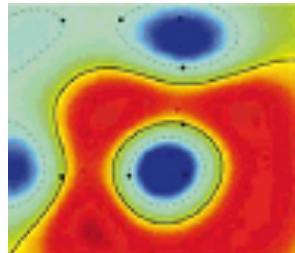
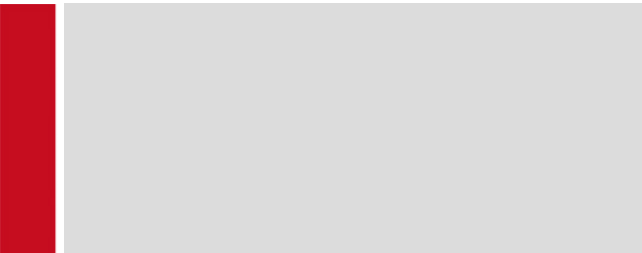




WiSe 2024/25

Deep Learning 1



Lecture 3

Optimization (Part 1)

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

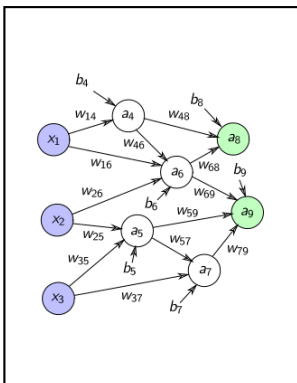
Recap Lecture 2

Recap: How to Learn in a Neural Network

Observation:

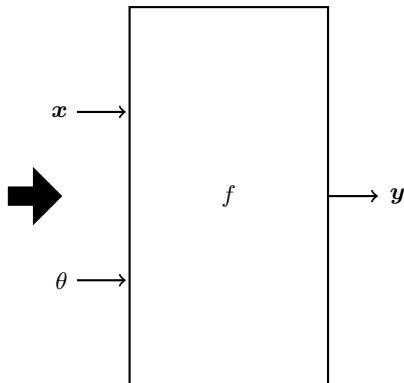
A neural network is a function of both its inputs and parameters.

graph view



$$\mathbf{x} = [x_1, x_2, x_3] \quad \mathbf{y} = [a_8, a_9]$$

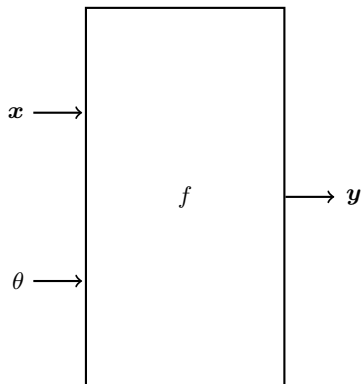
function view



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

Recap: How to Learn in a Neural Network

function view



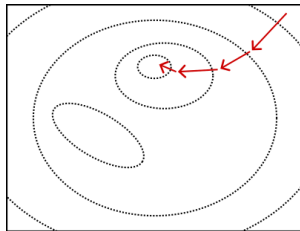
$$\theta = (w_{ij})_{ij}, (b_j)_j$$

Define an error function

$$\mathcal{E}(\theta) = \sum_{n=1}^N (f(\mathbf{x}_n; \theta) - t_n)^2$$

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:

linear

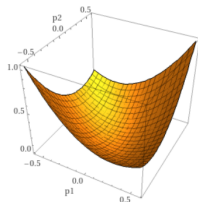
$x \xrightarrow{w} y$

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

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

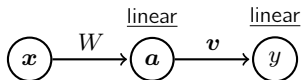
$$\mathcal{E}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^\top \mathbf{x}_n - t_n)^2 + \lambda \|\mathbf{w}\|^2$$

- 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.



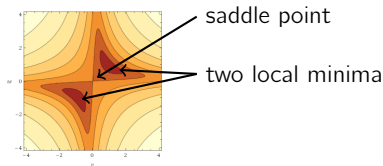
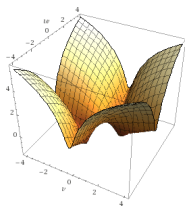
Characterizing the Error Function: Two Layers

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



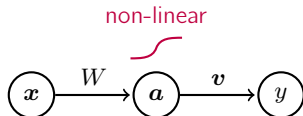
$$\mathcal{E}(W, \mathbf{v}) = \frac{1}{N} \sum_{n=1}^N (\mathbf{v}^\top W \mathbf{x}_n - t_n)^2 + \lambda (\|\mathbf{v}\|^2 + \|W\|_F^2)$$

- 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:



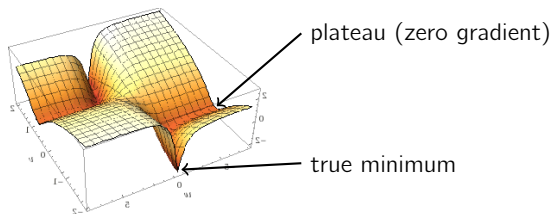
Characterizing the Error Function: Two Layers

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



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

- ▶ 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.



Practical Recommendations

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).

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

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):

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

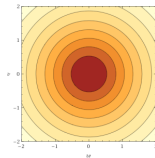
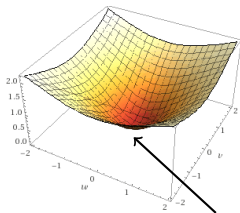
- ▶ Cyclical learning rates (reduce and grow the learning rate repeatedly).

Is it All About Escaping Local Minima?

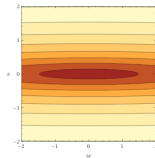
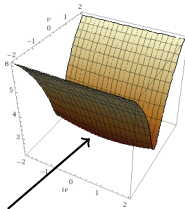
Answer: No. We must also verify that the function is *well-conditioned*.

Examples:

well-conditioned
error function



poorly conditioned
error function



minima of
the function

Well-conditioned functions
are easier to optimize.

Analyzing Gradient Descent (Special Case)

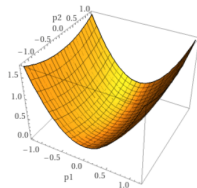
Special case: Suppose the error function takes the simple form:

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

with α_i s fixed coefficients that are strictly positive, θ_i s parameters that we would like to optimize, and θ^* 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:

- ▶ 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

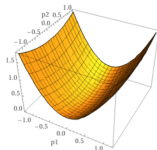
$$\mathcal{E}(\theta) = \sum_{i=1}^d \alpha_i (\theta_i - \theta_i^*)^2$$

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

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

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

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



Analyzing Gradient Descent (Special Case)

- Recall that:

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

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

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

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

\vdots

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

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 α_{\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 α_{\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 do not interact.
- ▶ However, using the framework of Taylor expansions, any error function can be rewritten near some local minimum θ^* as:

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

where $\textcolor{green}{H}$ 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:

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

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

$$\begin{aligned}\tilde{\mathcal{E}}(\theta) &= \frac{1}{2}(\theta - \theta^*)^\top \left(\sum_{i=1}^d \lambda_i u_i u_i^\top \right) (\theta - \theta^*) \\ &\vdots \\ &= \sum_{i=1}^d \frac{1}{2} \lambda_i ((\theta - \theta^*)^\top u_i)^2\end{aligned}$$

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

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

Exercise: Deriving the Hessian of an Error Function

Consider the simple linear model with mean square error

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

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

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

$$\begin{aligned}\frac{\partial}{\partial w_i} \mathcal{E}(\theta) &= 2\mathbb{E}[(\mathbf{w}^\top \mathbf{x} - t)x_i] + 2\lambda w_i \\ H_{ij} &= \frac{\partial}{\partial w_j} \left(\frac{\partial}{\partial w_i} \mathcal{E}(\theta) \right) = 2\mathbb{E}[x_i x_j] + 2\lambda 1_{i=j}\end{aligned}$$

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

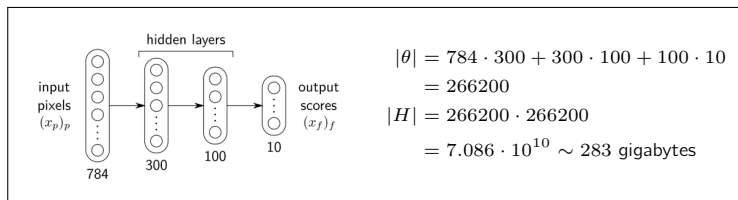
$$H = 2\mathbb{E}[\mathbf{x}\mathbf{x}^\top] + 2\lambda I$$

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:



Idea:

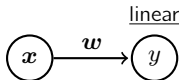
- ▶ For most practical tasks, we don't need to evaluate the Hessian and the condition number. We only need to apply a set of recommendations and tricks that keep the condition number low.

Part 3

Improving the Conditioning

Improving Conditioning of the Error Function

Example: The linear model



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

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

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}]$, i.e. the dataset mean.

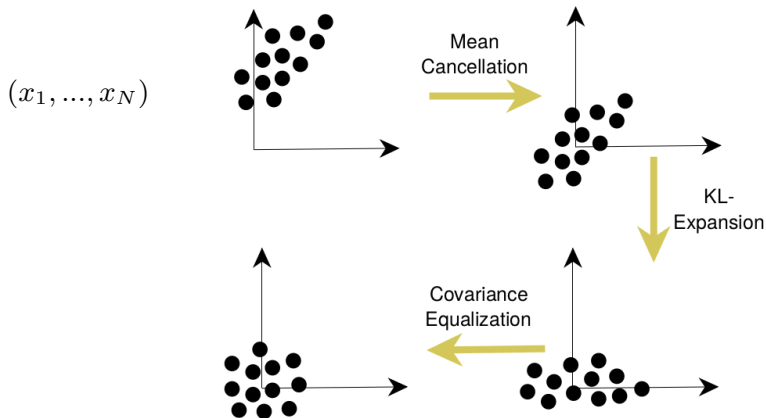
Observation:

- ▶ The Hessian, and therefore the condition number, are influenced by the mean $\boldsymbol{\mu}$ (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.

Trick: Normalize the data

Data Normalization to Improve Conditioning

Data pre-processing *before* training:



(image from LeCun'98/12)

Decomposition of the Hessian



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

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

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

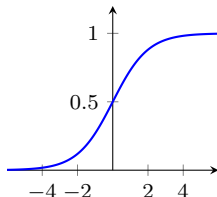
$$[H_k]_{jj'} = \frac{\partial^2 \mathcal{E}}{\partial w_{jk} \partial w_{j'k}} = \mathbb{E} \left[\underbrace{a_j a_{j'} \delta_k^2}_{\substack{\text{similar to} \\ \text{the simple} \\ \text{linear} \\ \text{model}}} \right] + \mathbb{E} \left[\underbrace{a_j \cdot \frac{\partial \delta_k}{\partial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

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.

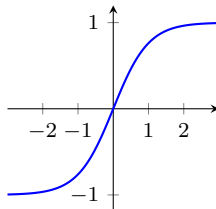
logistic sigmoid



activations are
not centered

⇒ **high condition number**

hyperbolic tangent

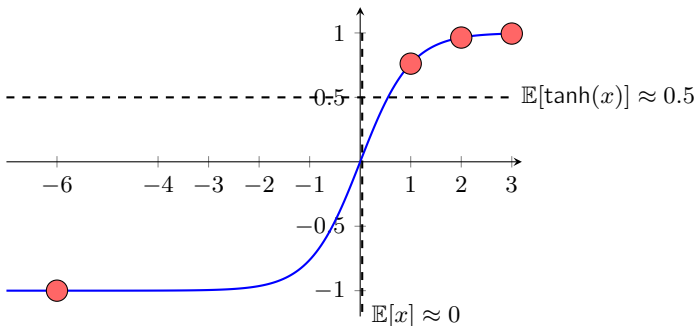


activations approximately
centered at zero

⇒ **low condition number**

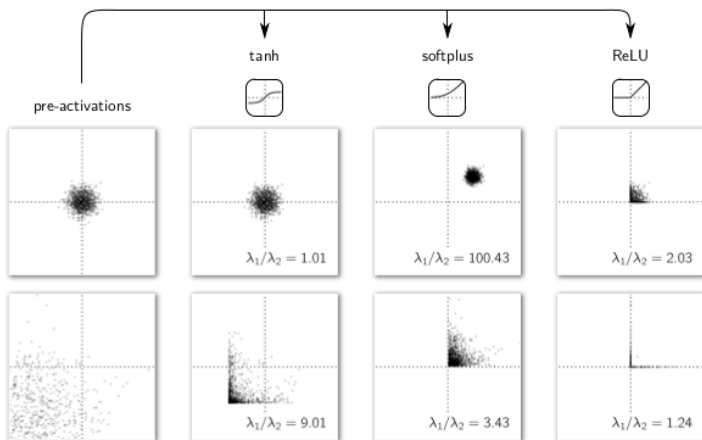
Limitation of tanh

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.



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")

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

- ▶ Use a similar number of neurons in each layer.

[Optional material] 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:

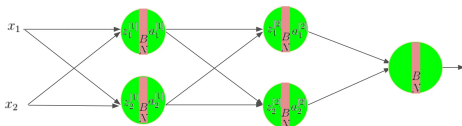
$$H = \text{diag}\{H_j, H_{j'}, H_{j''}, \dots, H_k, H_{k'}, H_{k''}, \dots, H_{\text{out}}\}$$

- ▶ 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.

Further Improving Optimization / the Hessian

Batch Normalization

(Ioffe et al.
arXiv:1502.03167, 2015)



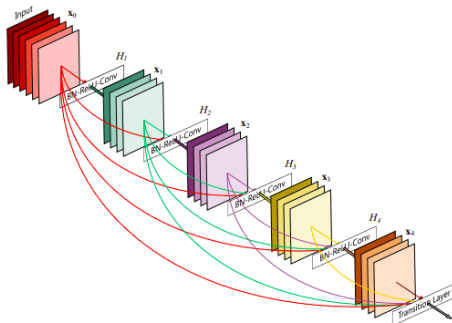
$$z^{[l]} = W^{[l]} a^{[l-1]} \longrightarrow \begin{aligned} \mu^{[l]} &= \frac{1}{m} \sum_i z^{[l](i)} \\ \sigma^{[l]2} &= \frac{1}{m} \sum_i (z^{[l](i)} - \mu^{[l]})^2 \\ z_{norm}^{[l](i)} &= \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}} \\ \tilde{z}^{[l](i)} &= \gamma^{[l]} z_{norm}^{[l](i)}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

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

- ▶ 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, homogenizing scales of activations across various layers and reducing interaction between parameters of different 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.