Advanced probabilistic methods

Lecture 9: Variational Bayes by backpropagation

Pekka Marttinen

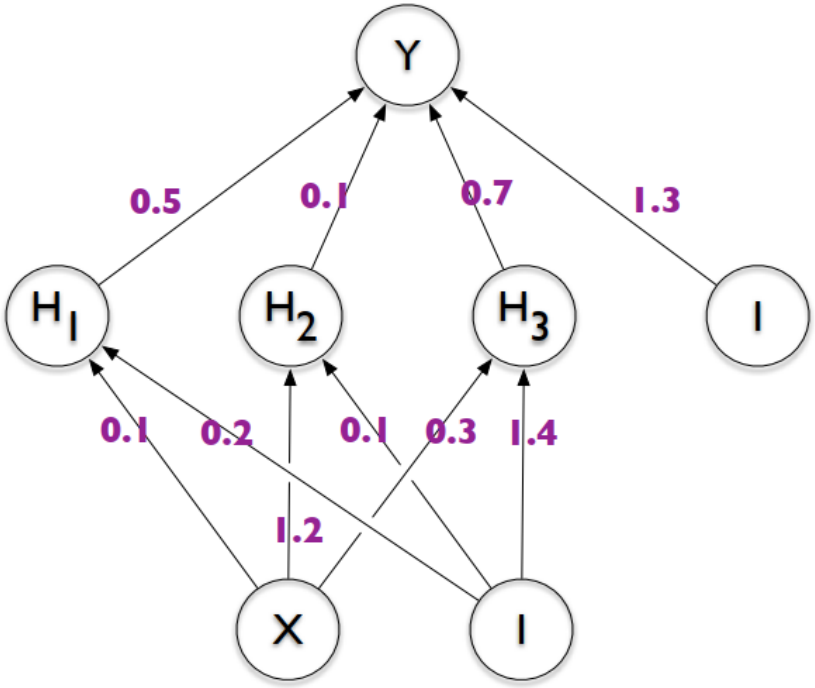
Aalto University

March, 2021

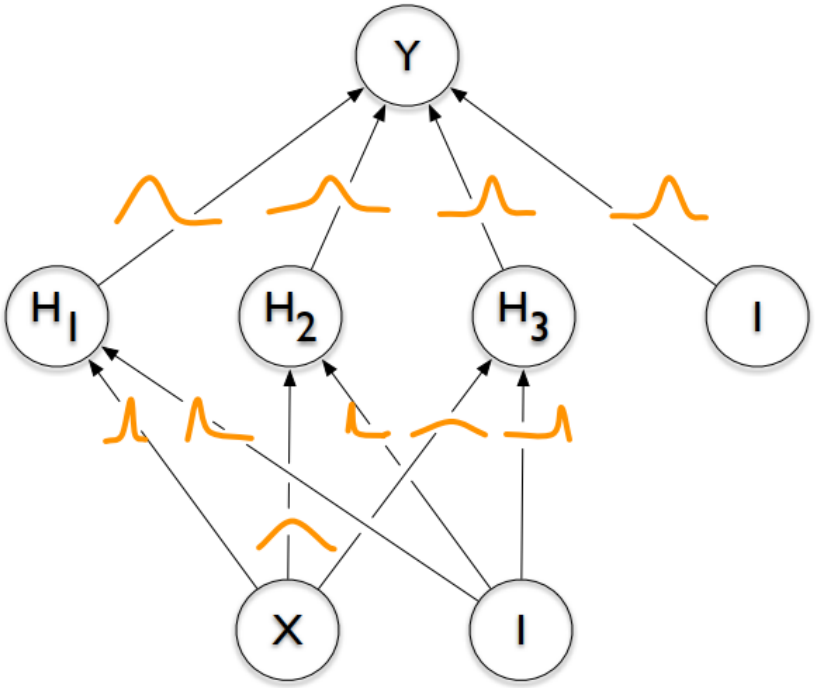
Lecture 9 overview

- Recap and caveats of VB
- Idea of variational Bayes by backpropagation
- Gradient of the ELBO
  - Monte Carlo sampling and backpropagation
- Computation using a mini-batch
- Lecture based on:
  - Blundell et al. (2015). Weight uncertainty in neural networks. *ICML*. https://arxiv.org/pdf/1505.05424.pdf
- Also relevant, for example:
  - Hoffman et al. (2013). Stochastic Variational Inference.
  - Kingma, Welling (2014). Auto-encoding variational Bayes.
  - Wilson, Izmailov (2020). Bayesian Deep Learning and a Probabilistic Perspective of Generalization

Motivation: Bayesian Neural Networks (BNNs)

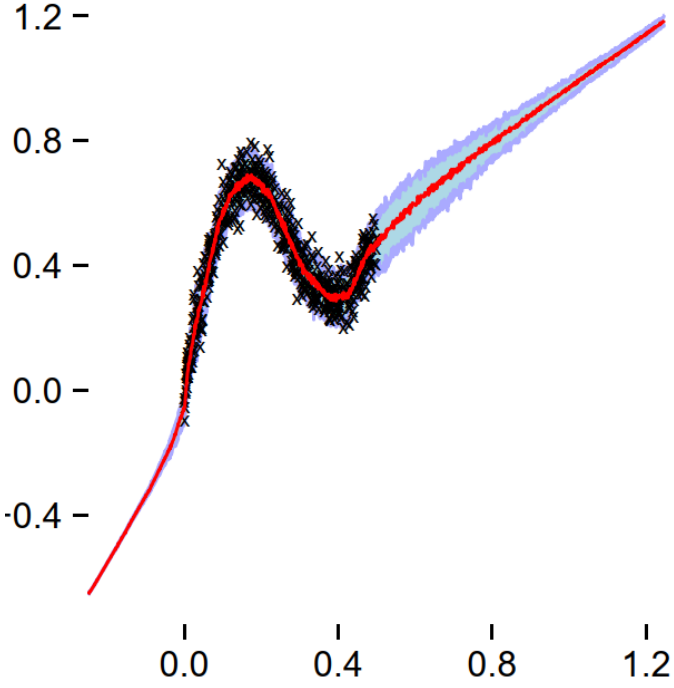


Classical neural network: each weight has a fixed value. (Blundell, Fig. 1)

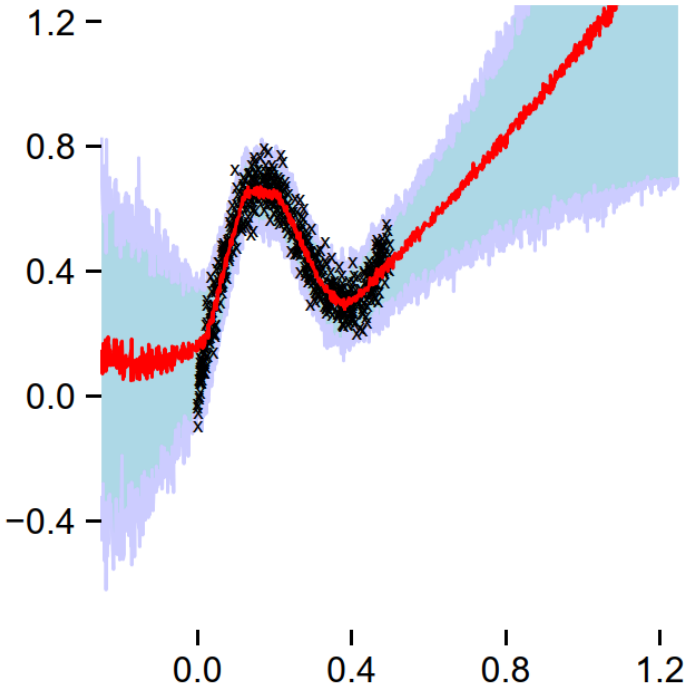


Bayesian neural network: each weight is assigned a probability distribution.

Benefits of being Bayesian (1/2)



Classical NN severely underestimates uncertainty in out-of-data regions.



Bayesian NN captures uncertainty better. (Blundell, Fig. 5)

Benefits of being Bayesian (2/2)

- Uncertainty properly quantified
  - Important in decision making
  - Critical in: medical applications, autonomous driving, ...
  - Active learning, reinforcement learning, ...
- Improved generalization (prediction accuracy)
  - Cheap model averaging over the posterior uncertainty
  - Automated complexity cost: regularization, robustness to small perturbations

Notation

- Model:

$$y_i = f(x_i, \mathbf{w}) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_l^2), \quad i = 1, \ldots, N.$$

- The log-likelihood:

$$\log p(\mathcal{D}|\mathbf{w}) = \sum_{i=1}^{N} \log p(y_i|x_i, \mathbf{w}) = \sum_{i=1}^{N} \log \mathcal{N}(y_i|f(x_i, \mathbf{w}), \sigma_l^2)$$

- Prior: $\mathbf{w} \sim \mathcal{N}(0, \alpha^2 I)$
- Hyperparameters α^2 and σ_l^2 are assumed known constants.
- f can be a NN or linear regression (exercise)

Predictive uncertainty

- Classical NN:

$$p(y^*|x^*, \mathcal{D}) = \mathcal{N}(y^*|f(x^*, \mathbf{w}^{\text{MLE}}), \sigma_l^2), \text{ where}$$
$$\mathbf{w}^{\text{MLE}} = \arg\max_{\mathbf{w}} \log p(\mathcal{D}|\mathbf{w}).$$

- Bayesian NN:

$$p(y^*|x^*, \mathcal{D}) = \int_{\mathbf{w}} p(y^*|x^*, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w}$$
$$= \int_{\mathbf{w}} \mathcal{N}(y^*|f(x^*, \mathbf{w}), \sigma_l^2) p(\mathbf{w}|\mathcal{D}) d\mathbf{w}$$

- Both models include noise uncertainty σ_l^2, but only the BNN accounts for the uncertainty in $\mathbf{w}$.

Classical way of training neural networks

- ML-estimate

$$\mathbf{w}^{\mathrm{MLE}} = \arg\max_{\mathbf{w}} \log p(\mathcal{D}|\mathbf{w})$$
$$= \arg\min_{\mathbf{w}} \underbrace{-\log p(\mathcal{D}|\mathbf{w})}_{\text{Loss (MSE)}}$$

- (Stochastic) gradient descent:
  - Calculate loss (for a mini-batch m): $-\log p(D_m|\mathbf{w})$
  - Backpropagate to get the gradient: $-\nabla_{\mathbf{w}} \log p(D_m|\mathbf{w})$
  - Update $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \log p(D_m|\mathbf{w})$
  - Repeat
- Very simple compared to the lengthy VB derivations!

Simple example: VB for linear regression

- Set $f(x, \mathbf{w}) = w_1 x + w_0$ such that

$$y_i = w_0 + w_1 x_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_l^2), \quad i = 1, \ldots, N,$$

  where $\mathbf{w} = (w_0, w_1)$.
- Mean field assumption:

$$p(w_0, w_1 | \mathcal{D}) \approx q(w_0) q(w_1),$$

  where

$$q(w_0) = \mathcal{N}(w_0 | \mu_0, \sigma_0^2),$$
$$q(w_1) = \mathcal{N}(w_1 | \mu_1, \sigma_1^2).$$

- Parameters $\lambda = \{\mu_0, \sigma_0, \mu_1, \sigma_1\}$ are the **variational parameters**.

VB for linear regression

- The goal of VB is to learn the values of $\lambda = \{\mu_0, \sigma_0, \mu_1, \sigma_1\}$.
- Previously, we derived factor updates using formulas:

$$\log q^*(w_0) = \mathbb{E}_{q(w_1)}\left[\log p(\mathbf{x}, \mathbf{y}, w_0, w_1)\right] + \text{const.}$$
$$\log q^*(w_1) = \mathbb{E}_{q(w_0)}\left[\log p(\mathbf{x}, \mathbf{y}, w_0, w_1)\right] + \text{const.}$$

- And: exponentiate, normalize, figure out the values of the respective variational parameters.

Problems in VB

- **Problem 1**: Closed form update for:

$$\log q^*(w_0) = \mathbb{E}_{q(w_1)}\left[\log p(\mathbf{x}, \mathbf{y}, w_0, w_1)\right]$$

  available only when conjugate priors are assumed.
- **Problem 2**: Computing a single update slow when N large:

$$\log q^*(w_0) = \mathbb{E}_{q(w_1)} \underbrace{\left[\sum_{i=1}^{N} \log p(y_i | x_i, w_0, w_1)\right]}_{O(N)} + \log p(w_0).$$

- **Problem 3**: Lengthy model-specific derivations needed $\rightarrow$ developing models slow.

VB by backpropagation, ideas

- **Idea 1**: Use Monte Carlo integration to calculate the required expectations.
  - No need for conjugate priors.
- **Idea 2**: Calculate updates using a minibatch.
  - Speed-up when N large.
- **Idea 3**: Use SGD and backpropagation to calculate the gradient of the ELBO
  - Avoids lengthy manual model-specific derivations.

Terminology

- Many methods have been introduced for VB which use SGD to optimize the ELBO.
  - Stochastic variational inference
  - Black-box variational inference
  - Stochastic gradient variational Bayes
  - Doubly-stochastic variational inference
  - Bayes by backprop
- Details of these methods may differ.
- The method presented here is called *Bayes by backprop* in Blundell *et al.* (2015).

A closer look at the variational objective (1/2)

- The ELBO for the linear regression model:

$$\mathcal{L}(\lambda) = \int q(\mathbf{w}|\lambda) \log \frac{p(\mathbf{x}, \mathbf{y}, \mathbf{w})}{q(\mathbf{w}|\lambda)} d\mathbf{w},$$

which can be written as

$$\mathcal{L}(\lambda) = \mathbb{E}_{q(\mathbf{w}|\lambda)} \left[\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})\right] - KL(q(\mathbf{w}|\lambda)||p(\mathbf{w})) + \text{const}$$

A closer look at the variational objective (2/2)

- Instead of maximizing the ELBO, in SGD we minimize the negative ELBO:

$$\text{Loss}(\lambda) = -\mathcal{L}(\lambda) = \underbrace{\mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x},\mathbf{w})\right]}_{\text{Likelihood cost}} + \underbrace{KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))}_{\text{Complexity cost}}$$

- Gradient of the loss:

$$\nabla_\lambda \text{Loss}(\lambda) = \nabla_\lambda \mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x},\mathbf{w})\right] + \nabla_\lambda KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$$

- In general both the loss and its gradient are intractable.

Gradient of the complexity cost

- In the special case considered here (q factorized, distributions Gaussian), $KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$ has a closed form.
  - Can be relaxed (details skipped).
- Hence, $KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$ is a deterministic function of λ and can be computed in a forward pass.
- The gradient $\nabla_\lambda KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$ can be calculated simply by using backpropagation with the chain rule.

Gradient of the likelihood cost (1/3)

- In principle, any expectation w.r.t. $q(\mathbf{w}|\lambda)$ could be approximated using Monte Carlo sampling, e.g.,

$$\mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x},\mathbf{w})\right] \approx -\frac{1}{S}\sum_{s=1}^{S}\log p(\mathbf{y}|\mathbf{x},\mathbf{w}^{(s)}),$$

  where $\mathbf{w}^{(s)} \sim q(\mathbf{w}|\lambda)$.

- However, this can't be applied to compute the gradient because:

$$\begin{aligned}\nabla_\lambda \mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x},\mathbf{w})\right] &= -\nabla_\lambda \int q(\mathbf{w}|\lambda)\log p(\mathbf{y}|\mathbf{x},\mathbf{w})d\mathbf{w} \\ &\overset{1}{=} -\int \log p(\mathbf{y}|\mathbf{x},\mathbf{w})\nabla_\lambda q(\mathbf{w}|\lambda)d\mathbf{w}\end{aligned}$$

  is not an expectation w.r.t. $q(\mathbf{w}|\lambda)$.

[1] Exchanging gradient and integration is ok if the variable w.r.t. which we integrate is different from the variable w.r.t. which we differentiate (assuming regularity conditions).

Gradient of the likelihood cost (2/3)

- Reparameterization trick: instead of sampling $\mathbf{w}^{(s)} \sim q(\mathbf{w}|\lambda)$ directly, do as follows:
  1. Sample $\mathbf{e}^{(s)} \sim N(0, I)$.
  2. Transform $\mathbf{w}^{(s)} = g_\lambda(\mathbf{e})$.
- To sample from $w_i \sim q(w_i|\lambda_i) = N(w_i|\mu_i, \sigma_i^2)$, where $\lambda_i = (\mu_i, \sigma_i)$, we need to select

$$g_{\lambda_i}(e^{(s)}) = \mu_i + e^{(s)}\sigma_i.$$

- Then, if $e^{(s)} \sim N(0, 1)$, $w_i^{(s)}$ has the correct distribution:

$$w_i^{(s)} = g_{\lambda_i}(e^{(s)}) = \mu_i + e^{(s)}\sigma_i \sim N(\mu_i, \sigma_i^2).$$

Gradient of the likelihood cost (3/3)

- After reparameterization, the gradient can be approximated using Monte Carlo sampling:

$$\begin{aligned}\nabla_\lambda \mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x},\mathbf{w})\right] &= \nabla_\lambda \mathbb{E}_{q(\mathbf{e})}\left[-\log p(\mathbf{y}|\mathbf{x},g_\lambda(\mathbf{e}))\right] \\ &= -\mathbb{E}_{q(\mathbf{e})}\left[\nabla_\lambda \log p(\mathbf{y}|\mathbf{x},g_\lambda(\mathbf{e}))\right] \\ &\approx -\frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log p(\mathbf{y}|\mathbf{x},g_\lambda(\mathbf{e}^{(s)})),\end{aligned}$$

  where $\mathbf{e}^{(s)} \sim N(0, I)$, $s = 1, \ldots, S$.
- In practice it's common to use $S = 1$.
- The gradient $\nabla_\lambda \log p(\mathbf{y}|\mathbf{x},g_\lambda(\mathbf{e}^{(s)}))$ can be obtained by backpropagation.

Using minibatches

- Suppose data $\mathcal{D}$ is divided into M minibatches: $\mathcal{D}_1, \ldots, \mathcal{D}_M$.
- Objective with the full data:

$$-\mathcal{L}(\lambda) = \mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})\right] + KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$$

- Objective for a mini-batch:

$$-\mathcal{L}_m(\lambda) = \mathbb{E}_{q(\mathbf{w}|\lambda)}\left[-\log p(\mathbf{y}_m|\mathbf{x}_m, \mathbf{w})\right] + \frac{1}{M} KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$$

- Or, averaged per individual:

$$-\mathcal{L}_m(\lambda) = -\frac{1}{|\mathcal{D}_m|}\mathbb{E}_{q(\mathbf{w}|\lambda)}\left[\sum_{i\in\mathcal{D}_m} \log p(y_i|x_i, \mathbf{w})\right] + \frac{1}{N} KL(q||p)$$

- Scaling the two terms to correspond to the same number of individuals ensures that the expectation of the stochastic gradient for the mini-batch is aligned with the gradient of the full cost.

Putting it all together

- One iteration of the *Bayes-by-backprop* for linear regression and mini-batch $\mathcal{D}_m$
  1. Sample $\mathbf{e}^{(s)} \sim N(0, I)$
  2. Transform $w_i^{(s)} = \mu_i + e_i^{(s)}\sigma_i$ for $i = 0, 1$, where $\lambda = (\mu_0, \sigma_0, \mu_1, \sigma_1)$[1]
  3. Forward pass to calculate the noisy objective:

$$\text{Loss}(\lambda) = -\frac{1}{|\mathcal{D}_m|}\sum_{i\in\mathcal{D}_m} \log p(y_i|x_i, \mathbf{w}^{(s)}) + \frac{1}{N} KL(q(\mathbf{w}|\lambda)||p(\mathbf{w}))$$

  4. Backward pass to get the stochastic gradient: $\nabla_\lambda \text{Loss}(\lambda)$.
  5. Update the variational parameters

$$\lambda \leftarrow \lambda - \eta \nabla_\lambda \text{Loss}(\lambda).$$

[1] The exercise uses a slightly different parameterization to ensure std stays positive.

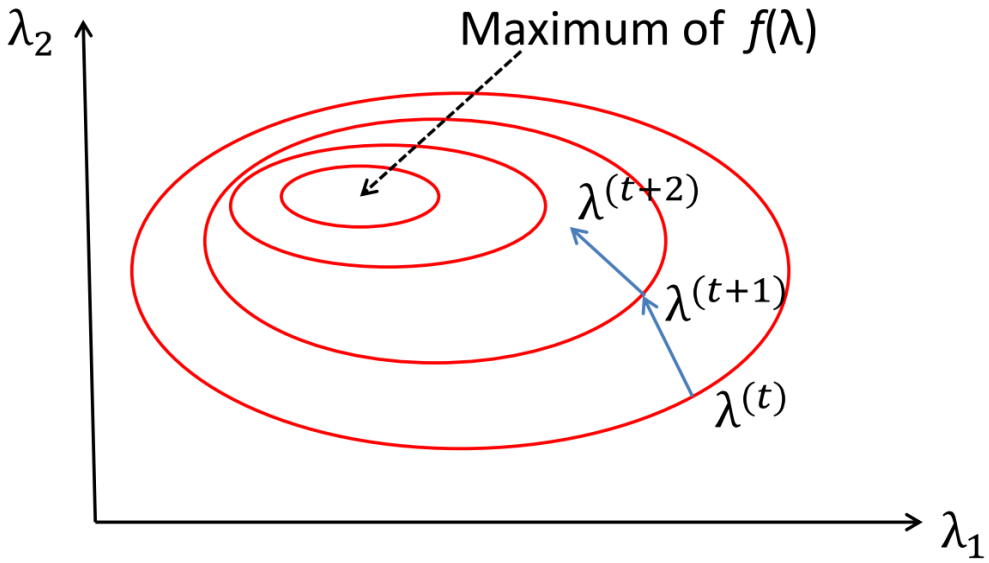
Summary

- Mean-field VB can be seen as an optimization problem: the variational parameters for each factor are updated in turn to maximize the ELBO $\mathcal{L}(q)$.
- In stochastic variational inference the negative ELBO is minimized directly using SGD.
- Stochastic gradient of the ELBO is obtained by
  - Monte Carlo sampling to approximate the loss during the forward pass.
  - Samples from $\mathbf{w}^{(s)} \sim q(\mathbf{w}|\lambda)$ are obtained using the reparameterization.
  - Backpropagation to calculate the gradient.
- Scaling up to massive data sets can be achieved using a mini-batch.

Reminder: gradient ascent algorithm*

- Gradient ascent algorithm maximizes a given function f by taking steps of length ρ to the direction of the gradient ∇f.

$$\lambda^{(t+1)} = \lambda^{(t)} + \rho \nabla_\lambda f(\lambda^{(t)}), \text{ where } \nabla_\lambda f = \left(\frac{\partial f}{\partial \lambda_1}, \ldots, \frac{\partial f}{\partial \lambda_D} \right)$$



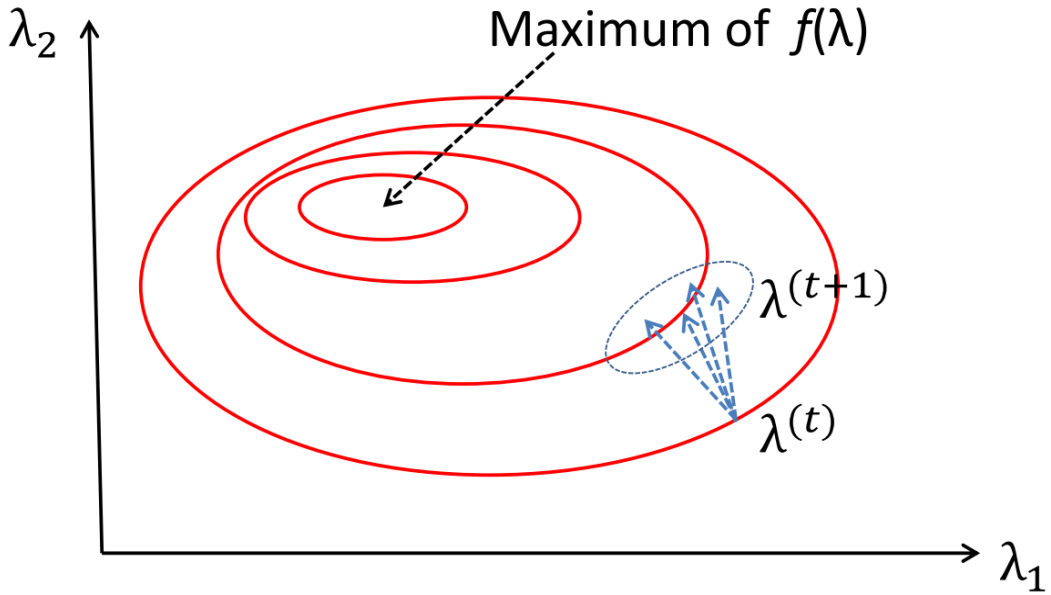
- $\lambda^{(t+1)} = \lambda^{(t)} - \rho \nabla_\lambda f(\lambda^{(t)})$ gives gradient descent.

Reminder: stochastic gradient ascent*

- Stochastic gradient ascent takes **random steps**, that are **on average to the correct direction**:

$$\lambda^{(t+1)} = \lambda^{(t)} + \rho b_t(\lambda^{(t)}),$$

- $b_t(\lambda)$ is a random variable s.t. $E(b_t(\lambda)) = \nabla_\lambda f(\lambda)$.



cartoonstock.com

Reminder: SGA with a mini-batch*

- To find a maximum likelihood estimate $\widehat{\lambda}$,

$$f(\lambda) = \frac{1}{N}\sum_{n=1}^{N} \log p(x_n|\lambda), \text{ and } \nabla_\lambda f(\lambda) = \frac{1}{N}\sum_{n=1}^{N} \nabla_\lambda \log p(x_n|\lambda)$$

  and we have to differentiate $\log p(x_n|\lambda)$ for all n.
- It is cheaper to sample a **minibatch** of S data points x_s and compute a noisy gradient

$$b(\lambda) = \frac{1}{S}\sum\nolimits_s \nabla_\lambda \log p(x_s|\lambda),$$

  which points approximately to the direction of $\nabla_\lambda f(\lambda)$.