UNIVERSITÀ PARTHENOPE
DiST
DIPARTIMENTO DI SCIENZE E TECNOLOGIE

Machine Learning (part II)

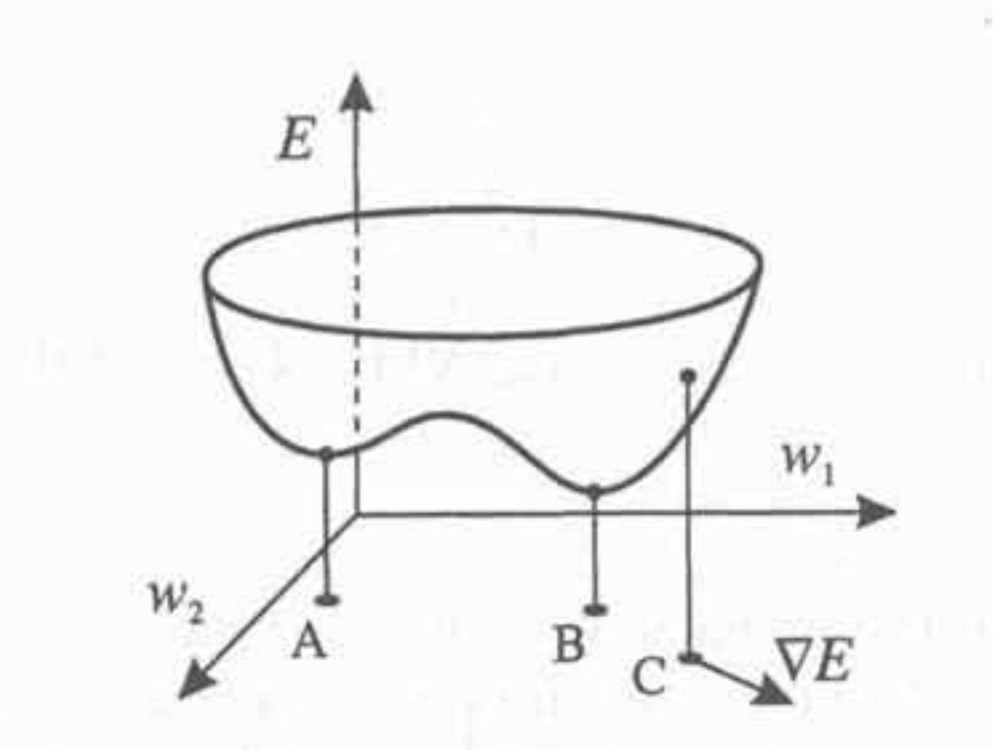
Optimization algorithms

Angelo Ciaramella

Introduction

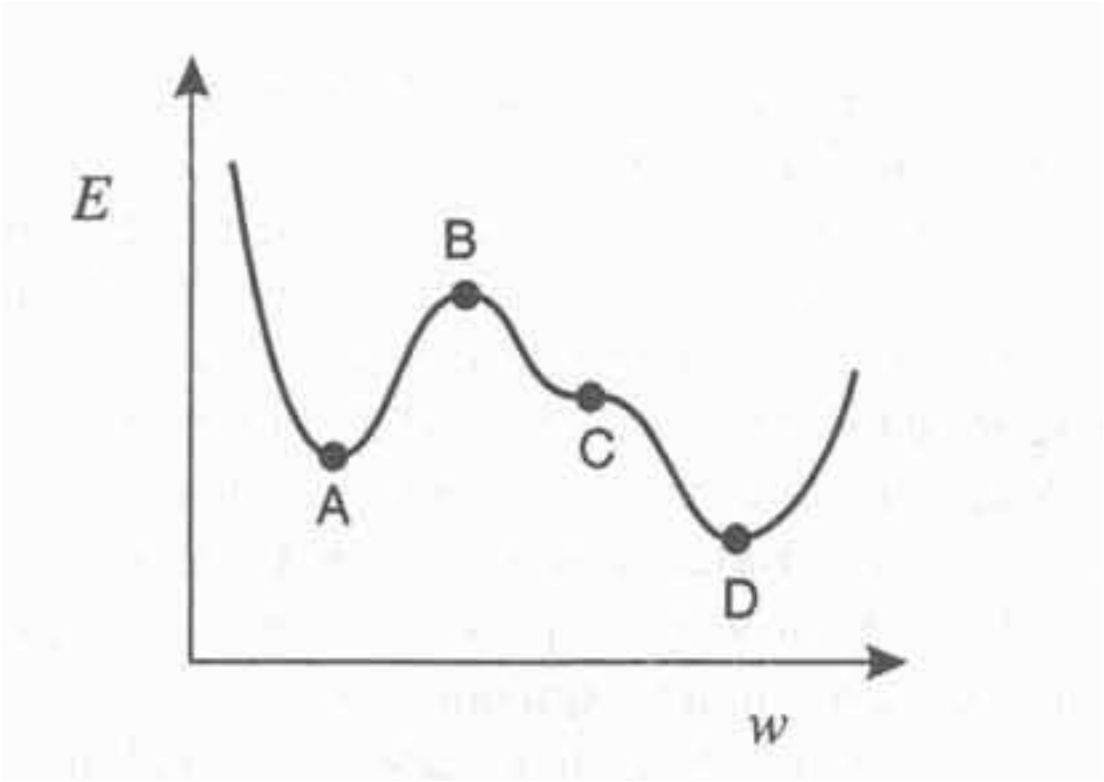
- Goal
  - find weight vector which minimizes the error function

$$E(\mathbf{w})$$



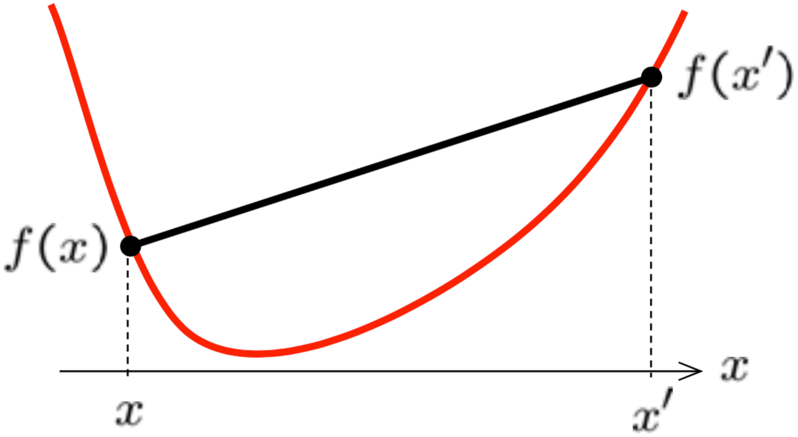
A - local minimum
B - global minimum

Error function

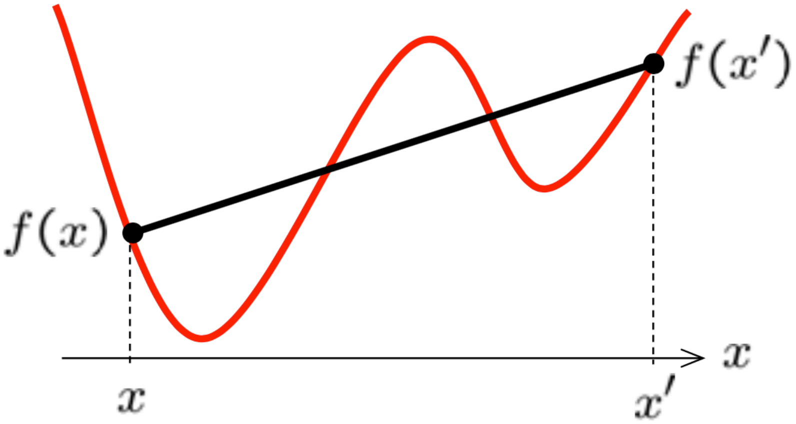


A - local minimum
B - lacal maximum
C - saddle point
D - global minimum

Error function



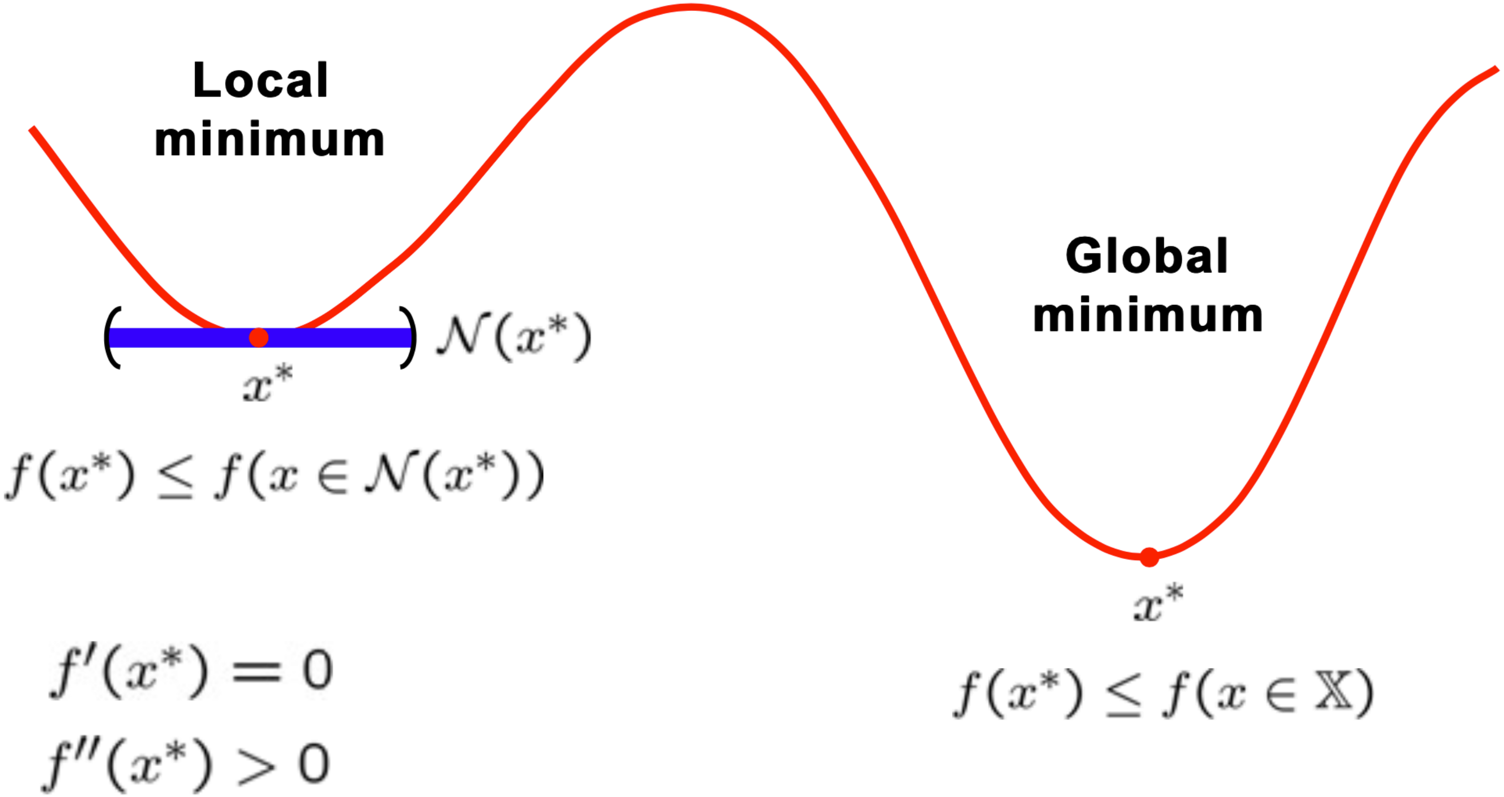
Convex



Non-convex

$$f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda) f(x')$$

Error function



$$f(x^*) \leq f(x \in \mathcal{N}(x^*))$$

$$f'(x^*) = 0$$

$$f''(x^*) > 0$$

$$f(x^*) \leq f(x \in \mathbb{X})$$

ML – Optimization algorithms

Gradient descent

- Learning approach

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau}$$

- Gradient descent

$$\Delta\mathbf{w}^{\tau} = -\eta \nabla E^{n} \Big|_{\mathbf{w}^{\tau}}$$

- Limitations
  - To choose a suitable value for the learning rate

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x^1}, \ldots, \frac{\partial f(x)}{\partial x^N} \right)$$

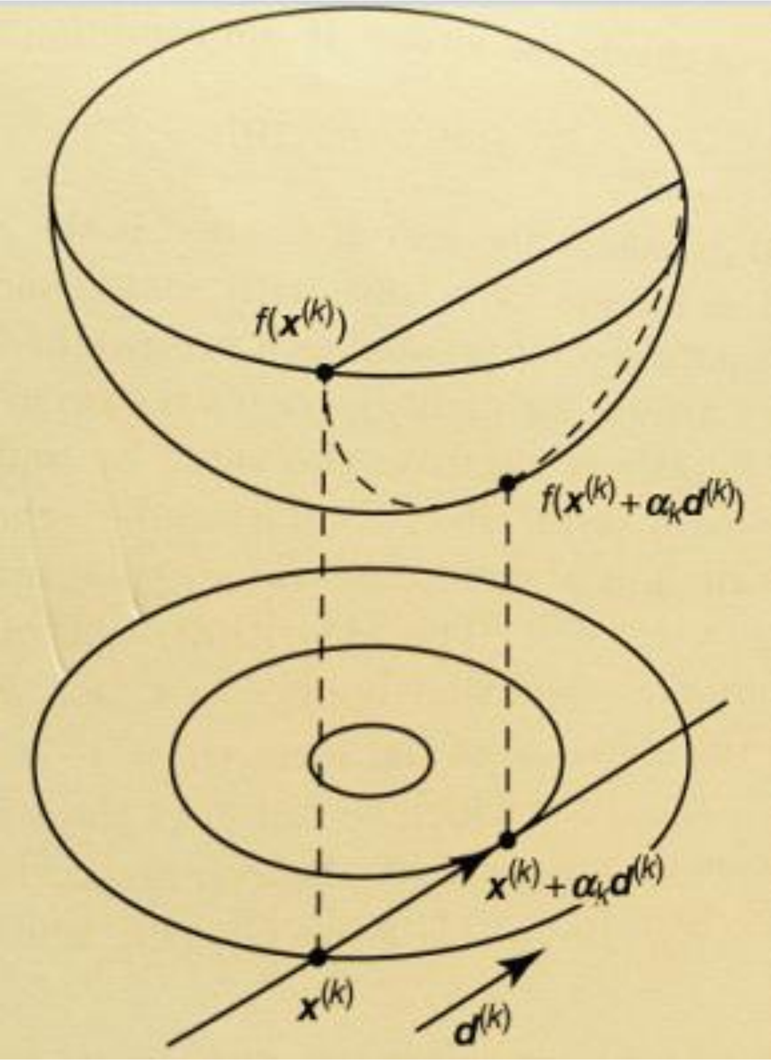
Gradient

$$\nabla^2 f(x) = \left(\frac{\partial^2 f(x)}{\partial x^i \partial x^j} \right)$$

Hessian

ML – Optimization algorithms

Minimization rule



$f(\mathbf{x}^{(k)})$
$f(\mathbf{x}^{(k)}+\alpha_k \mathbf{d}^{(k)})$
$\mathbf{x}^{(k)}+\alpha_k \mathbf{d}^{(k)}$
$\mathbf{x}^{(k)}$
$\mathbf{d}^{(k)}$

$$\min_{x} f(x), \qquad (f: \mathbb{R}^n \to \mathbb{R})$$

General form of the iteration

$$x_{k+1} = x_k + \alpha_k d_k$$

$$f(x^{(k)} + \alpha^{(k)} d^{(k)}) < f(x^{(k)})$$

Optimality conditions - garantees

$\nabla f(x^*) = 0$ Minimum

$x^\top \nabla^2 f(x^*) x > 0$ Convex

Positive definite

Taylor expansion

- Taylor expansion *f* (one variable), m times continuously differentiable

$$f:\mathbb{R}\to\mathbb{R}$$

$$h = b - a$$

$$f(b) = f(a) + \frac{h}{1!}f^{(1)}(a) + \frac{h^2}{2!}f^{(2)}(a) + \cdots + \frac{h^m}{m!}f^{(m)}(a) + o(h^m).$$

Point *a* around wich we will Taylor expand

- Multi-variable

$$f:\mathbb{R}^n \to \mathbb{R}$$

Second order

$$f(x) = f(a) + \nabla f(a)^T(x-a) + \frac{1}{2}(x-a)^T\nabla^2 f(a)(x-a) + o(\|x-a\|^2)$$

Local quadratic approximation

- Taylor exapansion around some point in weight space

$$E(\mathbf{w}) = E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^T\mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T\mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

- where

$$\mathbf{b} \equiv \nabla E\,\Big|_{\hat{\mathbf{w}}} \qquad (\mathbf{H})_{\boldsymbol{ij}} \equiv \frac{\partial^2 E}{\partial w_i \partial w_j}\Big|_{\hat{\mathbf{w}}}$$

Hessian matrix

- Local approximation of the gradient

$$\nabla E = \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

Local quadratic approximation

- Local quadratic approximation around a point

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- where

$$\nabla E = 0$$

- eigenvalues and eigenvectors of Hessian matrix

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \qquad \qquad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$$

Local quadratic approximation

- from expansion

$$(\mathbf{w}-\mathbf{w}^*) = \sum_i \alpha_i \mathbf{u}_i$$

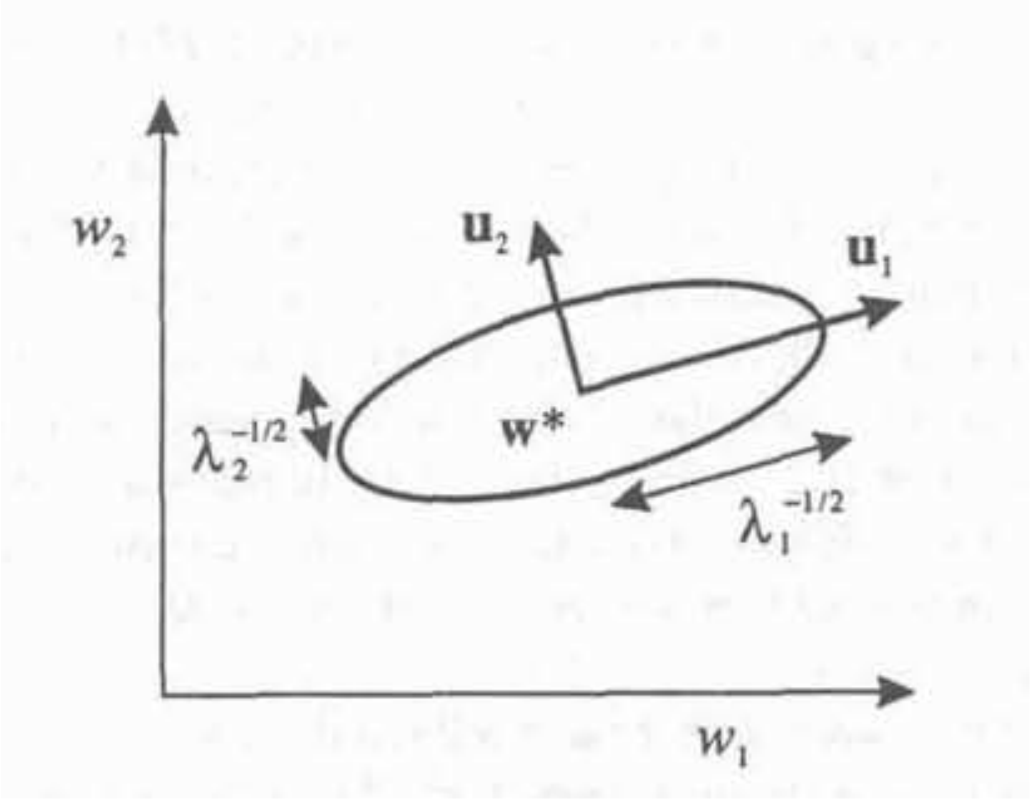
- where

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

- Moreover, Hessian matrix is positive definite when

$$\mathbf{v}^T\mathbf{H}\,\mathbf{v} > 0\ \forall\ \mathbf{v} = \sum_{\boldsymbol{i}} \beta_i \mathbf{u}_i \qquad \mathbf{v}^T\mathbf{H}\,\mathbf{v} = \sum_i \beta_i^2 \lambda_i$$

Local quadratic approximation



- Result
  - Countours of costant error are ellipses whose axes are aligned with the eigenvetors of the Hessian matrix, with length inversely proportional to the square roots of the corresponding eigenvectors

Gradient descent

- Learning approach

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau}$$

- Gradient descent

$$\Delta\mathbf{w}^{\tau} = -\eta\nabla E^{n}\Big|_{\mathbf{w}^{\tau}}$$

- Limitations
  - To choose a suitable value for the learning rate

Convergence

- Gradient approximation

$$\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i$$

- moreover

$$\Delta \mathbf{w} \quad = \sum_i \Delta\alpha_i \mathbf{u}_i$$

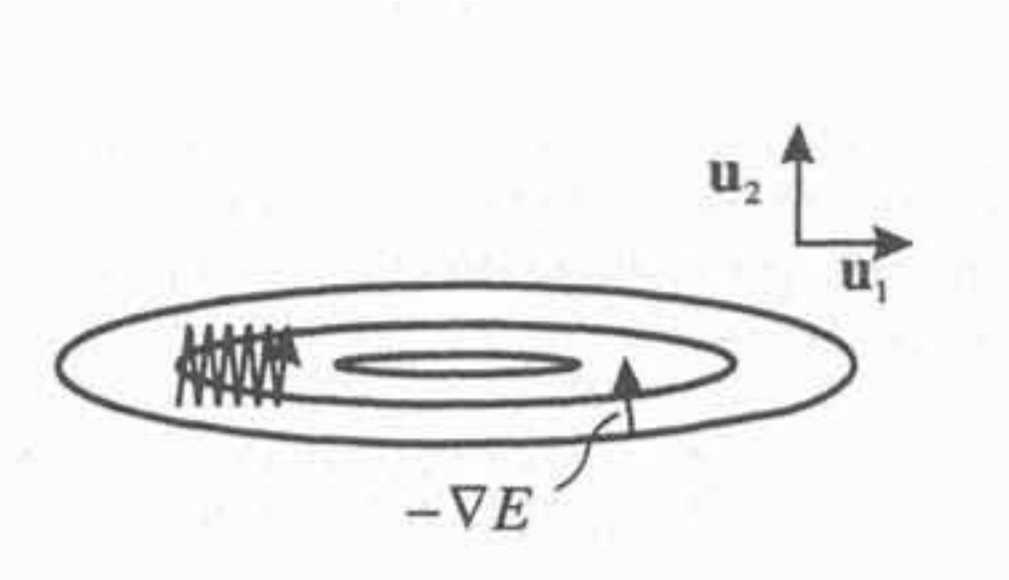
- Learning algorithm

$$\Delta\alpha_i = -\eta\alpha_i\lambda_i$$

$$\alpha_i^{new} = (1 - \eta\lambda_i)\alpha_i^{old}$$

ML – Optimization algorithms

Convergence



- Distance to the minimum and linear convergence

$$\mathbf{u}_i^T(\mathbf{w} - \mathbf{w}^*) = \alpha_i$$

- After T step

$$\alpha_i^{(T)} = (1 - \eta\lambda_i)^T \alpha_i^{(0)} \qquad |1 - \eta\lambda_i| < 1$$

Gradient descent

- We prove that

$$\alpha_i^{(T} = (1-\eta\lambda_i)^T\alpha_i^{(0)}$$

- Convergence for

$$|1-\eta\lambda_i| < 1$$

- Convergence along the direction

$$\left(1-\frac{2\lambda_{min}}{\lambda_{max}}\right)$$

ML – Optimization algorithms

Momentum

- Modified gradient formula

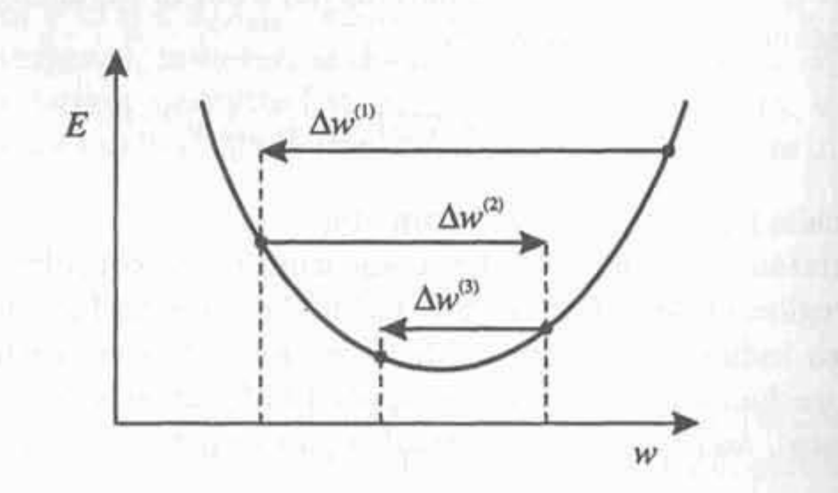
$$\Delta \mathbf{w}^{\tau} = -\eta \nabla E^{n}\big|_{\mathbf{w}^{\tau}} + \mu\, \Delta \mathbf{w}^{\tau-1}$$

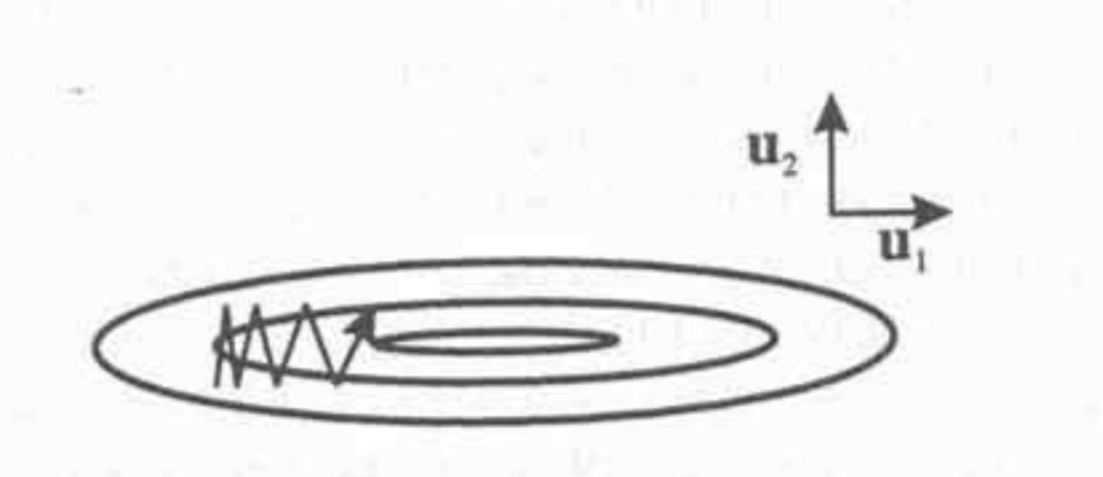
- Iteratively

$$\Delta \mathbf{w} \quad = -\eta \nabla E\{1 + \mu + \mu^{2} + \cdots\} = -\frac{\eta}{1-\mu} \nabla E$$

  - Increse the effective learning rate

Momentum





Optimization for deep models

- Optimization function

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

- Emprical risk minimization

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

Optimization for deep models

- Maximum likelihood estimation

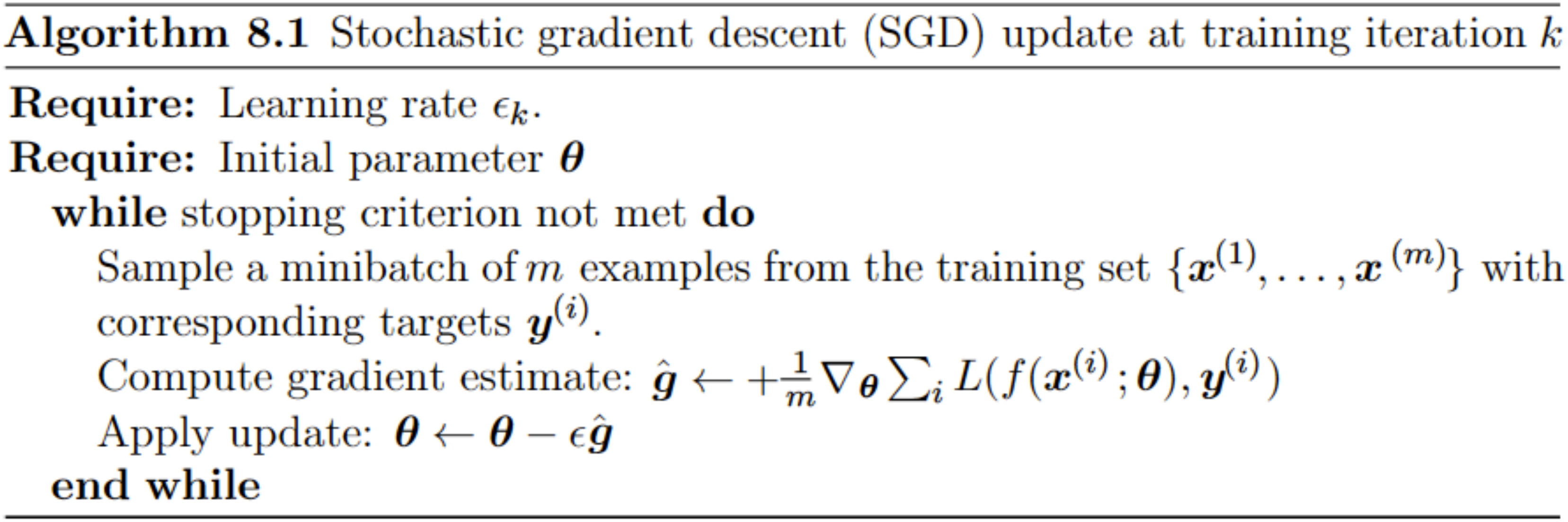
$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta})$$

- Gradient of the loss function

$$\hat{\boldsymbol{g}} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

Stochastic Gradient Descent (SGD)



Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

```
Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while
```

- Sufficient condition for convergence

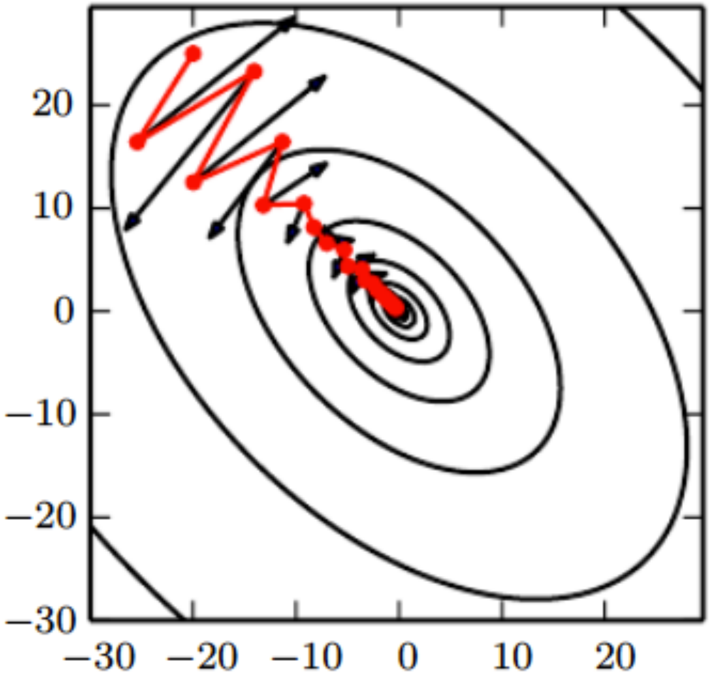
$$\sum_{k=1}^{\infty} \epsilon_k = \infty \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

SGD and momentum

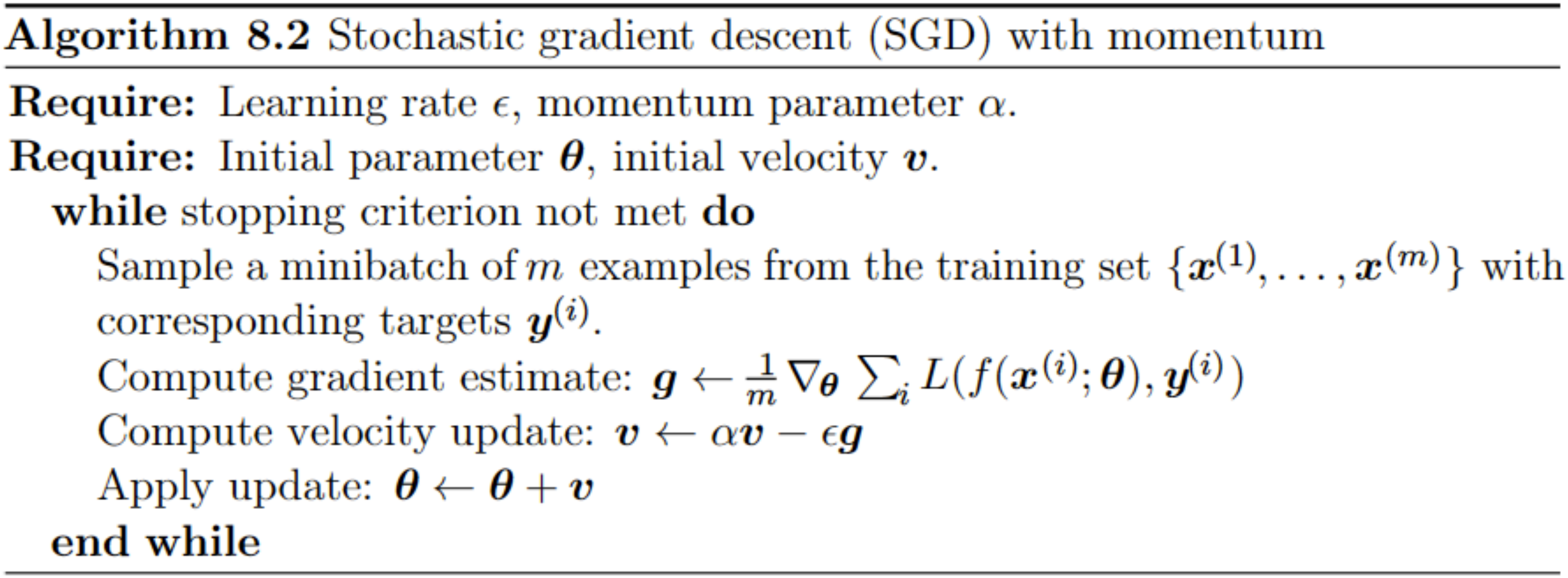
- Update rule

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$



SGD and momentum



Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

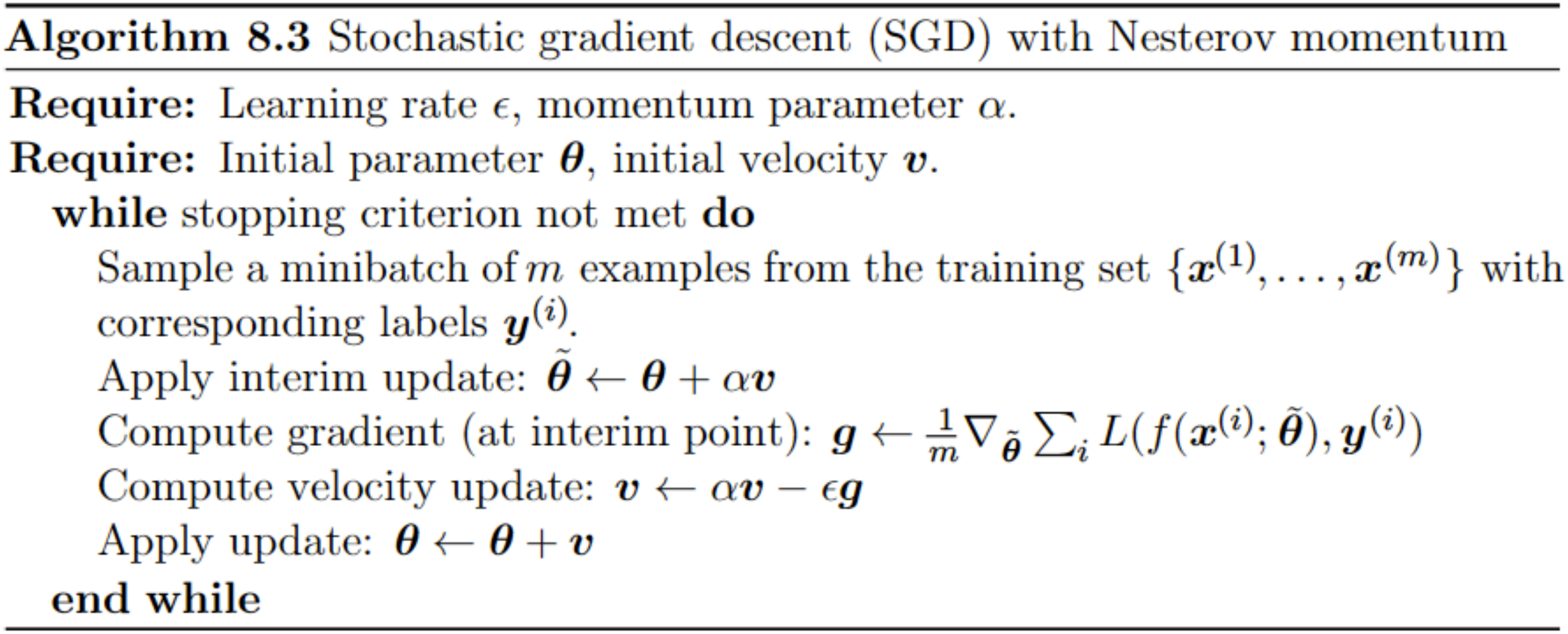
Nesterov Momentum

- Nesterov's accelerated gradient method (Sutskever et al., 2013)

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

Nesterov Momentum



Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

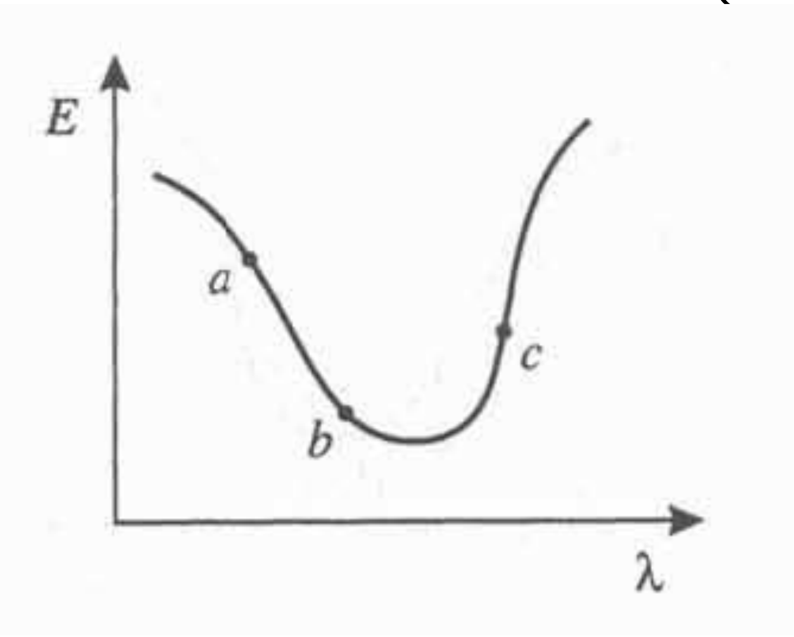
Line search

- Linear search direction in weight space

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \lambda^{(\tau)}\mathbf{d}^{(\tau)}$$

- Where the parameter is chosen to minimize

$$E(\lambda) = E\left(\mathbf{w}^{(\tau)} + \lambda\mathbf{d}^{(\tau)}\right)$$



Error function

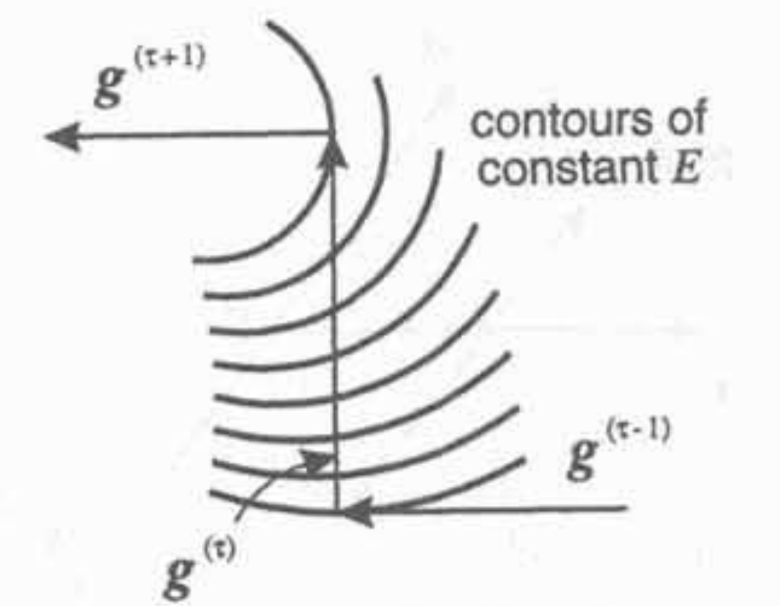
Conjugate gradient

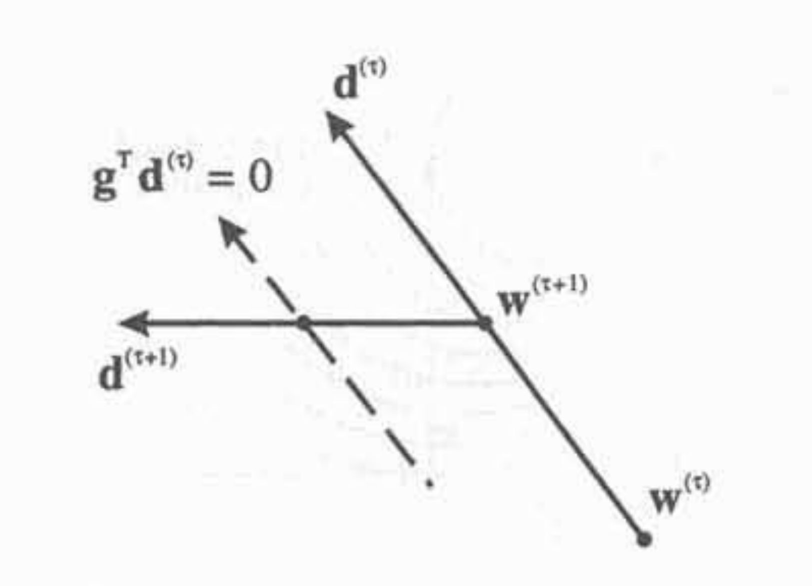
- At the minimum of the line search

$$\frac{\partial}{\partial\lambda} E(\mathbf{w}^{\tau} + \lambda \ \ \mathbf{d}^{(\tau)}) = 0$$

- which gives

$$\mathbf{g}^{(\tau+1)T}\mathbf{d}^{(\tau)} = 0 \qquad\qquad \mathbf{g} \equiv \nabla E$$





On a line minimization the new gradient is orthogonal to the line-search direction

Conjugate gradient algorithm

- Initialization of the weight vector

$$\mathbf{w}_1$$

- Evaluate

$$\mathbf{g}_1 \quad \rightarrow \mathbf{d}_1 = -\mathbf{g}_1$$

- At step *j*, minimize $E(\mathbf{w}_j + \alpha \mathbf{d}_j)$ with

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_{min} \mathbf{d}_j$$

$$\alpha_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j}$$

- Stopping criterion

Conjugate gradient

- Evaluate the successive gradient e search direction

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d_j}$$

- Hestness-Stiefel

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{d}_j^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}$$

- Polak-Ribiere

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T\mathbf{g}_j}$$

- Fletcher-Reeves

$$\beta_j = \frac{\mathbf{g}_{j+1}^T\mathbf{g}_{j+1}}{\mathbf{g}_j^T\mathbf{g}_j}$$

ML – Optimization algorithms

Scaled conjugate gradient

- Denominator can be negative increasing the error

$$\alpha_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j}$$

- Ensure that the Hessian matrix is positive defined

$$\mathbf{H} + \lambda \mathbf{I}$$

- Step length is defined as

$$\alpha_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j + \lambda_j \|\mathbf{d}\|^2}$$

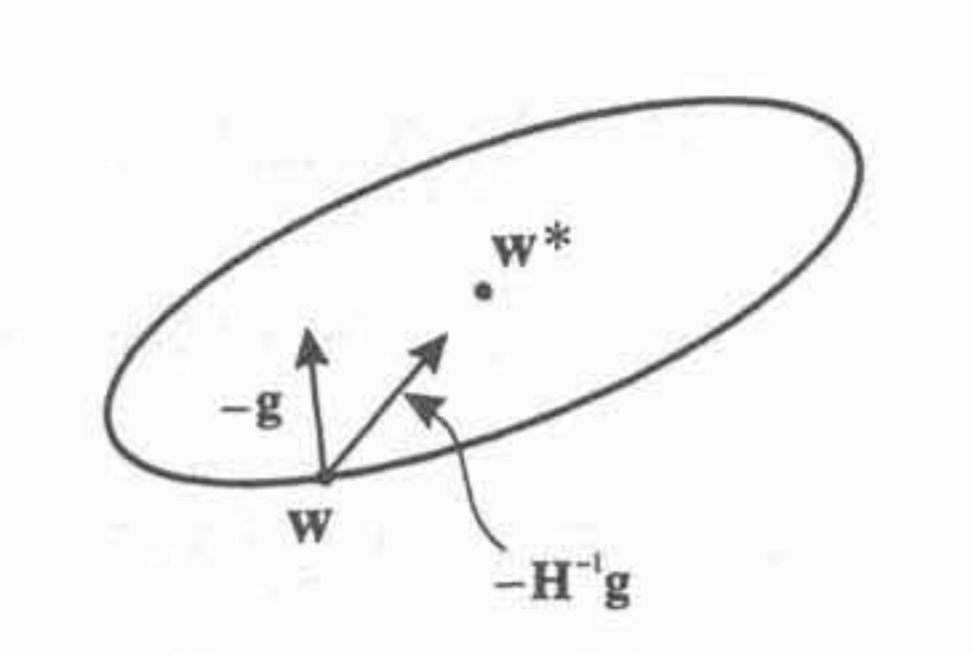
Newton's method

- Explicit use of the Hessian matrix

$$\mathbf{g} = \nabla E = \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- The minimum of the error function satisfies

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1}\mathbf{g}$$ Newton direction



Newton's method

- Second-order gradient method

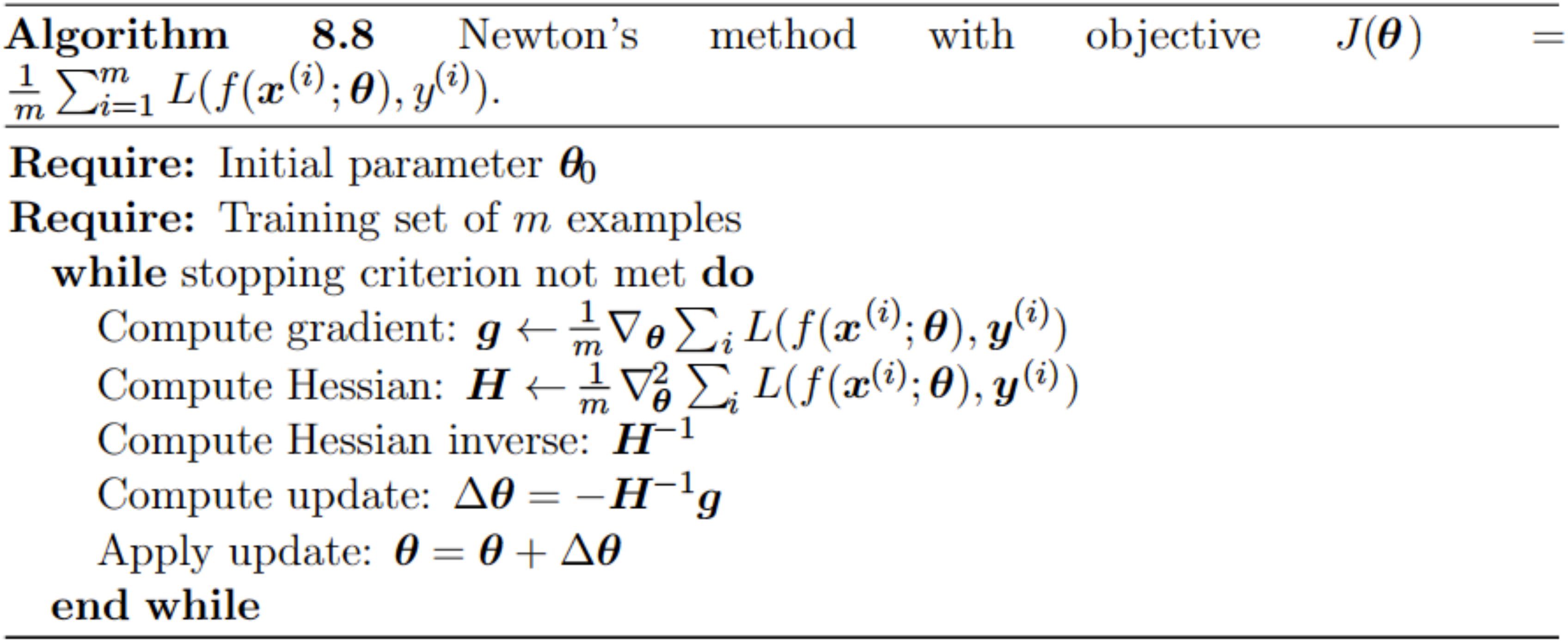
- Second order Taylor series expansion

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Newton parameter update rule

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

Newton's method



Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

```
Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of $m$ examples
  while stopping criterion not met do
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Approximation of Newton's method

- If the eigenvalues of the Hessian matrix are not all positive, we move in the wrong direction

- Regularization

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \left[H\left(f(\boldsymbol{\theta}_0)\right) + \alpha \boldsymbol{I}\right]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0)$$

- Levenberg-Marquardt algorithm

Conjugate gradients

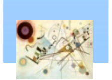
- Find a search direction that is conjugate to the previous line search direction

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

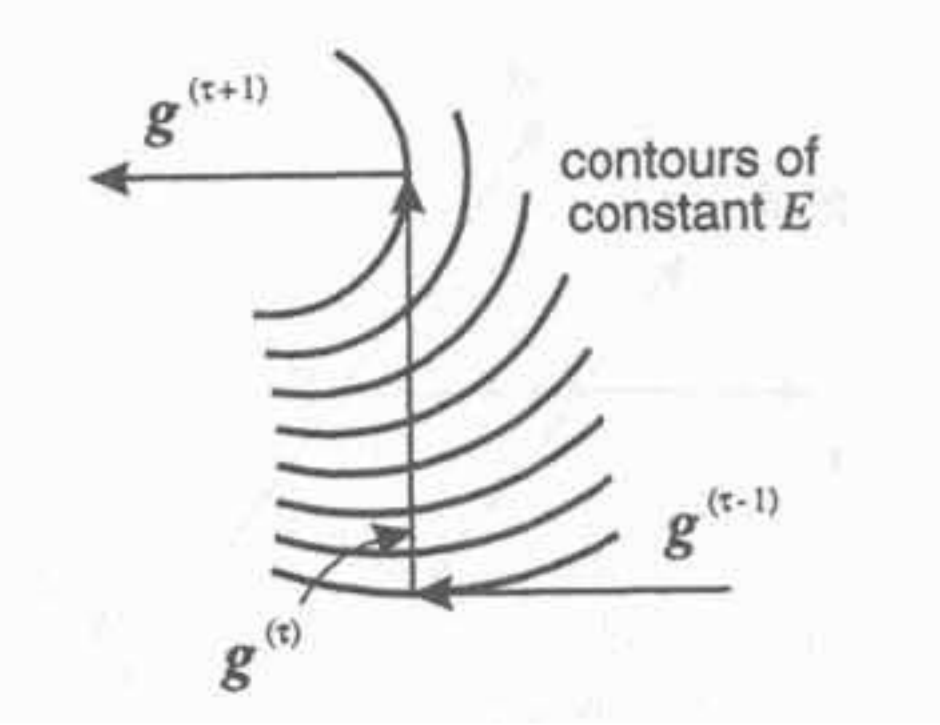
- Conjugate direction

$$\boldsymbol{d}_t^\top \boldsymbol{H} \boldsymbol{d}_{t-1} = 0$$

- Calculation of the eigenvectors of *H* to choose β_t



Conjugate gradients



Conjugate gradients

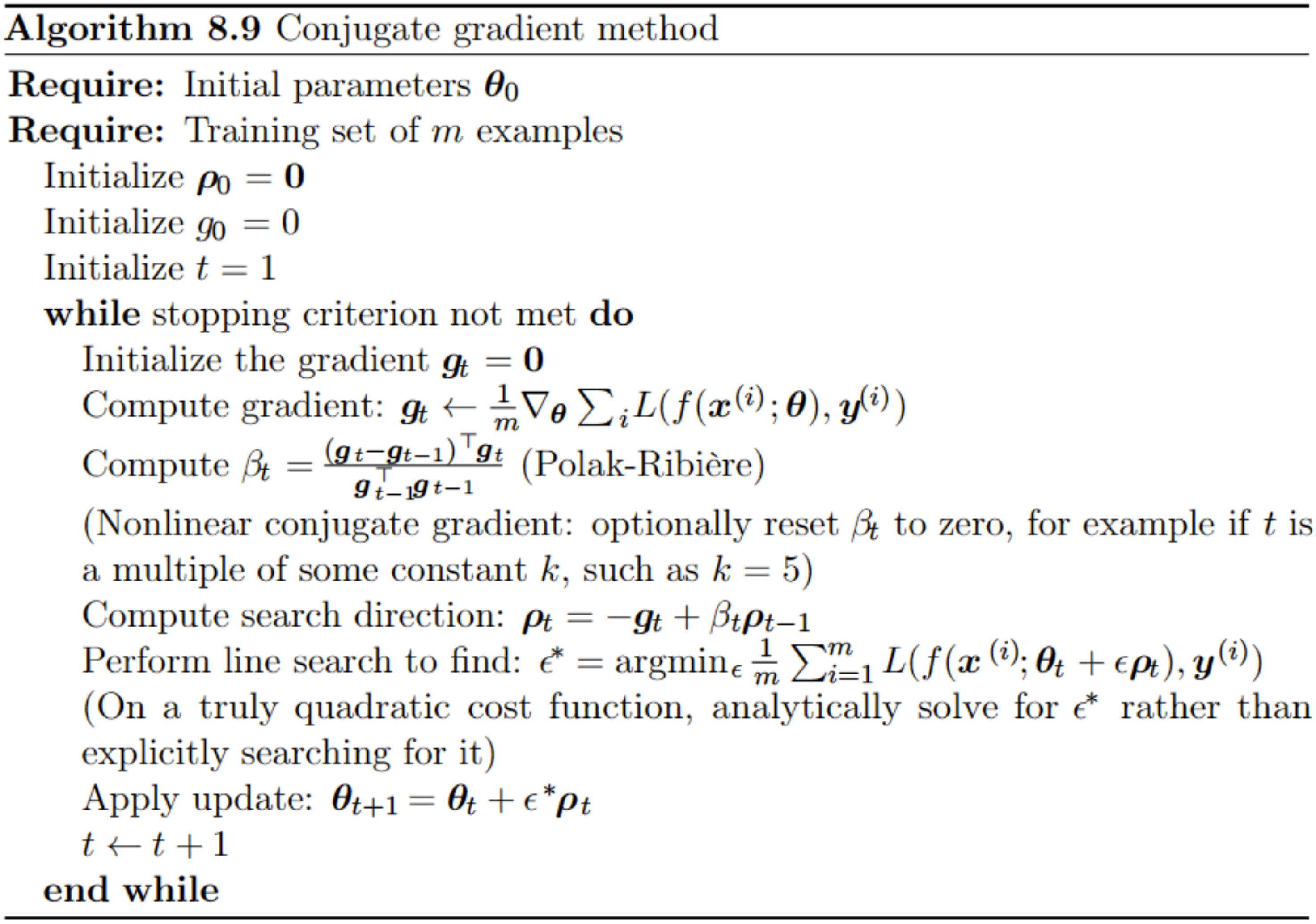
- Fletcher-Reeves

$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

- Polak-Ribiere

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

Conjugate gradients



Algorithm 8.9 Conjugate gradient method

```
Require: Initial parameters $\boldsymbol{\theta}_0$
Require: Training set of $m$ examples
  Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$
  Initialize $g_0 = 0$
  Initialize $t = 1$
  while stopping criterion not met do
    Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$
    Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)
    (Nonlinear conjugate gradient: optionally reset $\beta_t$ to zero, for example if $t$ is a multiple of some constant $k$, such as $k = 5$)
    Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
    Perform line search to find: $\epsilon^* = \text{argmin}_{\epsilon} \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$
    (On a truly quadratic cost function, analytically solve for $\epsilon^*$ rather than explicitly searching for it)
    Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$
    $t \leftarrow t + 1$
  end while
```

Quasi-Newton methods

- Broyden-Fletcher-Goldfarb-Shanno algorithm

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- Approximation of the Hessian matrix

$$\boldsymbol{M}_t$$

- Descent direction

$$\boldsymbol{\rho}_t = \boldsymbol{M}_t \boldsymbol{g}_t.$$

- Parameter update

Quasi-Newton method

- Newton’s method computationally prohibitive for evalutaing the Hessian matrix
- Sequence of matrices representing increasingly approximations to the inverse Hessian matrix
- Broyden-Fletcher-Godfarb-Shanno procedure

$$\mathbf{G}^{(\tau+1)} = \mathbf{G}^{(\tau)} + \frac{\mathbf{p}\mathbf{p}^{\mathrm{T}}}{\mathbf{p}^{\mathrm{T}}\mathbf{v}} - \frac{(\mathbf{G}^{(\tau)}\mathbf{v})\mathbf{v}^{\mathrm{T}}\mathbf{G}^{(\tau)}}{\mathbf{v}^{\mathrm{T}}\mathbf{G}^{(\tau)}\mathbf{v}} + (\mathbf{v}^{\mathrm{T}}\mathbf{G}^{(\tau)}\mathbf{v})\mathbf{u}\mathbf{u}^{\mathrm{T}}$$

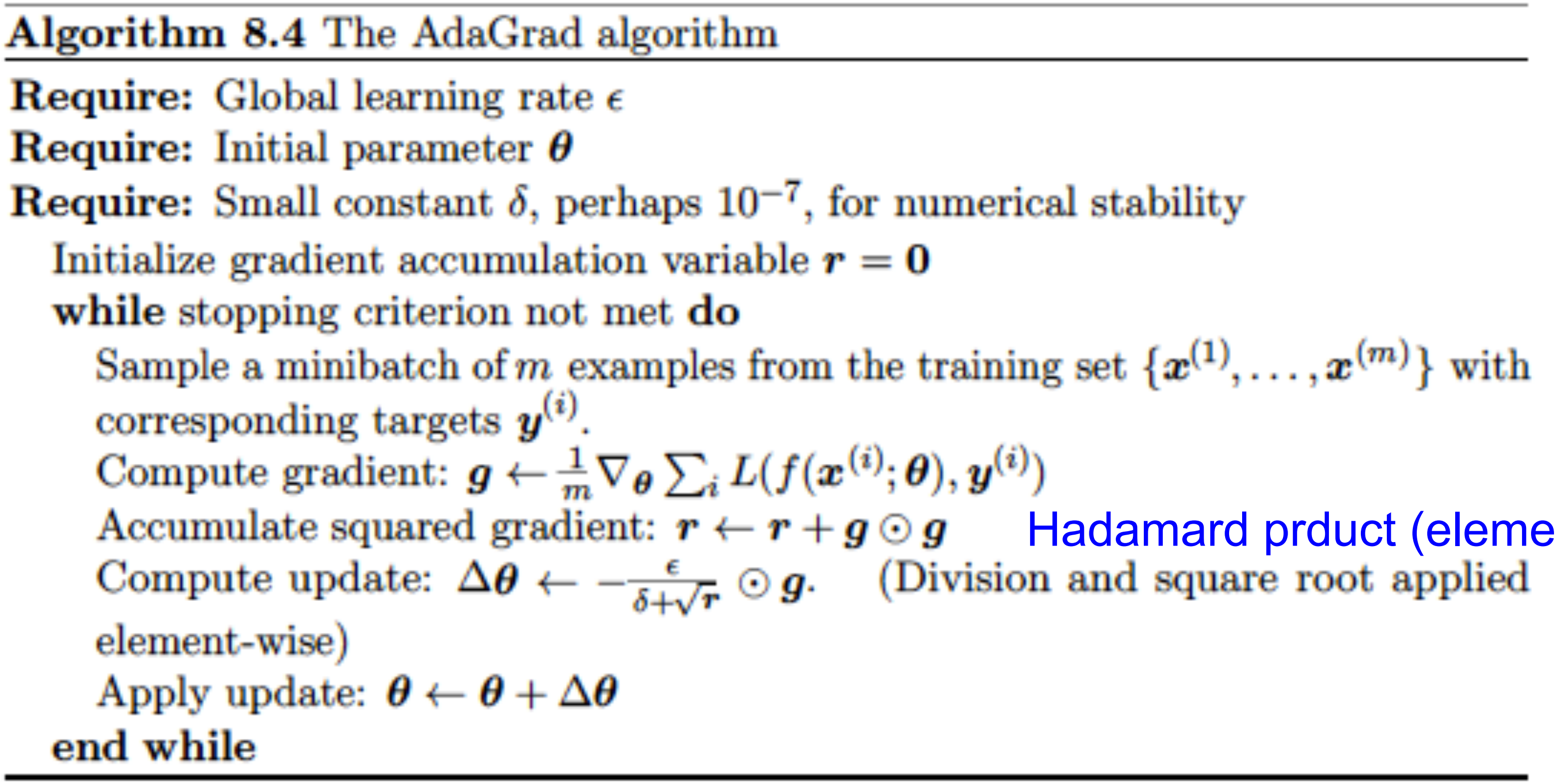
$$\mathbf{p} = \mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)} \qquad \mathbf{v} = \mathbf{g}^{(\tau+1)} - \mathbf{g}^{(\tau)} \qquad \mathbf{u} = \frac{\mathbf{p}}{\mathbf{p}^{\mathrm{T}}\mathbf{v}} - \frac{\mathbf{G}^{(\tau)}\mathbf{v}}{\mathbf{v}^{\mathrm{T}}\mathbf{G}^{(\tau)}\mathbf{v}}.$$

Adaptive learning rates

- Sensitivity to some directions in parameter space
  - Seperate learning rate for each parameter
  - Adapt learning rates throughout the course of learning

- Delta-bar-delta algorithm
  - Heuristic approach
  - Idea – if the partial derivative of the loss rimains the same sign, then the learning rate should be increase, decreased otherwise

AdaGrad



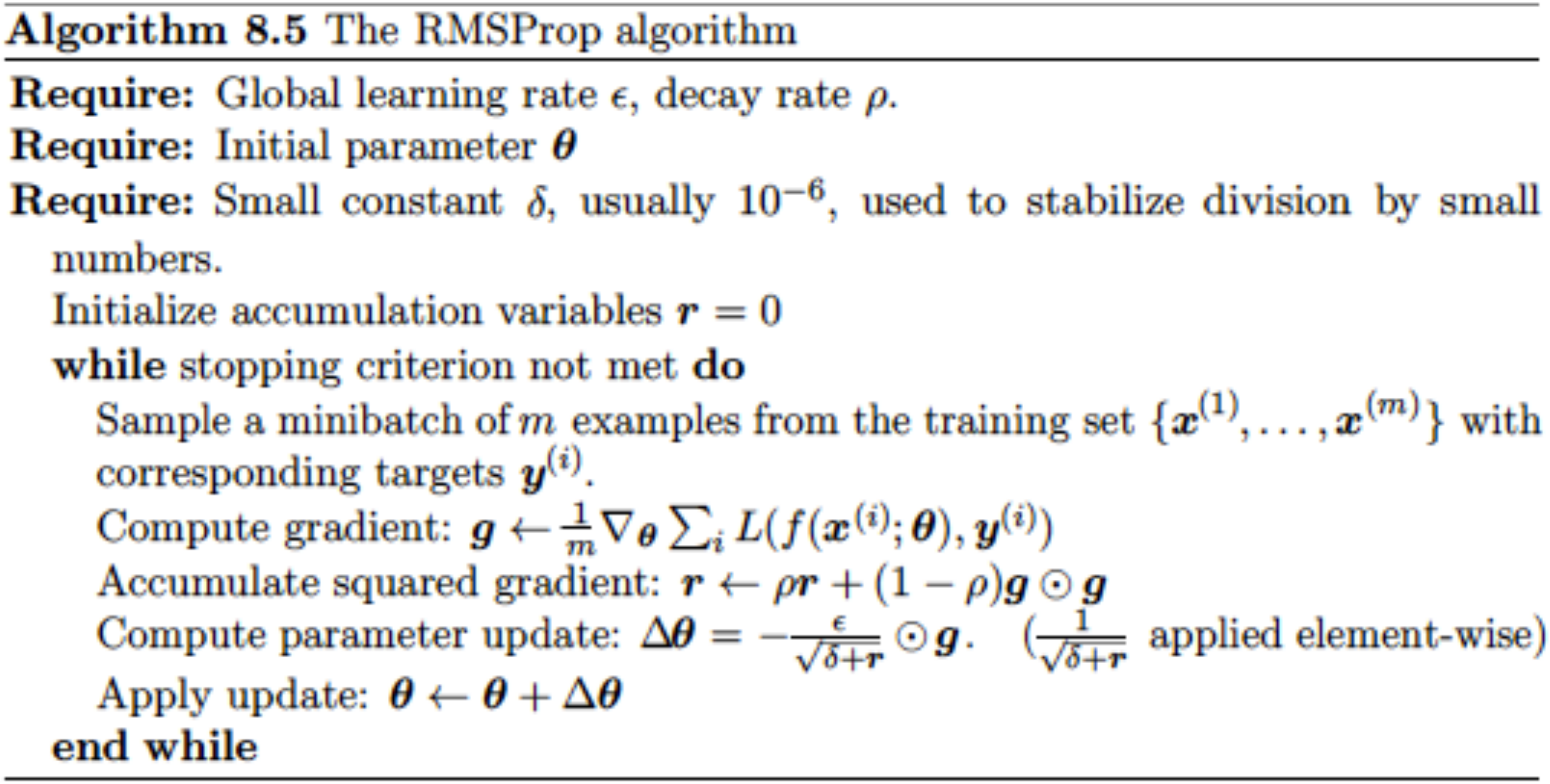
Algorithm 8.4 The AdaGrad algorithm

```
Require: Global learning rate $\epsilon$
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, perhaps $10^{-7}$, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.  (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Hadamard prduct (element-wise

Adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of the historical squared values of the gradient

RMSProp



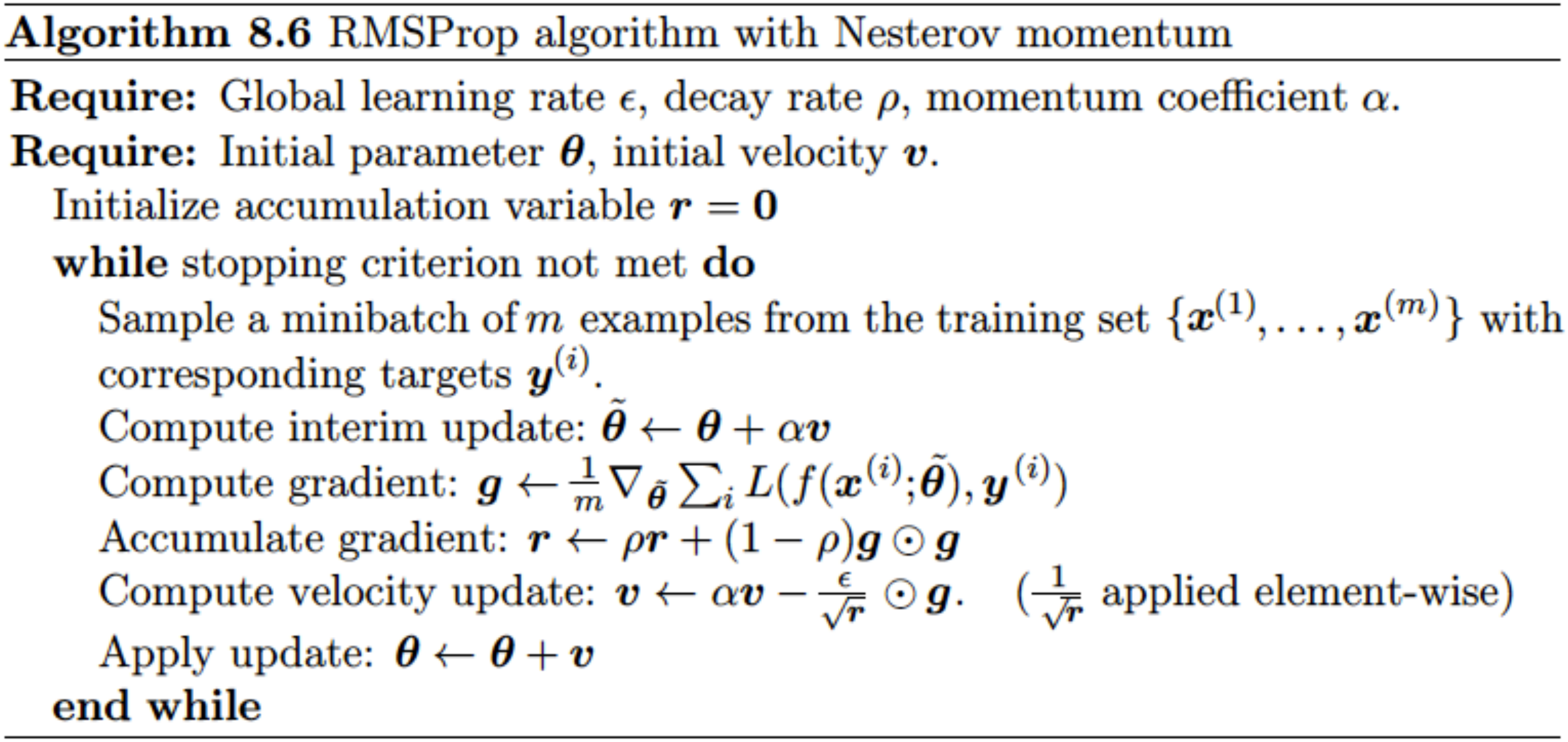
Algorithm 8.5 The RMSProp algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta+\boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Perform better in the nonconvex setting by changing the gradient accumulation Into an exponentially weighted moving average. Shrinks the learning rate according to the entire history of the squared gradient.

ML – Optimization algorithms

RMSProp

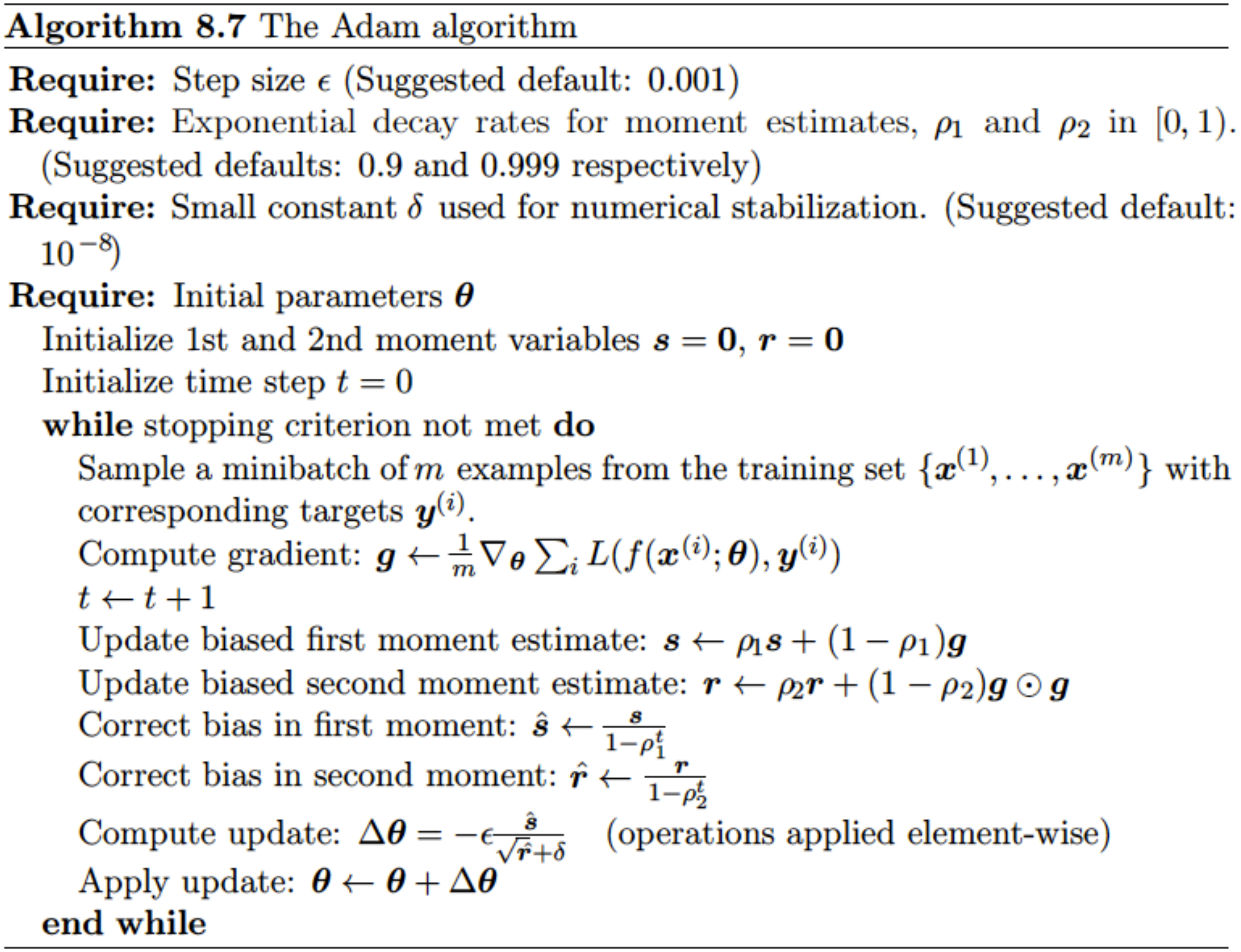


Algorithm 8.6 RMSProp algorithm with Nesterov momentum

```
Require: Global learning rate $\epsilon$, decay rate $\rho$, momentum coefficient $\alpha$.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while
```

ML – Optimization algorithms

Adam algorithm



Algorithm 8.7 The Adam algorithm

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
    (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
    $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
    Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
    Initialize time step $t = 0$
    while stopping criterion not met do
        Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with
        corresponding targets $\boldsymbol{y}^{(i)}$.
        Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
        $t \leftarrow t + 1$
        Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
        Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
        Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
        Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
        Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$    (operations applied element-wise)
        Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
    end while
```

Variant of RMSProp with momentum.

Adaptive Moments: 1) first order moment; 2) bias correction second order moment

Initialization strategies

- Heuristic

$$W_{i,j} \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right)$$

- Tipically
  - Biases for each unit heuristically chosen constant
  - weights random (Gaussian or Uniform distributions)
- Large initial weights
  - Recurrent NNs result chaos
- Further strategies
  - Sparse matrices initialization