CSC 2541: Neural Net Training Dynamics

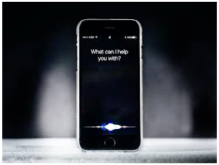
Lecture 1 - A Toy Model: Linear Regression

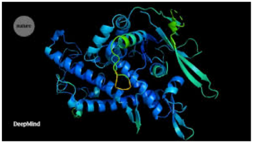
Roger Grosse

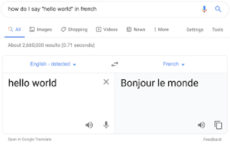
University of Toronto, Winter 2021

Introduction

Neural nets are everywhere:







how do I say "hello world" in french

About 2,660,000 results (0.71 seconds)

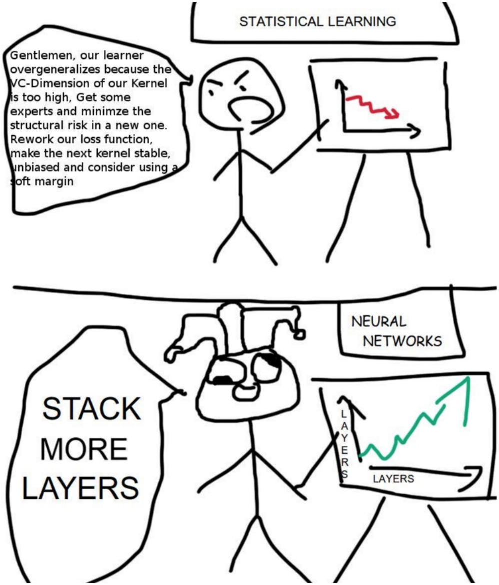
hello world

Bonjour le monde

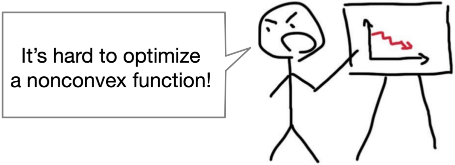
Introduction

- But how do they work?
- Some things (most) machine learning researchers believed 10 years ago:
  - It's hard to optimize nonconvex functions.
  - It's hard to train neural nets with more than 2 layers.
  - If you have way more parameters than data points, you'll overfit.
  - Regularization and optimization can be studied separately.
  - Your learning algorithm and feature representation need to be carefully designed for a particular application.
- Our experience from the last 10 years has turned each of these claims on its head — and we are just beginning to understand why!

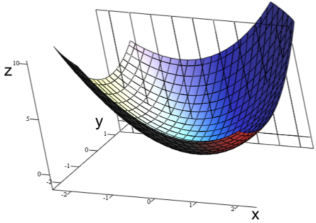
Introduction

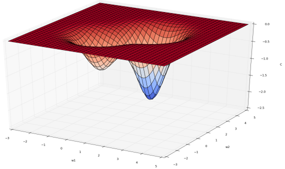

STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

Introduction

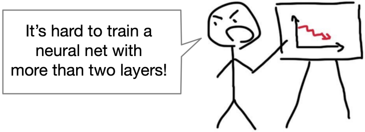


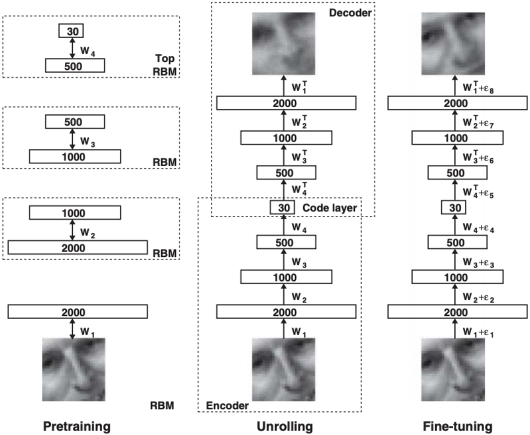
It's hard to optimize a nonconvex function!





Introduction

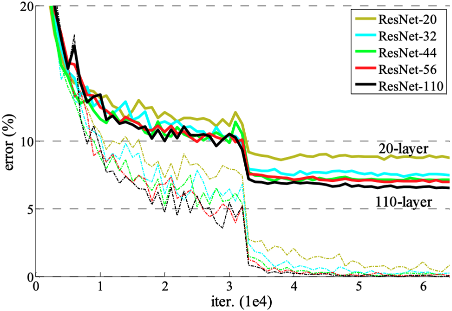




Hinton and Salakhutdinov, 2006

Introduction

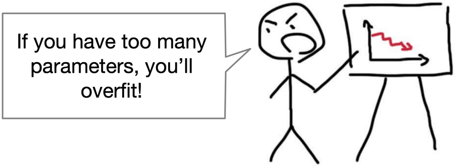
- But here are the training curves for an image classifier with millions of parameters:



He et al., 2016

- I could have picked basically any modern example!
- Generic optimization routines (only a few lines of code!)

Introduction



Theorem. Let $\mathcal{H}$ be given, and let $d = \mathrm{VC}(\mathcal{H})$. Then with probability at least $1 - \delta$, we have that for all $h \in \mathcal{H}$,

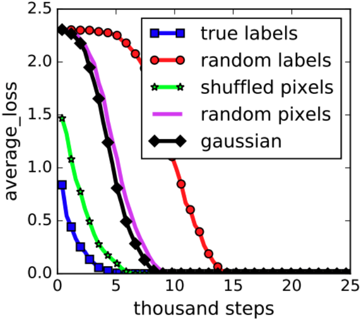
$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{d}{m}\log\frac{m}{d} + \frac{1}{m}\log\frac{1}{\delta}}\right).$$

Thus, with probability at least $1 - \delta$, we also have that:

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + O\left(\sqrt{\frac{d}{m}\log\frac{m}{d} + \frac{1}{m}\log\frac{1}{\delta}}\right).$$

Introduction

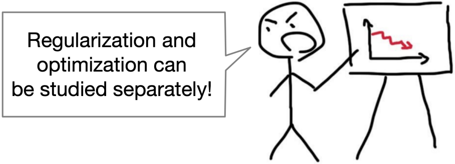
- But this image classification network is able to generalize well even though it can fit random labels:



Zhang et al., 2017

- So capacity constraints are not necessary for generalization.

Introduction

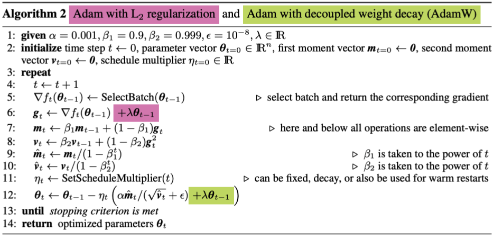


- E.g., weight decay was understood as implementing Tikhonov regularization, a well-understood concept from statistics.

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} \left(\mathcal{J} + \frac{\lambda}{2} \lambda \|\mathbf{w}\|^2 \right) \\ &= (1 - \alpha\lambda)\mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}\end{aligned}$$

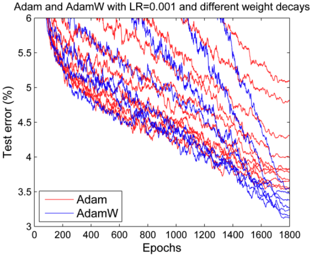
Introduction

- This analysis predicts that if you change the optimization algorithm, you should keep the same objective function (with the $\|\mathbf{w}\|^2$ penalty).
- But for various optimization algorithms, it works much better to literally apply weight decay, even though this appears to be optimizing a different function!



Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

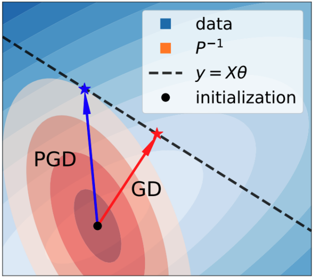
```
given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    $t \leftarrow t + 1$
    $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$    ▷ select batch and return the corresponding gradient
    $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
    $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$    ▷ here and below all operations are element-wise
    $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
    $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$    ▷ $\beta_1$ is taken to the power of $t$
    $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$    ▷ $\beta_2$ is taken to the power of $t$
    $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$    ▷ can be fixed, decay, or also be used for warm restarts
    $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left( \alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1} \right)$
until stopping criterion is met
return optimized parameters $\boldsymbol{\theta}_t$
```



Loschilov and Hutter, 2019

Introduction

- Also, for overparameterized models, different optimization algorithms can converge to different optimal solutions.
- This is a type of *implicit regularization*.



Amari et al., 2020

Introduction

- Common theme: need to understand the training dynamics
  - If you're minimizing a (strongly) convex function, you only have to understand the properties of the unique global optimum.
  - Neural net training is nonconvex, so we could wind up at various local optima depending on the path the optimizer takes
  - Also, most modern nets are overparameterized, so there are (infinitely) many different global optimal we could wind up at.

This Course

This Course

Course staff

- Instructor: Roger Grosse
- TAs: Cem Anil, Guodong Zhang

Course web page:

`https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/`

This Course

Topics

- **Weeks 1–4:** Foundational concepts
    - **Today:** Linear regression as a toy model
    - **Week 2:** Taylor approximations (Jacobian-vector products, Hessians, etc.)
    - **Week 3:** Metrics (function space distance, natural gradient)
    - **Week 4:** Second-order optimization
- **Weeks 5–9:** Understanding neural net training
- **Weeks 10–12:** Game dynamics and bilevel optimization

This Course

Topics

- **Weeks 1–4:** Foundational concepts
- **Weeks 5–9:** Understanding neural net training
  - **Week 5:** Adaptive gradient methods, normalization, weight decay
  - **Week 6:** Infinite limits (neural tangent kernel, infinite depth)
  - **Week 7:** Stochasticity
  - **Week 8:** Implicit regularization
  - **Week 9:** Momentum
- **Weeks 10–12:** Game dynamics and bilevel optimization

This Course

Topics

- **Weeks 1–4:** Foundational concepts
- **Weeks 5–9:** Understanding neural net training
- **Weeks 10–12:** Game dynamics and bilevel optimization
    - **Week 10:** Differential games and minmax optimization (e.g. GANs)
    - **Weeks 11–12:** Bilevel optimization (e.g. MAML, hyperparameter adaptation)

This Course

Prerequisites

- Linear algebra
- Probability theory
- Multivariable calculus
- **A course in machine learning (enforced!)**
  - You will need to fill out a Google Form saying what course you're using to meet this prerequisite. Due Wednesday 1/20.
- A course in neural nets (e.g. CSC2516) is useful but not required.
  - You only need to know the very basics of neural nets to follow the lectures.
  - You may want to read the CSC2516 materials for tasks/architectures you're using for your final project.

This Course: Coursework

- Marking scheme:
  - **15%** Problem set (due 2/10)
  - **25%** Colab notebook and paper presentation
  - **10%** Final project proposal (due 2/17)
  - **50%** Final project report (due 4/7)
- The problem set is already posted.
  - Covers Lectures 1–3.
  - The longest/hardest problem only depends on this lecture, so you can get started now.

This Course: Coursework

Colab notebook and paper presentation

- Form groups of 2–3
- Pick a paper from the spreadsheet (posted to Quercus)
- Create a Colab notebook that illustrates at least one of the key ideas from the paper.
- Give a 20-minute presentation to the class that summarizes the paper and presents your notebook.
- This assignment is due on the class meeting that the paper is discussed.

The sign-up sheet will be posted once the enrollment list is finalized.

This Course: Coursework

Final project

- Form groups of 2–3
- Carry out a small research project related to the course content, e.g.
    - invent a new algorithm/architecture
    - explain a phenomenon
    - apply the techniques to make a DL system work better
    - test the course ideas in new settings (e.g. transformers, graph neural nets, generative models, etc.)
- Project proposal (due 2/17): main purpose is for us to give you feedback on your project
- Final report (due 4/7): conference paper style, about 8 pages
- See website for more details

This Course: Software

- For software, we'll use JAX, a deep learning framework for Python
- Newer than TensorFlow/PyTorch, so maybe still some rough edges
- Advantages
  - Clean, NumPy-like API
  - Excellent support for forward derivatives and higher-order derivatives
  - Functional style, user maintains state explicitly. Avoids lots of potential bugs (especially random number generation).
  - Easily target TPU architectures.
  - Parallelize/vectorize your code with pmap/vmap
  - Fun to program in
- JAX tutorial today, right after this lecture, same Zoom meeting
- You're welcome to use whatever framework you like for the final project.

Gradient Descent for Linear Regression

Recap: Linear Regression

- Linear regression assumes a linear model with parameters $\mathbf{w}$, b.

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b$$

- Loss function penalizes the squared error from the true label:

$$\mathcal{L}(y, t) = \tfrac{1}{2}\|y - t\|^2$$

- Given a finite training set $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$
- The cost function is the mean of the losses over all training examples:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}^{(i)}, \mathbf{w}), t^{(i)})$$

- Simplifying the notation with a homogeneous coordinate:

$$\breve{\boldsymbol{\Phi}}(\mathbf{x}) = \begin{pmatrix} \boldsymbol{\Phi}(\mathbf{x}) \\ 1 \end{pmatrix} \qquad \breve{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix}$$

- In vectorized form, this is a quadratic cost function:

$$\mathcal{J}(\mathbf{w}) = \tfrac{1}{2N}\|\breve{\boldsymbol{\Phi}}\breve{\mathbf{w}} - \mathbf{t}\|^2$$

Recap: Gradient Descent

- Gradient descent update rule:

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}^{(k)}) \tag{1}$$

- It's a local search algorithm, so in general it can get stuck in local optima. But linear regression is a convex optimization problem, so for a small enough learning rate, it will converge to a global optimum.
- We can exactly analyze the dynamics of gradient descent for convex quadratics, gaining a lot of insight:

$$\mathcal{J}(\mathbf{w}) = \tfrac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w} + \mathbf{b}^\top \mathbf{w} + c,$$

  where **A** is symmetric
- Gradient descent update:

$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \alpha(\mathbf{A}\mathbf{w} + \mathbf{b})$$

Gradient Descent: Some Observations

- Perhaps the first question to ask about an iterative algorithm: what are its fixed points? I.e., for what values of $\mathbf{w}^{(k)}$ does $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)}$?
- For gradient descent on a differentiable function, the fixed points are the stationary points of $\mathcal{J}$:

$$\mathbf{w} = \mathbf{w} - \alpha \nabla \mathcal{J}(\mathbf{w}) \qquad \Longleftrightarrow \qquad \nabla \mathcal{J}(\mathbf{w}) = \mathbf{0}$$

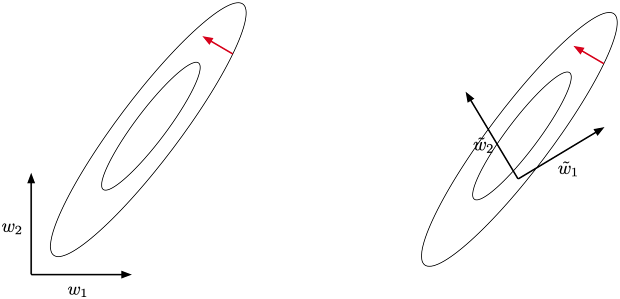
- In general, fixed points may be stable (e.g. local optima) or unstable (e.g. saddle points). Linear regression is convex, so all fixed points are stable.
- For a convex quadratic:

$$\nabla \mathcal{J}(\mathbf{w}) = \mathbf{A}\mathbf{w} + \mathbf{b} = \mathbf{0} \qquad \Longleftrightarrow \qquad \mathbf{A}\mathbf{w} = -\mathbf{b}$$

  - When does this have a solution? A unique solution?
  - If the solution isn't unique, where we end up depends on the initialization!

Gradient Descent: Some Observations

- Another important question: what are the algorithm's invariances?
- Claim: gradient descent is invariant to rigid transformations (rotation, reflection, translation)



- We often informally call this rotation invariance
- Can you give me an example of an optimization algorithm which isn't rotation invariant?

Gradient Descent: Invariance

- Rigid transformations can be written as:

$$\tilde{\mathbf{w}} = \mathcal{T}(\mathbf{w}) = \mathbf{Q}^\top(\mathbf{w}^{(0)} - \mathbf{t})$$

for orthogonal $\mathbf{Q}$

- Inverse transformation:

$$\mathbf{w} = \mathcal{T}^{-1}(\tilde{\mathbf{w}}) = \mathbf{Q}\tilde{\mathbf{w}} + \mathbf{t}$$

- This is a reparameterization, or change-of-basis. The cost function can be re-expressed in the new coordinate system:

$$\tilde{\mathcal{J}}(\tilde{\mathbf{w}}) = \mathcal{J}(\mathcal{T}^{-1}(\tilde{\mathbf{w}})) = \mathcal{J}(\mathbf{Q}\tilde{\mathbf{w}} + \mathbf{t}).$$

- Want to show that gradient descent in both coordinate systems results in equivalent trajectories.
- Mathematically: initialize $\tilde{\mathbf{w}}^{(0)} = \mathcal{T}(\mathbf{w}^{(0)})$. Want to show that

$$\tilde{\mathbf{w}}^{(k)} = \mathcal{T}(\mathbf{w}^{(k)}) \qquad \text{for all } k.$$

Gradient Descent: Invariance

Proof by Induction:

- Base case: covered by initialization,

$$\tilde{\mathbf{w}}^{(0)} = \mathcal{T}(\mathbf{w}^{(0)})$$

- Inductive step: assuming $\tilde{\mathbf{w}}^{(k)} = \mathcal{T}(\mathbf{w}^{(k)})$,

$$\begin{aligned}\tilde{\mathbf{w}}^{(k+1)} &= \tilde{\mathbf{w}}^{(k)} - \alpha \nabla \tilde{\mathcal{J}}(\tilde{\mathbf{w}}^{(k)}) \\ &= \tilde{\mathbf{w}}^{(k)} - \alpha \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{w}^{(k)}) \\ &= \mathbf{Q}^\top (\mathbf{w}^{(k)} - \mathbf{t}) - \alpha \mathbf{Q}^\top \nabla \mathcal{J}(\mathbf{w}^{(k)}) \\ &= \mathcal{T}(\mathbf{w}^{(k+1)})\end{aligned}$$

Gradient Descent: Invariance

- Because of rotation invariance, we are free to rotate to another coordinate system where the dynamics are easier to analyze.
- Recall the Spectral Decomposition of symmetric matrices:

  $$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top,$$

  where $\mathbf{Q}$ is an orthogonal matrix whose columns are the eigenvectors of $\mathbf{A}$, and $\mathbf{D}$ is a diagonal matrix whose diagonal entries are the eigenvalues.
- $\tilde{\mathbf{w}} = \mathcal{T}(\mathbf{w}) = \mathbf{Q}^\top\mathbf{w}$ is a very convenient coordinate system to rotate to because $\tilde{\mathbf{A}} = \mathbf{Q}^\top\mathbf{A}\mathbf{Q}$ is diagonal!

Gradient Descent: Invariance

- **A shorthand:** we may assume without loss of generality (WLOG) that [property P].
- **Translation:** The problem can be transformed to an equivalent one where P holds.
- E.g.,
  - When analyzing gradient descent, we may assume WLOG that $\mathbf{A}$ is diagonal, because the algorithm is rotation invariant.
  - When analyzing Adam or coordinate descent, we can't assume this, since the algorithims aren't rotation invariant.
  - We can't assume WLOG that matrices $\mathbf{A}$ and $\mathbf{B}$ are both diagonal, since diagonalizing $\mathbf{A}$ fixes a rotation.

Gradient Descent: Coordinatewise Dynamics

- If $\tilde{\mathbf{A}}$ is diagonal, then each coordinate evolves independently as:

$$\tilde{w}_j^{(k+1)} \leftarrow \tilde{w}_j^{(k)} - \alpha(\tilde{a}_j \tilde{w}_j^{(k)} + \tilde{b}_j).$$

- We can analyze this into different cases.
- **Case 1:** $\tilde{a}_j > 0$
  - Unique fixed point given by

$$\tilde{w}_{\star j} = -\tilde{b}_j / \tilde{a}_j$$

  - Solving the recurrence:

$$\tilde{w}_j^{(k)} = \tilde{w}_{\star j} + (1 - \alpha \tilde{a}_j)^k (\tilde{w}_j^{(0)} - \tilde{w}_{\star j}).$$

  - **Case 1(a):** $0 < \alpha \tilde{a}_j < 2$. Iterates converge exponentially to $\tilde{w}_{\star j}$.
    - Converges monotonically if $0 < \alpha \tilde{a}_j < 1$, oscillates if $1 < \alpha \tilde{a}_j < 2$.
  - **Case 1(b):** $\alpha \tilde{a}_j = 2$. Iterates oscillate and never converge.
  - **Case 1(c):** $\alpha \tilde{a}_j > 2$. Iterates diverge exponentially.

Gradient Descent: Coordinatewise Dynamics

- **Case 2:** $\tilde{a}_j = 0$ and $\tilde{b}_j \neq 0$. Recurrence is solved by

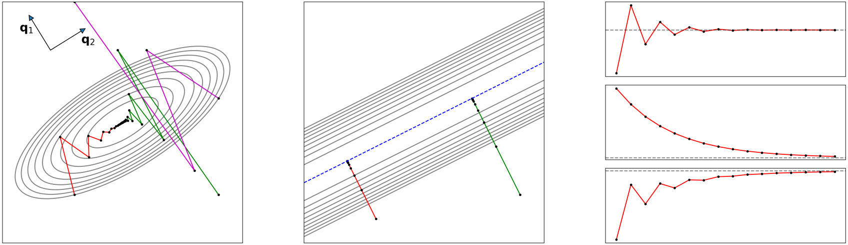
$$\tilde{w}_j^{(k)} = \tilde{w}_j^{(0)} - \alpha k \tilde{b}_j,$$

so iterates diverge linearly.

- **Case 3:** $\tilde{a}_j = 0$ and $\tilde{b}_j = 0$. Then $\tilde{w}_j$ is never updated, so

$$\tilde{w}_j^{(k)} = \tilde{w}_j^{(0)} \qquad \text{for all } k.$$

Gradient Descent: Coordinatewise Dynamics



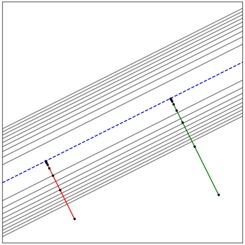
Gradient Descent

- Summarizing all the above analysis into an equation:

$$\mathbf{w}^{(k)} = \mathbf{w}^{(\infty)} + (\mathbf{I} - \alpha \mathbf{A})^k (\mathbf{w}^{(0)} - \mathbf{w}^{(\infty)})$$

- The stationary solution $\mathbf{w}^{(\infty)}$ is, among all min-cost solutions, the one closest to the initialization.
- I.e., it is the projection of $\mathbf{w}^{(0)}$ onto the min-cost subspace:

$$\mathbf{w}^{(\infty)} = \arg\min_{\mathbf{w}} \|\mathbf{w} - \mathbf{w}^{(0)}\|^2 \quad \text{s.t.} \quad \mathbf{w} \in \arg\min_{\mathbf{w}'} \mathcal{J}(\mathbf{w}'),$$



Gradient Descent: Stationary Solution

- $\mathbf{A}^\dagger$ denotes the pseudo-inverse of $\mathbf{A}$:

  $$\mathbf{A}^\dagger = \mathbf{Q}\mathbf{D}^\dagger\mathbf{Q}^\top,$$

  where $\mathbf{Q}\mathbf{D}^\dagger\mathbf{Q}^\top$ is the spectral decomposition, and $\mathbf{D}^\dagger$ is a diagonal matrix that inverts the nonzero diagonal entries of $\mathbf{D}$.
  - $\mathbf{A}^\dagger = \mathbf{A}^{-1}$ if $\mathbf{A}$ is invertible.

Gradient Descent: Stationary Solution

- Closed form for the stationary solution starting from $\mathbf{w}^{(0)} = \mathbf{0}$:

$$\mathbf{w}^{(\infty)} = -\mathbf{A}^{\dagger}\mathbf{b}$$

- For a matrix $\mathbf{B}$, the pseudoinverse is defined as:

$$\mathbf{B}^{\dagger} = \mathbf{V}\mathbf{S}^{\dagger}\mathbf{U}^{\top},$$

  where $\mathbf{U}\mathbf{S}\mathbf{V}^{\top}$ is the SVD of $\mathbf{B}$, and $\mathbf{S}^{\dagger}$ is defined like $\mathbf{D}^{\dagger}$.

- This can be written as

$$\mathbf{B}^{\dagger} = (\mathbf{B}^{\top}\mathbf{B})^{-1}\mathbf{B}^{\top}$$

  if $(\mathbf{B}^{\top}\mathbf{B})^{-1}$ is invertible.

- If $\mathbf{B}$ is square and invertible, then $\mathbf{B}^{\dagger} = \mathbf{B}^{-1}$.

- If $\mathbf{A}$ is symmetric, then:

$$\mathbf{A}^{\dagger} = \mathbf{Q}\mathbf{D}^{\dagger}\mathbf{Q}^{\top},$$

  where $\mathbf{Q}\mathbf{D}^{\dagger}\mathbf{Q}^{\top}$ is the spectral decomposition, and $\mathbf{D}^{\dagger}$ is a diagonal matrix that inverts the nonzero diagonal entries of $\mathbf{D}$.

Gradient Descent: Convergence

What happens to the cost function?

- Keeping the transformed coordinate system, the loss decomposes into an independent term for each coordinate:

$$\tilde{\mathcal{J}}(\tilde{\mathbf{w}}) = \sum_{j:\tilde{a}_j > 0} \frac{\tilde{a}_j}{2}(\tilde{w}_j - \tilde{w}_{\star j})^2 + c$$

  for a constant c.

- Recall:

$$\tilde{w}_j^{(k)} - \tilde{w}_{\star k} = (1 - \alpha \tilde{a}_j)^k (\tilde{w}_j^{(0)} - \tilde{w}_{\star k})$$

  So its contribution to the loss decreases exponentially, by a factor of $(1 - \alpha \tilde{a}_j)^2$ in each iteration.

- Speed of convergence

$$-\ln(1 - \alpha \tilde{a}_j)^2 \approx 2\alpha \tilde{a}_j \qquad \text{if } \alpha \tilde{a}_j \ll 1$$

Gradient Descent: Convergence

- Speed of convergence:

$$-\ln(1-\alpha\tilde{a}_j)^2 \approx 2\alpha\tilde{a}_j \qquad \text{if } \alpha\tilde{a}_j \ll 1$$

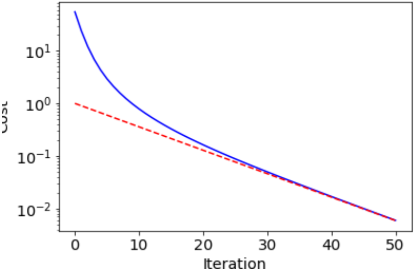
- Set $\alpha \leq \tilde{a}_{\max}^{-1}$ for stability
- Lowest (nonzero) curvature direction converges the slowest, at rate $\alpha\tilde{a}_{\min} < \tilde{a}_{\max}^{-1}\tilde{a}_{\min}$
- These dimensions will eventually dominate the loss, so the convergence rate is bounded by κ^{-1}, where κ is the condition number:

$$\kappa = \tilde{a}_{\max}/\tilde{a}_{\min}$$

  - $\kappa \approx 1$: well-conditioned, fast convergence
  - $\kappa \gg 1$: ill-conditioned, slow convergence

Gradient Descent: Convergence

E.g.,



Blue = total cost
Red = cost from min curvature direction

Back to linear regression...

Gradient Descent for Linear Regression

- Cost function:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\breve{\mathbf{\Phi}}\breve{\mathbf{w}} - \mathbf{t}\|^2$$

- Convex quadratic cost:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{A}\mathbf{w} + \mathbf{b}^\top \mathbf{w} + c,$$

- So

$$\mathbf{A} = \frac{1}{N}\breve{\mathbf{\Phi}}^\top \breve{\mathbf{\Phi}}$$
$$\mathbf{b} = -\frac{1}{N}\breve{\mathbf{\Phi}}^\top \mathbf{t}$$

- Stationary solution (starting from $\breve{\mathbf{w}}^{(0)} = \mathbf{0}$):

$$\breve{\mathbf{w}}^{(\infty)} = -\mathbf{A}^\dagger \mathbf{b} = \breve{\mathbf{\Phi}}^\dagger \mathbf{t}$$

Gradient Descent for Linear Regression

- Compare with ridge regression, or L_2-regularized linear regression:

$$\mathcal{J}_\lambda(\breve{\mathbf{w}}) = \tfrac{1}{2N}\|\breve{\mathbf{\Phi}}\breve{\mathbf{w}} - \mathbf{t}\|^2 + \frac{\lambda}{2}\|\breve{\mathbf{w}}\|^2,$$

- For $\lambda > 0$, there's a unique optimal solution:

$$\breve{\mathbf{w}}_\lambda = \arg\min_{\breve{\mathbf{w}}} \mathcal{J}_\lambda(\breve{\mathbf{w}}) = (\breve{\mathbf{\Phi}}^\top \breve{\mathbf{\Phi}} + \lambda \mathbf{I})^{-1} \breve{\mathbf{\Phi}}^\top \mathbf{t}.$$

- Can show that

$$\lim_{\lambda \to 0} \breve{\mathbf{w}}_\lambda = \breve{\mathbf{\Phi}}^\dagger \mathbf{t},$$

  which agrees with $\breve{\mathbf{w}}^{(\infty)}$ for gradient descent on an unregularized model. This is an example of implicit regularization.

Why do we normalize the features?

Why Normalize?

- A common trick when training machine learning models is to normalize, or standardize, the inputs to zero mean and unit variance:

$$\tilde{\phi}_j(\mathbf{x}) = \frac{\phi_j(\mathbf{x}) - \mu_j}{\sigma_j}$$

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} \phi_j(\mathbf{x}^{(i)})$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (\phi_j(\mathbf{x}^{(i)}) - \mu_j)^2$$

- Why is this a good idea?

Why Normalize?

- Recall: the convergence rate of gradient descent depends on the condition number κ, the ratio of the largest and smallest eigenvalues.
- Can show that

$$\mathbf{A} = \begin{pmatrix} \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^\top & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{pmatrix},$$

  where $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}(\mathbf{x}^{(i)})$ is the empirical mean and $\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{\phi}(\mathbf{x}^{(i)}) - \boldsymbol{\mu})(\boldsymbol{\phi}(\mathbf{x}^{(i)}) - \boldsymbol{\mu})^\top$ is the empirical covariance.
- **Example 1:** Suppose $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$. (The data are said to be white, as in "white noise".) Then $\mathbf{A} = \mathbf{I}$, so $\kappa = 1$, and the problem is perfectly well-conditioned. Gradient descent converges in one step.

Why Normalize?

$$\mathbf{A} = \begin{pmatrix} \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^\top & \boldsymbol{\mu} \\ \boldsymbol{\mu}^\top & 1 \end{pmatrix}$$

- **Example 2:** Suppose $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma}$ is diagonal. Then

  $$\mathbf{A} = \begin{pmatrix} \boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix},$$

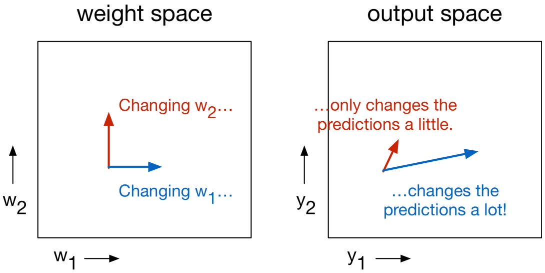
  and κ depends on the eigenvalues of $\boldsymbol{\Sigma}$. Convergence is slower.

- **Example 3:** Suppose the data are uncentered, i.e. $\boldsymbol{\mu} \neq \mathbf{0}$, and $\boldsymbol{\Sigma}$ is diagonal with bounded entries. It turns out that $\mathbf{A}$ has an outlier eigenvalue of roughly $\|\boldsymbol{\mu}\|^2 + 1$, in roughly the direction $(\boldsymbol{\mu}^\top \; 1)^\top$.
  - Note that $\|\boldsymbol{\mu}\|^2$ grows linearly in the dimension D, while the remaining eigenvalues are bounded.
  - This can be really badly conditioned in high-dimensional spaces!

Why Normalize?

Intuition

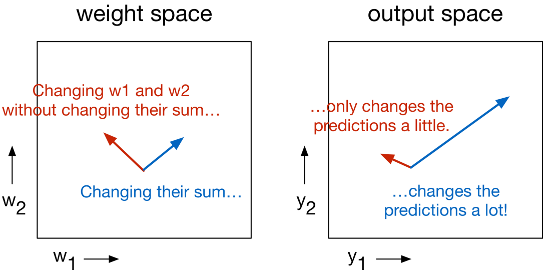
x_1	x_2	t
114.8	0.00323	5.1
338.1	0.00183	3.2
98.8	0.00279	4.1
⋮	⋮	⋮


weight space
Changing w2…
Changing w1…
w2
w1
output space
…only changes the predictions a little.
…changes the predictions a lot!
y2
y1

Why Normalize?

Intuition

x_1	x_2	t
1003.2	1005.1	3.3
1001.1	1008.2	4.8
998.3	1003.4	2.9
$\vdots$	$\vdots$	$\vdots$


weight space
output space
Changing w1 and w2 without changing their sum…
Changing their sum…
w2
w1
…only changes the predictions a little.
…changes the predictions a lot!
y2
y1

Why Normalize?

- So convergence is faster when $\boldsymbol{\mu}$ is closer to $\mathbf{0}$ and $\boldsymbol{\Sigma}$ is closer to $\mathbf{I}$.
  - Centering sets $\boldsymbol{\mu} = \mathbf{0}$, which eliminates the outlier eigenvalue.
  - Normalization corrects for the variances of different features, but not for correlations between features.
- It's possible to go a step further and whiten the data.
  - Map $\mathbf{x} \to \mathbf{S}^{-1}\mathbf{x}$, where $\mathbf{S}$ is a matrix such that $\mathbf{S}\mathbf{S}^\top = \boldsymbol{\Sigma}$. (For instance, the matrix square root $\boldsymbol{\Sigma}^{1/2}$.)
  - Then $\tilde{\Sigma} = \mathbf{I}$, so the problem is perfectly well conditioned.
- Is whitening a good idea?

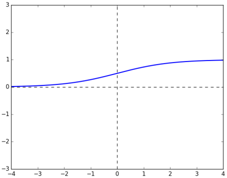
Pitfalls of Full Whitening

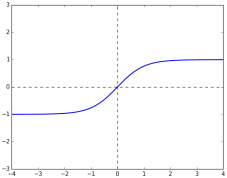
- Whitening always improves convergence of gradient descent, for linear regression, on the training set.
- But we also care about generalization!
- The highest variance directions are the principal components, which we believe contain a lot of the signal.
  - Converging faster in these directions can be a useful inductive bias, which is removed by whitening.
  - By contrast, the means and variances contain less useful information, since they depend on on arbitrary choice of units. So normalization is generally OK.
- **The lesson:** when making a change to speed up convergence, ask if it's doing it in a way that's useful for generalization!

Normalization: Neural Nets

Does this apply to neural nets?

- Centering and normalizing the inputs are a standard preprocessing step, even for neural nets
- Uncentered hidden activations create ill-conditioning too, and this is not straightforward to prevent.
  - **Classic trick:** use tanh rather than logistic activation function





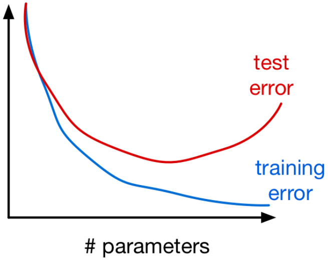
  - Activations can become uncentered due to internal covariate shift, and batch normalization was designed partly to counteract this
  - We can show that uncentered hidden activations create large eigenvalues in the Hessian (but this requires more machinery)

Double Descent

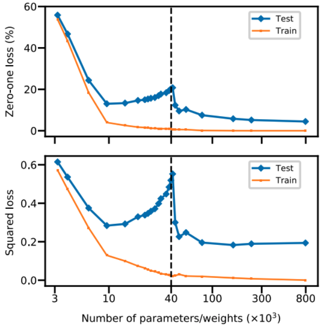
Double Descent

How does generalization depend on dimensionality?

The cartoon picture



What often happens

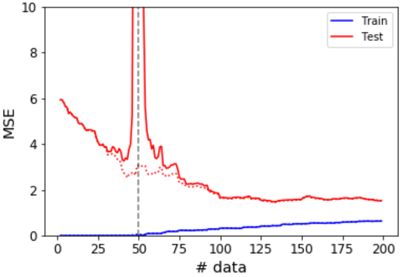


Belkin et al. (2019)

This phenomenon is known as double descent

Double Descent

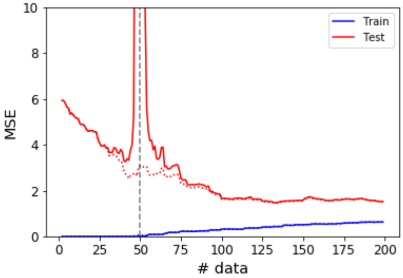
- Can we build a linear regression model of this phenomenon? If so, then maybe we can analyze it.
- No straightforward way to vary the complexity of the model. Instead, we'll fix the dimension $D = 50$ and vary N, the number of training examples.
- Double descent still happens!



Double Descent

Intuition:

- **Case 1:** $N \gg D$. There's way more than enough data to pin down the optimal parameters, so it generalizes well.
- **Case 2:** $N \approx D$. It can memorize the training set, but just barely. It might need a large $\|\mathbf{w}\|$ to do so.
- **Case 3:** $N \ll D$. It can fit the training set easily. The implicit regularization of gradient descent makes it do so with a small $\|\mathbf{w}\|$.

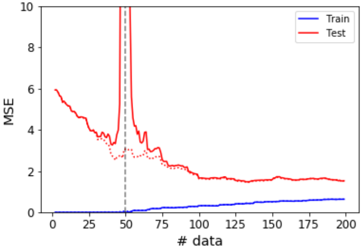


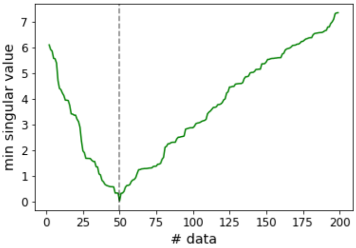
Double Descent

- Recall the stationary solution

$$\breve{\mathbf{w}}^{(\infty)} = \breve{\boldsymbol{\Phi}}^{\dagger}\mathbf{t}$$

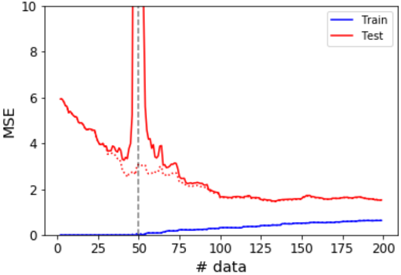
- Roughly speaking, $\breve{\mathbf{w}}^{(\infty)}$ is large when $\breve{\boldsymbol{\Phi}}^{\dagger}$ is large. This happens when $\breve{\boldsymbol{\Phi}}$ has small singular values.
- The minimum singular value is small exactly at the double descent point! (Basic result from random matrix theory, but beyond the scope of this course.)





Double Descent

- Adding explicit regularization removes any trace of the double descent phenomenon:



- Double descent is best regarded as a pathology, but it's one that still applies to a lot of state-of-the-art neural nets.

Discussion

Discussion

- When we prove something about linear regression, what does that tell us about neural nets?
    - In principle, nothing. Neural nets are much more complicated and can behave completely differently.
    - We have no guarantees.
    - But it's hard to prove any nontrivial guarantees about practical neural nets anyway. Proofs about neural nets usually require very idealized conditions.
- Instead, the goal of mathematical analysis is to provide *insight* into what happens during training.
    - Simpler model systems can provide more insight precisely because they yield more detailed predictions.
    - Like an empirical science, we need to validate our model by seeing if it makes surprising predictions that we can confirm experimentally.

Discussion

Why spend so much time on linear regression?

- It’s an important model system that we can analyze in detail, and often yields good predictions about neural nets.
    - Part of a toolbox that also includes noisy quadratic objectives, linear neural networks, Gaussian processes, matrix completion, bilinear games, ...
- We can approximate local convergence behavior by taking the second-order Taylor approximation around the optimum, reducing it to the convex quadratic case. (Topic of next lecture.)
- Amazingly, neural nets in certain settings are approximated well by linear regression with random features! (Topic of Lecture 6.)