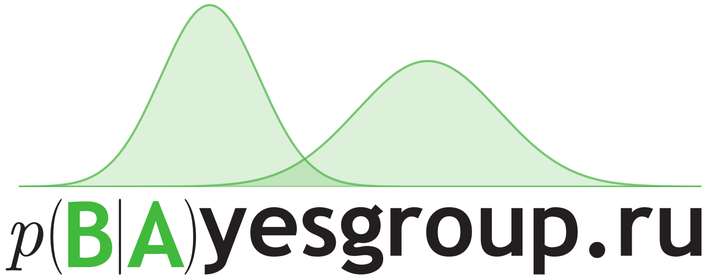
Bayesian neural networks and their applications

Nadezhda Chirkova

Higher School of Economics, Samsung AI Laboratory
Moscow, Russia

Lecture is here

Machine Learning in High Energy Physics
Summer School 2019
July 1–10, 2019. Hamburg, Germany



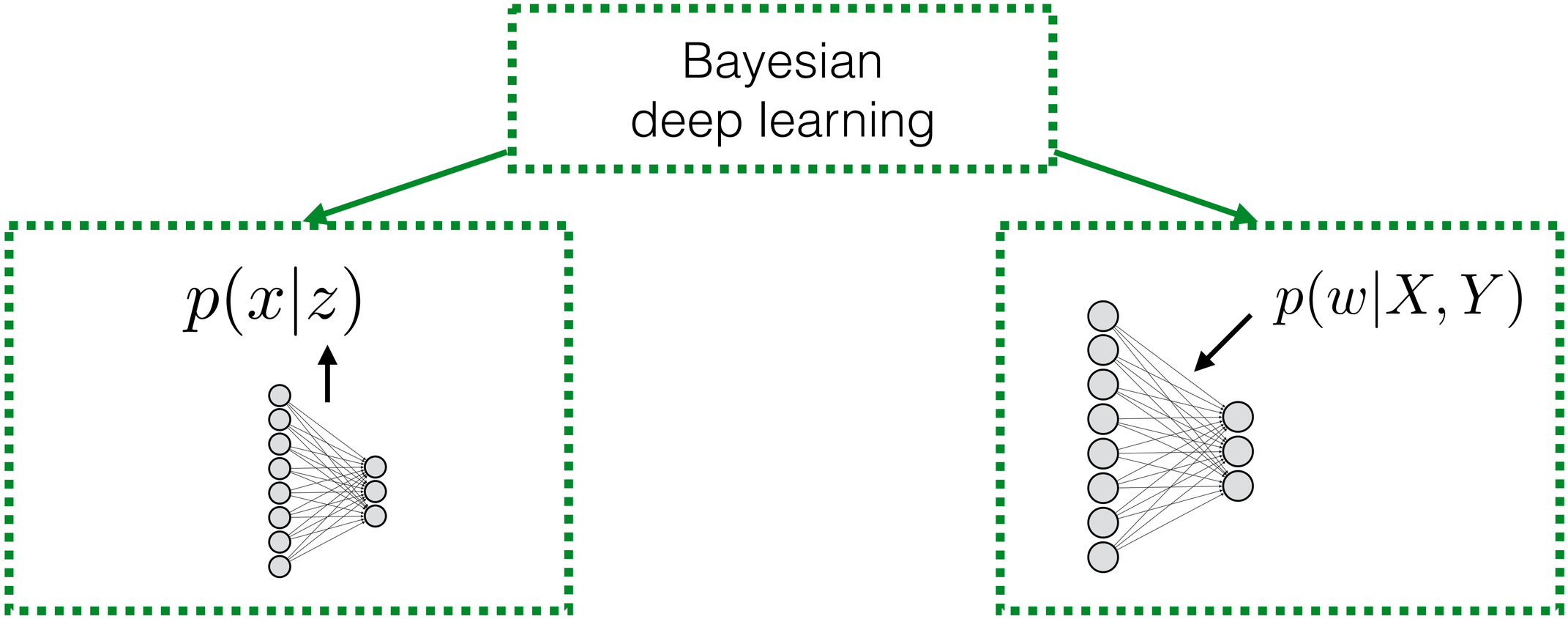
Plan of the first part: Bayesian neural networks

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- Example

Plan of the first part: Bayesian neural networks

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- Example

Bayesian deep learning



Deep generative models:
variational autoencoders,
normalizing flows …

Bayesian neural networks
(discriminative)

Variational autoencoders

Data
(no labels):

Probabilistic
model of data
(can generate
new objects):

$$p(x|z) = \prod_{i,j} p(x_{ij}|z_{ij}) = \prod_{i,j} \mathcal{N}(\mu_{ij}(z), \sigma^2_{ij}(z))$$

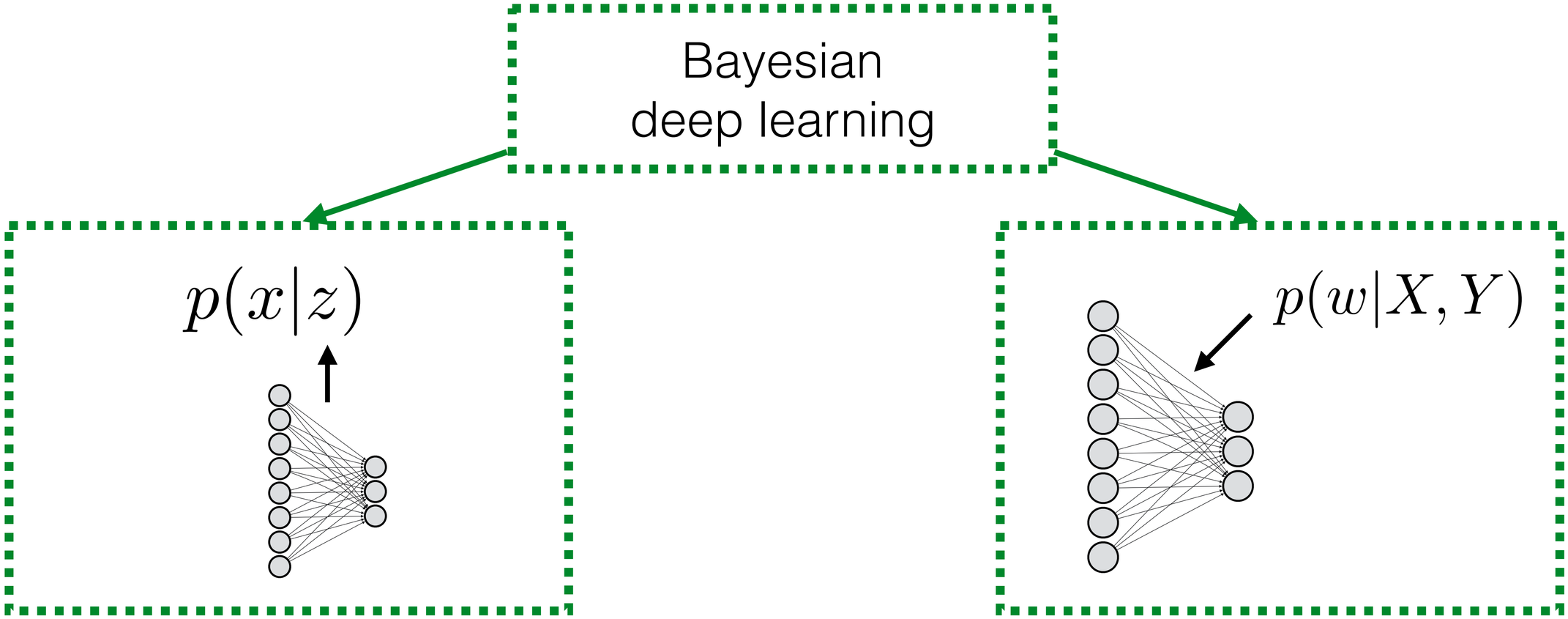
$$p(z) = \mathcal{N}(0, I)$$

Neural network

Neural network

How to train such models: Friday lecture

Bayesian deep learning



Deep generative models:
variational autoencoders,
normalizing flows …

Bayesian neural networks
(discriminative)

Regularization by noise

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

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

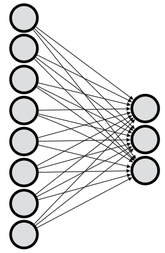
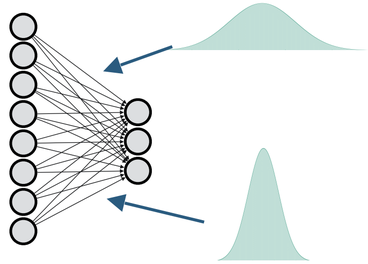
- More recent (1990+) approaches: regularization by noise
    - Data augmentation, dropout, gradient noise

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

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

Why go Bayesian?

A principled framework with many useful applications

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

Ensembling

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

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

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

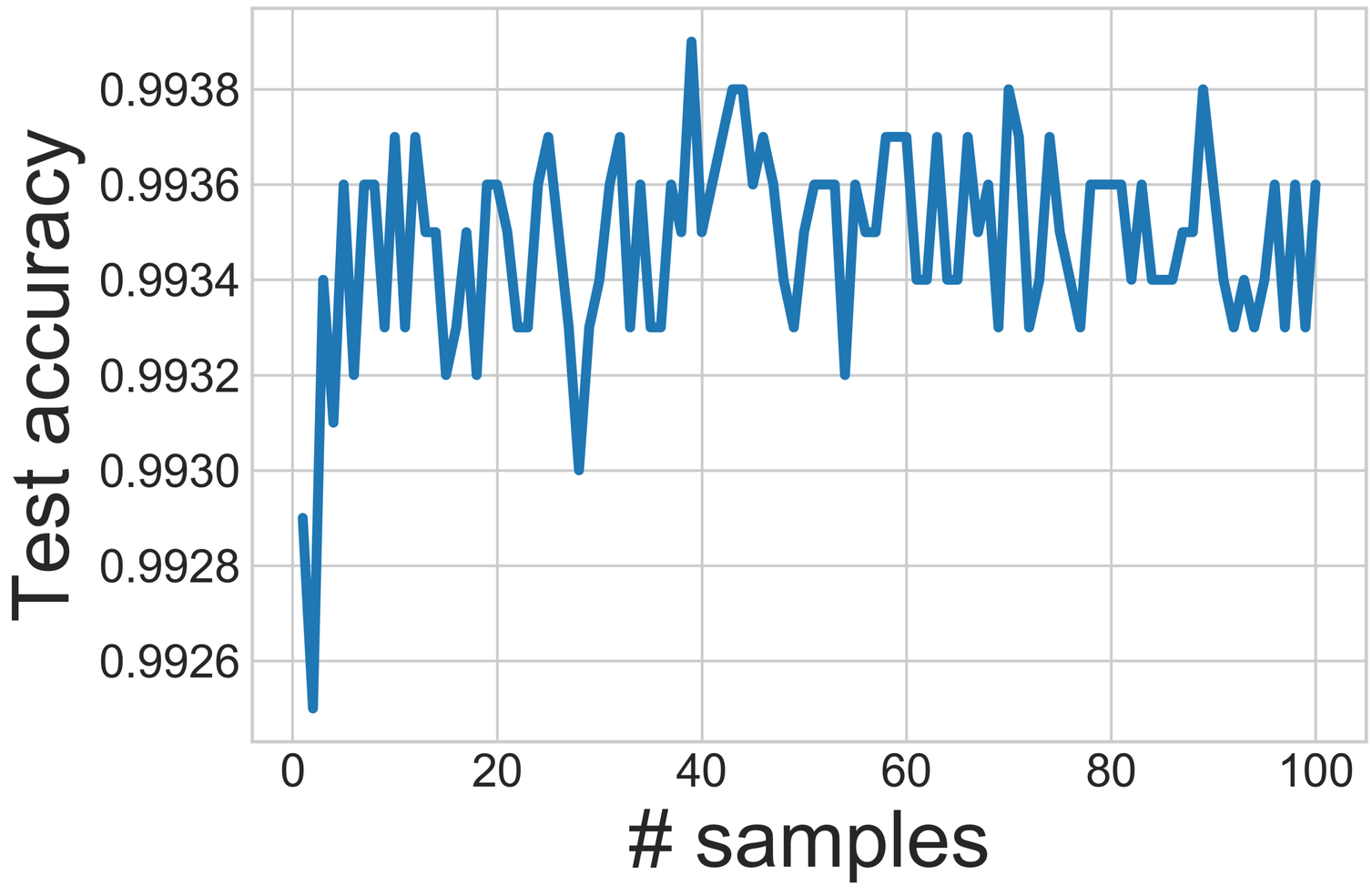
- Higher quality

(models compensate each other's errors)

- Better uncertainty estimation

Average SoftMax outputs across several samples

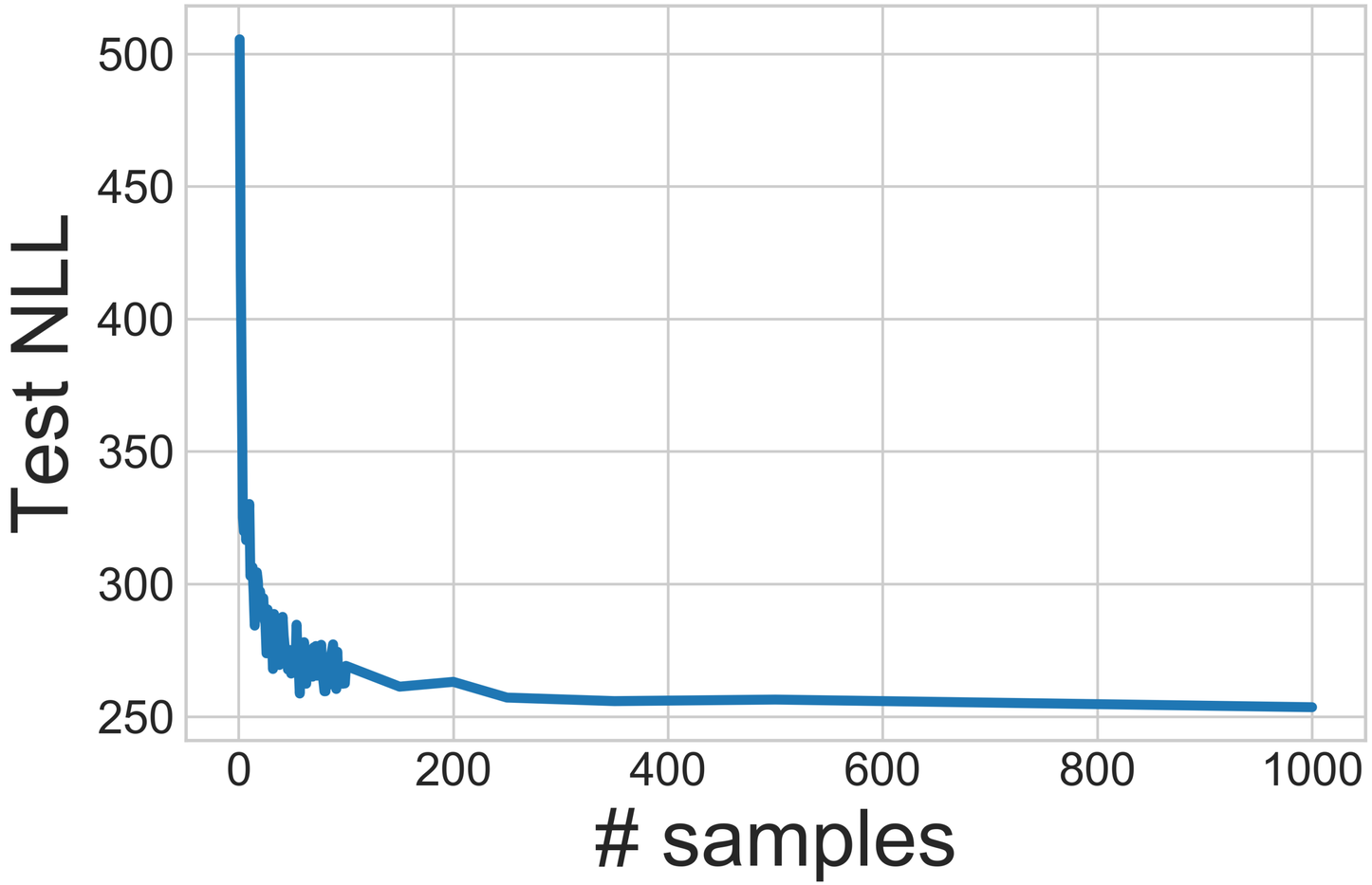
Ensembling



Accuracy quickly saturates

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

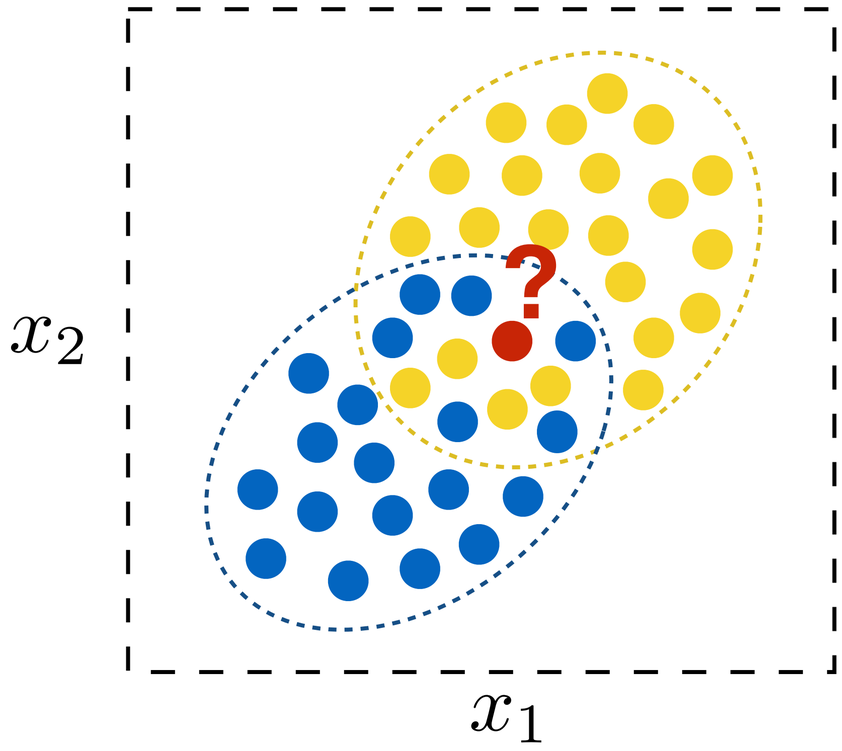
Ensembling



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

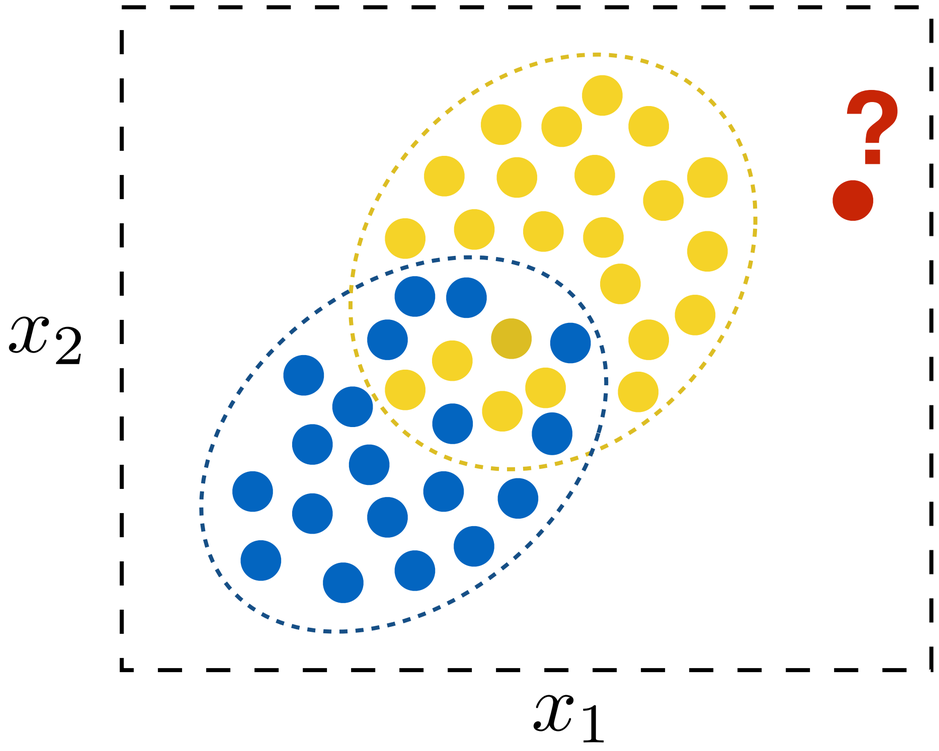
Two sources of uncertainty in classification

Data Uncertainty
(Aleatoric)



uncertain **in-domain** point

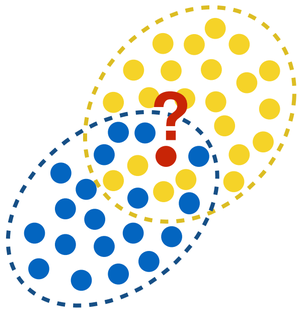
Model Uncertainty
(Epistemic)



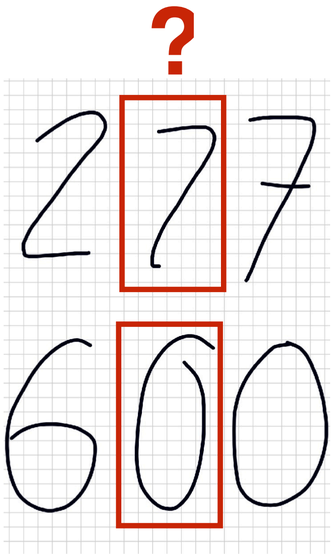
out-of-domain point

Two sources of uncertainty in classification

Data Uncertainty (Aleatoric)



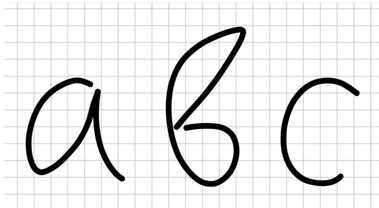
Overlapping classes:

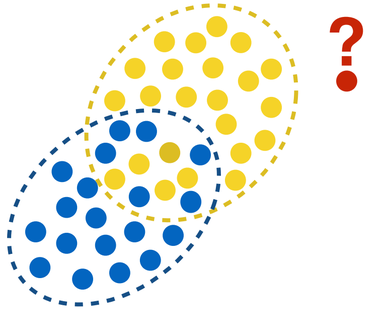


Examples

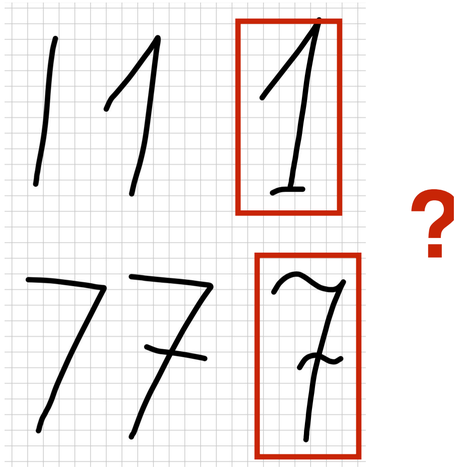
Model Uncertainty (Epistemic)

Unseen classes:





Unseen variations of known classes:



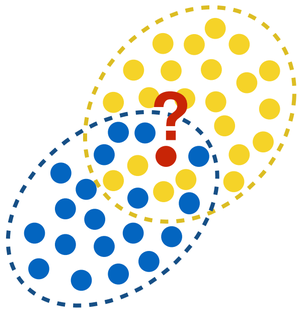
Inspired by Andrey Malinin's presentation, http://mi.eng.cam.ac.uk/~am969/

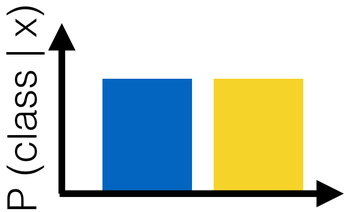
Two sources of uncertainty in classification

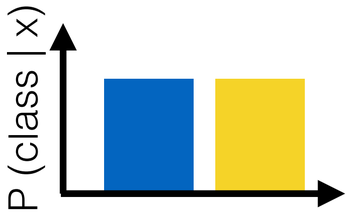
Data Uncertainty (Aleatoric)

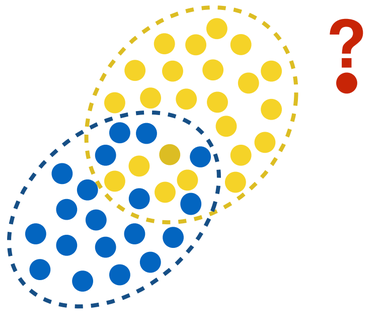
Model Uncertainty (Epistemic)

What do we expect from model?







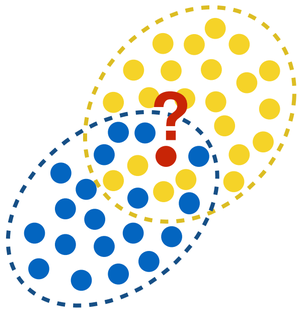


Inspired by Andrey Malinin's presentation, http://mi.eng.cam.ac.uk/~am969/

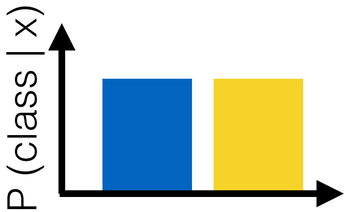
Two sources of uncertainty in classification

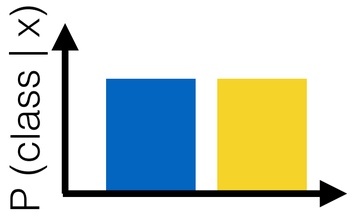
Data Uncertainty (Aleatoric)

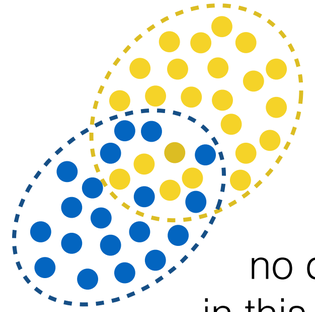
Model Uncertainty (Epistemic)



What do we expect from model?

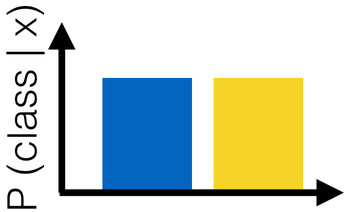


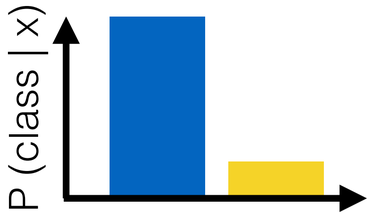




no data
in this region

What does single model output?





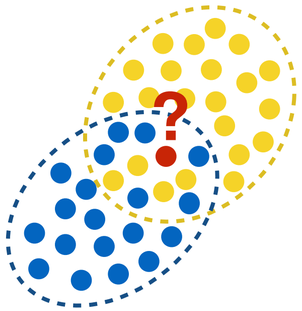
undefined
model
behavior

Inspired by Andrey Malinin's presentation, http://mi.eng.cam.ac.uk/~am969/

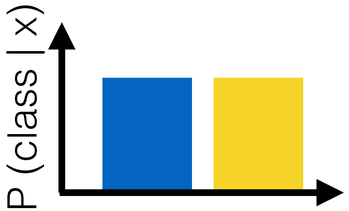
Two sources of uncertainty in classification

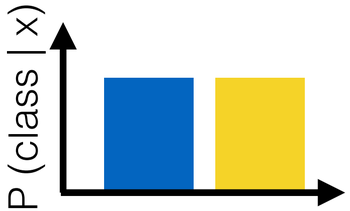
Data Uncertainty (Aleatoric)

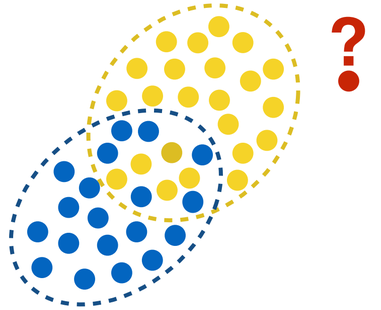
Model Uncertainty (Epistemic)



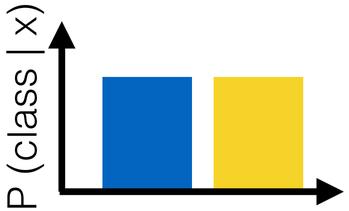
What do we expect from model?

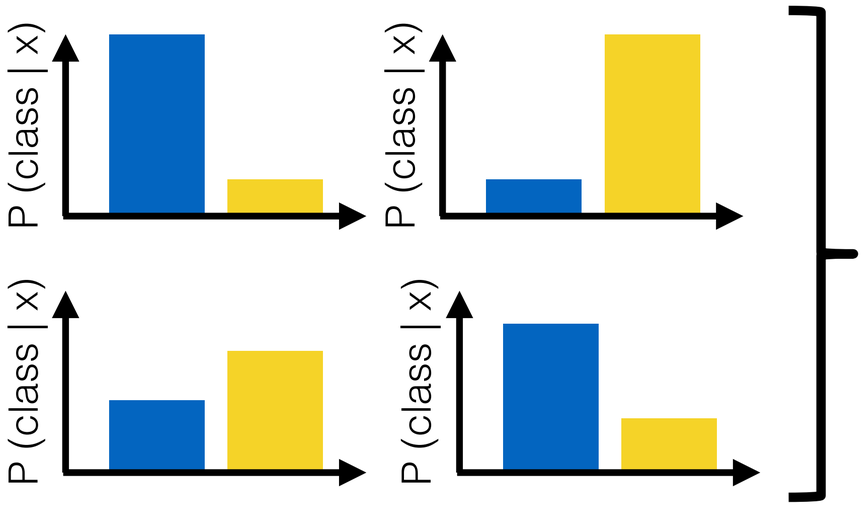




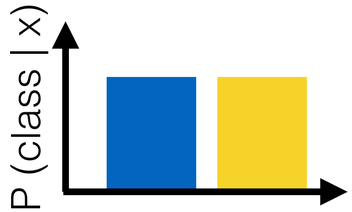


What does single model output?





ensemble



Inspired by Andrey Malinin's presentation

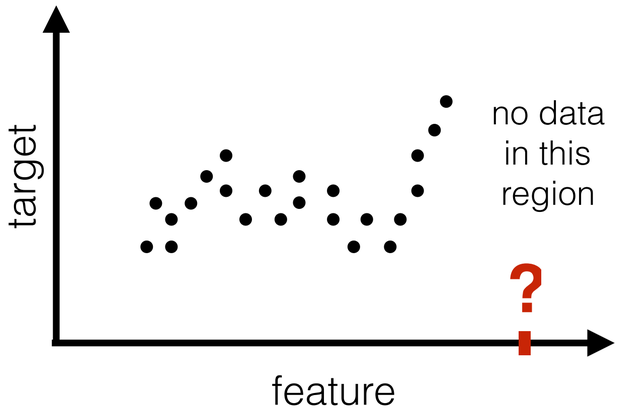
Two sources of uncertainty in regression

Data Uncertainty (Aleatoric)

- **Example**: noisy train data: same (or close) inputs with different target value
- **Solution**: model not only mean but also std

$$Loss = \frac{|y - \hat{y}|}{2\hat{\sigma}^2} + \log \hat{\sigma}$$

Model Uncertainty (Epistemic)

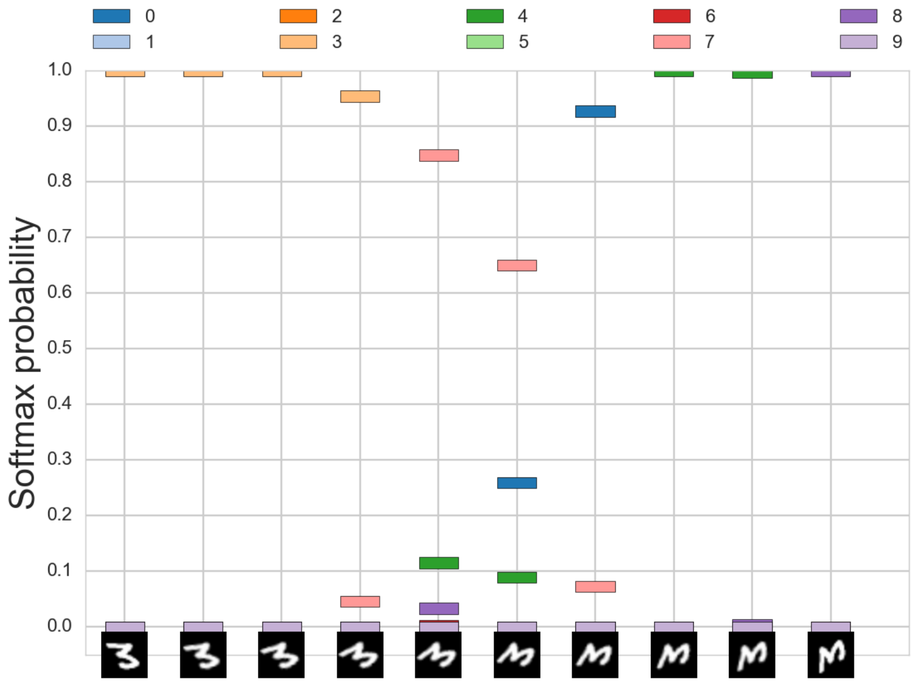


- **Example**: out-of-domain test data: undefined model behavior
- **Solution**: ensembling

https://alexgkendall.com/computer_vision/bayesian_deep_learning_for_safe_ai/

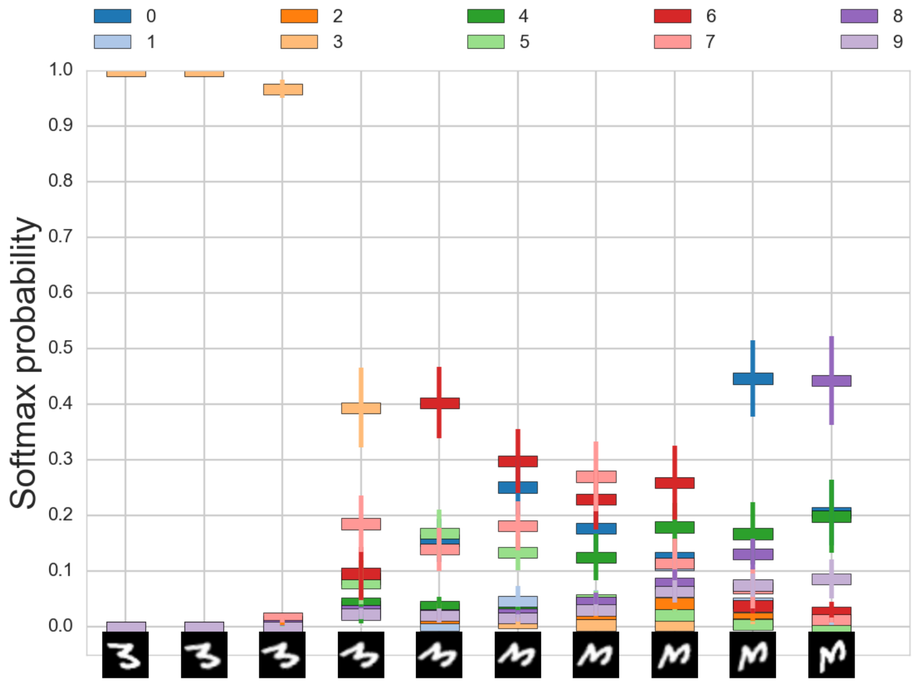
Uncertainty in classification: experiment

Deterministic NN



(a) LeNet with weight decay

Bayesian NN



(b) LeNet with multiplicative formalizing flows

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

Why go Bayesian?

A principled framework with many useful applications

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

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

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!

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

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

Why go Bayesian?

A principled framework with many useful applications

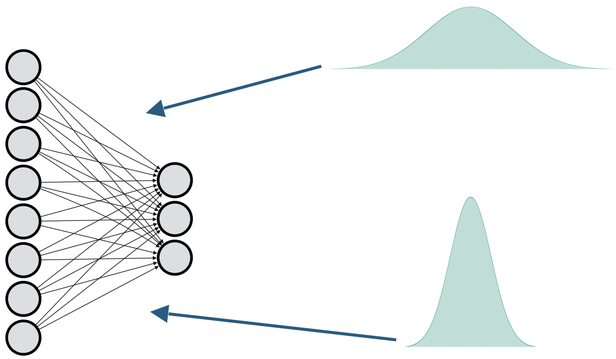
- Regularization ✓
- Ensembling ✓
- Uncertainty estimation ✓
- On-line / continual learning ✓
- Automatic hyperparameter choice ✓
- Different prior ⇒ different properties of the network (examples later)

Plan of the first part: Bayesian neural networks

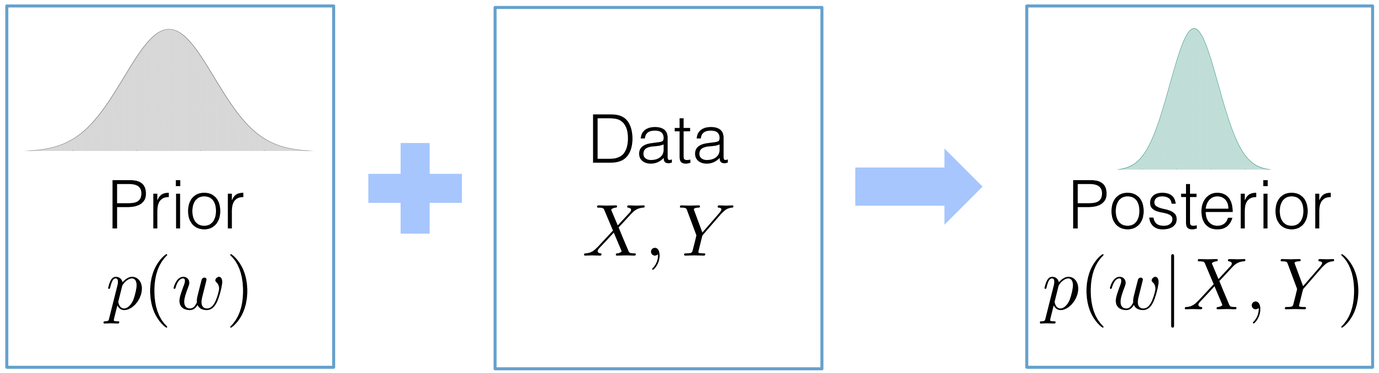
- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- Example

Training Bayesian neural networks

Stochastic weights:



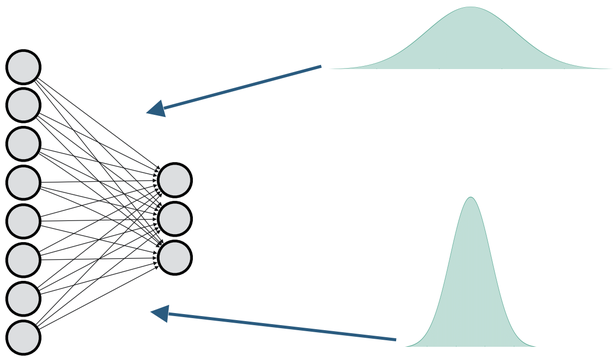
Bayesian Inference:



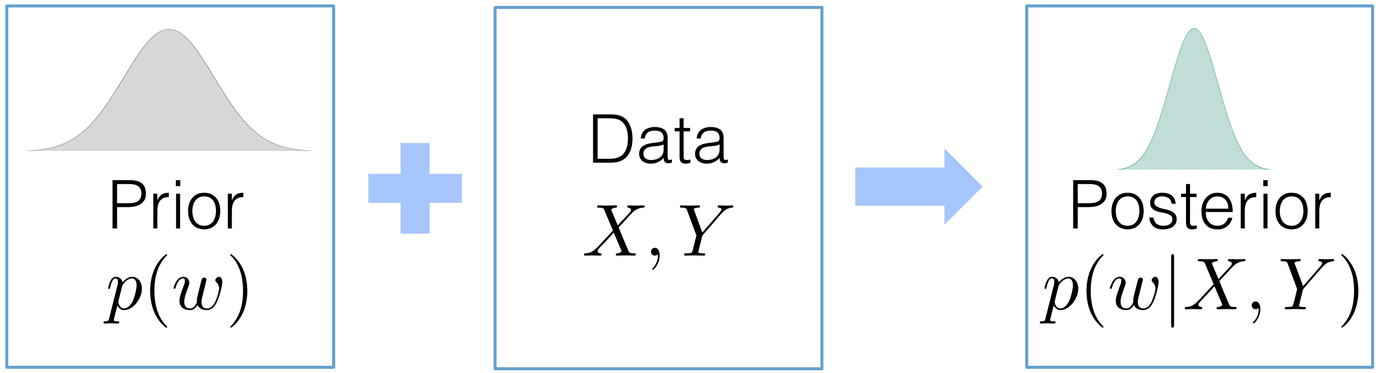
$$p(w|X,Y) = \frac{p(Y|X,w)p(w)}{\int p(Y|X,\tilde{w})p(\tilde{w})d\tilde{w}}$$

Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:

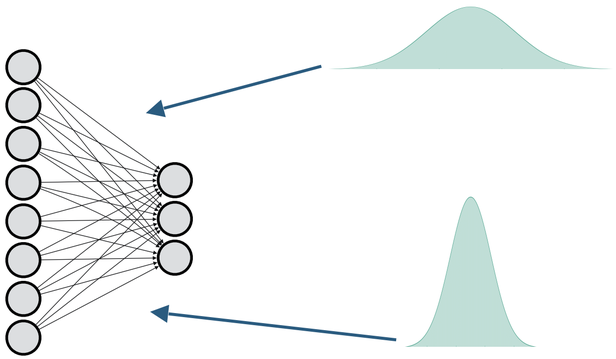


Posterior is intractable in neural networks ➡ approximate it with $q(w|\lambda)$:

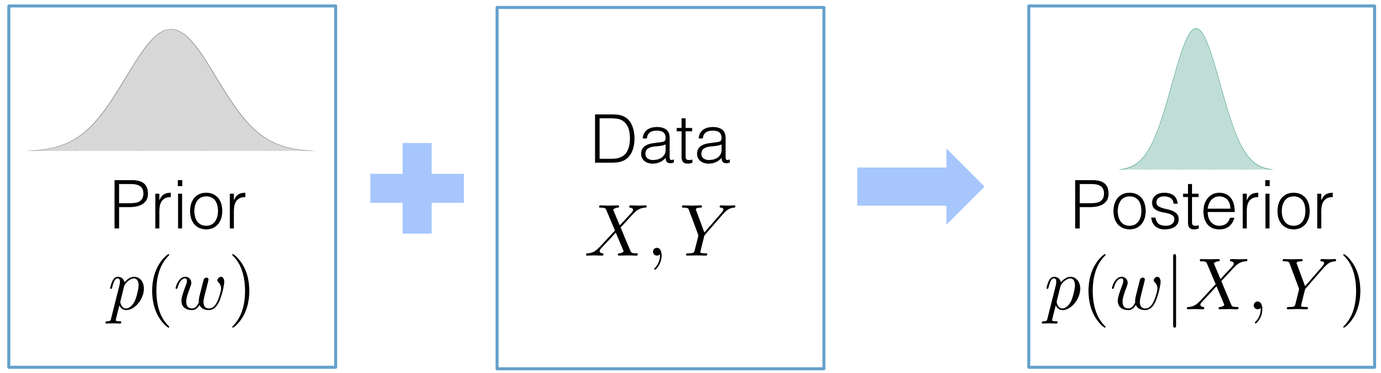
$$KL(q(w|\lambda)||p(w|X,y)) \rightarrow \min_{\lambda}$$

Training Bayesian neural networks

Stochastic weights:



Bayesian Inference:



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

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

Data term: popular examples of likelihood function

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

Classification: cross entropy:

$$\log p(y|x, w) = \log \underbrace{[\text{softmax}(NN(x, w))]_y}_{\substack{\text{probabilities of all classes} \\ \text{(output of neural network)}}}$$

selecting needed class

Regression: mean squared error

$$\log p(y|x, w) = \log \mathcal{N}(NN(x, w), \sigma^2) \overset{\substack{\text{assume } \sigma \text{ is} \\ \text{known and fixed}}}{\propto} |y - NN(x, w)|$$

KL-divergence: note there are two of them!

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

constant | **we optimize** | **non-negative we wish to optimize**

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

- approx. posterior || prior
- tractable!
- acts like a regularizer

- approx. posterior || true posterior
- intractable :(
- represents our ideal objective

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 the KL-divergence?
- How to estimate the expectation?

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 the KL-divergence?
- **How to estimate the expectation?**

Doubly stochastic variational inference

How to estimate the expectation?

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

???

Doubly stochastic variational inference

How to estimate the expectation?

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

???

sample mini-batch of the training objects

sample weights from q

Doubly stochastic variational inference

How to estimate the expectation?

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

???

sample mini-batch of the training objects

sample weights from

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

Doubly stochastic variational inference

How to estimate the expectation?

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

sample mini-batch of the training objects

sample weights from

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

But wait,

how to differentiate?

Doubly stochastic variational inference

How to estimate the expectation?

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

$\nabla_\lambda ?$

sample mini-batch of the training objects

sample weights from

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

But wait,

how to differentiate?

Gradient of expectation

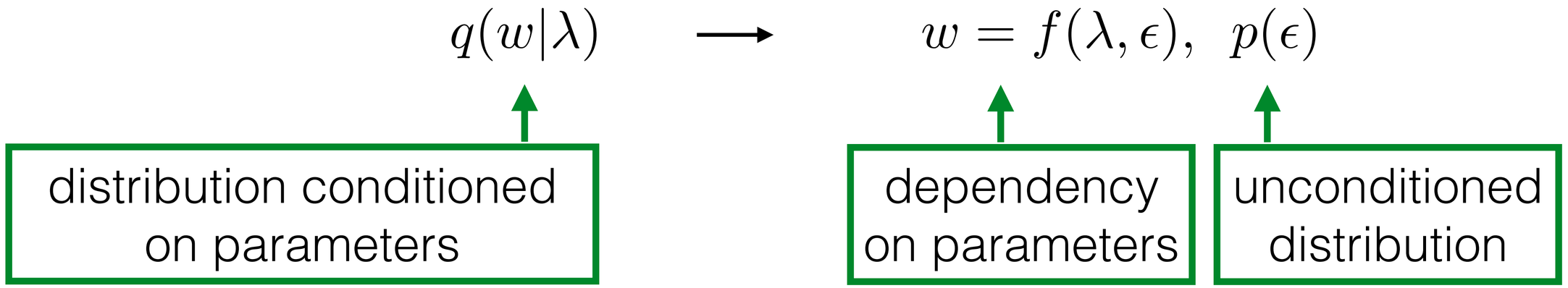
$$\nabla_\lambda \mathbb{E}_{q(w|\lambda)} \log p(y^i|x^i, w) = \nabla_\lambda \int q(w|\lambda) \log p(y^i|x^i, w) dw =$$

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

Not an expectation over q anymore!

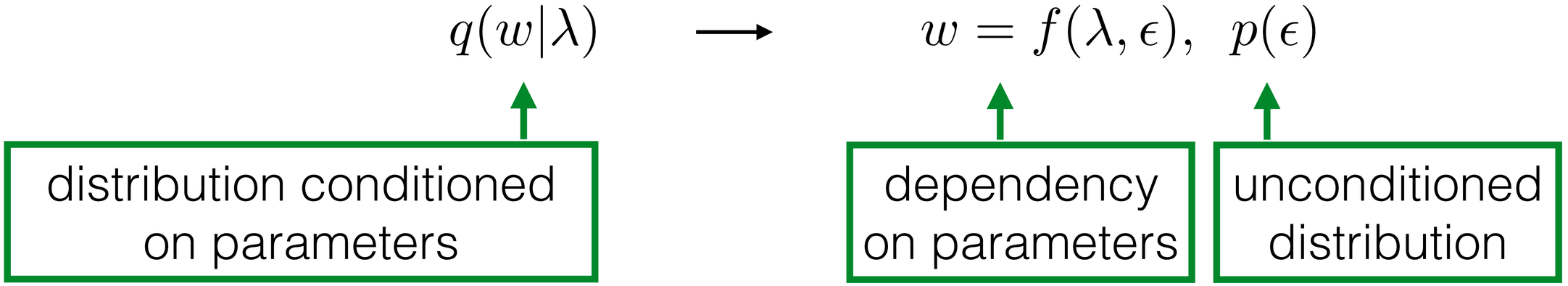
(One possible) solution: **reparametrization trick**

Reparametrization trick



Diederik P Kingma, Max Welling . Auto-Encoding Variational Bayes. ICLR 2013

Reparametrization trick



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

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

Diederik P Kingma, Max Welling . Auto-Encoding Variational Bayes. ICLR 2013

Reparametrization trick: examples

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

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

Reparametrization trick: examples

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

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

noise on weights!

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 the KL-divergence?
- **How to estimate the expectation?** **Doubly stochastic VI + reparametrization trick**

Plan of the first part: Bayesian neural networks

- Advantages of using Bayesian neural networks
- Training Bayesian neural networks
- Example

Example: binary dropout

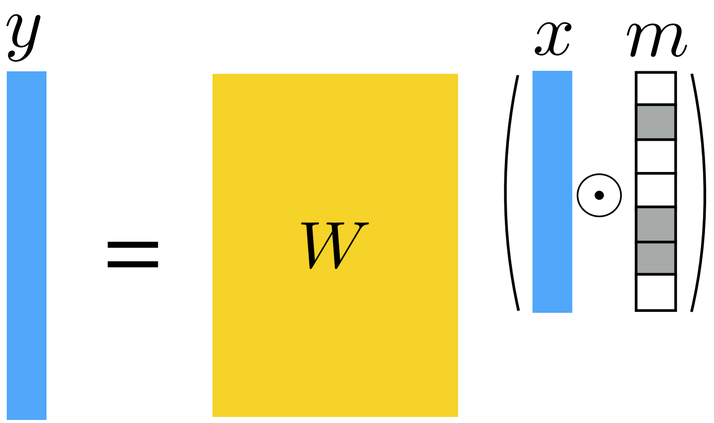
Linear layer: $x_2 = Wx_1,\ W$ — weight matrix

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

Example: binary dropout

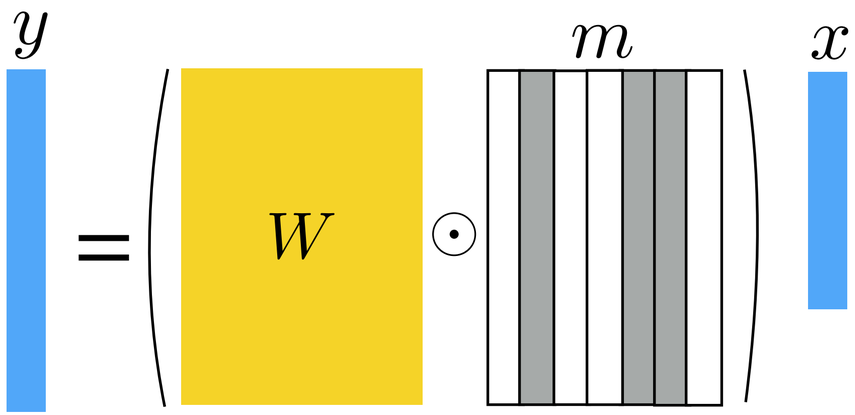
Linear layer: $x_2 = W x_1, \; W$ — weight matrix

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



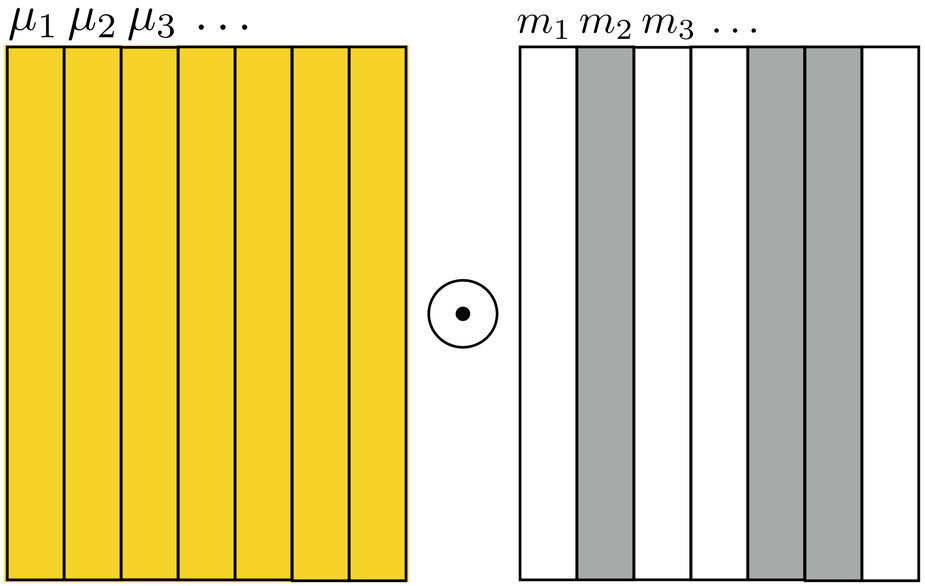
$$x_{2,j} = \sum_i w_{ij} x_{1,i} m_i$$

$$\Leftrightarrow$$



Example: binary dropout

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



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

reparametrization

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

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

p is a fixed hyperparameter!

Gal, Yarin and Ghahramani, Zoubin.
Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ICML 2016

Example: binary dropout

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

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

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

p is a fixed hyper-parameter!

Approximate KL-divergence: $KL(q(W|\mu)||p(W)) \approx \alpha||\mu||_2^2$ (see paper for the proof)

Estimate expectation: Doubly stochastic VI + reparametrization trick

Gal, Yarin and Ghahramani, Zoubin.
Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ICML 2016

Example: binary dropout, putting all together

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

$$\sum_{i=1}^{N} \mathbb{E}_{p(m)} \log p(y^i|x^i, w = \mu \cdot \text{diag}(m)) - ||\mu||_2^2 \rightarrow \max_{\mu}$$

sampling dropout masks

usual feed-forward pass & loss

applying dropout

applying L_2 regularization

Usual NN training with binary dropout

Gal, Yarin and Ghahramani, Zoubin.
Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ICML 2016

Example: binary dropout, Bayesian benefits

$$\sum_{i=1}^{N} \mathbb{E}_{p(m)} \log p(y^i|x^i, w = \mu \cdot \text{diag}(m)) - ||\mu||_2^2 \to \max_{\mu}$$

Usual NN training with binary dropout

Key messages of this example:

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

Gal, Yarin and Ghahramani, Zoubin.
Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ICML 2016

Bayesian neural networks: summary

- A lot of advantages: ensembling, uncertainty estimation, etc
- To train BNN, one should optimize ELBO using DSVI & RT
- Four steps towards a particular method
- Using binary dropout means being Bayesian

Next: detailed discussion of a method that compresses neural networks

5-min break

Plan of the second part: Bayesian sparsification

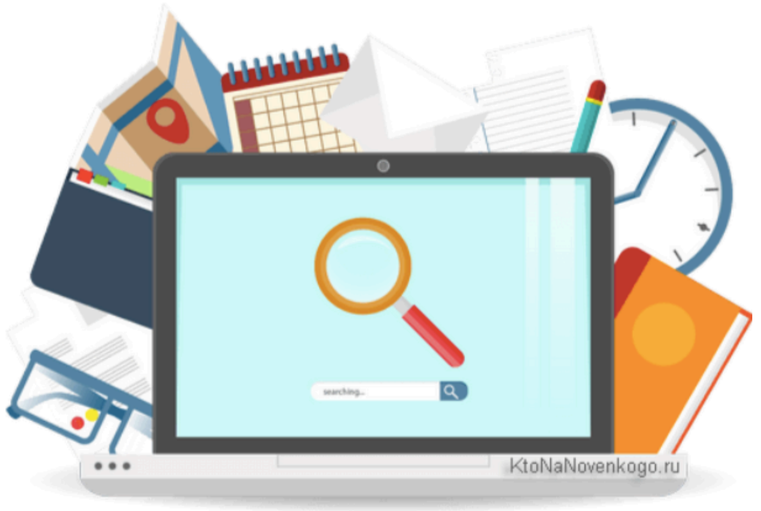
- Sparsification: what and why
- Sparse variational dropout
- Practical assignment: implementation of SparseVD (seminar)
- Model enhancements (seminar)

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! 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

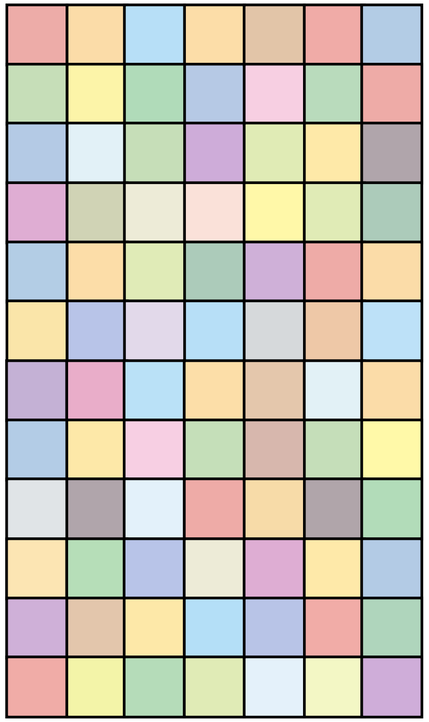
Compression of neural networks

- 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**

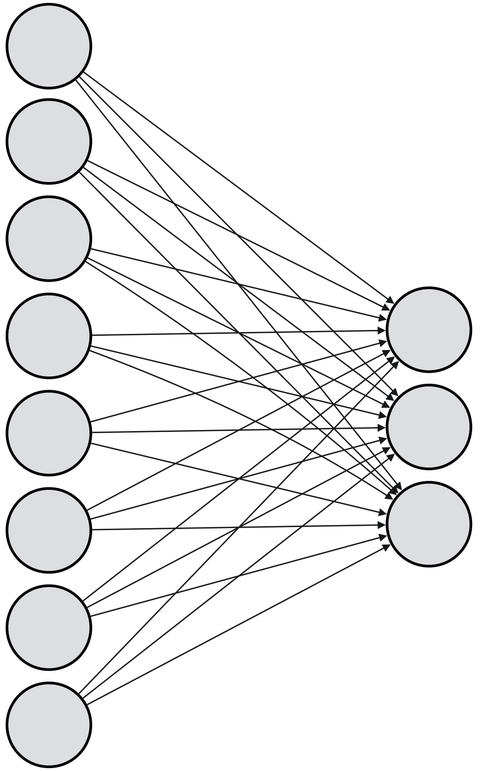
Methods for neural network compression:

- Quantization
- Matrix factorizations
- Sparsification

Neural network

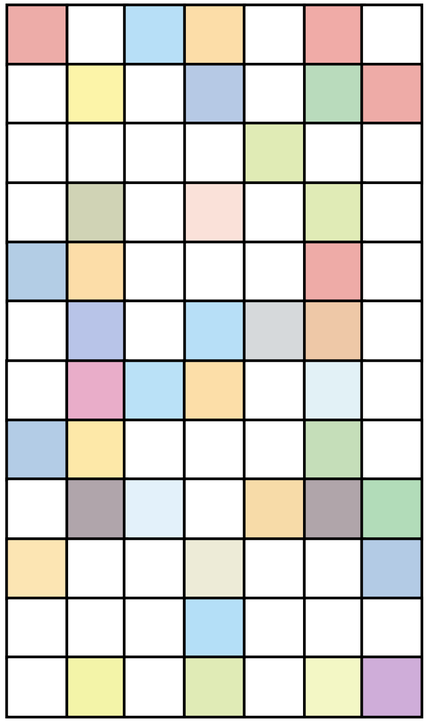


Weight matrix W

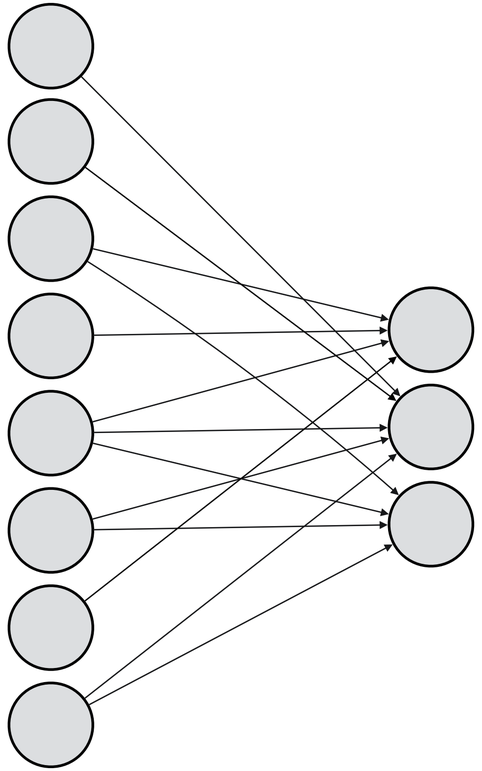


Computational graph

Sparse neural network



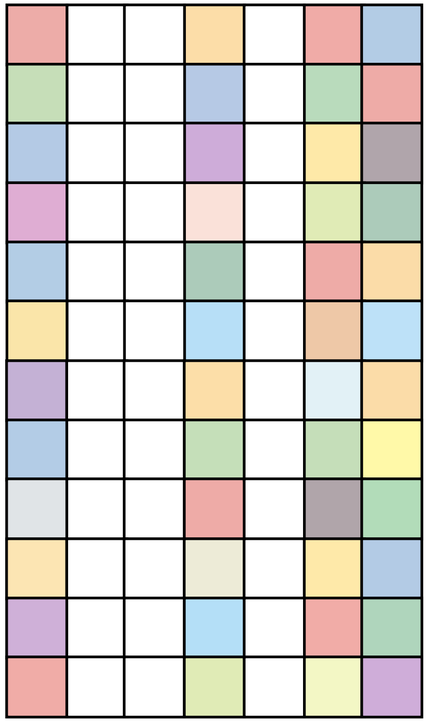
A lot of weights
set to zero



Weight matrix W

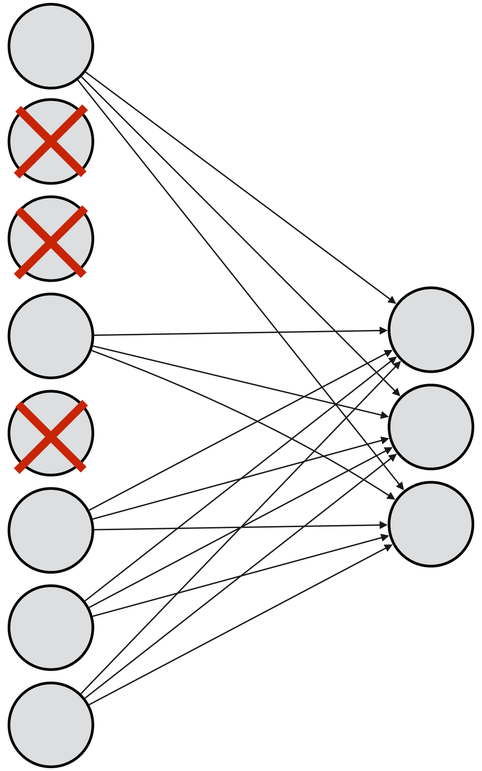
Computational graph

Structured sparsity



Weight matrix W

No outgoing edges
⇒ remove neuron



Computational graph

Sparsification of neural networks

Benefits:
- sparse matrices $\Rightarrow$ less memory consumption
- structured sparsification $\Rightarrow$ faster testing stage (prediction)
- regularization
- a bit more interpretable model

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)
  - Prune based on threshold T / prune percent of weights
  - Propagate gradients through zero weights or not
  - Prune from scratch / train - prune - retrain
  - …

huge number of hyperparameters

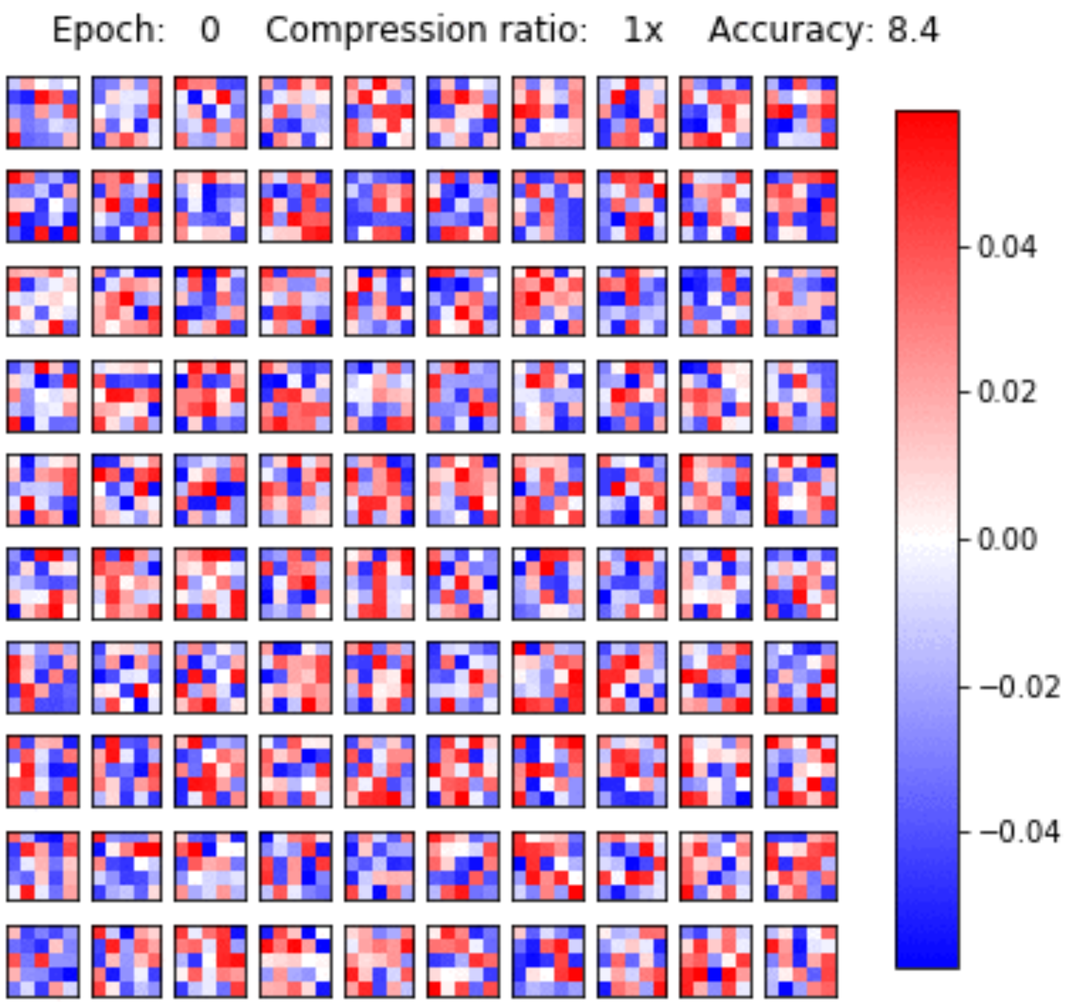
Two popular frameworks for sparsification

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!

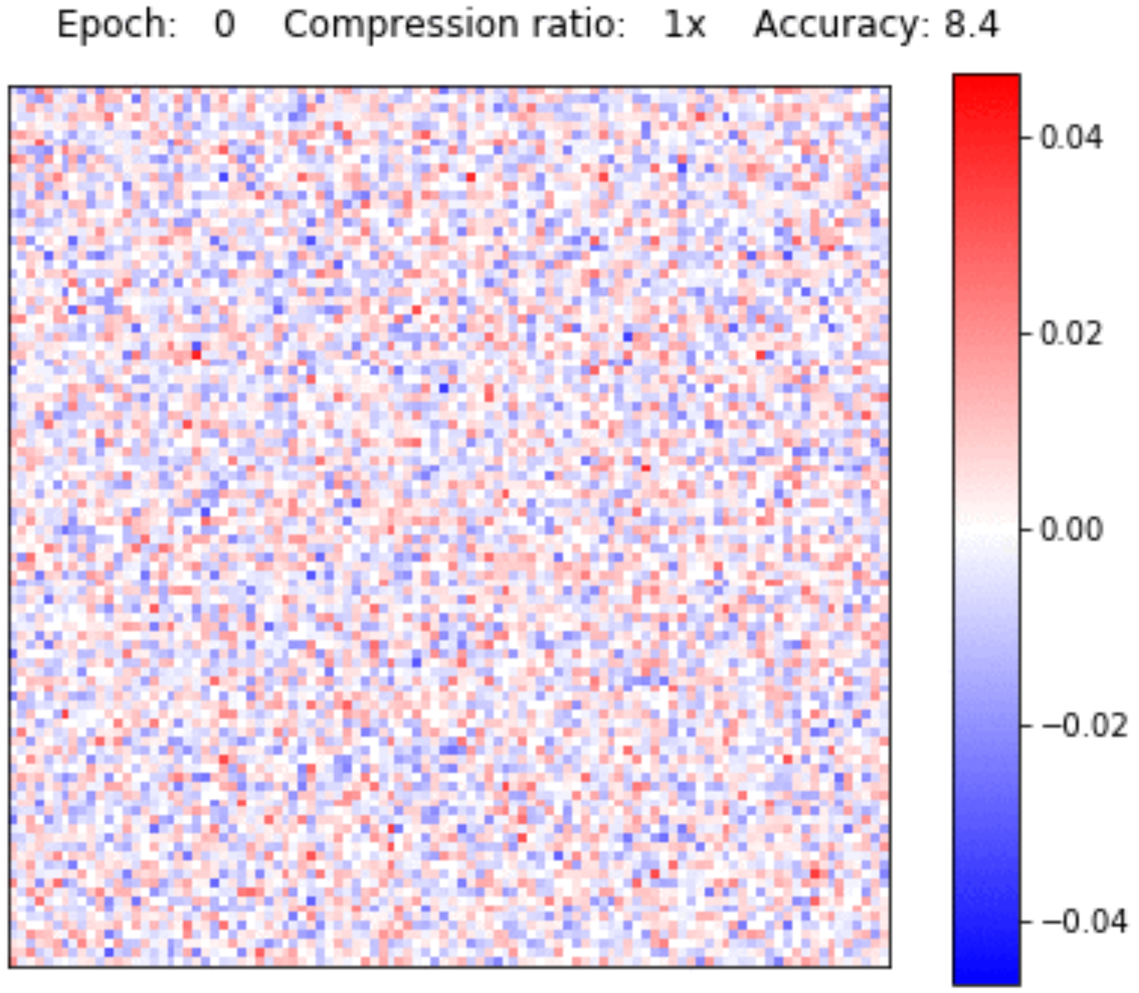
Plan of the second part: Bayesian sparsification

- Sparsification: what and why
- Sparse variational dropout
- Practical assignment: implementation of SparseVD (seminar)
- Model enhancements (seminar)

Sparse variational dropout: visualization

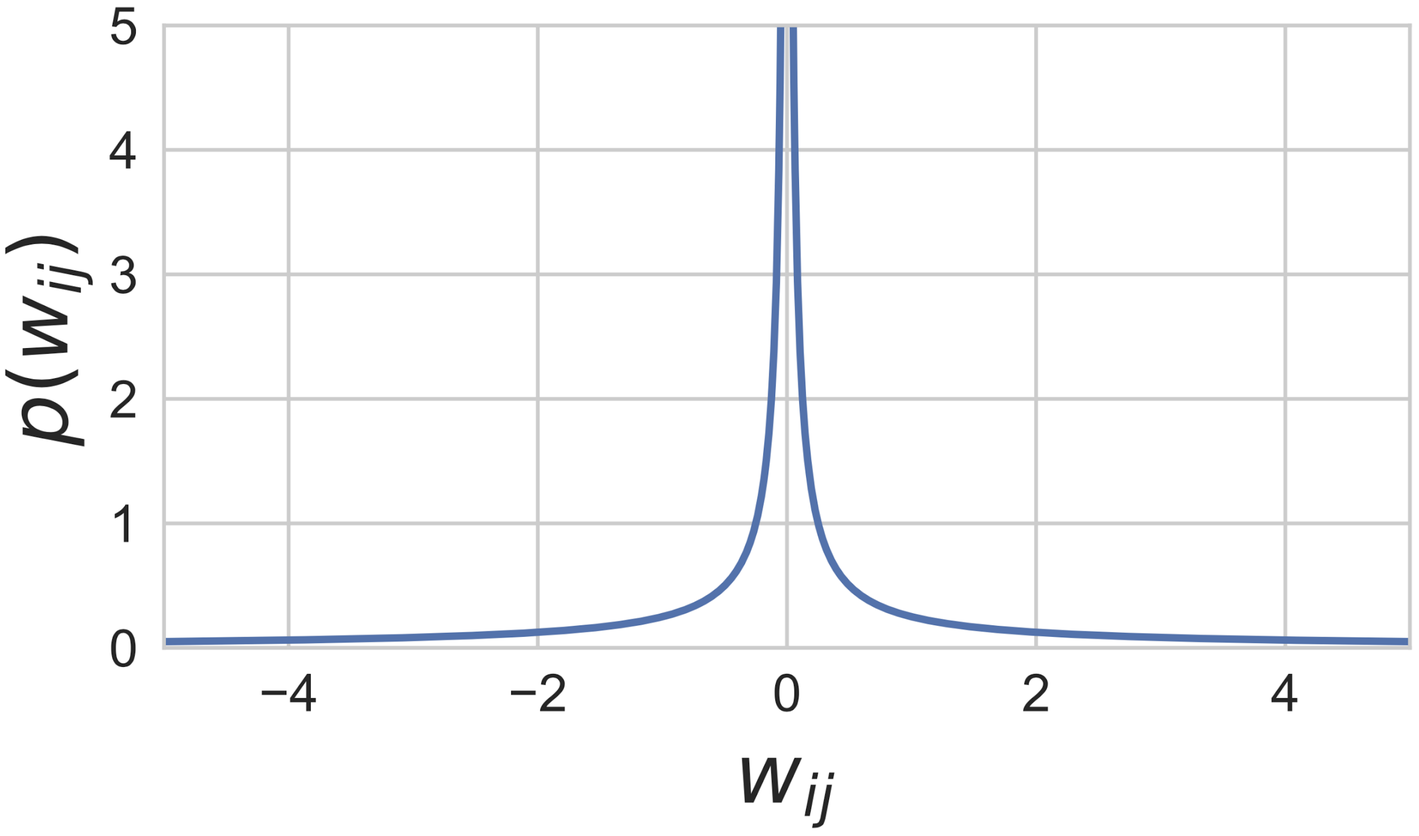


LeNet-5: convolutional layer



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

Log-uniform prior distribution

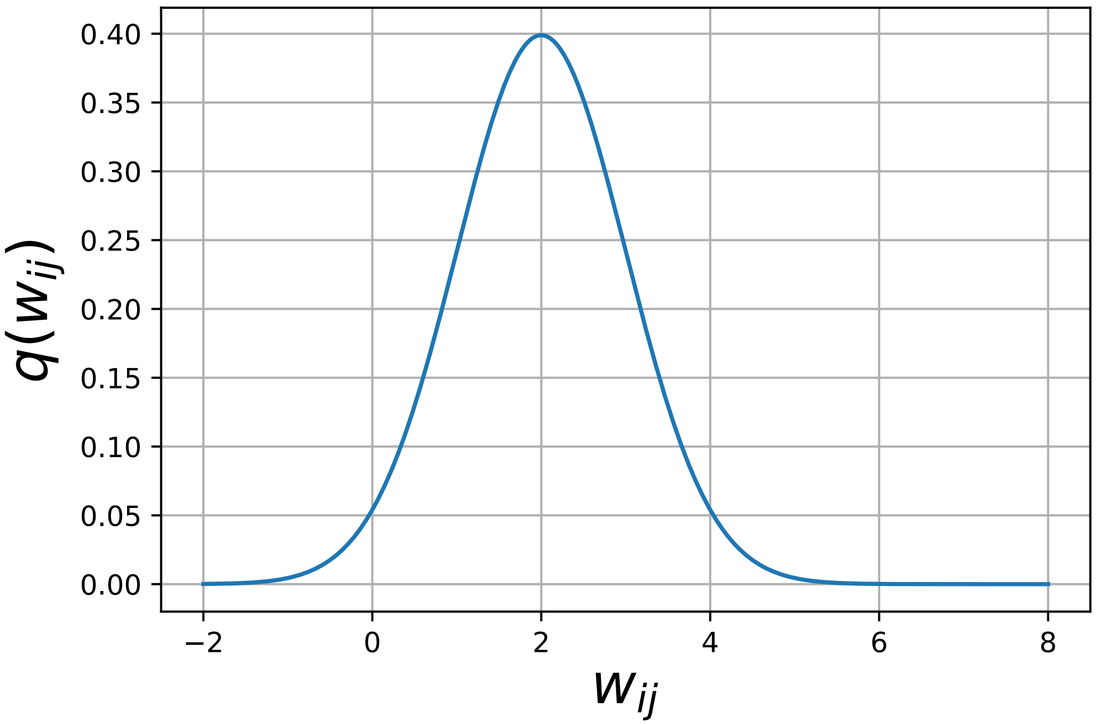


$$p(w_{ij}) \propto \frac{1}{|w_{ij}|}$$

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)



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

Reparametrization trick:

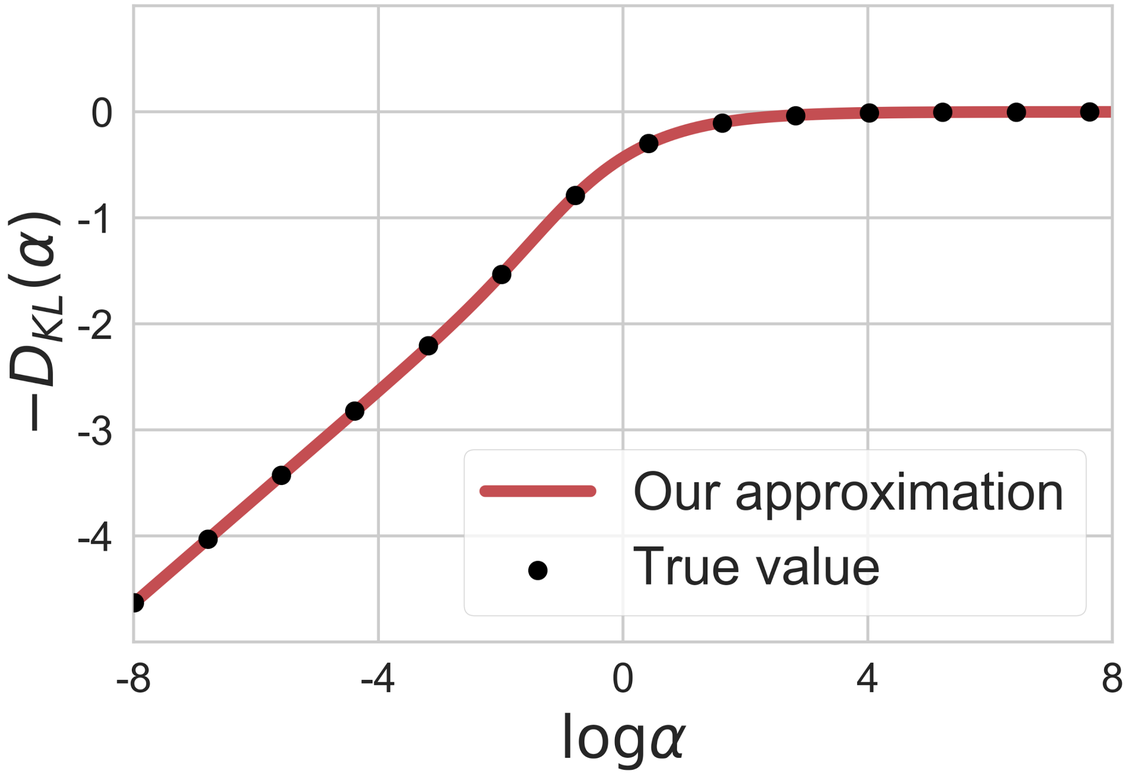
$$\hat{w}_{ij} = \mu_{ij} + \hat{\epsilon}_{ij}\sigma_{ij}, \quad \hat{\epsilon}_{ij} \sim \mathcal{N}(0, 1)$$

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 α_{ij}
- Favors large α_{ij} $\Rightarrow$

  removing noisy weights

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

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

Doubly stochastic variational inference

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

???

Analytical approximation

Doubly stochastic variational inference

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

Data term
???

Regularizer
Analytical approximation

sample mini-batch
of the training objects

sample weights from q

Estimating expectation in ELBO

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

Data term
Sample weights from q

Regularizer
Analytical approximation

Estimating expectation in ELBO

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

Data term
Sample weights from q

Regularizer
Analytical approximation

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

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

Data term
Sample weights from q

Regularizer
Analytical approximation

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! ($|w|\times$ mini-batch size)

Estimating expectation in ELBO

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

Data term
Sample weights from q

Regularizer
Analytical approximation

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

Estimating expectation in ELBO

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

Data term
Sample weights from q

Regularizer
Analytical approximation

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:

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

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

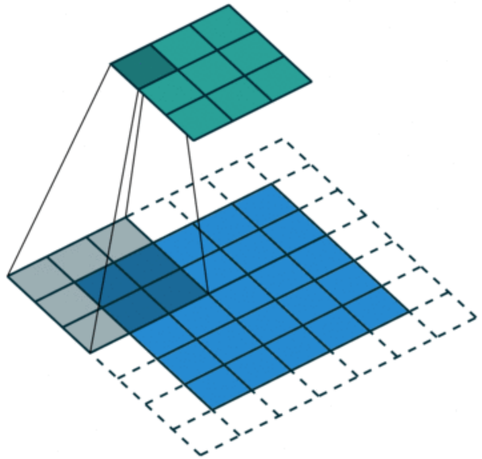
Predictions have **zero** correlation because there is one weight sample **per object**

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

LRT for convolutions

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

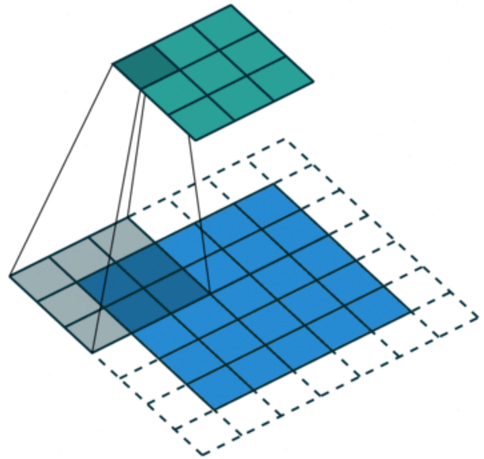


Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

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

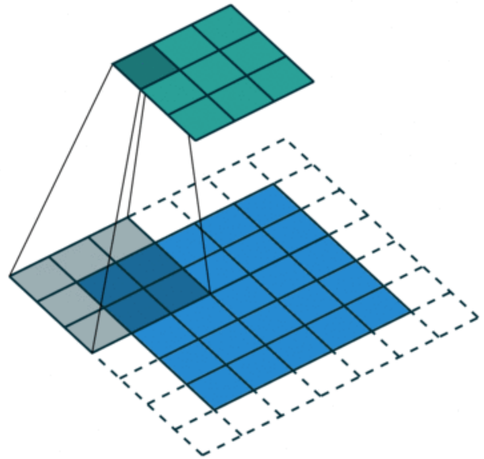


Diederik P. Kingma et al. Variational dropout and the local reparameterization trick. NIPS 2015

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:

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

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

Final algorithm (without LRT)

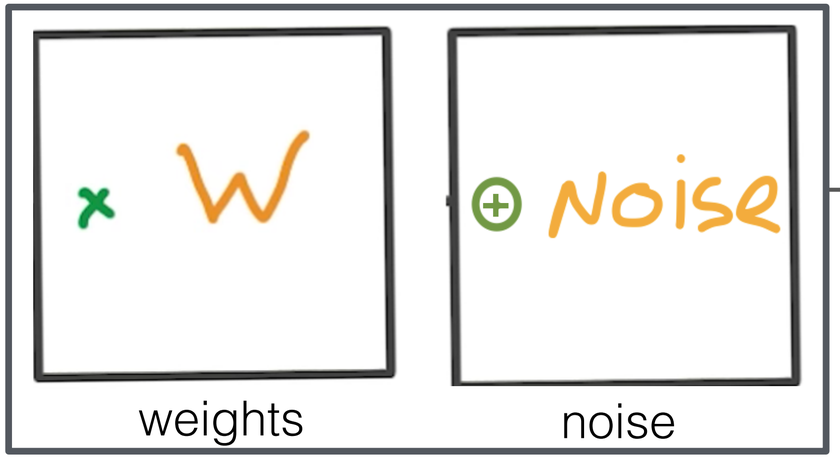
Training on a mini-batch X with labels Y:

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:

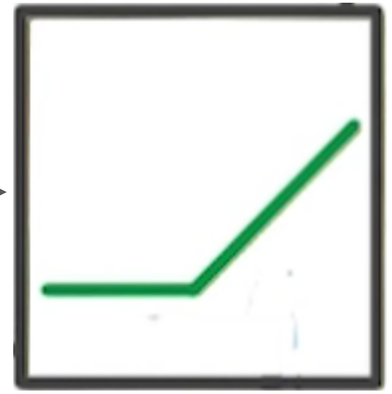
$$\nabla_{\mu, \log \sigma, b} \left(N \cdot \text{Loss}\left(Y, Y_{\text{pred}}\right) + \text{SparseReg}(\sigma/\mu) \right)$$

Forward pass with stochastic weights

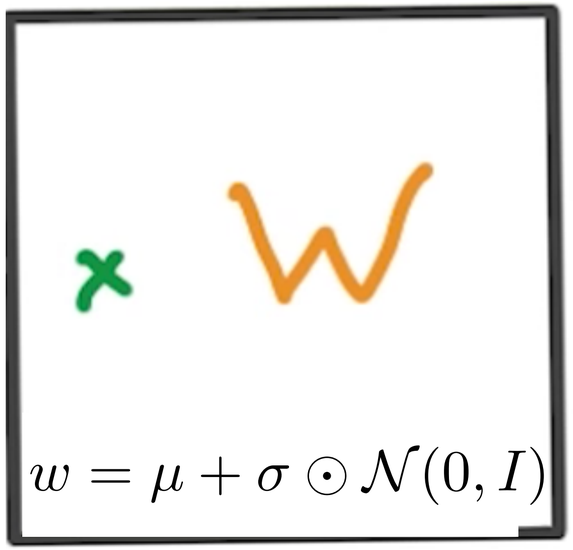
input



activations



nonlinearity



it does not change the mean value

additive noise

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

Slide from KirIII Neklyudov's presentation, deepbayes.ru/2018

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:

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

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:

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

Pruning after training:

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

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

signal-to-noise ratio

Final algorithm (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:

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

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

Automatic relevance determination

(other prior but very similar method)

- 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 method)

- 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 method)

- 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:

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

normal

normal

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

Automatic relevance determination

(other prior but very similar method)

- 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:

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

- Analytical optimization w. r. t. prior parameters τ_{ij} :

$$\tau_{ij}^* = (\mu_{ij}^2 + \sigma_{ij}^2)^{-1}$$

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

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

Comparing two models

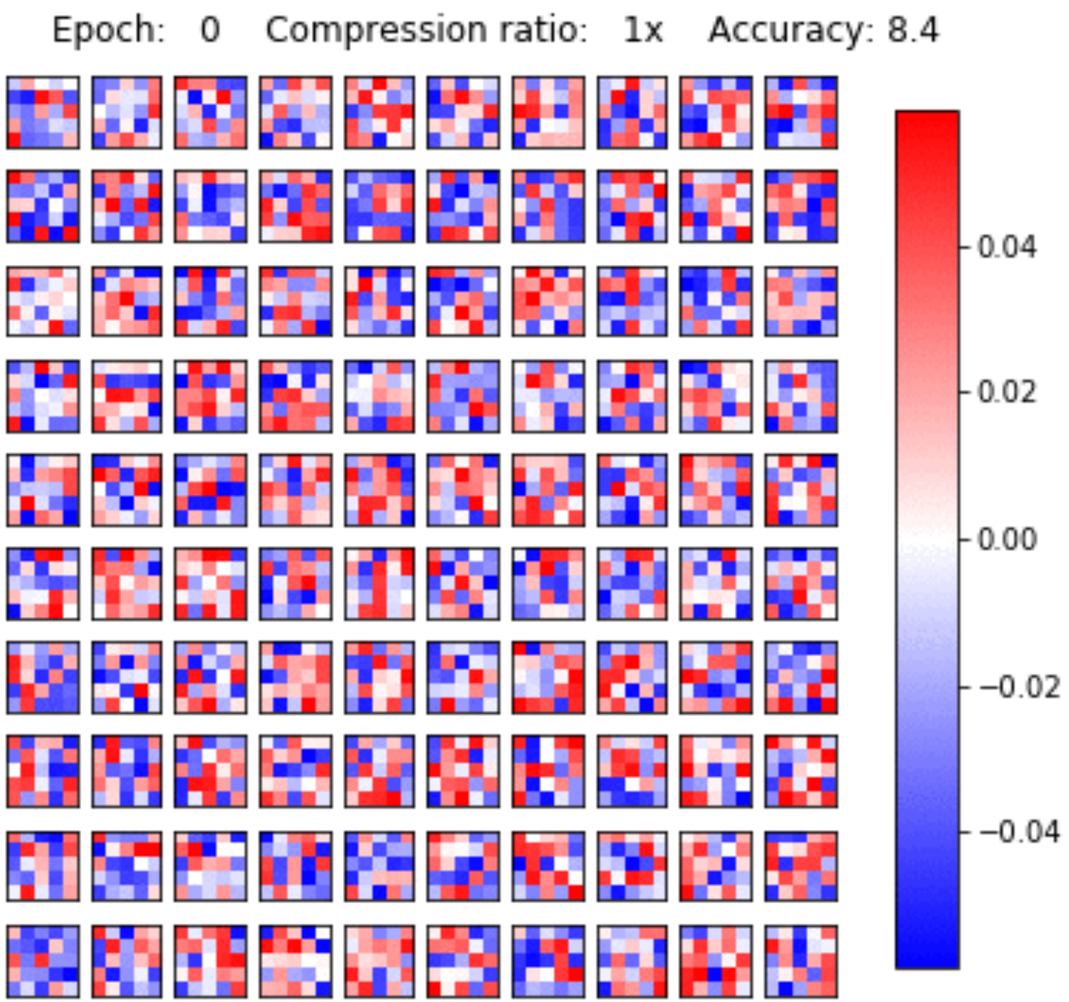
	SparseVD	**ARD**
Prior	$p(w_{ij}) \propto \frac{1}{\lvert w_{ij} \rvert}$	$p(w_{ij}\lvert\tau_{ij}) = \mathcal{N}(0, \tau_{ij}^{-1})$
Prior parameters	None	τ_{ij}
KL-divergence	$k_1\sigma(k_2 + k_3 \log \alpha_{ij}))- -0.5\log(1+\alpha_{ij}^{-1}) + \mathrm{C}$ bounded	$-0.5\log(1+\alpha_{ij}^{-1})$

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

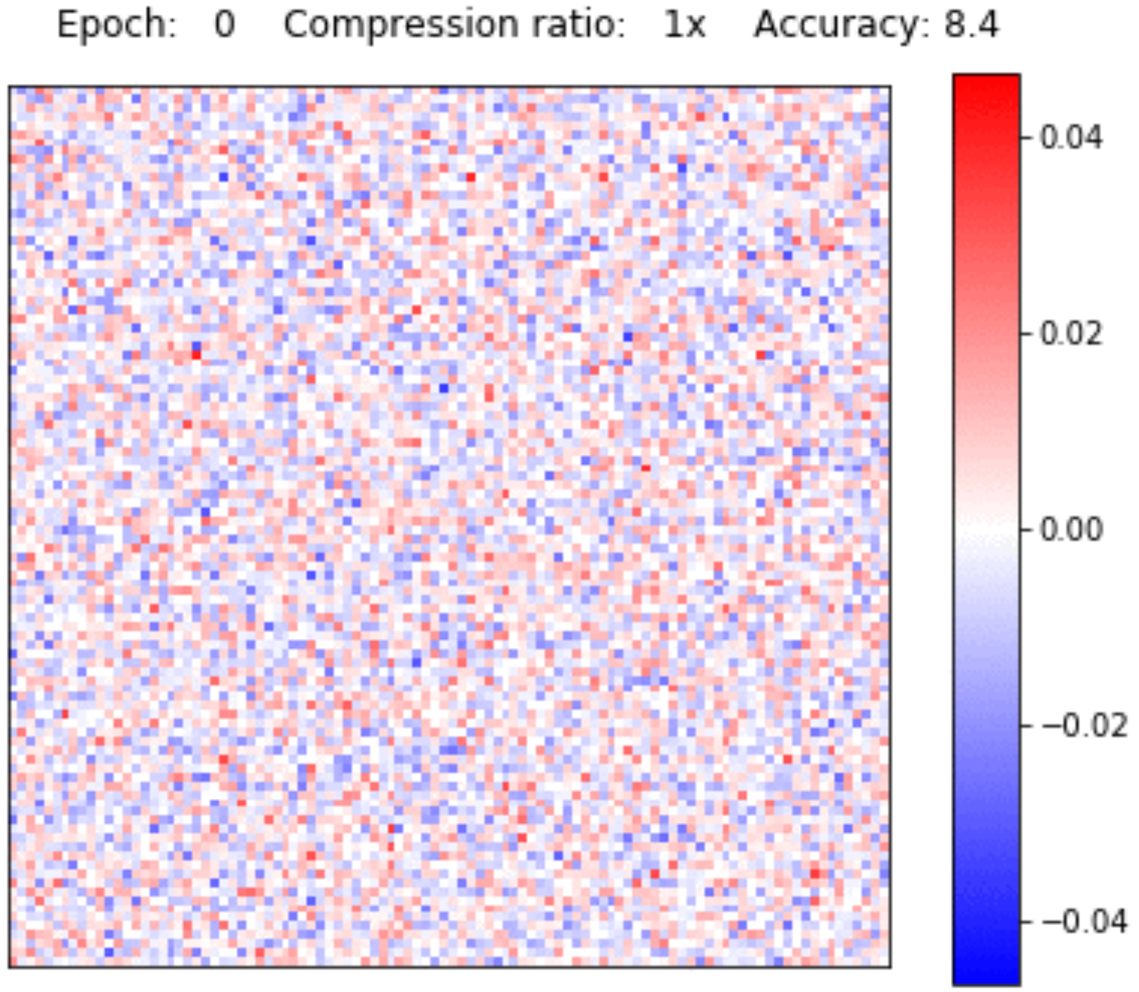
Training procedure is the same (approx. posterior, RT, LRT etc.)

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

Sparse variational dropout: visualization



LeNet-5: convolutional layer

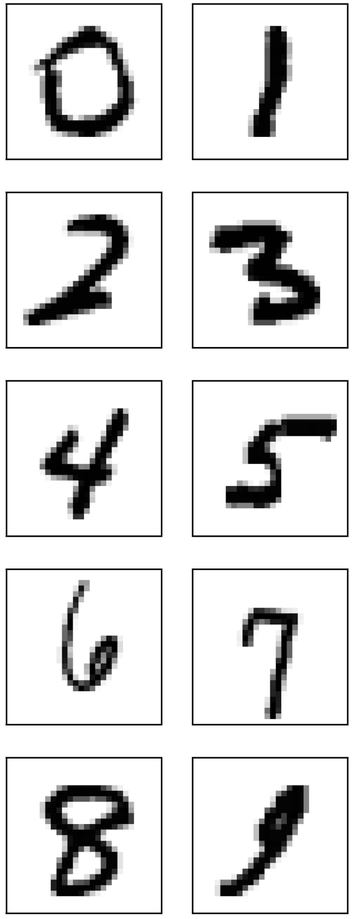


LeNet-5: fully-connected layer
(100 x 100 patch)

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

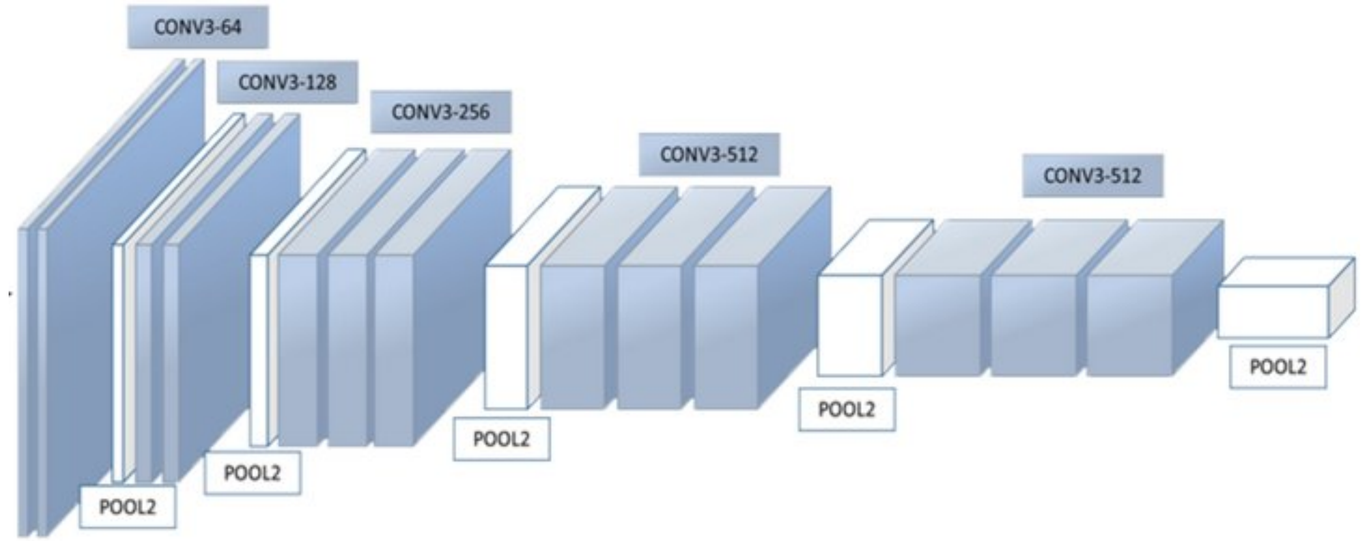
Fully Connected network: LeNet-300-100 **Convolutional network**: Lenet-5-Caffe

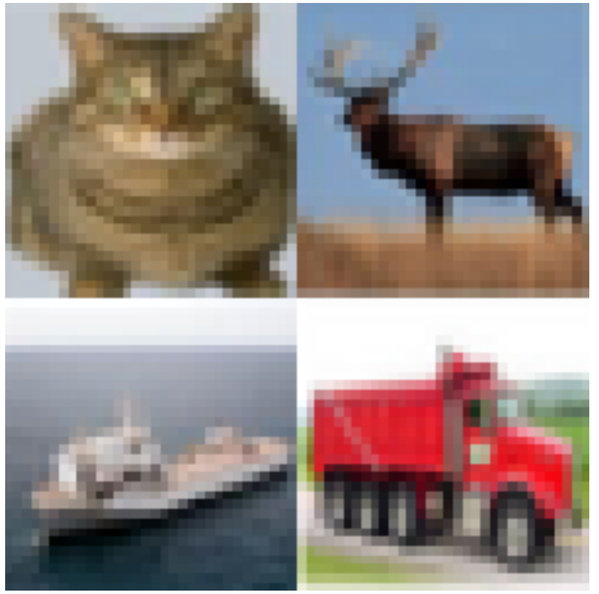
Network	Method	Error %	Sparsity per Layer %	$\frac{\lvert\mathbf{W}\rvert}{\lvert\mathbf{W}_{\neq 0}\rvert}$
LeNet-300-100	Original	1.64		1
	Pruning	1.59	$92.0 - 91.0 - 74.0$	12
	DNS	1.99	$98.2 - 98.2 - 94.5$	56
	SWS	1.94		23
(ours)	Sparse VD	1.92	$98.9 - 97.2 - 62.0$	**68**
LeNet-5-Caffe	Original	0.80		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**



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

VGG-like on CIFAR-10



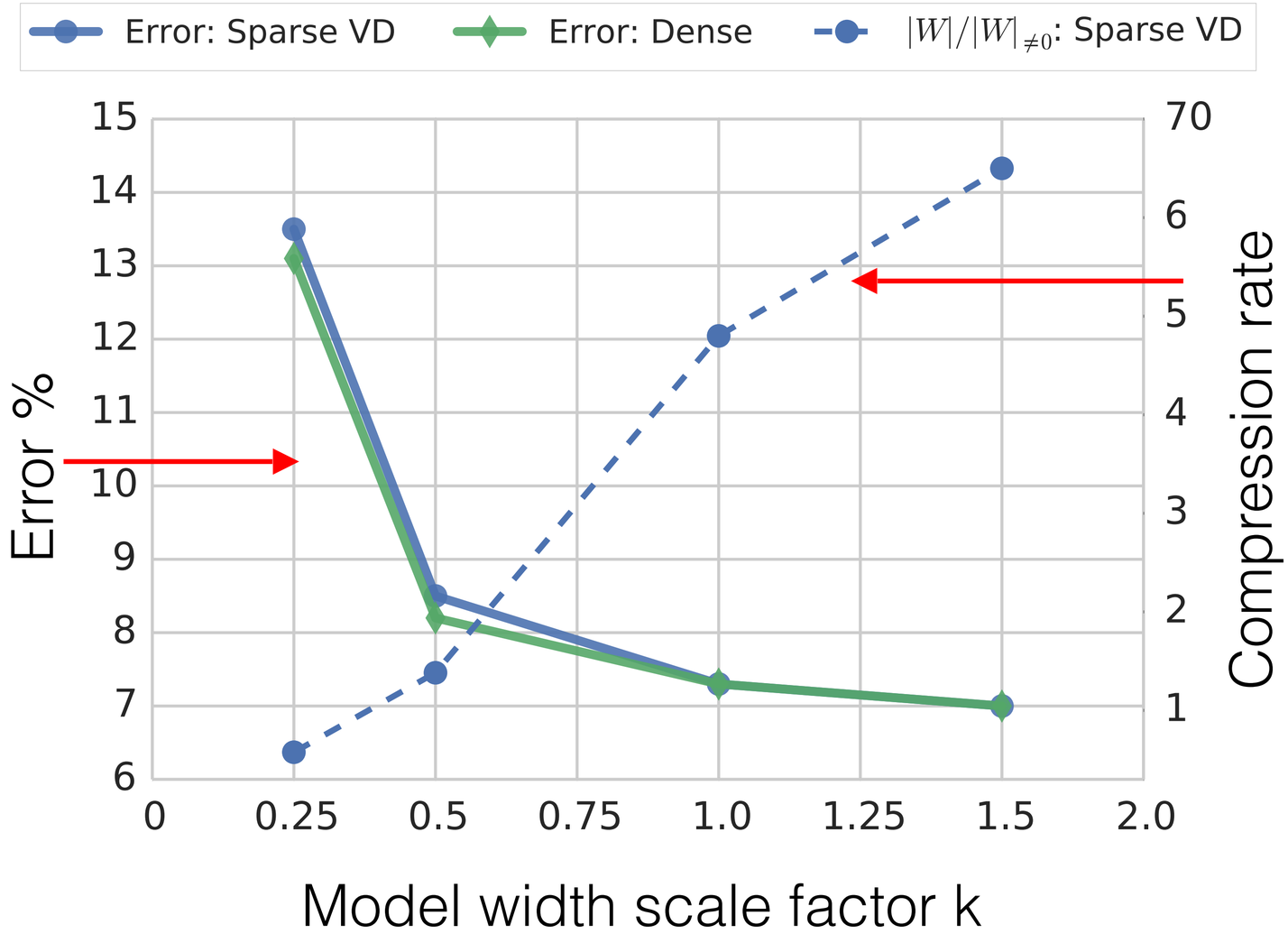


- 13 Convolutional layers and 2 Fully-Connected layers
- Pre-Activation Batch Norm and Binary Dropout after each layer

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."

Plan of the second part: Bayesian sparsification

- Sparsification: what and why
- Sparse variational dropout
- Practical assignment: implementation of SparseVD (seminar)
- Model enhancements (seminar)

Practical assignment

1. Take a look at an example (or skip it) : FC for a MNIST in PyTorch
2. Implement two python classes for SparseVD and incorporate them into model training
3. Train SparseVD and visualize weights

https://github.com/yandexdataschool/mlhep2019/tree/master/notebooks/day-4/Bayesian/SparseVD_assignment.ipynb

Plan of the second part: Bayesian sparsification

- Sparsification: what and why
- Sparse variational dropout
- Practical assignment: implementation of SparseVD (seminar)
- Model enhancements (seminar)

What’s next?

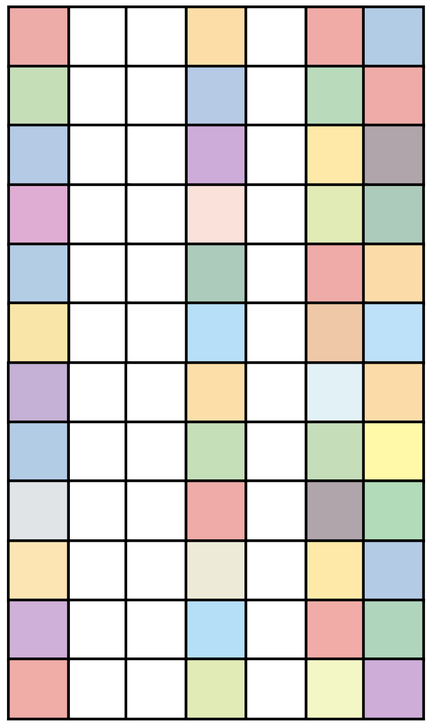
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

What’s next?

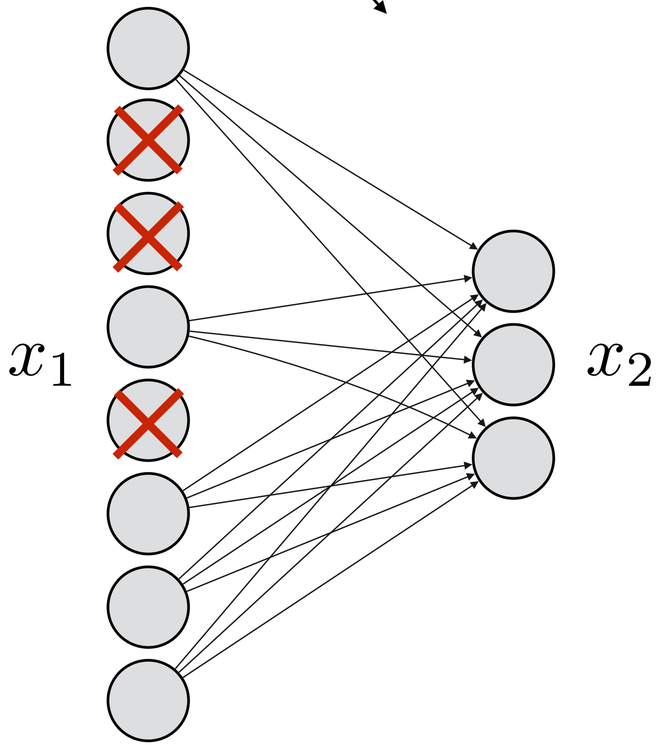
- Structured Bayesian sparsification
- How does it all works on large datasets and networks

Structured Bayesian sparsification

No outgoing edges
⇒ remove neuron



Weight matrix W



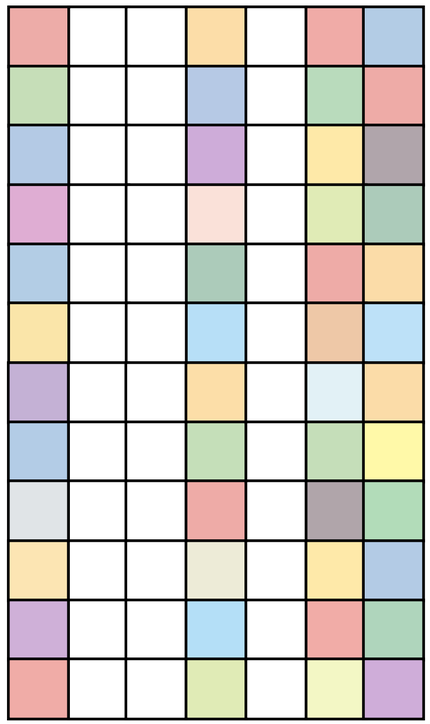
Computational graph

Benefits of structured sparsification:

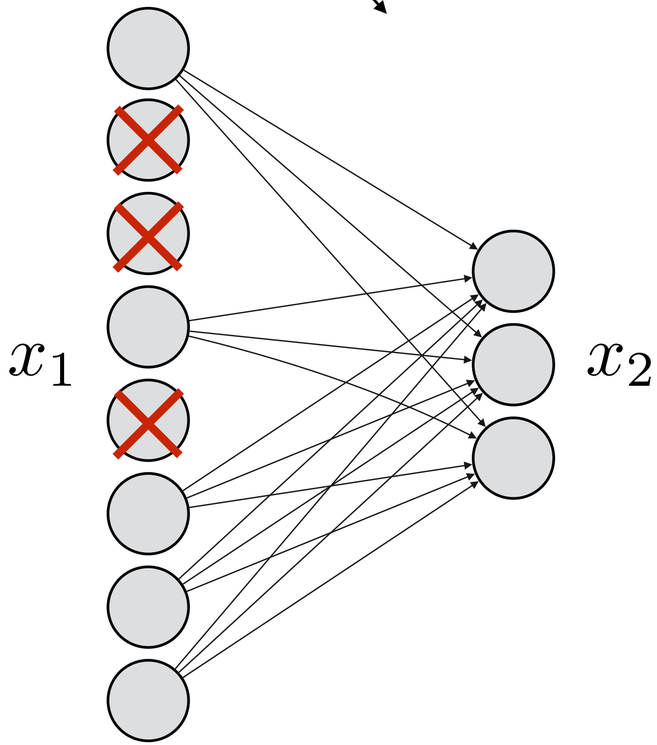
- more efficient compression
- speed-up of forward pass (faster testing stage)

Structured Bayesian sparsification

No outgoing edges
⇒ remove neuron



Weight matrix W

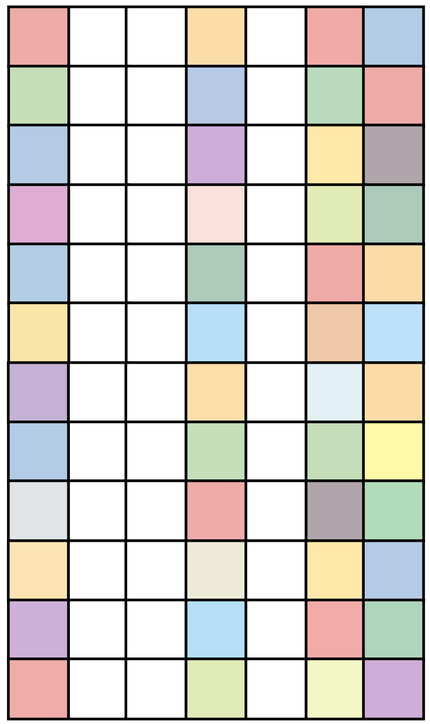


Computational graph

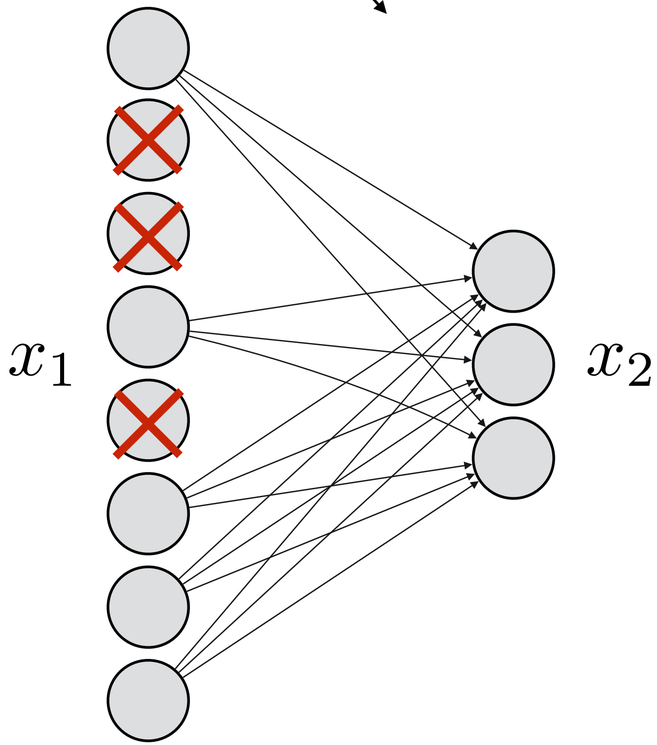
$$x_2 = \sigma(W x_1 + b)$$

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:

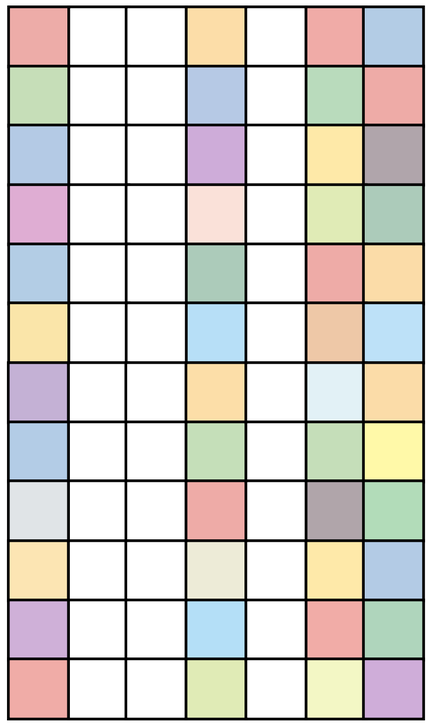
$$x_2 = \sigma(W(x_1 \odot z) + b)$$

zero component in z ⇒
no outgoing edges ⇒
remove neuron

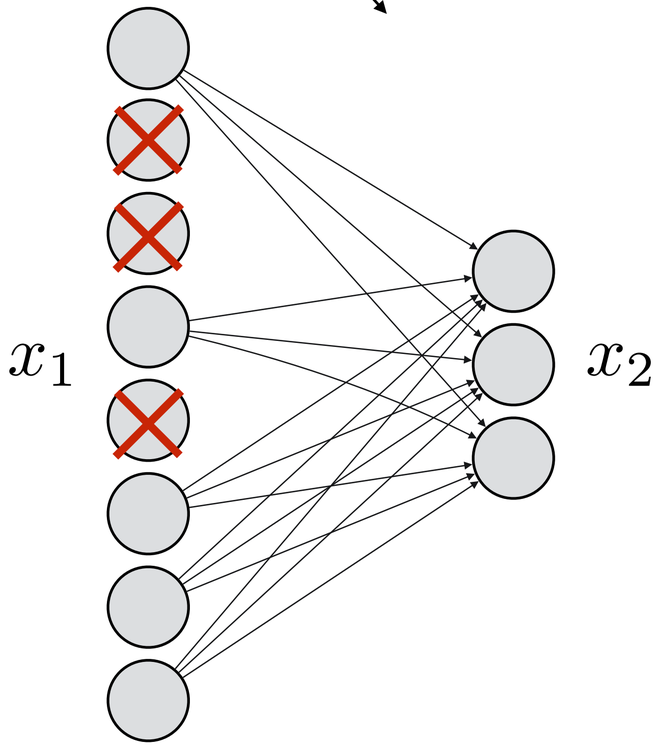
Louizos, Christos et al. Bayesian compression for deep learning. NIPS 2017

Structured Bayesian sparsification: training

No outgoing edges
⇒ remove neuron



Weight matrix W



Computational graph

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

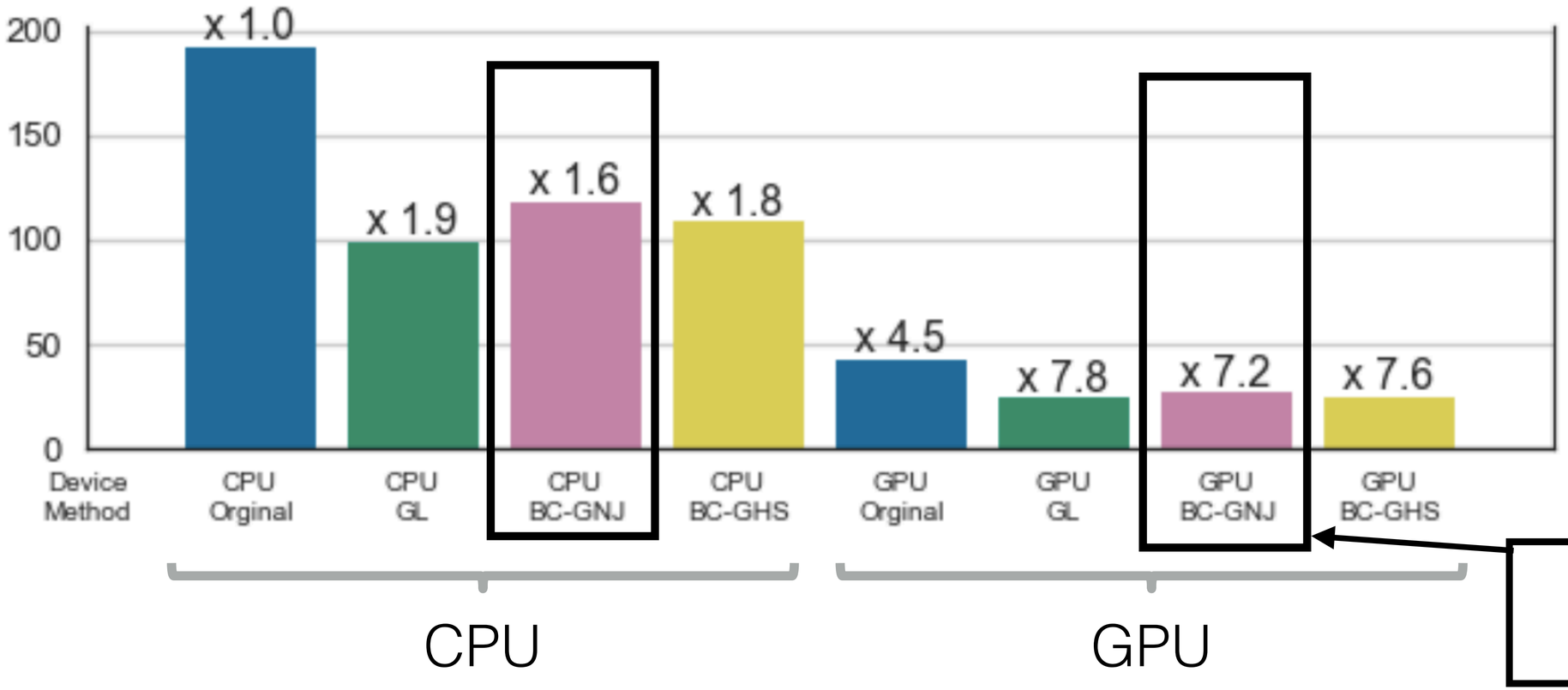
minor modification of computational graph

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)

Structured Bayesian sparsification: results

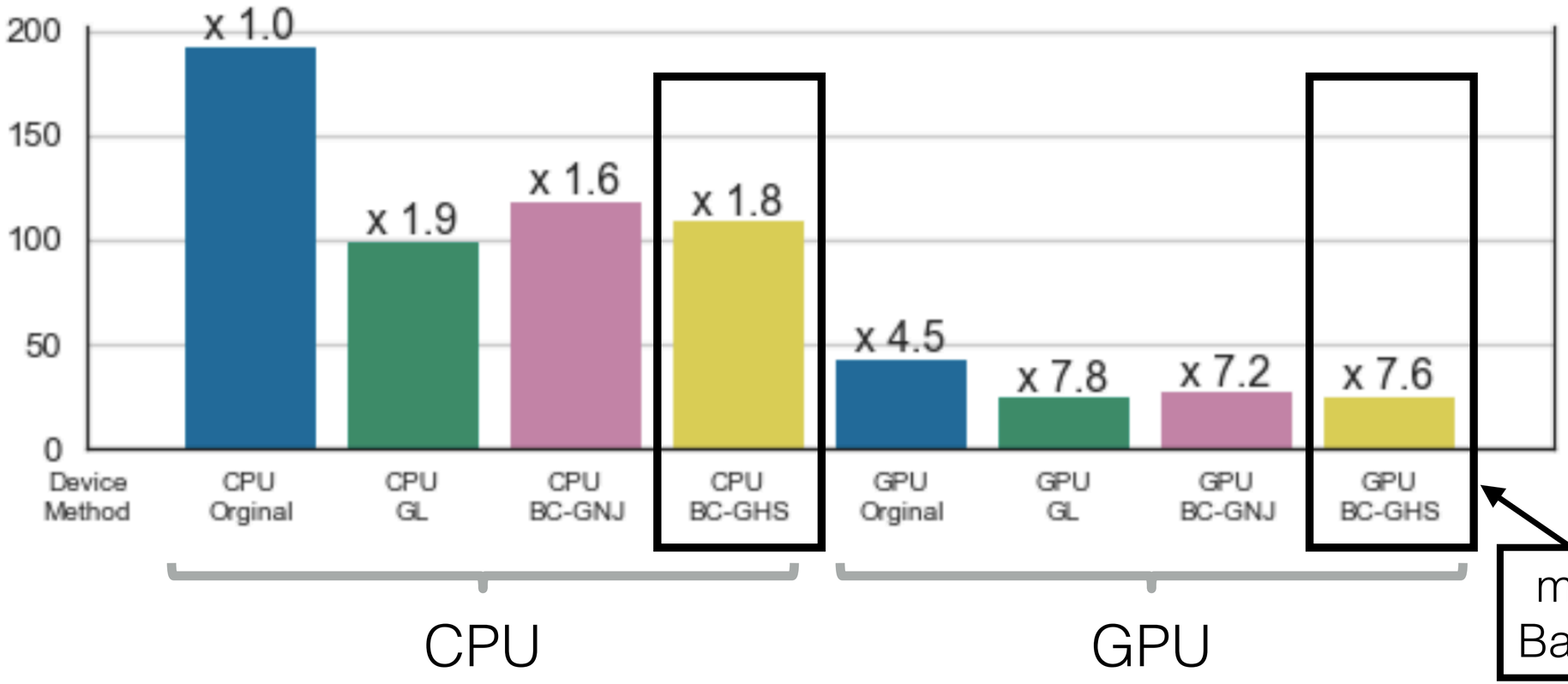
Speed-up of forward pass (testing stage) for Lenet-5-Caffe



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?

- 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

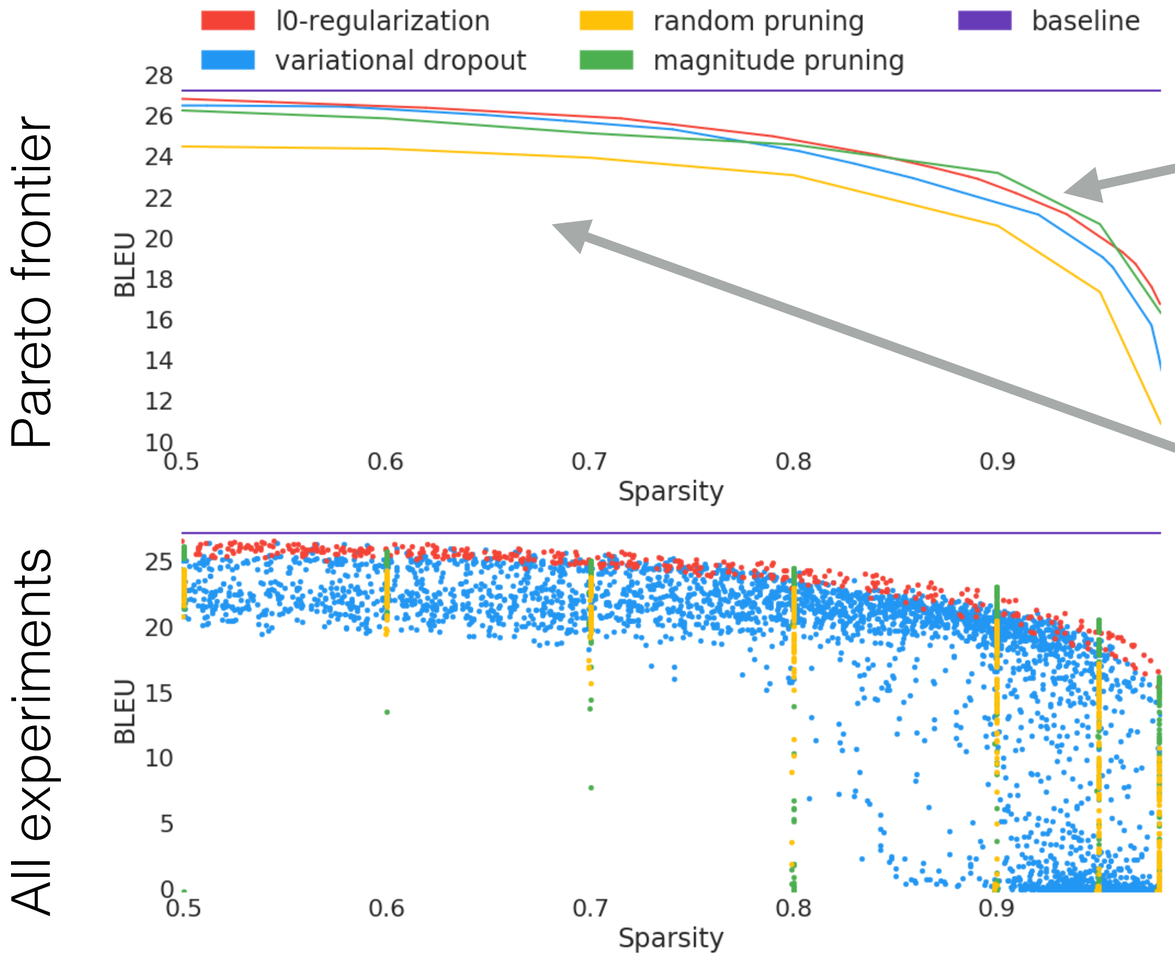
Hyperparameters for both tasks

- 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



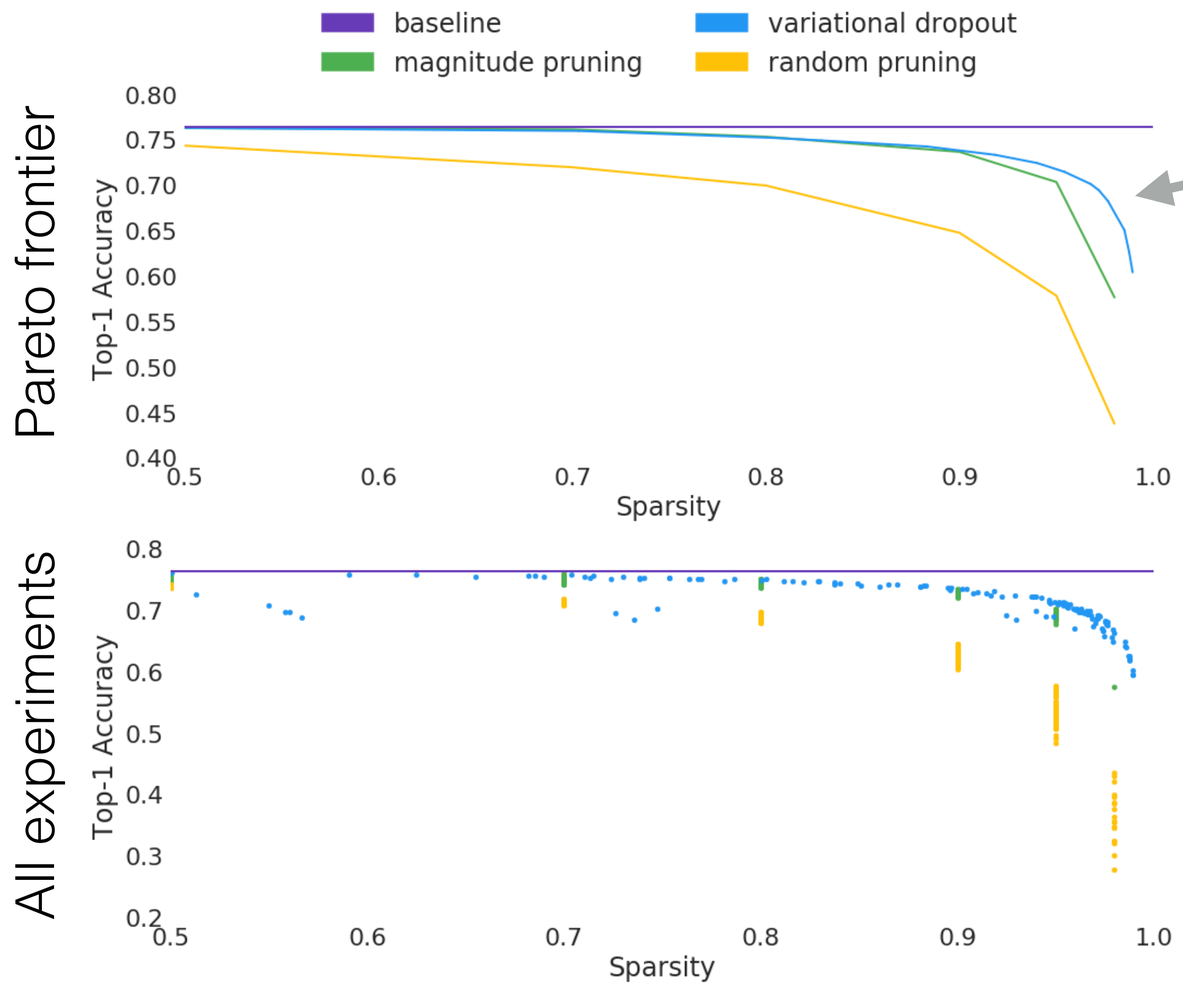
Magnitude pruning outperforms SparseVD a bit for high compression ratio

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

Summary

- Bayesian neural networks provide regularization, fast ensembling, uncertainty estimation etc.

- Sparsification of neural networks is an urgent industrial problem that is well solved using Bayesian deep learning

- Bayesian sparsification is a theoretically grounded approach but several tricks are needed to train the model