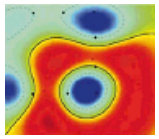
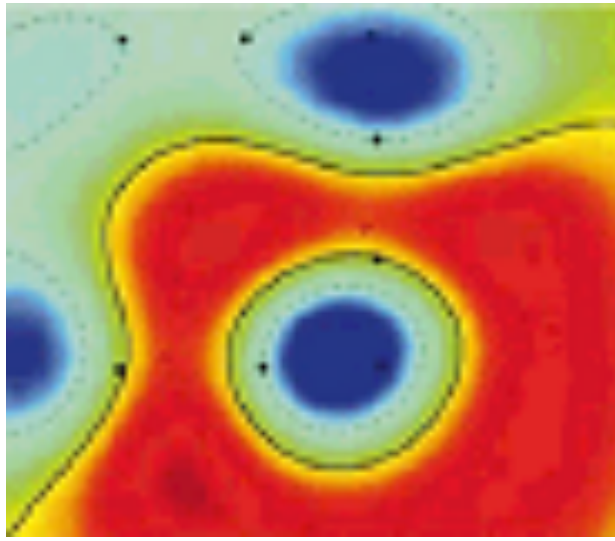


WiSe 2024/25

**Deep Learning 1**

Technische Universität Berlin



Lecture 4

# Optimization (Part 2)

# Outline

**Recap Lecture 3**

**Post-Hoc Mitigation of Poor Conditioning**

- Momentum
- Adam algorithm

**Efficient Optimization**

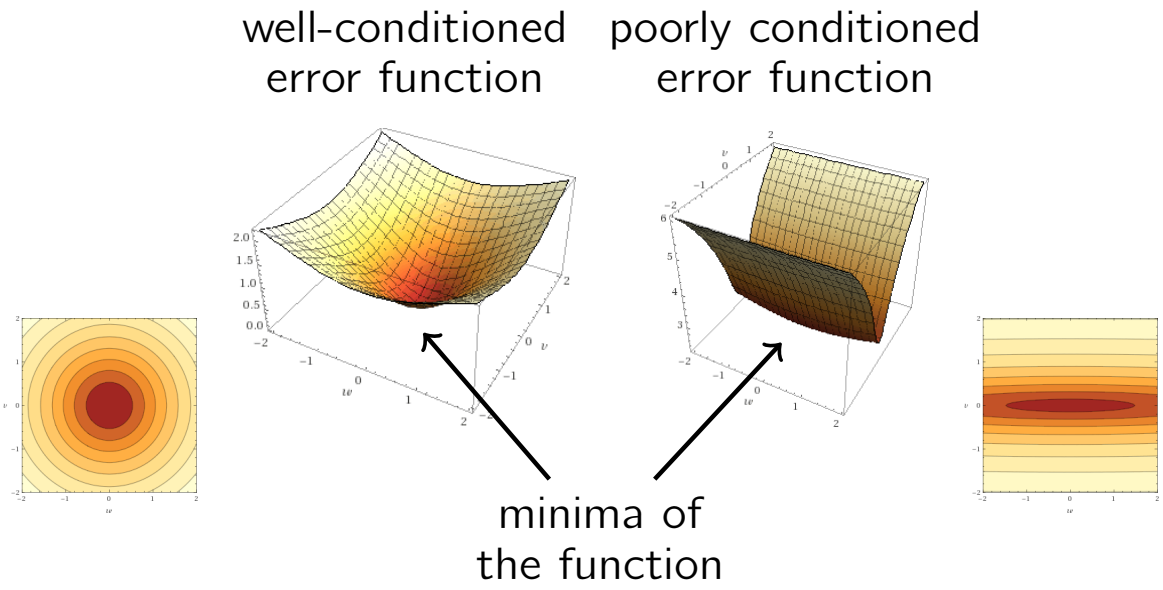
- Redundancies in the error function
- The stochastic gradient descent procedure
- Efficient networks
- Local connectivity

**Implementation Aspects**

- Computations as matrix-vector operations
- Training on specific hardware
- Distributed training schemes

# Recap Lecture 3

# Recap: Hessian-Based Analysis




Using the framework of Taylor expansions, any error function can be expanded at its minimum $\theta^\star$ by the quadratic function:

$$\mathcal{E}(\theta) = \mathcal{E}(\theta^\star) + 0 + \frac{1}{2}(\theta - \theta^\star)^\top H (\theta - \theta^\star) + \text{higher-order terms}$$

where $H$ is the Hessian. The condition number is then the ratio of largest and smallest eigenvalues of the Hessian (the lower the better):

$$\boxed{\text{Condition number} = \lambda_{\max}/\lambda_{\min}}$$

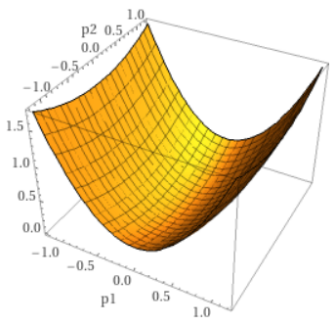
# Better Conditioning $\mathcal{E}(\theta)$

**Various techniques:**

- Centering/whitening the data
- Centering the activations
- Properly scaling the weights
- Design the architecture appropriately (limited depth, shortcut connections, batch-normalization layers, no bottlenecks, ...)

**Note:**

- Despite all measures to reduce the condition number of $\mathcal{E}(\theta)$, the latter may still be poorly conditioned.
- We also need to adapt the optimization procedure in a way that it deals better with poor conditioning.



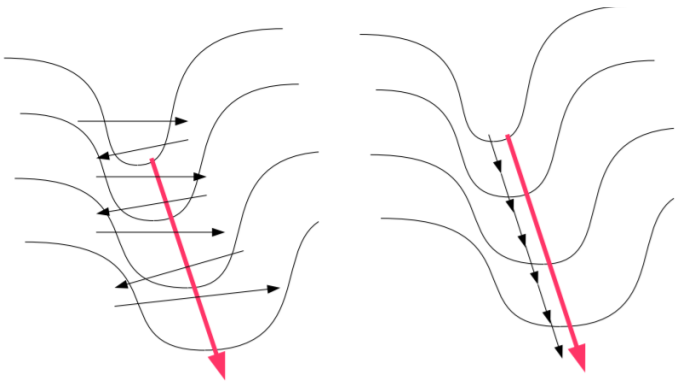
Part 1

# Post-Hoc Mitigations

# Momentum

**Idea:**

- Descent along directions of low curvature can be accelerated by imitating a physical momentum.



**Algorithm:**

- Compute the direction of descent as a cumulation of previous gradients:

$$\Delta \leftarrow \mu \cdot \Delta + (-\nabla \mathcal{E}(\theta))$$

  where $\mu \in [0, 1[$. The higher $\mu$, the stronger the momentum.

- Update the parameters $\theta$ by performing a step along the obtained direction of descent.

$$\theta \leftarrow \theta + \gamma \cdot \Delta$$

**动量（Momentum）**

**概念：**

- 沿着低曲率方向下降可以通过模拟物理动量来加速。

**算法：**

- 计算下降方向，作为之前梯度的累积：

$$\Delta \leftarrow \mu \cdot \Delta + (-\nabla \mathcal{E}(\theta))$$

  其中，$\mu \in [0, 1[$。$\mu$ 越大，动量越强。

- 通过沿着获得的下降方向执行一步更新参数 $\theta$：

$$\theta \leftarrow \theta + \gamma \cdot \Delta$$

（包含的图片展示了动量梯度下降的视觉化效果。）

# Momentum

**性质（Property）：**

- 如果所有梯度估计值 $\nabla\mathcal{E}(\theta)$ 在某个特定方向上保持一致，则该方向上的有效学习率变为：

$$\gamma' = \gamma \cdot \frac{1}{1-\mu}$$

- 这个公式可以通过几何级数的闭式形式推导出来。

**启发式方法（Heuristic）：**

- 当误差函数被认为是条件较差的（即优化困难），建议选择动量参数 $\mu$ 为 0.9 或 0.99。

**Recall:**

- Gradient descent with momentum proceeds as

$$\Delta \leftarrow \mu \cdot \Delta + (-\nabla\mathcal{E}(\theta))$$
$$\theta \leftarrow \theta + \gamma \cdot \Delta$$

**Property:**

If all gradient estimates $\nabla\mathcal{E}(\theta)$ coincide along a particular direction, then the effective learning rate along that direction becomes:

$$\gamma' = \gamma \cdot \frac{1}{1-\mu}$$

This can be derived as the closed form of a geometric series.

**Heuristic:**

- When error function is believed to be poorly conditioned, choose a momentum of $0.9$ or $0.99$.

# The Adam Algorithm

- **Adam** 是我们提出的用于随机优化（stochastic optimization）的算法。详情请参见第 2 节，该算法采用稍微更高效（但较不直观）的计算顺序。
- $g_t^2$ 表示梯度的按元素平方运算（elementwise square）。
- 在测试的机器学习问题中，推荐的默认超参数设置为：

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

- 所有向量运算均为逐元素操作（element-wise）。
- 其中，$\beta_1^t$ 和 $\beta_2^t$ 表示 $\beta_1$ 和 $\beta_2$ 的 t 次幂。

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector

$m_0 \leftarrow 0$ (Initialize $1^{\text{st}}$ moment vector)
$v_0 \leftarrow 0$ (Initialize $2^{\text{nd}}$ moment vector)
$t \leftarrow 0$ (Initialize timestep)
**while** $\theta_t$ not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
**end while**
**return** $\theta_t$ (Resulting parameters)

因为初始时刻m和v会被初始化为0，导致偏差

from Kingma'15

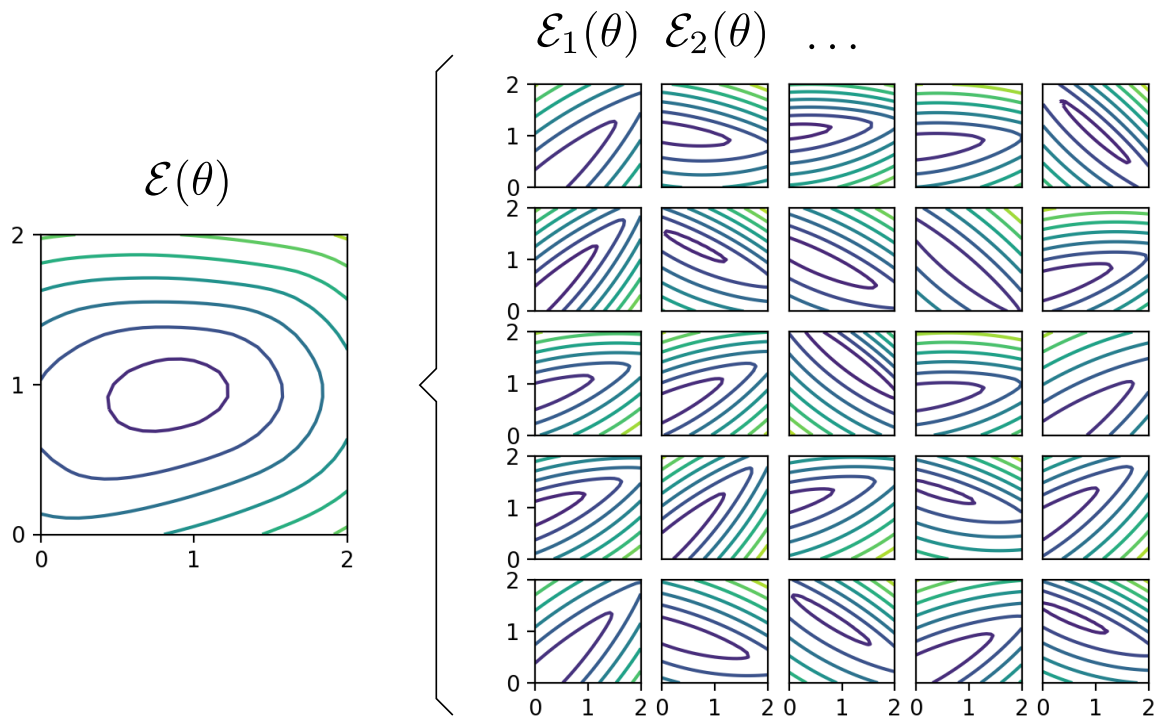
Part 2

# Avoiding Redundancies

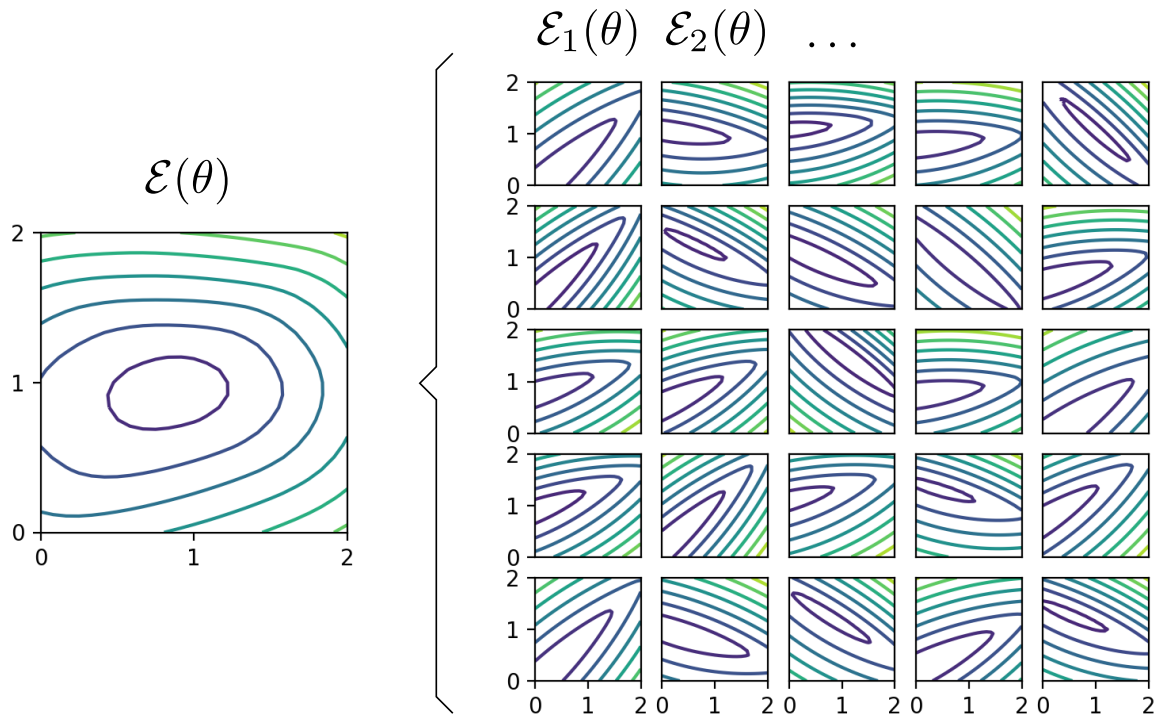
# Data Redundancies

**Observation:**

- The error function can usually be decomposed as the sum of errors on individual data points, i.e. $\mathcal{E}(\theta) = \sum_{i=1}^{N} \mathcal{E}_i(\theta)$.
- Error terms associated to different data points have similar shapes, e.g. for a linear model $y = w \cdot x + b$ with $\theta = (w, b)$, the overall and individual error functions typically look like this:



# Data Redundancies




**Conclusion:**

- It is redundant (and computationally inefficient) to compute the error function for every data point at each step of gradient descent.

**Question:**

- Can we perform gradient descent only on a subset of the data, or alternatively, pick at each iteration a random subset of data?

# Stochastic Gradient Descent

**Gradient descent:**

**for** $t = 1 \dots T$ **do**

$$\theta \leftarrow \theta - \gamma \underbrace{\nabla\Big(\frac{1}{N}\sum_{i=1}^{N}\mathcal{E}_i(\theta)\Big)}_{\nabla\mathcal{E}(\theta)}$$

**end for**

**Stochastic gradient descent:**

**for** $t = 1 \dots T$ **do**

$$\mathcal{I} = \text{choose}(\{1, 2, \dots, N\}, K)$$

$$\theta \leftarrow \theta - \gamma \underbrace{\nabla\Big(\frac{1}{K}\sum_{i\in\mathcal{I}}\mathcal{E}_i(\theta)\Big)}_{\widehat{\nabla}\mathcal{E}(\theta)}$$

**end for**

- Gradient descent costs $O(N)$ at each iteration whereas stochastic gradient descent costs $O(K)$ where $K \ll N$.
- $\widehat{\nabla}$ is an unbiased estimator of $\nabla$.
- SGD may never stabilize to a fixed solution due to the random sampling.

# Stochastic Gradient Descent

**Idea:**

- Make the learning rate decrease over time, i.e., replace the fixed learning rate $\gamma$ by a time-dependent learning rate $\gamma^{(t)}$.

**Stochastic gradient descent (improved):**

**for** $t = 1 \dots T$ **do**

$$\mathcal{I} = \text{choose}(\{1, 2, \dots, N\}, K)$$

$$\theta \leftarrow \theta - \gamma^{(t)} \underbrace{\nabla\Big(\frac{1}{K} \sum_{i \in \mathcal{I}} \mathcal{E}_i(\theta)\Big)}_{\widehat{\nabla}\mathcal{E}(\theta)}$$

**end for**

- SGD is guaranteed to converge if the learning rate satisfies the following two conditions:

$$\lim_{t \to \infty} \gamma^{(t)} = 0 \qquad \text{(i)}$$

$$\sum_{t=1}^{\infty} \gamma^{(t)} = \infty \qquad \text{(ii)}$$

# Choosing the Learning Rate Schedule of SGD

| | $\gamma^{(t)} = 1$ | $\gamma^{(t)} = t^{-1}$ | $\gamma^{(t)} = e^{-t}$ |
|---|---|---|---|
| $\lim_{t\to\infty} \gamma^{(t)} = 0$ | ✗ | ✓ | ✓ |
| $\sum_{t=1}^{\infty} \gamma^{(t)} = \infty$ | ✓ | ✓ | ✗ |

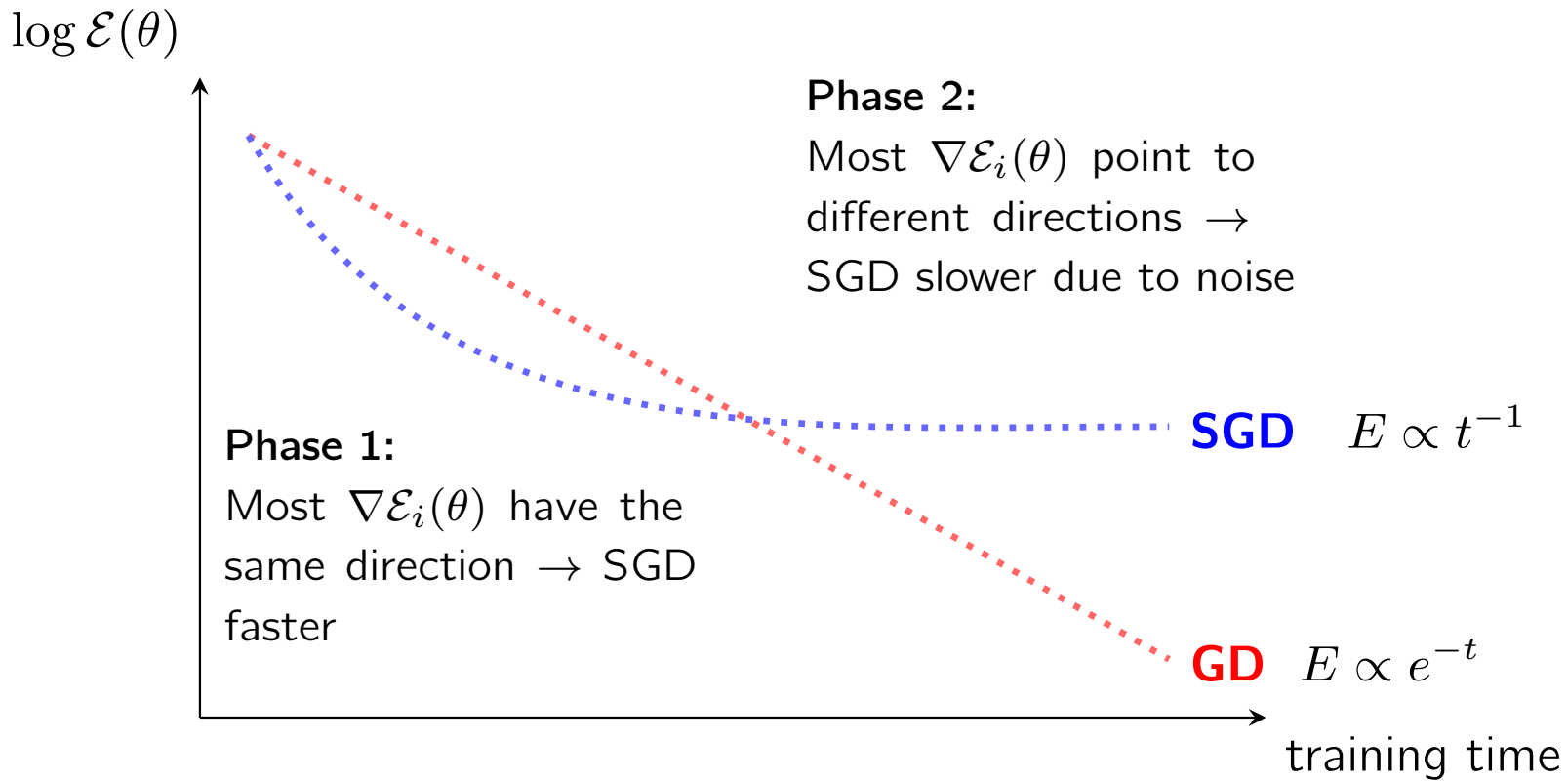
**Observation:**

- The learning rate should decay but not too quickly.
- Because of ths required slow decay, one also gets a slow convergence rate, e.g. $t^{-1}$. (Compare with the exponential convergence of GD near the optimum).

**Question:**

- Is SGD useful at all?

# GD vs. SGD Convergence




**Observations:**

- In Phase 1, constants ($K$ vs. $N$) matter. SGD moves way faster initially.
- Phase 2 is often irrelevant, because the model already starts overfitting before reaching it.
- $K$ can be increased over the course of training in order to perform efficiency in both phases.

# Further advantages of SGD vs. GD

- May escape local minima due to noise.
- May arrive at better generalizing solution (cf. Regularization in lectures 5–6).

Part 3

# Model Efficiency

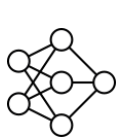
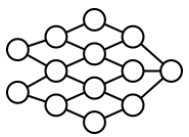
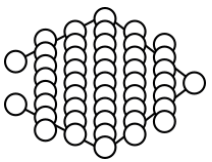
# Model Efficiency

**Observation:**

- Another factor that can have a strong effect on training efficiency is how much time/resource it takes to produce one forward pass.

**General guidelines:**

- The number of neurons in the network should not be chosen larger than needed for the task.

| |  |  |  |
|---|---|---|---|
| Solves the task | ✗ | ✓ | ✓ |
| Cheap to evaluate | ✓ | ✓ | ✗ |

- The network should be organized in a way that only relevant computations are performed.

# Model Efficiency

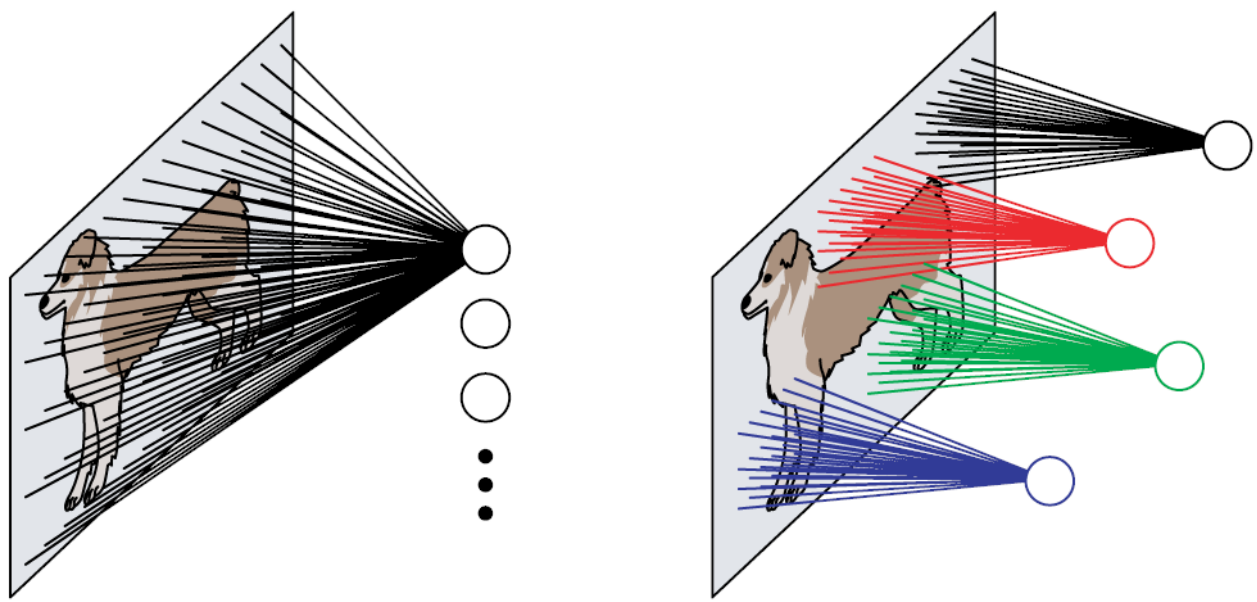
模型效率（Model Efficiency）

全局连接 vs. 局部连接（Global connectivity vs. local connectivity）

- 仅保留局部连接可以显著减少每个神经元的计算量。
- 仅当该层计算的表示不需要长距离交互时，此方法才有效。

## Global connectivity vs. local connectivity

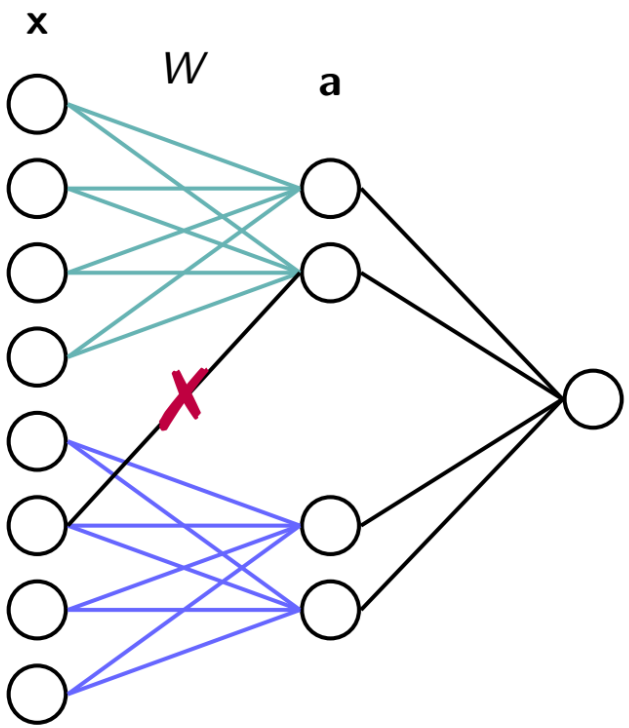
- Keeping only local connections can substantially reduce the number of computations for each neuron.
- Only works if the representation computed at a the layer does not require long-range interactions.



Adapted from B. Sick, O. Durr, Deep Learning Lecture, ETHZ.

# Model Efficiency

## Global connectivity vs. local connectivity

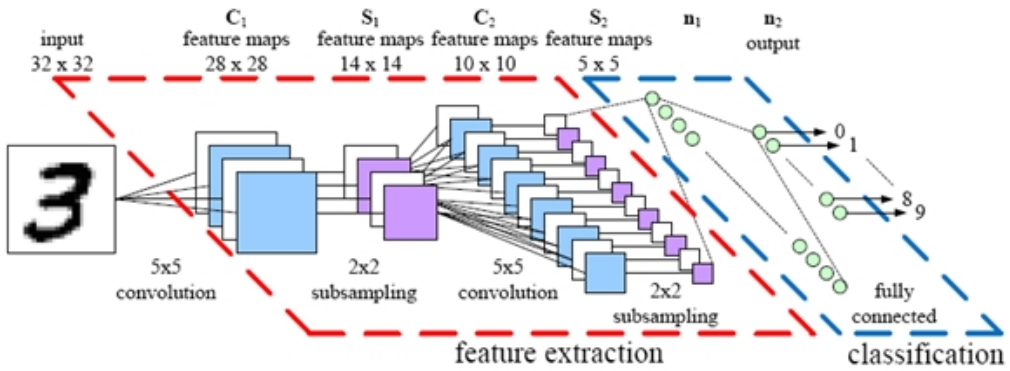



$$\mathbf{a} = \underbrace{W^\top \mathbf{x}}_{8\times 4=32 \text{ computations}} = \underbrace{W_\mathrm{A}^\top \mathbf{x}_\mathrm{A} + W_\mathrm{B}^\top \mathbf{x}_\mathrm{B}}_{2\times(2\times 4)=16 \text{ computations}}$$

$$W = \begin{pmatrix} W_\mathrm{A} & 0 \\ 0 & W_\mathrm{B} \end{pmatrix}$$

# Avoiding Computational Bottlenecks

## The CNN Architecture:




- Lower layers detect simple features at exact locations.
- Higher layers detect complex features at approximate locations.

## Key Computational Benefit of the CNN:

- Spatial information is progressively replaced by with semantic information as we move from the input layer to the top layer.
- The dimensionality and number of connections is never too high at any layer.

**CNN 的关键计算优势（Key Computational Benefit of the CNN）**

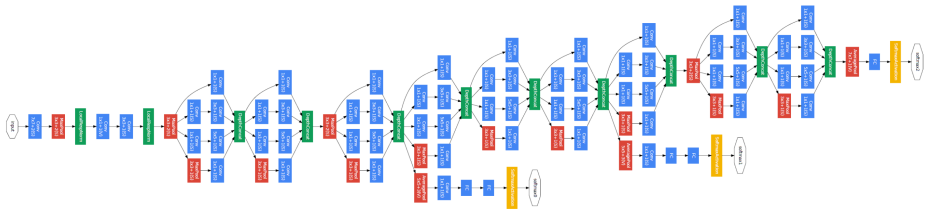
- 随着从输入层到顶层的推进，**空间信息逐渐被语义信息所取代。**
- **每一层的维度和连接数量不会过高**，从而提高计算效率。

（图示部分展示了 CNN 结构，从 5×5 卷积、2×2 采样、特征提取，到最终的全连接分类。）

# Avoiding Computational Bottlenecks

**Example:**

The Inception-v1 (GoogleNet) architecture



| type | patch size/ stride | output size | output filters | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112 | 64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56 | 64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56 | 192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28 | 192 | 0 | | | | | | | | |
| inception (3a) | | 28×28 | 256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28 | 480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14 | 480 | 0 | | | | | | | | |
| inception (4a) | | 14×14 | 512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14 | 512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14 | 512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14 | 528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14 | 832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7 | 832 | 0 | | | | | | | | |
| inception (5a) | | 7×7 | 832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7 | 1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1 | 1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1 | 1024 | 0 | | | | | | | | |
| linear | | 1×1 | 1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1 | 1000 | 0 | | | | | | | | |

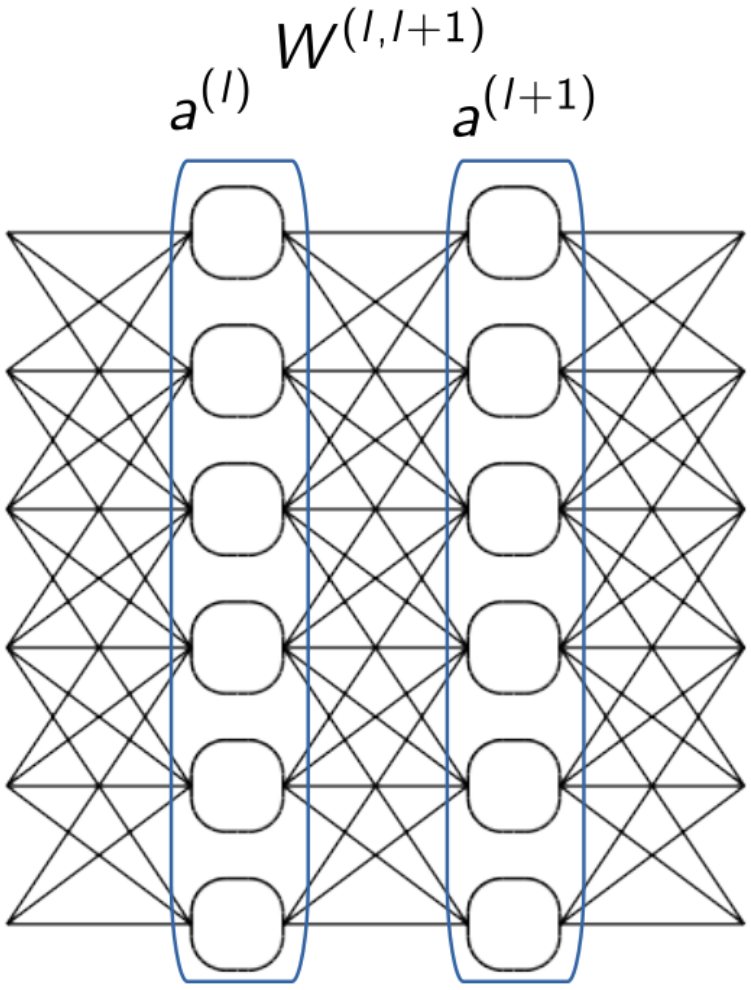
**Observation:**

- No specific layer strongly dominate in terms of number of operations.

Part 4

# Systemize / Parallellize Computations

# Systemize Computations




Per-neuron forward computations

$$\forall_j : a_j = g\left(\sum_i a_i w_{ij} + b_j\right)$$

**Whole-layer computation**

$$a^{(l+1)} = g\left(W^{(l,l+1)} \cdot a^{(l)} + b^{(l+1)}\right)$$

matrix-vector products (e.g. numpy.dot)

element-wise application of nonlinearity

# Choosing Batch Size in SGD

**在 SGD 中选择批量大小（Choosing Batch Size in SGD）**

决定批量大小的选择涉及两个因素：

- **数据点的梯度是否冗余**，通常取决于我们处于训练的**第一阶段**还是**第二阶段**。
- **用于训练模型的计算机是否足够强大**，以便可以在该设备上以 $O(1)$ 的复杂度执行批量操作。

Two factors enter into the decision of the batch size.

- Whether gradient of data points are redundant, typically, whether we are in phase 1 or phase 2 of training.
- Whether the machine used for training the model is sufficiently big so that the batch operation can be performed in $O(1)$ on that machine.

| | Phase 1 of training (correlated gradients) | Phase 2 of training (decorrelated gradients) |
|---|---|---|
| small machine | small batch | medium batch |
| big machine | medium batch | large batch |

# Map Neural Network to Hardware

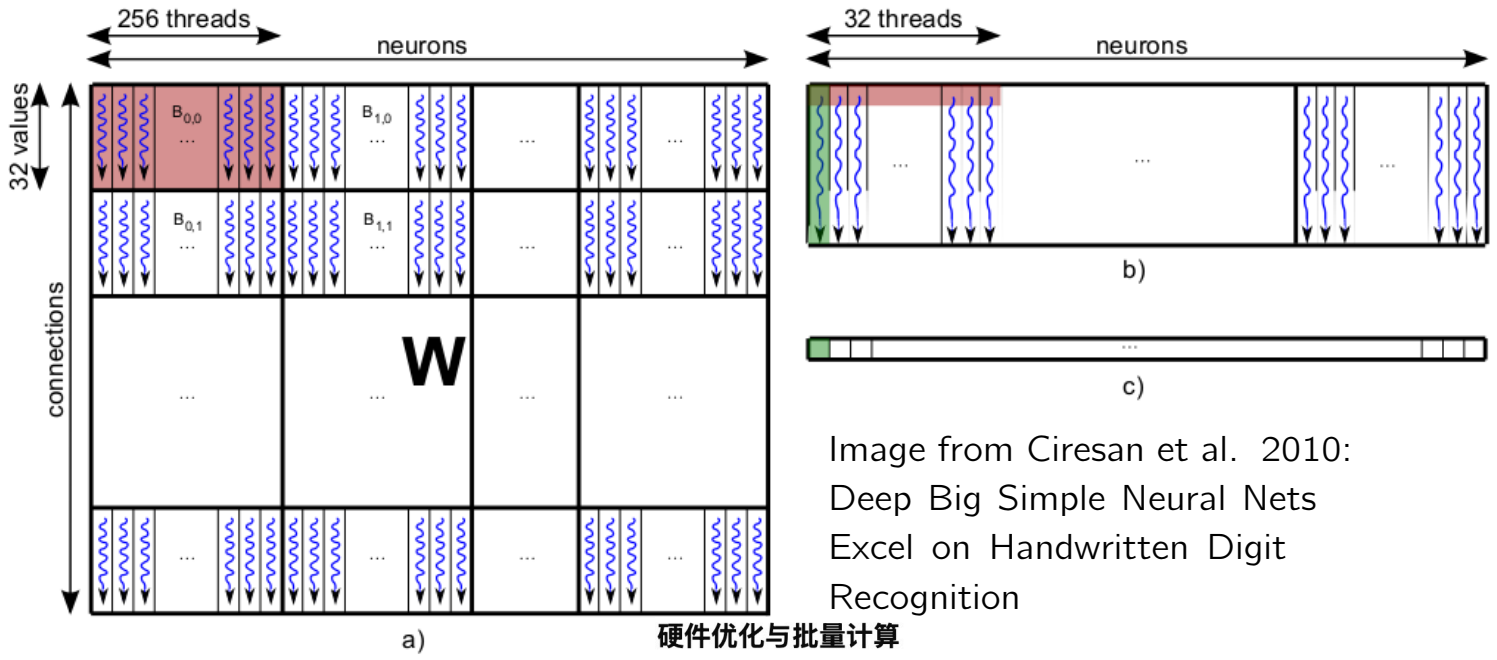



Image from Ciresan et al. 2010: Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition

**硬件优化与批量计算**

- **为了使训练过程与硬件规格（如 CPU 缓存、GPU 计算块大小）最佳匹配，神经网络计算（如批量计算）必须被划分为合适大小的计算块。**
- **这些特定于硬件的优化已经内置于大多数快速神经网络库中**（例如 PyTorch、TensorFlow、cuDNN 等）。

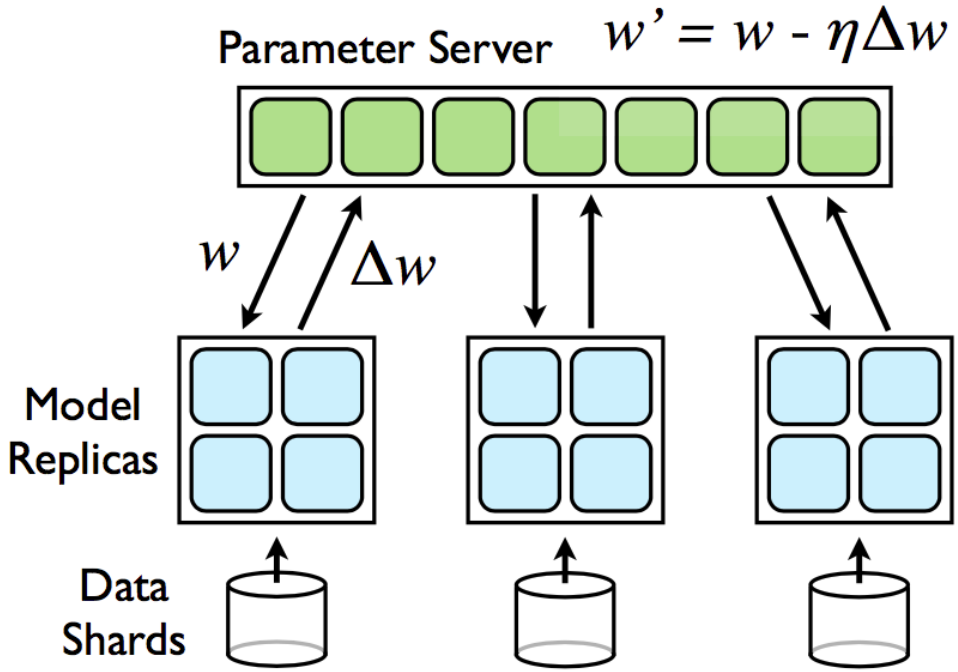
- In order for the training procedure to match the hardware specifications (e.g. CPU cache, GPU block size) optimally, neural network computations (e.g. batch computations) must be decomposed into blocks of appropriate size.
- These hardware-specific optimizations are already built in most fast neural network libraries (e.g. PyTorch, Tensorflow, cuDNN, ...).

Part 5

# Distributed Training

# Distributed Training

**Example:** Google's DistBelief Architecture [Dean'12]

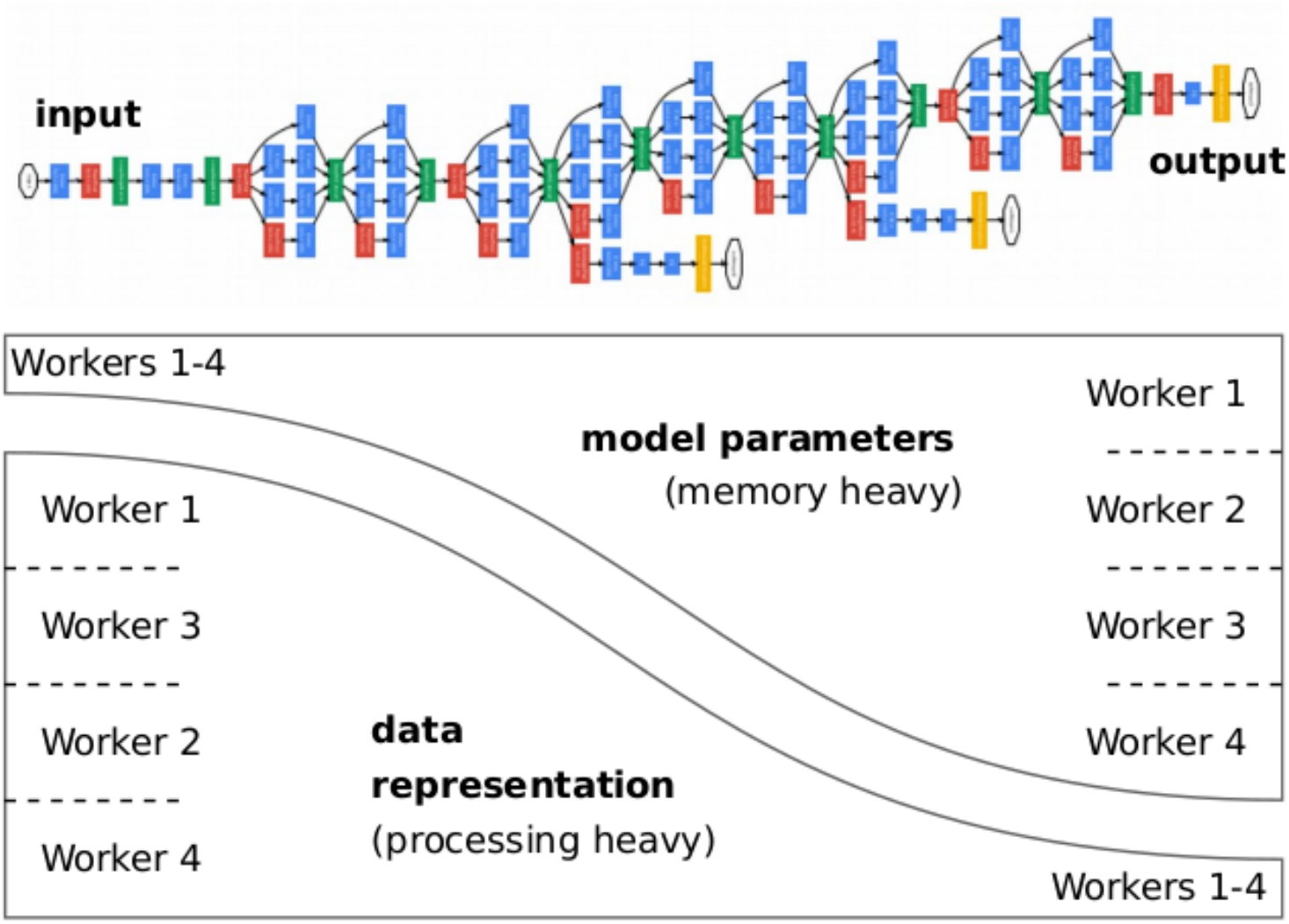



Each model replica trains on its own data, and synchronizes the model parameters it has learned with other replica via a dedicated parameter server.

# Distributed Training

- **模型并行（Model Parallelism）**：在左侧，**模型参数（model parameters）** 由多个计算节点（Worker 1-4）分担存储和计算。这种方法适用于 **模型较大，占用内存较多** 的情况（**memory heavy**）。
- **数据并行（Data Parallelism）**：在右侧，**数据表示（data representation）** 由多个计算节点（Worker 1-4）分别处理，这种方法适用于 **计算量较大的场景（processing heavy）**。

Combining data-parallelism and model-parallelism




see also Krizhevsky'14: One weird trick for parallelizing convolutional neural networks

**左侧（模型并行）**：模型参数主要由不同 Worker 共享，计算过程依赖于多个 Worker 之间的**通信**。

**右侧（数据并行）**：数据被分割成不同批次，每个 Worker 处理**一部分数据**，并最终聚合结果。

**数据并行** 适用于 **计算密集型任务**（如大批量数据训练）。

**模型并行** 适用于 **存储密集型任务**（如超大规模神经网络）。

# Summary

- **即使采用最佳实践**（如数据中心化、设计良好的架构等）来优化误差函数 $\mathcal{E}(\theta)$，其优化仍然计算量大，具有挑战性。
- **对于条件较差的误差函数**，可以通过在**梯度下降**（Gradient Descent）中**引入动量（Momentum）**来改进优化效果。
- **不同数据点对误差函数的贡献通常高度相关**，因此**随机选择数据子集进行梯度计算（随机梯度下降 Stochastic Gradient Descent**）有助于优化计算效率。
- **模型应尽量避免不必要的计算**（例如，去除已知无关特征之间的权重连接），并**避免计算瓶颈**。
- **高效的神经网络训练**需要考虑**硬件的计算能力**（例如，硬件能够在 $O(1)$ 复杂度下执行哪些操作）。
- **超大规模模型和数据集**无法适应单台机器，在这种情况下，需要**设计分布式计算方案**，合理**利用数据并行或模型并行**。

# Summary

- Even with the best practices for shaping the error function $\mathcal{E}(\theta)$ such as data centering, designing a good architecture, etc., the optimization of $\mathcal{E}(\theta)$ remains computationally demanding.
- A poorly conditioned error function can be addressed by enhancing the simple gradient descent procedure with momentum.
- The contributions of different data points to the error function are initially highly correlated $\rightarrow$ it is beneficial to approximate the error gradient from only a random subset of points at each iteration (stochastic gradient descent).
- The model can be shaped in a way that avoids unnecessary computations (e.g. weights connecting features known to be unrelated), and in a way that avoids computational bottlenecks.
- For most efficient neural network training, it is important to consider what the hardware can achieve (e.g. what operation the hardware achieves in $O(1)$).
- Very large models and very large datasets do not fit on a single machine. In that case, we need to design distributed schemes, with appropriate use of data/model parallelism.