CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Deep Learning Essentials

7. Optimization

SGD, Momentum, RMSProp, Adam, ...

Lukáš Neumann

Adapted from B3B33UROB slides of Karel Zimmerman

Mean and Average

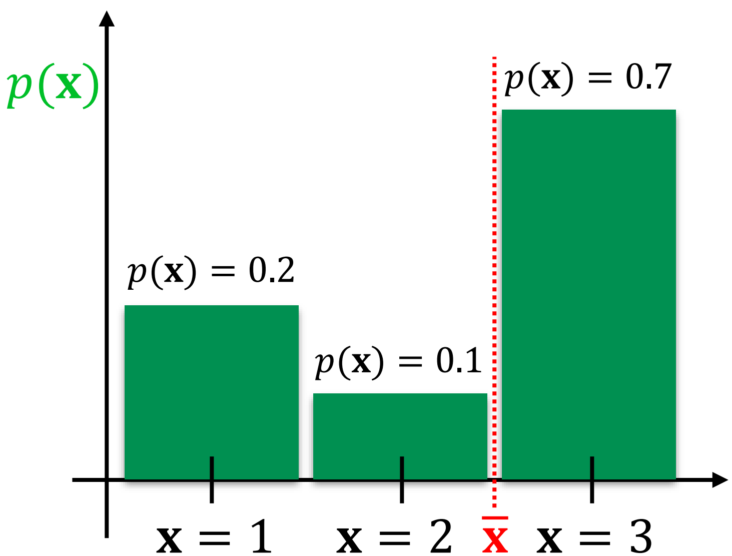
- Mean

$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\mathbf{x}] = 0.2 \cdot 1 + 0.1 \cdot 2 + 0.7 \cdot 3 = 2.5$$

- Average

$$\approx \frac{1}{N}\sum_i \mathbf{x}_i \quad = \frac{1}{10}(1+1+2+3+3+3+3+3+3+3) = 2.5$$

where $\mathbf{x}_i \sim p$



Mean and Average

- Mean

$$\bar{\mathbf{x}} = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot \mathbf{x} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\mathbf{x}] = 0.2 \cdot 1 + 0.1 \cdot 2 + 0.7 \cdot 3 = 2.5$$

- Average

$$\approx \frac{1}{N}\sum_i \mathbf{x}_i \quad = \frac{1}{10}(1+1+2+3+3+3+3+3+3+3) = 2.5$$

where $\mathbf{x}_i \sim p$

For $N \to \infty$

$$\mathcal{N}(\bar{\mathbf{x}}_i; \bar{\mathbf{x}}, \frac{\sigma_{\mathbf{x}}^2}{\sqrt{N}})$$

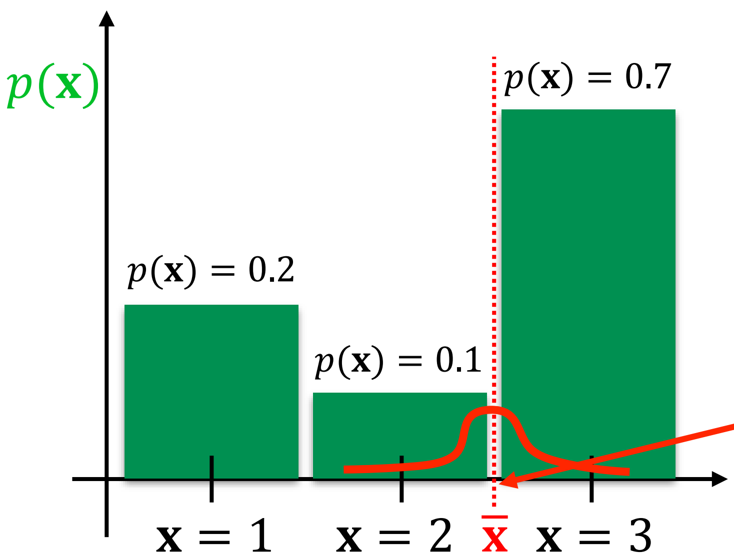
$$\bar{\mathbf{x}}_1 = \frac{1}{10}(1+1+1+1+3+3+3+3+3+3) = 2.2$$

$$\bar{\mathbf{x}}_2 = \frac{1}{10}(3+3+3+3+3+3+3+3+3+3) = 3.0$$

$$\bar{\mathbf{x}}_3 = \frac{1}{10}(2+2+2+2+3+3+3+3+3+3) = 2.6$$

$$\bar{\mathbf{x}}_4 = \frac{1}{10}(1+1+1+1+1+1+1+1+2+2) = 1.2$$

$$\bar{\mathbf{x}}_5 = \frac{1}{10}(1+1+1+1+1+3+3+3+3+3) = 2.0$$



Batches

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} D_{KL}(p_{\text{data}}(\mathbf{x}, y) \parallel p(\mathbf{x}, y|\mathbf{w})) = \arg\min_{\mathbf{w}} \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}[-\log p(y|\mathbf{x}, \mathbf{w})]$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}[\nabla_{\mathbf{w}} \log(p(y|\mathbf{x}, \mathbf{w}))]$$

True gradient
(we do not have access to)

$$\approx \frac{1}{M} \sum_i \nabla_{\mathbf{w}} \log(p(y_i|\mathbf{x}_i, \mathbf{w}))$$

Full gradient of the whole training set
(time-consuming estimation)

$$\approx \frac{1}{N} \sum_i \nabla_{\mathbf{w}} \log(p(y_i|\mathbf{x}_i, \mathbf{w}))$$

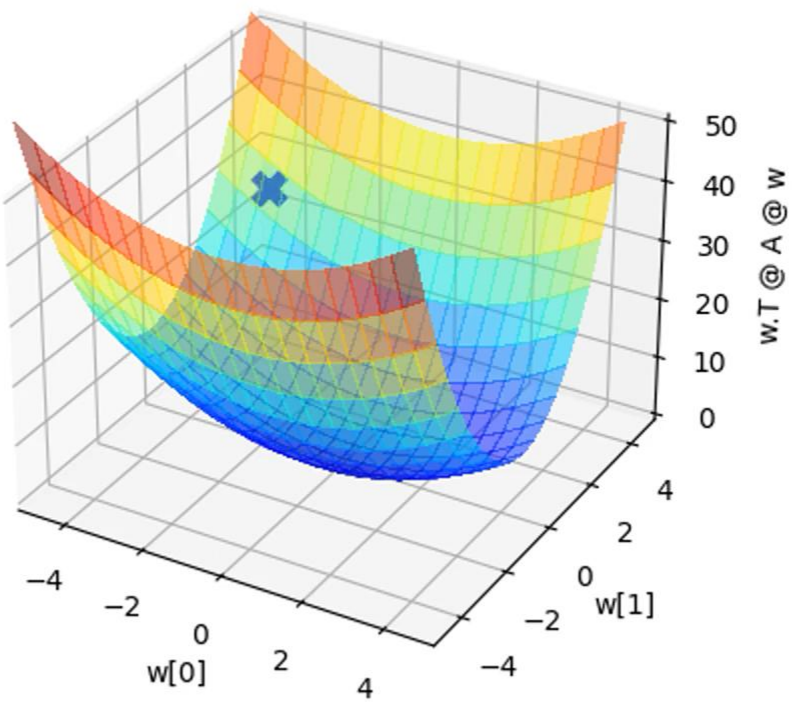
Gradient on a subset of the training set (**batch**)
(N<<M)

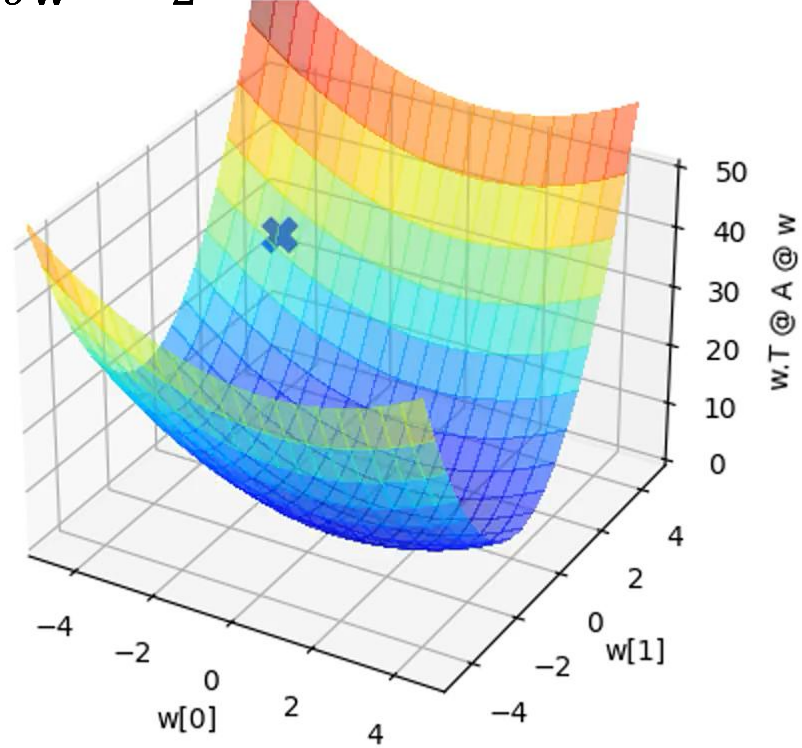
Batches

$$f(\mathbf{w}) = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A} (\mathbf{w} - \mathbf{w}_i)$$

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2 \cdot 1000} \sum_{i=1}^{1000} (\mathbf{w} - \boldsymbol{w}_i)^\top \mathbf{A}$$

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2} (\mathbf{w} - \mathbf{w}_i)^\top \mathbf{A}, i = \text{rand}(1{,}1000)$$





Batch size

- Is it worth to estimate the gradient from the whole training set?
- Standard error of the mean estimated from N samples is $\sigma/\sqrt{N}$, where σ^2 is true variance of input samples

- “Estimate of the gradient” based on $N = 10000$ vs $N = 100$
  - Standard error is $10 \times$ better
  - Computations are $100 \times$ slower !!!

- Using the large training set for estimating the gradient suffers from diminishing returns
- Convergence in the number of computations vs number of iterations

Batch size

- Large N => more accurate gradient with sub-linear returns
- Amount of required memory is linear in N

- GPU achieves better runtime with “power of 2” batch sizes
- Small batches yield regularization

- Use $N \in \{1,2,4,8,16,32,64,128,256,...\}$ or anything else that works and fits into your GPU ;-)

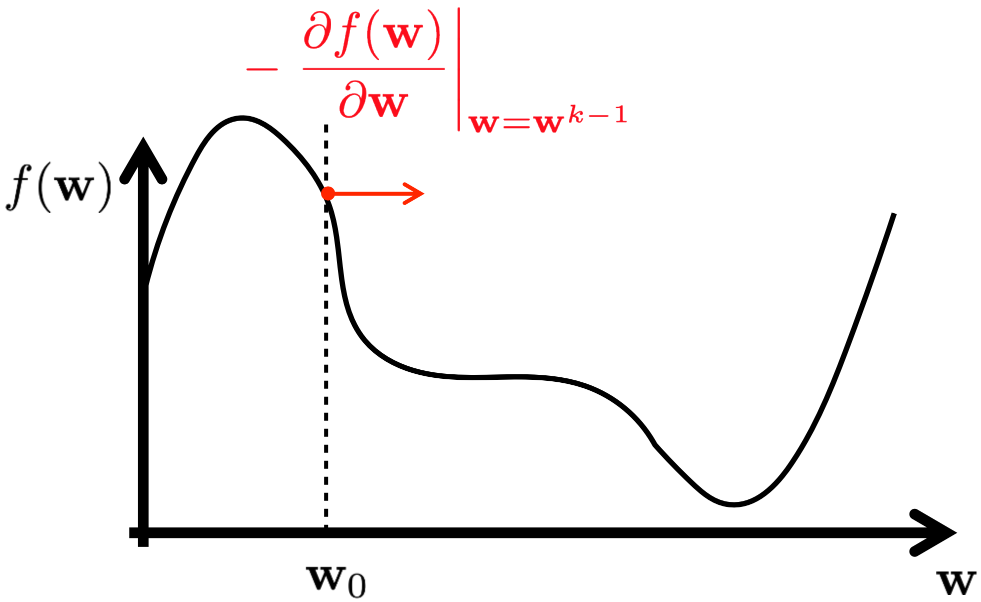
Stochastic Gradient Descent (SGD)

- Gradient Descent using randomized batches

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

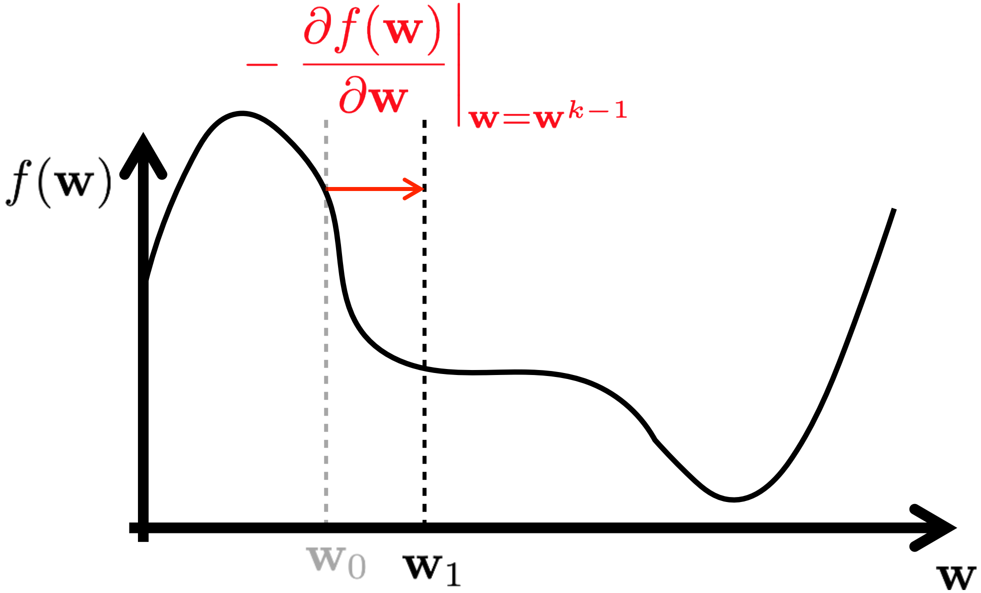
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



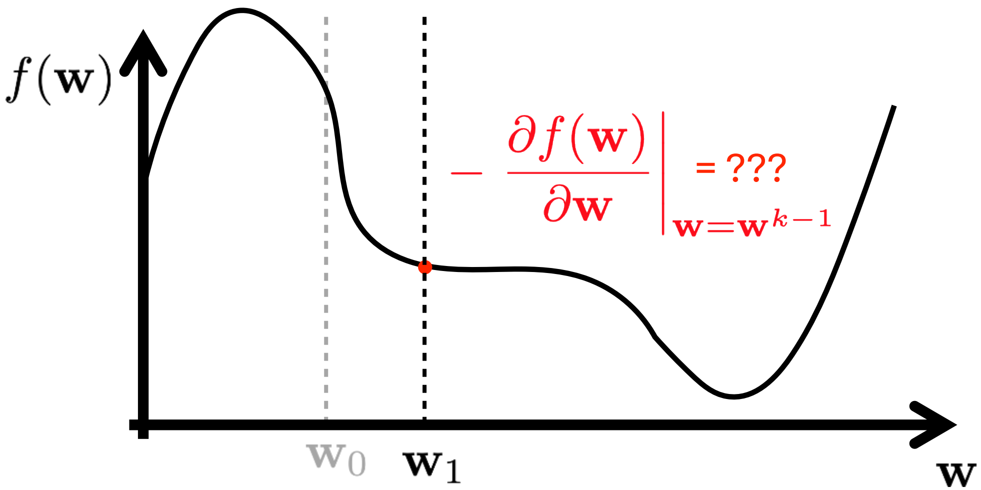
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



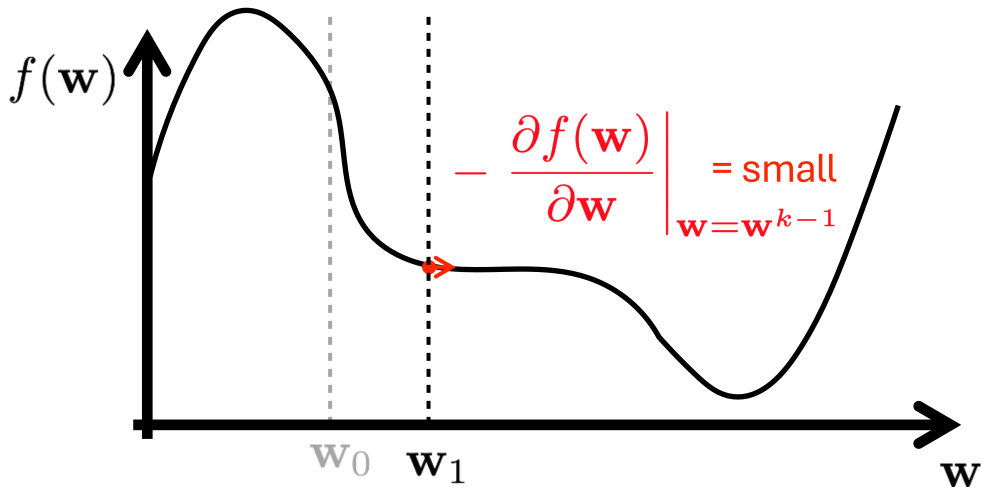
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



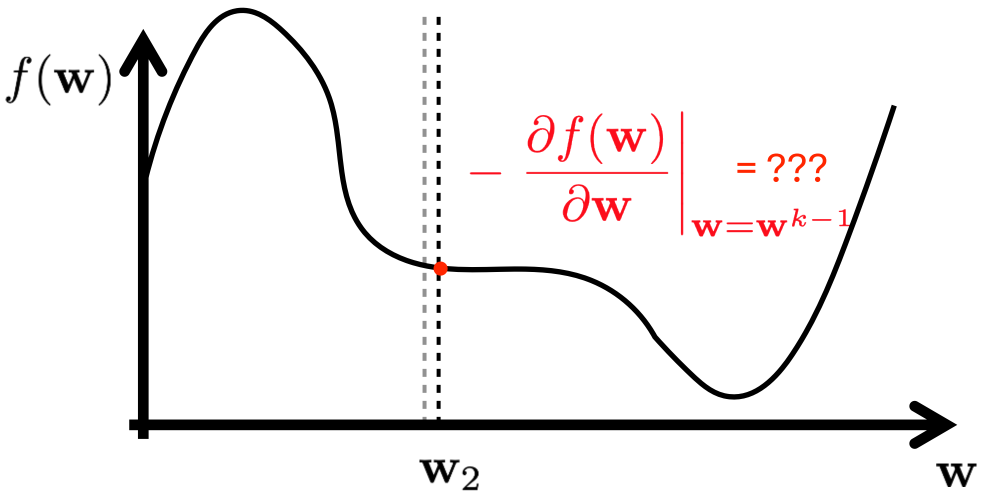
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



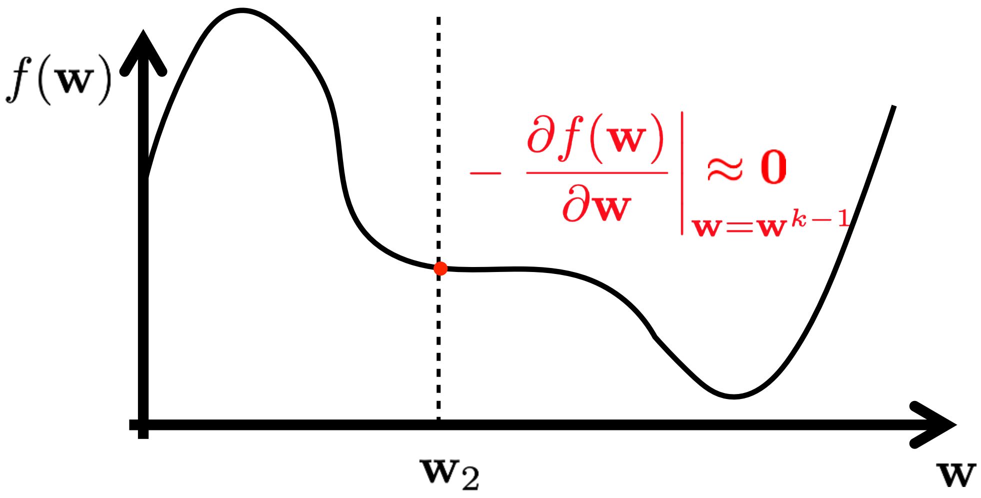
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



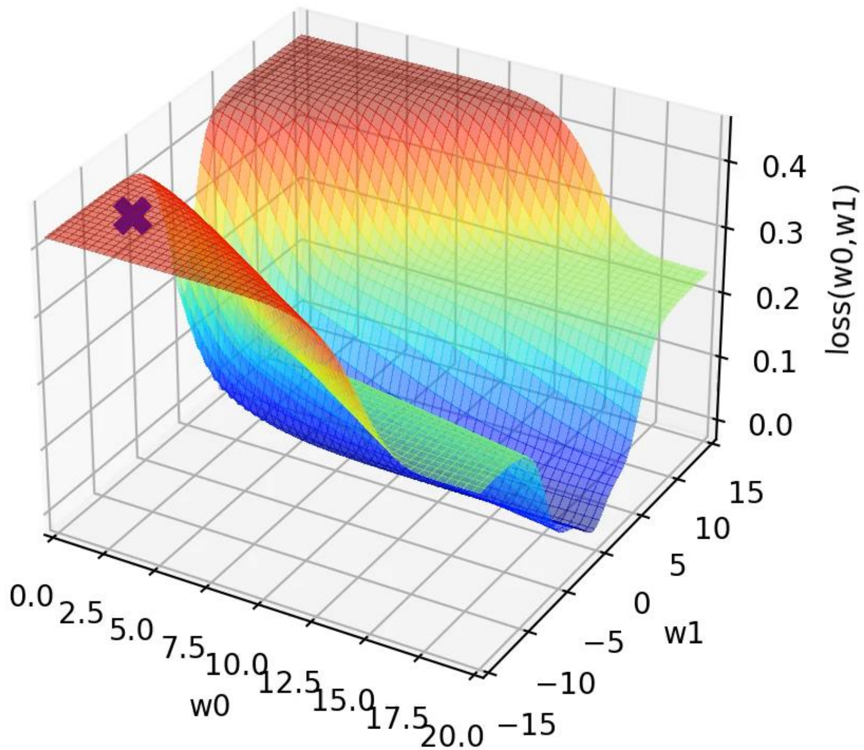
Stochastic Gradient Descent (SGD)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left. \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



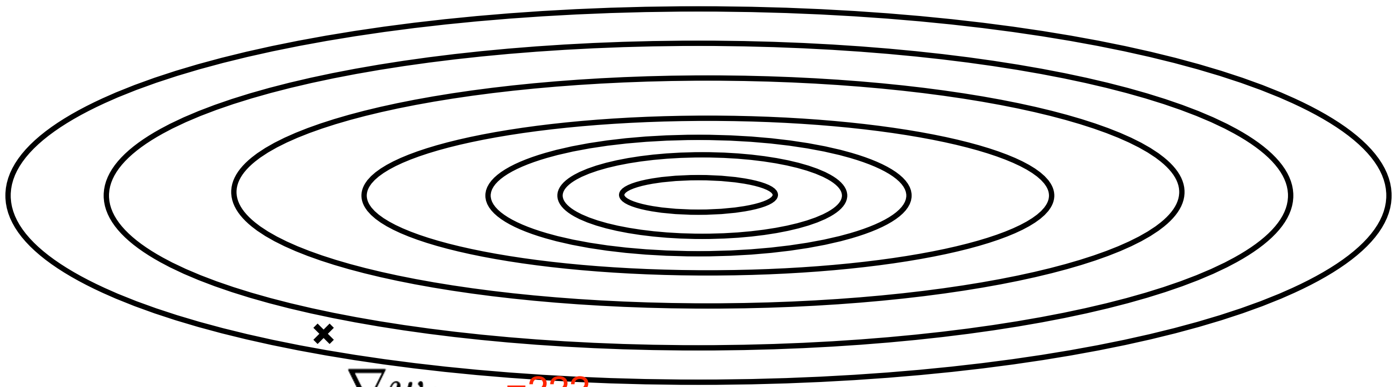
Stochastic Gradient Descent (SGD)

- Drawback: Slow convergence on plateaus (e.g. sigmoid fitting problem)



Stochastic Gradient Descent (SGD)

- Drawback: Oscillations



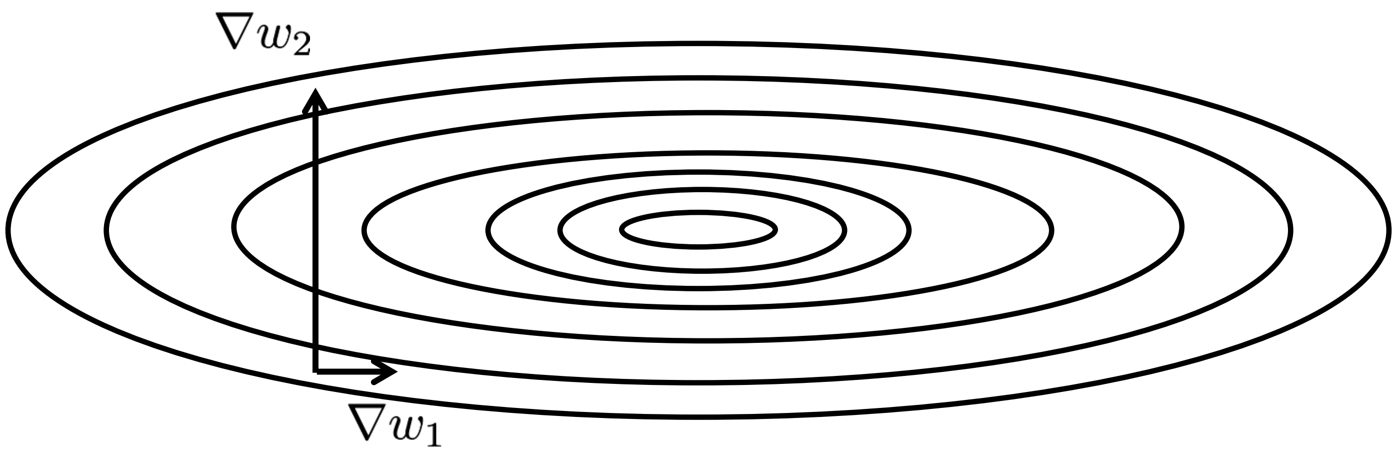
∇w_1 =???

∇w_2 =???

$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

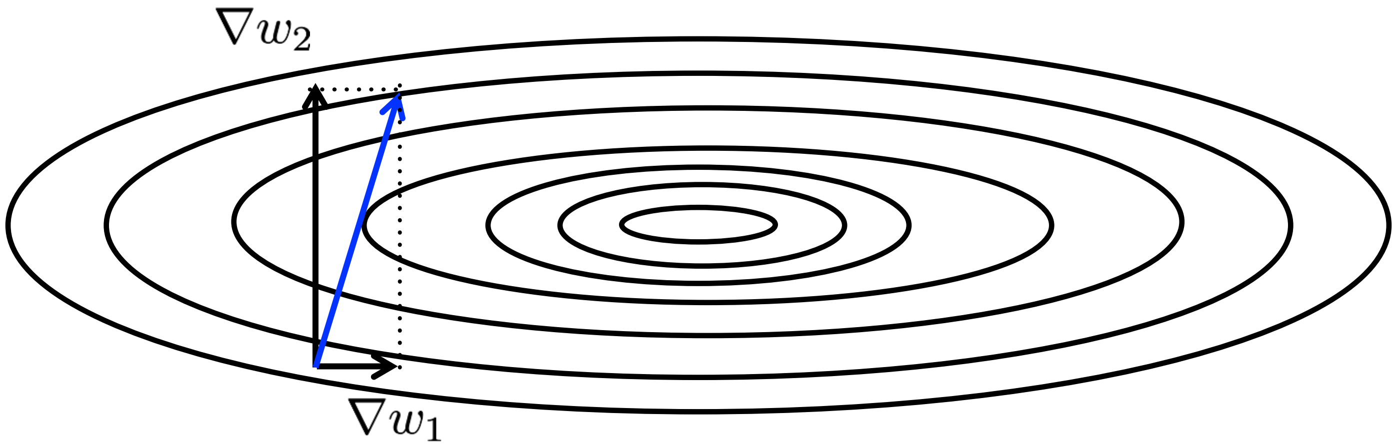
- Drawback: Oscillations



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

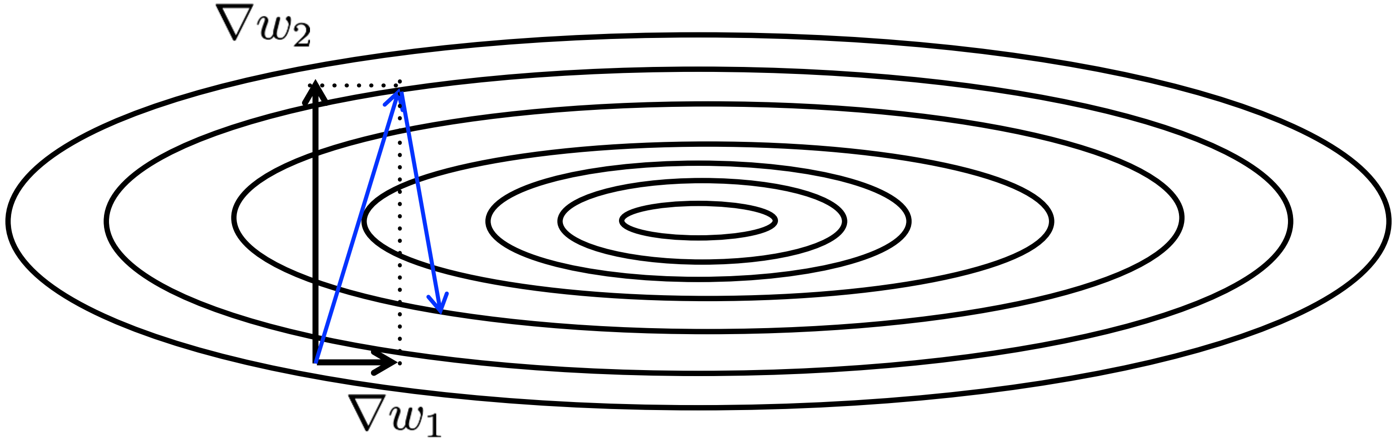
- Drawback: Oscillations



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

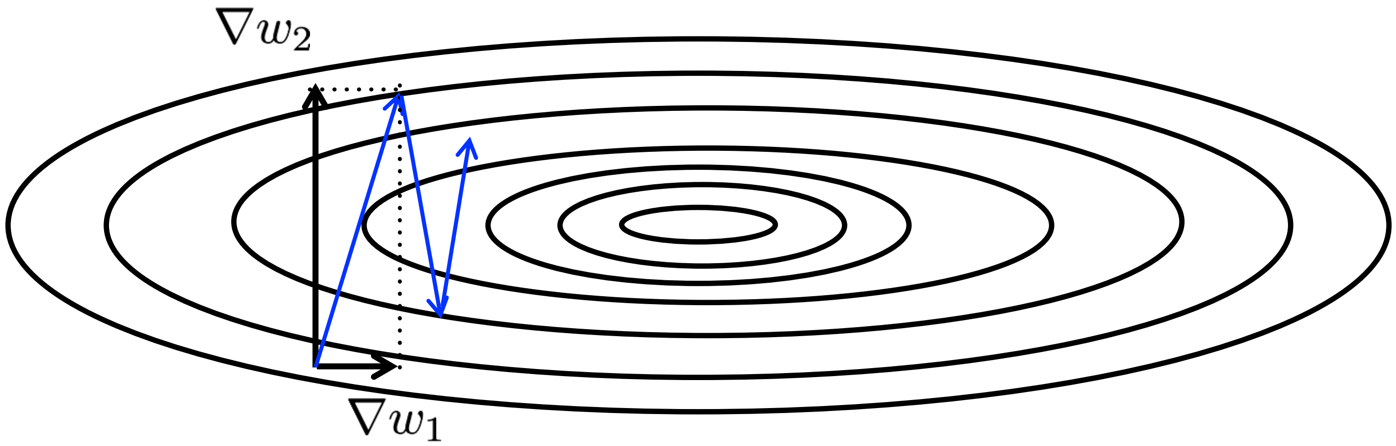
- Drawback: Oscillations



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

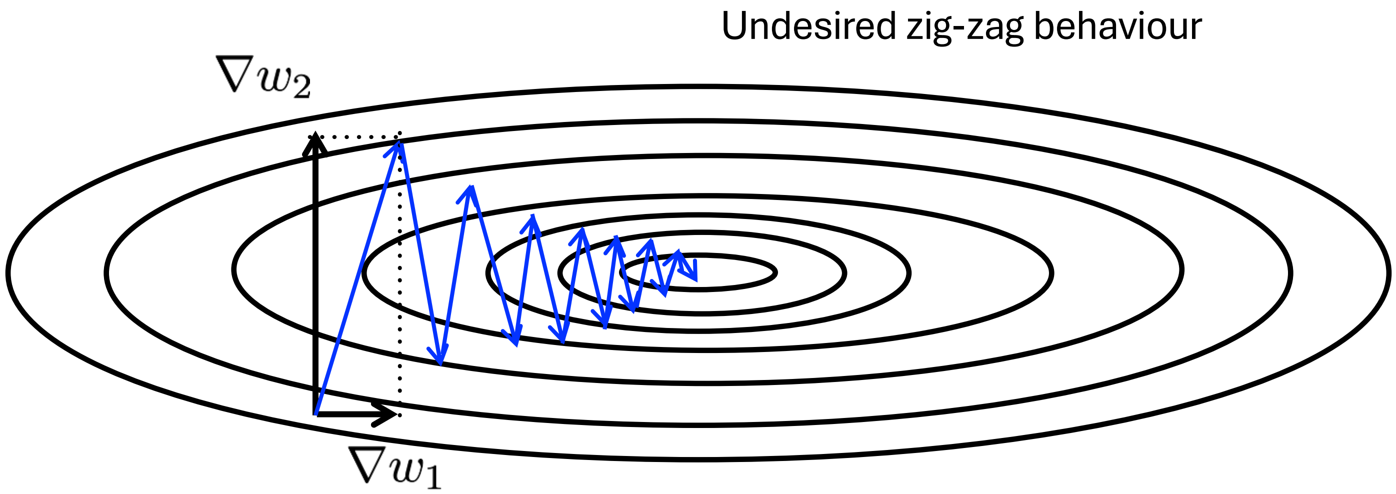
- Drawback: Oscillations



$$[\nabla w_1, \nabla w_2] = -\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

- Drawback: Oscillations



$$[\nabla w_1, \nabla w_2] = - \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

Stochastic Gradient Descent (SGD)

- Advantages
  - SGD is **faster** in term of computation time than GD
  - SGD does not get stuck in saddle-points as easy as GD
  - SGD yields **better generalization** due to inherent noise (similar to BN)

- Drawbacks
  - SGD **noisier** than GD (especially for small batch size)
  - SGD and GD **get stuck** on **flat** regions
  - SGD and GD **oscillate**

SGD with momentum

friction

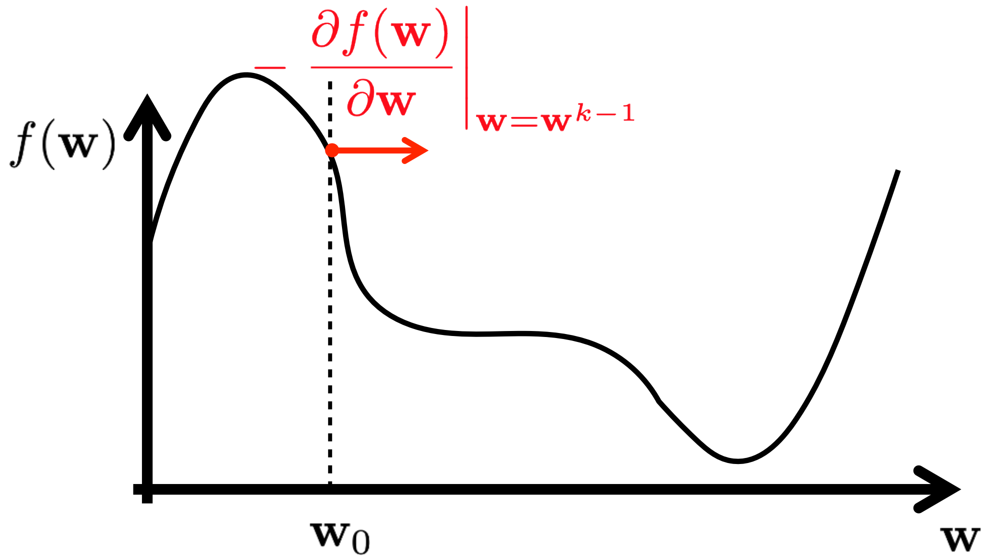
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

position

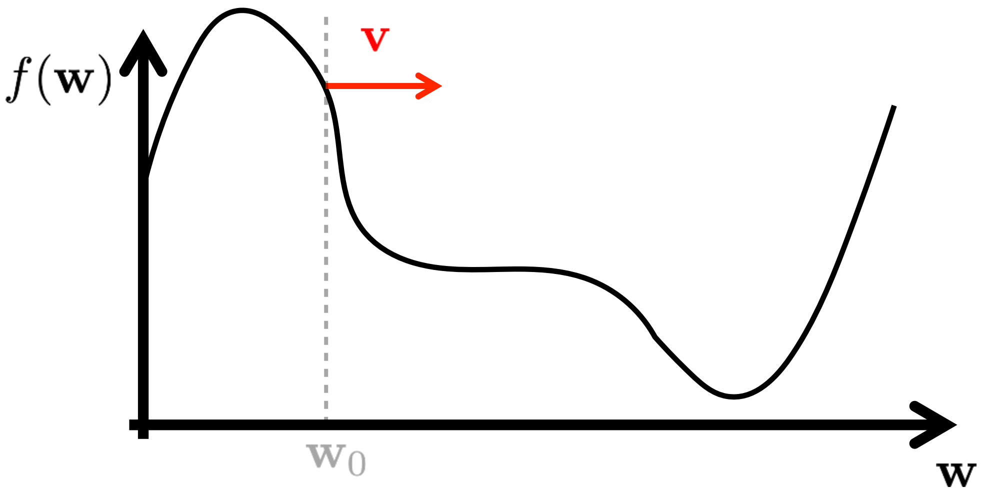
velocity

acceleration



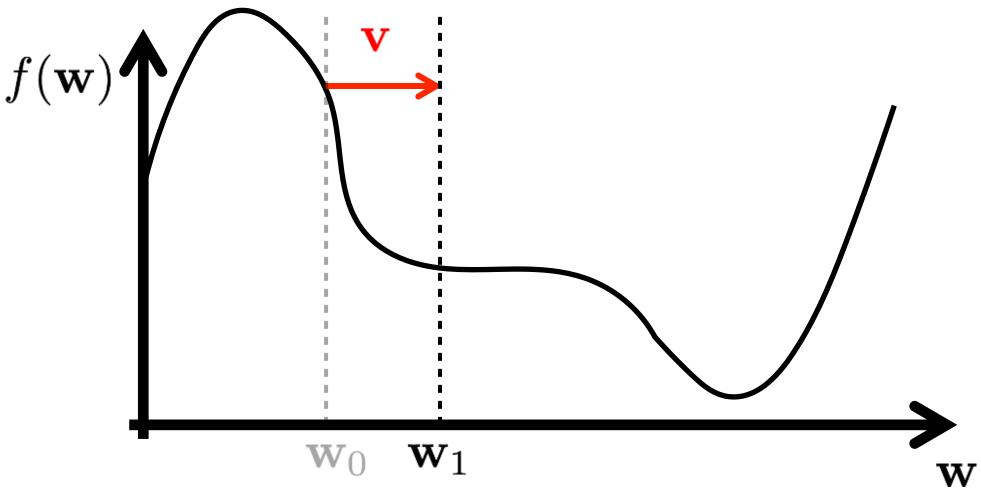
SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



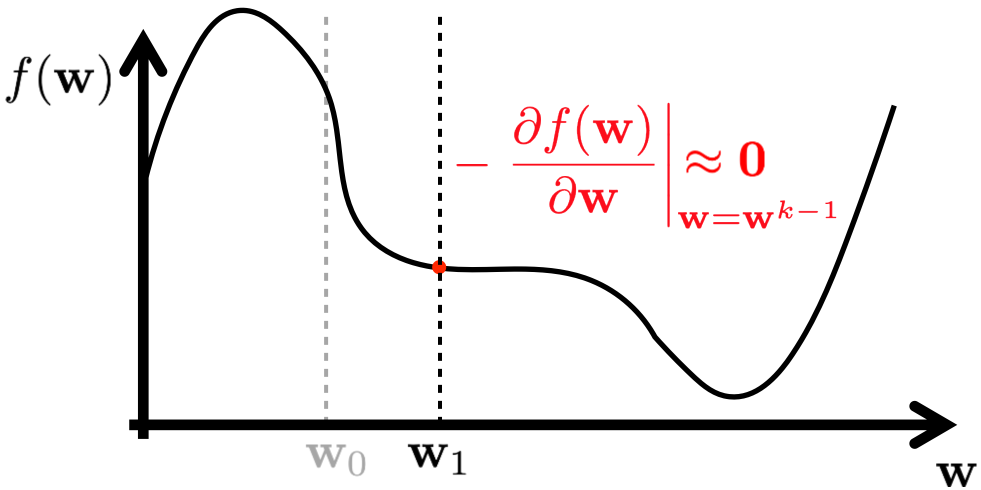
SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



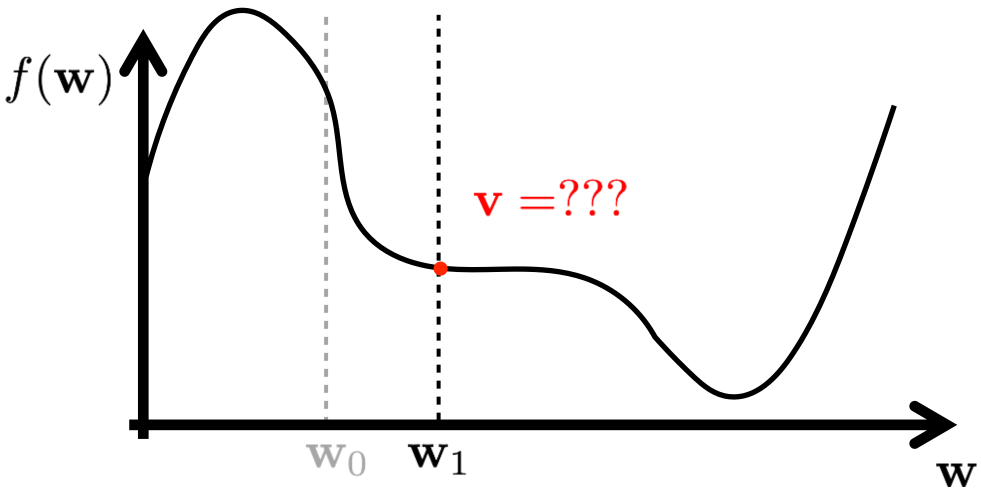
SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



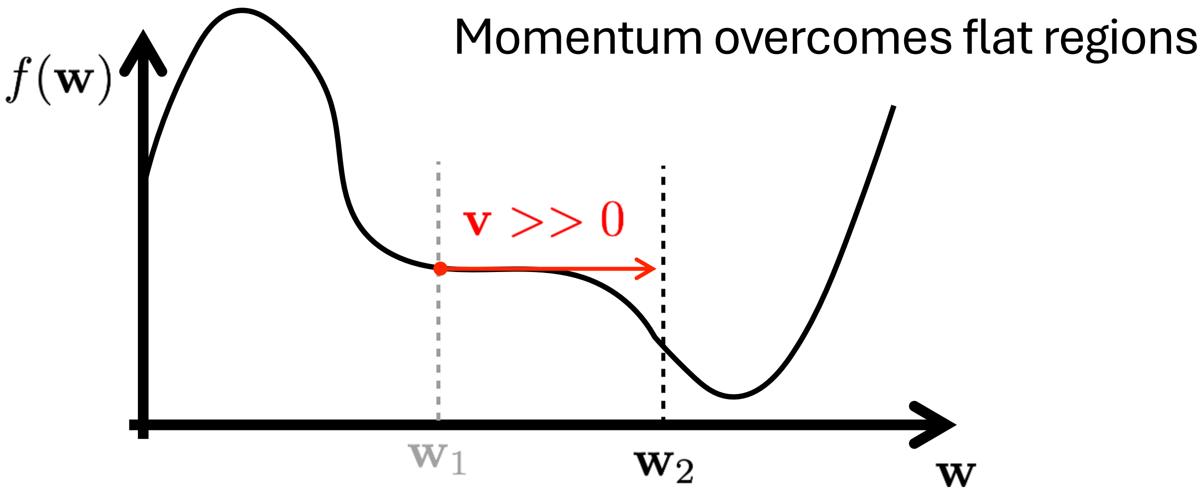
SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD with momentum

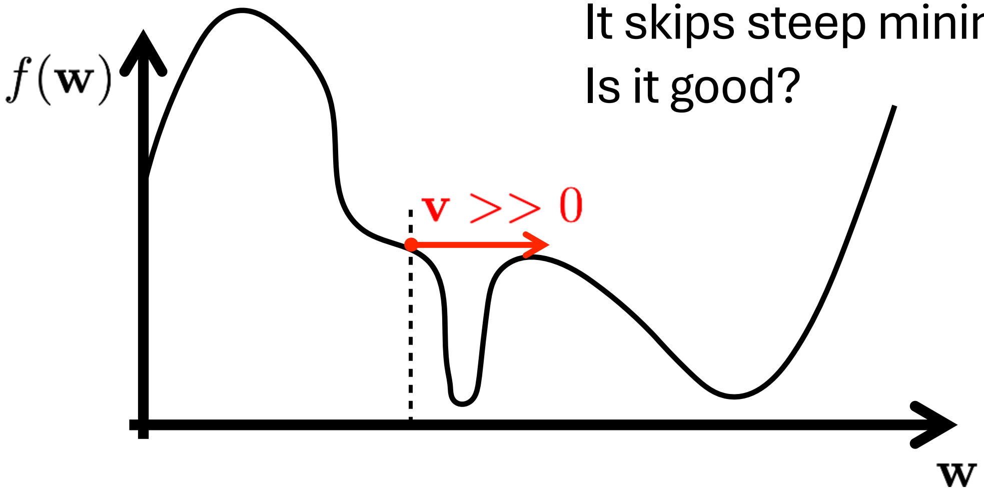
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



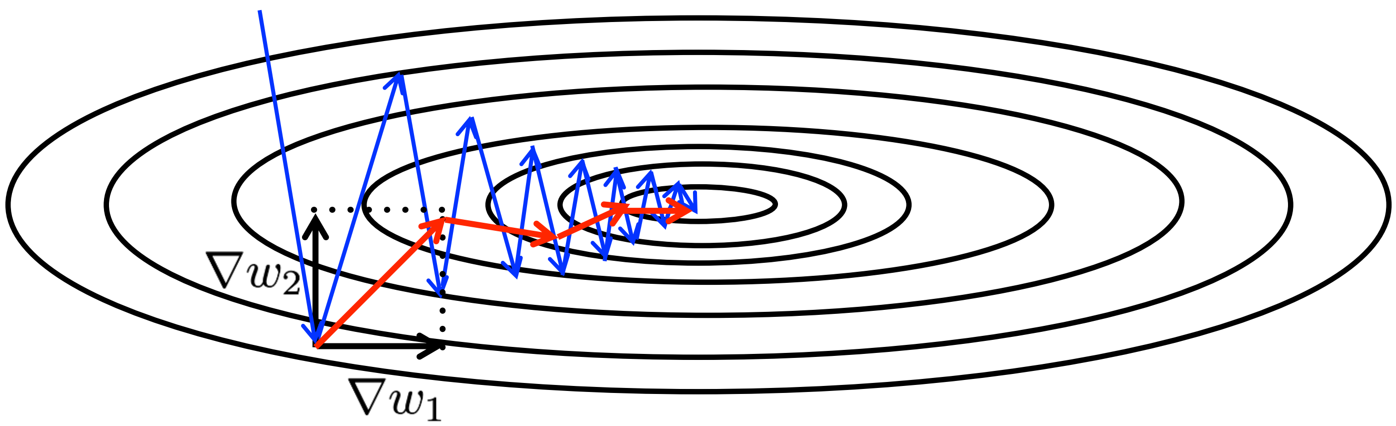
It skips steep minima.
Is it good?

SGD with momentum

- Momentum partially suppresses oscillations by averaging element-wise gradients

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

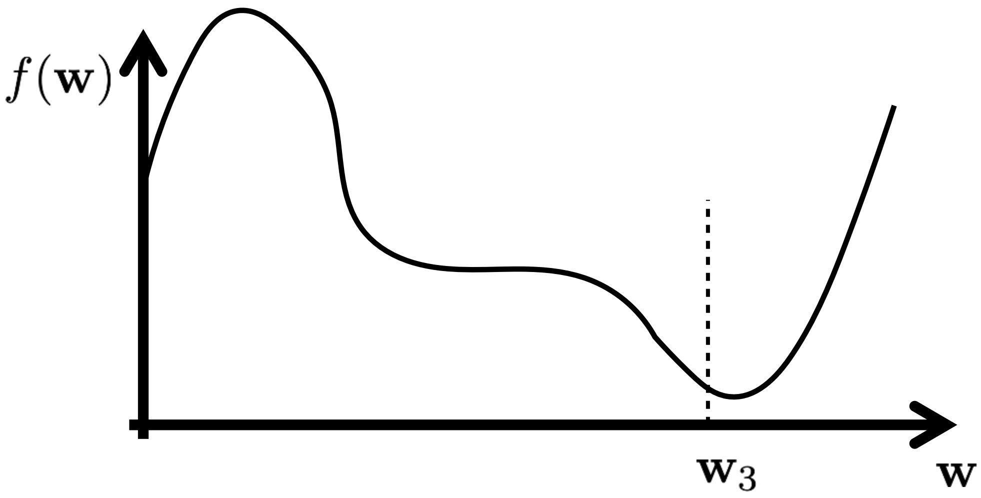
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



$$[\nabla w_1, \nabla w_2] = -\left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

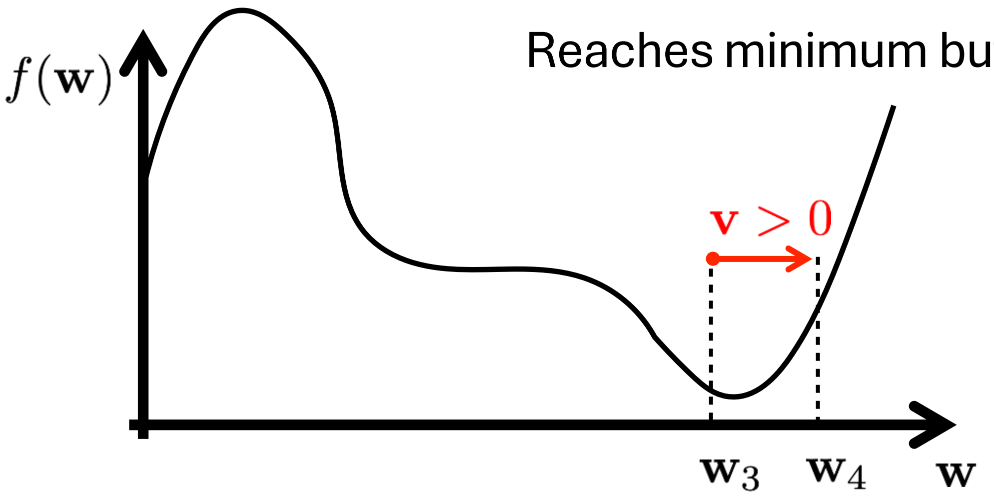
SGD with momentum

$$\mathbf{v}^k = \beta\mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha\mathbf{v}^k$$



SGD with momentum

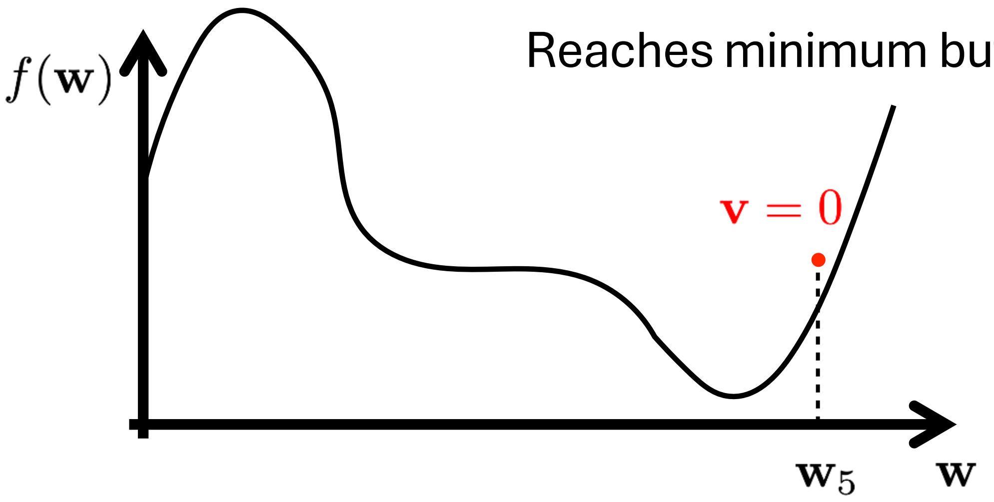
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



Reaches minimum but overshoots

SGD with momentum

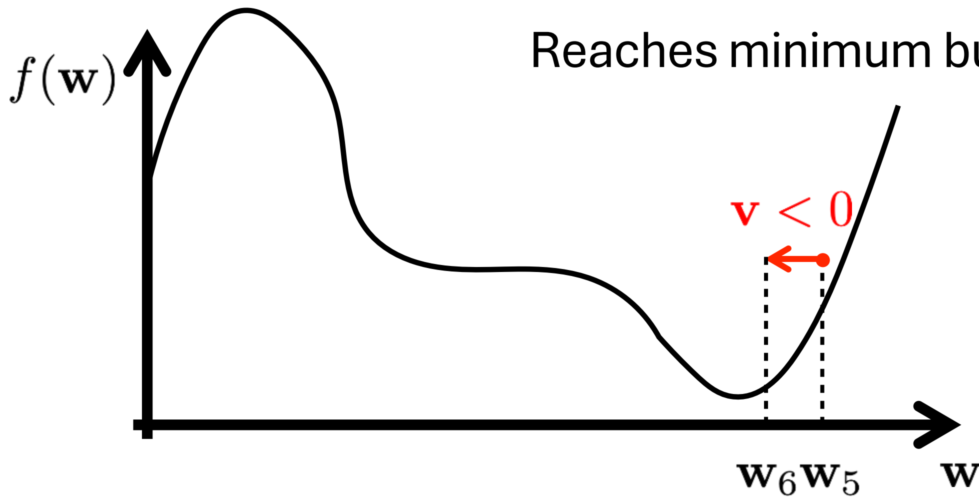
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



Reaches minimum but overshoots

SGD with momentum

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

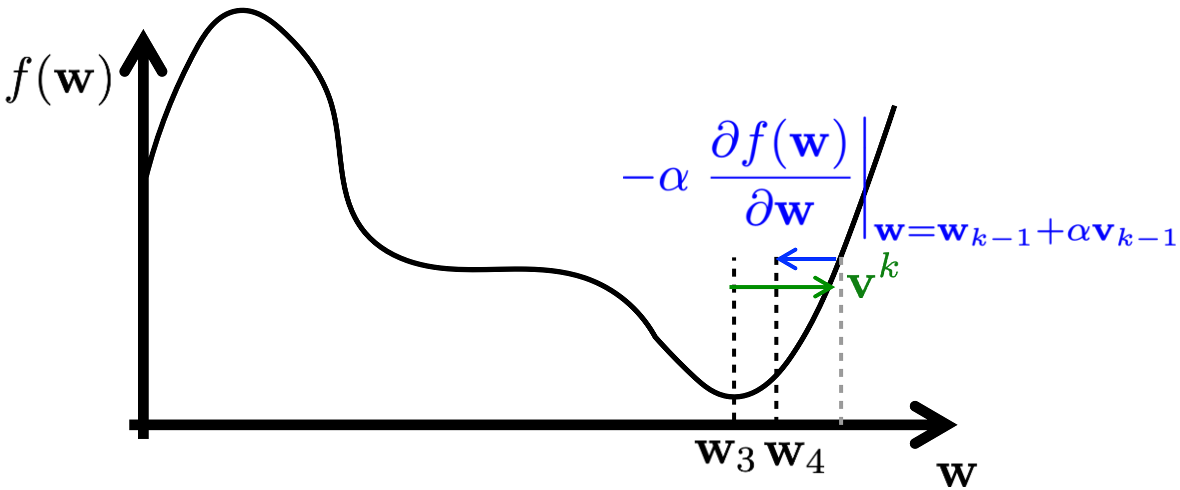


Reaches minimum but overshoots

SGD with Nesterov momentum

- Look one step ahead and reduce velocity by future gradient

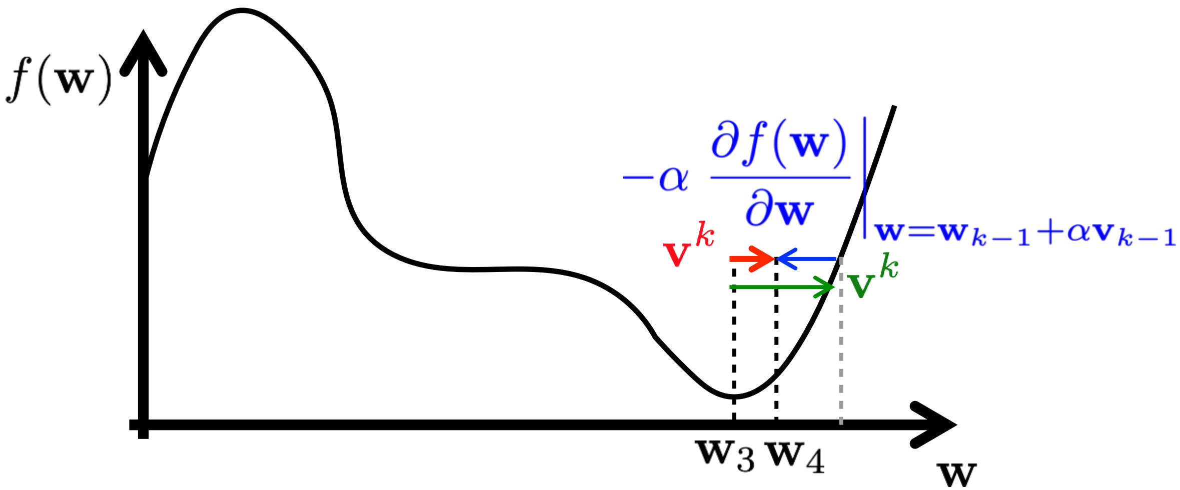
$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^{k-1}+\alpha \mathbf{v}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$



SGD with Nesterov momentum

- Look one step ahead and reduce velocity by future gradient

$$\mathbf{v}^k = \beta \mathbf{v}^{k-1} - \left.\frac{\partial f^\top(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}+\alpha\mathbf{v}^{k-1}}$$
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{v}^k$$

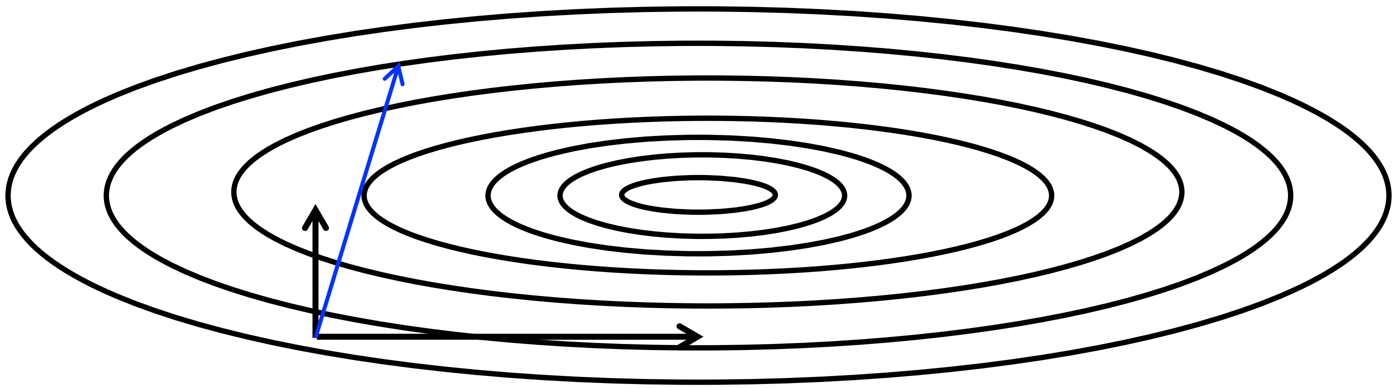


SGD with Nesterov momentum

- SGDM tends to overcome sharp minima and saddle-points due to momentum
- SGDM suppress oscillations by averaging out positive and negative gradients
- SGDM is less sensitive to large learning rates

Newton's method

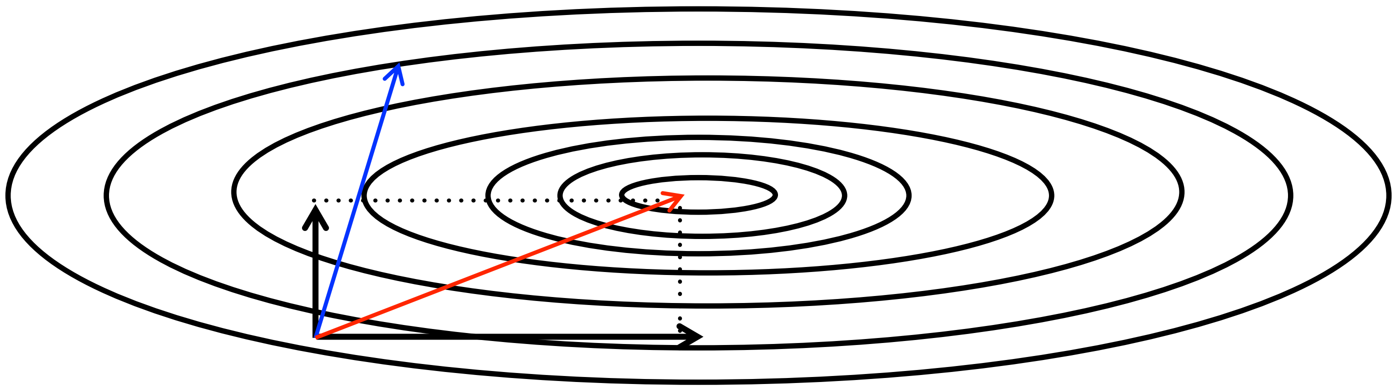
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1} \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



Hessian $H = \left. \frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ adjusts the direction of the gradient

Newton's method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1} \left. \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$



Hessian $H = \left. \frac{\partial^2 f(\mathbf{w})}{\partial^2 \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}^{k-1}}$ adjusts the direction of the gradient

Newton's method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \, H^{-1} \, \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

- **Why not to use the Hessian?**
  - Hessian has $M \times M$ elements for M-dimensional $\mathbf{w}$ → **memory-consuming**
  - Inverse of Hessian is $\mathcal{O}(M^3)$ → **time-consuming**
  - **Accurate** estimate of H^{-1} would require significantly **larger batches**

- **What does the Hessian actually do?**
  - It **slows down** each **component by** the amount corresponding the steepness in given direction
  - The faster the change, the shorter the step

Newton's method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\mathbf{g}$

Approximate Hessian as $\hat{H} \approx \mathrm{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}$

where $\mathbf{g} = \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$

Fisher information matrix

Newton's method

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\, H^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$\mathbf{g}$

Approximate Hessian as $\hat{H} \approx \operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}$

where $\mathbf{g} = \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$

Fisher information matrix

$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \alpha \left[\operatorname{diag}\left(\mathbf{g}\,\mathbf{g}^\top\right)^{1/2}\right]^{-1} \left.\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}^{k-1}}$$

$$\mathbf{w}^k \approx \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{g}^2 + \epsilon}} \odot \mathbf{g}$$

AdaGrad

$$\mathbf{q}^k = \mathbf{q}^{k-1} + \mathbf{g}^2$$

Gradient accumulation

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{g}$$

```
optimizer = torch.optim.Adagrad(params, lr=0.01, eps=1e-10)
```

RMSProp

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

Momentum on $\mathbf{g}^2$
(exponential averaging)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{g}$$

```
optimizer = torch.optim.RMSprop(params, lr=0.01, alpha=0.99)
```

Adam

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

Momentum on $\mathbf{g}$ (exponential averaging)

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

Momentum on $\mathbf{g}^2$ (exponential averaging)

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{v}^k$$

$\mathbf{v}^k, \mathbf{q}^k$ initialized in zero $\rightarrow$ slow start

Adam

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

Momentum on $\mathbf{g}$ (exponential averaging)

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

Momentum on $\mathbf{g}^2$ (exponential averaging)

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_1^k}$$

$$\hat{\mathbf{q}}^k = \frac{\mathbf{q}^k}{1-\beta_2^k}$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{v}^k$$

```
optimizer = torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999),
            eps=1e-08)
```

AdamW

$$\mathbf{v}^k = \beta_1 \mathbf{v}^{k-1} + (1-\beta_1)\mathbf{g}$$

Momentum on $\mathbf{g}$ (exponential averaging)

$$\mathbf{q}^k = \beta_2 \mathbf{q}^{k-1} + (1-\beta_2)\mathbf{g}^2$$

Momentum on $\mathbf{g}^2$ (exponential averaging)

$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_1^k}$$

$$\hat{\mathbf{q}}^k = \frac{\mathbf{q}^k}{1-\beta_2^k}$$

Correction of the slow start due to zero init.

$$\widehat{\mathbf{w}}^{k-1} = \mathbf{w}^{k-1} - \alpha\lambda\mathbf{w}^{k-1}$$

Decoupled weight-decay

$$\mathbf{w}^k = \widehat{\mathbf{w}}^{k-1} - \frac{\alpha}{\sqrt{\mathbf{q}^k} + \epsilon} \odot \mathbf{v}^k$$

```
optimizer = torch.optim.AdamW(params, lr=0.001, betas=(0.9, 0.999),
          eps=1e-08)
```

Optimizers summary

- **AdamW** is the most **popular** choice, since it is that **insensitive** to the choice **hyper-parameters**
- Anything **more complex** than AdamW typically **suffers** from diminishing improvements in **iteration quality** while **increasing** in computational **time**

- One more hack: If something is too big, you can always restrict it manually
  - Gradient clipping

```
for _, (inputs, labels) in enumerate(training_data):
    optimizer.zero_grad()                     # Zero gradients
    logits = model(inputs)                    # Make predictions

    loss = loss_fn(logits, labels)            # 1. Compute the loss
    loss.backward()                           # 2. Calculate gradients

    torch.nn.utils.clip_grad_norm_(
        model.parameters(), max_norm=1.0)

    optimizer.step()                          # 3. Update weights
```

Competencies gained for the test

- Batches
- Stochastic Gradient Descent (SGD), its pros and cons
- Momentum
- Second-order optimizers (Adam)