# Deep Learning

## Lecture 5 – Optimization

Sebastian Bordt
Theory of Machine Learning Group
University of Tübingen

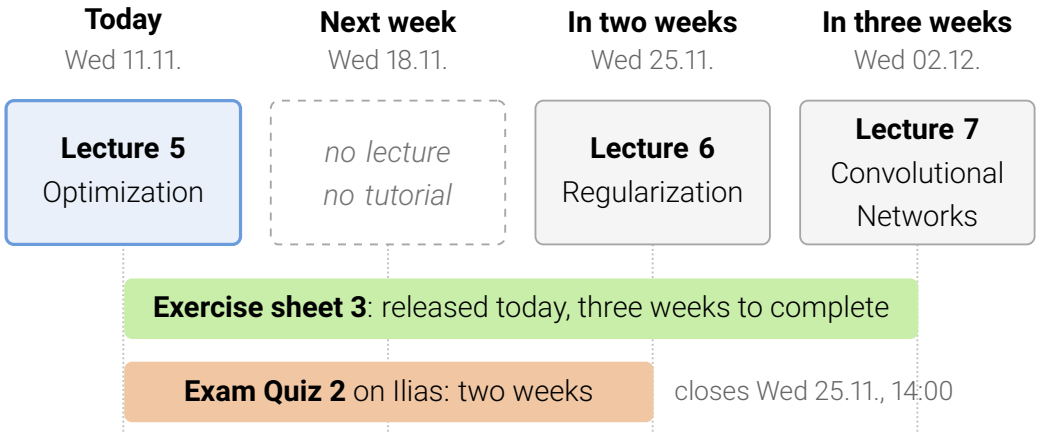








# Plan for the Next Weeks



# Agenda

**5.1** Optimization Challenges

**5.2** Optimization Algorithms

**5.3** Optimization Strategies

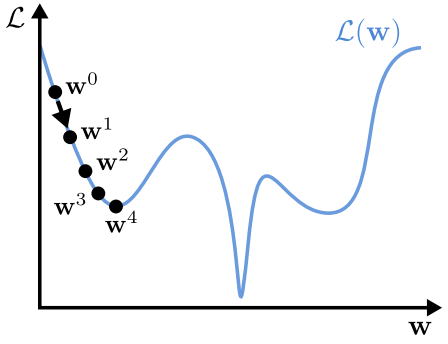
**5.4** Normalization Layers

# 5.1
# Optimization Challenges

# Optimization Challenges

**Gradient Descent:**

$$\mathbf{w}^0 = \mathbf{w}^{\text{init}}$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$$



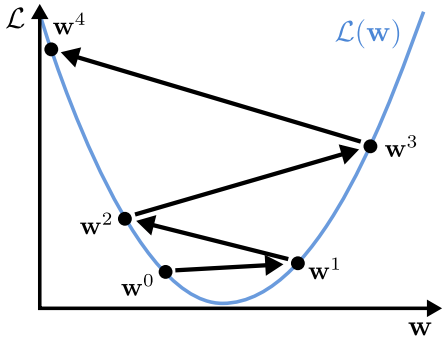

- Neural network loss $\mathcal{L}(\mathbf{w})$ is **not convex** wrt. the network parameters $\mathbf{w}$
- There exist **multiple local minima**, but we will find only one through optimization
- Example: we can permute all hidden units in a layer and get the **same solution**
- Good news: it is known that **many local minima** in deep networks **are good ones**

Choromanska, Henaff, Mathieu, Arous and LeCun: The Loss Surfaces of Multilayer Networks. AISTATS, 2015.

# Optimization Challenges

**Gradient Descent:**

$$\mathbf{w}^0 = \mathbf{w}^{\text{init}}$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$$

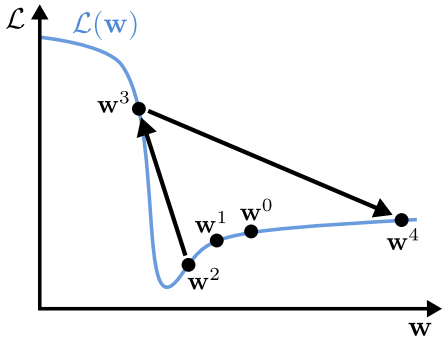



- Choosing the learning rate too low leads to very **slow progress**
- Choosing the learning rate too high might lead to **divergence**

# Optimization Challenges

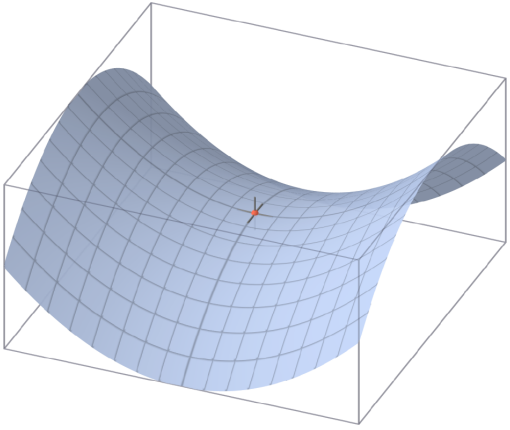
**Gradient Descent:**

$$\mathbf{w}^0 = \mathbf{w}^{\text{init}}$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$$



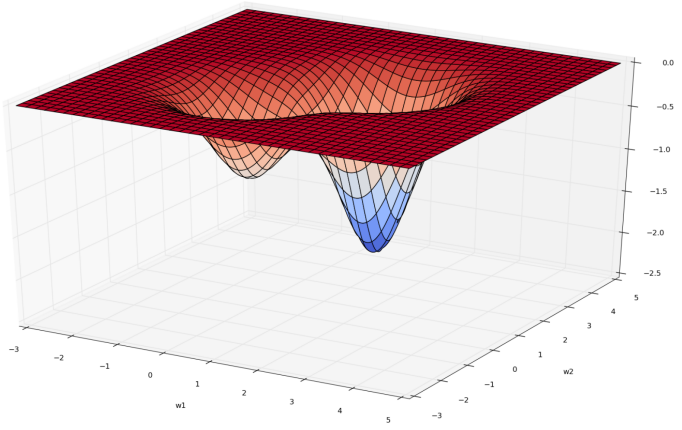

- Steep **cliffs** can pose great challenges to optimization
- Very high derivatives **catapult** the parameters $\mathbf{w}$ very far off
- **Gradient clipping** is a common heuristics to counteract such effects

# Optimization Challenges



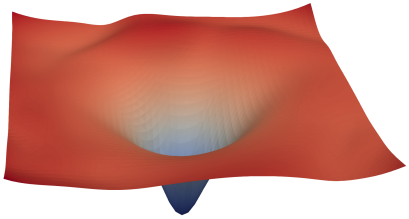
- At a **saddle point**, we have $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \mathbf{0}$, but we are not at a minimum
- Many saddle points in DL, but only problematic if we exactly “hit” the saddle point

# Optimization Challenges

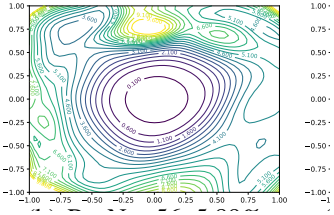



- A flat region is called a **plateau** where $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \approx \mathbf{0} \Rightarrow$ **Slow progress**
- Example: Saturated sigmoid activation function, dead ReLUs, etc.

# Loss Surface and Contour Plot



Loss surface (ResNet-56)



Contour plot (same network)

- The loss of a ResNet-56 on CIFAR-10 around its trained weights, as a surface and as a **contour plot**: lines of equal loss

# 5.2
# Optimization Algorithms

# Gradient Descent

**Algorithm:**

1. Initialize weights $\mathbf{w}^0$ and pick learning rate $\eta$
2. For all data points $i \in \{1, \dots, N\}$ do:
   2.1 Forward propagate $\mathbf{x}_i$ through network to calculate prediction $\hat{\mathbf{y}}_i$
   2.2 Backpropagate to obtain gradient $\nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^t) \equiv \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i, \mathbf{w}^t)$
3. Update gradients: $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{N} \sum_i \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w}^t)$
4. If validation error decreases, go to step 2, otherwise stop

**Remark:**

- Typically, millions of parameter $\Rightarrow \dim(\mathbf{w}) = 1$ million or more
- Typically, millions of training points $\Rightarrow N = 1$ million or more
- Becomes extremely expensive to compute and doesn't fit into memory

# Stochastic Gradient Descent

# Stochastic Gradient Descent

**Solution:**

- The total loss over the entire training set can be expressed as the expectation:

$$\frac{1}{N}\sum_i \mathcal{L}_i(\mathbf{w}^t) = \mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\mathcal{L}_i(\mathbf{w}^t)\right]$$

- This expectation can be approximated by a smaller subset $B \ll N$ of the data:

$$\mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\mathcal{L}_i(\mathbf{w}^t)\right] \approx \frac{1}{B}\sum_b \mathcal{L}_b(\mathbf{w}^t)$$

- Thus, the gradient can also be approximated:

$$\frac{1}{N}\sum_i \nabla_{\mathbf{w}}\mathcal{L}_i(\mathbf{w}^t) = \mathbb{E}_{i\sim\mathcal{U}\{1,N\}}\left[\nabla_{\mathbf{w}}\mathcal{L}_i(\mathbf{w}^t)\right] \approx \frac{1}{B}\sum_b \nabla_{\mathbf{w}}\mathcal{L}_b(\mathbf{w}^t)$$

# Stochastic Gradient Descent

**Remarks:**

- We call $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_B, \mathbf{y}_B)\} \subseteq \mathcal{X}$ a "minibatch"
- You should choose $B$ as large as your (GPU) memory allows [Goyal et al., 2017]
- Typically $B \ll N$, e.g., $B = 8, 16, 32, 64, 128$
- Smaller batch sizes lead to larger variance in the gradients (noisy updates)
- Batches can be chosen randomly or by partitioning the dataset
  (but they should be as independent as possible $\Rightarrow$ shuffle training set)

**Terminology:**

- Iteration = a single gradient update based on a single minibatch $\mathbf{w}^t \rightarrow \mathbf{w}^{t+1}$
- Epoch = complete pass through the training set ($= \frac{N}{B}$ iterations)

Goyal et al.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. Arxiv, 2017.

# Stochastic Gradient Descent

**Algorithm:**

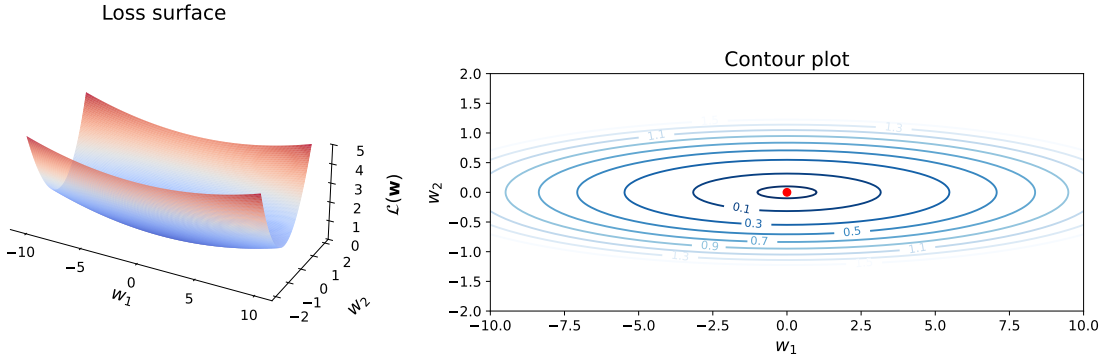
1. Initialize weights $\mathbf{w}^0$, pick learning rate $\eta$ and minibatch size $|\mathcal{X}_{\text{batch}}|$
2. Draw random minibatch $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_B, \mathbf{y}_B)\} \subseteq \mathcal{X}$ (with $B \ll N$)
3. For all minibatch elements $b \in \{1, \ldots, B\}$ do:
   3.1 Forward propagate $\mathbf{x}_b$ through network to calculate prediction $\hat{\mathbf{y}}_b$
   3.2 Backpropagate to obtain batch element gradient $\nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t) \equiv \nabla_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}_b, \mathbf{y}_b, \mathbf{w}^t)$
4. Update gradients: $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$
5. If validation error decreases, go to step 2, otherwise stop

**Remark:**

- Allows for training with limited (GPU) memory, by splitting data into chunks
- Introduces stochasticity, batch gradients approximate the true gradient

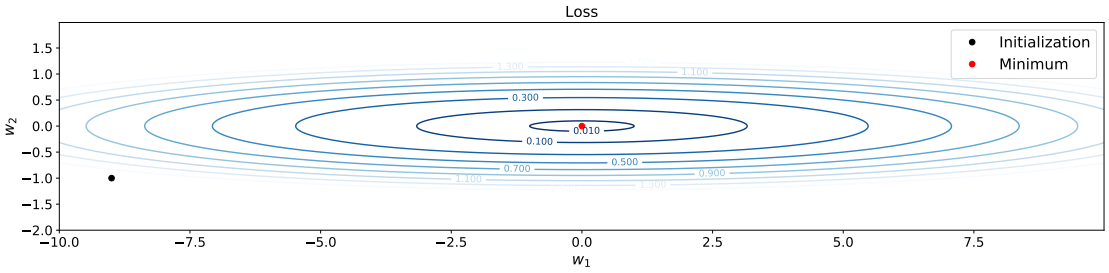
# Example

# Example: Loss Surface and Contour Plot




- Toy example for this chapter: $\mathcal{L}(\mathbf{w}) = (0.1\, w_1)^2 + w_2^2$, a quadratic bowl
- Non-isotropic: steep in the $w_2$ direction, 100 times flatter in the $w_1$ direction

# Stochastic Gradient Descent

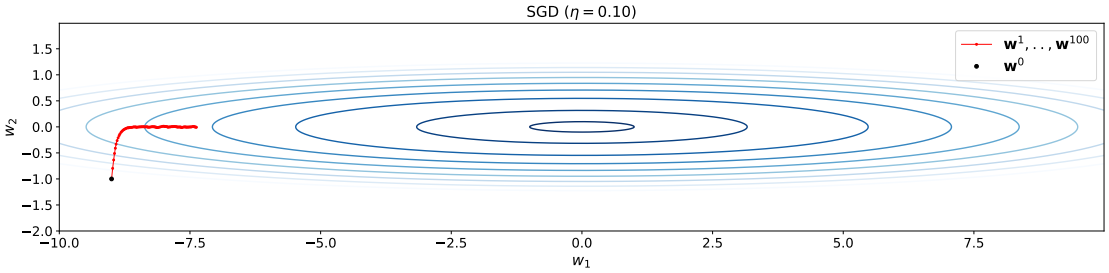



$$\mathcal{L}(\mathbf{w}) = (0.1\, w_1)^2 + w_2^2$$

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \begin{pmatrix} 0.02\, w_1 & 2 w_2 \end{pmatrix}^\top + \mathcal{N}(0, 0.03)$$

- Toy example: Loss is 2D non-isotropic quadratic of parameter $\mathbf{w} = (w_1, w_2)^\top$
- To simulate minibatches, we have added Gaussian noise to the gradient

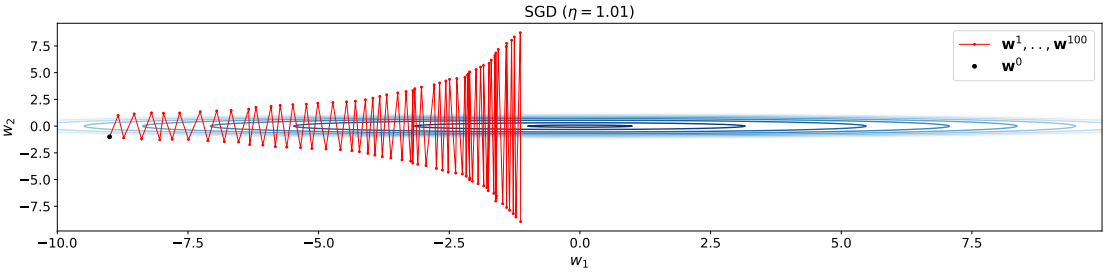
# Stochastic Gradient Descent




$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

- Choosing the learning rate $\eta$ too small does not lead to convergence (in 100 steps)

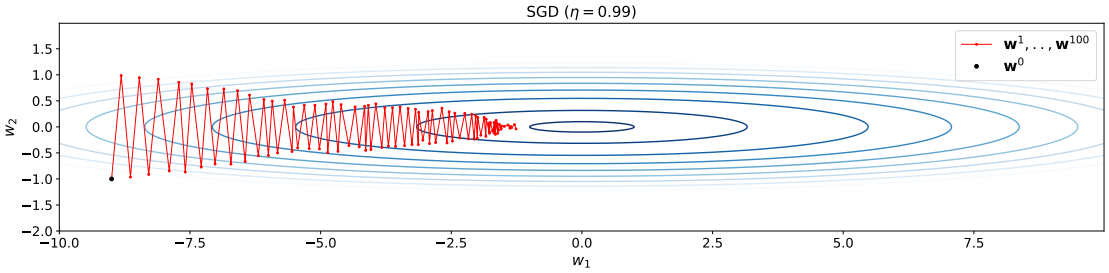
# Stochastic Gradient Descent




$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

- Choosing the learning rate $\eta$ too high leads to oscillations (also no convergence)

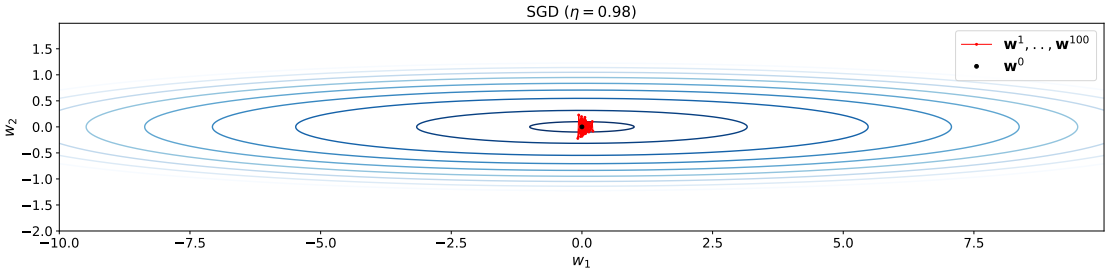
# Stochastic Gradient Descent




$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

- A good learning rate $\eta$ works better, but still slow, inefficient and no convergence

# Convergence of SGD




$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum_b \nabla_{\mathbf{w}} \mathcal{L}_b(\mathbf{w}^t)$$

- Remark: Due to stochasticity, a fixed learning rate $\eta$ will never lead to convergence
- In this example, the weights have been initialized at the optimum: $\mathbf{w}^0 = \mathbf{0}$

# Convergence of SGD

The SGD update leads to the following **sequence of parameters:**

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^1 = \mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0$$
$$\mathbf{w}^2 = \mathbf{w}^1 - \eta \nabla_{\mathbf{w}} \mathcal{L}_1 = \mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_1$$
$$\mathbf{w}^3 = \mathbf{w}^2 - \eta \nabla_{\mathbf{w}} \mathcal{L}_2 = \mathbf{w}^0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_0 - \eta \nabla_{\mathbf{w}} \mathcal{L}_1 - \eta \nabla_{\mathbf{w}} \mathcal{L}_2$$

Optimization **converges** if there exists a vector $\mathbf{w}^*$ such that for every arbitrarily small positive number $\epsilon$, there exists an integer $T$ such that for all $t \geq T$:

$$\|\mathbf{w}^t - \mathbf{w}^*\| < \epsilon$$

# Convergence of SGD in the Convex Case

**Theorem**

Let $\mathcal{L}(\mathbf{w})$ be convex and smooth and let $(\eta_1, \eta_2, \dots)$ be positive step sizes with

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

and let $\mathbf{g}_t$ be an unbiased estimate of $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$ with bounded variance.

Then, the sequence

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \mathbf{g}_t \qquad t \to \infty$$

converges to a global minimum of $\mathcal{L}(\mathbf{w})$ [Garrigos and Gower, 2023]. Example: $\eta_t = \frac{\eta}{t}$

Robbins, Monro: A Stochastic Approximation Method. 1951. · Garrigos, Gower: Handbook of Convergence Theorems for (Stochastic) Gradient Methods. 2023.

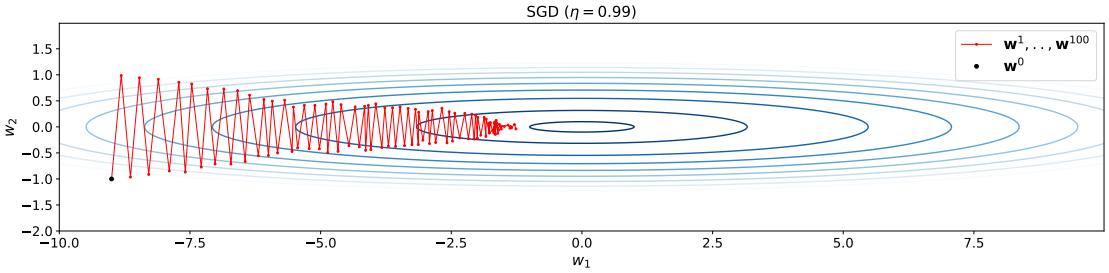
# Problems of SGD

**Problems of SGD:**

- Gradient scaled equally across all dimensions
- Requires conservative learning rate to avoid divergence
- However, in this case the updates become very small $\Rightarrow$ slow progress
- Finding a good learning rate is difficult

# SGD with Momentum

# SGD with Momentum

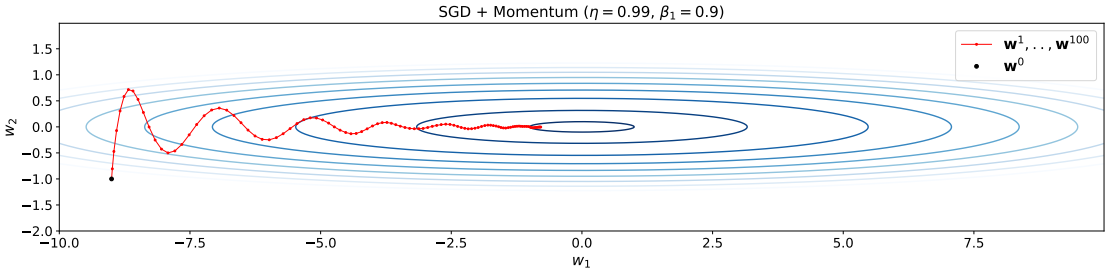



**Motivation for SGD with Momentum:**

- SGD oscillates along $w_2$ axis $\Rightarrow$ we should dampen, e.g., by averaging over time
- SGD makes slow progress along $w_1$ axis $\Rightarrow$ we like to accelerate in this direction
- Idea of momentum: update weights with exponential moving average of gradients

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# SGD with Momentum




$$\mathbf{m}^{t+1} = \beta_1 \, \mathbf{m}^t - \eta \, \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{m}^{t+1}$$

- With velocity $\mathbf{m}$ and momentum $\beta_1$, typically $\beta_1 = 0.9$

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# SGD with Momentum

Traditionally, momentum is introduced as

$$\mathbf{m}^{t+1} = \beta_1 \, \mathbf{m}^t - \eta \, \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{m}^{t+1}$$

However, this expression **couples** the momentum $\beta_1$ and learning rate $\eta$ hyperparameters. A better parameterization is the following decoupled one:

$$\mathbf{m}^{t+1} = \beta_1 \, \mathbf{m}^t + (1 - \beta_1) \, \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \, \mathbf{m}^{t+1}$$

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# Exponential Moving Average

Let us abbreviate the gradient at iteration $t$ with $\mathbf{g}_t \equiv \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$. We have:

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1-\beta_1)\mathbf{g}_t \quad (\text{with } \mathbf{m}^0 = \mathbf{0})$$

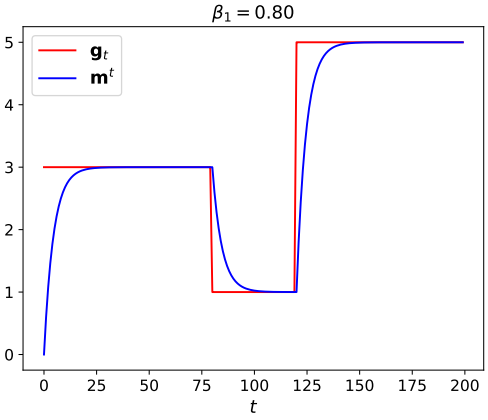
$$\begin{aligned}
\mathbf{m}^1 &= \beta_1 \mathbf{m}^0 + (1-\beta_1)\mathbf{g}_0 = (1-\beta_1)\mathbf{g}_0 \\
\mathbf{m}^2 &= \beta_1 \mathbf{m}^1 + (1-\beta_1)\mathbf{g}_1 \\
&= \beta_1 (1-\beta_1)\mathbf{g}_0 + (1-\beta_1)\mathbf{g}_1 \\
\mathbf{m}^3 &= \beta_1 \mathbf{m}^2 + (1-\beta_1)\mathbf{g}_2 \\
&= \beta_1^2 (1-\beta_1)\mathbf{g}_0 + \beta_1(1-\beta_1)\mathbf{g}_1 + (1-\beta_1)\mathbf{g}_2
\end{aligned}$$

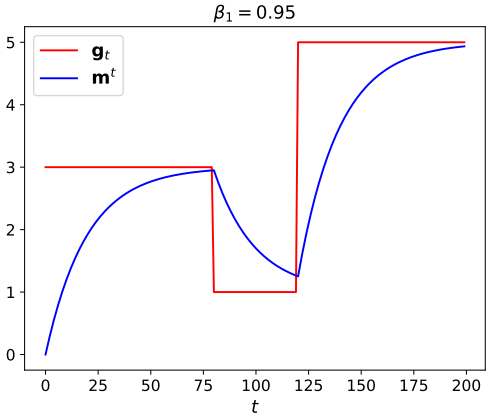
We see that the **weight decays exponentially:**

$$\mathbf{m}^t = (1-\beta_1) \sum_{i=0}^{t-1} \beta_1^{t-i-1} \mathbf{g}_i$$

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# Exponential Moving Average

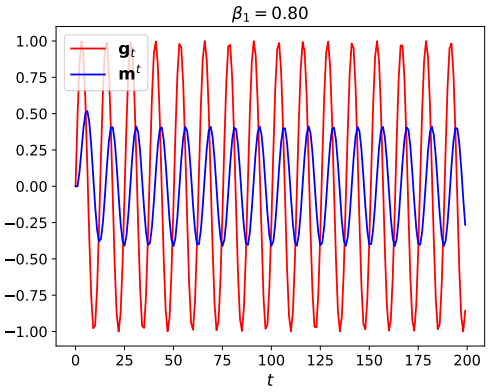


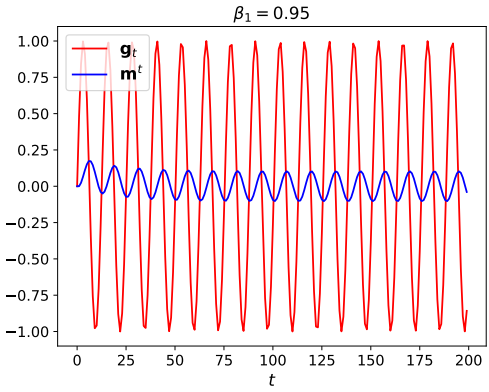




- The moving average effectively **dampens** the behavior of the function

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# Exponential Moving Average



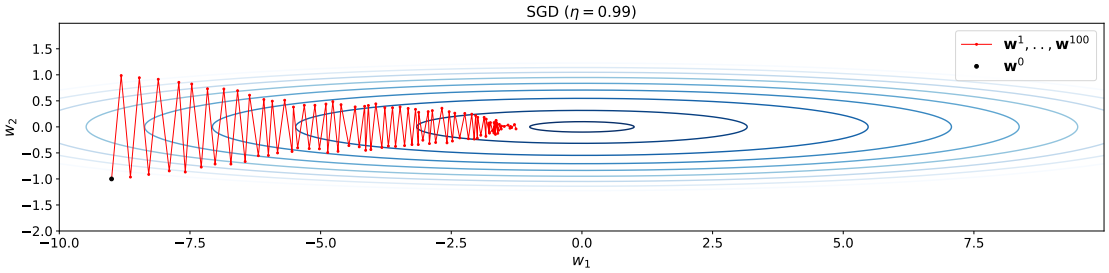




- The moving average effectively **dampens** the behavior of the function

Sutskever, Martens, Dahl and Hinton: On the importance of initialization and momentum in deep learning. ICML, 2013.

# RMSprop

# RMSprop

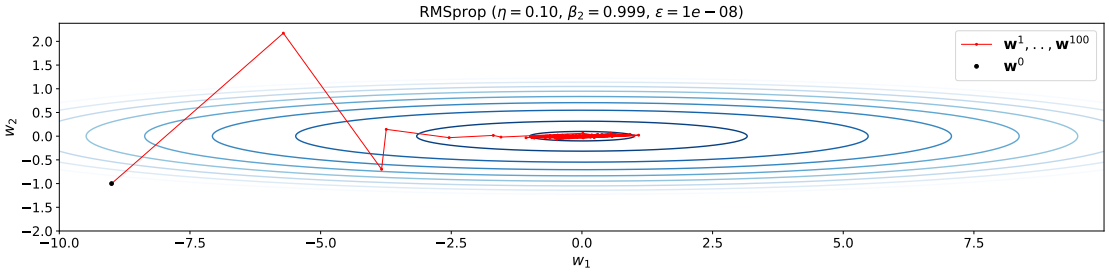



**Motivation for RMSprop:**

- Gradient distribution is very **uneven** and thus requires conservative learning rates
- In this SGD example, gradients are very large in $w_2$ but small in $w_1$
- Idea of RMSprop: **divide learning rate by moving average of squared gradients**

Hinton: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012.

# RMSprop



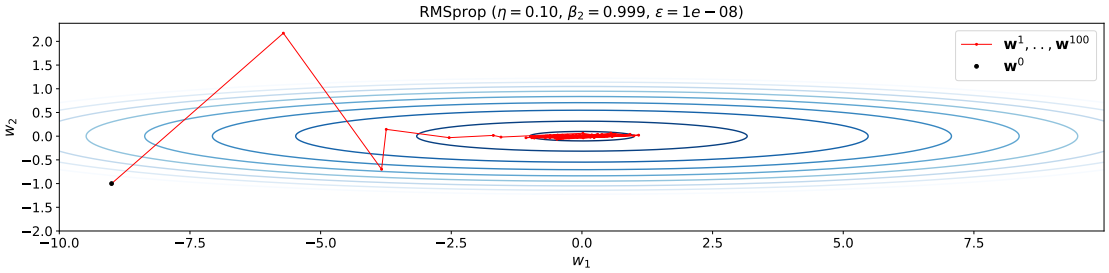

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1 - \beta_2) \left( \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

- With uncentered variance of gradient $\mathbf{v}$, momentum $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$
- Note: All operations (division, square root) are **elementwise** operations

Hinton: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012.

# RMSprop



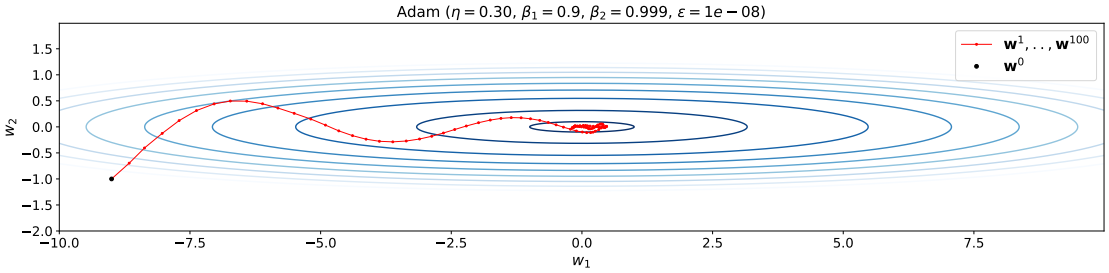

**Remarks:**

- Division by running average of squared gradients **adjusts per-weight step size**
  $\Rightarrow$ Division in $w_2$ direction will be large, division in $w_1$ direction will be small
- This **allows for increasing the learning rate** compared to vanilla SGD
- However: In first iterations, moving average biased towards zero

Hinton: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012.

# Adam

# Adam




$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1-\beta_1)\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1-\beta_2)\left(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)\right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\mathbf{m}^{t+1}}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

- Adam combines the benefits of Momentum and RMSprop

Kingma and Ba: Adam: A Method for Stochastic Optimization. ICLR, 2015.

# Adam with Bias Correction

**Adam Update Equation:**

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1-\beta_1) \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1-\beta_2) \left(\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)\right)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\mathbf{m}^{t+1}}{\sqrt{\mathbf{v}^{t+1}} + \epsilon}$$

- What happens at $t = 1$?
- As $\mathbf{m}^0 = \mathbf{v}^0 = \mathbf{0}$, both $\mathbf{m}^1$ and $\mathbf{v}^1$ are strongly biased towards their initial value $\mathbf{0}$
- We should correct for this bias at the beginning of training

Kingma and Ba: Adam: A Method for Stochastic Optimization. ICLR, 2015.

# Adam with Bias Correction

**Adam Update Equation:**

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1-\beta_1)\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)$$

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1-\beta_2)\left(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t) \odot \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}^t)\right)$$

$$\hat{\mathbf{m}}^{t+1} = \frac{\mathbf{m}^{t+1}}{1-\beta_1^{t+1}} \qquad \hat{\mathbf{v}}^{t+1} = \frac{\mathbf{v}^{t+1}}{1-\beta_2^{t+1}}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\hat{\mathbf{m}}^{t+1}}{\sqrt{\hat{\mathbf{v}}^{t+1}} + \epsilon}$$

- Why is this modification removing the bias from the Adam update equations?

Kingma and Ba: Adam: A Method for Stochastic Optimization. ICLR, 2015.

# Adam with Bias Correction

Let $\mathbf{g}_t = \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^t)$ denote the gradient of the stochastic objective $\mathcal{L}(\mathbf{w}^t)$.
Let further $\mathbf{m}^0 = \mathbf{0}$. Then, the update $\mathbf{m}^{t+1} = \beta_1 \, \mathbf{m}^t + (1 - \beta_1) \, \mathbf{g}_t$ can be written as:

$$\mathbf{m}^t = (1 - \beta_1) \sum_{i=0}^{t-1} \beta_1^{t-i-1} \, \mathbf{g}_i$$

Thus, the expectation over $\mathbf{m}^t$ is given by:

$$\begin{aligned}
\mathbb{E}[\mathbf{m}^t] &= \mathbb{E}\left[ (1 - \beta_1) \sum_{i=0}^{t-1} \beta_1^{t-i-1} \, \mathbf{g}_i \right] \\
&\approx \mathbb{E}\left[\mathbf{g}_t\right] \cdot (1 - \beta_1) \sum_{i=0}^{t-1} \beta_1^i \\
&= \mathbb{E}\left[\mathbf{g}_t\right] \cdot (1 - \beta_1^t)
\end{aligned}$$

# Stretching Break

5 Minutes

# First-order Methods

**There exist many variants:**

- SGD
- SGD with Momentum
- RMSprop
- Adagrad
- Adadelta
- Adam
- AdaMax
- AMSGrad

**Adam** is mostly the method of choice due to its robustness. Visualizations:

- Sebastian Ruder: `https://ruder.io/optimizing-gradient-descent/`
- Jaewan Yun: `https://github.com/Jaewan-Yun/optimizer-visualization`

# What Do People Use Today?

- Adam is from 2015. Since then, many new optimizers have been proposed
- In careful benchmarks, well-tuned Adam still comes out among the best; newer methods give at most modest speedups
- Today, many large models use **AdamW**, Adam with weight decay (next lecture)
- However, recently people have explored novel optimizers such as **Muon**

Dahl et al.: Benchmarking Neural Network Training Algorithms. 2023. · Wen et al.: Fantastic Pretraining Optimizers and Where to Find Them. 2025.

# Matrix Preconditioners: Muon

Adam divides every entry of the update by its own scalar: a **diagonal** preconditioner. A **matrix preconditioner** multiplies the update of a weight matrix $\mathbf{W}$ with a matrix.
Example: **Muon** [Jordan et al., 2024]

$$\mathbf{M}^{t+1} = \beta_1 \mathbf{M}^t + \nabla_{\mathbf{W}} \mathcal{L}_{\mathcal{B}}(\mathbf{W}^t)$$
$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \left( \mathbf{M}^{t+1} (\mathbf{M}^{t+1})^\top \right)^{-1/2} \mathbf{M}^{t+1}$$

- With the SVD $\mathbf{M}^{t+1} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$, the update is $\mathbf{U}\mathbf{V}^\top$: all singular values become 1

Jordan et al.: Muon: An Optimizer for Hidden Layers in Neural Networks. 2024.

# Second-order Methods

**Second-order methods:**

- Newton
- Gauss-Newton
- BFGS / L-BFGS
- Levenberg-Marquardt

**Remarks:**

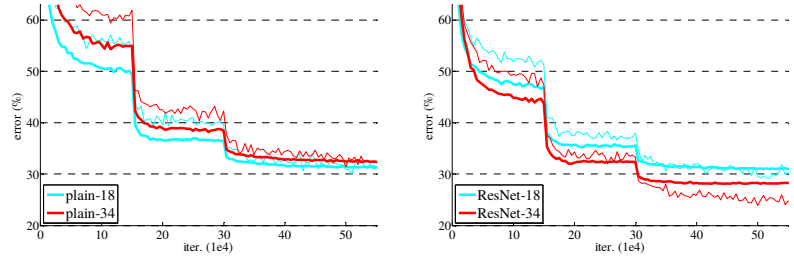
- Second-order methods are more powerful as they exploit the second derivative of the loss, i.e., its curvature
- The Hessian can be computed with backpropagation, but this takes $\mathcal{O}(W^2)$ operations and memory for $W$ parameters [Bishop, 1992]
- Thus, second-order methods are typically not applied for training deep models

Bishop: Exact Calculation of the Hessian Matrix for the Multilayer Perceptron. 1992. · Abreu et al.: The Potential of Second-Order Optimization for LLMs. 2025.

# 5.3
# Optimization Strategies

# Learning Rate Schedules

# Learning Rate Schedules

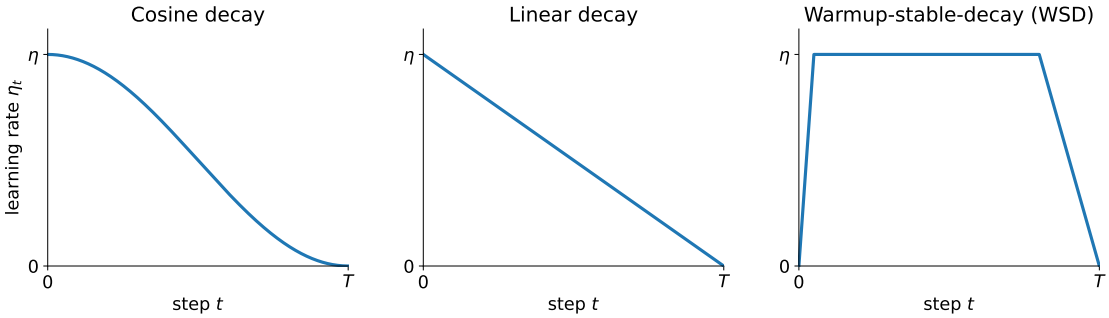



**Learning rate schedules:**

- Fixed learning rate (not a good idea: too slow in the beginning and fast in the end)
- Inverse proportional decay: $\eta_t = \eta/t$ (Robbins and Monro)
- Exponential decay: $\eta_t = \eta\alpha^t$
- Step decay: $\eta \leftarrow \alpha\eta$ (every $K$ iterations/epochs, common in practice: $\alpha = 0.5$)

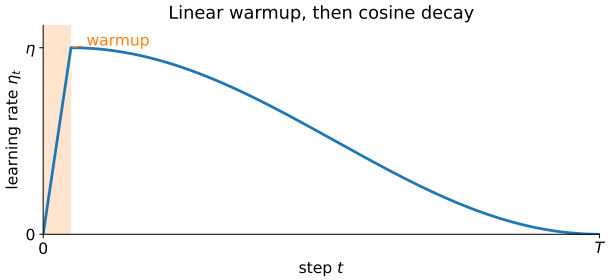
He, Zhang, Ren andSun: Deep Residual Learning for Image Recognition. CVPR, 2016.

# Modern Learning Rate Schedules




- **Cosine decay:** $\eta_t = \frac{\eta}{2}\left(1 + \cos(\pi t/T)\right)$ for $T$ steps, the default for many years
- **Linear decay:** $\eta_t = \eta\left(1 - t/T\right)$, works as well as cosine decay or better
- **Warmup-stable-decay:** warmup, constant $\eta$, decay at the end (last 10–20%)

# Learning Rate Warmup

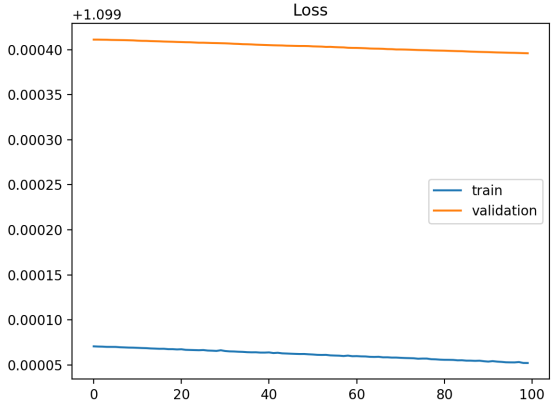



- **Warmup:** increase $\eta_t$ linearly from 0 to $\eta$ during the first steps of training
- Large steps right after initialization make training unstable
- Warmup allows larger learning rates
- Today's default: a short warmup, then cosine or linear decay

Goyal et al.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017.

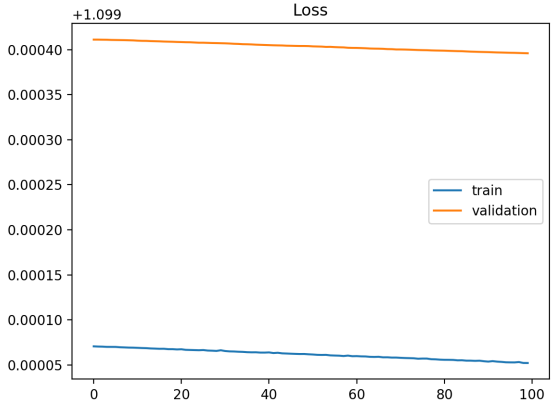
# Monitoring the Training Process

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

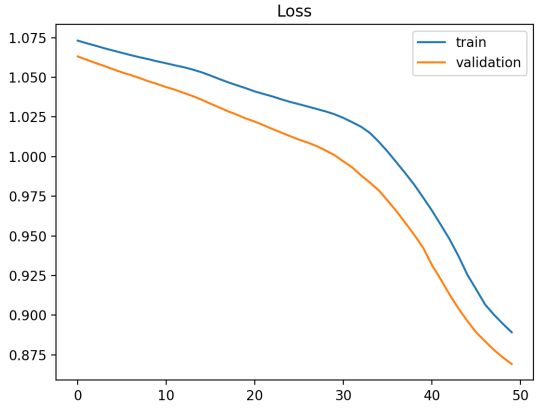
# Monitoring the Training Process




- Underfitting: Model does not have enough capacity to decrease losses.

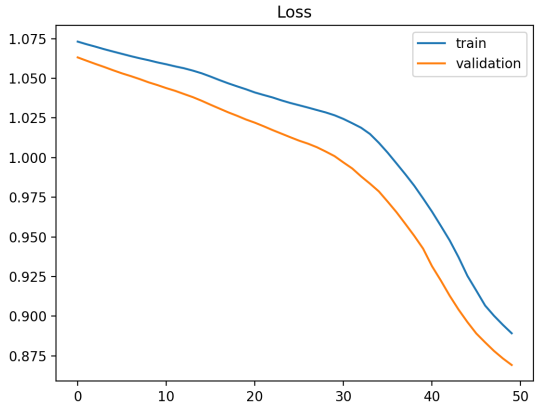
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

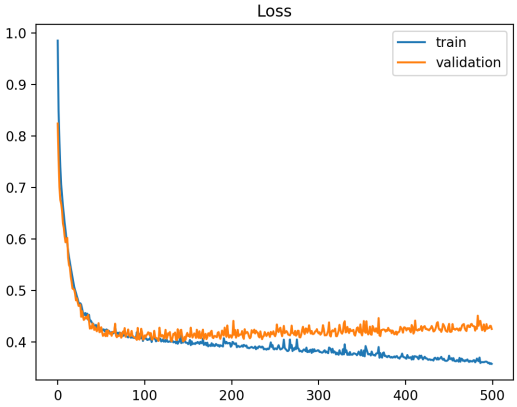
# Monitoring the Training Process




- Not converged: Model requires more iterations to converge.

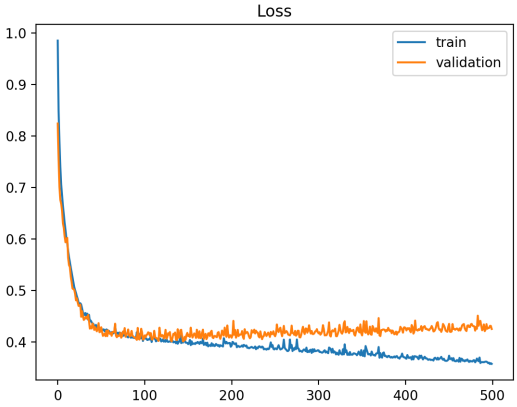
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

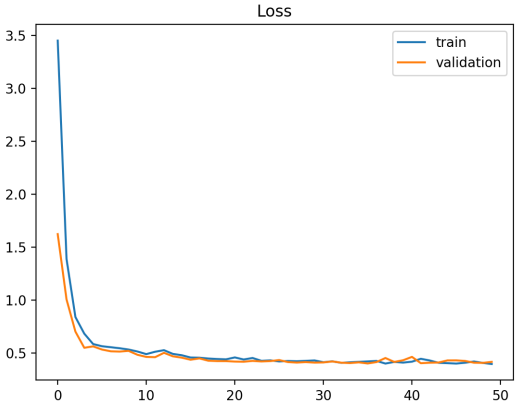
# Monitoring the Training Process




- Overfitting: Training loss decreases but validation loss increases.

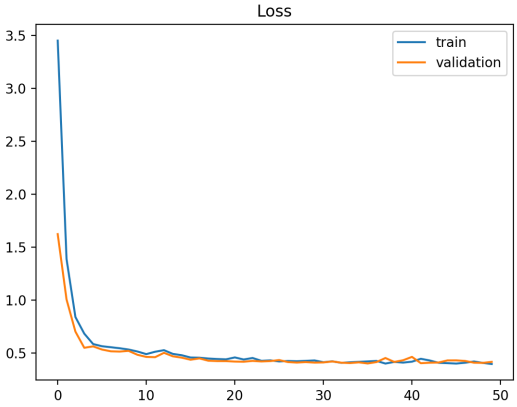
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

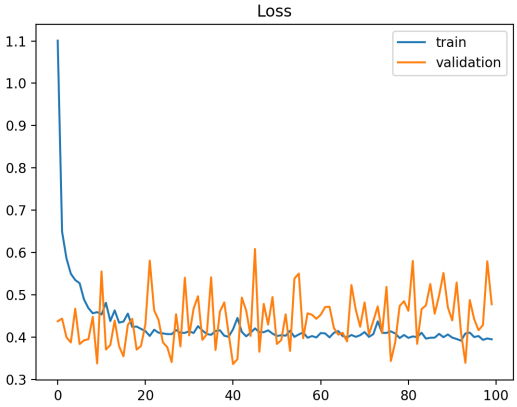
# Monitoring the Training Process




- Example of train and validation curves showing a good fit.

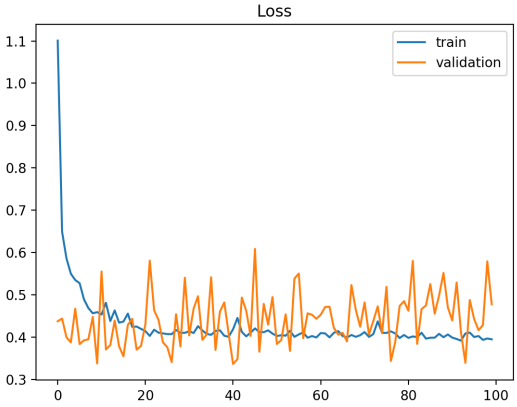
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

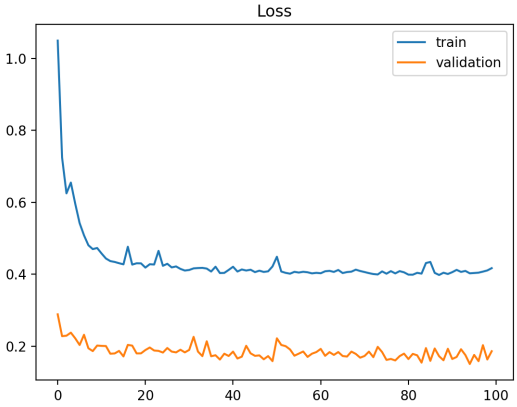
# Monitoring the Training Process




- Noisy validation cuves: Validation set might be too small.

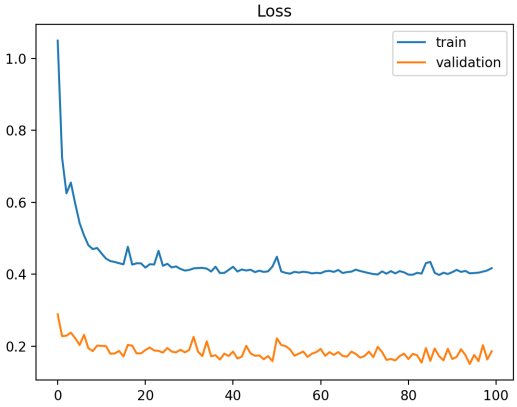
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Monitoring the Training Process




https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

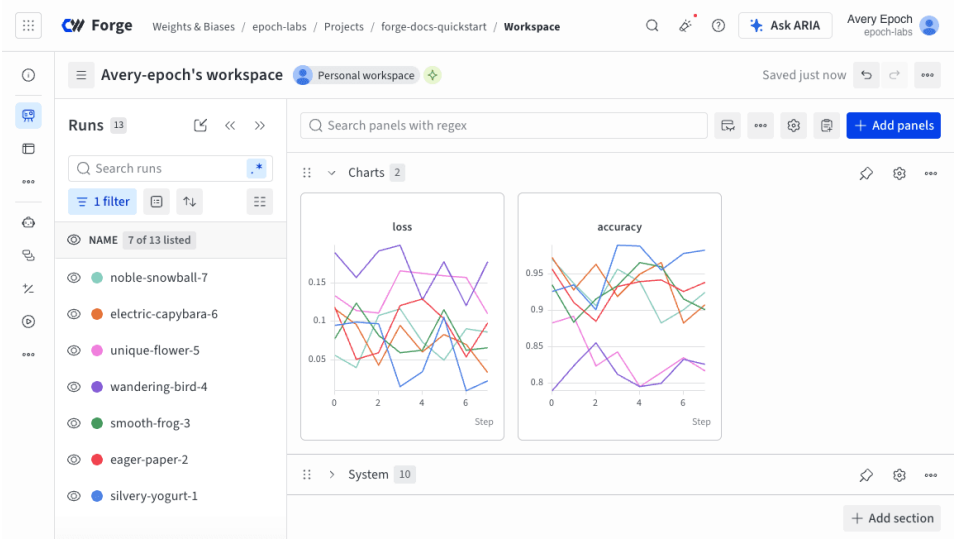
# Monitoring the Training Process




- Might also happen: Validation set is easier than training set.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

# Weights & Biases



# Hyperparameter Search

# Hyperparameter Search

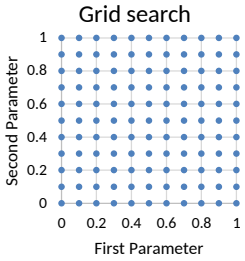
**Hyperparameters:**

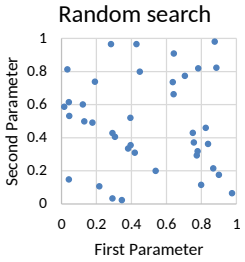
- Network architecture
- Number of iterations
- Batch size
- Learning rate schedule
- Regularization

# Hyperparameter Search

**Methods:**

- Manual search
  - Most common
  - Build intuitions
- Grid search
  - Define ranges
  - Systematically evaluate
  - Requires large resources
- Random search
  - Like grid search but hyperparameters selected based on random draws



# How to Start

**1. Start with single training sample and use a small network**

- First verify that the output is correct
- Then overfit, accuracy should be 100%, fast training/debug cycles
- Choose a good learning rate $(0.1, 0.01, 0.001, ..)$

**2. Increase to 10 training samples**

- Again, verify that the output is correct
- Measure time for one iteration $(< 1s) \Rightarrow$ identify bottlenecks (e.g., data loading)
- Overfit to 10 samples, accuracy should be near 100%

**3. Increase to 100, 1000, 10000 samples and increase network size**

- Plot train and validation error $\Rightarrow$ now you should start to see generalization
- Important: Make only one change at a time to identify causes

# Training Schedules

# Training Schedules

**Pretraining:** (common practice)

- Pretrain backbone (all layers except last few layers) on another task for which a large dataset with labels is available
- Finetune last layers/full architecture on target task/dataset

**Self Supervision:**

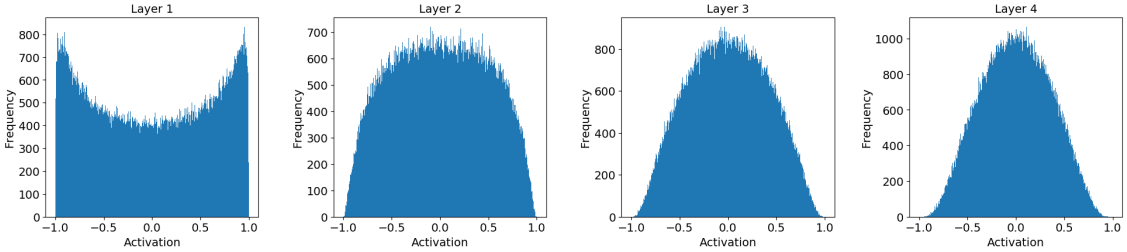
- Pretrain backbone on a task for which supervision is generated from the data itself (e.g., denoising, inpainting, contrastive learning)

**Curriculum Learning:**

- Start learning on easy datasets and successively increase difficulty

# 5.4
# Normalization Layers

# Recap: Initialization (Lecture 4)




**Xavier/He Initialization:**

- Initialize such that activation distribution constant Gaussian across all layers

Glorot and Bengio: Understanding the difficulty of training deep feedforward neural networks. AISTAT, 2010.

# Why Normalization Layers?

**Last lecture:**

- Zero-centered inputs with unit variance are good for optimization
- Xavier/He initialization keeps the activations $\Theta(1)$ across all layers

**But:** this holds only at initialization

- During training, the weights change and the activation statistics drift
- Activation functions destroy zero-centering, e.g., sigmoid and ReLU are $\geq 0$

**Idea:** if zero mean and unit variance are good, enforce them with a **normalization layer**

# A Normalization Layer

For a set $S$ of activations, compute

$$\mu = \frac{1}{|S|} \sum_{j \in S} x_j \qquad \sigma^2 = \frac{1}{|S|} \sum_{j \in S} (x_j - \mu)^2$$

and output, for every $j \in S$,

$$\hat{x}_j = \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}} \qquad y_j = \gamma \, \hat{x}_j + \beta$$

- Learnable scale $\gamma$ and shift $\beta$
- Differentiable, backpropagation as usual

# Batch Normalization

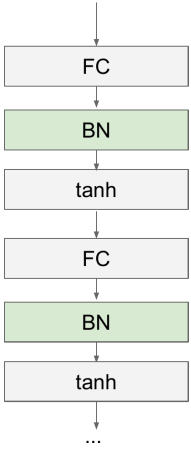
$$\mu_c = \frac{1}{B}\sum_{b=1}^{B} x_{b,c}$$

$$\sigma_c^2 = \frac{1}{B}\sum_{b=1}^{B}(x_{b,c} - \mu_c)^2$$

$$\hat{x}_{b,c} = \frac{x_{b,c} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

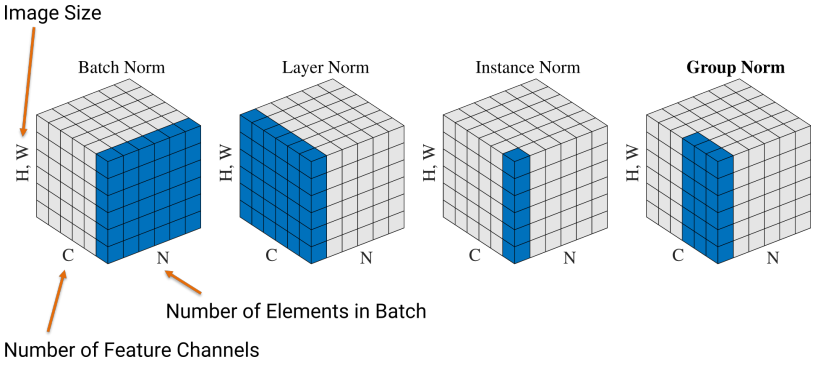
$$y_{b,c} = \gamma_c \hat{x}_{b,c} + \beta_c$$

- Normalize each neuron (=channel) $c$ by mean and variance over batch
- At test time: average during training




- Insert before *or* after activation

# Normalization Layers for Images




- Activations $x[n, c, h, w]$: sample $n$ in the batch, channel $c$, position $(h, w)$
- **Batch norm** normalizes each channel $c$ over the batch and all positions
- **Layer, instance, group norm:** within each sample, hence batch-independent

Wu, He: Group Normalization. ECCV, 2018.

# RMS Normalization

**Layer normalization without the mean:**

$$y_i = \gamma_i \frac{x_i}{\text{RMS}(\mathbf{x})} \qquad \text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2 + \epsilon}$$

- Only re-scaling, no $\mu$ and no $\beta$: cheaper and in practice as good as layer norm
- Used in almost all current language models (Llama, Qwen, DeepSeek, Gemma)

Zhang and Sennrich: Root Mean Square Layer Normalization. NeurIPS, 2019.

# Normalization Layers in Practice

- **Convolutional networks:** batch norm after every convolution (ResNet)
- **Transformers:** layer norm or RMS norm at the start of every block (“pre-norm”)
- **Test time:** batch norm needs running averages from training, layer norm does not
- Normalization layers make training less sensitive to initialization and learning rate

# Break

- **Now: 30 minute break**
- **After that: Tutorial**
  - Numerical stability and SGD



**Exam Quiz 2**

is now live on **Ilias**

**two weeks** to participate

today — Wed 25.11., 14:00