Mathematics of Deep Learning

Stability and robustness

Lessons: Kevin Scaman

Class overview

1. Introduction and general overview 16/01
2. Non-convex optimization 23/01
3. Structure of ReLU networks and group invariances 06/02
4. Approximation guarantees 13/02
5. **Stability and robustness** 20/02
6. Infinite width limit of NNs 27/02
7. Generative models 12/03
8. Exam 19/03

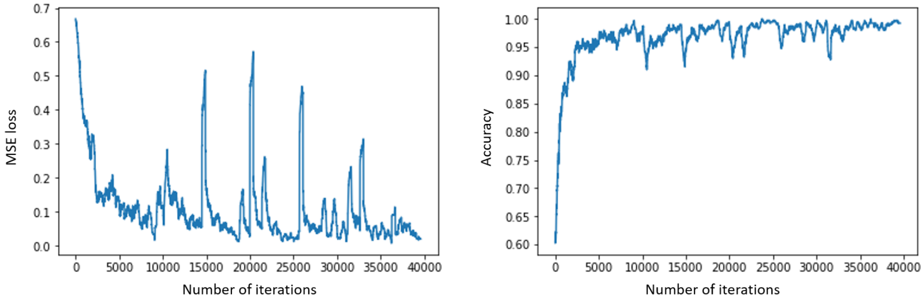
Stability during training

Weights initialization, gradient vanishing and explosion

Stability during training

Example with simple RNNs (Elman networks, no gating mechanisms)

- The gradients are sometimes **very large**.
- This leads to a large **drop in accuracy**.
- Results are **quite random**, final performance depends on initialization.



Gradient vanishing and explosion

Breaking gradient descent

- If θ_t are the iterates of the parameters learned using stochastic gradient descent on minibatches $(x_{t,i}, y_{t,i})_{i\in[\![1,K]\!]}$ at time t, then we have

$$\theta_{t+1} = \theta_t - \frac{\eta}{K}\sum_i \nabla\mathcal{L}_{x_{t,i},y_{t,i}}(\theta)\,,$$

  where $\mathcal{L}_{x,y}(\theta) = \ell(g_\theta(x), y)$.
- **Gradient vanishing:** When the gradients $\nabla\mathcal{L}_{x_{t,i},y_{t,i}}(\theta)$ are very small compared to θ_t, the iteration does not modify the parameters.
- **Gradient explosion:** When the gradients $\nabla\mathcal{L}_{x_{t,i},y_{t,i}}(\theta)$ are very large compared to θ_t, the iteration will push the parameters to extreme values.

Gradient vanishing and explosion

Why is it a problem for deep learning?

- By chain rule, the gradient tends to multiply along the layers.
- Example: If $g^{(L)}(x) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(x)$ where $f^{(L)} : \mathbb{R} \to \mathbb{R}$, then

$$g^{(L)'}(x) = \prod_{l=1}^{L} f^{(l)'}(g^{(l-1)}(x))$$

- If $f^{(l)'}(g^{(l-1)}(x)) \approx c$, then $g^{(L)'}(x) \approx c^L$.
- **Exponentially small** w.r.t. L if $c < 1$ (gradient vanishing).
- **Exponentially large** w.r.t. L if $c > 1$ (gradient explosion).

Mitigation techniques: how to avoid this?

Gradient clipping

- `torch.nn.utils.clip_grad_norm_(model.parameters(), threshold)`
- **Pros:** Easiest method, just limits the gradient norm to a fixed value.
- **Cons:** Only for gradient explosion, adds an extra hyper-parameter.

Mitigation techniques: how to avoid this?

Gradient clipping

- `torch.nn.utils.clip_grad_norm_(model.parameters(), threshold)`
- **Pros:** Easiest method, just limits the gradient norm to a fixed value.
- **Cons:** Only for gradient explosion, adds an extra hyper-parameter.

Architecture changes

- Gates in RNNs, residuals in CNNs, dropout, batch normalization, ...
- **Pros:** More principled, usually leads to better performance.
- **Cons:** Requires to change the network architecture, application dependent.

Mitigation techniques: how to avoid this?

Gradient clipping

- `torch.nn.utils.clip_grad_norm_(model.parameters(), threshold)`
- **Pros:** Easiest method, just limits the gradient norm to a fixed value.
- **Cons:** Only for gradient explosion, adds an extra hyper-parameter.

Architecture changes

- Gates in RNNs, residuals in CNNs, dropout, batch normalization, ...
- **Pros:** More principled, usually leads to better performance.
- **Cons:** Requires to change the network architecture, application dependent.

Weight initialization

- Automatically implemented, but can have an **large impact** on performance

Weights initialization

Ideal initialization scheme

- The better the model is at initialization, the more changes we have of find good weights.
- We would like to have values that are reasonable, $\forall i \in [\![1, d^{(L)}]\!]$, $|g_\theta(x)_i| \approx 1$.
- We would like to have gradients that are neither too large nor too small

$$\forall i \in [\![1, p]\!], \qquad |\nabla \mathcal{L}_{x,y}(\theta)_i| \approx 1$$

Weights initialization

Ideal initialization scheme

- The better the model is at initialization, the more changes we have of find good weights.
- We would like to have values that are reasonable, $\forall i \in [\![1, d^{(L)}]\!]$, $|g_\theta(x)_i| \approx 1$.
- We would like to have gradients that are neither too large nor too small

$$\forall i \in [\![1, p]\!], \qquad |\nabla \mathcal{L}_{x,y}(\theta)_i| \approx 1$$

Simple solution

- Set $b^{(l)} = 0$ and sample the weights $W_{ij}^{(l)} \sim \mathcal{P}$ i.i.d. with expectation 0 and variance $V^{(l)}$.
- Choose $V^{(l)}$ so that the variance is constant across layers.

Weights initialization

Ideal initialization scheme

- The better the model is at initialization, the more changes we have of find good weights.
- We would like to have values that are reasonable, $\forall i \in [\![1, d^{(L)}]\!]$, $|g_\theta(x)_i| \approx 1$.
- We would like to have gradients that are neither too large nor too small

$$\forall i \in [\![1, p]\!], \qquad |\nabla \mathcal{L}_{x,y}(\theta)_i| \approx 1$$

Simple solution

- Set $b^{(l)} = 0$ and sample the weights $W_{ij}^{(l)} \sim \mathcal{P}$ i.i.d. with expectation 0 and variance $V^{(l)}$.
- Choose $V^{(l)}$ so that the variance is constant across layers.
- Technical assumptions:
  - The probability distribution is symmetric w.r.t. 0 and $\mathcal{P}(\{0\}) = 0$.
  - The activation function is ReLU $\sigma(x) = \max\{0, x\}$.

Derivation of optimal weights

Preliminary results

- Let $x \in \mathbb{R}^{d^{(0)}}$ a fixed input and, $\forall l \in [\![1, L]\!]$, $X^{(l)} = g_\theta^{(2l-1)}(x)$.
- For any $l \in [\![1, L]\!]$, the variables $(X_i^{(l)})_{i \in [\![1, d^{(2l-1)}]\!]}$ are identically distributed.
- The distribution of $X_i^{(l)}$ is symmetric w.r.t. 0 (and thus $\mathbb{E}(X_j^{(l)}) = 0$).

Derivation of optimal weights

Preliminary results

- Let $x \in \mathbb{R}^{d^{(0)}}$ a fixed input and, $\forall l \in [\![1, L]\!]$, $X^{(l)} = g_\theta^{(2l-1)}(x)$.
- For any $l \in [\![1, L]\!]$, the variables $(X_i^{(l)})_{i \in [\![1, d^{(2l-1)}]\!]}$ are identically distributed.
- The distribution of $X_i^{(l)}$ is symmetric w.r.t. 0 (and thus $\mathbb{E}(X_j^{(l)}) = 0$).

Proof.

- The proof follows a simple recurrence:

Derivation of optimal weights

Preliminary results

- Let $x \in \mathbb{R}^{d^{(0)}}$ a fixed input and, $\forall l \in [\![1, L]\!]$, $X^{(l)} = g_\theta^{(2l-1)}(x)$.
- For any $l \in [\![1, L]\!]$, the variables $(X_i^{(l)})_{i \in [\![1, d^{(2l-1)}]\!]}$ are identically distributed.
- The distribution of $X_i^{(l)}$ is symmetric w.r.t. 0 (and thus $\mathbb{E}(X_j^{(l)}) = 0$).

Proof.

- The proof follows a simple recurrence:
- Initialization: $X_i^{(1)} = \sum_j W_{ij}^{(1)} x_j$ is identically distributed and symmetric.

Derivation of optimal weights

Preliminary results

- Let $x \in \mathbb{R}^{d^{(0)}}$ a fixed input and, $\forall l \in [\![1, L]\!]$, $X^{(l)} = g_\theta^{(2l-1)}(x)$.
- For any $l \in [\![1, L]\!]$, the variables $(X_i^{(l)})_{i \in [\![1, d^{(2l-1)}]\!]}$ are identically distributed.
- The distribution of $X_i^{(l)}$ is symmetric w.r.t. 0 (and thus $\mathbb{E}(X_j^{(l)}) = 0$).

Proof.

- The proof follows a simple recurrence:
- Initialization: $X_i^{(1)} = \sum_j W_{ij}^{(1)} x_j$ is identically distributed and symmetric.
- If the properties are verified for l, then $X_i^{(l)} = \sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})$, which is identically distributed and symmetric.

□

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$\mathrm{var}(X_i^{(l)}) \quad = \quad \mathrm{var}(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)}))$$

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$\begin{aligned} \operatorname{var}(X_i^{(l)}) &= \operatorname{var}\left(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})\right) \\ &= \textstyle\sum_j \operatorname{var}(W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \end{aligned}$$

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$\begin{aligned}
\operatorname{var}(X_i^{(l)}) &= \operatorname{var}(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= \textstyle\sum_j \operatorname{var}(W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= d^{(l-1)} \operatorname{var}(W_{ij}^{(l)}) \, \mathbb{E}(\sigma(X_j^{(l-1)})^2)
\end{aligned}$$

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$\begin{aligned}
\mathrm{var}(X_i^{(l)}) &= \mathrm{var}(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= \textstyle\sum_j \mathrm{var}(W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= d^{(l-1)}\ \mathrm{var}(W_{ij}^{(l)})\, \mathbb{E}(\sigma(X_j^{(l-1)})^2) \\
&= d^{(l-1)}\, V^{(l)}\, \mathbb{E}({X_j^{(l-1)}}^2 \mathbb{1}\{X_j^{(l-1)} > 0\})
\end{aligned}$$

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$
\begin{aligned}
\mathrm{var}(X_i^{(l)}) &= \mathrm{var}(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= \textstyle\sum_j \mathrm{var}(W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= d^{(l-1)} \ \mathrm{var}(W_{ij}^{(l)})\, \mathbb{E}(\sigma(X_j^{(l-1)})^2) \\
&= d^{(l-1)}\, V^{(l)}\, \mathbb{E}({X_j^{(l-1)}}^2 \mathbb{1}\{X_j^{(l-1)} > 0\}) \\
&= d^{(l-1)}\, V^{(l)} \ \mathrm{var}(X_j^{(l-1)})/2
\end{aligned}
$$

Derivation of optimal weight variance

Variance of the intermediate outputs

- For any $l \in [\![2, L]\!]$ and $i \in [\![1, d^{(l)}]\!]$, we have

$$\begin{aligned}
\mathrm{var}(X_i^{(l)}) &= \mathrm{var}(\textstyle\sum_j W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= \textstyle\sum_j \mathrm{var}(W_{ij}^{(l)} \sigma(X_j^{(l-1)})) \\
&= d^{(l-1)} \, \mathrm{var}(W_{ij}^{(l)}) \, \mathbb{E}(\sigma(X_j^{(l-1)})^2) \\
&= d^{(l-1)} V^{(l)} \, \mathbb{E}({X_j^{(l-1)}}^2 \mathbb{1}\{X_j^{(l-1)} > 0\}) \\
&= d^{(l-1)} V^{(l)} \, \mathrm{var}(X_j^{(l-1)})/2
\end{aligned}$$

- Hence, the **variance is constant across layers** if $V^{(l)} = 2/d^{(l-1)}$, and

$$\mathrm{var}(g_\theta(x)_i) = 2\|x\|_2^2/d^{(0)}$$

Kaiming initialization (Kaiming He et.al., 2015)

Gaussian weights

Our assumptions are satisfied if we use Gaussian weights $W_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{d^{(l-1)}}\right)$.

Uniform weights

If we take uniform weights $W_{ij}^{(l)} \sim \mathcal{U}([-r^{(l)}, r^{(l)}])$, then $V^{(l)} = r^2/3$ and

$$r^{(l)} = \sqrt{\frac{6}{d^{(l-1)}}}$$

Variance of the gradient

Variance propagation during backprop

- Same analysis for backprop, but in **reverse**.
- This gives an optimal variance $V^{(l)} = 2/d^{(l)}$.

Variance of the gradient

Variance propagation during backprop

- Same analysis for backprop, but in **reverse**.
- This gives an optimal variance $V^{(l)} = 2/d^{(l)}$.
- In order to have both the variances of gradients and of values constant, we thus need $V^{(l)} = 2/d^{(l)}$ and $V^{(l)} = 2/d^{(l-1)}$...

Variance of the gradient

Variance propagation during backprop

- Same analysis for backprop, but in **reverse**.
- This gives an optimal variance $V^{(l)} = 2/d^{(l)}$.
- In order to have both the variances of gradients and of values constant, we thus need $V^{(l)} = 2/d^{(l)}$ and $V^{(l)} = 2/d^{(l-1)}$...
- A reasonable heuristic consists in taking the average: $V^{(l)} = \frac{4}{d^{(l)}+d^{(l-1)}}$.

Variance of the gradient

Variance propagation during backprop

- Same analysis for backprop, but in **reverse**.
- This gives an optimal variance $V^{(l)} = 2/d^{(l)}$.
- In order to have both the variances of gradients and of values constant, we thus need $V^{(l)} = 2/d^{(l)}$ and $V^{(l)} = 2/d^{(l-1)}$...
- A reasonable heuristic consists in taking the average: $V^{(l)} = \frac{4}{d^{(l)}+d^{(l-1)}}$.

Xavier initialization (Xavier Glorot & Yoshua Bengio, 2010)

Let $c > 0$ be a hyper-parameter. The weights are initialized using the heuristic

$$W_{ij}^{(l)} \sim \mathcal{U}([-r^{(l)}, r^{(l)}]) \qquad \text{and} \qquad r^{(l)} = \sqrt{\frac{6c^2}{d^{(l)} + d^{(l-1)}}}$$

Impact of initialization in practice: `https://www.deeplearning.ai/ai-notes/initialization/index.html`

Batch normalization

Idea

- Normalize the input of each layer by **removing mean and dividing by std**.
- Also uses a **learnable affine map**.

Batch normalization

Idea

- Normalize the input of each layer by **removing mean and dividing by std**.
- Also uses a **learnable affine map**.

Definition

- If $(x_i)_i$ is a batch of b inputs (to the layer), then the output is:

$$y_i = \frac{x_i - E}{\sqrt{V + \varepsilon}} \cdot \gamma + \beta$$

where $E = \frac{1}{b}\sum_i x_i$ and $V = \frac{1}{b}\sum_i (x_i - E)^2$ (coord.-wise), γ and β are learnable vectors.

Batch normalization

The output depends on the whole batch, not just single inputs!

Train and eval

- The behavior of batch norm is different between training and evaluation (e.g. `model.train()` and `model.eval()` in Pytorch).
- At evaluation, the model uses a (moving) average of **all training batches**.
- Stores E and V for each training batch, and then computes

$$(1-\rho)\sum_t \rho^t E_t \quad \text{and} \quad (1-\rho)\sum_t \rho^t V_t$$

where (typically) $\rho = 0.9$.

Recap

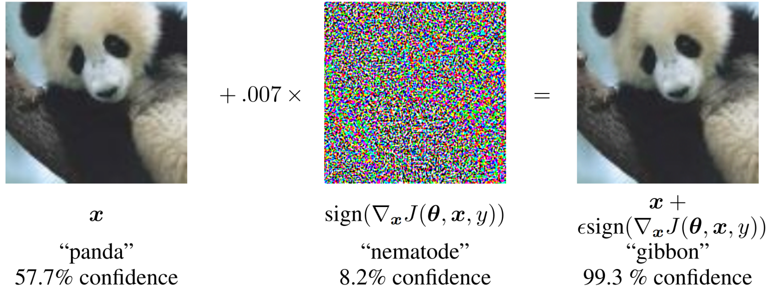
- Gradient **vanishing** and **explosion** can happen during training of **deep** NNs.
- **Gradient clipping**, **batch normalization**, **regularisation** and proper **weight initialization** can help stabilize training.
- The variance of the weights at initialization should be **inversely proportional to the layer width**.

Robustness and adversarial attacks

Confusing a neural network with noise

Adversarial attacks

- Can a small (invisible) noise change the prediction of a vision model?
- Vision models are robust to random input noise.
- Vision models are **extremely fragile** to **well-crafted** input noise.



source: Explaining and Harnessing Adversarial Examples, Goodfellow et al, ICLR 2015.

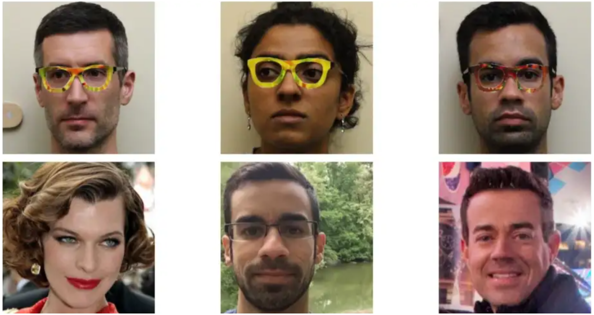
Adversarial attacks

- Can a small (invisible) noise change the prediction of a vision model?
- Vision models are robust to random input noise.
- Vision models are **extremely fragile** to **well-crafted** input noise.

source: Robust Physical-World Attacks on Deep Learning Visual Classification, Eykholt et al, CVPR 2018.

Adversarial attacks

- Can a small (invisible) noise change the prediction of a vision model?
- Vision models are robust to random input noise.
- Vision models are **extremely fragile** to **well-crafted** input noise.



source: Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition, Sharif et.al., CCS 2016.

Adversarial attacks: examples

Fast gradient sign method (Goodfellow et.al., 2014)

- **Idea:** Take one gradient step in the direction that **maximizes the loss**.
- To control the maximum pixel noise, use the coordinates' sign instead of value.
- **Limitations:** Destroys performance, but cannot target a specific class.

$$x^{\mathsf{att}} = x^{\mathsf{true}} + \varepsilon\,\mathsf{sign}(\nabla_x \mathcal{L}(\theta, x^{\mathsf{true}}, y^{\mathsf{true}}))$$

Adversarial attacks: examples

Fast gradient sign method (Goodfellow et.al., 2014)

- **Idea:** Take one gradient step in the direction that **maximizes the loss**.
- To control the maximum pixel noise, use the coordinates' sign instead of value.
- **Limitations:** Destroys performance, but cannot target a specific class.

$$x^{\text{att}} = x^{\text{true}} + \varepsilon \, \mathsf{sign}(\nabla_x \mathcal{L}(\theta, x^{\text{true}}, y^{\text{true}}))$$

Iterative Target Class Method (Kurakin et.al., 2016)

- **Idea:** Perform gradient descent on the loss with **labels swaped**.
- To control the maximum pixel noise, project on a ball of radius ε around x.
- **Limitations:** Requires to know the model weights (white box setting).

$$x^{\text{att}}_{k+1} = \mathsf{Clamp}_{x^{\text{true}},\varepsilon} \left(x^{\text{att}}_k + \varepsilon \, \mathsf{sign}(\nabla_x \mathcal{L}(\theta, x^{\text{att}}_k, y^{\text{att}}))\right)$$

Beyond the white box setting

White-box attacks

- Use the knowledge of the model to create the perturbation.
- Gradient descent on a modified objective (classes swapped).

Beyond the white box setting

White-box attacks

- Use the knowledge of the model to create the perturbation.
- Gradient descent on a modified objective (classes swapped).

Black-box attacks

- Attacks without access the parameters of the model.
- Use a similar model, usually works relatively well.

Beyond the white box setting

White-box attacks

- Use the knowledge of the model to create the perturbation.
- Gradient descent on a modified objective (classes swapped).

Black-box attacks

- Attacks without access the parameters of the model.
- Use a similar model, usually works relatively well.

Defenses

- Augment the dataset with adversarial attacks (brute-force).
- Control the smoothness of the model (see next).

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

Lipschitz continuity

- First order approximation: $g_\theta(x+\varepsilon) - g_\theta(x) = J_{g,x}(x,\theta)\varepsilon + o(\|\varepsilon\|)$.

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

Lipschitz continuity

- First order approximation: $g_\theta(x+\varepsilon) - g_\theta(x) = J_{g,x}(x,\theta)\varepsilon + o(\|\varepsilon\|)$.
- Control on $\|J_{g_\theta}(x)\|_2 = \max_{u\neq 0} \frac{\|J_{g_\theta}(x)u\|_2}{\|u\|_2}$ (operator norm) leads to robustness.

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

Lipschitz continuity

- First order approximation: $g_\theta(x+\varepsilon) - g_\theta(x) = J_{g,x}(x,\theta)\varepsilon + o(\|\varepsilon\|)$.
- Control on $\|J_{g_\theta}(x)\|_2 = \max_{u\neq 0} \frac{\|J_{g_\theta}(x)u\|_2}{\|u\|_2}$ (operator norm) leads to robustness.
- **Lipschitz constant:** $L_{g_\theta} = \sup_x \|J_{g_\theta}(x)\|_2$.

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

Lipschitz continuity

- First order approximation: $g_\theta(x+\varepsilon) - g_\theta(x) = J_{g,x}(x,\theta)\varepsilon + o(\|\varepsilon\|)$.
- Control on $\|J_{g_\theta}(x)\|_2 = \max_{u\neq 0} \frac{\|J_{g_\theta}(x)u\|_2}{\|u\|_2}$ (operator norm) leads to robustness.
- **Lipschitz constant:** $L_{g_\theta} = \sup_x \|J_{g_\theta}(x)\|_2$.
- For piece-wise linear interpolation, Lipschitz constant is **smaller than target function**.

Robustness of neural networks

What makes a model robust?

- **Vital** for practical applications in **engineering** or **medicine**.
- If **black-box**, then trusting the model requires **hard constraints**.
- **Small input perturbation leads to small output perturbation**.

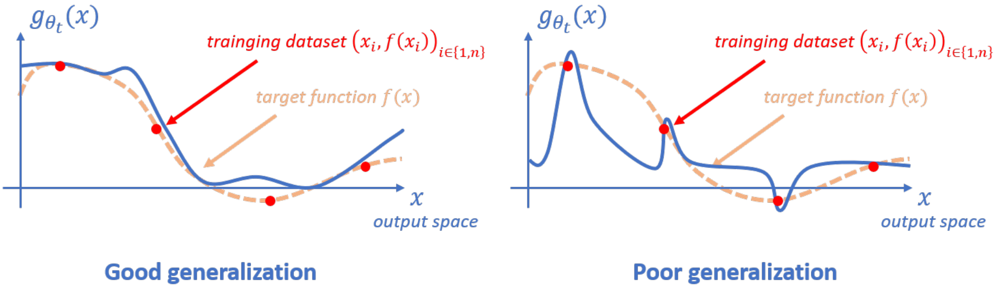
Lipschitz continuity

- First order approximation: $g_\theta(x+\varepsilon) - g_\theta(x) = J_{g,x}(x,\theta)\varepsilon + o(\|\varepsilon\|)$.
- Control on $\|J_{g_\theta}(x)\|_2 = \max_{u\neq 0} \frac{\|J_{g_\theta}(x)u\|_2}{\|u\|_2}$ (operator norm) leads to robustness.
- **Lipschitz constant:** $L_{g_\theta} = \sup_x \|J_{g_\theta}(x)\|_2$.
- For piece-wise linear interpolation, Lipschitz constant is **smaller than target function**.
- For neural networks: $L_{g_\theta} \leqslant \prod_l L_{f^{(l)}} \ldots$ can be exponential in number of layers!

Generalization beyond the training samples

From train accuracy to test accuracy

Beyond the training samples



- **Left model:** More regular, worst on the training set, better on the whole space.
- **Right model:** Less regular, better on the training set, worst on the whole space.
- How does the model behaves when the **test samples are different from the training samples**?

Beyond the training samples

Training objective and risk minimization

- Let $g_\theta : \mathcal{X} \to \mathcal{Y}$ be a model and $\mathcal{D}$ be a distribution of data points in $\mathcal{X} \times \mathcal{Y}$.

$$\min_{\theta \in \mathbb{R}^d} \mathcal{L}_{\mathcal{D}}(\theta) \triangleq \mathbb{E}_{(X,Y)\sim\mathcal{D}}(\ell(g_\theta(X), Y))$$

- During training we minimize $\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta)$ where $\widehat{\mathcal{D}}_n = \frac{1}{n}\sum_i \delta_{(x_i,y_i)}$ is the empirical distribution over the training dataset $(x_i, y_i)_{i\in[\![1,n]\!]}$.

Beyond the training samples

Training objective and risk minimization

- Let $g_\theta : \mathcal{X} \to \mathcal{Y}$ be a model and $\mathcal{D}$ be a distribution of data points in $\mathcal{X} \times \mathcal{Y}$.

$$\min_{\theta \in \mathbb{R}^d} \mathcal{L}_{\mathcal{D}}(\theta) \triangleq \mathbb{E}_{(X,Y)\sim\mathcal{D}}(\ell(g_\theta(X), Y))$$

- During training we minimize $\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta)$ where $\widehat{\mathcal{D}}_n = \frac{1}{n}\sum_i \delta_{(x_i,y_i)}$ is the empirical distribution over the training dataset $(x_i, y_i)_{i\in[\![1,n]\!]}$.

Statistical error

- If $\theta \in \mathbb{R}^d$ is independent of the training samples, then, with probability $1-\delta$,

$$\left|\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta) - \mathcal{L}_{\mathcal{D}}(\theta)\right| \leqslant \|\ell\|_\infty \sqrt{\frac{2\ln(2/\delta)}{n}}$$

- Unfortunately, the SGD iterates $\widehat{\theta}_{n,t}$ depend on the training dataset $\widehat{\mathcal{D}}_n$...

Decomposition of the error

Decomposition of the error

- Let $\widehat{\theta}_{n,t}$ be the parameters after t training steps and $\theta^* \in \mathrm{argmin}_\theta \mathcal{L}_{\mathcal{D}}(\theta)$. Then,

$$\mathcal{L}_{\mathcal{D}}(\widehat{\theta}_{n,t}) = \underbrace{\mathcal{L}_{\mathcal{D}}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t})}_{\text{Generalization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*)}_{\text{Optimization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*) - \mathcal{L}_{\mathcal{D}}(\theta^*)}_{\text{Statistical error}} + \underbrace{\mathcal{L}_{\mathcal{D}}(\theta^*)}_{\text{Approx.}}$$

Decomposition of the error

Decomposition of the error

- Let $\widehat{\theta}_{n,t}$ be the parameters after t training steps and $\theta^* \in \operatorname{argmin}_\theta \mathcal{L}_\mathcal{D}(\theta)$. Then,

$$\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) = \underbrace{\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t})}_{\text{Generalization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*)}_{\text{Optimization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*) - \mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Statistical error}} + \underbrace{\mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Approx.}}$$

- **Approximation error:** by the universality of MLPs, is arbitrarily small. $d \searrow$

Decomposition of the error

Decomposition of the error

- Let $\widehat{\theta}_{n,t}$ be the parameters after t training steps and $\theta^* \in \operatorname{argmin}_\theta \mathcal{L}_\mathcal{D}(\theta)$. Then,

$$\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) = \underbrace{\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t})}_{\text{Generalization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*)}_{\text{Optimization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*) - \mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Statistical error}} + \underbrace{\mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Approx.}}$$

- **Approximation error:** by the universality of MLPs, is arbitrarily small. $d \searrow$
- **Optimization error:** Convergence for SGD if function is sufficiently regular. $t \searrow$

Decomposition of the error

Decomposition of the error

- Let $\widehat{\theta}_{n,t}$ be the parameters after t training steps and $\theta^* \in \mathrm{argmin}_\theta \mathcal{L}_\mathcal{D}(\theta)$. Then,

$$\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) = \underbrace{\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t})}_{\text{Generalization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*)}_{\text{Optimization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*) - \mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Statistical error}} + \underbrace{\mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Approx.}}$$

- **Approximation error:** by the universality of MLPs, is arbitrarily small. $d \searrow$
- **Optimization error:** Convergence for SGD if function is sufficiently regular. $t \searrow$
- **Statistical error:** Convergence in $O\left(\frac{1}{\sqrt{n}}\right)$ by Tchebyshev concentration. $n \searrow$

Decomposition of the error

Decomposition of the error

- Let $\widehat{\theta}_{n,t}$ be the parameters after t training steps and $\theta^* \in \operatorname{argmin}_\theta \mathcal{L}_\mathcal{D}(\theta)$. Then,

$$\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) = \underbrace{\mathcal{L}_\mathcal{D}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t})}_{\text{Generalization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\widehat{\theta}_{n,t}) - \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*)}_{\text{Optimization error}} + \underbrace{\mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta^*) - \mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Statistical error}} + \underbrace{\mathcal{L}_\mathcal{D}(\theta^*)}_{\text{Approx.}}$$

- **Approximation error:** by the universality of MLPs, is arbitrarily small. $d \searrow$
- **Optimization error:** Convergence for SGD if function is sufficiently regular. $t \searrow$
- **Statistical error:** Convergence in $O\left(\frac{1}{\sqrt{n}}\right)$ by Tchebyshev concentration. $n \searrow$
- **Generalization error:** Difficult part. Depends on the model and opt. $d \nearrow, t \nearrow, n \searrow$

Overfitting in ML

Usual analysis

- Optimization error decreases
- Generalization error increases
- There is a trade-off

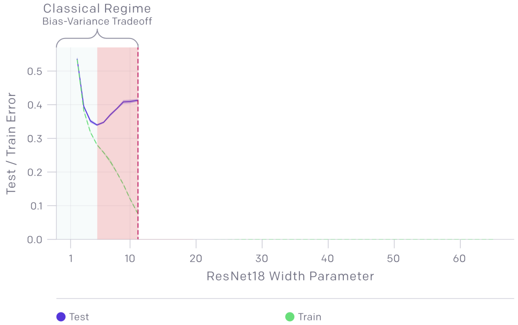
Usual mitigation strategies

- Early stopping
- Hyper-parameter selection via cross-validation
- Regularization: $\min_\theta \mathcal{L}_{\widehat{\mathcal{D}}_n}(\theta) + g(\theta)$ (usually $g(\theta) = \gamma\|\theta\|_2^2$).

But...Double descent!

Overfitting mitigated by over-parameterization

- After a certain model size, test error starts decreasing again.
- Over-parameterizing tends to create **implicit regularization**.

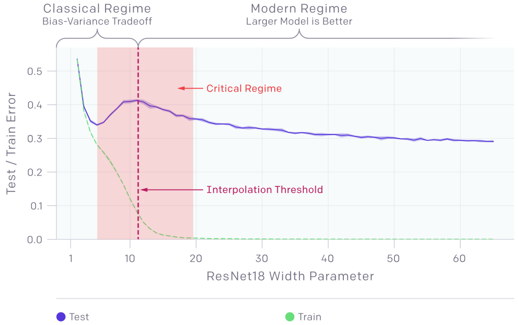


source: `https://openai.com/blog/deep-double-descent/`

But...Double descent!

Overfitting mitigated by over-parameterization

- After a certain model size, test error starts decreasing again.
- Over-parameterizing tends to create **implicit regularization**.

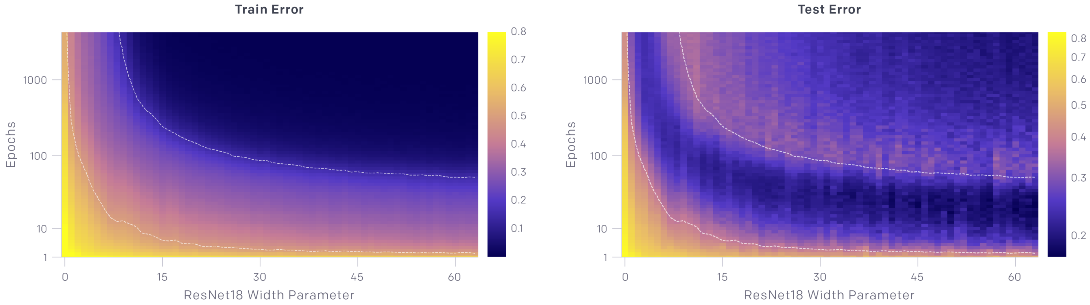


source: `https://openai.com/blog/deep-double-descent/`

But...Double descent!

Overfitting mitigated by over-parameterization

- After a certain model size, test error starts decreasing again.
- Over-parameterizing tends to create **implicit regularization**.

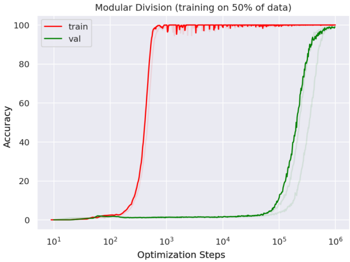


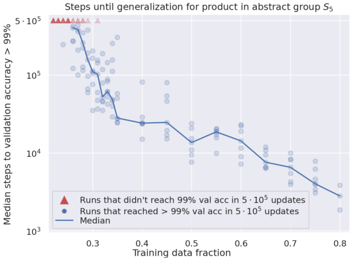
source: `https://openai.com/blog/deep-double-descent/`

But...Grokking!?

Generalization beyond overfitting

- All hope is lost... until you forget to turn your computer off during the holidays.
- Very (very) large plateaux during training.
- Still not a satisfactory explanation (don't do this at home. ;-)).





★	a	b	c	d	e
a	a	d	?	c	d
b	c	d	d	a	c
c	?	e	d	b	d
d	a	?	?	b	c
e	b	b	c	?	a

source: Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, Power et.al., 2022.

Recap

- **Overfitting** to the training dataset can be an issue when the number of parameters is larger than the number of samples.
- In practice, **overparameterization can help generalization** (often called implicit regularization).
- The training curves can exhibit a **double descent behavior**.
- Long plateaux can appear on the test loss/accuracy.