Moderne Methoden der Datenanalyse

Task, loss function, risk

Roger Wolf

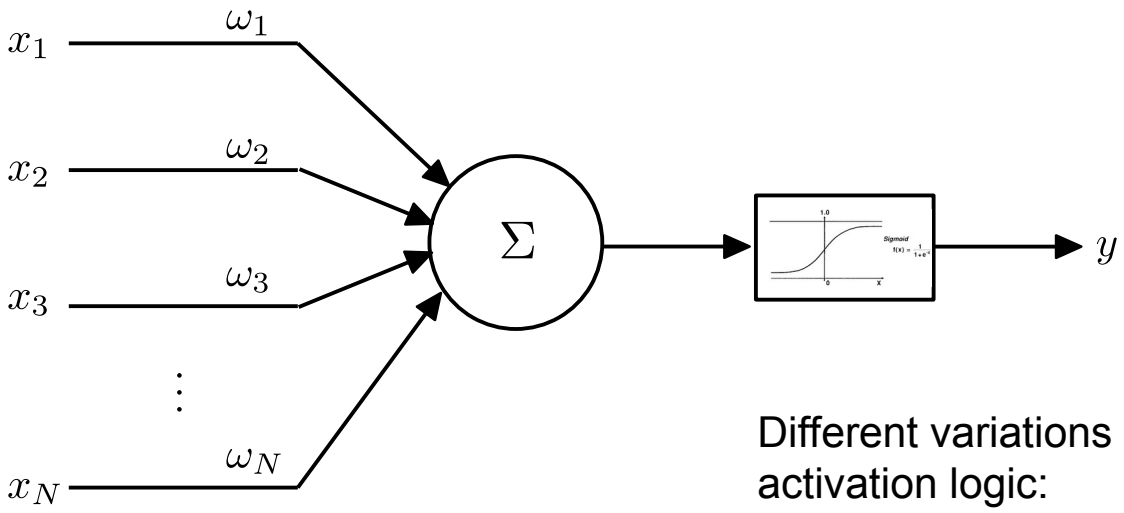
07. July 2023

Content of this lecture

- Training as optimization task.
- The NN model, loss function & risk functional.
- Numerical minimization (*gradient descent*):
  - Learning rate and convergence properties.
  - Momentum methods.
  - Backpropagation → forward and backward pass.

Continuous activation function

- Last time we discussed the importance of the continuous activation function for the NN performance:



$x_1 \dots x_N \in \mathbb{R}$

$\omega_1 \dots \omega_N \in \mathbb{R}$

Unit fires if $\sum_i \omega_i x_i \geq T$

Different variations to express the activation logic:

$$\sum_i \omega_i x_i \geq T\,,$$

$$\sum_i \omega_i x_i - T \geq 0\,,$$

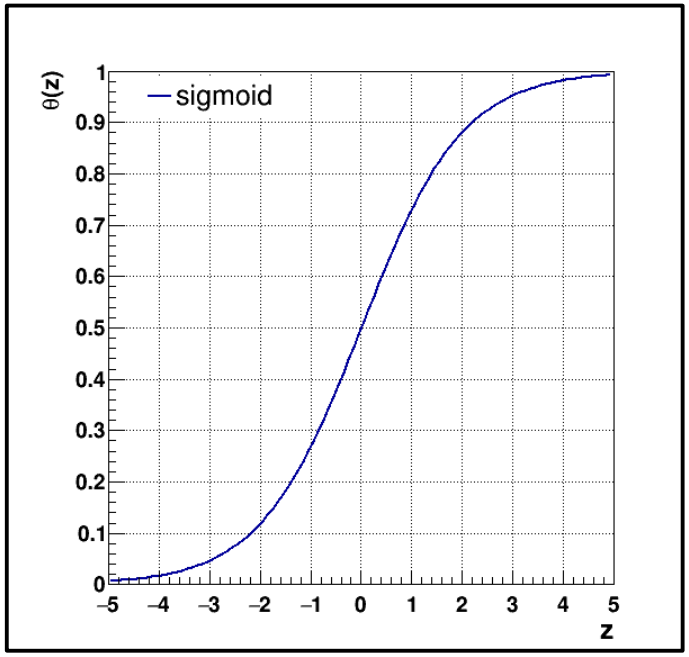
$$\theta\left(\sum_i \omega_i x_i - T\right)$$

We choose the sigmoid function as concrete example.

The sigmoid function

- The sigmoid function (a.k.a. logistic function) is a common activation function for perceptrons:

$$\theta(z) = \frac{1}{1+\exp(-z)} = \frac{1}{2}\left(1+\tanh\left(\frac{z}{2}\right)\right) \quad ; \quad \frac{\mathrm{d}\theta}{\mathrm{d}z} = \theta(z)\left(1-\theta(z)\right)$$



- Maps $\mathbb{R}$ to $(0,1)$.
- Resembles a continuous threshold behavior.
- Is used to model saturation processes in statistics.
- Provides an interpretation as conditional PDF.

The NN task

- NNs are designed to solve specific *tasks*.

The NN task

- NNs are designed to solve specific *tasks*.
- Each concrete realization of a task requires:

The NN task

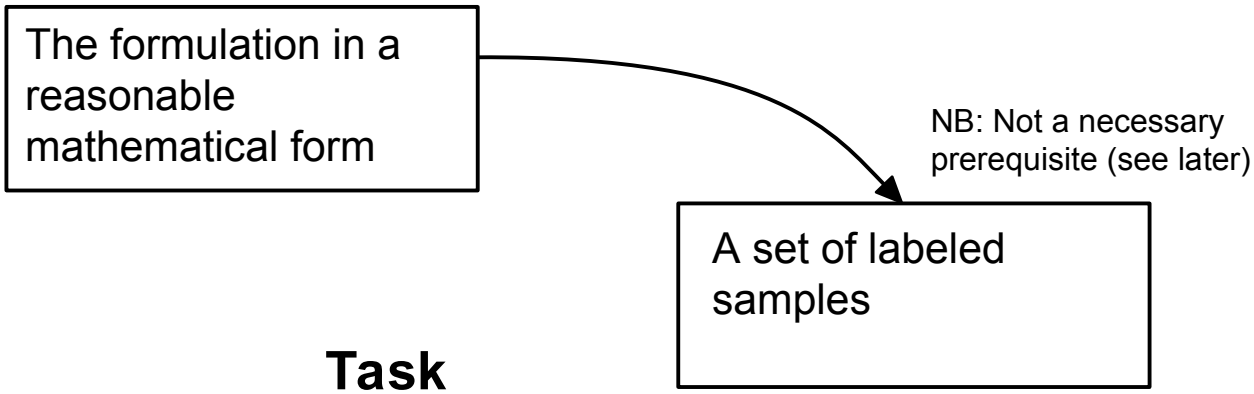
- NNs are designed to solve specific *tasks*.
- Each concrete realization of a task requires:

The formulation in a reasonable mathematical form

Task

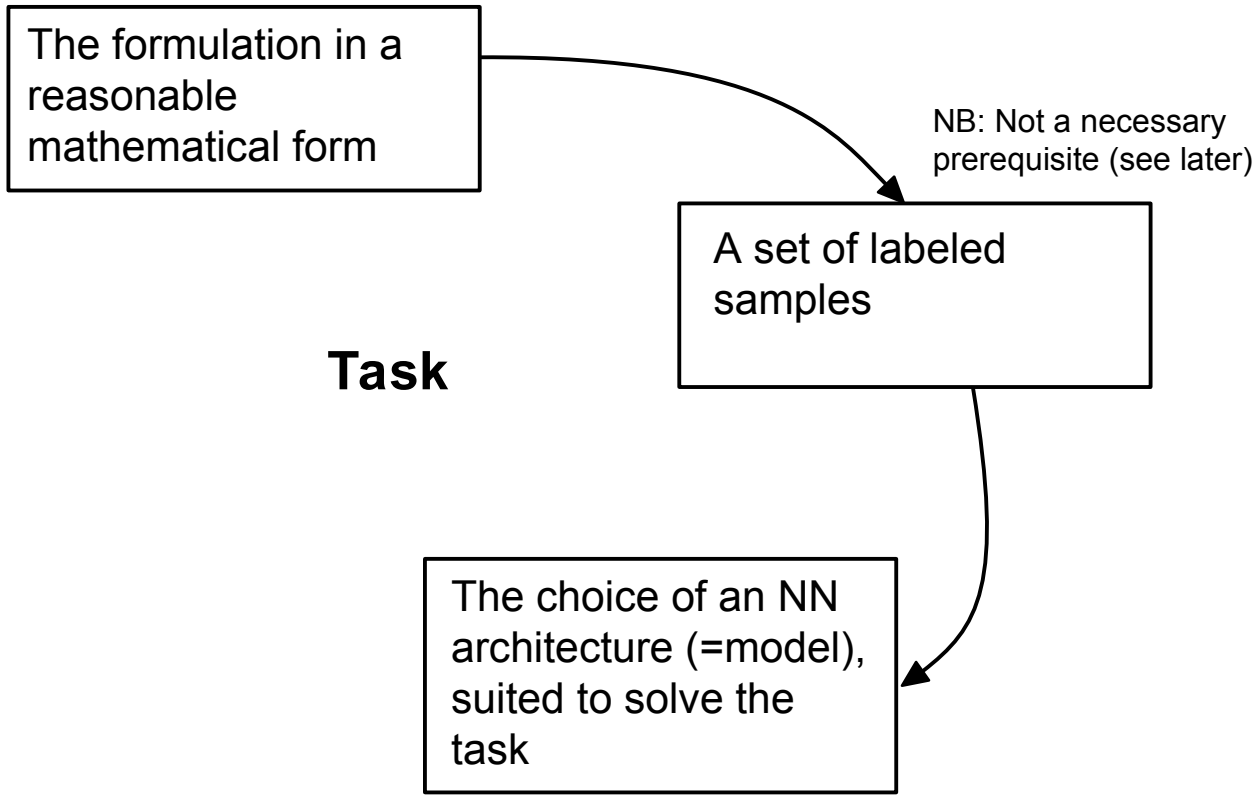
The NN task

- NNs are designed to solve specific *tasks*.
- Each concrete realization of a task requires:



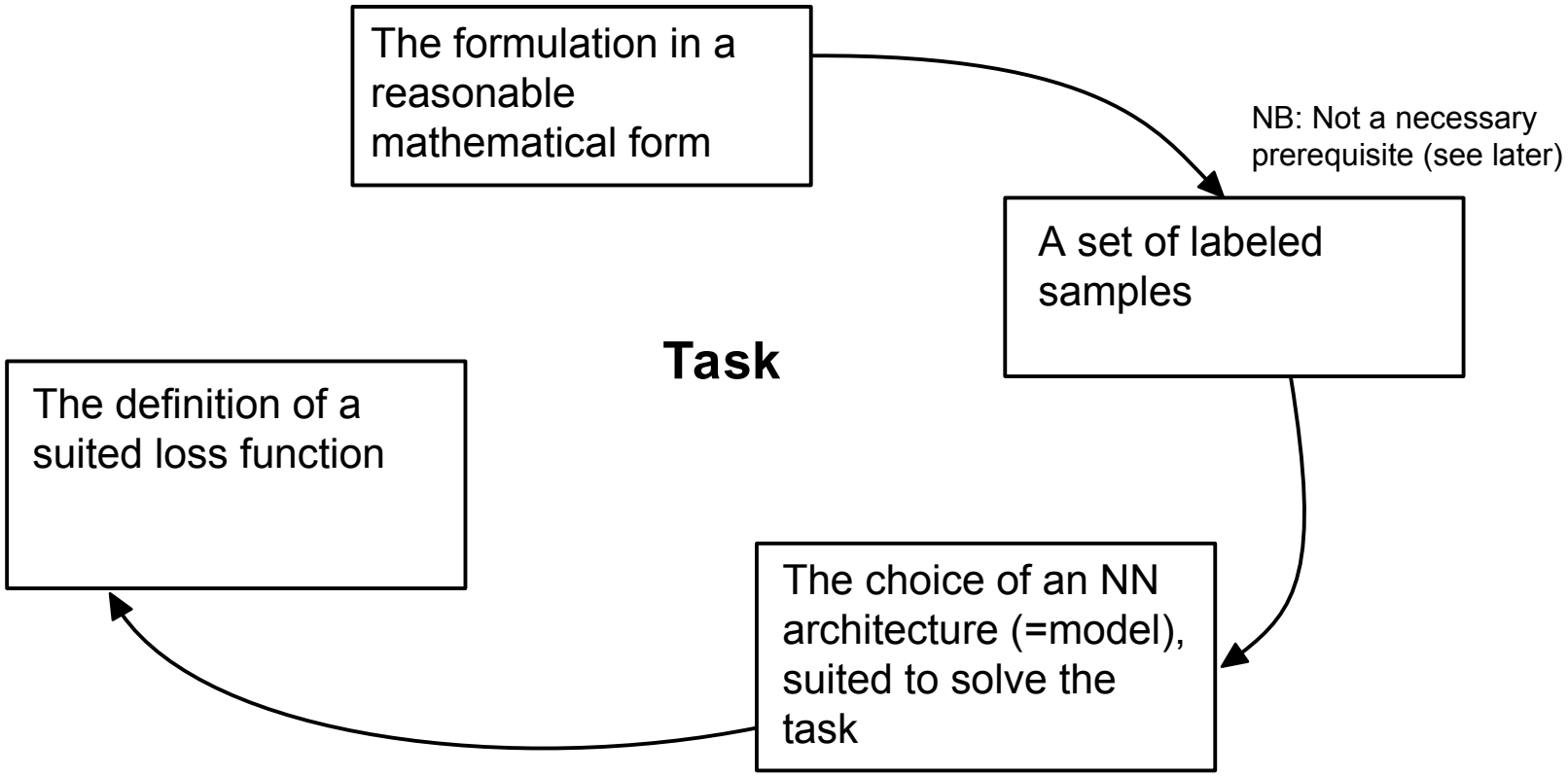
The NN task

- NNs are designed to solve specific *tasks*.
- Each concrete realization of a task requires:



The NN task

- NNs are designed to solve specific *tasks*.
- Each concrete realization of a task requires:



Training as optimization task

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

Training as optimization task

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

- The adaptation of the weights $\boldsymbol{\omega}$ for the MLP to match the target function/contour $y(\mathbf{x})$ thus turns into the known problem of an optimization task.

Training as optimization task

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

- The adaptation of the weights $\boldsymbol{\omega}$ for the MLP to match the target function/contour $y(\mathbf{x})$ thus turns into the known problem of an optimization task.

  - The dimension of this task may still be extraordinarily large, requiring *robust* numerical optimization algorithms.

Training as optimization task

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

- The adaptation of the weights $\boldsymbol{\omega}$ for the MLP to match the target function/contour $y(\mathbf{x})$ thus turns into the known problem of an optimization task.

  - The dimension of this task may still be extraordinarily large, requiring *robust* numerical optimization algorithms.

  - What is the difference between an NN training and a numerical optimization task?

Training as optimization task

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

- The adaptation of the weights $\boldsymbol{\omega}$ for the MLP to match the target function/contour $y(\mathbf{x})$ thus turns into the known problem of an optimization task.

  - The dimension of this task may still be extraordinarily large, requiring *robust* numerical optimization algorithms.

  - What is the difference between an NN training and a numerical optimization task? – Usually we don‘t know $y(\mathbf{x})$. We have to approximate $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ based on **samples**.

An editorial note

- Using differentiable activation functions turns the MLP output function $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ into a function that is *differentiable in any variable*.

- The adaptation of the weights $\boldsymbol{\omega}$ for the MLP to match the target function/contour $y(\mathbf{x})$ thus turns into the known problem of an optimization task.

  - The dimension of this task may still be extraordinarily large, requiring *robust* numerical optimization algorithms.

  - What is the difference between an NN training and a numerical optimization task? – Usually we don‘t know $y(\mathbf{x})$. We have to approximate $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ based on **samples**.

Since we will reserve y for the truth label in the following we will use $\hat{y}$ (the estimate of y made by the applied NN model) for the NN output.

Supervised learning

- In the context of this lecture we will mostly restrict ourselves to *supervised learning*, i.e. for the *training* the MLP has access to the following information, for each sample:

Supervised learning

- In the context of this lecture we will mostly restrict ourselves to *supervised learning*, i.e. for the *training* the MLP has access to the following information, for each sample:
  - The input features $\mathbf{x}$.

Supervised learning

- In the context of this lecture we will mostly restrict ourselves to *supervised learning*, i.e. for the *training* the MLP has access to the following information, for each sample:
  - The input features $\mathbf{x}$.
  - The truth label $y(\mathbf{x})$ (i.e. the association to signal/background for classification, the true value for regression).

Supervised learning

- In the context of this lecture we will mostly restrict ourselves to *supervised learning*, i.e. for the *training* the MLP has access to the following information, for each sample:
  - The input features $\mathbf{x}$.
  - The truth label $y(\mathbf{x})$ (i.e. the association to signal/background for classification, the true value for regression).
  - The MLP does not learn from explicit rules, but from **labeled samples**.

Supervised learning

- In the context of this lecture we will mostly restrict ourselves to *supervised learning*, i.e. for the *training* the MLP has access to the following information, for each sample:
  - The input features $\mathbf{x}$.
  - The truth label $y(\mathbf{x})$ (i.e. the association to signal/background for classification, the true value for regression).
  - The MLP does not learn from explicit rules, but from **labeled samples**.

Supervised learning: *Label* known for each sample.

Unsupervised learning

- Supervised training is contrasted e.g. by *unsupervised learning*:
  - Here the MLP identifies (and/or reproduces), during its later application, structures that it has learned from the training data before.

Unsupervised learning

- Supervised training is contrasted e.g. by *unsupervised learning*:
  - Here the MLP identifies (and/or reproduces), during its later application, structures that it has learned from the training data before.
  - **Example**: two bags of gummy bears. One bag contains **green** gummy bears, the other one doesn‘t. – **Task**: find the green gummy bears!

Unsupervised learning

- Supervised training is contrasted e.g. by *unsupervised learning*:
  - Here the MLP identifies (and/or reproduces), during its later application, structures that it has learned from the training data before.
  - **Example**: two bags of gummy bears. One bag contains **green** gummy bears, the other one doesn‘t. – **Task**: find the green gummy bears!

- **NB**: Here not each bear is labeled any more. The MLP has to identify the difference between the two bags on its own.

Reinforcement learning

- Another way of learning:
  - Here the MLP (in general with some delay) obtains a **reward/penalty** for the decisions that it has made before (e.g. machine-learning based gaming, like chess, Go, …).

The labels (for classification)

- For classification tasks *labeling* of the training data usually happens via **one-hot encoding**.
- **Example**:
  - *Binary classification*:

$$y(\mathbf{x}^{(k)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

  - How would you implement one-hot encoding for multi-classification?

Here $\mathbf{x}^{(k)}$ refers to a sample.

The labels (for classification)

- For classification tasks *labeling* of the training data usually happens via **one-hot encoding**.
- **Example**:
  - *Binary classification*:

$$y(\mathbf{x}^{(k)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

  - How would you implement one-hot encoding for multi-classification?

$$y(\mathbf{x}^{(k)}) = \begin{cases} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \text{Category–1} \\ \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} & \text{Category–2} \\ \cdots & \vdots \\ \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} & \text{Category–n} \end{cases}$$

As a vector with n components each

Here $\mathbf{x}^{(k)}$ refers to a sample.

The labels (for regression)

- For regression tasks the *labels* correspond just to the **real-valued truth**.

- **Example**: when calibrating the response of a calorimeter, it could be the energy of the inpinging pion.

The loss function

- The match of $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ with $y(\mathbf{x})$ is quantified by the **loss function** $L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$, which should also be chosen differentiable in each variable.

The loss function

- The match of $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ with $y(\mathbf{x})$ is quantified by the **loss function** $L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$, which should also be chosen differentiable in each variable.

- $L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$ is evaluated on a single sample.

The loss function

- The match of $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ with $y(\mathbf{x})$ is quantified by the **loss function** $L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$, which should also be chosen differentiable in each variable.

- $L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$ is evaluated on a single sample.

- $L(\cdot, \cdot)$ can be chosen arbitrarily. Very common (likelihood-based) choices are:

  - Cross entropy (CE, binary or categorical);
  - L2-norm squared ($\| \cdot \|_2^2$).

Cross entropy (CE)

- The (categorical) cross entropy (CE) for a (multi-)classification task with n categories for sample (ℓ) is defined as:

$$H\left(\mathbf{y}^{(\ell)}, \hat{\mathbf{y}}^{(\ell)}\right) = -\sum_{j=1}^{n} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right)$$

n : Number of categories

$y_j^{(\ell)}$: Label for category j and sample (ℓ)

$\hat{y}_j^{(\ell)}$: MLP prediction for category j and sample (ℓ)

L_2 norm

- The L_2 norm is a natural choice for regression tasks.

$$\left\| y^{(\ell)} - \hat{y}^{(\ell)} \right\|_2^2 = \left(y^{(\ell)} - \hat{y}^{(\ell)} \right)^2$$

$y_j^{(\ell)}$: Label for sample (ℓ)

$\hat{y}_j^{(\ell)}$: MLP prediction for sample (ℓ)

Risk minimization

NB. For our risk assessment we want to know what the probability is that the NN is right with its decision.

- Assume $p_{\hat{y}}(y(\mathbf{x}))$ as a conditional PDF to obtain a certain label $y(\mathbf{x})$ for given prediction $\hat{y}(\mathbf{x}, \boldsymbol{\omega})$ for fixed values of $\boldsymbol{\omega}$. We call the expectation value of L over $p_{\hat{y}}$

$$R[\boldsymbol{\omega}] = \int p_{\hat{y}}(y(\mathbf{x}))\, L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))\, \mathrm{d}\mathbf{x} = E_{p_{\hat{y}}}[L]$$

NB: R does not depend on $\mathbf{x}$ any more, in this case.

the risk functional.

- **Examples**:
  - CE:

$$R[\boldsymbol{\omega}] = E[H(\mathbf{y}(\mathbf{x}), \hat{\mathbf{y}}(\mathbf{x}, \boldsymbol{\omega}))]$$

  - L_2 norm:

$$R[\boldsymbol{\omega}] = \int p_{\hat{y}}(y(\mathbf{x}))\, \left(y^{(k)} - \hat{y}^{(k)}\right)^2 \mathrm{d}\mathbf{x}$$

- Statistical classification tasks are addressed by **minimizing the risk** (i.e. the expected loss).

Empirical risk minimization

- Since $p_{\hat{y}}(y(\mathbf{x}))$ is not known but has to be estimated from samples the NN training task is solved by minimizing the (sample based) **empirical risk**

Empirical risk minimization

- Since $p_{\hat{y}}(y(\mathbf{x}))$ is not known but has to be estimated from samples the NN training task is solved by minimizing the (sample based) **empirical risk**

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N} \sum_{j=1}^{N} L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$$

NB: R still depends on $\mathbf{x}$ through the choice of samples in this case.

for a sample of size N.

Empirical risk minimization

- Since $p_{\hat{y}}(y(\mathbf{x}))$ is not known but has to be estimated from samples the NN training task is solved by minimizing the (sample based) **empirical risk**

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{j=1}^{N} L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$$

NB: R still depends on $\mathbf{x}$ through the choice of samples in this case.

for a sample of size N.

- **Examples**:
  - CE:
    $$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = -\frac{1}{N}\sum_{\ell=1}^{N}\sum_{j=1}^{n} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right)$$
  - L_2 norm:
    $$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(y^{(\ell)} - \hat{y}^{(\ell)}\right)^2 = \mathrm{MSE}\left[y(\mathbf{x}), \hat{y}(\mathbf{x}, \boldsymbol{\omega})\right]$$

Empirical risk minimization

- Since $p_{\hat{y}}(y(\mathbf{x}))$ is not known but has to be estimated from samples the NN training task is solved by minimizing the (sample based) **empirical risk**

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{j=1}^{N} L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$$

NB: R still depends on $\mathbf{x}$ through the choice of samples in this case.

for a sample of size N.

- **Examples**:
  - CE:
    $$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = -\frac{1}{N}\sum_{\ell=1}^{N}\sum_{j=1}^{n} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right)$$
  - L_2 norm:
    $$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(y^{(\ell)} - \hat{y}^{(\ell)}\right)^2 = \mathrm{MSE}\left[y(\mathbf{x}), \hat{y}(\mathbf{x}, \boldsymbol{\omega})\right]$$

If the $\hat{y}(\mathbf{x}, \boldsymbol{\omega})^{(*)}$ are normal distributed the MSE is a **maximum likelihood estimate** for $y(\mathbf{x})$.

(*) i.e. the estimates of the NN

Empirical risk minimization

- Since $p_{\hat{y}}(y(\mathbf{x}))$ is not known but has to be estimated from samples the NN training task is solved by minimizing the (sample based) **empirical risk**

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{j=1}^{N} L(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$$

NB: R still depends on $\mathbf{x}$ through the choice of samples in this case.

for a sample of size N.

- **Examples**:
  - CE:

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = -\frac{1}{N}\sum_{\ell=1}^{N}\sum_{j=1}^{n} y_j^{(\ell)} \log\left(\hat{y}_j^{(\ell)}\right)$$

If the $\hat{y}(\mathbf{x}, \boldsymbol{\omega})^{(*)}$ are multinomial distributed the CE is a **maximum likelihood estimate** for $y(\mathbf{x})$.

  - L_2 norm:

$$\hat{R}\left[\mathbf{x}, \boldsymbol{\omega}\right] = \frac{1}{N}\sum_{\ell=1}^{N}\left(y^{(\ell)} - \hat{y}^{(\ell)}\right)^2 = \mathrm{MSE}\left[y(\mathbf{x}), \hat{y}(\mathbf{x}, \boldsymbol{\omega})\right]$$

If the $\hat{y}(\mathbf{x}, \boldsymbol{\omega})^{(*)}$ are normal distributed the MSEis a **maximum likelihood estimate** for $y(\mathbf{x})$.

(*) i.e. the estimates of the NN

Cross entropy: Example binary classification

Probability for signal
obtained from known truth

$$y(x^{(k)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Probability for background
obtained from known truth

$$1 - y(x^{(k)})$$

MLP prediction for $y(x^{(k)})$

$$\hat{y}(x^{(k)})$$

MLP prediction for $1 - y(x^{(k)})$

$$1 - \hat{y}(x^{(k)})$$

$$H(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \right)$$

Cross entropy: Example binary classification

Probability for signal
obtained from known truth

$$y(x^{(k)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Probability for background
obtained from known truth

$$1 - y(x^{(k)})$$

MLP prediction for $y(x^{(k)})$

$$\hat{y}(x^{(k)})$$

MLP prediction for $1 - y(x^{(k)})$

$$1 - \hat{y}(x^{(k)})$$

$$H(y, \hat{y}) = -\left(\underbrace{y \log(\hat{y})}_{\text{Correctly identified signal}} + \underbrace{(1-y)\log(1-\hat{y})}_{\text{Correctly identified background}}\right)$$

Cross entropy: Example binary classification

Probability for signal
obtained from known truth

$$y(x^{(k)}) = \begin{cases} 1 & \text{Signal} \\ 0 & \text{Background} \end{cases}$$

Probability for background
obtained from known truth

$$1 - y(x^{(k)})$$

MLP prediction for $y(x^{(k)})$

$$\hat{y}(x^{(k)})$$

MLP prediction for $1 - y(x^{(k)})$

$$1 - \hat{y}(x^{(k)})$$

$$H(y, \hat{y}) = -\left(\underbrace{y \log(\hat{y})}_{\text{Correctly identified signal}} + \underbrace{(1-y)\log(1-\hat{y})}_{\text{Correctly identified background}}\right)$$

- Based on one-hot encoding **CE quantifies only the correctly identified examples** (compare with Rosenblatts learning rule).

Binomial distribution

- Probability of a Bernoulli process for a sample to be identified as signal/background by the MLP:

$$P(\hat{y}, y) = \hat{y}^{y}(1-\hat{y})^{1-y} = \begin{cases} \hat{y} & \text{Signal} \\ 1-\hat{y} & \text{Background} \end{cases}$$

- Likelihood for N Bernoulli processes (keeping track of their order of appearance):

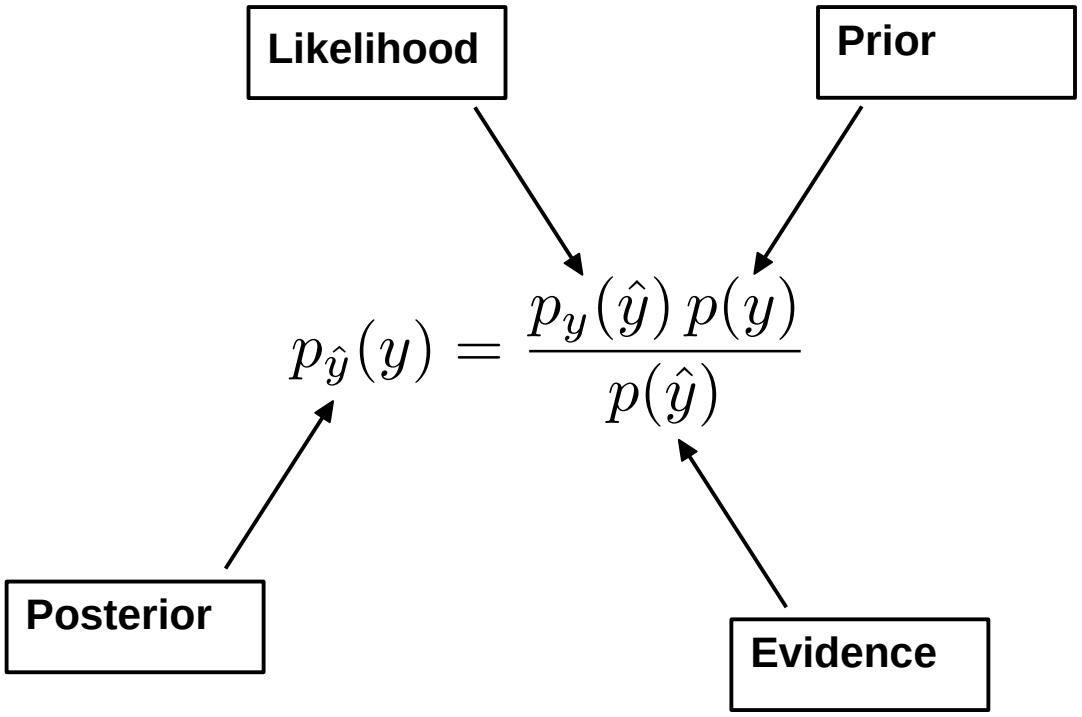
$$\begin{aligned} \mathcal{L}(\{\hat{y}^{(\ell)}\}, \{y^{(\ell)}\}) &= \prod_{\ell=1}^{N} P(\{\hat{y}^{(\ell)}\}, \{y^{(\ell)}\}) \\ &= \prod_{\ell=1}^{N} \hat{y}^{(\ell)y^{(\ell)}}(1-\hat{y}^{(\ell)})^{1-y^{(\ell)}}; \end{aligned}$$

$$\log(\mathcal{L}(\{\hat{y}^{(\ell)}\}, \{y^{(\ell)}\})) = \underbrace{\sum_{\ell=1}^{N}\left(y^{(\ell)}\log(\hat{y}^{(\ell)}) + (1-y^{(\ell)})\log(1-\hat{y}^{(\ell)})\right)}_{\equiv N\,\hat{R}[\mathbf{x}, \boldsymbol{\omega}] \text{ for } n=2}$$

Comparison with slide 17 shows that both expressions are equivalent.

Bayesian statistics

- You might have realized $p_{\hat{y}}(y)$ as the **Bayesian posterior**:



Bayesian statistics

- You might have realized $p_{\hat{y}}(y)$ as the **Bayesian posterior**:

Using simulation you might wonder what the practical difference between posterior, likelihood, and joint PDF is.

Likelihood

MC Simulation

Prior

- Flat (Bayesian prior);
- Output of the generator step.

$$p_{\hat{y}}(y) = \frac{p_y(\hat{y})\, p(y)}{p(\hat{y})}$$

Posterior

Measurement

Evidence

- This touches the heart of doing measurements in modern physics.

Take a break

Numerical minimization

- $\hat{R}(\hat{y}(\mathbf{x}, \boldsymbol{\omega}), y(\mathbf{x}))$ is a real-valued function on a high-dimensional input space:

  $$\hat{R} : \Omega \longrightarrow \mathbb{R}; \quad \boldsymbol{\omega} \longrightarrow \hat{R}(\mathbf{x}, \boldsymbol{\omega})$$

- In a high-dimensional input space the gradient $\nabla \hat{R}$ is perpendicular to the tangential hyperplane of the equipotential contour of $\hat{R}$, and it points in the direction of the steepest ascent of $\hat{R}$.

- For the minimum of $\hat{R}$ we have the condition:

  $$\nabla_{\boldsymbol{\omega}} \hat{R} = 0 \quad ; \quad H_{ij} = \left[\frac{\partial^2}{\partial \omega_i \partial \omega_j} \hat{R} \right] \text{ (positiv definite)}$$

Gradienten descent

- A well etablished (iterative) numerical method of minimization is **gradient descent**:
  - Initialize weights $\boldsymbol{\omega}^{(0)}$ randomly.
  - Calculate the gradient in $\boldsymbol{\omega}^{(k)}$.

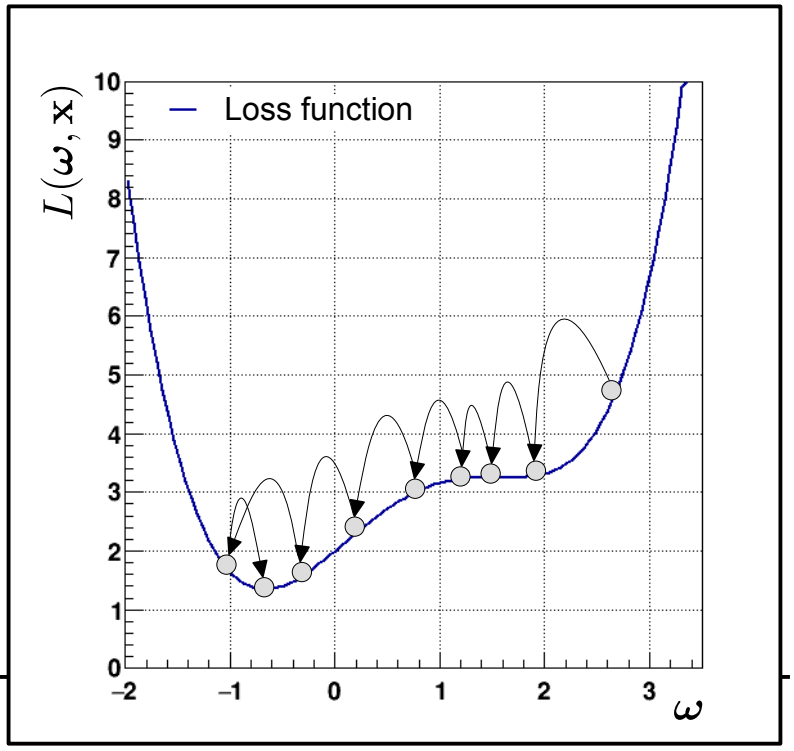
Apply the following update rule:

$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}_k - \eta \nabla_{\boldsymbol{w}^{(k)}} \hat{R}, \quad \eta > 0$$

as long as:

$$\left| \hat{R}(\mathbf{x}_k) - \hat{R}(\mathbf{x}_{k-1}) \right| > \epsilon$$

NB: here the superscripts in parentheses stand for iteration cycles.



Gradienten descent

- A well etablished (iterative) numerical method of minimization is **gradient descent**:
  - Initialize weights $\boldsymbol{\omega}^{(0)}$ randomly.
  - Calculate the gradient in $\boldsymbol{\omega}^{(k)}$.

Apply the following update rule:

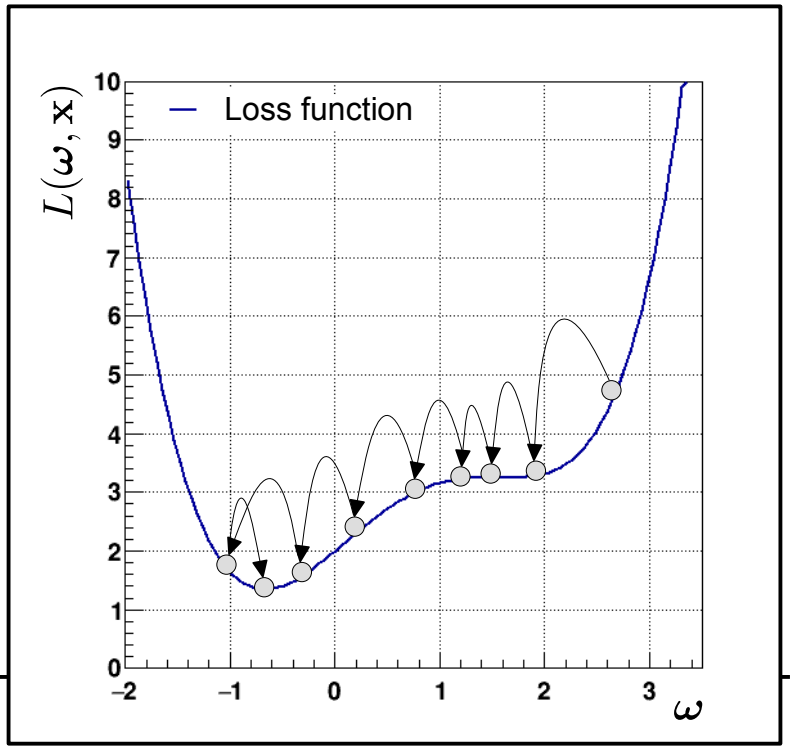
$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}_k - \eta \nabla_{\boldsymbol{w}^{(k)}} \hat{R}, \quad \eta > 0$$

as long as:

$$\left| \hat{R}(\mathbf{x}_k) - \hat{R}(\mathbf{x}_{k-1}) \right| > \epsilon$$

i.e. move in opposite direction of the largest ascent.

NB: here the superscripts in parentheses stand for iteration cycles.



Gradienten descent

- A well etablished (iterative) numerical method of minimization is **gradient descent**:
  - Initialize weights $\boldsymbol{\omega}^{(0)}$ randomly.
  - Calculate the gradient in $\boldsymbol{\omega}^{(k)}$.

η is called **learning rate**.

Apply the following update rule:

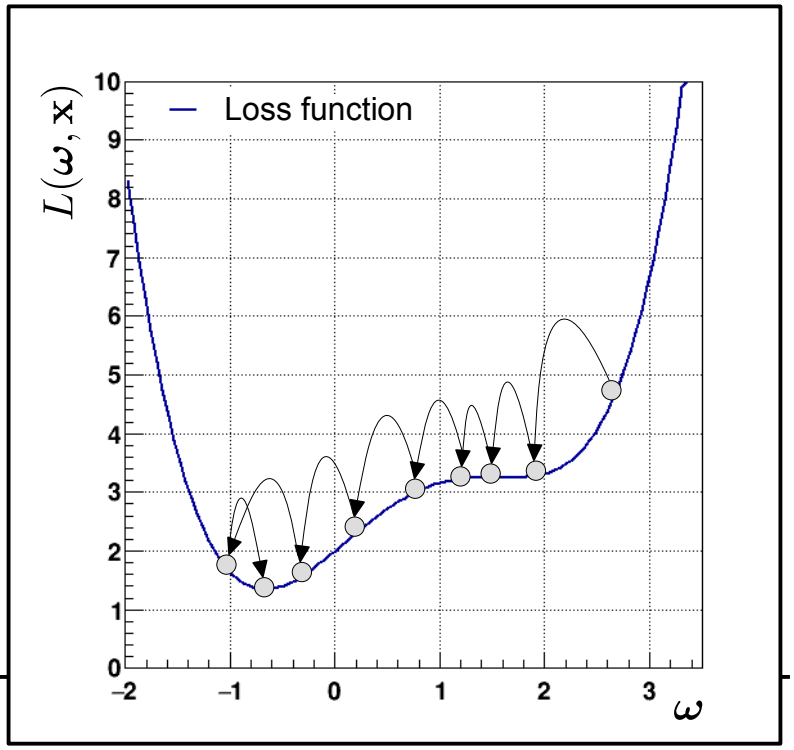
$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}_k - \eta \nabla_{\boldsymbol{w}^{(k)}} \hat{R}, \quad \eta > 0$$

as long as:

$$\left| \hat{R}(\mathbf{x}_k) - \hat{R}(\mathbf{x}_{k-1}) \right| > \epsilon$$

i.e. move in opposite direction of the largest ascent.

NB: here the superscripts in parentheses stand for iteration cycles.

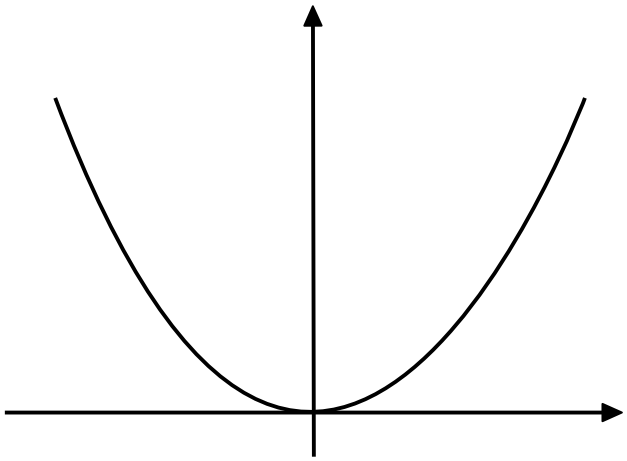


Optimal learning rate

- To identify the optimal learning rate we discuss the simple case of a polynomial of 2nd order:

$$F(z) = \frac{1}{2}az^2 + bz + c$$

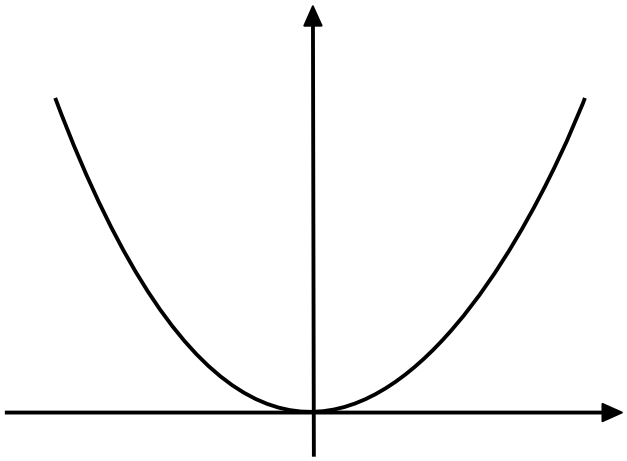
- How would you choose η to find the minimum of $F(z)$ as quickly as possible and how many steps do you think will be required?



Optimal learning rate

- To identify the optimal learning rate we discuss the simple case of a polynomial of 2nd order:

$$F(z) = \frac{1}{2}az^2 + bz + c$$



- How would you choose η to find the minimum of $F(z)$ as quickly as possible and how many steps do you think will be required?

$$F\,(z) = F(z_0) + F'(z_0)\,(z - z_0) + \frac{1}{2}F''(z_0)(z_0)\,(z - z_0)^2$$

$$F'(z) = F'(z_0) + F''(z_0)\,(z - z_0) = 0$$

$$z_{\min} = z_0 - \frac{F'(z_0)}{F''(z_0)}$$

Compare with previously defined update rule:

$$z_1 = z_0 - \eta F'(z_0) \qquad \Rightarrow \qquad \eta_{\text{opt}} = \frac{1}{F''(z_0)} \quad \text{mit: } F''(z) = a \quad (\forall z \in \mathbb{R})$$

i.e. you always reach the minimum in **one step**!

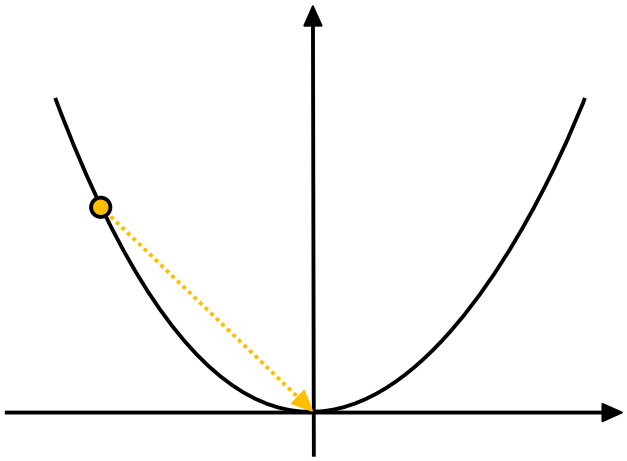
Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

$$(1) \quad \eta = \frac{1}{F''(z)}$$

} Convergence after one step



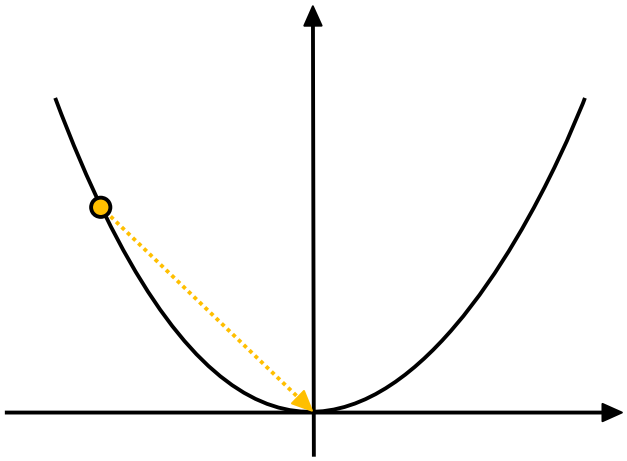
Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

(1) $\eta = \dfrac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \dfrac{1}{F''(z)}$



Discussion of convergence behavior

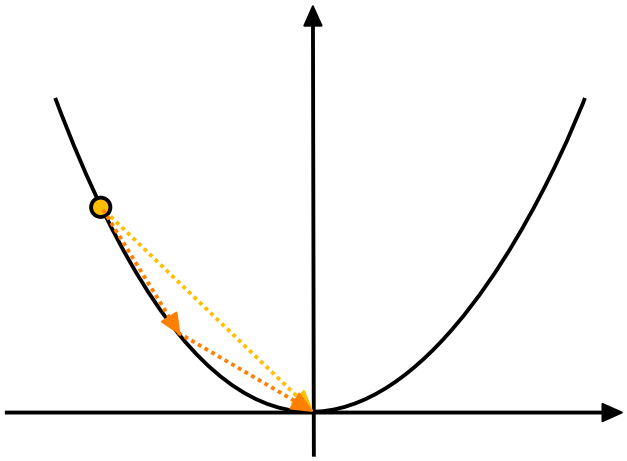
$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

(1) $\eta = \dfrac{1}{F''(z)}$

(2) $\eta < \dfrac{1}{F''(z)}$

} Convergence after one step



Discussion of convergence behavior

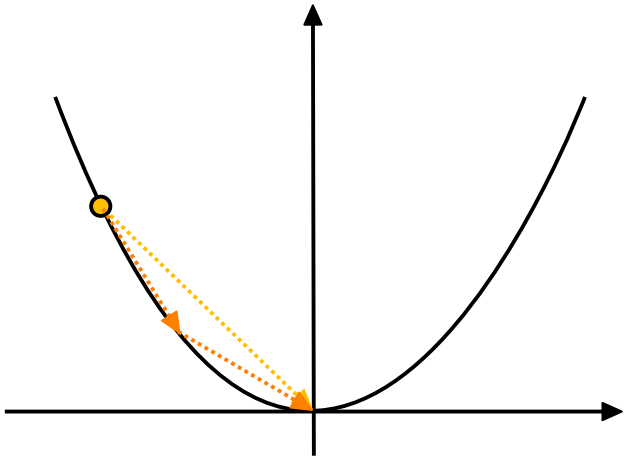
$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$

(3) $\eta > \frac{1}{F''(z)}$



Discussion of convergence behavior

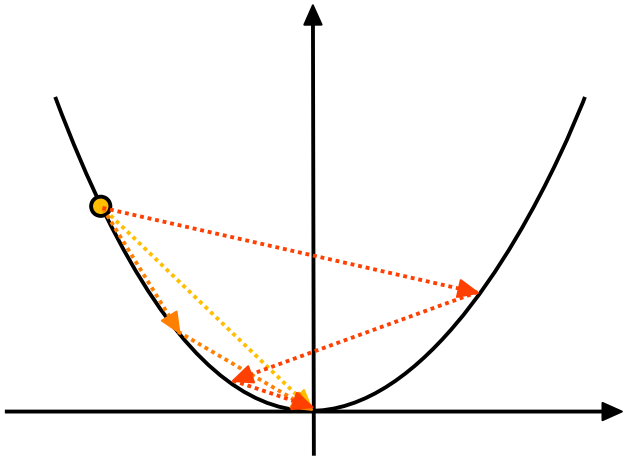
$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$ } Convergence after more than one step (potentially oscillating)

(3) $\eta > \frac{1}{F''(z)}$ }



Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

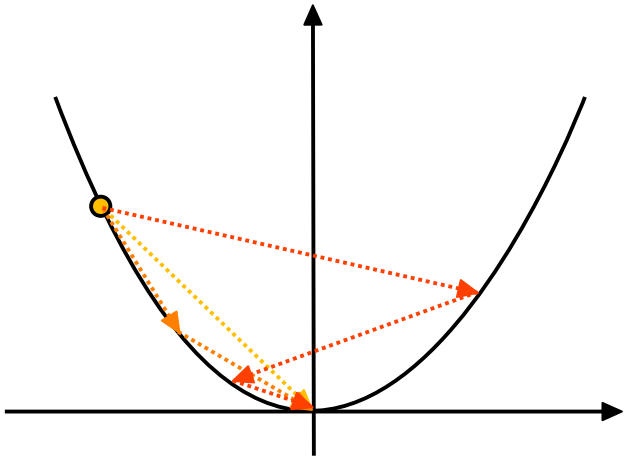
- We check the convergence behavior for different values of η :

(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$ } Convergence after more than one step (potentially oscillating)

(3) $\eta > \frac{1}{F''(z)}$ }

(4) $\eta = 2\frac{1}{F''(z)}$



Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

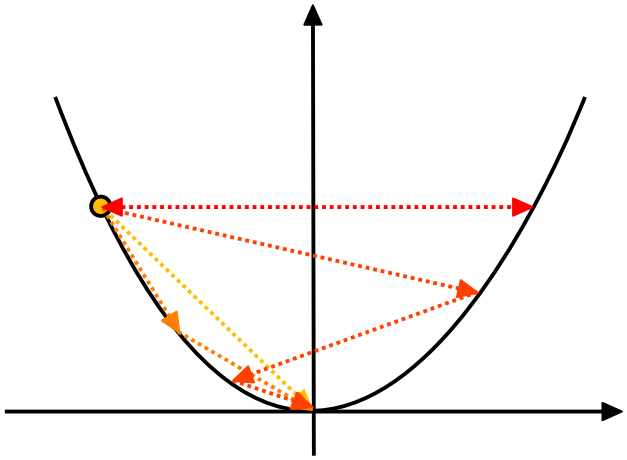
- We check the convergence behavior for different values of η :

(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$ } Convergence after more than one step (potentially oszillating)

(3) $\eta > \frac{1}{F''(z)}$ }

(4) $\eta = 2\frac{1}{F''(z)}$ } Stabile oscillation



Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

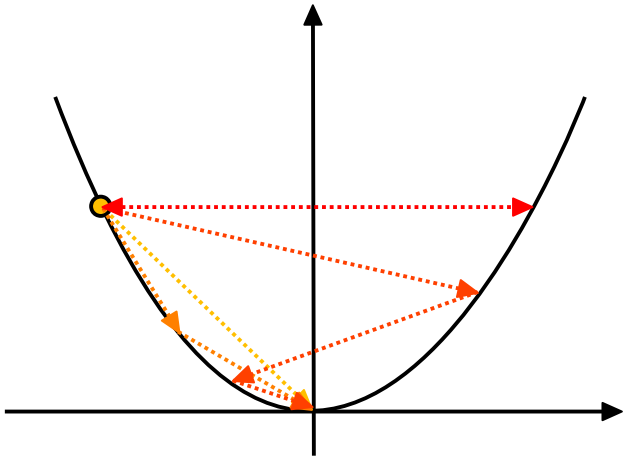
(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$ } Convergence after more than one step (potentially oszillating)

(3) $\eta > \frac{1}{F''(z)}$ }

(4) $\eta = 2\frac{1}{F''(z)}$ } Stabile oscillation

(5) $\eta > 2\frac{1}{F''(z)}$



Discussion of convergence behavior

$$F(z) = \frac{1}{2}az^2 + bz + c$$

- We check the convergence behavior for different values of η :

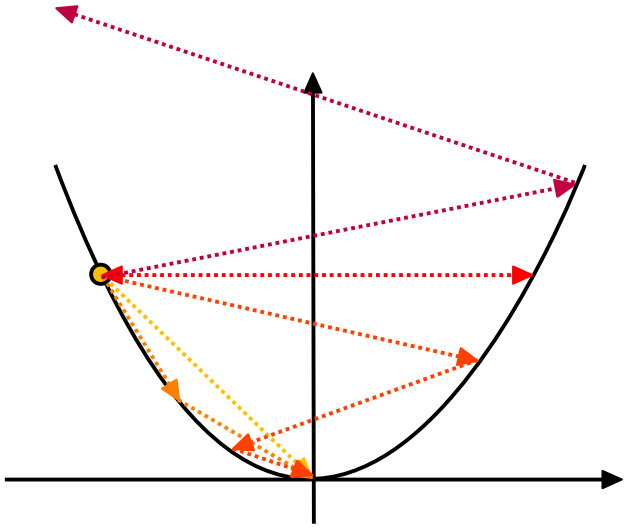
(1) $\eta = \frac{1}{F''(z)}$ } Convergence after one step

(2) $\eta < \frac{1}{F''(z)}$ } Convergence after more than one step (potentially oszillating)

(3) $\eta > \frac{1}{F''(z)}$ }

(4) $\eta = 2\frac{1}{F''(z)}$ } Stabile oscillation

(5) $\eta > 2\frac{1}{F''(z)}$ } Divergence

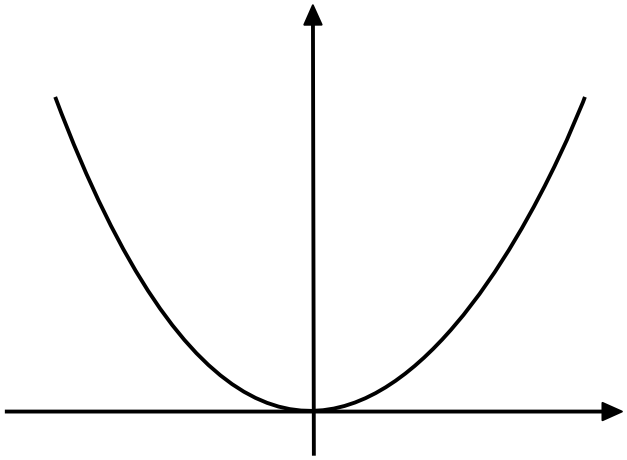


Optimal learning rate

- To identify the optimal learning rate we discuss the simple case of a polynomial of 2nd order:

$$F(z) = \frac{1}{2}az^2 + bz + c$$

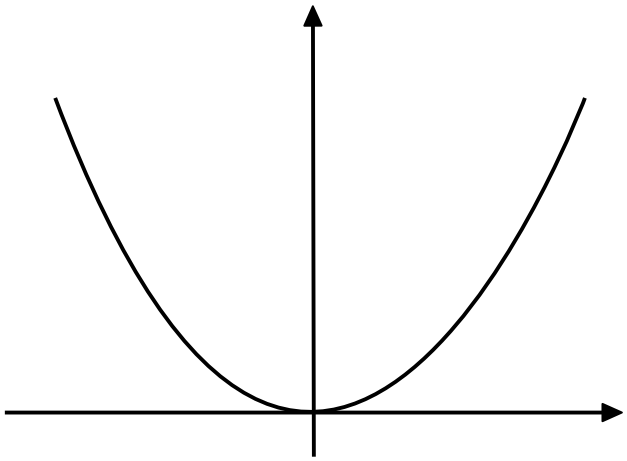
- The discussion up to here gan be **generalized** to any twice differentiable function as long as we expand it only up to second order in the Taylor series.



Optimal learning rate

- To identify the optimal learning rate we discuss the simple case of a polynomial of 2nd order:

$$F(z) = \frac{1}{2}az^2 + bz + c$$



- The discussion up to here gan be **generalized** to any twice differentiable function as long as we expand it only up to second order in the Taylor series.

- The update rule takes the following form:

$$z_{k+1} = z_k - \frac{F'(z_k)}{F''(z_k)}$$

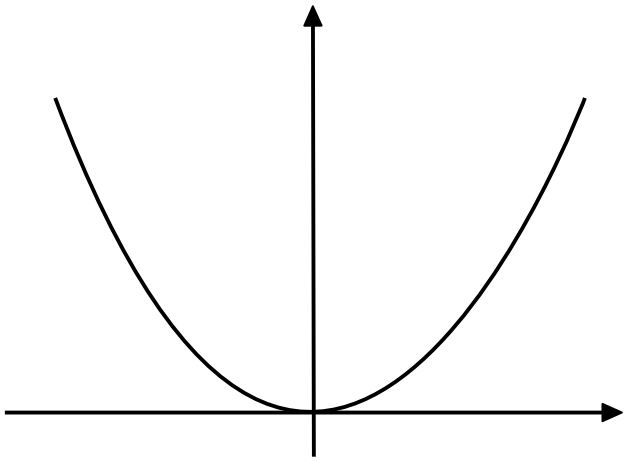
as long as:

$$|F(z_k) - F(z_{k-1})| > \epsilon$$

Optimal learning rate

- To identify the optimal learning rate we discuss the simple case of a polynomial of 2nd order:

$$F(z) = \frac{1}{2}az^2 + bz + c$$



- The discussion up to here gan be **generalized** to any twice differentiable function as long as we expand it only up to second order in the Taylor series.

- The update rule takes the following form:

$$z_{k+1} = z_k - \frac{F'(z_k)}{F''(z_k)}$$

as long as:

$$|F(z_k) - F(z_{k-1})| > \epsilon$$

This is the well-known Newton method for root finding, here applied to $F'(z)$.

Newton method in higher dimensions

- When expanding to higher dimensions $F''(z)^{-1}$ is replaced by the inverse of the **Hessian** $H^{-1}(\mathbf{z})$ in the update rule:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - H^{-1}(\mathbf{z}_k)\nabla F(\mathbf{z}_k)$$

as long as:

$$|F(\mathbf{z}_k) - F(\mathbf{z}_{k-1})| > \epsilon$$

- This is **too computing expensive** for applications in modern ML.

Adaptive learning rate

- Another possibility to prevent convergence issues is to start with a potentially large learning rate that is successively reduced after some „burn-in“ time.

- Starting with a larger learning rate is supposed to prevent getting stuck in a side minimum by unfortunate choice of the initial parameters.

- When following this road the following boundary conditions *should* apply to the learning rates:

$$\sum_{k=1}^{\infty} \eta_k \rightarrow \infty$$

It must be possible to cover the complete input space of the (loss) function with an infinite sequence of steps.

Adaptive learning rate

- Another possibility to prevent convergence issues is to start with a potentially large learning rate that is successively reduced after some „burn-in“ time.

- Starting with a larger learning rate is supposed to prevent getting stuck in a side minimum by unfortunate choice of the initial parameters.

- When following this road the following boundary conditions *should* apply to the learning rates:

$$\sum_{k=1}^{\infty} \eta_k \rightarrow \infty$$

It must be possible to cover the complete input space of the (loss) function with an infinite sequence of steps.

$$\sum_{k=1}^{\infty} \eta_k^2 < \infty$$

The sequence of steps should still be convergent and its elements should become smaller and smaller.

Adaptive learning rate

- Another possibility to prevent convergence issues is to start with a potentially large learning rate that is successively reduced after some „burn-in“ time.

- Starting with a larger learning rate is supposed to prevent getting stuck in a side minimum by unfortunate choice of the initial parameters.

- Established adaptive learning rate algorithms are(*):

$$\eta_k = \frac{\eta_0}{k+1} \qquad \text{(Linear)}$$

$$\eta_k = \frac{\eta_0}{(k+1)^2} \qquad \text{(Quadratic)}$$

$$\eta_k = \eta_0\, e^{-\beta\, k} \quad \beta > 0 \qquad \text{(Exponential)}$$

(*) Not necessarily obeying the rules given before.

Momentum methods

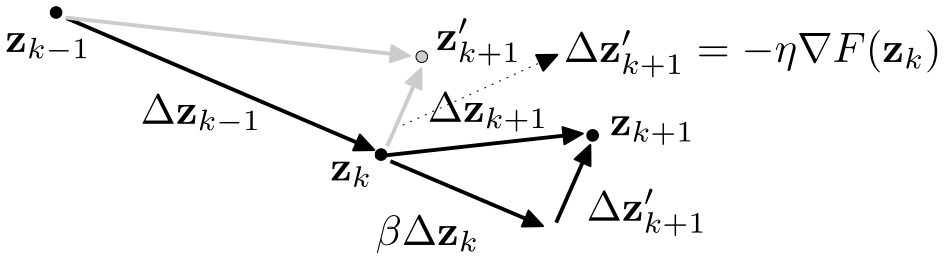
- Momentum methods rely on the assumption that the original direction of the descent was already a good choice for finding the minimum.

- Keep some „memory“ of this original information in the gradient-descent update rule, e.g. through a (weighted) mean:

$$\Delta \mathbf{z}_{k+1} = \beta \Delta \mathbf{z}_k - \eta \nabla F(\mathbf{z}_k)$$
$$\mathbf{z}_{k+1} = \mathbf{z}_k + \Delta \mathbf{z}_{k+1}$$

as long as:

$$|F(\mathbf{z}_k) - F(\mathbf{z}_{k-1})| > \epsilon$$



Simple gradient descent (primed variables) in gray.

Gradient descent by momentum method in **black**.

- This simple method does not require any second derivative. It supresses quickly varying changes in the descent and thus also oscillations.

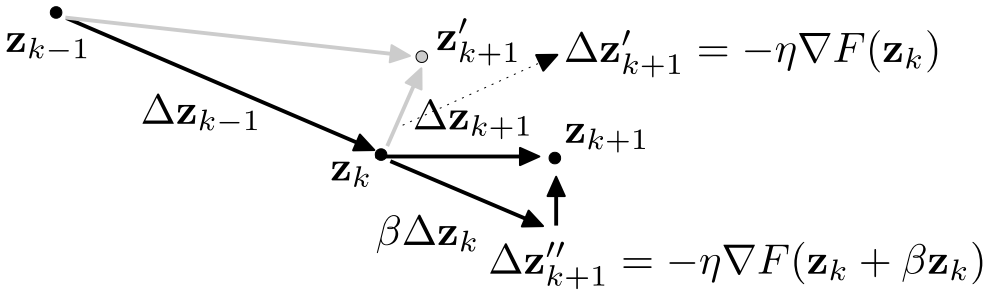
Nesterov's accelerated gradient (Y. Nesterov 1983)

- A variation of the momentum method with proven better convergence behavior first applies the step of the previous descent before evaluating the derivative, there:

$$\Delta\mathbf{z}_{k+1} = \beta\Delta\mathbf{z}_k - \eta\nabla F(\mathbf{z}_k + \beta\Delta\mathbf{z}_k)$$
$$\mathbf{z}_{k+1} = \mathbf{z}_k + \Delta\mathbf{z}_{k+1}$$

as long as:

$$|F(\mathbf{z}_k) - F(\mathbf{z}_{k-1})| > \epsilon$$



Simple gradient descent (primed variables) in gray.

Gradient descent by momentum method in **black**.

Putting things together…

- Up to this point we have discussed the update rule in general.
- We will next discuss how to calculate the gradients w.r.t. thousands of weights in a practical and efficient way.

Apply the following update rule:

$$\boldsymbol{\omega}^{(k+1)} = \boldsymbol{\omega}_k - \eta \nabla_{\boldsymbol{w}^{(k)}} \hat{R}, \quad \eta > 0$$

as long as:

$$\left| \hat{R}(\boldsymbol{\omega}_k) - \hat{R}(\boldsymbol{\omega}_{k-1}) \right| > \epsilon$$

Derivative at a fixed point in the MLP

NB: Here shown for a signalevent.

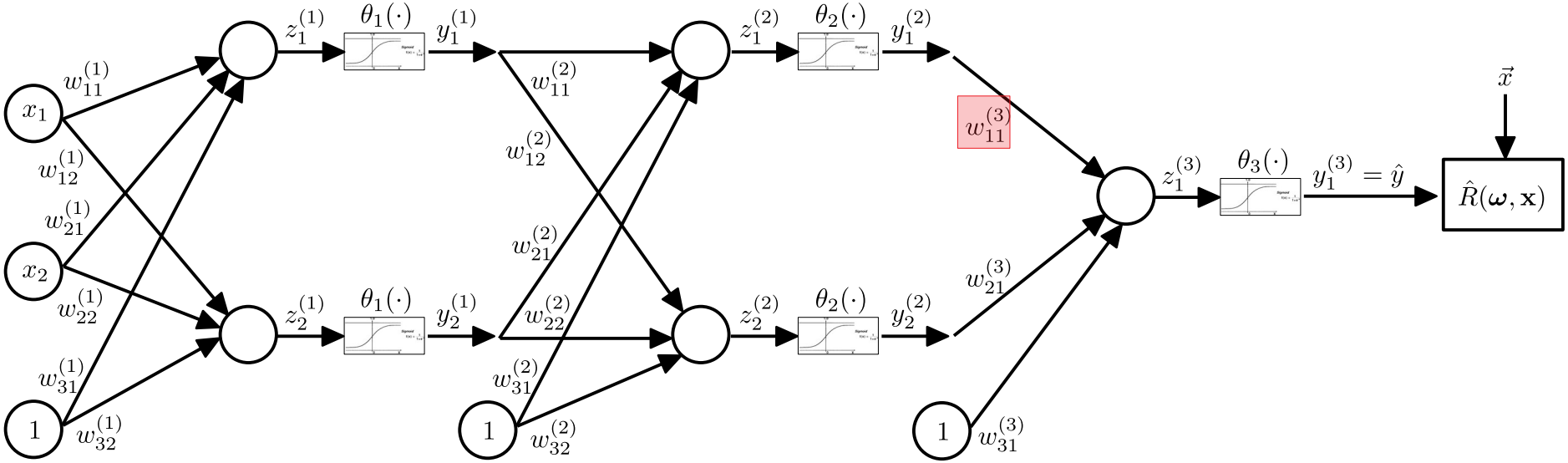
$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{11}^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} \cdot \frac{\mathrm{d}\hat{y}}{\mathrm{d}z_1^{(3)}} \cdot \frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}}$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} = \frac{1}{\hat{y}} \qquad \text{(for cross entropy)}$$

$$\frac{\mathrm{d}y}{\mathrm{d}z_1^{(3)}} = \theta_3(z)\,(1 - \theta_3(z)) \qquad \text{(for sigmoid)}$$

$$\frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}} = y_1^{(2)}$$

- Derivative happens following the chain rule.



Derivative at a fixed point in the MLP

NB: Here shown for a signalevent.

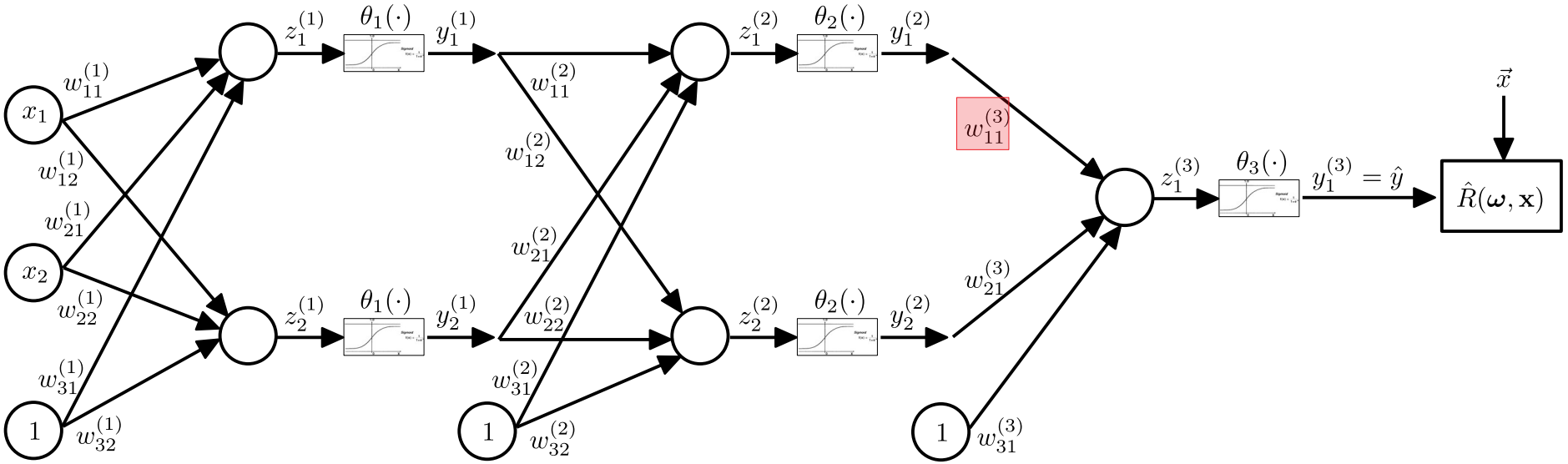
$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{11}^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} \cdot \frac{\mathrm{d}\hat{y}}{\mathrm{d}z_1^{(3)}} \cdot \frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}}$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} = \frac{1}{\hat{y}}\bigg|_{y(\boldsymbol{\omega},\mathbf{x})} \qquad \text{(for cross entropy)}$$

$$\frac{\mathrm{d}y}{\mathrm{d}z_1^{(3)}} = \theta_3(z)\,(1-\theta_3(z))\bigg|_{z=z_1^{(3)}(\boldsymbol{\omega},\mathbf{x})} \qquad \text{(for sigmoid)}$$

$$\frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}} = y_1^{(2)}\bigg|_{y_1^{(2)}(\boldsymbol{\omega},x)}$$

- Derivative happens following the chain rule.
- To obtain a concrete value of the derivative the function values in the arguments are required.



Derivative at a fixed point in the MLP

NB: Here shown for a signalevent.

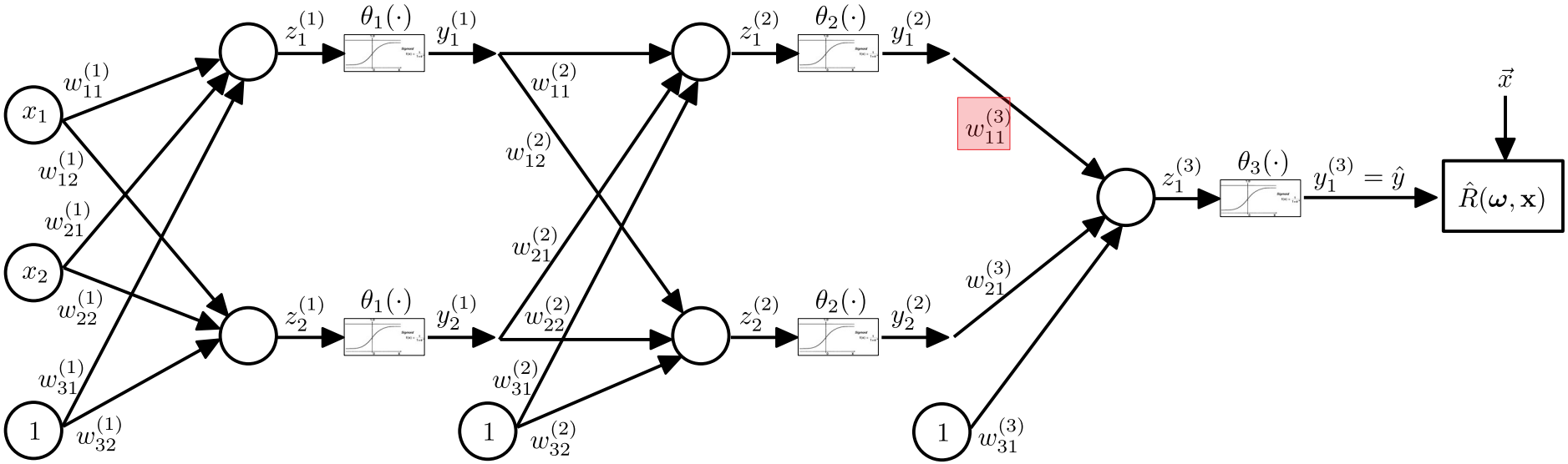
$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{11}^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} \cdot \frac{\mathrm{d}\hat{y}}{\mathrm{d}z_1^{(3)}} \cdot \frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}}$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}} = \frac{1}{\hat{y}}\bigg|_{y(\boldsymbol{\omega},\mathbf{x})} \qquad \text{(for cross entropy)}$$

$$\frac{\mathrm{d}y}{\mathrm{d}z_1^{(3)}} = \theta_3(z)\,(1-\theta_3(z))\bigg|_{z=z_1^{(3)}(\boldsymbol{\omega},\mathbf{x})} \qquad \text{(for sigmoid)}$$

$$\frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{11}^{(3)}} = y_1^{(2)}\bigg|_{y_1^{(2)}(\boldsymbol{\omega},x)}$$

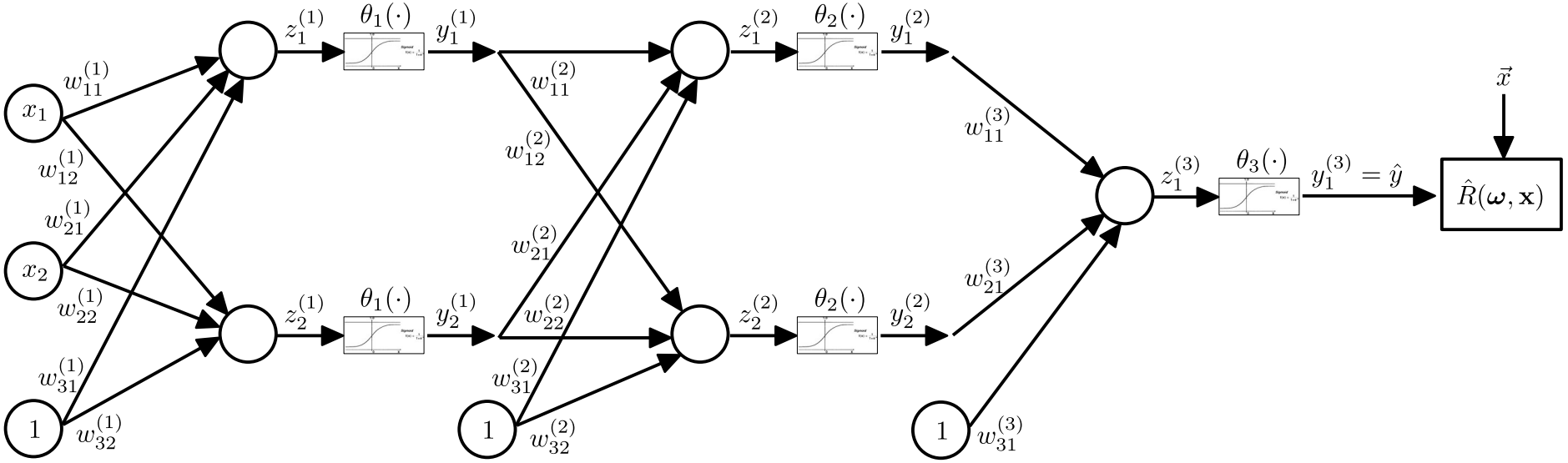
- Derivative happens following the chain rule.
- To obtain a concrete value of the derivative the function values in the arguments are required.
- i.e. each gradient descent requires **two steps**!



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

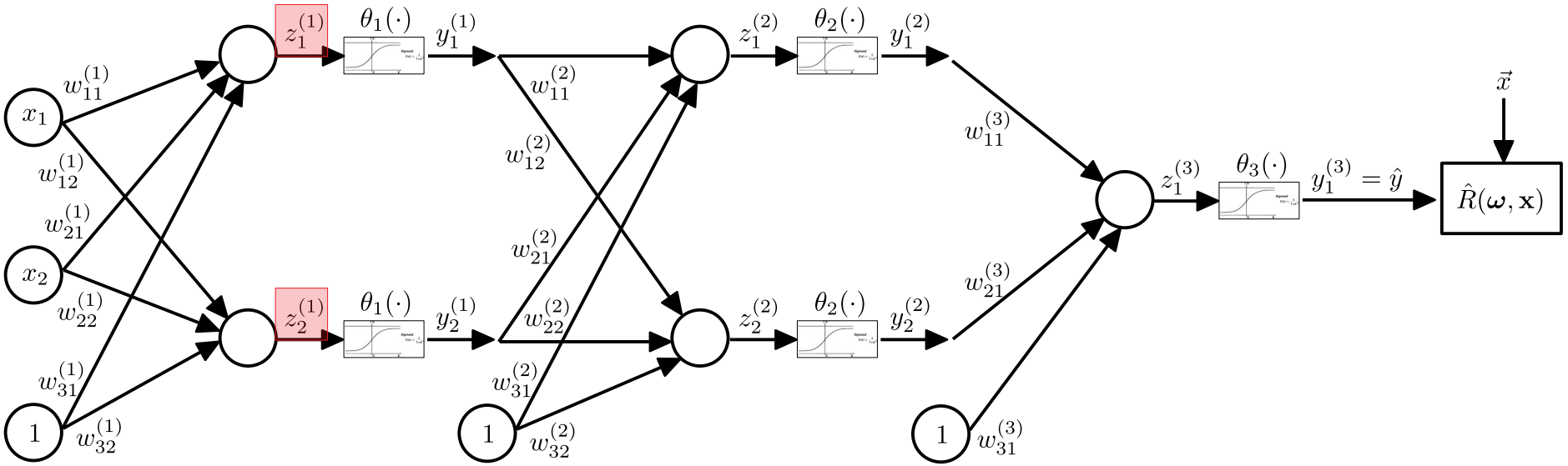
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3 \left(\sum \omega_{ij}^{(3)} \theta_2 \left(\sum \omega_{ij}^{(2)} \theta_1 \left(\sum \omega_{ij}^{(1)} x_i \right) \right) \right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

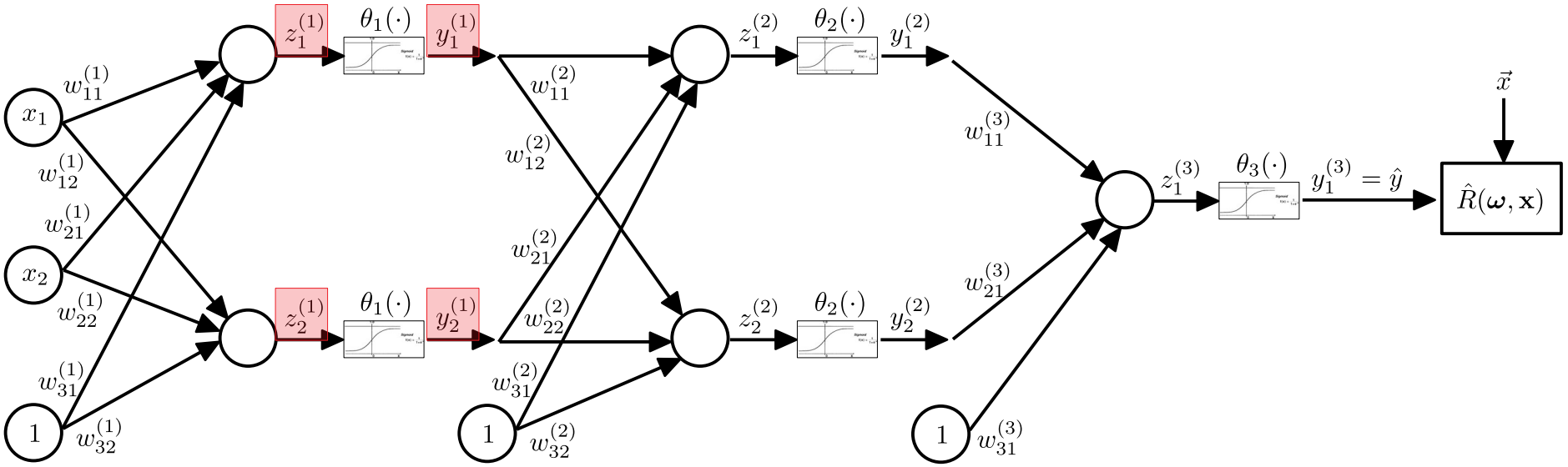
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3 \left(\sum \omega_{ij}^{(3)} \theta_2 \left(\sum \omega_{ij}^{(2)} \theta_1 \left(\sum \omega_{ij}^{(1)} x_i \right) \right) \right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

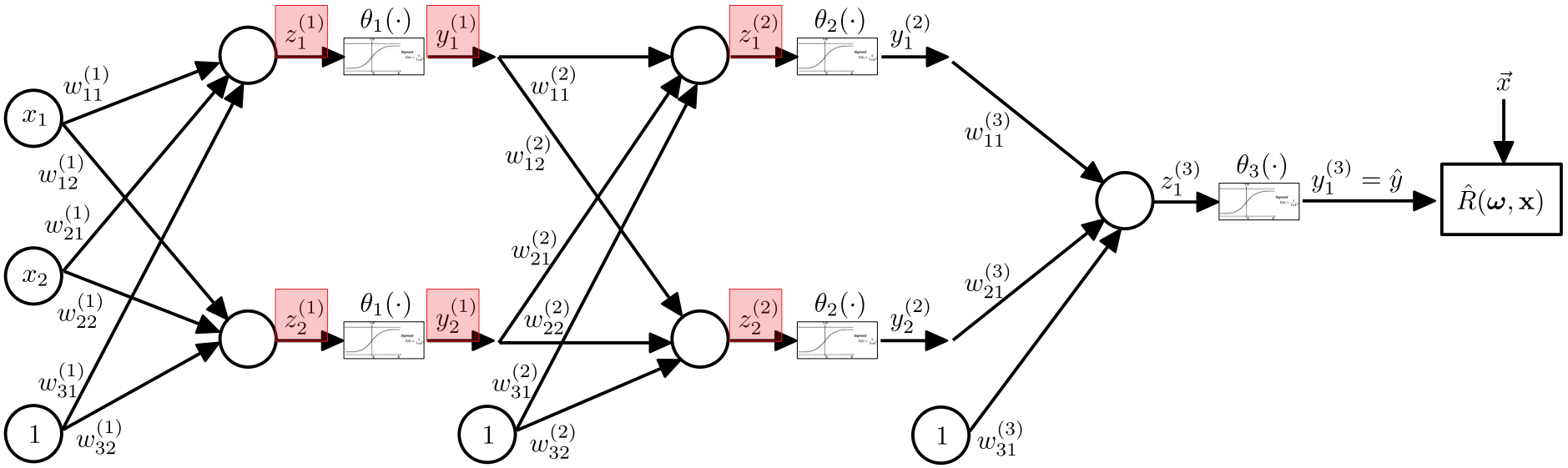
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3\left(\sum \omega_{ij}^{(3)} \theta_2\left(\sum \omega_{ij}^{(2)} \theta_1\left(\sum \omega_{ij}^{(1)} x_i\right)\right)\right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

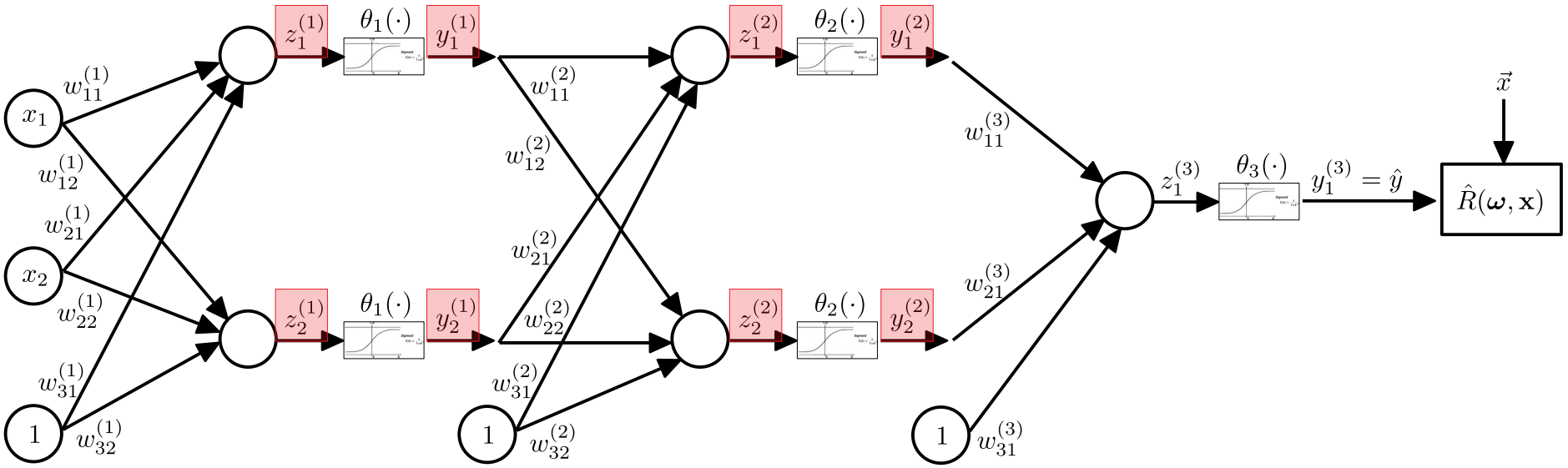
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3 \left(\sum \omega_{ij}^{(3)} \theta_2 \left(\sum \omega_{ij}^{(2)} \theta_1 \left(\sum \omega_{ij}^{(1)} x_i \right) \right) \right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

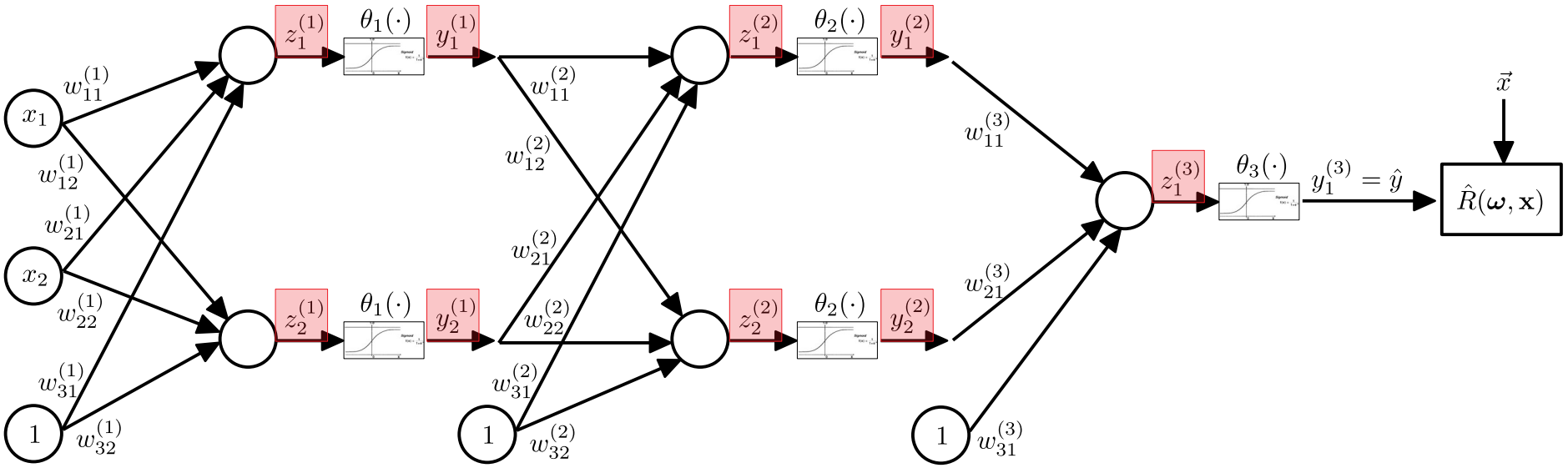
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3 \left(\sum \omega_{ij}^{(3)} \theta_2 \left(\sum \omega_{ij}^{(2)} \theta_1 \left(\sum \omega_{ij}^{(1)} x_i \right) \right) \right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration ,layer by layer:

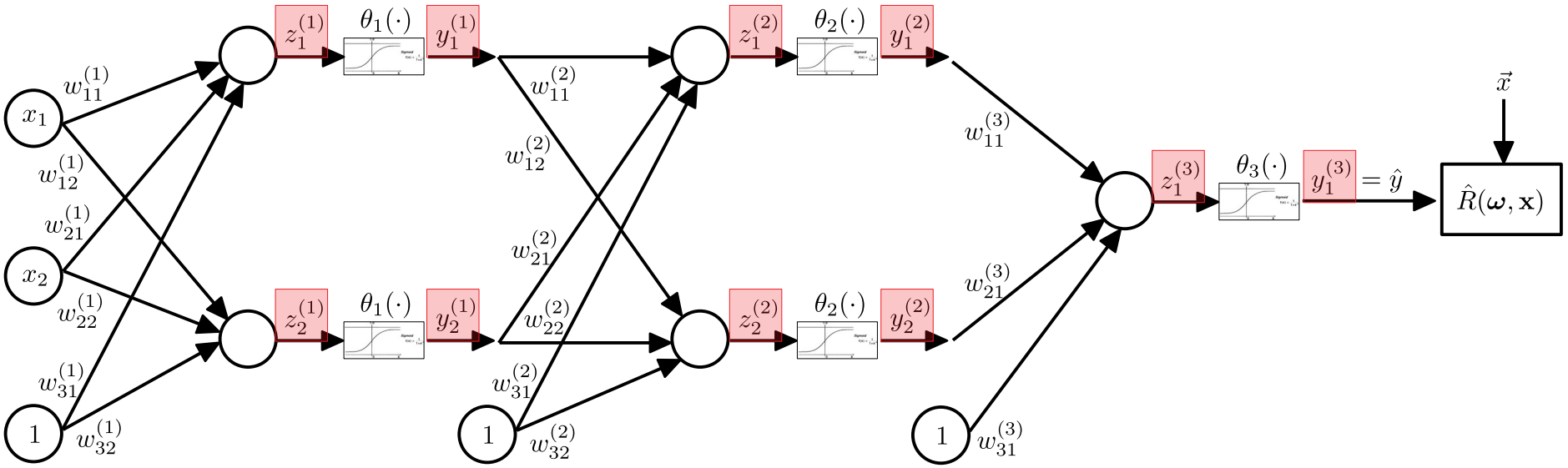
$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3 \left(\sum \omega_{ij}^{(3)} \theta_2 \left(\sum \omega_{ij}^{(2)} \theta_1 \left(\sum \omega_{ij}^{(1)} x_i \right) \right) \right)$$



Step-1: Forward pass

- Calculate and safe all $\{z_i^{(k)}\}$ and $\{y_i^{(k)}\}$ for the given MLP configuration, layer by layer:

$$\hat{y}(\boldsymbol{\omega}, \mathbf{x}) = y_1^{(3)}(\boldsymbol{\omega}, \mathbf{x}) = \theta_3\left(\sum \omega_{ij}^{(3)} \theta_2\left(\sum \omega_{ij}^{(2)} \theta_1\left(\sum \omega_{ij}^{(1)} x_i\right)\right)\right)$$



Backpropagation – part 1

Forward pass:

- Start with: $y_j^{(0)} = x_j$

- For each layer $k = 1 \ldots N$:

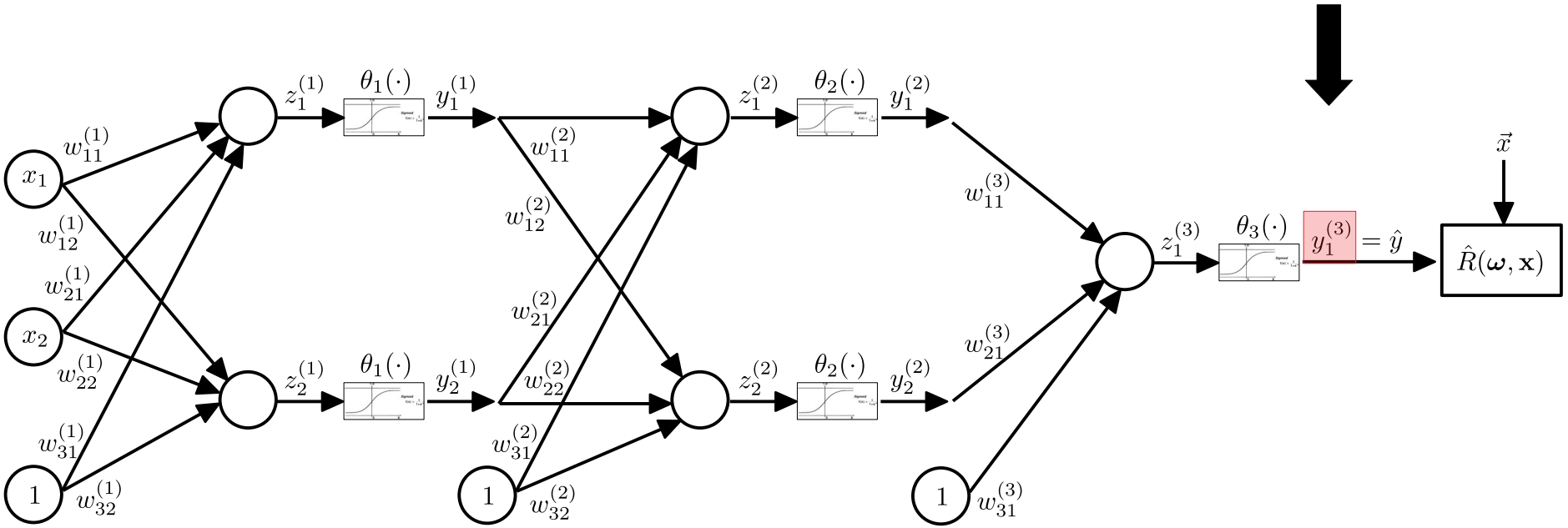
$$z_i^{(k)} = \sum_j y_j^{(k-1)} \omega_{ij}^{(k)}$$

$$y_i^{(k)} = \theta_k \left(z_i^{(k)} \right)$$

Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\hat{y}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1^{(3)}}$$

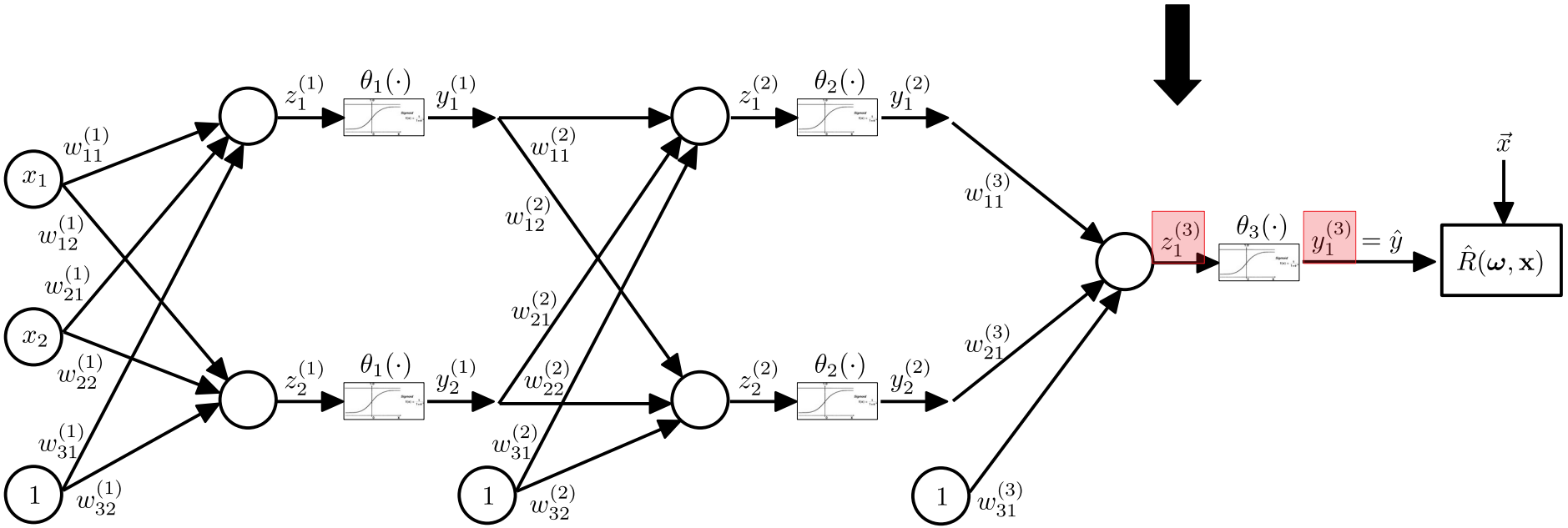


Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1^{(3)}} \cdot \theta_3'\left(z_1^{(3)}\right)$$

Already calculated in previous step



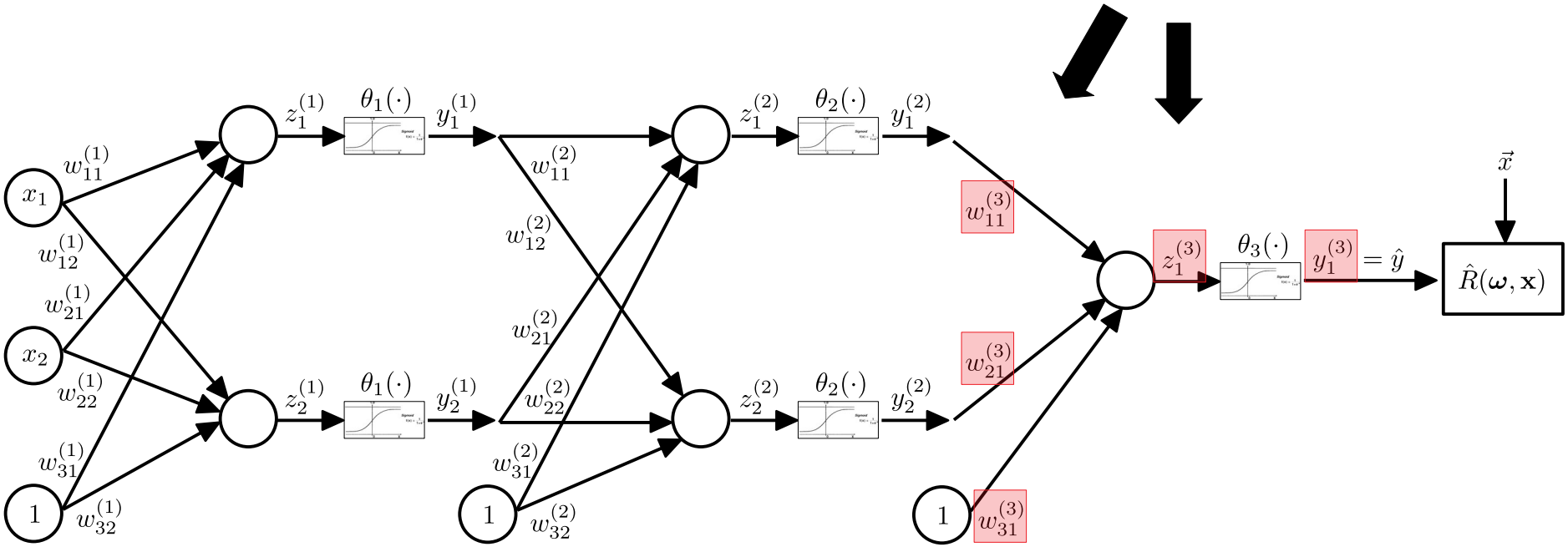
Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1^{(3)}} \cdot \theta_3'\left(z_1^{(3)}\right)$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{i1}^{(3)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}} \cdot \frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}\omega_{i1}^{(3)}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}} \cdot y_i^{(2)}$$

Already calculated in previous step

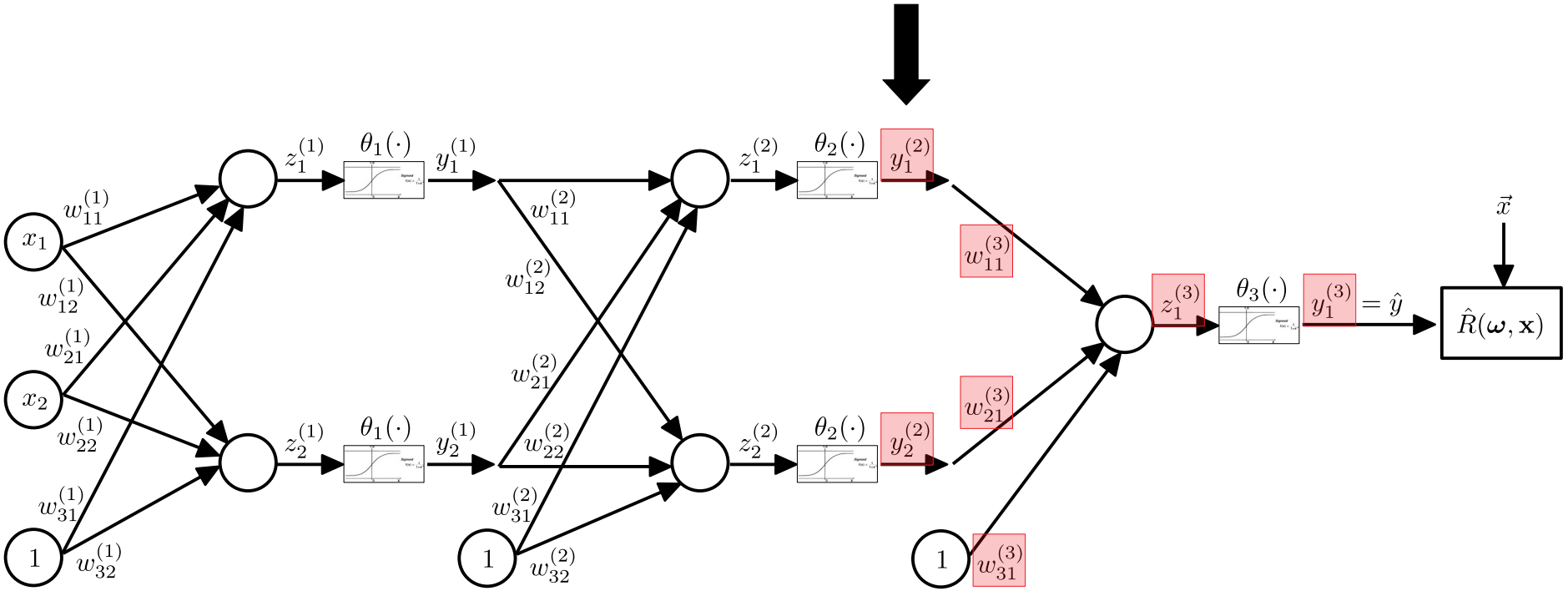


Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(2)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}} \cdot \frac{\mathrm{d}z_1^{(3)}}{\mathrm{d}y_i^{(2)}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_1^{(3)}} \cdot \omega_{i1}^{(3)}$$

Already calculated in previous step

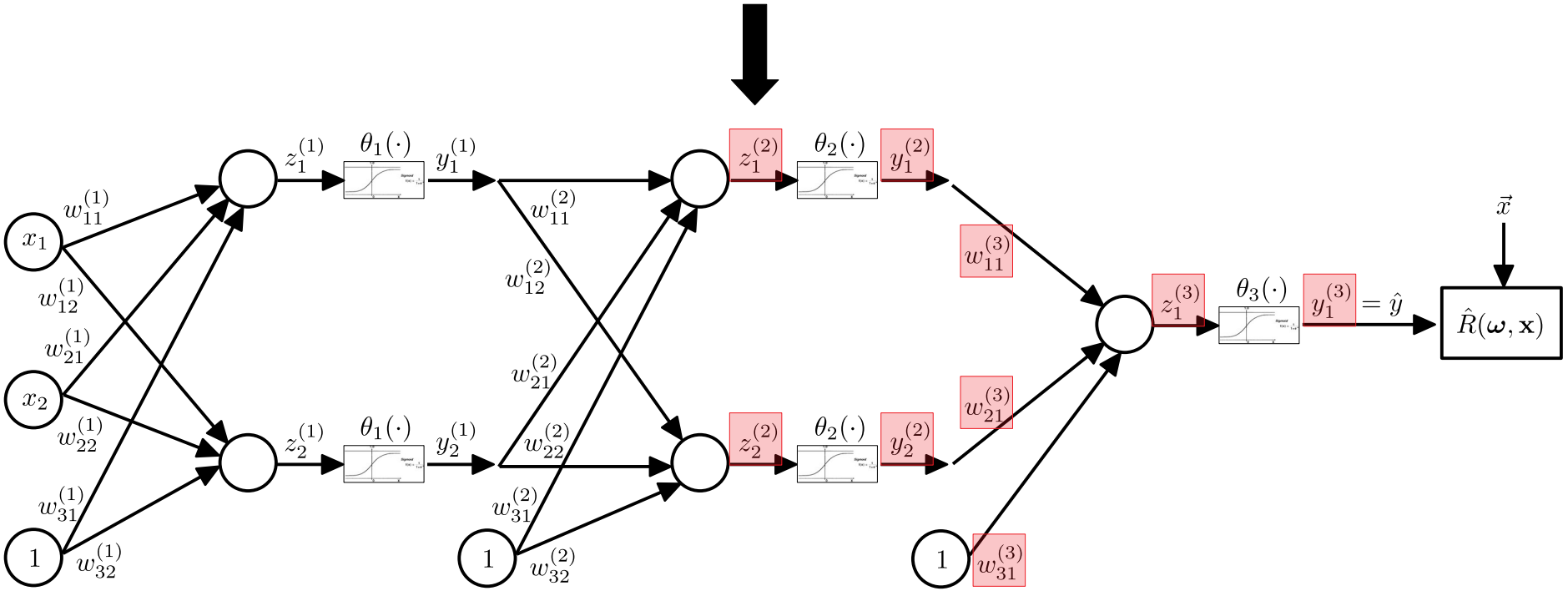


Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(2)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(2)}} \cdot \theta_2'\left(z_i^{(2)}\right)$$

Already calculated in previous step



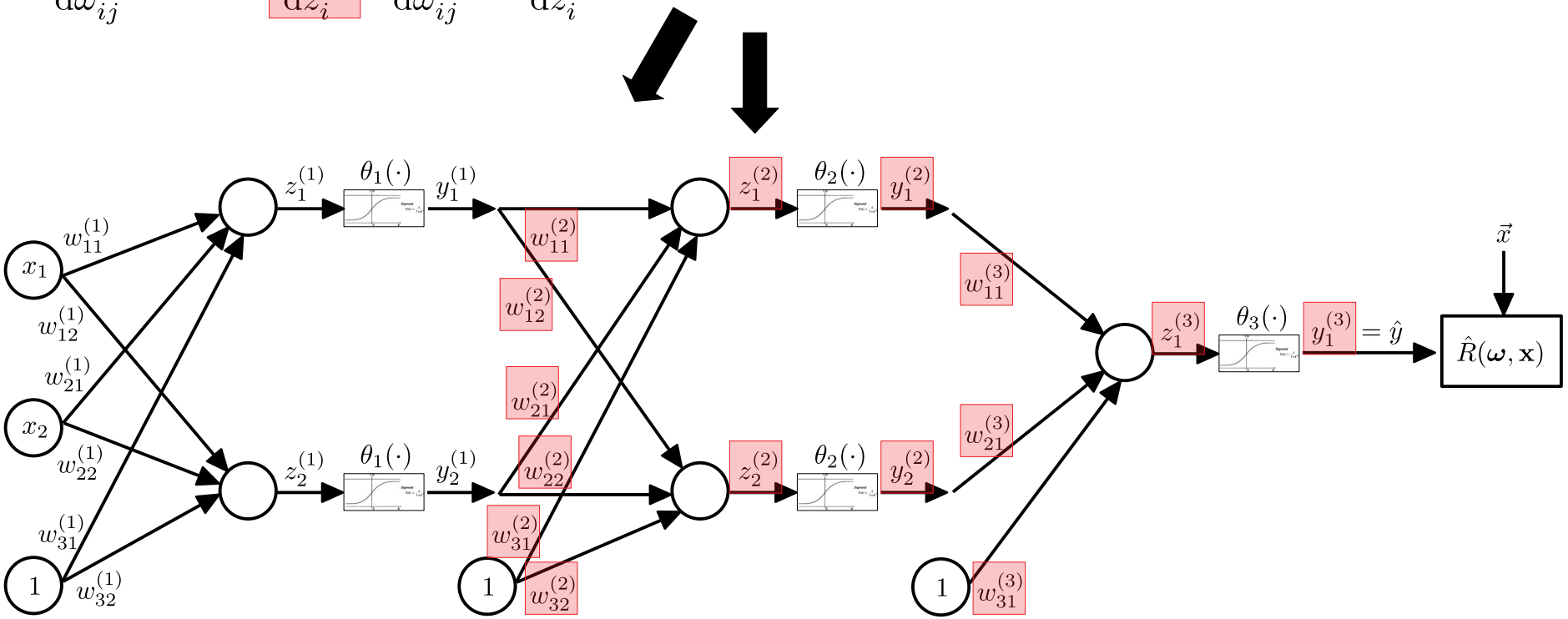
Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(2)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(2)}} \cdot \theta_2'\left(z_i^{(2)}\right)$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{ij}^{(2)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(2)}} \cdot \frac{\mathrm{d}z_i^{(2)}}{\mathrm{d}\omega_{ij}^{(2)}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(2)}} \cdot y_j^{(1)}$$

Already calculated in previous step

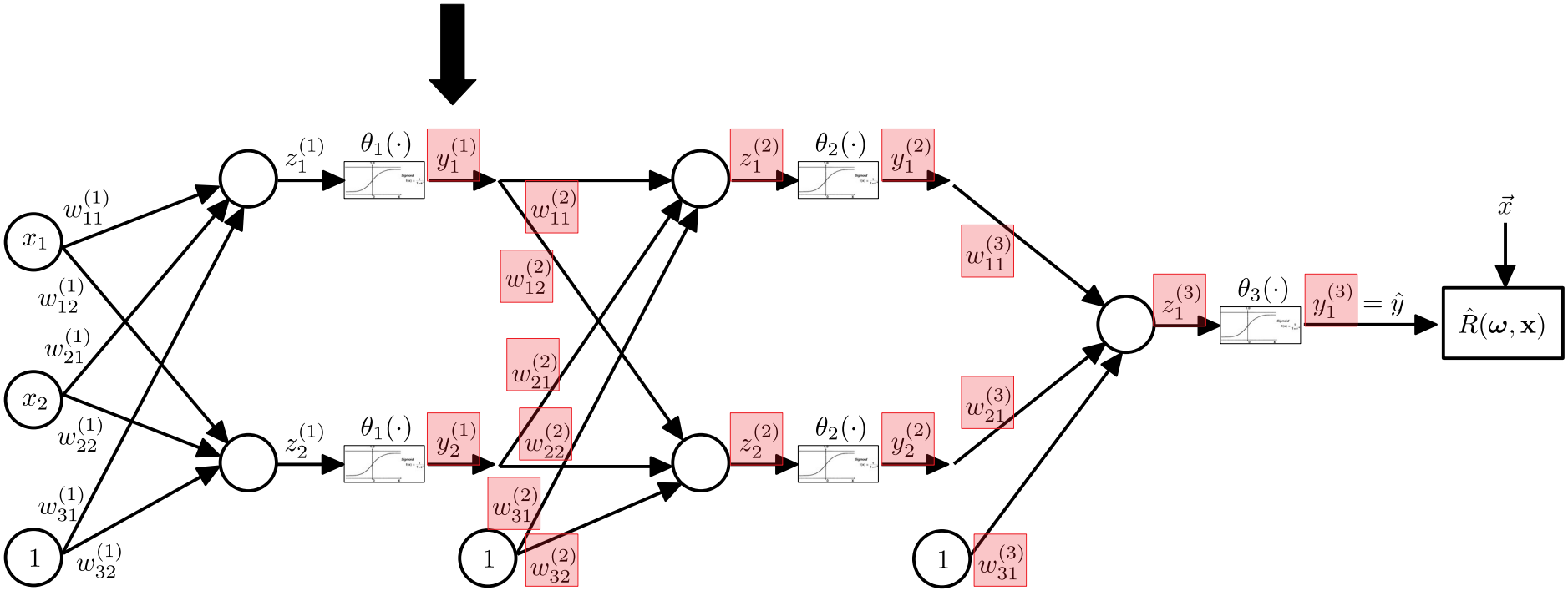


Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(1)}}(\boldsymbol{\omega}, \mathbf{x}) = \sum_{j=1}^{2} \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_j^{(2)}} \cdot \frac{\mathrm{d}z_j^{(2)}}{\mathrm{d}y_i^{(1)}} = \sum_{j=1}^{2} \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_j^{(2)}} \cdot \omega_{ij}^{(2)}$$

Already calculated in previous step

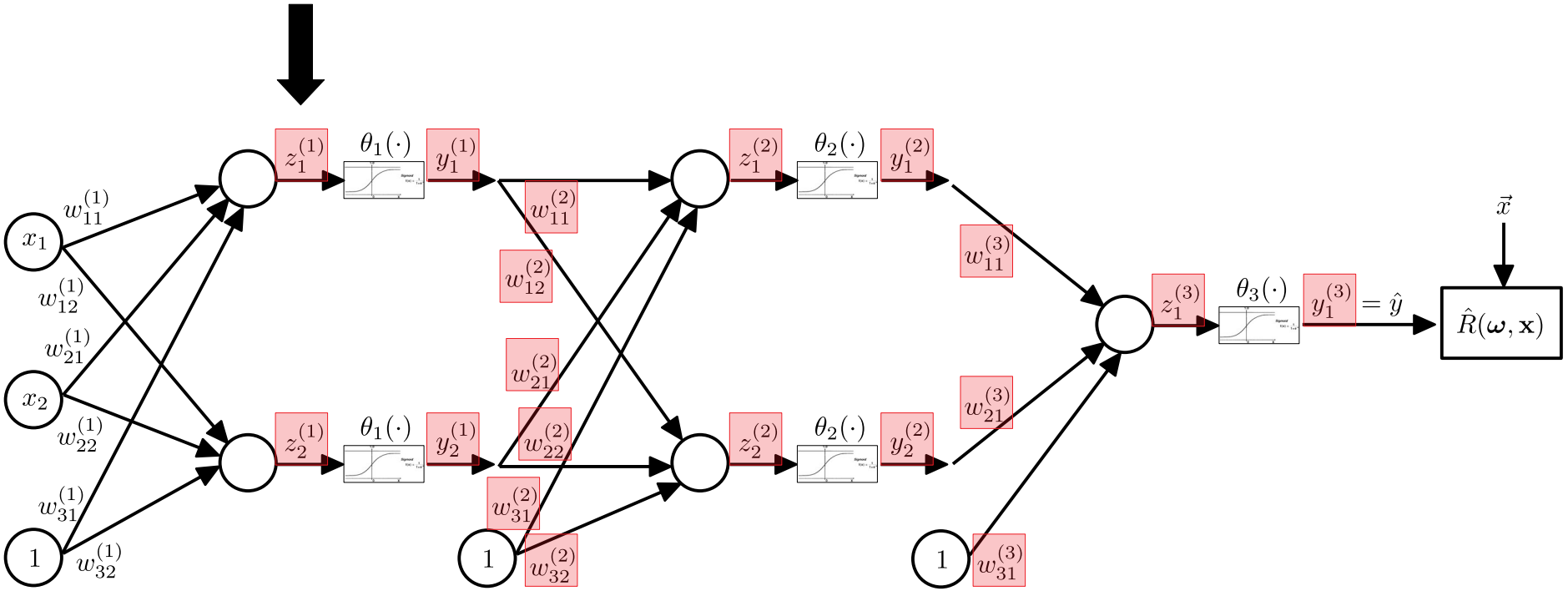


Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(1)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(1)}} \cdot \theta_1'\left(z_i^{(1)}\right)$$

Already calculated in previous step



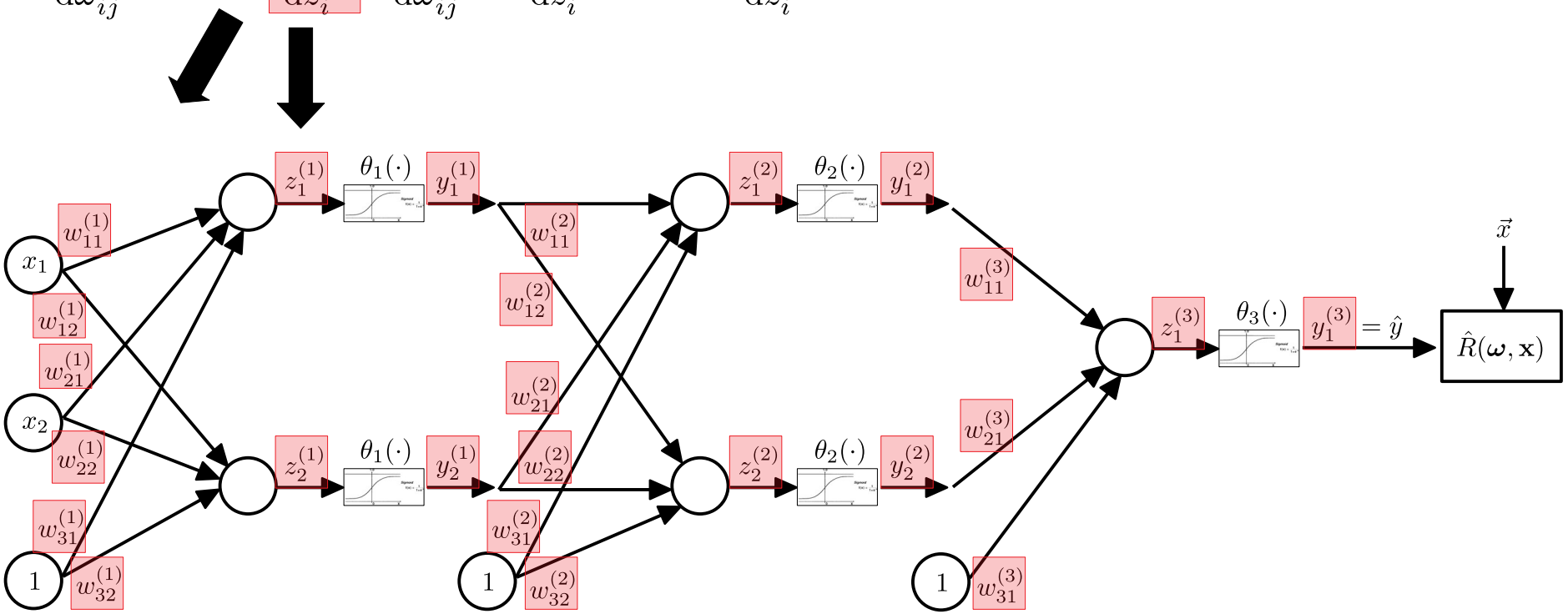
Step-2: Backward pass

- Calculate the derivatives w.r.t. the $\{\omega_{ij}^{(k)}\}$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(1)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(1)}} \cdot \theta_1'\left(z_i^{(1)}\right)$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{ij}^{(1)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(1)}} \cdot \frac{\mathrm{d}z_i^{(1)}}{\mathrm{d}\omega_{ij}^{(1)}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(1)}} \cdot y_j^{(0)} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(1)}} \cdot x_j$$

Already calculated in previous step



Backpropagation – part 2

Forward pass:

- Start with: $y_j^{(0)} = x_j$

- For each layer $k = 1 \dots N$:

$$z_i^{(k)} = \sum_j y_j^{(k-1)} \omega_{ij}^{(k)}$$

$$y_i^{(k)} = \theta_k \left(z_i^{(k)} \right)$$

Backward pass:

- Start for layer N:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}y}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y^{(N)}}$$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z^{(N)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y^{(N)}} \cdot \theta'_N \left(z^{(N)} \right)$$

- For each layer $k = (N-1) \dots 0$:

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}\omega_{ij}^{(k+1)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(k+1)}} \cdot y_j^{(k)}$$

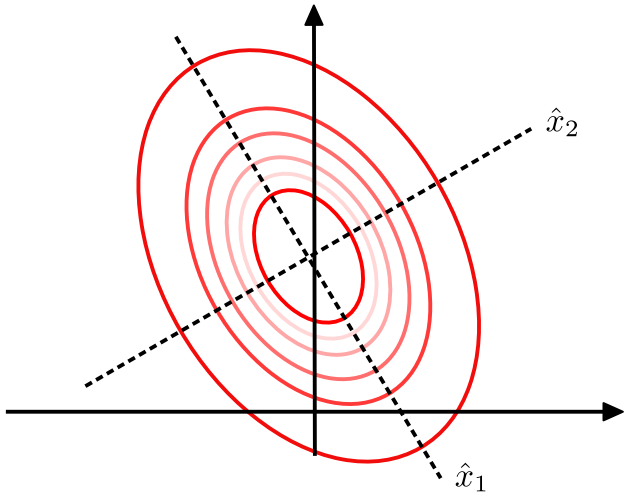
$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(k)}}(\boldsymbol{\omega}, \mathbf{x}) = \sum_{j=1}^{n} \frac{\mathrm{d}\hat{R}}{\mathrm{d}z_j^{(k+1)}} \cdot \omega_{ij}^{(k+1)}$$

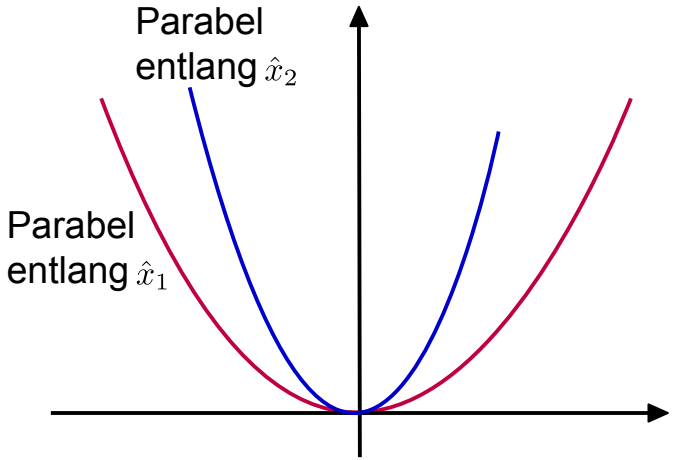
$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}z_i^{(k)}}(\boldsymbol{\omega}, \mathbf{x}) = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_i^{(k)}} \cdot \theta'_1 \left(z_i^{(k)} \right)$$

Backup

Gradientenabstieg in d Dimensionen

- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.



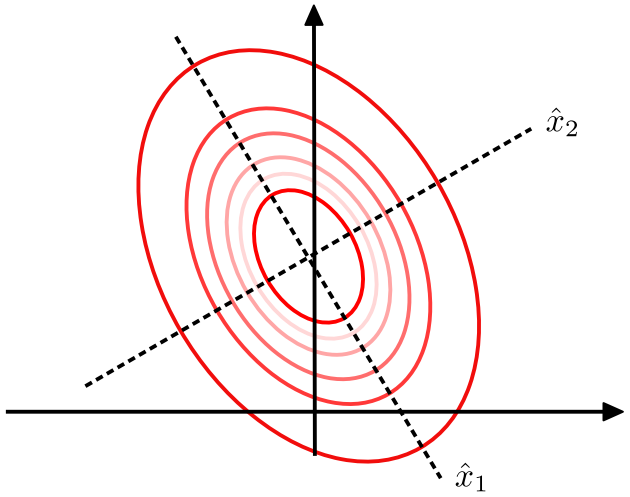


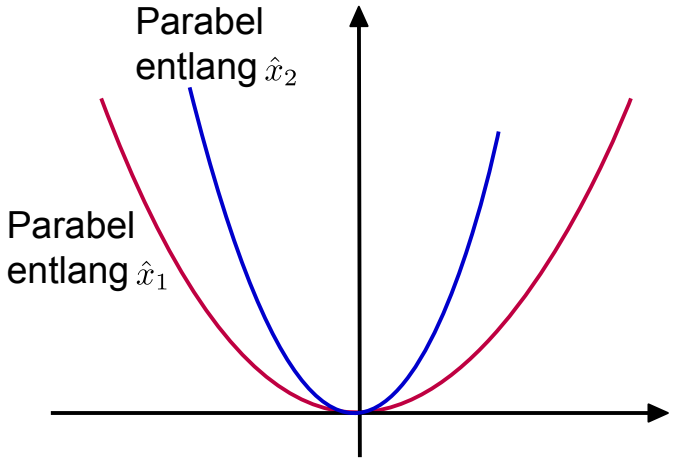
Priv. Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Gradientenabstieg in d Dimensionen

- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.

- **Problem**: Die Öffnungswinkel der Parabeln entlang der Hauptachsen – und damit verbunden die zweiten Richtungsableitungen entlang der Hauptachsen – sind i.a. unterschiedlich groß.





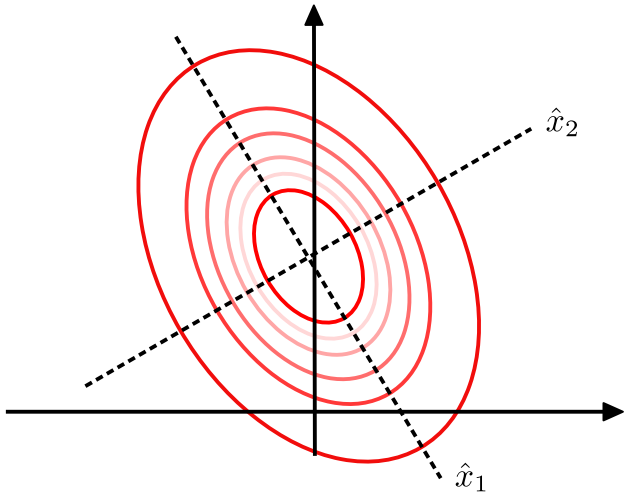
Priv. Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

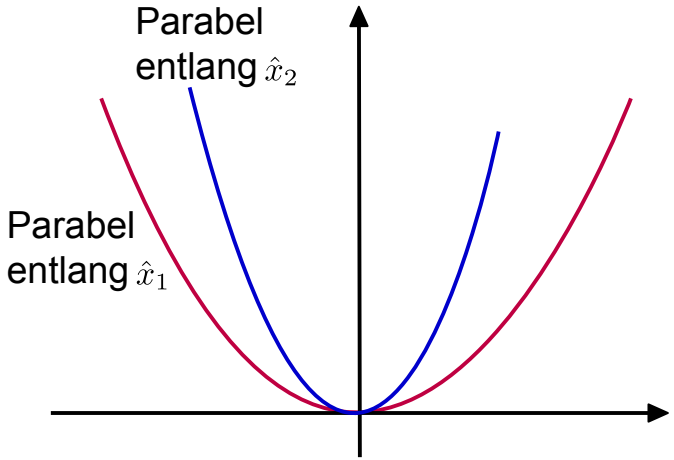
Gradientenabstieg in d Dimensionen

- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.

- **Problem**: Die Öffnungswinkel der Parabeln entlang der Hauptachsen – und damit verbunden die zweiten Richtungsableitungen entlang der Hauptachsen – sind i.a. unterschiedlich groß.

- Wie ist η zu wählen, um optimale und garantierte Konvergenz zu erreichen?





Priv. Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

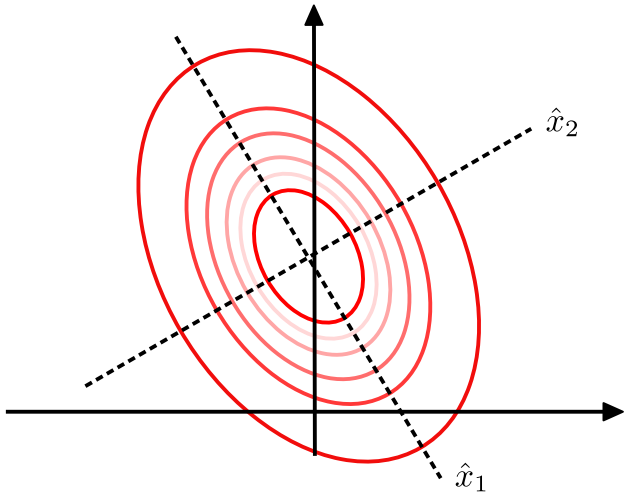
Gradientenabstieg in d Dimensionen

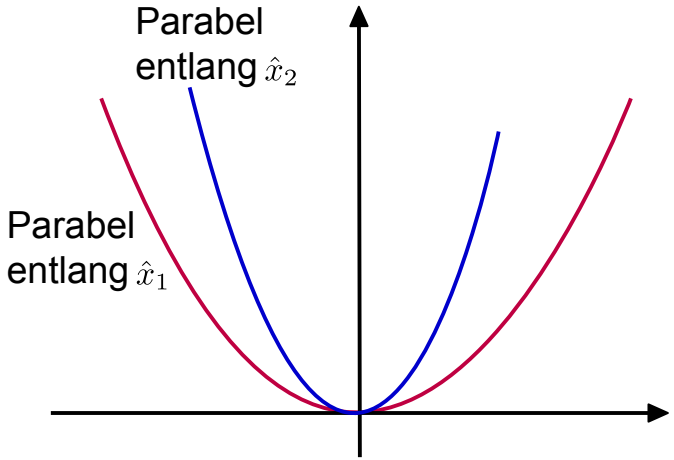
- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.

- **Problem**: Die Öffnungswinkel der Parabeln entlang der Hauptachsen – und damit verbunden die zweiten Richtungsableitungen entlang der Hauptachsen – sind i.a. unterschiedlich groß.

- Wie ist η zu wählen, um optimale und garantierte Konvergenz zu erreichen?

- Je höherdimensional der Definitionsbereich von $F(x)$ ist desto schwieriger lässt sich diese Frage im Rahmen des naiven Gradientenabstiegverfahrens beantworten.





Gradientenabstieg in d Dimensionen

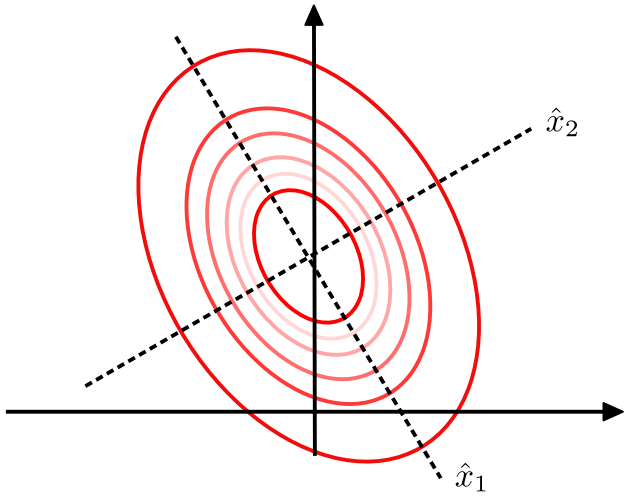
- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.

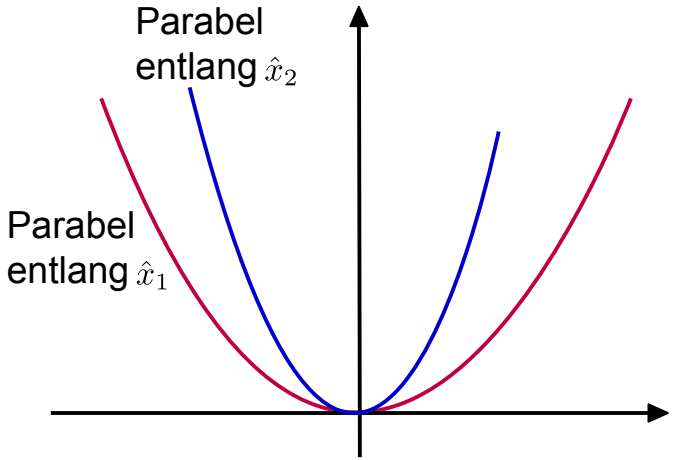
- **Problem**: Die Öffnungswinkel der Parabeln entlang der Hauptachsen – und damit verbunden die zweiten Richtungsableitungen entlang der Hauptachsen – sind i.a. unterschiedlich groß.

- Wie ist η zu wählen, um optimale und garantierte Konvergenz zu erreichen?

- Je höherdimensional der Definitionsbereich von $F(x)$ ist desto schwieriger lässt sich diese Frage im Rahmen des naiven Gradientenabstiegverfahrens beantworten.

- **NB**: Konvergenz ist umso schwieriger zu erreichen je größer die Konditionszahl der Hessematrix ist (d.h. je weiter die Eigenwerte der Hessematrix auseinander liegen).





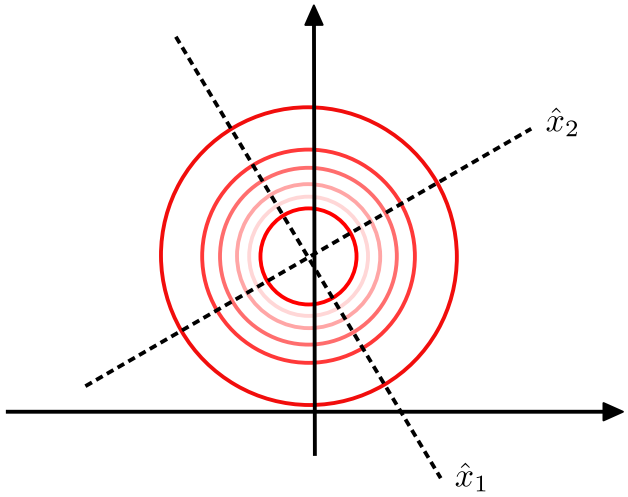
Priv. Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Gradientenabstieg in d Dimensionen

- In d Dimensionen lässt sich jede Parabel auf ihre Hauptachsen transformieren. Schnitte entlang der Hauptachsen führen wiederum auf Parabeln.

- **Problem**: Die Öffnungswinkel der Parabeln entlang der Hauptachsen – und damit verbunden die zweiten Richtungsableitungen entlang der Hauptachsen – sind i.a. unterschiedlich groß.

- Durch das Newtonverfahren wird das Problem automatisch gelöst. Die Multiplikation mit $H^{-1}(x)$ transformiert die Parabel auf eine Einheitsparabel.

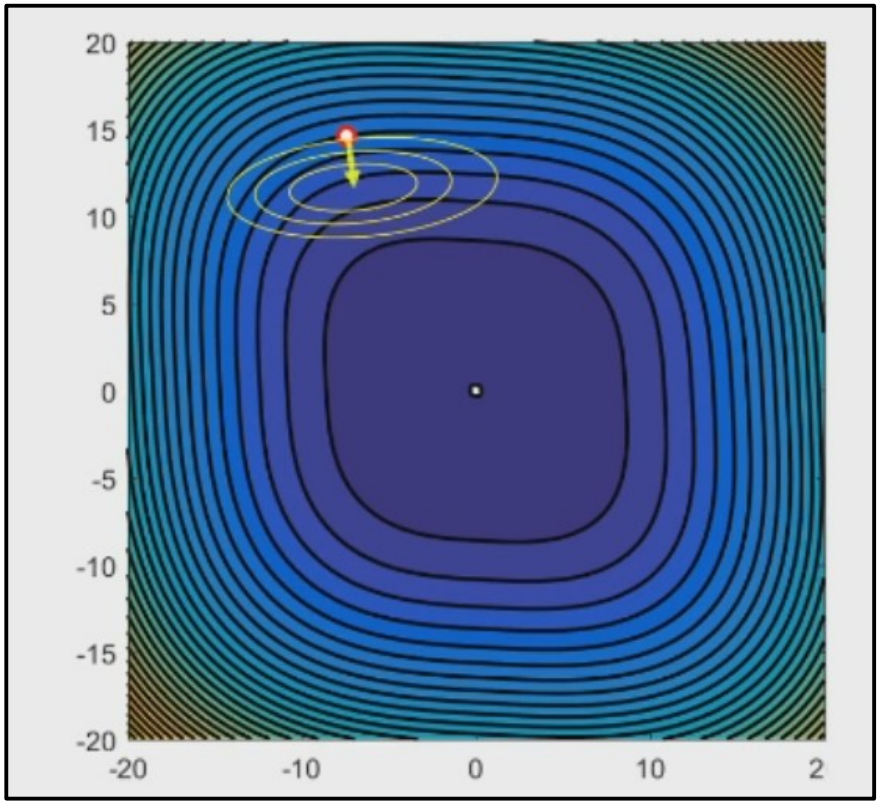


Sie können das auch durch die Aktualisierungsregel erkennen.

$$\vec{x}_{k+1} = \vec{x}_k - \underbrace{H^{-1}(\vec{x}_k)\vec{\nabla}F(x_k)}_{\text{Transformation des Gradienten mit } \eta \equiv 1}$$

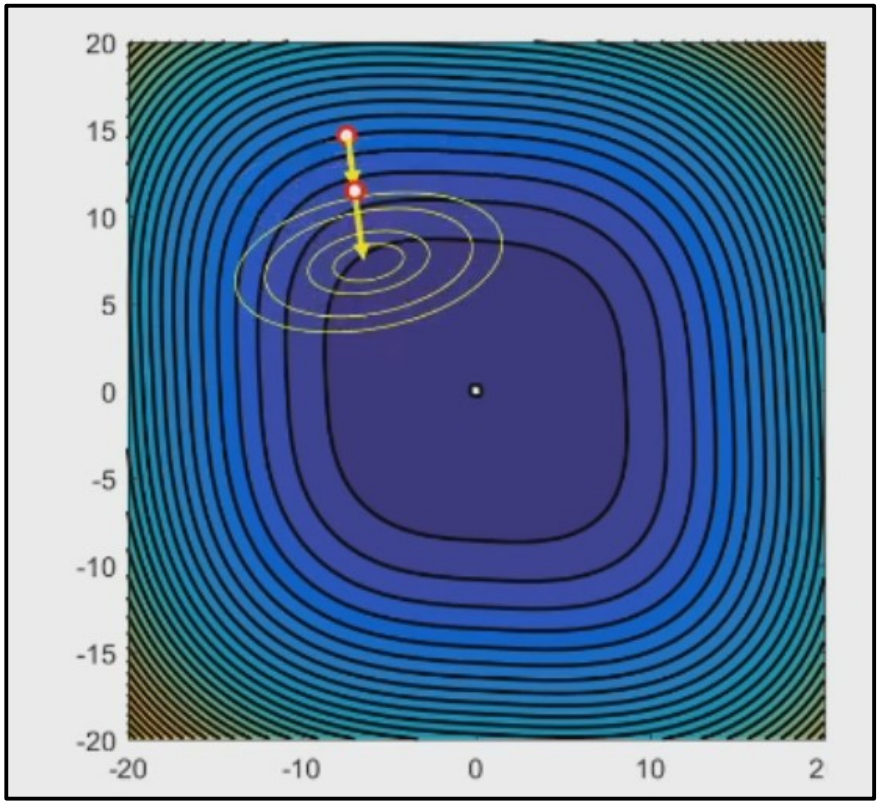
Newtonverfahren in 2 Dimensionen

- Die folgenden Folien veranschaulichen das Newtonverfahren in 2 Dimensionen:



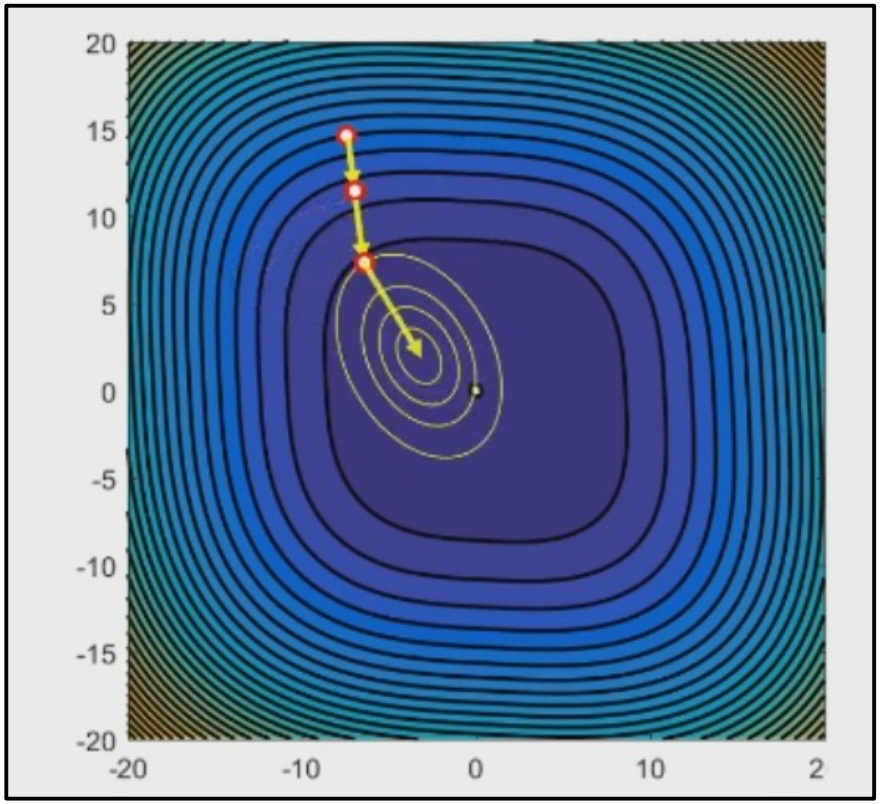
Newtonverfahren in 2 Dimensionen

- Die folgenden Folien veranschaulichen das Newtonverfahren in 2 Dimensionen:



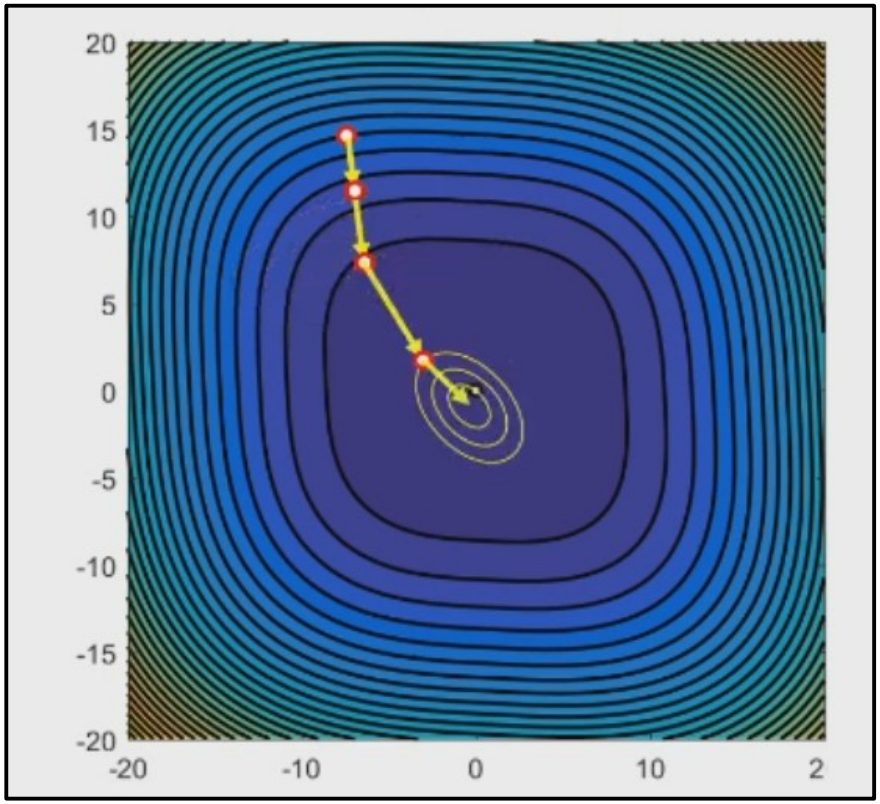
Newtonverfahren in 2 Dimensionen

- Die folgenden Folien veranschaulichen das Newtonverfahren in 2 Dimensionen:



Newtonverfahren in 2 Dimensionen

- Die folgenden Folien veranschaulichen das Newtonverfahren in 2 Dimensionen:



Newtonverfahren in 2 Dimensionen

- Die folgenden Folien veranschaulichen das Newtonverfahren in 2 Dimensionen: