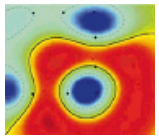
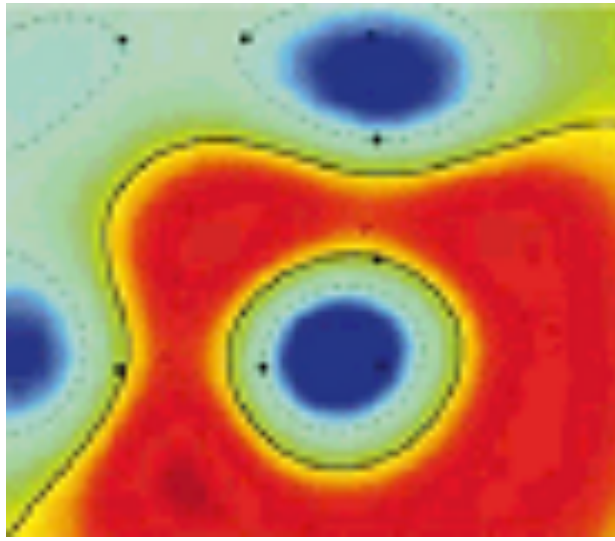


WiSe 2024/25

Deep Learning 1

Technische Universität Berlin



Lecture 3

Optimization (Part 1)

Outline

Recap Lecture 2

- Backpropagation and gradient descent

Characterizing the error function

- The problem of local minima
- The importance of initialization
- The problem of poor conditioning
- Characterizing conditioning with the Hessian

Improving the conditioning

- Data normalization & choice of non-linearities
- Scaling initial weights, batch normalization, skip connections

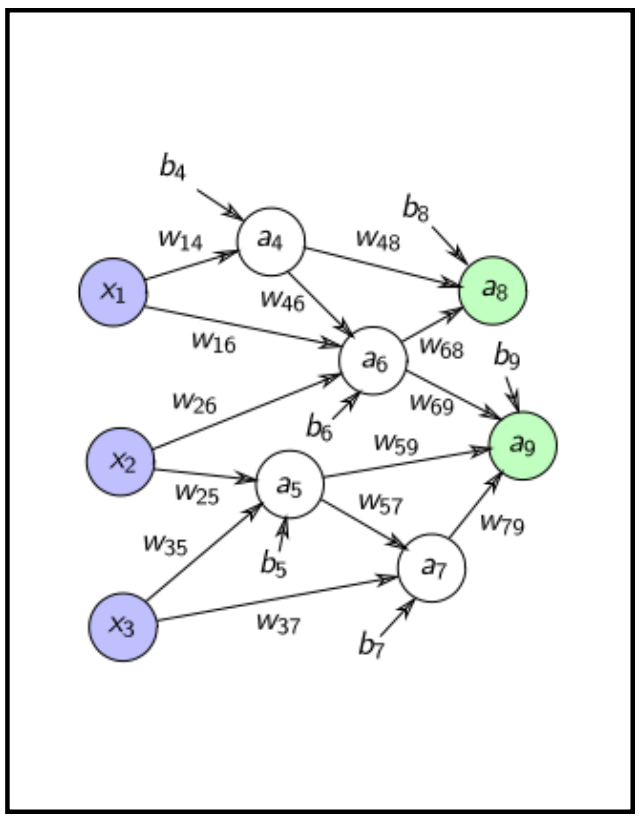
Part 1

Recap Lecture 2

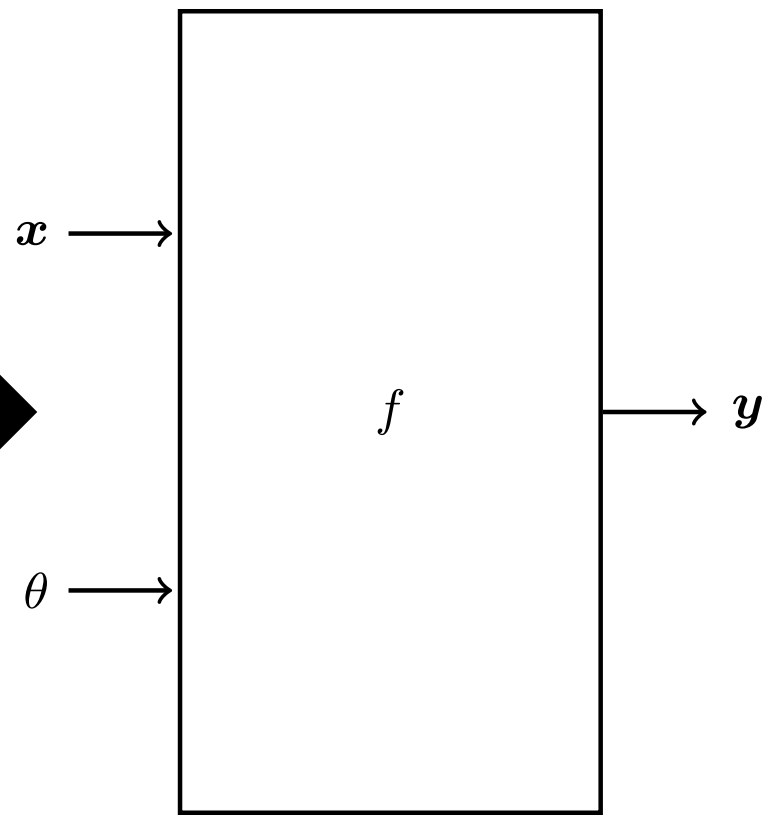
Recap: How to Learn in a Neural Network

Observation:
A neural network is a function of both its inputs and parameters.

graph view



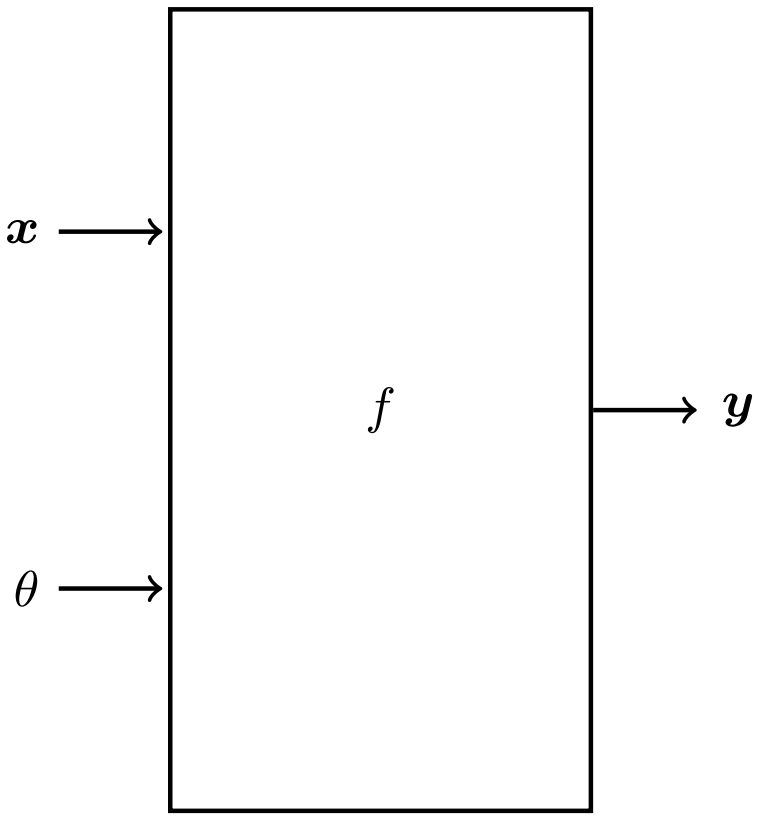
function view



$\boldsymbol{x} = [x_1, x_2, x_3]$ $\quad$ $\boldsymbol{y} = [a_8, a_9]$ $\quad$ $\theta = (w_{ij})_{ij}, (b_j)_j$

Recap: How to Learn in a Neural Network

function view



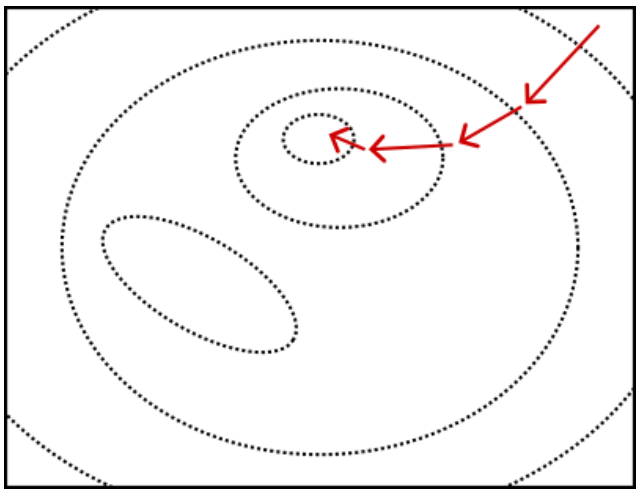
$$\theta = (w_{ij})_{ij}, (b_j)_j$$

Define an error function

$$\mathcal{E}(\theta) = \sum_n \left(f(\boldsymbol{x}_n; \theta) - t_n\right)^2)$$

and minimize it by gradient descent

$$\theta \leftarrow \theta - \gamma \cdot \nabla_\theta \mathcal{E}(\theta)$$

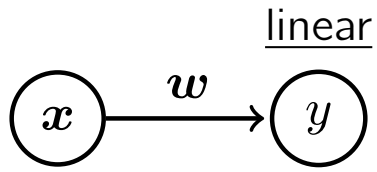


Part 2

Characterizing the Error Function

Characterizing the Error Function: One Layer

- Consider a simple linear neural network made of one layer of parameters:

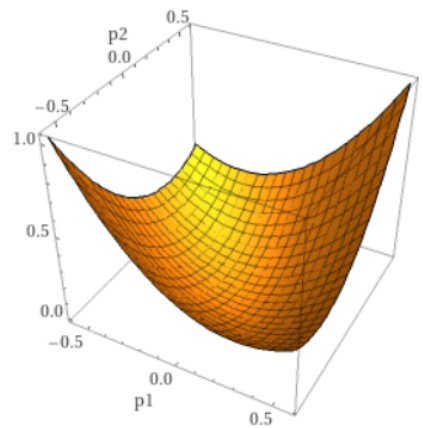


$$y = \boldsymbol{w}^\top \boldsymbol{x}$$

with prediction error averaged over a dataset $\mathcal{D}$ of inputs and their associated targets, i.e. $\mathcal{D} = \{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_N, t_N)\}$ given by:

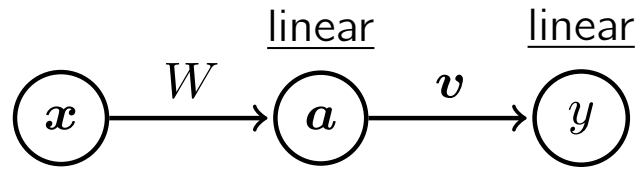
$$\mathcal{E}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{w}^\top \boldsymbol{x}_n - t_n)^2 + \lambda \|\boldsymbol{w}\|^2$$

- One can show that this objective function is convex (like for the perceptron). I.e. one can always reach the minimum of the function by performing gradient descent.



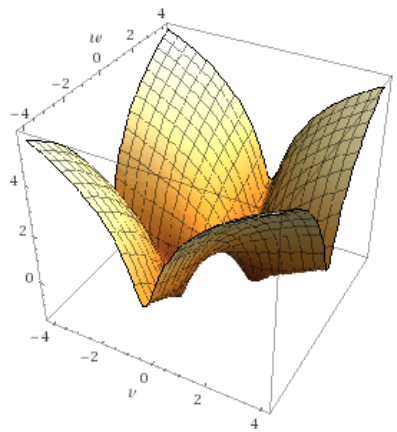
Characterizing the Error Function: Two Layers

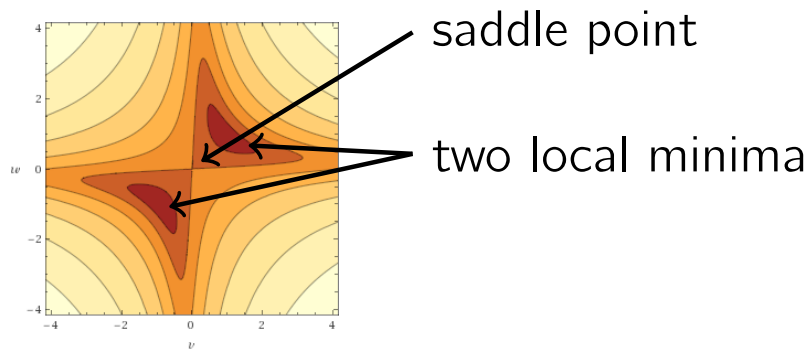
- Consider now a slightly extended version of the neural network above, where we add an extra layer. This gives the error function:



$$\mathcal{E}(W, \boldsymbol{v}) = \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{v}^\top W \boldsymbol{x}_n - t_n \right)^2 + \lambda \left(\|\boldsymbol{v}\|^2 + \|W\|_{\mathrm{F}}^2 \right)$$

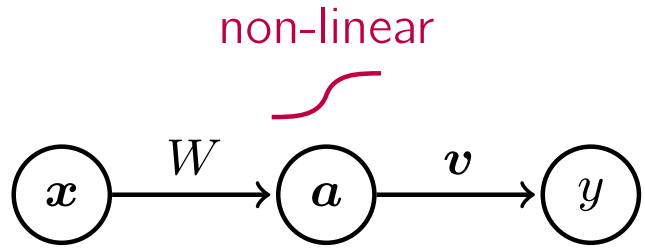
- One can show that this error function is non-convex, e.g. the simple case $N = 1,\ x_1 = 1,\ t_1 = 1,\ \lambda = 0.1$ gives:





Characterizing the Error Function: Two Layers

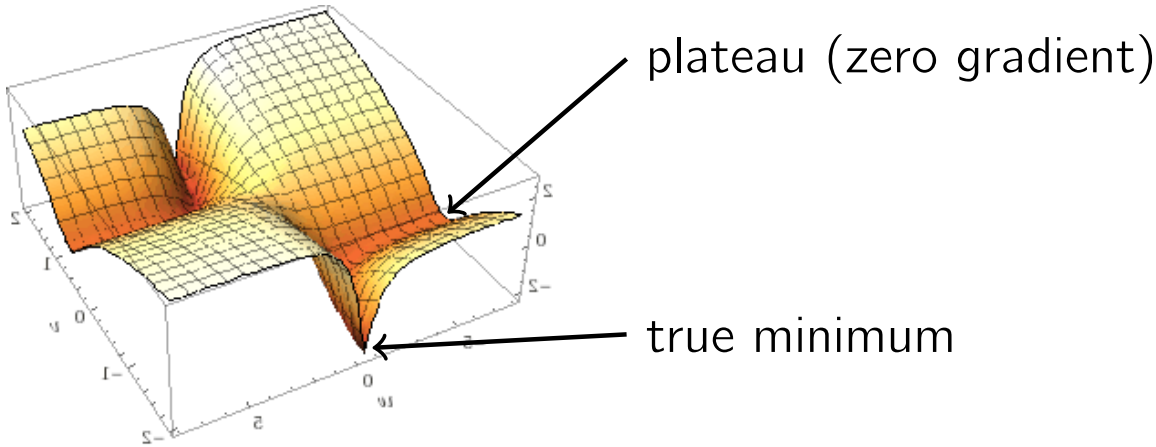
- Let's now use a tanh nonlinear activation function on the intermediate layer which leads to the following error function to minimize:



$$\mathcal{E}(W, \boldsymbol{v}) = \frac{1}{N} \sum_n \|\boldsymbol{v}^\top \tanh(W\boldsymbol{x}) - t_n\|^2 + \lambda(\|\boldsymbol{v}\|^2 + \|W\|^2)$$

- In addition to having several local minima, the error function now has plateaus (non-minima regions with near-zero gradients), which are hard to escape using gradient descent.

高原



Practical Recommendations

不要将参数初始化为零（否则会正好位于鞍点，梯度下降会卡在这个位置）。

最常见的替代方法是将参数随机初始化（例如，从固定尺度的高斯分布中抽取）。

尺度不应过大（以避免非线性函数饱和状态）。

这些基本的启发式方法可以帮助达到某个局部最小值，但不一定是一个好的局部最小值。

Basic recommendations:

- Do not initialize your parameters to zero (otherwise, it is exactly at a saddle point, and the gradient descent is stuck there).
- The most common alternative is to initialize the parameters at random (e.g. drawn from a Gaussian distribution of fixed scale).
- The scale should be not too large (in order avoid the saturated regime of the nonlinearities).

饱和状态

These basic heuristics help to land in some local minimum, but not necessary a good one.

如果条件允许，用多次随机初始化重新训练神经网络，并保留误差最小的训练结果。

设置足够大的学习率可以帮助逃离局部最小值。

在每一层中使用足够数量的神经元（更多的参数使算法更容易逃离局部最小值）。

不要在必要性之外增加神经网络的深度（更深的网络更难优化）。

More recommendations:

- If affordable, retrain the neural network with multiple random initializations, and keep the training run that achieves the lowest error.
- A learning rate set large enough can help to escape local minima.
- Use a sufficient number of neurons at each layer (more parameters makes it easier for the algorithm to escape local minima).
- Do not increase the depth of the neural network beyond necessity (a deeper network is harder to optimize).

Learning Rate Schedules

Idea:

- During training, apply a broad range of learning rates, specifically (1) large learning rates to jump out of local minima, and (2) small learning rate to finely adjust the parameters of the model.

Practical Examples:

- Step decay (every k iterations, decay the learning rate by a certain factor). For example:

$$\gamma(t) = \begin{cases} 0.1 & 0 \leq t \leq 1000 \\ 0.01 & 1000 \leq t < 2000 \\ \vdots & \end{cases}$$

- Exponential decay (learning rate decays smoothly over time):

$$\gamma(t) = \gamma_0 \exp(-\alpha t)$$

- Cyclical learning rates (reduce and grow the learning rate repeatedly).

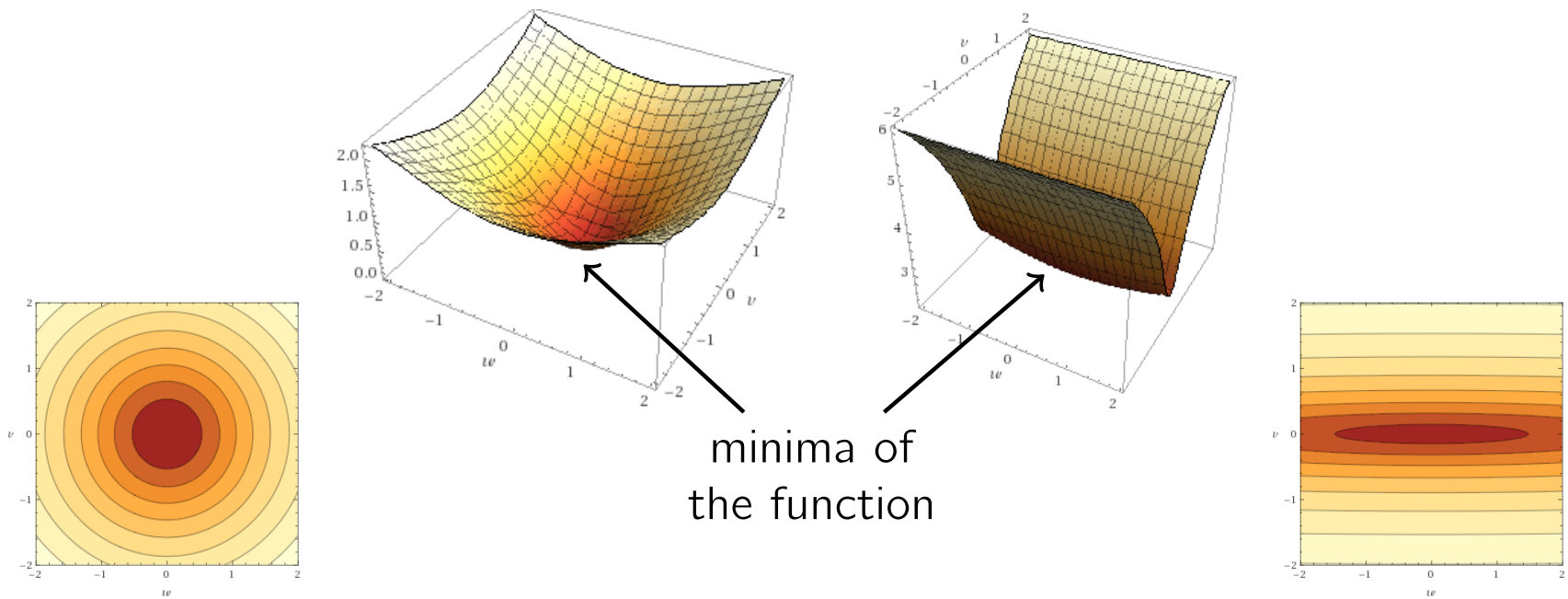
Is it All About Escaping Local Minima?

Answer: No. We must also verify that the function is *well-conditioned*.

Examples:

well-conditioned error function

poorly conditioned error function



Well-conditioned functions are easier to optimize.

Analyzing Gradient Descent (Special Case)

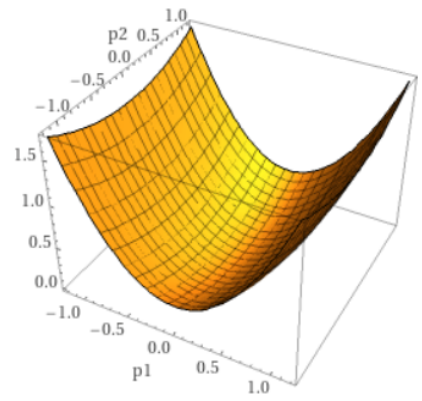
Special case: Suppose the error function takes the simple form:

$$\mathcal{E}(\theta) = \sum_{i=1}^{d} \alpha_i (\theta_i - \theta_i^\star)^2$$

with α_is fixed coefficients that are strictly positive, θ_is parameters that we would like to optimize, and $\theta^\star$ a (unique) minimum of the error function.

Observations:

- The error is easiest to optimize when all dimensions have the same curvature, i.e. $\forall_{ij} : \alpha_i = \alpha_j$.
- The error is hard to optimize when there is a strong divergence of curvature between the different dimensions (e.g. $\exists_{ij} : \alpha_i \gg \alpha_j$).



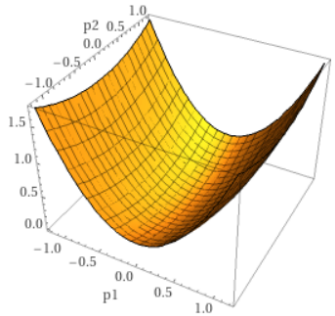
Idea: 当不同维度的曲率差异很大时，误差难以优化（例如，存在 i, j: $\alpha_i \gg \alpha_j$）。这种情况下，某些方向的曲率远大于其他方向，使得梯度下降在一些方向上变得非常慢。

- Quantify the difficulty of optimization by analyzing the process of gradient descent.

Analyzing Gradient Descent (Special Case)

- Recall that we have defined the error function

$$\mathcal{E}(\theta) = \sum_{i=1}^{d} \alpha_i (\theta_i - \theta_i^\star)^2$$



- A step along the gradient direction $\partial\mathcal{E}/\partial\theta_i$ gives:

$$\theta_i^{(\text{new})} = \theta_i - \gamma \cdot 2\alpha_i(\theta_i - \theta_i^\star)$$

- From it, one can characterize the convergence of gradient descent:

$$\begin{aligned}
\theta_i^{(\text{new})} &= \theta_i - \gamma \cdot 2\alpha_i(\theta_i - \theta_i^\star) \\
\theta_i^{(\text{new})} - \theta_i^\star &= \theta_i - \gamma \cdot 2\alpha_i\theta_i + \gamma \cdot 2\alpha_i\theta_i^\star - \theta_i^\star \\
\theta_i^{(\text{new})} - \theta_i^\star &= (1 - 2\gamma\alpha_i) \cdot (\theta_i - \theta_i^\star) \\
(\theta_i^{(\text{new})} - \theta_i^\star)^2 &= (1 - 2\gamma\alpha_i)^2 \cdot (\theta_i - \theta_i^\star)^2
\end{aligned}$$

Analyzing Gradient Descent (Special Case)

- Recall that:

$$(\theta_i^{(\text{new})} - \theta_i^\star)^2 = (1 - 2\gamma\alpha_i)^2 \cdot (\theta_i - \theta_i^\star)^2$$

- Applying t steps of gradient descent from an initial solution $\theta^{(0)}$, we get

$$(\theta_i^{(1)} - \theta_i^\star)^2 = (1 - 2\gamma\alpha_i)^2 \cdot (\theta_i^{(0)} - \theta_i^\star)$$

$$(\theta_i^{(2)} - \theta_i^\star)^2 = (1 - 2\gamma\alpha_i)^2 \cdot \overbrace{(1 - 2\gamma\alpha_i)^2 \cdot (\theta_i^{(0)} - \theta_i^\star)}^{(\theta_i^{(1)} - \theta_i^\star)^2}$$

$$\vdots$$

$$(\theta_i^{(T)} - \theta_i^\star)^2 = \underbrace{(1 - 2\gamma\alpha_i)^2 \cdot \ldots \cdot (1 - 2\gamma\alpha_i)^2}_{(1 - 2\gamma\alpha_i)^{2T}} \cdot (\theta_i^{(0)} - \theta_i^\star)^2$$

如果误差与最优点的平方距离在所有维度上都减少，即对于所有 α_i 满足 $|1 - 2\gamma\alpha_i| < 1$，那么总体的距离也会随着迭代次数呈指数级减小。

同样地，误差函数 $\mathcal{E}(\theta)$ 作为这些平方距离的线性组合，也会随着迭代次数呈指数级减小。

If the squared distance to the optimum decreases along all dimensions, i.e. if $|1 - 2\gamma\alpha_i| < 1$ for all α_i, then the overall distance to the optimum also decreases *exponentially fast* with the number of iterations.

- Likewise, $\mathcal{E}(\theta)$ being a linear combination of these square distances, it also decreases *exponentially fast* with the number of iterations.

Analyzing Gradient Descent (Special Case)

- Recall that gradient descent converges if for all dimensions $i = 1 \dots d$,

$$|1 - 2\gamma\alpha_i| < 1$$

of equivalently

$$0 < \gamma < \frac{1}{\alpha_i}$$

- Let us choose the maximum learning rate that avoids diverging along any of the dimensions:

$$\gamma^{(\text{best})} = 0.99 \cdot \min_i \frac{1}{\alpha_i} = 0.99 \cdot \frac{1}{\alpha_{\max}},$$

where $\alpha_{\max}$ is the coefficient of the dimension with highest curvature.

- Using this learning rate, the convergence rate along the direction of lowest curvature (with coefficient $\alpha_{\min}$) can be expressed as:

$$|1 - 2\gamma^{(\text{best})}\alpha_{\min}| = \left|1 - 2 \cdot 0.99 \frac{\alpha_{\min}}{\alpha_{\max}}\right|$$

the higher the ratio $\alpha_{\min}/\alpha_{\max}$ the faster it converges.

- The *difficulty* to optimize can therefore be quantified by the inverse ratio $\alpha_{\max}/\alpha_{\min}$, known as the *condition number*.

Analyzing Gradient Descent (General Case)

- The analysis in the previous slides assume a very specific form of $\mathcal{E}(\theta)$, where the parameters to not interact.
- However, using the framework of Taylor expansions, any error function can be rewritten near some local minimum $\theta^\star$ as:

$$\mathcal{E}(\theta) = \mathcal{E}(\theta^\star) + 0 + \overbrace{\frac{1}{2}(\theta - \theta^\star)^\top \underbrace{\frac{\partial^2 \mathcal{E}}{\partial\theta\partial\theta^\top}\Big|_{\theta=\theta^\star}}_{H}(\theta - \theta^\star)}^{\widetilde{\mathcal{E}}(\theta)} + \text{higher-order terms}$$

where H is the Hessian, a matrix of size $|\theta| \times |\theta|$ where $|\theta|$ denotes the number of parameters in the network.

Analyzing Gradient Descent (General Case)

- Let us start from the Hessian-based local approximation of the error function:

$$\widetilde{\mathcal{E}}(\theta) = \frac{1}{2}(\theta - \theta^\star)^\top H (\theta - \theta^\star)$$

- Diagonalizing the Hessian matrix, i.e. $H = \sum_{i=1}^{d} \lambda_i u_i u_i^\top$ with $\lambda_1, \ldots, \lambda_d$ the eigenvalues, we can rewrite the error as:

$$\begin{aligned}\widetilde{\mathcal{E}}(\theta) &= \frac{1}{2}(\theta - \theta^\star)^\top \Big(\sum_{i=1}^{d} \lambda_i u_i u_i^\top \Big)(\theta - \theta^\star) \\ &\vdots \\ &= \sum_{i=1}^{d} \frac{1}{2} \lambda_i ((\theta - \theta^\star)^\top u_i)^2\end{aligned}$$

- Repeating the analysis from before, but replacing the individual dimensions by the projections on eigenvectors, we get the condition number:

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

Exercise: Deriving the Hessian of an Error Function

Consider the simple linear model with mean square error

$$\mathcal{E}(\theta) = \mathbb{E}[(\boldsymbol{w}^\top \boldsymbol{x} - t)^2] + \lambda \|\boldsymbol{w}\|^2$$

where $\mathbb{E}[\cdot]$ denotes the expectation over the training data. *Derive* its Hessian.

Elements of the Hessian can be obtained by differenting the function twice:

$$\frac{\partial}{\partial w_i}\mathcal{E}(\theta) = 2\mathbb{E}[(\boldsymbol{w}^\top \boldsymbol{x} - t)x_i] + 2\lambda w_i$$

$$H_{ij} = \frac{\partial}{\partial w_j}\left(\frac{\partial}{\partial w_i}\mathcal{E}(\theta)\right) = 2\mathbb{E}[x_i x_j] + 2\lambda 1_{i=j}$$

The matrix can then also be stated in terms of vector operations:

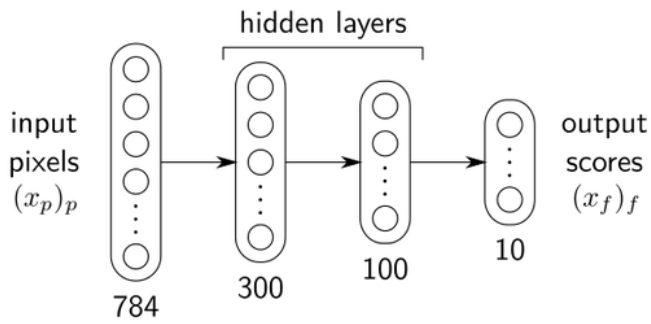
$$H = 2\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top] + 2\lambda I$$

Computing the Hessian in Practice?

Problem:

- The Hessian H (from which one can extract the condition number) is hard to compute and very large for neural networks with many parameters (e.g. fully connected networks).

Example:



$$
\begin{aligned}
|\theta| &= 784 \cdot 300 + 300 \cdot 100 + 100 \cdot 10 \\
&= 266200 \\
|H| &= 266200 \cdot 266200 \\
&= 7.086 \cdot 10^{10} \sim 283 \text{ gigabytes}
\end{aligned}
$$

Idea:

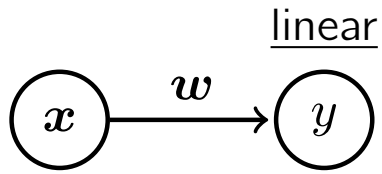
- For most practical tasks, we don't need to evaluate the Hessian and the condition number. We only need to apply a set of recommendations and tricks that keep the condition number low.

Part 3

Improving the Conditioning

Improving Conditioning of the Error Function

Example: The linear model



$$y = \boldsymbol{w}^\top \boldsymbol{x}$$

$$\begin{aligned}
\mathcal{E}(\boldsymbol{w}) &= \mathbb{E}[(\boldsymbol{w}^\top \boldsymbol{x} - t)^2] + \lambda \|\boldsymbol{w}\|^2 \\
&= \boldsymbol{w}^\top \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top + \lambda I]\boldsymbol{w} + \text{linear} + \text{constant} \\
&= \boldsymbol{w}^\top \underbrace{\mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top + \boldsymbol{\mu}\boldsymbol{\mu}^\top + \lambda I]}_{\propto \text{ Hessian}} \boldsymbol{w} + \text{linear} + \text{constant}
\end{aligned}$$

Observation:

- Hessian (and condition number) are influenced by the mean and covariance of the data.
- The closer the mean is to zero, and the closer the covariance is to the identity, the lower the condition number.

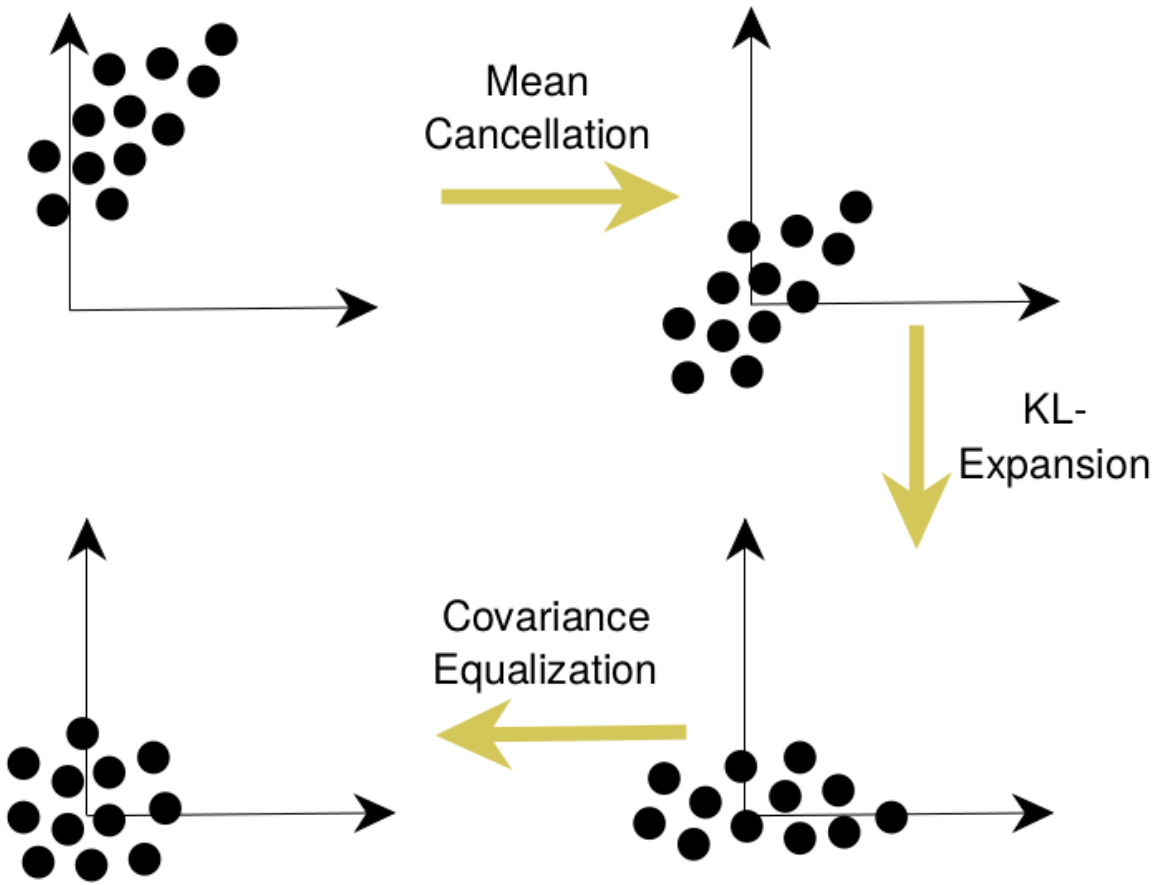
均值越接近零，协方差矩阵越接近单位矩阵，条件数就越低。

Trick: Normalize the data

Data Normalization to Improve Conditioning

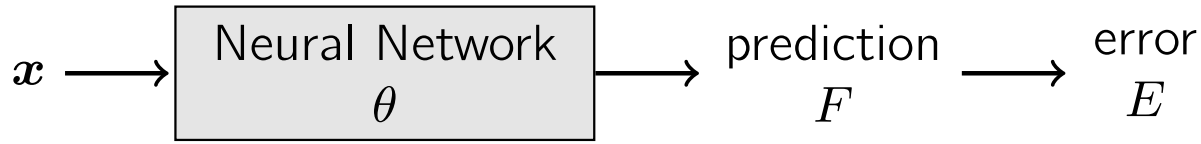
Data pre-processing *before* training:

$(x_1, ..., x_N)$



(image from LeCun'98/12)

Decomposition of the Hessian



General formula for the Hessian of a neural network (size: $|\theta| \times |\theta|$)

$$H = \frac{\partial^2 \mathcal{E}}{\partial \theta^2} = \frac{\partial F}{\partial \theta}^\top \frac{\partial^2 \mathcal{E}}{\partial F^2} \frac{\partial F}{\partial \theta} + \frac{\partial \mathcal{E}}{\partial F} \frac{\partial^2 F}{\partial \theta^2}$$

Hessian between weights of a single neuron (mean square error case):

$$[H_k]_{jj'} = \frac{\partial^2 \mathcal{E}}{\partial w_{jk} w_{j'k}} = \mathbb{E}\big[\underbrace{a_j a_{j'} \delta_k^2}_{\substack{\text{similar to} \\ \text{the simple} \\ \text{linear} \\ \text{model}}}\big] + \mathbb{E}\big[\underbrace{a_j \cdot \frac{\partial \delta_k}{\partial w_{j'k}} \cdot (y - t)}_{\text{complicated}}\big]$$

where δ_k denotes the derivative of the neural network output w.r.t. the pre-activation of neuron k.

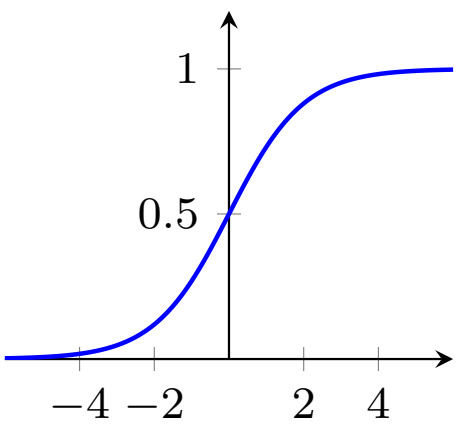
Improving Conditioning of Higher-Layers

To improve conditioning, not only the input data should be normalized, but also the representations built from this data at each layer. This can be done by carefully choosing the activation function.

图1：改善条件数与激活函数选择

- 为了改善神经网络的训练效果，不仅需要对输入调整。这可以通过仔细选择激活函数来实现。
- 左边图：**Logistic Sigmoid 函数**
  - 激活值不是以零为中心。
  - 导致高条件数（高条件数使得训练更不稳定
- 右边图：**双曲正切（tanh）函数**
  - 激活值大致以零为中心。
  - 导致较低的条件数（低条件数使得训练更加

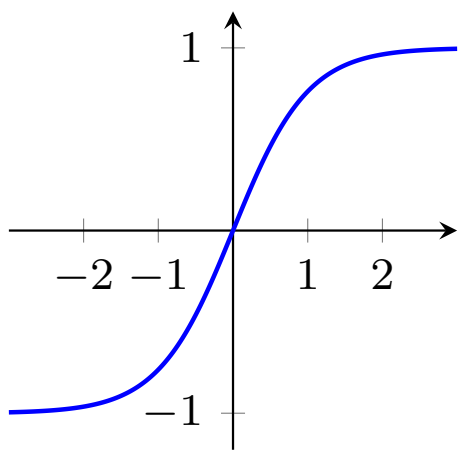
logistic sigmoid



activations are not centered

$\Rightarrow$ **high condition number**

hyperbolic tangent



activations approximately centered at zero

$\Rightarrow$ **low condition number**

Limitation of tanh

The tanh non-linearity works well initially, but after some training steps, it might no longer work as expected as the input distribution will drift to negative or positive values.

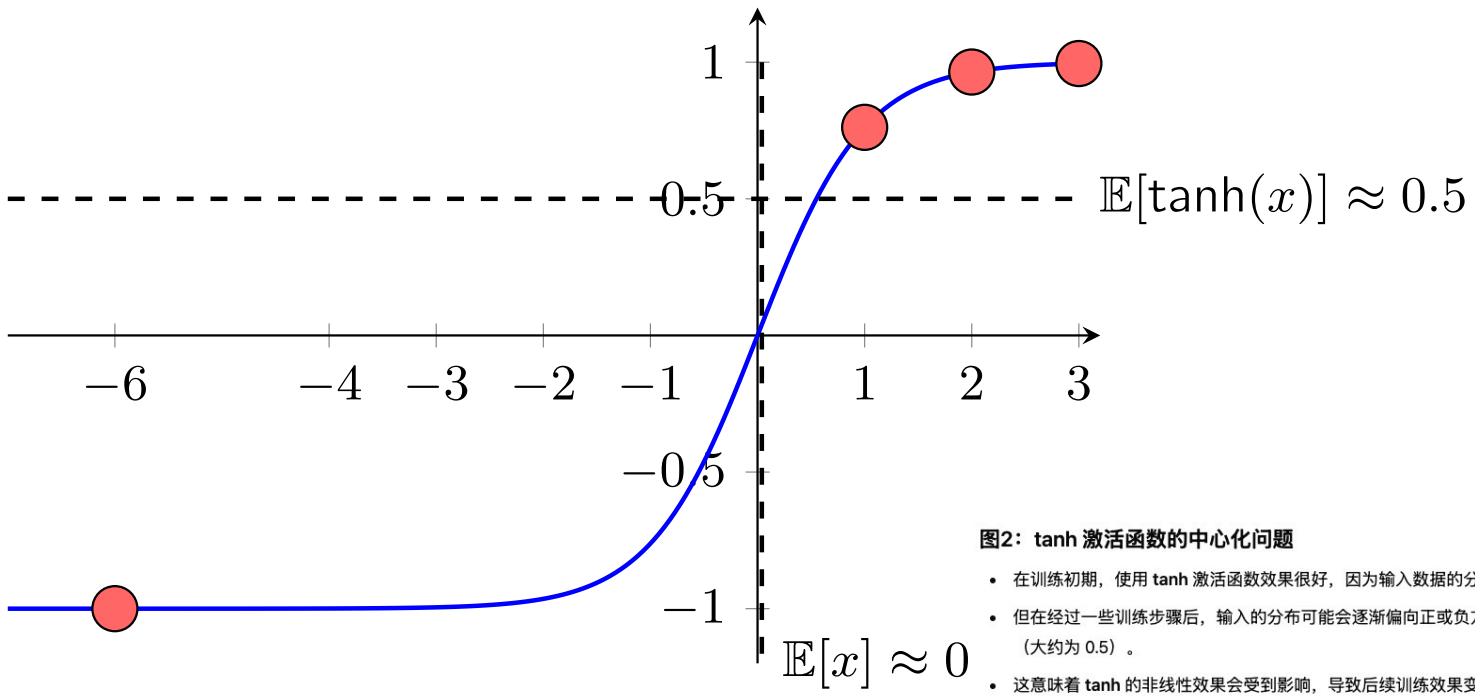


图2：tanh 激活函数的中心化问题

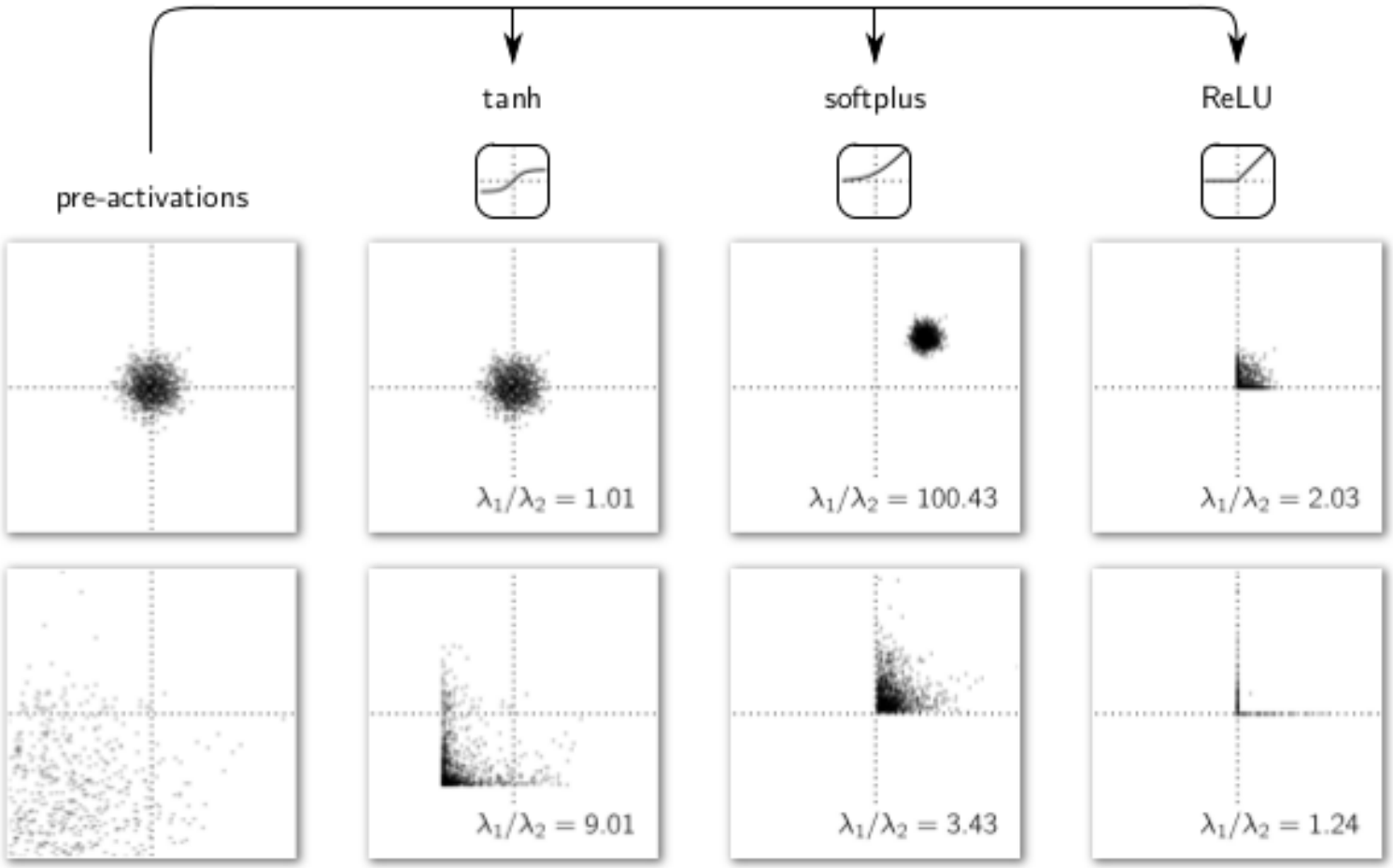
- 在训练初期，使用 tanh 激活函数效果很好，因为输入数据的分布接近零中心。
- 但在经过一些训练步骤后，输入的分布可能会逐渐偏向正或负方向，导致输出的期望值偏离零（大约为 0.5）。
- 这意味着 tanh 的非线性效果会受到影响，导致后续训练效果变差。

备注：

- 如果 tanh 的输入是中心化的但有偏斜，那么 tanh 的输出也将偏离中心。这在实际中很常见，尤其是在需要稀疏表示的问题中。

Remark: If the input of tanh is centered but skewed, the output of tanh will not be centered. This happens a lot in practice, e.g. when the problem representation needs to be sparse.

Comparing Non-Linearities



Further Improving the Hessian

Recommendation

- Scale parameters such that neuron outputs have variance ≈ 1 initially (LeCun'98/12 "Efficient Backprop")

$$\theta \sim \mathcal{N}(0, \sigma^2) \qquad \sigma^2 = \frac{1}{\#\text{input neurons}} \tag{1}$$

- Use a similar number of neurons in each layer.

- 构建 Hessian 矩阵的近似，其中忽略了不同神经元之间参数的相互作用。这种近似可以形成块对角矩阵的形式：

$$H = \text{diag}\{H_j, H_j', H_j'', \ldots, H_k, H_k', H_k'', \ldots, H_{\text{out}}\}$$

这种近似将 Hessian 分为若干块，每个块对应不同的神经元。

A Hessian-based justification:

- Build an approximation of the Hessian where interactions between parameters of different neurons are neglected. Such approximation takes the form of a block-diagonal matrix:

$$H = \text{diag}\{H_j, H_{j'}, H_{j''}, \ldots, H_k, H_{k'}, H_{k''}, \ldots, H_{\text{out}}\}$$

- Eigenvalues of H are given by the eigenvalues of the different blocks. Reducing the condition number requires ensuring each block has eigenvalues on a similar scale.

- Recall that the Hessian associated to a given neuron is of the form $[H_k]_{jj'} = 2\mathbb{E}[a_j a_{j'} \delta_k^2]$. This implies that activations and sensitivities to the output needs to be on the same scale at each layer.

特征值的计算：

- H 的特征值由各个块的特征值给出。为了减少条件数，需要确保每个块的特征值在相似的量级上。

Hessian 的形式：

- 与某个特定神经元相关的 Hessian 通常具有如下形式：

$$[H_k]_{jj'} = 2\mathbb{E}[a_j a_{j'} \delta_k^2]$$

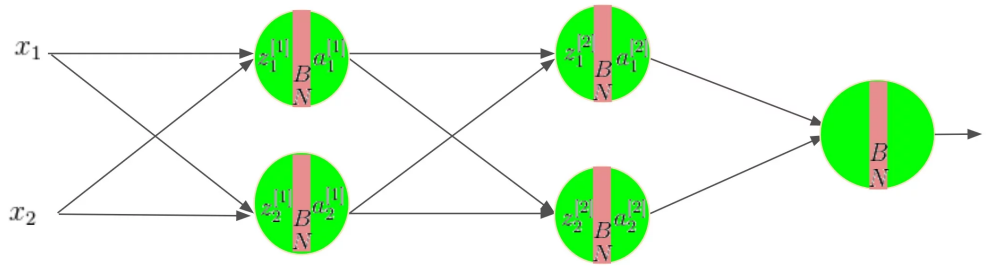
这意味着每一层中的激活值和对输出的敏感度需要保持在相同的尺度上，以确保训练过程的稳定

Further Improving Optimization / the Hessian

通过对神经网络的中间层激活值进行标准化，改善训练过程，使梯度的传播更加稳定。

Batch Normalization

(Ioffe et al.
arXiv:1502.03167, 2015)



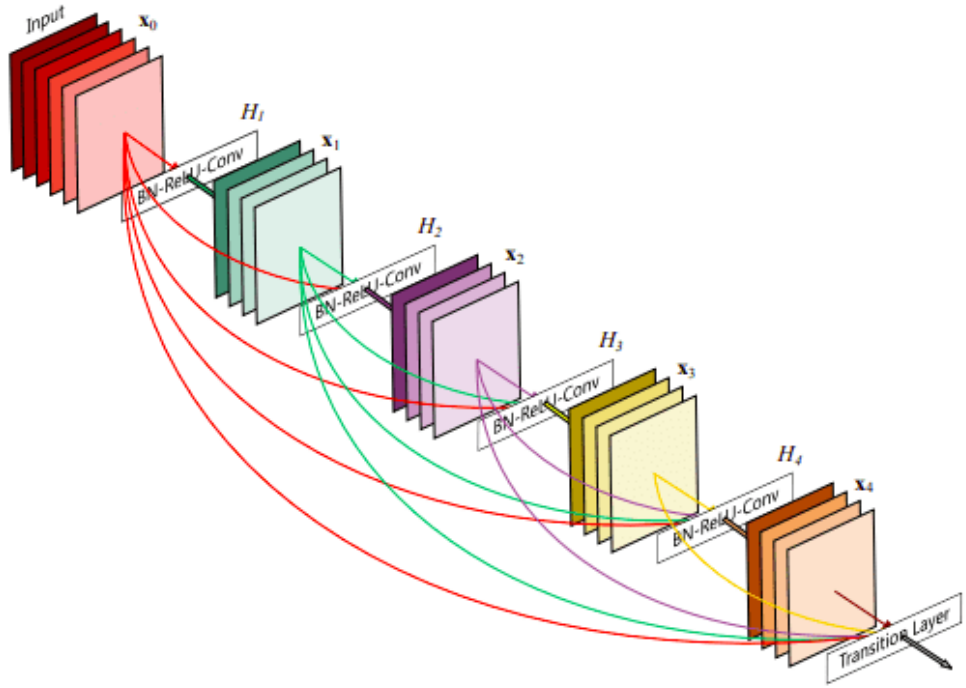
$$z^{[l]} = W^{[l]} a^{[l-1]} \longrightarrow \begin{aligned} \mu^{[l]} &= \frac{1}{m}\sum_i z^{[l](i)} \\ \sigma^{[l]2} &= \frac{1}{m}\sum_i (z^{[l](i)} - \mu^{[l]})^2 \\ z^{[l](i)}_{norm} &= \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}} \\ \tilde{z}^{[l](i)} &= \gamma^{[l]} z^{[l](i)}_{norm} + \beta^{[l]} \end{aligned} \longrightarrow a^{[l]} = g^{[l]}(\tilde{z}^{[l]})$$

Advantages:

- Ensures activations in multiple layers are centered.
- Reduce interactions between parameters at multiple layers.

Further Improving Optimization / the Hessian

Skip connections:



Advantages:

- Better propagate the relevant signal to the output of the network.
- Reduce interactions between parameters at different layers.

Summary

Summary

神经网络非常**强大**，但也**难以优化**（例如，非凸性，条件较差等）。

- **非凸性**是无法避免的，但可以通过选择合适的神经网络架构和参数初始化来减轻其不利影响。
- **条件较差**的问题，可以通过分析 Hessian 矩阵来应对。可通过应用一些技巧，如数据和表示的中心化、均衡各层激活值的尺度，以及减少各层之间的参数交互来改善条件数。
- 还有很多优化方面没有被讨论，例如优化过程本身，避免冗余计算，实现细节和分布式机器学习方案。这些内容将在第四讲中详细讲解。

- Neural networks are **powerful** but also **difficult to optimize** (e.g. non-convex, poorly conditioned, etc.)
- **Non-convexity** cannot be avoided, however, its adverse effects can be mitigated by selecting an appropriate neural network architecture and initialization of the parameters.
- **Poor conditioning**, characterized by analyzing the Hessian, can be tackled by applying different tricks such as centering data and representations, homogeneizing scales of activations across various layers and reducing interaction between parameters of diffent layers. Many of these tricks can be justified as improving the condition number.
- There are many more aspects of optimization that have not been covered yet. These include the optimization procedure itself, avoiding redundant computations, implementation aspects, and distributed ML schemes. They will be the focus of Lecture 4.