Generalization

Arto Klami

26 March, 2025

General news

- First exercise deadline passed – any feedback?
    - Workload?
    - Difficulty?
- Grading:
    - No specific deadline, but will be done as fast as we can
    - You will see overall points in Moodle, and can ask for more specific feedback in exercise sessions

Optimization recap

Last time:

- How to optimize the weights of a neural network, by SGD with automatic differentiation
- For large enough networks we can often find a global optimum
- Speed and difficulty of learning depends on *a lot* of things: Data, model, hyperparameters, ...

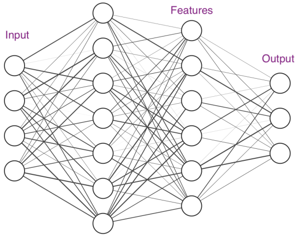
Today:

- Generalization: Why optimization is not quite enough?
- What should we do instead?
- Solely from the perspective of supervised learning

Optimization objective

Quick look at what we optimize (not worrying about generalization yet)

- The objective is always problem-specific and effectively always a *likelihood of some assumed noise model*
- Nothing specific to neural networks: The last layer is a generalized linear model, and everything before that is just for learning good features (a *representation*) $\mathbf{g}(\boldsymbol{x}) = \psi_K(\boldsymbol{W}_K(\psi_{K-1}(\boldsymbol{W}_{K-1} \dots \psi_1(\boldsymbol{W}_1\boldsymbol{x}))) = \psi_K(\boldsymbol{W}_K\boldsymbol{y}_{K-1})$
- For the loss it does not matter how the previous layers were computed, so $\boldsymbol{y}_{K-1}$ plays the role of inputs $\boldsymbol{x}$



See: Prince, Sections 5.1-5.2

Maximum log-likelihood

- Generalized linear model
    - $p = \psi(\boldsymbol{W}\boldsymbol{x})$, where p is a parameter of a distribition $p(y|p)$
    - y is a sample from that distribution: $y \sim p(y|p)$
- This means the shape and the activation function of the output layer needs to match the assumed likelihood: For binary classification, we need to output a probability, a value between $[0, 1]$ etc.
- The actual loss is often the *average* negative log-likelihood of the observation distribution (we want runs with different B to be comparable)

$$J(\boldsymbol{W}) = -\frac{1}{N}\sum_{n=1}^{N} \log p(y_n|\psi(\boldsymbol{W}\boldsymbol{x}_n))$$

Examples

- Regression:
  $y \sim \mathcal{N}(\boldsymbol{Wx}, \sigma^2)$ gives $-\frac{1}{2\sigma^2}\|y - Wx\|^2$. This is mean square error when σ^2 is just some constant
- Binary classification:
  $y \sim \text{Bernoulli}(p)$ where $p = \sigma(\boldsymbol{Wx})$, gives likelihood $p^y - (1-p)^{1-y}$ and hence log-likelihood $y\log(p) - (1-y)\log(1-p)$ (mathematically equivalent to cross-entropy)
- Multiple classes:
  - Multi-class: $p(y_k = 1) = \frac{e^{p_k}}{\sum_j e^{p_j}}$
  - Multi-label: $p(y_k = 1) = \sigma(p_k)$

Concepts

- A data set is a random sample of $\{x_n, y_n\}_{n=1}^N$ from some distribution $p(y, x)$ (in supervised learning)
- Independent samples, unless otherwise specified
- We can access the data set, but not the actual distribution
- What we care about is how well the model works on the distribution itself
- Good training data log-likelihood is not enough

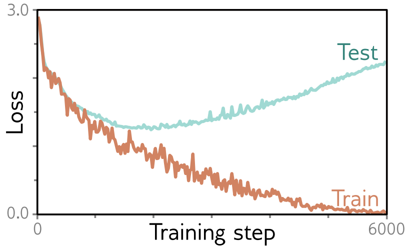
Concepts

- The real loss is sometimes called *risk*: $R(\boldsymbol{\theta}) = \mathrm{E}_{p(y,x)}[J(y,x,\boldsymbol{\theta})]$
- The training loss is called *empirical risk*: $R_e(\boldsymbol{\theta}) = \frac{1}{N}\sum_n J(y_n, x_n, \boldsymbol{\theta})$
- If we optimize $R_e(\boldsymbol{\theta})$ well, then in general $R_e < R$ and more careful optimization just makes things worse (*overfitting*)
- However, empirical loss $R_v(\boldsymbol{\theta}) = \frac{1}{N'}\sum_n J(y'_n, x'_n, \boldsymbol{\theta})$ computed on a separate *validation data* is an unbiased estimate of $R(\boldsymbol{\theta})$
- Hence: A solution with low $R_v(\boldsymbol{\theta})$ is what we want

See: *Introduction to Machine Learning*

Training, validation and test losses

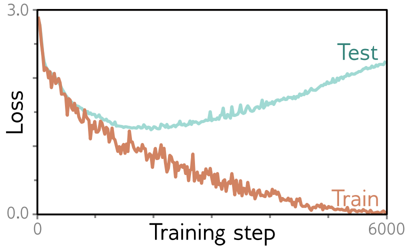
- Training and validation curves: Plot and inspect
- Comparison of $R_v(\boldsymbol{\theta})$ reveals good models
- If we compare multiple models then $R_v(\boldsymbol{\theta})$ cannot be used as estimate of the final quality. Why?



Source: Prince (2023) CC-BY-NC-ND

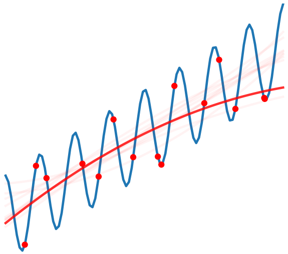
Training, validation and test losses

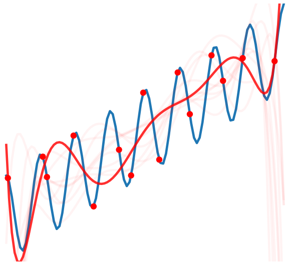
- Training and validation curves: Plot and inspect
- Comparison of $R_v(\boldsymbol{\theta})$ reveals good models
- If we compare multiple models then $R_v(\boldsymbol{\theta})$ cannot be used as estimate of the final quality. Why?
- We need yet another independent sample, to compute the test error $R_t(\boldsymbol{\theta})$

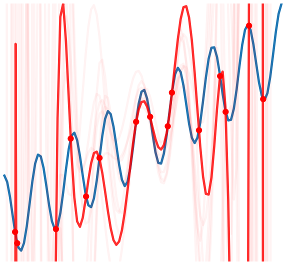


Source: Prince (2023) CC-BY-NC-ND

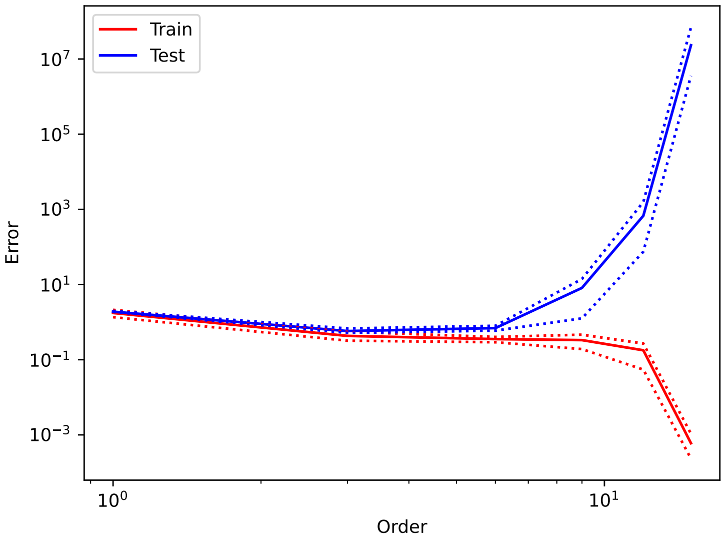
Generalization and model complexity: Example


Order=3


Order=9


Order=15

Generalization and model complexity: Example



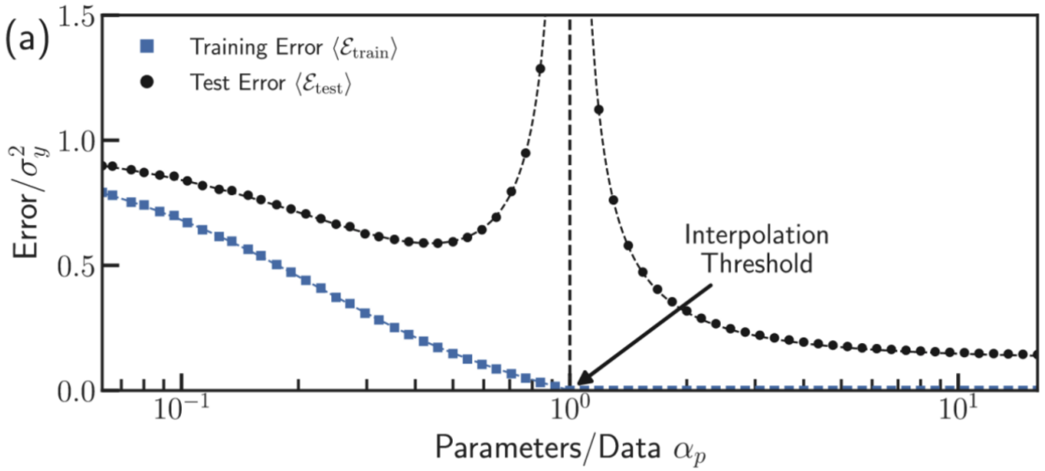
Generalization and model complexity: Double descent

Notation: N is amount of data, D is model complexity (order of polynomial, number of parameters, ...)

- Classical perspective of overfitting suggest we want a model with $D \ll N$ and that $D = N$ is horrible, but this is a bit misleading
- For NNs we need the double descent perspective, to discuss how the test error behaves for overparameterized models $D \gg N$
- For $D \gg N$ we (can) have *multiple solutions* with perfect $R_e(\boldsymbol{\theta})$
- With suitable algorithms we can find the one(s) that have better $R_v(\boldsymbol{\theta})$
- Hence the name: The test error (can) start descending again when increasing D
- Note: Not limited to deep learning – we can observe double descent also with linear models, polynomials etc.

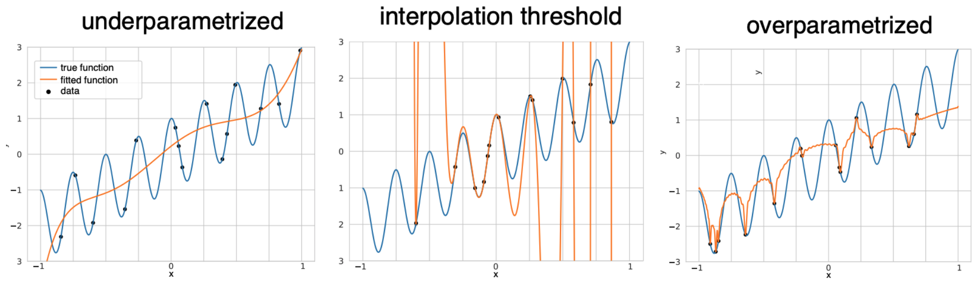
See: Prince, Section 8.4

Generalization and model complexity: Double descent



Source: Rocks et al. (2022)

Generalization and model complexity: Double descent



Source: Schaeffer et al. (2023)
https://arxiv.org/abs/2303.14151

Generalization and model complexity: Double descent

Observations and interpretations:

- Many modern DL models are severely overparameterized ($D \gg N$)
- They can work better than any of the models in the classical regime of $D < N$, but this is not guaranteed
- There is huge variety of different solutions with optimal R_e, some horribly bad and some great. It really matters which one we find.
- The interpolation threshold is poorly defined: It is a complex function of N, the model, the learning algorithm and its hyperparameters etc.

Generalization and model complexity: Double descent

Observations and interpretations: The Data paradox

- In classical regime having more data always helps: If $N_2 > N_1$ then (on average) $R_v(\boldsymbol{\theta}_2) \leq R_v(\boldsymbol{\theta}_1)$, where $\boldsymbol{\theta}_i$ refers to parameters learned on the data set with N_i
- This is not guaranteed for overparameterized models (that is, many modern DL models)!
- Denote the interpolation threshold as $\gamma(N)$. We have $\gamma(N_2) > \gamma(N_1)$
- If $D > \gamma(N_1)$ but $D \approx \gamma(N_2)$, then $R_v(\boldsymbol{\theta}_2) \gg R_v(\boldsymbol{\theta}_1)$
- Implications:
    - You need to know whether your model is in the classical or overparameterized regime
    - If you are in the overparameterized regime and observe more data, you should typically also make the model larger

See: Computer Assignment

Regularization

- We can reduce overfitting by regularization
- Explicit regularization refers to methods that manipulate the learning objective so that solutions that generalize better are being favored
- Implicit regularization refers to inherent properties of the learning algorithms or models that have regularising properties
- Explicit regularization already covered well in IML and AML and the techniques used in DL are pretty standard
- Hence: Focus here in implicit regularization

Explicit regularization

- For many models a solution with small weights corresponds to a simpler function, and hence many regularization techniques encourage small values
- Setting a parameter value exactly to zero:
  - For a polynomial can reduce the order
  - Makes a linear model sparser and helps with collinear features
  - Kills a connection in a neural network, making the model smaller
- Mostly we do not set them to exactly zero, but similar intuitions hold
- But: Zero does not always mean less complex. What about a regularizer for bias terms?

Explicit regularization

- Introduce a regularizer $\Omega(\boldsymbol{\theta}) : \mathbb{R}^D \rightarrow \mathbb{R}$ where small $\Omega(\boldsymbol{\theta})$ corresponds to a solution more likely to generalise
- A scalar multiplier is needed to balance between the loss and the regularizer $J(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta})$
- λ is a hyperparameter just as the parameters of the learning algorithm
- The choice still based on R_v, which is the actual loss on validation samples (not including the regularizer!)
- From a practical perspective, any differentiable regularizer could be used: As long as we can run AD, we can optimize the joint objective

Explicit regularization

- l_2 regularizer $\Omega = \boldsymbol{\theta}^T\boldsymbol{\theta}$ penalises for squared norm of the weights
- Historically called *weight-decay* in NN literature
- In `PyTorch` implemented as extra parameter for the *optimizer* (but could be added to a loss instead)
- Mathematically equivalent to changing from maximum likelihood to MAP estimate with Gaussian prior on the parameters

Explicit regularization: Sparse connectivity and parameter tying

- l_1 regularizer $\Omega = \sum_i |\boldsymbol{\theta}_i|$ encourages sparse connections and l_0 (that counts non-zero entries) would do so even more strongly but is not differentiable
- Somewhat rarely used in neural networks, but covered in the AML course
- In deep learning it is more common to design an architecture that *by construction* is sparse
- Example: Convolutional neural network, which replaces fully connected layer with
  - *Sparse connections:* Each neuron only connected to a few others
  - *Parameter tying:* Each neuron uses the same weights, but different inputs

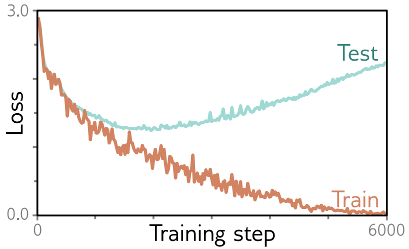
See: Lecture 7

Implicit regularization

- Implicit regularization refers to any mechanism of the model or learning algorithm that has regularising properties
- Practical DL solutions include several forms of regularization
    - The algorithm: SGD and its variants have regularising properties
    - Learning rate and batch size: Influence how the algorithm regularizes the solution
    - Initialization: Influences initial (and hence final) functions
    - Architecture: Depth and width, as well as strong inductive biases of specific architectures
    - Explicit regularization
- It is really difficult to say how big the different effects are relative to each other
- Most important thing: You need to be aware of these effects and (slowly) learn to decide how to improve your solutions

Semi-implicit regularization: Early stopping

- SGD eventually converges to some (local) optimum of R_e or $R_e + \lambda\Omega$
- The validation error R_v can start growing already before that
- We can (and perhaps should) stop the optimization before we reach the local optimum
- Both saves time and improves generalization
- Perhaps the simplest implicit regularization technique, but can still be theoretically interesting (e.g. `https://openreview.net/pdf?id=QM8oG0bz1o`)



Source: Prince (2023) CC-BY-NC-ND

Semi-implicit regularization: Early stopping

Not quite as easy as it sounds

- Mini-batch losses are highly stochastic and cannot be compared directly
- The loss may be quite stable for a long time and still start improving later, so there is a risk of terminating way too early
- Typical heuristics average the losses over a longer duration (e.g. running average over multiple iterations, full-batch loss over the whole epoch) and additionally require the validation loss to grow for a few iterations in a row
- The final choice would then be the model with the lowest validation error (in most cases a bit before we actually terminated the process)

Might be a good idea to use ready-made tools

Implicit regularization

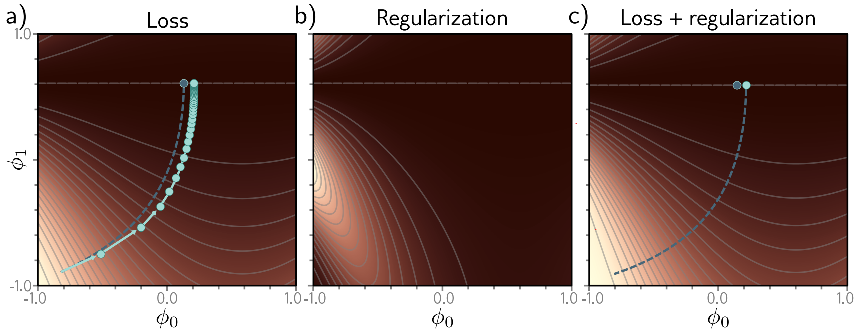
- NNs often generalize relatively well even without explicit regularization
- This must be a properly of either the model architecture or the optimization algorithms (or their combination)
- No clear single reason, but there is significant amount of alternative perspectives
- Today: Quick overview of some of these directions to encourage you to think about the aspects

Finite step size as regularization

- The GD update $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mu \nabla J \boldsymbol{\theta}_t$ can be interpreted as *approximation* for numerical integration of a differential equation $\frac{\partial \boldsymbol{\theta}}{\partial t} = -\frac{\partial J}{\partial \boldsymbol{\theta}}$
- This is the *Euler integrator*, known to be the crudest possible approximation. It diverges from the true path unless μ is really small
- The actual path we get with finite μ corresponds (approximately) to a exact solution of a *different* differential equation where the loss is replaced with

$$J(\boldsymbol{\theta}) + \frac{\mu}{4} \|\nabla J(\boldsymbol{\theta})\|^2$$

Finite step size as regularization



Source: Prince (2023) CC-BY-NC-ND

Finite step size as regularization

Basic idea of derivation

- Assume the discrete solution matches continuous solution for some modified problem
- Form second-order Taylor expansion for the modified problem
- The first two terms are exactly the discrete step, so we solve for the modification that makes the 2nd-order term zero

See: Prince, Section 9.2

Some implications

- GD corresponds to l_2 regularisation for the *gradient*, where the step length determines the amount of regularization

$$J(\boldsymbol{\theta}) + \frac{\mu}{4}\|\nabla J(\boldsymbol{\theta})\|^2$$

- Previously you were making the choice based on how well and fast the optimization converge, but you should at the same time be thinking about generalization
- Unless working with very large models, you mostly should not care that much about how long it takes to train the model, so maybe generalization should be the main aspect?

Finite step size as regularization

Why regularising gradient norm helps?

- At a local optimum we have $\|\nabla J(\boldsymbol{\theta})\| = 0$, but everywhere else the norm is non-zero
- Regularization keeps the norm smaller during the optimization, and likely also in the local neighborhood of the solution
- One explanation: Avoids ever getting to areas of the parameter space with very large gradients

Stochastic gradients as regularization

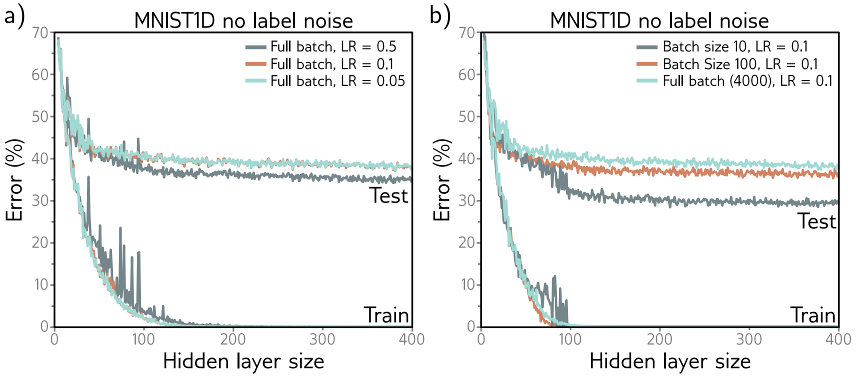
- Similar analysis can be repeated for SGD, which has both finite step size μ and noisy estimates for the gradients because of using only a subset of samples
- The objective that SGD actually optimizes is (approximately)

$$J(\boldsymbol{\theta}) + \frac{\mu}{4}\|\nabla J(\boldsymbol{\theta})\|^2 + \frac{\mu}{4B}\sum_{b=1}^{B}\|\nabla J_b(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta})\|^2$$

  where $J_b(\boldsymbol{\theta})$ is the loss for the bth mini-batch
- Intuition: The latter term is small when individual samples do not have a major effect on the gradient
- Consequence: Using small B can be good for generalization, not just for speeding up iterations

See: Assignment problem

Implicit regularization of (S)GD



Source: Prince (2023) CC-BY-NC-ND

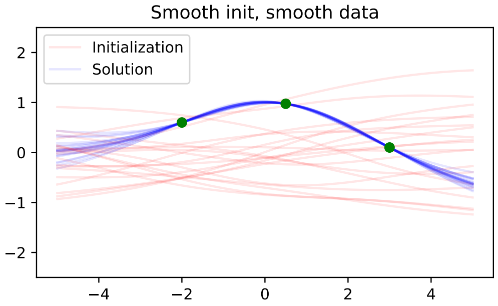
Flat minima

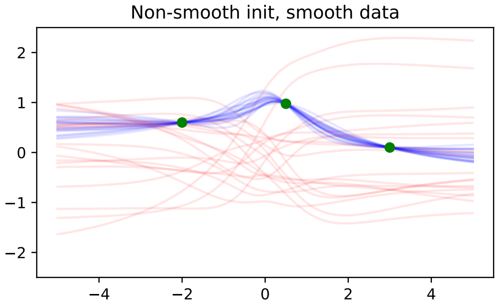
- Due to the previous effects (and maybe other reasons as well) SGD tends to find minima that are in some sense flat
- This explains why it finds such local optima, but not yet why they are good
- Intuitively: Small errors in parameter estimates matter less then then minimum is flatter
- Formal theoretical analysis still largely an open problem, but various methods have been developed to explicitly encourage flat minima and they seem to be working well: See e.g. `https://arxiv.org/abs/2010.01412`

Implicit regularization: Initialization

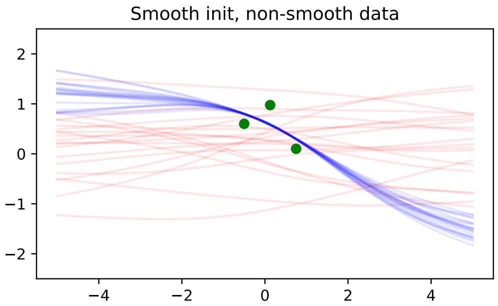
- The specific (local or global) optimum SGD finds is heavily influenced by the initial solution
- We already talked about how initialization influences the initial gradient magnitude, but it also influences the function itself
- Easiest to analyse in case of a regression NN with a single hidden layer:
  - Variance of w for the output layer influences the output scale
  - Variance of w for the hidden layer influence smoothness: With small w the $\mathbf{g}(x)$ is always smooth, since small difference in x cannot change the values much
- The final solution tends to retain the initial characteristics: If the initial function is smooth then so is the result
- The initial conditions could in principle be escaped, but it is hard for SGD

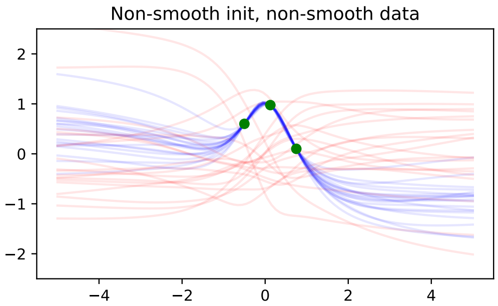
Implicit regularization: Initialization





Implicit regularization: Initialization





With smooth initialization we fail to find the global optimum, but instead SGD finds a solution that is still smooth. We implicitly regularised the solution towards smoothness

Implicit regularization: GP perspective

- Intuitively one could think a NN gets more flexible when it becomes wider (larger), but this is not the case
- For a single-layer NN with L neurons we can study the limit process where $L \to \infty$
- For $w \sim \mathcal{N}(0, \sigma^2/L)$ the function becomes a *Gaussian process* that models only smooth functions
- The examples on the previous slide had $L = 256$, and the curves look quite a bit like GP results if we re-interpret initializations as draws from a prior and the solutions as draws from the posterior
- The smoothness is governed by the shape and steepness of the activation function

Note: GPs are out of scope in this course

Batch normalization

- Batch normalization and its variants have regularizing properties: for instance, `https://arxiv.org/pdf/2103.01499.pdf` and `https://openreview.net/pdf?id=d-XzF81Wg1`
- Mentioned here for completeness; we anyway did not cover these techniques
- If you use batch normalization in your model, you should read something about the effects it has

Generalization summary

Some things to remember:

- We should not approach the question of "is this a good optimization algorithm" from the perspective of the training loss, but from the perspective of generalization
- SGD (and variants) work very well for NNs, especially for over-parameterized cases: We often do not need to worry about local optima
- Even better, SGD finds optima that are flat and tend to generalize well, and there are theoretical explanations on why this happens
- However: Does not mean we would not need to regularize the solutions in other ways too

Generalization summary

Some things to remember:

- Basic theory still holds: SGD finds optima that are close to the initial solutions, and hence initialisation has a large effect even when we find a global optimum
- Example: Smooth initializations often lead to smooth final results as well
- Double descent: Not all intuitions we have on model complexity are correct, but the counter-intuitive behaviors ("I got more data but it did not help") have theoretical explanations. You need to be aware of them
- Possible survivorshop bias: We use the kind of NN architectures we use today because they are the ones for which our current optimization tools happen to work