CIS 522: Lecture 4

Optimization

Lyle Ungar – with some slides from NMA : Ioannis Mitliagkas

Penn Engineering

Today

(1) Admin

(2) Representation, Loss Functions, Optimization

(3) Stochastic Gradient Descent for DL

(4) Double Descent and over-parameterization

Admin

No final exam

Final project: info coming soon

Ed: for communication

Deep Learning is Machine Learning:

Representation

Loss Function

Optimization

All models have an “inductive bias”

Linear network?

CNN?

LSTM (vs. Transformer)?

Other biases to build in?

Inductive bias examples

Locality of correlations
Translation invariance
Left-right symmetry
Top-bottom symmetry
Smoothness
Conservation of mass

Representations

Why are ReLUs popular?

Is deep better than shallow? Why?

Loss Functions

Depends on the problem (duh!)

Loss functions

- L_2=MSE
  - or L_1 = MAE

$$\mathcal{L}_N(X,Y) = \frac{1}{s}\sum_{i=1}^{s} ||y_i - N(x_i)||_2^2$$

$$\mathcal{L}_N(X,Y) = \frac{1}{s}\sum_{i=1}^{s} ||y_i - N(x_i)||_1$$

- Cosine Similarity

$$\mathcal{L}_N(X,Y) = \frac{1}{s}\sum_{i=1}^{s} \frac{y_i \cdot N(x_i)}{||y_i||_2||N(x_i)||_2}$$

- Cross entropy

$$\mathcal{L}_N(X,Y) = \frac{1}{s}\sum_{i=1}^{s} \left(-\sum_{j} y_{ij} \log N(x_i)_j\right)$$

Cross-entropy

$$\mathcal{L}_N(X,Y) = \frac{1}{s}\sum_{i=1}^{s}\left(-\sum_j y_{ij}\log N(x_i)_j\right)$$

- Popular for classification tasks.
- "Distance" between two probability distributions.
- target vector is one-hot encoded:
- which means that y_{ij} is 1 where x_i belongs to class *j*, and is otherwise 0.
- Needs probabilities (softmax).
- In PyTorch, softmax and cross-entropy can be applied in a single step.
- **Be careful** in that case not to apply it twice :)

Custom loss functions

- Different cost of false positives and negatives
- Augment standard loss with “soft constraints”
- Model should be similar to an existing model
- Regularization
- E.g., output probability close to 0.5
- ”Fair” outcomes adjust loss on different “protected classes”

Style transfer

Source: Gatys et al. 2016

Content Loss

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

Original

Generated

Are the features on layer l similar?

Style loss

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

Summation is over all channels in each layer.

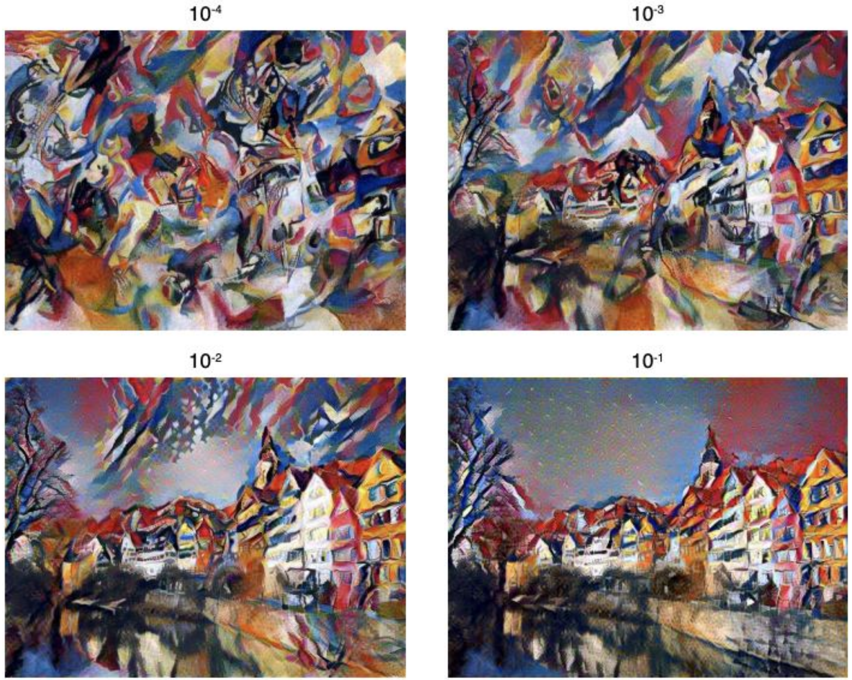
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

Difference between synthesized images with the style reference image.

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Custom Loss Functions: Style Transfer

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$



Source: Gatys et al. 2016

Computer Art?

Fairness

Setting: General group and "protected subclass"

One definition of fairness

(near) equal outcome for same features

- avoid "redlining"

See the book by Michael Kearns and Aaron Roth

Alternative: enforce equality of outcomes

Problem Statement: Given two groups S and T, ensure statistical parity when members of S are less likely to be "qualified".

Example: Given a set of low-income high school students (S) and a set of high-income high school students (T), ensure that proportions of students "accepted" to Penn are equal.

Longer term loss functions

Lifelong learning

Curiosity matters a great deal for RL.

How would you define it?

Optimization

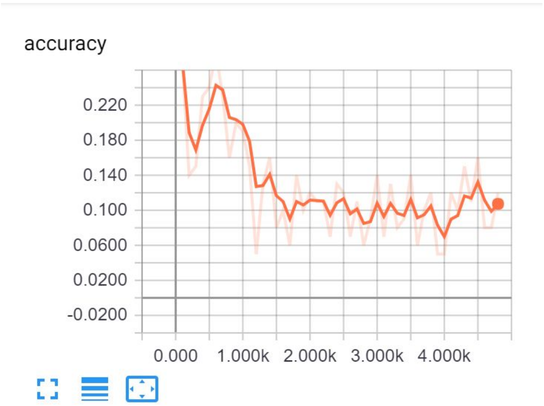
Make sure to optimize the right thing!

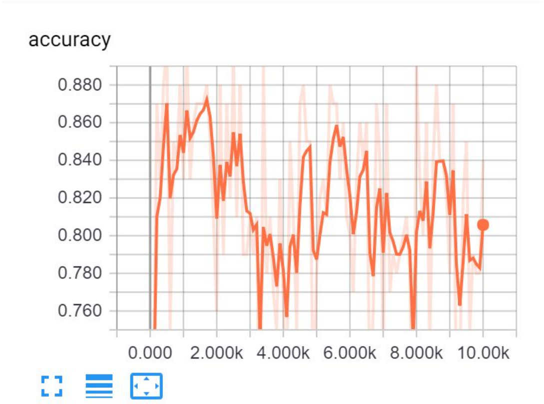
Goal: learn fast, learn good solutions

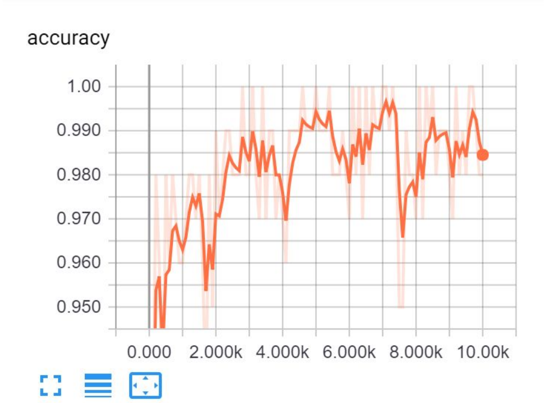
Optimization is not just about finding a minimum

It is about finding a minimum that generalizes well

Learning curves demonstrating problems







The Optimization Worksheets cover

- SGD – noisy, cheap approximation to GD
- Momentum
- Rate scheduling and adaptive learning rates
- Batch size for minibatch gradient descent
- Batch normalization
- Natural gradients
- Fairness

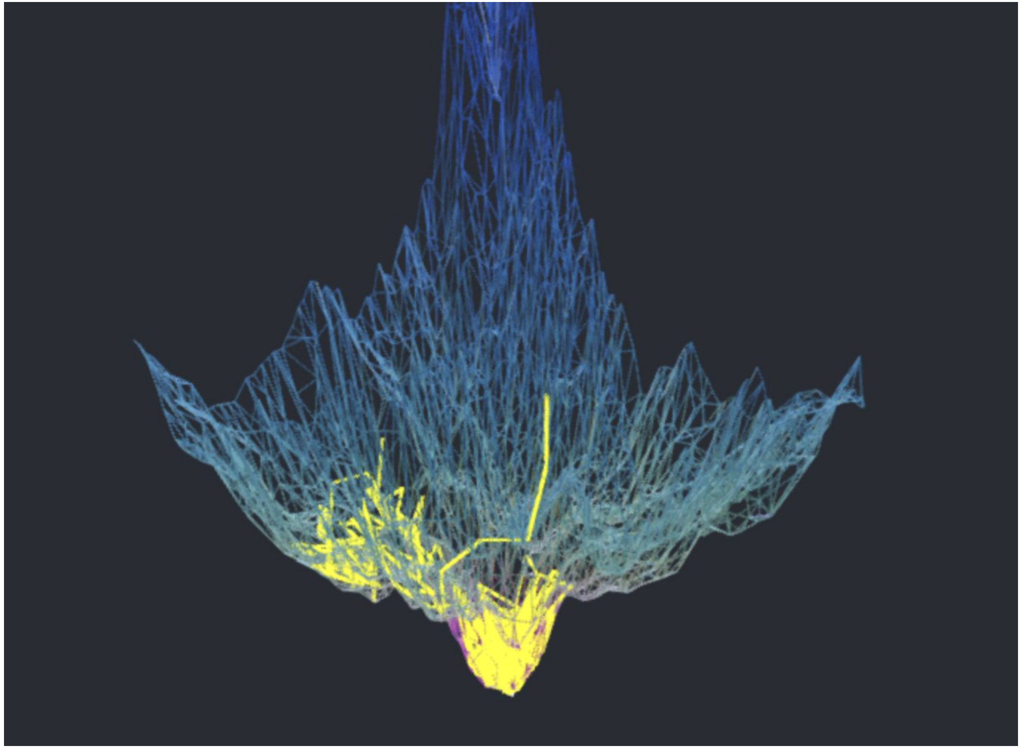
Optimization Background

- The optimization landscape
  - geometric intuition for SGD, momentum. …

Gradient descent

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

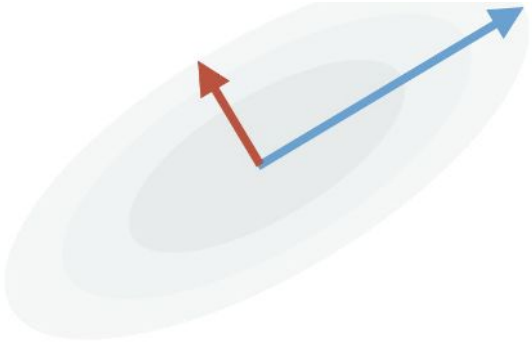
Gradient descent

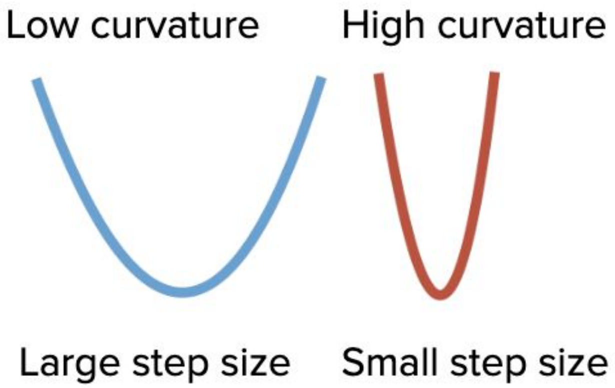


https://losslandscape.com/explorer

Training Challenges: Gradient Magnitude

Conditioning





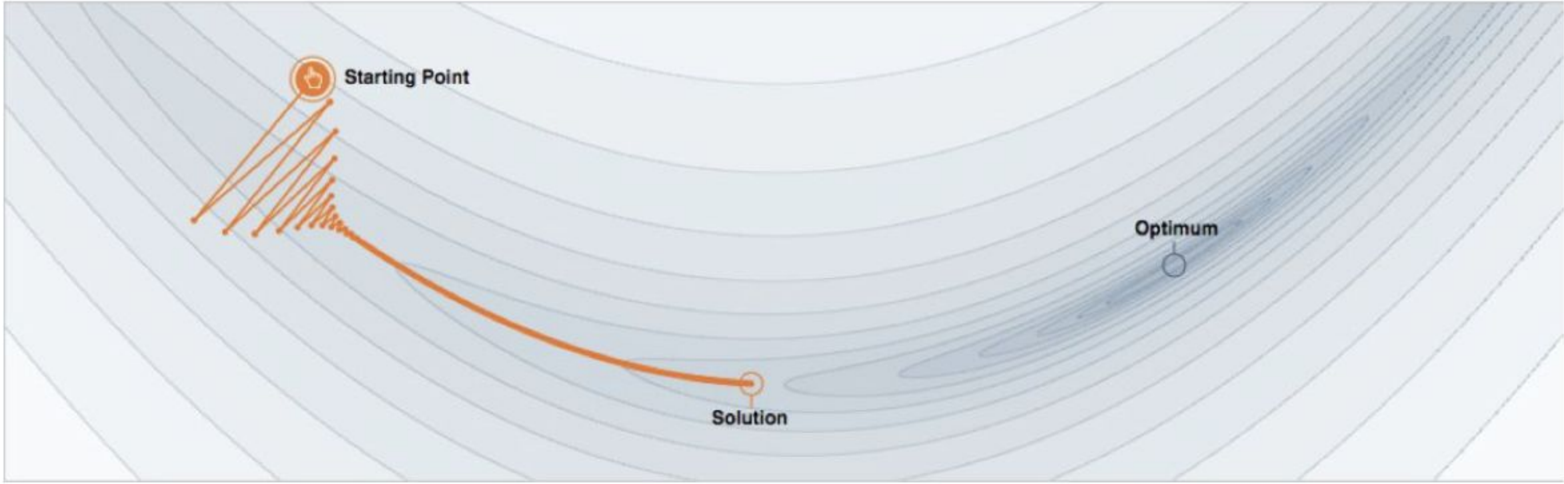
Step size: Need large for **one direction**
Need small for **other direction**

Poor conditioning ⇒ Slow convergence

Poor conditioning and gradient descent

Gradient descent: Moves slowly along **flat directions**

Oscillates along **sharp directions**



Momentum

Do a **gradient descent step**

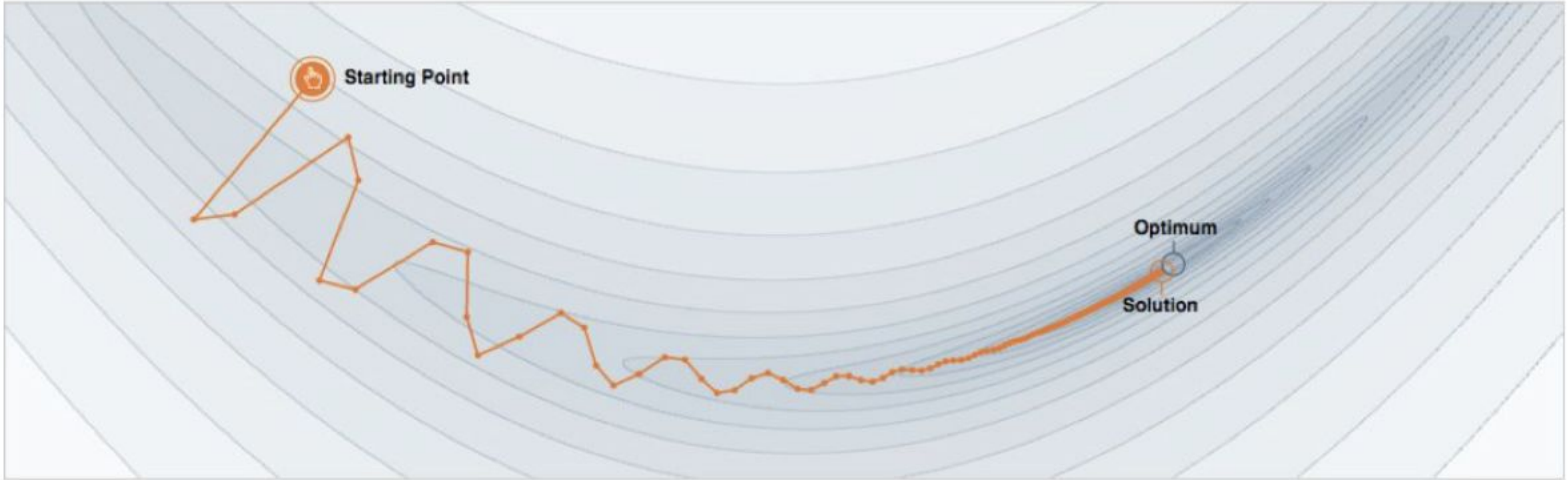
Apply the update from the last iteration, only smaller (**momentum step**)

$$w_{t+1} = w_t - \eta \nabla J(w_t) + \beta(w_t - w_{t-1})$$

Poor conditioning and momentum

Momentum: Accelerates along **flat directions**

Slows down along **sharp directions**



Rate tuning ("annealing")

$$w^{t+1} = w^t + \mu \cdot \nabla_w$$

$$w^{t+1} = w^t + \frac{\mu}{t} \cdot \nabla_w$$

Advanced optimizers: Adagrad

- "Adaptive gradient algorithm"
- Adapts a learning rate for each parameter based on size of previous gradients.

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2.$$

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.$$

Advanced optimizers: Adagrad

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Advanced optimizers: RMSprop

- "Root mean square prop"
- Adapts a learning rate for each parameter based on size of $v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

Advanced optimizers: RMSProp

RMSProp (green) vs AdaGrad (white). The first run just shows the balls; the second run also shows the sum of gradient squared represented by the squares.

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Advanced optimizers: Adam

- "Adaptive moment estimation"
- Similar to RMSprop, but with both the first and second moments of the gradients

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-(\beta_1)^{t+1}}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-(\beta_2)^{t+1}}$$

What to do when your gradients are still enormous

Gradient clipping

clip_grad_norm_

`torch.nn.utils.clip_grad_norm_(parameters, max_norm, norm_type=2)` [SOURCE]

Clips gradient norm of an iterable of parameters.

The norm is computed over all gradients together, as if they were concatenated into a single vector. Gradients are modified in-place.

Parameters:

- **parameters** (*Iterable*[*Tensor*] *or Tensor*) – an iterable of Tensors or a single Tensor that will have gradients normalized
- **max_norm** (*float or int*) – max norm of the gradients
- **norm_type** (*float or int*) – type of the used p-norm. Can be `'inf'` for infinity norm.

Returns: Total norm of the parameters (viewed as a single vector).

clip_grad_value_

`torch.nn.utils.clip_grad_value_(parameters, clip_value)` [SOURCE]

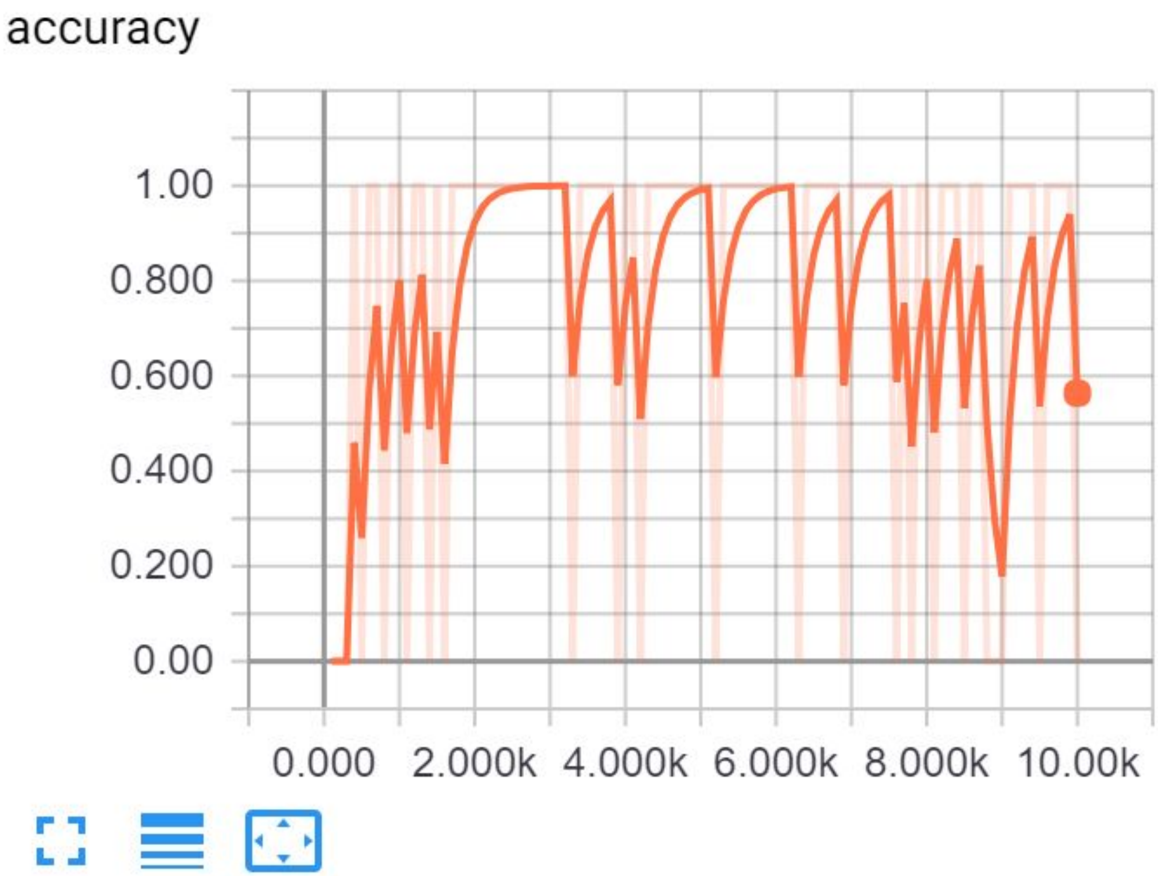
Clips gradient of an iterable of parameters at specified value.

Gradients are modified in-place.

Parameters:

- **parameters** (*Iterable*[*Tensor*] *or Tensor*) – an iterable of Tensors or a single Tensor that will have gradients normalized
- **clip_value** (*float or int*) – maximum allowed value of the gradients The gradients are clipped in the range [-clip_value, clip_value]

Training Challenges: Gradient Direction


accuracy
1.00
0.800
0.600
0.400
0.200
0.00
0.000
2.000k
4.000k
6.000k
8.000k
10.00k

Minibatching

- A minibatch is a small subset of a large dataset.
- For gradient descent, we need an accurate measure of the gradient of the loss with respect to the parameters. The best measure is the average gradient over all of the examples (batch gradient descent).
- Computing over 60K examples on MNIST for a single (extremely accurate) update is stupidly expensive.
- We use minibatches (say, 50 examples) to compute a noisy estimate of the true gradient. The gradient updates are worse, but there are **many** of them. This converts the neural net training into an **online** algorithm.

Minibatching Issues

- Size
- Normalization
  - Remember initialization?

Double Descent and Overparameterization

Modern Neural Nets are Big!

MODEL	Billion Parameters
GPT-3	175
Gopher	280
Megatron-Turing NLG	530
Wu Dao 2.0	1,750

GPT-3 was trained on ½ billion words
Why don't they overfit?

Bias-variance Tradeoff

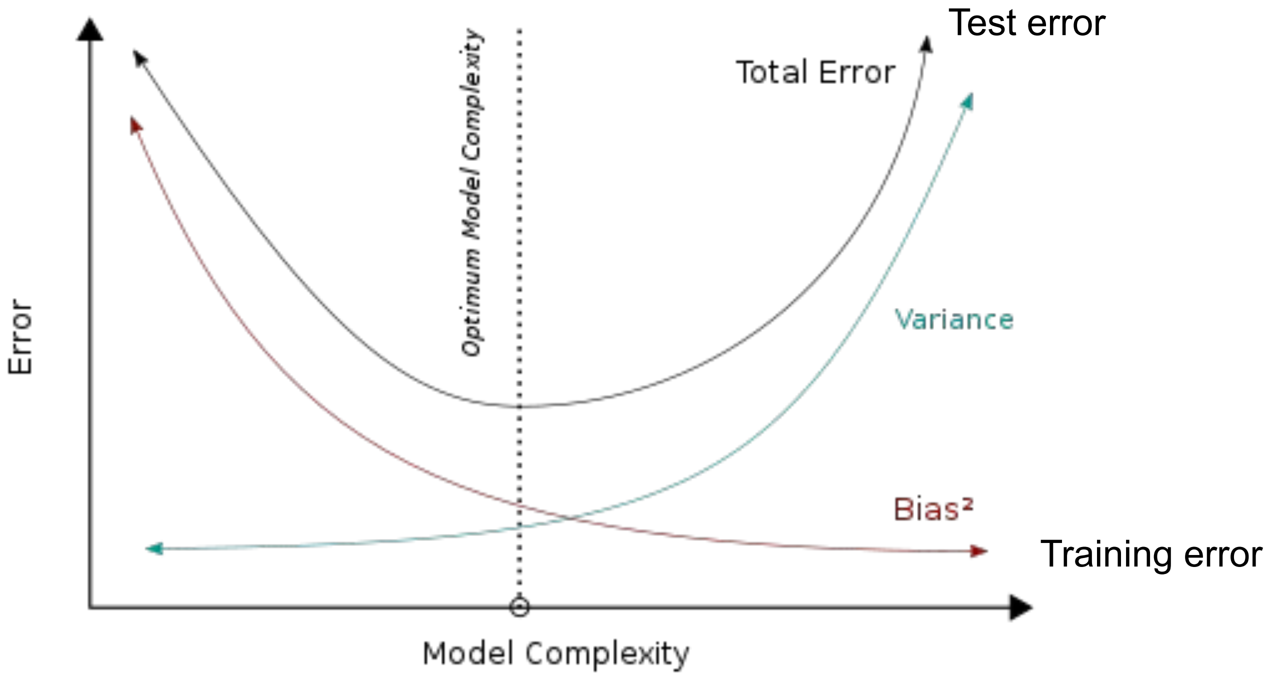
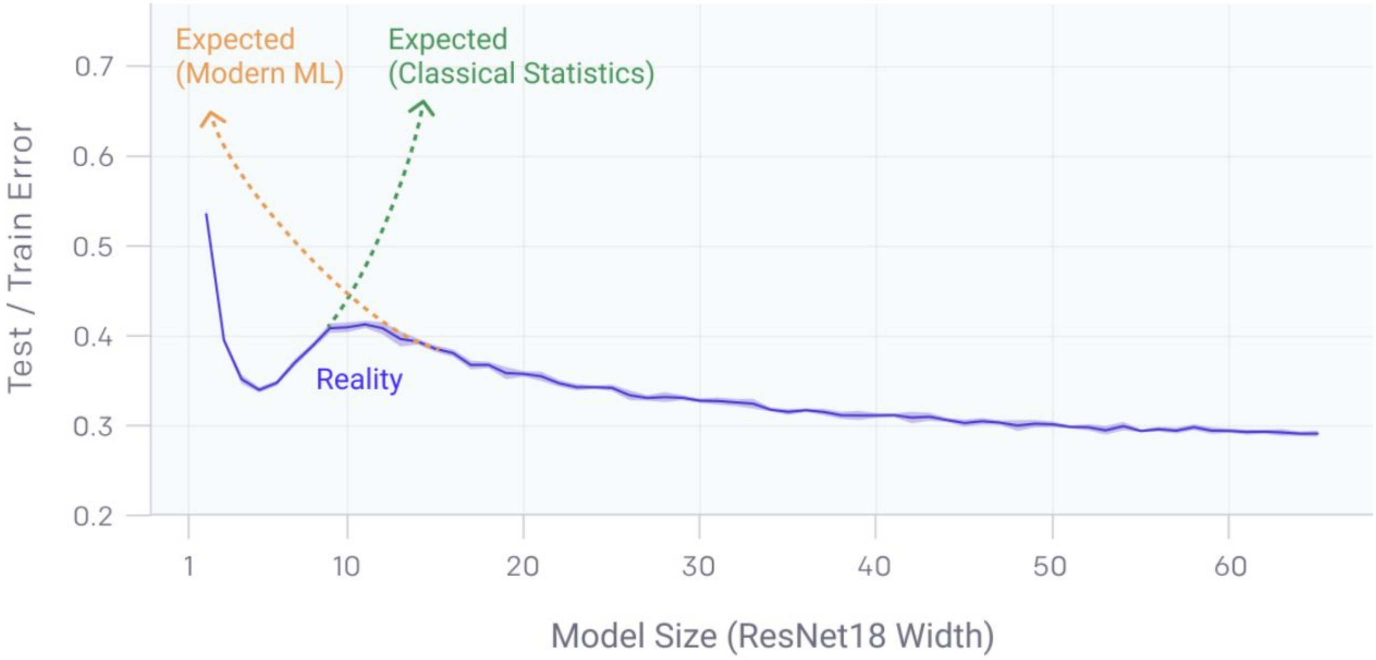


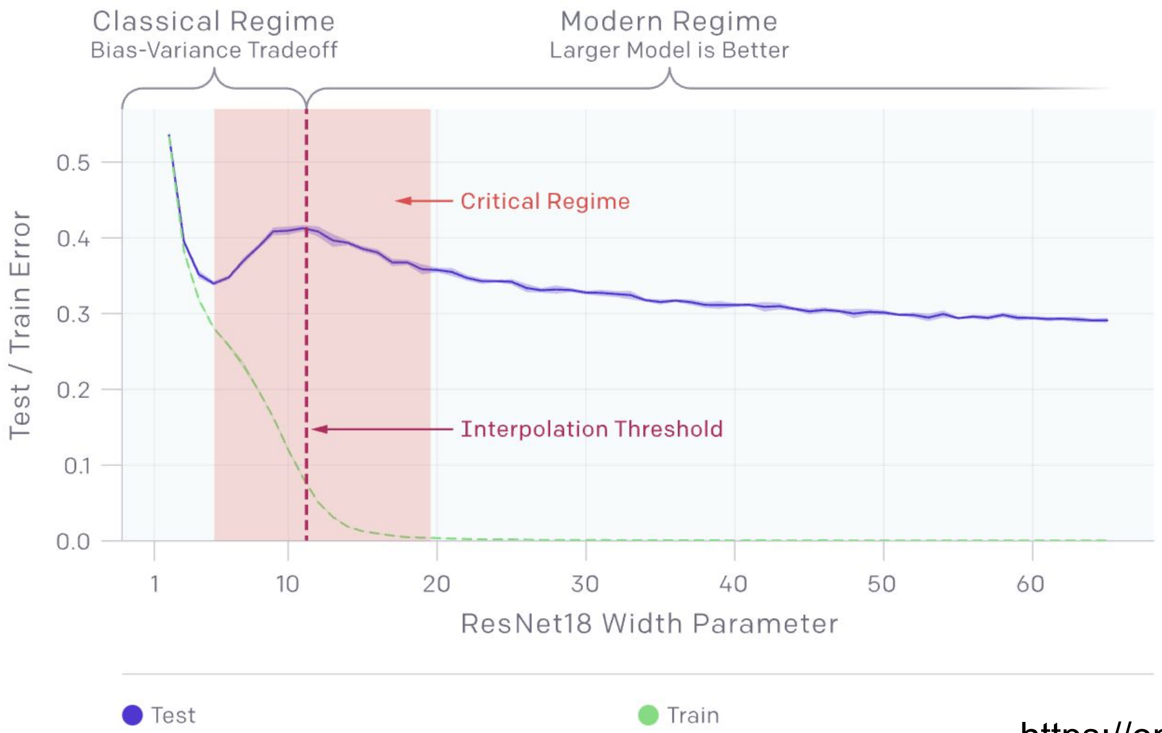
Image credit:
wikipedia

Bias-variance tradeoff fails in DL?



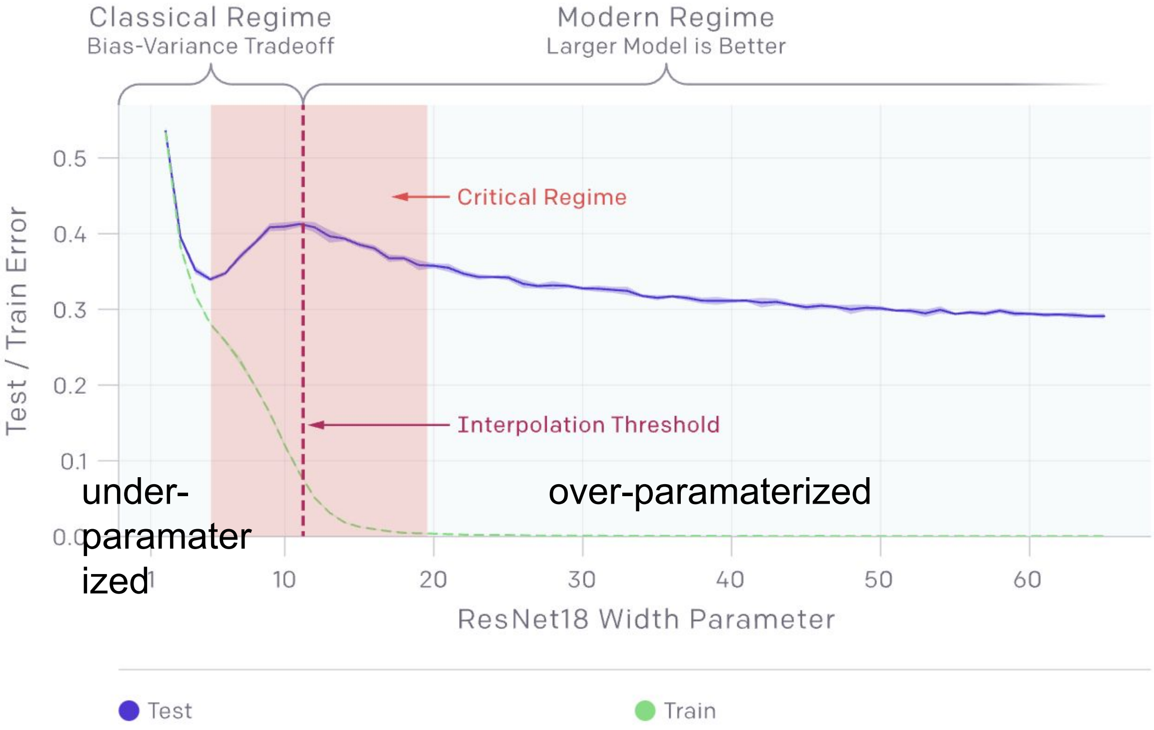
https://openai.com/blog/deep-double-descent/

Deep double descent: where bigger models and more data hurt



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

Deep double descent: where bigger models and more data hurt



Deep double descent: where bigger models and more data hurt

- In under- or over-parameterized regime, increasing complexity reduces test error
- In the middle, it may help or may hurt

Why can an "overfit" model be good?

- It interpolates more smoothly
- "in the over-parameterized regime, there are many models that fit the train set and there exist such good models. Moreover, the implicit bias of stochastic gradient descent (SGD) leads it to such good models, for reasons we don't yet understand."
    - Nakkiran, Kaplun, Bansal, Yang, Boaz, Sutskever

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

See also https://mlu-explain.github.io/double-descent/

What key part of optimization did we not cover?

- Hyperparameter optimization
  - Network architecture
  - Regularization
  - Search method

Have an awesome week!