Deep Learning

Lecture 3 – Optimization: Stochastic gradient descent and backpropagation

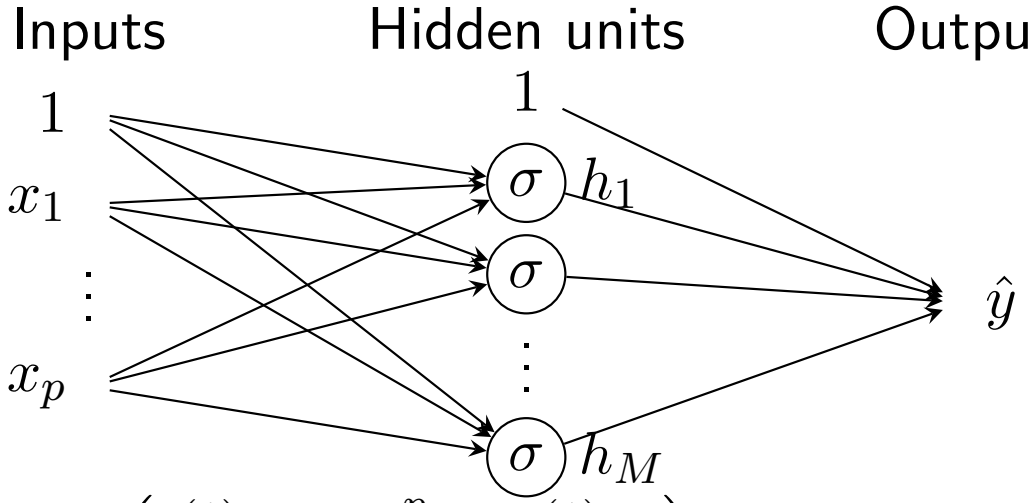
UPPSALA
UNIVERSITET

Thomas Schön
Department of Information Technology
Uppsala University

thomas.schon@it.uu.se
user.it.uu.se/~thosc112

Summary Lecture 2 (I/II) - Neural network

A neural network is a sequential construction of **several** generalized linear regression models.



$$h_1 = \sigma\left(b_1^{(1)} + \sum_{j=1}^{p} w_{1j}^{(1)} x_j\right)$$

$$h_2 = \sigma\left(b_2^{(1)} + \sum_{j=1}^{p} w_{2j}^{(1)} x_j\right)$$

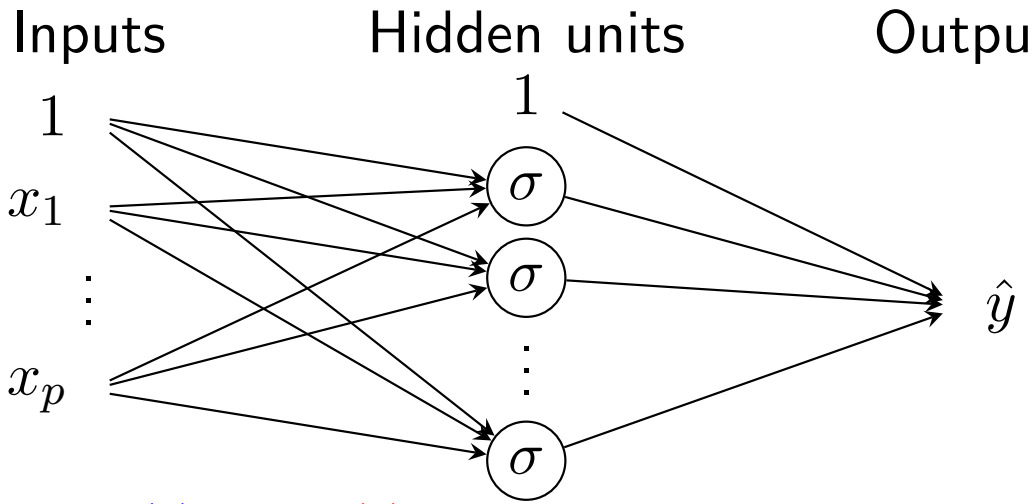
$$\vdots$$

$$h_M = \sigma\left(b_M^{(1)} + \sum_{j=1}^{p} w_{Mj}^{(1)} x_j\right)$$

$$\hat{y} = b^{(2)} + \sum_{m=1}^{M} w_m^{(2)} h_m$$

Summary Lecture 2 (I/II) - Neural network

A neural network is a sequential construction of **several** generalized linear regression models.



$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\hat{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \dots & w_{1p}^{(1)} \\ \vdots & & \vdots \\ w_{M1}^{(1)} & \dots & w_{Mp}^{(1)} \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_M^{(1)} \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} h_1 \\ \vdots \\ h_M \end{bmatrix}$$

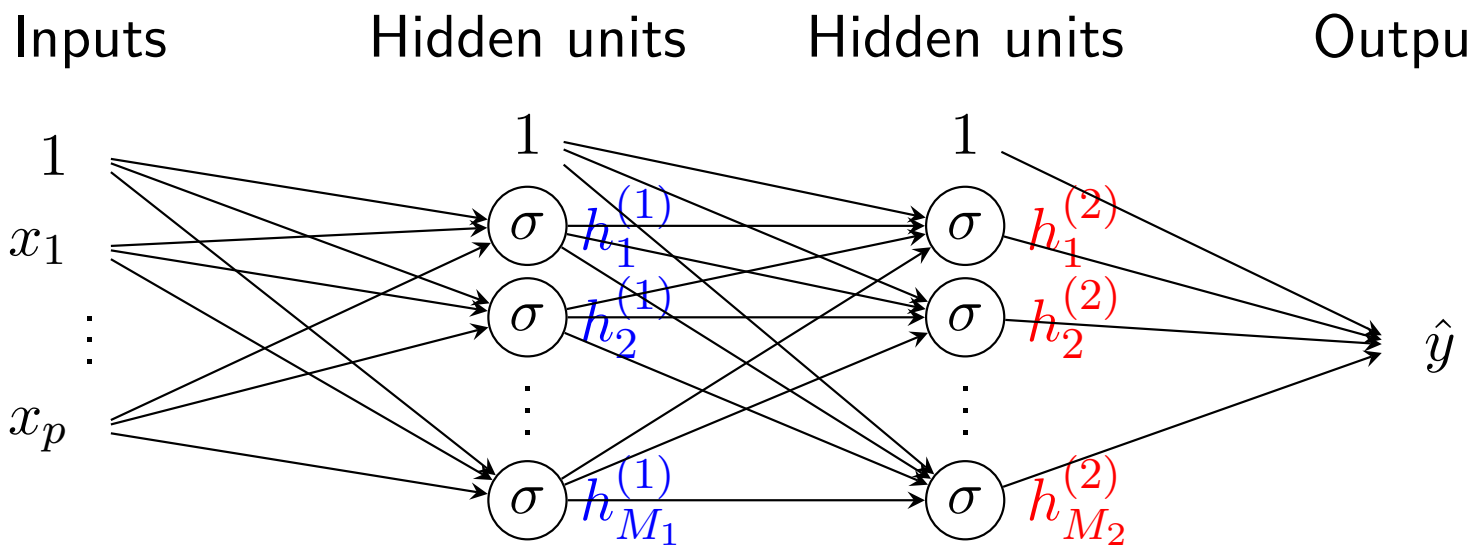
Weight matrix Offset vector Hidden units

$$\mathbf{b}^{(2)} = \begin{bmatrix} b^{(2)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_1^{(2)} & \dots & w_M^{(2)} \end{bmatrix}$$

Summary Lecture 2 (I/II) - Neural network

A neural network is a **sequential** construction of several generalized linear regression models.



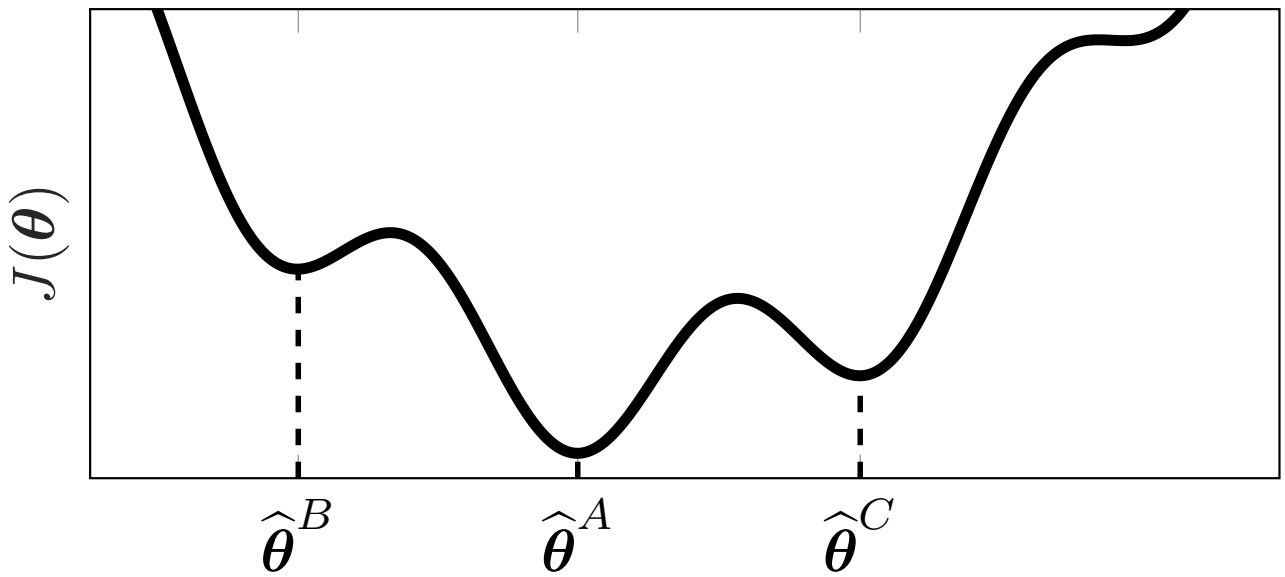
$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$
$$\hat{y} = \mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}$$

The model tends to learn better using a deep network (several layers) instead of a wide and shallow network.

Summary of Lecture 2 (II/II) - Optimization

We train our models by considering an optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$$



- The best possible solution $\widehat{\boldsymbol{\theta}}$ is the **global minimizer** ($\widehat{\boldsymbol{\theta}}^A$)
- The global minimizer is typically very hard to find, and we have to settle for a **local minimizer** ($\widehat{\boldsymbol{\theta}}^A$, $\widehat{\boldsymbol{\theta}}^B$, $\widehat{\boldsymbol{\theta}}^C$)

UPPSALA UNIVERSITET

Outline – Lecture 3

Aim: Motivate and introduce the Stochastic Gradient (SG) algorithm and derive the backpropagation algorithm to efficiently compute the gradient needed for SG.

Outline:

1. Summary – lecture 2
2. Optimization in machine learning
3. Stochastic gradient
4. Backpropagation

Reminder — forming the loss function

The model provides a prediction $\hat{y}_\theta(x) = f(x, \theta)$ of the output y.

For a good model we would expect $\hat{y}_\theta(x) \approx y$

Hence, it is natural to look for parameters θ that makes $\hat{y}_\theta(x)$ as close to y as possible.

Results in an optimization problem of the form

$$\min_\theta J(\theta)$$

where

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}_\theta(x_i), y_i)$$

Computational challenge 1 – n is large

At each optimization step we need to compute the gradient

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}).$$

Computational challenge 1 – n large: We typically have large-scale training data $(n \gg 1)$ available, which implies that it is costly to compute the gradient.

Solution: **Randomization**

Computational challenge 1 – n is large

At each iteration, we only use a small part of the dataset to compute an estimate of the gradient $\mathbf{g}_k$. This is called **minibatch stochastic gradient**.

Large scale machine learning requires **stochastic optimization**.

Randomization often produce reliability (in large scale problems).

Massive research on large scale optimization over the past decade.

Computational challenge 2 – dim(θ) large

At each optimization step we need to compute the gradient

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta}).$$

Computational challenge 2 – dim(θ) large: A neural network contains a lot of parameters. Computing the gradient is costly.

Solution: A deep NN is a composition of multiple layers. Hence, each term $\nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})$ can be computed efficiently by repeatedly applying the chain rule and storing certain intermediate results for later reuse. This is called **backpropagation**.

Stochastic gradient (SG)

The DL optimization problem

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{x}_i, \boldsymbol{\theta}), y_i) = \frac{1}{n}\sum_{i=1}^{n} L_i(\boldsymbol{\theta})$$

Basic characteristics:

1. Non-convex
2. Large-scale (both large n and large dimension of $\boldsymbol{\theta}$)

Two main classes of optimization methods (for this lecture):

1. Deterministic (batch)
2. Stochastic (online)

UPPSALA UNIVERSITET

Problem formulation – stochastic opt.

What? Solve the non-convex stochastic optimization problem

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

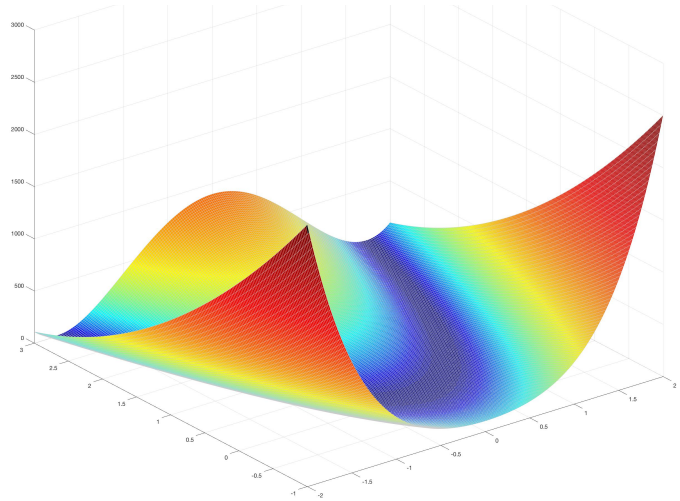
when we only have access to **noisy** evaluations of $J(\boldsymbol{\theta})$ and its derivatives.

Why? These stochastic optimization problems are common:

- When the cost function cannot be evaluated on the entire dataset. Large-scale ML and DL in particular.
- When numerical methods approximate $J(\boldsymbol{\theta})$ and $\nabla^i J(\boldsymbol{\theta})$.
- ...

Toy example – Rosenbrock's banana function

Let $J(\boldsymbol{\theta}) = (1 - \theta_1)^2 + 100(\theta_2 - \theta_1^2)^2$.

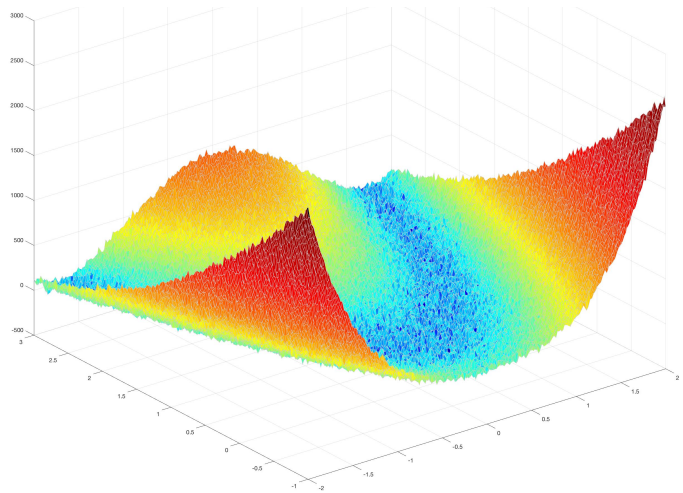


Deterministic problem

$$\min_{\theta} J(\boldsymbol{\theta})$$

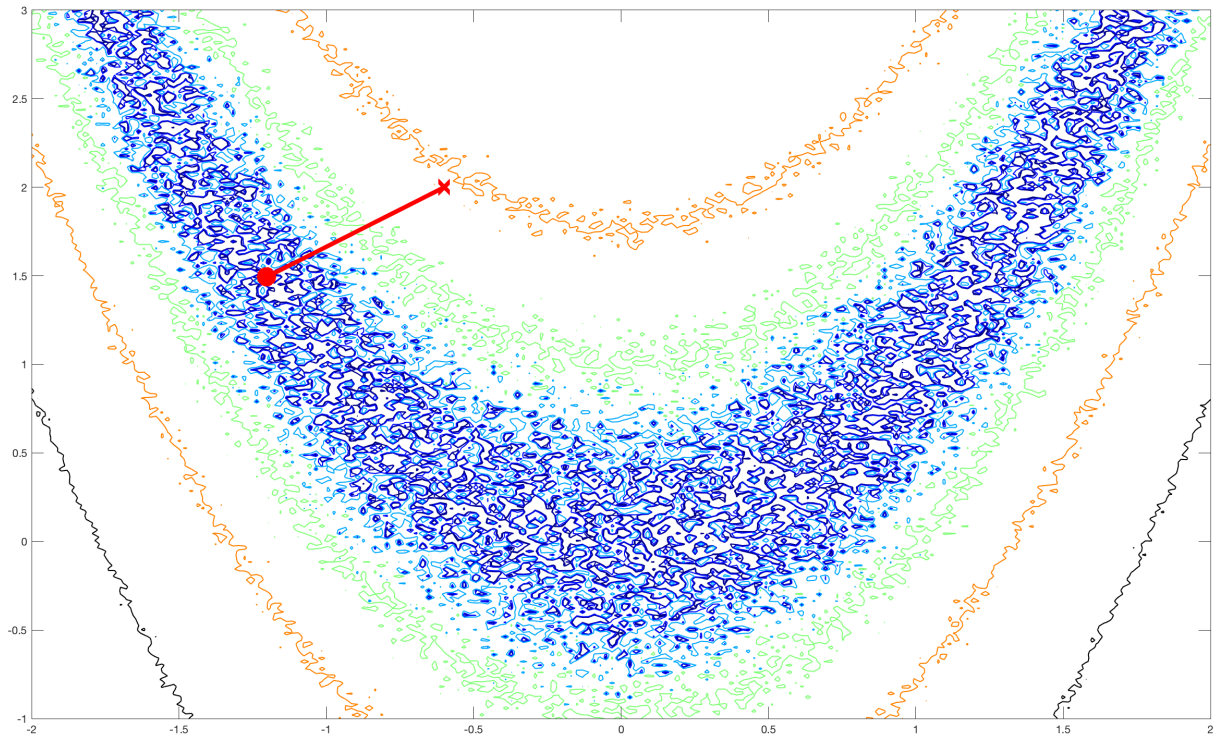
Stochastic problem

$$\min_{\theta} J(\boldsymbol{\theta})$$



when we only have access to noisy versions of the cost function $(\widetilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + e,\ e = \mathcal{N}(0, 30^2))$ and its noisy gradients.

Std. deterministic optimization at work



Terminates at the wrong solution after 3 iterations.

The true solution is $(1, 1)$.

SG is a Markov chain!

The SG idea was first proposed almost 70 years ago in

Robbins, H. and Monro, S. **A stochastic approximation method.** *The Annals of Mathematical Statistics*. 22(3):400–407, 1951.

They introduce SG as a particular **Markov chain**, which is since then 'forgotten'.

Motivating the use of SG

Advantages of the batch approach:

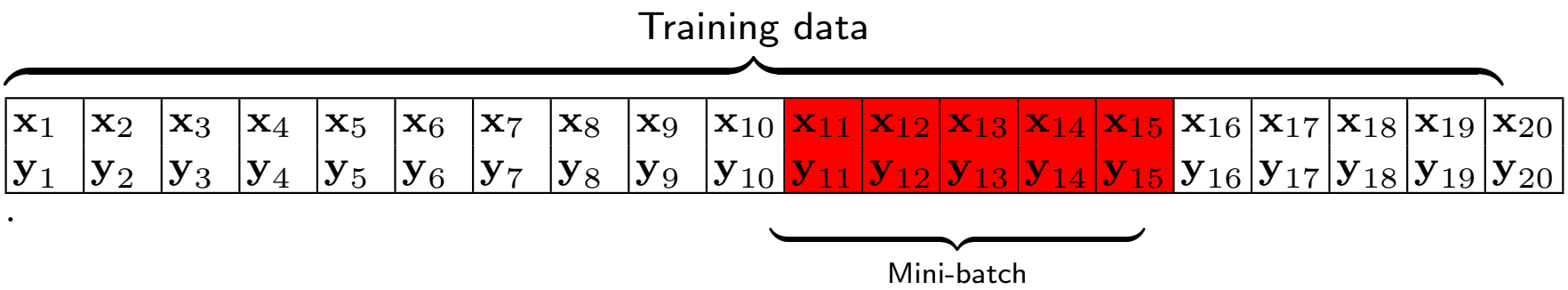
1. There are many good optimization methods available for the deterministic problem.
2. The sum-structure of J and ∇L allows for computation in a distributed manner.

Why is the stochastic gradient (SG) algorithm still the standard for large-scale machine learning (including DL)?

1. Intuitive/practical motivation
2. Theoretical motivation

We will mainly use a middle ground (combining the best properties of batch and stochastic algorithms): **minibatch SG**.

Minibatch SG



$$\boldsymbol{\theta}_3 = \boldsymbol{\theta}_2 - \gamma \frac{1}{5} \sum_{i=11}^{15} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}_2)$$

Recall:

- The extreme version is to use only one data point at each training step (called **online learning**)
- We typically do something in between (not one data point, and not all data). We use a smaller set called **minibatch**.

One pass through the training data is called an **epoch**.

UPPSALA UNIVERSITET

Minibatch SG

$\mathbf{x}_7$	$\mathbf{x}_{10}$	$\mathbf{x}_3$	$\mathbf{x}_{20}$	$\mathbf{x}_{16}$	$\mathbf{x}_2$	$\mathbf{x}_1$	$\mathbf{x}_{18}$	$\mathbf{x}_{19}$	$\mathbf{x}_{12}$	$\mathbf{x}_6$	$\mathbf{x}_{11}$	$\mathbf{x}_{17}$	$\mathbf{x}_{15}$	$\mathbf{x}_5$	$\mathbf{x}_{14}$	$\mathbf{x}_4$	$\mathbf{x}_9$	$\mathbf{x}_{13}$	$\mathbf{x}_8$
$\mathbf{y}_7$	$\mathbf{y}_{10}$	$\mathbf{y}_3$	$\mathbf{y}_{20}$	$\mathbf{y}_{16}$	$\mathbf{y}_2$	$\mathbf{y}_1$	$\mathbf{y}_{18}$	$\mathbf{y}_{19}$	$\mathbf{y}_{12}$	$\mathbf{y}_6$	$\mathbf{y}_{11}$	$\mathbf{y}_{17}$	$\mathbf{y}_{15}$	$\mathbf{y}_5$	$\mathbf{y}_{14}$	$\mathbf{y}_4$	$\mathbf{y}_9$	$\mathbf{y}_{13}$	$\mathbf{y}_8$

Iteration: 3

Epoch: 1

- If we pick the minibatches in order, they might be unbalanced and not representative for the whole data set.
- Therefore, we pick data points **at random** from the training data to form a minibatch.
- One implementation is to randomly reshuffle the data before dividing it into minibatches.
- After each epoch we do another reshuffling and another pass through the data set.

Minibatch SG

1. Initialize $\boldsymbol{\theta}_0$, set $t \leftarrow 1$, choose batch size n_b and number of epochs E.
2. For $i = 1$ to E
   (a) Randomly shuffle the training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$.
   (b) For $j = 1$ to $\frac{n}{n_b}$
      (i) Approximate the gradient of the loss function using the current minibatch $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=(j-1)n_b+1}^{jn_b}$
      $\widehat{\mathbf{g}}_t = \frac{1}{n_b} \sum_{i=(j-1)n_b+1}^{jn_b} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \approx \mathbf{g}_t.$
      (ii) Take a step in the gradient direction $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \widehat{\mathbf{g}}_t$.
      (iii) Update the iteration index $t \leftarrow t + 1$.

At each iteration we get a stochastic approximation $\widehat{\mathbf{g}}_t$ of the true gradient

$$\mathbf{g}_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L_i(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$$

Learning rate within minibatch SG

It is necessary to select a learning rate that decrease over time.

A sufficient condition to guarantee convergence is that

$$\sum_{t=1}^{\infty} \gamma_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \gamma_t^2 < \infty.$$

Practically common approach: decay the learning rate linearly

$$\gamma_t = (1 - \alpha)\gamma_0 + \alpha\gamma_\tau,$$

with

$$\alpha = \frac{t}{\tau}$$

until iteration τ and then leave γ constant after that.

Can we do better?

The answer to the title question is **Yes!**

The next question is then: **How?**

By using more information in computing:

- the search direction d_t
- and the learning rate γ_t

$$\theta_{t+1} = \theta_t + \gamma_t d_t.$$

Quasi-Newton – A non-standard take

Our problem is of the form

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Idea underlying (quasi-)Newton methods: Learn a local quadratic model $q(\boldsymbol{\theta}_t, \delta)$ of $J(\boldsymbol{\theta})$ around the current iterate $\boldsymbol{\theta}_t$

$$q(\boldsymbol{\theta}_t, \delta) = f(\boldsymbol{\theta}_t) + g(\boldsymbol{\theta}_t)^{\mathsf{T}}\delta + \frac{1}{2}\delta^{\mathsf{T}} H(\boldsymbol{\theta}_t)\delta$$

$$g(\boldsymbol{\theta}_t) = \nabla J(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}, \quad H(\boldsymbol{\theta}_t) = \nabla^2 J(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}, \quad \delta = \boldsymbol{\theta} - \boldsymbol{\theta}_k.$$

We have measurements of

- the cost function $J_t = J(\boldsymbol{\theta}_t)$,
- and its gradient $g_t = g(\boldsymbol{\theta}_t)$.

Question: How do we update the Hessian model?

Useful basic facts

Line segment connecting two adjacent iterates $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$:

$$r_t(\tau) = \boldsymbol{\theta}_t + \tau(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t), \qquad \tau \in [0, 1].$$

1. The **fundamental theorem of calculus** states that

$$\int_0^1 \frac{\partial}{\partial \tau} \nabla f(r_t(\tau)) \mathrm{d}\tau = \nabla J(r_t(1)) - \nabla f(r_t(0)) = \underbrace{\nabla J(\boldsymbol{\theta}_{t+1})}_{g_{t+1}} - \underbrace{\nabla J(\boldsymbol{\theta}_t)}_{g_t}$$

2. The **chain rule** tells us that

$$\frac{\partial}{\partial \tau} \nabla J(r_t(\tau)) = \nabla^2 J(r_t(\tau)) \frac{\partial r_t(\tau)}{\partial \tau} = \nabla^2 J(r_t(\tau))(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t).$$

$$\underbrace{g_{t+1} - g_t}_{=y_t} = \int_0^1 \frac{\partial}{\partial \tau} \nabla J(r_t(\tau)) \mathrm{d}\tau = \int_0^1 \nabla^2 J(r_t(\tau)) \mathrm{d}\tau \underbrace{(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)}_{s_t}.$$

Result – the quasi-Newton integral

With the definitions $y_t \triangleq g_{t+1} - g_t$ and $s_t \triangleq \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$ we have

$$y_t = \int_0^1 \nabla^2 J(r_t(\tau)) \mathrm{d}\tau s_t.$$

Interpretation: The difference between two consecutive gradients (y_t) constitute a **line integral observation of the Hessian**.

Problem: Since the Hessian is unknown there is no functional form available for it.

Solution 1 – existing quasi-Newton algorithms

Existing quasi-Newton algorithms (e.g. BFGS, DFP, Broyden's method) assume the Hessian to be constant

$$\nabla^2 J(r_t(\tau)) \approx H_{t+1}, \qquad \tau \in [0, 1],$$

implying the following approximation of the integral (**secant condition**)

$$y_t = H_{t+1} s_t.$$

Find H_{t+1} by **regularizing** H:

$$\begin{aligned} H_{t+1} = \min_{H} \quad & \|H - H_t\|_W^2, \\ \text{s.t.} \quad & H = H^{\mathsf{T}}, \quad H s_t = y_t, \end{aligned}$$

Equivalently, the existing quasi-Newton methods can be interpreted as **particular instances of Bayesian linear regression**.

Philipp Hennig. **Probabilistic interpretation of linear solvers**, *SIAM Journal on Optimization*, 25(1):234–260, 2015.

UPPSALA UNIVERSITET

Solution 2 – use a flexible nonlinear model

The approach used here is fundamentally different.

Recall that the problem is **stochastic** and **nonlinear**.

Hence, we need a model that can deal with such a problem.

Idea: Represent the Hessian using a **Gaussian process** learned from data.

Adrian Wills and Thomas B. Schön. **Stochastic quasi-Newton with line-search regularization.** *Automatica*, 127:109503, May 2021.

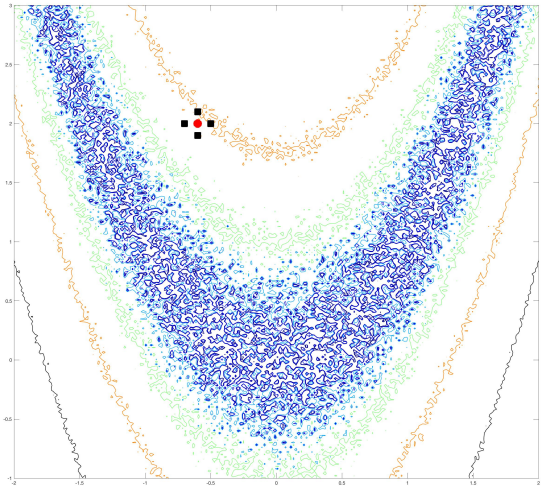
Adam at work

By not using the curvature information we expose ourselves to the ”banana-problem”.

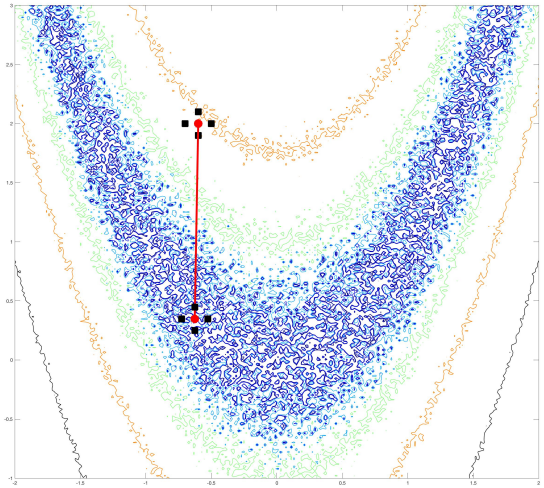
UPPSALA UNIVERSITET

2-order alg. at work — overall result

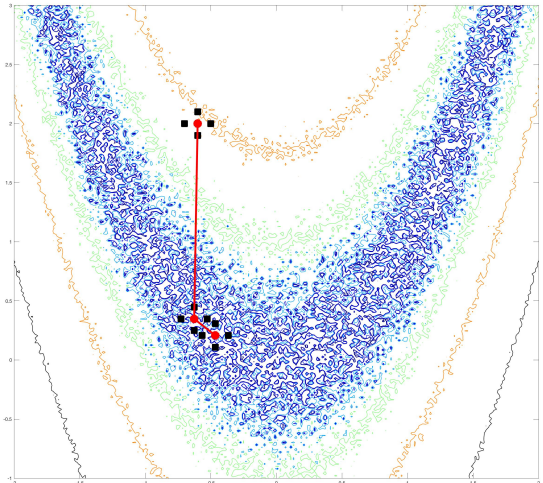
Initial value



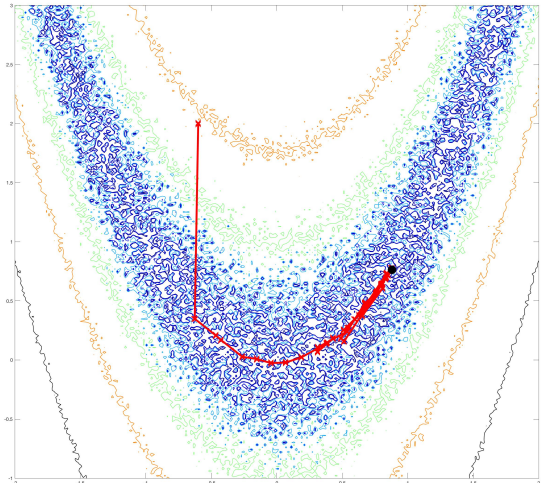
Iteration 1



Iteration 2



Iteration 50



Adaptive (momentum) methods

Adaptive (momentum) methods **accelerates SG** especially when

- high curvature,
- small but consistent gradients,
- or noisy gradients.

Idea: Consider different ways of using the earlier gradients to adapt learning rates

$$\gamma_t = \gamma(\nabla J_t, \nabla J_{t-1}, \ldots, \nabla J_0)$$

and search directions

$$d_t = d(\nabla J_t, \nabla J_{t-1}, \ldots, \nabla J_0).$$

Update parameters as

$$\theta_{t+1} = \theta_t - \gamma_t d_t.$$

Adaptive (momentum) methods

The most popular methods use an **exponential moving average**, where recent gradients have higher weights than older gradients.

Adam updates according to (typical values $\beta_1 = 0.9, \beta_2 = 0.999$)

$$d_t = \beta_1 d_{t-1} + (1 - \beta_1)\nabla J_t,$$
$$\gamma_t^2 = \beta_2 \gamma_{t-1}^2 + (1 - \beta_2)\|\nabla J_t\|^2.$$

Numerical opt. – Want to know more?

Great pedagogical introduction to the use of stochastic optimization for large-scale ML:

Bottou, L., Curtis, F. E. and Nocedal, J. **Optimization methods for large-scale machine learning**, *SIAM Review*, 60(2):223–311, 2018.

Solid and well-structured introduction to numerical optimization:

Nocedal, J. and Wright, S. **Numerical Optimization**, 2006.

To follow the research on optimization methods for ML, see the NeurIPS and ICML conferences.

neurips.cc

icml.cc

Backpropagation

Bookkeeping – chain rule

Just as we can compute gradients of z w.r.t. scalars and vectors, we can compute gradients of z w.r.t. matrices and tensors.

Let $\mathbf{X}$ denote a tensor. If $z = f(\mathbf{Y})$ and $\mathbf{Y} = g(\mathbf{X})$, then

$$\nabla_{\mathbf{X}} z = \sum_{j} (\nabla_{\mathbf{X}} \mathbf{Y}_j) \frac{\partial z}{\partial \mathbf{Y}_j}$$

Note that the index variable j now encodes the complete tuple of indices, e.g. for a matrix, it contains two coordinates.

Forward propagation and backpropagation

We only consider feedforward networks to focus on the key ideas. Generalizing this to a general network is then mainly a bookkeeping exercise.

Forward propagation: The input $\mathbf{x}$ is propagated through the network populating all hidden units with values and finally we reach the output $\widehat{\mathbf{y}}$. Based on this we can then compute the cost $J(\boldsymbol{\theta})$.

Backpropagation: Refers to the method for computing the gradient. It amounts to repeated use of the chain rule coupled with a memory that allows us to reuse computations.

Ex. scalar regression

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - z^{(L)}(x_i, \theta))^2$$

To keep the notation simple we only consider one data point, i.e.

$$J(\theta) = (y - z^{(L)}(x, \theta))^2$$

UPPSALA UNIVERSITET

Forward propagation – fully connected MLP

Weight matrices: $\boldsymbol{W}^{(i)}, i \in \{i = 1, \ldots, L\}$.
Offset terms: $\boldsymbol{b}^{(i)}, i \in \{i = 1, \ldots, L\}$.
Input: $\mathbf{x}$, output: $\mathbf{y}$
Network depth: L

1. $\boldsymbol{q}^{(0)} = \mathbf{x}$
2. **For** $l = 1$ **to** L **do**
   (a) $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{q}^{(l-1)} + \boldsymbol{b}^{(l)}$
   (b) $\boldsymbol{q}^{(l)} = h(\boldsymbol{z}^{(l)})$
3. **end for**
4. $\boldsymbol{z}^{L} = \boldsymbol{W}^{(L)}\boldsymbol{q}^{(L-1)} + \boldsymbol{b}^{(L)}$
5. $J = (y - z^{(L)})^2$

Backpropagation

$$d\boldsymbol{z}^{(l)} \triangleq \nabla_{\boldsymbol{z}^{(l)}} \boldsymbol{J}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial z_1^{(l)}} \\ \vdots \\ \frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial z_{U^{(l)}}^{(l)}} \end{bmatrix}, \qquad d\boldsymbol{q}^{(l)} \triangleq \nabla_{\boldsymbol{q}^{(l)}} \boldsymbol{J}(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial q_1^{(l)}} \\ \vdots \\ \frac{\partial \boldsymbol{J}(\boldsymbol{\theta})}{\partial q_{U^{(l)}}^{(l)}} \end{bmatrix}.$$

We need the gradients $d\boldsymbol{z}^{(l)}$ and $d\boldsymbol{q}^{(l)}$ for all layers.

Back to our example scalar regression problem

$$J(\boldsymbol{\theta}) = (y - z^{(L)}(x, \boldsymbol{\theta}))^2$$

$$dz^{(L)} = \nabla_{\boldsymbol{z}^{(L)}} \boldsymbol{J}(\boldsymbol{\theta}) = -2(y - z^{(L)})$$

Backpropagation

Applying the chain rule to

$$
\begin{aligned}
&\boldsymbol{q}^{(0)} = \boldsymbol{x}, \\
&\begin{cases} \boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{q}^{(l-1)} + \boldsymbol{b}^{(l)}, \\ \boldsymbol{q}^{(l)} = h(\boldsymbol{z}^{(l)}), \end{cases} \qquad \text{for } l = 1, \ldots, L-1 \\
&\boldsymbol{z}^{(L)} = \boldsymbol{W}^{(L)} \boldsymbol{q}^{(L-1)} + \boldsymbol{b}^{(L)},
\end{aligned}
$$

results in

$$
\begin{aligned}
d\boldsymbol{z}^{(l)} &= d\boldsymbol{q}^{(l)} \odot h'(\boldsymbol{z}^{(l)}), \\
d\boldsymbol{q}^{(l-1)} &= \boldsymbol{W}^{(l)\mathsf{T}} d\boldsymbol{z}^{(l)},
\end{aligned}
$$

where $\odot$ denotes the element-wise product.

Backpropagation

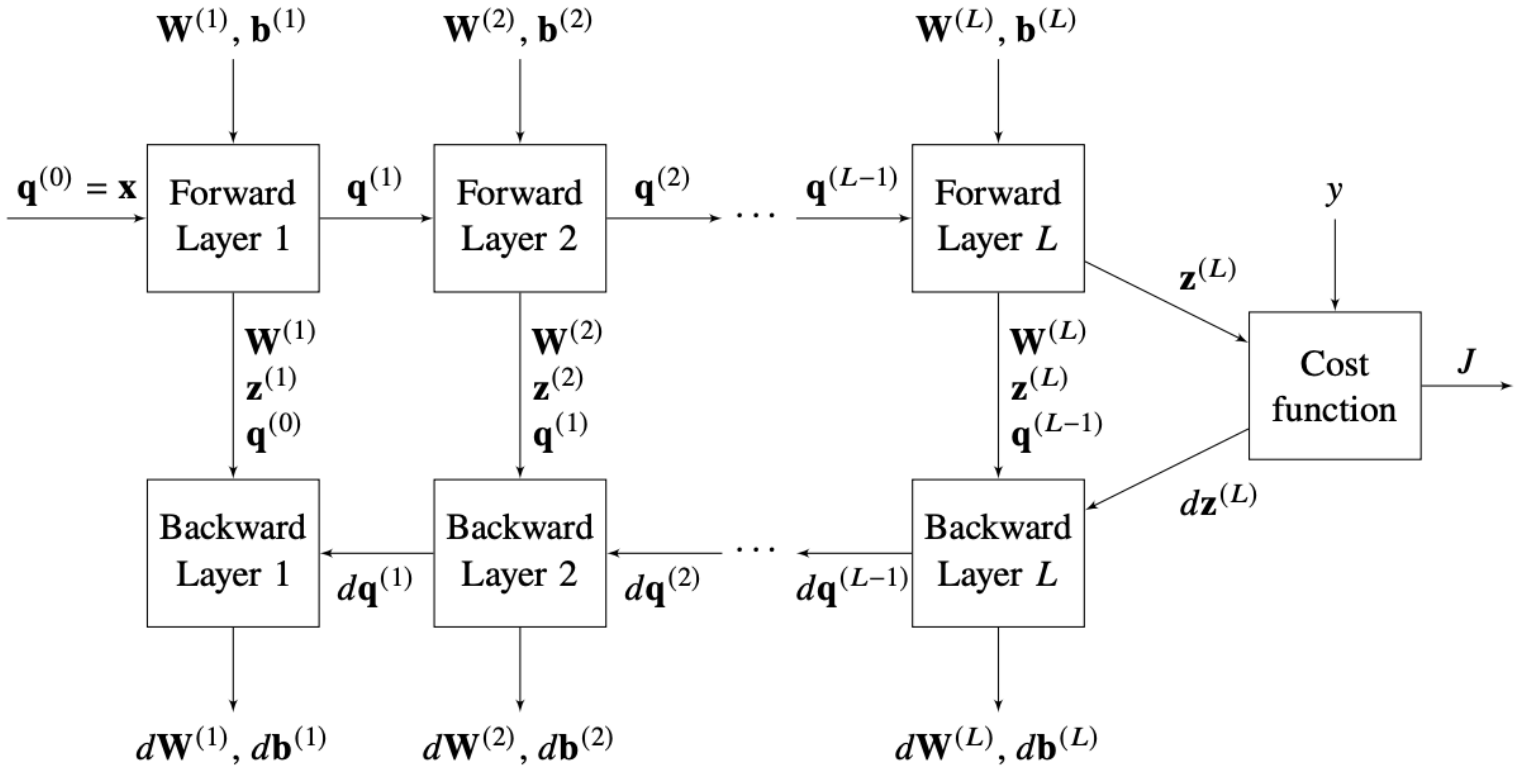
Finally, we can make use of $d\boldsymbol{z}^{(l)}$ to compute the gradients of the weight matrices and offset vectors as

$$d\boldsymbol{W}^{(l)} = d\boldsymbol{z}^{(l)} \boldsymbol{q}^{(l-1)\mathsf{T}},$$
$$d\boldsymbol{b}^{(l)} = d\boldsymbol{z}^{(l)}.$$

The equations on this and the previous slide are derived using the **chain rule**, see Appendix 6.A for detailed derivation.

This can be generalized (essentially a bookkeeping exercise) into any network structure. The **computational graph** is a useful tool.

Computational graph – backpropagation



Summarized in Algorithm 6.1.

UPPSALA UNIVERSITET

A few concepts to summarize lecture 3

Gradient descent: An iterative optimization algorithm where we at each iteration take a step proportional to the negative gradient.

Learning rate: (a.k.a step length): A scalar tuning parameter deciding the length of each gradient step in gradient descent.

Stochstic gradient (SG): A version of gradient descent where we at each iteration only use a small part of the training data (a mini-batch).

Minibatch: The group of training data that we use at each iteration in SG.

Batch size: The number of data points in one minibatch.

Epoch: One pass through the training data.

Backpropagation: A method based on the chain rule to compute the gradient by reusing information.