DAY 2: TOWARDS SCALABLE DEEP LEARNING

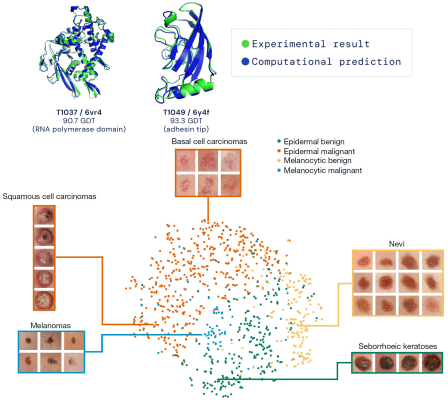
Motivation, Deep Learning Basics Recap

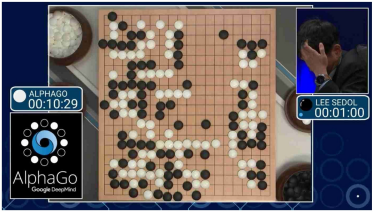
2023-05-09 | Jenia Jitsev | Scalable Learning & Multi-Purpose AI Lab, Helmholtz AI, LAION @ JSC

JÜLICH
Forschungszentrum

DEEP LEARNING: CURRENT STATE OF THE ART IN ML

- Breakthroughs in broad range of challenging domain problems
  - computer vision, language understanding, complex control, protein structure prediction, . . .

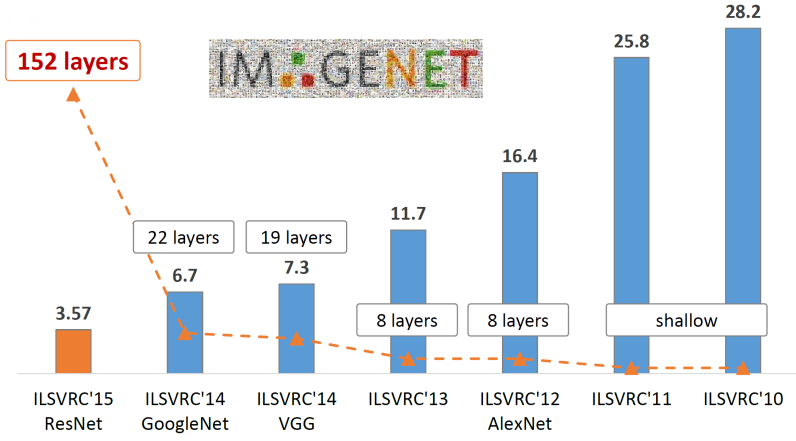




Karras et al., 2019; Esteva et al., 2017; Jumper et al., 2020

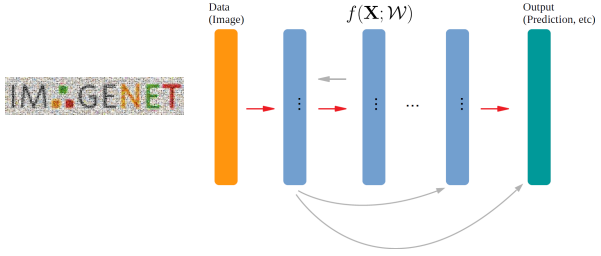
DEEP LEARNING: CURRENT STATE OF THE ART IN ML

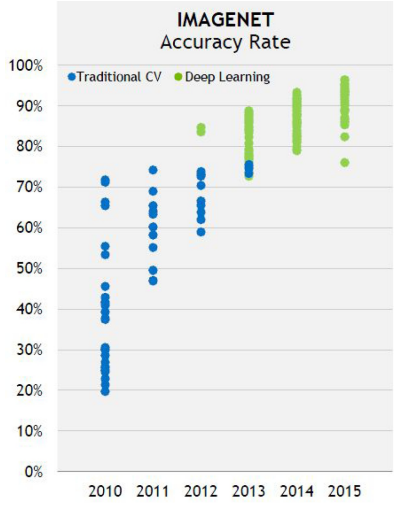
- Consistently outperforming all other ML methods on large dataset benchmarks
  - ImageNet-1k (1.4 M samples, 224x224, 0.4 TB), FFHQ (70k samples, 1024x1024, 2.5 TB), . . .
  - Recent development: very large scale multi-modal (language-vision) datasets: LAION-400M/5B (400M/5B image-text pairs)



DEEP LEARNING: CURRENT STATE OF THE ART IN ML

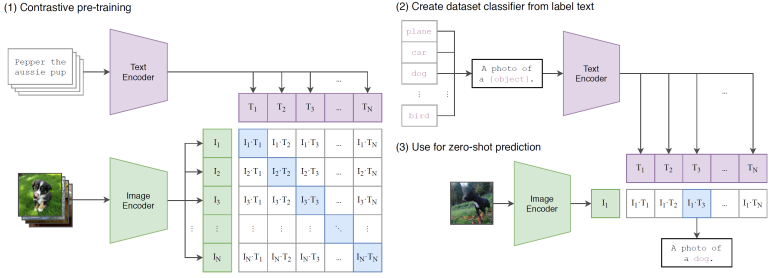
- Consistently outperforming all other ML methods on large dataset benchmarks
  - convolutional and transformer networks predominant

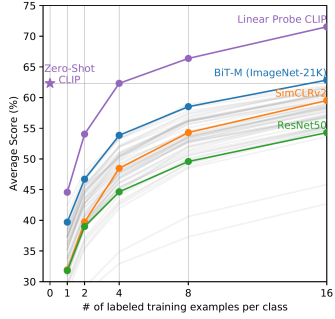




DEEP LEARNING: CURRENT STATE OF THE ART IN ML

- Self-supervised multi-modal (language-vision) learning: no explicit labels required
  - openAI CLIP: very strong zero- and few-shot transfer across various targets
  - requires very large data for pre-training (eg. publicly available LAION-400M/5B)
  - self-supervised learning from weakly aligned image-text pairs: public Internet as scalable data source

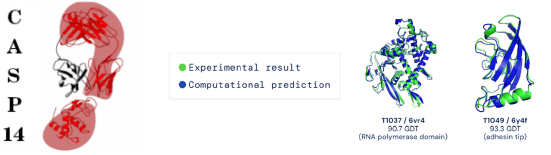


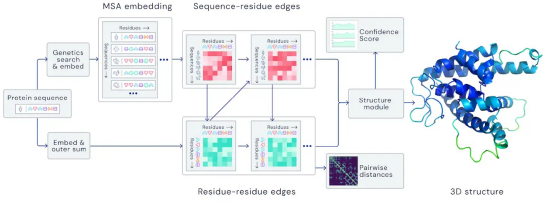


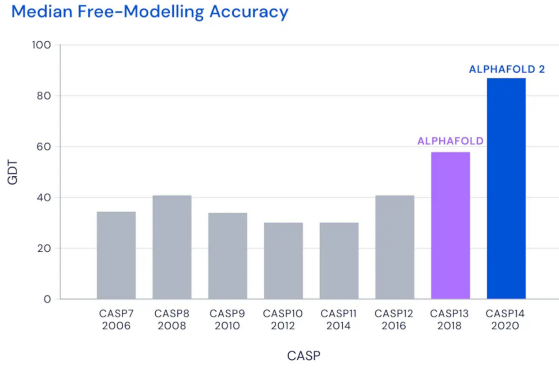
Radford et al., ICML, 2021

DEEP LEARNING: CURRENT STATE OF THE ART IN ML

- Consistently outperforming all other ML methods
  - AlphaFold 2: Transformer networks (predominant in natural language processing)
  - Big Fantastic Database (BFD): 2.4B protein sequences, 0.27 TB



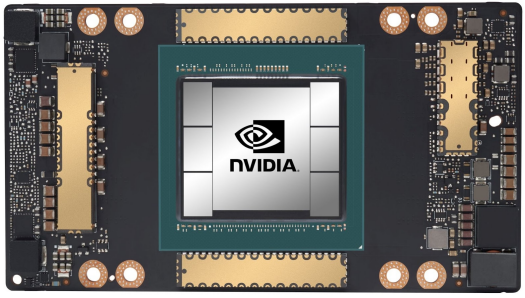


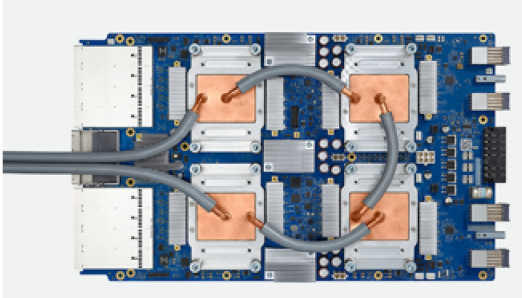


Jumper et al., Nature, 2021

DEEP LEARNING IS SUPERCOMPUTING

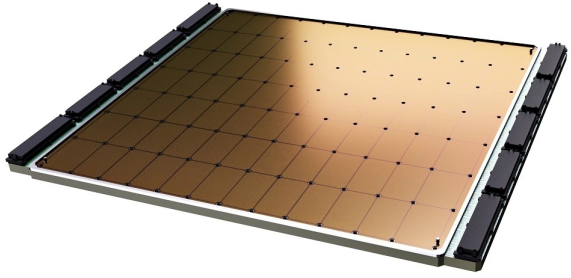
- Training of models requires accelerators
  - GPUs (currently NVIDIA dominant), TPUs (Google)
  - GPUs: generic deep learning hardware
    - parallelizing matrix/tensor operations via vectorization





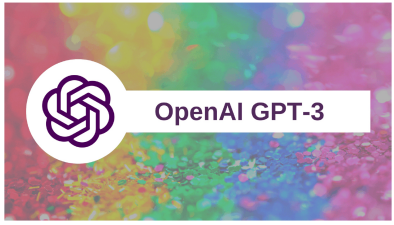
DEEP LEARNING IS SUPERCOMPUTING

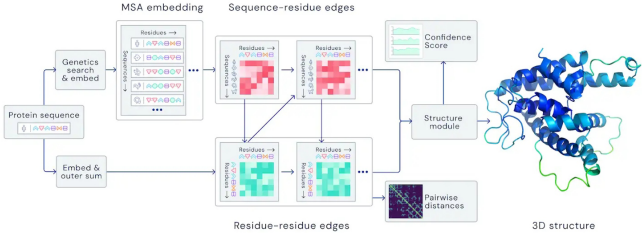
- Training of models requires accelerators
  - spezialized hardware, eg. in-memory computing chips
  - Graphcore IPU: Colossus MK2
  - Cerebras: Wafer Scale Engine 2 (850k cores!)



DEEP LEARNING IS SUPERCOMPUTING

- Most breakthroughs require heavy compute power, using many accelerators simultaneously
  - GPT-3: natural language generation, language understanding
  - CLIP, DALL-E 2, Stable Diffusion: image understanding and image generation
  - AlphaFold 2: protein structure prediction
  - AlphaZero, MuZero: learning control in highly dimensional state-action spaces





DEEP LEARNING IS SUPERCOMPUTING

- Many recent breakthroughs require heavy compute power, using many accelerators simultaneously
  - GPT-3: natural language generation, language understanding
  - CLIP, DALL-E: image understanding and image generation
  - AlphaFold 2: protein structure prediction
  - AlphaZero, MuZero: learning control in highly dimensional state-action spaces

DEEP LEARNING IS SUPERCOMPUTING

- GPT-3: natural language generation, language understanding
  - 1 model: 175 billion weight parameters
  - compare:
    - AlexNet, winner ILSRVC 2012 – 60M
    - ResNet-50, winner ILSRVC 2015 – 25M
    - CLIP ViT L/14, multi-modal learning, 2021 - 600M

DEEP LEARNING IS SUPERCOMPUTING

- GPT-3: natural language generation, language understanding
  - 1 model: $\approx$ 350 GB memory for training
  - splitting over 22 V100 GPUs required for one model to run

DEEP LEARNING IS SUPERCOMPUTING

- GPT-3: natural language generation, language understanding
  - 1 model full training: $\approx 3.14 \cdot 10^{23}$ FLOPS
  - $\approx$ 355 years for V100; $\approx$ 90 years for A100

DEEP LEARNING IS SUPERCOMPUTING

- GPT-3: natural language generation, language understanding
  - 1 model full training: $\approx 3.14 \cdot 10^{23}$ FLOPS
  - $\approx$ 16 days if scaled well with 2 000 A100 GPUs on JUWELS Booster

DEEP LEARNING IS SUPERCOMPUTING

- CLIP: self-supervised image-text learning, strong zero-shot transfer
  - openCLIP ViT g/14, 1480 A100 GPUs
  - 8 days on 34B samples from LAION-2B

  $\rightarrow$ for training a single language-vision model

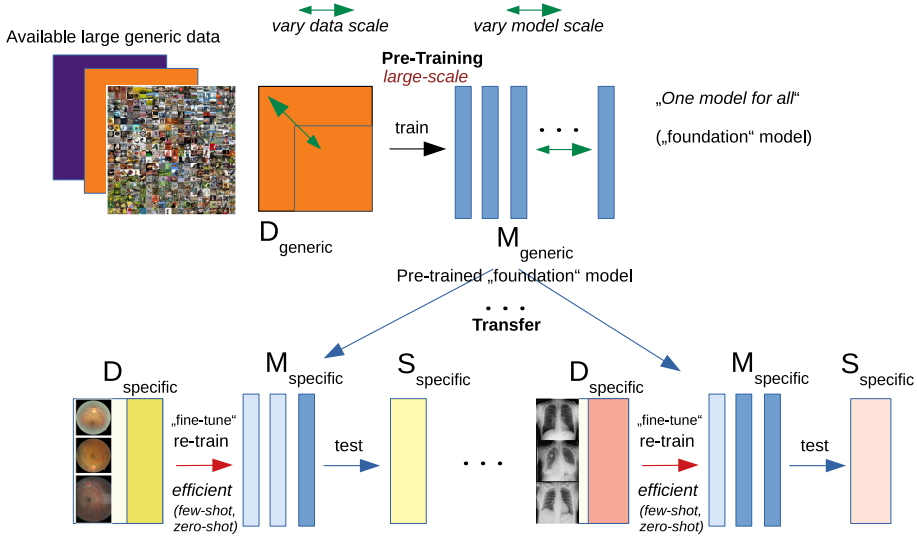
DEEP LEARNING IS SUPERCOMPUTING

- AlphaFold 2: protein structure prediction
  - 128 TPUs
  - few weeks

$\rightarrow$ for training a single model

DEEP LEARNING IS SUPERCOMPUTING

- Foundation models: transferable models pre-trained on large generic data
  - transfer across domains specific smaller datasets and tasks
  - strong, efficient transfer: requires large pre-trained models



DEEP LEARNING IS SUPERCOMPUTING

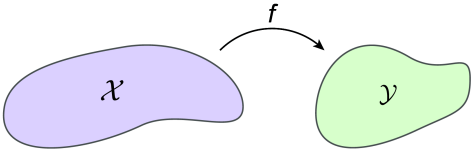
- Investigating strong transferable models: pre-training on a large dataset, requires a large machine
  - a rather compact ResNet-50 on ImageNet-1k: still $\approx$ **20 hours** for full training (single V100)
  - can be done in **a few minutes** if using thousands of GPU
  - High Performance Computing (HPC): many strong compute nodes, fast interconnect (InfiniBand)

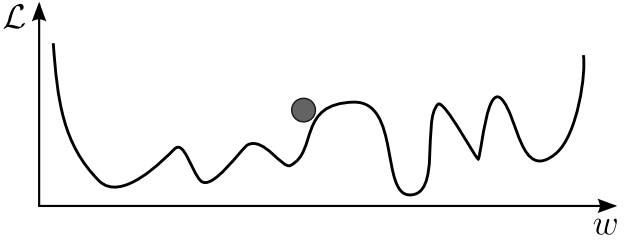
DEEP LEARNING: BASICS RECAP

Machine Learning

Optimizing loss (objective) of a (complex) model f using (a lot of) data $\mathcal{D}$

- a (complex) model: function (or distribution) family $f(X;\boldsymbol{\theta})$ ($p(X;\boldsymbol{\theta})$)
  - parameters $\boldsymbol{\theta}$ (often $\mathbf{W}$ is used) are to adapt ("fit") given the data samples $X \in \hat{\mathcal{D}}$
- optimization:
  - defining a loss $\mathcal{L}(f(X;\boldsymbol{\theta}), \hat{\mathcal{D}})$
  - loss $\mathcal{L}$: measure of quality ("fit") of $f(X;\boldsymbol{\theta})$ in terms of a task solution on $\hat{\mathcal{D}}$
  - seeking to minimize $\mathcal{L}(f, \hat{\mathcal{D}})$ with respect to all possible $\hat{\mathcal{D}} \sim P(\mathcal{D})$!





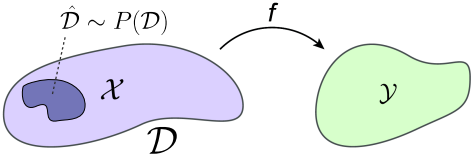
DEEP LEARNING: BASICS RECAP

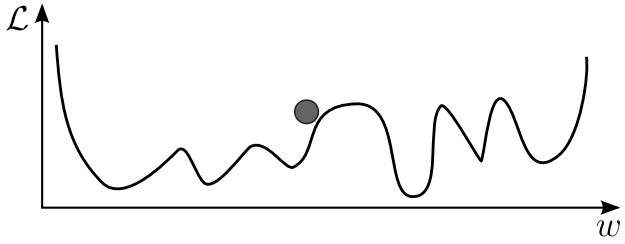
Machine Learning

Optimizing loss (objective) of a (complex) model f using (a lot of) data $\mathcal{D}$

- model $f(X; \mathbf{W})$: unknown recipe for "solutions" to a "problem" posed by $\mathcal{L}$
- optimization: looking for a "good" model f^* by minimizing $\mathcal{L}(f, \hat{\mathcal{D}})$ for all possible $\hat{\mathcal{D}} \sim P(\mathcal{D})$!
  - general principle of expected risk minimization:

$$f^* = \arg\min_f \mathbf{R}(f) = \mathbb{E}_{\hat{\mathcal{D}} \sim P(\mathcal{D})}[\mathcal{L}(f, \hat{\mathcal{D}})] = \int \mathcal{L}(f, \hat{\mathcal{D}})\, P(\hat{\mathcal{D}})\, d\hat{\mathcal{D}}$$





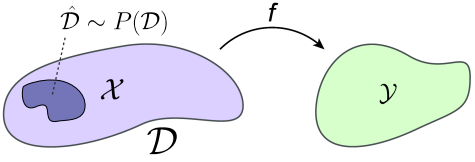
MACHINE LEARNING: GENERALIZATION

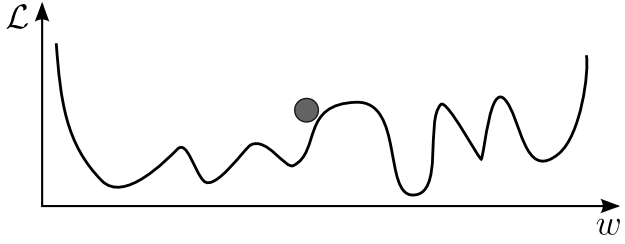
Important

Estimate loss $\mathcal{L}(f(X; \mathbf{W}), \mathcal{D})$ on data $\mathcal{D}$ yet unseen!

- estimating "true" $\mathcal{L}$, expected risk: **generalization** error
  - Challenge: limited data – "true" $\mathcal{L}$, true loss landscape, true data distribution $P(\mathcal{D})$ unknown

$$f^* = \arg\min_f \mathbf{R}(f) = \mathbb{E}_{\hat{\mathcal{D}} \sim P(\mathcal{D})}[\mathcal{L}(f, \hat{\mathcal{D}})] = \int \mathcal{L}(f, \hat{\mathcal{D}})\, P(\hat{\mathcal{D}})\, d\hat{\mathcal{D}}$$



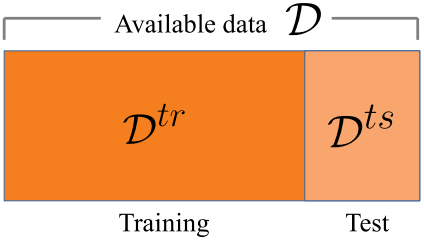


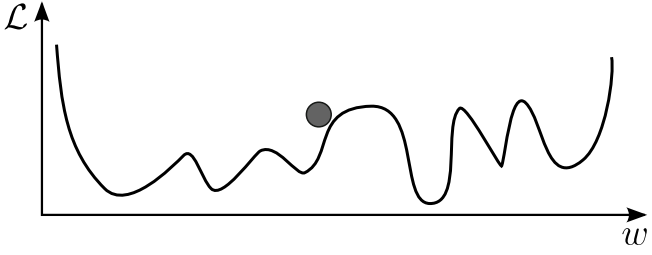
MACHINE LEARNING: GENERALIZATION

Important

- Estimate $\mathcal{L}(f(X; \mathbf{W}), \mathcal{D}^{unseen})$: aiming for good **generalization** capability

- General approach: split $\hat{\mathcal{D}}$ into disjoint $\mathcal{D}^{tr}$ and $\mathcal{D}^{ts}$, $\mathcal{D}^{tr} \cap \mathcal{D}^{ts} = \emptyset$!
  - learn, "train" on $\mathcal{D}^{tr}$: **training data set**
    - Do not show $\mathcal{D}^{ts}$ - **test data set** - during training!
  - estimate $\mathcal{L}$, **generalization error** on $\mathcal{D}^{ts}$ after training





MACHINE LEARNING: GENERALIZATION

Dataset split

- Estimate $\mathcal{L}(f(X; \mathbf{W}), \mathcal{D}^{unseen})$
  - Requires strict separation of data used for **any** parameter adaptation (training) from data for testing

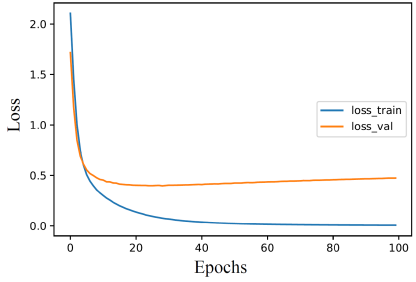
- Often, model has further **hyperparameters Θ** (eg. learning rate, . . .)
  - adapt **W**, then **Θ**: requires distinct sets
  - training $\mathcal{D}^{tr}$, **validation** set $\mathcal{D}^{val}$
- Estimate $\mathcal{L}$, **generalization error** on separate $\mathcal{D}^{ts}$
  - **never** use $\mathcal{D}^{ts}$ for parameter tuning followed by model selection, **only** for reporting metrics (loss, accuracy, . . .)

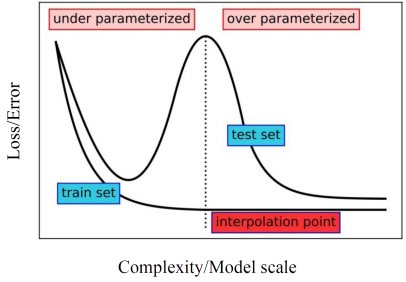
MACHINE LEARNING: GENERALIZATION

Remember

- $\mathcal{L}^{tr}$ measured on $\mathcal{D}^{tr}$ can be highly misleading for the quality of the trained model
- Minimizing $\mathcal{L}^{tr}$ is **not** the ultimate goal

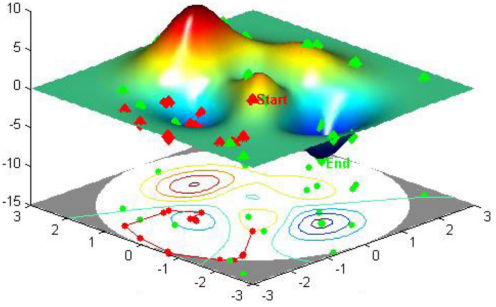
- **Overfitting**: very low $\mathcal{L}^{tr}$ on $\mathcal{D}^{tr}$, much higher $\mathcal{L}^{ts}$ on $\mathcal{D}^{ts}$
- **Generalization gap**
- Ultimate aim: minimize **generalization error**
  - "bad" outcomes on unseen data matter

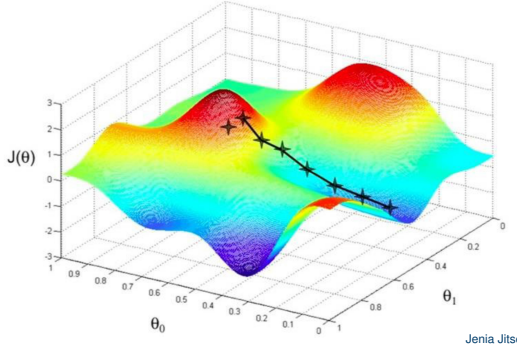




MACHINE LEARNING AS OPTIMIZATION PROBLEM

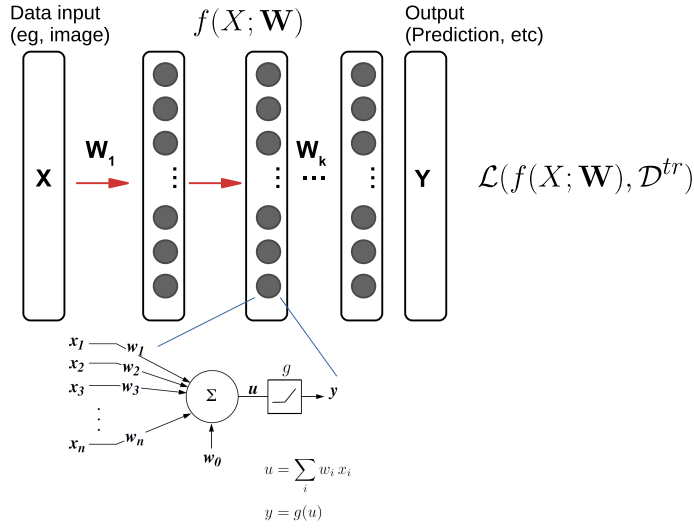
- Optimization procedure: minimizing $\mathcal{L}(X; \mathbf{W})$ by adapting $\mathbf{W}$ from $\mathcal{D}^{tr}$
  - corresponds to **searching for a yet unknown, good model** $f(X; \mathbf{W})$ – data-driven tuning of $f(X; \mathbf{W})$
- Different optimization methods depending on problem setting, model class, loss
  - evolutionary search (ES), genetic algorithms
  - expectation-maximization (EM)
  - coordinate descent
  - gradient-based (first, second order gradient descent): requires **differentiable** $\mathcal{L}$!
  - etc. . .





DEEP LEARNING AS OPTIMIZATION PROBLEM

- Deep Learning: using a generic, **differentiable** flexible model family $f(X; \mathbf{W})$
  - "neural" networks with multiple adaptive "layers", large number of those
  - based on stacked, adaptive, repetitive **differentiable** operations
  - non-linear operations executed by simple, generic units – a great number of those
  - Large number of weights **W** in the layers: adapted from incoming data
- Important: **differentiable** f allows for **differentiable** $\mathcal{L}$
  - using **simple first order gradient descent** methods turned out to be very successful

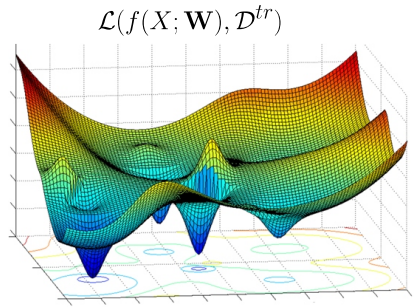


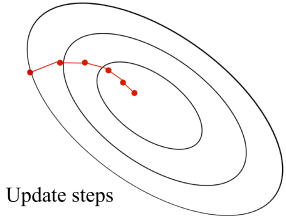
DEEP LEARNING: GRADIENT DESCENT

- First order gradient descent (GD)
  - first derivatives (the gradient) of loss $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ are sufficient

$$\nabla_{\mathbf{W}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \begin{pmatrix} \partial \mathcal{L}/\partial w_1 \\ \vdots \\ \partial \mathcal{L}/\partial w_m \end{pmatrix}$$

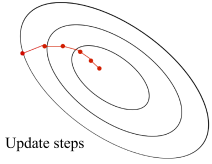
  - use gradients to update the weights: $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}$
    - η: update step size, **learning rate**
  - moving in direction of decreasing $\mathcal{L}$: $\mathcal{L}(f(\mathbf{W}_{t+1})) < \mathcal{L}(f(\mathbf{W}_t))$ (if $\Delta \mathbf{W}$ small)





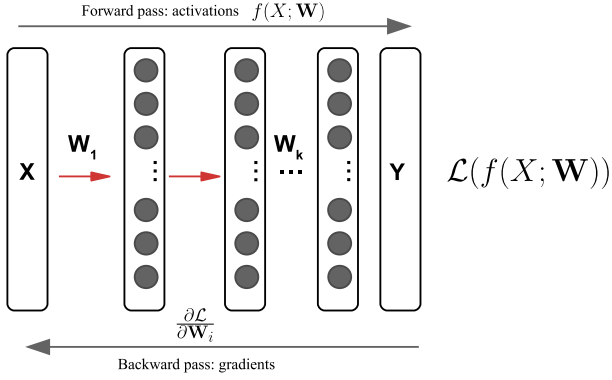
DEEP LEARNING: GRADIENT DESCENT

- Training: Loop until $\mathcal{L}^{tr}$, $\mathcal{L}^{ts}$, generalization gap are sufficiently small
  - **forward** pass, **backward** pass (**backpropagation** via **autodiff**, computing $\nabla_{\mathbf{W}}\mathcal{L}$ automatically), applying weight updates, . . .
- Thanks to **automatic differentiation** (reverse mode), computation of $\nabla_{\mathbf{W}}\mathcal{L}$ for any differentiable $\mathcal{L}(f)$ via dedicated libraries (TensorFlow, PyTorch, . . .)



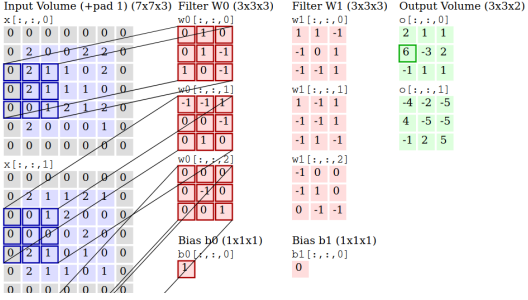
$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}$$

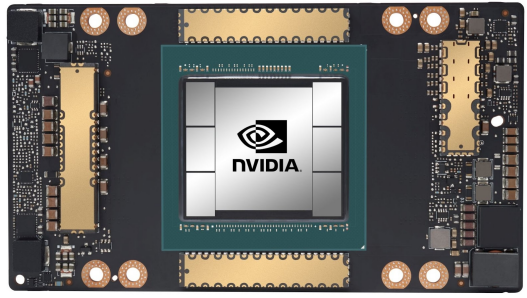
$$\mathcal{L}_{\mathbf{W_{t+1}}} \leq \mathcal{L}_{\mathbf{W_t}} \leq \cdots$$



DEEP LEARNING: GRADIENT DESCENT

- Convolutional and Transformer networks: dominant in many domains
  - Convolutional: vision (ResNets, EfficientNets, RegNets, . . .)
  - Transformers: language (GPT, BERT), vision (ViT, DeIT), language-vision (CLIP, DALL-E)
  - most compute intensive: tensor-tensor operations
- Tensor operations, simple non-linearities
  - highly optimized **vectorized** implementations for GPU
    - cuBLAS, cuDNN
  - both forward and backward path: extremely efficient execution on GPUs – fast training





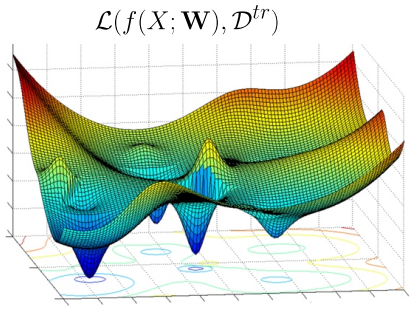
GRADIENT DESCENT AS OPTIMIZATION PROCEDURE

- Loss over a training set $\mathcal{D}^{tr}$ of size N as sum of losses for N single examples $X^i \in \mathcal{D}^{tr}$
  - remember: decomposition into sum due to data maximum likelihood estimation – i.i.d. assumption!

$$\mathcal{L} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i}_{\text{i.i.d.!}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(X^i, f(X^i; \mathbf{W}))$$

- Differentiation is a linear operator. Loss gradient over all X^i:

$$\nabla_{\mathbf{W}}\mathcal{L} = \nabla_{\mathbf{W}}\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{W}}\mathcal{L}_i$$



DEEP LEARNING: STOCHASTIC GRADIENT DESCENT

Two extremes:

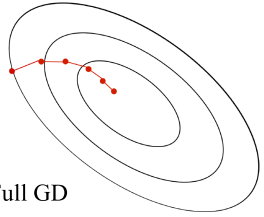
- full (batch) gradient descent:

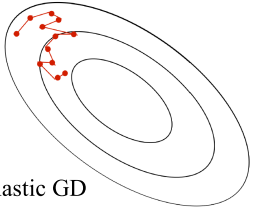
$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}$$

  - where $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i$: a single update step after computing the full gradient $\nabla_{\mathbf{W}} \mathcal{L}$ over the whole dataset!
  - 1 **epoch** – 1 update step!
- stochastic gradient descent (**SGD**):

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_i$$

  - performing update step after computing gradient from a single data point $X_i \in \mathcal{D}$
  - "noisy" gradient estimate $\nabla_{\mathbf{W}} \mathcal{L}_i$
  - 1 **epoch**: N update steps visiting all $X^i \in \mathcal{D}$





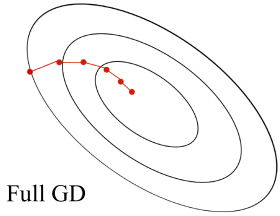
DEEP LEARNING: STOCHASTIC GRADIENT DESCENT

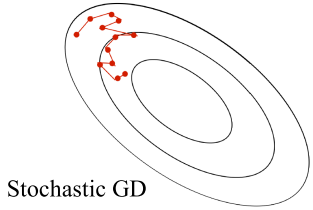
Inbetween two extremes: **mini-batch SGD**

- Perform an update step using loss gradient $\nabla_{\mathbf{w}}\mathcal{L}_B$ over a **mini-batch** of size $|B| = n \ll N$

$$\nabla_{\mathbf{w}}\mathcal{L}_B = \nabla_{\mathbf{w}}\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{w}}\mathcal{L}_i$$

- Combines benefits of both extremes:
  - can use **vectorization** for efficient gradient computation; affordable memory demand
  - is still noisy – escaping "bad" loss landscape regions; good for generalization? - still debated
  - fewer steps to converge than pure SGD

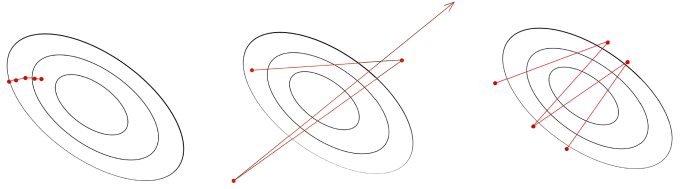




DEEP LEARNING: MINI-BATCH SGD

Training via mini-batch SGD

- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - **learning rate** η
  - **batch size** $|B|$



Slow | Instable, diverging | Instable, oscillating

Figure Roger Grosse, U Toronto, CSC421

DEEP LEARNING: MINI-BATCH SGD

Training via mini-batch SGD

- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on many other factors:
  - **network architecture**, **weight initialization**, **input & activity normalization**, **regularization** (weight decay, . . .), **data augmentation**, **optimizer type** (e.g. SGD with Nesterov momentum, . . .)

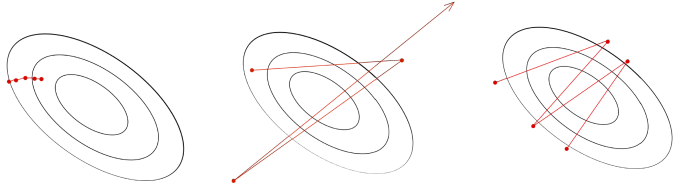
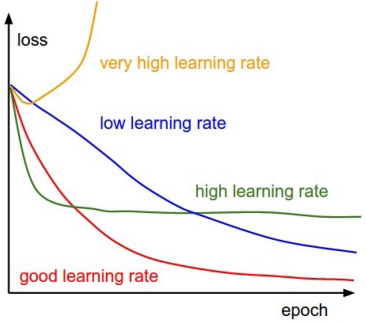


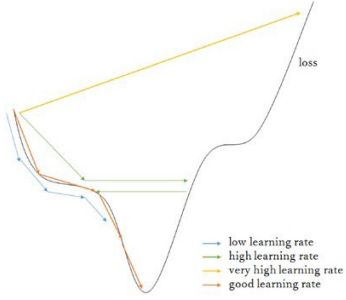
Figure Roger Grosse, U Toronto, CSC421

DEEP LEARNING: WEIGHT UPDATE DYNAMICS

Training via mini-batch SGD

- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - **learning rate** η
  - batch size $|B|$

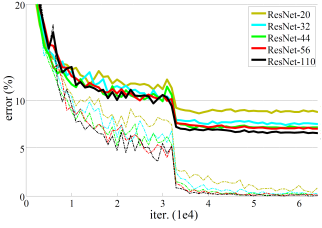




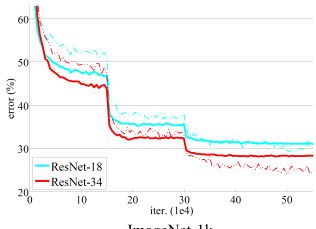
DEEP LEARNING: WEIGHT UPDATE DYNAMICS

Training via mini-batch SGD

- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - **learning rate** η : rate schedules
  - batch size $|B|$



CIFAR-10



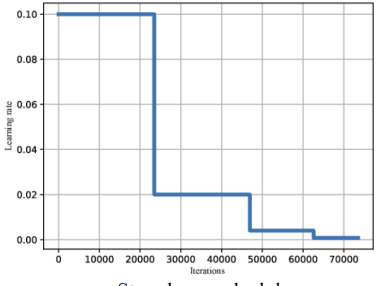
ImageNet-1k

He et al, CVPR, 2016

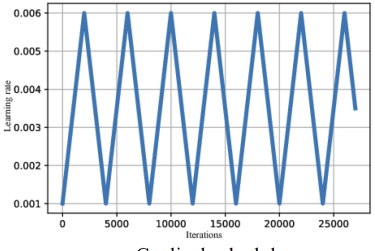
DEEP LEARNING: WEIGHT UPDATE DYNAMICS

Training via mini-batch SGD

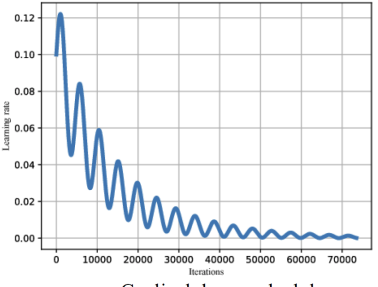
- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - **learning rate** η : rate schedules
  - batch size $|B|$



Step-decay schedule



Cyclical schedule

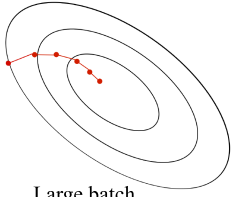


Cyclical decay schedule

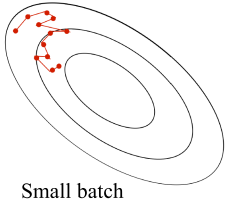
DEEP LEARNING: WEIGHT UPDATE DYNAMICS

Training via mini-batch SGD

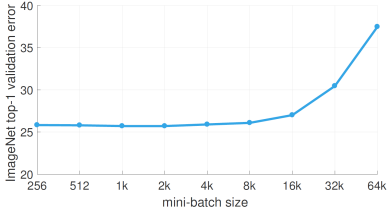
- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - learning rate η
  - **batch size** $|B|$ (changing $|B|$ often requires changing η)



Large batch



Small batch

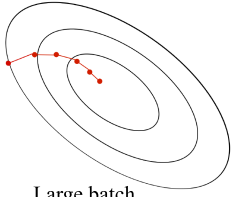


Goyal et al., 2017

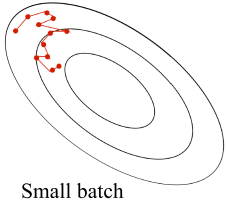
DEEP LEARNING: WEIGHT UPDATE DYNAMICS

Training via mini-batch SGD

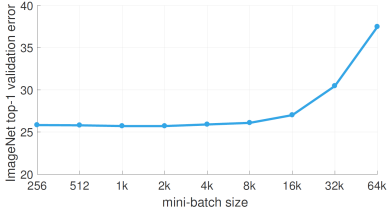
- Learning dynamics $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_B$
  - convergence speed
  - final loss region
- Strongly dependent on
  - learning rate η
  - **batch size** $|B|$: SGD type dependent (Nesterov momentum vs plain SGD, . . .)



Large batch



Small batch



Goyal et al., 2017

DEEP LEARNING: RECAP

Mini-batch SGD on large data

- Minimizing loss $\mathcal{L}^{tr}$ on very large $\mathcal{D}^{tr}$
- Adapting differentiable complex $f(X; \mathbf{W})$, $\mathbf{W}$ very large
- Using mini-batch loss gradient $\nabla_{\mathbf{W}} \mathcal{L}_B$ for weight updates
- Loss landscape complex and unknown, many pitfalls
- Optimization critically depends on proper initialization and hyperparams
  - interaction of **learning rate** η and **batch size** $|B|$
    - **batch size** issues: relevant for distributed training
- **Remember**: minimizing $\mathcal{L}^{tr}$ is not main target
  - generalization matters most: estimate gap on $\mathcal{L}^{ts}$