# Bayesian Sparsification of Neural Networks

Nadezhda Chirkova

Higher School of Economics, Samsung AI Laboratory
Moscow, Russia

Probabilistic AI summer school
June 3-7, 2019. Trondheim, Norway

# Agenda

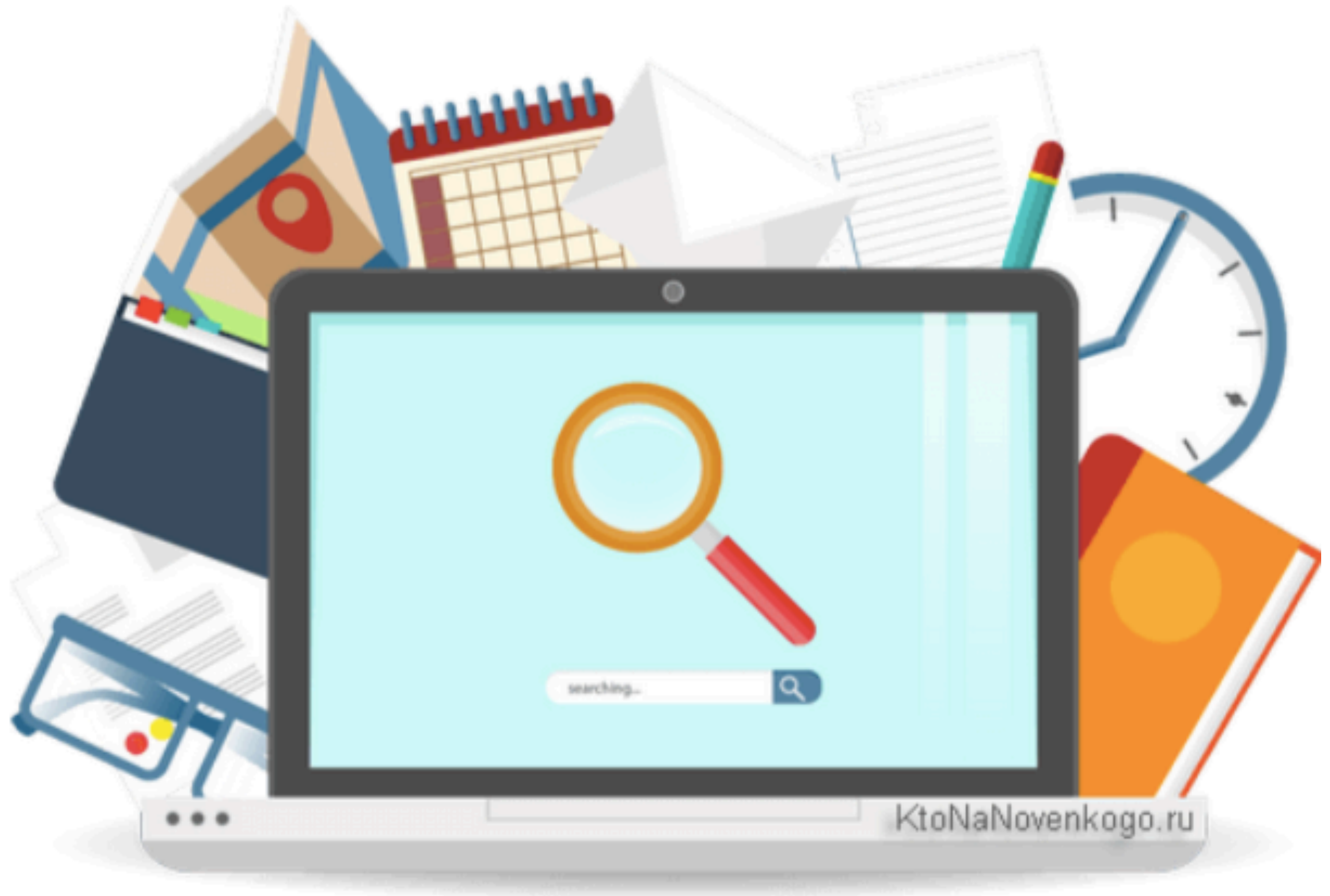
- Sparsification: what and why
- Bayesian neural networks
- Sparse variational dropout
- Practical assignment: implementation of SparseVD
- Model enhancements

# Agenda

- $\boxed{\text{Sparsification: what and why}}$
- Bayesian neural networks
- Sparse variational dropout
- Practical assignment: implementation of SparseVD
- Model enhancements

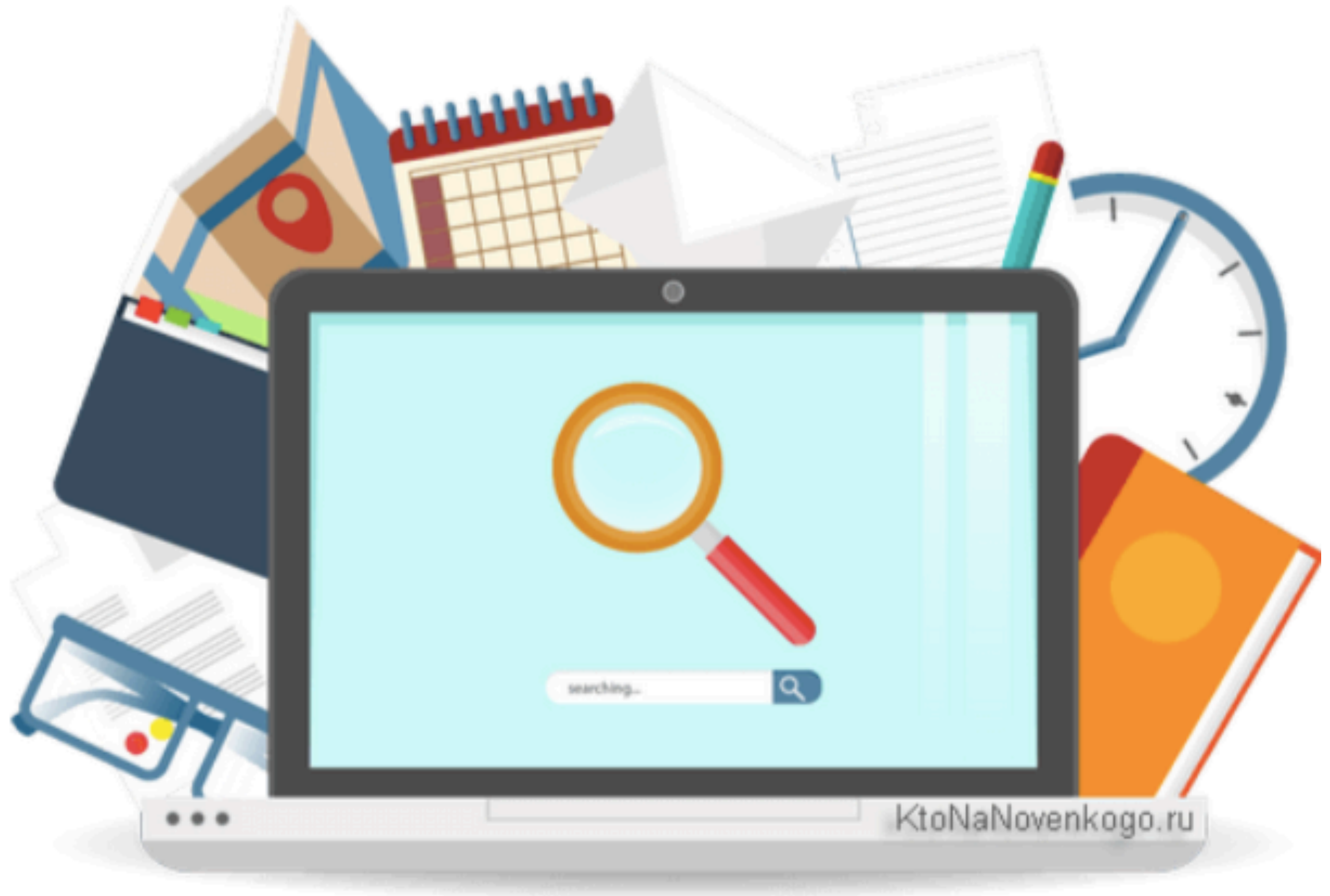
# Compression of neural networks

- 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**

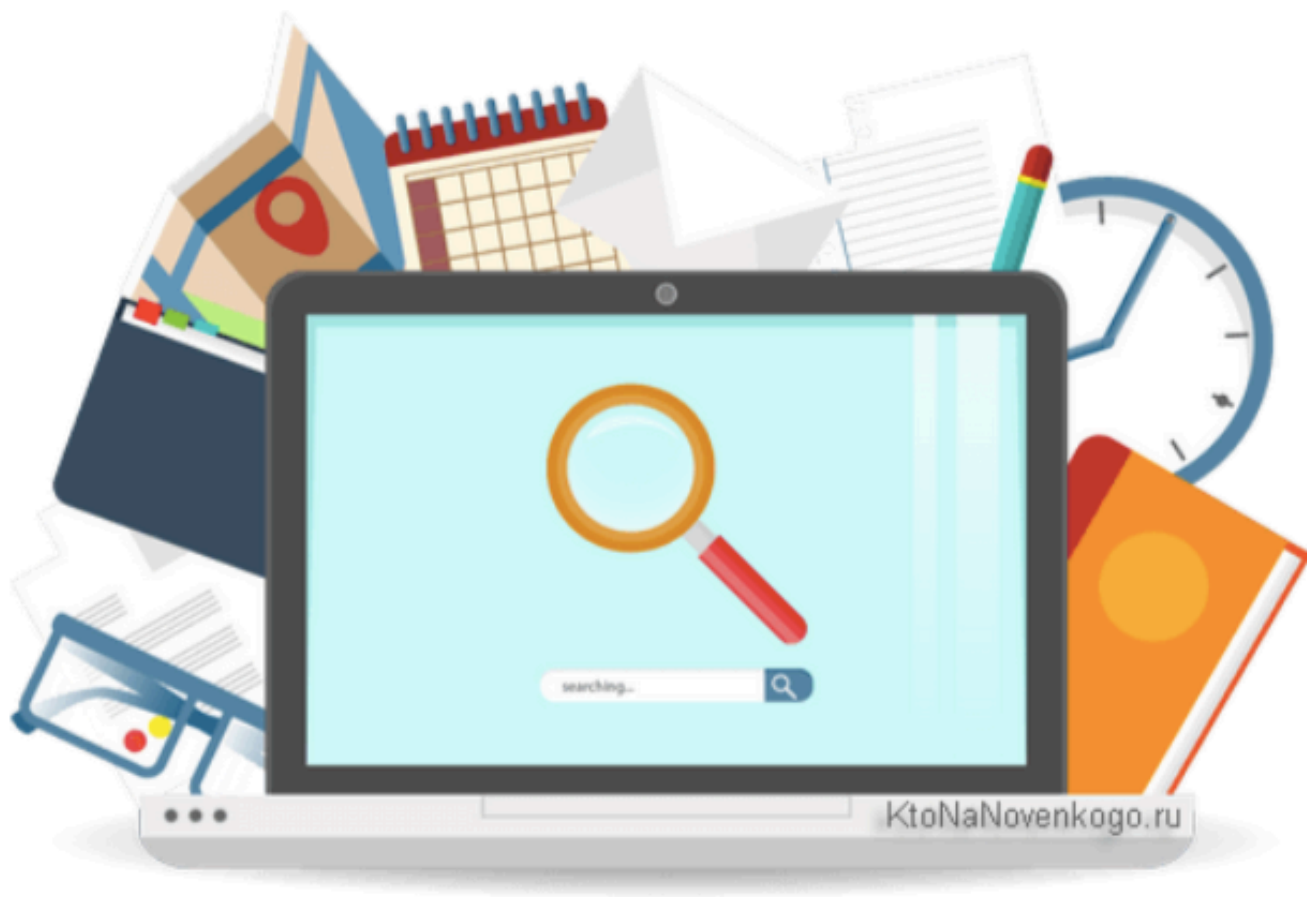



**Urgent industrial problem!**

# 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**






**Urgent industrial problem! well solved using Bayesian deep learning!**

# 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**

**Methods for neural network compression:**

- Quantization
- Matrix factorizations
- Sparsification

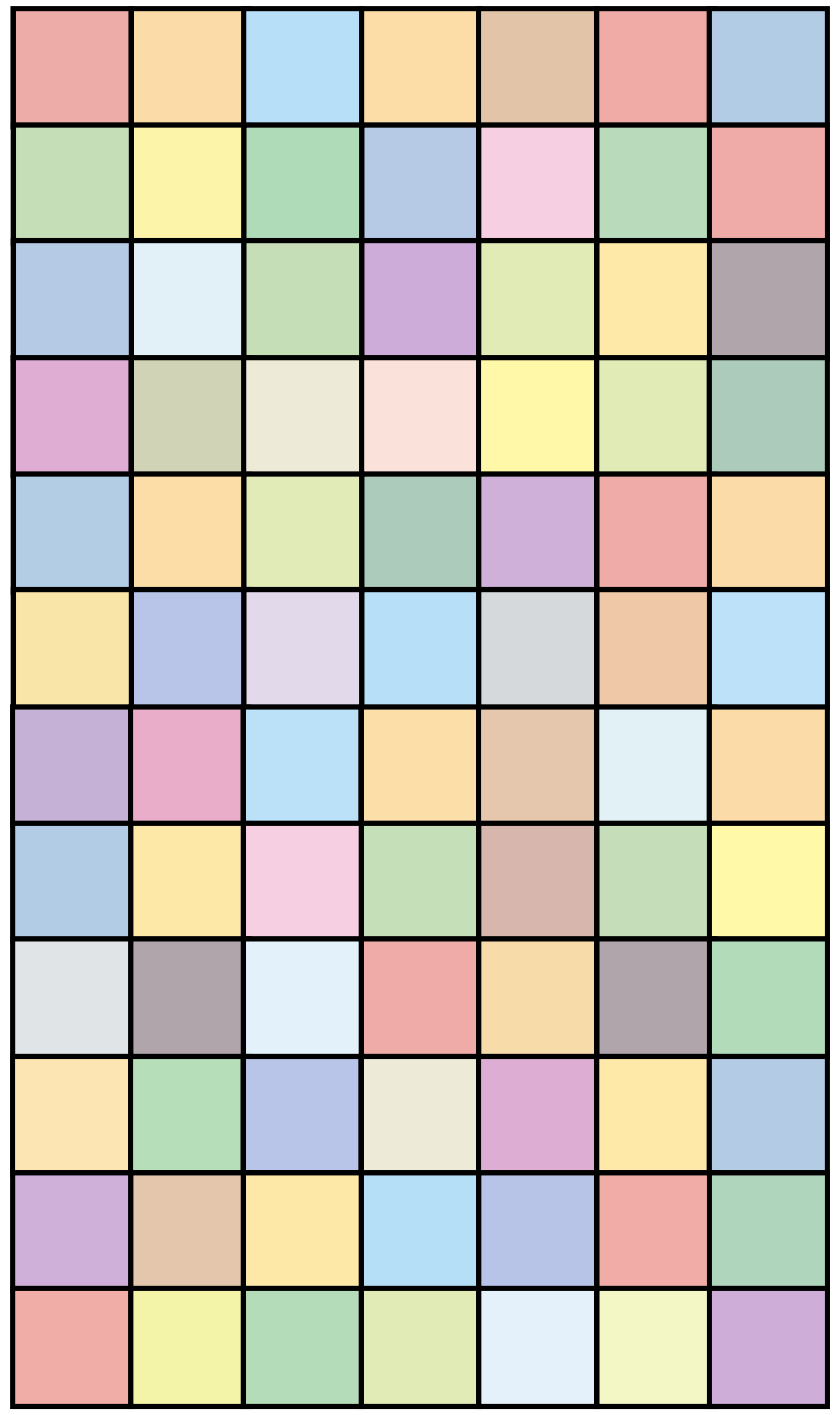
# Compression of neural networks

- 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**

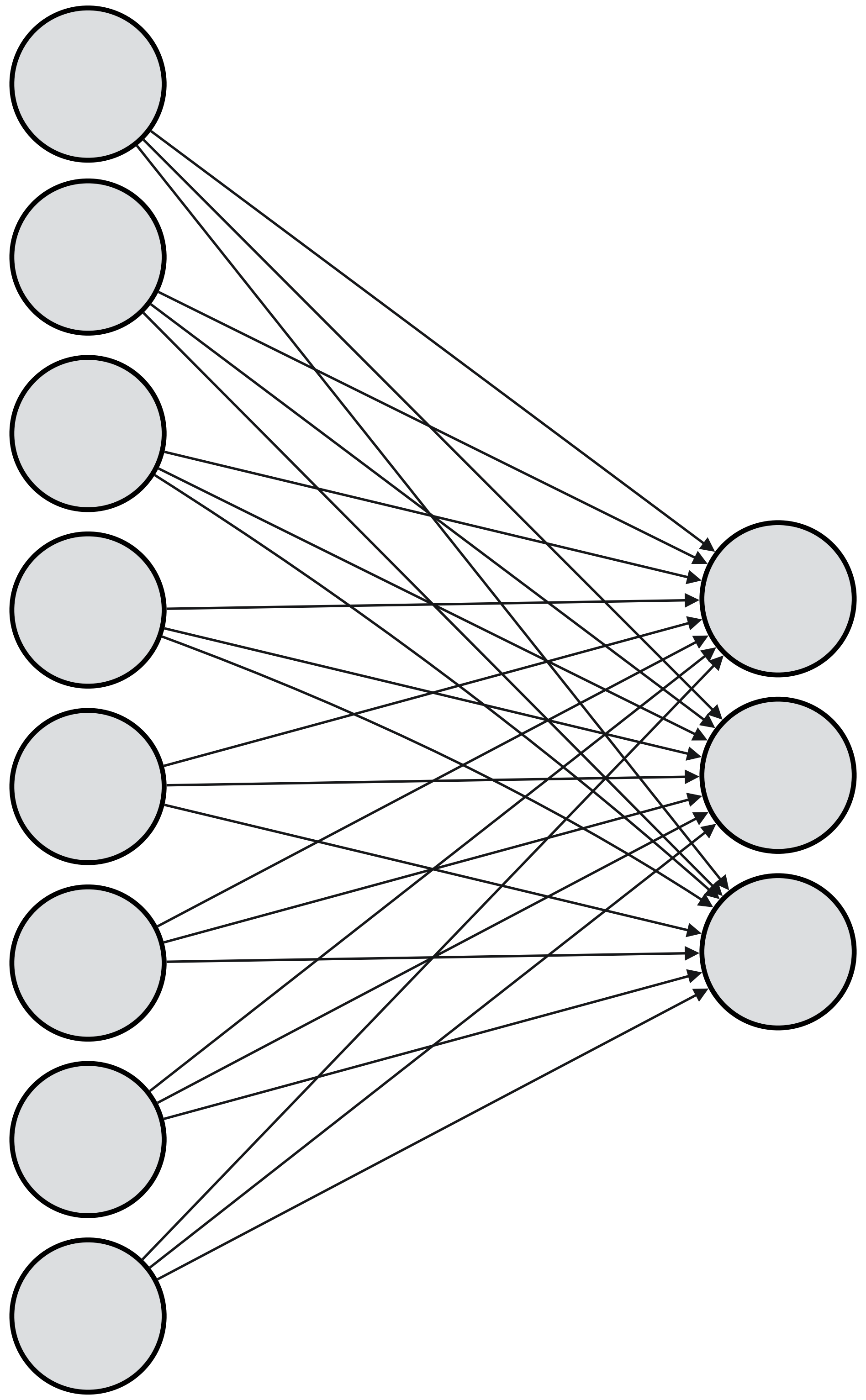
**Methods for neural network compression:**

- Quantization
- Matrix factorizations
- Sparsification

# Neural network

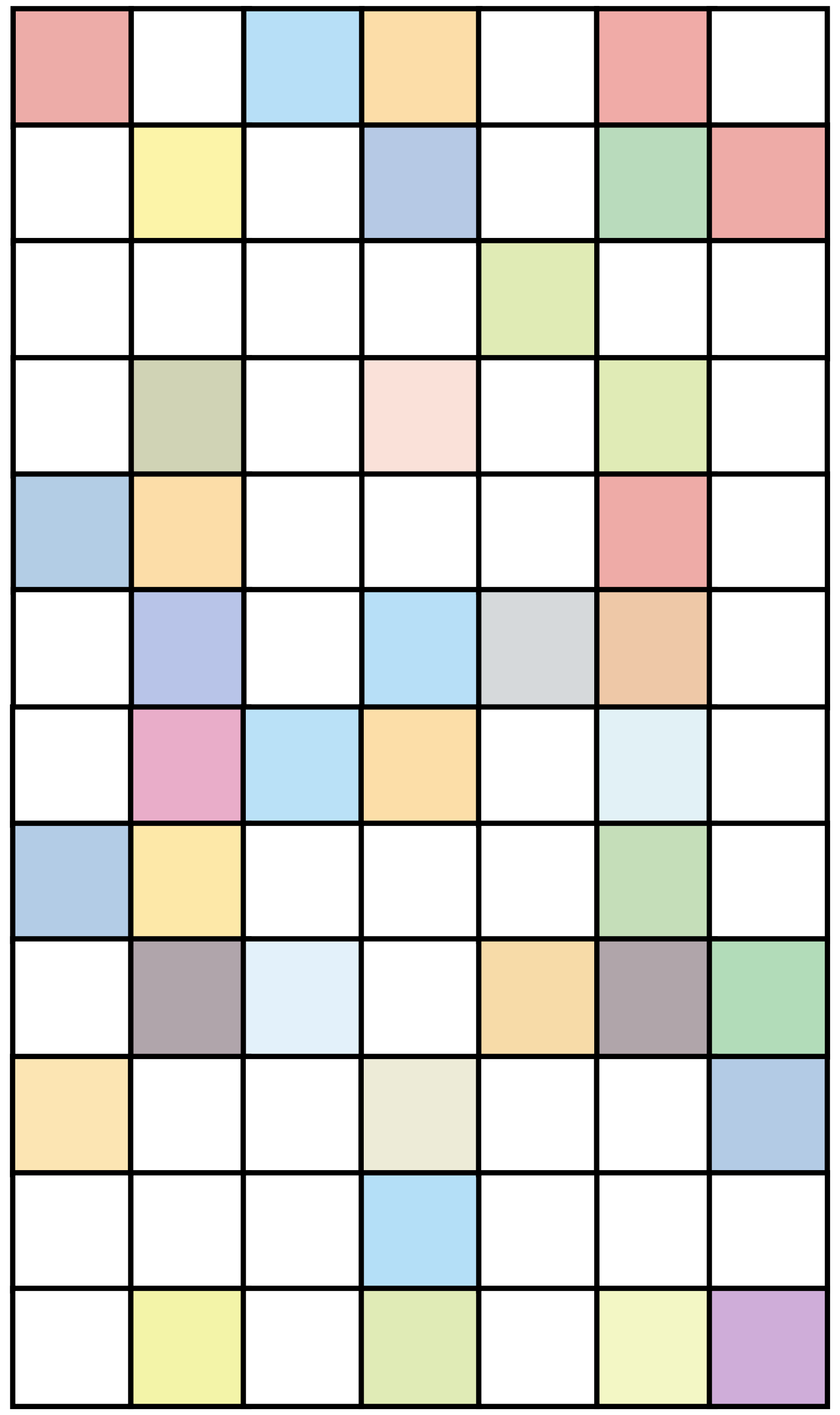


Weight matrix W



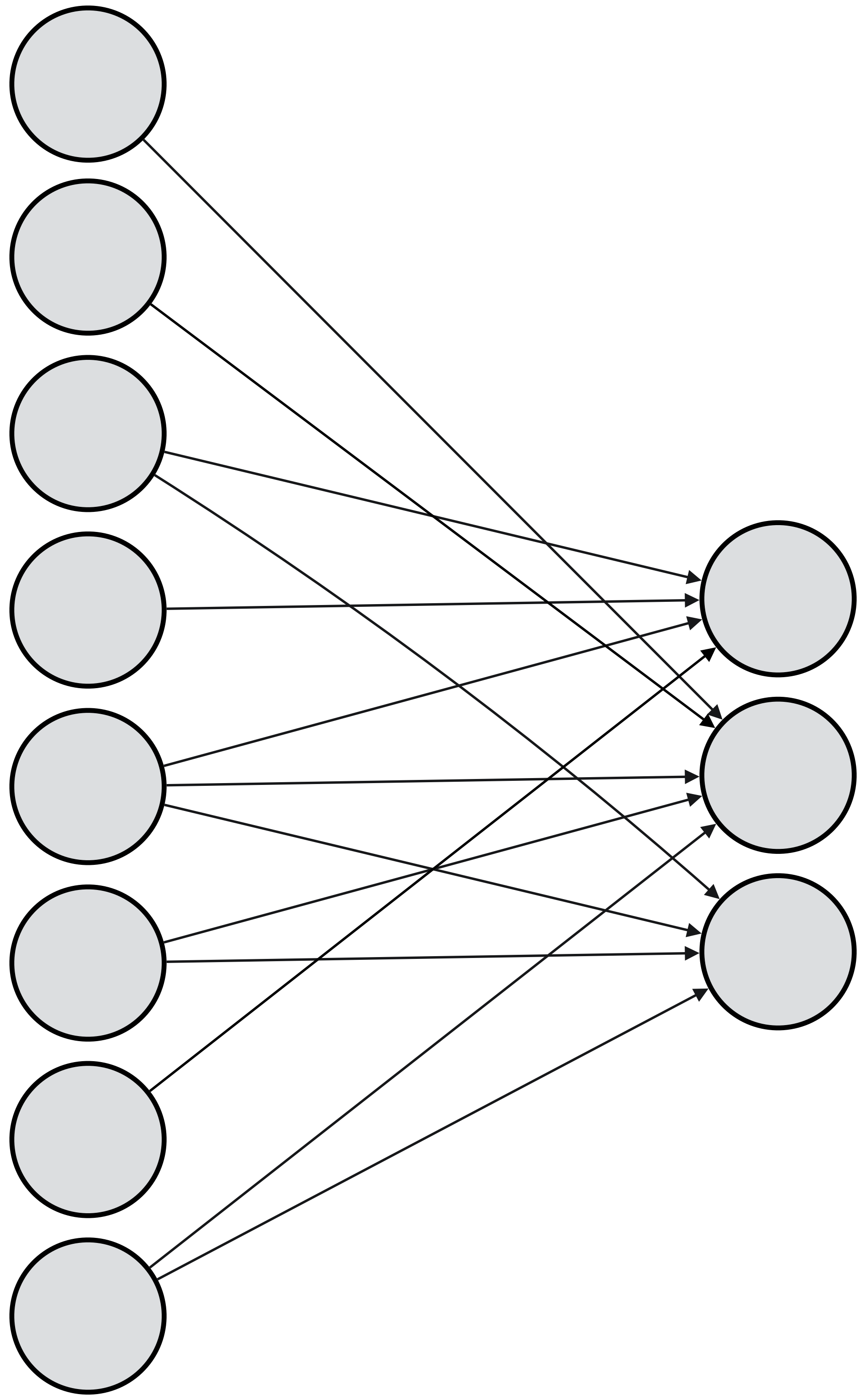
Computational graph

# Sparse neural network



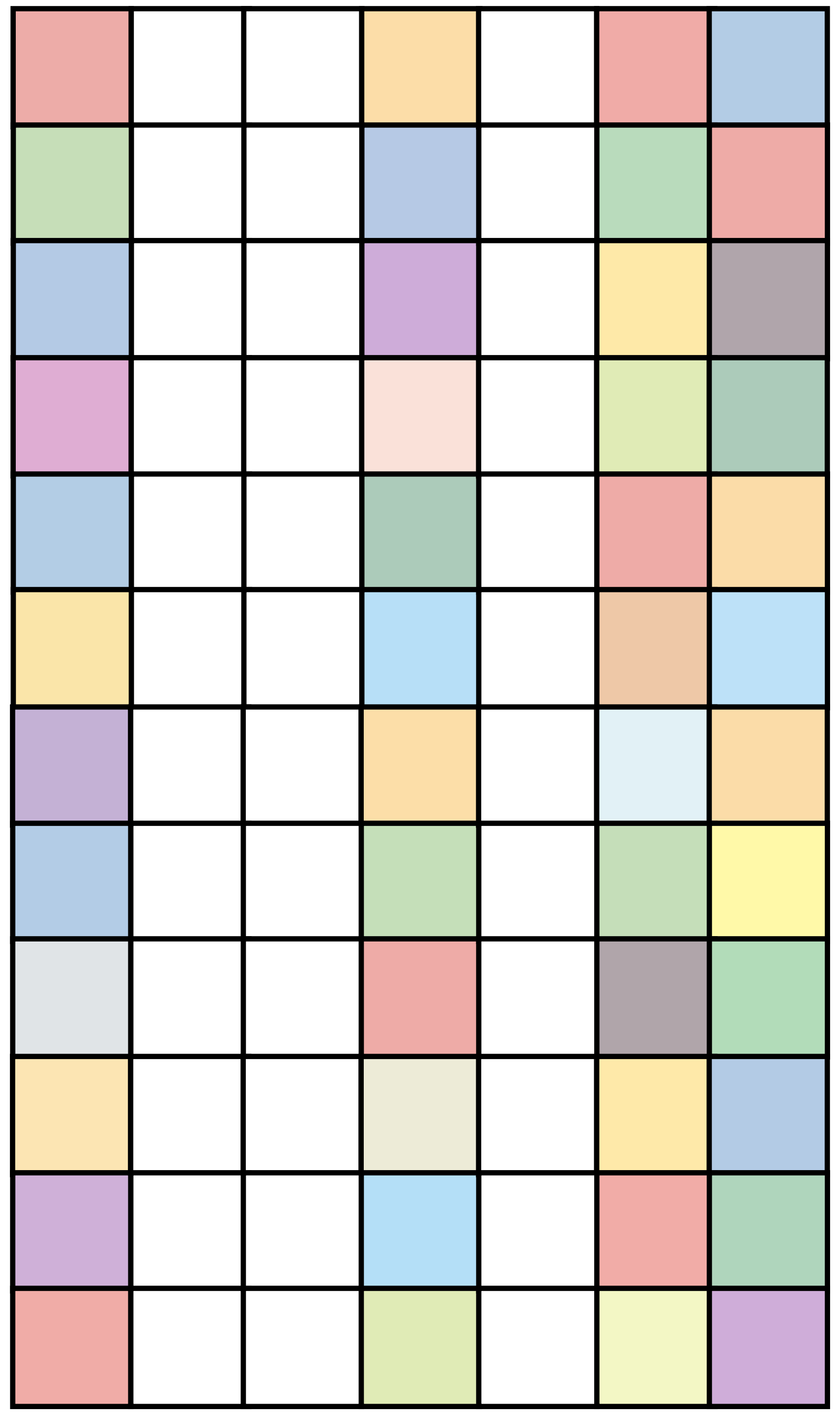
Weight matrix W

A lot of weights
set to zero

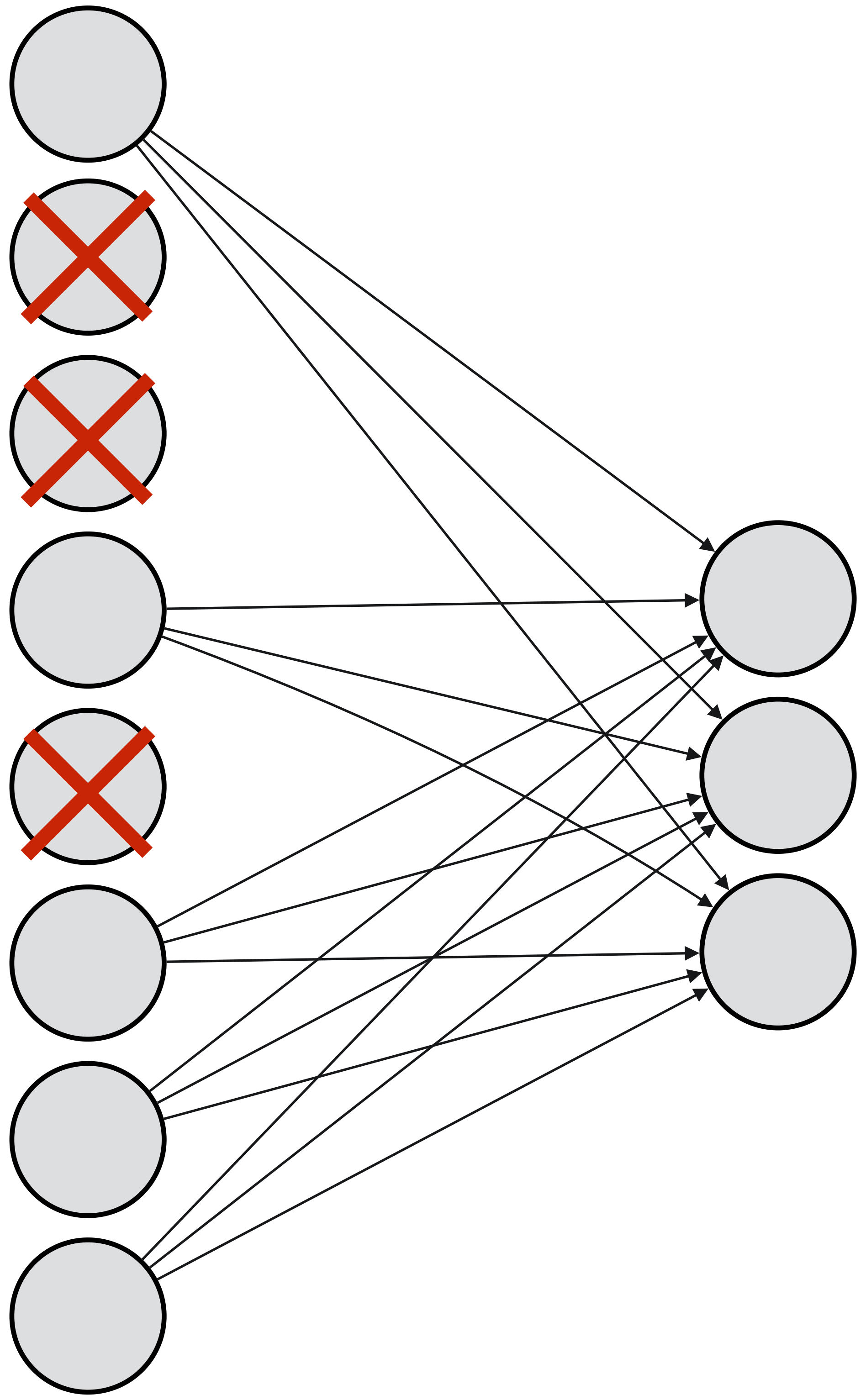


Computational graph

# Structured sparsity



No outgoing edges
⇒ remove neuron



Weight matrix W

Computational graph

# Sparsification of neural networks

**Benefits**:

- sparse matrices ⇒ less memory consumption
- structured sparsification ⇒ faster testing stage (prediction)

**Drawbacks:**

- Sometimes leads to small quality drop

**Applications**:

- Mobile devices, smartphones
- Online services (where fast reply is needed)

# Pruning

- Popular approach to NN sparsification
- Example training algorithm:
  - Optimize $L_1$-regularized loss (induces sparsity)
  - At the beginning of each epoch, set to 0 all weights satisfying $|w| < T$.

Wei Wen et al. Learning Intrinsic Sparse Structures within Long Short-Term Memory. ICLR 2018

# Pruning

- Popular approach to NN sparsification
- Example training algorithm:
  - Optimize $L_1$-regularized loss
  - At the beginning of each epoch, set to 0 all weights satisfying $|w| < T$.
- Variants:
  - Different pruning schedule (increase T gradually / constant T)
  - Propagate gradients through zero weights or not
  - Prune from scratch / train - prune - retrain
  - Prune based on threshold T / prune percent of weights
  - …

huge number of hyperparameters

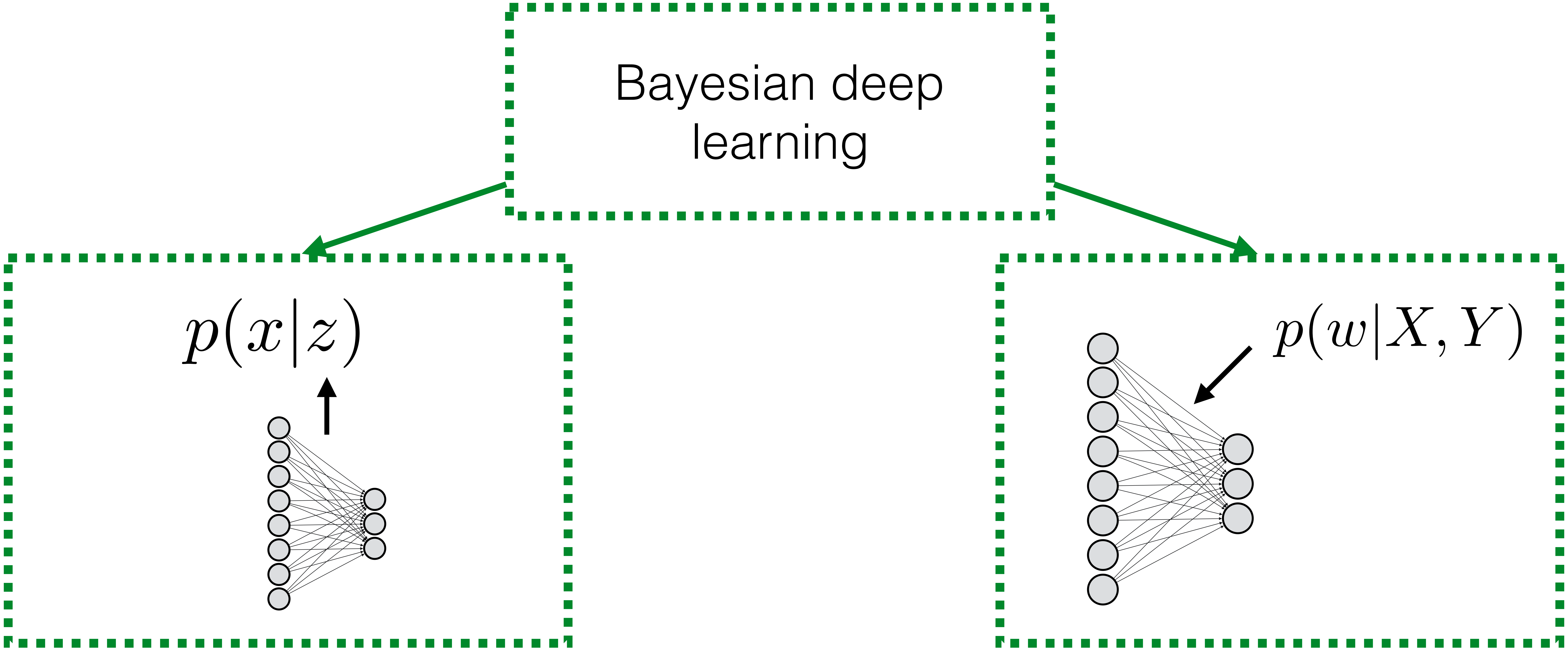
# Two popular frameworks for sparsification

| **Pruning** | **Bayesian sparsification** |
|---|---|
| A lot of method hyperparameters | (Almost) no method hyperparameters |
| Need to choose training schedule | Need to choose training schedule |
| non-Bayesian | Bayesian! |

# Agenda

- Sparsification: what and why
- Bayesian neural networks
- Sparse variational dropout
- Practical assignment: implementation of SparseVD
- Model enhancements

# Bayesian deep learning




Variational autoencoders, normalizing flows

Bayesian neural networks

# Regularization by noise

- Traditional (1943+) regularization: add some penalty for model complexity
  - $L_2$, $L_1$ - regularization, max norm constraint

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

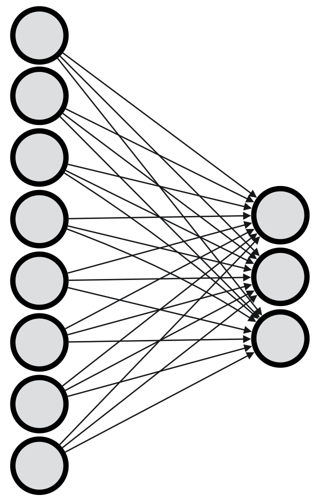
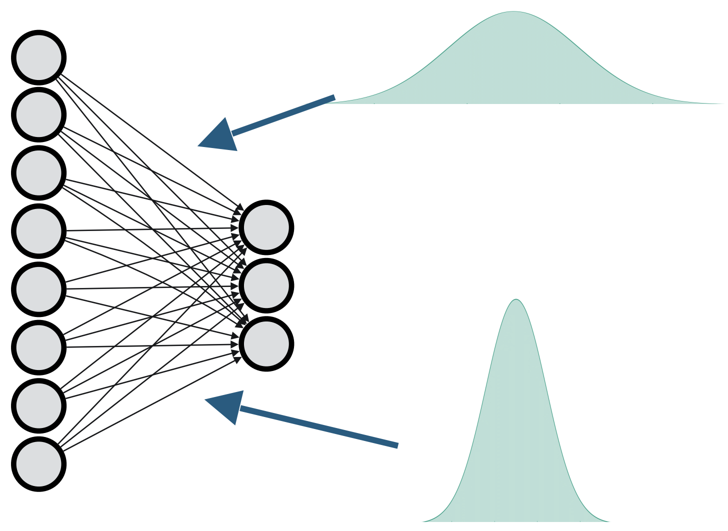
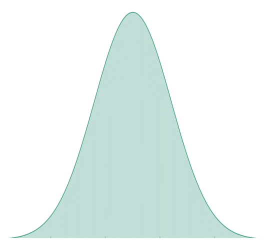
- More recent (1990+) approaches: regularization by noise
  - Data augmentation, dropout, gradient noise

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

Bayesian framework provides a principled approach to training with noise!

Slide From Dmitry Molchanov's presentation, deepbayes.ru/2018

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

# 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

# Ensembling

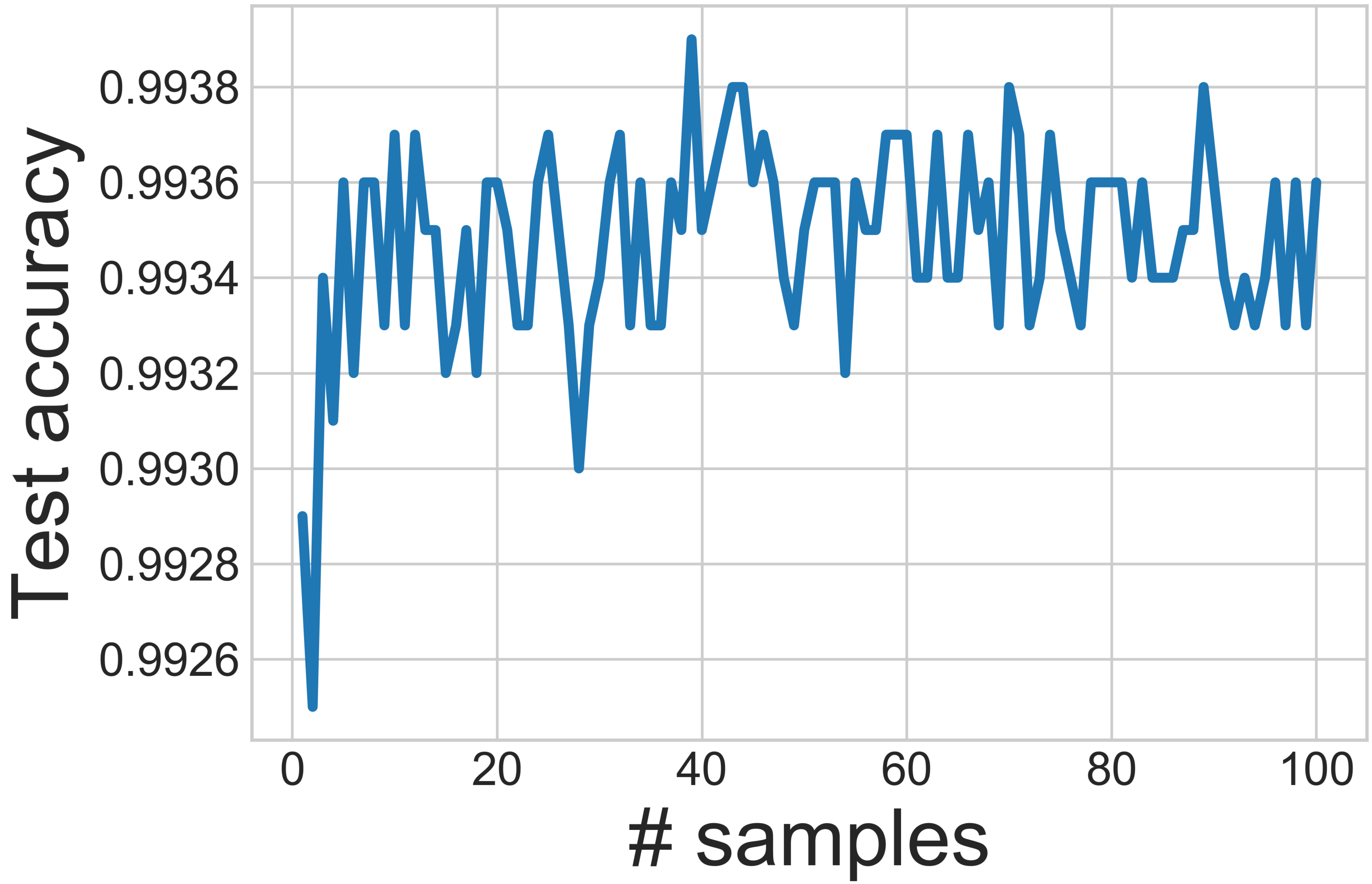
A Bayesian neural network is **an infinite ensemble** of neural networks

$$p(y_*|x_*, X, Y) = \int p(y_*|x_*, w)p(w|X, Y)dw \approx \frac{1}{K}\sum_{i=1}^{K} p(y_*|x_*, w^k)$$

$$w^k \sim p(w|X, Y)$$

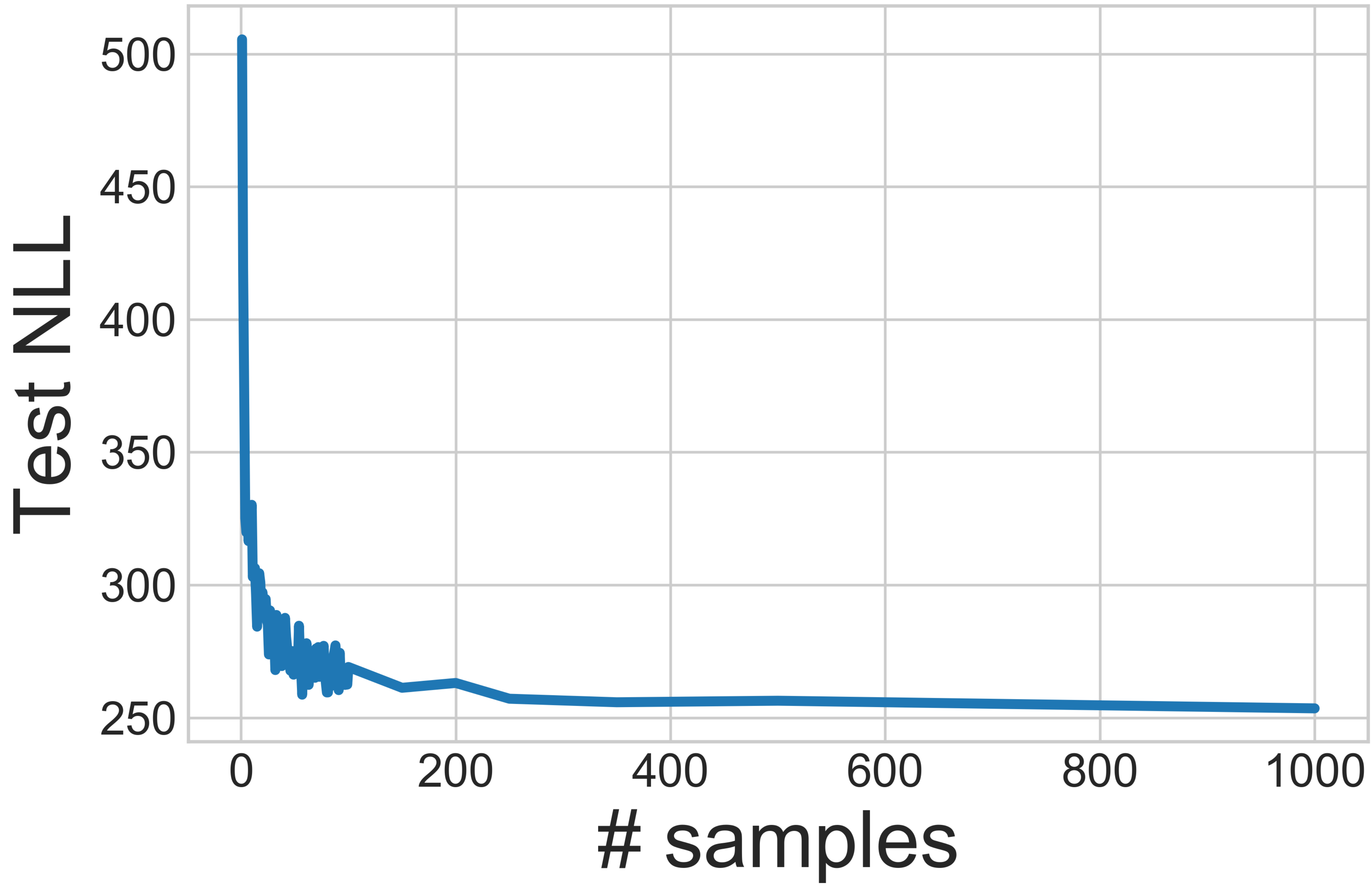
Average SoftMax outputs
across several samples

# Ensembling




Accuracy quickly saturates

# Ensembling



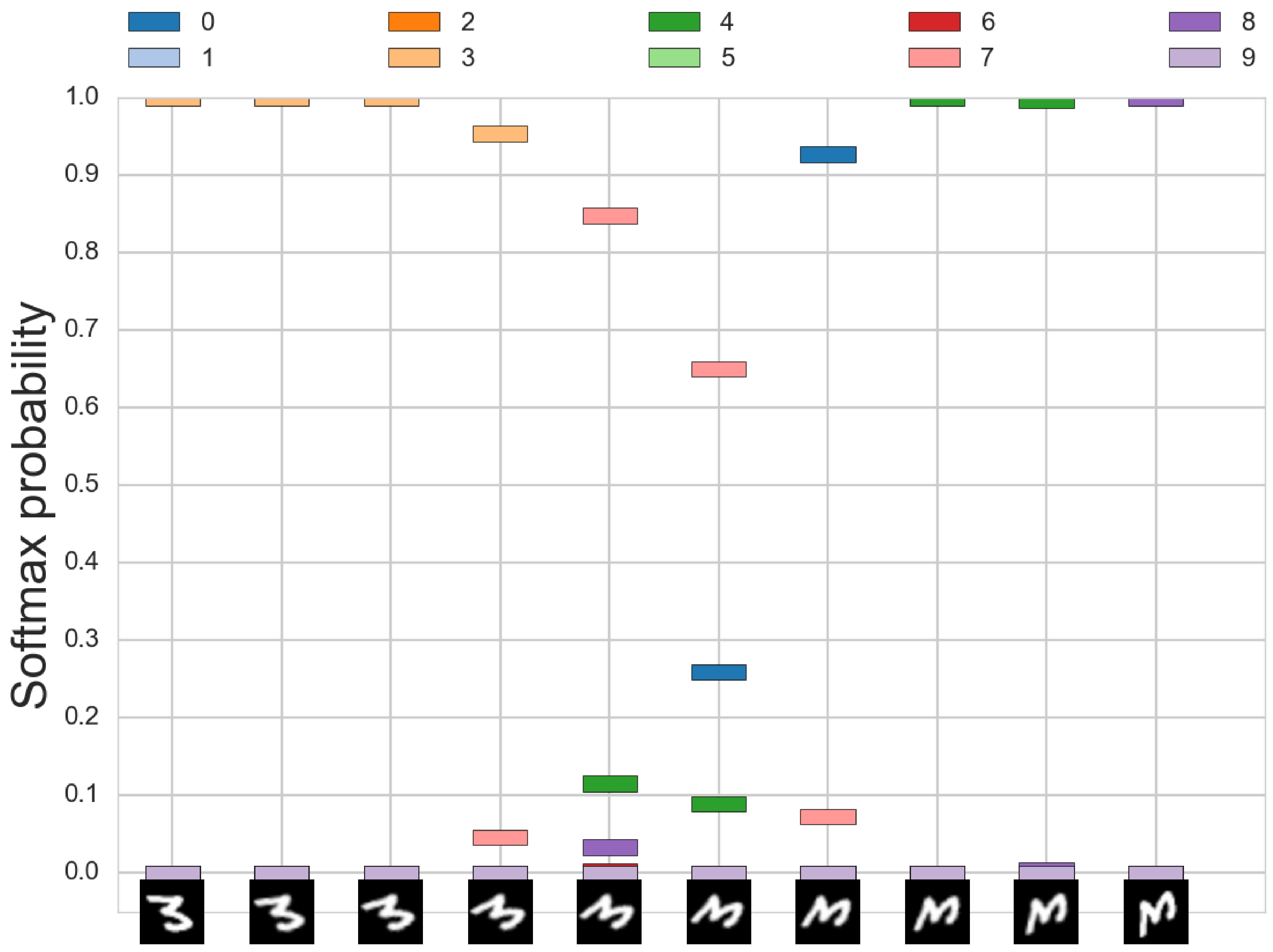

But the negative log—likelihood keeps improving! This is a measure of “uncertainty”

Slide From Dmitry Molchanov’s presentation, deepbayes.ru/2018

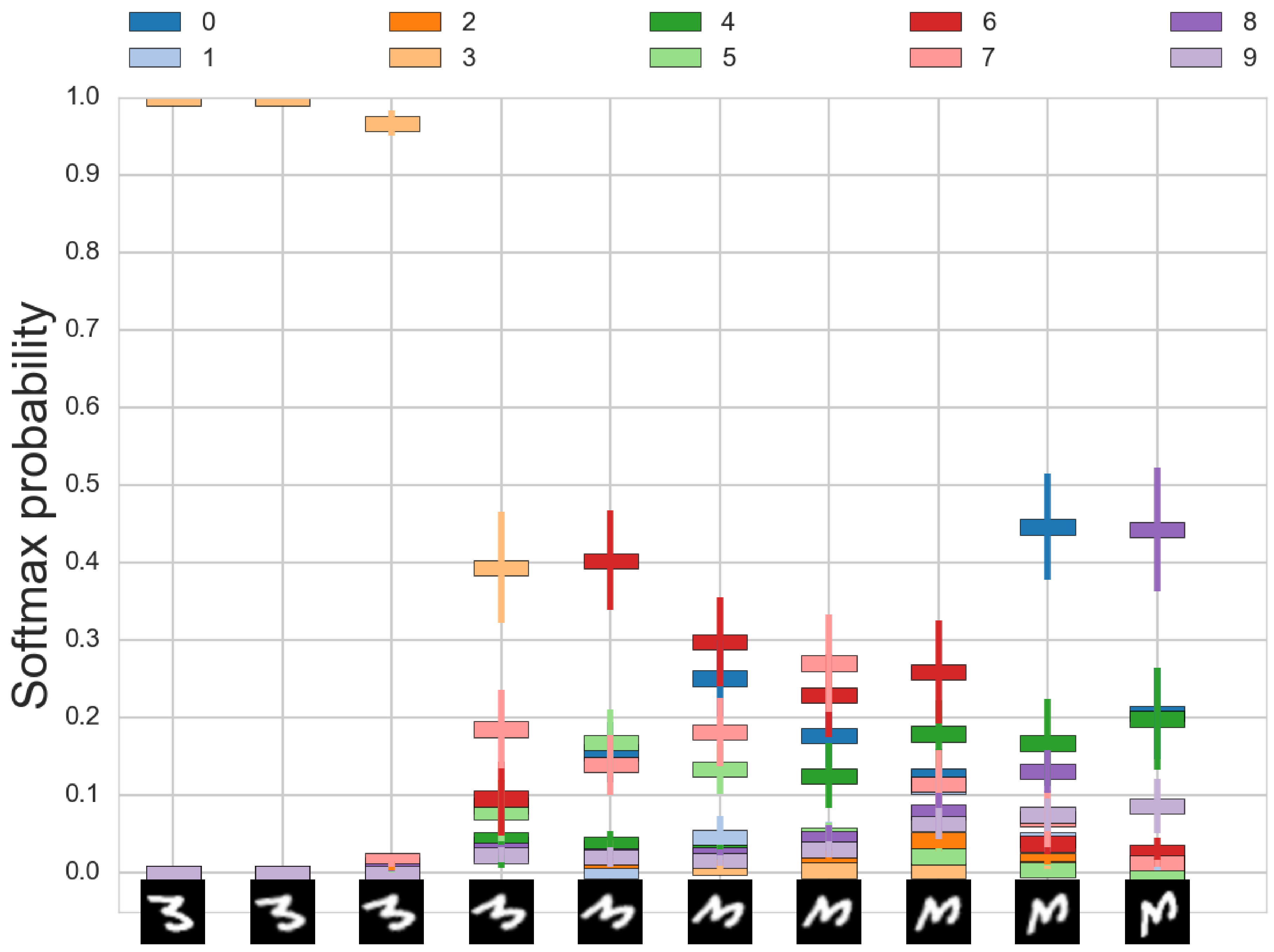
# Uncertainty estimation

Deterministic NNs: a point estimate of the output, overconfident

Bayesian framework: a distribution over the outputs




(a) LeNet with weight decay




(b) LeNet with multiplicative formalizing flows

Louizos, Christos, and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. ICML 2017

# Model selection

- Empirical Bayes (maximum evidence)
  - Optimize w. r. t. prior parameters ⇒ Choose hyperparameters
- Discuss later

# Online / incremental learning

Assume that dataset arrives gradually in independent parts:

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3 \cup \cdots \cup \mathcal{D}_M$$

We can first train on the first dataset as usual:

$$p(w|\mathcal{D}_1) = \frac{p(\mathcal{D}_1|w)p(w)}{\int p(\mathcal{D}_1|\tilde{w})p(\tilde{w})d\tilde{w}}$$

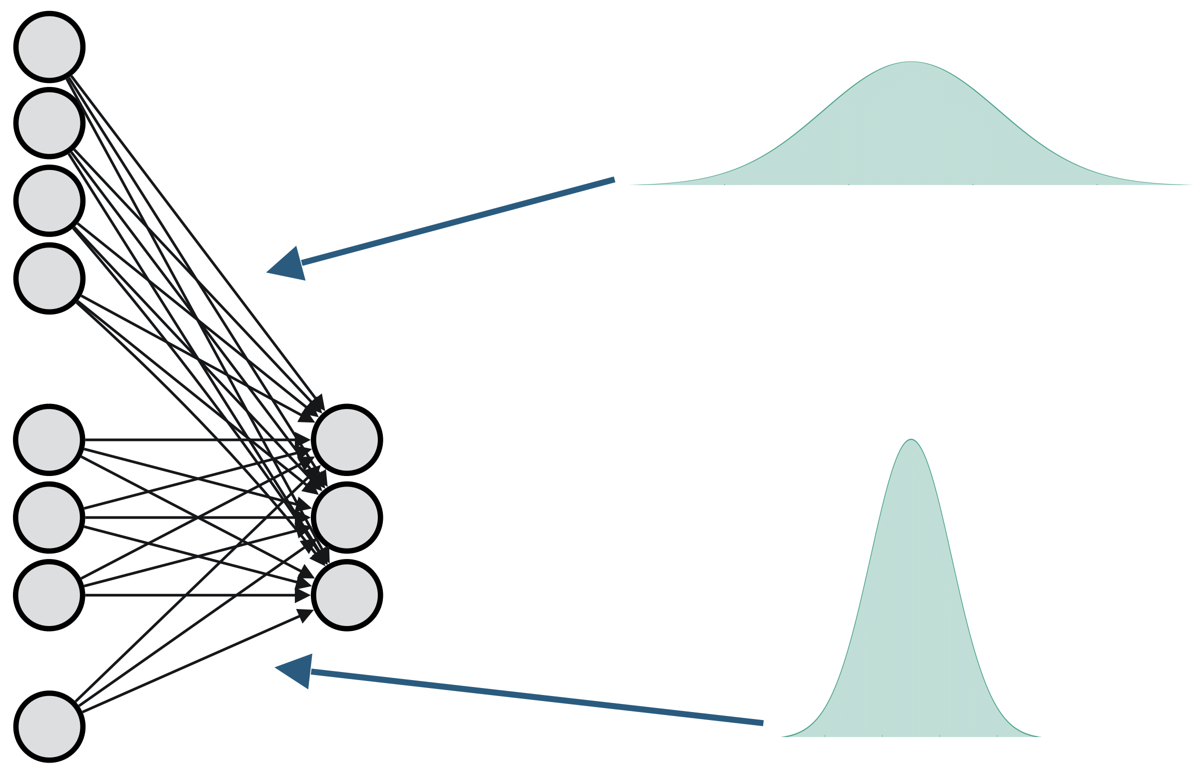
And then use the obtained posterior as the prior for the next step!

$$p(w|\mathcal{D}_2, \mathcal{D}_1) = \frac{p(\mathcal{D}_2|w)\color{blue}{p(\mathcal{D}_1|w)p(w)}}{\int p(\mathcal{D}_2|\tilde{w})\color{blue}{p(\mathcal{D}_1|\tilde{w})p(\tilde{w})}d\tilde{w}}$$

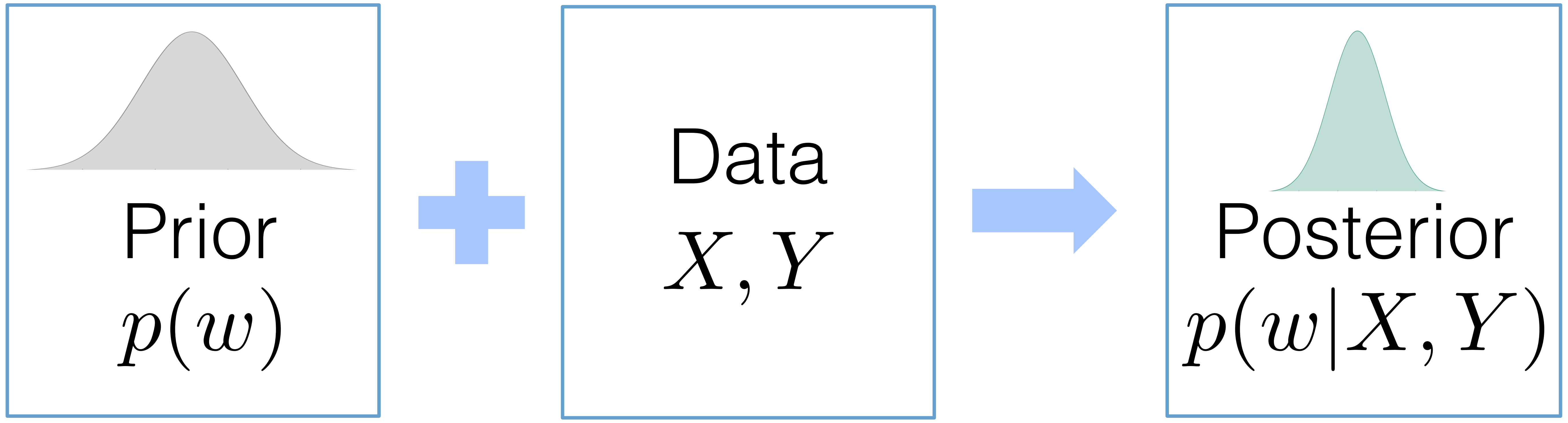
Using these sequential updates, we can find $p(w|\mathcal{D})$.

# Training Bayesian neural networks

Stochastic weights:



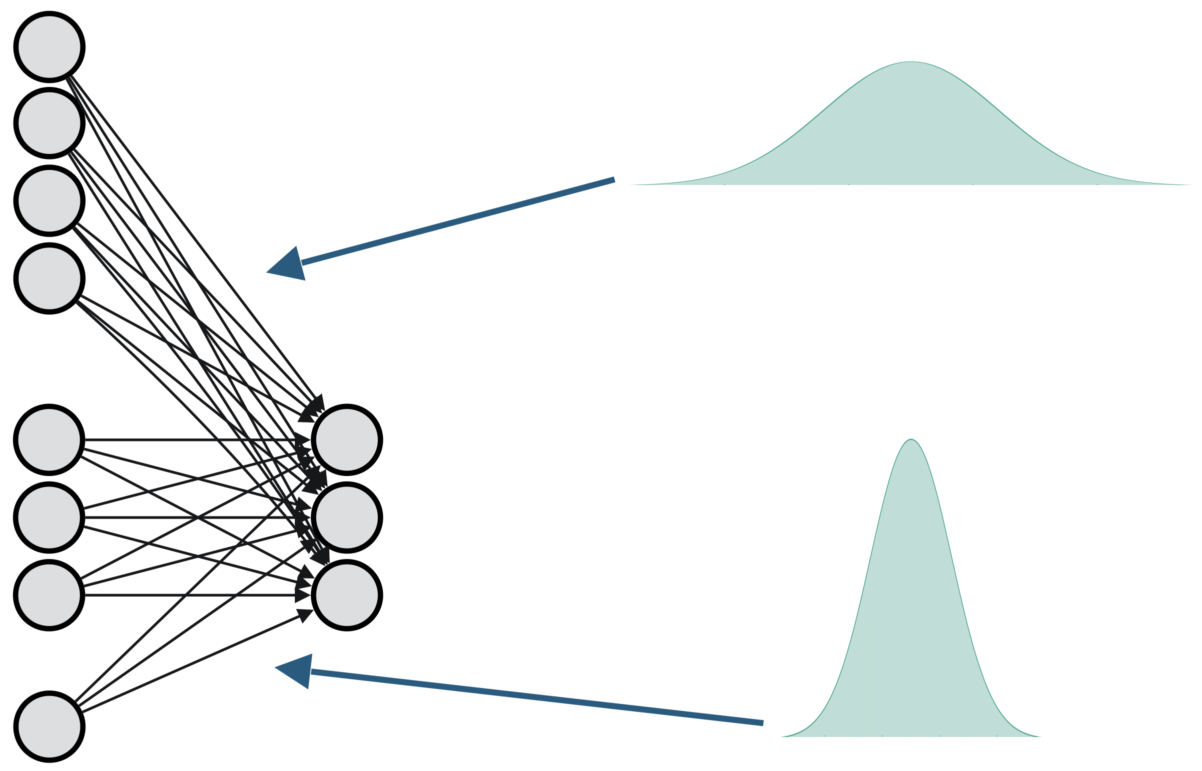
Bayesian Inference:



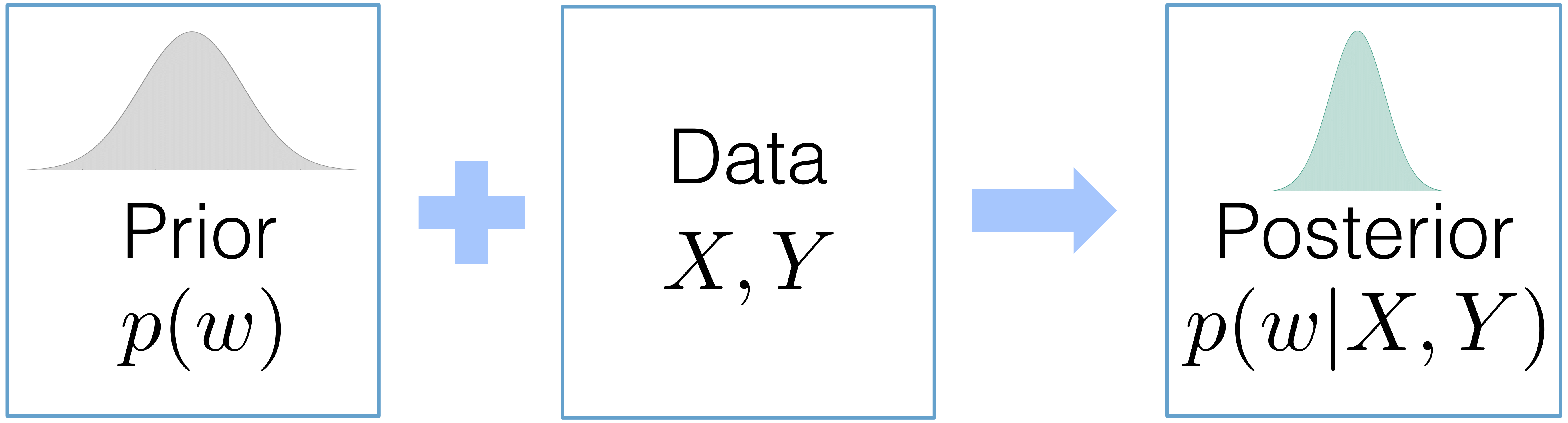

$$p(w|X,Y) = \frac{p(Y|X,w)p(w)}{\int p(Y|X,\tilde{w})p(\tilde{w})d\tilde{w}}$$

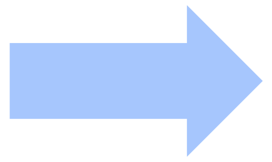
# Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:

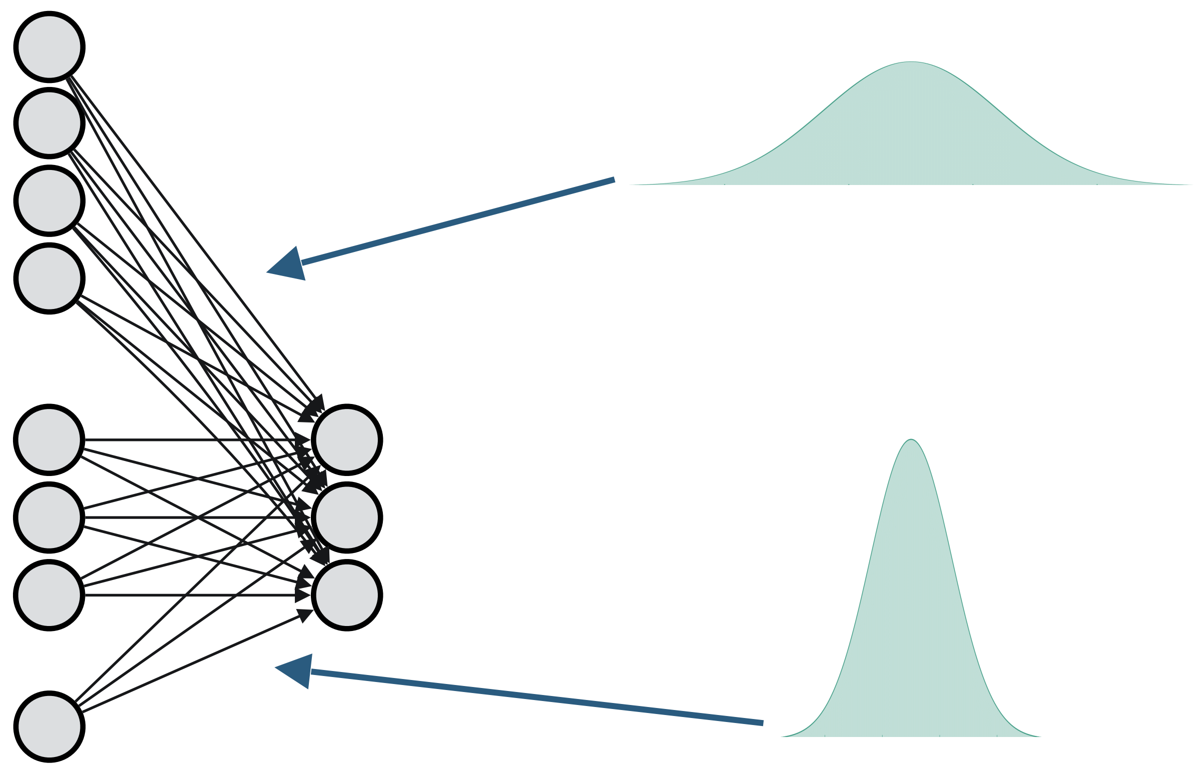



Posterior is intractable in neural networks  approximate it with $q(w|\lambda)$:

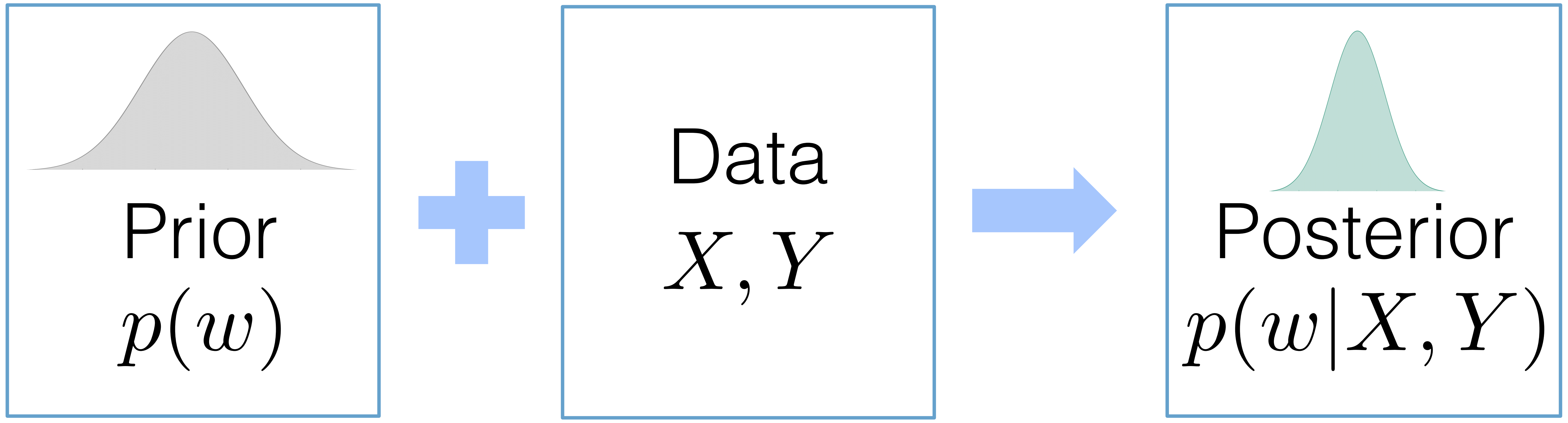
$$KL(q(w|\lambda)||p(w|X,y)) \to \min_{\lambda}$$

# Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:




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

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

# Training Bayesian neural networks

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

Variational autoencoders are also trained using variational inference.

Compared to them:

- KL-term is global
- Extremely high-dimensional approximate posterior

# From general framework to particular method

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

Model specification:

- Choose particular prior

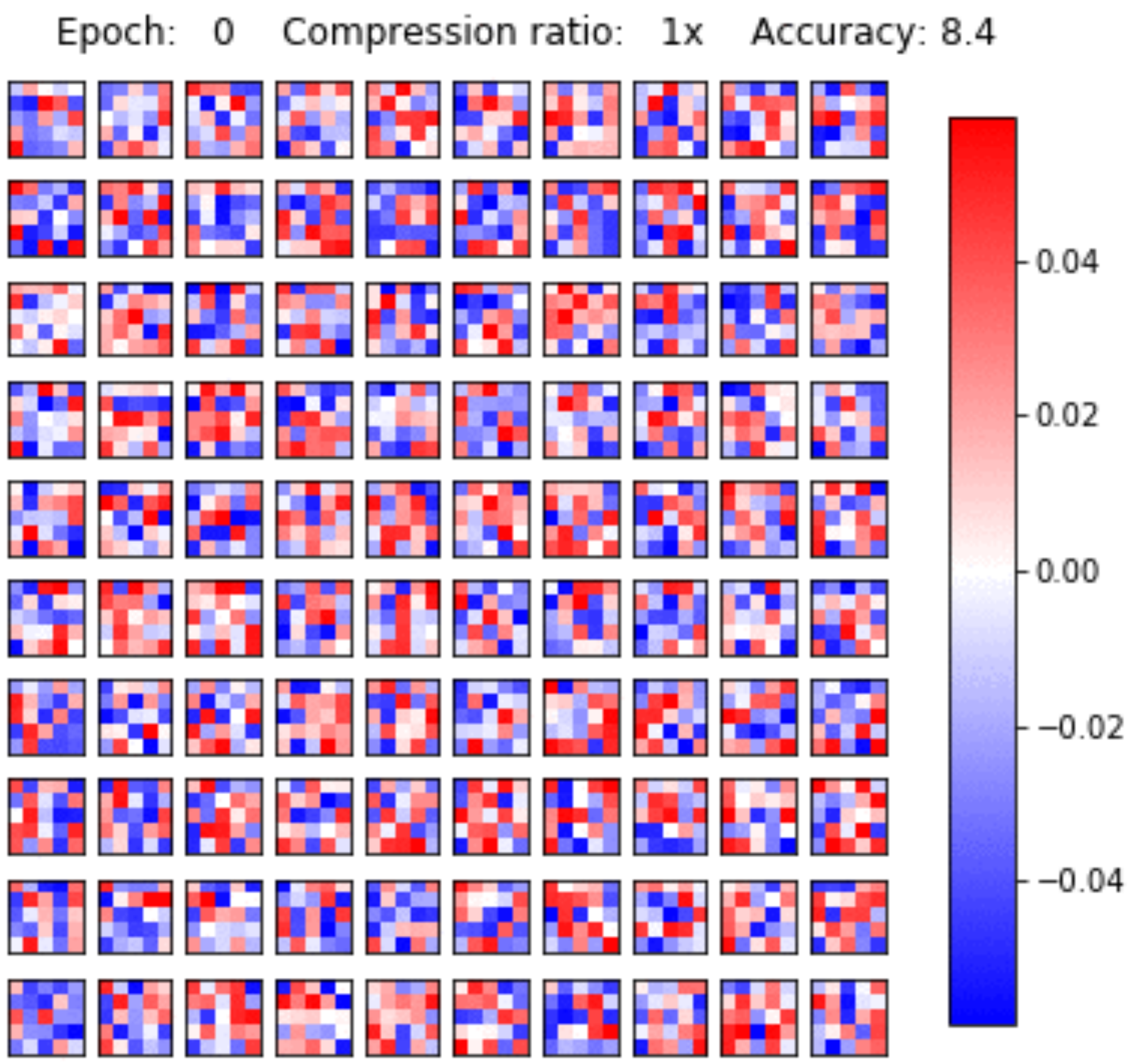
Training:

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

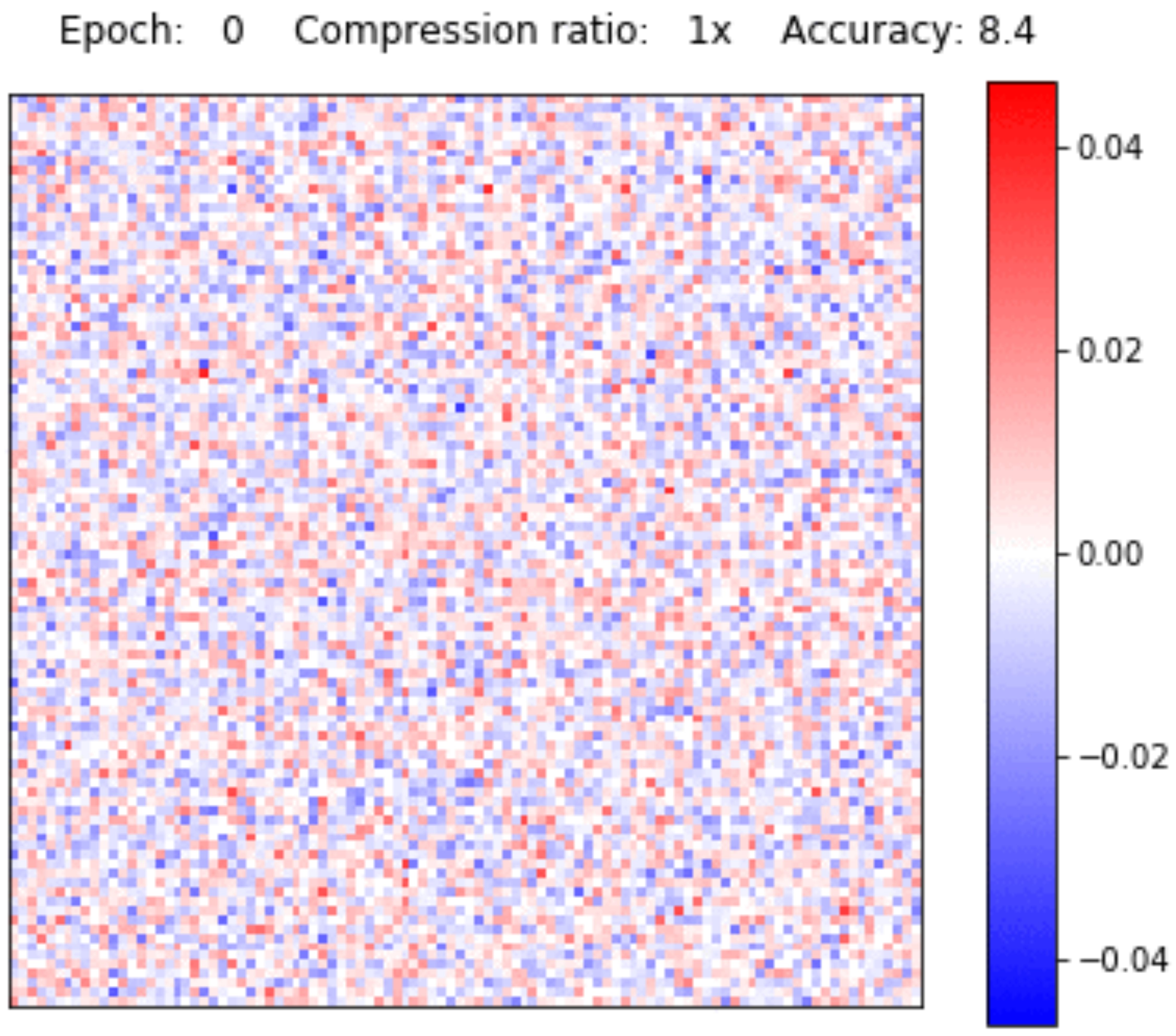
# Agenda

- Sparsification: what and why
- Bayesian neural networks
- $\boxed{\text{Sparse variational dropout}}$
- Practical assignment: implementation of SparseVD
- Model enhancements

# Sparse variational dropout: visualization

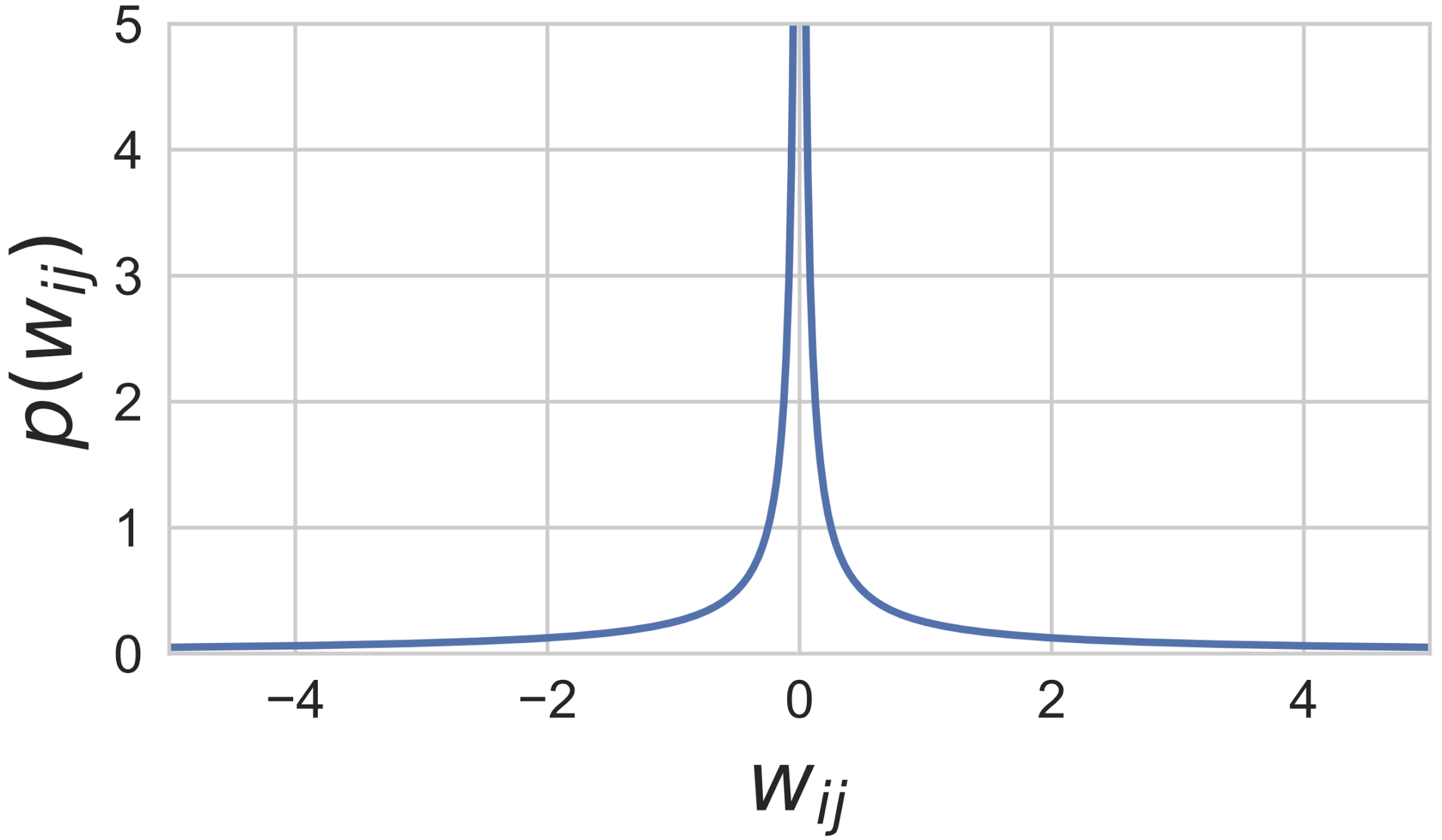



LeNet-5: convolutional layer




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

# Log-uniform prior distribution

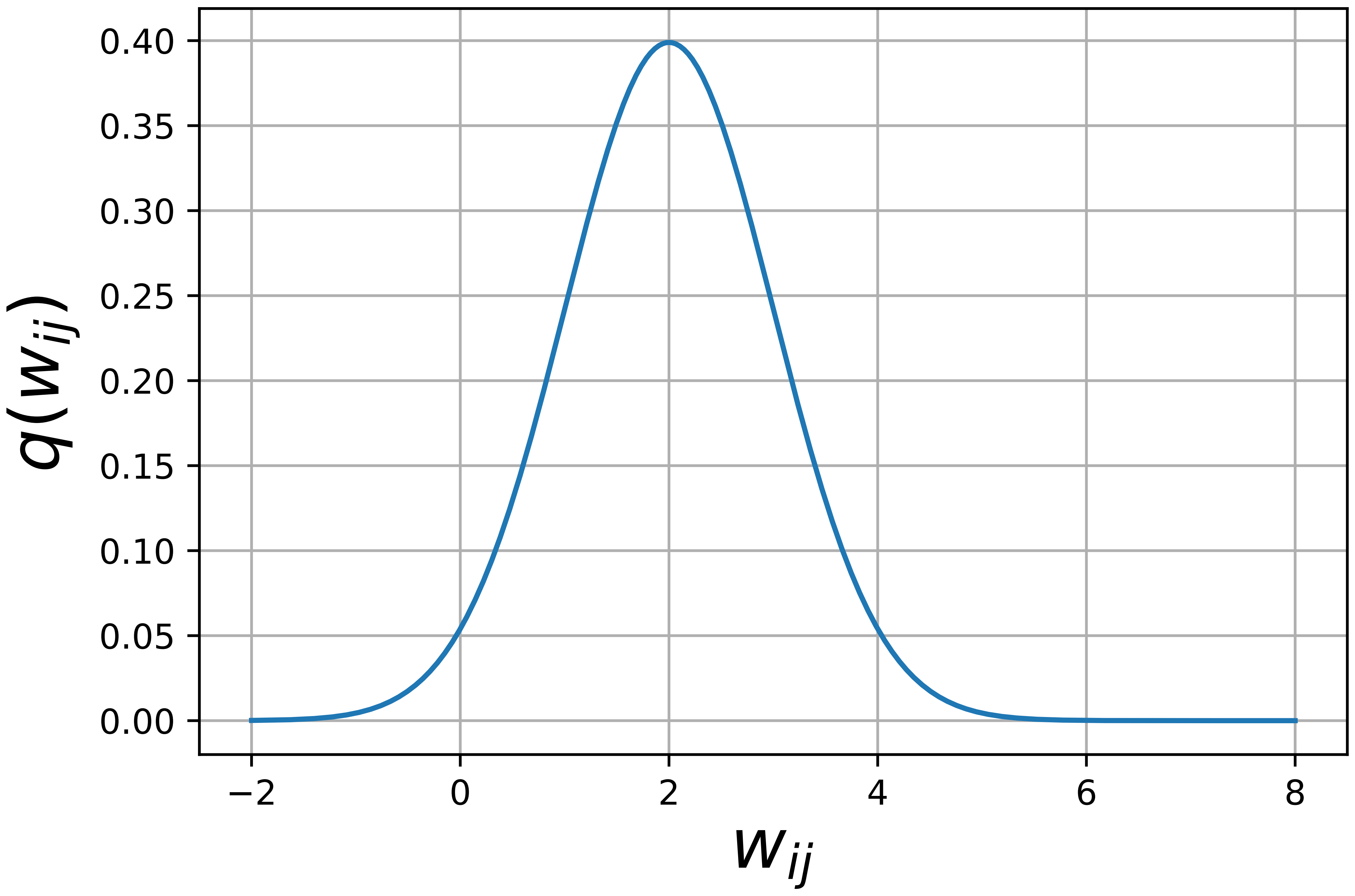



$$p(w_{ij}) \propto \frac{1}{|w_{ij}|}$$

Favors removing noisy weights
(few slides later)

Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# Normal approximate posterior distribution
## (fully factorized)




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

Reparametrization trick:

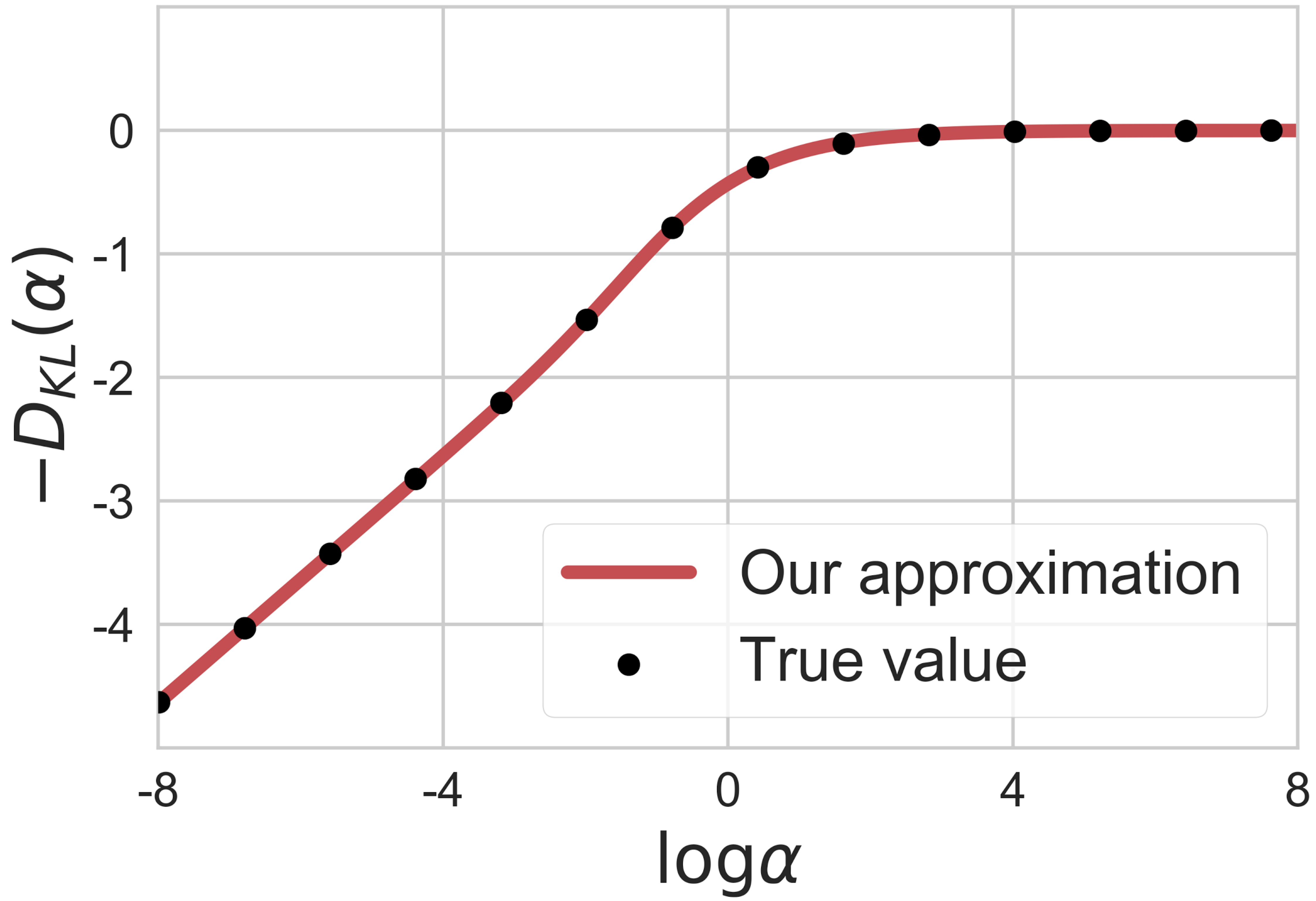
$$\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}, \quad \hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$$

weight mean

zero-centered noise with learnable variance

Molchanov, Dmitry et al. Variational dropout sparsifies deep neural networks. ICML 2017

# Approximating KL-divergence
(fully factorized)




$$-KL(q(w_{ij}|\mu_{ij},\sigma_{ij}) \,\|\, p(w_{ij})) \approx$$
$$\approx k_1 \sigma(k_2 + k_3 \log \alpha_{ij})) - 0.5 \log(1 + \alpha_{ij}^{-1}) + \mathrm{C}$$
$$k_1 = 0.63576 \qquad k_2 = 1.87320 \qquad k_3 = 1.48695$$
$$\alpha_{ij} = \frac{\sigma_{ij}^2}{\mu_{ij}^2}$$

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

Molchanov, Dmitry et al. Variational dropout sparsifies deep neural networks. ICML 2017

# Estimating expectation in ELBO

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

# Estimating expectation in ELBO

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(w|\mu,\sigma)} \log p(y^i|x^i, w)}_{\substack{\text{Data term} \\ \text{Sample weights from } q}} - \underbrace{KL(q(w|\mu, \sigma)||p(w))}_{\substack{\text{Regularizer} \\ \text{Analytical approximation}}} \to \max_{\mu, \log \sigma}$$

# Estimating expectation in ELBO

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(w|\mu,\sigma)} \log p(y^i|x^i, w)}_{\substack{\text{Data term} \\ \text{Sample weights from } q}} - \underbrace{KL(q(w|\mu,\sigma)||p(w))}_{\substack{\text{Regularizer} \\ \text{Analytical approximation}}} \to \max_{\mu, \log\sigma}$$

1. Reparametrization trick ⇒ unbiased estimate of the gradients

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

# Estimating expectation in ELBO

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(w|\mu,\sigma)} \log p(y^i|x^i, w)}_{\substack{\text{Data term} \\ \text{Sample weights from } q}} - \underbrace{KL(q(w|\mu,\sigma)||p(w))}_{\substack{\text{Regularizer} \\ \text{Analytical approximation}}} \to \max_{\mu, \log\sigma}$$

1. Reparametrization trick ⇒ unbiased estimate of the gradients

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

- Too expensive! ( $|w| \times N$ samples)

# Estimating expectation in ELBO

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(w|\mu,\sigma)} \log p(y^i|x^i, w)}_{\substack{\text{Data term} \\ \text{Sample weights from } q}} - \underbrace{KL(q(w|\mu,\sigma)||p(w))}_{\substack{\text{Regularizer} \\ \text{Analytical approximation}}} \rightarrow \max_{\mu, \log\sigma}$$

1. Reparametrization trick ⇒ unbiased estimate of the gradients

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

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

# Estimating expectation in ELBO

$$\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(w|\mu,\sigma)} \log p(y^i|x^i, w)}_{\substack{\text{Data term} \\ \text{Sample weights from } q}} - \underbrace{KL(q(w|\mu,\sigma)||p(w))}_{\substack{\text{Regularizer} \\ \text{Analytical approximation}}} \to \max_{\mu, \log\sigma}$$

1. Reparametrization trick ⇒ unbiased estimate of the gradients

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

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

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

$$\mathbb{E}B = A\mu \quad VarB = A^2\sigma^2$$
$$B \sim \mathcal{N}(A\mu, A^2\sigma^2)$$
$$B = A\mu + \sqrt{A^2\sigma^2} \odot \epsilon$$

blue means element-wise

# The local reparametrization trick

Consider a linear layer with weight matrix $W$, input $A$ and output $B$:

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

Predictions have **high**
correlation because there is
one weight sample **per mini-batch**

Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# The local reparametrization trick

Consider a linear layer with weight matrix $W$, input $A$ and output $B$:

$$w_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma^2_{ij})$$
$$B = AW$$

Predictions have **high** correlation because there is one weight sample **per mini-batch**

$$\mathbb{E}B = A\mu \quad VarB = A^2\sigma^2$$
$$B \sim \mathcal{N}(A\mu, A^2\sigma^2)$$
$$B = A\mu + \sqrt{A^2\sigma^2} \odot \epsilon$$

blue means element-wise

Predictions have **zero** correlation because there is one weight sample **per object**

Less correlation ⇒ lower variance of the stochastic gradients **for a mini-batch**

Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# The local reparametrization trick

The LRT also reduces the variance of the stochastic gradient w.r.t $\sigma_{i,j}$ for **an object.**

**Sketch of the proof:**

ELBO

$$\frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial \sigma_{i,j}^2} = \frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial b_{m,j}} \frac{\epsilon_{i,j} a_{m,i}}{2\sigma_{i,j}} \qquad \frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial \sigma_{i,j}^2} = \frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial b_{m,j}} \frac{\zeta_{m,j} a_{m,i}^2}{2\sqrt{\delta_{m,j}}}$$

$$\text{Var}_{q_\phi,\mathcal{D}}\left[\frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial \sigma_{i,j}^2}\right] = \text{Var}_{b_{m,j}}\left[\mathbb{E}_{q_\phi,\mathcal{D}}\left[\frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial \sigma_{i,j}^2}|b_{m,j}\right]\right] + \mathbb{E}_{b_{m,j}}\left[\text{Var}_{q_\phi,\mathcal{D}}\left[\frac{\partial L_{\mathcal{D}}^{\text{SGVB}}}{\partial \sigma_{i,j}^2}|b_{m,j}\right]\right]$$

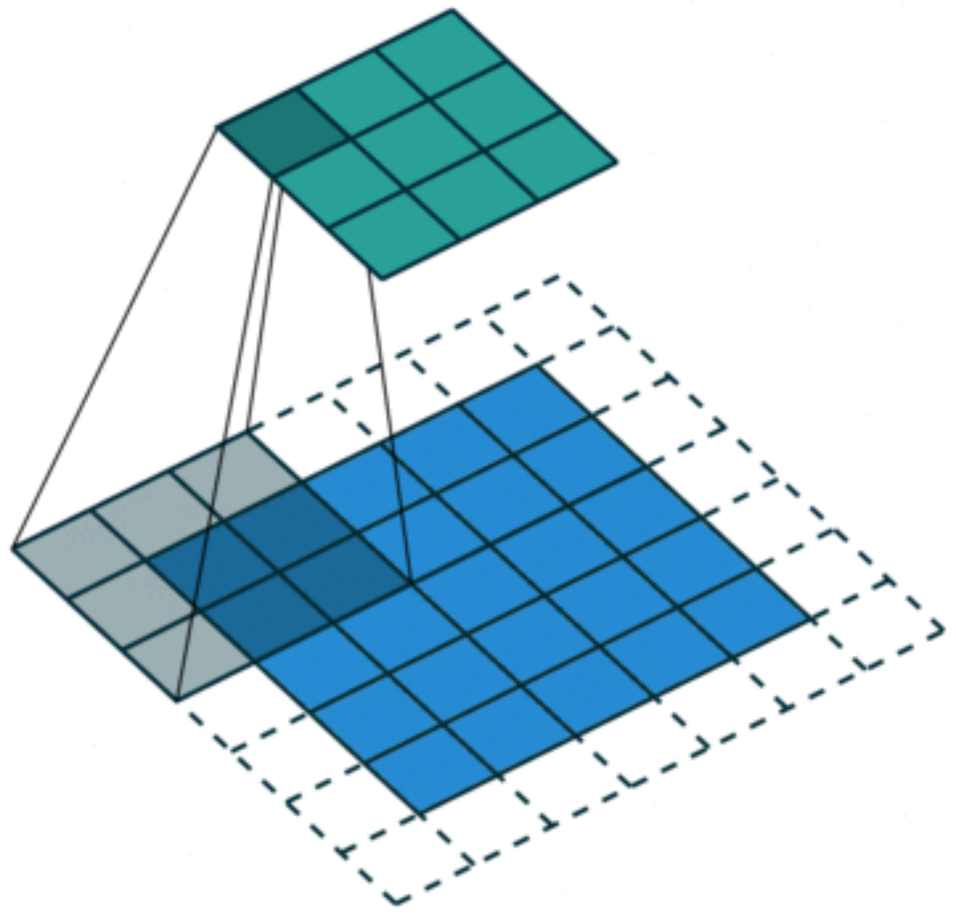
same for both parametrizations

**for LRT**: vanishes because $b_{m,j} = \ldots + \ldots \cdot \zeta_{m,j}$
and $\zeta_{m,j}$ is uniquely determined given $b_{m,j}$
**for RT**: doesn't vanish (and positive)

Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# LRT for convolutions

- $B$ no longer factorizes in convolutional layers
  - Same weights sample should be used for different spatial positions

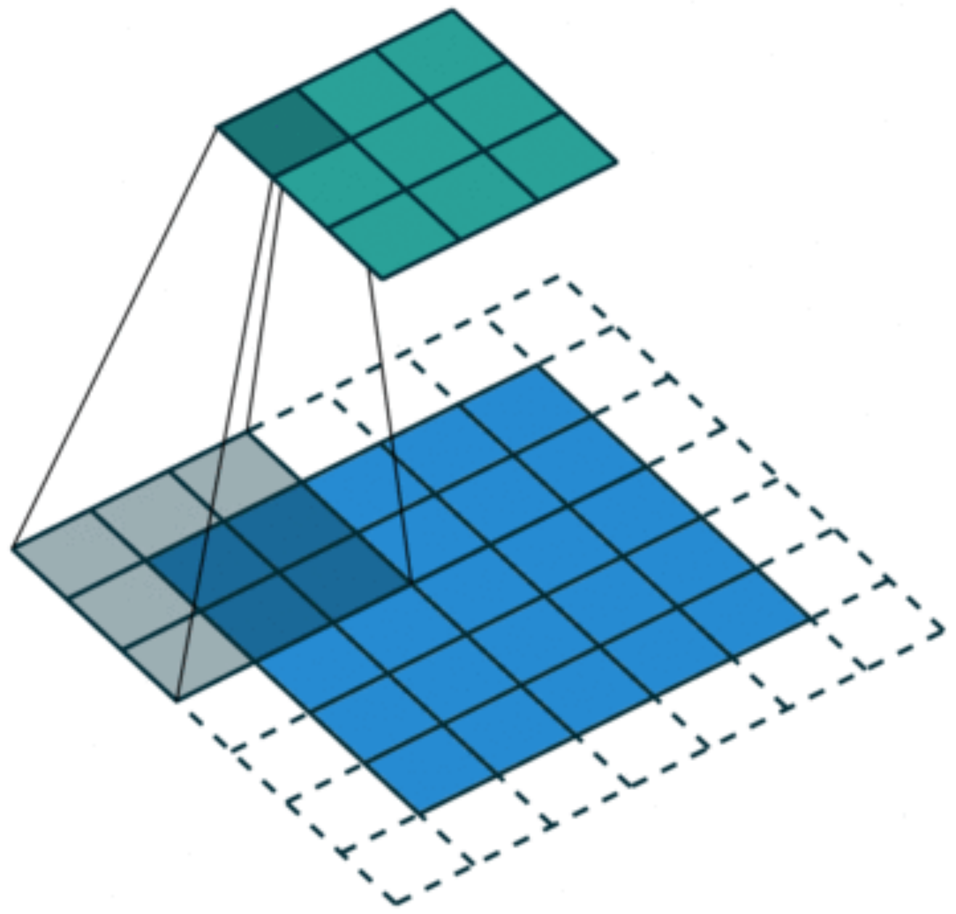


Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# LRT for convolutions

- $B$ no longer factorizes in convolutional layers
  - Same weights sample should be used for different spatial positions
- Exact local reparametrization is too complex
  - Preactivations are normal but correlated

    $\Rightarrow$ we need to estimate the full covariance matrix for each preactivation

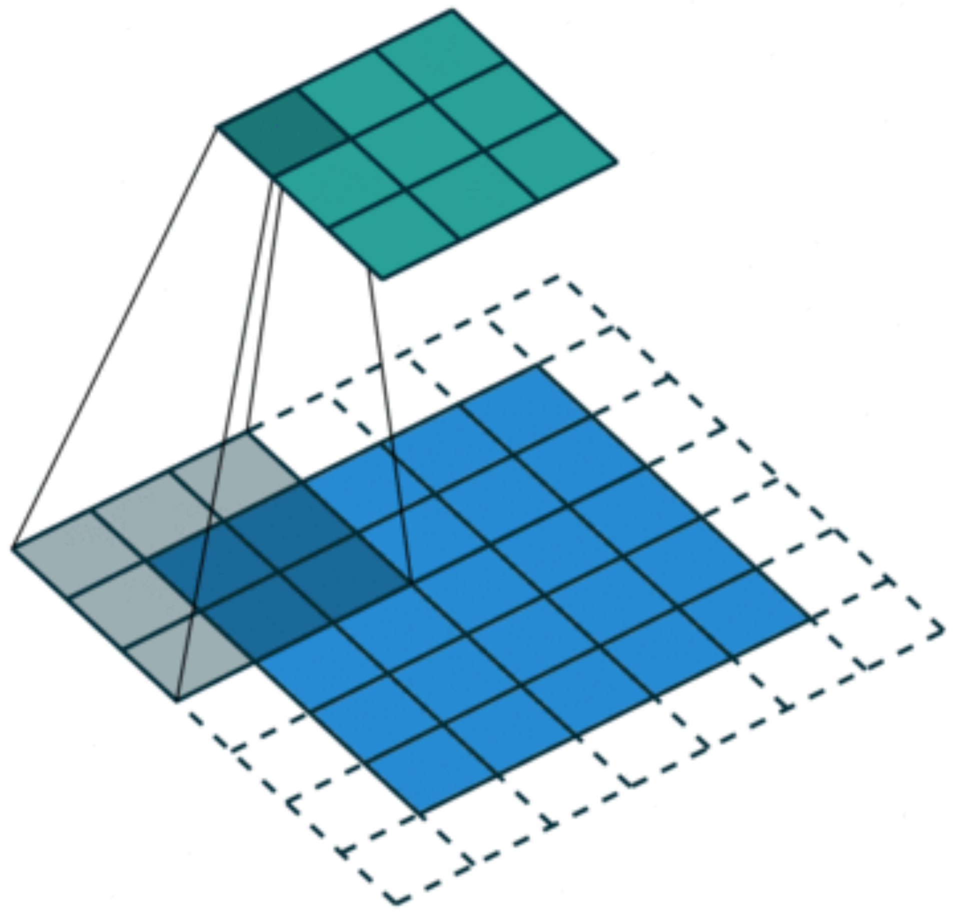


Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# LRT for convolutions

- $B$ no longer factorizes in convolutional layers
  - Same weights sample should be used for different spatial positions
- Exact local reparametrization is too complex
  - Preactivations are normal but correlated

    $\Rightarrow$ we need to estimate the full covariance matrix for each preactivation



- Mean-field approximation performs much better that sampling weights:

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

Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

# From general framework to particular method

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

Model specification:

- Choose particular prior ✓ log-uniform distribution

Training:

- Choose particular family for approximate posterior ✓ normal distribution
- How to compute KL-divergence? ✓ analytical approximation
- How to estimate the expectation? ✓ RT & LRT

# From general framework to particular method

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

What about biases? batch-norm parameters?

# From general framework to particular method

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

What about biases? batch-norm parameters?

Treat them as deterministic parameters and find point estimate:

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

Eventually, it assumes a flat prior and a delta-peak posterior.

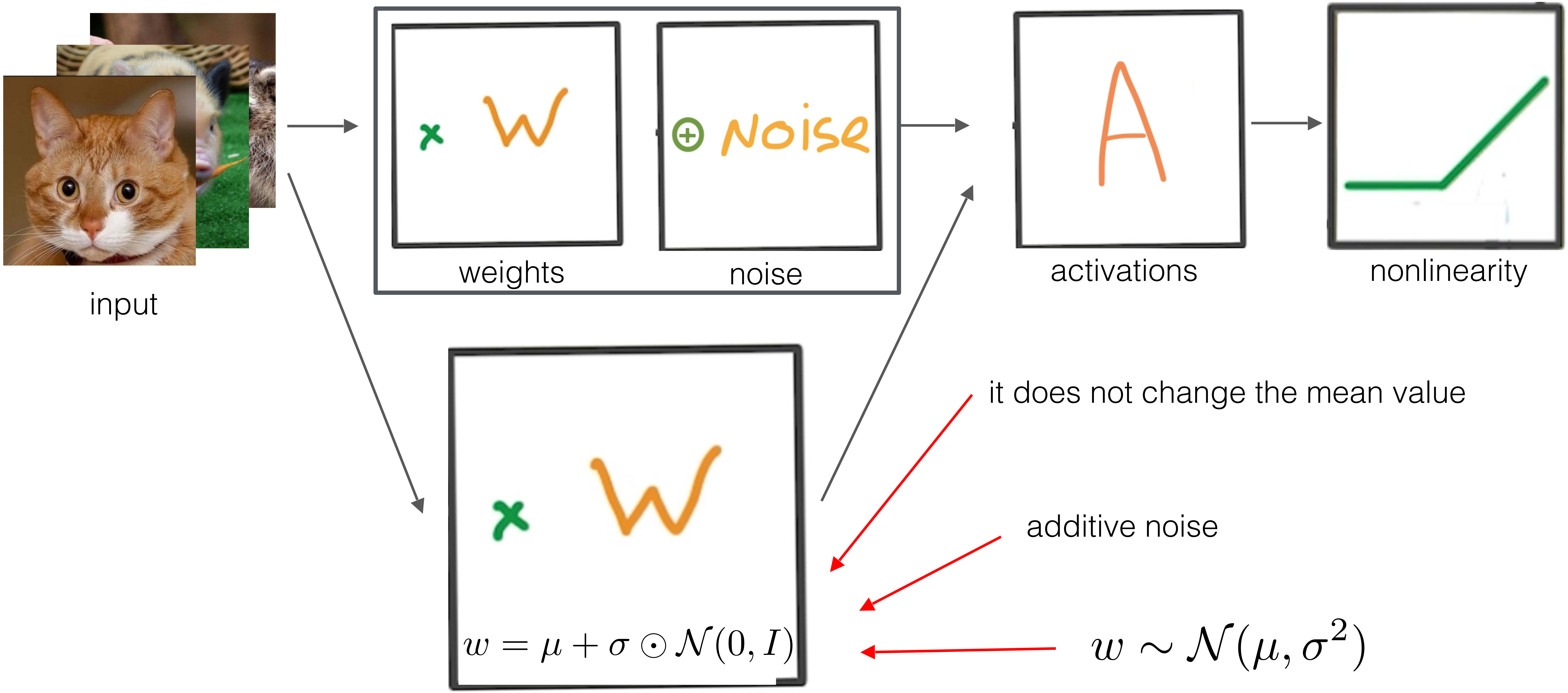
# Final algorithm (without LRT)

Training on a mini-batch $X$ with labels $Y$:

1. Sample weights: $\hat{w} = \mu + \sigma \odot \epsilon, \ \epsilon \sim \mathcal{N}(0, I)$
2. Forward pass: $Y_{\text{pred}} = NN(X, \hat{w}, b)$
3. Backward pass: compute stochastic gradients of ELBO:

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

# Forward pass with stochastic weights



# Final algorithm (without LRT)

Training on a mini-batch $X$ with labels $Y$:

1. Sample weights: $\hat{w} = \mu + \sigma \odot \epsilon,\ \epsilon \sim \mathcal{N}(0, I)$
2. Forward pass: $Y_{\text{pred}} = NN(X, \hat{w}, b)$
3. Backward pass: compute stochastic gradients of ELBO:

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

# Final algorithm (with LRT)

Training on a mini-batch $X$ with labels $Y$:

1. Sample noise $\epsilon \sim \mathcal{N}(0, I)$
2. Forward pass with LRT: $Y_{\text{pred}} = NN(X, \mu, \sigma, \epsilon, b)$
3. Backward pass: compute stochastic gradients of ELBO:

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

# Final algorithm (with LRT)

**Training on a mini-batch $X$ with labels $Y$:**

1. Sample noise $\epsilon \sim \mathcal{N}(0, I)$
2. Forward pass with LRT: $Y_{\text{pred}} = NN(X, \mu, \sigma, \epsilon, b)$
3. Backward pass: compute stochastic gradients of ELBO:

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

**Pruning after training:**

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

$$\theta_{ij} = 0, \;\; \sigma_{ij} = 0$$

signal-to-noise ratio

# Final algorithm (with LRT)

Training on a mini-batch $X$ with labels $Y$:

1. Sample noise $\epsilon \sim \mathcal{N}(0, I)$
2. Forward pass with LRT: $Y_{\text{pred}} = NN(X, \mu, \sigma, \epsilon, b)$
3. Backward pass: 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 $\mu_{ij}^2 / \sigma_{ij}^2 <$ threshold:

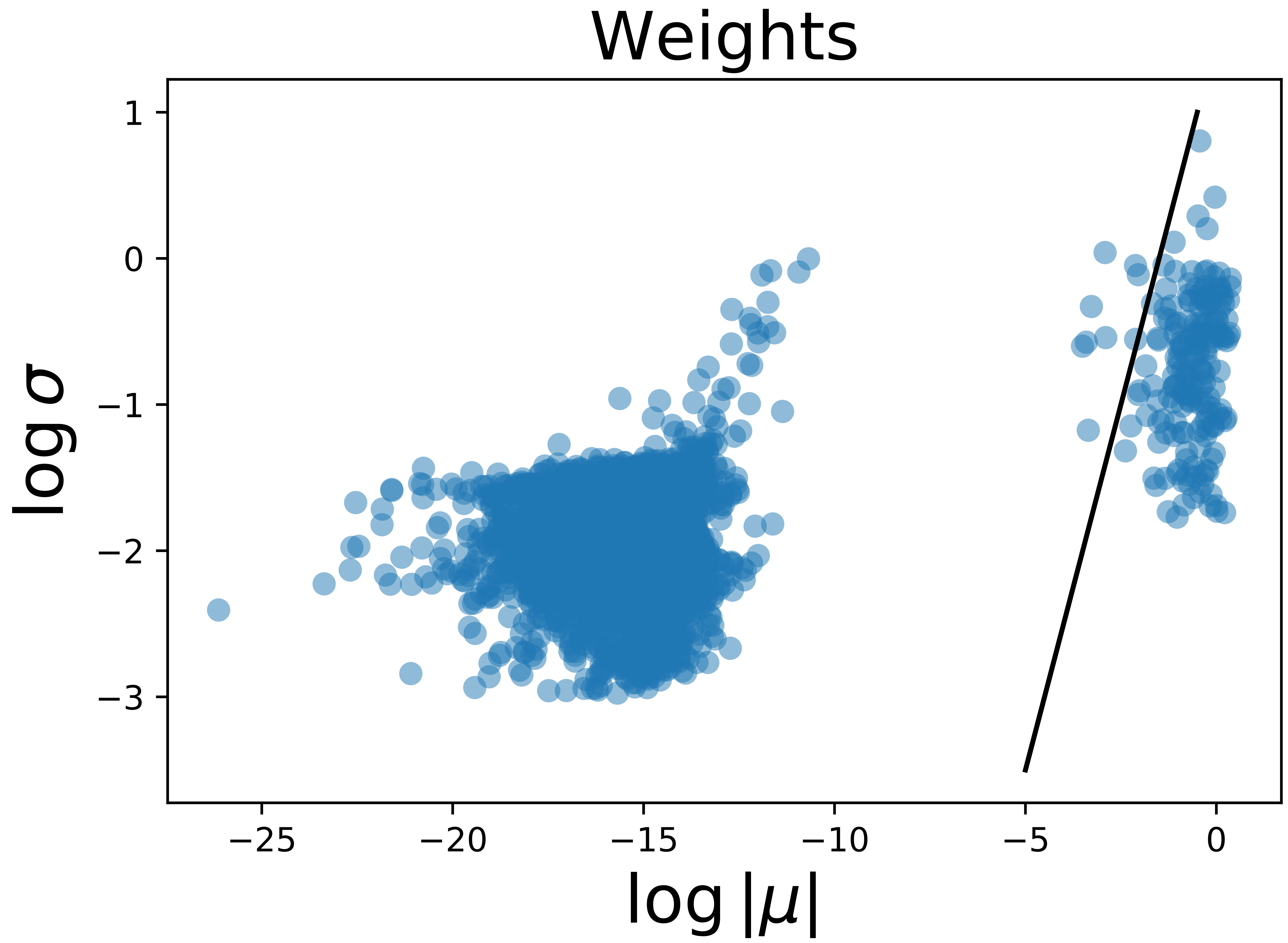
$$\theta_{ij} = 0, \;\; \sigma_{ij} = 0$$

Prediction for a mini-batch $X$ :

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

do not ensemble because we want
the most compact network

# Selecting threshold




Pruning criteria:

$$\frac{\mu_{ij}^2}{\sigma_{ij}^2} < 0.05$$

$$\Updownarrow$$

$$\log \sigma_{ij} - \log |\mu_{ij}| > 3$$

# Automatic relevance determination

(other prior but very similar model)

- Empirical Bayes
  - optimize w. r. t. prior parameters ⇒ automatically choose hyperparameters

Kharitonov, V. et al. Variational dropout via Empirical Bayes. NeurIPS Workshop on Bayesian Deep Learning, 2018.

# Automatic relevance determination

(other prior but very similar model)

- Empirical Bayes
  - optimize w. r. t. prior parameters ⇒ automatically choose hyperparameters
- Normal prior with **learnable** variance for **each weight**:

$$p(w_{ij}|\tau_{ij}) = \mathcal{N}(0, \tau_{ij}^{-1})$$

Kharitonov, V. et al. Variational dropout via Empirical Bayes. NeurIPS Workshop on Bayesian Deep Learning, 2018.

# Automatic relevance determination

(other prior but very similar model)

- Empirical Bayes
  - optimize w. r. t. prior parameters ⇒ automatically choose hyperparameters
- Normal prior with **learnable** variance for **each weight**:

$$p(w_{ij}|\tau_{ij}) = \mathcal{N}(0, \tau_{ij}^{-1})$$

- KL-divergence:

$$KL\big(\underbrace{q(w_{ij}|\mu_{ij}, \sigma_{ij})}_{\text{normal}}||\underbrace{p(w_{ij}|\tau_{ij})}_{\text{normal}}\big) = -\log \sigma_{ij} - \frac{1}{2}\log \tau_{ij} + \frac{1}{2}\tau_{ij}(\mu_{ij}^2 + \sigma_{ij}^2)$$

Kharitonov, V. et al. Variational dropout via Empirical Bayes. NeurIPS Workshop on Bayesian Deep Learning, 2018.

# Automatic relevance determination

(other prior but very similar model)

- Empirical Bayes
  - optimize w. r. t. prior parameters ⇒ automatically choose hyperparameters

- Normal prior with **learnable** variance for **each weight**:

$$p(w_{ij}|\tau_{ij}) = \mathcal{N}(0, \tau_{ij}^{-1})$$

- KL-divergence:

$$KL\big(q(w_{ij}|\mu_{ij}, \sigma_{ij})||p(w_{ij}|\tau_{ij})\big) = -\log\sigma_{ij} - \frac{1}{2}\log\tau_{ij} + \frac{1}{2}\tau_{ij}(\mu_{ij}^2 + \sigma_{ij}^2) \circeq$$

- Analytical optimization w. r. t. prior parameters $\tau_{ij}$ :

$$\boxed{\tau_{ij}^* = (\mu_{ij}^2 + \sigma_{ij}^2)^{-1}} \qquad \overset{\tau_{ij}^*}{\circeq} -\frac{1}{2}\log(\mu_{ij}^2 + \sigma_{ij}^2)^{-1} = -\frac{1}{2}\log\left(1 + \frac{\mu_{ij}^2}{\sigma_{ij}^2}\right)$$

Kharitonov, V. et al. Variational dropout via Empirical Bayes. NeurIPS Workshop on Bayesian Deep Learning, 2018.

# Comparing two models

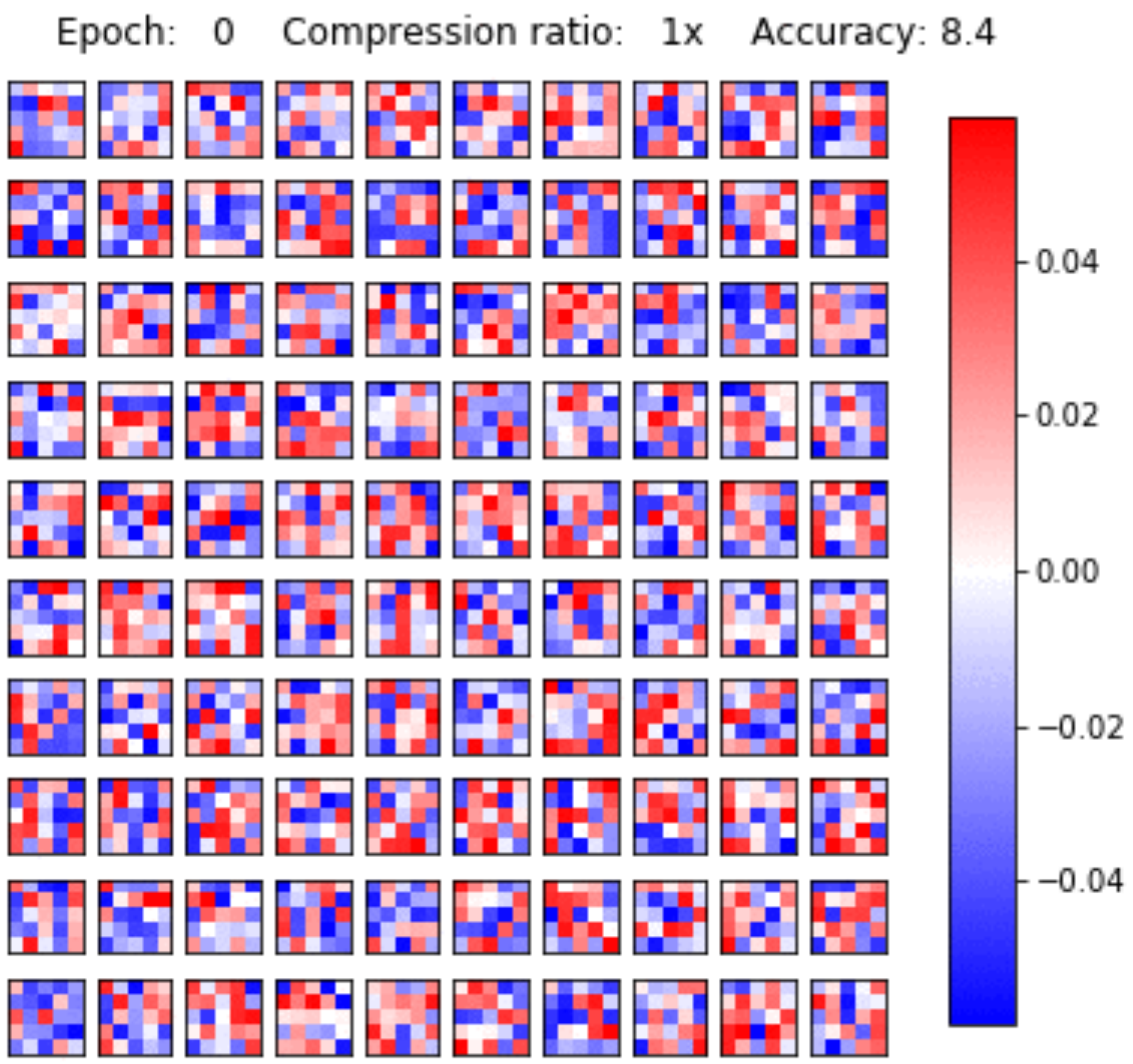
| | **SparseVD** | **ARD** |
|---|---|---|
| Prior | $p(w_{ij}) \propto \frac{1}{\lvert w_{ij} \rvert}$ | $p(w_{ij} \mid \tau_{ij}) = \mathcal{N}(0, \tau_{ij}^{-1})$ |
| Prior parameters | None | $\tau_{ij}$ |
| KL-divergence | $k_1 \sigma(k_2 + k_3 \log \alpha_{ij})) - \underbrace{-0.5 \log(1 + \alpha_{ij}^{-1})}_{\text{bounded}} + \mathrm{C}$ | $-0.5 \log(1 + \alpha_{ij}^{-1})$ |

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

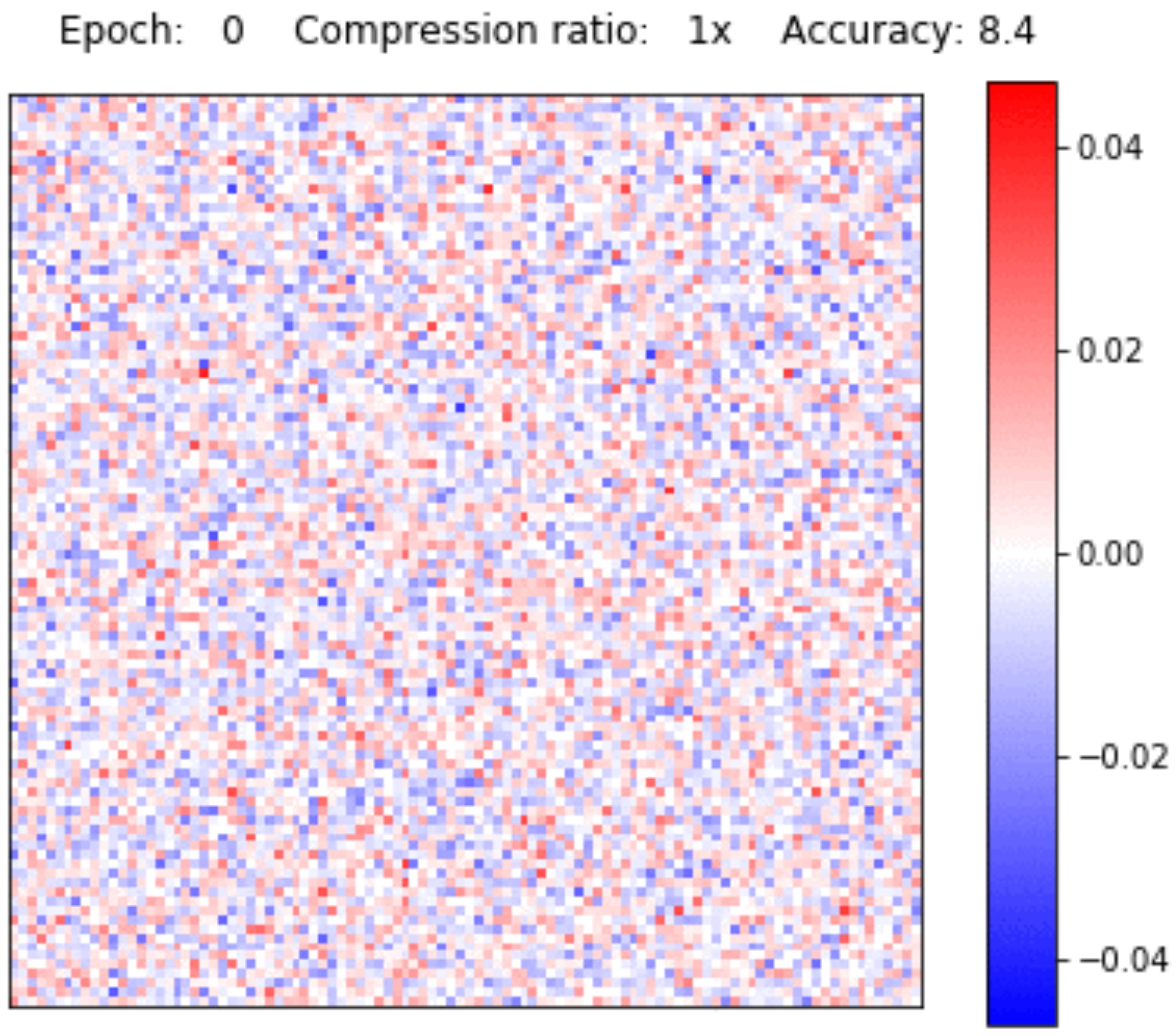
Training procedure is the same (approx. posterior, RT, LRT etc.)

Kharitonov, V. et al. Variational dropout via Empirical Bayes. NeurIPS Workshop on Bayesian Deep Learning, 2018.

# Sparse variational dropout: visualization




LeNet-5: convolutional layer



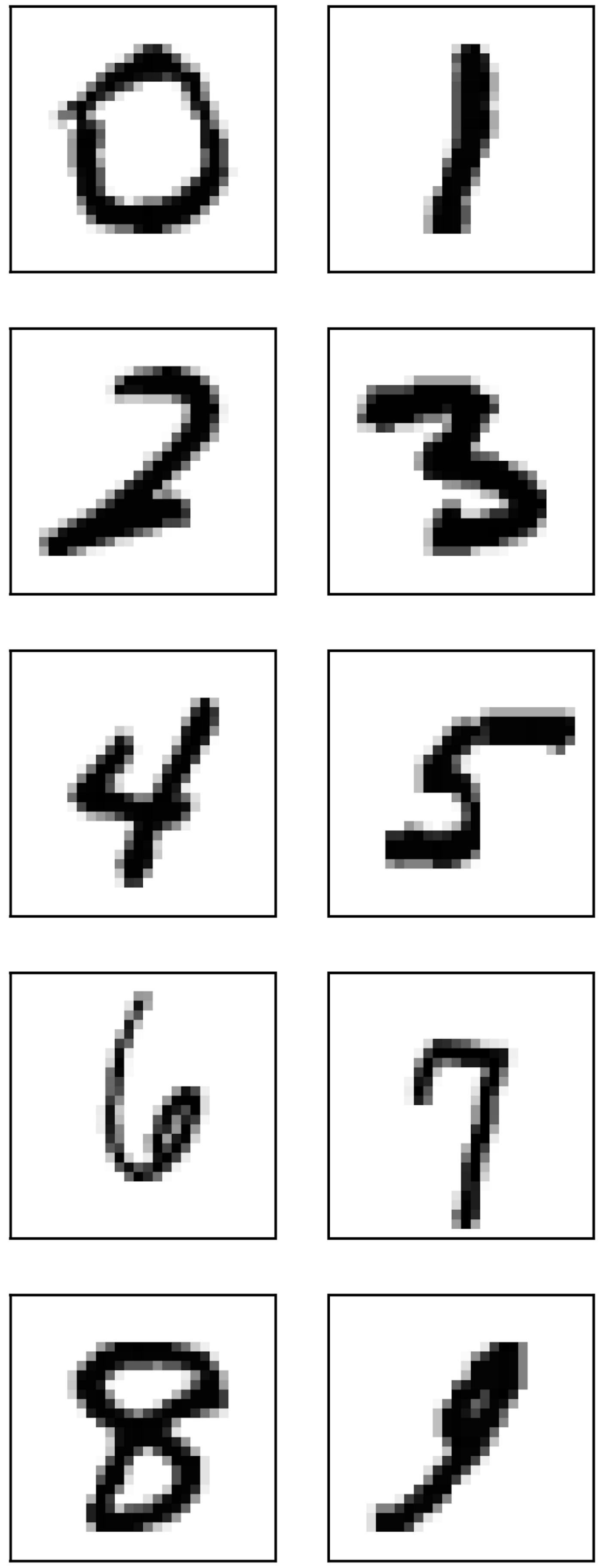

LeNet-5: fully-connected layer
(100 x 100 patch)

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

**Fully Connected network**: LeNet-300-100

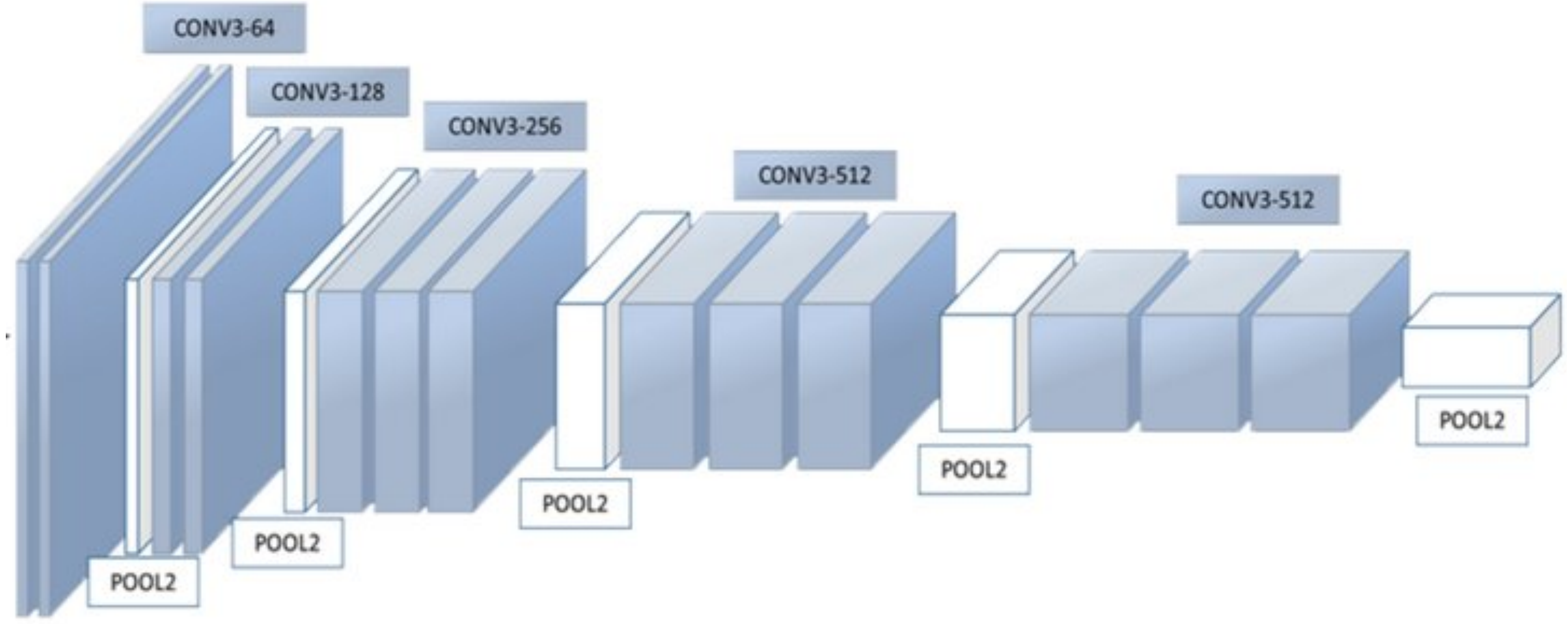
**Convolutional network**: Lenet-5-Caffe

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



Molchanov, Dmitry et al. Variational dropout sparsifies deep neural networks. ICML 2017

# VGG-like on CIFAR-10

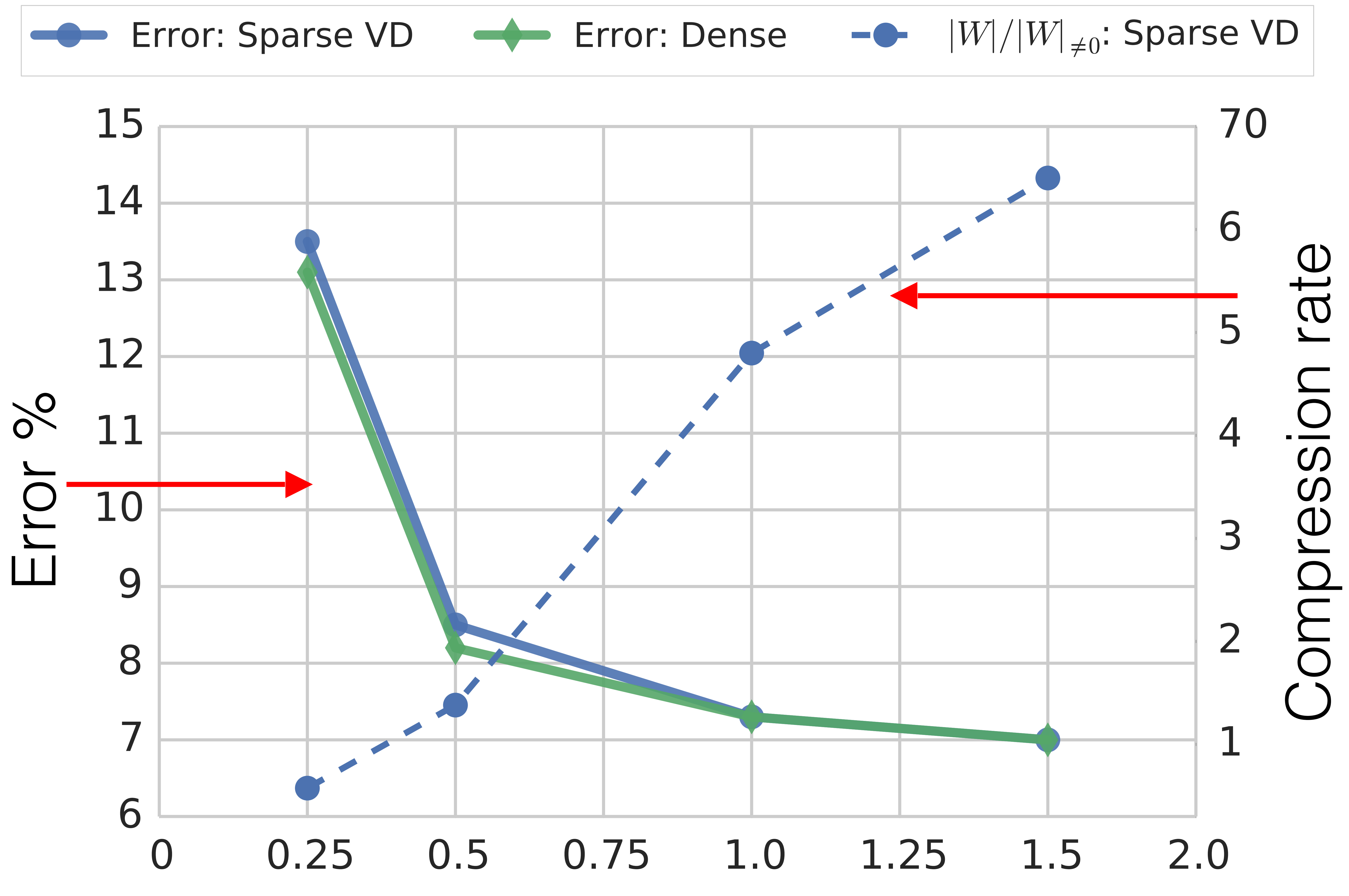



- 13 Convolutional layers and 2 Fully-Connected layers
- Pre-Activation Batch Norm and Binary Dropout after each layer

Official Torch Blog: Zagoruyko, http://torch.ch/blog/2015/07/30/cifar.html

# 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 ⇒ Sparse VD yields an empty model where conventional models easily overfit.

Zhang, Chiyuan, et al. "Understanding deep learning requires rethinking generalization."

# Agenda

- Sparsification: what and why
- Bayesian neural networks
- Sparse variational dropout
- $\boxed{\text{Practical assignment: implementation of SparseVD}}$
- Model enhancements

# Practical assignment

# Agenda

- Sparsification: what and why
- Bayesian neural networks
- Sparse variational dropout
- Practical assignment: implementation of SparseVD
- $\boxed{\text{Model enhancements}}$

# What's next?

- SparseVD for RNNs
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

# What’s next?

- $\boxed{\text{SparseVD for RNNs}}$
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

# SparseVD for recurrent neural networks

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

Model specification:

- Choose particular prior ✓ log-uniform distribution

Training:

- Choose particular family for approximate posterior ✓ normal distribution
- How to compute KL-divergence? ✓ analytical approximation
- How to estimate the expectation? ?

# Important RNN specifics

Input is a sequence: $x = x_1 \dots x_T$ :

$$-\sum_{i=1}^{N} \int q(w|\theta,\sigma) \log p(y^i|\overbrace{x_0^i,\dots,x_T^i}^{\textbf{sequence}}, w, B)dw +$$

$$+ \sum_{w_{ij}\in\omega} KL(q(w_{ij}|\theta_{ij},\sigma_{ij})||p(w_{ij})) \to \min_{\theta,\sigma,B}$$

1 Use one sample $w$ for all $t$ for 1 sequence

Gal, Yarin and Ghahramani, Zoubin. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. NIPS 2016

# Important RNN specifics

Input is a sequence: $x = x_1 \dots x_T$ :

$$-\sum_{i=1}^{N} \int q(w|\theta,\sigma) \log p(y^i|\overbrace{x_0^i,\dots,x_T^i}^{\text{sequence}}, w, B) dw +$$

$$+ \sum_{w_{ij} \in \omega} KL(q(w_{ij}|\theta_{ij},\sigma_{ij})||p(w_{ij})) \to \min_{\theta,\sigma,B}$$

1 Use one sample $w$ for all $t$ for 1 sequence

2 LRT is not applicable $\Rightarrow$ sample one $w$ for a whole mini-batch

Chirkova, Nadezhda et al. Bayesian compression for natural language processing. EMNLP 2018

# Why LRT is not applicable in RNNs?

$$h_{t+1} = g(W^x x_{t+1} + W^h h_t + b^h)$$

For $W^h$:

$h_t$ is a random variable depending on $W^h$ → $W^h h_t$ is not normal

For $W^x$:

Sampling same noise on weights for all t $\neq$ Sampling same noise on preactivation for all t

Chirkova, Nadezhda et al. Bayesian compression for natural language processing. EMNLP 2018

# Experiments: text classification

Datasets: IMDb (25K texts, 2 classes), AGNews (120K texts, 4 classes)

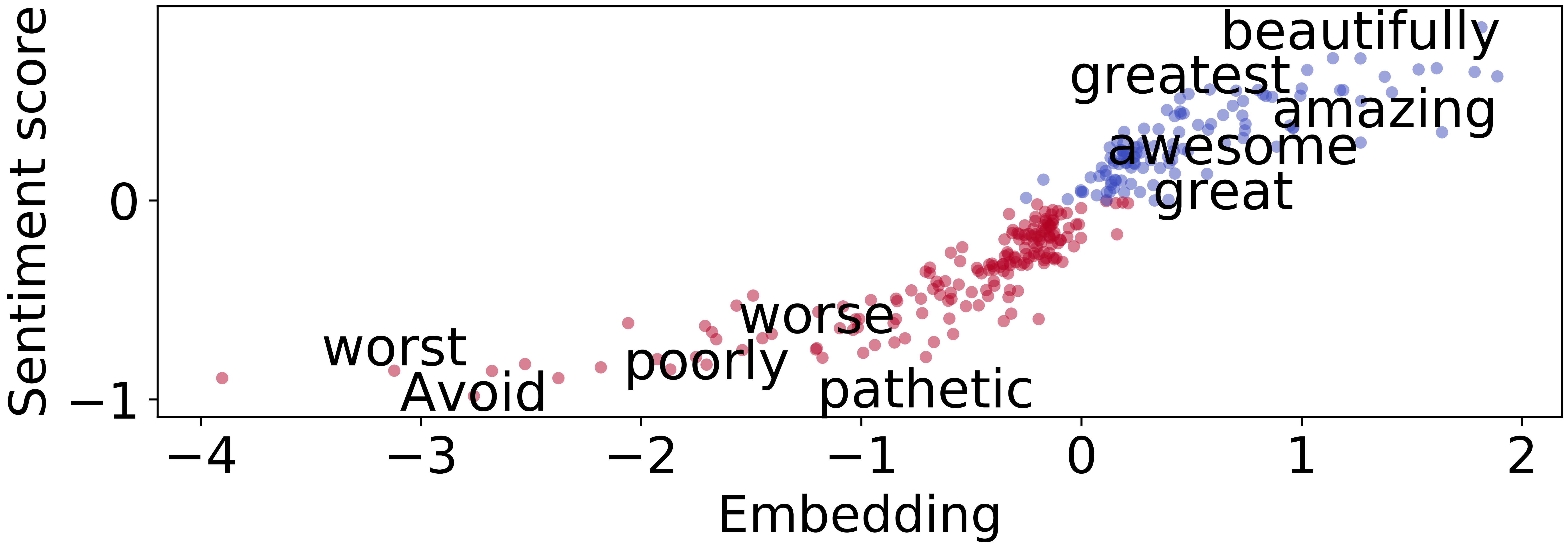
Architecture: Embedding → LSTM → FC on $h_T$

| Task | Method | Accuracy % | Compression | Vocabulary |
|---|---|---|---|---|
| IMDb | Original | 84.1 | 1x | 20000 |
| | SparseVD | **85.1** | 1135x | 4611 |
| | SparseVD-Voc | 83.6 | **12985x** | **292** |
| AGNews | Original | **90.6** | 1x | 20000 |
| | SparseVD | 88.8 | 322x | 5727 |
| | SparseVD-Voc | 89.2 | **469x** | **2444** |

Chirkova, Nadezhda et al. Bayesian compression for natural language processing. EMNLP 2018

# Experiments: text classification

IMDb: the only remaining embedding component for remaining words




sentiment score = (#pos. — #neg.) / # all texts with word

Chirkova, Nadezhda et al. Bayesian compression for natural language processing. EMNLP 2018

# Experiments: text generation

PTB Corpus: char-level (6M chars), word-level (0.9M words)

Architecture: LSTM → FC

| Task | Method | Valid | Test | Compression | Vocabulary |
|---|---|---|---|---|---|
| | Original | 1.498 | 1.454 | 1x | 50 |
| Char PTB | SparseVD | 1.472 | 1.429 | **7.6x** | 50 |
| (Bits-per-char) | SparseVD-Voc | **1.4584** | **1.4165** | 5.8x | **48** |
| | Original | 135.6 | 129.5 | 1x | 10000 |
| Word PTB | SparseVD | **115.0** | **109.0** | **14.0x** | 9985 |
| (Perplexity) | SparseVD-Voc | 126.3 | 120.6 | 11.1x | **4353** |

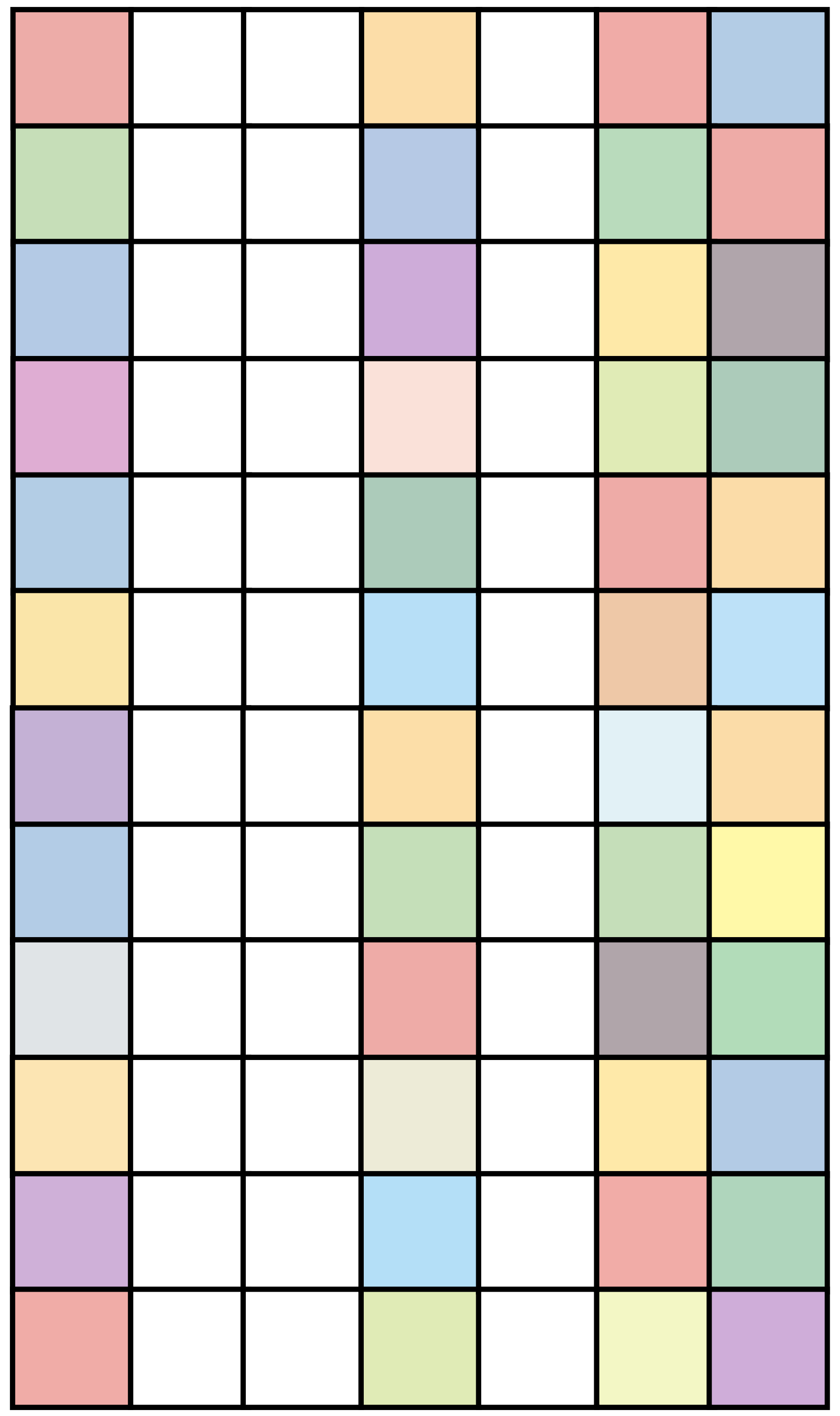
Chirkova, Nadezhda et al. Bayesian compression for natural language processing. EMNLP 2018

# What's next?

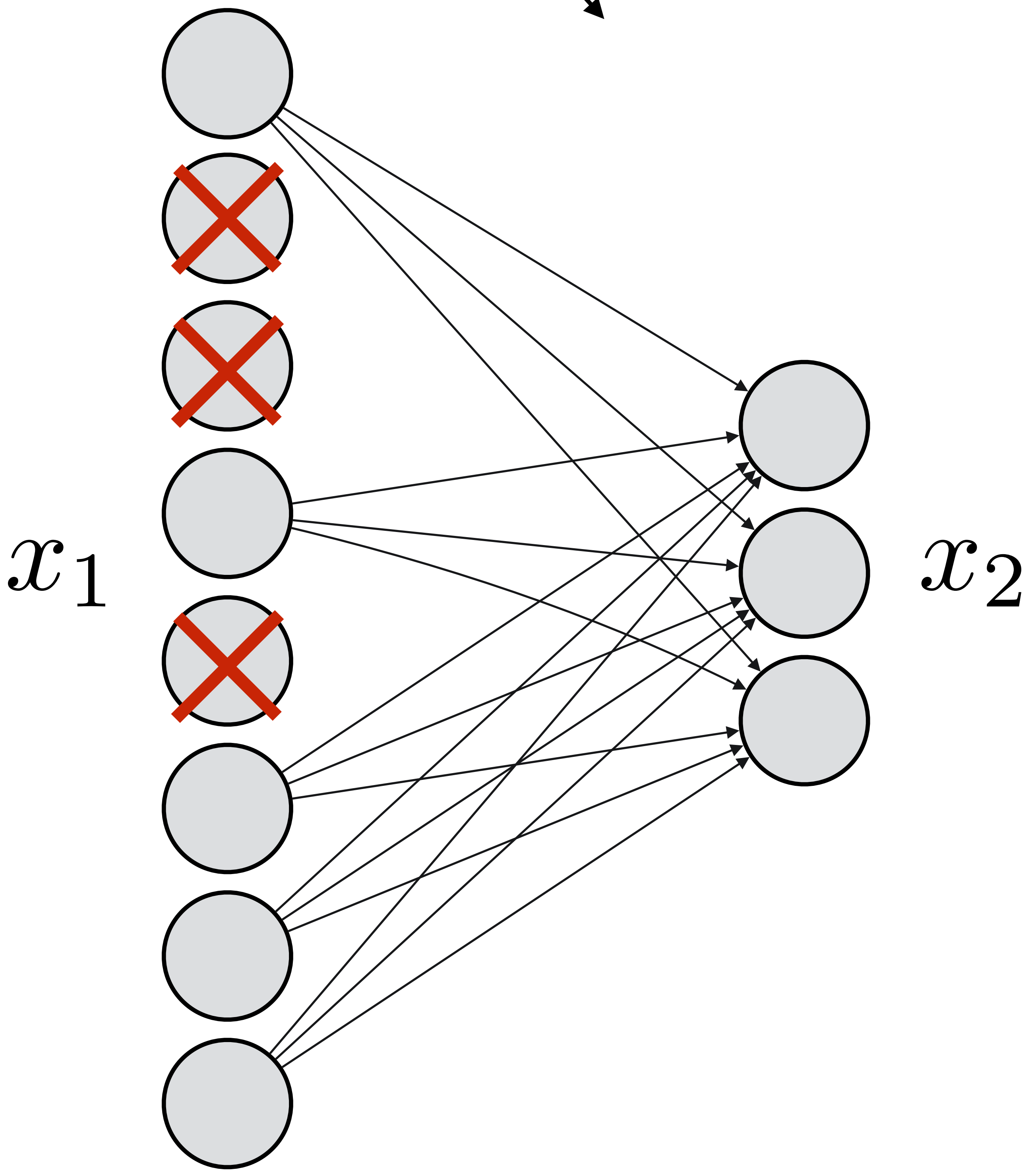
- SparseVD for RNNs
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

# Structured Bayesian sparsification

No outgoing edges
⇒ remove neuron



Weight matrix $W$




Computational graph

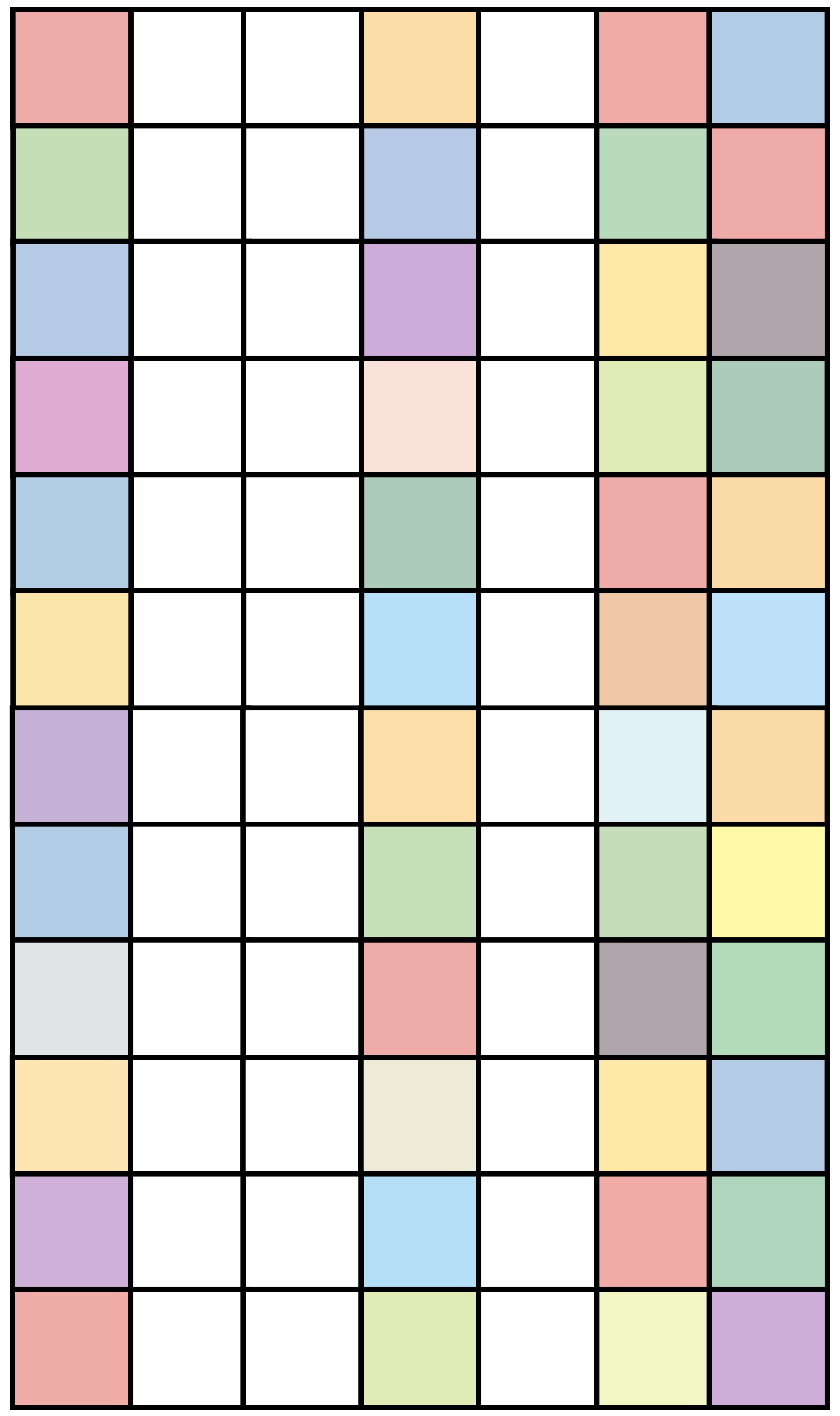
Benefits of structured sparsification:

- more efficient compression
- speed-up of forward pass (faster testing stage)

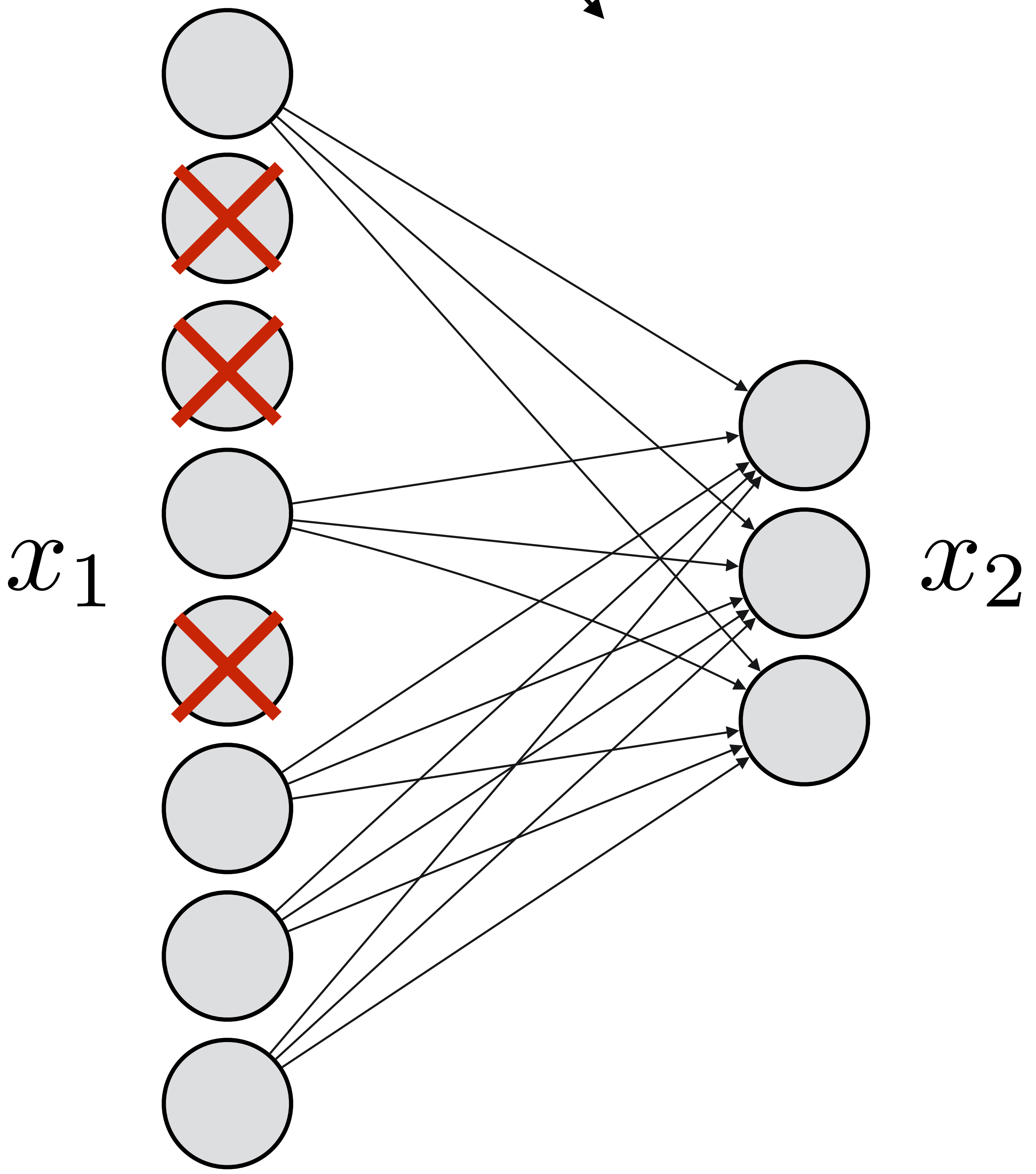
# Structured Bayesian sparsification

No outgoing edges
$\Rightarrow$ remove neuron

$$x_2 = \sigma(Wx_1 + b)$$



Weight matrix $W$

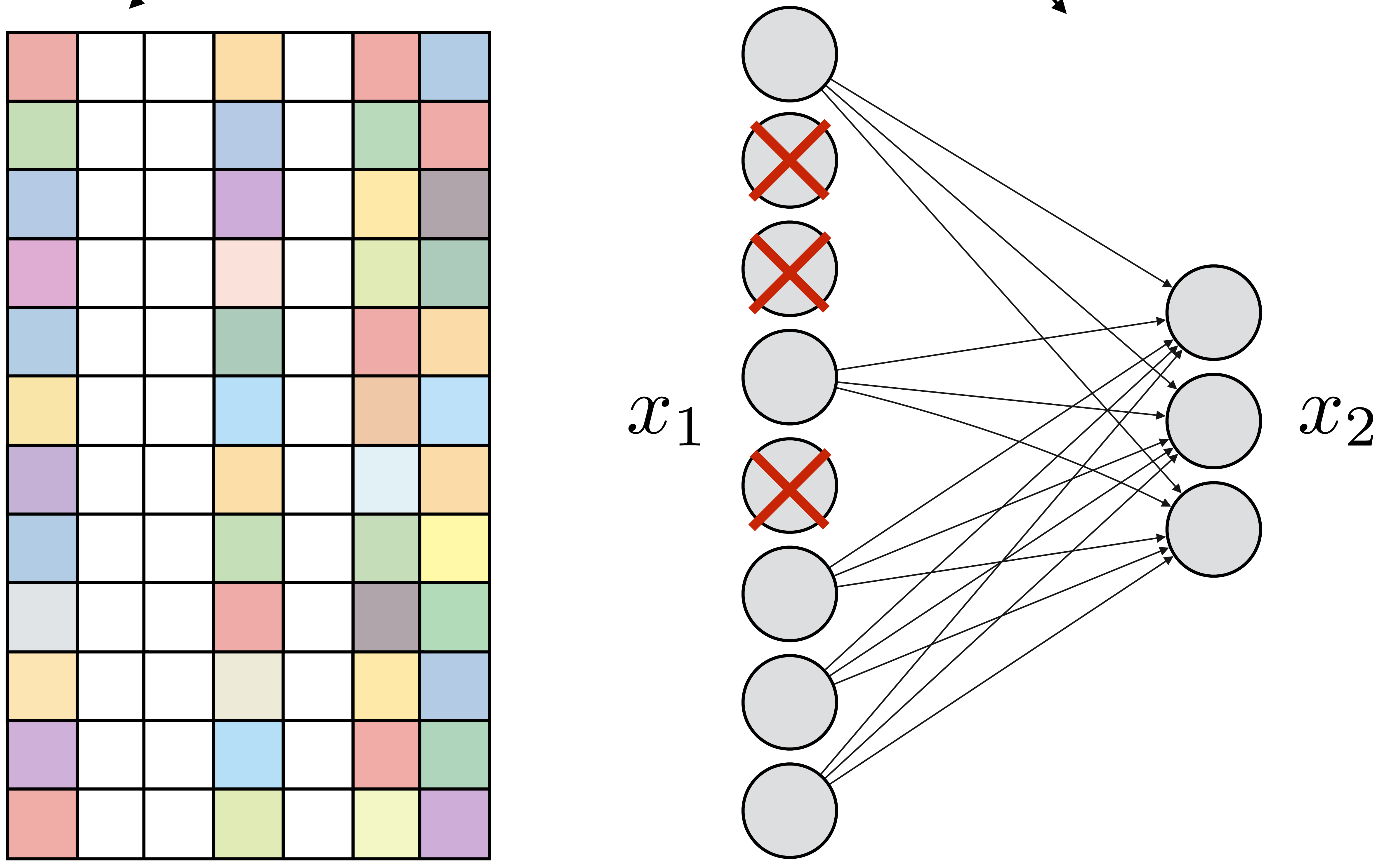



Computational graph

# Structured Bayesian sparsification

No outgoing edges

⇒ remove neuron




Weight matrix $W$ Computational graph

$$x_2 = \sigma(Wx_1 + b)$$

Multiply $x_1$ by group variable $z$ :

$$x_2 = \sigma(W(x_1 \odot z) + b)$$

zero component in $z$ ⇒

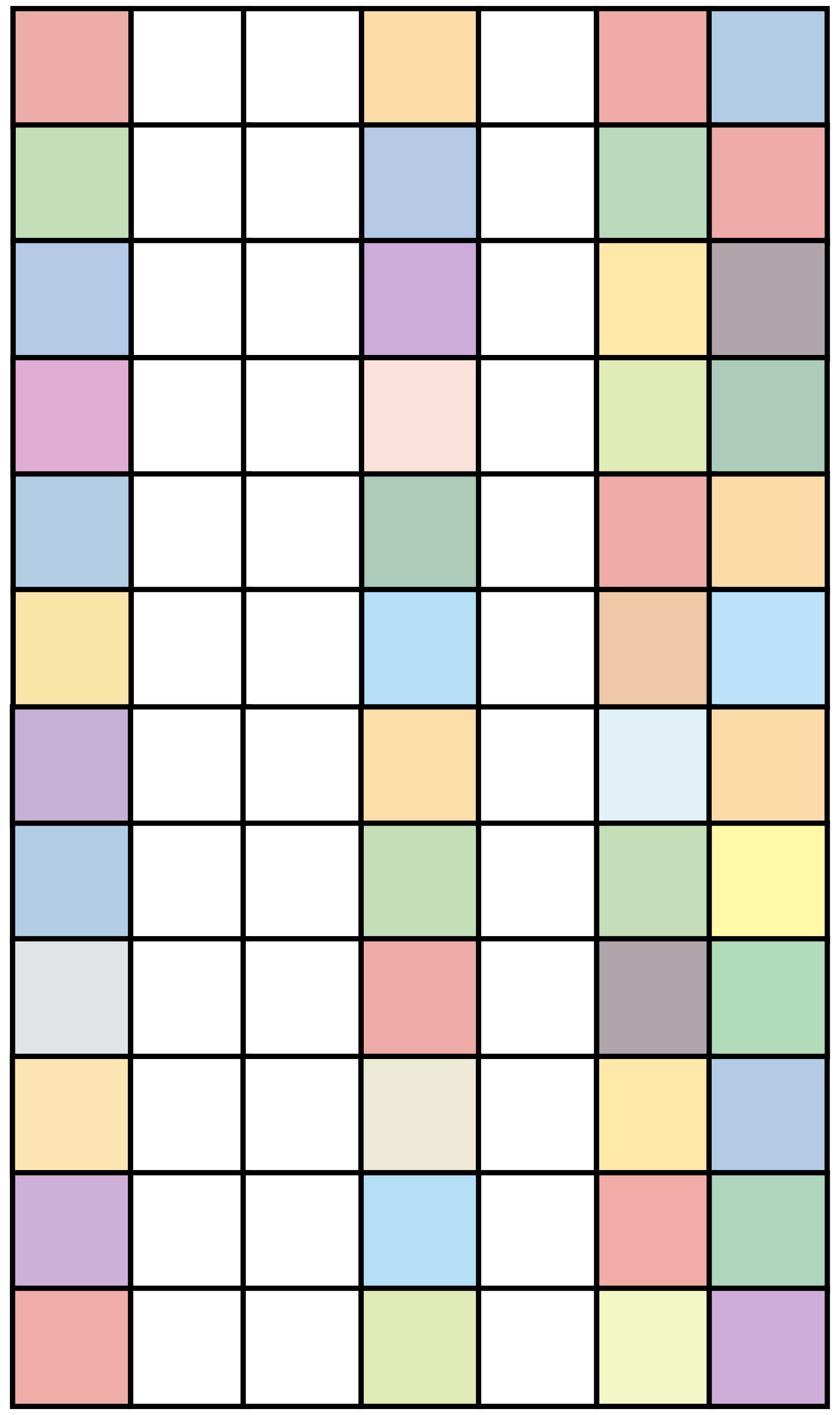
no outgoing edges ⇒

remove neuron

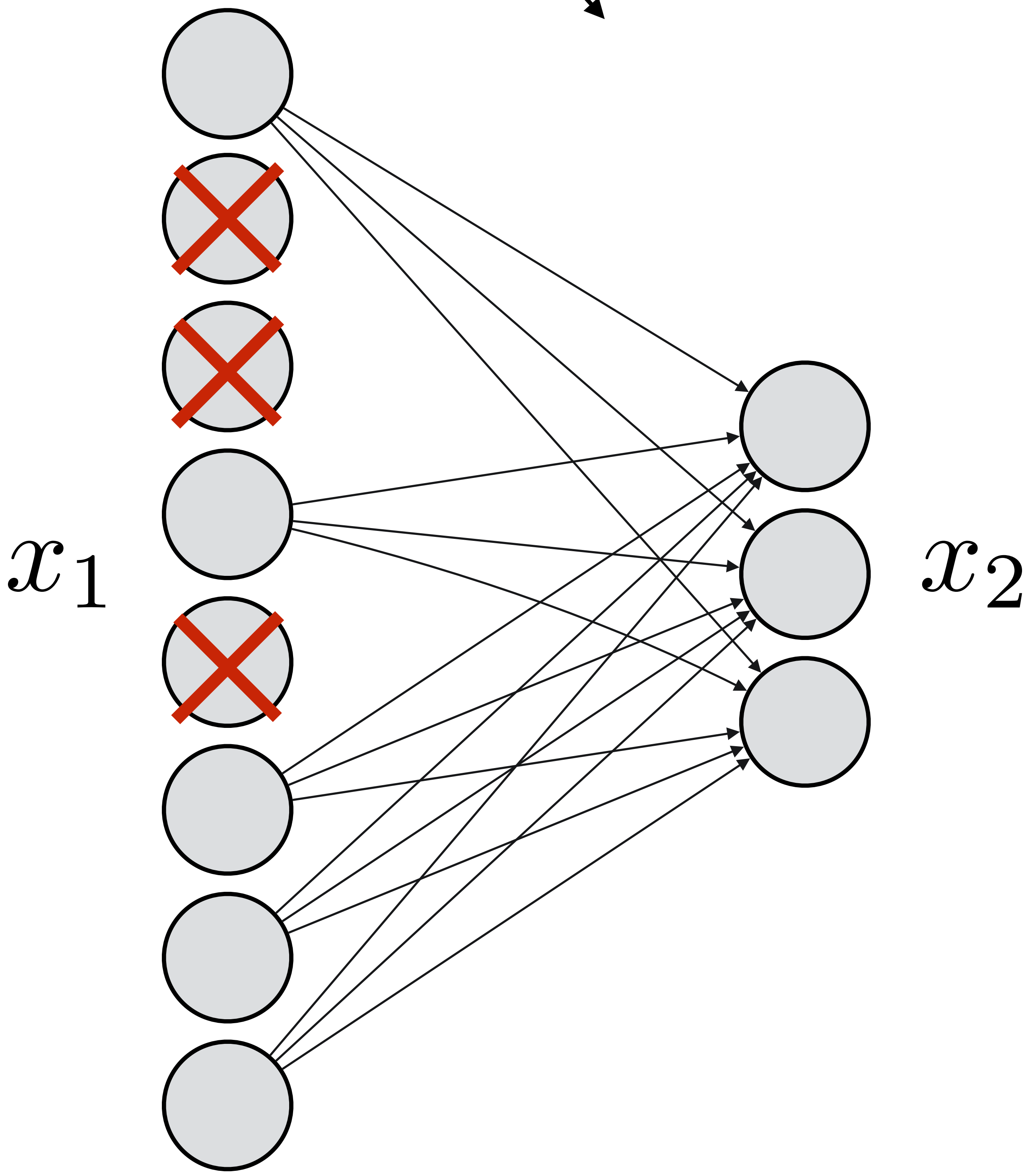
Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# Structured Bayesian sparsification: training

No outgoing edges
$\Rightarrow$ remove neuron



Weight matrix $W$




Computational graph

$$x_2 = \sigma(W(x_1 \odot z) + b)$$

minor modification of computational graph

Treat $z$ in **the same way** as $W$ :

- Log-uniform prior on $z$ and on $W$
- Normal approx. posterior for both
- RT & LRT for $W$
- RT for $z$ (LRT is not needed)

# A bit more mathematical motivation

Let's choose standard normal prior on $W$ :

$$x_2 = \sigma(W(x_1 \odot z) + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(w_{ij}) = \mathcal{N}(0, 1)$$

$z$ $W$

independent

Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# A bit more mathematical motivation

Let's choose standard normal prior on $W$ :

$$x_2 = \sigma(W(x_1 \odot z) + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(w_{ij}) = \mathcal{N}(0, 1)$$

$z$ $W$

independent

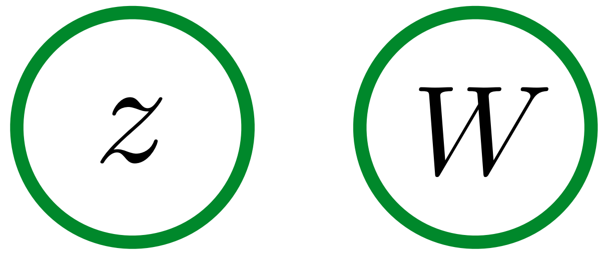
$$\boxed{p(\underbrace{w_{ij} z_i}_{\tilde{w}_{ij}} | z_i) = \mathcal{N}(0, z_i^2)}$$

Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# A bit more mathematical motivation

Let's choose standard normal prior on $W$ :

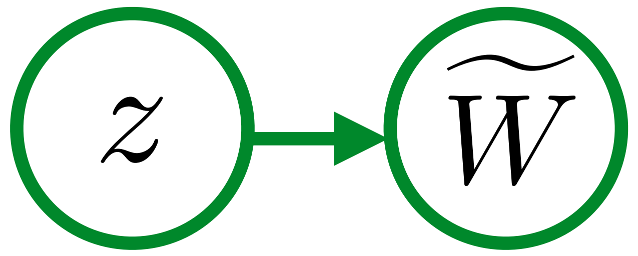
$$x_2 = \sigma(W(x_1 \odot z) + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(w_{ij}) = \mathcal{N}(0,1)$$




independent

Equivalent model:

$$x_2 = \sigma(\widetilde{W} x_1 + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(\tilde{w}_{ij}|z_i) = \mathcal{N}(0, z_i^2)$$




dependent!

Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# A bit more mathematical motivation

Let's choose standard normal prior on $W$ :

$$x_2 = \sigma(W(x_1 \odot z) + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(w_{ij}) = \mathcal{N}(0, 1)$$

$z$ $\quad$ $W$

independent

Equivalent model:

$$x_2 = \sigma(\widetilde{W} x_1 + b) \qquad p(z_i) \propto \frac{1}{|z_i|}; \quad p(\tilde{w}_{ij} | z_i) = \mathcal{N}(0, z_i^2)$$

$z \rightarrow \widetilde{W}$

dependent!

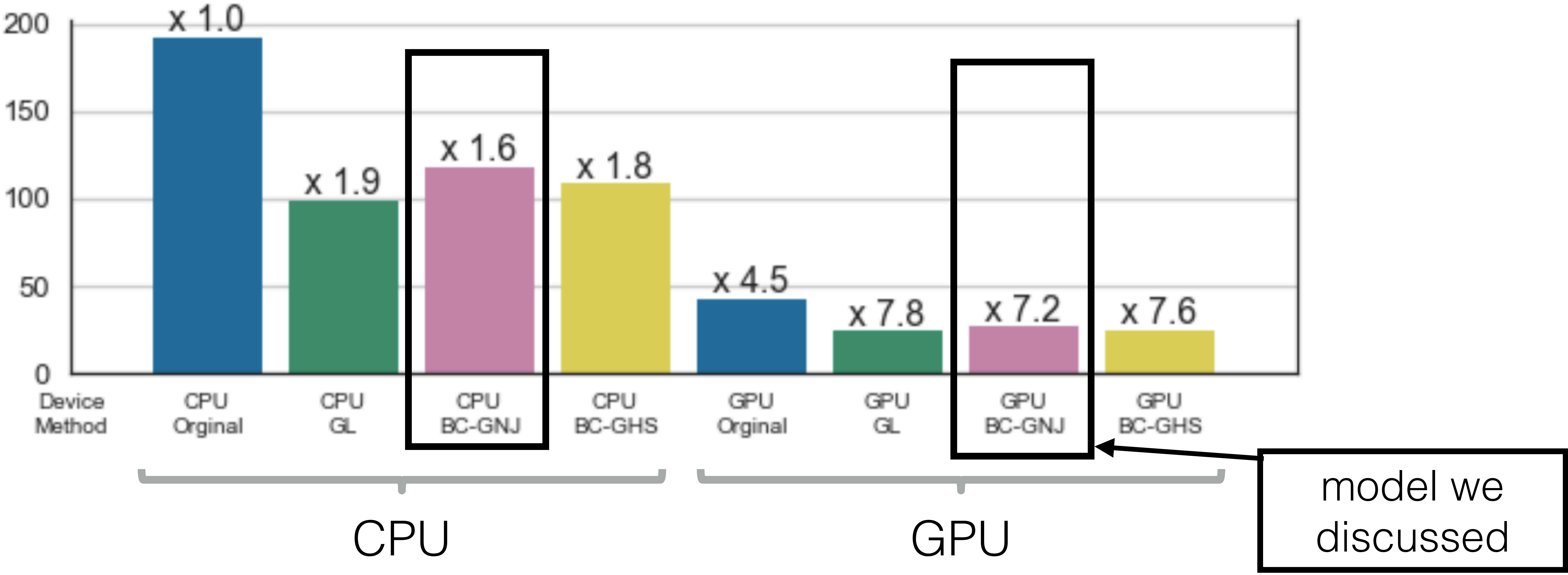
$$p(\tilde{w}_{ij}) \propto \int \frac{1}{|z_i|} \mathcal{N}(0, z_i^2) dz_i = \frac{1}{|\tilde{w}_{ij}|}$$

SparseVD model!
But with structured sparsity

Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# Structured Bayesian sparsification: results

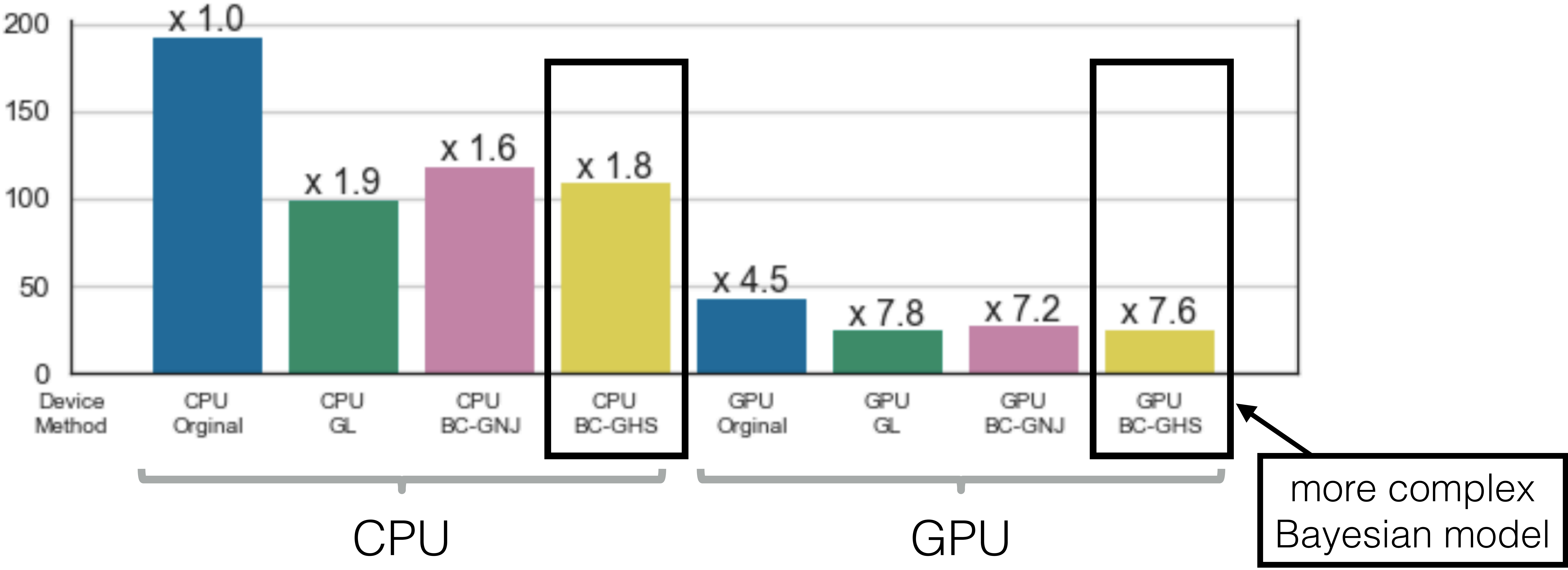
Speed-up of forward pass (testing stage) for Lenet-5-Caffe




Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# Structured Bayesian sparsification: results

Speed-up of forward pass (testing stage) for Lenet-5-Caffe




Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

# What's next?

- SparseVD for RNNs
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

# Two popular frameworks for sparsification

| **Magnitude pruning** | **Bayesian sparsification** |
|---|---|
| A lot of method hyperparameters | (Almost) no method hyperparameters |
| Need to choose training schedule | Need to choose training schedule |
| non-Bayesian | Bayesian! |

# The state of sparsity in deep neural networks

- Recent (2019) work on comparing different (unstructured) sparsification techniques on **large** datasets and models:
  - Transformer for neural machine translation (WMT 2014 English-to-German)
  - ResNet-50 for ImageNet
- Open-source code and top performing model checkpoints for reproducibility
- **Thousands** of experiments!

Gale, Trevor et al. The State of Sparsity in Deep Neural Networks. Arxiv, 2019

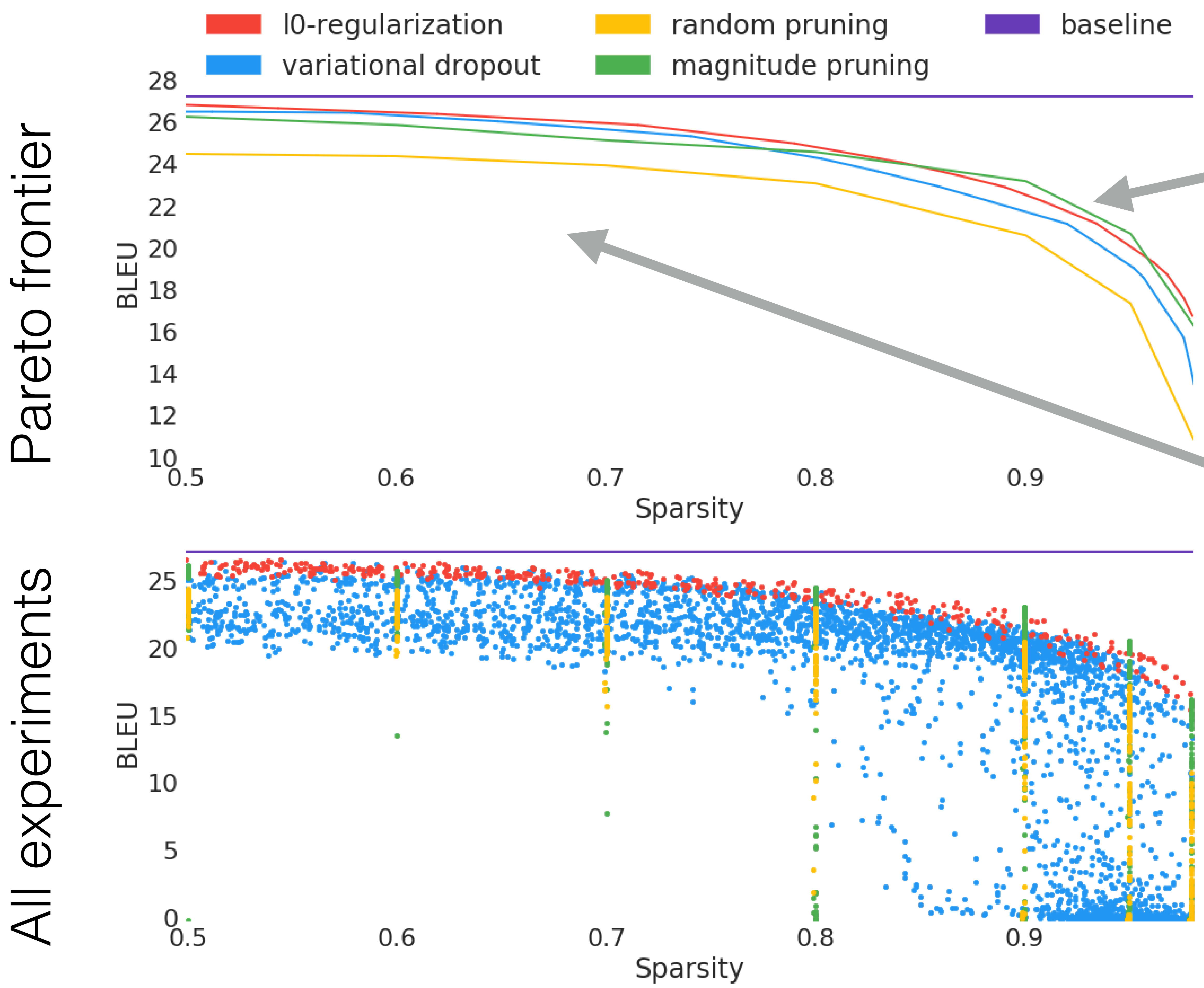
# Hyperparameters for both tasks

- Hyperparameters tuned in Sparse variational dropout:
  - KL-divergence weight & annealing schedule (not grounded theoretically, but works in practice)
  - Sparsification threshold

- Hyperparameters tuned in Magnitude pruning:
  - target sparsity
  - starting iteration of the sparsification process
  - ending iteration of the sparsification process
  - frequency of pruning steps

Gale, Trevor et al. The State of Sparsity in Deep Neural Networks. Arxiv, 2019

# Transformer for neural machine translation



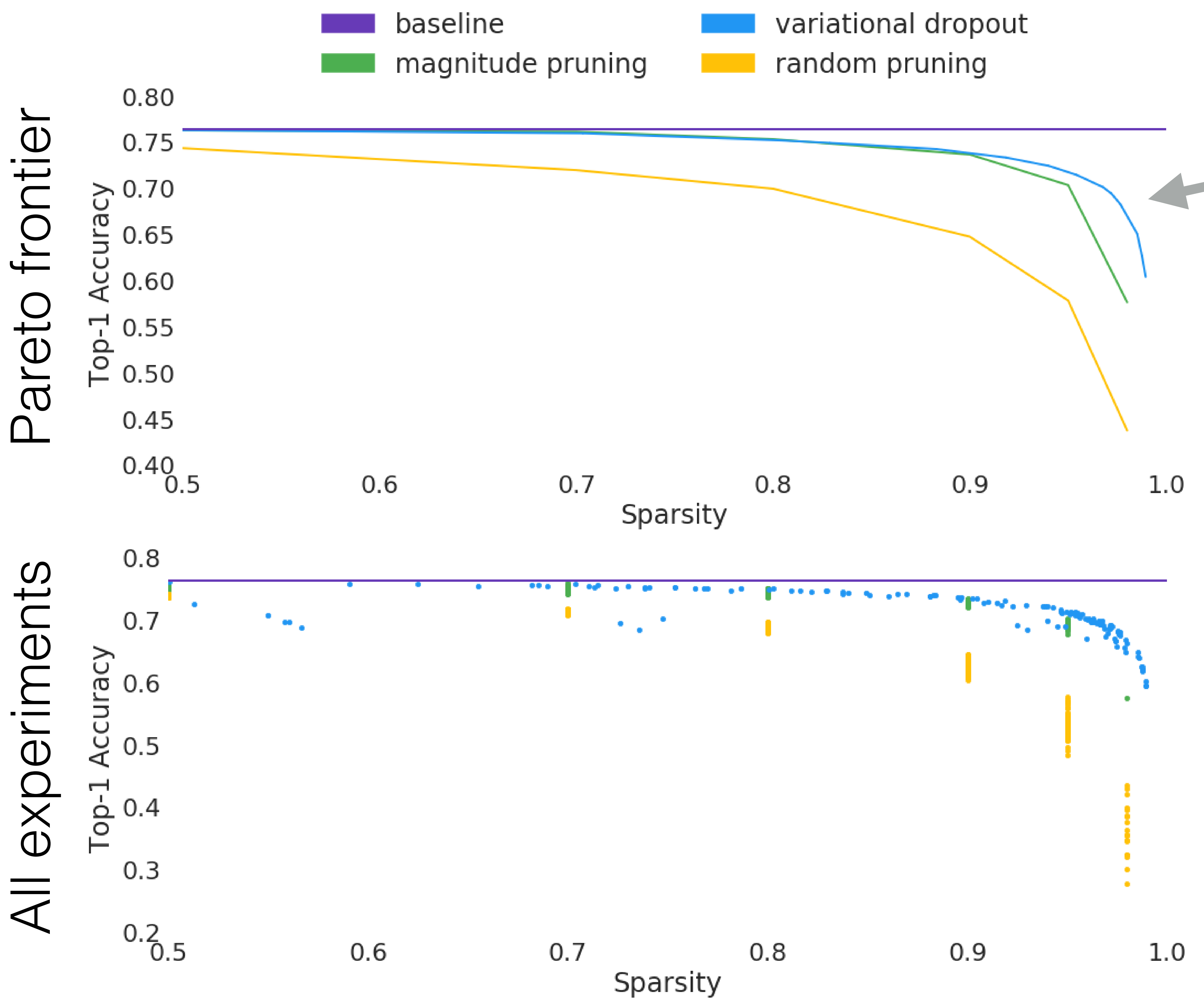

Magnitude pruning outperforms SparseVD a bit for high compression ratio

L0-regularization (another Bayesian approach) outperform others for small compression ratio

The relative performance of all three techniques is remarkably consistent

Gale, Trevor et al. The State of Sparsity in Deep Neural Networks. Arxiv, 2019

# ResNet-50 for ImageNet




SparseVD outperforms magnitude pruning for high compression ratio

The relative performance of all three techniques is remarkably consistent

Gale, Trevor et al. The State of Sparsity in Deep Neural Networks. Arxiv, 2019

# Other Bayesian sparsification models

- Group horseshoe with half-Cauchy scale priors:

  more levels is model hierarchy & usually better compression

$$s \sim \mathcal{C}^+(0, \tau_0); \qquad \tilde{z}_i \sim \mathcal{C}^+(0, 1); \qquad \tilde{w}_{ij} \sim \mathcal{N}(0, 1); \qquad w_{ij} = \tilde{w}_{ij}\tilde{z}_i s$$

- $L_0$ regularization

  relax non-differentiable $L_0$ regularizer & obtain exact zero weights

$$\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{N}\left(\sum_{i=1}^{N} \mathcal{L}\big(h(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i\big)\right) + \lambda \|\boldsymbol{\theta}\|_0$$

  relaxation is based on spike-and-slab prior (as well as approox. posterior)

$$p(z) = \mathbf{Bernoulli}(\pi), \qquad p(\theta | z = 0) = \delta(\theta), \qquad p(\theta | z = 1) = \mathcal{N}(\theta | 0, 1)$$

Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017
Louizos, Learning Sparse Neural Networks through L0 Regularization. ICLR 2018

# Summary

- Sparsification of neural networks is an urgent industrial problem that is well solved using Bayesian deep learning
- Bayesian sparsification relies on Bayesian neural networks that provide regularization, fast ensembling, uncertainty estimation etc.
- Bayesian sparsification is a theoretically grounded approach but several tricks are needed to train the model

p(B|A)yesgroup.ru