UNIVERSITY OF CAMBRIDGE

# Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference

Yarin Gal

yg279@cam.ac.uk

## Bayesian Convnets

UNIVERSITY OF CAMBRIDGE

In this talk we'll —

- Offer insights into why recent state-of-the-art models in image processing work so well.
- Start with an approximation of a BNP model and through a (beautiful) derivation...
- **... obtain insights into getting state-of-the-art results on CIFAR-10 dataset ($7.71\%$ test error).**

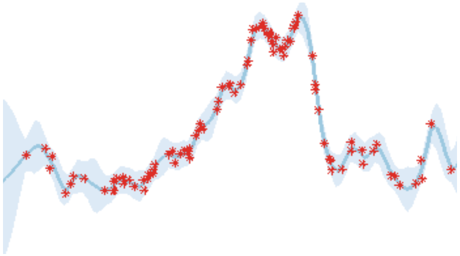
# GP regression

Gaussian processes (GPs) are a powerful tool for probabilistic inference over functions.

- GP regression captures non-linear functions

GPs offer:

- uncertainty estimates,
- robustness to over-fitting,
- and principled ways for tuning hyper-parameters



## GP regression

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

  $$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution
- This implies a joint Gaussian distribution over function values:

  $$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$
- $Y$ consists of noisy observations, making the functions $F$ latent:

  $$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$
- Prior and likelihood conjugate:

  $$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

## GP regression

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

$$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution

- This implies a joint Gaussian distribution over function values:

$$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- $Y$ consists of noisy observations, making the functions $F$ latent:

$$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$

- Prior and likelihood conjugate:

$$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

## GP regression

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

$$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution

- This implies a joint Gaussian distribution over function values:

$$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- $Y$ consists of noisy observations, making the functions $F$ latent:

$$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$

- Prior and likelihood conjugate:

$$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

# GP regression

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

  $$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution
- This implies a joint Gaussian distribution over function values:

  $$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$
- $Y$ consists of noisy observations, making the functions $F$ latent:

  $$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$
- Prior and likelihood conjugate:

  $$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

## GP regression

UNIVERSITY OF CAMBRIDGE

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

$$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution

- This implies a joint Gaussian distribution over function values:

$$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- $Y$ consists of noisy observations, making the functions $F$ latent:

$$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$

- Prior and likelihood conjugate:

$$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

## GP regression

- Training dataset with $N$ inputs $X \in \mathbb{R}^{N \times Q}$ ($Q$ dimensional)
- Corresponding $D$ dimensional outputs $F_n = \mathbf{f}(X_n)$
- We place a *Gaussian process prior* over the space of functions

$$\mathbf{f} \sim \mathcal{GP}(\text{mean } \mu(\mathbf{x}), \text{covariance } k(\mathbf{x}, \mathbf{x}'))$$

  Every finite subset of variables follows a joint Gaussian distribution

- This implies a joint Gaussian distribution over function values:

$$p(F|X) = \mathcal{N}(F; \mu(X), K), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- $Y$ consists of noisy observations, making the functions $F$ latent:

$$p(Y|F) = \mathcal{N}(Y; F, \tau^{-1} I_n)$$

- Prior and likelihood conjugate:

$$p(Y|X) = \mathcal{N}(Y; 0, K + \tau^{-1} I_n)$$

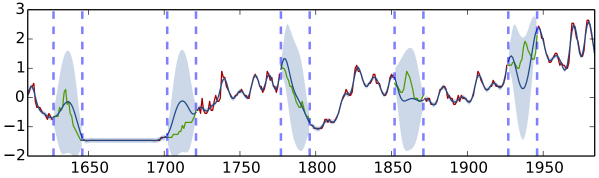
## However...

Problem – time and space complexity

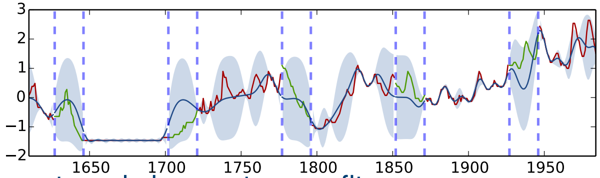
- Evaluating $p(Y|X)$ directly is an expensive operation
- Involves the inversion of the $N$ by $N$ matrix $K$
- requiring $\mathcal{O}(N^3)$ time complexity

## Many Approximations

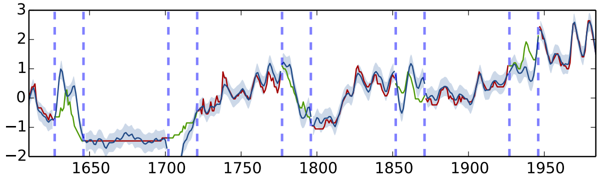
**Full GP:**



- Sparse pseudo-input cannot handle complex functions well:



- Sparse spectrum is known to over-fit:



# Main Idea:

- **Variational Sparse Spectrum GP (VSSGP)**
  - use variational inference for the sparse spectrum approximation
  - avoids over-fitting, efficiently captures globally complex behaviour
- **In short—**
  - we replace the GP covariance function with a finite Monte Carlo approximation
  - we view this as a **random covariance function**
  - conditioned on data this random variable has an intractable posterior
  - we approximate this posterior with variational inference

## Main Idea:

- **Variational Sparse Spectrum GP (VSSGP)**
  - use variational inference for the sparse spectrum approximation
  - avoids over-fitting, efficiently captures globally complex behaviour
- **In short—**
  - we replace the GP covariance function with a finite Monte Carlo approximation
  - we view this as a **random covariance function**
  - conditioned on data this random variable has an intractable posterior
  - we approximate this posterior with variational inference

## Main Idea:

- **Variational Sparse Spectrum GP (VSSGP)**
  - use variational inference for the sparse spectrum approximation
  - avoids over-fitting, efficiently captures globally complex behaviour
- **In short—**
  - we replace the GP covariance function with a finite Monte Carlo approximation
  - we view this as a **random covariance function**
  - conditioned on data this random variable has an intractable posterior
  - we approximate this posterior with variational inference

## Main Idea:

- **Variational Sparse Spectrum GP (VSSGP)**
  - use variational inference for the sparse spectrum approximation
  - avoids over-fitting, efficiently captures globally complex behaviour
- **In short—**
  - we replace the GP covariance function with a finite Monte Carlo approximation
  - we view this as a **random covariance function**
  - conditioned on data this random variable has an intractable posterior
  - we approximate this posterior with variational inference

## Main Idea:

- **Variational Sparse Spectrum GP (VSSGP)**
  - use variational inference for the sparse spectrum approximation
  - avoids over-fitting, efficiently captures globally complex behaviour
- **In short—**
  - we replace the GP covariance function with a finite Monte Carlo approximation
  - we view this as a **random covariance function**
  - conditioned on data this random variable has an intractable posterior
  - we approximate this posterior with variational inference

## 2-Slide Review of Variational Inference

- Condition model on a finite set of random variables $\omega$.
- Predictive distribution

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

- Can't evaluate $p(\omega|\mathbf{X}, \mathbf{Y})$ analytically —
- define an "easier" approximating *variational* distribution $q_\theta(\omega)$ parametrised by variational parameters $\theta$.

## 2-Slide Review of Variational Inference

- Minimise the Kullback–Leibler (KL) divergence:

$$\text{argmin}_\theta \text{KL}(q_\theta(\omega) \mid p(\omega|\mathbf{X}, \mathbf{Y})).$$

- Minimising KL = maximising *log evidence lower bound* with respect to $\theta$:

$$\mathcal{L}_{\text{VI}} := \int q_\theta(\omega) \log p(\mathbf{Y}|\mathbf{X}, \omega) \text{d}\omega - \text{KL}(q_\theta(\omega)||p(\omega)).$$

- Gives approximate predictive distribution:

$$q_\theta(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}^*|\mathbf{x}^*, \omega) q_\theta(\omega) \text{d}\omega.$$

**How do we apply this to our GP situation?** (with a squared exponential covariance function)

**Given Fourier transform of the covariance function:**

$$\begin{aligned}\mathbf{K}(\mathbf{x}-\mathbf{y}) &= \sigma^2 e^{-\frac{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})}{2}} \\ &= \sigma^2 \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) \cos\big(2\pi \mathbf{w}^T(\mathbf{x}-\mathbf{y})\big) d\mathbf{w}.\end{aligned}$$

Fourier transform of the squared exponential covariance function:

$$\mathbf{K}(\mathbf{x}-\mathbf{y}) = \sigma^2 \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) \cos\left(2\pi \mathbf{w}^T(\mathbf{x}-\mathbf{y})\right) \mathrm{d}\mathbf{w},$$

**Auxiliary variable *b*:**

$$\mathbf{K}(\mathbf{x}-\mathbf{y}) = 2\sigma^2 \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) \mathrm{Unif}[0, 2\pi] \\ \cos\left(2\pi \mathbf{w}^T \mathbf{x} + b\right) \cos\left(2\pi \mathbf{w}^T \mathbf{y} + b\right) \mathrm{d}\mathbf{w}\mathrm{d}b.$$

Auxiliary variable $b$:

$$\mathbf{K}(\mathbf{x}-\mathbf{y}) = 2\sigma^2 \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q)\mathrm{Unif}[0, 2\pi] \cos\left(2\pi\mathbf{w}^T\mathbf{x} + b\right)\cos\left(2\pi\mathbf{w}^T\mathbf{y} + b\right)\mathrm{d}\mathbf{w}\mathrm{d}b,$$

**Monte Carlo integration with $K$ terms:**

$$\widehat{\mathbf{K}}(\mathbf{x}-\mathbf{y}) = \frac{2\sigma^2}{K}\sum_{k=1}^{K}\cos\left(2\pi\mathbf{w}_k^T\mathbf{x} + b_k\right)\cos\left(2\pi\mathbf{w}_k^T\mathbf{y} + b_k\right)$$

with $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{I}_Q)$, $b_k \sim \mathrm{Unif}[0, 2\pi]$.

Monte Carlo integration with $K$ terms:

$$\widehat{\mathbf{K}}(\mathbf{x}-\mathbf{y}) = \frac{2\sigma^2}{K}\sum_{k=1}^{K}\cos\left(2\pi\mathbf{w}_k^T\mathbf{x}+b_k\right)\cos\left(2\pi\mathbf{w}_k^T\mathbf{y}+b_k\right),$$

**Rewrite the covariance function with $\Phi \in \mathbb{R}^{N\times K}$**

$$\mathbf{w}_k \sim \mathcal{N}(0,\mathbf{I}_Q), \quad b_k \sim \mathrm{Unif}[0,2\pi], \quad \boldsymbol{\omega} = \{\mathbf{w}_k, b_k\}_{k=1}^{K}$$

$$\Phi_{n,k}(\boldsymbol{\omega}) = \sqrt{\frac{2\sigma^2}{K}}\cos\left(2\pi\mathbf{w}_k^T\mathbf{x}_n+b_k\right),$$

$$\widehat{\mathbf{K}}(\mathbf{x}-\mathbf{y}) = \Phi(\boldsymbol{\omega})\Phi(\boldsymbol{\omega})^T.$$

Rewrite the covariance function with $\Phi \in \mathbb{R}^{N \times K}$

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{I}_Q), \quad b_k \sim \text{Unif}[0, 2\pi], \quad \omega = \{\mathbf{w}_k, b_k\}_{k=1}^K$$

$$\Phi_{n,k}(\omega) = \sqrt{\frac{2\sigma^2}{K}} \cos\left(2\pi \mathbf{w}_k^T \mathbf{x}_n + b_k\right),$$

$$\widehat{\mathbf{K}}(\mathbf{x} - \mathbf{y}) = \Phi(\omega)\Phi(\omega)^T,$$

## Integrate the GP over the random covariance function

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{I}_Q), \quad b_k \sim \text{Unif}[0, 2\pi], \quad \omega = \{\mathbf{w}_k, b_k\}_{k=1}^K$$

$$p(\mathbf{Y}|\mathbf{X}, \omega) = \mathcal{N}\left(\mathbf{Y}; \mathbf{0}, \Phi(\omega)\Phi(\omega)^T + \tau^{-1}\mathbf{I}_N\right)$$

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \omega) p(\omega) \mathrm{d}\omega$$

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

Integrate the GP over the random covariance function

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{I}_Q), \quad b_k \sim \text{Unif}[0, 2\pi], \quad \omega = \{\mathbf{w}_k, b_k\}_{k=1}^K$$
$$p(\mathbf{Y}|\mathbf{X}, \omega) = \mathcal{N}\left(\mathbf{Y}; \mathbf{0}, \Phi(\omega)\Phi(\omega)^T + \tau^{-1}\mathbf{I}_N\right)$$
$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \omega)p(\omega)\mathrm{d}\omega$$
$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \omega)p(\omega|\mathbf{X}, \mathbf{Y})\mathrm{d}\omega.$$

**Use variational distribution $q(\omega) = \prod q(\mathbf{w}_k)q(b_k)$ to approximate posterior $p(\omega|\mathbf{X}, \mathbf{Y})$:**

$$q(\mathbf{w}_k) = \mathcal{N}(\mu_k, \Sigma_K), \quad q(b_k) = \text{Unif}(\alpha_k, \beta_k),$$

with $\Sigma_K$ diagonal.

## Maximise log evidence lower bound

$$\mathcal{L}_{VSSGP} = \frac{1}{2}\sum_{d=1}^{D}\Big(\log(|\tau^{-1}\boldsymbol{\Sigma}|) + \tau \mathbf{y}_d^T E_{q(\omega)}(\Phi)\, \boldsymbol{\Sigma}\, E_{q(\omega)}(\Phi^T)\, \mathbf{y}_d + \ldots\Big) - \mathrm{KL}(q(\omega)||p(\omega))$$

with $\boldsymbol{\Sigma} = (E_{q(\omega)}(\Phi^T\Phi) + \tau^{-1}I)^{-1}$. We can evaluate the KL and the expectations analytically using the identity

$$E_{q(\mathbf{w})}\big(\cos(\mathbf{w}^T\mathbf{x} + b)\big) = e^{-\frac{1}{2}\mathbf{x}^T\Sigma\mathbf{x}}\cos(\mu^T\mathbf{x} + b).$$

Requires $\mathcal{O}(NK^2 + K^3)$ **time complexity**.

## Factorised VSSGP

UNIVERSITY OF CAMBRIDGE

**Factorised VSSGP (fVSSGP)**

- We often use large $K$.
- $K$ by $K$ matrix inversion is still slow: $\mathcal{O}(K^3)$.
- **It is silly to invert the whole matrix every time**
— slightly changing the parameters we expect the inverse to not change too much.
- We can do better with an additional **auxiliary variable**.

We integrated the GP over the random covariance function

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{I}_Q), \quad b_k \sim \text{Unif}[0, 2\pi], \quad \omega = \{\mathbf{w}_k, b_k\}_{k=1}^{K}$$

$$p(\mathbf{Y}|\mathbf{X}, \omega) = \mathcal{N}\big(\mathbf{Y}; \mathbf{0}, \Phi(\omega)\Phi(\omega)^T + \tau^{-1}\mathbf{I}_N\big)$$

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \omega)p(\omega)\text{d}\omega,$$

**Introduce auxiliary random variables $\mathbf{A} \in \mathbb{R}^{K \times D}$**

$$\mathbf{A} \sim \mathcal{N}(0, \mathbf{I}_{K \times D}),$$

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{A}, \omega) = \mathcal{N}\big(\mathbf{Y}; \Phi(\omega)\mathbf{A}, \tau^{-1}\mathbf{I}_N\big)$$

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, A, \omega)p(A)p(\omega)\text{d}\omega\text{d}\mathbf{A}.$$

**Introduce auxiliary random variables A** $\in \mathbb{R}^{K \times D}$

$$\mathbf{A} \sim \mathcal{N}(0, \mathbf{I}_{K \times D}),$$
$$p(\mathbf{Y}|\mathbf{X}, \mathbf{A}, \omega) = \mathcal{N}\left(\mathbf{Y}; \Phi(\omega)\mathbf{A}, \tau^{-1}\mathbf{I}_N\right)$$
$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, A, \omega)p(A)p(\omega)\mathrm{d}\omega\mathrm{d}\mathbf{A},$$

**Use variational distribution** $q(\omega) = \prod q(\mathbf{w}_k)q(b_k) \prod q(\mathbf{a}_d)$ to approximate posterior $p(\omega, \mathbf{A}|\mathbf{X}, \mathbf{Y})$:

$$q(\mathbf{a}_d) = \mathcal{N}(\mathbf{m}_d, \mathbf{s}_d)$$

over the rows of $A$ with $\mathbf{s}_d$ diagonal.

## Maximise log evidence lower bound

$$\mathcal{L}_{fVSSGP} = \sum_{d=1}^{D} \Big( \tau \mathbf{y}_d^T E_{q(\omega)}(\Phi)\mathbf{m}_d - \frac{\tau}{2}\mathrm{tr}\big( E_{q(\omega)}(\Phi^T \Phi)(\mathbf{s}_d + \mathbf{m}_d \mathbf{m}_d^T) \big)$$

$$+ \ldots \Big) - \mathrm{KL}(q(\mathbf{A})||p(\mathbf{A})) - \mathrm{KL}(q(\omega)||p(\omega)).$$

Requires $\mathcal{O}(NK^2)$ **time complexity** — no matrix inversion.

## Looks Familiar?

**Let's rewrite the last model with different notation:**

$$\mathbf{W}_1 \sim \mathcal{N}(0, \mathbf{I}_{Q \times K}), \quad \mathbf{W}_2 \sim \mathcal{N}(0, \mathbf{I}_{K \times D}),$$

$$\omega = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}\}$$

$$p(\mathbf{y}|\mathbf{x}, \omega) = \mathcal{N}\big(\mathbf{y}; \mathbf{W}_2 \cos\big(\mathbf{W}_1 \mathbf{x} + \mathbf{b}\big), \tau^{-1}\mathbf{I}_N\big)$$

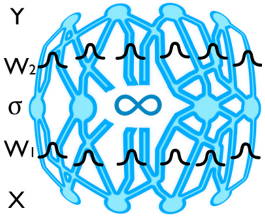
$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \omega) p(\omega|\mathbf{X}, \mathbf{Y}) \mathrm{d}\omega.$$

## Looks Familiar?

**Let's rewrite the last model with different notation:**

$$\mathbf{W}_1 \sim \mathcal{N}(0, \mathbf{I}_{Q\times K}), \quad \mathbf{W}_2 \sim \mathcal{N}(0, \mathbf{I}_{K\times D}),$$
$$\omega = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}\}$$
$$p(\mathbf{y}|\mathbf{x}, \omega) = \mathcal{N}\Big(\mathbf{y}; \mathbf{W}_2 \cos\big(\mathbf{W}_1\mathbf{x} + \mathbf{b}\big), \tau^{-1}\mathbf{I}_N\Big)$$
$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \omega)p(\omega|\mathbf{X}, \mathbf{Y})\mathrm{d}\omega.$$

This is a **Bayesian neural network**.



## Looks Familiar?

- Started with a GP and using variational inference $\rightarrow$ Bayesian neural network
- We (approximately) integrate over the weights of the NN
- But we have weird cosine non-linearities
- Let's replace the covariance function with

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) p(b) \sigma(\mathbf{w}^T \mathbf{x} + b) \sigma(\mathbf{w}^T \mathbf{y} + b) \mathrm{d}\mathbf{w} \mathrm{d}b$$

  with non-linear function $\sigma(\cdot)$ (ReLU/TanH) and distribution $p(b)$
- We get $\sigma(\cdot)$ non-linearities (ReLU/TanH) in our Bayesian NN

## Looks Familiar?

- Started with a GP and using variational inference $\rightarrow$ Bayesian neural network
- We (approximately) integrate over the weights of the NN
- But we have weird cosine non-linearities
- Let's replace the covariance function with

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) p(b) \sigma(\mathbf{w}^T \mathbf{x} + b) \sigma(\mathbf{w}^T \mathbf{y} + b) \mathrm{d}\mathbf{w} \mathrm{d}b$$

  with non-linear function $\sigma(\cdot)$ (ReLU/TanH) and distribution $p(b)$
- We get $\sigma(\cdot)$ non-linearities (ReLU/TanH) in our Bayesian NN

## Looks Familiar?

- Started with a GP and using variational inference $\rightarrow$ Bayesian neural network
- We (approximately) integrate over the weights of the NN
- But we have weird cosine non-linearities
- Let's replace the covariance function with

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) p(b) \sigma(\mathbf{w}^T \mathbf{x} + b) \sigma(\mathbf{w}^T \mathbf{y} + b) \mathrm{d}\mathbf{w} \mathrm{d}b$$

  with non-linear function $\sigma(\cdot)$ (ReLU/TanH) and distribution $p(b)$
- We get $\sigma(\cdot)$ non-linearities (ReLU/TanH) in our Bayesian NN

## Looks Familiar?

- Started with a GP and using variational inference $\rightarrow$ Bayesian neural network
- We (approximately) integrate over the weights of the NN
- But we have weird cosine non-linearities
- Let's replace the covariance function with

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) p(b) \sigma(\mathbf{w}^T \mathbf{x} + b) \sigma(\mathbf{w}^T \mathbf{y} + b) \mathrm{d}\mathbf{w} \mathrm{d}b$$

  with non-linear function $\sigma(\cdot)$ (ReLU/TanH) and distribution $p(b)$

- We get $\sigma(\cdot)$ non-linearities (ReLU/TanH) in our Bayesian NN

## Looks Familiar?

- Started with a GP and using variational inference $\rightarrow$ Bayesian neural network
- We (approximately) integrate over the weights of the NN
- But we have weird cosine non-linearities
- Let's replace the covariance function with

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}_Q) p(b) \sigma(\mathbf{w}^T \mathbf{x} + b) \sigma(\mathbf{w}^T \mathbf{y} + b) \mathrm{d}\mathbf{w} \mathrm{d}b$$

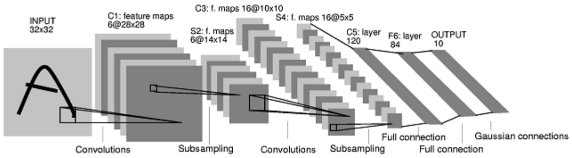
  with non-linear function $\sigma(\cdot)$ (ReLU/TanH) and distribution $p(b)$

- We get $\sigma(\cdot)$ non-linearities (ReLU/TanH) in our Bayesian NN

## Convolutional NNs

We have **Bayesian NNs with arbitrary non-linearities** approximating various Gaussian processes.

- Replace the neural network with a convolutional neural network

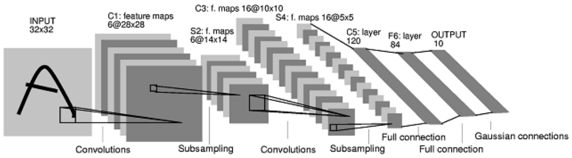


- Convolution operation = inner-product of transformed input
- Integrate over the filters
- But these are often HUGE
- We have wayy too many parameters in our approximation

## Convolutional NNs

We have **Bayesian NNs with arbitrary non-linearities** approximating various Gaussian processes.

- Replace the neural network with a convolutional neural network

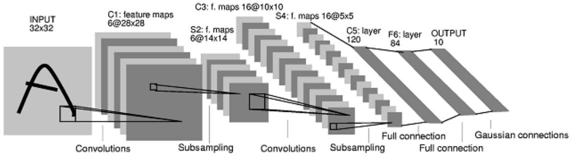


- Convolution operation = inner-product of transformed input
- Integrate over the filters
- But these are often HUGE
- We have wayy too many parameters in our approximation

## Convolutional NNs

We have **Bayesian NNs with arbitrary non-linearities** approximating various Gaussian processes.

- Replace the neural network with a convolutional neural network

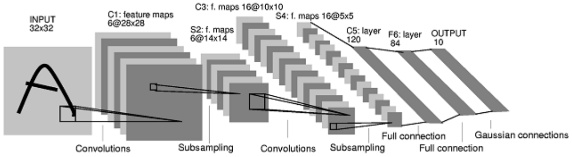


- Convolution operation = inner-product of transformed input
- Integrate over the filters
- But these are often HUGE
- We have wayy too many parameters in our approximation

## Convolutional NNs

We have **Bayesian NNs with arbitrary non-linearities** approximating various Gaussian processes.

- Replace the neural network with a convolutional neural network

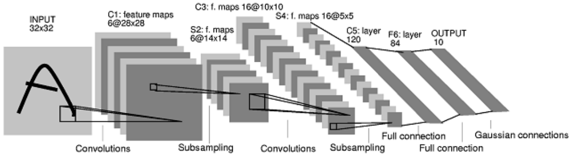


- Convolution operation = inner-product of transformed input
- Integrate over the filters
- But these are often HUGE
- We have wayy too many parameters in our approximation

## Convolutional NNs

We have **Bayesian NNs with arbitrary non-linearities** approximating various Gaussian processes.

- Replace the neural network with a convolutional neural network



- Convolution operation = inner-product of transformed input
- Integrate over the filters
- But these are often HUGE
- We have wayy too many parameters in our approximation

## Computational Efficiency

**Too many parameters...**

- Gaussian approximating distributions $\rightarrow$ Bernoullis
- Random weights defined as $\mathbf{W}_i = \mathbf{M}_i B_i$ with variational parameters $\mathbf{M}_i$ and $B_i$ diagonal: $B_{i,jj} \sim \text{Bern}(p_i)$
- Doesn't use more parameters than normal NNs

## Computational Efficiency

**Problem**: can't integrate analytically

- Use MC integration instead with stochastic optimisation

**In practice**

- Sample Bernoulli realisations and multiply rows of $\mathbf{M}_i$
  $\rightarrow$ identical to setting NN units to zero with probability $p_i$

## Computational Efficiency

**Problem**: can't integrate analytically

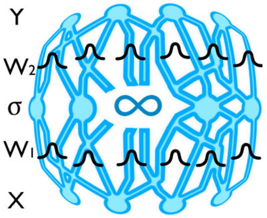
- Use MC integration instead with stochastic optimisation

**In practice**

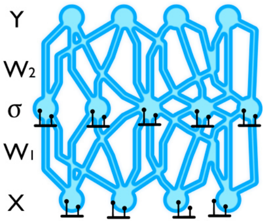
- Sample Bernoulli realisations and multiply rows of $\mathbf{M}_i$
  $\rightarrow$ identical to setting NN units to zero with probability $p_i$

This is **dropout**

(an empirical technique in deep learning to avoid over-fitting)



$\rightarrow$



## Bayesian convnets

UNIVERSITY OF CAMBRIDGE

So...

**Dropout = Bayesian NN = Bernoulli approximate variational inference in GP**

- Can implement Bayesian convnets with Bernoulli approximate variational inference with existing tools!
- Dropout implemented in every deep learning package
- Just do dropout after every convolution layer

**Someone must have tried it in the past?**

## Bayesian convnets

**Dropout after every convolution layer:**

- On CIFAR-10 with a small LeNet model
- Implemented only after inner-product layers in existing literature — 23.46 test error
- We got test error 41.82.

## Bayesian convnets

**Why?**

- Dropout implementation uses full weight matrices at test time

**But...**

## Bayesian convnets

**Why?**

- Dropout implementation uses full weight matrices at test time

**But...**

- Dropout is a Bayesian model

- Should estimate mean of posterior (e.g. MC integration):

$$
\begin{aligned}
p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) &= \int p(\mathbf{y}^*|\mathbf{x}^*, \omega) p(\omega|\mathbf{X}, \mathbf{Y}) \mathrm{d}\omega \\
&\approx \int p(\mathbf{y}^*|\mathbf{x}^*, \omega) q(\omega) \mathrm{d}\omega \\
&\approx \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \omega_t) q(\omega_t), \quad \big(\omega_t \sim q(\omega)\big).
\end{aligned}
$$

**referred to as MC dropout.**

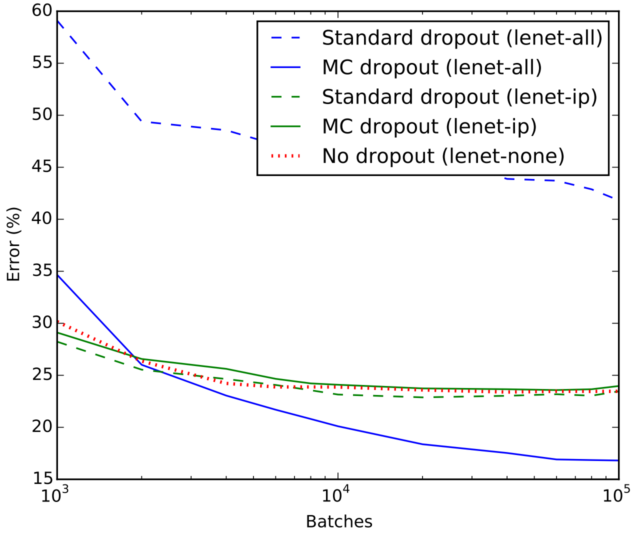
## Bayesian convnets

**MC dropout after every convolution layer**:

- On CIFAR-10 with a small LeNet model
- Training same as before, at test time average $T$ stochastic samples from the network
- **We get** $16.05 \pm 0.07$ **test error averaging 100 samples!** (41.82 before, 23.46 after inner-product alone)

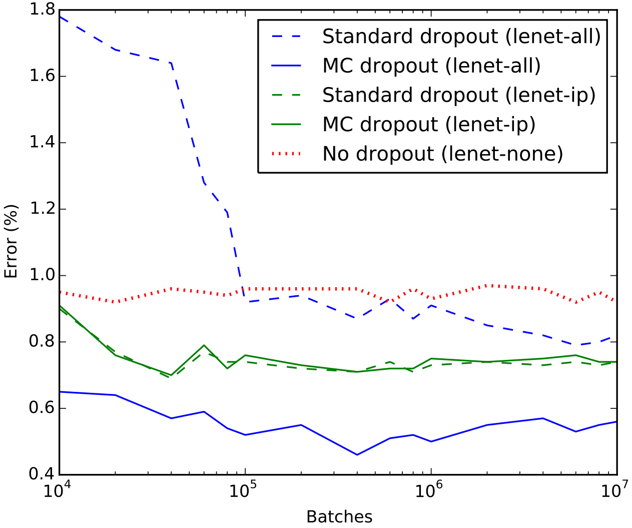
## Bayesian convnets

### How do we compare to standard techniques? (CIFAR-10 LeNet)



## Bayesian convnets

### How do we compare to standard techniques? (MNIST LeNet)



## State of the art

MC dropout in state-of-the-art models ($T = 100$ averaged with 5 repetitions):

| **Model** | **Test error (Standard dropout)** | **Test error (MC dropout)** |
|---|---|---|
| NIN | 10.43 | $\mathbf{10.27 \pm 0.05}$ |
| DSN | 9.37 | $9.32 \pm 0.02$ |
| Augmented-DSN | 7.95 | $\mathbf{7.71 \pm 0.09}$ |

Lowest error obtained is **7.51**

## What's next

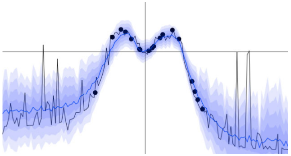
Many new insights

- Existing techniques in deep learning approximate Bayesian non-parametrics models

- Dropout integrates over network weights

- Dropout approximation doesn't work in convnets but integrating over the weights is still good

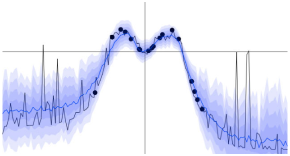
- Opens the door for **many new applications** (model uncertainty, principled extensions, etc.)



## What's next

Many new insights

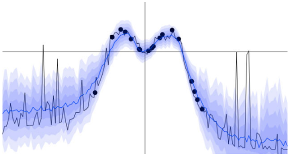
- Existing techniques in deep learning approximate Bayesian non-parametrics models
- Dropout integrates over network weights
- Dropout approximation doesn't work in convnets but integrating over the weights is still good
- Opens the door for **many new applications** (model uncertainty, principled extensions, etc.)



## What's next

Many new insights

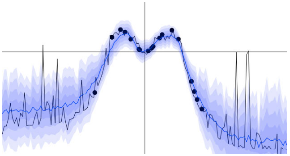
- Existing techniques in deep learning approximate Bayesian non-parametrics models
- Dropout integrates over network weights
- Dropout approximation doesn't work in convnets but integrating over the weights is still good
- Opens the door for **many new applications** (model uncertainty, principled extensions, etc.)



## What's next

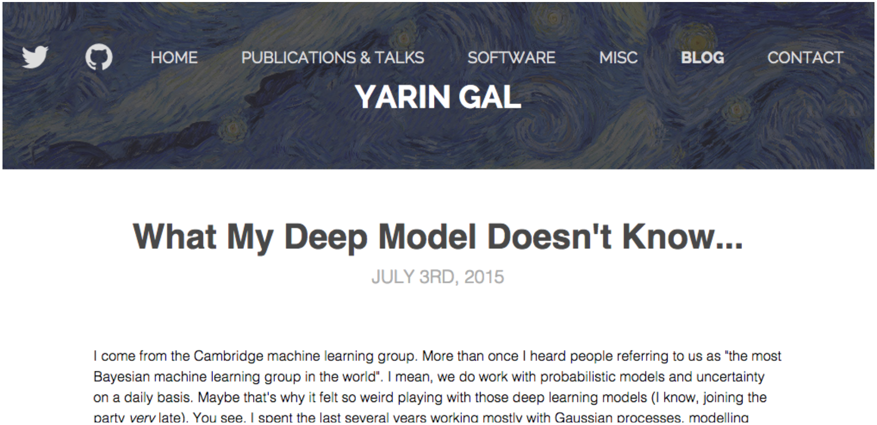
Many new insights

- Existing techniques in deep learning approximate Bayesian non-parametrics models
- Dropout integrates over network weights
- Dropout approximation doesn't work in convnets but integrating over the weights is still good
- Opens the door for **many new applications** (model uncertainty, principled extensions, etc.)



## What's next

Finally, have a look at `http://goo.gl/q8O1GK`



What My Deep Model Doesn't Know...

JULY 3RD, 2015

I come from the Cambridge machine learning group. More than once I heard people referring to us as "the most Bayesian machine learning group in the world". I mean, we do work with probabilistic models and uncertainty on a daily basis. Maybe that's why it felt so weird playing with those deep learning models (I know, joining the party *very* late). You see, I spent the last several years working mostly with Gaussian processes, modelling

Thank you for listening