Bayesian Neural Networks

Evgeny Burnaev

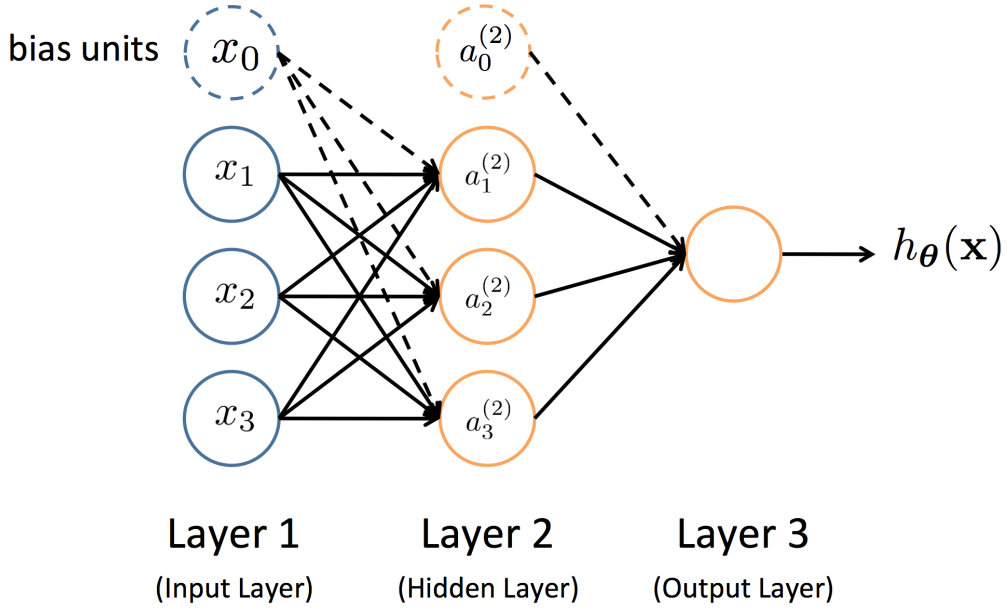
Skoltech, Moscow, Russia

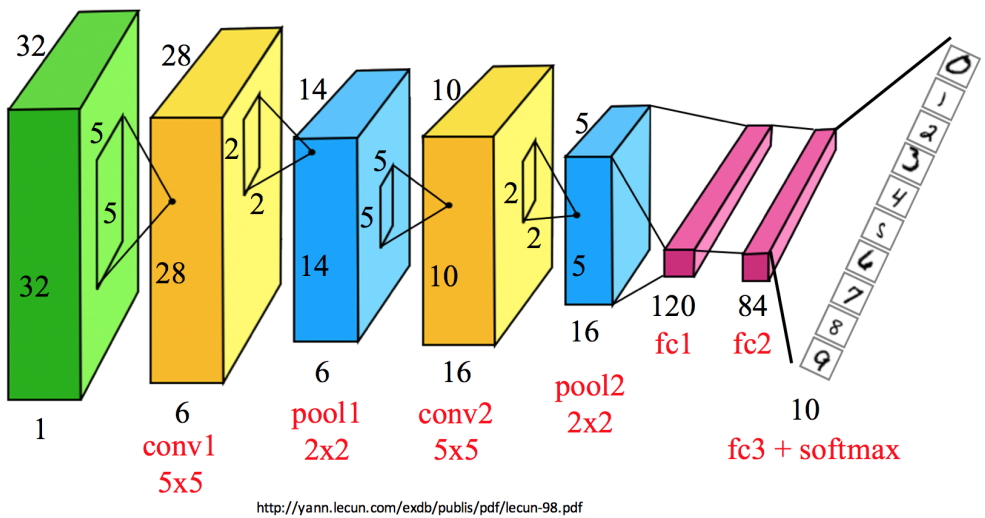
Skoltech

Skolkovo Institute of Science and Technology

Outline

Multi-Layered Neural Network

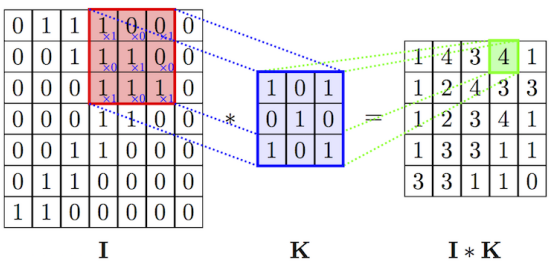




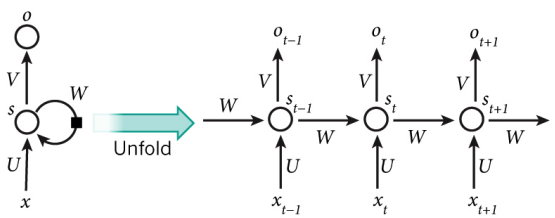
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Different architectures

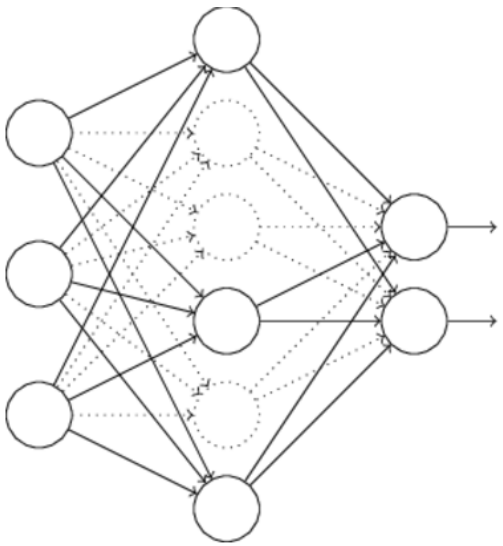
Convolutional layer



Recurrent layer



Dropout layer

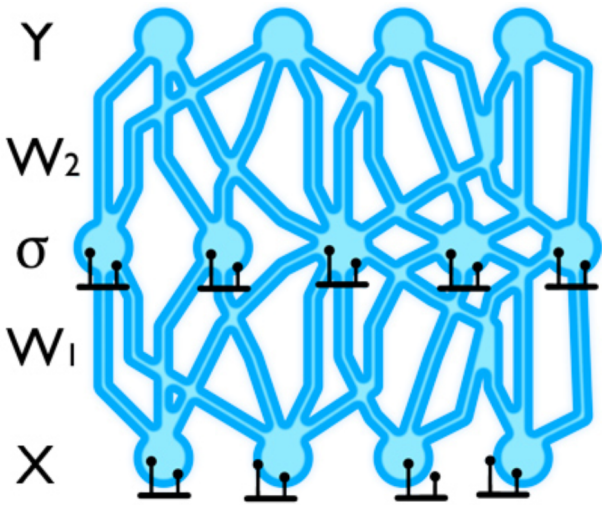


Modern Deep Learning

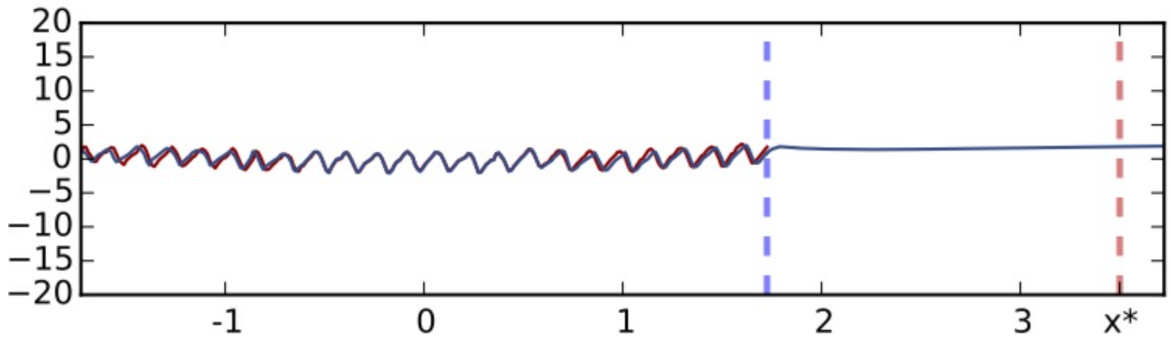
- Attracts tremendous attention from popular media
- Fundamentally affected the way ML is used in industry
- Driven by pragmatic developments of models that work well and scale well

Understand recipes

- Why does my model work?
- We don't understand many of the tools that we use
- E.g. stochastic reg. techniques (dropout) are used in most deep learning models to avoid over-fitting

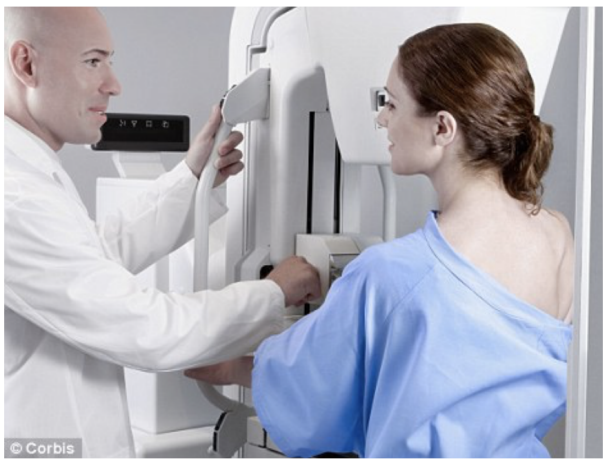


- How to estimate model uncertainty? E.g. what would be the CO2 concentration level in 20 years' time?



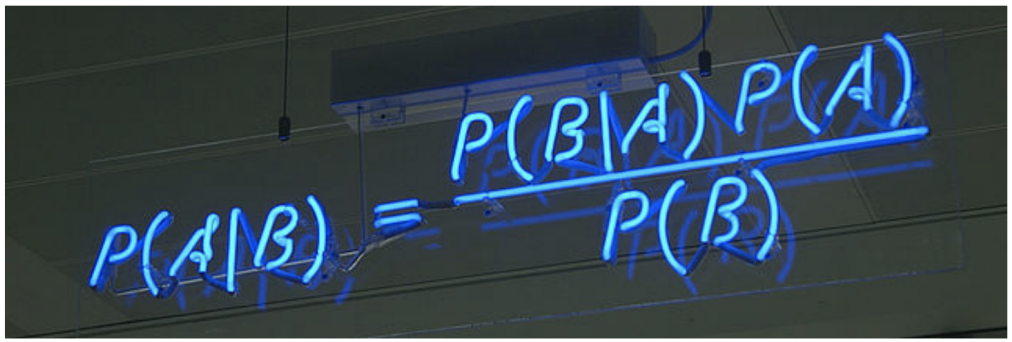
Interpretability

- Why does my model predict this and not that? Our models are black boxes and not interpretable



Bayesian framework

- Why does my model work
- What does my model know?
- Why does my model predict this and not that?



We can use Bayesian modelling to answer the questions above!

- Stochastic models for inputs $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^m$ and outputs $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^m$
- Prior $p(\mathbf{w})$ over parameters $\mathbf{w}$
- Likelihood $p(\mathbf{Y}|\mathbf{w}, \mathbf{X})$
- Posterior

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{w}, \mathbf{X})p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X})}$$

- Predictive distribution given new input $\mathbf{x}^*$

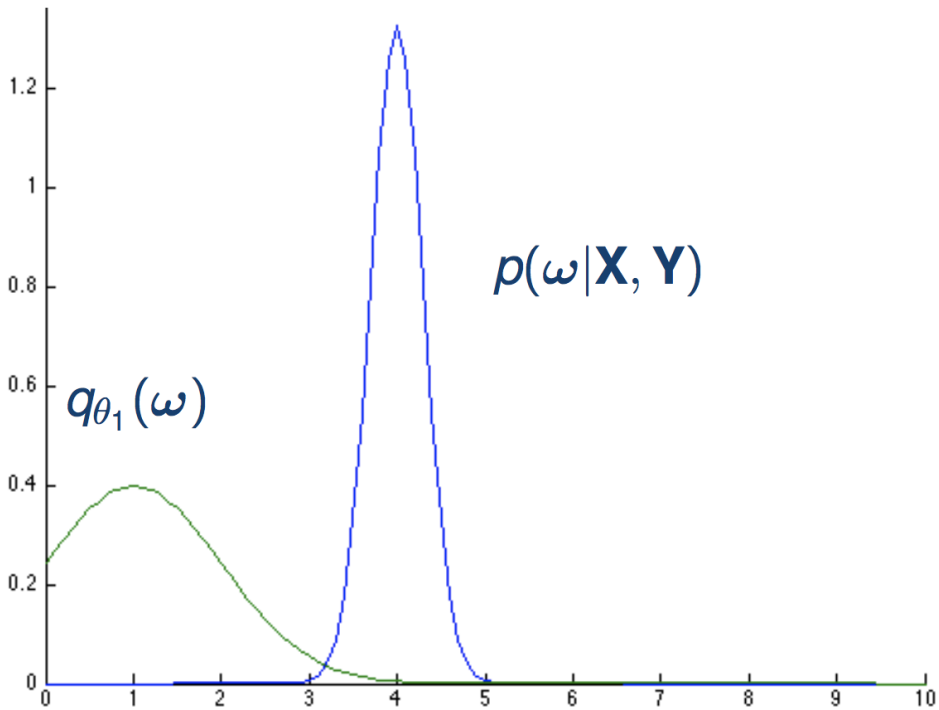
$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{Y})d\mathbf{w}$$

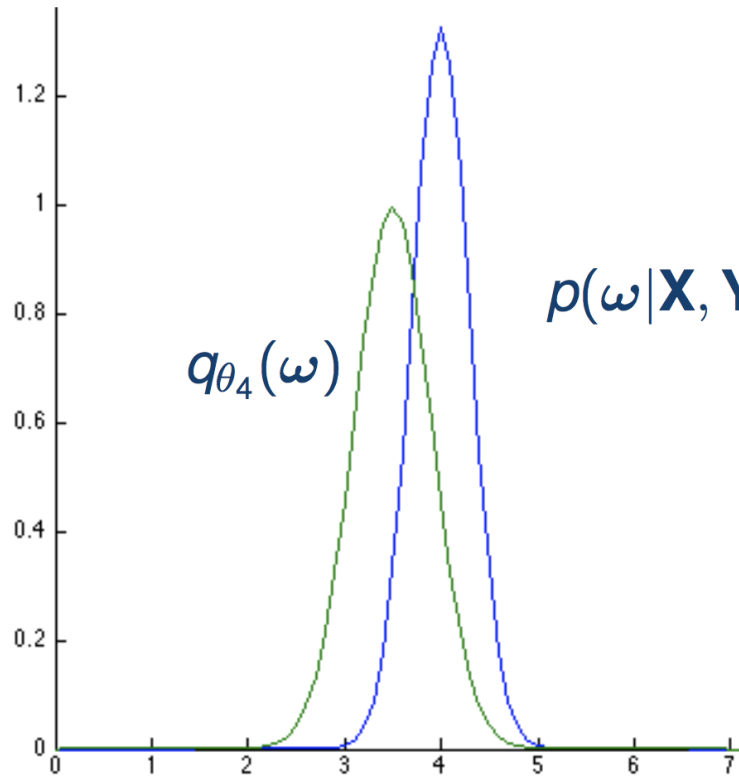
- Posterior $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ is often intractable

Approximate inference

- Approximate $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ with simple dist. $q(\mathbf{w}|\boldsymbol{\theta})$
- Minimize divergence from posterior w.r.t. $\boldsymbol{\theta}$

$$KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}|\mathbf{X}, \mathbf{Y}))$$





- Approximate $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ with simple dist. $q(\mathbf{w}|\boldsymbol{\theta})$
- Minimize divergence from posterior w.r.t. $\boldsymbol{\theta}$

$$KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}|\mathbf{X}, \mathbf{Y}))$$

- Identical to minimizing —ELBO

$$\mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{w}\sim q(\mathbf{w}|\boldsymbol{\theta})}[\underbrace{\log p(\mathbf{Y}|\mathbf{X}, \mathbf{w})}_{\text{likelihood}}] + KL(q(\mathbf{w}|\boldsymbol{\theta})||\underbrace{p(\mathbf{w})}_{\text{prior}})$$

- We can approximate the predictive distribution

$$q(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\theta}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w})q(\mathbf{w}|\boldsymbol{\theta})d\mathbf{w}$$

Bayesian framework: so what?

We'll look at dropout specifically:

- Used in most modern deep learning models
- It somehow circumvents over-fitting
- And improves performance
- With Bayesian modelling we can explain why (modern deep learning as approximate inference)

Standard Neural Network with dropout

- Let $\widehat{\mathbf{y}}$ be the output of a NN model with L layers and a loss function $E(\cdot, \cdot)$ (softmax loss or square loss)
- We denote by $\mathbf{w}_i$ the NN's weight matrices of dimensions $K_i \times K_{i-1}$, and by $\mathbf{b}_i \in \mathbb{R}^{K_i}$ the bias vectors, $i = 1, \ldots, L$
- Output is (denote $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{L}$)

$$\mathbf{f}(\mathbf{x}, \mathbf{w}) = \mathbf{w}_L \sigma(\ldots \mathbf{w}_2 \sigma(\mathbf{w}_1 \mathbf{x} + \mathbf{b}_1) \ldots)$$

- To train $\mathbf{f}(\mathbf{x}, \mathbf{w})$ we use dropout (see the next slides)
- Besides dropout we often use L_2 regularisation

$$\mathcal{L}_{dropout} = \frac{1}{m} \sum_{i=1}^{m} E(\mathbf{y}_i, \widehat{\mathbf{y}}_i) + \lambda \sum_{i=1}^{L} (\|\mathbf{w}_i\|_2^2 + \|\mathbf{b}_i\|_2^2)$$

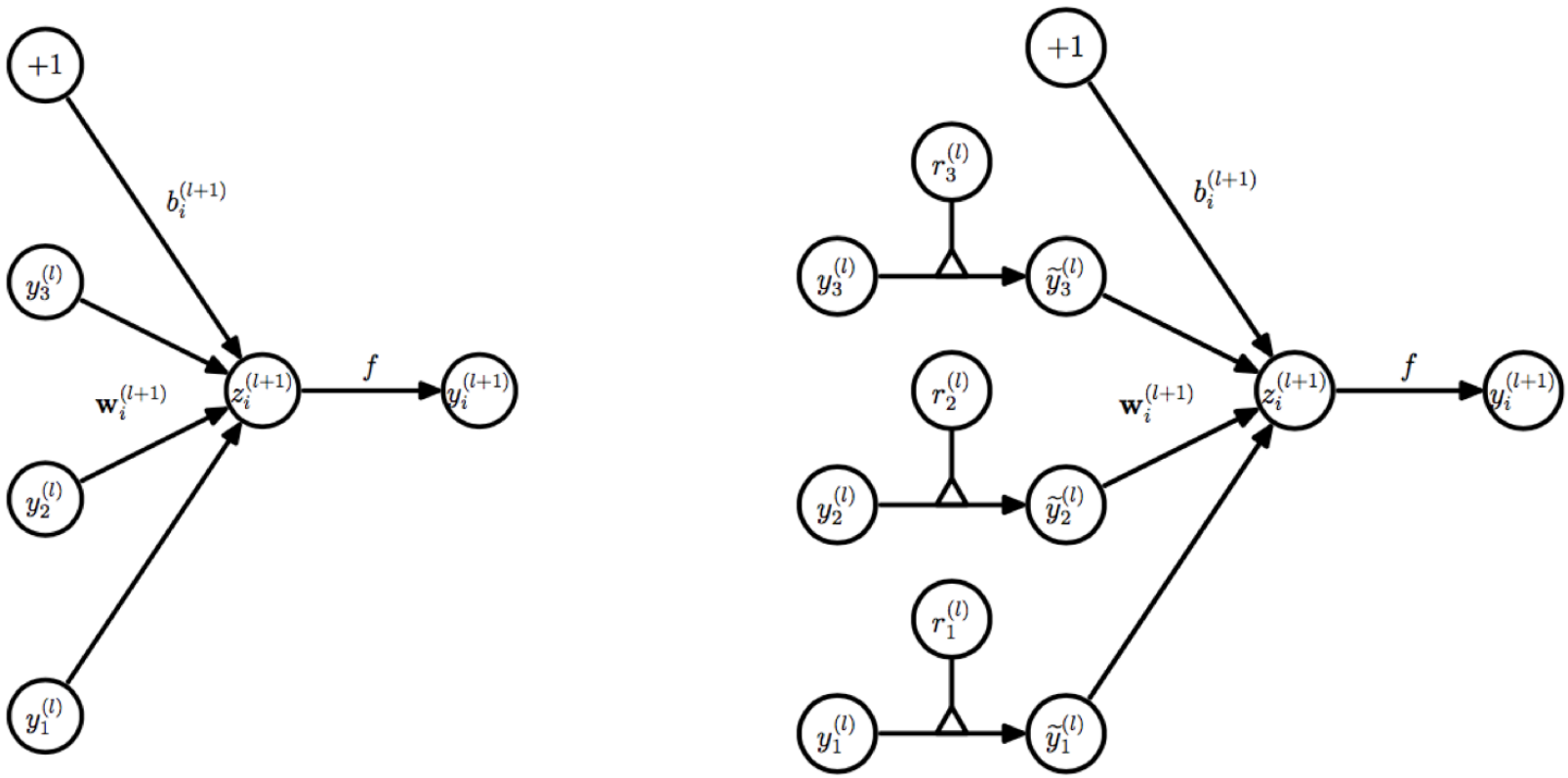
Dropout

- Randomly disable outputs of a layer:

$$r_{jk} \sim \text{Bernoulli}(p)$$

$$v_{j+1}(\mathbf{x}) = f_{j+1}(v_j(\mathbf{x}) \otimes r_j; \mathbf{w}_{j+1})$$

- Schematically this looks as



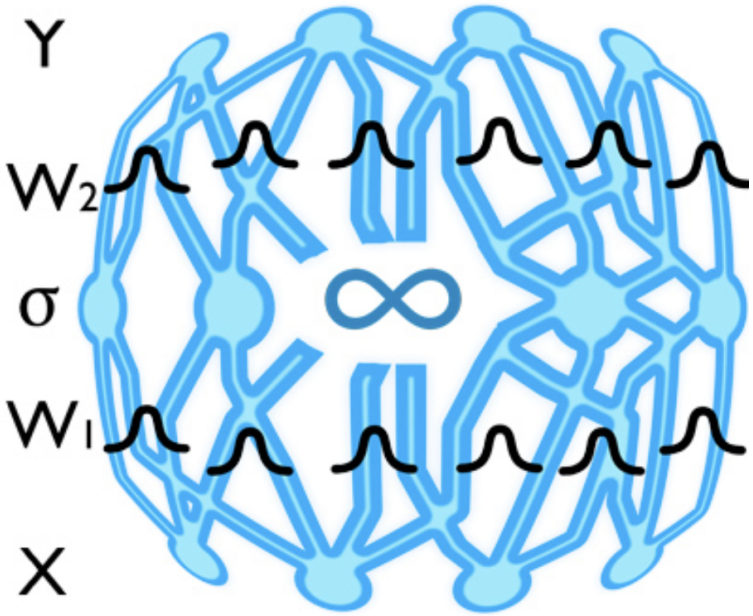
Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights (n is the number of parameters)
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- **Next lecture**: variational dropout can learn separate dropout weight for each node

- Place prior $p(\mathbf{w}_i)$

$$\text{vec}[\mathbf{w}_i] \sim \mathcal{N}(\mathbf{0}, l^{-2}\mathbf{I})$$

for $i \leq L$ (and denote $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{L}$)



- Place prior $p(\mathbf{w}_i)$

$$\mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, l^{-2}\mathbf{I})$$

  for $i \leq L$ (and denote $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{L}$)

- Output is a r.v.

$$\mathbf{f}(\mathbf{x}, \mathbf{w}) = \sqrt{\frac{1}{K_L}}\mathbf{w}_L\sigma(\dots\sqrt{\frac{1}{K_1}}\mathbf{w}_2\sigma(\mathbf{w}_1\mathbf{x} + \mathbf{b}_1)\dots)$$

- Softmax likelihood for class.: $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \mathrm{softmax}(\mathbf{f}(\mathbf{x}, \mathbf{w}))$ or a Gaussian for regression: $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{y}; \mathbf{f}(\mathbf{x}, \mathbf{w}), \tau^{-1})$
- Intractable posterior

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$$

Approximate inference in Bayesian NNs

- Def. $q(\mathbf{w}|\boldsymbol{\theta})$ to approximate posterior $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$
- KL divergence to minimize

$$KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}|\mathbf{X}, \mathbf{Y}))$$
$$\sim -\int q(\mathbf{w}|\boldsymbol{\theta}) \log p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) d\mathbf{w} + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w})) = \mathcal{L}(\boldsymbol{\theta})$$

- We rewrite the first term as a sum and get

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \int q(\mathbf{w}|\boldsymbol{\theta}) \log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}) d\mathbf{w} + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

- Approximate the integral with MC integration $\widehat{\mathbf{w}}_i \sim q(\mathbf{w}|\boldsymbol{\theta})$ and get the unbiased estimate of the first term $-\log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i)$, i.e.

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i) + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

- Here we assume that parameters $\mathbf{w}$ explicitly depend on parameters $\boldsymbol{\theta}$. E.g. $\mathbf{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \circ \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$

- Unbiased estimator

$$\mathbb{E}_{\widehat{\mathbf{w}} \sim q(\mathbf{w}|\boldsymbol{\theta})}(\widehat{\mathcal{L}}(\boldsymbol{\theta})) = \mathcal{L}(\boldsymbol{\theta})$$

- Converges to the same optima as $\mathcal{L}(\boldsymbol{\theta})$
- For inference repeat
  - Sample $\widehat{\mathbf{w}}_i \sim q(\mathbf{w}|\boldsymbol{\theta})$
  - And minimize (one step)

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i) + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

w.r.t. $\boldsymbol{\theta}$

Variational Family

- Given variational parameters $\boldsymbol{\theta} = \{\mathbf{m}_{ik}\}_{i=1,k=1}^{i=L,k=K_{i-1}}$

$$q(\mathbf{w}|\boldsymbol{\theta}) = \prod_i q(\mathbf{w}_i|\boldsymbol{\theta})$$

$$q(\mathbf{w}_i|\boldsymbol{\theta}) = \prod_k q(\mathbf{w}_{ik}|\mathbf{m}_{ik})$$

$$q(\mathbf{w}_{ik}|\mathbf{m}_{ik}) = p_i\delta_0(\mathbf{w}_{ik}) + (1-p_i)\delta_{\mathbf{m}_{ik}}(\mathbf{w}_{ik})$$

- That is k-th column of the i-th layer is a mixture of two components
- Or, in a more compact way: given variational parameters $\boldsymbol{\theta} = \{\mathbf{M}_i\}_{i=1}^{L}$

$$\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i) \text{ for each layer } i \text{ and column } k$$

$$\mathbf{w}_i = \mathbf{M}_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$$

$$q(\mathbf{w}|\boldsymbol{\theta}) = \prod_i q(\mathbf{w}_i|\mathbf{M}_i)$$

- Repeat
  - Sample $\widehat{\mathbf{z}}_{i,j} \sim \text{Bernoulli}(p_i)$ and set

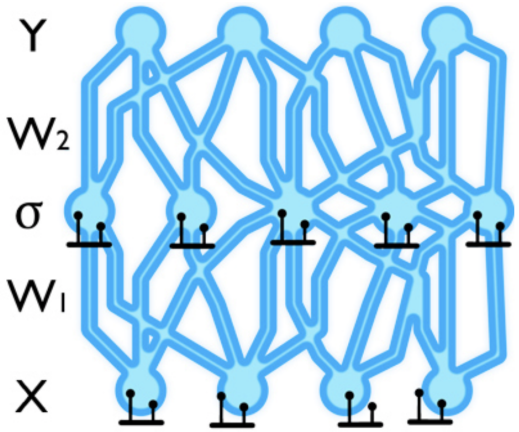
$$\widehat{\mathbf{w}}_i = \mathbf{M}_i \cdot \text{diag}([\widehat{\mathbf{z}}_{i,j}]_{j=1}^{K_i})$$

$$\widehat{\mathbf{w}} = \{\widehat{\mathbf{w}}_i\}_{i=1}^{L}$$

  - Minimize (one step)

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i) + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

w.r.t. $\boldsymbol{\theta} = \{\mathbf{M}_i\}_{i=1}^{L}$ (set of matrices)



- Repeat
  - Randomly set columns of $\mathbf{M}_i$ to zero
  - Minimize (one step)

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i) + KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

w.r.t. $\boldsymbol{\theta} = \{\mathbf{M}_i\}_{i=1}^{L}$ (set of matrices)

- Given model precision τ (in case of the regression) we scale the result by the constant $\frac{1}{\tau m}$ to obtain the objective:

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\frac{-\log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i)}{\tau} + \frac{1}{\tau m} KL(q(\mathbf{w}|\boldsymbol{\theta})||p(\mathbf{w}))$$

- It can be proved (see Gal and Ghahramani, 2015) that

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) \approx \underbrace{\frac{1}{m}\sum_{i=1}^{m}\frac{-\log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i)}{\tau}}_{=\text{ loss}} + \underbrace{\sum_{i=1}^{L}\left(\frac{p_i l^2}{2\tau m}\|\mathbf{M}_i\|_2^2 + \frac{l^2}{2\tau m}\|\mathbf{b}_i\|_2^2\right)}_{=\ L_2 \text{ reg}},$$

- The first term is a data log-likelihood with parameters influenced by a Bernoulli dropout on each iteration
- The second term is approximately equal to the L_2-regularization
- Setting $E(\mathbf{y}_i, \widehat{\mathbf{y}}(\mathbf{x}_i, \widehat{\mathbf{w}}_i)) = -\log p(\mathbf{y}_i|\mathbf{x}_i, \widehat{\mathbf{w}}_i)/\tau$ we recover a standard criterion for training of NN with dropout

Implementing VI with $q(\mathbf{w}|\theta)$ above = implementing dropout in deep network

Better use of dropout

How do we use dropout with convolutional neural networks (convnets)?

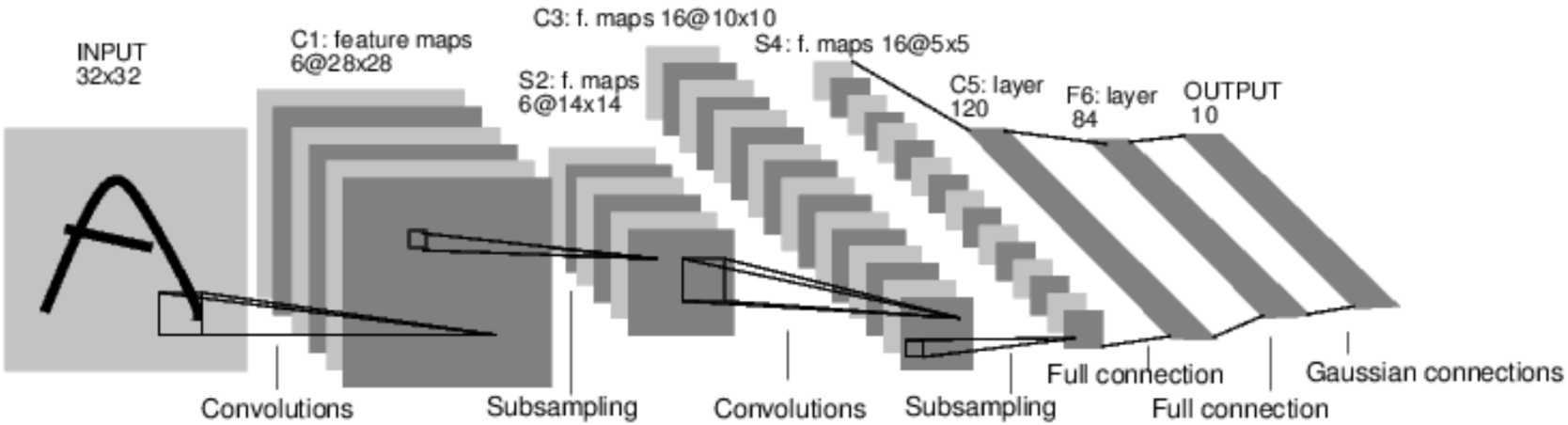


Figure: LeNet convnet structure

Better use of dropout

How do we use dropout with convolutional neural networks (convnets)?

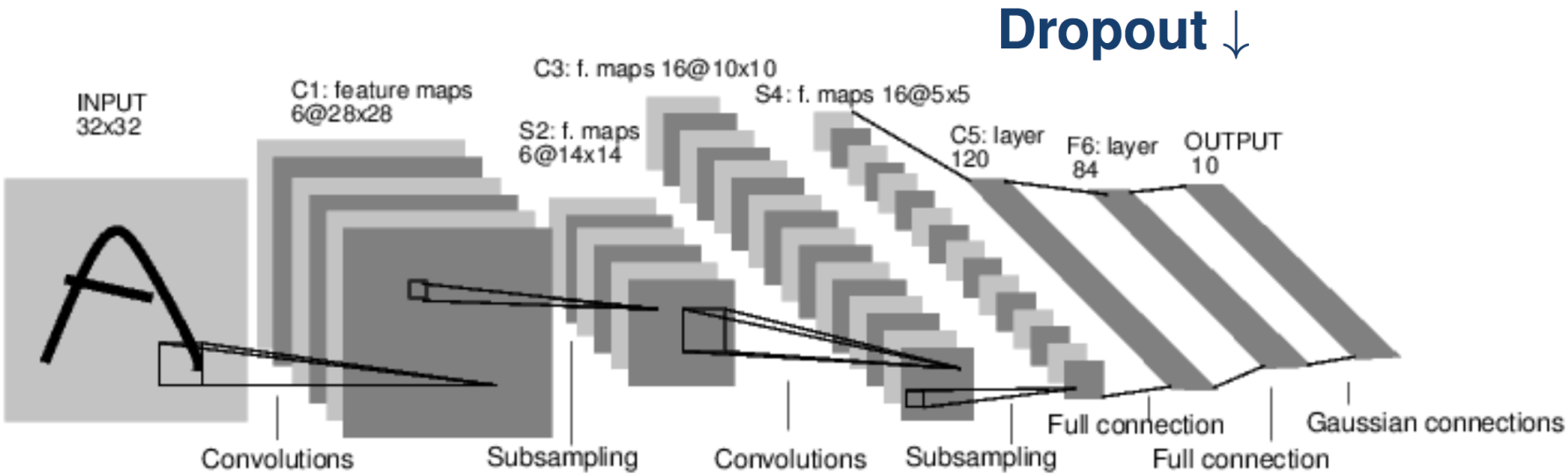


Figure: LeNet convnet structure

How do we use dropout with convolutional neural networks (convnets)?

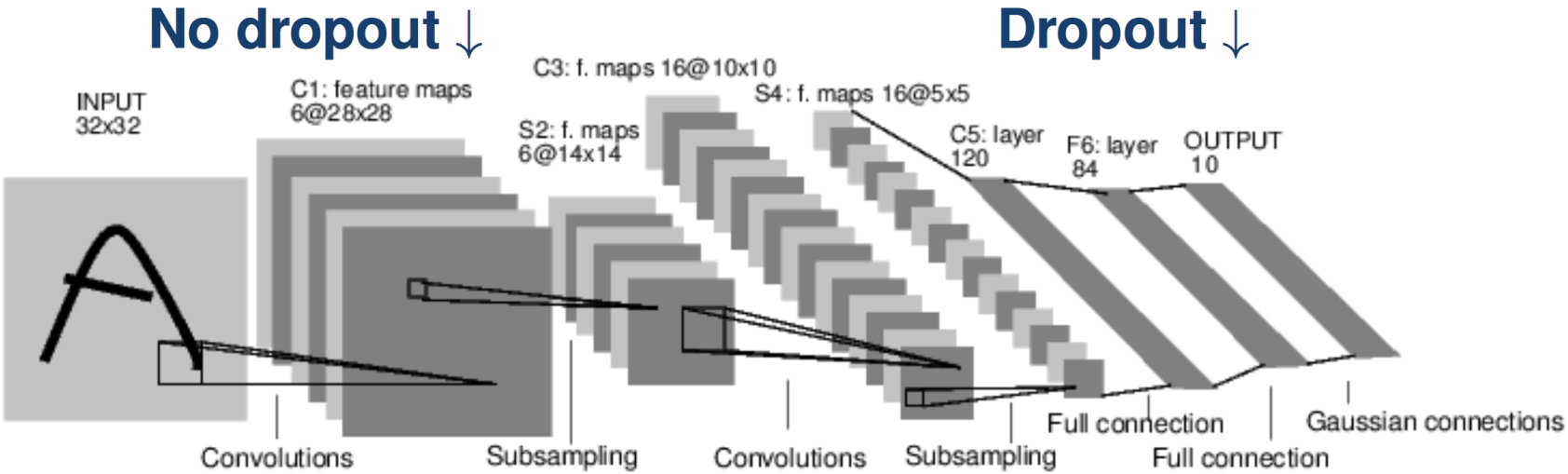


Figure: LeNet convnet structure

- Why not use dropout et al. with convolutions?
- Because it's not used correctly
- Standard dropout averages weights at test time

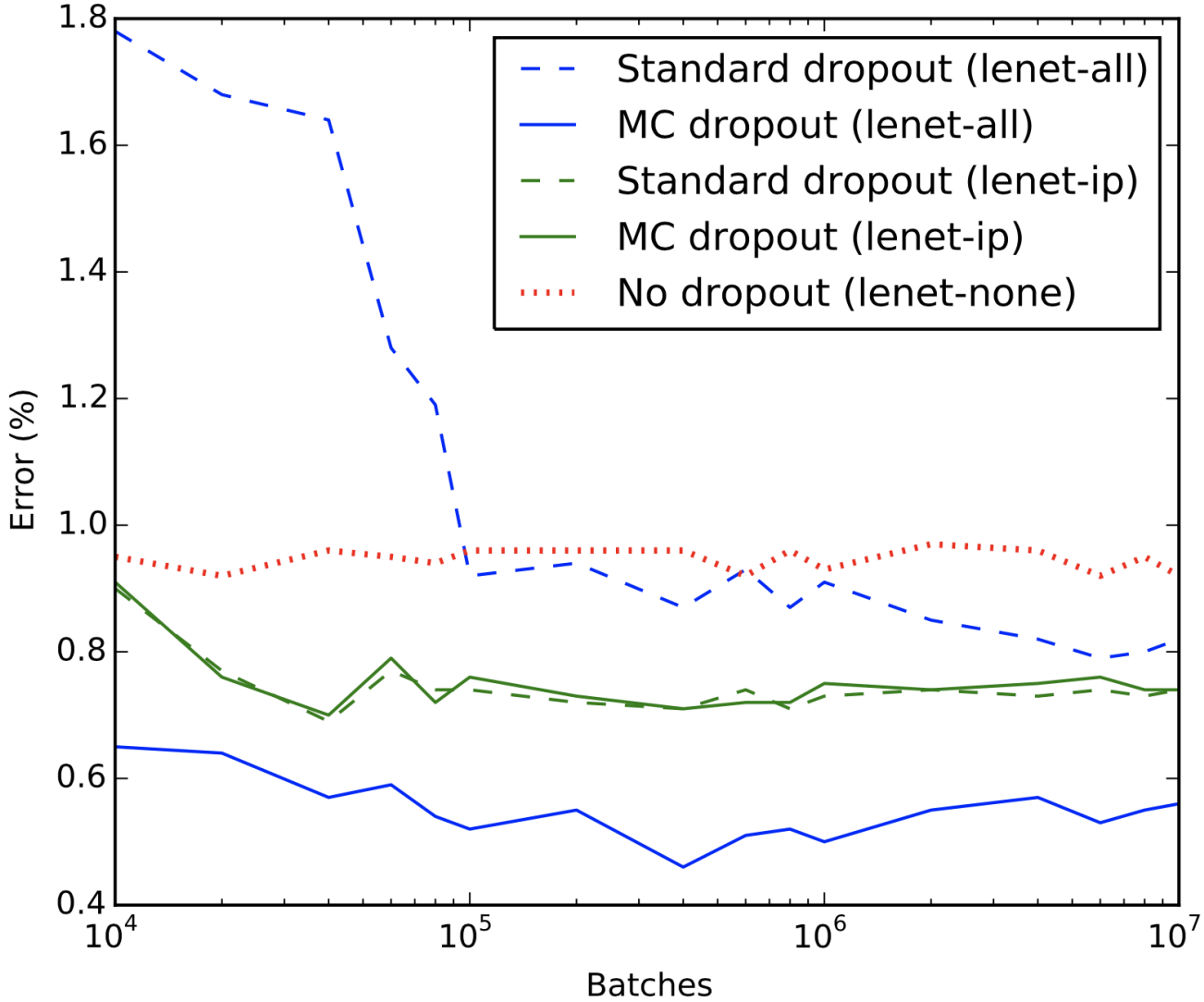
Instead, predictive mean, approx. with MC integration:

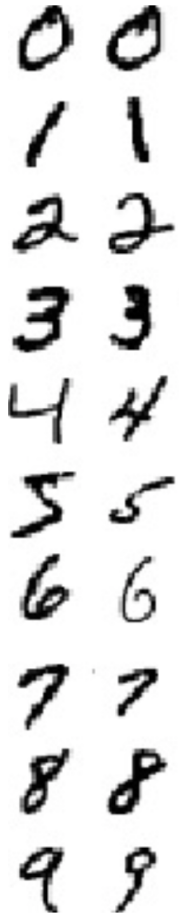
$$\widehat{\mathbb{E}}_{q(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})}(\mathbf{y}^*) = \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}(\mathbf{x}^*, \widehat{\mathbf{w}}_t)$$

with $\widehat{\mathbf{w}}_t \sim q(\mathbf{w}|\boldsymbol{\theta})$

- In practice, average stochastic forward passes through the network (referred to as "MC dropout")
- Dropout after convolutions and averaging forward passes = approximate inference in Bayesian convnets
- Very simple: drop units at test time, repeat T times and look at mean and sample variance!

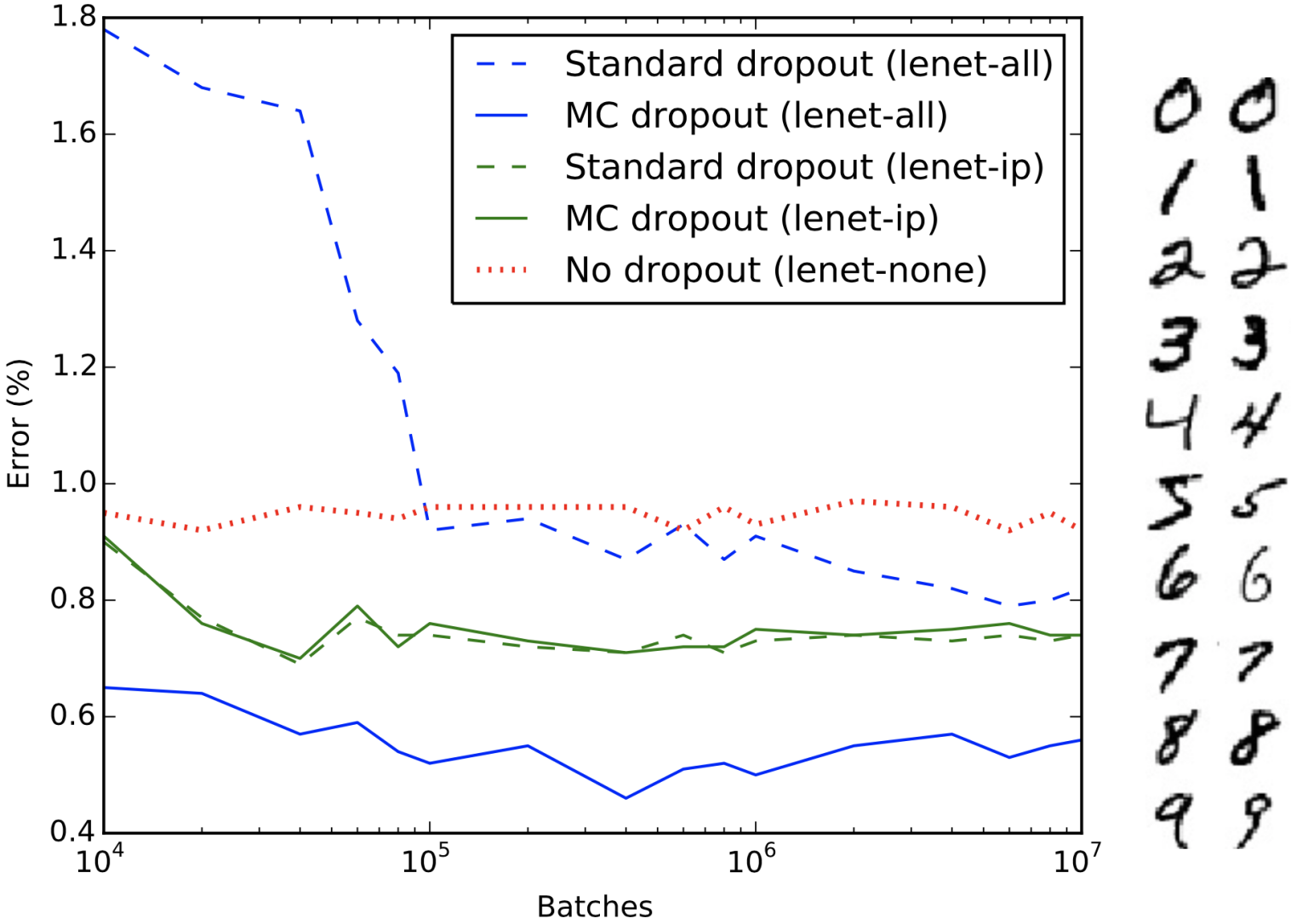
Huge improvement (MNIST)





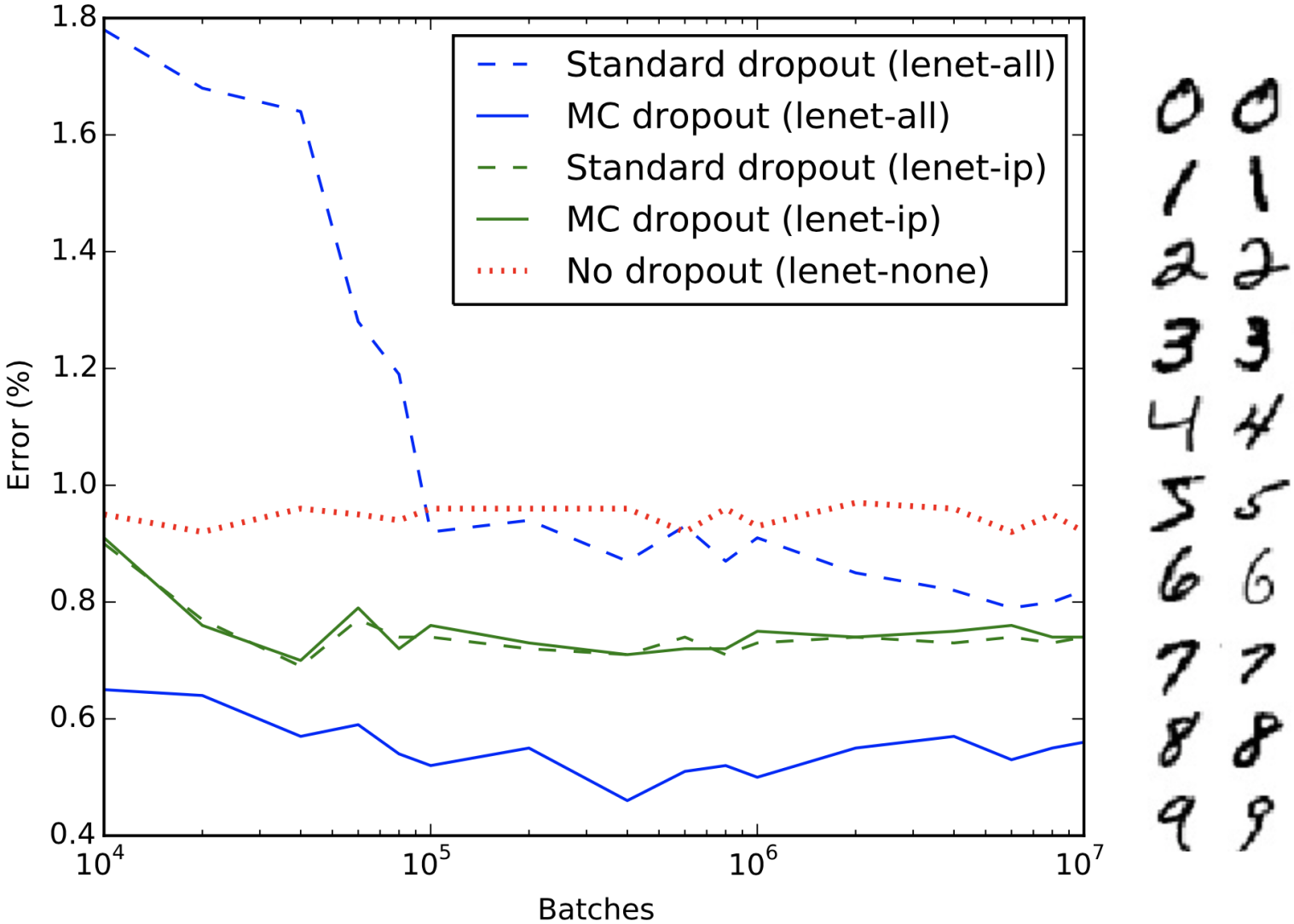
Red: standard LeNet (no dropout)

Huge improvement (MNIST)



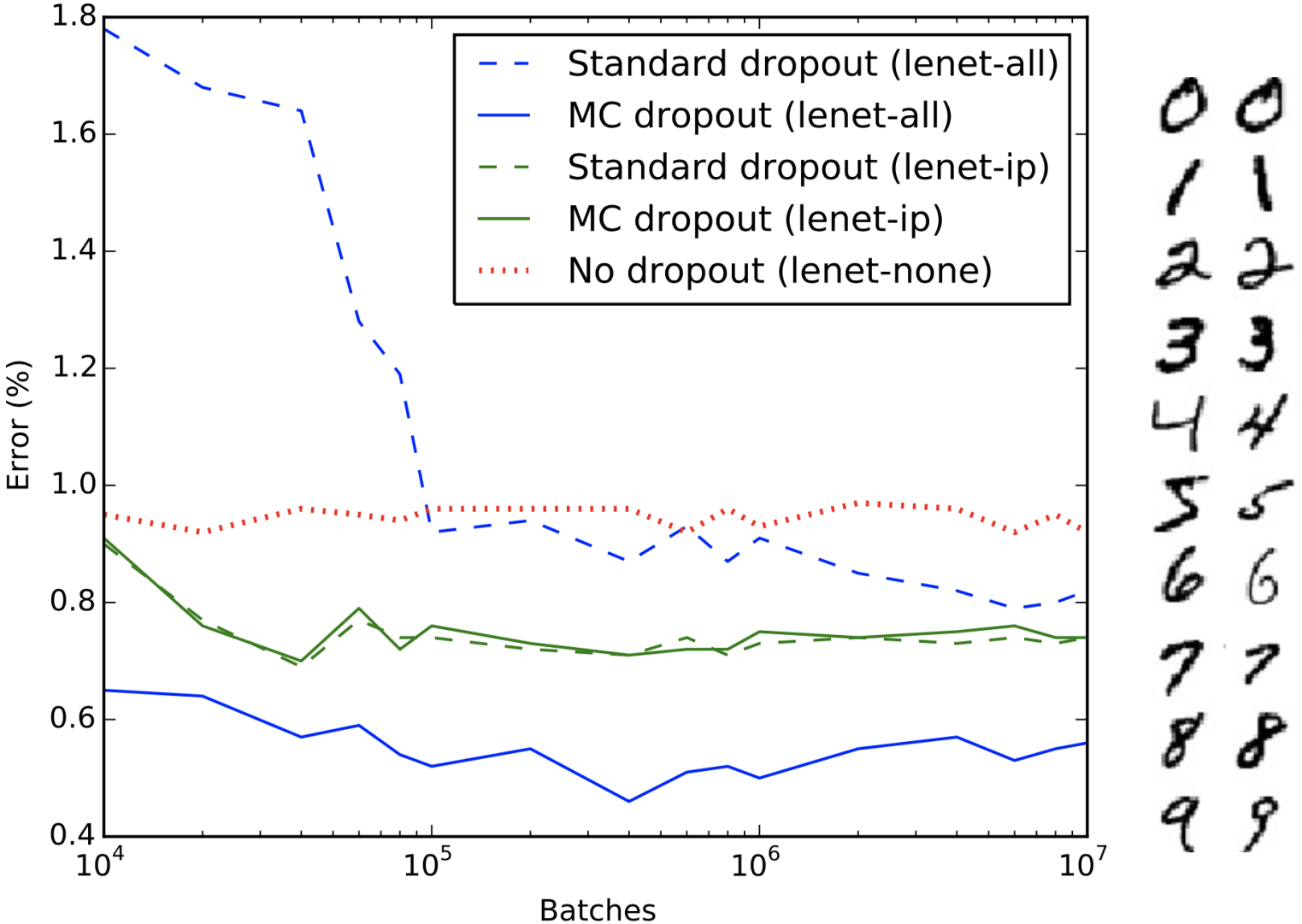
Green: standard dropout LeNet (dropout at the end)

Huge improvement (MNIST)



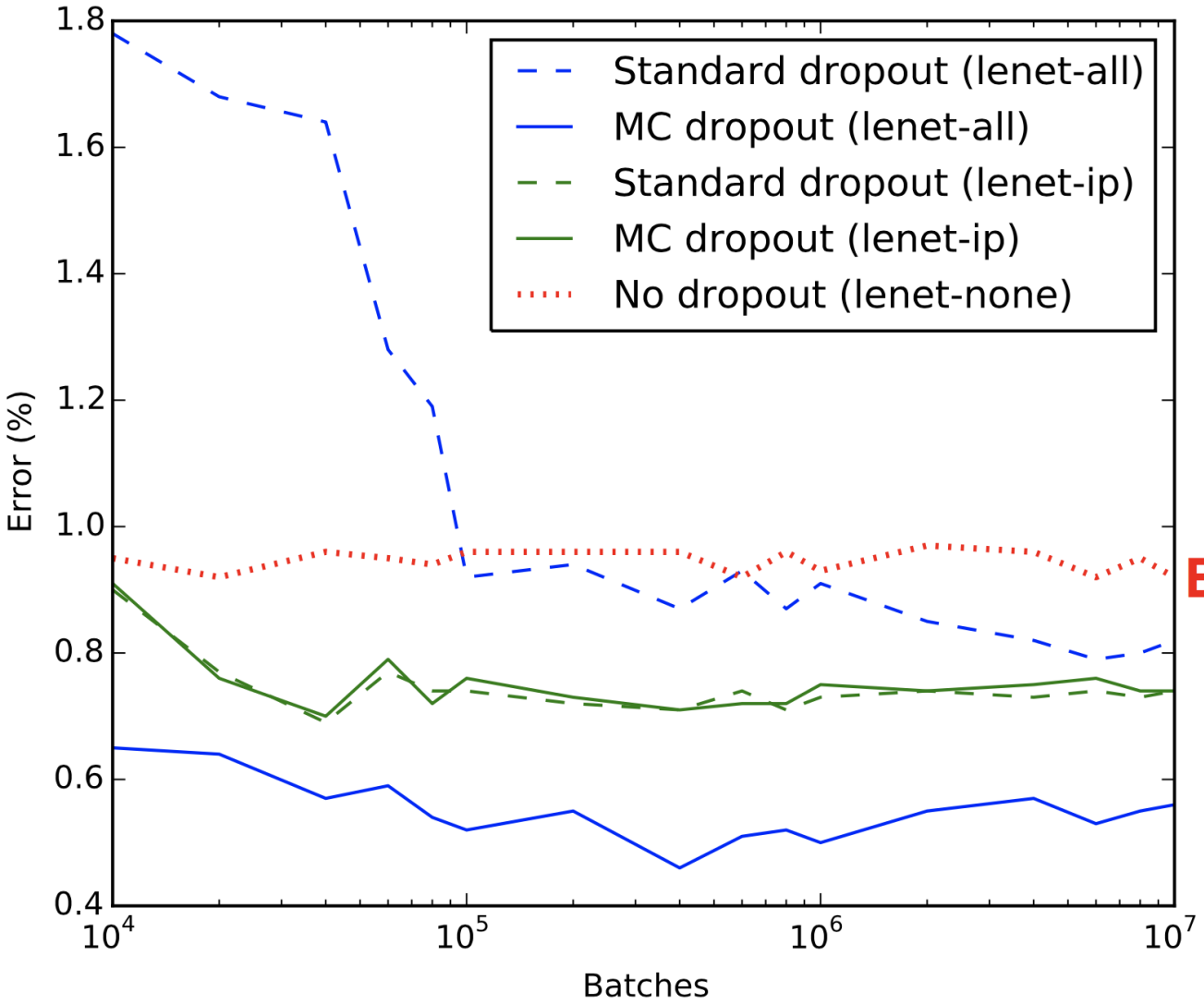
Dashed blue: Bayesian LeNet (weight averaging – FAIL)

Huge improvement (MNIST)



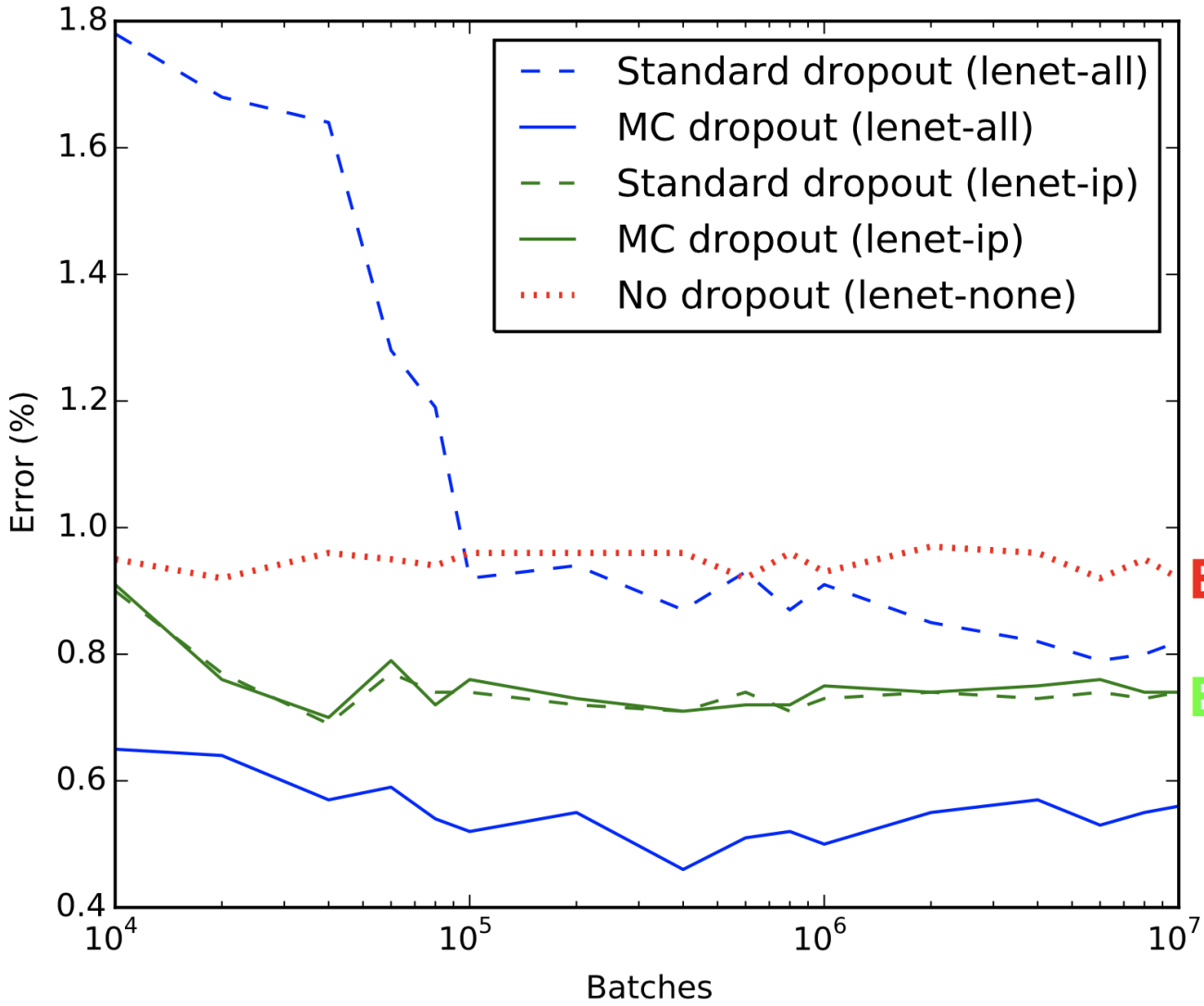
Solid blue: Bayesian LeNet (MC dropout)

Huge improvement (MNIST)



Before dropout

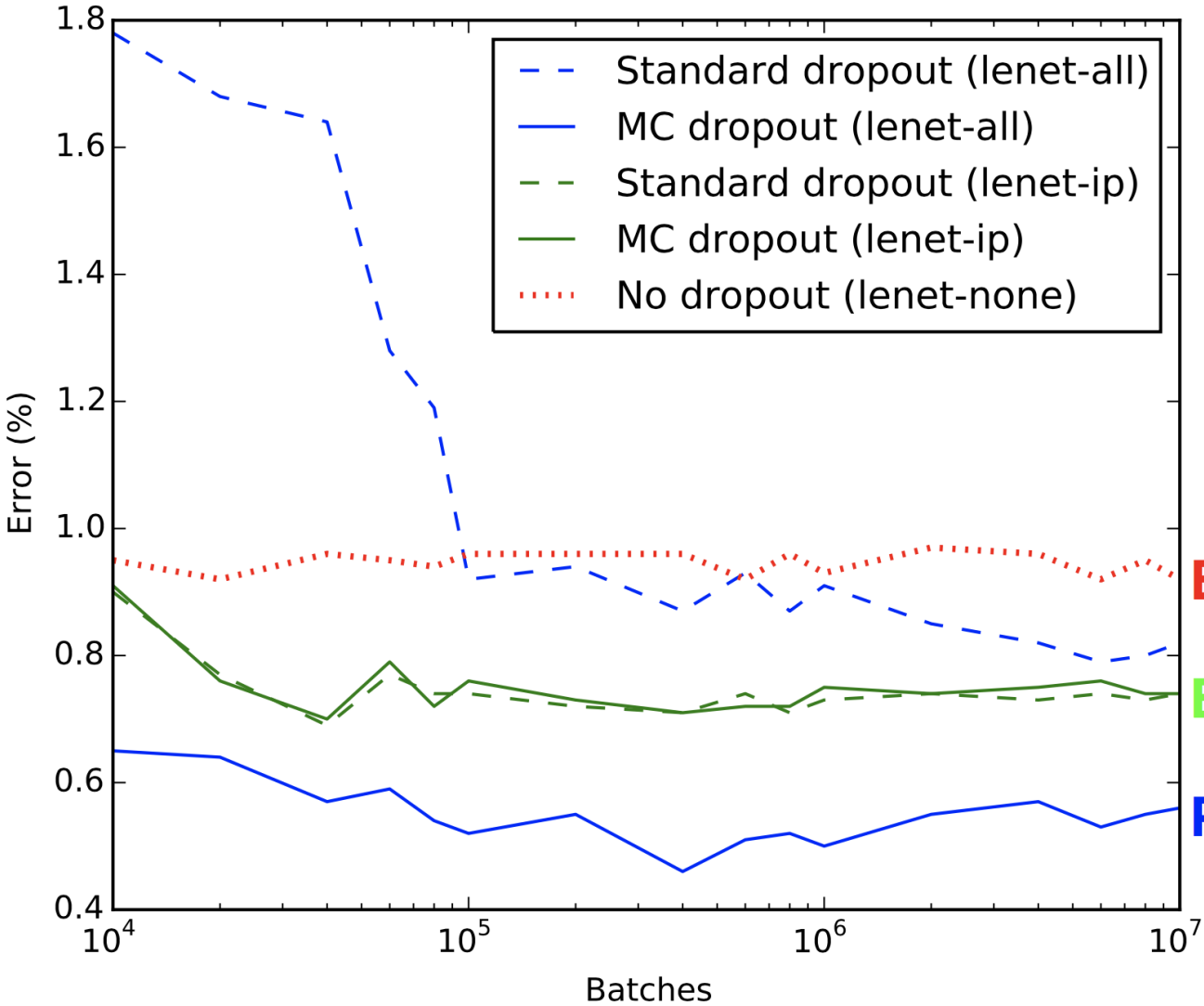
Huge improvement (MNIST)



Before dropout

Empirical dropout

Huge improvement (MNIST)

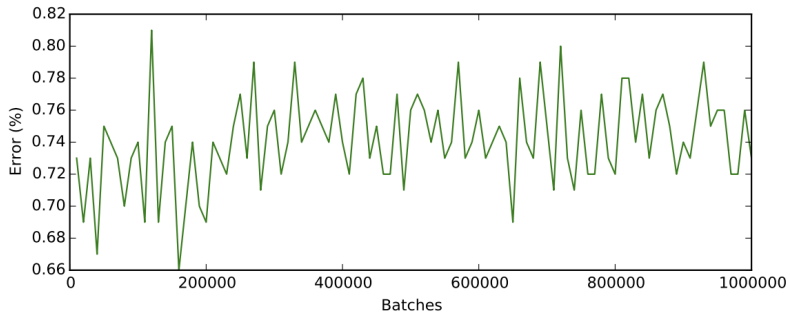


Before dropout

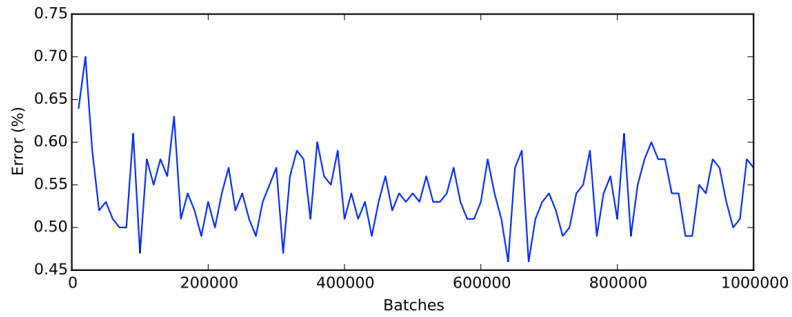
Empirical dropout

Principled dropout

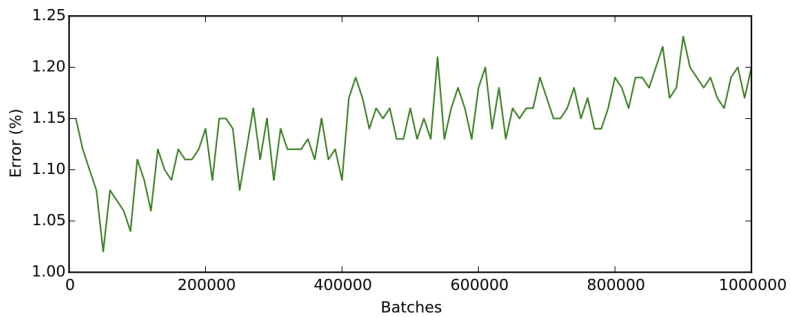
Over-fittting on small data



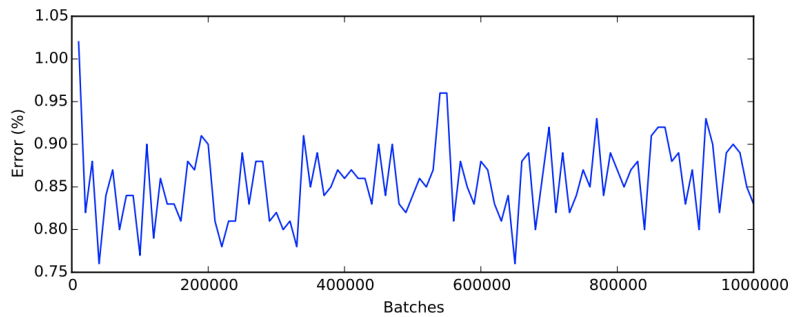
(a) Entire MNIST
Standard dropout convnet



(b) Entire MNIST
Bayesian convnet



(c) 1/4 of MNIST
Standard dropout convnet



(d) 1/4 of MNIST
Bayesian convnet

Results on CIFAR-10

	CIFAR Test Error (and Std.)	
Model	**Standard Dropout**	**MC Dropout**
NIN	10.43 (Lin et al., 2013)	**10.27 ± 0.05**
DSN	9.37 (Lee et al., 2014)	**9.32 ± 0.02**
Augmented-DSN	7.95 (Lee et al., 2014)	**7.71 ± 0.09**

Table: Bayesian techniques (MC dropout) with some typical DL approaches

Predictive Log-likelihood

- Predictive log-likelihood can be approximated by Monte Carlo integration

$$\begin{aligned}\log p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) &= \log \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) d\mathbf{w} \\ &\approx \log \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) q(\mathbf{w}|\boldsymbol{\theta}) d\mathbf{w} \\ &\approx \log \left(\frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}_t) \right), \ \mathbf{w}_t \sim q(\mathbf{w}|\boldsymbol{\theta})\end{aligned}$$

- For regression we get

$$\begin{aligned}\log p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \log \operatorname{sum} \exp \left(-\frac{1}{2} \tau \|\mathbf{y}^* - \widehat{\mathbf{y}}_t\|_2^2 \right) - \log T \\ - \frac{1}{2} \log 2\pi - \frac{1}{2} \log \tau^{-1}\end{aligned}$$

with a log-sum-exp of T terms and $\widehat{\mathbf{y}}_t$ stochastic forward passes through the network

Why does dropout work?

- Because it approximately integrates over model parameters
- The noise is a side-effect of approx. integration
- Explains model over specification , “adaptive model capacity”
- We fit the process that generated our data

Comments on dropout

What can we say about $q(\cdot|\boldsymbol{\theta})$?

- Many Bernoullis = cheap multi-modality
- Dropout at test time $\approx$ propagate the mean $\mathbb{E}(\mathbf{w}_i) = p_i \mathbf{M}_i$
- Constrains the weights to near the origin:
  - Posterior uncertainty decreases with more data
  - We get that
$$\mathrm{Var}(\mathbf{w}_i) = \mathbf{M}_i \mathbf{M}_i^\top (p_i - p_i^2)$$
  - For fixed p_i to decrease uncertainty must decrease $\|\mathbf{M}_i\|$
- Smallest $\|\mathbf{M}_i\|$ = strongest reg. at $p_i = 0.5$
- Can combine model with Bayesian techniques in apractical way ...
- Have uncertainty estimates in the network

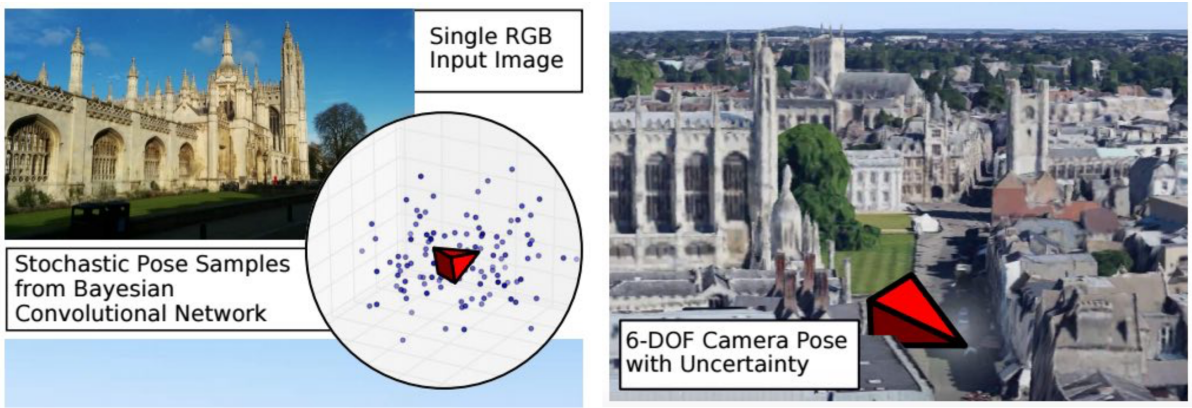
Other stochastic reg. techniques

- Multiplicative Gaussian noise (Srivastava et al.2014)
- Multiply network units by $\mathcal{N}(1, 1)$
- Same performance as dropout
- Multiplicative Gaussian noise as approximate inference

$$\mathbf{z}_{i,j} \sim \mathcal{N}(1, 1),\ i = 1, \ldots, L,\ j = 1, \ldots, K_{i-1}$$
$$\mathbf{w}_i = \mathbf{M}_i \cdot \mathrm{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i})$$
$$q(\mathbf{w}|\boldsymbol{\theta}) = \prod q(\mathbf{w}_i|\mathbf{M}_i)$$

Camera pose localisation

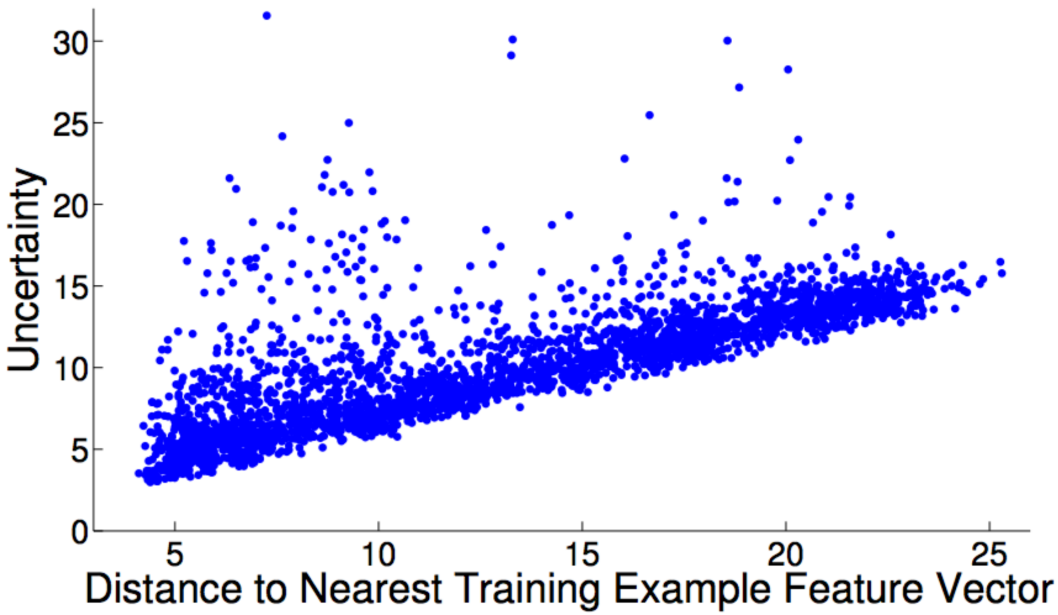
- Find the location from which a picture was taken



- Test photos with high uncertainty (strong occlusion from vehicles, pedestrians or other objects)

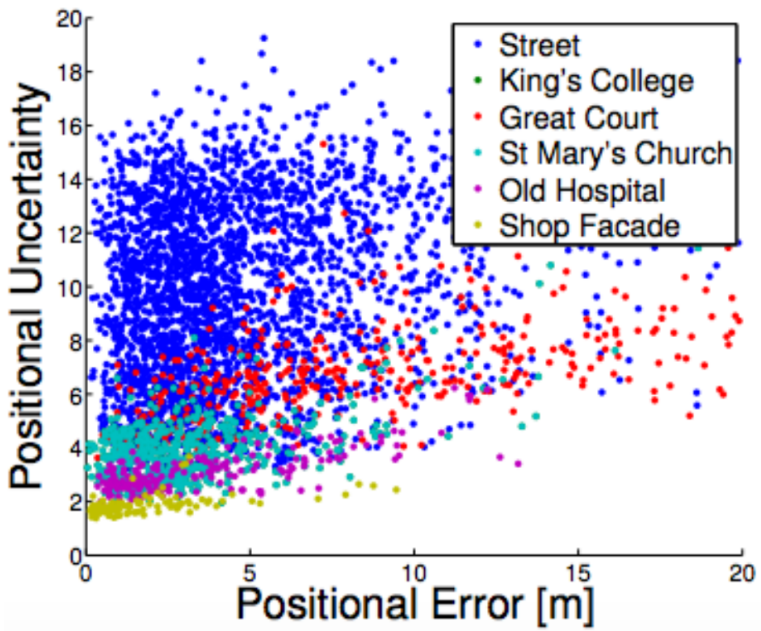
Camera pose localisation

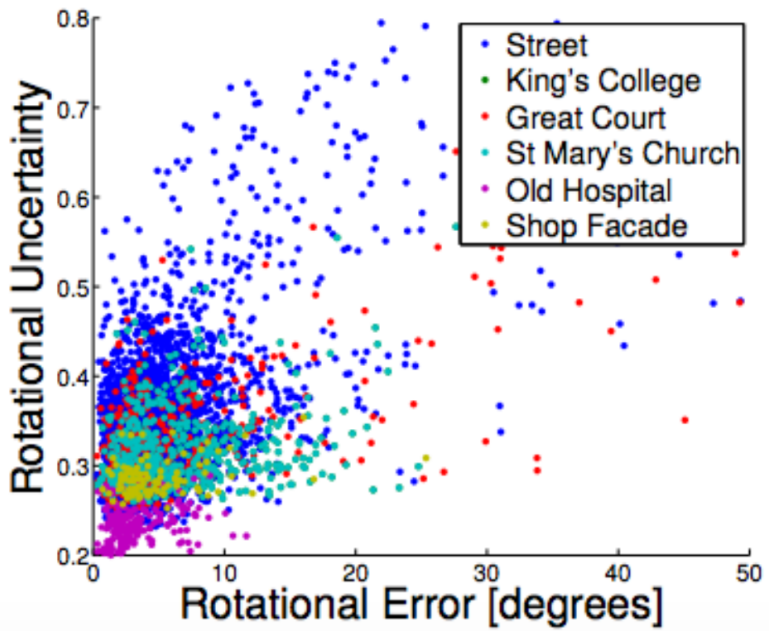
- Where was a picture taken? (Kendall and Cipolla, 2015)
- Uncertainty increases as a test photo diverges from training distribution



Camera pose localisation

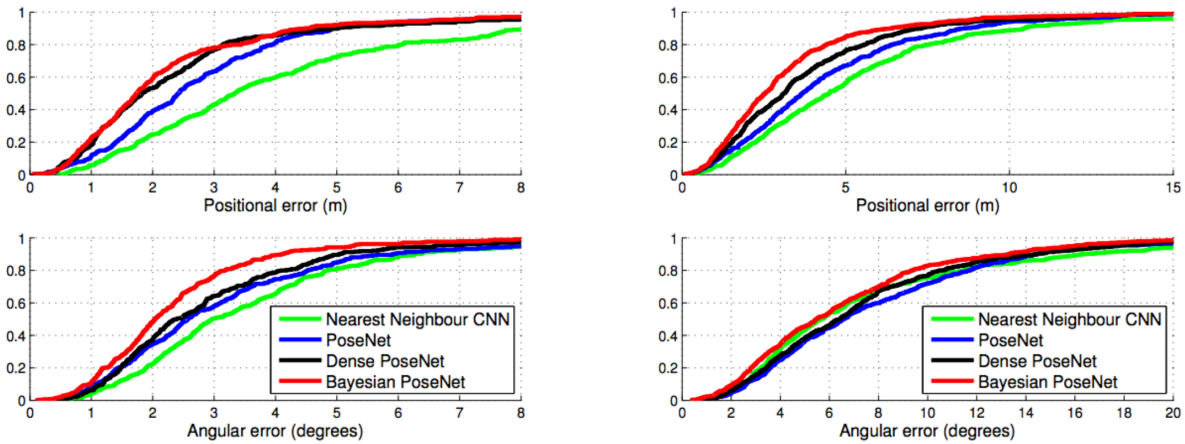
- Where was a picture taken? (Kendall and Cipolla, 2015)
- Test photos with high uncertainty (strong occlusion from vehicles, pedestrians or other objects)
- Uncertainty increases as a test photo diverges from training distribution
- Localisation error correlates with uncertainty





Camera pose localisation

- Kendall and Cipolla (2015) show 10-15% improvement on state-of-the-art with Bayesian convnets



(a) King's College (b) St Mary's Church

Figure – Localisation accuracy for different error thresholds

Model Uncerntainty

- We train a model to recognise dog breeds

Model Uncerntainty

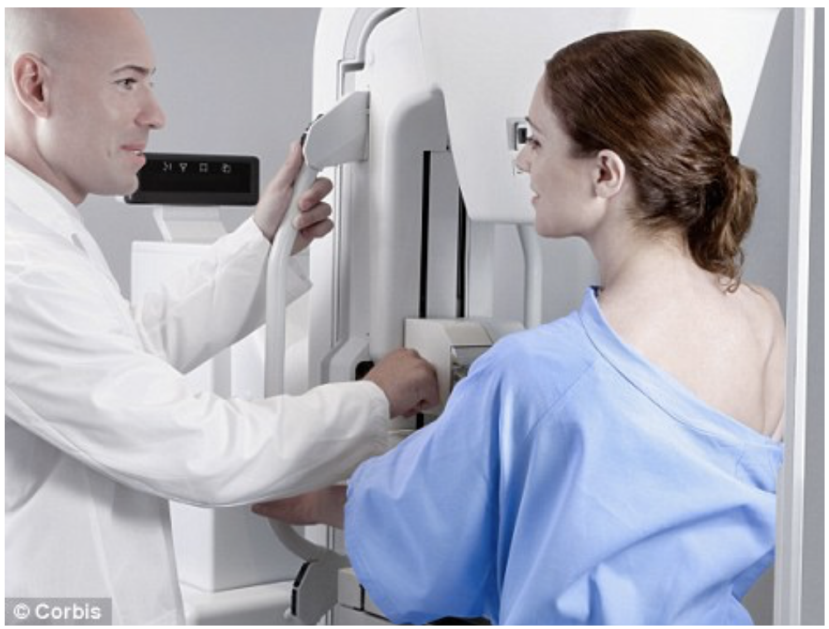
- We train a model to recognise dog breeds
- And are given a cat to classify

Model Uncerntainty

- We train a model to recognise dog breeds
- And are given a cat to classify
- What would you want your model to do?

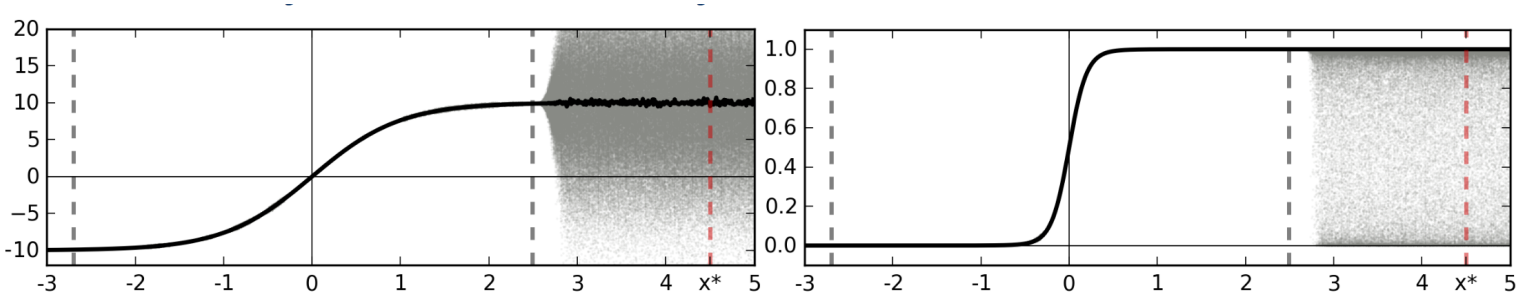
Model Uncerntainty

- We train a model to recognise dog breeds
- And are given a cat to classify
- What would you want your model to do?
- Similar problems in decision making, physics, lifescience, etc.



Model Uncerntainty

- We train a model to recognise dog breeds
- And are given a cat to classify
- What would you want your model to do?
- Similar problems in decision making, physics, lifescience, etc.
- Classifier score is not about the uncertainty!



(a) Softmax *input* as a function of data $\mathbf{x}$: $f(\mathbf{x})$

(b) Softmax *output* as a function of data $\mathbf{x}$: $\sigma(f(\mathbf{x}))$

- We need to be able to tell what our model knows and what it doesn't

- We fit a distribution; the first moment:

$$\widehat{\mathbb{E}}(\mathbf{y}^*) = \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}(\mathbf{x}^*, \widehat{\mathbf{w}}_t)$$

  with $\widehat{\mathbf{w}}_t \sim q(\mathbf{w}|\boldsymbol{\theta})$

- For uncertainty (in regression):

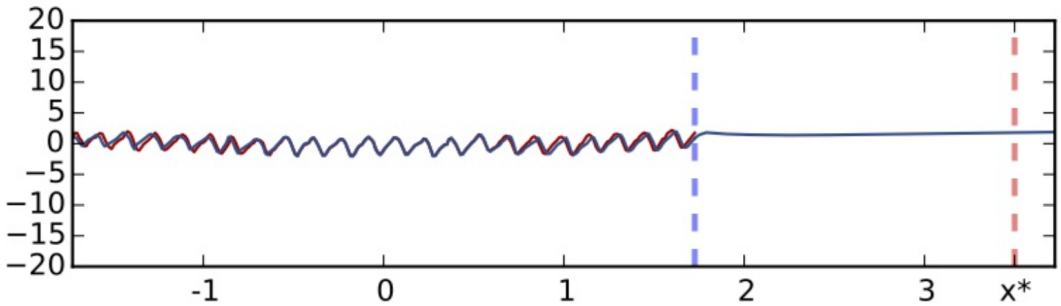
$$\mathrm{Var}(\mathbf{y}^*) = \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}(\mathbf{x}^*, \widehat{\mathbf{w}}_t)^\top\widehat{\mathbf{y}}(\mathbf{x}^*, \widehat{\mathbf{w}}_t) - \widehat{\mathbb{E}}(\mathbf{y}^*)^\top\widehat{\mathbb{E}}(\mathbf{y}^*)$$

- As simple as looking at the sample variance of stochastic forward passes through the network (plus obs. noise)

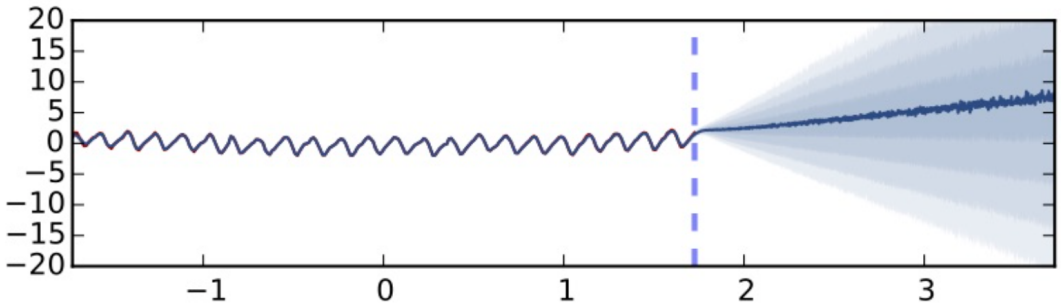
Example

What would be the CO2 concentration level in Mauna Loa, Hawaii, in 20 years' time?

- Normal dropout (weight averaging, 5 layers, ReLU units):



- Same network, Bayesian perspective:

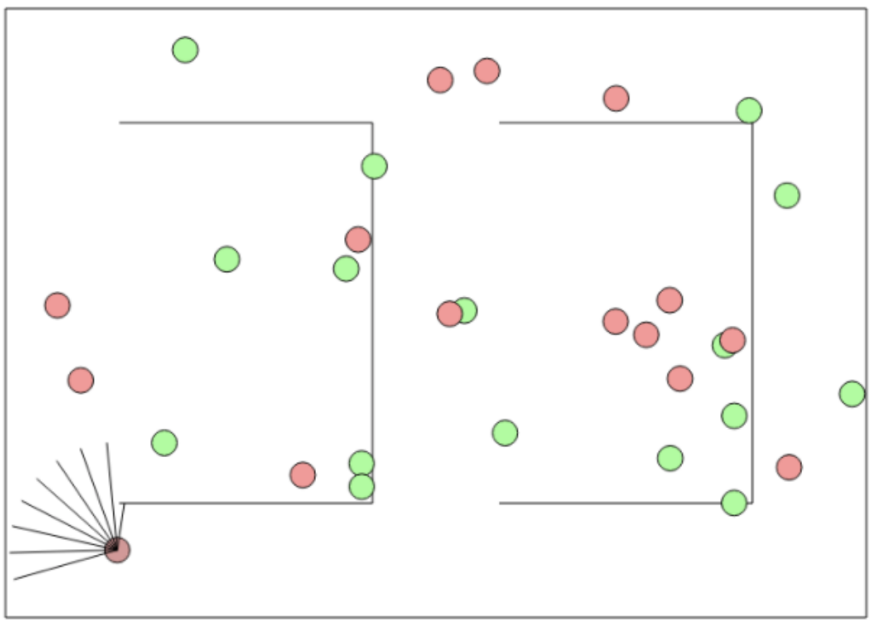


How good is our prediction?

	Avg. Test RMSE and Std. Errors			Avg. Test LL and Std. Errors		
Dataset	**VI**	**PBP**	**Dropout**	**VI**	**PBP**	**Dropout**
Boston Housing	4.32 ±0.29	3.01 ±0.18	**2.97 ±0.85**	-2.90 ±0.07	-2.57 ±0.09	**-2.46 ±0.25**
Concrete Strength	7.19 ±0.12	5.67 ±0.09	**5.23 ±0.53**	-3.39 ±0.02	-3.16 ±0.02	**-3.04 ±0.09**
Energy Efficiency	2.65 ±0.08	1.80 ±0.05	**1.66 ±0.19**	-2.39 ±0.03	-2.04 ±0.02	**-1.99 ±0.09**
Kin8nm	**0.10 ±0.00**	**0.10 ±0.00**	**0.10 ±0.00**	0.90 ±0.01	0.90 ±0.01	**0.95 ±0.03**
Naval Propulsion	**0.01 ±0.00**	**0.01 ±0.00**	**0.01 ±0.00**	3.73 ±0.12	3.73 ±0.01	**3.80 ±0.05**
Power Plant	4.33 ±0.04	4.12 ±0.03	**4.02 ±0.18**	-2.89 ±0.01	-2.84 ±0.01	**-2.80 ±0.05**
Protein Structure	4.84 ±0.03	4.73 ±0.01	**4.36 ±0.04**	-2.99 ±0.01	-2.97 ±0.00	**-2.89 ±0.01**
Wine Quality Red	0.65 ±0.01	0.64 ±0.01	**0.62 ±0.04**	-0.98 ±0.01	-0.97 ±0.01	**-0.93 ±0.06**
Yacht Hydrodynamics	6.89 ±0.67	**1.02 ±0.05**	1.11 ±0.38	-3.43 ±0.16	-1.63 ±0.02	**-1.55 ±0.12**
Year Prediction MSD	9.034 ±NA	8.879 ±NA	**8.849 ±NA**	-3.622 ±NA	-3.603 ±NA	**-3.588 ±NA**

Table 1: **Average test performance in RMSE and predictive log likelihood** for a popular variational inference method (VI, Graves [20]), Probabilistic back-propagation (PBP, Hernández-Lobato and Adams [27]), and dropout uncertainty (Dropout).

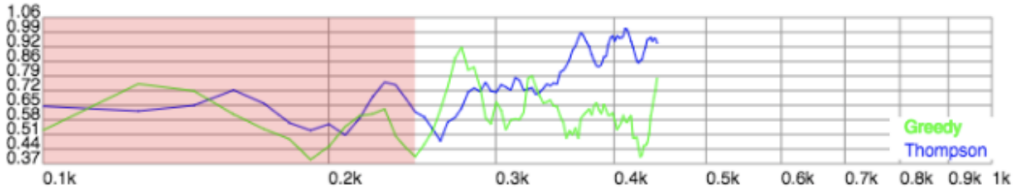
- We consider a "Roomba" (`github.com/karpathy/convnetjs`)
- Penalised -5 for walking into a wall, +10 reward for collecting poisson
- Our environment is stochastic and ever changing
- We want a net to learn what actions to do in different situations

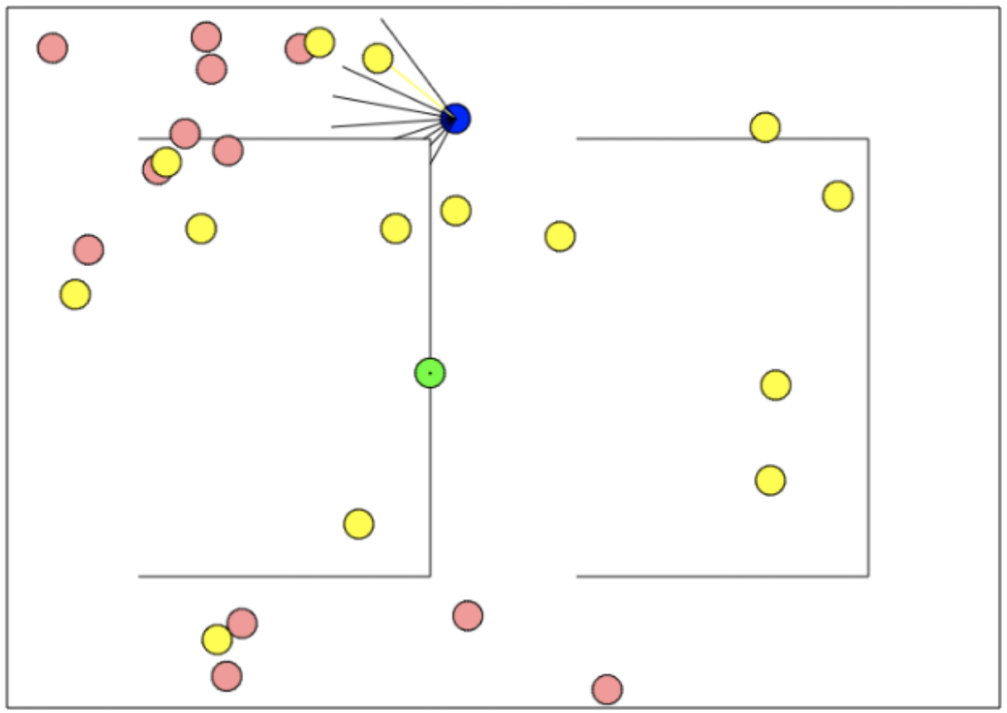


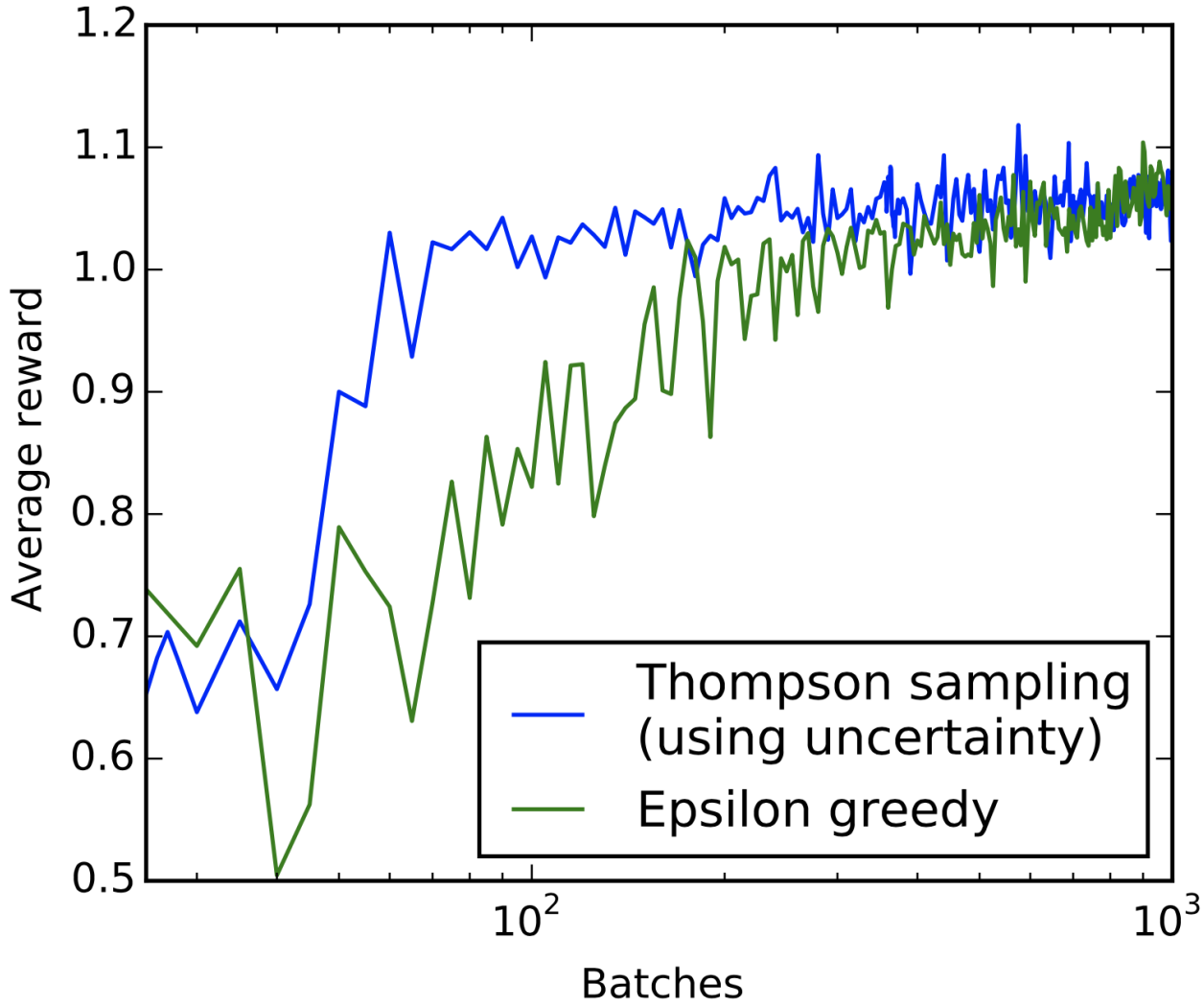
Behavioural policies:

- Epsilon-greedy — take random actions with probability ϵ and optimal actions otherwise
- Using uncertainty we can learn faster
- Thompson sampling — draw realisation from current belief over world, choose action with highest value
- In practice: simulate a stochastic forward pass through the dropout network and choose action with highest value

Deep Reinforcement Learning







Average reward over time (log scale)

Image segmentation

- Scene understanding: what's in a photo and where? (Kendall, Badrinarayanan, and Cipolla, 2015)

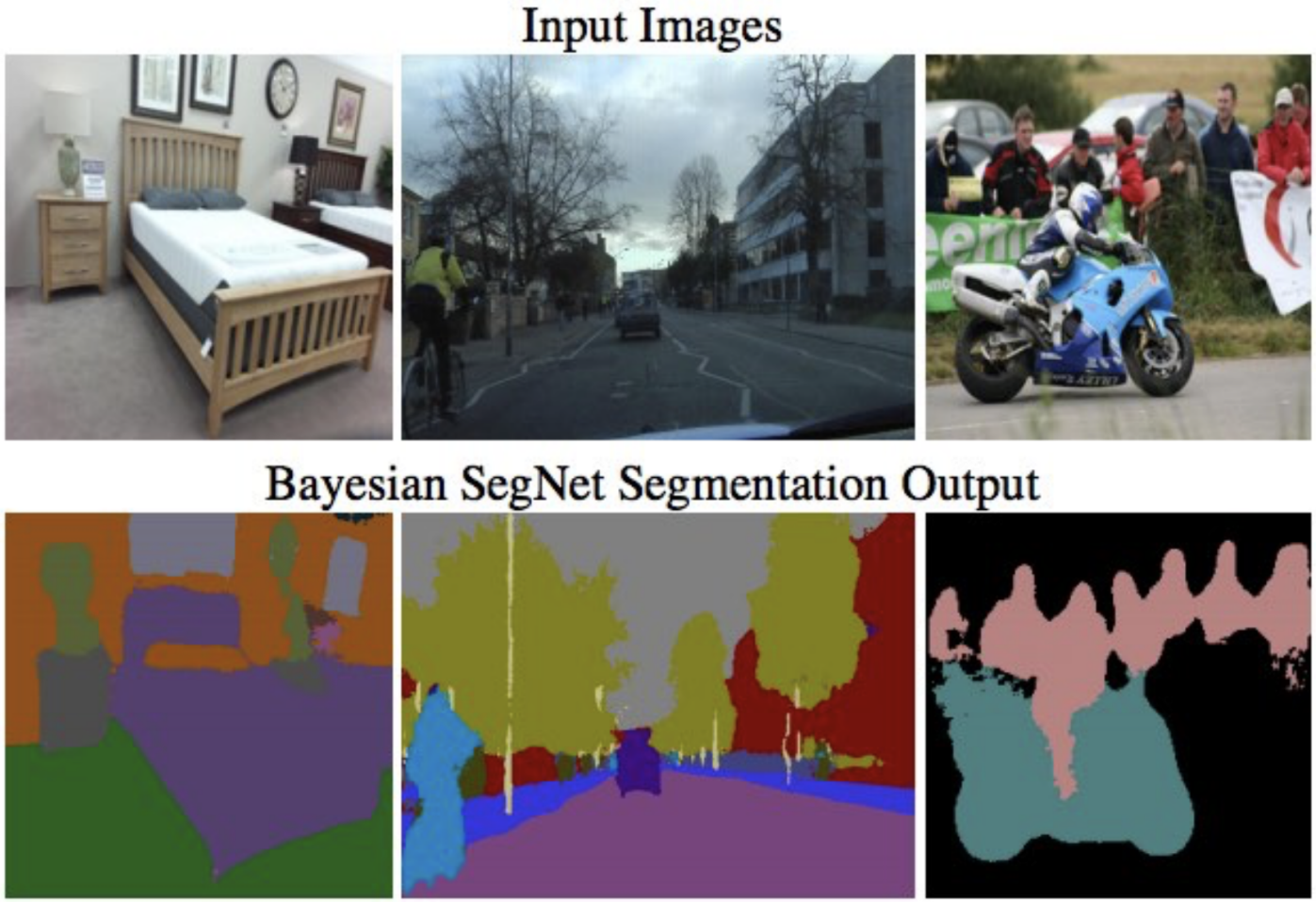


Image segmentation

- Scene understanding: what's in a photo and where? (Kendall, Badrinarayanan, and Cipolla, 2015)