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

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

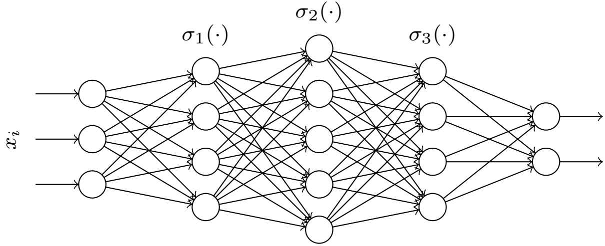
  where L is same as in convex regression and classification models
- Difference to previous convex methods: *Nonlinear model* $m(x;\theta)$
  - Deep learning regression generalizes least squares
  - DL classification generalizes multiclass logistic regression
  - Nonlinear model makes training problem nonconvex

Deep learning – Model

- Nonlinear model of the following form is often used:

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

  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

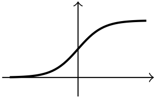
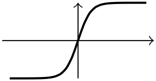
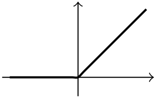
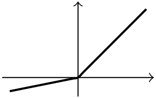
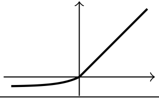
No final layer activation in classification

- 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
- We will not have activation in final layer

Activation functions

- Activation function σ_j takes as input the output of $W_j(\cdot) + b_j$
- Often a function $\bar{\sigma}_j : \mathbb{R} \to \mathbb{R}$ is applied to each element
    - Example: $\sigma_j : \mathbb{R}^3 \to \mathbb{R}^3$ 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

Loss functions

- The most common loss functions are
  - Regression: least squares loss
  - Binary classification: logistic loss
  - Multiclass classification: multiclass logistic loss

  which gives generalizations of LS and (multiclass) logistic regression
- Can also use
  - Regression: Huber loss, 1-norm loss
  - Binary classification: hinge loss (as in SVM)
  - Multiclass classification: Multiclass SVM loss functions

Prediction

- Prediction as for convex methods
- Assume model $m(x;\theta)$ trained and "optimal" $\theta^\star$ found
- Regression:
  - Predict response for new data x using $\hat{y} = m(x;\theta^\star)$
- Binary classification
  - Predict class beloning for new data x using $\mathrm{sign}(m(x;\theta^\star))$
- Multiclass classification (with no final layer activation):
  - We have one model $m_j(x;\theta^\star)$ output for each class
  - Predict class belonging for new data x according to

$$\underset{j\in\{1,\ldots,K\}}{\mathrm{argmax}}\ m_j(x;\theta^\star)$$

  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:

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

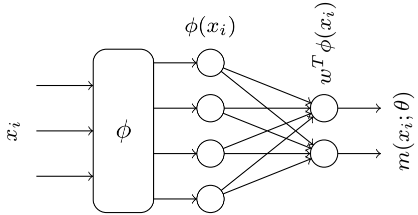
  - Model becomes $m(x;\theta) = W_n\phi(x;\theta) + b_n$
  - Inserted into training problem:

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

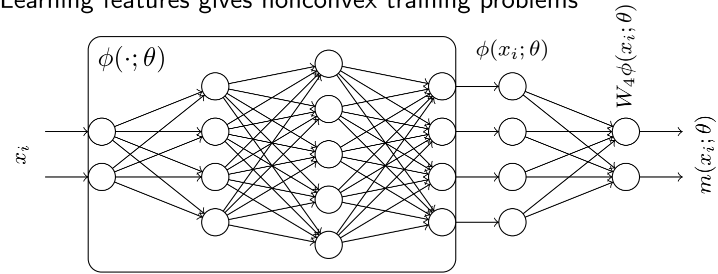
    same as before, but with learned (parameter-dependent) features
- Learning features at training makes training nonconvex

Learning features – Graphical representation

- Fixed features gives convex training problems



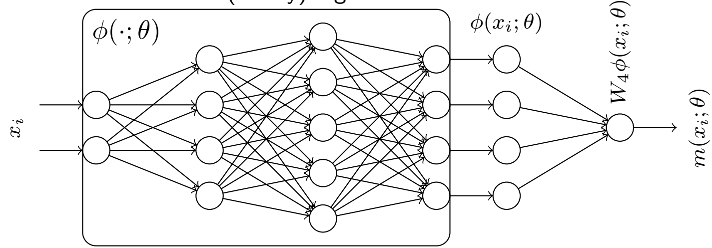
- Learning features gives nonconvex training problems



- Output of last activation function is feature vector

Optimizing only final layer

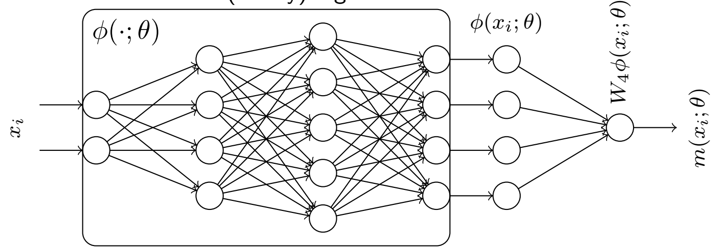
- Assume:
  - that parameters $\bar{\theta}_f$ in the layers in the square are fixed
  - that we optimize only the final layer parameters
  - that the loss is a (binary) logistic loss



- 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

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

  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

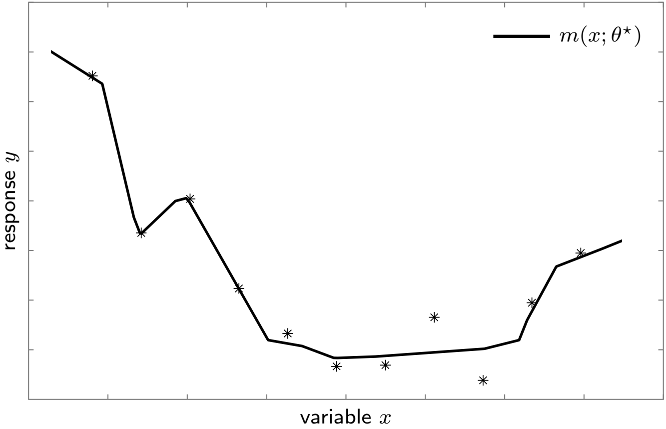
- Recall model

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

  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

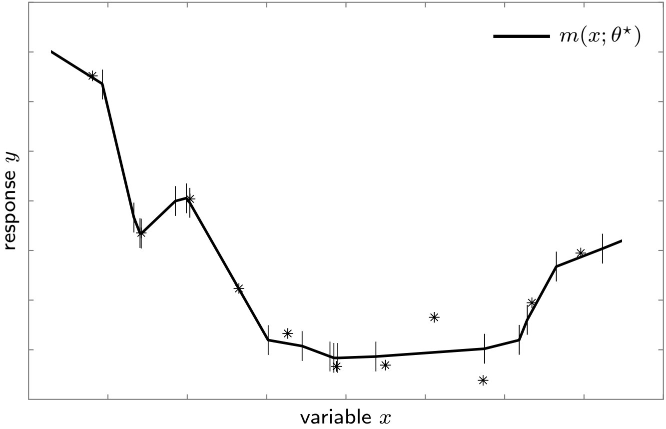
1D Regression – Model properties

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



1D Regression – Model properties

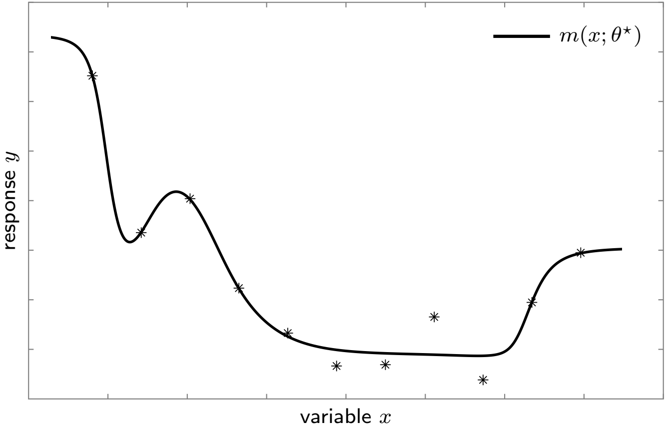
- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyReLU



- 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 $\sigma_j = \mathrm{Id}$ in

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

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 $\sigma_j = \mathrm{Id}$ in

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

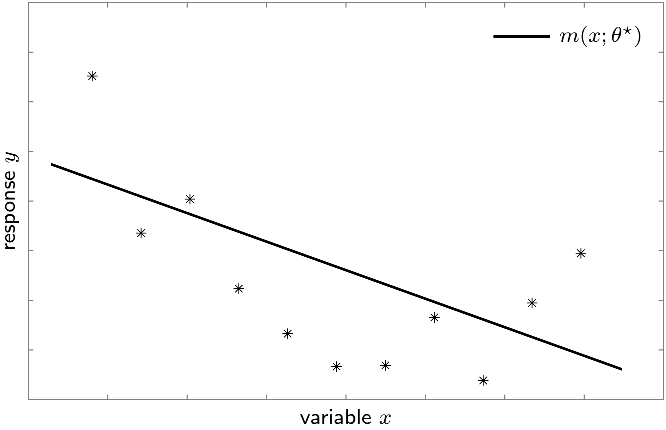
  where θ contains all W_j and b_j
- We then get

$$\begin{aligned} m(x;\theta) &:= W_n(W_{n-1}(\cdots(W_2(W_1x+b_1)+b_2)\cdots)+b_{n-1})+b_n \\ &= \underbrace{W_nW_{n-1}\cdots W_2W_1}_{W}x + \underbrace{b_n + \sum_{l=2}^{n} W_n\cdots W_l b_{l-1}}_{b} \\ &= Wx+b \end{aligned}$$

  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

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

where θ includes all W_j and b_j and training problem

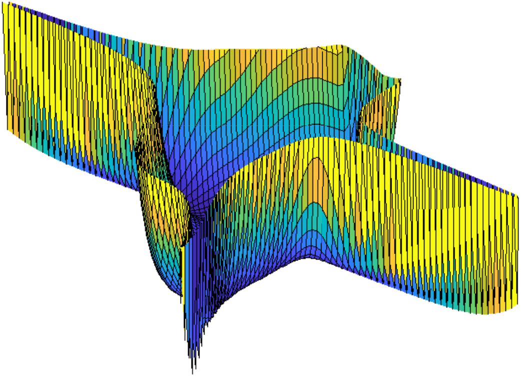
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{N} L(m(x_i;\theta), y_i)$$

- 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 $\sigma_j : \mathbb{R} \to (-1, 1)$

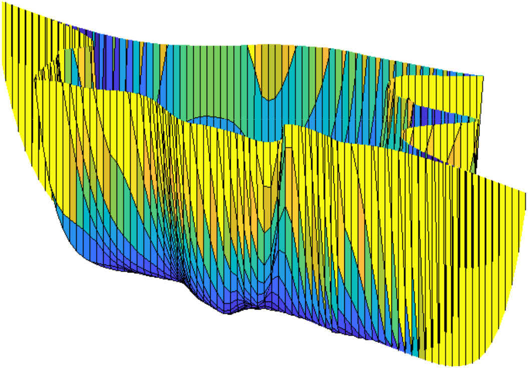
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^{N} L(m(x_i; \theta^\star + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
  - $\theta^\star$ 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:



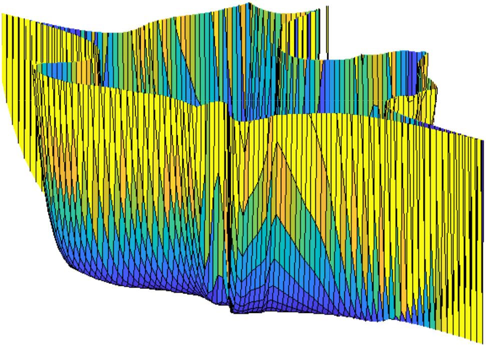
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^{N} L(m(x_i;\theta^\star + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
  - $\theta^\star$ 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:



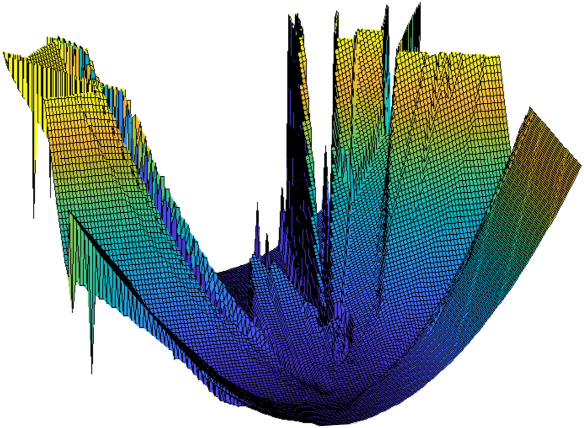
Loss landscape – Leaky ReLU

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^{N} L(m(x_i; \theta^\star + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
  - $\theta^\star$ 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:



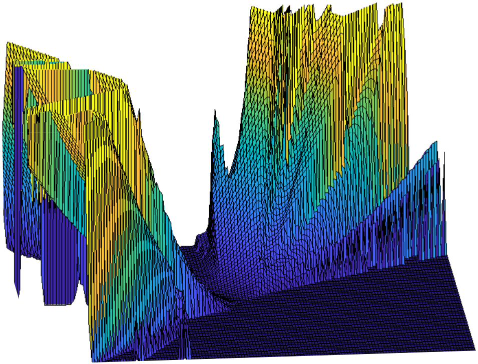
Loss landscape – Tanh

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^{N} L(m(x_i; \theta^\star + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
  - $\theta^\star$ 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:



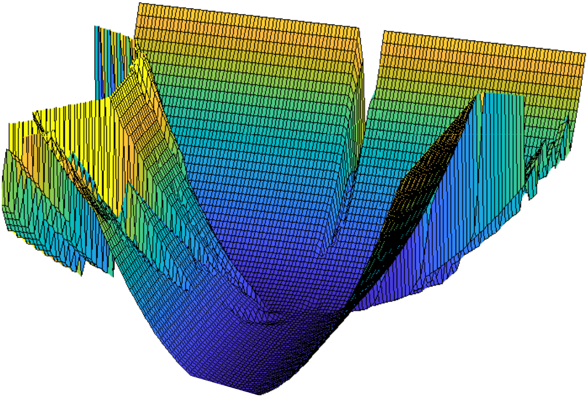
Loss landscape – Tanh

- Fully connected, layers widths: 5,5,5,1,1 (78 params), LeakyRelu
- Regression problem, least squares loss
- Plot: $\sum_{i=1}^{N} L(m(x_i; \theta^{\star} + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
  - $\theta^{\star}$ 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: $\sum_{i=1}^{N} L(m(x_i; \theta^\star + t_1\theta_1 + t_2\theta_2), y_i)$ vs scalars t_1, t_2, where
    - $\theta^\star$ 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

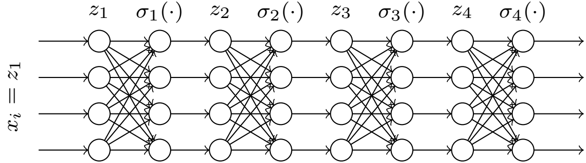
- Add skip connections between layers
- Instead of network architecture with $z_1 = x_i$ (see figure):

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

  use residual architecture

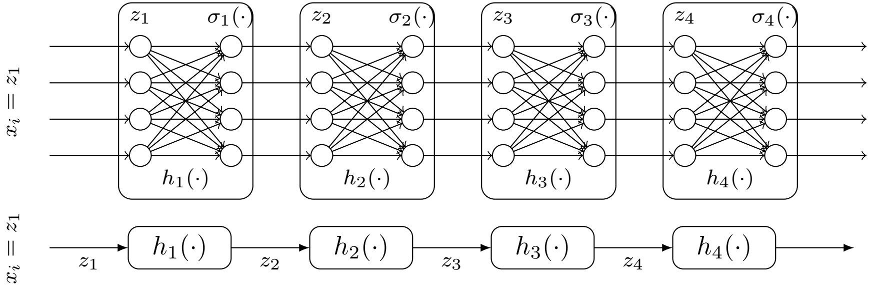
$$z_{j+1} = z_j + \sigma_j(W_j z_j + b_j) \text{ for } j \in \{1, \dots, n-1\}$$

- 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)



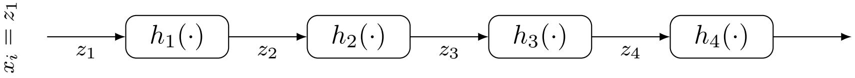
Graphical representation

For graphical representation, first collapse nodes into single node

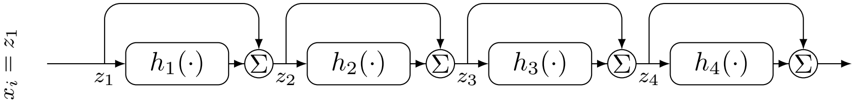


Graphical representation

- Collapsed network representation



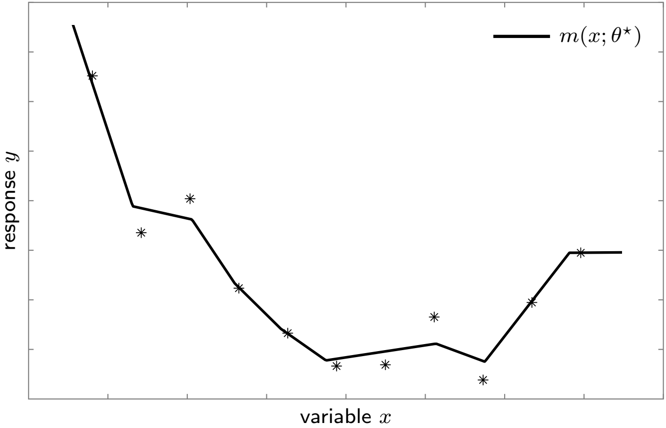
- Residual network



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

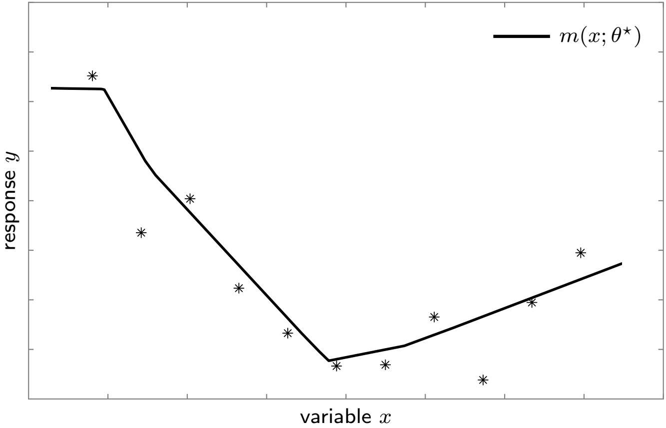
Residual network – Example

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



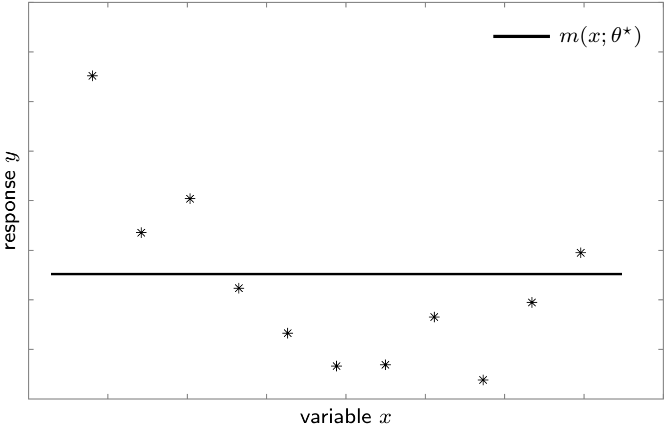
Residual network – Example

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



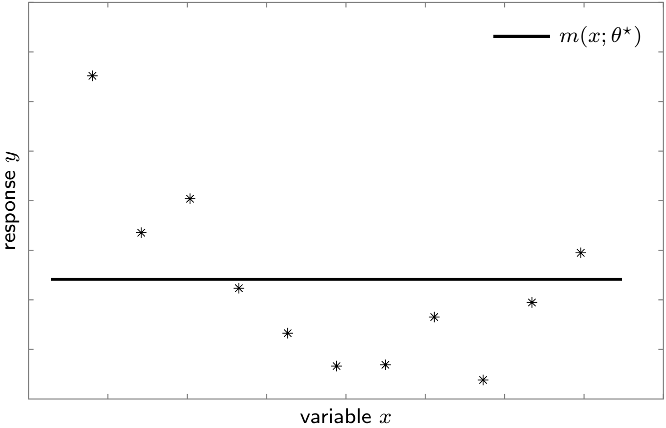
Residual network – Example

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



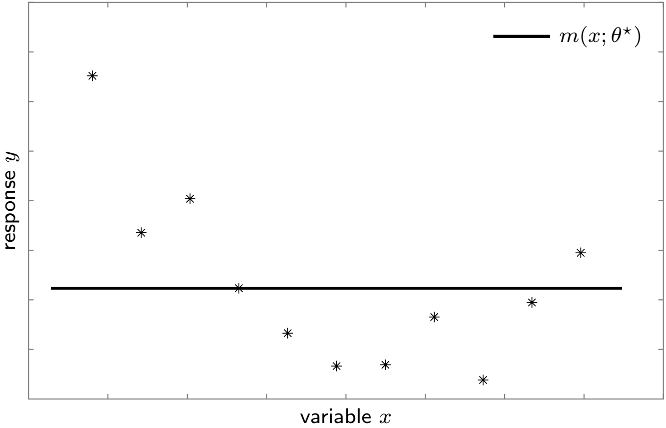
Residual network – Example

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



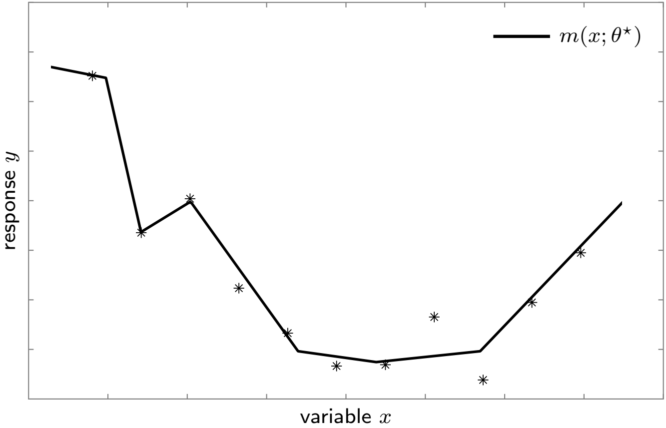
Residual network – Example

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



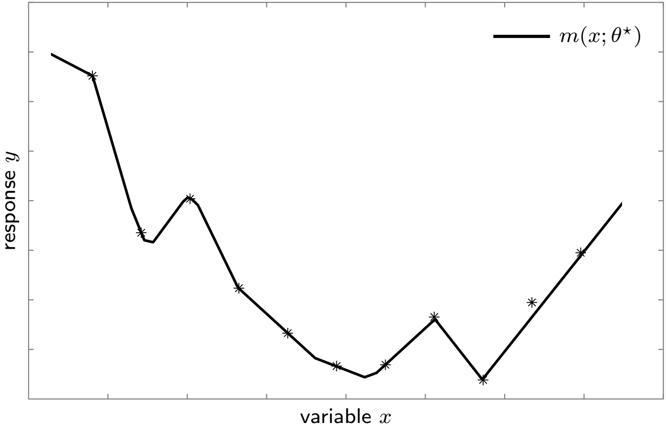
Residual network – Example

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



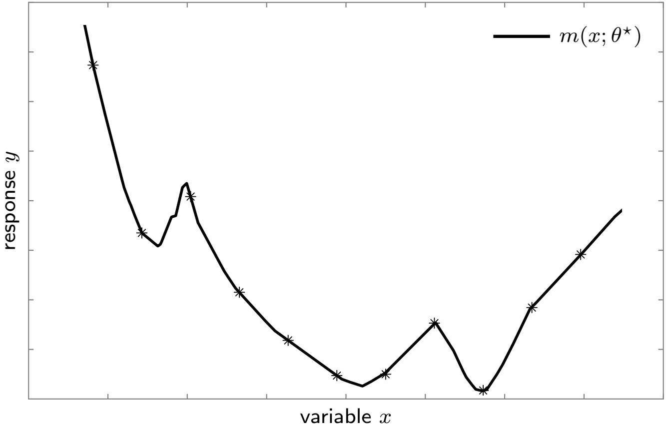
Residual network – Example

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



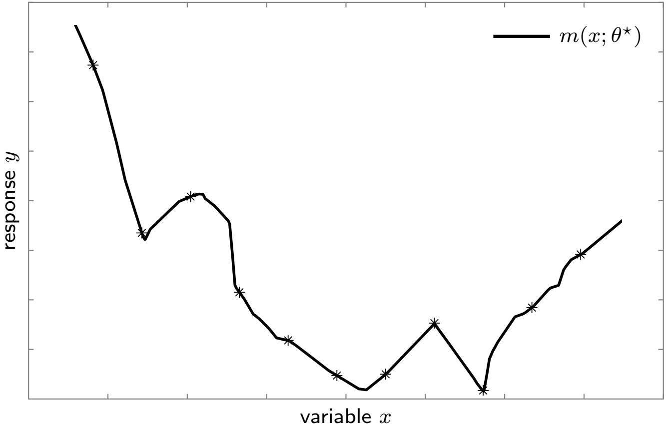
Residual network – Example

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



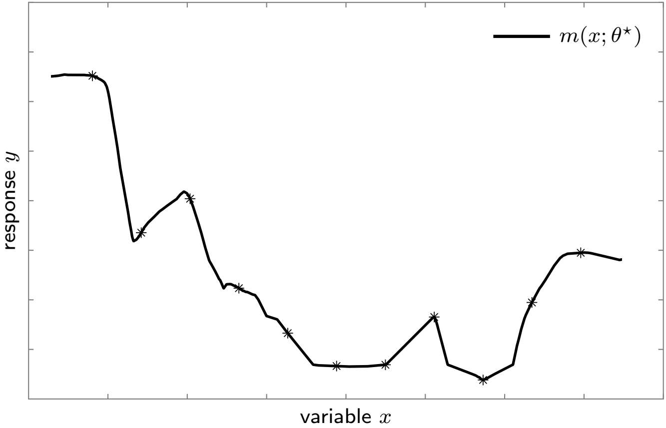
Residual network – Example

- 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

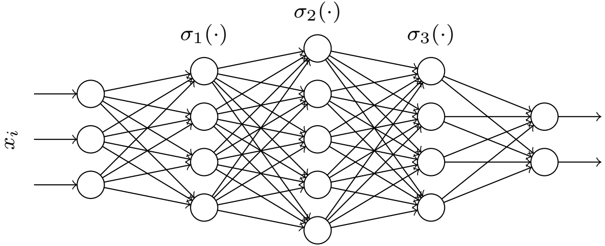
- Deep learning
- Learning features
- Model properties and activation functions
- Loss landscape
- Residual networks
- **Overparameterized networks**
- Generalization and regularization
- Generalization – Norm of weights
- Generalization – Flatness of minima
- Backpropagation
- Vanishing and exploding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

Why overparameterization?

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

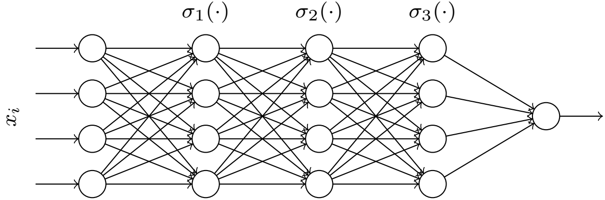
What is overparameterization?

- 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?



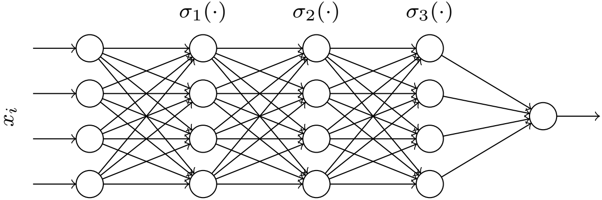
Overparameterization – An example

- 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,

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

  for fixed θ, e.g., the θ obtained after training
- This means output differerences 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

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

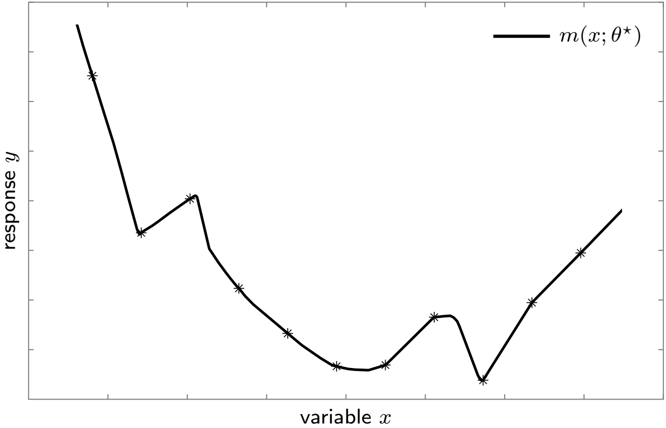
  with a Lipschitz constant is given by

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

  or with $\|W_j\|_2$ replaced by $(1 + \|W_j\|_2)$ for residual layers
- Smaller weight norms give better generalization if perfect fit

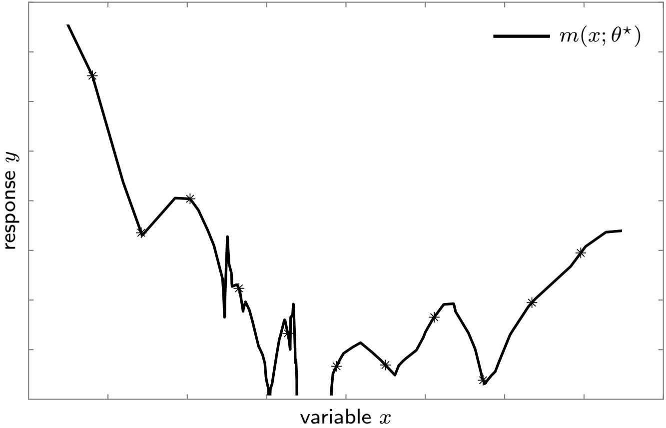
Generalization – Norm of weights

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



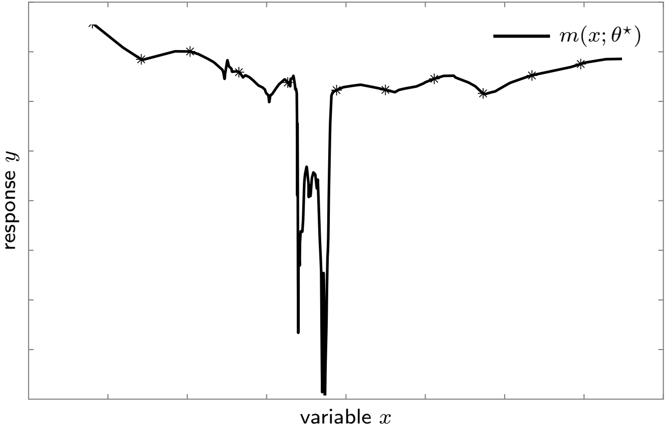
Generalization – Norm of weights

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



Generalization – Norm of weights

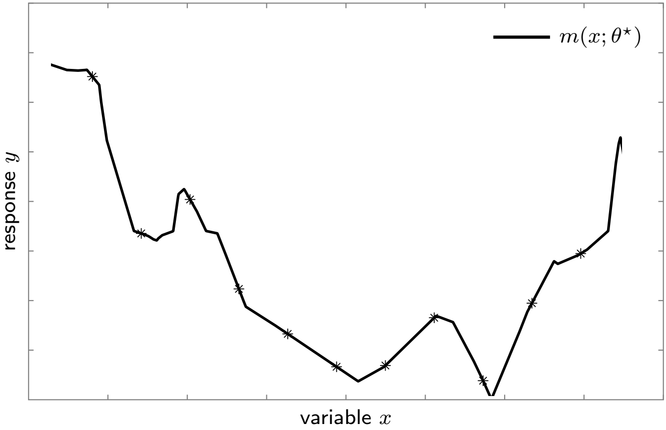
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 540



- Same as previous, new scaling

Generalization – Norm of weights

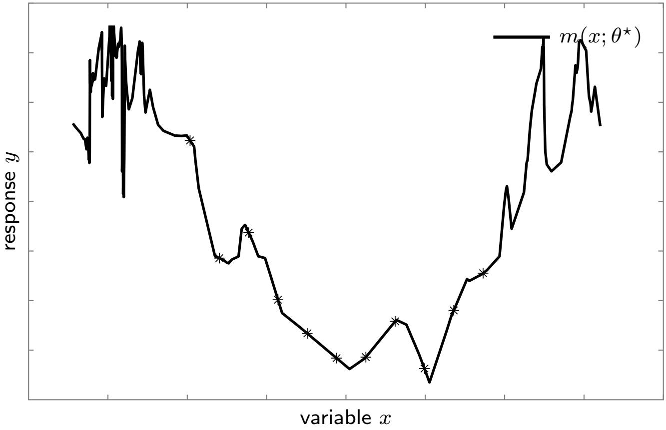
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 595



- Large norm, but seemingly fair generalization

Generalization – Norm of weights

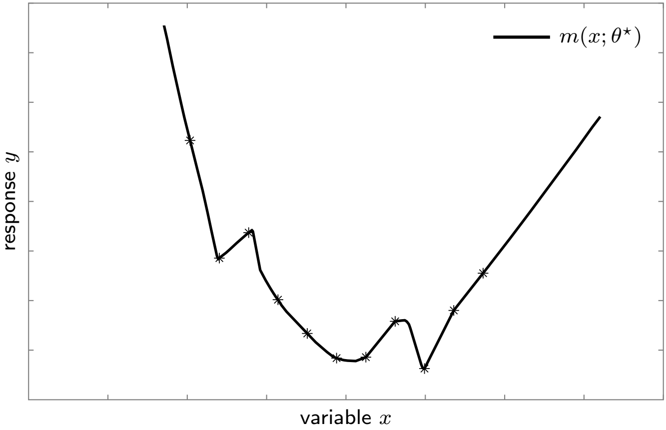
- Fully connected – residual layers, LeakyReLU
- Layers widths: 30x5,1,1 (888 params)
- Norm of weights (with perfect fit): 595



- 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



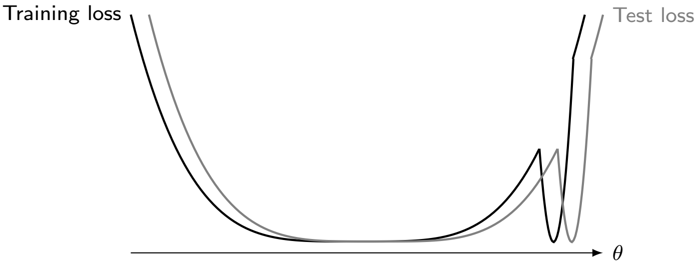
- Same as first, new scaling – overfits less than large norm solutions

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

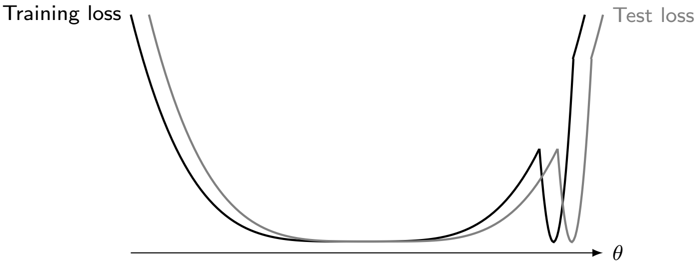
- Consider the following illustration of *average* loss:



- 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

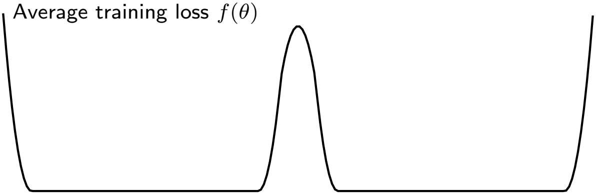
$$L(m(\hat{x}_i;\theta), \hat{y}_i) = L(m(x_{j_i};\theta + \hat{\theta}_i), y_{j_i})$$

  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

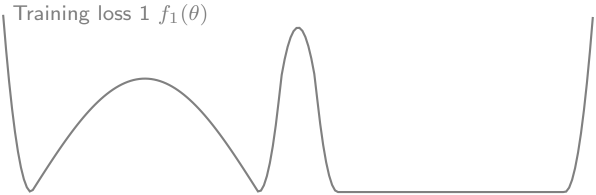
Example

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



Example

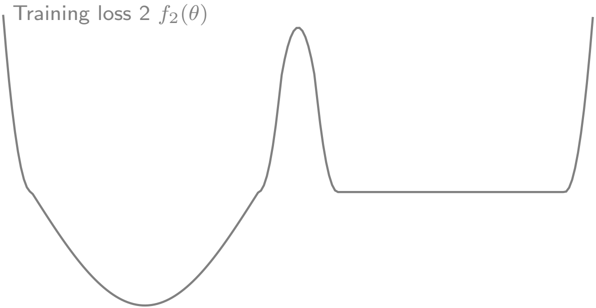
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

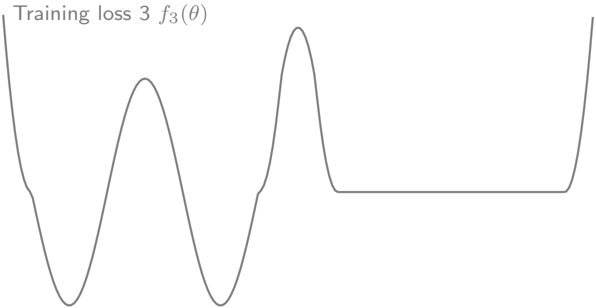
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

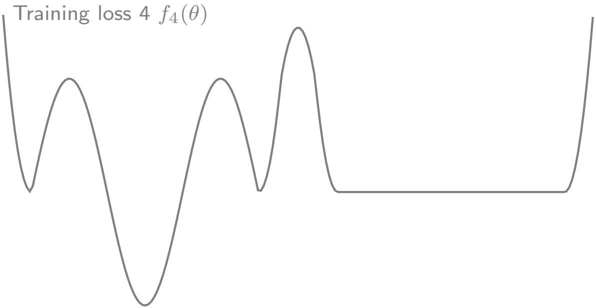
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

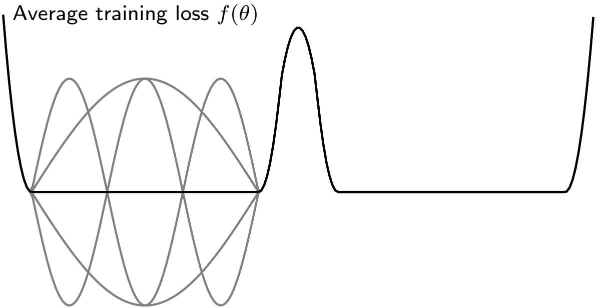
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

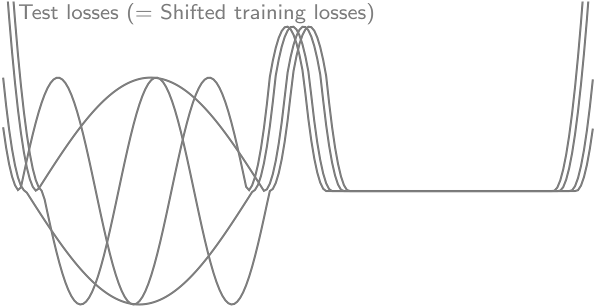
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- It depends on individual losses

Example

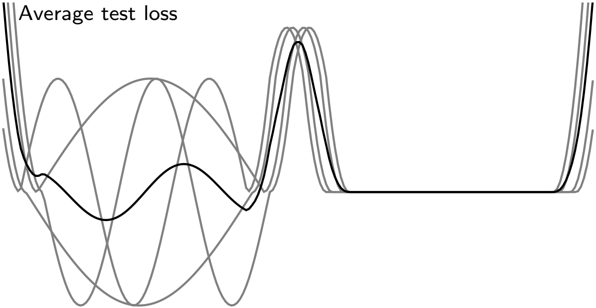
- Can flat (local) minima be different?
- Does one of the following minima generalize better?



- 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 $\|\nabla f_i(\theta)\|_2$
  - Other minima has small individual gradients $\|\nabla f_i(\theta)\|_2$
  - The latter (individually flat minima) seems to generalize better
- Want individually flat minima (with small $\|\nabla f_i(\theta)\|_2$)
  - 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 $\sigma_j(v) = (\sigma_j(v_1), \ldots, \sigma_j(v_p))$ 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 $f : \mathbb{R}^n \to \mathbb{R}^m$ 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 $f : \mathbb{R}^{p \times n} \to \mathbb{R}$ 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 $f : \mathbb{R}^{p \times n} \to \mathbb{R}^m$ 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 $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$

- The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ 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 $f : \mathbb{R}^n \to \mathbb{R}$

- 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 $f : \mathbb{R}^m \to \mathbb{R}$ 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

$$m(x_i;\theta) = W_n\sigma_{n-1}(W_{n-1}\sigma_{n-2}(\cdots(W_2\sigma_1(W_1x_i + b_1) + b_2)\cdots) + b_{n-1}) + b_n$$

- Rewrite as function with states z_j

$$\begin{aligned} & L(z_{n+1}, y_i) \\ \text{where} \quad & z_{j+1} = \sigma_j(W_jz_j + b_j) \text{ for } j \in \{1,\ldots,n\} \\ \text{and} \quad & z_1 = x_i \end{aligned}$$

where $\sigma_n(u) \equiv u$

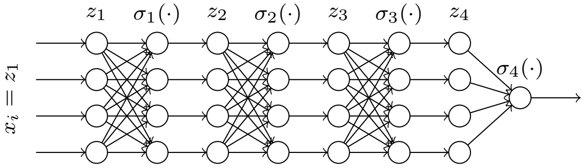
Graphical representation

- Per sample loss function

$$
\begin{aligned}
& L(z_{n+1}, y_i) \\
\text{where} \quad & z_{j+1} = \sigma_j(W_j z_j + b_j) \text{ for } j \in \{1, \dots, n\} \\
\text{and} \quad & z_1 = x_i
\end{aligned}
$$

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

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

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

- Backpropagation evaluates partial Jacobians as follows

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

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:

$$\frac{\partial L}{\partial z_{j+1}} = \underbrace{\Bigg(\underbrace{\Bigg(\underbrace{\frac{\partial L}{\partial z_{n+1}}}_{\delta_{n+1}^T} \frac{\partial z_{n+1}}{\partial z_n} \Bigg)}_{\delta_n^T} \cdots \frac{\partial z_{j+2}}{\partial z_{j+1}} \Bigg)}_{\delta_{j+1}^T}$$

- Compute

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

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

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

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

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

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

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

where $\odot$ is element-wise (Hadamard) product

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

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

- Consequently, using induction:

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

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

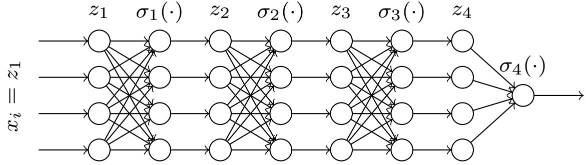
- To compute first Jacobian $\frac{\partial L}{\partial z_n}$, we need $z_n \Rightarrow$ forward pass
- Computing

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

  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:

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

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

- 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 $\sigma_n = \mathrm{Id}$

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)

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

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 $\sigma_n = \mathrm{Id}$

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 gradients

- Backpropagation composes n layers in the two passes
- Composing scalars $C = \alpha^n$ is exponential in n
  - if $\alpha \in (0, 1)$ exponential decrease (vanishing)
  - if $\alpha > 1$ exponential increase (exploding)
  - if $\alpha = 1$, we have $C = 1$
- Want gain per layer to be around 1 in backpropagation
- Achieved gain depends on
  - Choice of activation functions
  - Norms of weights

Avoiding vanishing and exploding gradients

- Assume L-Lipschitz activation with $\sigma(0) = 0$
- Forward pass estimation:

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

- Backward pass estimation:

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

- Gradients do not explode or vanish if

$$\|z_{j+1}\|_2 \approx \|z_j\|_2 \qquad \text{and} \qquad \|\delta_j\|_2 \approx \|\delta_{j+1}\|_2$$

- Suggests $L\|W_j\| \approx 1$ and $L\|b_j\|_2$ small

Residual networks

- Assume L-Lipschitz activation with $\sigma(0) = 0$
- Forward pass estimation:

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

- Backward pass estimation:

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

- Larger estimates than for non-residual networks
- To achieve $\|z_{j+1}\|_2 \approx \|z_j\|_2$ and $\|\delta_j\|_2 \approx \|\delta_{j+1}\|_2$ suggests

$L\|W_j\|$ and $\|b_j\|_2$ small

Suggestions based on upper bounds

- Suggestions
  - $L\|W_j\| \approx 1$ and $L\|b_j\|_2$ small for standard networks
  - $L\|W_j\|$ and $L\|b_j\|_2$ small for residual networks

  are based on upper bounds
- Safe to go a bit larger w.r.t. explosion
- Replace L by "average" Lipschitz constant for better estimates
  - ReLU: 0.5, α-LeakyReLU: $(1+\alpha)/2)$
  - Tanh: depends on active region (larger region, smaller constant)
- Replace operator norm $\|W_j\|$, e.g., by average singular value
  - Operator norm is maximum gain of vector (max singular value)
  - Average singular value is "average gain of vector"
- Tanh outputs are constrained to $(-1,1)$ – not taken into account

Initialization

- Initialize network to avoid vanishing and exploding gradients
- To initialize according to suggestions rely on computing
  - operator norms $\|W_j\|$ (largest singular value)
  - average non-zero singular values of W_j

  where first is expensive and second even more so
- Not possible for large networks $\Rightarrow$ Randomization!

The power of random initialization

- Random iid matrices have operator norm close to expected value
  - Probability distribution concentrated around mean
  - "Concentration of measures"
- It turns out that if $M \in \mathbb{R}^{n\times m}$ with $M \sim \mathcal{N}(0,1)$

$$\mathbb{E}[\|M\|] \approx (\sqrt{n} + \sqrt{m})$$

- If we select $(M)_{i,l} \sim \mathcal{N}(0, \frac{1}{(\sqrt{n}+\sqrt{m})^2 L^2})$

$$\|M\| = \tfrac{1}{(\sqrt{n}+\sqrt{m})L}\|L(\sqrt{n}+\sqrt{m})W\| \approx \tfrac{1}{(\sqrt{n}+\sqrt{m})L}(\sqrt{n}+\sqrt{m}) = \frac{1}{L}$$

  which for ReLU suggests $(W_j)_{i,l} \sim \mathcal{N}(0, \frac{4}{(\sqrt{n_j}+\sqrt{n_{j+1}})^2})$
- For residual networks weights can be initalized smaller

Initialization example

- Claim: $(W_j)_{i,l} \sim \mathcal{N}(0, \frac{1}{(\sqrt{n_j}+\sqrt{n_{j+1}})^2 L^2})$ implies $\|W_j\| \approx \frac{1}{L}$
- Let $L = 0.5$ and we get the following $\|W_j\|$ which should be ≈ 2

n_j / n_{j+1}	100 / 1	100 / 10	100 / 100	1000 / 1	1000 / 100	1000 / 1000
	1.91	1.97	1.96	2.02	1.98	2.00
	1.99	1.86	1.91	1.89	1.99	1.99
	1.80	1.93	1.94	1.94	1.97	2.00
	1.79	1.82	1.94	2.00	1.95	1.98
	1.73	2.02	1.90	1.87	1.98	2.00
	1.73	1.83	2.00	1.92	1.98	2.00
	1.83	1.82	1.98	1.96	1.97	1.99
	1.83	1.98	1.94	1.93	2.00	2.01
	1.69	1.85	1.97	2.00	2.00	1.99
	1.65	1.93	1.98	1.95	1.98	1.98

- Very close to $\frac{1}{L} = 2$, especially for larger dimensions
- Same results if $n_{j+1} > n_j$

Estimation from upper bounds

- Suggestion $\|W_j\| \approx \frac{1}{L}$ from upper bounds
- Can use average non-zero singular value instead of largest ($\|W\|_j$)
- For Gaussian iid matrices:
  - Average singular value typically $\alpha\|W_j\|$ with $\alpha \in [0.4, 1]$
  - Factor α smaller for square and larger for wide/thin matrices
  - Also concentrated around mean

Average singular value vs operator norm

- Claim: Average non-zero SVD typically $\alpha\|W_j\|$ with $\alpha \in [0.4, 1]$
- Table of α for different dimensions and different random matrices

n_j / n_{j+1}	100 / 1	100 / 10	100 / 100	1000 / 1	1000 / 100	1000 / 1000
	1.000	0.774	0.430	1.000	0.755	0.427
	1.000	0.767	0.443	1.000	0.762	0.425
	1.000	0.745	0.432	1.000	0.763	0.427
	1.000	0.812	0.432	1.000	0.758	0.428
	1.000	0.789	0.435	1.000	0.751	0.427
	1.000	0.800	0.436	1.000	0.754	0.427
	1.000	0.806	0.403	1.000	0.752	0.428
	1.000	0.765	0.419	1.000	0.759	0.428
	1.000	0.810	0.438	1.000	0.764	0.428
	1.000	0.787	0.433	1.000	0.753	0.427

- Concentrated around mean, especially for large square matrices
- Initialize: $(W_j)_{i,l} \sim \mathcal{N}(0, \frac{1}{(\sqrt{n_j}+\sqrt{n_{j+1}})^2 L^2 \alpha^2})$ with average L