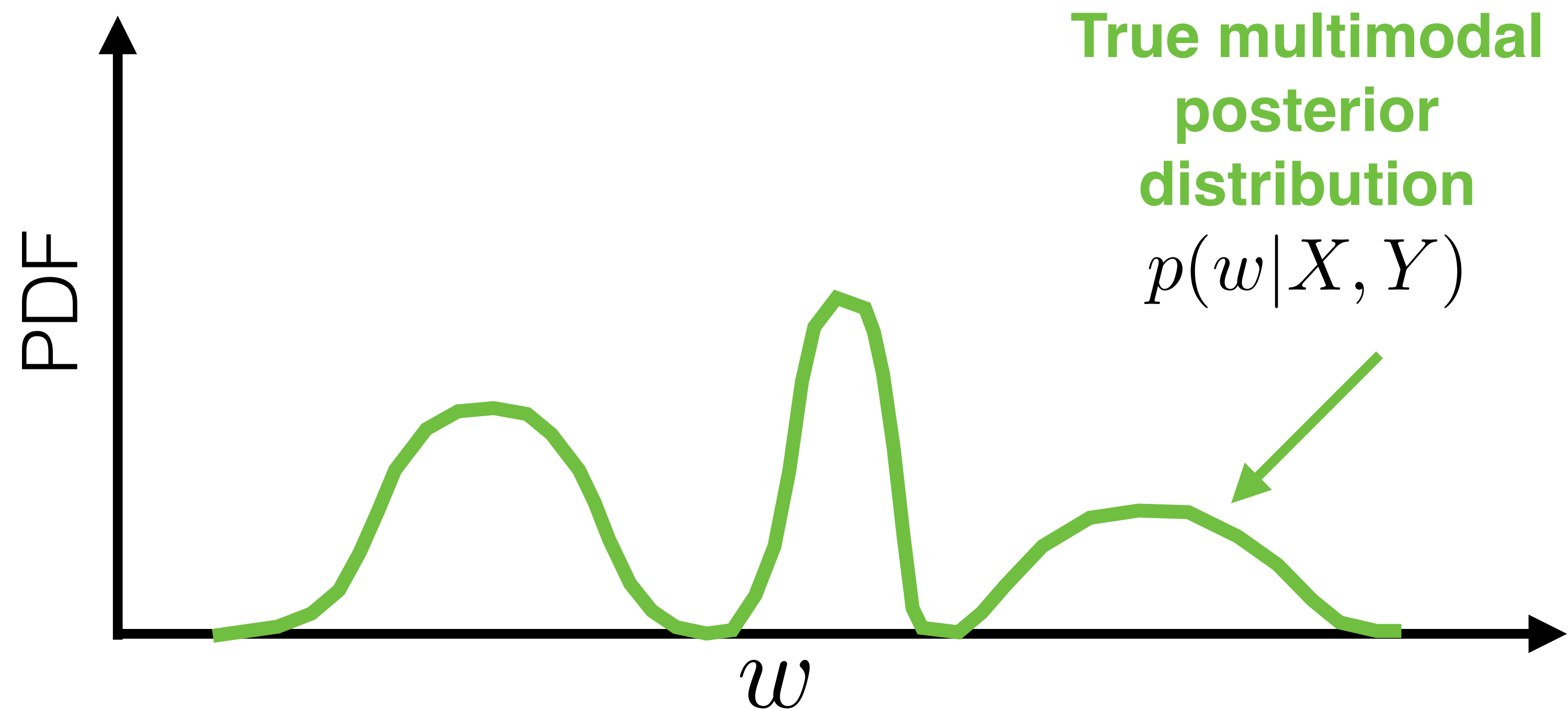
Bayesian Inference:



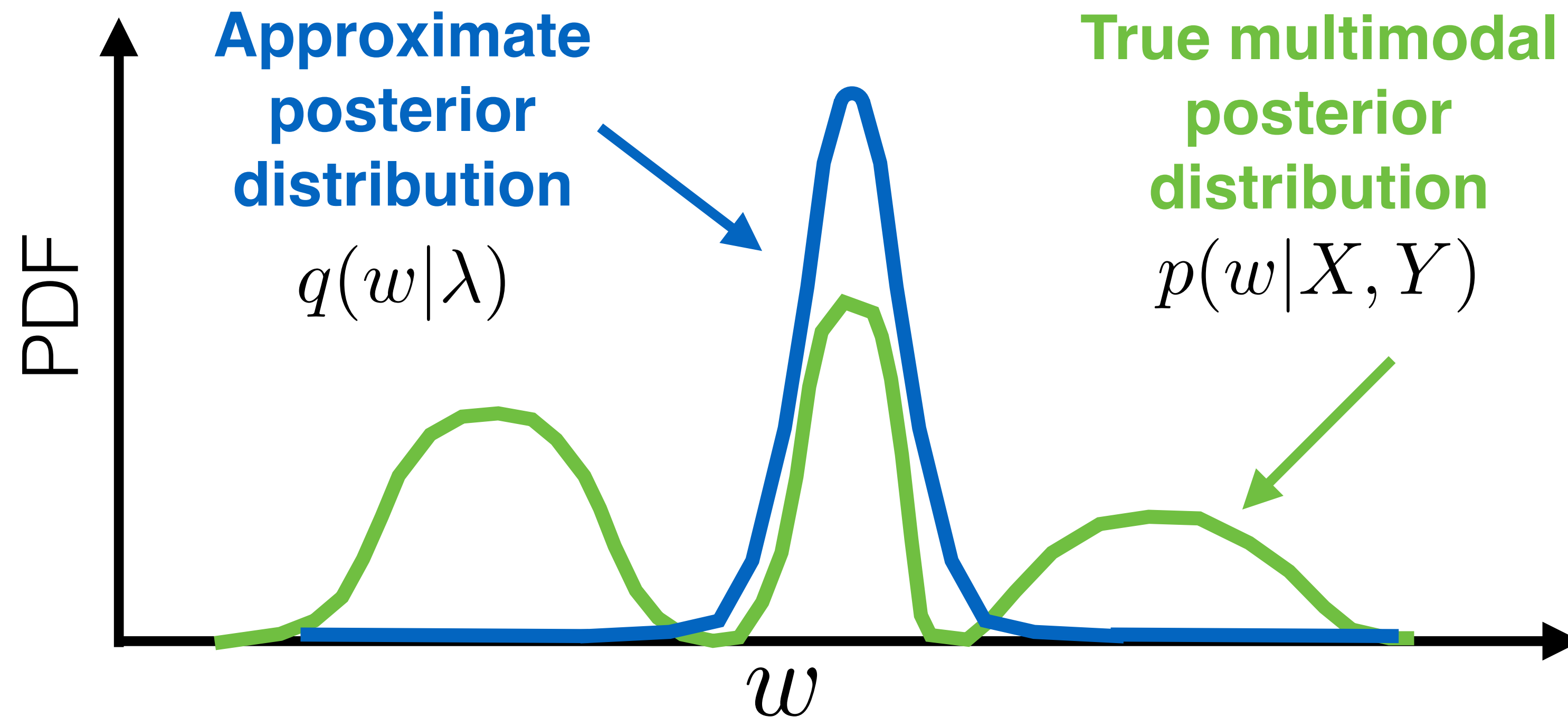
$$p(w|X, Y) = \frac{p(Y|X, w)p(w)}{\int p(Y|X, \tilde{w})p(\tilde{w})d\tilde{w}}$$

Intractable!

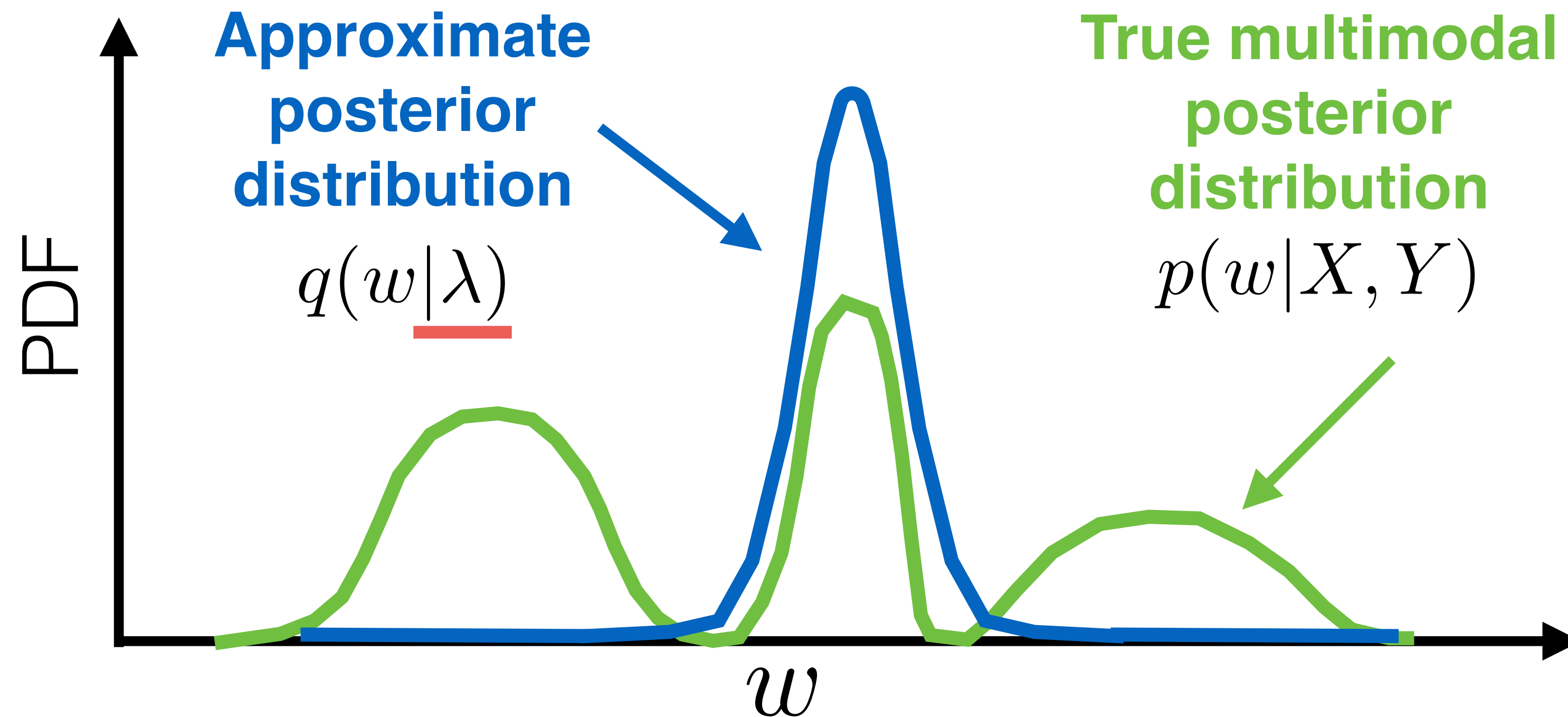
Training Bayesian neural networks



Training Bayesian neural networks

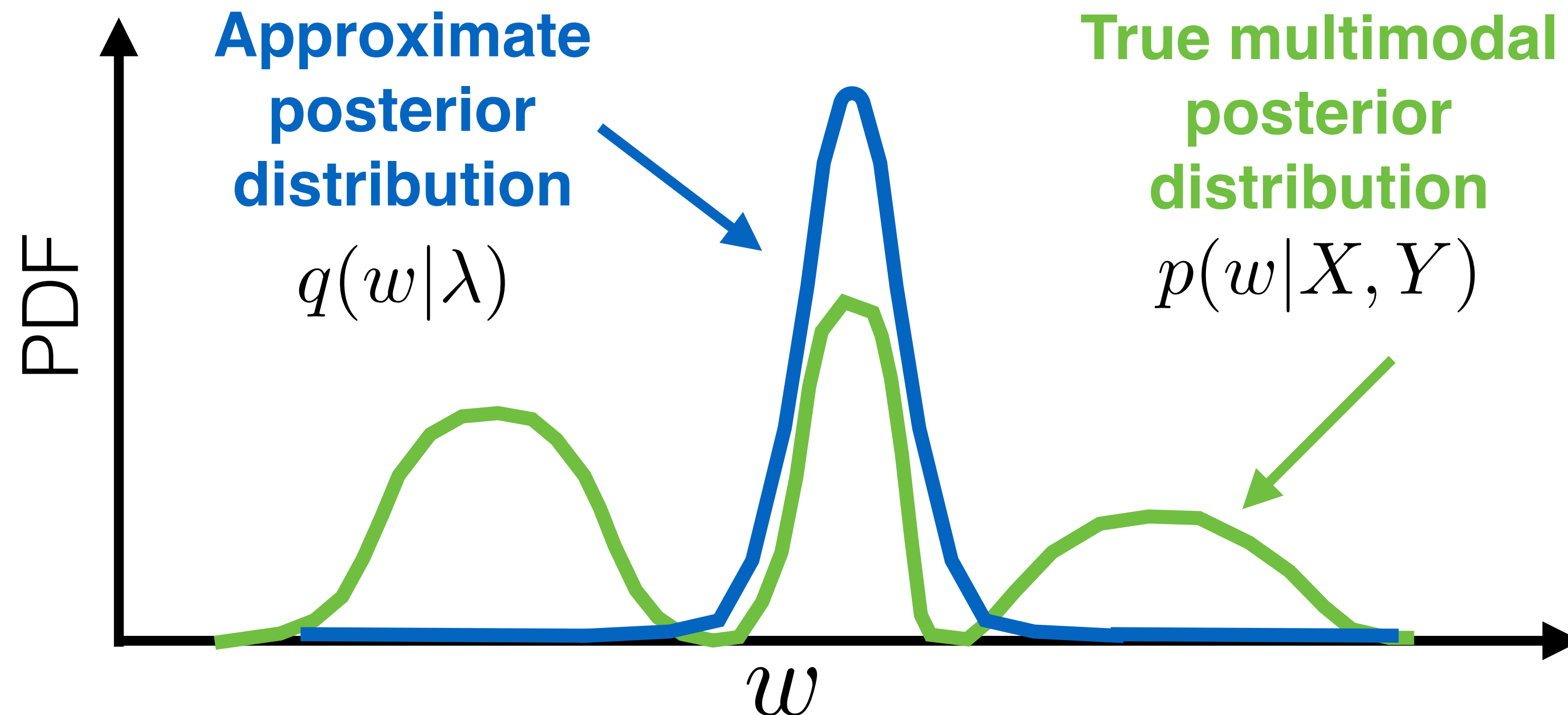


Training Bayesian neural networks



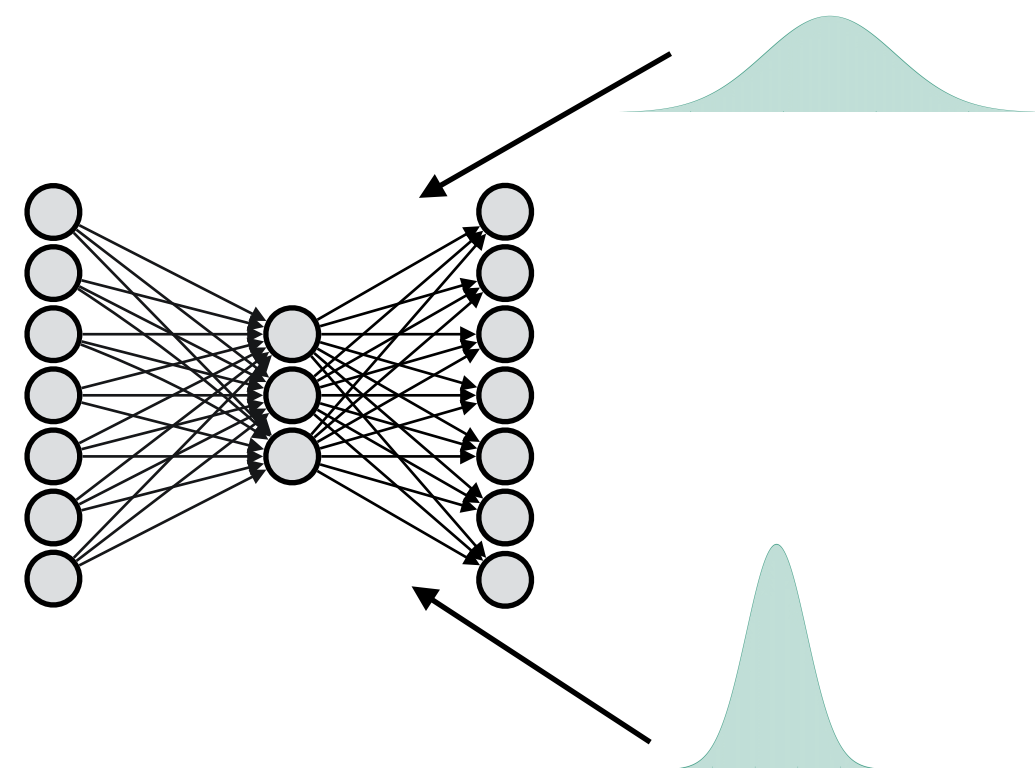
Training Bayesian neural networks

“distance”
between two distributions $\longrightarrow KL(q(w|\lambda) || p(w|X, Y)) \longrightarrow \min_{\lambda}$

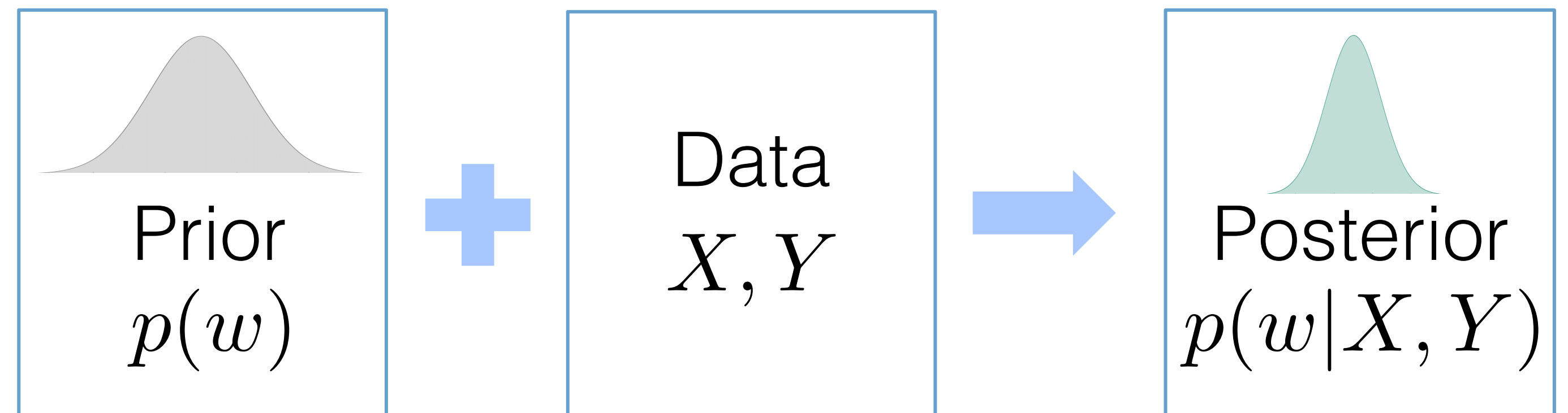


Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:

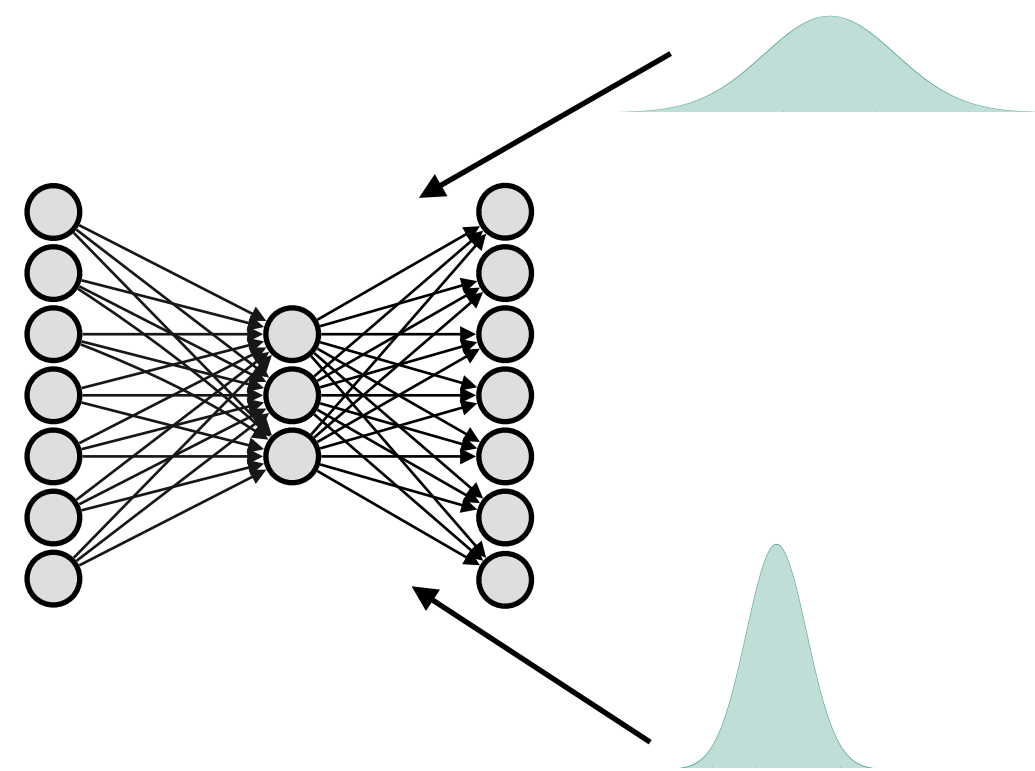


Posterior is intractable in neural networks \rightarrow approximate it with $q(w|\lambda)$:

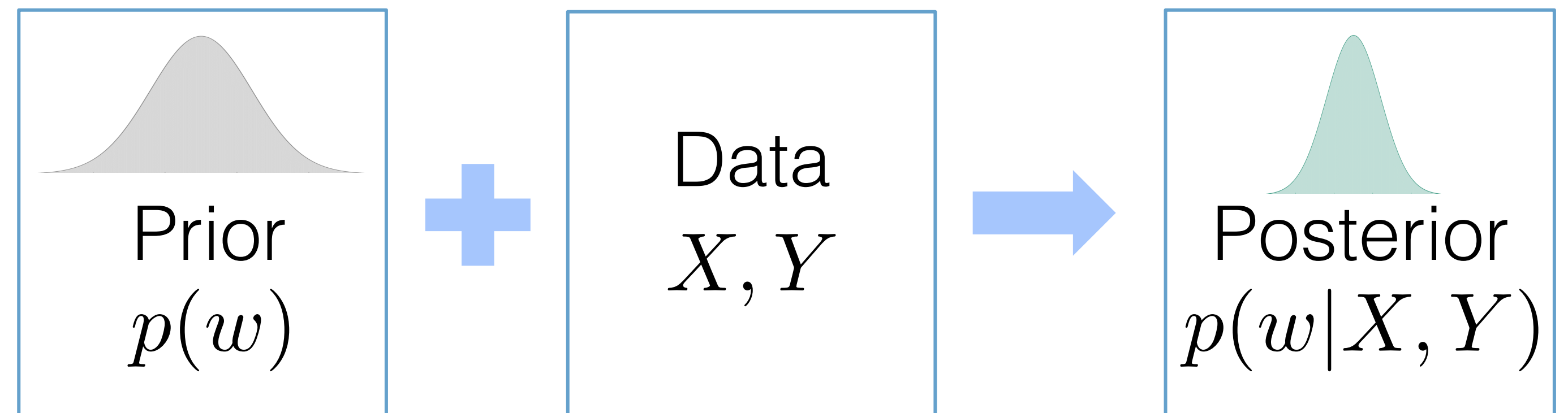
$$KL(q(w|\lambda) || p(w|X, Y)) \rightarrow \min_{\lambda}$$

Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:



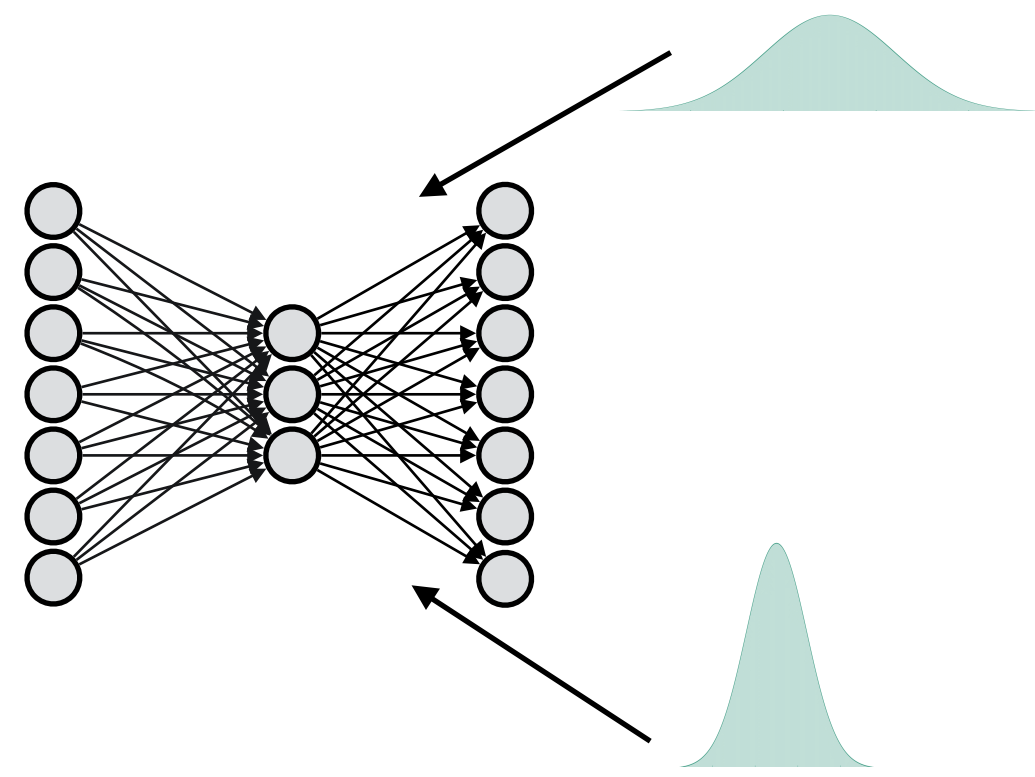
Posterior is intractable in neural networks \rightarrow approximate it with $q(w|\lambda)$:

$$KL(q(w|\lambda) || p(w|X, Y)) \rightarrow \min_{\lambda}$$

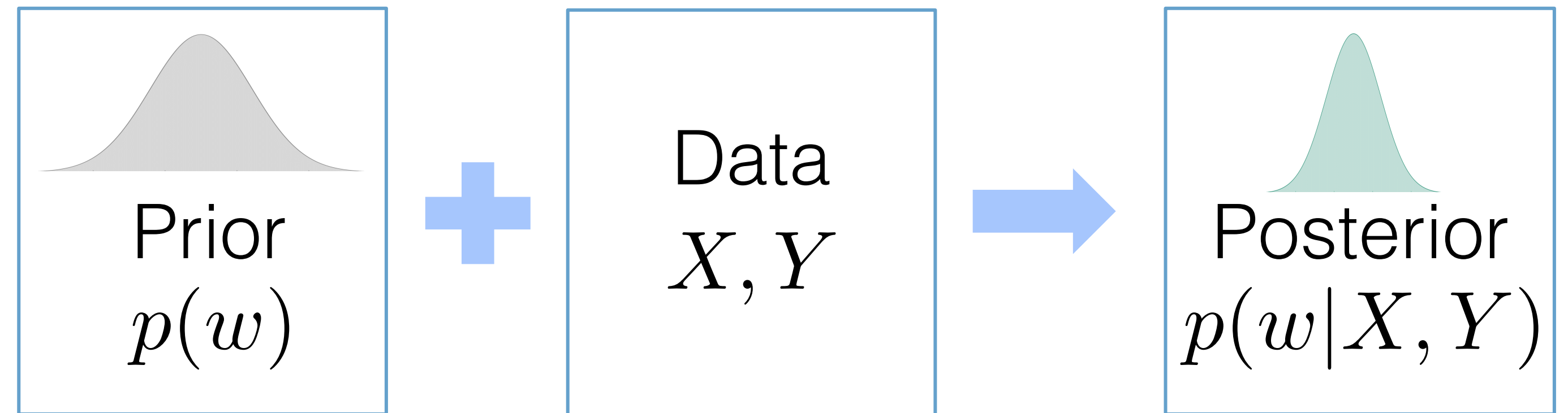
Intractable!

Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:



Equivalently, we can optimize ELBO to find approximate posterior $q(w|\lambda)$:

$$\sum_{i=1}^N \underbrace{\mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w)}_{\text{Data term}} - \underbrace{KL(q(w|\lambda) || p(w))}_{\text{Regularizer}} \rightarrow \max_{\lambda}$$

Doubly stochastic variational inference

How to estimate the data term?

$$\sum_{i=1}^N \underbrace{\mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w)}_{\substack{\text{Data term} \\ \text{???}}} - \underbrace{KL(q(w|\lambda) || p(w))}_{\text{Regularizer}} \rightarrow \max_{\lambda}$$

Doubly stochastic variational inference

How to estimate the data term?

$$\sum_{i=1}^N \underbrace{\mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w)}_{\text{Data term}} - \underbrace{KL(q(w|\lambda) || p(w))}_{\text{Regularizer}} \rightarrow \max_{\lambda}$$

sample mini-batch
of the training objects

sample weights from q

Doubly stochastic variational inference

How to estimate the data term?

$$\sum_{i=1}^N \underbrace{\mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w)}_{\text{Data term}} - \underbrace{KL(q(w|\lambda) || p(w))}_{\text{Regularizer}} \rightarrow \max_{\lambda}$$

sample mini-batch
of the training objects

sample weights from q

$$\sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, \hat{w}_j), \quad \hat{w}_j \sim q(w|\lambda) \quad i_j \sim \text{Unif}(1, \dots, N)$$

Doubly stochastic variational inference

How to estimate the data term?

$$\sum_{i=1}^N \mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w) - KL(q(w|\lambda) || p(w)) \rightarrow \max_{\lambda}$$

sample mini-batch
of the training objects

sample weights from q

$$\sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, \hat{w}_j), \quad \hat{w}_j \sim q(w|\lambda) \quad i_j \sim \text{Unif}(1, \dots, N)$$

$\lambda?$ **How to differentiate?**

Doubly stochastic variational inference

How to estimate the data term?

$$\sum_{i=1}^N \mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w) - KL(q(w|\lambda) || p(w)) \rightarrow \max_{\lambda}$$

$\frac{\partial}{\partial \lambda} ?$

sample mini-batch
of the training objects

sample weights from q

$$\sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, \hat{w}_j), \quad \hat{w}_j \sim q(w|\lambda) \quad i_j \sim \text{Unif}(1, \dots, N)$$

$\lambda?$ **How to differentiate?**

Gradient of expectation

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q(w|\lambda)} \log p(y^i|x^i, w) = \frac{\partial}{\partial \lambda} \int q(w|\lambda) \log p(y^i|x^i, w) dw =$$

$$= \int \left(\frac{\partial}{\partial \lambda} q(w|\lambda) \right) \log p(y^i|x^i, w) dw$$

Not an expectation over q anymore!

Most popular solution: **reparametrization trick**

Reparameterization trick

As an example, consider 1-dim normal approximate posterior:

$$q(w|\lambda) = \mathcal{N}(\mu, \sigma^2), \quad \lambda = \{\mu, \sigma\}$$

Reparameterization trick

As an example, consider 1-dim normal approximate posterior:

$$q(w|\lambda) = \mathcal{N}(\mu, \sigma^2), \quad \lambda = \{\mu, \sigma\}$$

Property of normal distribution:

$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Reparameterization trick

As an example, consider 1-dim normal approximate posterior:

$$q(w|\lambda) = \mathcal{N}(\mu, \sigma^2), \quad \lambda = \{\mu, \sigma\}$$

Property of normal distribution:

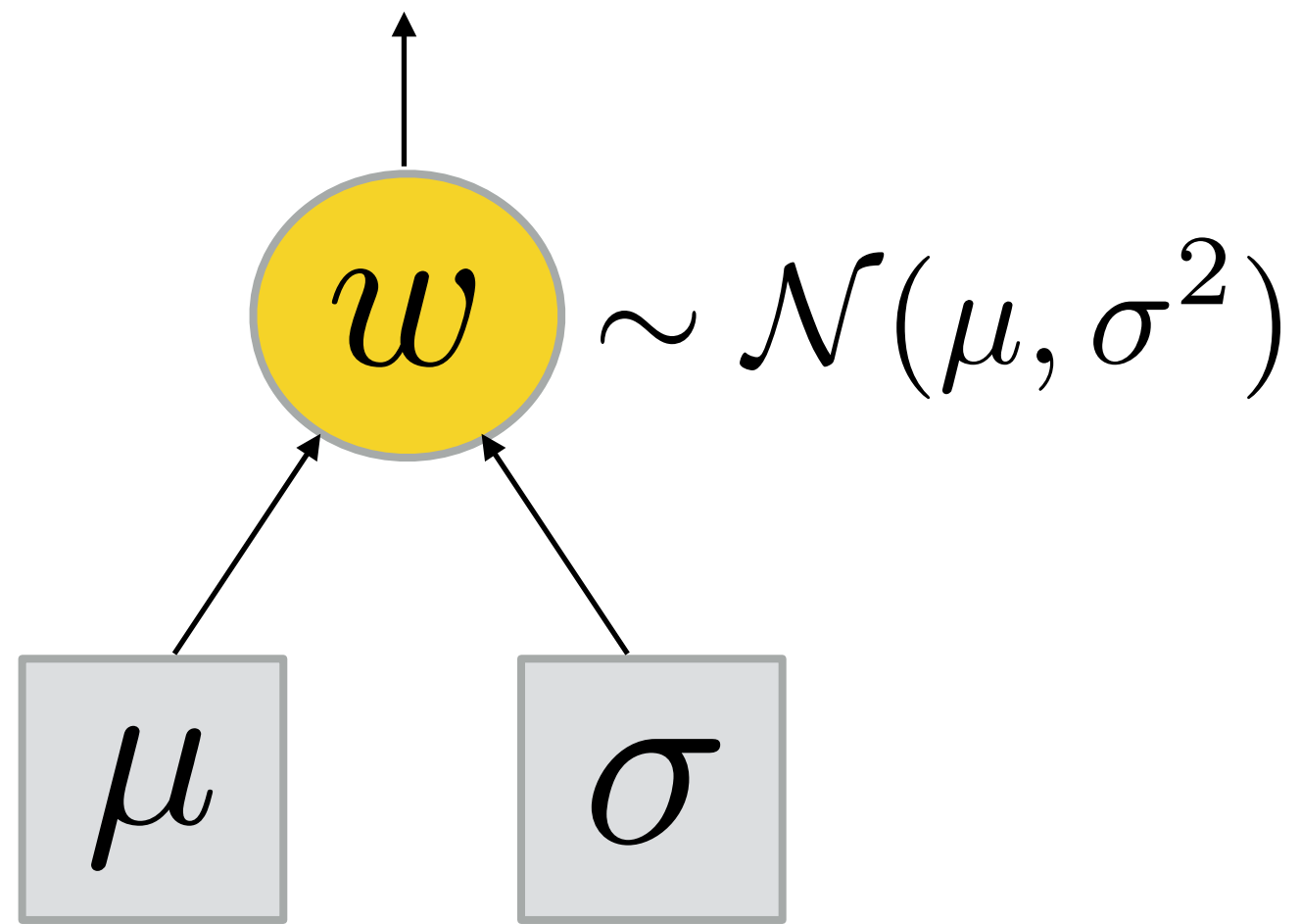
$$w \sim \underbrace{\mathcal{N}(\mu, \sigma^2)}_{\text{distribution conditioned on parameters}} \Leftrightarrow w = \mu + \sigma \epsilon, \quad \epsilon \sim \underbrace{\mathcal{N}(0, 1)}_{\text{unconditioned distribution}}$$

**distribution conditioned
on parameters**

**unconditioned
distribution**

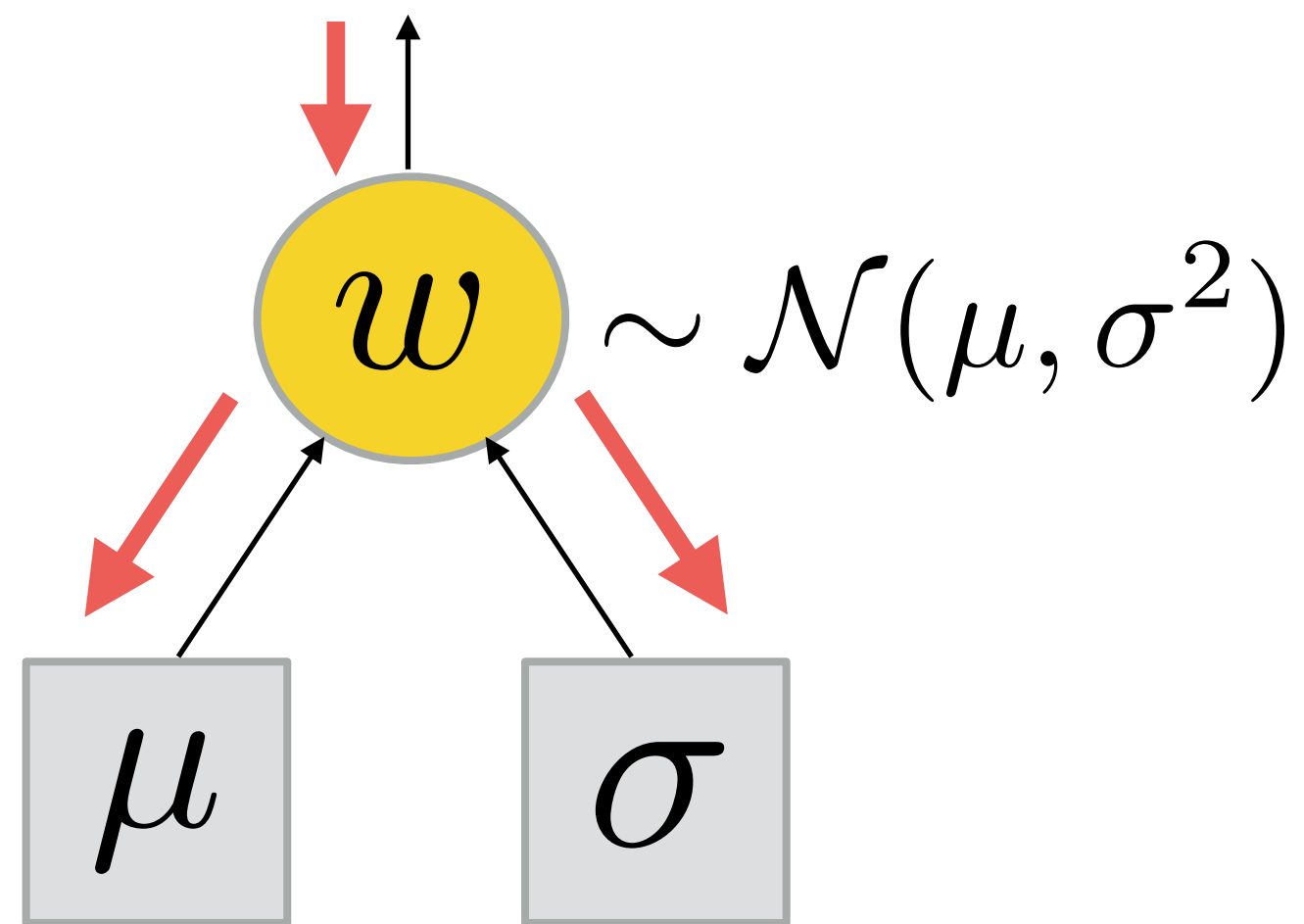
Reparameterization trick

$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$



Reparametrization trick

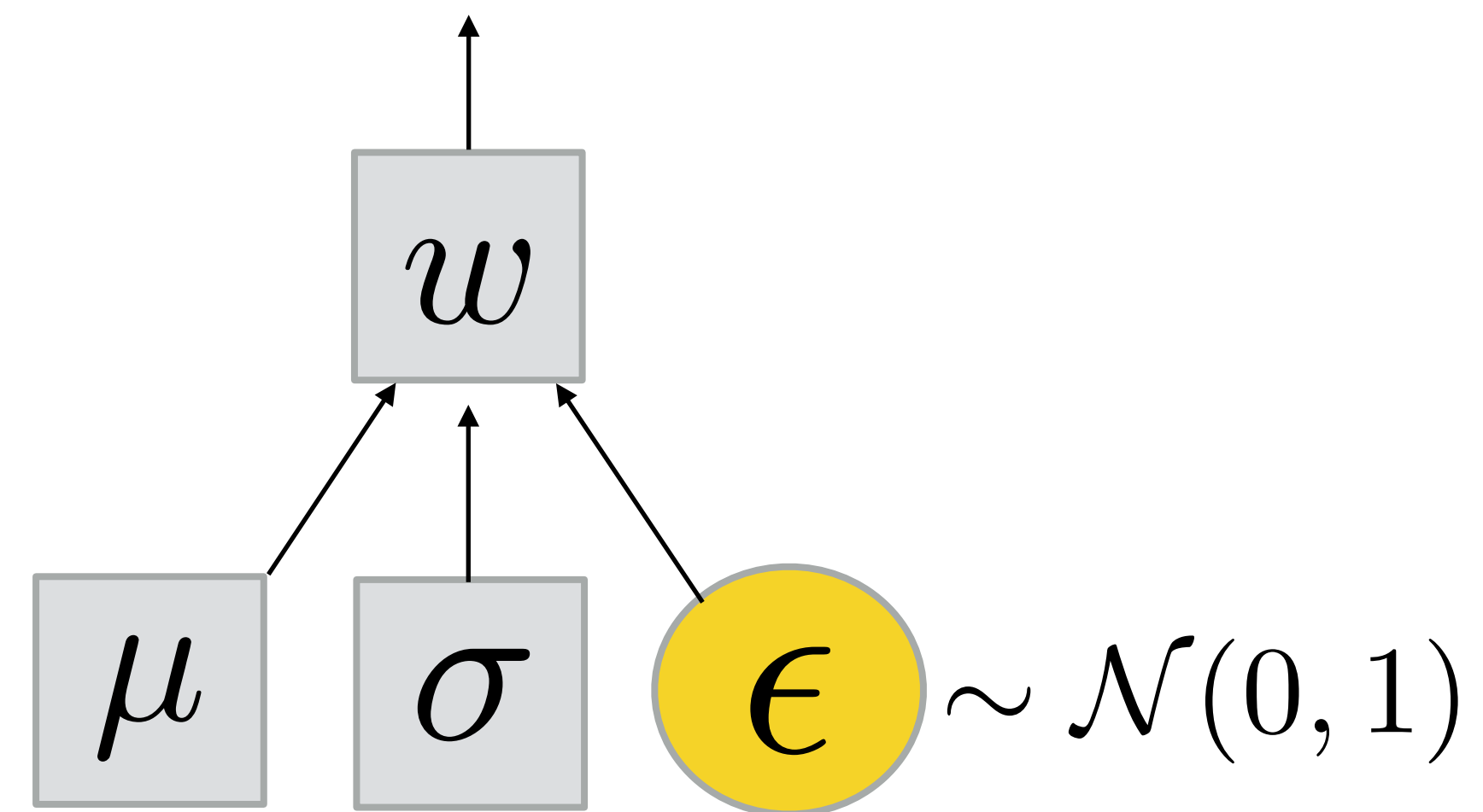
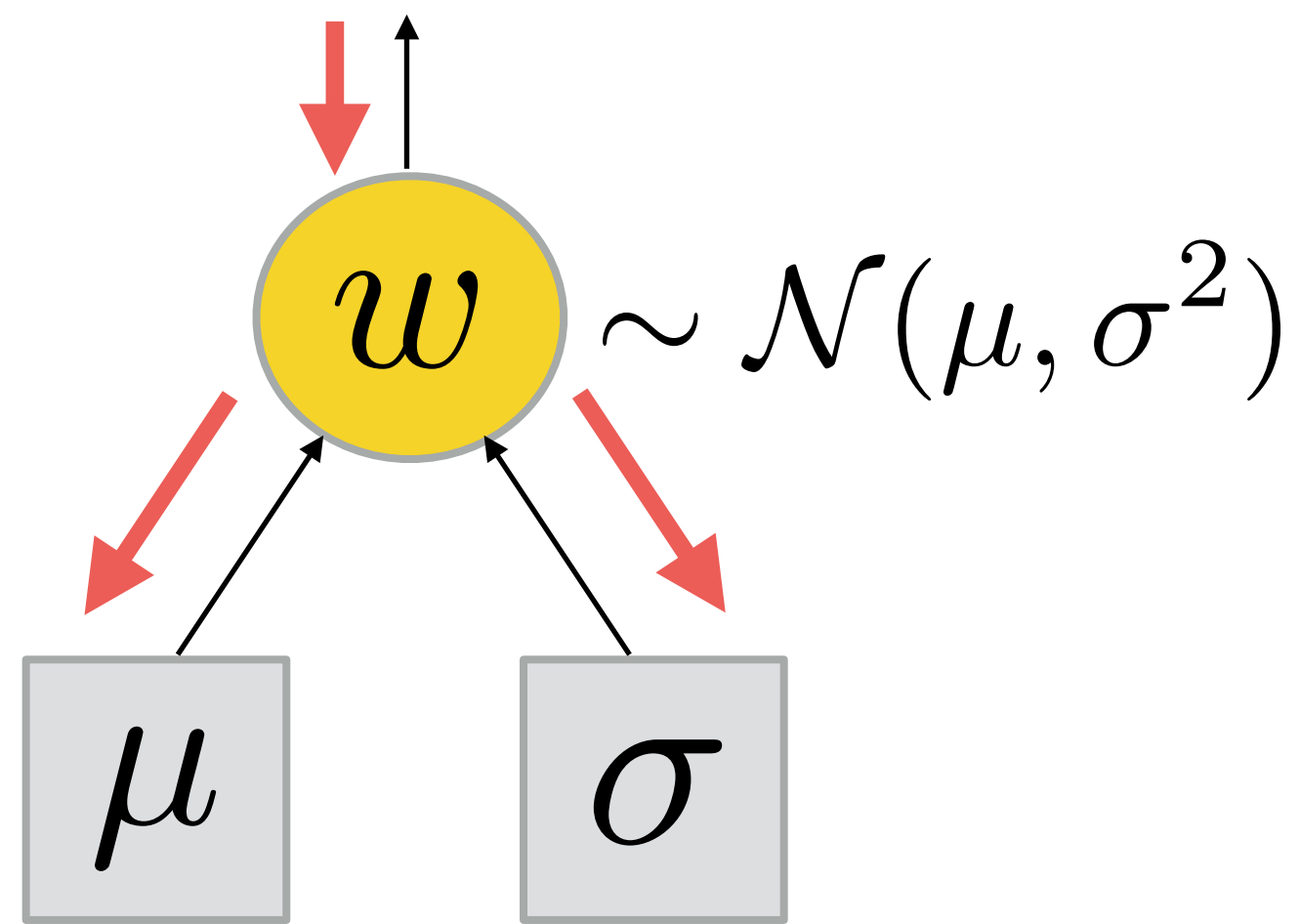
$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$



**Gradients propagate
through randomness**

Reparametrization trick

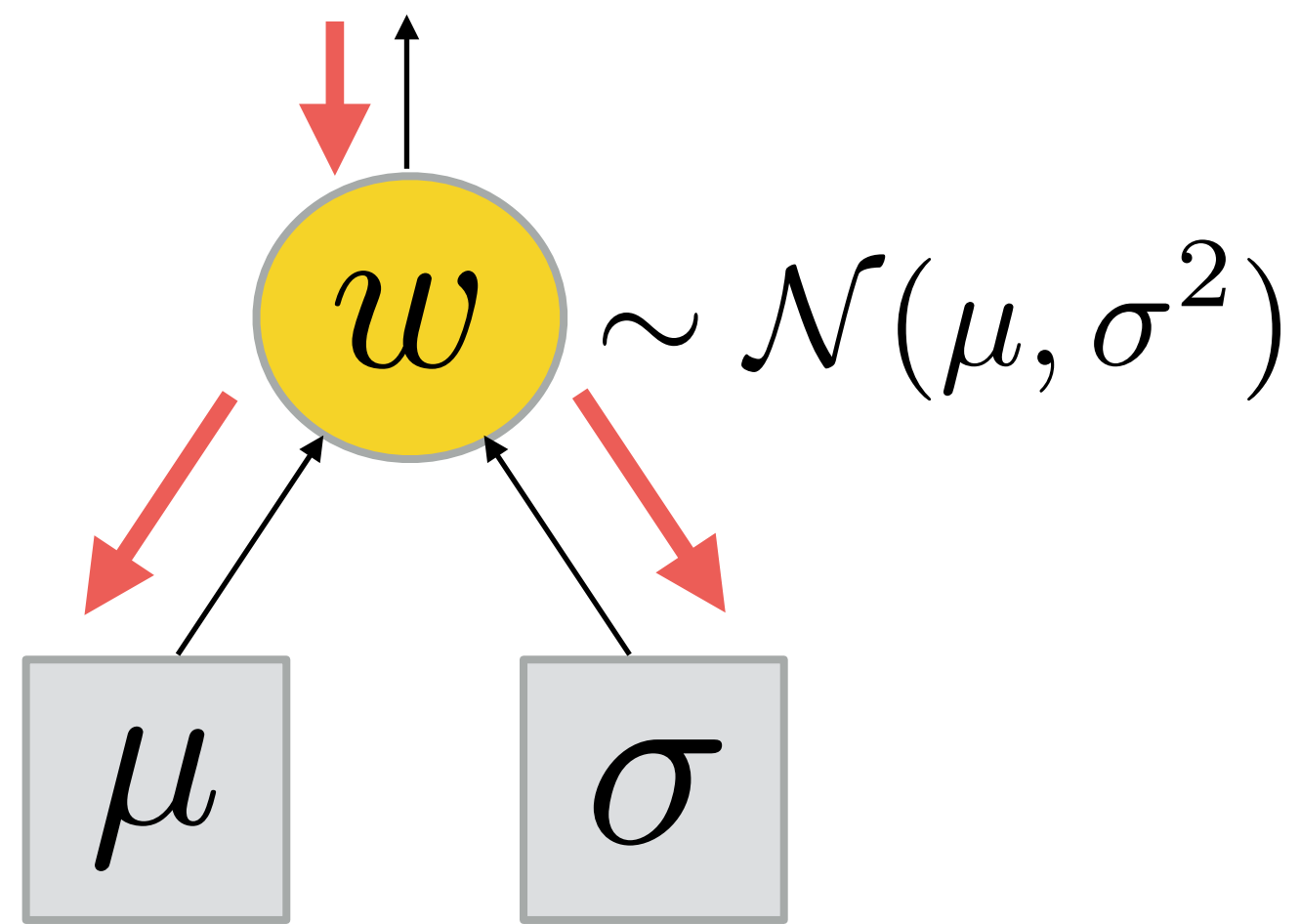
$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$



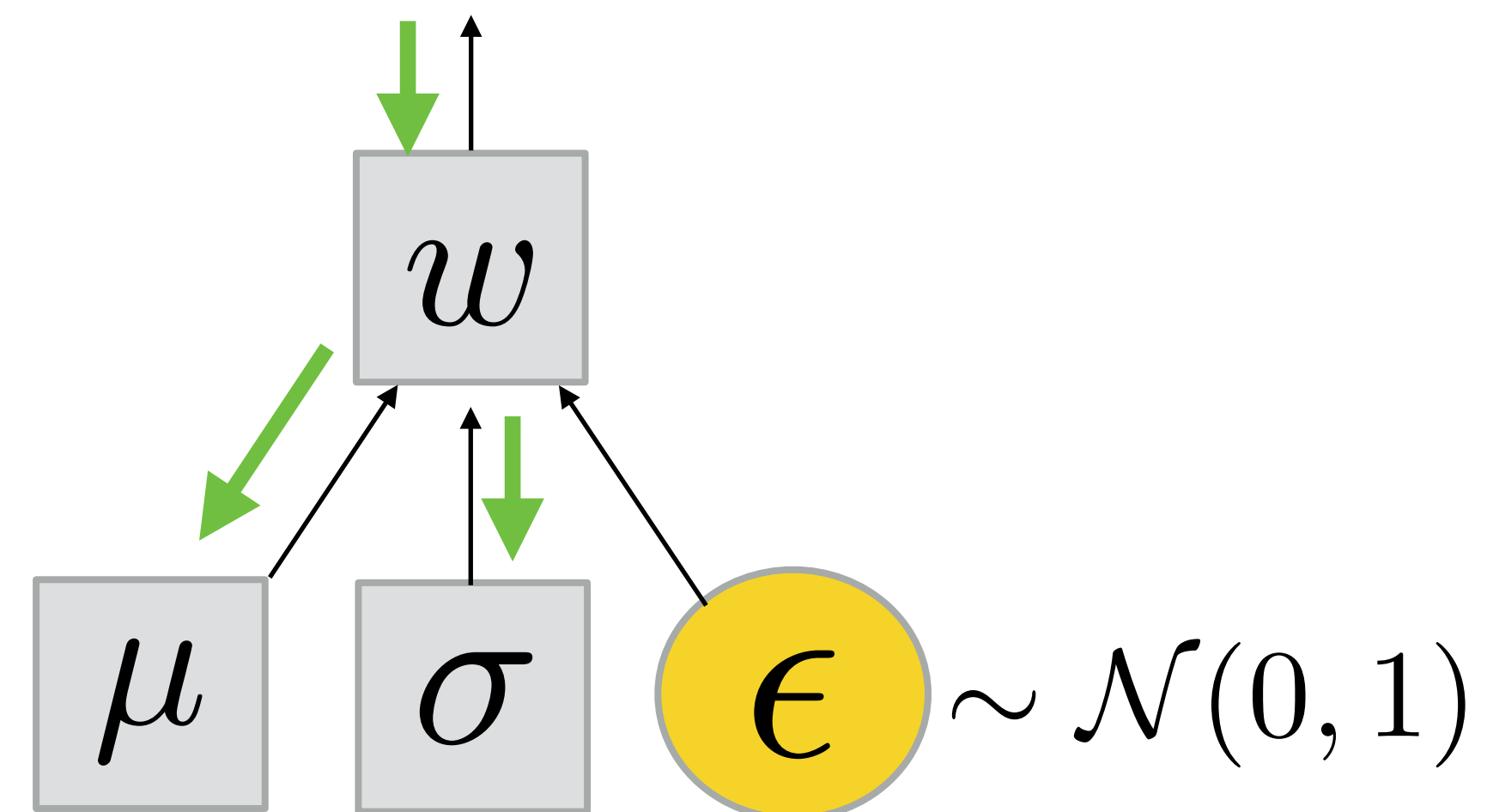
**Gradients propagate
through randomness**

Reparametrization trick

$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$



Gradients propagate through randomness



Gradients propagate only through deterministic nodes

Reparameterization trick: general form

$$w \sim \mathcal{N}(\mu, \sigma^2) \quad \Leftrightarrow \quad w = \mu + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

$$w \sim q(w|\lambda) \quad \Leftrightarrow \quad w = f(\lambda, \epsilon), \quad \epsilon \sim p(\epsilon)$$


**distribution
conditioned
on parameters**


**dependency
on parameters**


**unconditioned
distribution**

Reparameterization trick

$$\underbrace{w \sim q(w|\lambda)}_{\substack{\text{distribution} \\ \text{conditioned} \\ \text{on parameters}}} \Leftrightarrow \underbrace{w = f(\lambda, \epsilon),}_{\substack{\text{dependency} \\ \text{on parameters}}} \quad \underbrace{\epsilon \sim p(\epsilon)}_{\substack{\text{unconditioned} \\ \text{distribution}}}$$

$$\begin{aligned} \nabla_{\lambda} \mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w) &= \nabla_{\lambda} \mathbb{E}_{p(\epsilon)} \log p(y^i | x^i, w = f(\lambda, \epsilon)) = \\ &= \mathbb{E}_{p(\epsilon)} \nabla_{\lambda} \log p(y^i | x^i, w = f(\lambda, \epsilon)) \end{aligned}$$

Reparameterization trick: examples

$$q(w|\lambda) \longrightarrow w = f(\lambda, \epsilon), p(\epsilon)$$

$q(w \lambda)$	$p(\epsilon)$	$f(\lambda, \epsilon)$
$\mathcal{N}(\mu, \sigma^2)$	$\mathcal{N}(0, 1)$	$w = \sigma\epsilon + \mu$
$\mathcal{G}(1, \beta)$	$\mathcal{G}(1, 1)$	$w = \beta\epsilon$
$\mathcal{E}(\lambda)$	$\mathcal{U}(0, 1)$	$w = -\frac{\log \epsilon}{\lambda}$
$\mathcal{N}(\mu, \Sigma)$	$\mathcal{N}(0, I)$	$w = A\epsilon + \mu, \text{ where } AA^T = \Sigma$

Reparametrization trick: examples

$$q(w|\lambda) \longrightarrow w = f(\lambda, \epsilon), p(\epsilon)$$

$q(w \lambda)$	$p(\epsilon)$	$f(\lambda, \epsilon)$
$\mathcal{N}(\mu, \sigma^2)$	$\mathcal{N}(0, 1)$	$w = \sigma\epsilon + \mu$
$\mathcal{G}(1, \beta)$	$\mathcal{G}(1, 1)$	$w = \beta\epsilon$
$\mathcal{E}(\lambda)$	$\mathcal{U}(0, 1)$	$w = -\frac{\log \epsilon}{\lambda}$
$\mathcal{N}(\mu, \Sigma)$	$\mathcal{N}(0, I)$	$w = A\epsilon + \mu, \text{ where } AA^T = \Sigma$



noise on weights!

Local reparametrization trick

1. Reparametrization trick \Rightarrow unbiased estimate of the gradients

$$w_{ij} = \mu_{ij} + \epsilon_{ij}\sigma_{ij}, \quad \epsilon_{ij} \sim \mathcal{N}(0, 1)$$

- Too expensive! ($|w| \times$ mini-batch size)
- One sample per mini-batch? High variance of stochastic gradients & correlated predictions

Local reparametrization trick

1. Reparametrization trick \Rightarrow unbiased estimate of the gradients

$$w_{ij} = \mu_{ij} + \epsilon_{ij}\sigma_{ij}, \quad \epsilon_{ij} \sim \mathcal{N}(0, 1)$$

2. Local reparametrization trick: sample preactivations instead of weights
 \Rightarrow reduced variance of the gradients & uncorrelated predictions

$$w_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$$

$$B = AW$$

$$\mathbb{E}B = A\mu \quad \text{Var}B = A^2\sigma^2$$

$$B \sim \mathcal{N}(A\mu, A^2\sigma^2)$$

$$B = A\mu + \sqrt{A^2\sigma^2} \odot \epsilon$$

blue means
element-wise

Training: putting everything together

How to estimate the data term?

$$\sum_{i=1}^N \underbrace{\mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w)}_{\text{Data term}} - \underbrace{KL(q(w|\lambda) || p(w))}_{\text{Regularizer}} \rightarrow \max_{\lambda}$$

sample mini-batch
of the training objects

sample weights
from q using
reparametrization trick

$$\sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, w = f(\lambda, \epsilon^j)), \quad \epsilon^j \sim p(\epsilon) \quad i_j \sim \text{Unif}(1, \dots, N)$$

Training: putting everything together

Deterministic neural network:

$$w^{new} = w^{old} + \eta \frac{\partial}{\partial w} \sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, w^{old}) \quad i_j \sim \text{Unif}(1, \dots, N)$$

Bayesian neural network:

$$\lambda^{new} = \lambda^{old} + \eta \frac{\partial}{\partial \lambda} \sum_{j=1}^m \log p(y^{i_j} | x^{i_j}, w = f(\lambda^{old}, \epsilon^j)), \quad i_j \sim \text{Unif}(1, \dots, N)$$
$$\epsilon^j \sim p(\epsilon)$$

m — mini-batch size

η — learning rate

From general framework to particular method

$$\sum_{i=1}^N \mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w) - KL(q(w|\lambda) || p(w)) \rightarrow \max_{\lambda}$$

Model specification:

- Choose particular prior

Training:

- Choose particular family for approximate posterior
- How to compute the KL-divergence?

Plan

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- First example: binary dropout
- Second example: Bayesian sparsification

Example: binary dropout

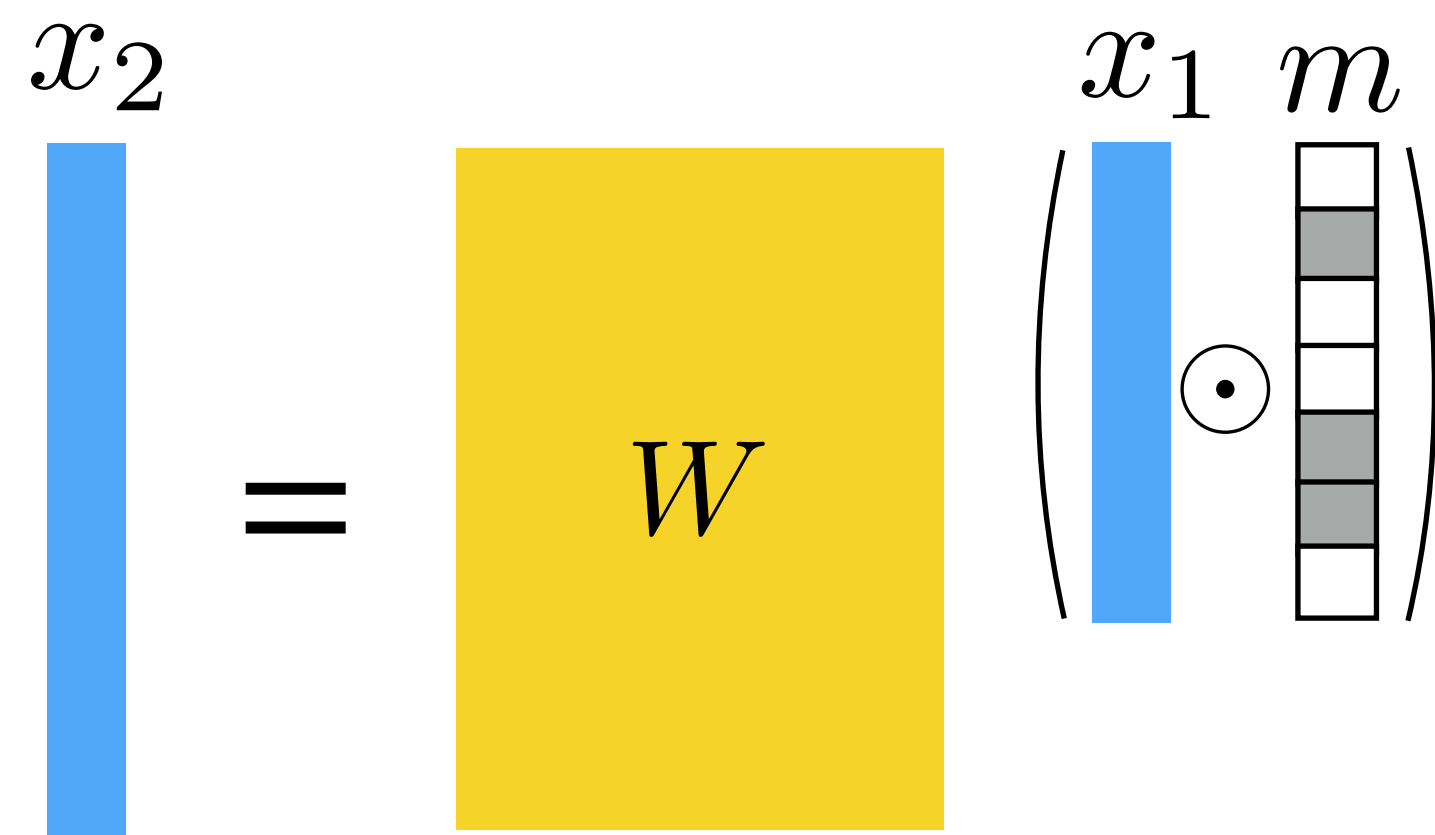
Linear layer: $x_2 = Wx_1$, W — weight matrix

With dropout: $x_2 = W(x_1 \odot m)$, $m_i \sim \text{Bernoulli}(p)$, p — dropout rate

Example: binary dropout

Linear layer: $x_2 = W x_1$, W — weight matrix

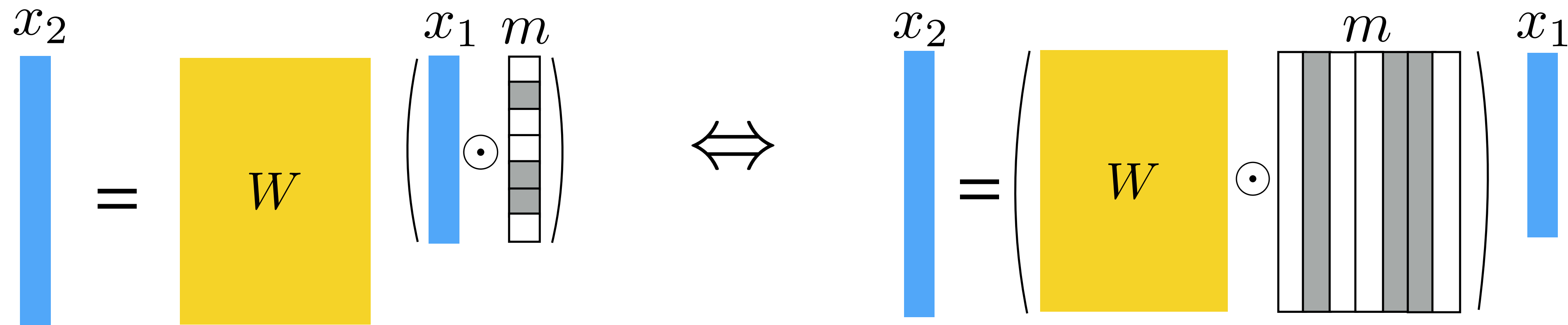
With dropout: $x_2 = W(x_1 \odot m)$, $m_i \sim \text{Bernoulli}(p)$, p — dropout rate



Example: binary dropout

Linear layer: $x_2 = W x_1$, W — weight matrix

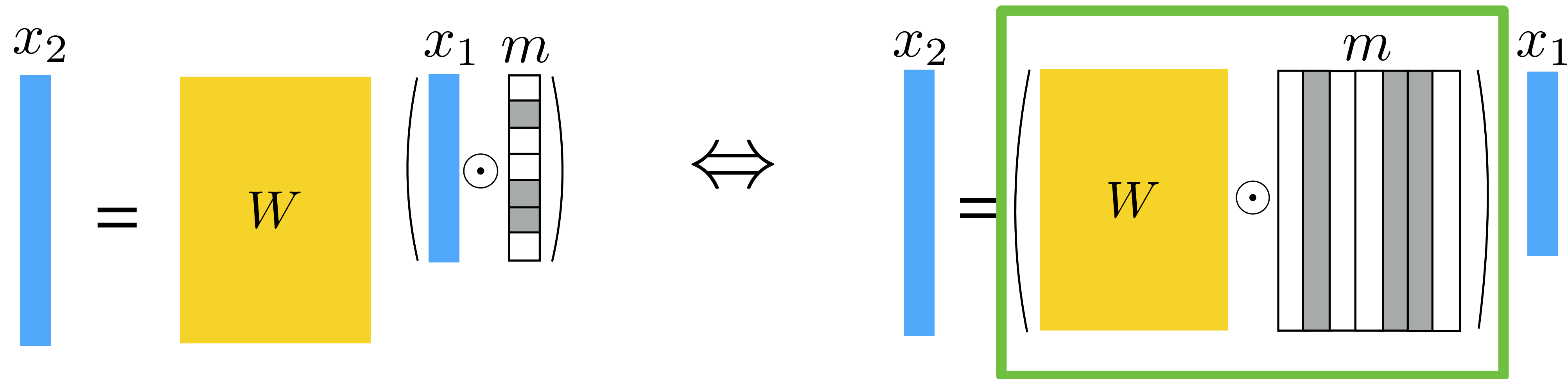
With dropout: $x_2 = W(x_1 \odot m)$, $m_i \sim \text{Bernoulli}(p)$, p — dropout rate



Example: binary dropout

Linear layer: $x_2 = W x_1$, W — weight matrix

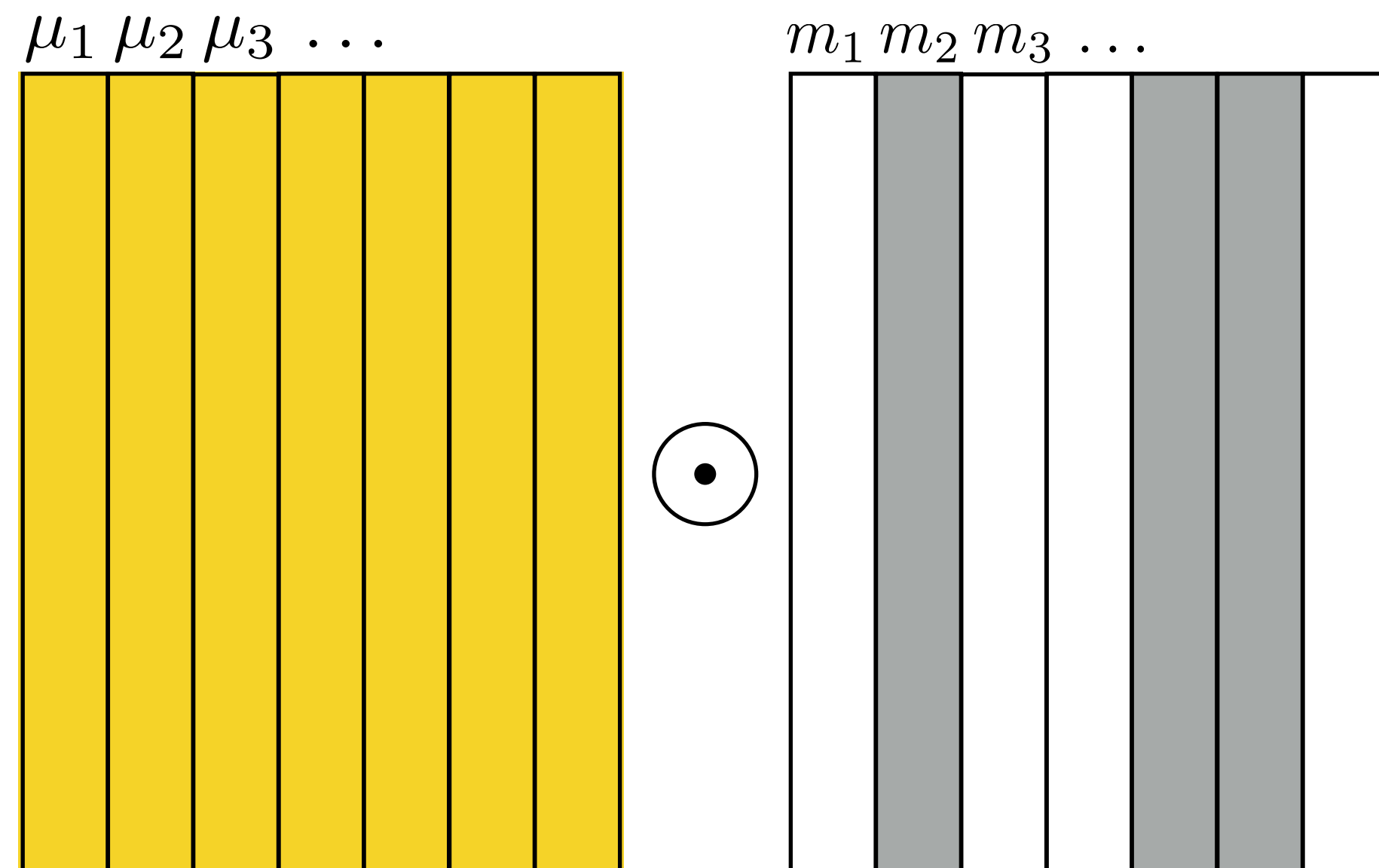
With dropout: $x_2 = W(x_1 \odot m)$, $m_i \sim \text{Bernoulli}(p)$, p — dropout rate



**Applying dropout means
sampling weights!**

Example: binary dropout

Weight matrix:



reparametrization

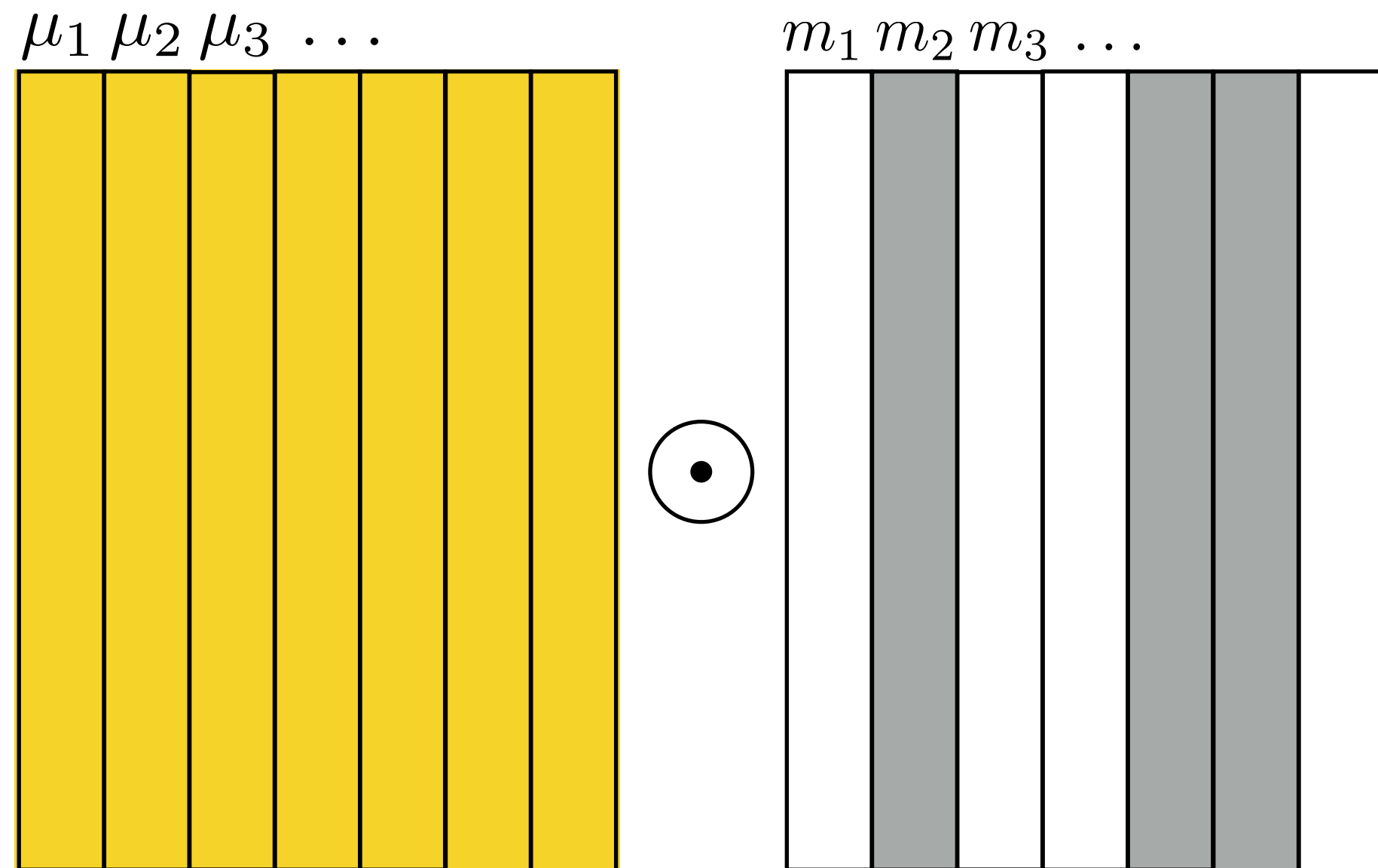
$$w_i = f(\mu_i, m_i) = \mu_i \cdot m_i$$

$$m_i \sim \text{Bernoulli}(p) \quad \leftarrow \text{0 or 1!}$$

p is a fixed hyperparameter!

Example: binary dropout

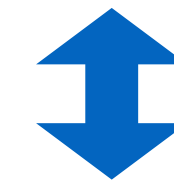
Weight matrix:



reparametrization

$$w_i = f(\mu_i, m_i) = \mu_i \cdot m_i$$

$$m_i \sim \text{Bernoulli}(p)$$



$$q(W | \mu) = \prod_i q(w_i | \mu_i)$$

$$q(w_i | \mu_i) = p \delta(0) + (1 - p) \delta(\mu_i)$$

p is a fixed hyperparameter!

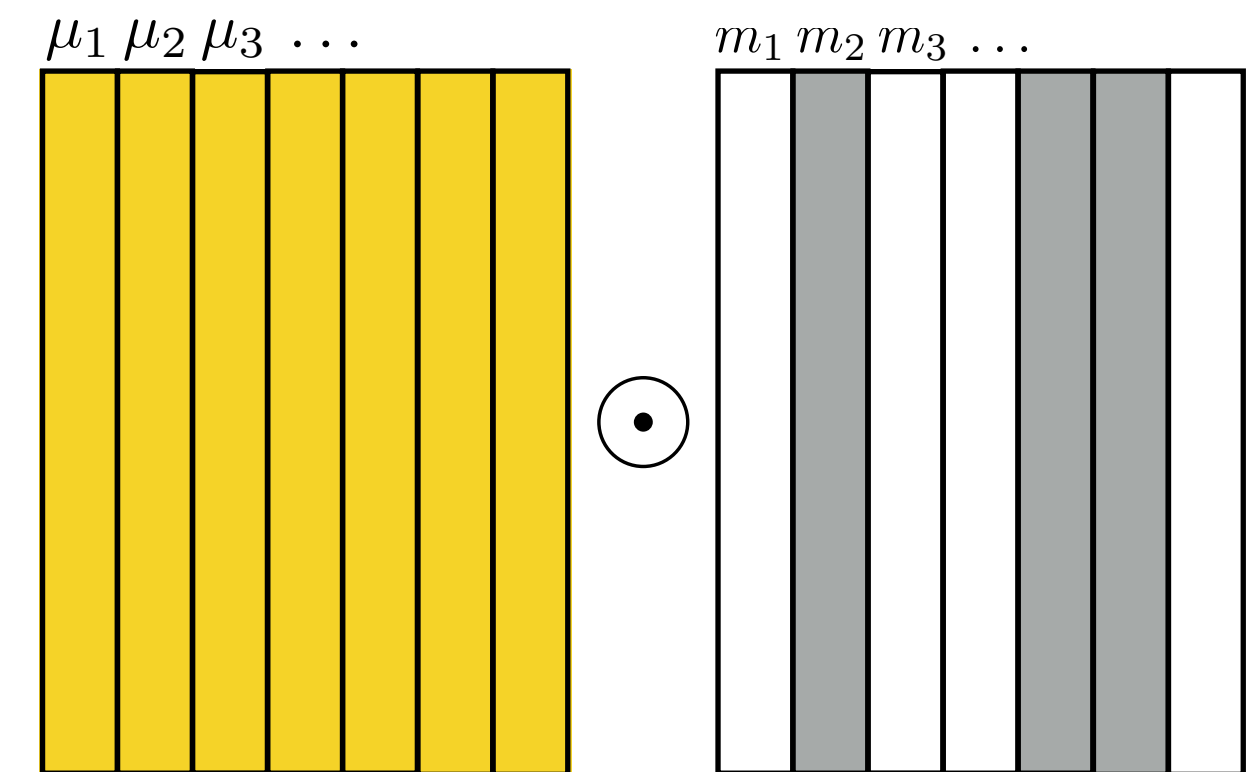
Example: binary dropout

Prior: ?

Approximate posterior: $q(W|\mu) = \prod_i q(w_i|\mu_i)$

$$q(w_i|\mu_i) = p \delta(0) + (1 - p) \delta(\mu_i)$$

Weight matrix:



Approximate KL-divergence: ?

Example: binary dropout

Prior: $p(W) = \prod_{i,j} p(w_{ij}), \quad p(w_{ij}) = \mathcal{N}(0, 1)$

Approximate posterior: $q(W|\mu) = \prod_i q(w_i|\mu_i)$

$$q(w_i|\mu_i) = p \delta(0) + (1 - p) \delta(\mu_i)$$

Approximate KL-divergence: ?

Example: binary dropout

Prior: $p(W) = \prod_{i,j} p(w_{ij}), \quad p(w_{ij}) = \mathcal{N}(0, 1)$

Approximate posterior: $q(W|\mu) = \prod_i q(w_i|\mu_i)$

$$q(w_i|\mu_i) = p \delta(0) + (1 - p) \delta(\mu_i)$$

Approximate KL-divergence: $KL(q(W|\mu)||p(W)) \approx \alpha \|\mu\|_2^2$

L2-regularization

Example: binary dropout, Bayesian benefits

Key messages of this example:

- Using binary dropout means being Bayesian!
- There are other dropout profits beyond regularization:
 - ensembling
 - uncertainty estimation

Plan

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- First example: binary dropout
- Second example: Bayesian sparsification

Compression of neural networks

- Deep neural networks achieve state-of-the-art performance in a variety of domains
- Model quality scales with model and dataset size
- State-of-the-art models usually incorporate **tens of millions of parameters**
- But **resources** (memory, processing time) **may be limited**



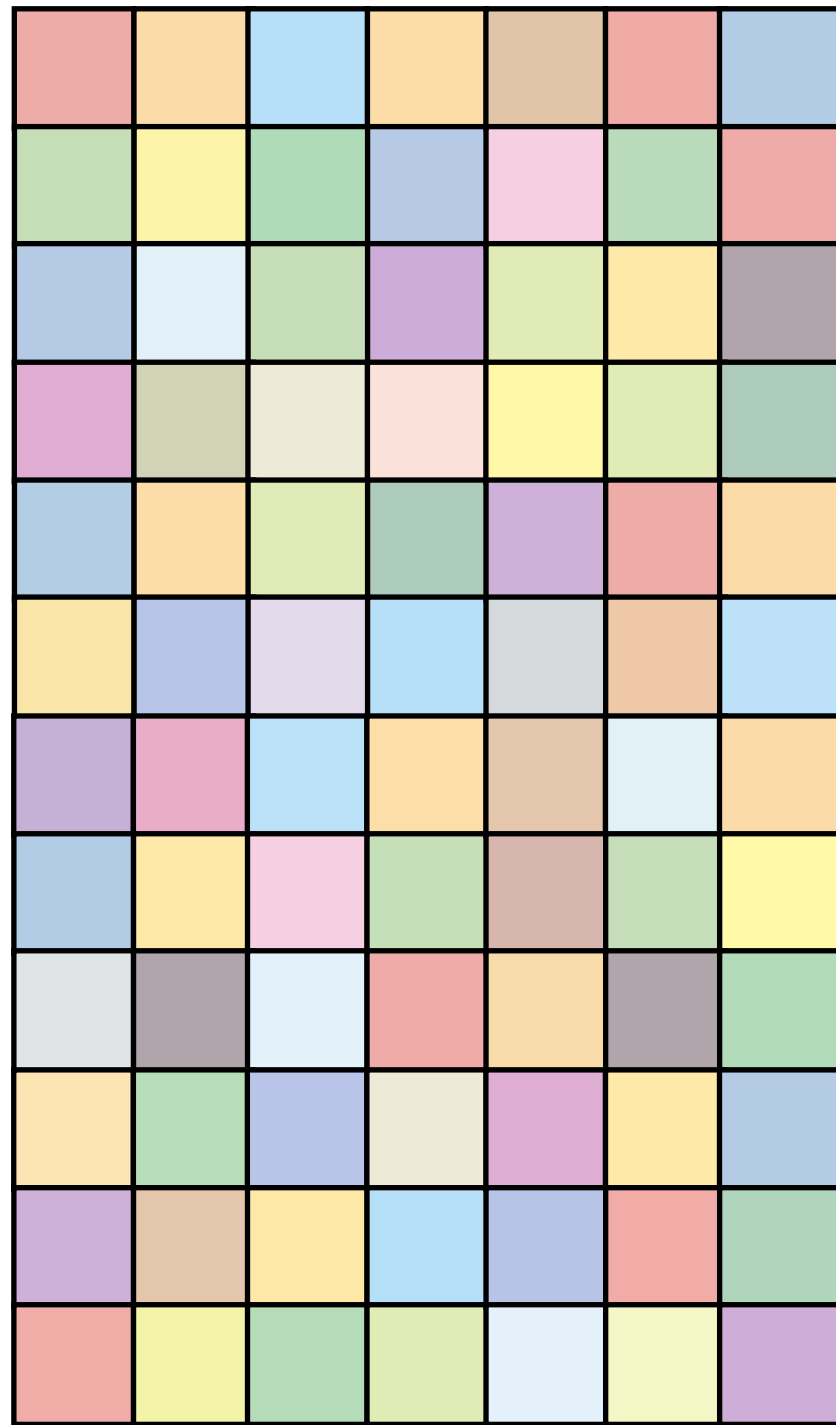
Compression of neural networks

- Deep neural networks achieve state-of-the-art performance in a variety of domains
- Model quality scales with model and dataset size
- State-of-the-art models usually incorporate **tens of millions of parameters**
- But **resources** (memory, processing time) **may be limited**

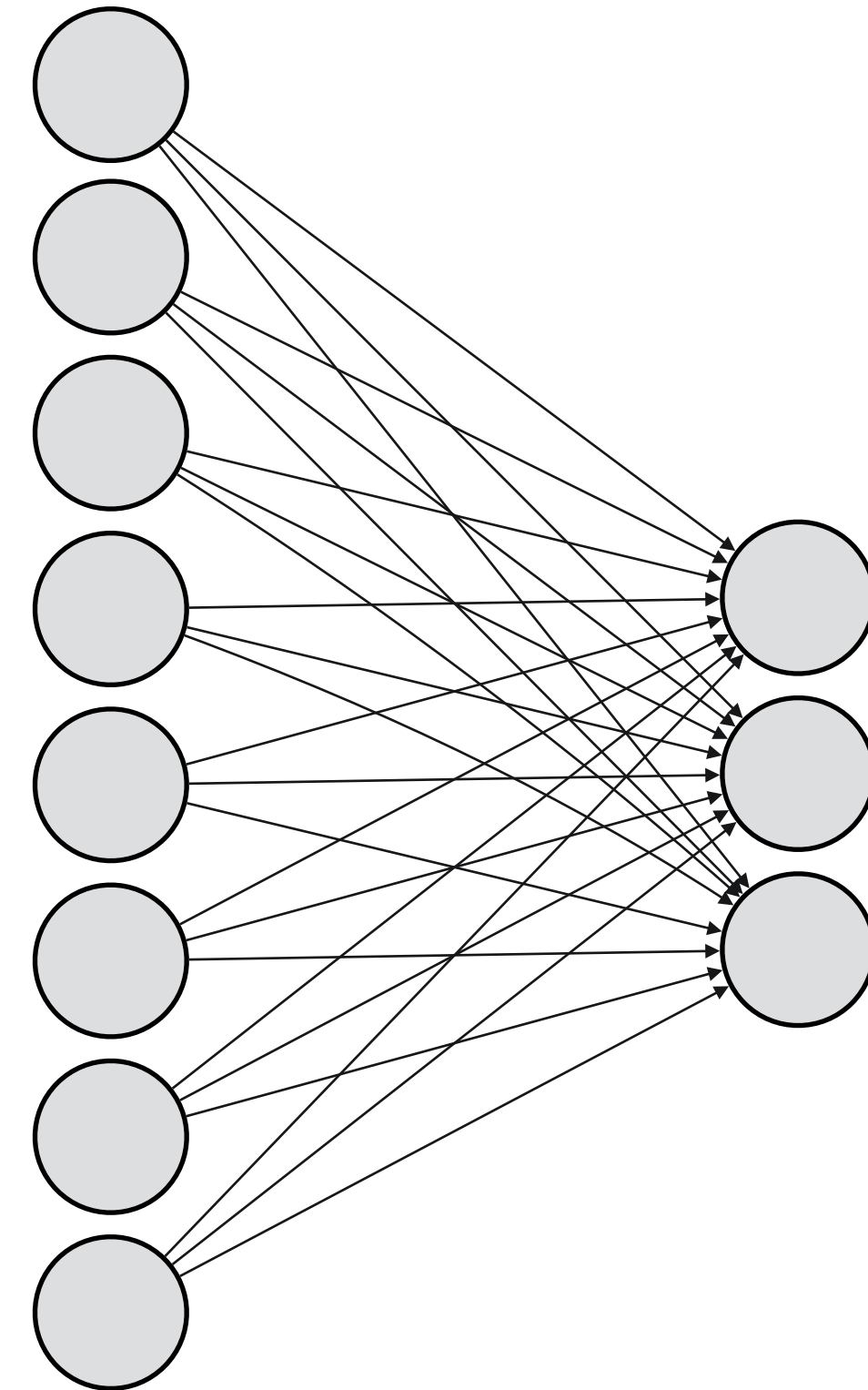


- One of the solutions — sparsification

Neural network

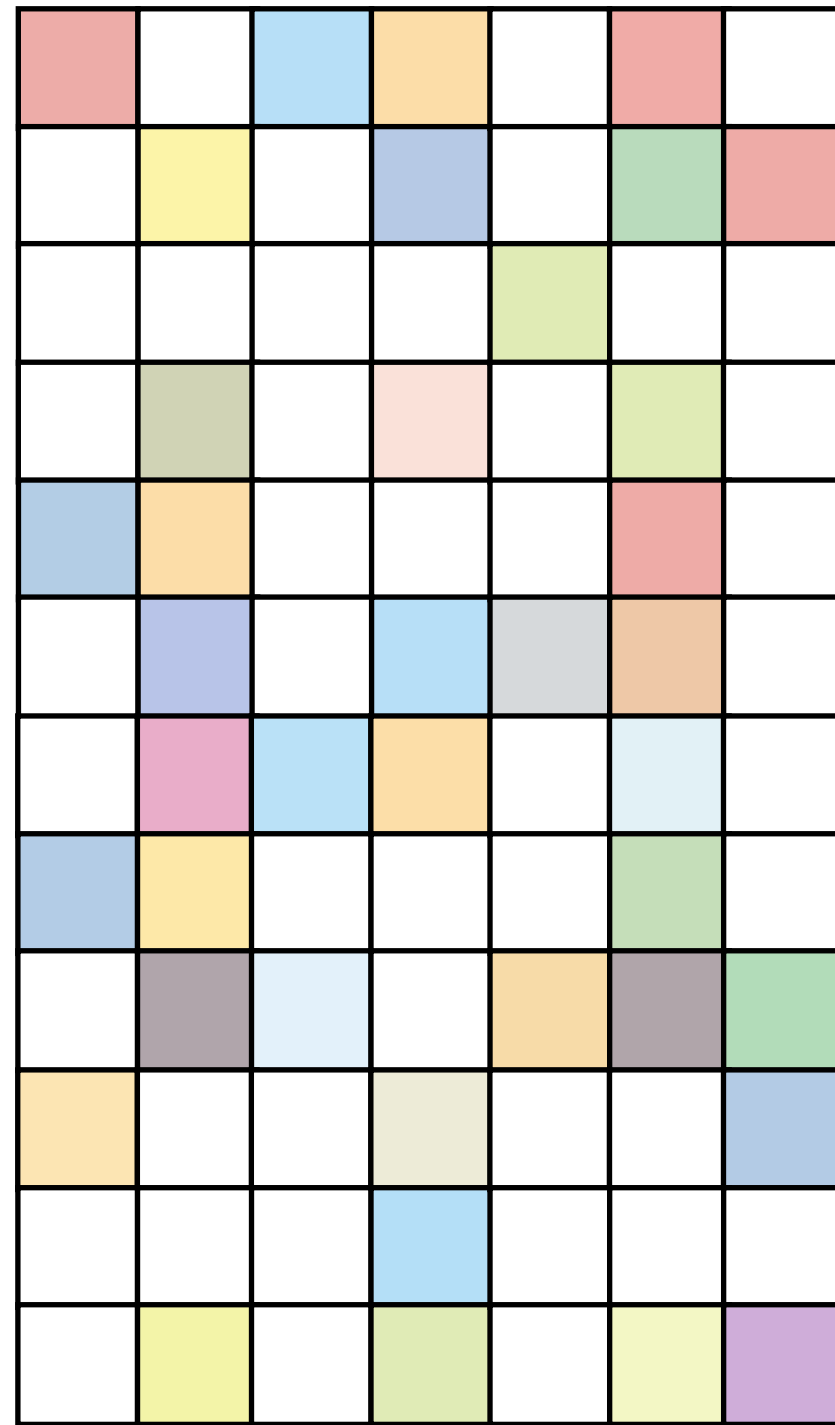


Weight matrix W



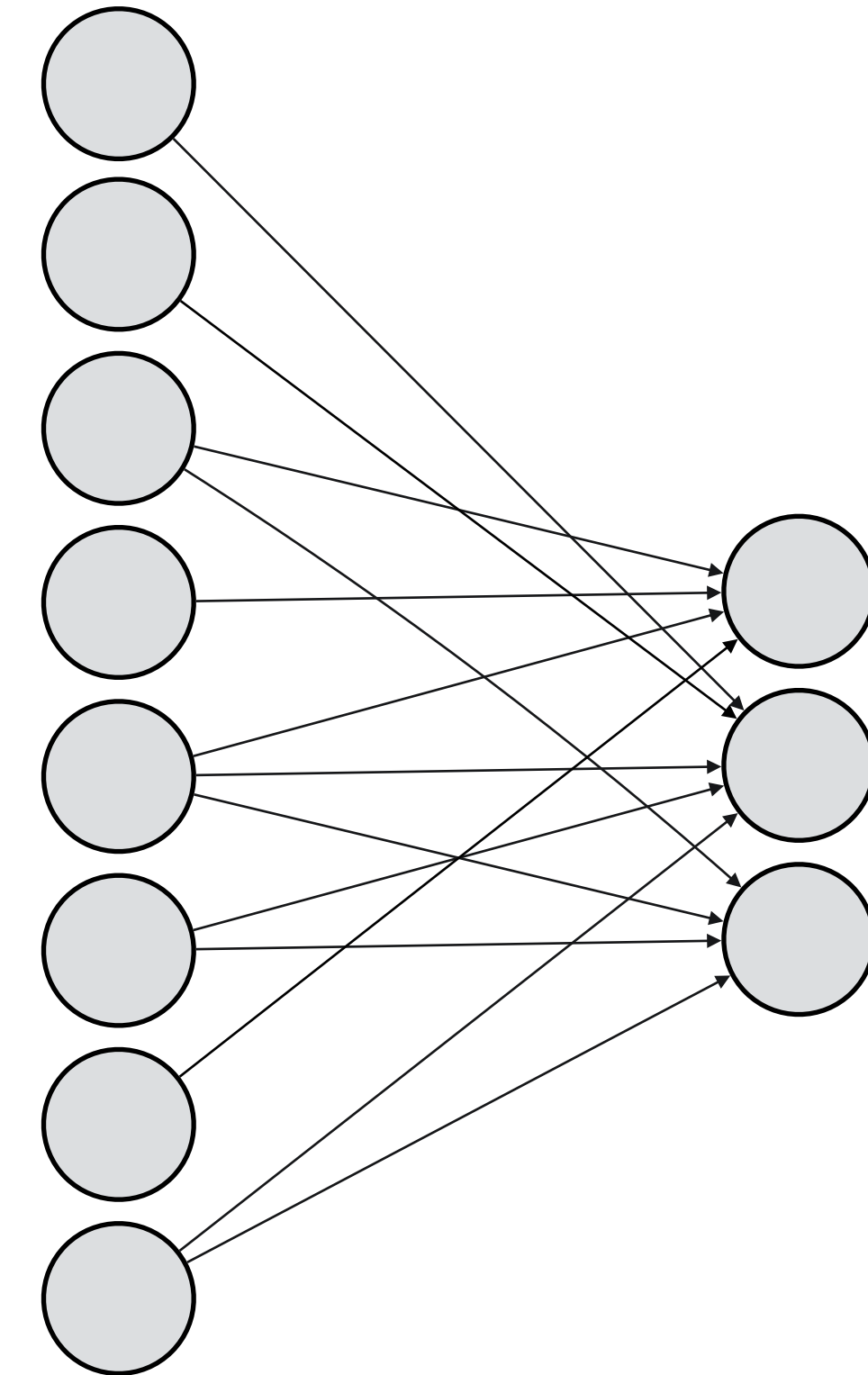
Computational graph

Sparse neural network



Weight matrix W

A lot of weights set to zero



Computational graph

From general framework to particular method

$$\sum_{i=1}^N \mathbb{E}_{q(w|\lambda)} \log p(y^i | x^i, w) - KL(q(w|\lambda) || p(w)) \rightarrow \max_{\lambda}$$

Model specification:

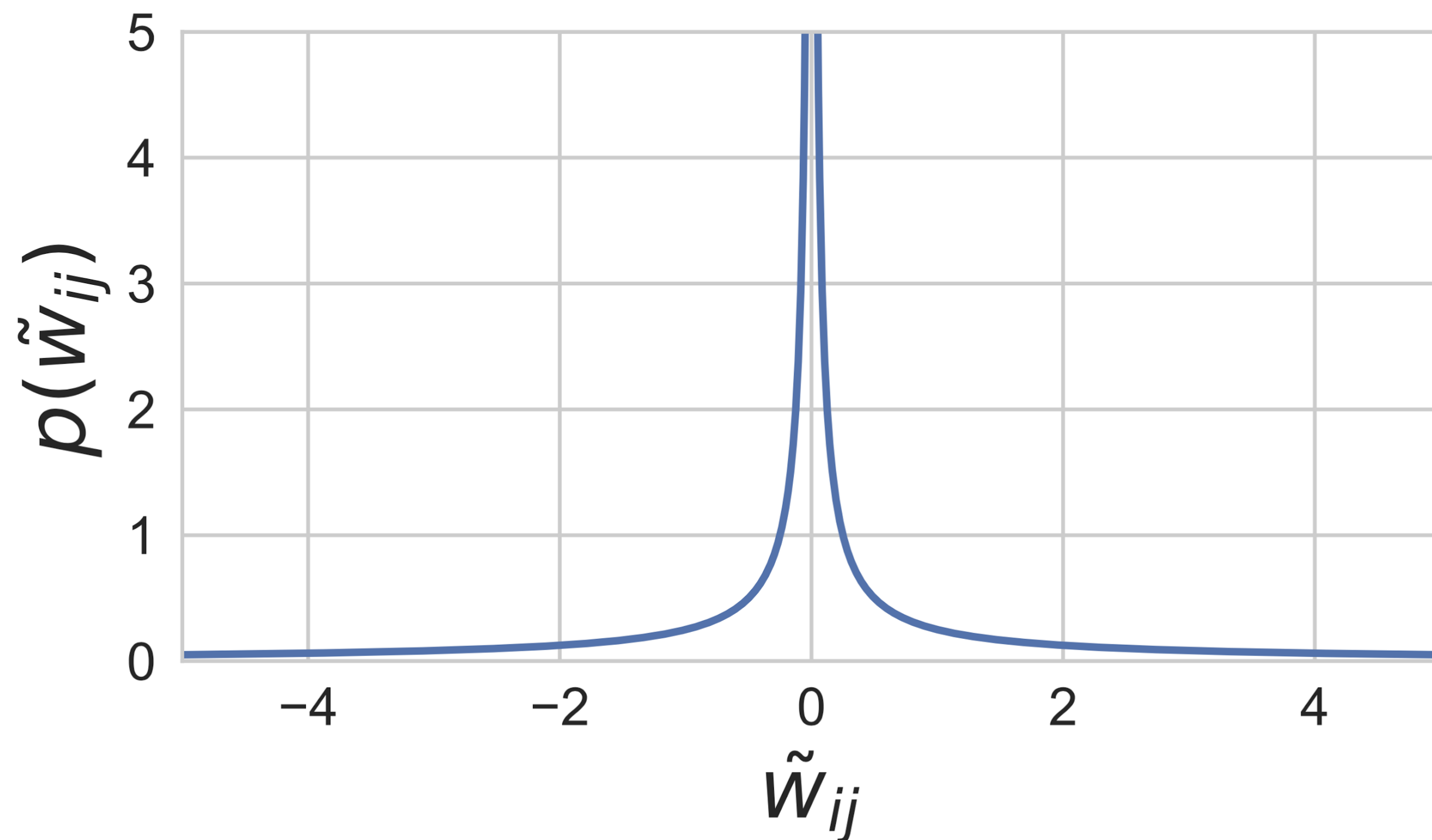
- Choose particular prior

Training:

- Choose particular family for approximate posterior
- How to compute the KL-divergence?

Example: sparse variational dropout

Prior: $p(w_{ij}) \propto \frac{1}{|w_{ij}|}$



Favors removing noisy weights!

Example: sparse variational dropout

Prior: $p(w_{ij}) \propto \frac{1}{|w_{ij}|}$

Approximate posterior: ?

Approximate KL-divergence: ?

Example: sparse variational dropout

Prior: $p(w_{ij}) \propto \frac{1}{|w_{ij}|}$

Approximate posterior: $q(w_{ij} | \mu_{ij}, \sigma_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$

Approximate KL-divergence: ?

Example: sparse variational dropout

Prior: $p(w_{ij}) \propto \frac{1}{|w_{ij}|}$

Approximate posterior: $q(w_{ij} | \mu_{ij}, \sigma_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$

Reparametrization: $\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}, \quad \hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$

Approximate KL-divergence: ?

Example: sparse variational dropout

Prior: $p(w_{ij}) \propto \frac{1}{|w_{ij}|}$

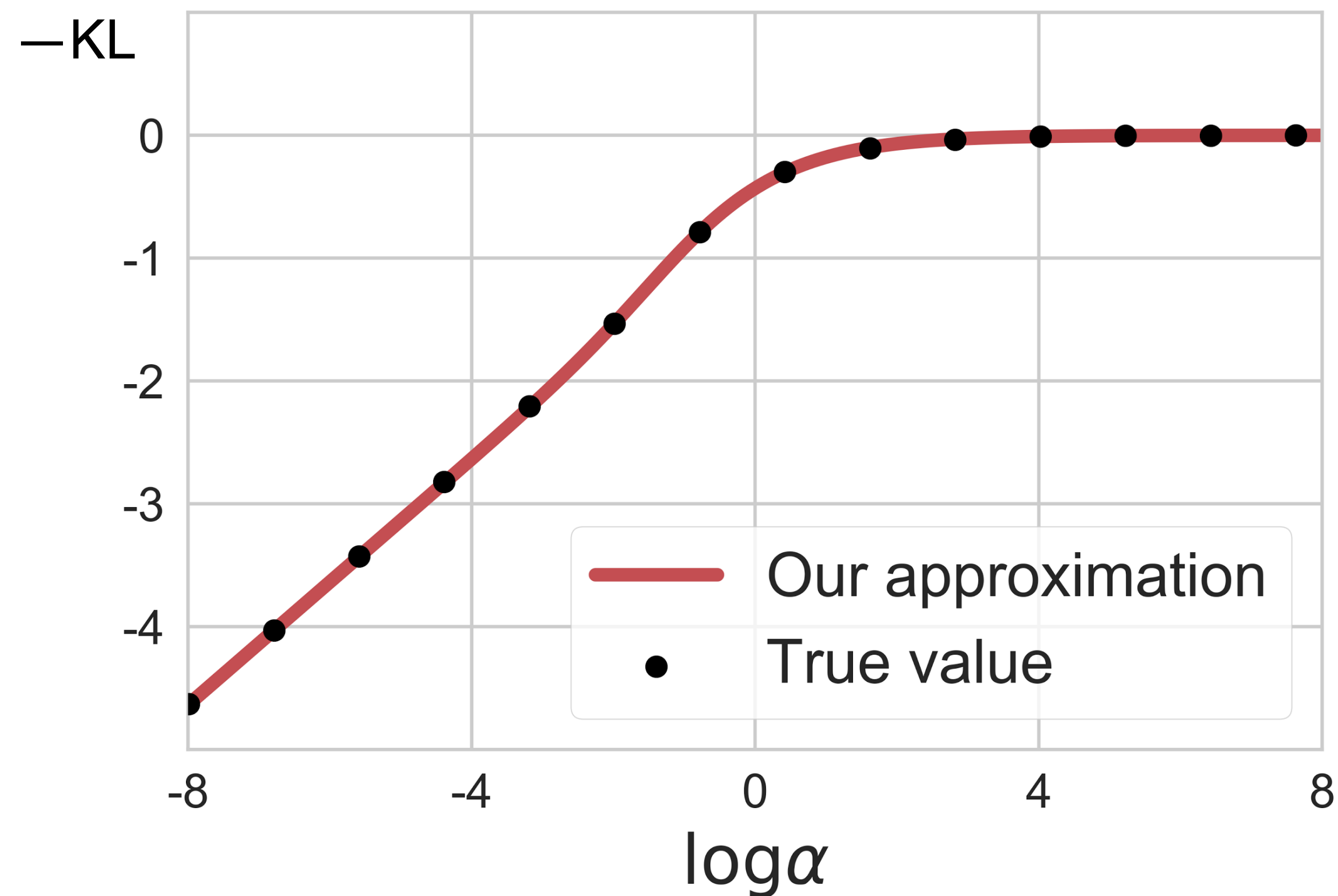
Approximate posterior: $q(w_{ij} | \mu_{ij}, \sigma_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$

Approximate KL-divergence: $-KL(q(w_{ij} | \mu_{ij}, \sigma_{ij}) || p(w_{ij})) \approx f_{KL}(\alpha_{ij})$

$$\alpha_{ij} = \frac{\sigma_{ij}^2}{\mu_{ij}^2}$$

Favors large $\alpha_{ij} \Rightarrow$ removing noisy weights

Approximating KL-divergence (fully factorized)



$$\begin{aligned} -KL(q(w_{ij}|\mu_{ij}, \sigma_{ij}) \parallel p(w_{ij})) &\approx \\ &\approx k_1 \sigma(k_2 + k_3 \log \alpha_{ij}) - 0.5 \log(1 + \alpha_{ij}^{-1}) + C \\ k_1 &= 0.63576 \quad k_2 = 1.87320 \quad k_3 = 1.48695 \end{aligned}$$

$$\alpha_{ij} = \frac{\sigma_{ij}^2}{\mu_{ij}^2}$$

- KL depends only on α_{ij}
- Favors large $\alpha_{ij} \Rightarrow$
removing noisy weights

Ok, sparsify weights. What about biases?

$$\sum_{i=1}^N \mathbb{E}_{q(w|\mu, \sigma)} \log p(y^i | x^i, w) - KL(q(w|\mu, \sigma) || p(w)) \rightarrow \max_{\mu, \log \sigma}$$

Treat biases as deterministic parameters and find a point estimate:

$$\sum_{i=1}^N \mathbb{E}_{q(w|\mu, \sigma)} \log p(y^i | x^i, w, \mathbf{b}) - KL(q(w|\mu, \sigma) || p(w)) \rightarrow \max_{\mu, \log \sigma, \mathbf{b}}$$

Final algorithm

Training on a mini-batch X with labels Y :

1. Sample weights: $\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}, \quad \hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$
2. Forward pass: $Y_{\text{pred}} = NN(X, \hat{w}, b)$
3. Backward pass + SGD step: compute stochastic gradients of ELBO:
$$\nabla_{\mu, \log \sigma, b} \left(N \cdot \text{Loss}(Y, Y_{\text{pred}}) + \text{SparseReg}(\sigma / \mu) \right)$$

Final algorithm

Training on a mini-batch X with labels Y :

1. Sample weights: $\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}$, $\hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$
2. Forward pass: $Y_{\text{pred}} = NN(X, \hat{w}, b)$
3. Backward pass + SGD step: compute stochastic gradients of ELBO:

$$\nabla_{\mu, \log \sigma, b} \left(N \cdot \text{Loss}(Y, Y_{\text{pred}}) + \text{SparseReg}(\sigma/\mu) \right)$$

Pruning after training:

If $\mu_{ij}^2 / \sigma_{ij}^2 < \text{threshold}$:

$$\mu_{ij} = 0, \sigma_{ij} = 0$$

signal-to-noise ratio

Final algorithm

Training on a mini-batch X with labels Y :

1. Sample weights: $\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}$, $\hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$
2. Forward pass: $Y_{\text{pred}} = NN(X, \hat{w}, b)$
3. Backward pass + SGD step: compute stochastic gradients of ELBO:

$$\nabla_{\mu, \log \sigma, b} \left(N \cdot \text{Loss}(Y, Y_{\text{pred}}) + \text{SparseReg}(\sigma/\mu) \right)$$

Pruning after training:

If $\mu_{ij}^2 / \sigma_{ij}^2 < \text{threshold}$:

$$\mu_{ij} = 0, \sigma_{ij} = 0$$

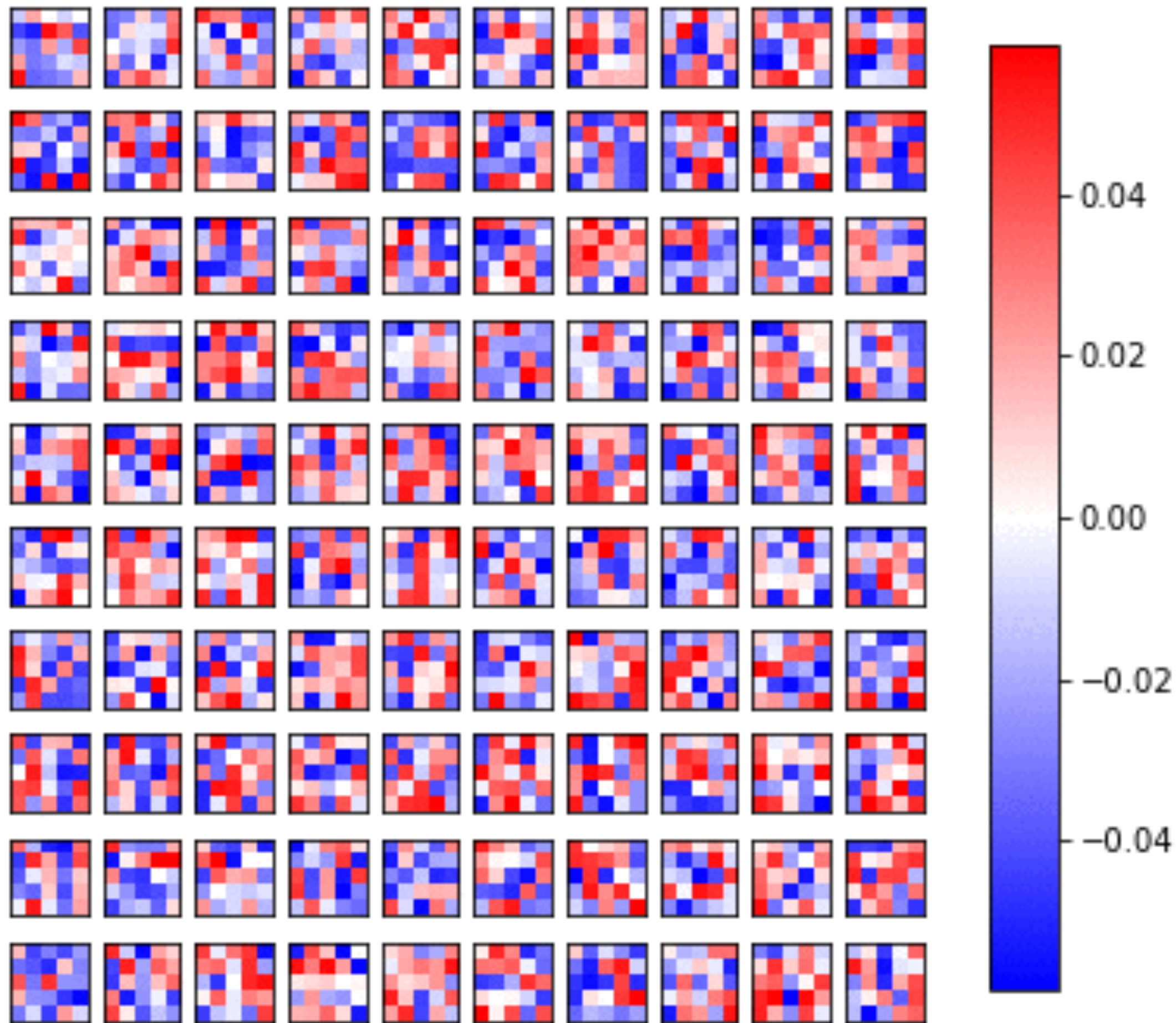
Prediction for a mini-batch X :

Return $Y_{\text{pred}} = NN(X, \mu, b)$

do not ensemble because we want the most compact and fast network

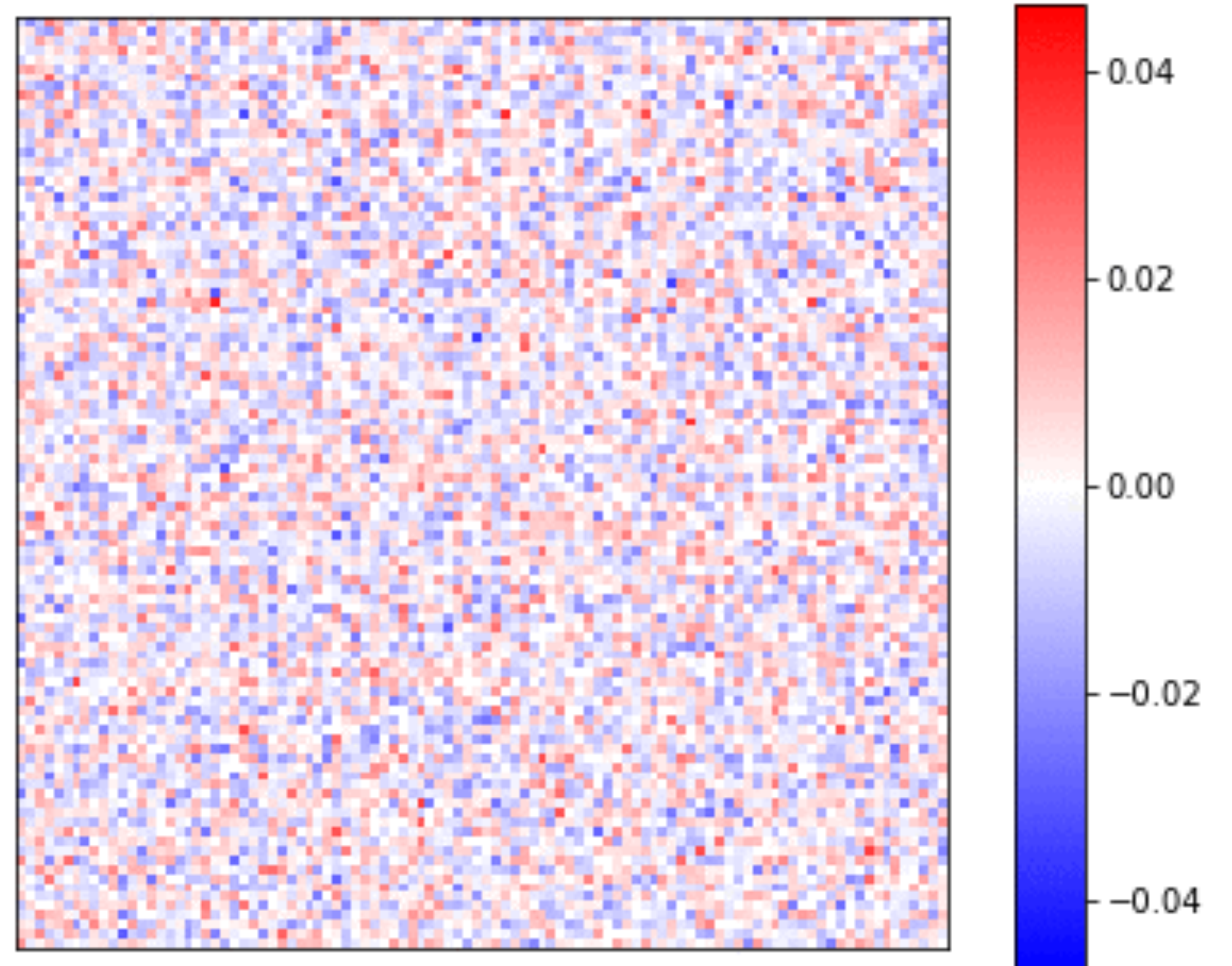
Sparse variational dropout: visualization

Epoch: 0 Compression ratio: 1x Accuracy: 8.4



LeNet-5: convolutional layer

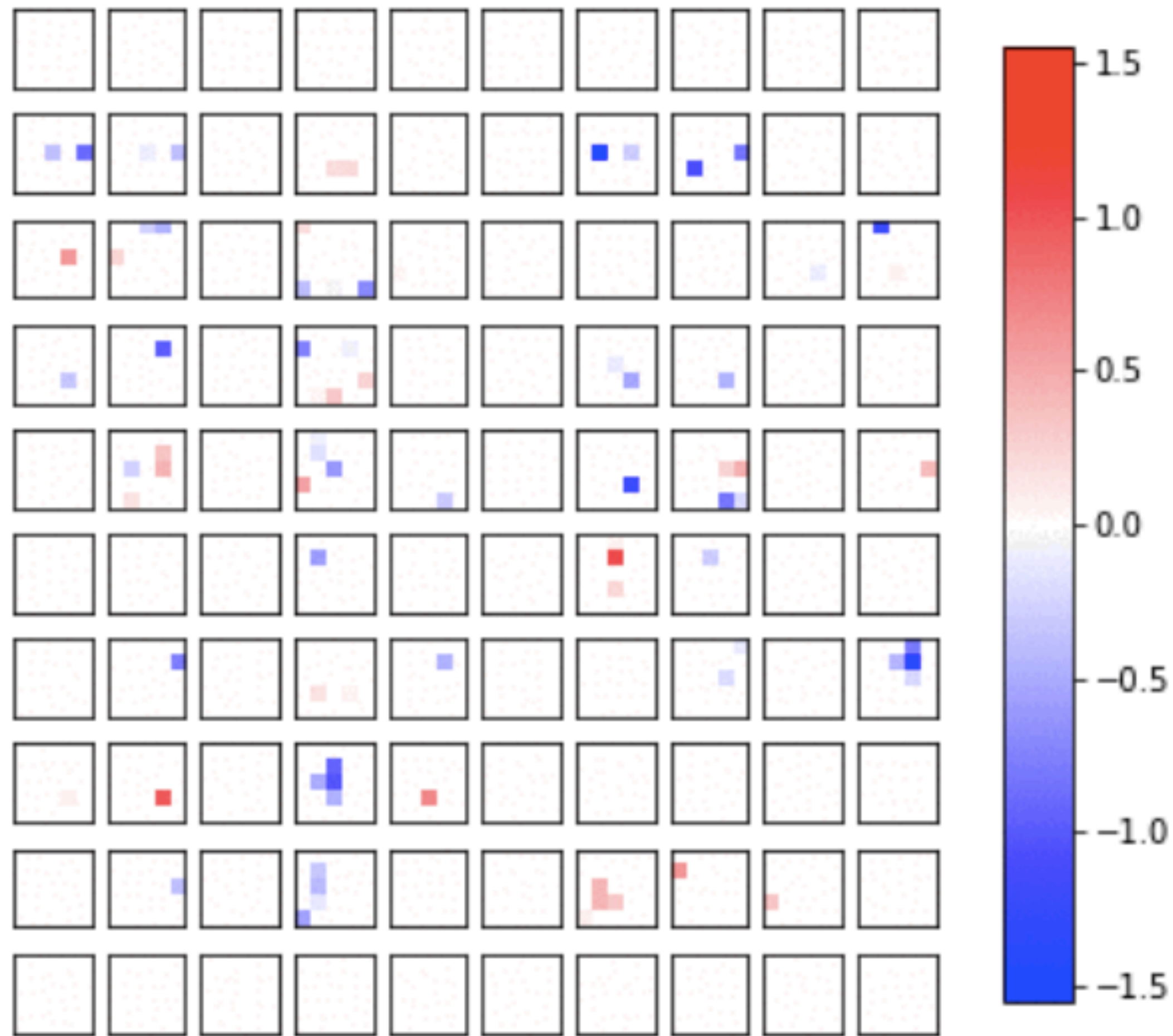
Epoch: 0 Compression ratio: 1x Accuracy: 8.4



LeNet-5: fully-connected layer
(100 x 100 patch)

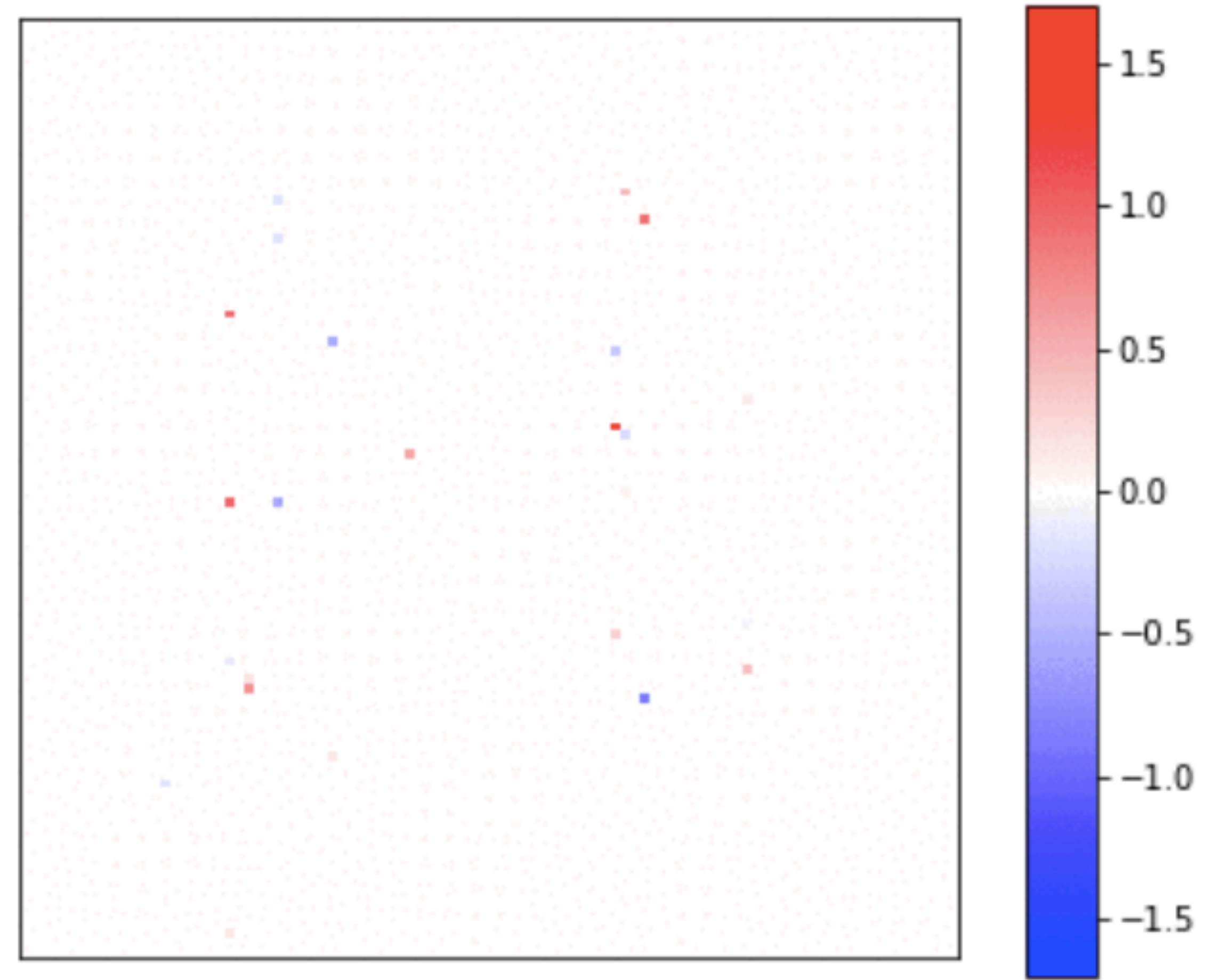
Sparse variational dropout: visualization

Epoch: 200 Compression ratio: 270x Accuracy: 99.3



LeNet-5: convolutional layer

Epoch: 200 Compression ratio: 270x Accuracy: 99.3



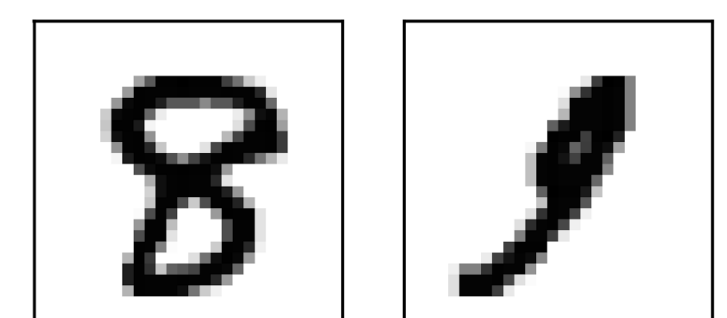
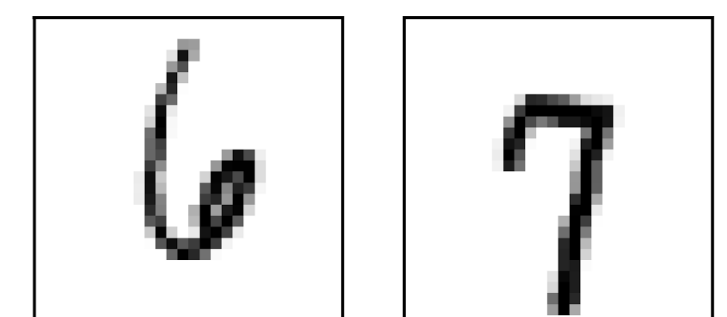
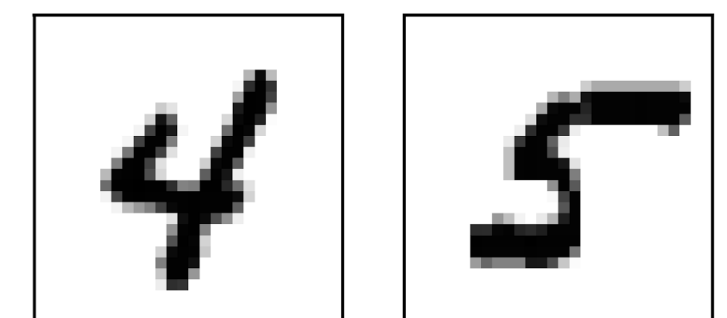
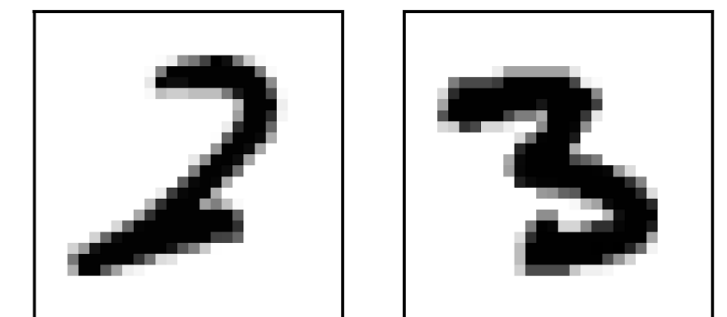
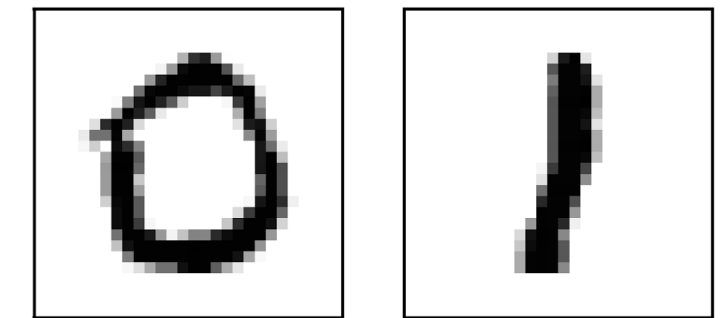
LeNet-5: fully-connected layer
(100 x 100 patch)

Lenet-5-Caffe and Lenet-300-100 on MNIST

Fully Connected network: LeNet-300-100

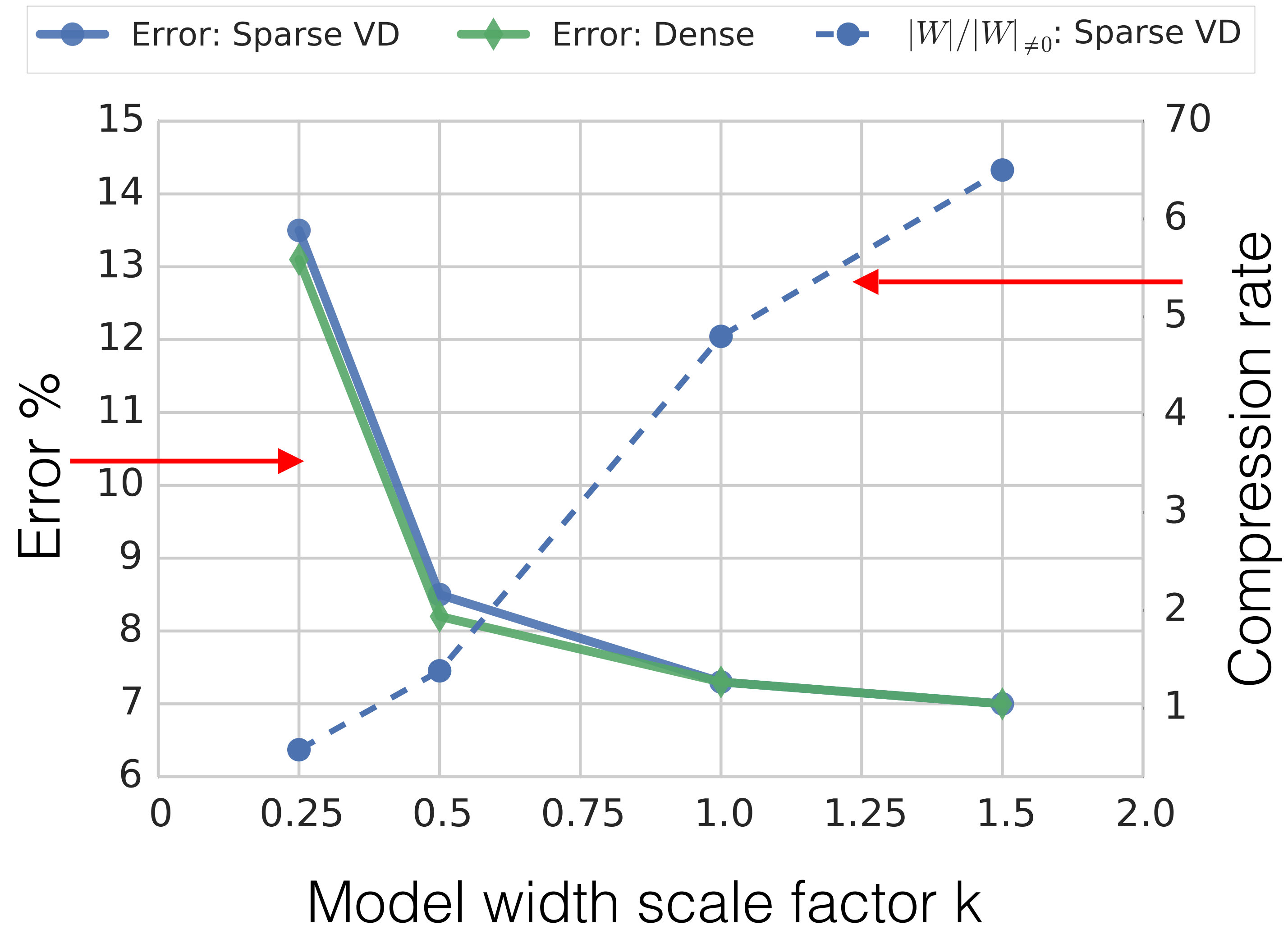
Convolutional network: Lenet-5-Caffe

Network	Method	Error %	Sparsity per Layer %	$\frac{ W }{ W_{\neq 0} }$
LeNet-300-100	Original	1.64		1
	Pruning	1.59	92.0 – 91.0 – 74.0	12
	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
LeNet-5-Caffe	Original	0.80		1
	Pruning	0.77	34 – 88 – 92.0 – 81	12
	DNS	0.91	86 – 97 – 99.3 – 96	111
	SWS	0.97		200
	(ours) Sparse VD	0.75	67 – 98 – 99.8 – 95	280



VGG-like on CIFAR-10

Number of filters / neurons is linearly scaled by k (the width of the network)



Random Labeling



Dataset	Architecture	Train Acc.	Test Acc.	Sparsity
MNIST	FC + BD	100%	10%	—
MNIST	FC + Sparse VD	10%	10%	100%
CIFAR-10	VGG + BD	100%	10%	—
CIFAR-10	VGG + Sparse VD	10%	10%	100%

No dependency between data and labels \Rightarrow Sparse VD yields an empty model where conventional models easily overfit.

Sparse variational dropout: key messages

- Prior distribution can encode our desirable model properties (e. g. sparse weights)
- Other Bayesian compression techniques:
 - group sparsification (removing neurons / filters)
 - quantization (low-precision weights)

Summary

- A lot of BNN advantages: regularization, ensembling, uncertainty estimation, ...
- To train BNN, one should optimize ELBO using DSVI & RT
- Three steps towards a particular method
- Using binary dropout means being Bayesian
- Prior distribution can encode our desirable model properties

Software

- Pyro (based on PyTorch)
- TensorFlow Probability (based on TensorFlow)
- Edward (based on TensorFlow)
- PyMC (based on Theano, pre-release with TensorFlow Probability)
- <https://github.com/JavierAntoran/Bayesian-Neural-Networks> - PyTorch implementations of popular Bayesian deep learning papers

Practical assignment

https://github.com/nadiinchi/bm_tutorial
assignment_practice.ipynb

