

Deep Learning

Pontus Giselsson

Outline

- **Deep learning**
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Deep learning

- Can be used both for classification and regression
- Deep learning training problem is of the form

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^N L(m(x_i; \theta), y_i)$$

where typically

- $L(u, y) = \frac{1}{2} \|u - y\|_2^2$ is used for regression
- $L(u, y) = \log \left(\sum_{j=1}^K e^{u_j} \right) - y^T u$ is used for K -class classification
- Difference to previous convex methods: *Nonlinear model* $m(x; \theta)$
 - Deep learning regression generalizes least squares
 - DL classification generalizes multiclass logistic regression
 - Nonlinear model makes training problem nonconvex

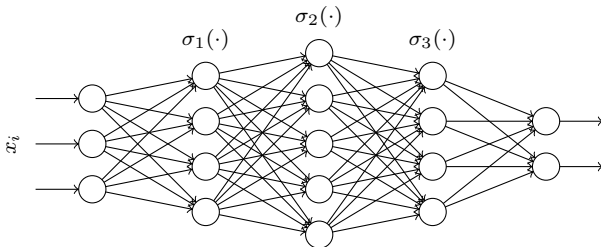
Deep learning – Model

- Nonlinear model of the following form is often used:

$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ contains all W_i and b_i

- Each activation σ_j constitutes a hidden layer in the model network
- We have no final layer activation (is instead part of loss)
- Graphical representation with three hidden layers



- Some reasons for using this structure:
 - (Assumed) universal function approximators
 - Efficient gradient computation using backpropagation

No final layer activation in classification


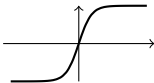
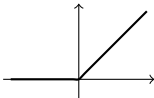
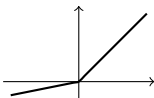

- In classification, it is common to use
 - Softmax final layer activation
 - Cross entropy loss function
- Equivalent to
 - no (identity) final layer activation
 - multiclass logistic loss

which is what we use

Activation functions

- Activation function σ_j takes as input the output of $W_j(\cdot) + b_j$
- Often a function $\bar{\sigma}_j : \mathbb{R} \rightarrow \mathbb{R}$ is applied to each element
 - Example: $\sigma_j : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is $\sigma_j(u) = \begin{bmatrix} \bar{\sigma}_j(u_1) \\ \bar{\sigma}_j(u_2) \\ \bar{\sigma}_j(u_3) \end{bmatrix}$
- We will use notation over-loading and call both functions σ_j

Examples of activation functions

Name	$\sigma(u)$	Graph
Sigmoid	$\frac{1}{1+e^{-u}}$	
Tanh	$\frac{e^u - e^{-u}}{e^{-u} + e^u}$	
ReLU	$\max(u, 0)$	
LeakyReLU	$\max(u, \alpha u)$	
ELU	$\begin{cases} u & \text{if } u \geq 0 \\ \alpha(e^u - 1) & \text{else} \end{cases}$	

Examples of affine transformations

- Dense (fully connected): Dense W_j
- Sparse: Sparse W_j
 - Convolutional layer (convolution with small pictures)
 - Fixed (random) sparsity pattern
- Subsampling: reduce size, W_j fat (smaller output than input)
 - average pooling

Prediction

- Prediction as in least squares and multiclass logistic regression
- Assume model $m(x; \theta)$ trained and “optimal” θ^* found
- Regression:
 - Predict response for new data x using $\hat{y} = m(x; \theta^*)$
- Classification (with no final layer activation):
 - We have one model $m_j(x; \theta^*)$ output for each class
 - Predict class belonging for new data x according to

$$\operatorname{argmax}_{j \in \{1, \dots, K\}} m_j(x; \theta^*)$$

i.e., class with largest model value (since loss designed this way)

Outline

- Deep learning
- **Learning features**
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Learning features

- Convex methods use *prespecified* feature maps (or kernels)
- Deep learning instead *learns* feature map during training
 - Define parameter dependent feature vector:

$$\phi(x; \theta) := \sigma_{n-1}(W_{n-1}\sigma_{n-2}(\cdots(W_2\sigma_1(W_1x+b_1)+b_2)\cdots)+b_{n-1})$$

- Model becomes $m(x; \theta) = W_n\phi(x; \theta) + b_n$
- Inserted into training problem:

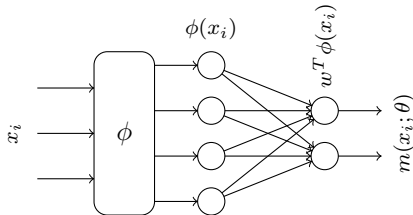
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^N L(W_n\phi(x_i; \theta) + b_n, y_i)$$

same as before, but with learned (parameter-dependent) features

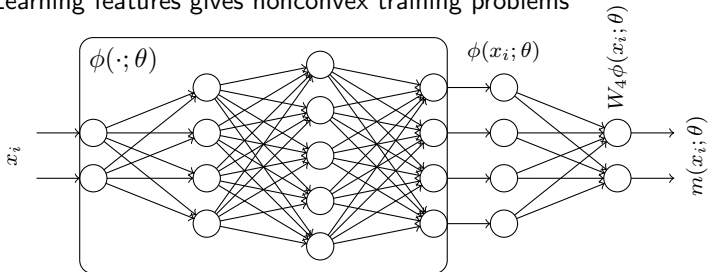
- Learning features at training makes training nonconvex

Learning features – Graphical representation

- Fixed features gives convex training problems



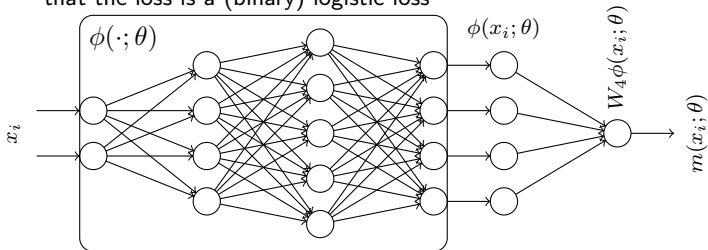
- Learning features gives nonconvex training problems



- Output of last activation function is feature vector

Optimizing only final layer

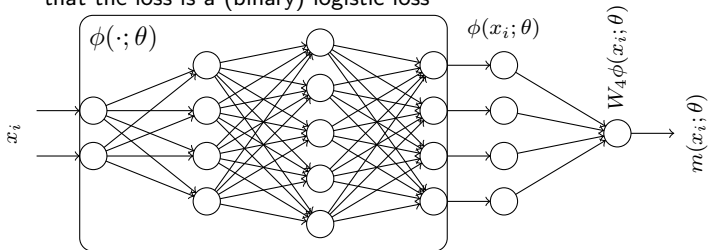
- Assume:
 - that parameters $\bar{\theta}_f$ in the layers in the square are fixed
 - that we optimize only the final layer parameters
 - that the loss is a (binary) logistic loss



- What can you say about the training problem?

Optimizing only final layer

- Assume:
 - that parameters $\bar{\theta}_f$ in the layers in the square are fixed
 - that we optimize only the final layer parameters
 - that the loss is a (binary) logistic loss



- What can you say about the training problem?
 - It reduces to logistic regression with fixed features $\phi(x_i; \bar{\theta}_f)$

$$\underset{\theta=(W_n, b_n)}{\text{minimize}} \sum_{i=1}^N L(W_n \phi(x_i; \bar{\theta}_f) + b_n, y_i)$$

- The training problem is convex

Design choices

Many design choices in building model to create good features

- Number of layers
- Width of layers
- Types of layers
- Types of activation functions
- Different model structures (e.g., residual network)

Outline

- Deep learning
- Learning features
- **Model properties and activation functions**
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Model properties – ReLU networks

- Recall model

$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ contains all W_i and b_i

- Assume that all activation functions are (Leaky)ReLU
- What can you say about the properties of $m(\cdot; \theta)$ for fixed θ ?

Model properties – ReLU networks

- Recall model

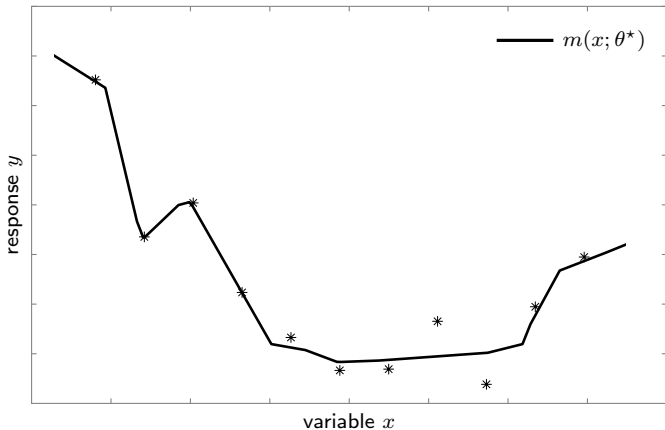
$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ contains all W_i and b_i

- Assume that all activation functions are (Leaky)ReLU
- What can you say about the properties of $m(\cdot; \theta)$ for fixed θ ?
 - It is continuous piece-wise affine

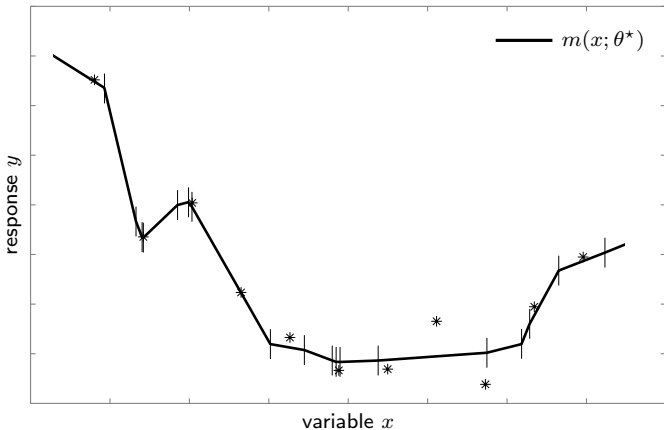
1D Regression – Model properties

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyReLU



1D Regression – Model properties

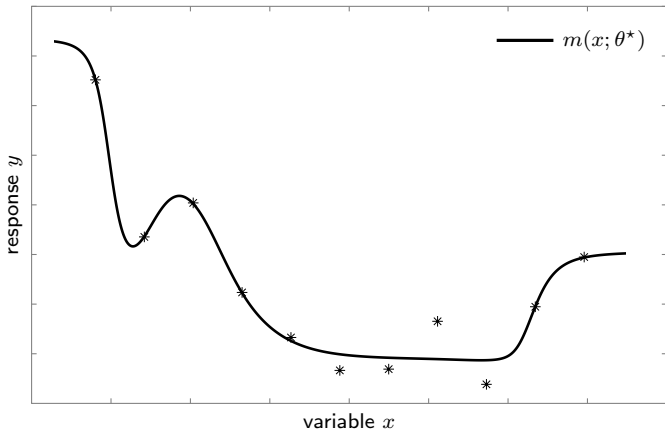
- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyReLU



- Vertical lines show kinks

1D Regression – Model properties

- Fully connected, layers widths: 5,5,5,1,1 (78 params), Tanh



- No kinks for Tanh

Identity activation

- Do we need nonlinear activation functions?
- What can you say about model if all $\sigma_j = \text{Id}$ in

$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ contains all W_j and b_j

Identity activation

- Do we need nonlinear activation functions?
- What can you say about model if all $\sigma_j = \text{Id}$ in

$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ contains all W_j and b_j

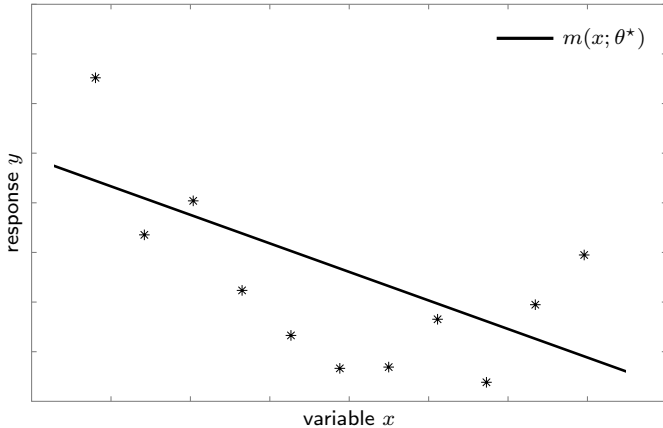
- We then get

$$\begin{aligned} m(x; \theta) &:= W_n(W_{n-1}(\cdots (W_2(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n \\ &= \underbrace{W_n W_{n-1} \cdots W_2 W_1}_{W} x + b_n + \underbrace{\sum_{l=2}^{n-1} W_n \cdots W_l b_{l-1}}_b \\ &= Wx + b \end{aligned}$$

which is linear in x (but training problem nonconvex)

Network with identity activations – Example

- Fully connected, layers widths: 5,5,5,1,1 (78 params), Identity



Outline

- Deep learning
- Learning features
- Model properties and activation functions
- **Loss landscape**
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Training problem properties

- Recall model

$$m(x; \theta) := W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

where θ includes all W_j and b_j and training problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^N L(m(x_i; \theta), y_i)$$

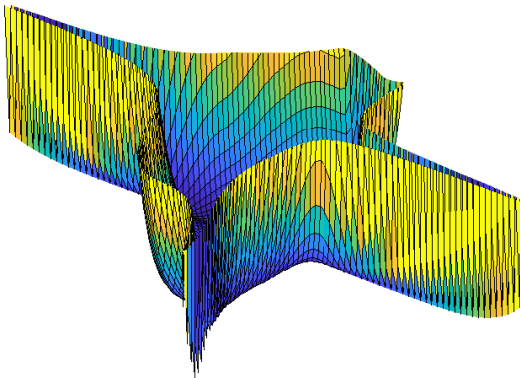
- If all σ_j LeakyReLU and $L(u, y) = \frac{1}{2} \|u - y\|_2^2$, then for fixed x, y
 - $m(x; \cdot)$ is continuous piece-wise polynomial (cpp) of degree n in θ
 - $L(m(x; \theta), y)$ is cpp of degree $2n$ in θ

where both model output and loss can grow fast

- If σ_j is instead Tanh
 - model no longer piece-wise polynomial (but “more” nonlinear)
 - model output grows slower since $\sigma_j : \mathbb{R} \rightarrow (-1, 1)$

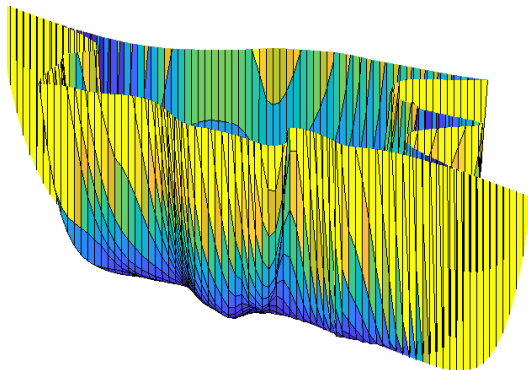
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- First choice of θ_1 and θ_2 :



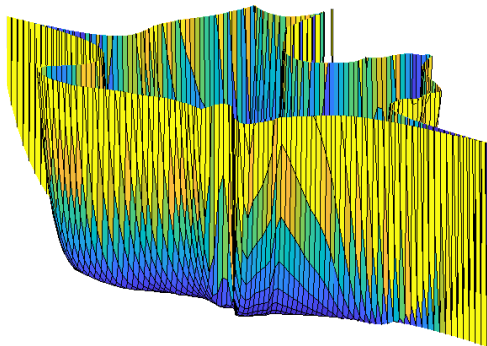
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- Second choice of θ_1 and θ_2 :



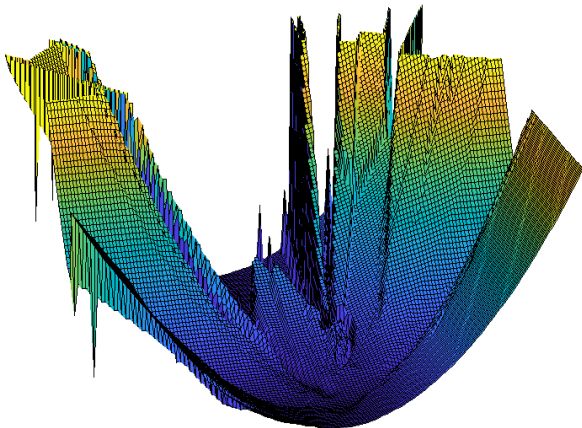
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- Third choice of θ_1 and θ_2 :



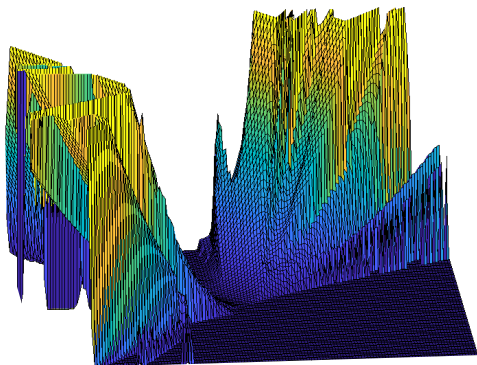
Loss landscape – Tanh

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- First choice of θ_1 and θ_2 :



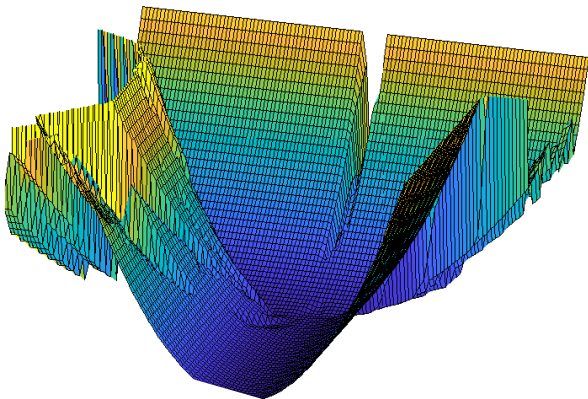
Loss landscape – Tanh

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- Second choice of θ_1 and θ_2 :



Loss landscape – Tanh

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^N L(m(x_i; \theta^* + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2 , where
 - θ^* is numerically found solution to training problem
 - θ_1 and θ_2 are random directions in parameter space
- Third choice of θ_1 and θ_2 :



ReLU vs Tanh

Previous figures suggest:

- ReLU: more regular and similar loss landscape?
- Tanh: less steep (on macro scale)?
- Tanh: Minima extend over larger regions?

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- **Residual networks**
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Performance with increasing depth

- Increasing depth can deteriorate performance
- Deep networks may even have worse training errors than shallow
- Intuition: deeper layers bad at approximating identity mapping

Residual networks

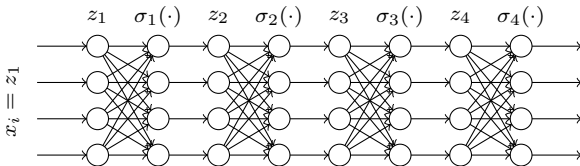
- Add skip connections between layers
- Instead of network architecture with $z_1 = x_i$ (see figure):

$$z_{j+1} = \sigma_j(W_j z_j + b_j) \text{ for } j \in \{1, \dots, n-1\}$$

use residual architecture

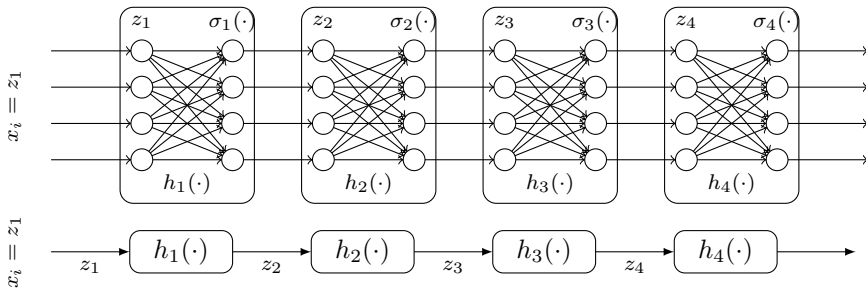
$$z_{j+1} = z_j + \sigma_j(W_j z_j + b_j) \text{ for } j \in \{1, \dots, n-1\}$$

- Assume $\sigma(0) = 0$, $W_j = 0$, $b_j = 0$ for $j = 1, \dots, m$ ($m < n-1$)
 \Rightarrow deeper part of network is identity mapping and does no harm
- Learns variation from identity mapping (residual)



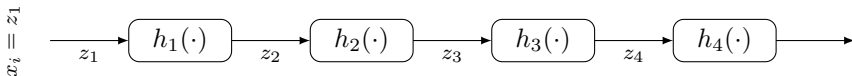
Graphical representation

For graphical representation, first collapse nodes into single node

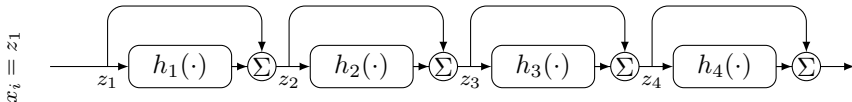


Graphical representation

- Collapsed network representation



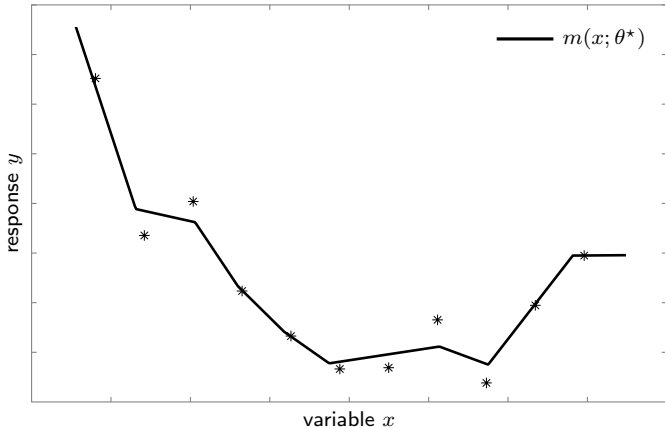
- Residual network



- If some $h_j = 0$ gives same performance as shallower network

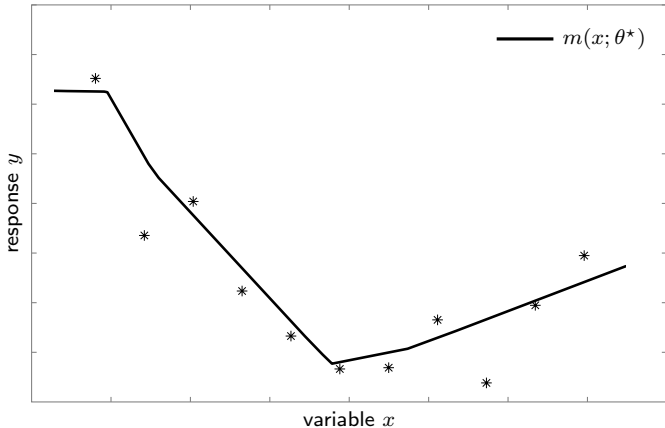
Residual network – Example

- Fully connected – no residual layers, LeakyReLU activation
- Layers widths: 3x5,1,1 (depth: 5, 78 params)
- Trained for 5000 epochs



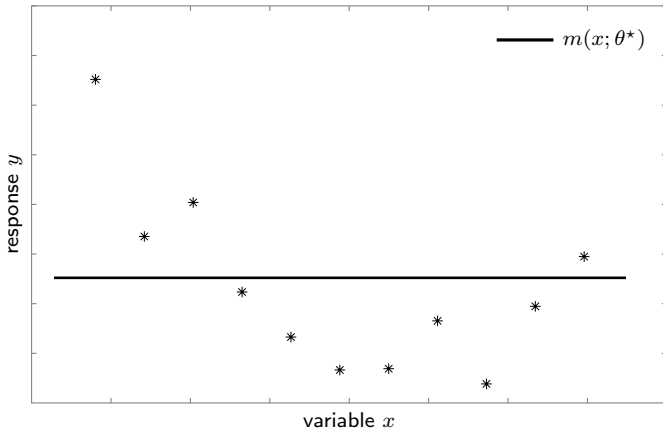
Residual network – Example

- Fully connected – no residual layers, LeakyReLU activation
- Layers widths: 5x5,1,1 (depth: 7, 138 params)
- Trained for 5000 epochs



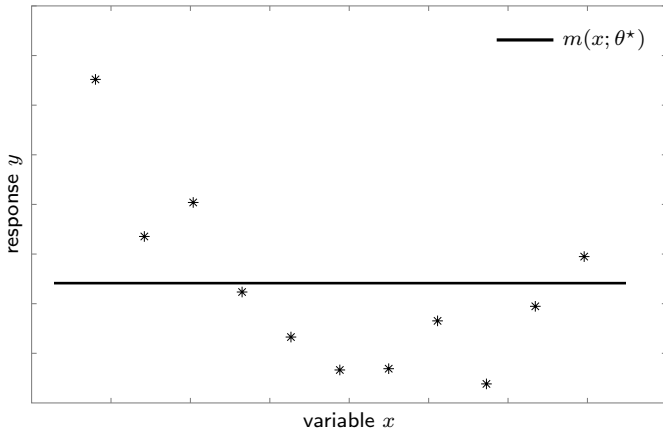
Residual network – Example

- Fully connected – no residual layers, LeakyReLU activation
- Layers widths: 10x5,1,1 (depth: 12, 288 params)
- Trained for 5000 epochs



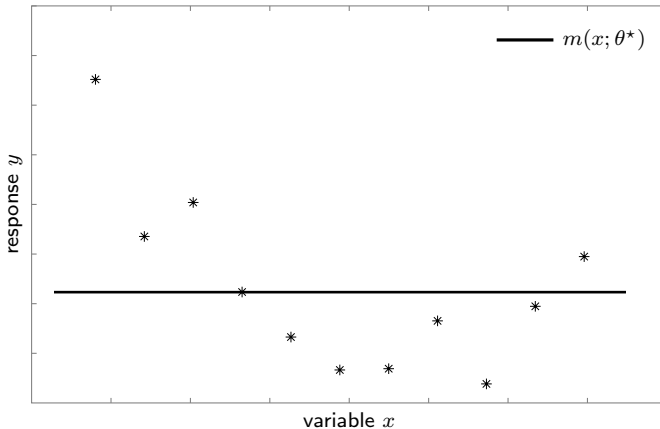
Residual network – Example

- Fully connected – no residual layers, LeakyReLU activation
- Layers widths: 15x5,1,1 (depth: 17, 438 params)
- Trained for 5000 epochs



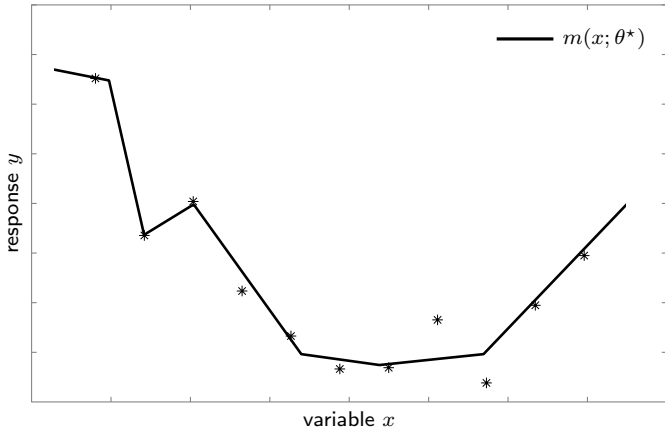
Residual network – Example

- Fully connected – no residual layers, LeakyReLU activation
- Layers widths: 45x5,1,1 (depth: 47, 1,338 params)
- Trained for 5000 epochs



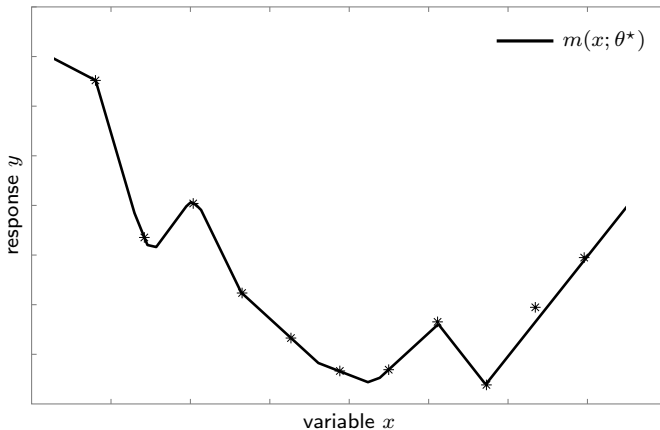
Residual network – Example

- Fully connected – residual layers, LeakyReLU activation
- Layers widths: 3x5,1,1 (depth: 5, 78 params)
- Trained for 5000 epochs



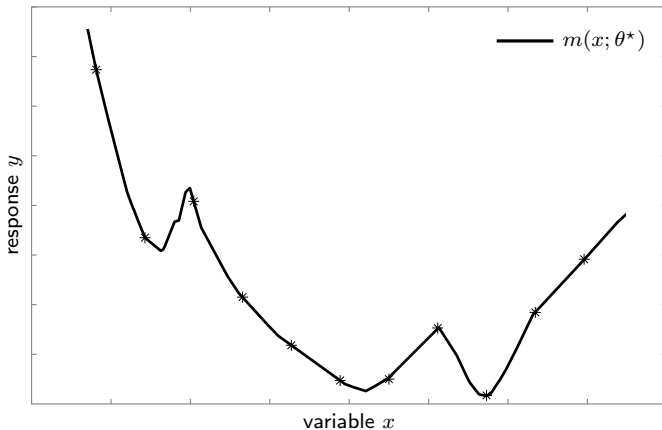
Residual network – Example

- Fully connected – residual layers, LeakyReLU activation
- Layers widths: 5x5,1,1 (depth: 7, 138 params)
- Trained for 5000 epochs



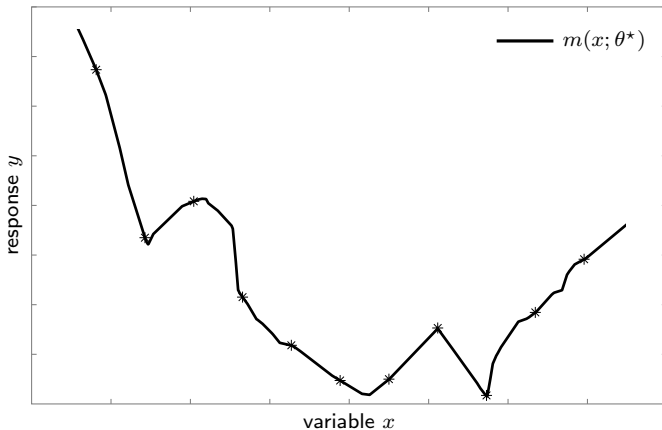
Residual network – Example

- Fully connected – residual layers, LeakyReLU activation
- Layers widths: 10x5,1,1 (depth: 12, 288 params)
- Trained for 5000 epochs



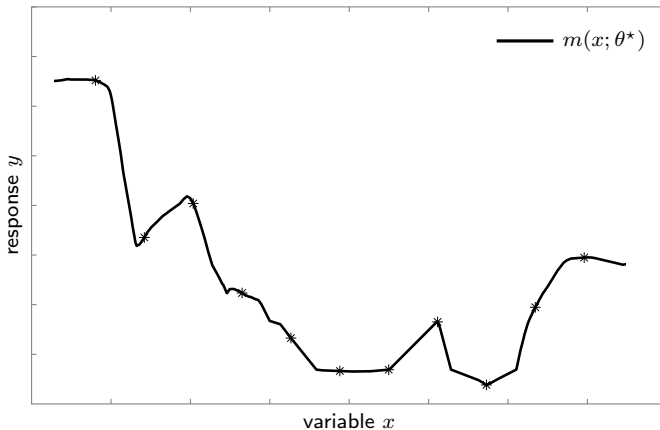
Residual network – Example

- Fully connected – residual layers, LeakyReLU activation
- Layers widths: 15x5,1,1 (depth: 17, 438 params)
- Trained for 5000 epochs



Residual network – Example

- Fully connected – residual layers, LeakyReLU activation
- Layers widths: 45x5,1,1 (depth: 47, 1,338 params)
- Trained for 5000 epochs



Outline

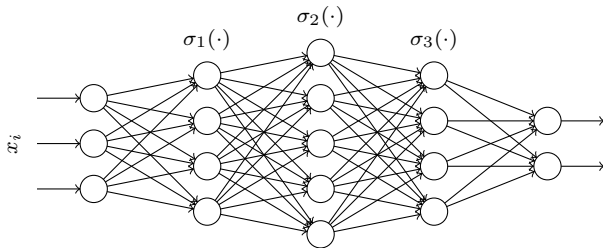
- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- **Overparameterized networks**
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Why overparameterization?

- Neural networks are often overparameterized in practice
- Why? They often perform better than underparameterized

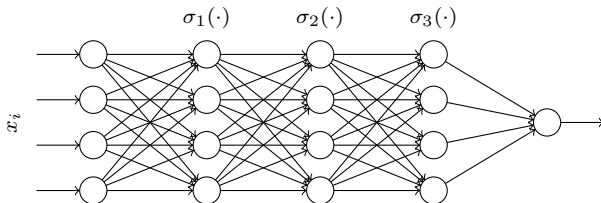
What is overparameterization?

- We mean that many solutions exist that can:
 - fit all data points (0 training loss) in regression
 - correctly classify all training examples in classification
- This requires (many) more parameters than training examples
 - Need wide and deep enough networks
 - Can result in overfitting
- Questions:
 - Which of all solutions give best generalization?
 - (How) can network design affect generalization?



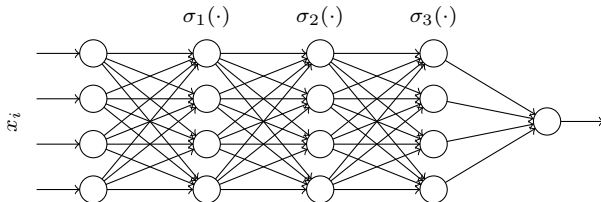
Overparameterization – An example

- Assume fully connected network with
 - input data $x_i \in \mathbb{R}^p$
 - n layers and $N \approx p^2$ samples
 - same width throughout (except last layer, which can be neglected)
- What is the relation between number of weights and samples?



Overparameterization – An example

- Assume fully connected network with
 - input data $x_i \in \mathbb{R}^p$
 - n layers and $N \approx p^2$ samples
 - same width throughout (except last layer, which can be neglected)
- What is the relation between number of weights and samples?



- We have:
 - Number of parameters approximately: $(W_j)_{lk}$: $p^2 n$ and $(b_j)_l$: pn
 - Then $\frac{\text{\#weights}}{\text{\#samples}} \approx \frac{p^2 n}{p^2} = n$ more weights than samples

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- **Generalization and regularization**
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Generalization

- Most important for model to generalize well to unseen data
- General approach in training
 - Train a model that is too expressive for the underlying data
 - Overparameterization in deep learning
 - Use regularization to
 - find model of appropriate (lower) complexity
 - favor models with desired properties

Regularization

What regularization techniques in DL are you familiar with?

Regularization techniques

- Reduce number of parameters
 - Sparse weight tensors (e.g., convolutional layers)
 - Subsampling (gives fewer parameters deeper in network)
- Explicit regularization term in cost function, e.g., Tikhonov
- Data augmentation – more samples, artificial often OK
- Early stopping – stop algorithm before convergence
- Dropouts
- ...

Implicit vs explicit regularization

- Regularization can be explicit or implicit
- Explicit – Introduce something with intent to regularize:
 - Add cost function to favor desirable properties
 - Design (adapt) network to have regularizing properties
- Implicit – Use something with regularization as byproduct:
 - Use algorithm that finds favorable solution among many
 - Will look at implicit regularization via SGD

Generalization – Our focus

Will here discuss generalization via:

- Norm of parameters – leads to implicit regularization via SGD
- Flatness of minima – leads to implicit regularization via SGD

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- **Generalization – Norm of weights**
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding gradients

Lipschitz continuity of ReLU networks

- Assume that all activation functions 1-Lipschitz continuous
- The neural network model $m(\cdot; \theta)$ is Lipschitz continuous in x ,

$$\|m(x_1; \theta) - m(x_2; \theta)\|_2 \leq L\|x_1 - x_2\|_2$$

for fixed θ , e.g., the θ obtained after training

- This means output differences are bounded by input differences
- A Lipschitz constant L is given by

$$L = \|W_n\|_2 \cdot \|W_{n-1}\|_2 \cdots \|W_1\|_2$$

since activation functions are 1-Lipschitz continuous

- For residual layers each $\|W_j\|_2$ replaced by $(1 + \|W_j\|_2)$

Desired Lipschitz constant

- Overparameterization gives many solutions that perfectly fit data
- Would you favor one with high or low Lipschitz constant L ?

Small norm likely to generalize better

- Smaller Lipschitz constant probably generalizes better if perfect fit
- “Similar inputs give similar outputs”, recall

$$\|m(x_1; \theta) - m(x_2; \theta)\|_2 \leq L\|x_1 - x_2\|_2$$

with a Lipschitz constant is given by

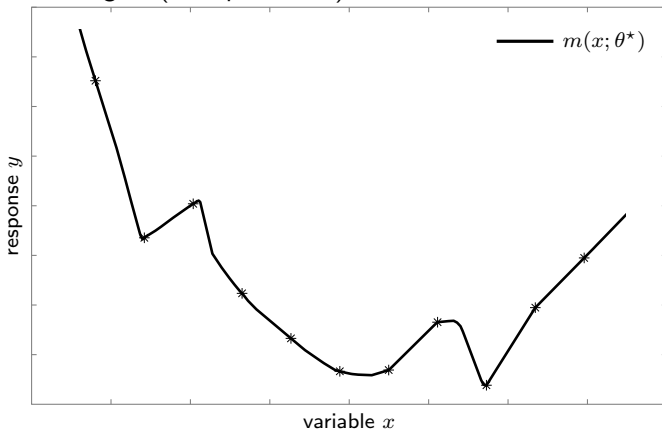
$$L = \|W_n\|_2 \cdot \|W_{n-1}\|_2 \cdots \|W_1\|_2$$

or with $\|W_j\|_2$ replaced by $(1 + \|W_j\|_2)$ for residual layers

- Smaller weight norms give better generalization if perfect fit

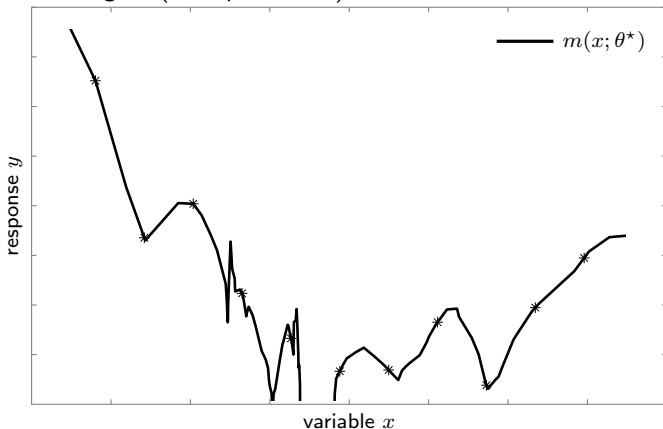
Generalization – Norm of weights

- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 72



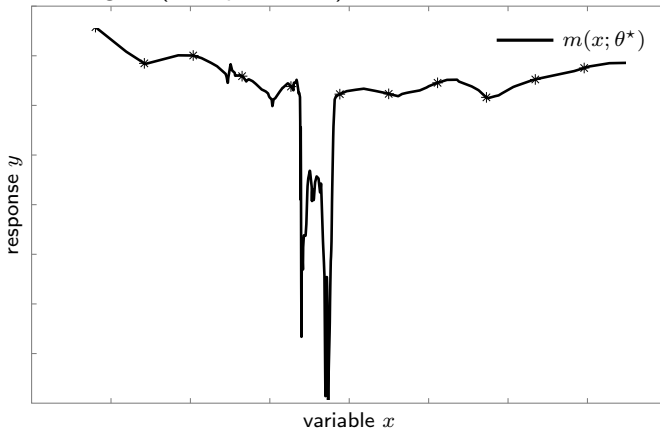
Generalization – Norm of weights

- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 540



Generalization – Norm of weights

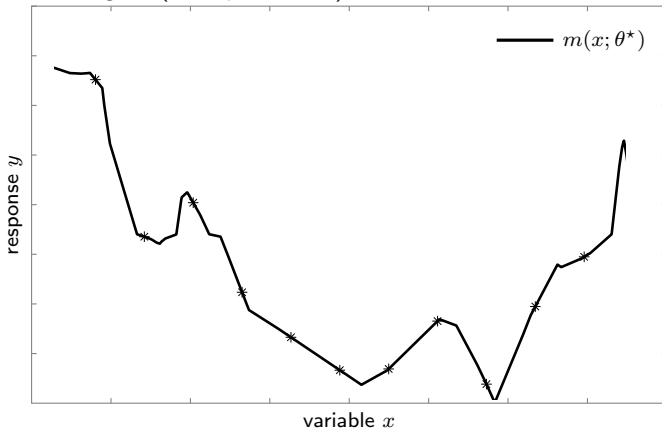
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 540



- Same as previous, new scaling

Generalization – Norm of weights

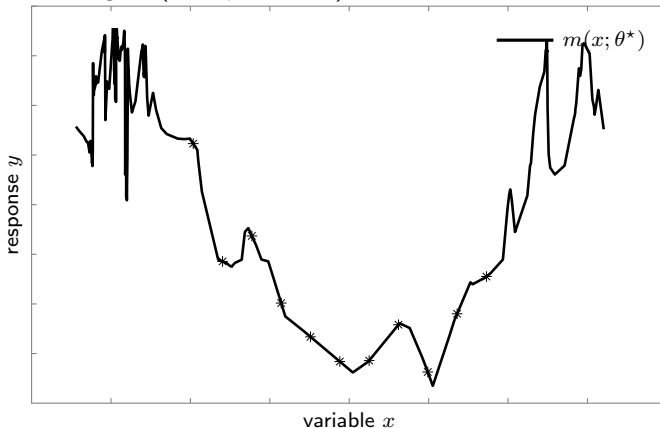
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 595



- Large norm, but seemingly fair generalization

Generalization – Norm of weights

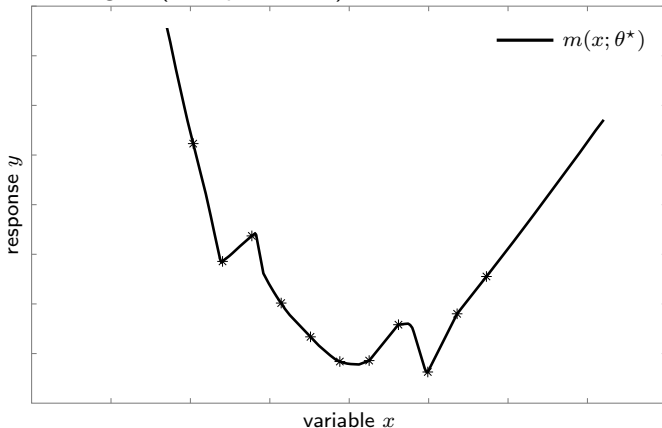
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 595



- Same as previous, new scaling

Generalization – Norm of weights

- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 72



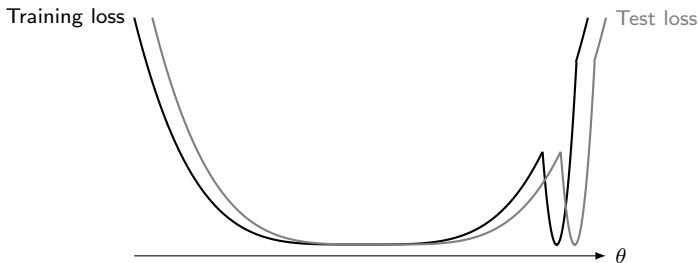
- Same as first, new scaling – overfits less than large norm solutions

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- **Generalization – Flatness of minima**
- Backpropagation
- Vanishing and exploding gradients

Flatness of minima

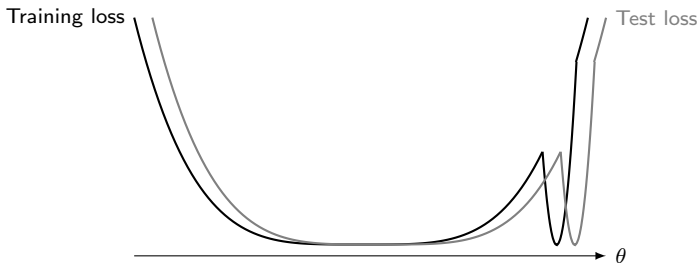
- Consider the following illustration of *average* loss:



- Depicts test loss as shifted training loss
- Motivation to that flat minima generalize better than sharp

Flatness of minima

- Consider the following illustration of *average* loss:



- Depicts test loss as shifted training loss
- Motivation to that flat minima generalize better than sharp
- Is there a limitation in considering the average loss only?

Generalization from loss landscape

- Training set $\{(x_i, y_i)\}_{i=1}^N$ and training problem:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^N L(m(x_i; \theta), y_i)$$

- Test set $\{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{\hat{N}}$, θ generalizes well if test loss small

$$\sum_{i=1}^{\hat{N}} L(m(\hat{x}_i; \theta), \hat{y}_i)$$

- By overparameterization, we can for each (\hat{x}_i, \hat{y}_i) find $\hat{\theta}_i$ so that

$$L(m(\hat{x}_i; \theta), \hat{y}_i) = L(m(x_{j_i}; \theta + \hat{\theta}_i), y_{j_i})$$

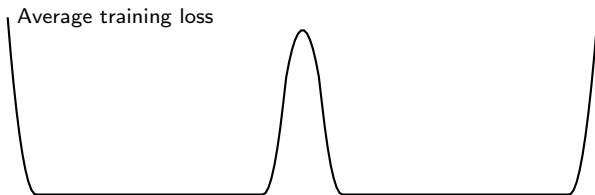
for all θ given a (similar) (x_{j_i}, y_{j_i}) pair in training set

- Evaluate test loss by training loss at shifted points $\theta + \hat{\theta}_i$ ¹⁾
- Test loss small if original individual loss small at all $\theta + \hat{\theta}_i$
- Previous figure used same $\hat{\theta}_i = \hat{\theta}$ for all i

¹⁾ Don't compute in practice, just thought experiment to connect generalization to training loss

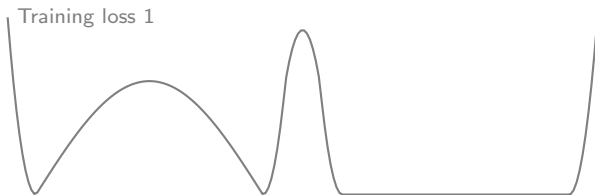
Example

- Can flat (local) minima be different?
- Does one of the following minima generalize better?



Example

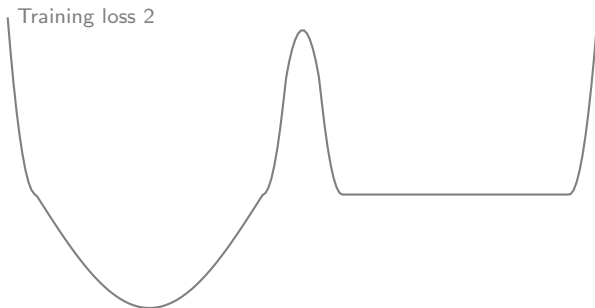
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

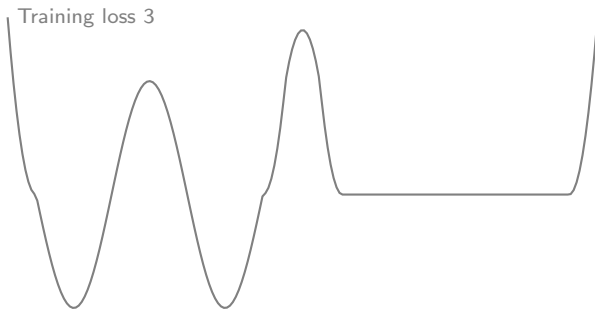
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

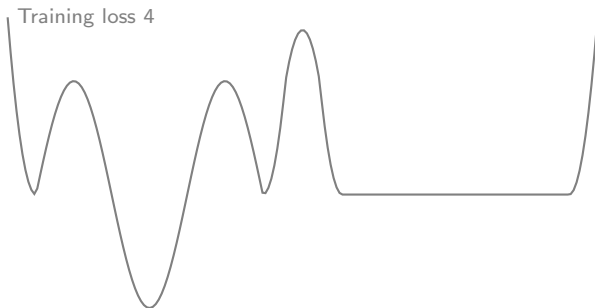
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

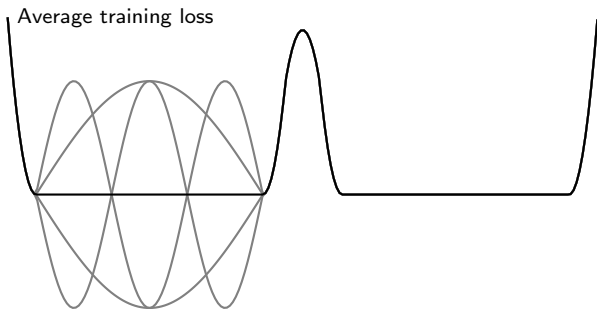
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

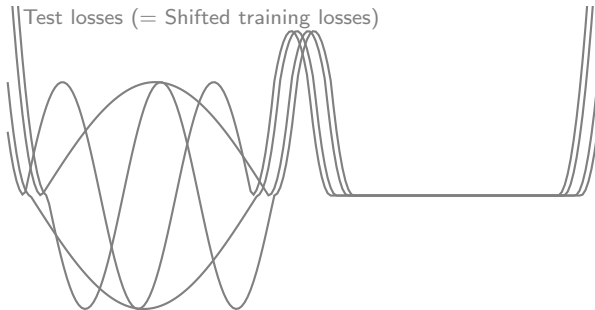
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

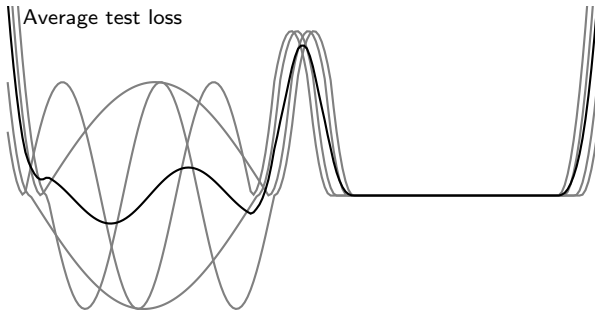
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses
- Let us evaluate test loss by shifting individual training losses

Example

- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses
- Let us evaluate test loss by shifting individual training losses
- Do not only want flat minima, want individual losses flat at minima

Individually flat minima

- Both flat minima have $\nabla f(\theta) = 0$, but
 - One minima has large individual gradients $\|\nabla f_i(\theta)\|$
 - Other minima has small individual gradients $\|\nabla f_i(\theta)\|$
 - The latter (individually flat minima) seems to generalize better
- Want individually flat minima (with small $\|\nabla f_i(\theta)\|$)
 - This implies average flat minima
 - The reverse implication may not hold
 - Overparameterized networks:
 - The reverse implication may often hold at global minima
 - Why? $f(\theta) = 0$ and $\nabla f(\theta) = 0$ implies $f_i(\theta) = 0$ and $\nabla f_i(\theta) = 0$

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- **Backpropagation**
- Vanishing and exploding gradients

Training algorithm

- Neural networks often trained using stochastic gradient descent
- DNN weights are updated via gradients in training
- Gradient of cost is sum of gradients of summands (samples)
- Gradient of each summand computed using backpropagation

Backpropagation

- Backpropagation is reverse mode automatic differentiation
- Based on chain-rule in differentiation
- Backpropagation must be performed per sample
- Our derivation assumes:
 - Fully connected layers (W full, if not, set elements in W to 0)
 - Activation functions $\sigma_j(v) = (\sigma_j(v_1), \dots, \sigma_j(v_p))$ element-wise (overloading of σ_j notation)
 - Weights W_j are matrices, samples x_i and responses y_i are vectors
 - No residual connections

Jacobians

- The Jacobian of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is given by

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

- The Jacobian of a function $f : \mathbb{R}^{p \times n} \rightarrow \mathbb{R}$ is given by

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1n}} \\ \vdots & \vdots & \vdots \\ \frac{\partial f}{\partial x_{p1}} & \cdots & \frac{\partial f}{\partial x_{pn}} \end{bmatrix} \in \mathbb{R}^{p \times n}$$

- The Jacobian of a function $f : \mathbb{R}^{p \times n} \rightarrow \mathbb{R}^m$ is at layer j given by

$$\left[\frac{\partial f}{\partial x} \right]_{:,j,:} = \begin{bmatrix} \frac{\partial f_1}{\partial x_{j1}} & \cdots & \frac{\partial f_1}{\partial x_{jn}} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_{j1}} & \cdots & \frac{\partial f_m}{\partial x_{jn}} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

the full Jacobian is a 3D tensor in $\mathbb{R}^{m \times p \times n}$

Jacobian vs gradient

- The Jacobian of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is given by

$$\frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial x_1} \quad \cdots \quad \frac{\partial f}{\partial x_n} \right]$$

- The gradient of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is given by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

i.e., transpose of Jacobian for $f : \mathbb{R}^n \rightarrow \mathbb{R}$

- Chain rule holds for Jacobians:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial x}$$

Jacobian vs gradient – Example

- Consider differentiable $f : \mathbb{R}^m \rightarrow \mathbb{R}$ and $M \in \mathbb{R}^{m \times n}$
- Compute Jacobian of $g = (f \circ M)$ using chain rule:
 - Rewrite as $g(x) = f(z)$ where $z = Mx$
 - Compute Jacobian by partial Jacobians $\frac{\partial f}{\partial z}$ and $\frac{\partial z}{\partial x}$:

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial z} \frac{\partial z}{\partial x} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial x} = \nabla f(z)^T M = \nabla f(Mx)^T M \in \mathbb{R}^{1 \times n}$$

- Know gradient of $(f \circ M)(x)$ satisfies

$$\nabla(f \circ M)(x) = M^T \nabla f(Mx) \in \mathbb{R}^n$$

which is transpose of Jacobian

Backpropagation – Introduce states

- Compute gradient/Jacobian of

$$L(m(x_i; \theta), y_i)$$

w.r.t. $\theta = \{(W_j, b_j)\}_{j=1}^n$, where

$$m(x_i; \theta) = W_n \sigma_{n-1}(W_{n-1} \sigma_{n-2}(\cdots (W_2 \sigma_1(W_1 x_i + b_1) + b_2) \cdots) + b_{n-1}) + b_n$$

- Rewrite as function with states z_j

$$L(z_{n+1}, y_i)$$

where $z_{j+1} = \sigma_j(W_j z_j + b_j)$ for $j \in \{1, \dots, n\}$

and $z_1 = x_i$

where $\sigma_n(u) \equiv u$

Graphical representation

- Per sample loss function

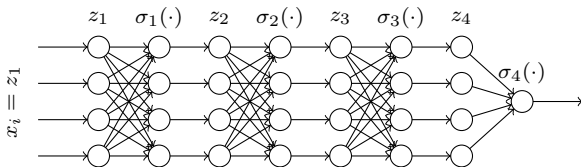
$$L(z_{n+1}, y_i)$$

where $z_{j+1} = \sigma_j(W_j z_j + b_j)$ for $j \in \{1, \dots, n\}$

and $z_1 = x_i$

where $\sigma_n(u) \equiv u$

- Graphical representation



Backpropagation – Chain rule

- Jacobian of L w.r.t. W_j and b_j can be computed as

$$\begin{aligned}\frac{\partial L}{\partial W_j} &= \frac{\partial L}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial z_n} \dots \frac{\partial z_{j+2}}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial W_j} \\ \frac{\partial L}{\partial b_j} &= \frac{\partial L}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial z_n} \dots \frac{\partial z_{j+2}}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial b_j}\end{aligned}$$

where we mean derivative w.r.t. first argument in L

- Backpropagation evaluates partial Jacobians as follows

$$\begin{aligned}\frac{\partial L}{\partial W_j} &= \left(\left(\frac{\partial L}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial z_n} \right) \dots \frac{\partial z_{j+2}}{\partial z_{j+1}} \right) \frac{\partial z_{j+1}}{\partial W_j} \\ \frac{\partial L}{\partial b_j} &= \left(\left(\frac{\partial L}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial z_n} \right) \dots \frac{\partial z_{j+2}}{\partial z_{j+1}} \right) \frac{\partial z_{j+1}}{\partial b_j}\end{aligned}$$

Backpropagation – Forward and backward pass

- Jacobian of $L(z_{n+1}, y_i)$ w.r.t. z_{n+1} (transpose of gradient)
- Computing Jacobian of $L(z_{n+1}, y_i)$ requires z_{n+1}
 \Rightarrow forward pass: $z_1 = x_i, z_{j+1} = \sigma_j(W_j z_j + b_j)$
- Backward pass, store δ_j :

$$\frac{\partial L}{\partial z_{j+1}} = \underbrace{\left(\underbrace{\left(\frac{\partial L}{\partial z_{n+1}} \frac{\partial z_{n+1}}{\partial z_n} \right) \dots \frac{\partial z_{j+2}}{\partial z_{j+1}}}_{\delta_{n+1}^T} \right)}_{\delta_n^T} \underbrace{\hspace{10em}}_{\delta_{j+1}^T}$$

- Compute

$$\begin{aligned} \frac{\partial L}{\partial W_j} &= \frac{\partial L}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial W_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial W_j} \\ \frac{\partial L}{\partial b_j} &= \frac{\partial L}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial b_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial b_j} \end{aligned}$$

Dimensions

- Let $z_j \in \mathbb{R}^{n_j}$, consequently $W_j \in \mathbb{R}^{n_{j+1} \times n_j}$, $b_j \in \mathbb{R}^{n_{j+1}}$
- Dimensions

$$\frac{\partial L}{\partial W_j} = \underbrace{\left(\underbrace{\left(\underbrace{\frac{\partial L}{\partial z_{n+1}}}_{1 \times n_{n+1}} \underbrace{\frac{\partial z_{n+1}}{\partial z_n}}_{n_{n+1} \times n_n} \right) \dots \frac{\partial z_{j+2}}{\partial z_{j+1}}}_{n_{j+2} \times n_{j+1}} \right)}_{1 \times n_n} \underbrace{\frac{\partial z_{j+1}}{\partial W_j}}_{n_{j+1} \times n_{j+1} \times n_j}$$

$$\frac{\partial L}{\partial b_j} = \underbrace{\left(\underbrace{\left(\underbrace{\frac{\partial L}{\partial z_{n+1}}}_{1 \times n_{n+1}} \underbrace{\frac{\partial z_{n+1}}{\partial z_n}}_{n_{n+1} \times n_n} \right) \dots \frac{\partial z_{j+2}}{\partial z_{j+1}}}_{1 \times n_{j+1}} \right)}_{1 \times n_{j+1}} \underbrace{\frac{\partial z_{j+1}}{\partial b_j}}_{n_{j+1} \times n_{j+1}}$$

- Vector matrix multiplies except for in last step
- Multiplication with tensor $\frac{\partial z_{j+1}}{\partial W_j}$ can be simplified
- Backpropagation variables $\delta_j \in \mathbb{R}^{n_j}$ are vectors (not matrices)

Partial Jacobian $\frac{\partial z_{j+1}}{\partial z_j}$

- Recall relation $z_{j+1} = \sigma_j(W_j z_j + b_j)$ and let $v_j = W_j z_j + b_j$
- Chain rule gives

$$\begin{aligned}\frac{\partial z_{j+1}}{\partial z_j} &= \frac{\partial z_{j+1}}{\partial v_j} \frac{\partial v_j}{\partial z_j} = \mathbf{diag}(\sigma'_j(v_j)) \frac{\partial v_j}{\partial z_j} \\ &= \mathbf{diag}(\sigma'_j(W_j z_j + b_j)) W_j\end{aligned}$$

where, with abuse of notation (notation overloading)

$$\sigma'_j(u) = \begin{bmatrix} \sigma'_j(u_1) \\ \vdots \\ \sigma'_j(u_{n_{j+1}}) \end{bmatrix}$$

- Reason: $\sigma_j(u) = [\sigma_j(u_1), \dots, \sigma_j(u_{n_{j+1}})]^T$ with $\sigma_j : \mathbb{R}^{n_{j+1}} \rightarrow \mathbb{R}^{n_{j+1}}$, gives

$$\frac{d\sigma_j}{du} = \begin{bmatrix} \sigma'_j(u_1) & & \\ & \ddots & \\ & & \sigma'_j(u_{n_{j+1}}) \end{bmatrix} = \mathbf{diag}(\sigma'_j(u))$$

Partial Jacobian $\delta_j^T = \frac{\partial L}{\partial z_j}$

- For any vector $\delta_{j+1} \in \mathbb{R}^{n_{j+1} \times 1}$, we have

$$\begin{aligned} \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial z_j} &= \delta_{j+1}^T \mathbf{diag}(\sigma'_j(W_j z_j + b_j)) W_j \\ &= (W_j^T (\delta_{j+1}^T \mathbf{diag}(\sigma'_j(W_j z_j + b_j))))^T \\ &= (W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)))^T \end{aligned}$$

where \odot is element-wise (Hadamard) product

- We have defined $\delta_{n+1}^T = \frac{\partial L}{\partial z_{n+1}}$, then

$$\delta_n^T = \frac{\partial L}{\partial z_n} = \delta_{n+1}^T \frac{\partial z_{n+1}}{\partial z_n} = \underbrace{(W_n^T (\delta_{n+1} \odot \sigma'_n(W_n z_n + b_n)))^T}_{\delta_n}$$

- Consequently, using induction:

$$\delta_j^T = \frac{\partial L}{\partial z_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial z_j} = \underbrace{(W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)))^T}_{\delta_j}$$

Information needed to compute $\frac{\partial L}{\partial z_j}$

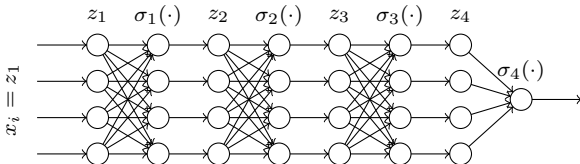
- To compute first Jacobian $\frac{\partial L}{\partial z_n}$, we need $z_n \Rightarrow$ forward pass
- Computing

$$\frac{\partial L}{\partial z_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial z_j} = (W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)))^T = \delta_j^T$$

is done using a backward pass

$$\delta_j = W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))$$

- All z_j (or $v_j = W_j z_j + b_j$) need to be stored for backward pass



Partial Jacobian $\frac{\partial L}{\partial W_j}$

- Computed by

$$\frac{\partial L}{\partial W_j} = \frac{\partial L}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial W_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial W_j}$$

where $z_{j+1} = \sigma_j(v_j)$ and $v_j = W_j z_j + b_j$

- Recall $\frac{\partial z_{j+1}}{\partial W_l}$ is 3D tensor, compute Jacobian w.r.t. row l (W_j) _{l}

$$\delta_{j+1}^T \frac{\partial z_{j+1}}{\partial (W_j)_l} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial v_j} \frac{\partial v_j}{\partial (W_j)_l} = \delta_{j+1}^T \mathbf{diag}(\sigma'_j(v_j)) \begin{bmatrix} 0 \\ \vdots \\ z_j^T \\ \vdots \\ 0 \end{bmatrix}$$

$$= (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))^T \begin{bmatrix} 0 \\ \vdots \\ z_j^T \\ \vdots \\ 0 \end{bmatrix} = (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))_l z_j^T$$

Partial Jacobian $\frac{\partial L}{\partial W_j}$ cont'd

- Stack Jacobians w.r.t. rows to get full Jacobian:

$$\begin{aligned}\frac{\partial L}{\partial W_j} &= \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial W_j} = \begin{bmatrix} \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial (W_j)_1} \\ \vdots \\ \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial (W_j)_{n_{j+1}}} \end{bmatrix} = \begin{bmatrix} (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))_1 z_j^T \\ \vdots \\ (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))_{n_{j+1}} z_j^T \end{bmatrix} \\ &= (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)) z_j^T\end{aligned}$$

for all $j \in \{1, \dots, n-1\}$

- Dimension of result is $n_{j+1} \times n_j$, which matches W_j
- This is used to update W_j weights in algorithm

Partial Jacobian $\frac{\partial L}{\partial b_j}$

- Recall $z_{j+1} = \sigma_j(v_j)$ where $v_j = W_j z_j + b_j$
- Computed by

$$\begin{aligned}\frac{\partial L}{\partial b_j} &= \frac{\partial L}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial v_j} \frac{\partial v_j}{\partial b_j} = \delta_{j+1}^T \frac{\partial z_{j+1}}{\partial v_j} \frac{\partial v_j}{\partial b_j} = \delta_{j+1}^T \mathbf{diag}(\sigma'_j(v_j)) \\ &= (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))^T\end{aligned}$$

Backpropagation summarized

1. Forward pass: Compute and store z_j (or $v_j = W_j z_j + b_j$):

$$z_{j+1} = \sigma_j(W_j z_j + b_j)$$

where $z_1 = x_i$ and $\sigma_n = \text{Id}$

2. Backward pass:

$$\delta_j = W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))$$

with $\delta_{n+1} = \frac{\partial L}{\partial z_{n+1}}$

3. Weight update Jacobians (used in SGD)

$$\begin{aligned}\frac{\partial L}{\partial W_j} &= (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)) z_j^T \\ \frac{\partial L}{\partial b_j} &= (\delta_{j+1} \odot \sigma'_j(W_j x_j + b_j))^T\end{aligned}$$

Backpropagation – Residual networks

1. Forward pass: Compute and store z_j (or $v_j = W_j z_j + b_j$):

$$z_{j+1} = \sigma_j(W_j z_j + b_j) + z_j$$

where $z_1 = x_i$ and $\sigma_n = \text{Id}$

2. Backward pass:

$$\delta_j = W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)) + \delta_{j+1}$$

with $\delta_{n+1} = \frac{\partial L}{\partial z_{n+1}}$

3. Weight update Jacobians (used in SGD)

$$\frac{\partial L}{\partial W_j} = (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)) z_j^T$$

$$\frac{\partial L}{\partial b_j} = (\delta_{j+1} \odot \sigma'_j(W_j x_j + b_j))^T$$

Outline

- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- Overparameterized networks
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- **Vanishing and exploding gradients**

Vanishing and exploding gradient problem

- For some activation functions, gradients can vanish
- For other activation functions, gradients can explode

Vanishing gradient example: Sigmoid

- Assume $\|W_j\| \leq 1$ for all j and $\|\delta_{n+1}\| \leq C$
- Maximal derivative of sigmoid (σ) is 0.25
- Then

$$\begin{aligned}\left\| \frac{\partial L}{\partial z_j} \right\| &= \|\delta_j\| = \|W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))\| \leq 0.25 \|\delta_{j+1}\| \\ &\leq 0.25^{n-j+1} \|\delta_{n+1}\| \leq 0.25^{n-j+1} C\end{aligned}$$

- Hence, as n grows, gradients can become very small for small i
- In general, vanishing gradient if $\sigma' < 1$ everywhere
- Similar reasoning: exploding gradient if $\sigma' > 1$ everywhere
- Hence, need $\sigma' = 1$ in important regions

Vanishing gradients – Residual networks

- Residual networks with forward pass

$$z_{j+1} = \sigma_j(W_j z_j + b_j) + z_j$$

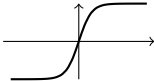
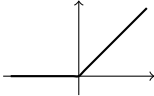
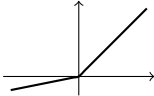
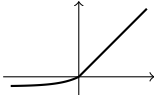
and backward pass

$$\delta_j = W_j^T (\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)) + \delta_{j+1}$$

- Gradients do not vanish in passes despite small σ gain

Examples of activation functions

Activation functions that (partly) avoid vanishing gradients

Name	$\sigma(u)$	Graph
Tanh	$\frac{e^u - e^{-u}}{e^{-u} + e^u}$	
ReLU	$\max(u, 0)$	
LeakyReLU	$\max(u, \alpha u)$	
ELU	$\begin{cases} u & \text{if } u \geq 0 \\ \alpha(e^u - 1) & \text{else} \end{cases}$	

Exploding gradient – Example

- Assume L -Lipschitz activation (ReLU, Tanh etc have $L = 1$)
 - Forward pass estimation:

$$\begin{aligned}\|z_{j+1}\|_2 &= \|\sigma_j(W_j z_j + b_j)\|_2 \leq L\|W_j z_j + b_j\|_2 \leq L(\|W_j z_j\|_2 + \|b_j\|_2) \\ &\leq L\|W_j\|_2\|z_j\|_2 + L\|b_j\|_2\end{aligned}$$

- Backward pass estimation:

$$\begin{aligned}\|\delta_j\|_2 &= \|W_j^T(\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))\|_2 \\ &\leq \|W_j^T\|_2\|\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j)\|_2 \\ &\leq L\|W_j\|_2\|\delta_{j+1}\|_2\end{aligned}$$

- If $L \leq 1$, $\|W_j\|_2 \leq 1$ and $\|b_j\|_2$ small, gradients do not explode
 - ReLU “average” $L = 0.5$ reduces “average estimate”
 - Tanh reduces “average estimates” more since
 - σ_j -outputs are constrained to $(-1, 1)$
 - “average Lipschitz constant” is smaller

Exploding gradient – Residual network

- Assume L -Lipschitz activation (ReLU, Tanh have $L = 1$)

- Forward pass estimation:

$$\|z_{j+1}\|_2 = \|\sigma_j(W_j z_j + b_j)\|_2 + \|z_j\|_2 \leq (1 + L\|W_j\|_2)\|z_j\|_2 + L\|b_j\|_2$$

- Backward pass estimation:

$$\begin{aligned}\|\delta_j\|_2 &= \|W_j^T(\delta_{j+1} \odot \sigma'_j(W_j z_j + b_j))\|_2 + \delta_{j+1} \\ &\leq (1 + L\|W_j\|_2)\|\delta_{j+1}\|_2\end{aligned}$$

- Larger estimates than for non-residual networks
- Activations with $L \leq 1$ to avoid exploding and vanishing gradients:
 - $\alpha \times \text{ReLU}$ with $\alpha \in (0, 1)$
 - $\alpha \times \text{Tanh}$ with $\alpha \in (0, 1)$