Bayesian Non-linear Regression

Václav Šmídl

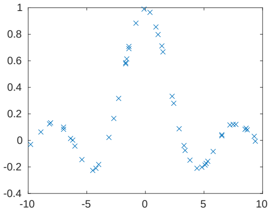
May 5, 2020

Limit of linear models – basis functions

Fit by a linear combination of basis functions:

$$y_i \quad = a\phi_1(x_i) \quad + b\phi_2(x_i) \quad + e_i$$

What should be the basis function?



Limit of linear models – basis functions

Fit by a linear combination of basis functions:

$$y_i \quad = a\phi_1(x_i) \quad +b\phi_2(x_i) \quad +e_i$$

What should be the basis function?

Adaptive basis functions:

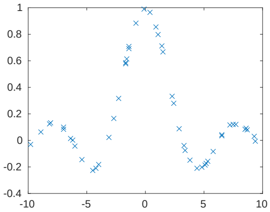
$$y_i \quad = a\phi_1(\psi_1, x_i) \quad +b\phi_2(\psi_2, x_i) \quad +e_i$$

where ϕ_j are non-linear functions with parameters ψ_k.

Estimating new set of parameters $\theta = [a, b, \psi_1, \psi_2]$

$$\frac{d(y_i - a\phi_1(\psi_1, x_i) + b\phi_2(\psi_2, x_i))^2}{d\theta} = 0.$$

$$\hat{\theta} = ?$$

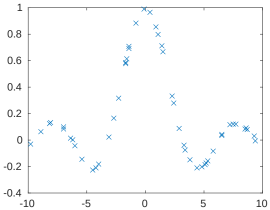


Limit of linear models – basis functions

Fit by a linear combination of basis functions:

$$y_i \quad = a\phi_1(x_i) \quad + b\phi_2(x_i) \quad + e_i$$

What should be the basis function?



What is the shape of ϕ_j?

Adaptive basis functions:

$$y_i \quad = a\phi_1(\psi_1, x_i) \quad + b\phi_2(\psi_2, x_i) \quad + e_i$$

where ϕ_j are non-linear functions with parameters ψ_k.

$$\phi_j(x_i, W, b) = \tanh(Wx_i + b),$$
$$\phi_j(x_i, W, b) = \max(0, Wx_i + b),$$
$$\vdots$$

Estimating new set of parameters $\theta = [a, b, \psi_1, \psi_2]$

$$\frac{d(y_i - a\phi_1(\psi_1, x_i) + b\phi_2(\psi_2, x_i))^2}{d\theta} = 0.$$

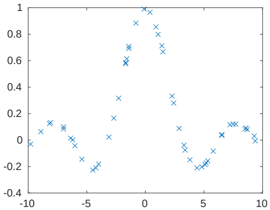
$$\hat{\theta} = ?$$

Limit of linear models – basis functions

Fit by a linear combination of basis functions:

$$y_i \quad = a\phi_1(x_i) \quad + b\phi_2(x_i) \quad + e_i$$

What should be the basis function?



What is the shape of ϕ_j?

Adaptive basis functions:

$$y_i \quad = a\phi_1(\psi_1, x_i) \quad + b\phi_2(\psi_2, x_i) \quad + e_i$$

where ϕ_j are non-linear functions with parameters ψ_k.

$$\phi_j(x_i, W, b) = \tanh(Wx_i + b),$$
$$\phi_j(x_i, W, b) = \max(0, Wx_i + b),$$
$$\vdots$$

Estimating new set of parameters $\theta = [a, b, \psi_1, \psi_2]$

$$\frac{d(y_i - a\phi_1(\psi_1, x_i) + b\phi_2(\psi_2, x_i))^2}{d\theta} = 0.$$

$$\hat{\theta} = ?$$

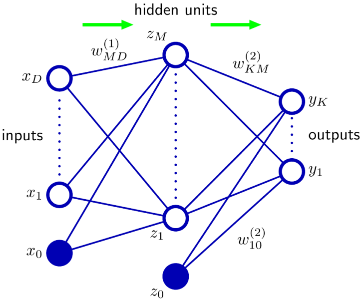
Nested functions

$$\tanh(W_1 \tanh(W_2 x_i + b_2) + b_1),$$

Neural networks

Feed forward NN:

$$z_1 = \sigma_1 (W_1 x + b_1),$$
$$z_2 = \sigma_2 (W_2 z_1 + b_2), \ldots$$
$$y = \sigma_2 (w_m z_m + b_m) + e$$



with vector-valued
– **activation** functions $\sigma_j()$,
– **weights** w_j
– **biases** b_i.

For Gaussian noise, maximum log-likelihood is

$$\hat{\theta} = \arg\min \mathcal{L}(x, y, \theta), \quad \mathcal{L} = \boldsymbol{e}^T \boldsymbol{e} = \sum_{i=1}^{n} (y_i - \sigma_1 (w_1 \sigma_2 (\cdots) + b_1))^2 .$$

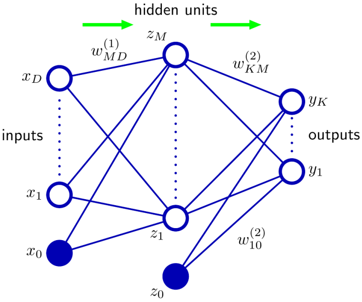
MSE (mean square error) loss function with unknowns
$\theta = [w_1, b_1, w_2, b_2, \ldots,]$.

Neural networks

Feed forward NN:

$$z_1 = \sigma_1 \left(W_1 x + b_1 \right),$$
$$z_2 = \sigma_2 \left(W_2 z_1 + b_2 \right), \ldots$$
$$y = \sigma_2 \left(w_m z_m + b_m \right) + e$$

with vector-valued
– **activation** functions $\sigma_j()$,
– **weights** w_j
– **biases** b_i.



For Gaussian noise, maximum log-likelihood is

$$\hat{\theta} = \arg\min \mathcal{L}(x, y, \theta), \quad \mathcal{L} = \boldsymbol{e}^T \boldsymbol{e} = \sum_{i=1}^{n} \left(y_i - \sigma_1 \left(w_1 \sigma_2 \left(\cdots \right) + b_1 \right) \right)^2 .$$

MSE (mean square error) loss function with unknowns $\theta = [w_1, b_1, w_2, b_2, \ldots,]$. Gradient descent method

$$\hat{\theta}^{(\tau+1)} = \hat{\theta}^{(\tau)} - \eta \nabla \mathcal{L}(\hat{\theta}^{(\tau)}),$$

where η is the (small) learning rate.

Example

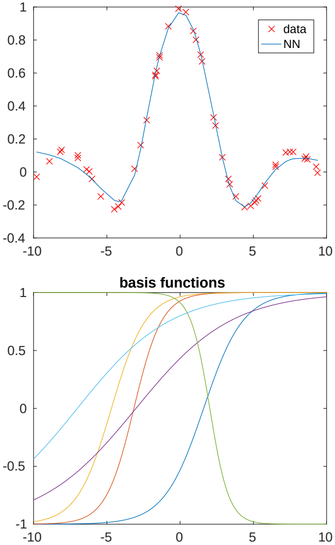
Trivial NN with one hidden layer:

$$y_i = \sum_{i=1}^{6} w_{2,i} \tanh(w_{1,j} x_i + b_{1,j}) + b_2,$$

tanh activation function on hidden layer and linear activation function on output.
Training by GD:

1. random initialization,
2. 50000 steps,
3. rate $\eta = 0.001$,

Main issue: reliability, slow convergence,...

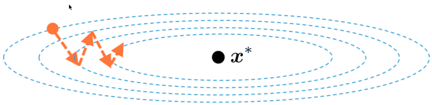


Faster gradient descent

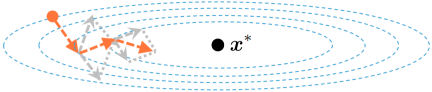
In general, gradient descent requires $O(1/\epsilon)$ steps

$$\frac{2L(\mathcal{L}\{(\hat{\theta}^{(0)}) - \mathcal{L}\{(\theta^*))}{\epsilon} \leq \tau_{max}$$

where L is the Lipschitz constant of $\mathcal{L}$, for **convex** function.



gradient descent



heavy-ball method

Heavy-ball (momentum): accumulate velocity

$$\hat{\theta}^{(\tau+1)} = \hat{\theta}^{(\tau)} - \eta \nabla L(\hat{\theta}^{(\tau)}) + \beta(\hat{\theta}^{(\tau)} - \hat{\theta}^{(\tau-1)})$$

has theoretical asymptotic number of steps $O(1/\sqrt{\epsilon})$.

- Nesterov: theoretically the fastest first-order method. Tuning: η, β (via L?)

Second-order: Newton method

Optimize:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

using Taylor expansion

$$\mathcal{L}(\theta^{(\tau)} + \boldsymbol{h}) \approx \mathcal{L}(\theta^{(\tau)}) + \nabla \mathcal{L}(\theta^{(\tau)})\boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T H_{\mathcal{L}}(\theta_k)\boldsymbol{h}$$

where $H_{\mathcal{L}}(\theta) = \nabla^2 \mathcal{L}(\theta)$.

We wish that $\theta^{(\tau+1)} = \theta^{(\tau)} + \boldsymbol{h}$ is an optimum, i.e. $\nabla_{\boldsymbol{h}} \mathcal{L}(\theta_k + \boldsymbol{h}) \equiv 0$:

$$\nabla \mathcal{L}(\theta^{(\tau)}) + H_{\mathcal{L}}(\theta^{(\tau)})\boldsymbol{h} = 0 \quad \Leftrightarrow \quad \boldsymbol{h} = -\left(H_{\mathcal{L}}(\theta^{(\tau)})\right)^{-1} \nabla \mathcal{L}(\theta^{(\tau)})$$

yielding

$$\theta^{(\tau+1)} = \theta^{(\tau)} - H_{\mathcal{L}}(\theta^{(\tau)})^{-1} \nabla \mathcal{L}(\theta^{(\tau)}).$$

with theoretical asymptotic number of steps $O(\log(\log \epsilon))$. (Expensive steps!)

Approximation of the Hessian: **LBFGS**.

Stochastic Gradient Descent

Original loss function

$$\mathcal{L}(y,x,\theta) = \sum_{i=1}^{n} \left(y_i - \sigma_1\left(w_1\sigma_2\left(\cdots\right) + b_1\right)\right)^2 .$$

is replaced by:

$$\tilde{\mathcal{L}}(y,x,\theta) = \sum_{i\in\mathcal{I}} \left(y_i - \sigma_1\left(w_1\sigma_2\left(\cdots\right) + b_1\right)\right)^2 .$$

where $\mathcal{I} \subset \{1,\ldots,n\}, |\mathcal{I}| \ll n$. For random samples of indeces $j = 1,\ldots m$,

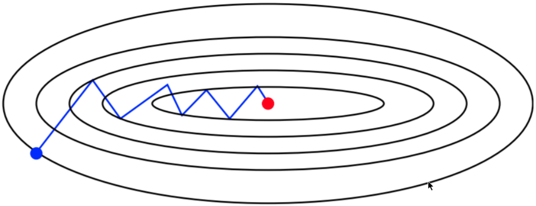
$$\nabla_\theta \mathcal{L}(y,x,\theta) = \mathrm{E}\left(\nabla\tilde{\mathcal{L}}(y,x,\theta)\right)$$

yielding

$$\hat{\theta}^{(\tau+1)} = \hat{\theta}^{(\tau)} - \eta\nabla\tilde{L}(\hat{\theta}^{(\tau)}),$$

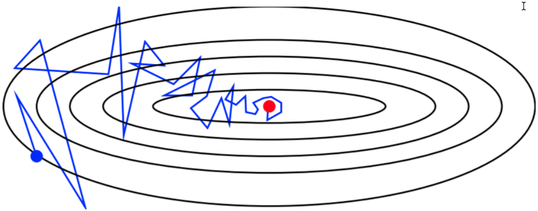
Stochastic Gradient Descent

Deterministic gradient:



Stochastic gradient: will converge only if $\eta_\tau \to 0$.

For constant η_τ it "walks" around optima.



Adaptive Learning Rate SGD

AdaGrad (Duchi, 2011) method uses estimate of the Hessian

$$H_{\mathcal{L}}(\hat{\theta}) \approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}),$$
$$\boldsymbol{r}_{\tau+1} = \boldsymbol{r}_{\tau} + \left[\nabla \tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^{2}$$

accumulates all values from the beginning (infinite window) .

Adaptive Learning Rate SGD

AdaGrad (Duchi, 2011) method uses estimate of the Hessian

$$H_{\mathcal{L}}(\hat{\theta}) \approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}),$$
$$\boldsymbol{r}_{\tau+1} = \boldsymbol{r}_{\tau} + \left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^{2}$$

accumulates all values from the beginning (infinite window) .

RMSProp (Hinton, 2012) methods adds forgetting

$$H_{\mathcal{L}}(\hat{\theta}) \approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}),$$
$$\boldsymbol{r}_{\tau+1} = \rho\boldsymbol{r}_{\tau} + (1-\rho)\left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^{2}$$

Adaptive Learning Rate SGD

AdaGrad (Duchi, 2011) method uses estimate of the Hessian

$$
\begin{aligned}
H_{\mathcal{L}}(\hat{\theta}) &\approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}), \\
\boldsymbol{r}_{\tau+1} &= \boldsymbol{r}_{\tau} + \left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2
\end{aligned}
$$

accumulates all values from the beginning (infinite window) .

RMSProp (Hinton, 2012) methods adds forgetting

$$
\begin{aligned}
H_{\mathcal{L}}(\hat{\theta}) &\approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}), \\
\boldsymbol{r}_{\tau+1} &= \rho\boldsymbol{r}_{\tau} + (1-\rho)\left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2
\end{aligned}
$$

ADAM (Kingma&Ba, 2014) combines adaptive rate with adaptive momentum

Adaptive Learning Rate SGD

AdaGrad (Duchi, 2011) method uses estimate of the Hessian

$$
\begin{aligned}
H_{\mathcal{L}}(\hat{\theta}) &\approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}), \\
\boldsymbol{r}_{\tau+1} &= \boldsymbol{r}_{\tau} + \left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2
\end{aligned}
$$

accumulates all values from the beginning (infinite window) .

RMSProp (Hinton, 2012) methods adds forgetting

$$
\begin{aligned}
H_{\mathcal{L}}(\hat{\theta}) &\approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}), \\
\boldsymbol{r}_{\tau+1} &= \rho\boldsymbol{r}_{\tau} + (1-\rho)\left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2
\end{aligned}
$$

ADAM (Kingma&Ba, 2014) combines adaptive rate with adaptive momentum

Bayesian filtering (Aichison, 2018) explains ADAM, as an extended state in Kalman filter.

Adaptive Learning Rate SGD

AdaGrad (Duchi, 2011) method uses estimate of the Hessian

$$H_{\mathcal{L}}(\hat{\theta}) \approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}),$$
$$\boldsymbol{r}_{\tau+1} = \boldsymbol{r}_{\tau} + \left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2$$

accumulates all values from the beginning (infinite window) .

RMSProp (Hinton, 2012) methods adds forgetting

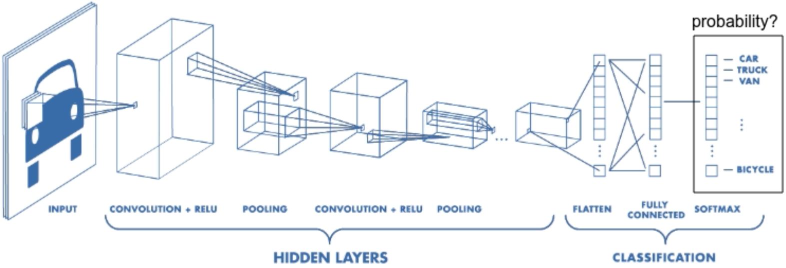
$$H_{\mathcal{L}}(\hat{\theta}) \approx \mathrm{diag}(\sqrt{\boldsymbol{r}_{\tau+1}}),$$
$$\boldsymbol{r}_{\tau+1} = \rho\boldsymbol{r}_{\tau} + (1-\rho)\left[\nabla\tilde{\mathcal{L}}(\hat{\theta}^{(\tau)})\right].^2$$

ADAM (Kingma&Ba, 2014) combines adaptive rate with adaptive momentum

Bayesian filtering (Aichison, 2018) explains ADAM, as an extended state in Kalman filter.

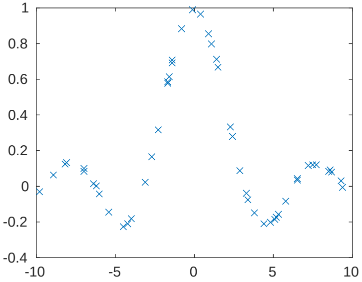
Controversy: adaptation can help but can also harm convergence

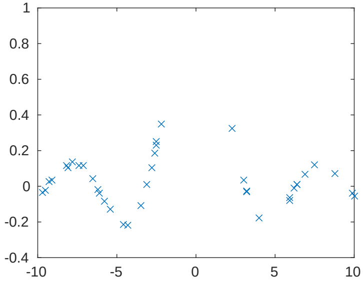
Deep Learning



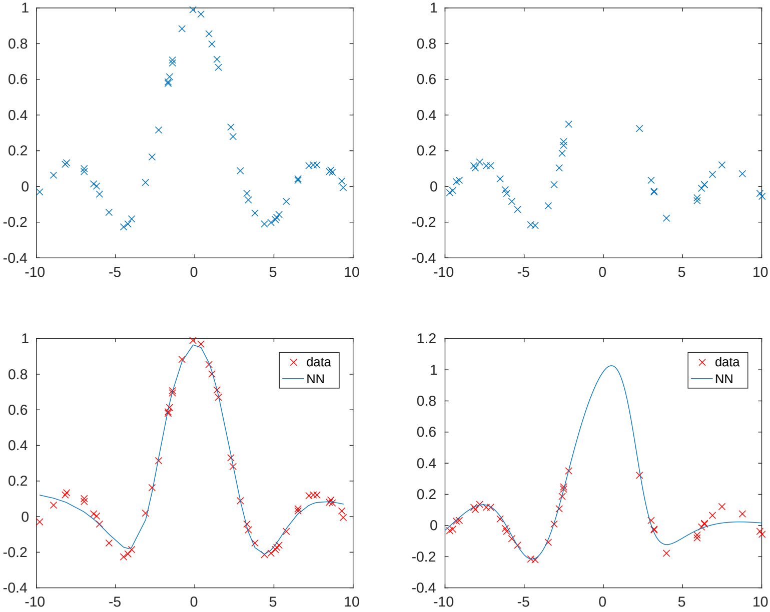
- Large networks with many layers
- Special layers that allow to compute gradients
- Training by a first-order methods
- Excellent at supervised tasks (regression)

Can we trust the result?

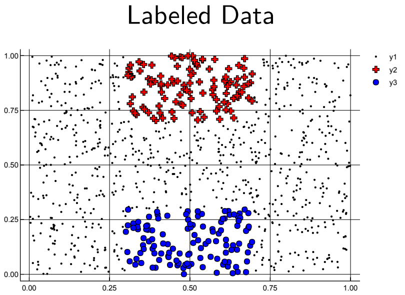


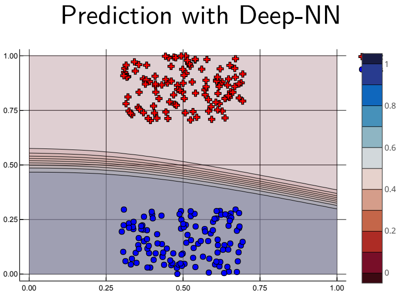


Can we trust the result?



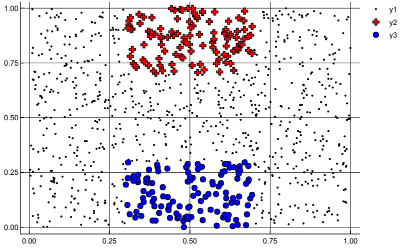
False confidence in classification



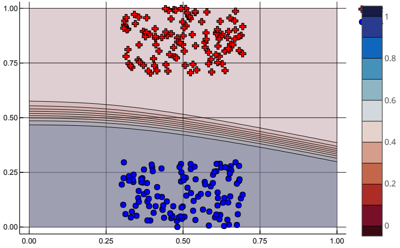


False confidence in classification

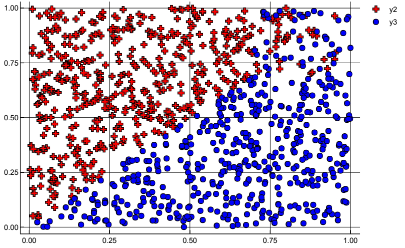
Labeled Data



Prediction with Deep-NN

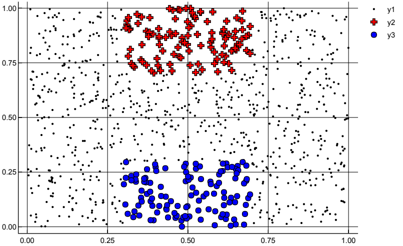


True labels

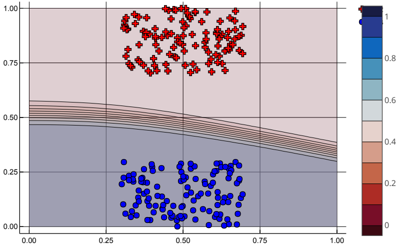


False confidence in classification

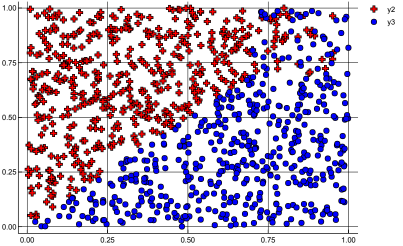
Labeled Data



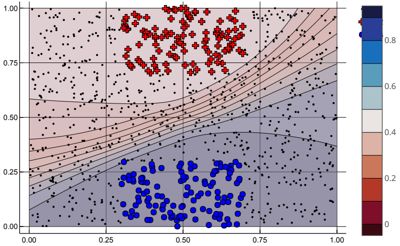
Prediction with Deep-NN



True labels



$\int p(y|\theta, x)p(\theta)d\theta$ from 500 HMC



Posterior distribution of the weights

Replace parameter estimate $\hat{\theta}$ by $p(\theta)$. Harder than with linear models:

Posterior distribution of the weights

Replace parameter estimate $\hat{\theta}$ by $p(\theta)$. Harder than with linear models:

Laplace build covariance around $\hat{\theta}$, (MacKay, 1992):

$$p(\theta) \approx \mathcal{N}(\hat{\theta}, \Sigma), \qquad \Sigma = (-\nabla\nabla \log p(\hat{x}))^{-1},$$

Posterior distribution of the weights

Replace parameter estimate $\hat{\theta}$ by $p(\theta)$. Harder than with linear models:

Laplace build covariance around $\hat{\theta}$, (MacKay, 1992):

$$p(\theta) \approx \mathcal{N}(\hat{\theta}, \Sigma), \qquad \Sigma = (-\nabla\nabla \log p(\hat{x}))^{-1},$$

HMC (Metropolis Hastings) uses gradient for solving Hamiltonian ODE

- golden standard for posterior estimates
- computationally expensive

Posterior distribution of the weights

Replace parameter estimate $\hat{\theta}$ by $p(\theta)$. Harder than with linear models:

Laplace build covariance around $\hat{\theta}$, (MacKay, 1992):

$$p(\theta) \approx \mathcal{N}(\hat{\theta}, \Sigma), \qquad \Sigma = (-\nabla\nabla \log p(\hat{x}))^{-1},$$

HMC (Metropolis Hastings) uses gradient for solving Hamiltonian ODE

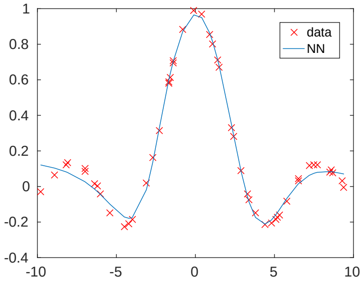
- golden standard for posterior estimates
- computationally expensive

Langevin Dynamics (Welling, Teh, 2011):

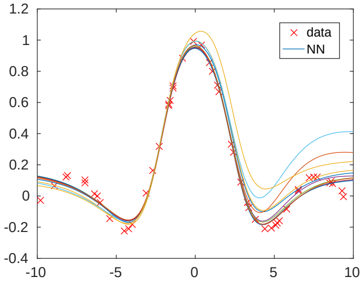
$$\hat{\theta}^{(\tau+1)} = \hat{\theta}^{(\tau)} - \eta \nabla \tilde{L}(\hat{\theta}^{(\tau)}) + e_t, \quad e_t \sim \mathcal{N}(0, \epsilon),$$

where ϵ and η needs to be carefully balanced. (Asymptotic proof of acceptance rate=1).

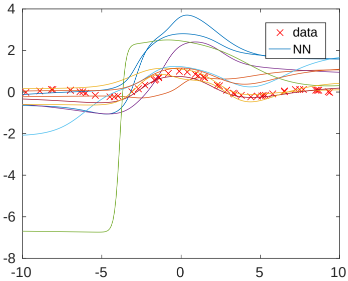
Laplace



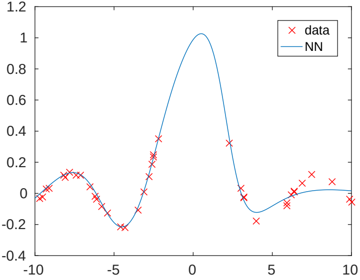
max+0.1std



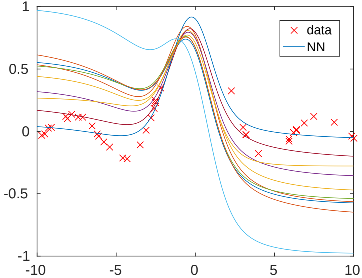
max+std



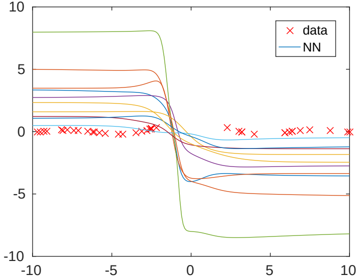
Laplace II



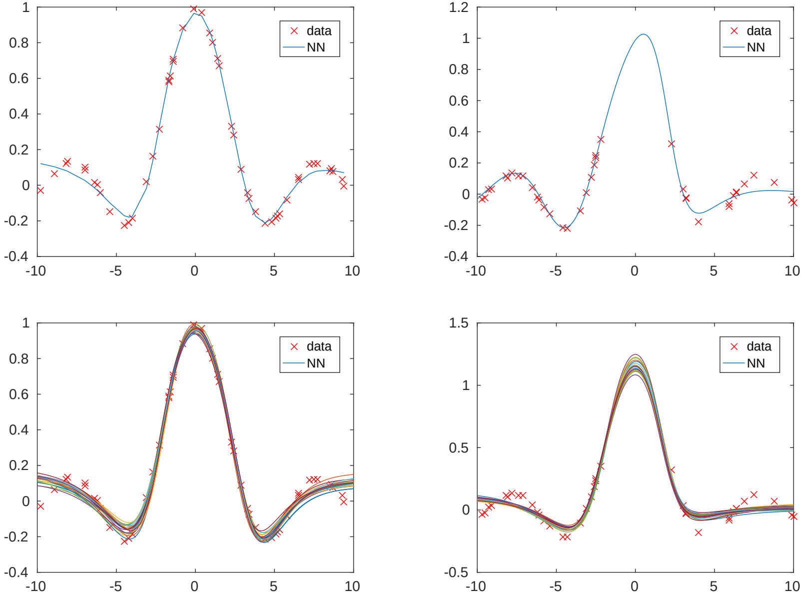
max+0.1std



max+std



Langevin MCMC (tweaked)



Uncertainty in the number of basis functions

- Knowing that the "true" value of the basis functions is 6 is very strong information.
- How many basis functions for an arbitrary function? Universal approximation property of NN. Infinity.
- Limit of 2 layer NN is the Gaussian process (Neal, 1994):

$$y(x) \sim GP(0, K(\theta, x, x'))),$$

  where $K(x, x')$ is a covariance function of all possible x, x' pairs, and θ is a hyperparameter.
- For a chosen vector of $\boldsymbol{x}$ has Gaussian distribution with

$$y \sim \mathcal{N}(0, K(\theta, \boldsymbol{x}, \boldsymbol{x}'))).$$

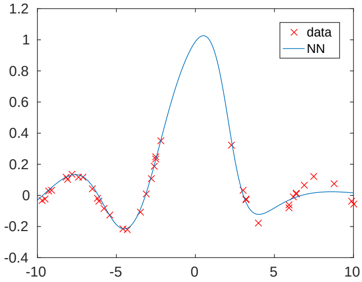
- Joint process of testing new data $\overline{x}$ using training data x (Rasmussen, 2006)

$$\begin{bmatrix} y(x) \\ \overline{y}(\overline{x}) \end{bmatrix} \sim GP\left(0, \begin{bmatrix} K(x, x') & K(x, \overline{x}') \\ K(\overline{x}, x') & K(\overline{x}, \overline{x}') \end{bmatrix}\right),$$

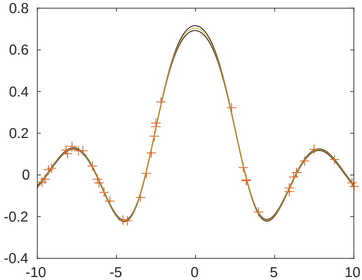
  and predictive points are marginal

$$\overline{y}(\overline{x}) \sim \mathcal{N}(K(\overline{x}, x')K(x, x')^{-1}y(x), K(\overline{x}, \overline{x}') - K(\overline{x}, x')K(x, x')^{-1}K(x, \overline{x}'))$$

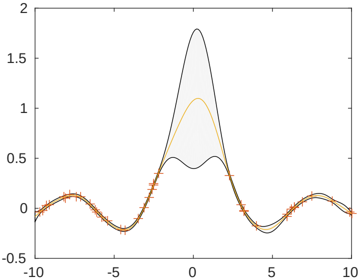
Gaussian process and its hyperparameters



GP: empirical Bayes kernel



GP: inflated kernel



Dropout MC

Standard Network Model:

$$
\begin{aligned}
z_i &= \sigma_i\left(W_i x + b_i\right), \quad i = 1 : m-1, \\
y &= \sigma_2\left(w_m z_m + b_m\right),
\end{aligned}
$$

Dropout Network Model:

$$
\begin{aligned}
z_i &= \sigma_i\left(W_i\left(\xi_i \circ x\right) + b_1\right), \\
y &= \sigma_2\left(w_m(\xi_m \circ z_m) + b_m\right)
\end{aligned}
$$

where ξ_i are vectors of zeros and ones sampled from Bernouli distribution.

Works also for Gaussian distribution, can be explained by Variational Inference.

- Dropout is an approximation of GP (Gal, Ghahramani, 2016),
- Deep Neural Networks as Gaussian Processes (Lee, et. al. 2018).

SGD is Approximate Bayesian Inference

SDG is a discretization of approximation of random walk model

$$\nabla \tilde{\mathcal{L}}(\theta) \approx \nabla \mathcal{L}(\theta) + \frac{1}{\sqrt{S}}\Delta, \qquad \Delta \sim \mathcal{N}(0, C(\theta))$$

If the loss function can be approximated by quadratic function

$$\mathcal{L}(\theta) = \frac{1}{2}\theta^\top A\theta,$$

then posterior factor $q(\theta) = \mathcal{N}(\hat{\theta}, \Sigma)$ satisfies:

$$\Sigma A + A\Sigma = \frac{\eta}{S} C(\theta).$$

Minimizing KL to $p(\theta)$ yields (Mandt, Hoffman, Blei, 2017):

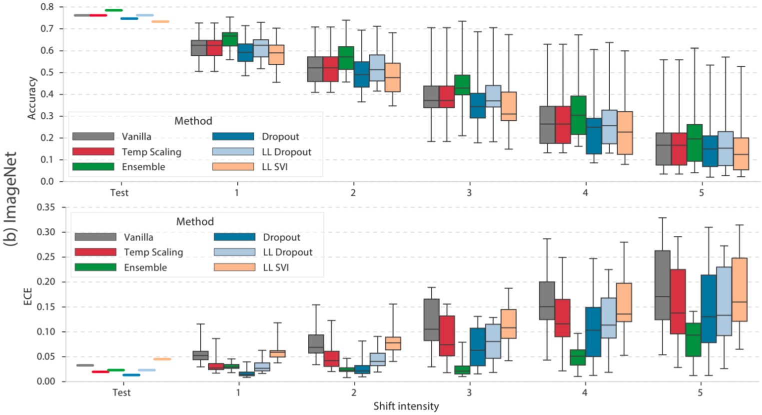
$$\eta^* = \frac{2S}{N}\frac{\dim(\theta)}{\mathrm{tr}(C)}, \text{ or } \qquad H^* = \frac{2S}{N} C^{-1}, \text{ (matrix learning rate)}$$

Can be used to tune learning rate using

$$C_\tau = (1 - \kappa_\tau)C_{\tau-1} + \kappa_\tau \mathrm{cov}(\nabla\tilde{\mathcal{L}}).$$

Sad story: Large scale comparison

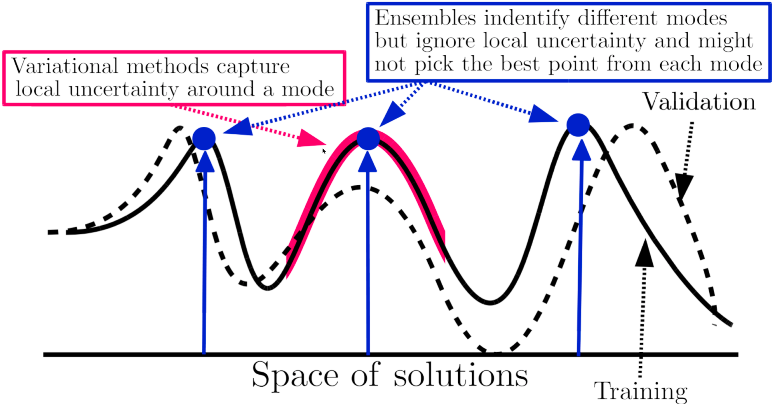
Various methods were compared in (Ovadia, et. al 2019):



The winner is **ensemble**: parallel run of NN from different starts.

Landscape of Deep networks

Hypothesis (Fort et. al. 2019): The probability of weights in networks is multimodal:



Assignment:

- Create a 1d regression problem with missing data
- train neural network fro minimum loss
- try one of the Bayesian approaches
  - Laplace,
  - Dropout
  - Ensemble
  - Langevin
  - HMC...