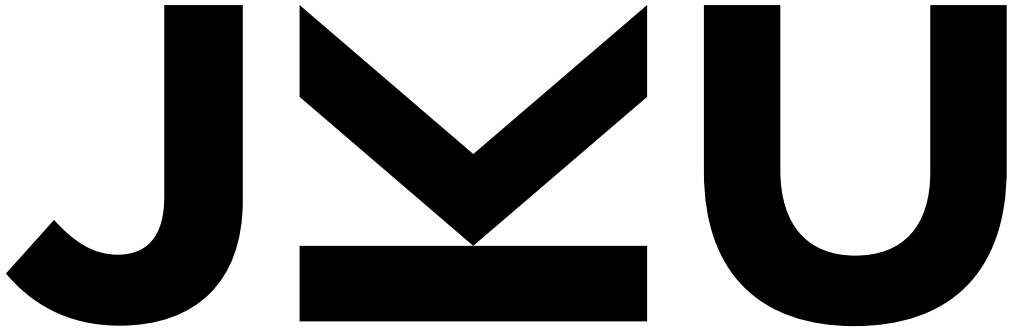
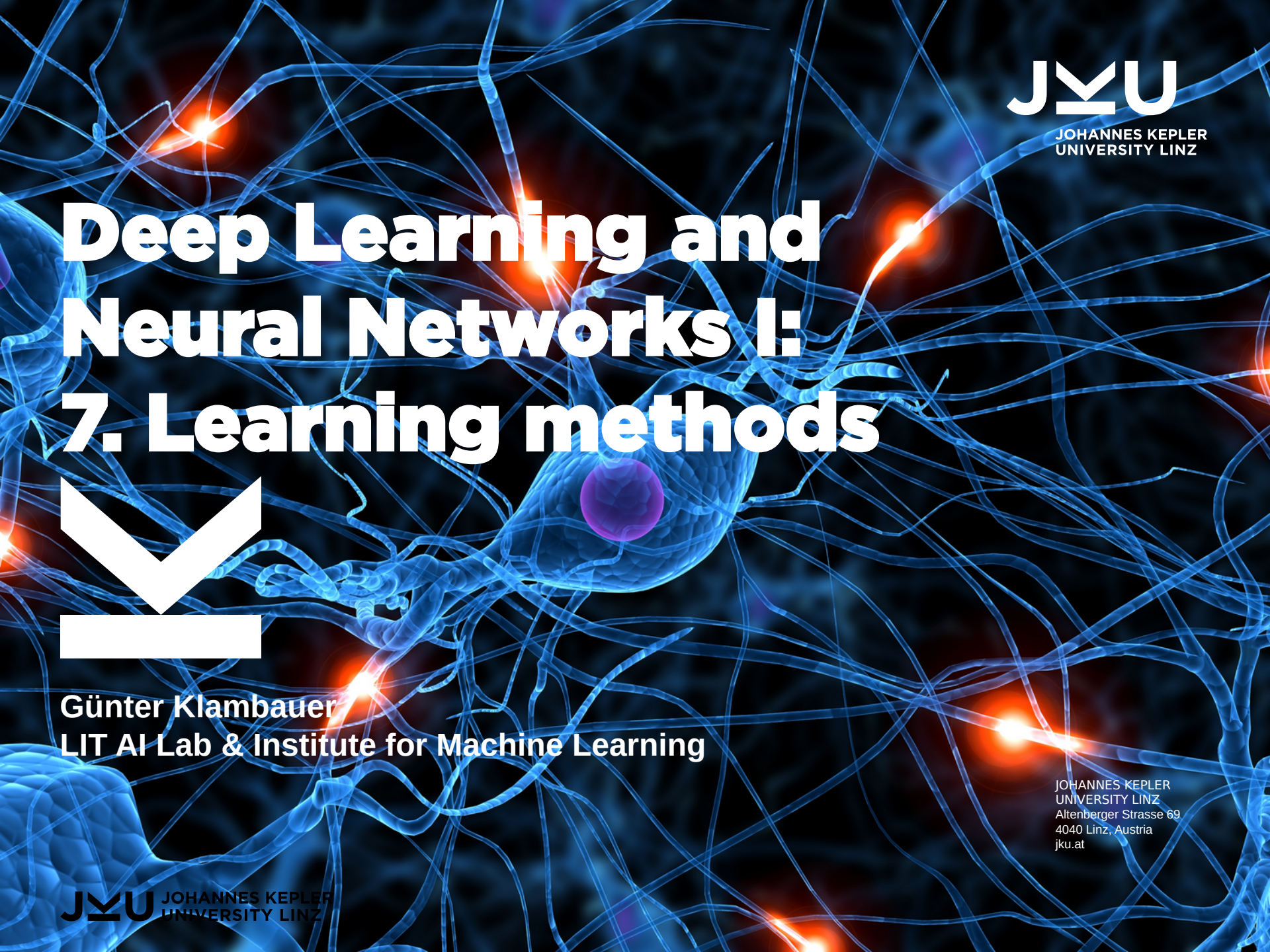


JOHANNES KEPLER
UNIVERSITY LINZ



JOHANNES KEPLER UNIVERSITY LINZ

Deep Learning and Neural Networks I: 7. Learning methods

Günter Klambauer
LIT AI Lab & Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Strasse 69
4040 Linz, Austria
jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Current topic

- Largest and most important Machine Learning conference takes place next week: **NeurIPS**
- https://neurips.cc/
- Fully virtual conference; participation is at relatively low-cost ($25 for full-time students)

- Side event on December 13: https://moleculediscovery.github.io/workshop2021/

Learning methods: basic algorithms

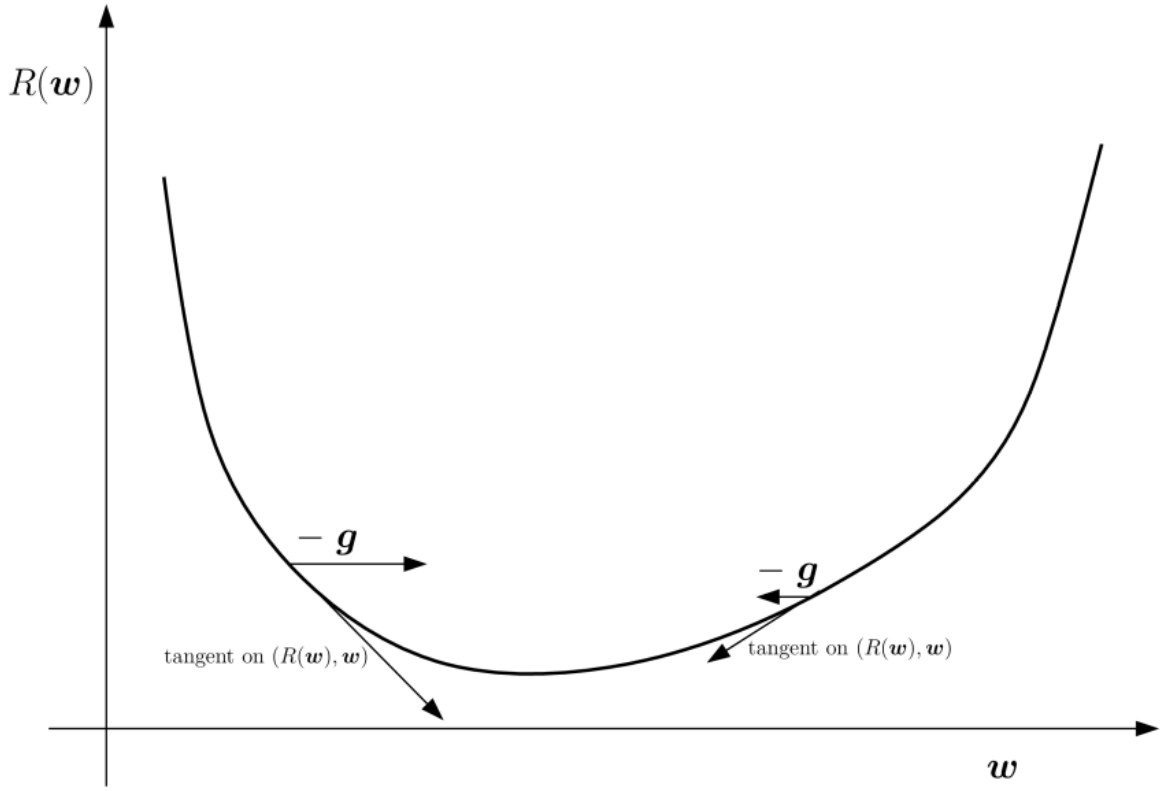
- In machine learning, we often adjust parameters $\boldsymbol{w}$ to optimize some objective function $R(\boldsymbol{w})$
- Objective functions, e.g. : $R(\boldsymbol{w}) = \sum_{n=1}^{N} (y^n - \boldsymbol{w}^T \boldsymbol{x}^n)^2 + \|\boldsymbol{w}\|^2$
- Adapting those parameters is called *learning*
- Difference between “learning” and pure optimization:
  - e.g. we use the empirical error as surrogate to maximize accuracy.
- In ML, we often have access to the gradient of the objective function

Overview

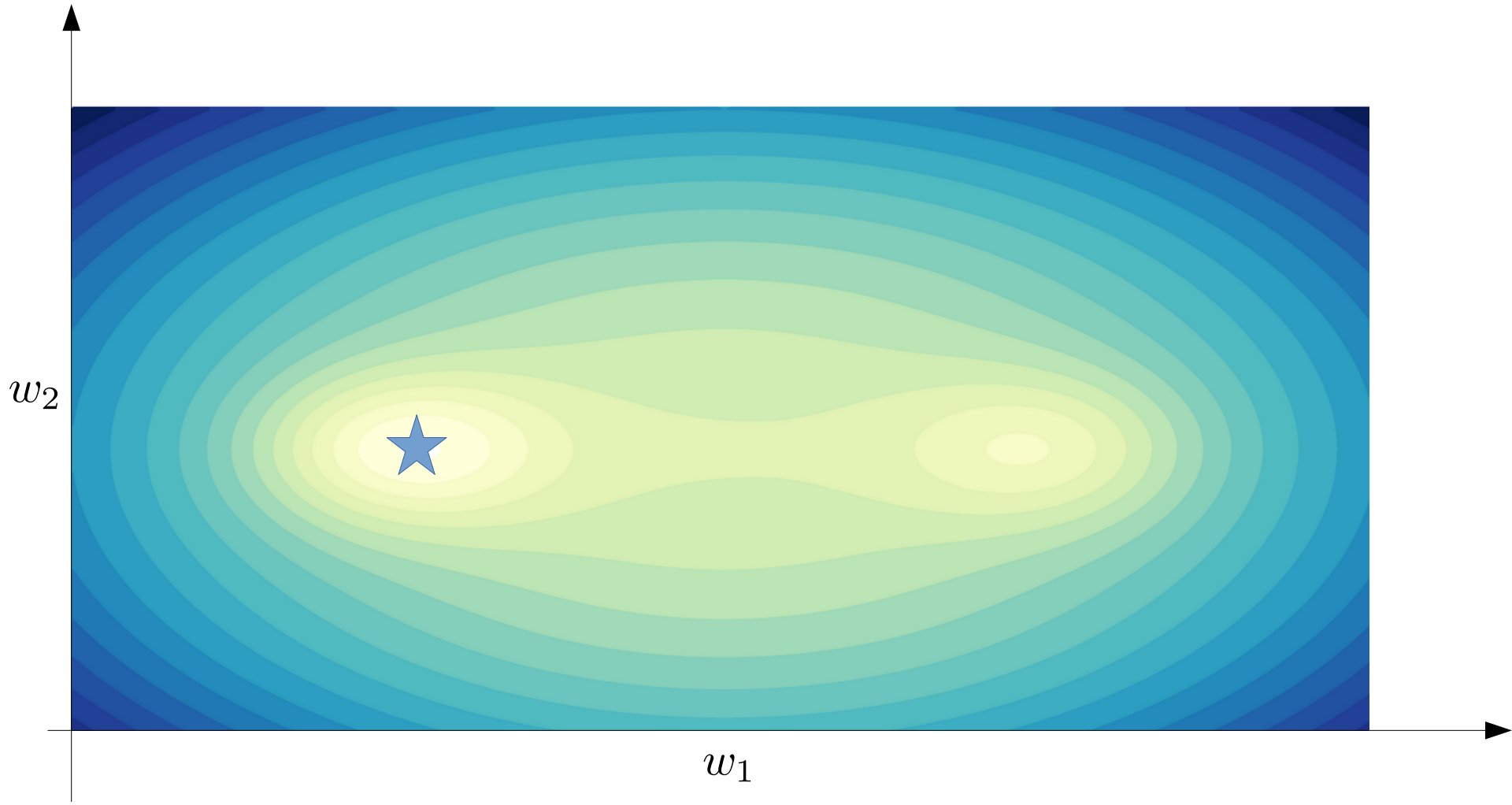
- 7.1 Gradient Descent
- 7.2 Gradient Descent with Momentum Term
- 7.3 Stochastic Gradient Descent (SGD)
- 7.4 Adaptive Learning Rate Optimizers
- 7.5 First-order Gradient Alternatives
- 7.6 Second-order Methods

7.1 Gradient descent

The negative gradient $-\boldsymbol{g}$ gives the direction of the *steepest decent*, depicted by the tangent on $(R(\boldsymbol{w}), \boldsymbol{w})$:



Optimization problems



7.1 Gradient descent

Assume that the objective function $R(g(.;\boldsymbol{w}))$is a differentiable function w. r. t $\boldsymbol{w}$.

The gradient is

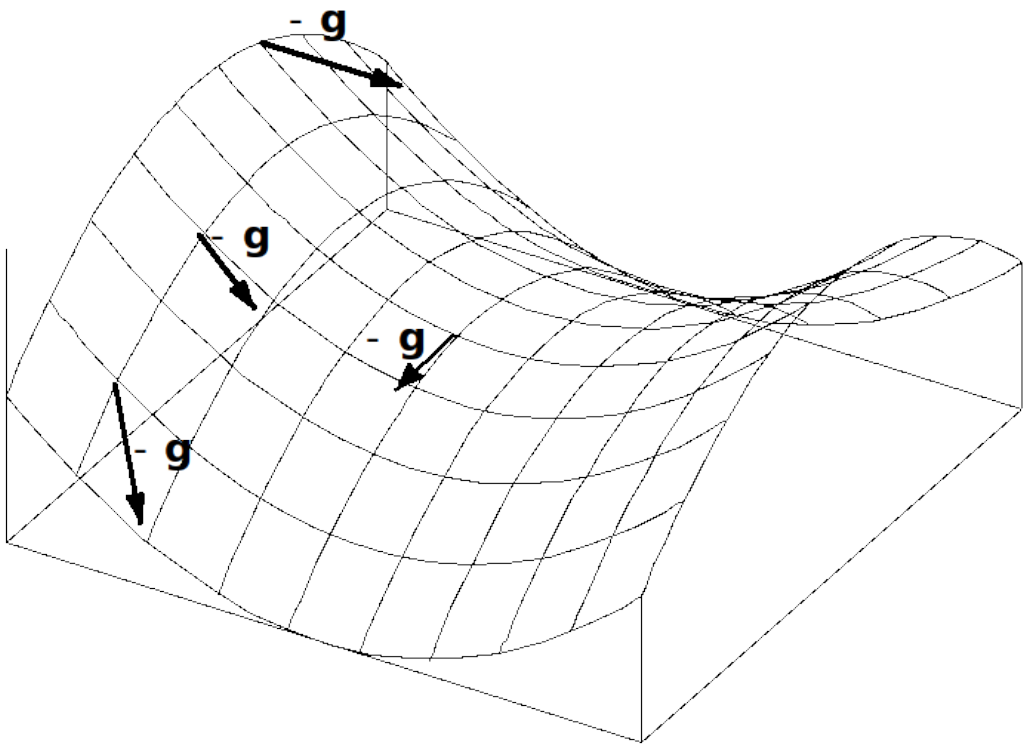
$$\frac{\partial R(\boldsymbol{w})}{\partial \boldsymbol{w}} = (\nabla_{\boldsymbol{w}} R(\boldsymbol{w}))^T = \left(\frac{\partial R(\boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial R(\boldsymbol{w})}{\partial w_W} \right).$$

- For the gradient of $\mathrm{R}(\boldsymbol{w})$ we use the *column vector*

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}} R(\boldsymbol{w}).$$

7.1 Gradient descent

The negative gradient $-g$ attached at different positions on a two dimensional error surface $(R(\boldsymbol{w}), \boldsymbol{w})$:



7.1 Gradient descent

- The gradient descent update is

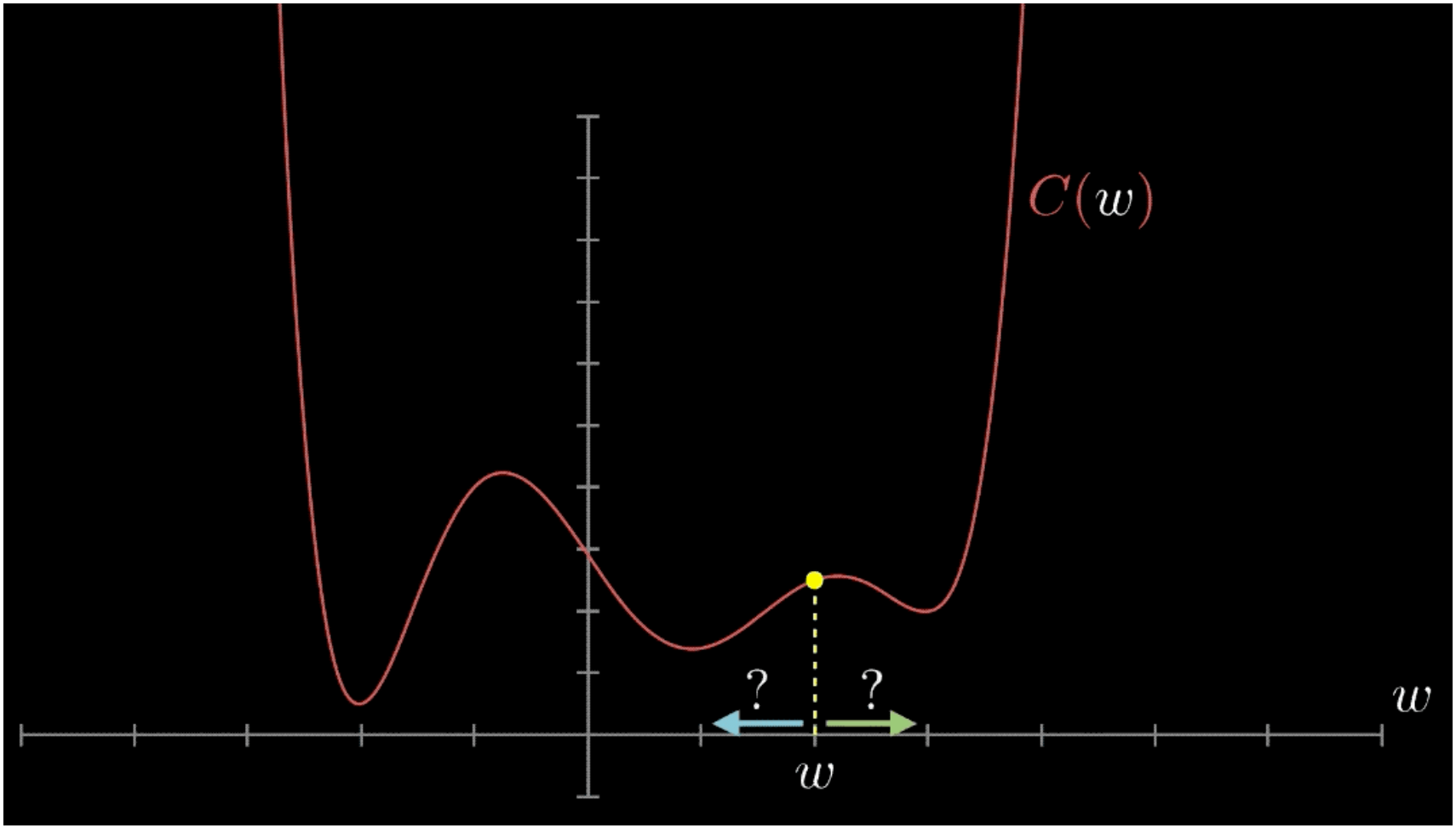
$$\Delta \boldsymbol{w}^{(t)} = -\eta \nabla_{\boldsymbol{w}} R(\boldsymbol{w})|_{\boldsymbol{w}^{(t)}}$$
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \Delta \boldsymbol{w}^{(t)}$$

- We gradually change by update steps

$$\boldsymbol{w}^{(0)} \rightarrow \boldsymbol{w}^{(1)} \rightarrow \ldots \rightarrow \boldsymbol{w}^{(t)}$$

```
while True:
    dw = get_gradient(w)
    w  = w - eta*dw
```

7.1 Gradient Descent



7.1 Gradient Descent: Convergence

Theorem 7.1. *Suppose the function $R : \mathbb{R}^W \mapsto \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$, i.e. $\|\nabla R(\boldsymbol{w}) - \nabla R(\boldsymbol{v})\|_2 \leq L\|\boldsymbol{w} - \boldsymbol{v}\|_2$ for any $\boldsymbol{w}, \boldsymbol{v}$. Then gradient descent with a fixed step size $\eta \leq 1/L$ will yield a solution $\boldsymbol{w}^{(t)}$ which satisfies*

$$R(\boldsymbol{w}^{(t)}) - R(\boldsymbol{w}^*) \leq \frac{\|\boldsymbol{w}^{(0)} - \boldsymbol{w}^*\|_2^2}{2\eta t}, \tag{7.5}$$

where $R(\boldsymbol{w}^*)$ is the optimal value.

- Intuitively, Gradient Descent converges when the error function is convex and sufficiently smooth
  - The learning rate has too be sufficiently small
- (proof on blackboard/whiteboard)

7.1 Gradient Descent: problems

- Local minima in loss function?
- Saddle points in loss function?
- Interplay with dimension of parameter space

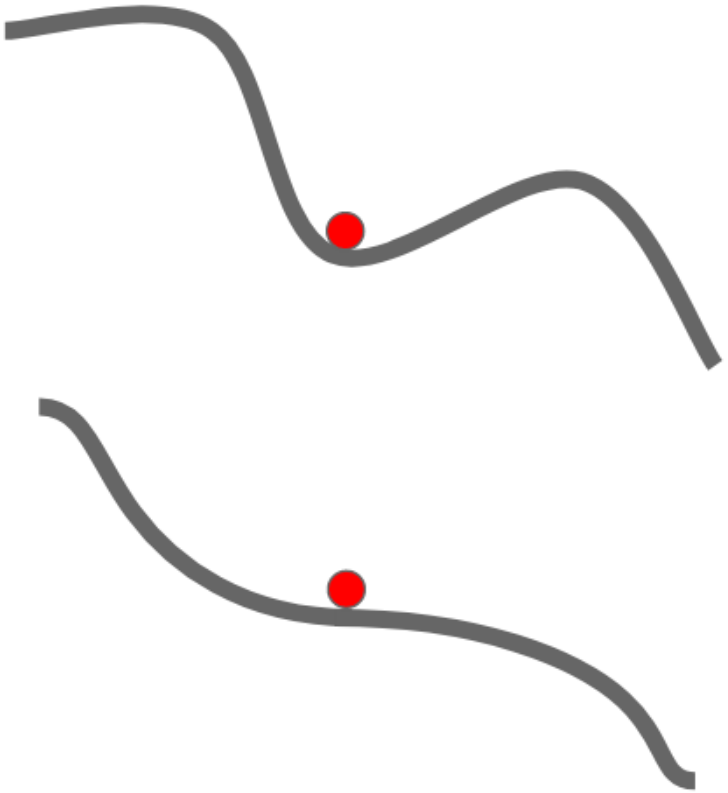
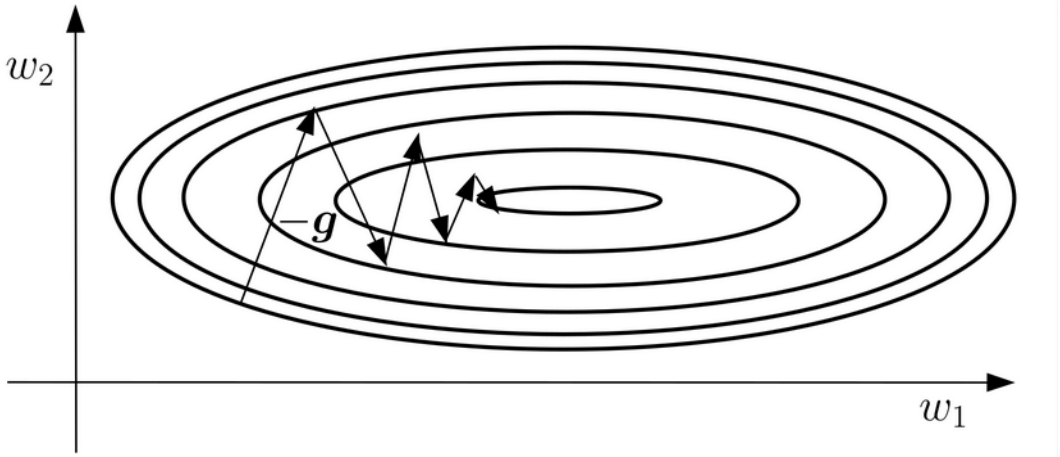


Image Source: Fei-Fei Li, Justin Johson, and Serena Yeung
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf

7.1 Gradient Descent: problems

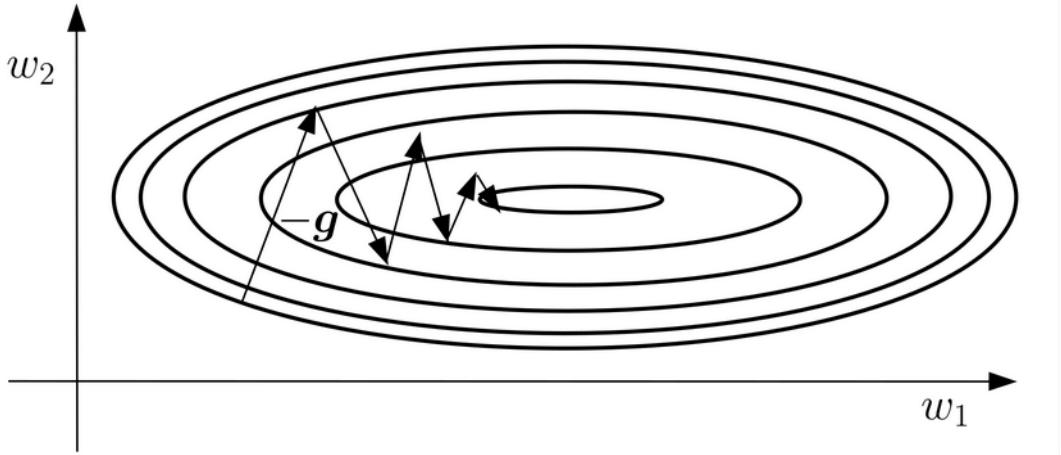
- Oscillation



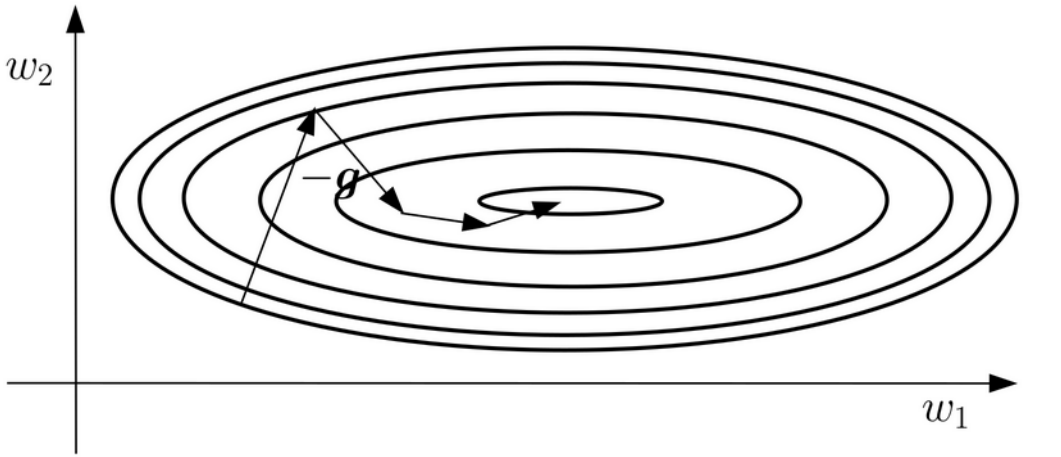
7.2 Gradient Descent with Momentum Term

Oscillation of the negative gradient $-g$

- without momentum:



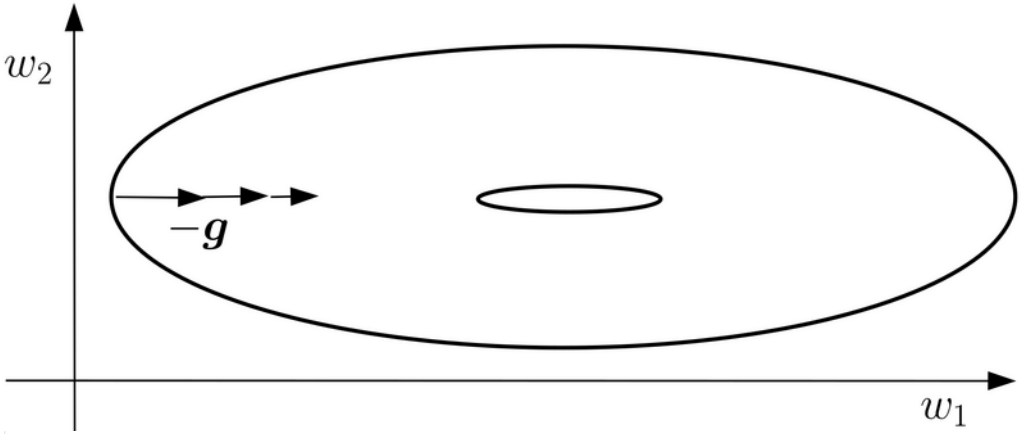
- with momentum:



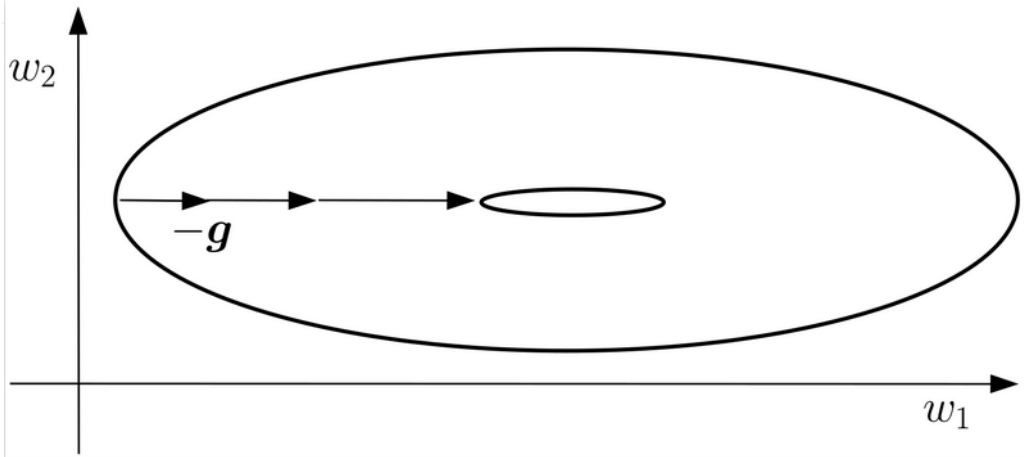
7.2 Gradient Descent with Momentum Term

Another effect of the momentum term is that in flat regions the gradients pointing in the same directions are accumulated and the learning rate is implicitly increased.

- without momentum:



- with momentum:



7.2 Gradient Descent with Momentum Term

The gradient descent update with momentum term is

$$\boldsymbol{m}^{(t)} = \mu \boldsymbol{m}^{(t-1)} + (1-\mu)\boldsymbol{g}^{(t)}$$
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{m}^{(t)}$$

where $0 \leq \mu \leq 1$ is the *momentum parameter* or *momentum factor*, which is usually close to one.

```
vm = 0
while True:
    dw = get_gradient(w)
    vm = mu*vm + (1-mu)*dw
    w  = w - eta*vm
```

7.2 Gradient Descent with Momentum Term Interpretation as physical quantity

Physical view of momentum

Learning with momentum has a particular interpretation as the velocity m of a physical object:

$$
\begin{aligned}
m^{(t)} =& \mu m^{(t-1)} + (1-\mu) g^{(t)} = \\
=& \mu^2 m^{(t-2)} + \mu(1-\mu) g^{(t-1)} + (1-\mu) g^{(t)} = \\
=& \mu^3 m^{(t-3)} + \mu^2(1-\mu) g^{(t-2)} + \mu(1-\mu) g^{(t-1)} + (1-\mu) g^{(t)} = \\
=& \mu^t m^{(0)} + (1-\mu) \sum_{i=1}^{t} \mu^{t-i} g^{(i)}
\end{aligned}
$$

7.2 Gradient Descent with Momentum Term Interpretation as physical quantity

From this and $m^{(0)} = 0$ follows that

$$\Delta w^{(t)} = -\sum_{i=1}^{t} \mu^{t-i} \left(\eta(1-\mu) g^{(i)} \right).$$

Which means that we could also write the algorithm as

$$\begin{aligned} \boldsymbol{m}^{(t)} &= \mu \boldsymbol{m}^{(t-1)} - \eta' \boldsymbol{g} \\ \Delta \boldsymbol{w}^{(t)} &= \boldsymbol{m}^{(t)} \end{aligned} \quad , \eta' = -\eta(1-\mu).$$

We can interpret the right hand side as force times a time interval (which is equal to one).

$$\boldsymbol{m}^{(t)} - \boldsymbol{m}^{(t-1)} = \left(-(1-\mu)\boldsymbol{m}^{(t-1)} - \eta' \boldsymbol{g}^{(t)} \right) \Delta t$$

$$\frac{d\boldsymbol{m}(t)}{dt} = \underbrace{-(1-\mu)\boldsymbol{m}(t)}_{\text{friction}} \underbrace{-\eta' \boldsymbol{g}(t)}_{\text{Lossforce}}$$

7.2 Gradient Descent with Momentum Term Interpretation as physical quantity

- First we look at the case when $\boldsymbol{g}(t) = 0$:

$$\boldsymbol{m}(t) = \boldsymbol{m}(t_0)\ e^{-(1-\mu)t}.$$

- Now we take a look at the parameter update in continuous time:

$$\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)} = \boldsymbol{m}^{(t)} \Delta t$$

$$\frac{\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}}{\Delta t} = \boldsymbol{m}^{(t)}.$$

  In the limit $\Delta t \to 0$ we get:

$$\frac{d\boldsymbol{w}(t)}{dt} = \boldsymbol{m}(t).$$

- Differential equation which describes the motion of $\boldsymbol{w}(t)$:

$$\frac{d^2\boldsymbol{w}(t)}{dt^2} = -(1-\mu)\frac{d\boldsymbol{w}(t)}{dt} - \eta'\boldsymbol{g}(t).$$

7.3 Stochastic Gradient Descent (SGD)

Objective function for neural networks:

$$R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} L\left(y^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right)$$

- **Full-batch-gradient:**

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{w}} L\left(y^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right).$$

- **Stochastic gradient descent:**
  Minibatch: $\mathcal{B} = \{j_1, ..., j_B\} \subset \{1, ..., N\}$

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) \approx \frac{1}{B} \sum_{b=1}^{B} \nabla_{\boldsymbol{w}} L\left(y^{j_b}, \boldsymbol{g}\left(\boldsymbol{x}^{j_b}; \boldsymbol{w}\right)\right).$$

- **On-line learning:** SGD when $B = 1$.

7.3 Stochastic Gradient Descent
Noisy estimates of gradient

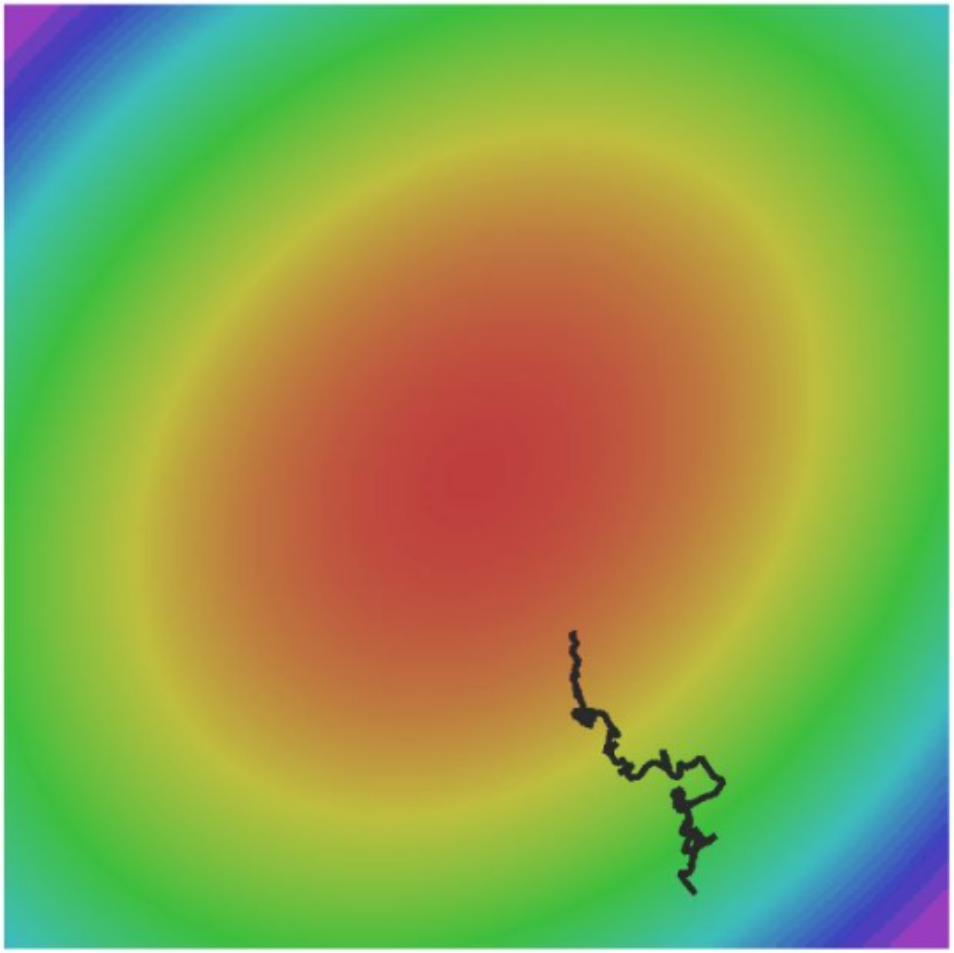


Image Source: Fei-Fei Li, Justin Johson, and Serena Yeung
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf

7.3 Stochastic Gradient Descent (SGD)

Efficiency of stochastic gradient descent:

- The expression for the empirical error is an expectation of the loss of the training set:

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) = \mathbb{E}_{\boldsymbol{x}, y \sim \tilde{p}_{\text{data}}} \left[\nabla_{\boldsymbol{w}} L\left(y, \boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{w}\right)\right) \right],$$

- The improvement of the estimate does not scale linearly with the batch size B, since the standard error of the mean of Bsamples is given by

$$\frac{\sigma}{\sqrt{B}},$$

where σ is the true standard deviation in the samples.

- Example: 100,000 samples in training data; gradient calculated on 1,000. 100 times less computations; gradient standard error 10 times larger

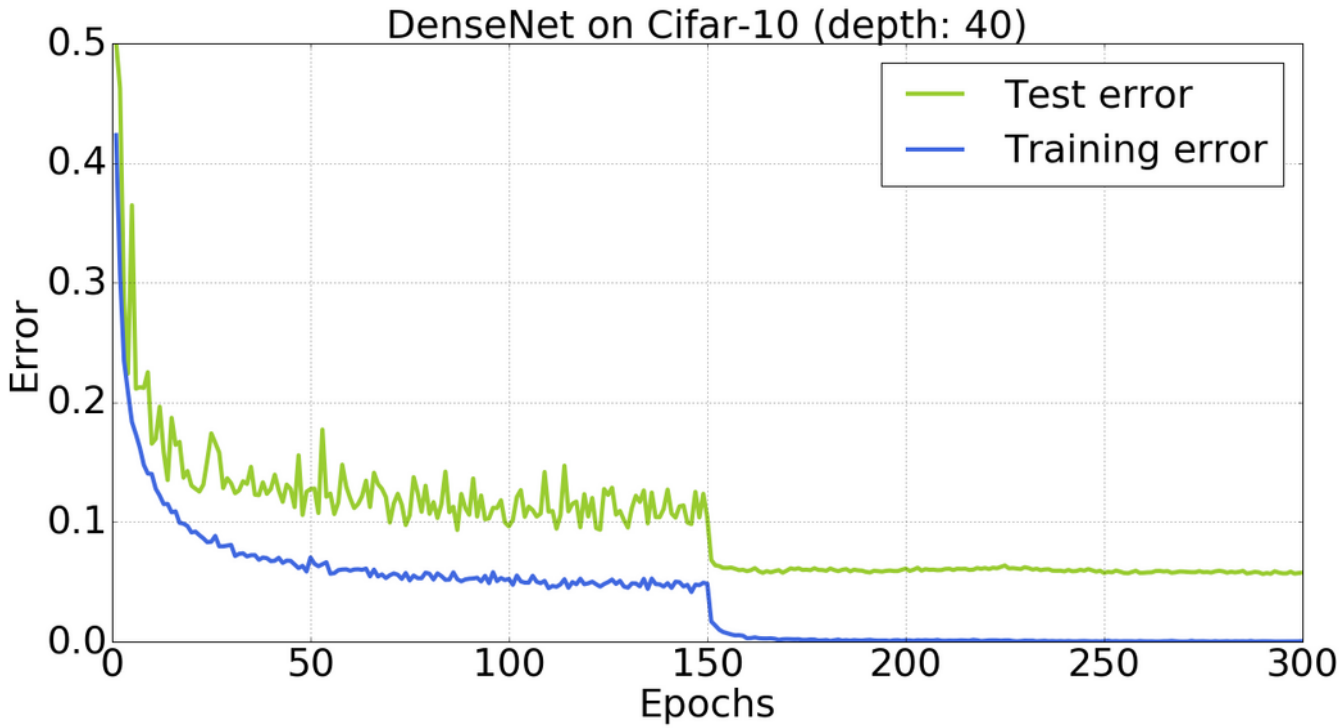
7.3 Stochastic Gradient Descent (SGD)

Implementation of learning with stochastic gradient descent:

- Sampling without replacement: A mini-batch is drawn and removed from training set
- Update step:
  - The gradient on this mini-batch is calculated
  - The parameters are updated
- When training set is depleted, an **epoch** has ended.
  Number of updates per epoch: $\lceil N/B \rceil$.
- Practitioners monitor the loss and performance metric on the training set (and a validation set).

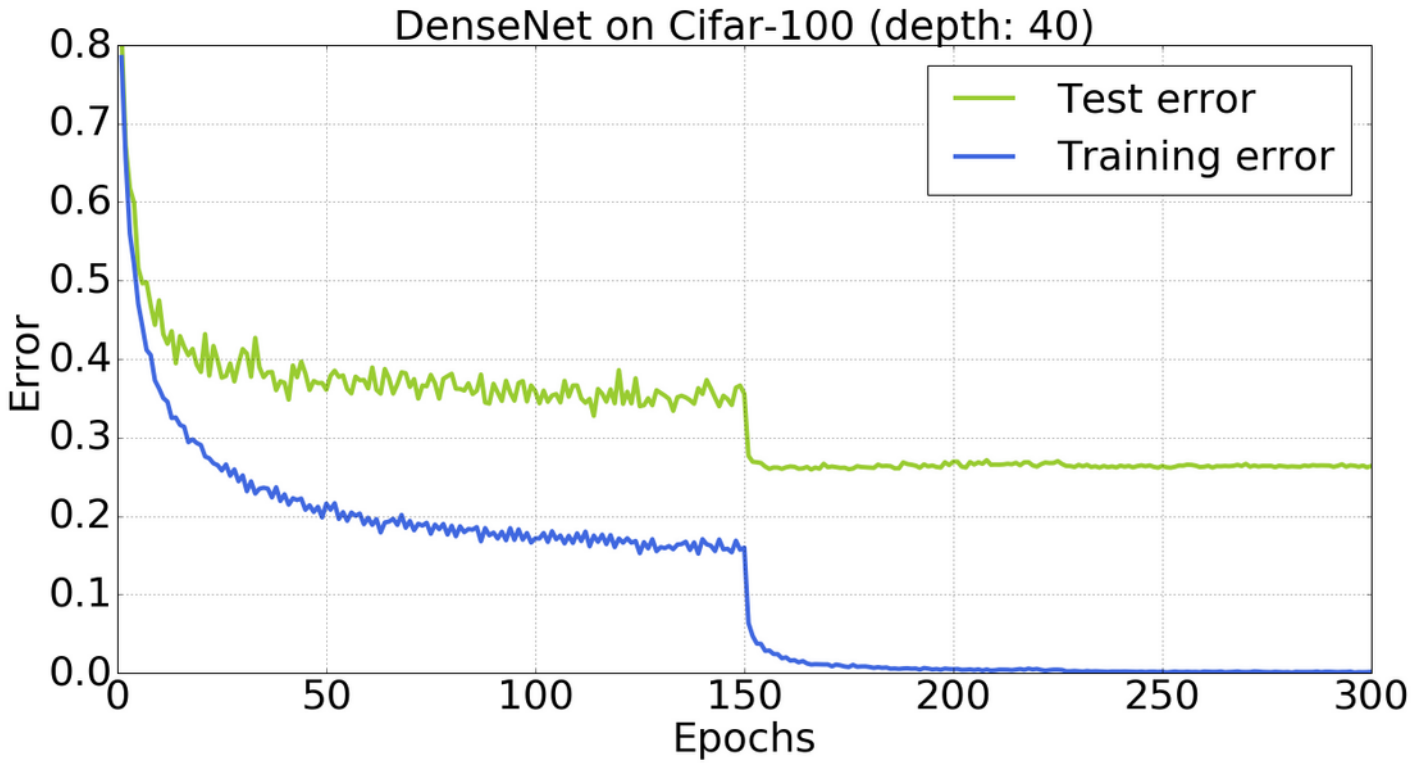
7.3 Stochastic Gradient Descent (SGD)

Learning curves for training of a neural network (DenseNet)



7.3 Stochastic Gradient Descent (SGD)

Learning curves for training of a neural network (DenseNet)



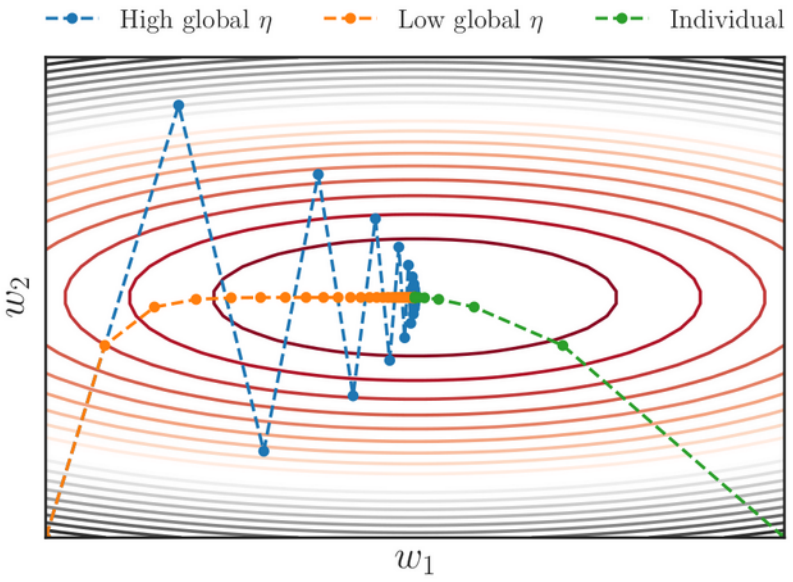
7.4 Adaptive Learning Rate Optimizers

Choosing the learning rate is critical for success

- Learning rate to high: update may “overshoot” its goal
- Learning rate to slow: progress may be slow
- Each parameter might benefit from different learning rates
- A specific learning η_i rate for each parameter w_i can be beneficial:

$$\Delta w_i = -\eta_i g_i$$

or $$\Delta \boldsymbol{w} = -\boldsymbol{\eta} \odot \boldsymbol{g}$$



7.4 Adaptive Learning Rate Optimizers

$\Delta \boldsymbol{w}$ leads to a point with lower error: $\Delta \boldsymbol{w} = -\boldsymbol{\eta} \odot \boldsymbol{g}$

$$R(\boldsymbol{w} + \boldsymbol{\Delta w}) = R(\boldsymbol{w}) + \boldsymbol{g}^T \Delta \boldsymbol{w} + \mathcal{O}(\Delta \boldsymbol{w}^2)$$

With $-\boldsymbol{g}^T \ (\boldsymbol{\eta} \odot \boldsymbol{g}) = -\sum_i g_i g_i \eta_i =$

$$\leq -\sum_i g_i g_i \eta_{\min}$$

$$= \ -\eta_{\min} \sum_i g_i g_i \leq 0$$

we get $R(\boldsymbol{w} + \boldsymbol{\Delta w}) \leq R(\boldsymbol{w})$.

7.4 Adaptive Learning Rate Optimizers

Further issue of a fixed learning rate:
This assumes that the loss surface looks similar everywhere

Therefore useful: Adaption of the learning rate to the topology of the loss surface at the current point.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \boldsymbol{\eta}^{(t)} \odot \boldsymbol{g}^{(t)}.$$

the vector $\boldsymbol{\eta}$ provides an individual learning rate for each parameter. The elements of $\boldsymbol{\eta}$ must be non-negative.

7.4 Adaptive Learning Rate Optimizers AdaGrad

- AdaGrad: adapts step size by remembering past gradients.

  With the sum of squared past gradients for each parameter

$$v_i^{(t)} = \sum_{s=1}^{t} (g_i^{(s)})^2, g_i^{(s)} : i\text{-th component of the gradient at step } s$$

we get:

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{1}{\sqrt{v_i^{(t)} + \epsilon}} g_i^{(t)},$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{1}{\sqrt{\boldsymbol{v}^{(t)} + \epsilon}} \odot \boldsymbol{g}^{(t)}.$$

typica)

```
grad_squared=0
while True:
    dw = get_gradient(w)
    grad_squared = grad_squared + dw * dw
    w = w - eta*dw/(np.sqrt(grad_squared+1e-7))
```

7.4 Adaptive Learning Rate Optimizers: RMSprop, AdaDelta

- RMSprop:

$$\boldsymbol{v}^{(t)} = \gamma\, \boldsymbol{v}^{(t-1)} + (1-\gamma)\, (\boldsymbol{g}^{(t)})^2$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{1}{\sqrt{\boldsymbol{v}^{(t)} + \epsilon}} \odot \boldsymbol{g}^{(t)}.$$

```
grad_squared=0
while True:
    dw = get_gradient(w)
    grad_squared = gamma * grad_squared + (1-gamma) * dw * dw
    w = w - eta*dw/(np.sqrt(grad_squared+1e-7))
```

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{\sqrt{\boldsymbol{d}^{(t-1)} + \epsilon}}{\sqrt{\boldsymbol{v}^{(t)} + \epsilon}} \odot \boldsymbol{g}^{(t)},$$

where: $\gamma \in [0, 1)$.

7.4 Adaptive Learning Rate Optimizers Adam

- Combines the idea of momentum term and Adelta by memorizing both exponentially decaying averages of *past gradients* $\boldsymbol{g}^{(t)}$ and *past squared gradients* $(\boldsymbol{g}^{(t-1)})^2$.

$$\boldsymbol{m}^{(t)} = \beta_1 \boldsymbol{m}^{(t-1)} + (1-\beta_1)\boldsymbol{g}^{(t)}$$
$$\boldsymbol{v}^{(t)} = \beta_2 \boldsymbol{v}^{(t-1)} + (1-\beta_2)(\boldsymbol{g}^{(t)})^2$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{1}{\sqrt{\boldsymbol{v}^{(t)}} + \epsilon} \odot \boldsymbol{m}^{(t)}$$

```
first_moment = 0
second_moment = 0
while True:
    dw = get_gradient(w)
    first_moment = beta1 * first_moment + (1-beta1) * dw
    second_moment = beta2 * second_moment + (1-beta2) * dw * dw
    w  = w - eta*first_moment / (np.sqrt(second_moment)+1e-7)
```

J:

7.4 Adaptive Learning Rate Optimizers: Delta-delta rule; delta-bar-delta rule

- Delta-delta rule:
  Learning rate at step t

$$\eta_i^{(t)} = \eta_i^{(t-1)} + \gamma g_i^{(t)} g_i^{(t-1)} \qquad , \gamma : \text{step size}$$

- Delta-bar-delta rule:

$$\eta_i^{(t)} = \eta_i^{(t-1)} + \Delta\eta_i^{(t)}$$

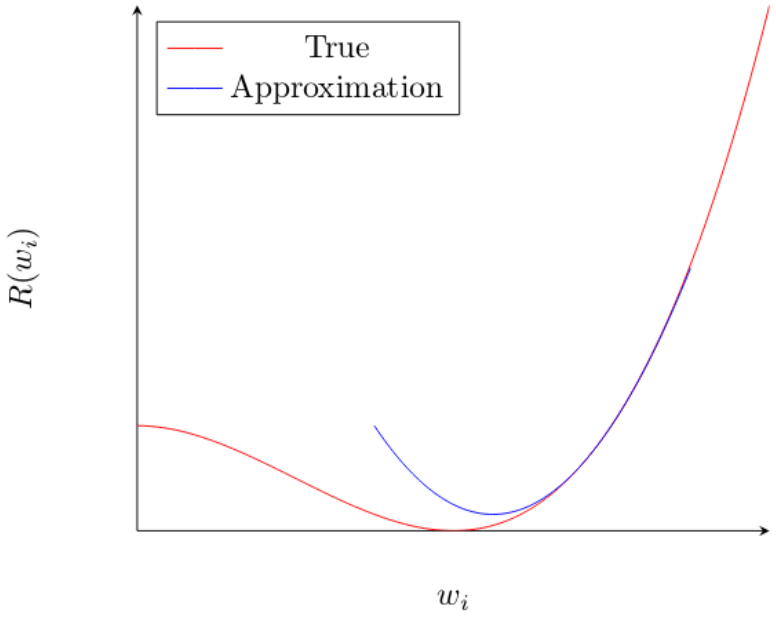
$$\Delta\eta_i^{(t)} = \begin{cases} \kappa & \bar{g}_i^{(t)} g_i^{(t)} > 0 \\ -\phi\eta_i^{(t-1)} & \bar{g}_i^{(t)} g_i^{(t)} < 0 \\ 0 & \text{otherwise} \end{cases},$$

  where $\bar{g}_i^{(t)} = \theta \bar{g}_i^{(t-1)} + (1-\theta) g_i^{(t)}$

Demo on Optimizers

7.5 First Order Gradient Alternatives Quickprop

- Quickprop:
  Main Idea: Assume that loss surface is locally quadratic.



The resulting update is: $\Delta^{\text{new}} w_i = \frac{g_i^{\text{new}}}{g_i^{\text{old}} - g_i^{\text{new}}} \Delta^{\text{old}} w_i.$

7.5 First Order Gradient Alternatives Quickprop

- Assume $\mathrm{R}(\boldsymbol{w})$ is a piece-wiese function so that: $R(\boldsymbol{w})_i = R(w_i)$.
- Taylor expansion:

$$R(w_i + \Delta w_i) = R(w_i) + g_i \Delta w_i + \frac{1}{2} g_i' (\Delta w_i)^2 + \mathcal{O}\left((\Delta w_i)^3\right)$$

- To maximize the change in the objective, one can minimize

$$R(w_i + \Delta w_i) - R(w_i) = g_i \Delta w_i + \frac{1}{2} g_i' (\Delta w_i)^2$$

- We set the derivative w. r. t. Δw_i to zero and obtain: $\Delta w_i = -\frac{g_i}{g_i'}$
- Now, approximate: $g_i' \approx \frac{g_i^{\text{new}} - g_i^{\text{old}}}{w_i - (w_i - \Delta^{\text{old}} w_i)}$
- Plugging this approximation into $\Delta w_i = -\frac{g_i}{g_i'}$ gives the update

7.5 First Order Gradient Alternatives Rprop

- Gradient is completely replaced

$$\Delta w_i = -\eta_i^{\text{new}} \text{sign}(g_i^{\text{new}}).$$

- Learning rate adjustment similar to delta-bar-delta rule:

$$\eta_i^{\text{new}} = \begin{cases} \eta^+ \eta_i^{\text{old}} & g_i^{\text{old}} g_i^{\text{new}} > 0 \\ \eta^- \eta_i^{\text{old}} & g_i^{\text{old}} g_i^{\text{new}} < 0 \\ \eta_i^{\text{old}} & \text{otherwise} \end{cases},$$

- Hyperparameters: η^+, η^-

Choosing learning rates; learning rate decay

- Learning rate (LR) is a crucial hyperparameter
- LR can be decreased during learning

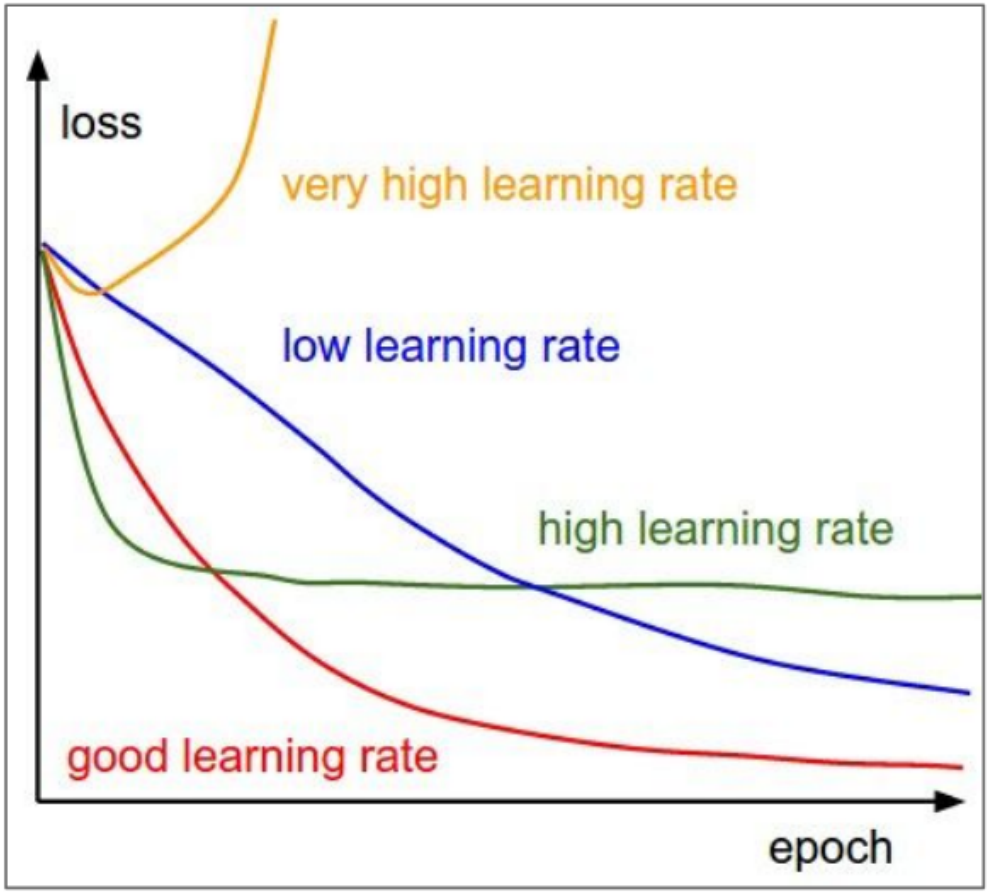


Image Source: Fei-Fei Li, Justin Johson, and Serena Yeung
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf

Choosing learning rates; learning rate decay

- Learning rate decay strategies:
  - No decay
  - Exponential decay
  - 1/t decay $\eta = \eta_0/(1+kt)$
  - Decay at plateaus
  - “Cosine annealing”

$$\eta = 1/2\eta_0(1+\cos(t\pi/T))$$

Loshchilov, I., & Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983.

  - Linear,
    Inverse sqrt,..

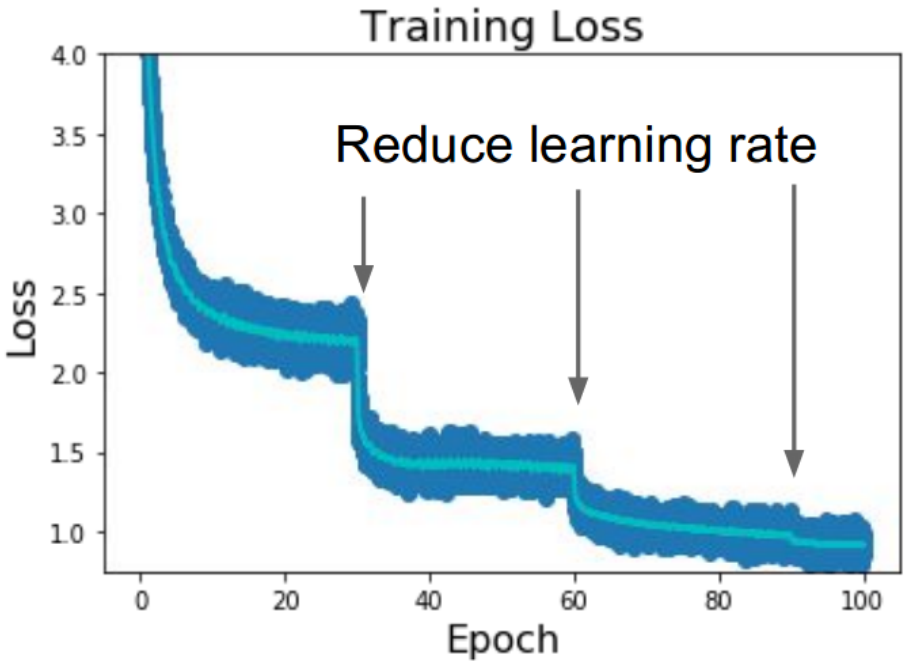
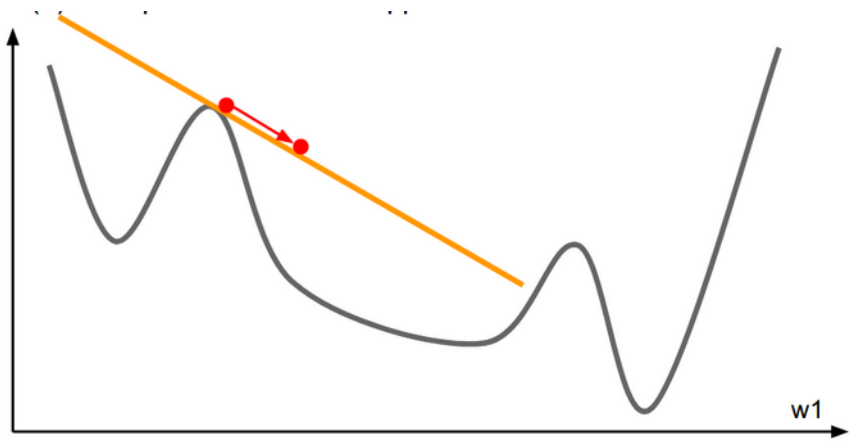


Image Source: Fei-Fei Li, Justin Johson, and Serena Yeung
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf

7.5 Second order methods

- Use the information on curvature (Hessian)
- Currently not used to train neural networks
  - Why?
- See also part II of this lecture.



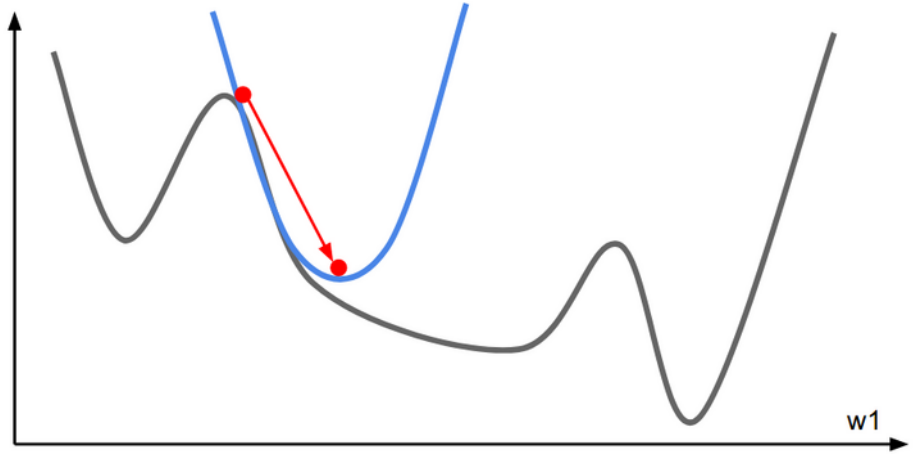
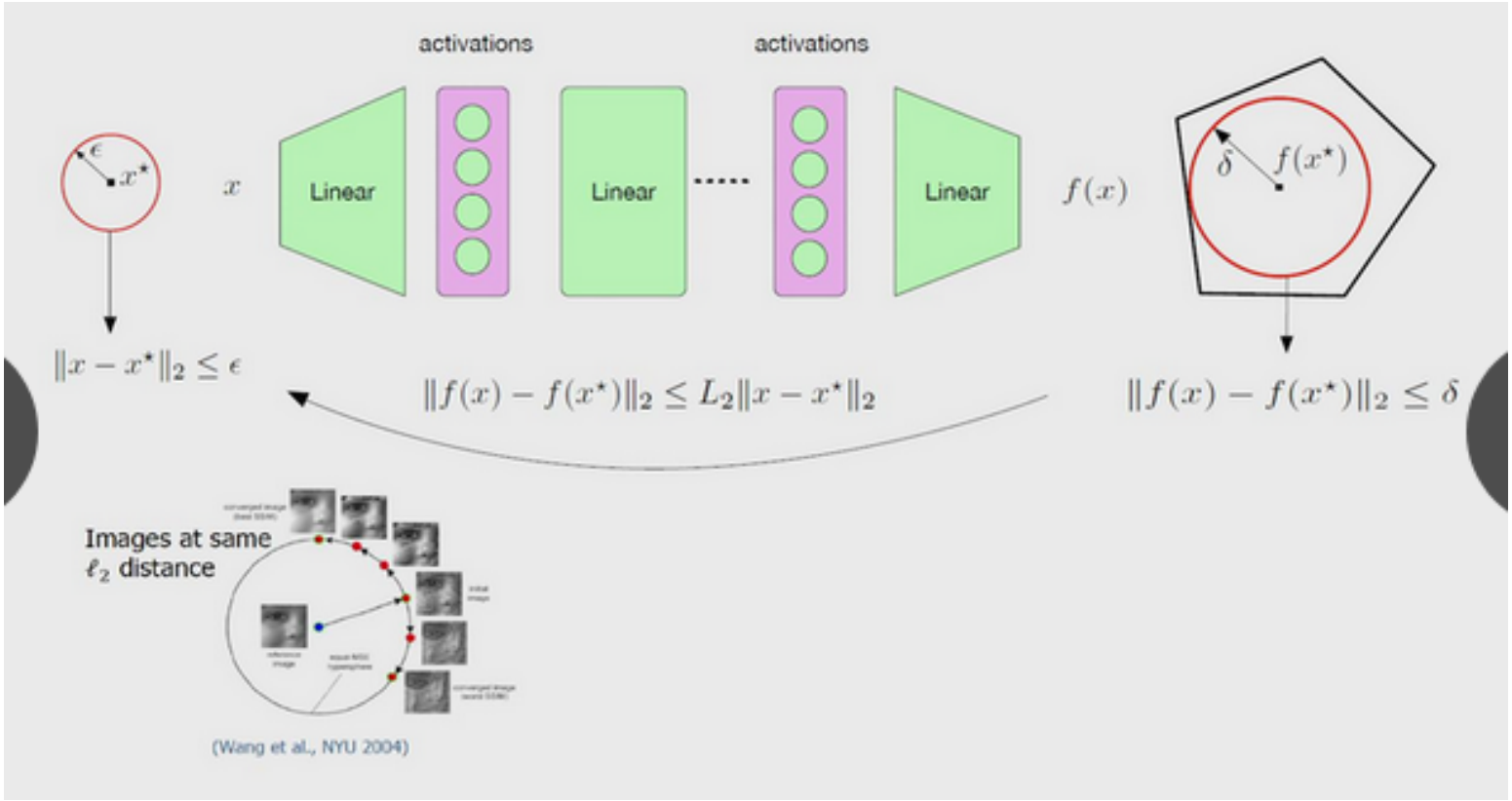


Image Source: Fei-Fei Li, Justin Johson, and Serena Yeung
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture08.pdf

NeurIPS2019 and Lipschitz continuity



Fazlyab, M., Robey, A., Hassani, H., Morari, M., & Pappas, G. J. (2019). Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks. arXiv preprint arXiv:1906.04893.