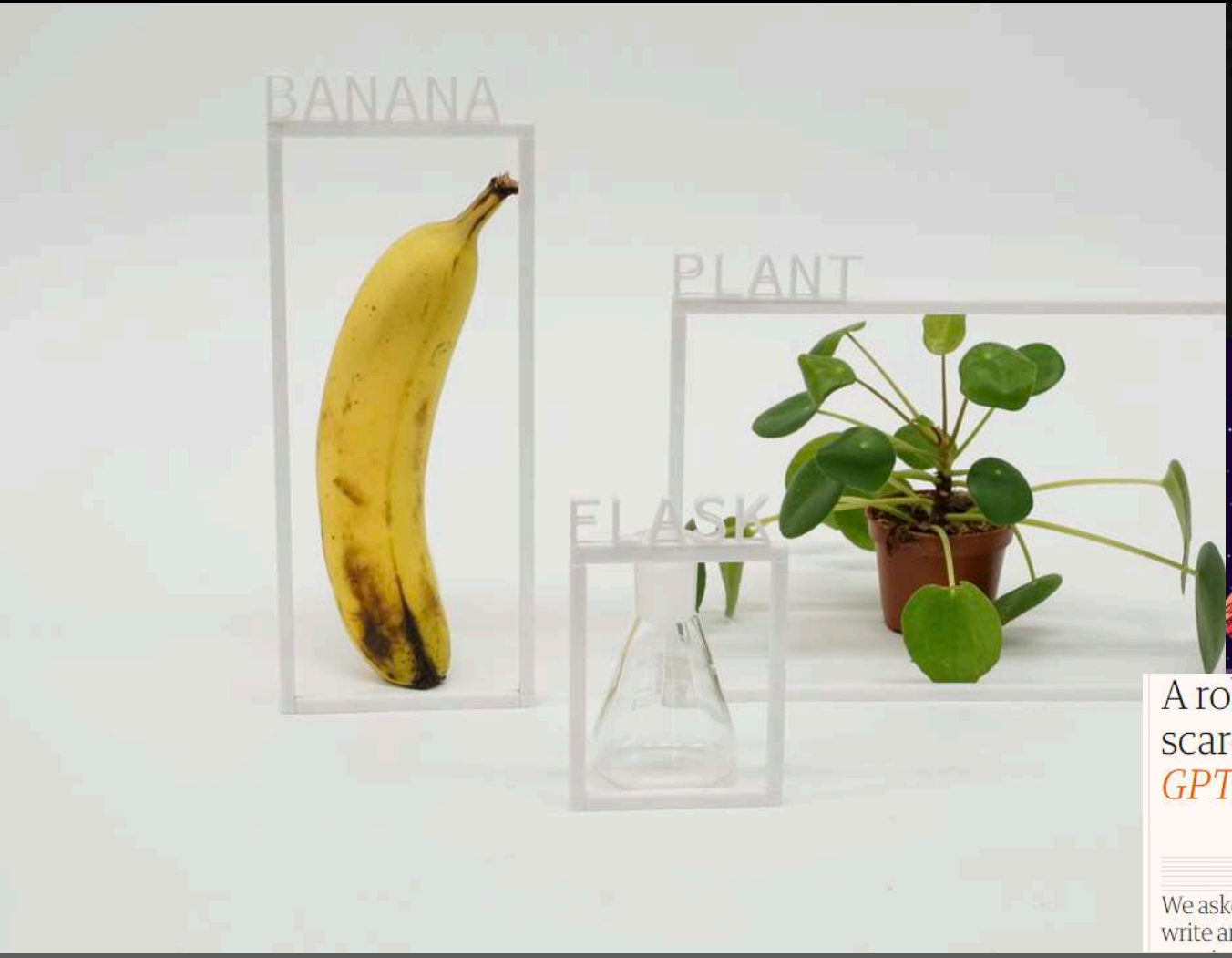


Deep Learning Lecture -- MLx Fundamentals 2024

Yuki M. Asano

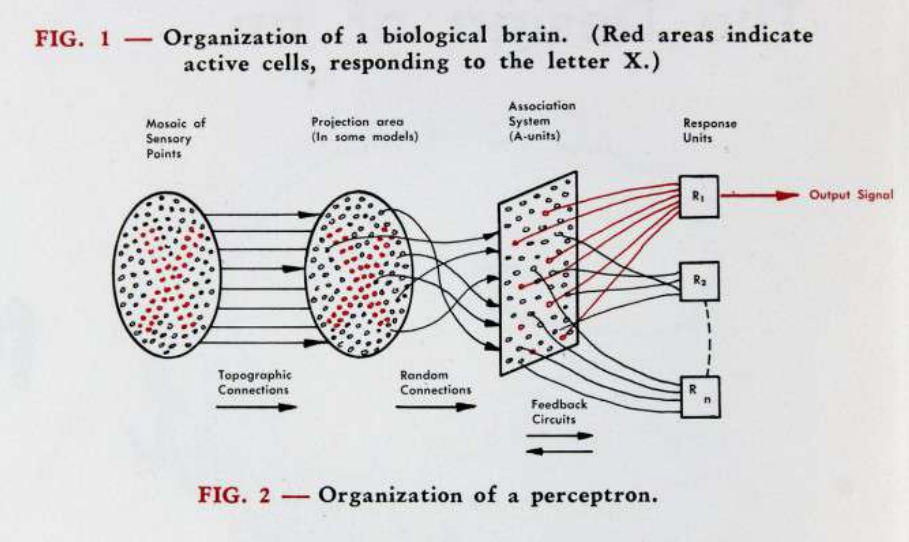
Hi, I'm Yuki M. Asano.

- Assistant Professor at Univ. Amsterdam
  - Self-supervised Learning
  - Multi-modal Learning
  - Large Model Adaptation
  - Privacy and Bias in Computer Vision
- Prior to this:
  - PhD at VGG in Oxford; Applied Mathematics/Physics/Economics at Oxford/Munich/Hagen
- Find more about myself here: https://yukimasano.github.io/ or just ask ☺

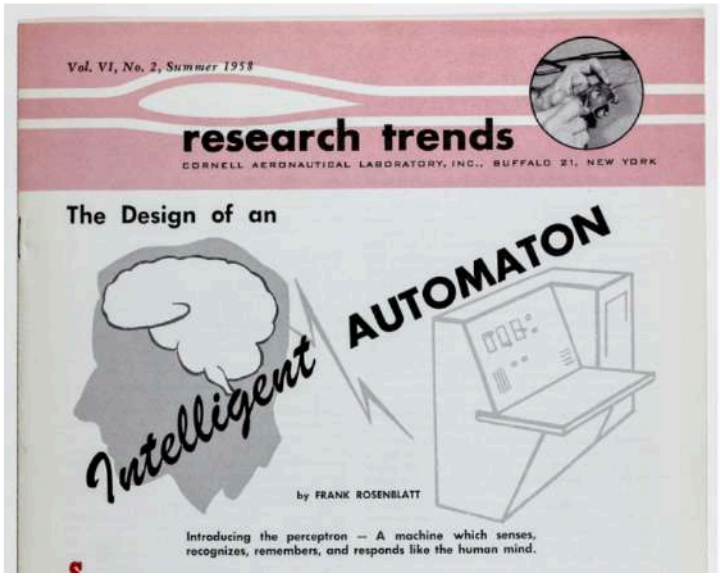
History & why is it "deep" learning

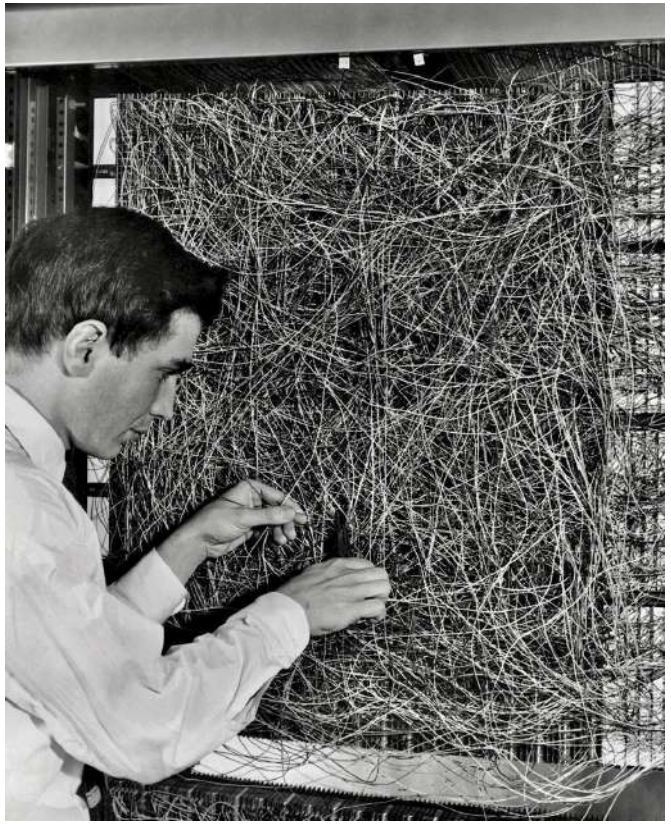
First appearance (roughly)

Rosenblatt: *The Design of an Intelligent Automaton* (1958)



"a machine which senses, recognizes, remembers, and responds like a human mind"





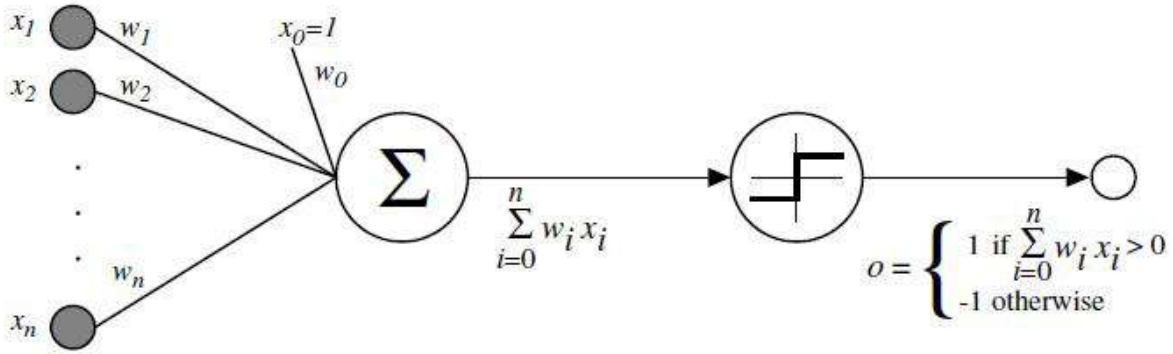
You think your wiring is chaotic?

Perceptrons

- (McCulloch & Pitts: binary inputs & outputs, no weights/learning)
- Rosenblatt proposed perceptrons for binary classifications
- A model comprising one weight w_i per input continuous x_i
- Multiply weights with respective inputs and add bias ($b = w_0,\ x_0 = +1$)

$$y = \sum_{j=1}^{n} w_j x_j + b = \sum_{j=0}^{n} w_j x_j$$

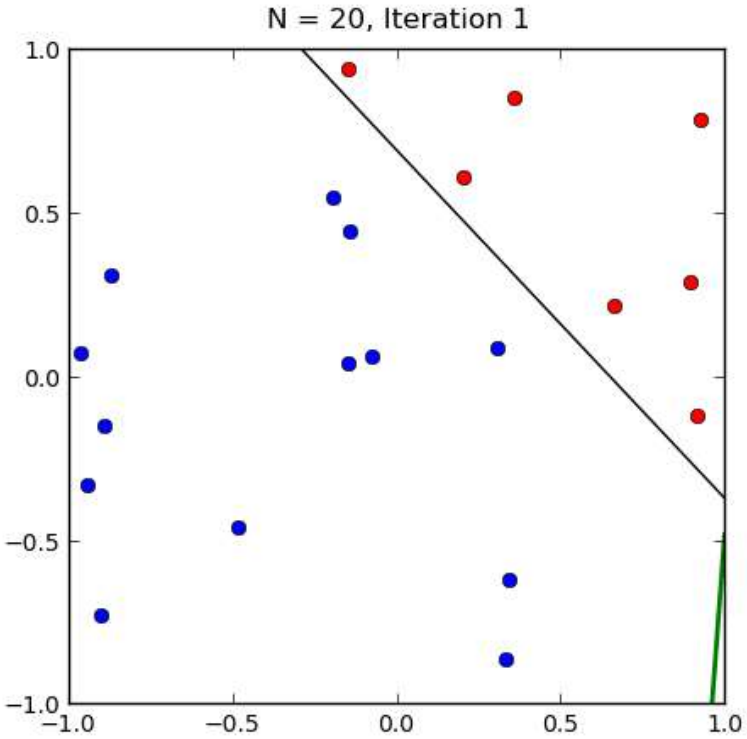
- If score y positive then return 1, otherwise -1



Training a perceptron

- Main innovation: a learning algorithm for perceptrons (Hebbian theory)

Perceptron learning algorithm	**Comments**
1. Set	
2. Sample new	New train image, label
3. Compute	: indicator function
4. If	Score too low. Increase weights!
5. If	Score too high. Decrease weights!
6. Go to 2	Repeat till happy ☺



XOR & 1-layer perceptrons

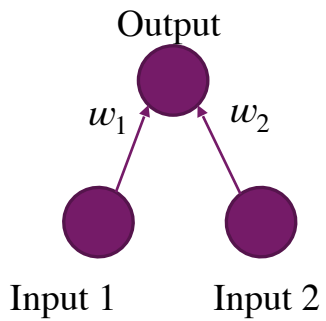
- The original perceptron has trouble with simple non-linear tasks though
- E.g., imagine a NN with two inputs that imitates the "exclusive-or" (XOR)
  - τ is the threshold for either +1 or -1 prediction

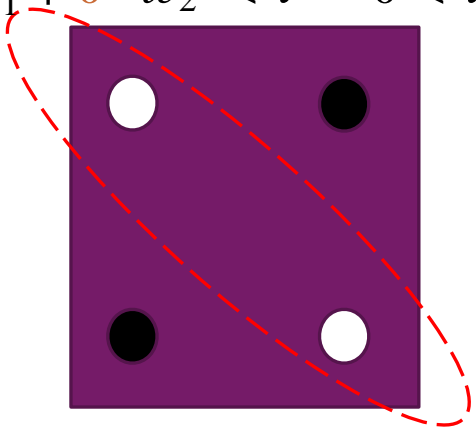
Input 1	Input 2	XOR
1	1	-1
1	0	+1
0	1	+1
0	0	-1

$$1 \cdot w_1 + 1 \cdot w_2 < \tau \rightarrow w_1 + w_2 < \tau$$
$$1 \cdot w_1 + 0 \cdot w_2 > \tau \rightarrow w_1 > \tau$$
$$0 \cdot w_1 + 1 \cdot w_2 > \tau \rightarrow w_2 > \tau$$
$$0 \cdot w_1 + 0 \cdot w_2 < \tau \rightarrow 0 < \tau$$

$$w_1 + w_2 > 2\tau$$
$$w_1 + w_2 < \tau$$

Inconsistent



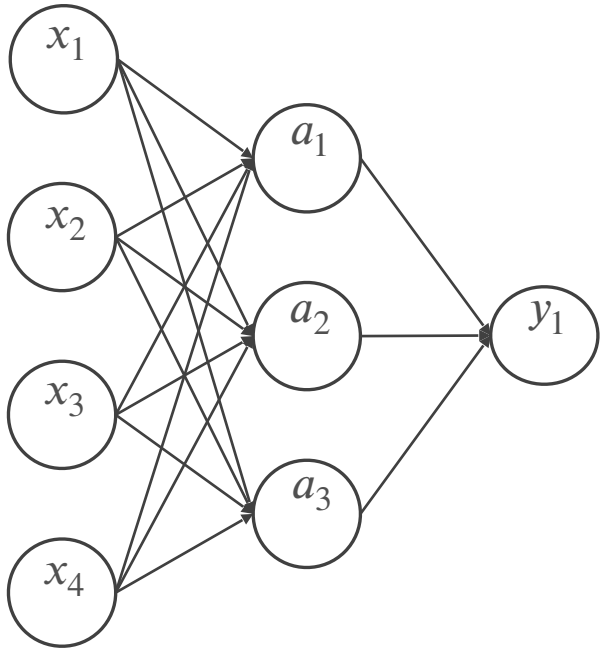


No line can separate the white from the black

Minsky and Papert, "Perceptrons", 1969

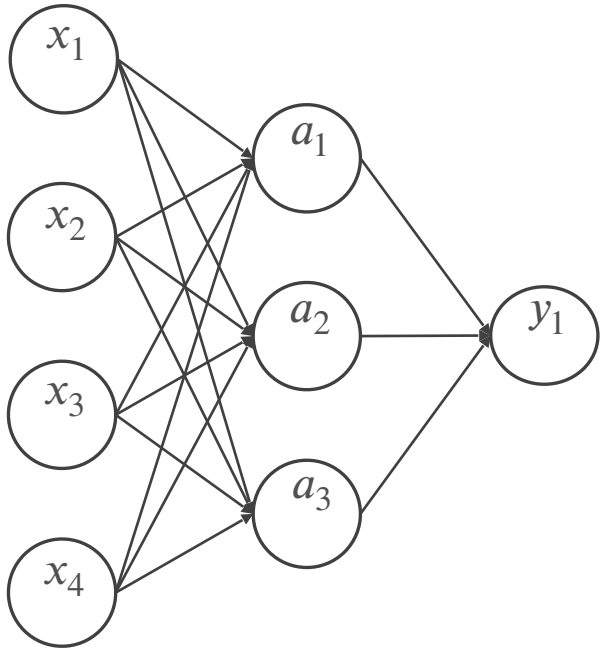
Multi-layer perceptrons to the rescue

- Minsky **never said** XOR cannot be solved by neural networks
  - Only that XOR cannot be solved with **1-layer** perceptrons
- Multi-layer perceptrons (MLP) can solve XOR
  - One layer's output is input to the next layer
  - Add nonlinearities between layers, e.g., sigmoids
  - Or even single layer with "feature engineering"
- Problem: how to train a multi-layer perceptron?
- Rosenblatt's algorithm not applicable.



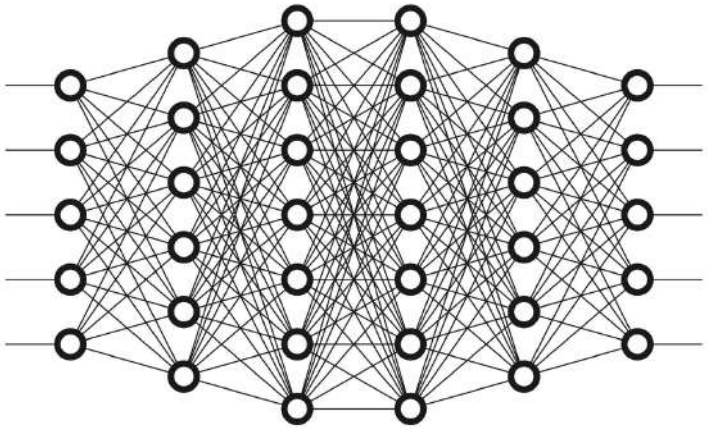
Multi-layer perceptrons to the rescue

- Rosenblatt's algorithm not applicable. Why?
  - Learning depends on "ground truth" l_i for updating weights
  - For the intermediate neurons a_j there is no "ground truth"
  - The Rosenblatt algorithm cannot train intermediate layers



Deep feedforward networks

- Feedforward neural networks
  - Also called multi-layer perceptrons (MLPs)
  - The goal is to approximate some function f
  - A feedforward network defines a mapping



$$y = f(x; \theta)$$

  - Learns the value of the parameters θ that result in the best function approximation.
- No feedback connections
  - When including feedback connections, we obtain recurrent neural networks.
  - Nb: brains have many feedback connections

Deep feedforward networks

- In a formula of a composite of functions:

$$\mathrm{y} = f(x;\theta) = a_L\left(x;\theta_{1,\ldots,\mathrm{L}}\right) = h_L\left(h_{L-1}\left(\ldots(h_1\left(x,\theta_1\right),\ \ldots),\ \theta_{L-1}\right),\theta_L\right)$$

where θ_l is the parameter in the l-th layer

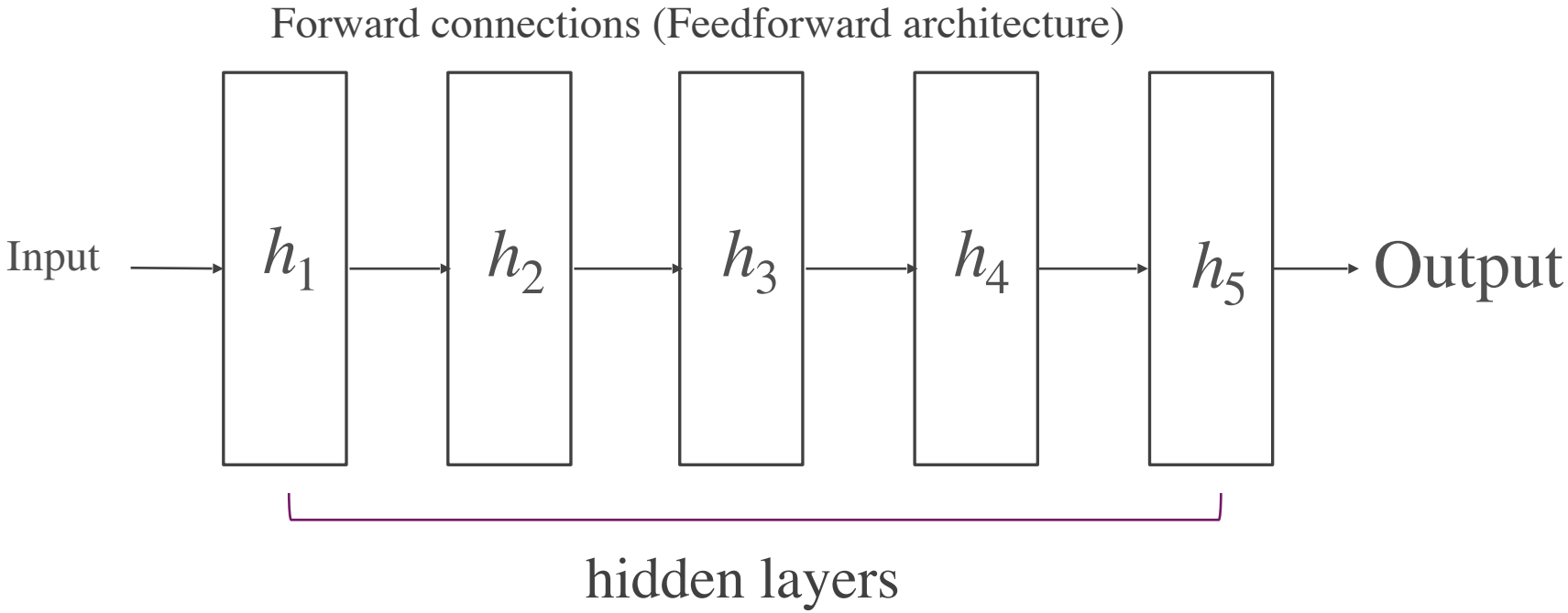
- We can simplify the notation by

$$a_L = f(x;\theta) = h_L \circ h_{L-1} \circ \ldots \circ h_1 \circ x$$

where each functions h_l is parameterized by the parameter θ_l

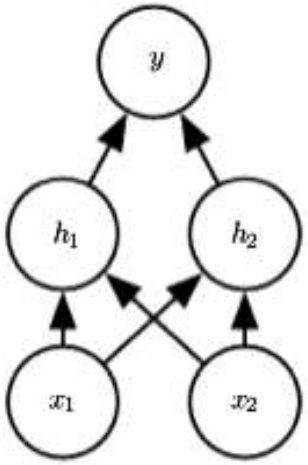
Neural networks in blocks

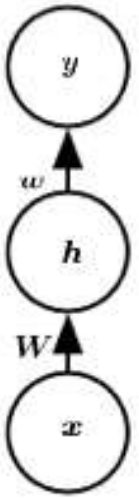
- We can visualize $a_L = h_L \circ h_{L-1} \circ \ldots \circ h_1 \circ x$ as a cascade of blocks



What is a module?

- Module ⇔ Building block ⇔ Transformation ⇔ Function
- A module receives as input either data x or another module's output
- A module returns an output a based on its activation function $h(\ldots)$
- A module may or may not have trainable parameters w
- Examples: f = Ax, f= exp(x)





Requirements

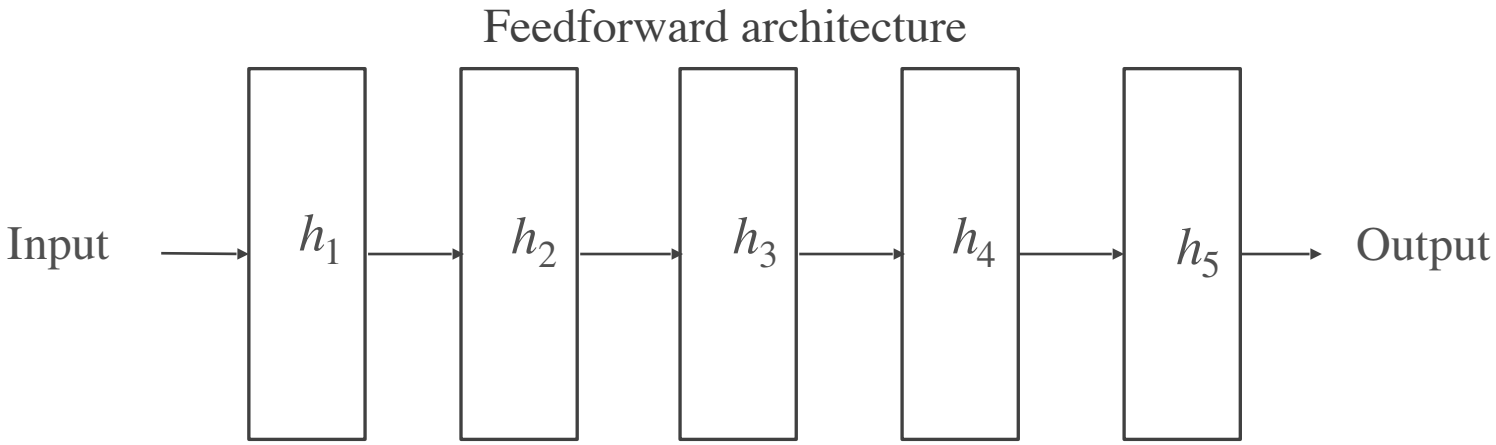
(1) The activation functions must be **1st-order differentiable (almost) everywhere**

(2) Take special care when there are cycles in the architecture of blocks

- No other requirements
- We can build as complex hierarchies as we want

Feedforward model

- The vast majority of models
- Almost all CNNs/Transformers
- As simple as it gets



Non-linear feature learning perspective

- Linear models
  - logistic regression, linear regression
  - convex, with closed-form solution
  - can be fit efficiently and reliably
  - limited capacity

- Extend to nonlinear models
  - apply the linear model not to x itself but to a transformed input $\phi(x)$, where ϕ is a nonlinear transformation
  - kernel trick, e.g., RBF kernel
  - nonlinear dimension reduction

Non-linear feature learning perspective

- The strategy of deep learning is to learn ϕ: $y = f(x; \theta, w) = \phi(x; \theta)^{\mathrm{T}} w$
  - ϕ defines a hidden layer
  - find the θ that corresponds to a good* representation.
  - no longer a convex training problem
- We design families $\phi(x; \theta)$ rather than the right function
  - Encode human knowledge to help generalization

* *good* = linearly separable (in the case of classification)

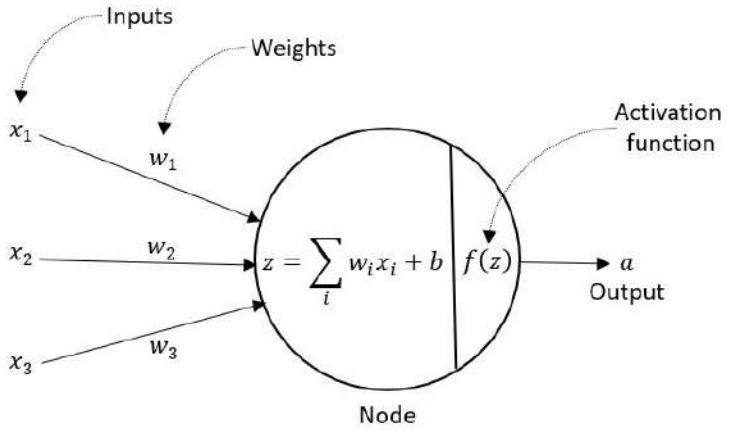
Cost functions

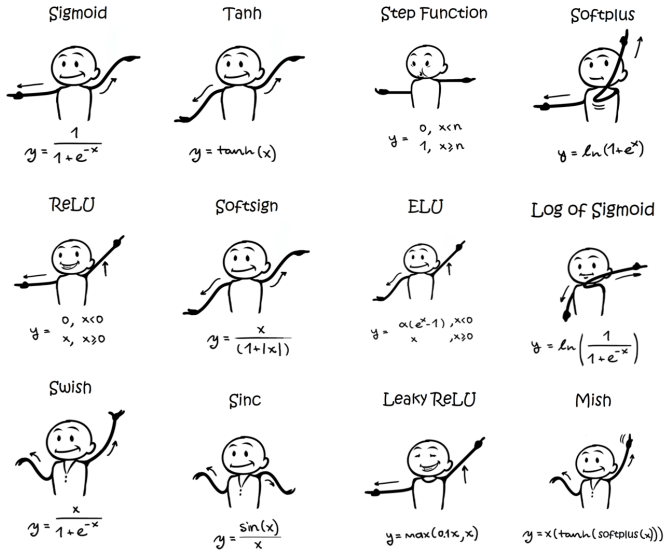
- Main point: cost functions describe what the model should do
- The gradient of the cost function must be large and predictable enough to serve as a good guide for learning algorithms
- Functions that saturate (become very flat) undermine this objective.
- In many cases, this is due to the activation functions saturation.
- The negative log-likelihood help to avoid this problem for many models because it can undo the exponentiation of the output (eg see softmax definition later)

Deep Learning Modules

Activation functions

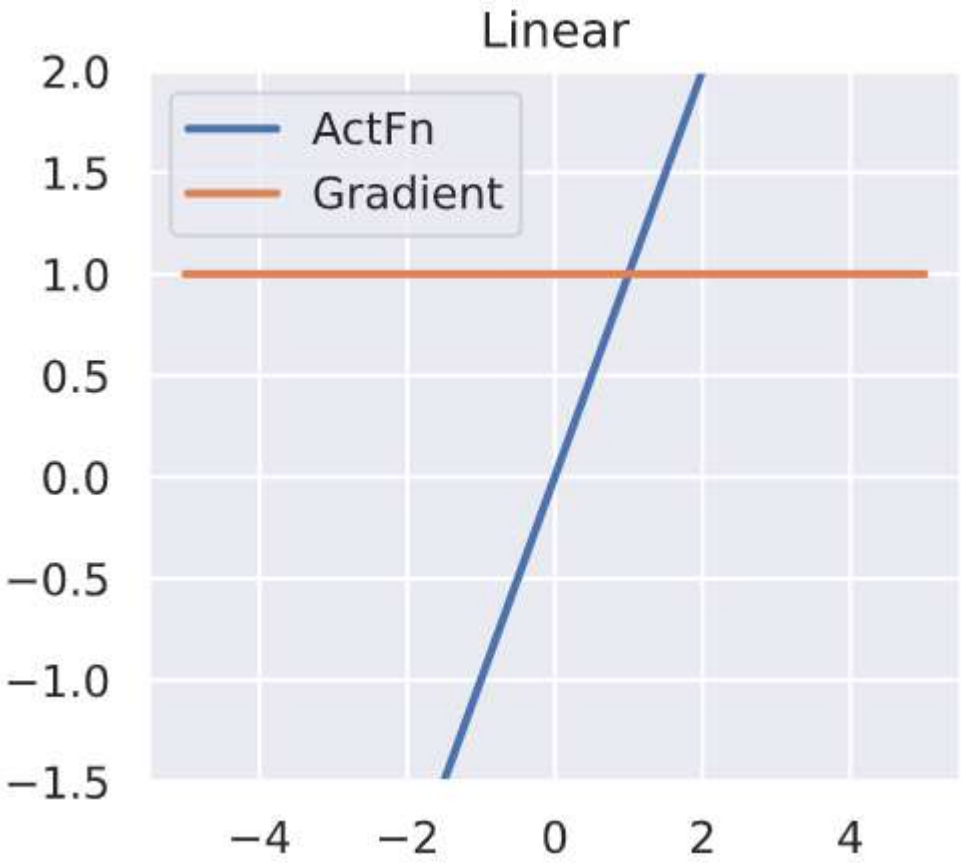
- Defined how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.
- If output range limited, then called a "*squashing function*."
- The choice of activation function has a large impact on the capability and performance of the neural network.
- Different activation functions may be combined, but rare
- All hidden layers typically use the same activation function
- Need to be differentiable at most points





Linear Units / "Fully connected layer"

$$\boldsymbol{x} \in \mathbb{R}^{1\times M},\ \boldsymbol{w} \in \mathbb{R}^{N\times M}$$
$$h(x;w) = \boldsymbol{x}\cdot\boldsymbol{w}^T + b$$
$$\frac{dh}{d\boldsymbol{x}} = \boldsymbol{w}$$



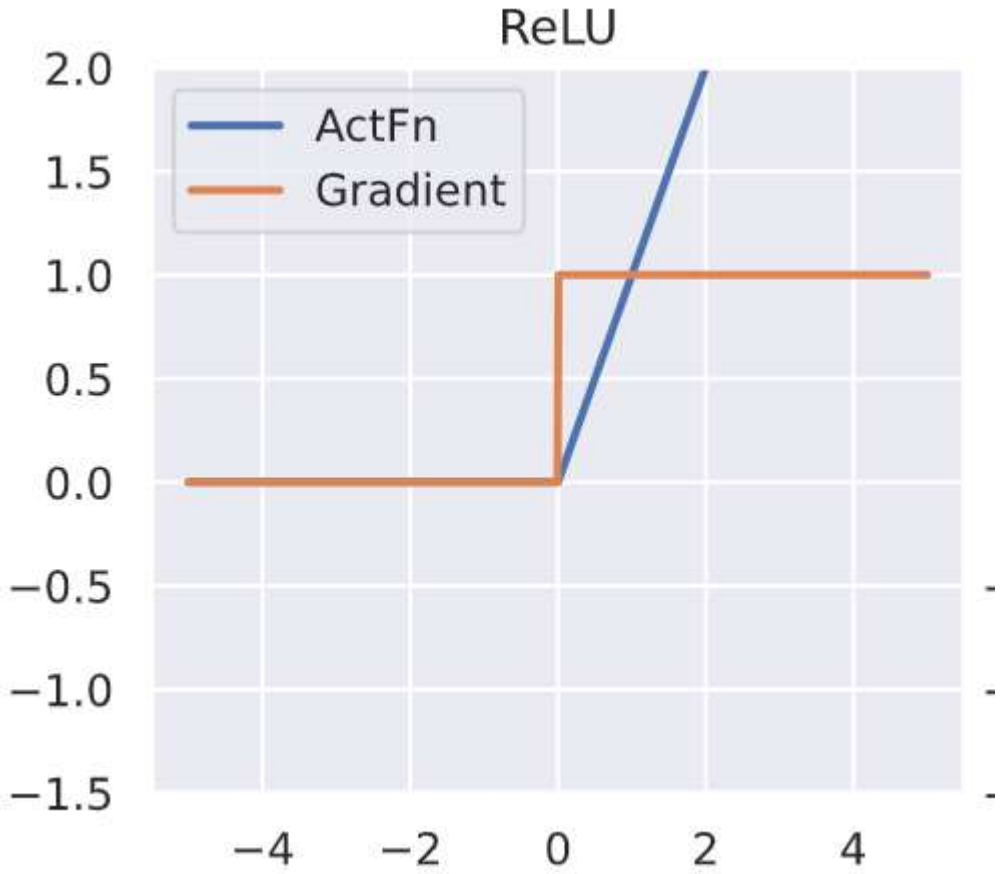
- Identity activation function
- No activation saturation
- Hence, strong & stable gradients
  - Reliable learning with linear modules

Rectified Linear Unit (ReLU)

ReLU

$$h(x) = \max(0, x)$$

$$\frac{\partial h}{\partial w} = \begin{cases} 1 \text{ when } x > 0 \\ 0, \text{ when } x \leq 0 \end{cases}$$



Rectified Linear Unit (ReLU)

- Advantages
  - Sparse activation: In randomly initialized network, ~50% active
  - Better gradient propagation: Fewer vanishing gradient problems compared to sigmoidal activation functions that saturate in both directions.
    - Eg for sin(x), x<<1: (small number) * (small number) * …. --> 0
  - Efficient computation: Only comparison, addition and multiplication.
  - Scale-invariant

Rectified Linear Unit (ReLU)

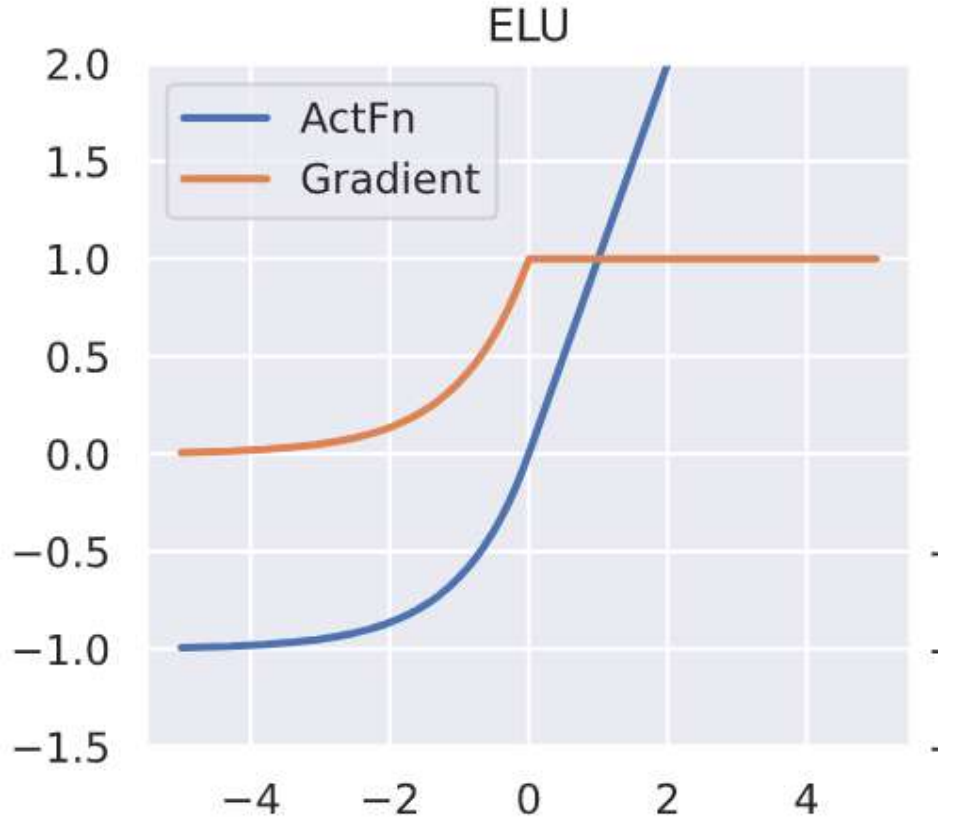
- Potential problems
  - Non-differentiable at zero; however, it is differentiable anywhere else, and the value of the derivative at zero can be arbitrarily chosen to be 0 or 1.
  - Not zero-centered.
  - Unbounded.
  - Dead neurons problem: neurons can sometimes be pushed into states in which they become inactive for essentially all inputs. Higher learning rates might help

Exponential Linear Unit (ELU)

ELU

$$h(x) = \begin{cases} x, \text{ when } x > 0 \\ \exp(x) - 1, \ x \leq 0 \end{cases}$$

$$\frac{\partial h}{\partial x} = \begin{cases} 1, \text{ when } x > 0 \\ \exp(x), x \leq 0 \end{cases}$$



- ELU is a smooth approximation to the rectifier.
- It has a non-monotonic “bump” when x < 0.
- It serves as the default activation for models such as BERT.

Gaussian Error Linear Unit

GELU

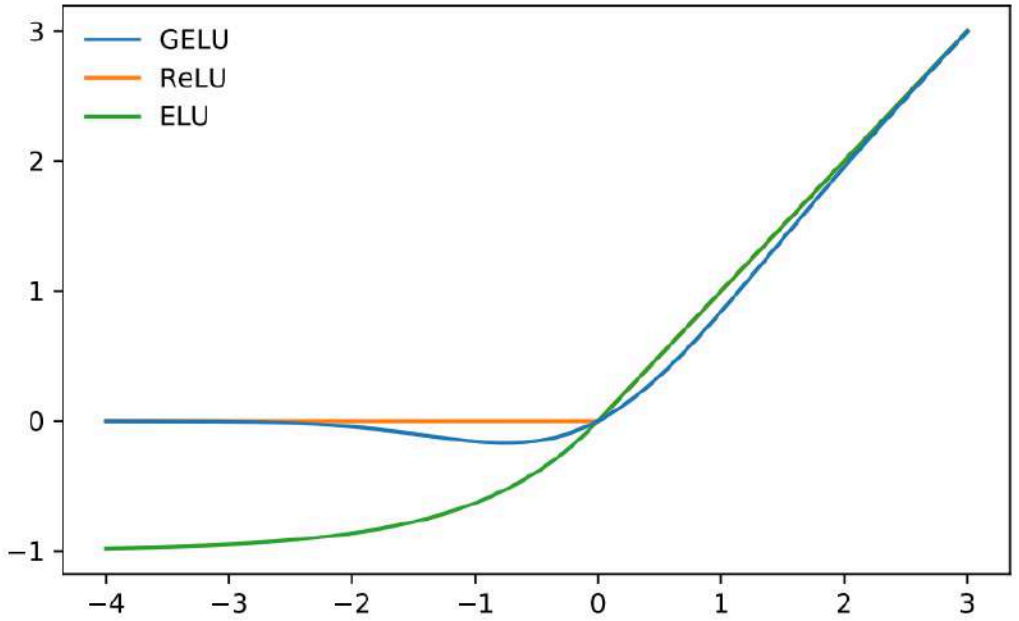
$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right].$$

We can approximate the GELU with

$$0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

or

$$x\sigma(1.702x),$$



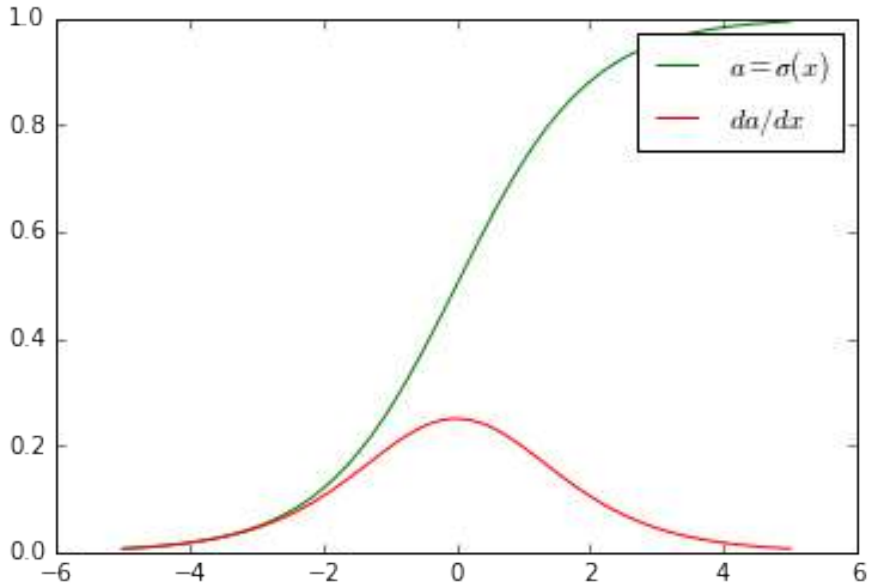
- Similar to ELU, but non-monotonic (change in gradient sign)
- Default for Vision Transformers & state of the art

https://arxiv.org/pdf/1710.05941

Softmax

Softmax

$$h(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

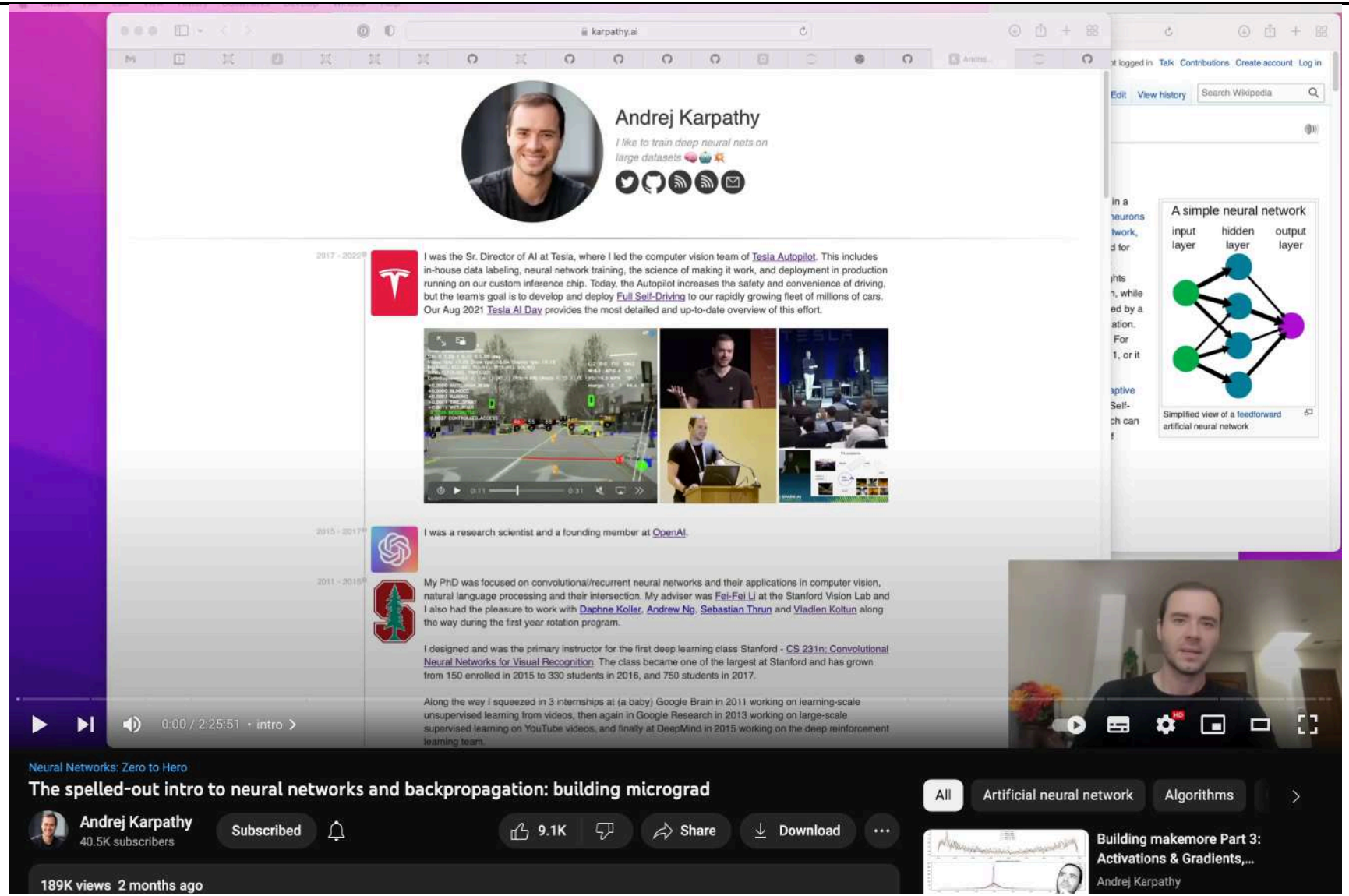


- Outputs probability distribution, (why?)
- $\sum_{i=1}^{K} h(x_i) = 1$ for K classes or simply normalizes in a non-linear manner.
- Avoid exponentianting too large/small numbers for better stability

$$h(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - \mu}}{\sum_j e^{x_j - \mu}} \quad , \ \mu = \max_i x_i$$

Backpropagation

S


karpathy.ai
Andrej Karpathy
I like to train deep neural nets on large datasets
I was the Sr. Director of AI at Tesla, where I led the computer vision team of Tesla Autopilot. This includes in-house data labeling, neural network training, the science of making it work, and deployment in production running on our custom inference chip. Today, the Autopilot increases the safety and convenience of driving, but the team's goal is to develop and deploy Full Self-Driving to our rapidly growing fleet of millions of cars. Our Aug 2021 Tesla AI Day provides the most detailed and up-to-date overview of this effort.
I was a research scientist and a founding member at OpenAI.
My PhD was focused on convolutional/recurrent neural networks and their applications in computer vision, natural language processing and their intersection. My adviser was Fei-Fei Li at the Stanford Vision Lab and I also had the pleasure to work with Daphne Koller, Andrew Ng, Sebastian Thrun and Vladlen Koltun along the way during the first year rotation program.
I designed and was the primary instructor for the first deep learning class Stanford - CS 231n: Convolutional Neural Networks for Visual Recognition. The class became one of the largest at Stanford and has grown from 150 enrolled in 2015 to 330 students in 2016, and 750 students in 2017.
Along the way I squeezed in 3 internships at (a baby) Google Brain in 2011 working on learning-scale unsupervised learning from videos, then again in Google Research in 2013 working on large-scale supervised learning on YouTube videos, and finally at DeepMind in 2015 working on the deep reinforcement learning team.
A simple neural network
input layer
hidden layer
output layer
Simplified view of a feedforward artificial neural network
0:00 / 2:25:51 • intro
Neural Networks: Zero to Hero
The spelled-out intro to neural networks and backpropagation: building micrograd
Andrej Karpathy
40.5K subscribers
Subscribed
9.1K
Share
Download
189K views 2 months ago
All
Artificial neural network
Algorithms
Building makemore Part 3: Activations & Gradients,...
Andrej Karpathy

Backpropagation $\Longleftrightarrow$ Chain rule

- The neural network loss is a composite function of modules
- We want the gradient w.r.t. to the parameters of the l layer

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_L} \bullet \frac{dh_L}{dh_{L-1}} \bullet \ldots \bullet \frac{dh_l}{dw_l} \quad \Rightarrow \quad \frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

Gradient of loss w.r.t. the module output

Gradient of a module w.r.t. its parameters

- Back-propagation is an algorithm that computes the chain rule, with a specific **order of operations that is highly efficient**.

Backpropagation $\Longleftrightarrow$ Chain rule!!!

- Backpropagating gradients means repeating computation of 2 quantities

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

- For $\frac{dh_l}{dw_l}$ just compute the Jacobian of the l-th module w.r.t. to its parameters w_l
- Very local rule → "every module looks for its own"
- Since computations can be very local, this means that
  - graphs can be complex
  - modules can be complex if differentiable

Backpropagation $\Longleftrightarrow$ Chain rule but as an algorithm

- Backpropagating gradients means repeating computation of 2 quantities

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

- For $\frac{d\mathcal{L}}{dh_l}$ we apply chain rule again to recursively reuse computations

$$\frac{d\mathcal{L}}{dh^l} = \frac{d\mathcal{L}}{dh_{l+1}} \cdot \frac{dh_{l+1}}{dh_l}$$

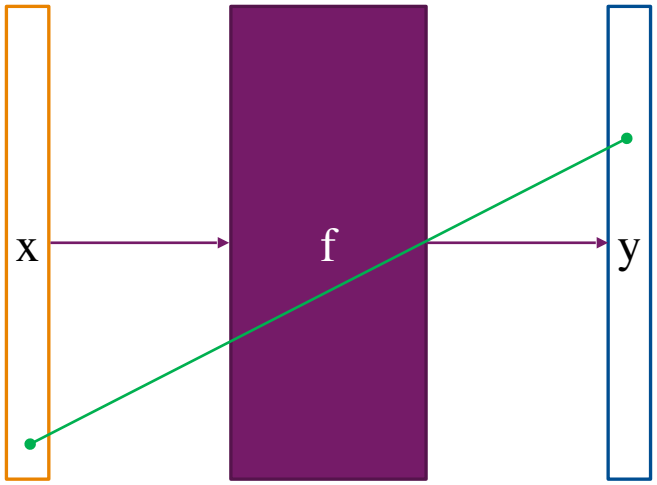
Recursive rule → computation-friendly

Gradient of module w.r.t. its module input

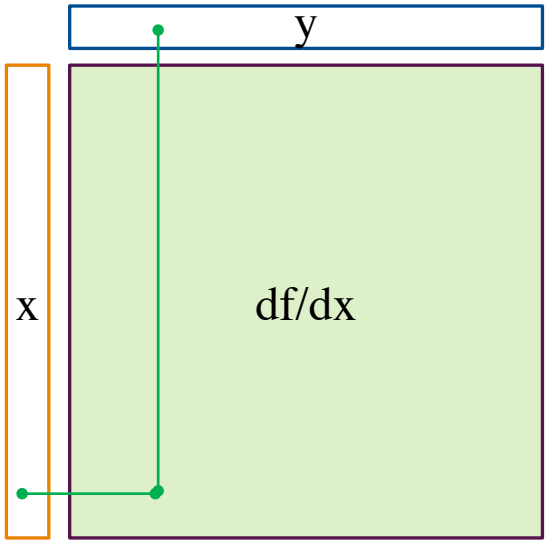
- Remember, the output of a module is the input for the next one: $a_l = x_{l+1}$

Computational feasibility

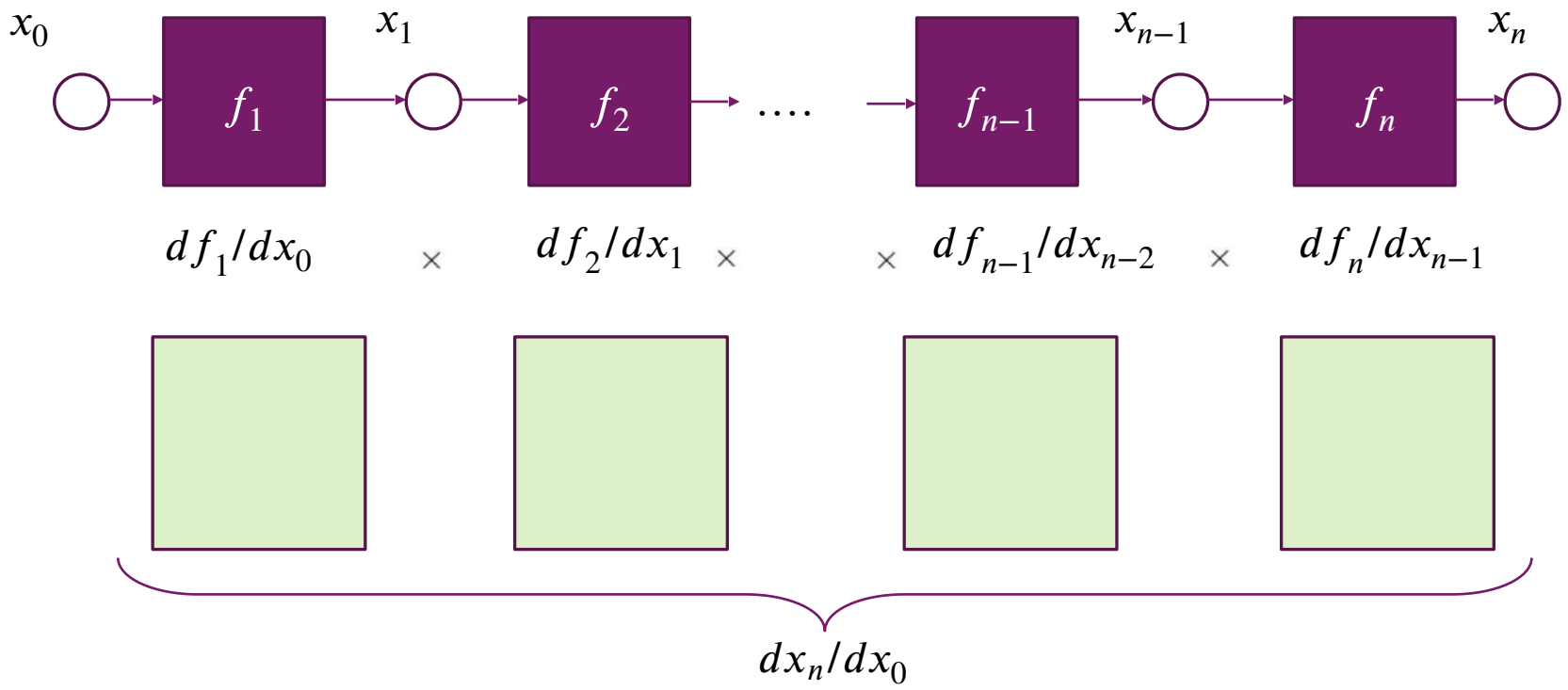
- $y = f(x)$
- Each x's contribution to y is given by the Jacobian, df/dx



- Suppose x and y are some intermediate outputs of size 32x32x512
- Then storing the Jacobian would take 1TB of memory.



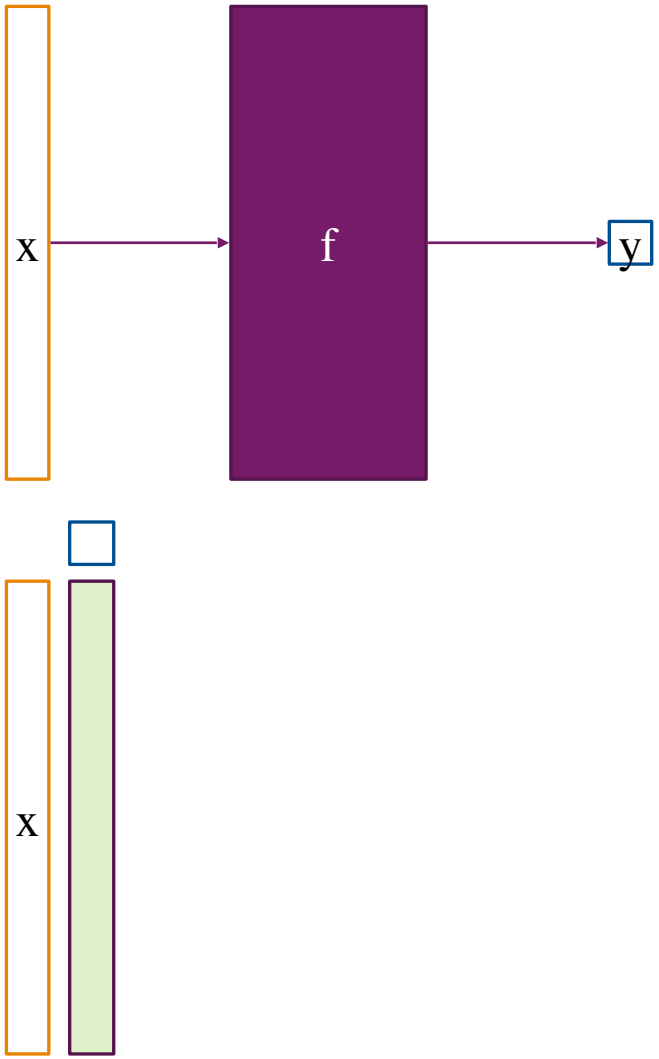
Chain rule visualized



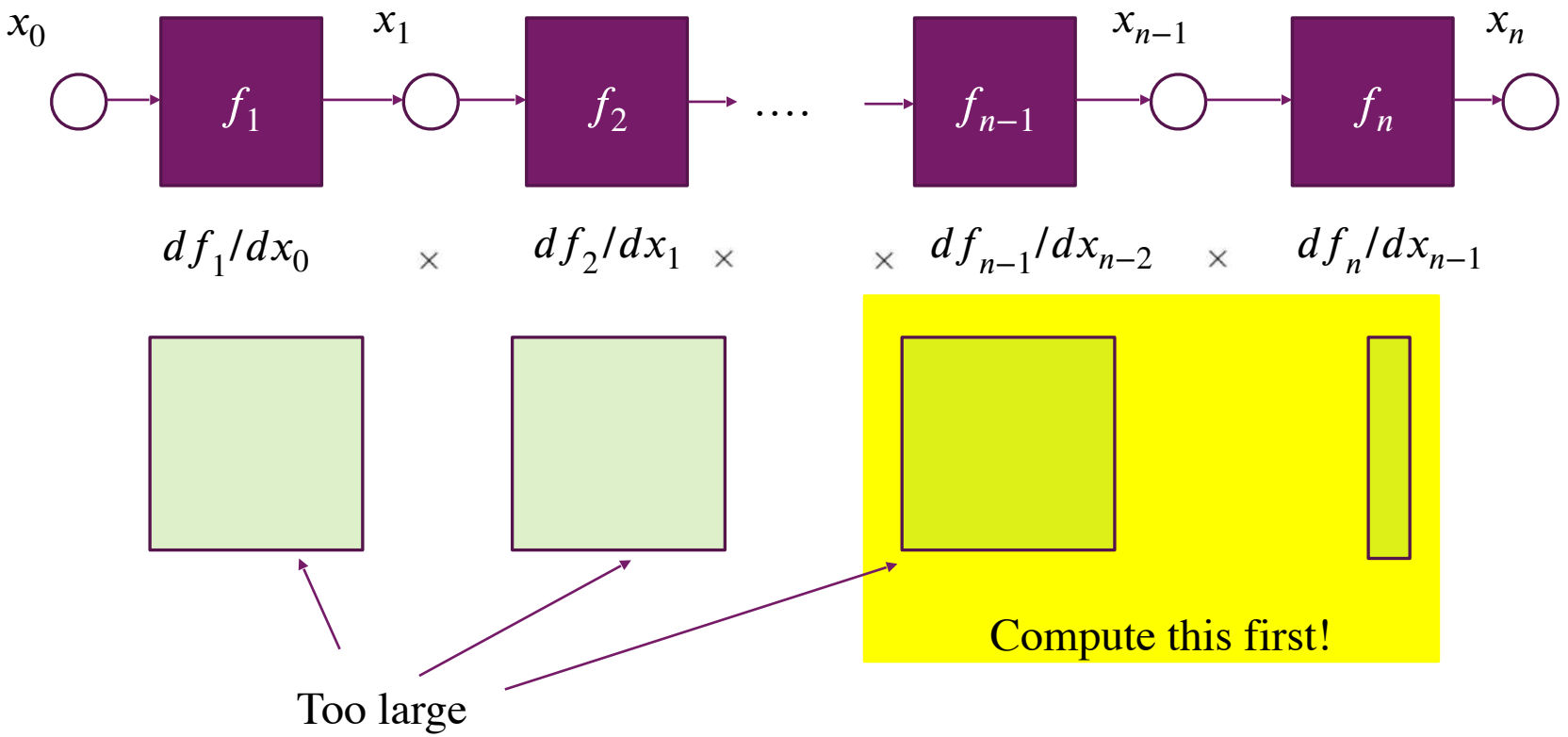
How to adjust x_1 to minimize x_n?
"just multiply Jacobians"
…
But this is not possible.

What if the output is a scalar?

- With x of size 32x32x512 and y=1,
- df/dx is only 32*32*512=524K elements ~ 2MB
- Of size $D_x \times 1$



Chain rule visualized

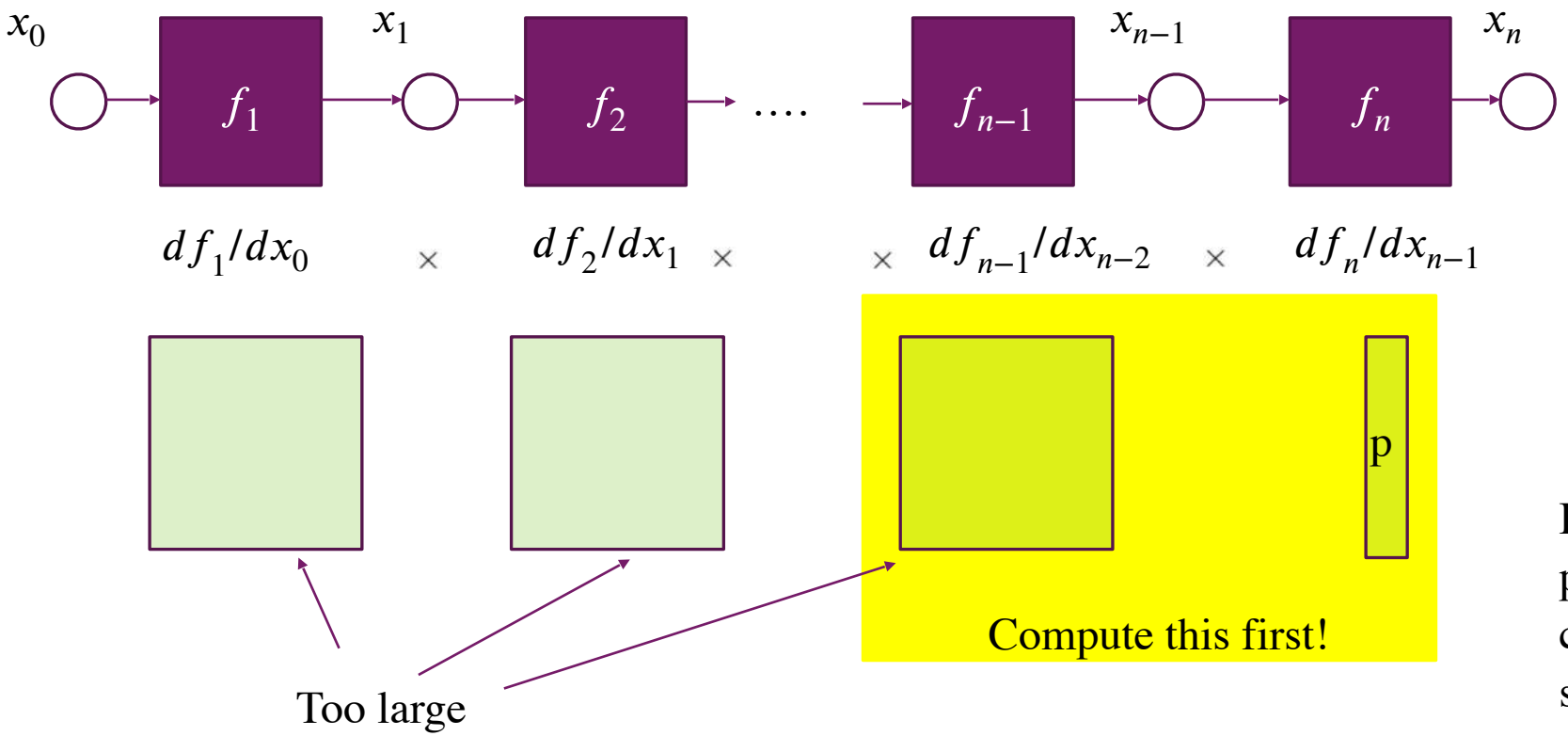


A simple matrix-vector product:

$D_{n-1} \times D_n \quad D_n \times 1$

Result again low size: $D_{n-1} \times 1$

Chain rule visualized

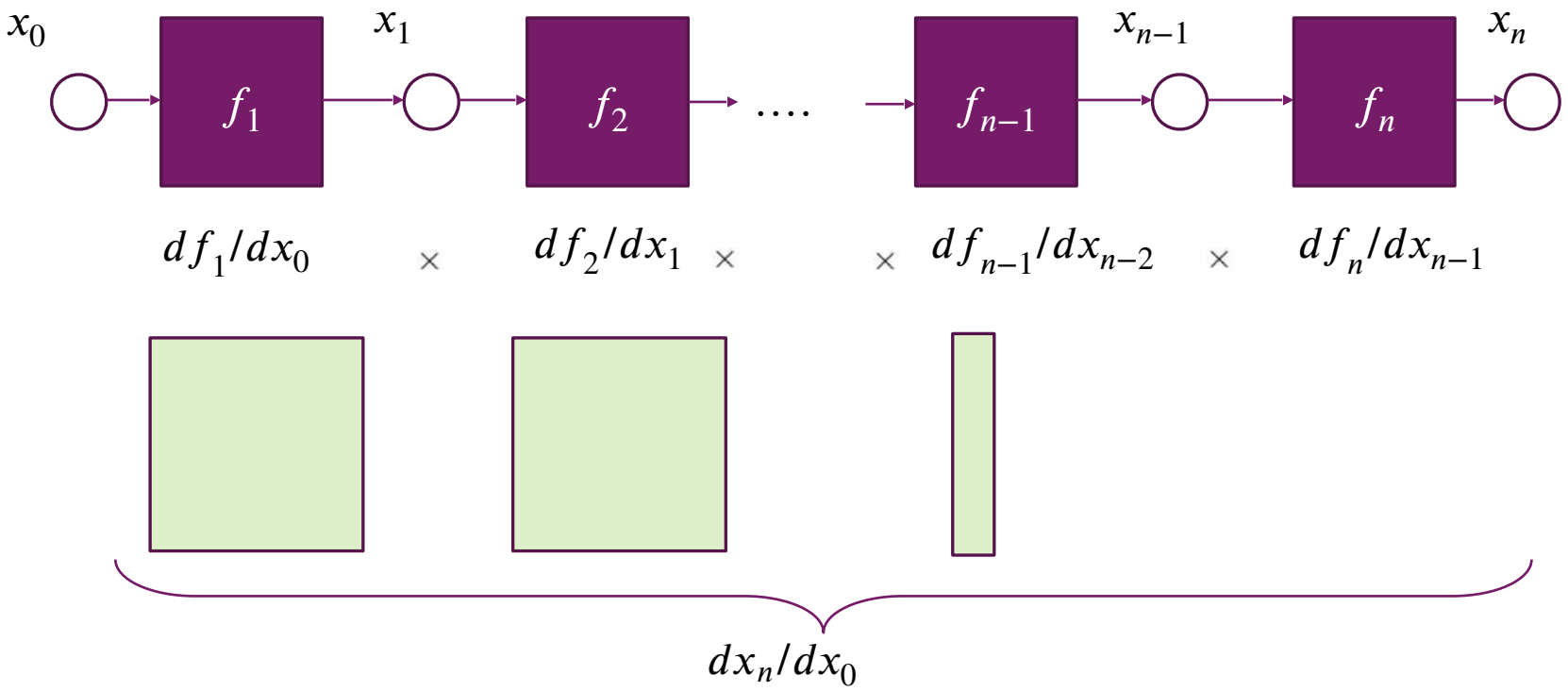


$$\mathbf{p} \cdot \frac{df}{d\mathbf{x}}(\mathbf{x}) = \frac{d(\mathbf{p} \cdot f)}{d\mathbf{x}}(\mathbf{x}).$$

In other words, the vector-matrix product in the left hand side can be computed as the derivative of the scalar-valued projected function p · f to the right.

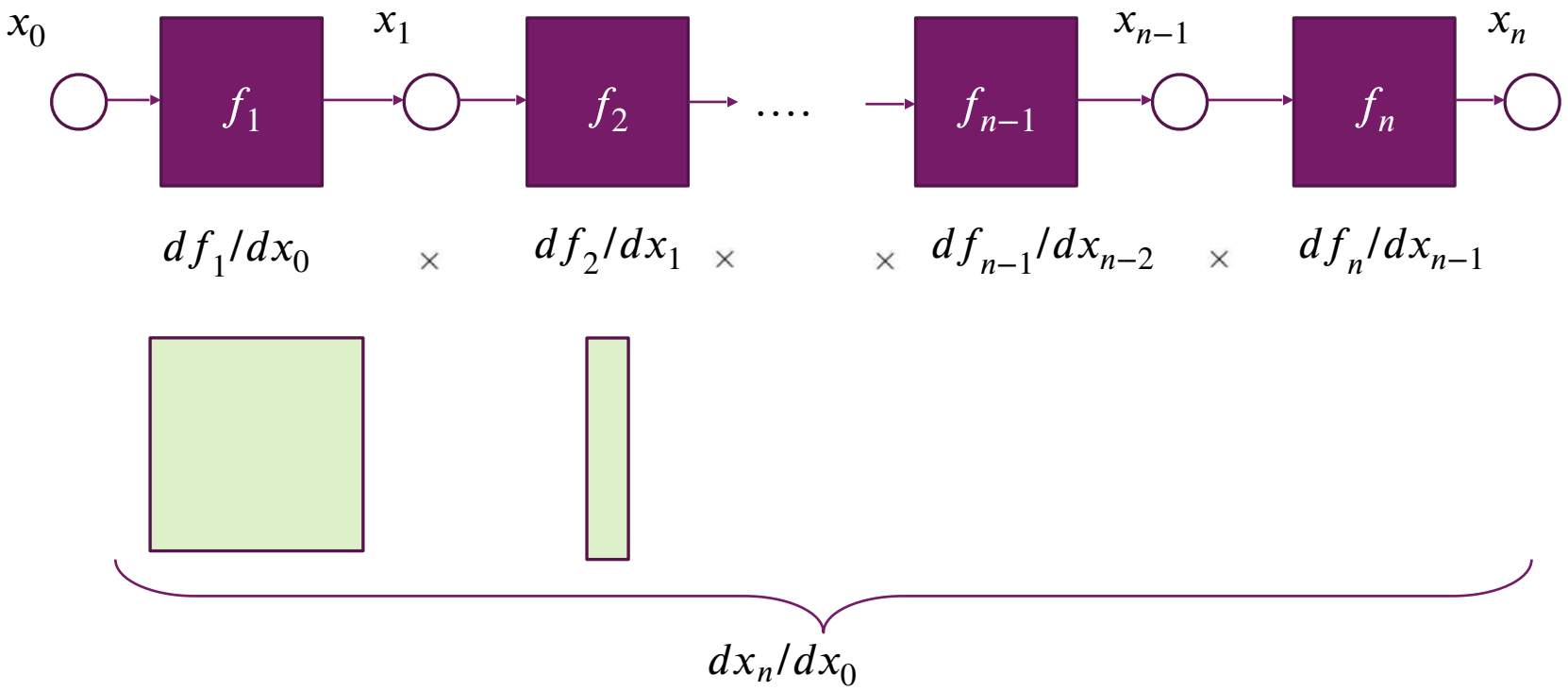
AutoDiff toolboxes allow you to write efficient derivatives of <p,f> , and take care of the rest.

Chain rule visualized



Keep going

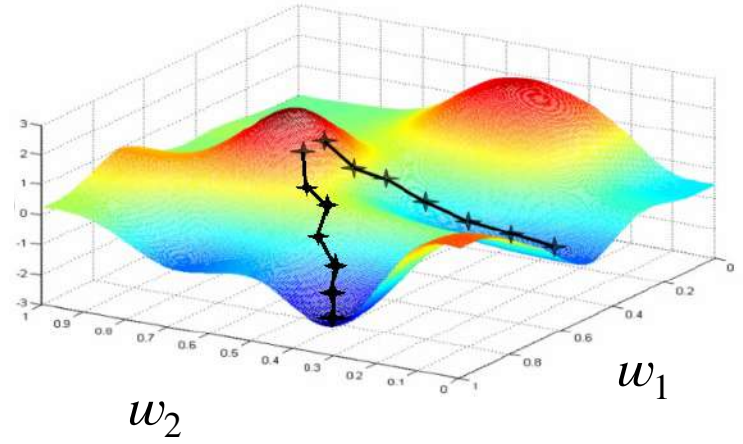
Chain rule visualized



Keep going

Optimization

Gradient descent



- Gradient descent is the iterative method

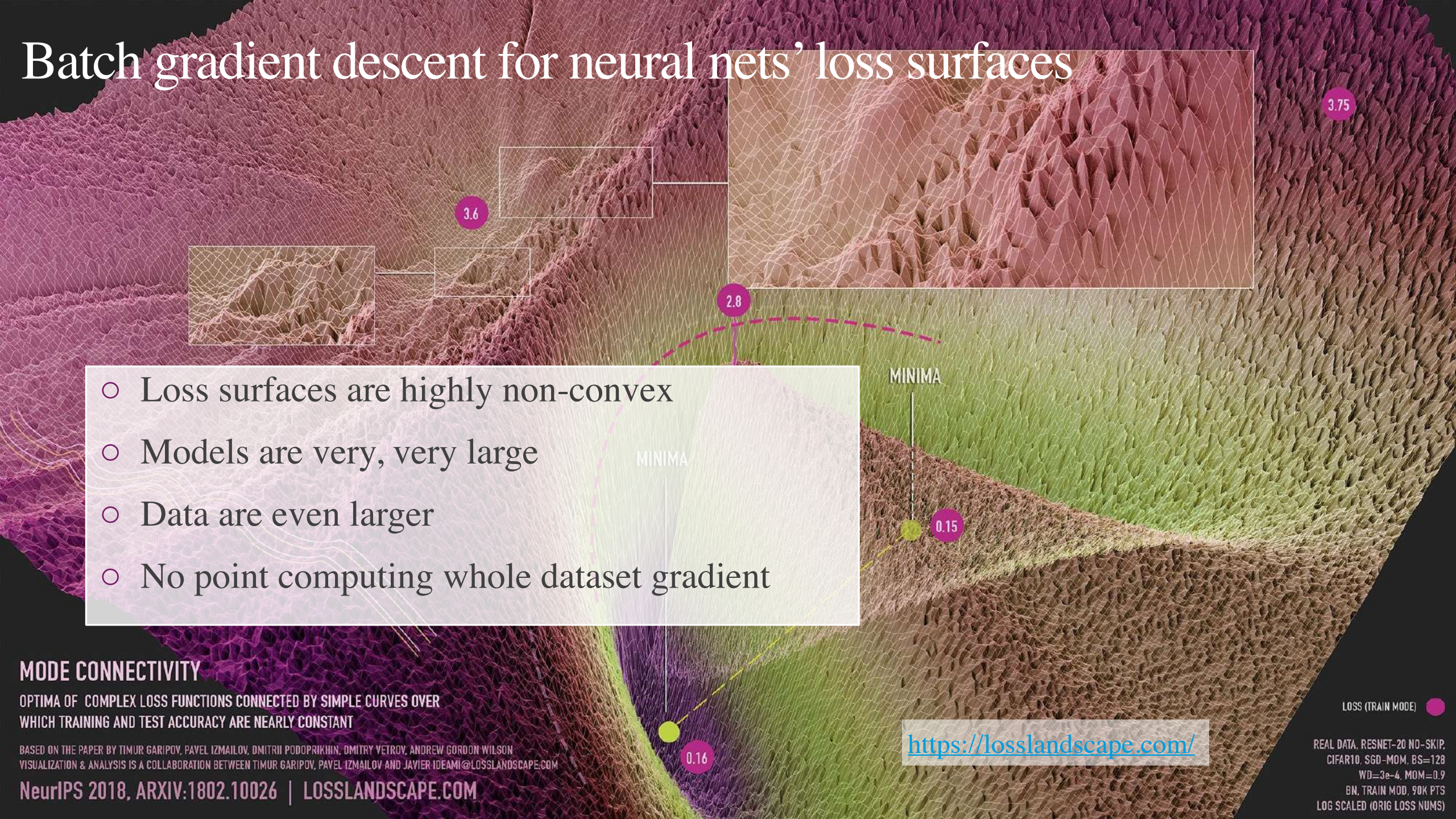
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{d\mathscr{L}}{d\boldsymbol{w}}$$

  - $\boldsymbol{w}^{(t)}$ are the parameters of our neural network at t
  - $\frac{\partial \mathscr{L}}{\partial \boldsymbol{w}}$ is the gradient of the loss w.r.t. the parameters "step direction"
  - η is the learning rate or "step-size"
- If $\frac{\partial \mathscr{L}}{\partial \boldsymbol{w}}$ is computed from whole dataset → gradient descent
- If $\frac{\partial \mathscr{L}}{\partial \boldsymbol{w}}$ is computed from a mini-batch of samples → mini

"SGD"

- If $\frac{\partial \mathscr{L}}{\partial \boldsymbol{w}}$ is computed from a *single* sample → Stochastic/online gradient descent
- *On non-convex landscapes no guarantee for global optimum*

Batch gradient descent for neural nets' loss surfaces

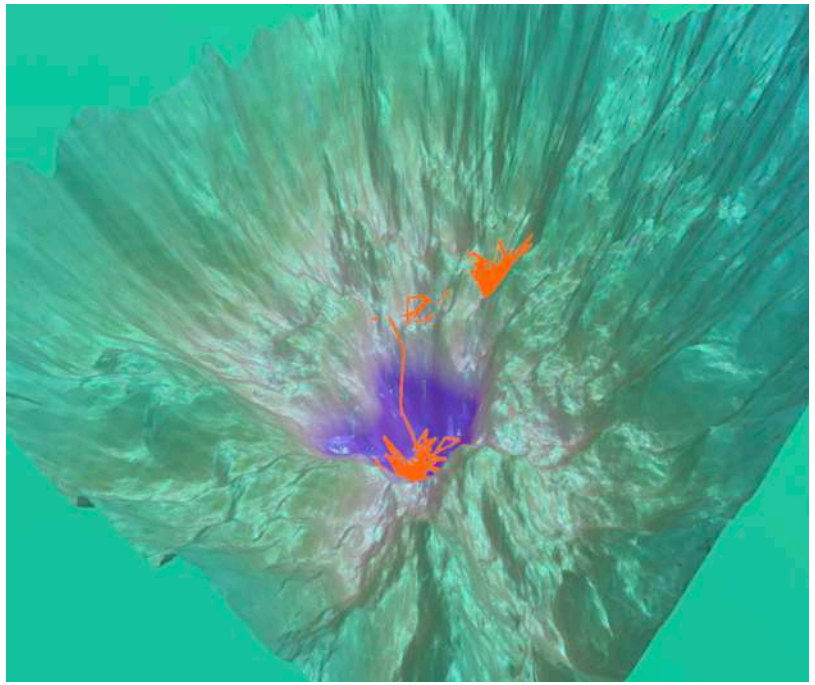


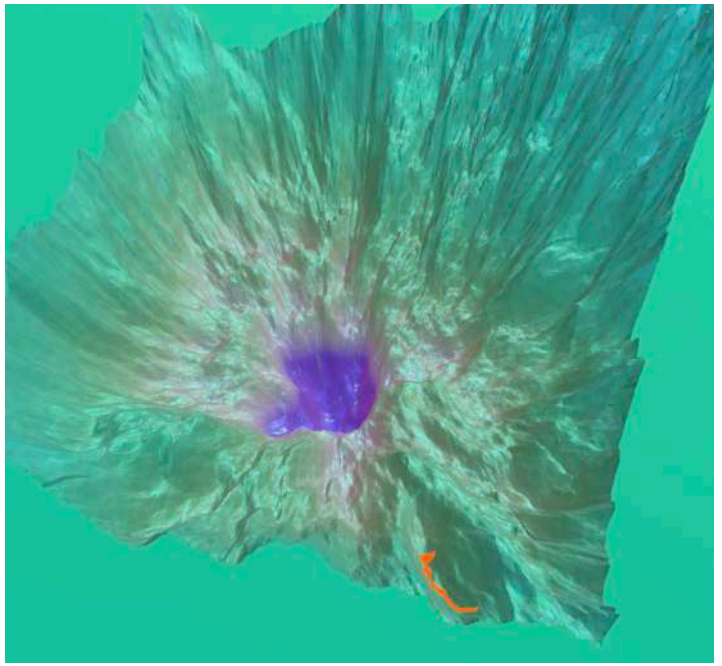
- Loss surfaces are highly non-convex
- Models are very, very large
- Data are even larger
- No point computing whole dataset gradient

https://losslandscape.com/

https://losslandscape.com/explorer

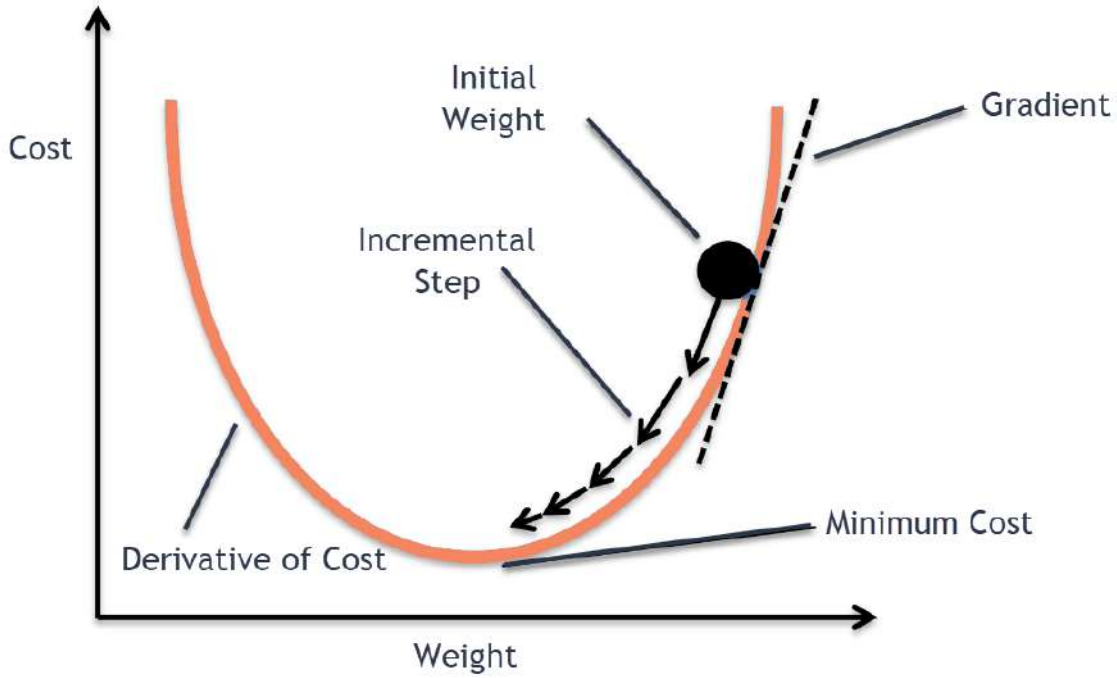
- Explore: play around with momentum, lr etc.





Gradient descent vs. Stochastic Gradient Descent

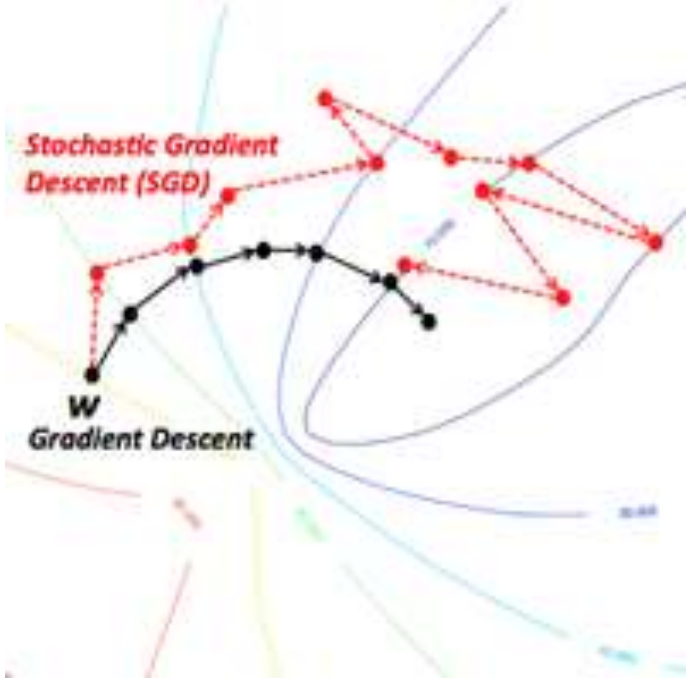
Gradient descent



Link

Calculate the gradients for the whole dataset to perform just one update

Stochastic gradient descent



Link

Perform a parameter update for *each* mini-batch

SGD with Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
  dw = compute_gradient(w)
  w -= learning_rate * dw
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  v = rho * v + dw
  w -= learning_rate * v
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

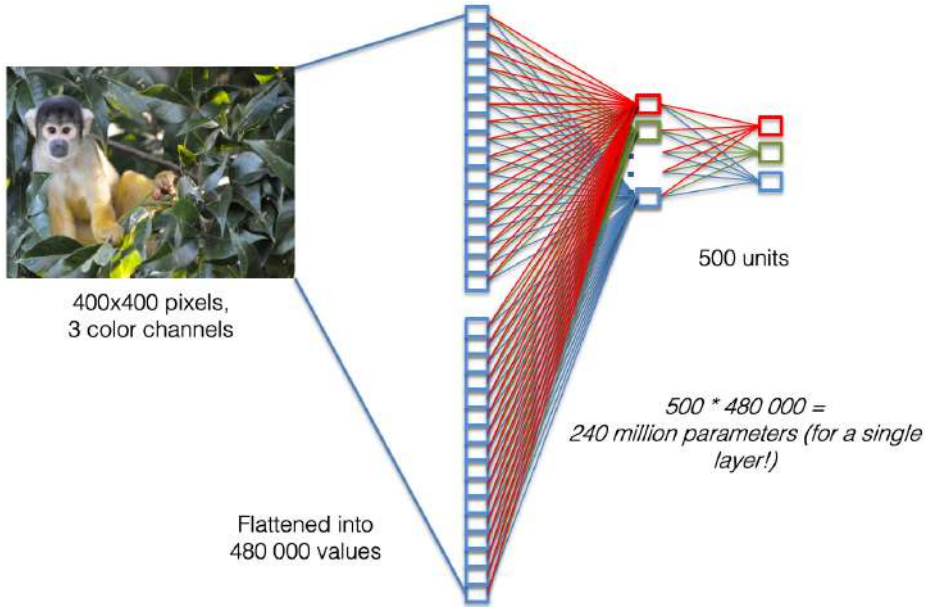
Batch size

- Large batch size
  - more accurate estimate of the gradient, but less than linear returns
- Extremely small batch size (BS) underutilizes hardware, why?
  - Re-use of computations (matrix multiplies)
- Power of 2 batch size achieves better runtime for GPUs
  - 32 ~ 8192
- Small batches can offer regularizing effect
  - add noise to the learning process
  - small batches requires very small learning rate, longer total runtime
- General rule: Batchsize ~ learning rate (ie double BS -> double the LR
  (*Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.* Goyal et al. 2017: Batchsize of 8K)
- Generally: use largest batch size that fits on GPU, but stick to powers of 2.

Convolutional Neural Networks

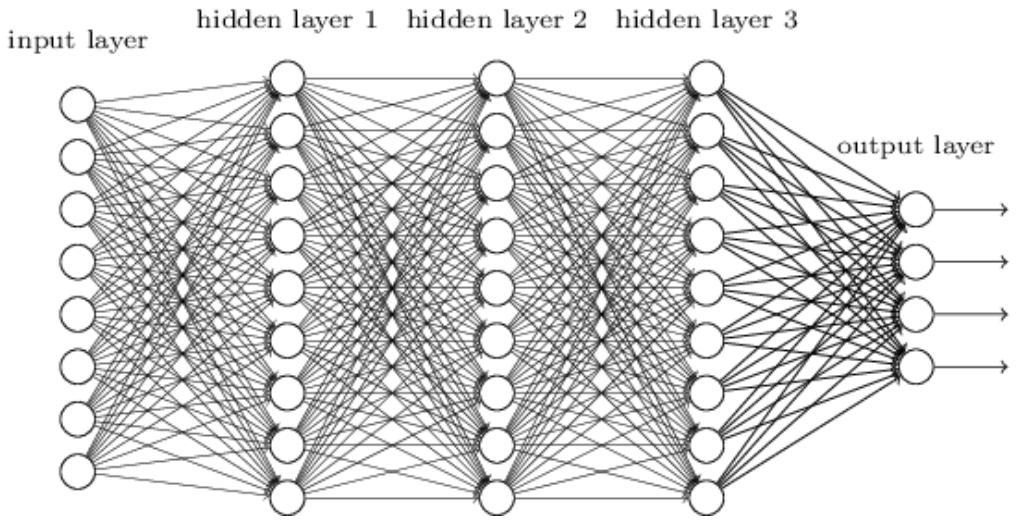
Multi-layer perceptrons

- An input image of 400x400 pixels and 3 colour channels (red, green, blue)
- flattened into a vector of 480 000 input values
- fed into an MLP with a single hidden layer with 500 hidden units
- 500*480 000 = 240 million parameters for a single layer



Multi-layer perceptrons

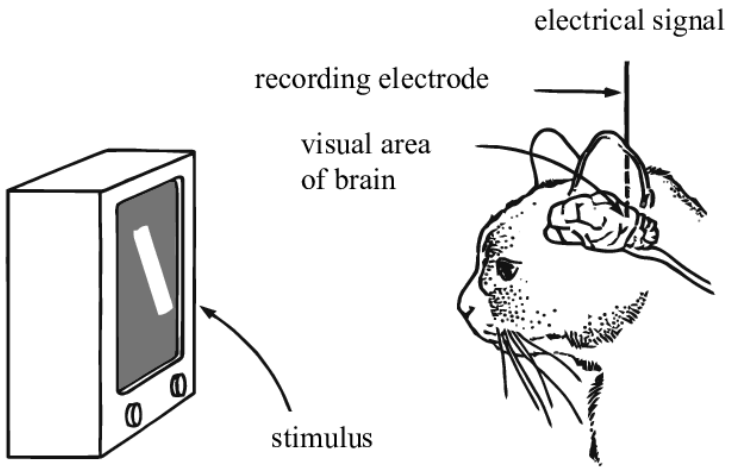
- From this fully-connected model, do we really need all the edges?
- Can some of these be "shared" (equal weight)?
- Can prior knowledge ("inductive biases") be incorporated into the design?



Consider an image

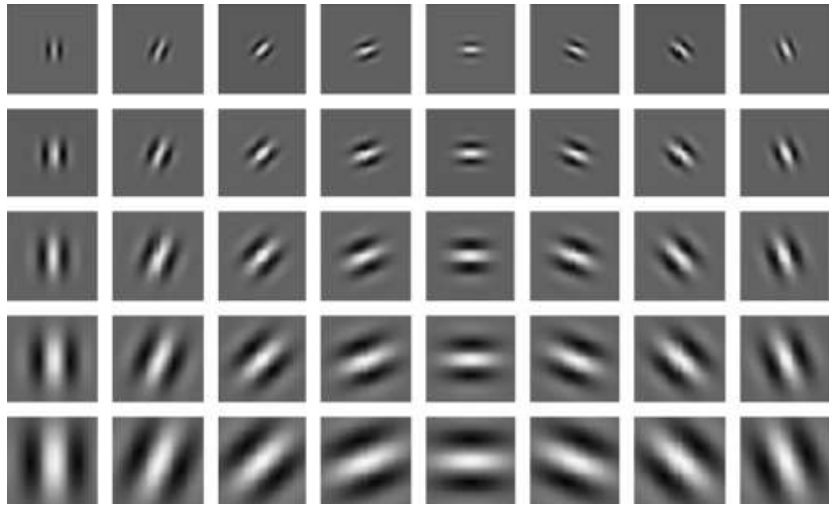
- Some patterns are much smaller than the whole image
- Can represent a small region with fewer parameters
- What about training a lot of such “small” detectors and each detector must be able to “move around”? (C.f. Barlow 1961)

Hubel and Wiesel: Nobel Prize for Physiology or Medicine in 1981



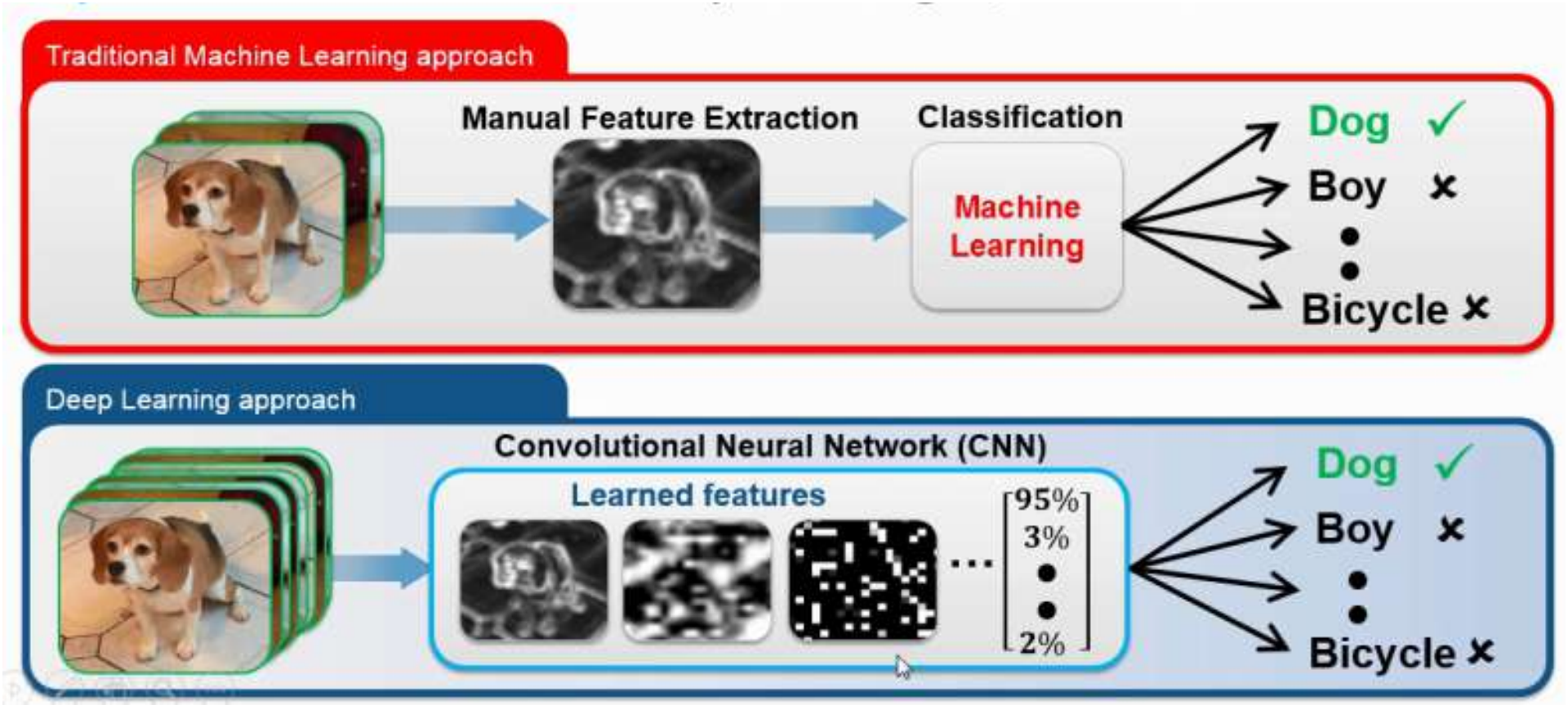
Filters, yes. How about learnable filters

- Hubel and Wiesel: Edge detectors in visual cortex
- See also “Grandmother cells” (next lecture)
- Handcrafted filters
  - Canny, Gabor filters
- Are they optimal for recognition?
- Can we learn optimal filters from our data instead?
- Are they going to resemble the handcrafted filters?



Gabor filters

Filters, yes. How about learnable filters



Manual features:
* Eg HOG, SIFT, GIST

Convolution for 2D images

- For a two-dimensional image I as our input, we want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Convolution is commutative, and we can equivalently write:

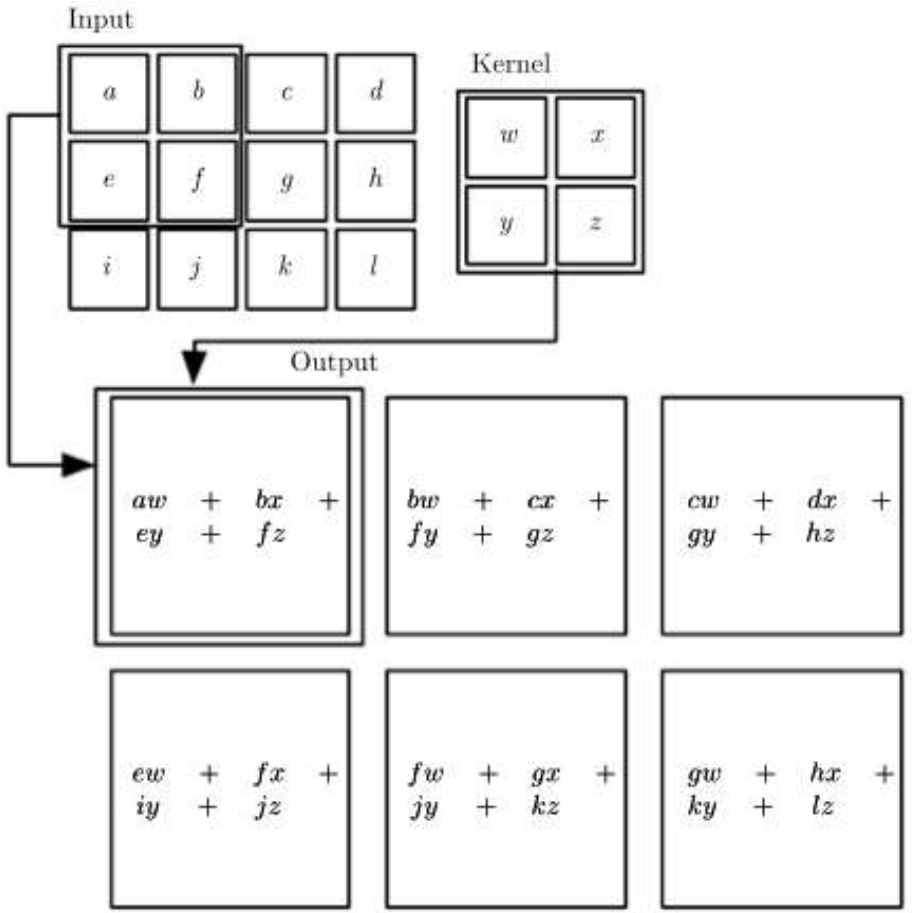
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

- Neural networks libraries implement cross-correlation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

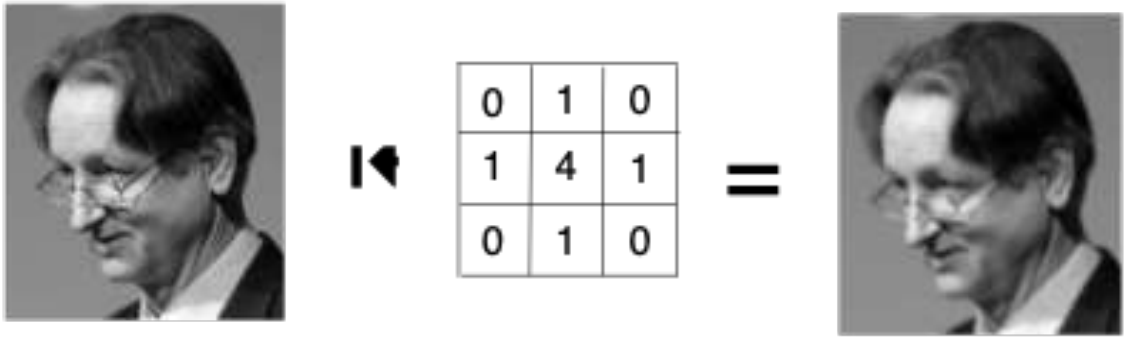
Convolution for 2D images

- An example of 2-D convolution

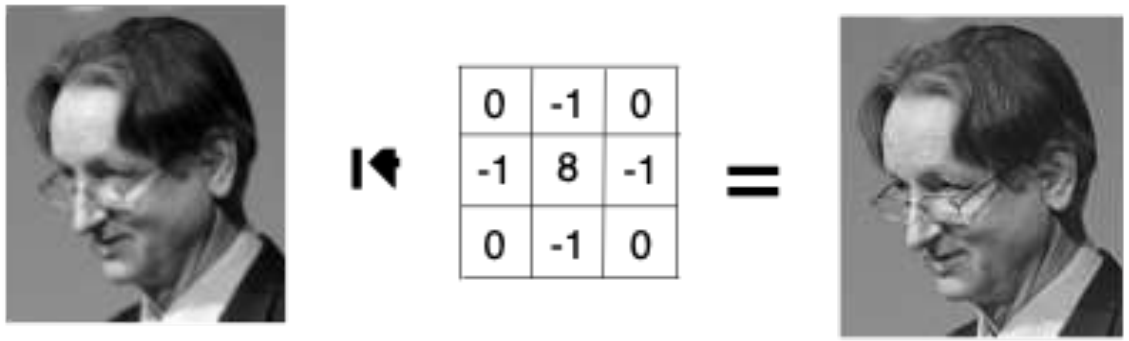


Examples

- We can blur images



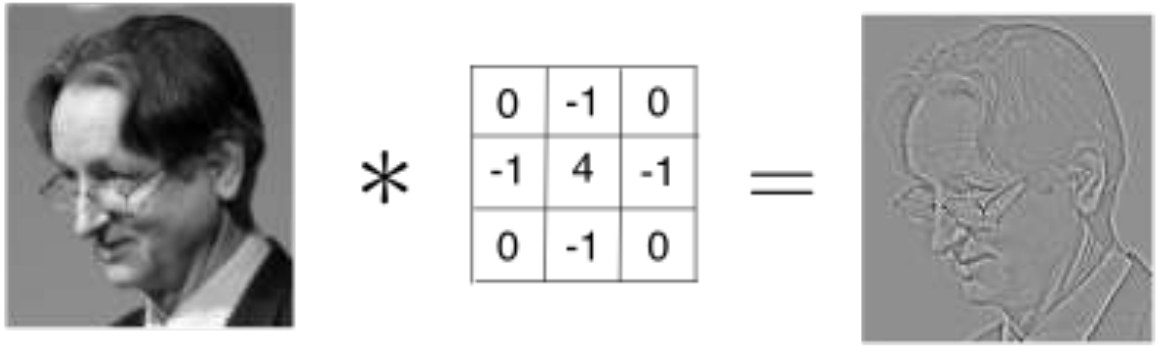
- We can sharpen images



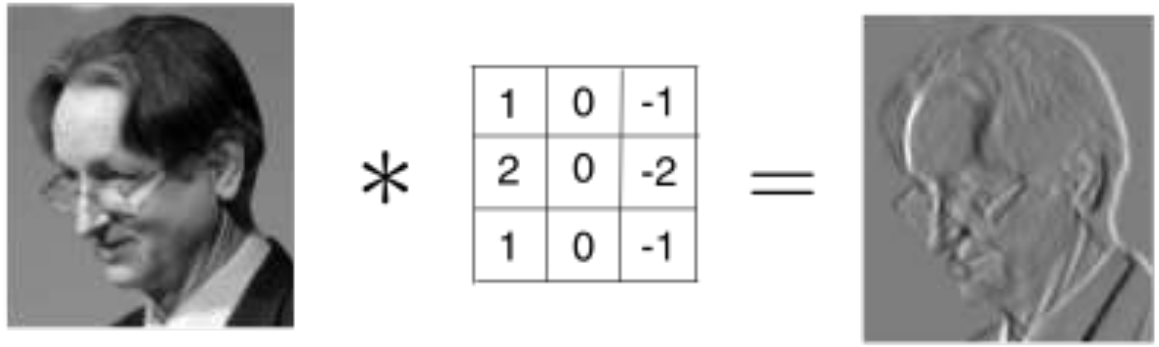
Picture credits from Roger Grosse

Examples

- We can detect edges



- Sobel filter



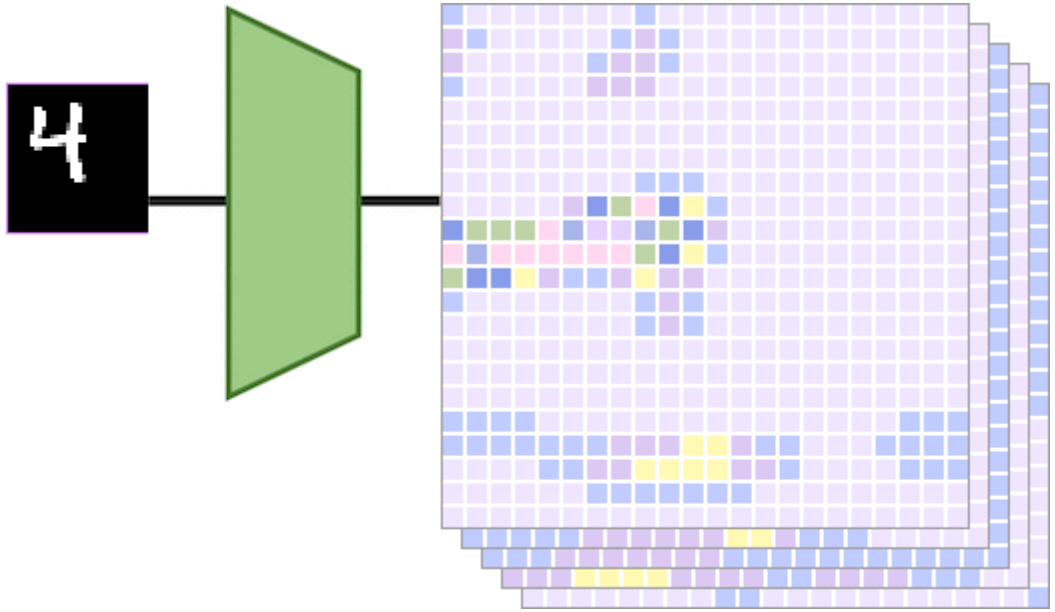
Picture credits from Roger Grosse

The motivation of convolutions

- Sparse interaction, or local connectivity
  - The receptive field of the neuron, or the filter size.
  - The connections are local in space (width and height), but always full in depth
  - A set of learnable filters
- Parameters sharing, the weights are tied
- Equivariant representation: same operation at different places

The motivation of convolutions

- Equivariant representation
  - parameter sharing causes the layer to have a property called *equivariance* to translation.*
  - A function is equivariant if the input changes, the output changes in the same way.
  - why do we want equivariance to translation?



A simple convolution: saves space!

1D example

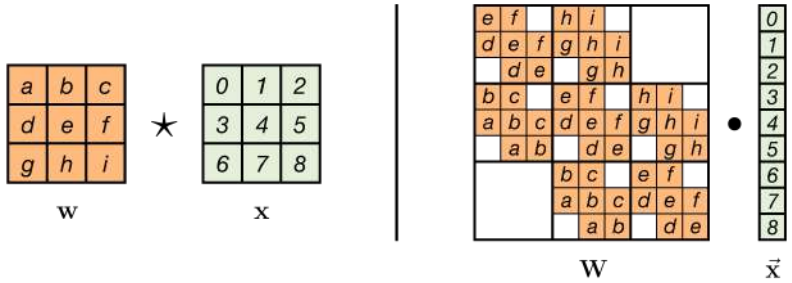
We can express convolution in terms of a matrix multiplication

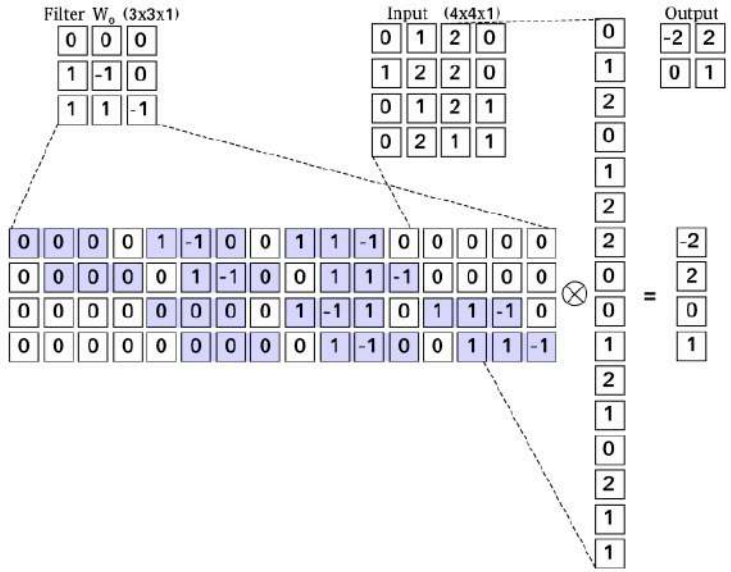
$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

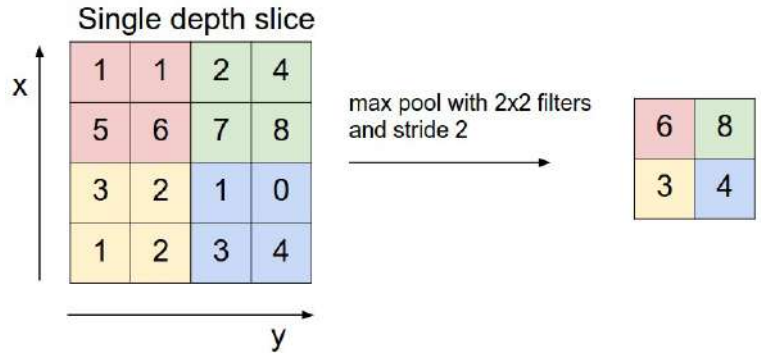
2D examples





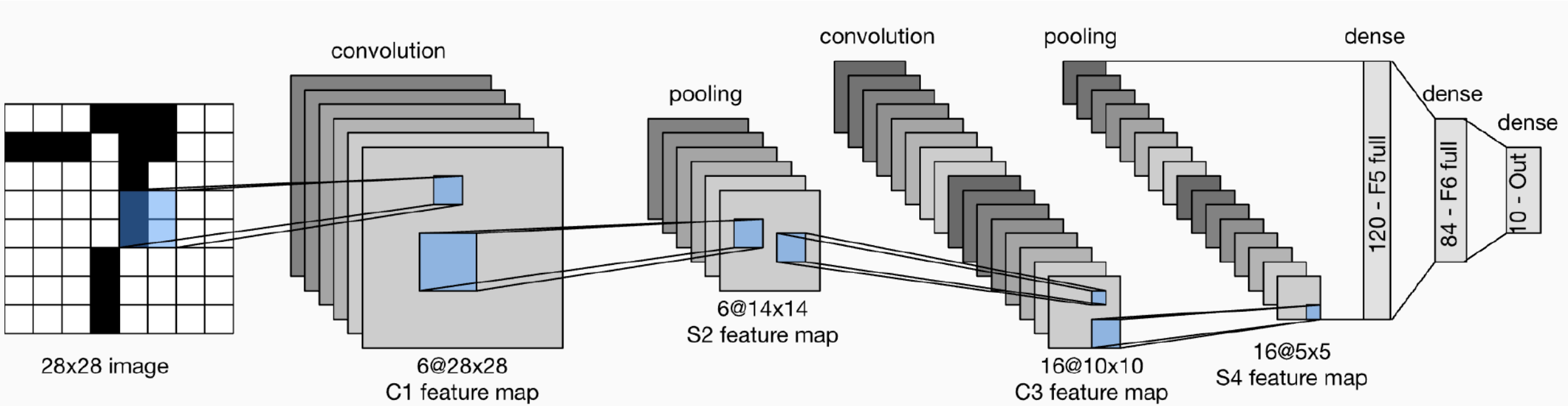
The pooling operations

- A pooling function
  - replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
  - e.g., max pooling operation reports the maximum output within a rectangular neighbourhood.
    - How could you implement circle-shaped pooling?
  - can reduce spatial size and thus improve the computational efficiency
  - pooling over spatial regions produces invariance to translation
  - pooling is essential for handling inputs of varying size.



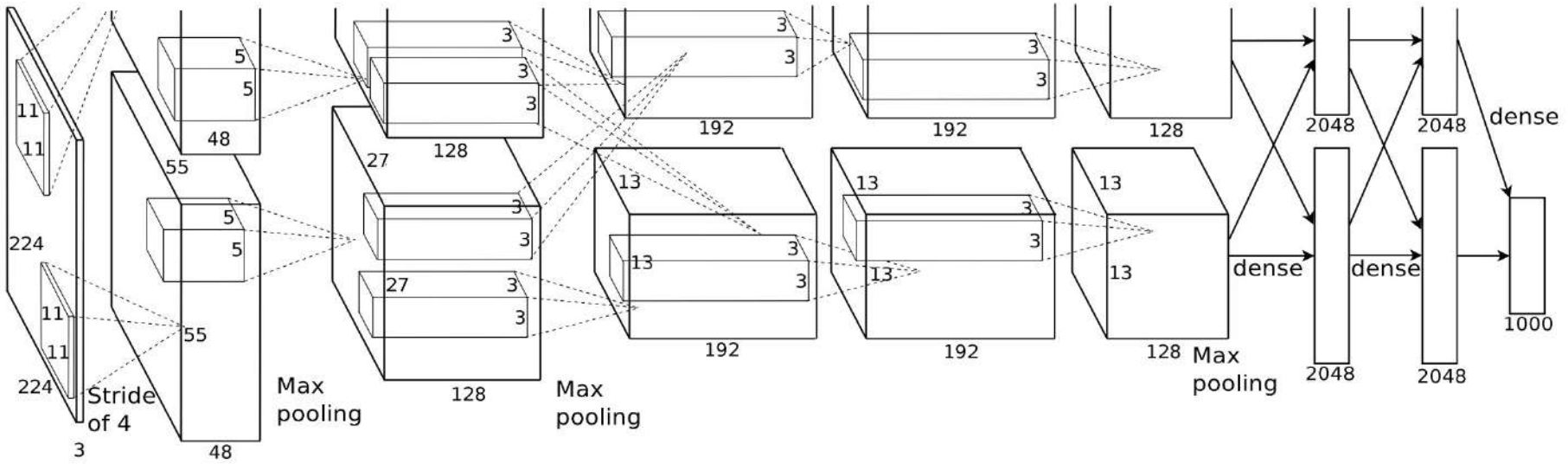
LeNet-5

- LeNet-5 was one of the earliest convolutional neural networks
- Yann LeCun et al. first applied the backpropagation algorithm
- Two convolutional layers and three fully-connected layers (--> 5-layer deep)



AlexNet: similar principles, but some extra engineering.

"AlexNet" — Won the ILSVRC2012 Challenge



Major breakthrough: 15.3% Top-5 error on ILSVRC2012
(Next best: 25.7%)

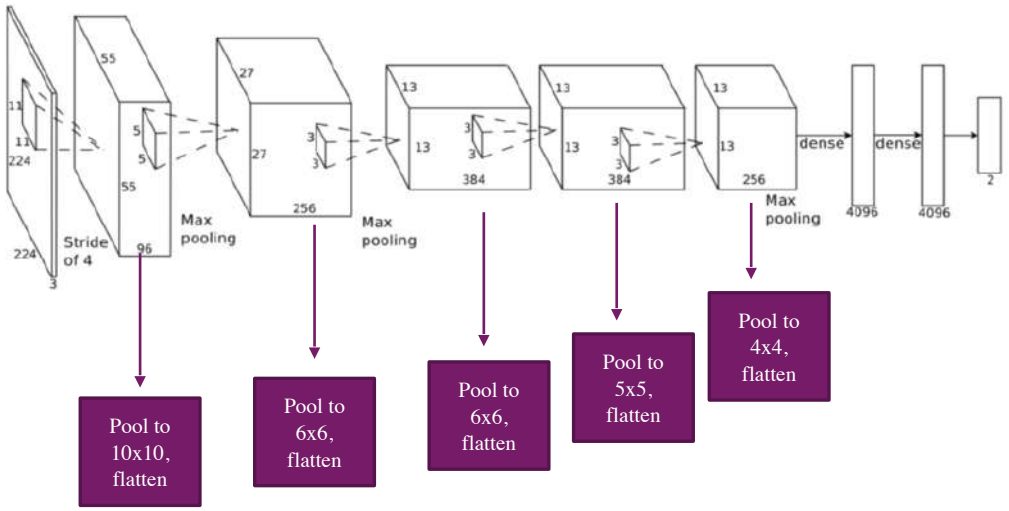
Fig es
bet ts
at t nd
the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

[Krizhevsky, Sutskever, Hinton. NIPS 2012]

How do these layers correspond to "semantics": numerical evaluation

Linear probing method: "exit" network at different locations & evaluate how good the model is at a given depth
How? Keep backbone (the whole AlexNet) frozen and only train a simple model, such as a linear layer

AlexNet



Method	ILSVRC-12 c1	c2	c3	c4	c5
ImageNet supervised, (Zhang et al., 2017)	19.3	36.3	44.2	48.3	50.5
Random, (Zhang et al., 2017)	11.6	17.1	16.9	16.3	14.1

Observations

- Performance gradually improves --> deeper representations are contain more semantic information of ILSVRC-12 (ImageNet)
- (Even randomly initialised networks extract some useful features)

yields features of roughly equal sizes [9600, 9216, 9600, 9600, 9216]

now train a linear layer on top of each to predict the 1K classes.
call these classifiers [c1, c2, c3, c4, c5]

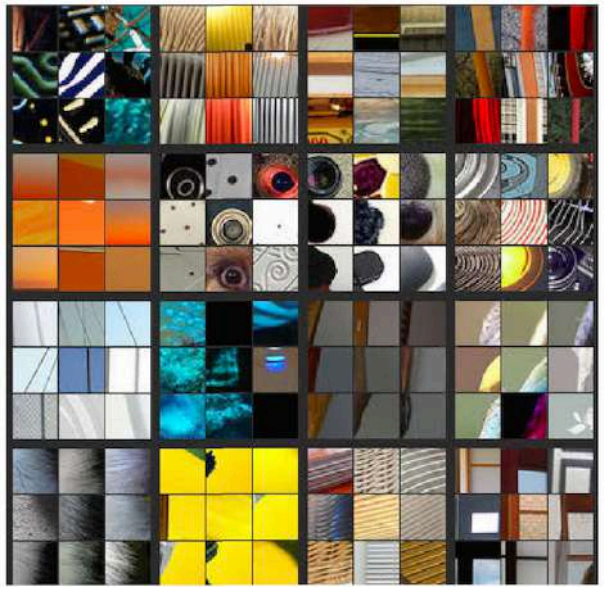
What do the different layers in a deep neural network learn

Layer 1

Filters

Image patches that activate this neuron maximally

Layer 2



(note that patches are bigger now, as receptive field size is larger)

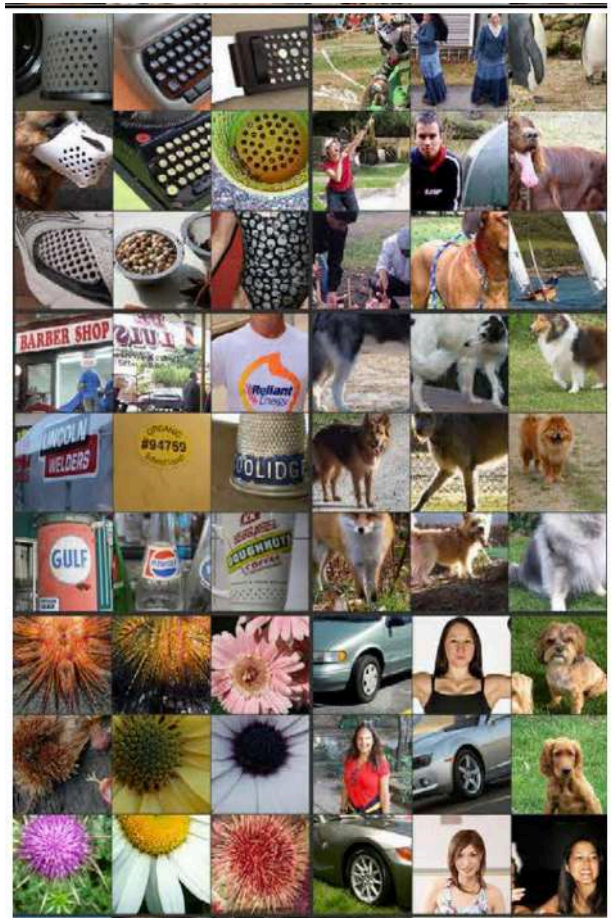
Layer 3

Visualizing and Understanding Convolutional Networks. Zeiler & Fergus. ECCV 2014

What do the different layers in a deep neural network learn

Layer 4

Layer 5



Actually same for self-supervised methods:

Last layer. (border color indicates true image class ID)

Visualizing and Understanding Convolutional Networks. Zeiler & Fergus. ECCV 2014

ImageNet 2012 winner: AlexNet

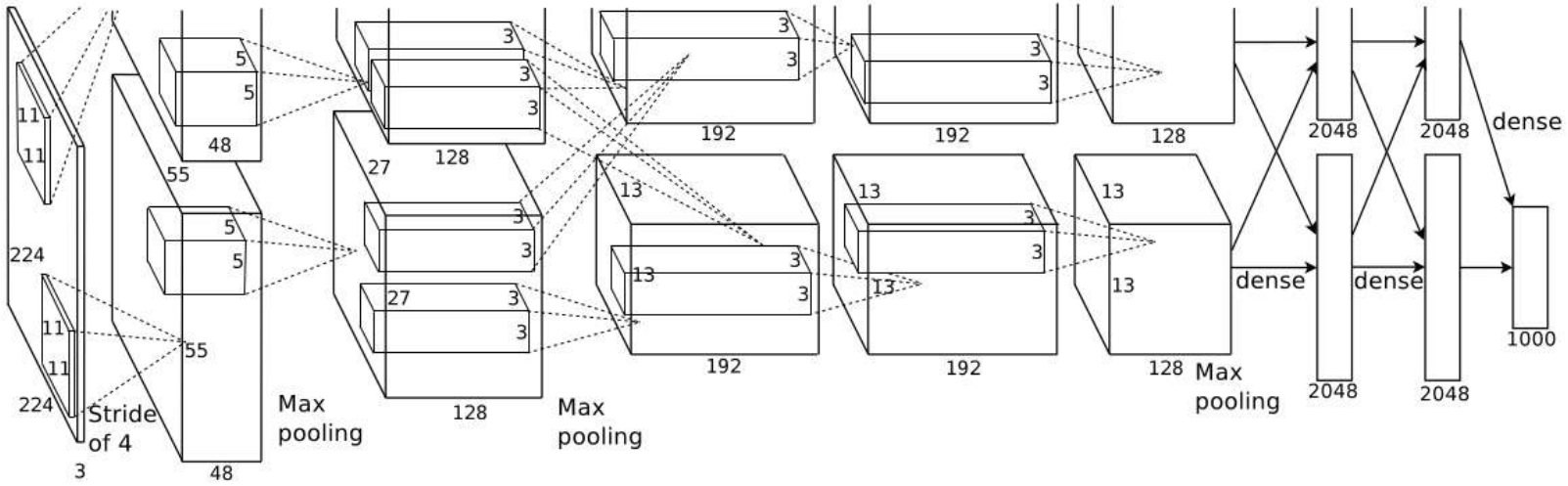
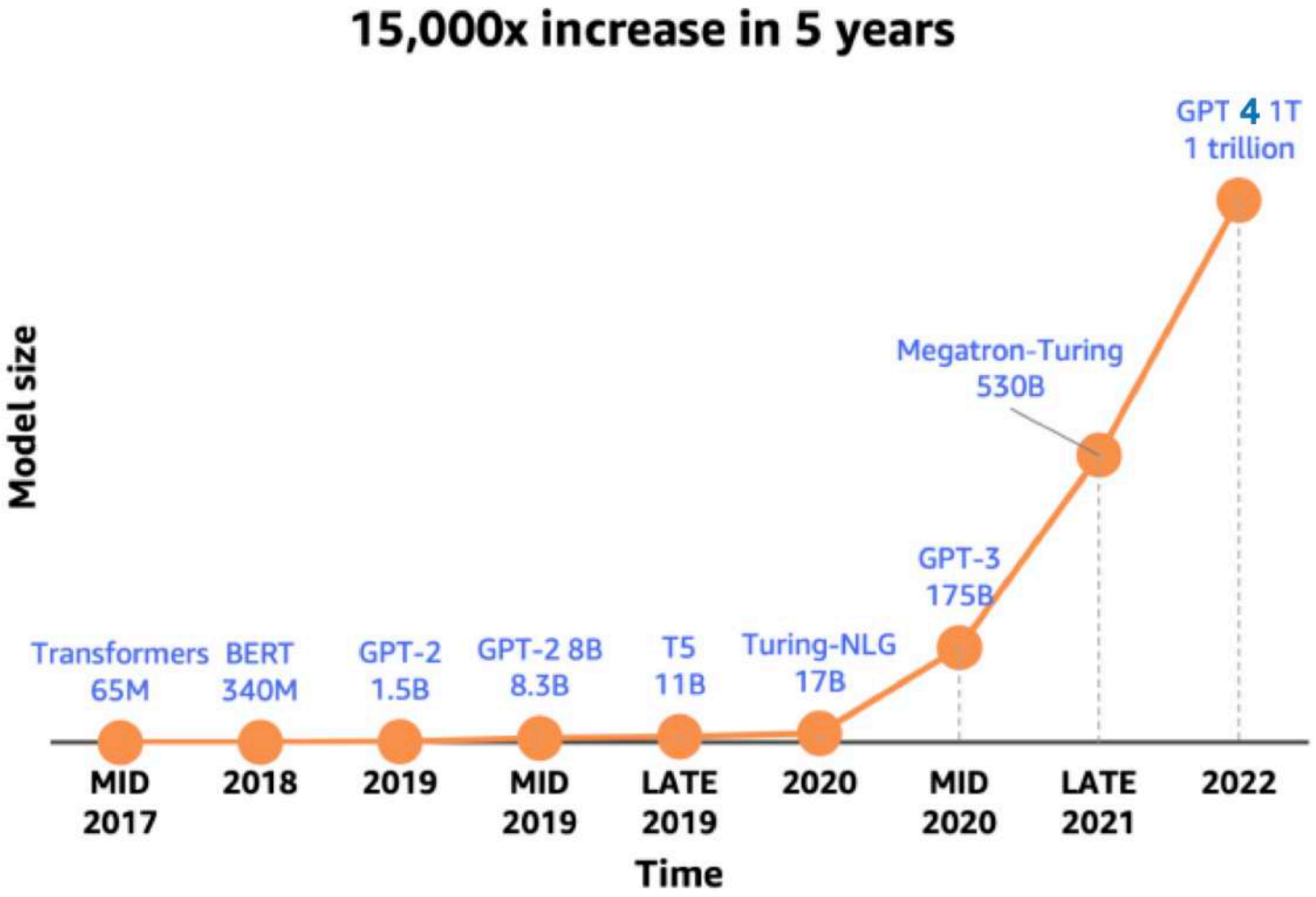


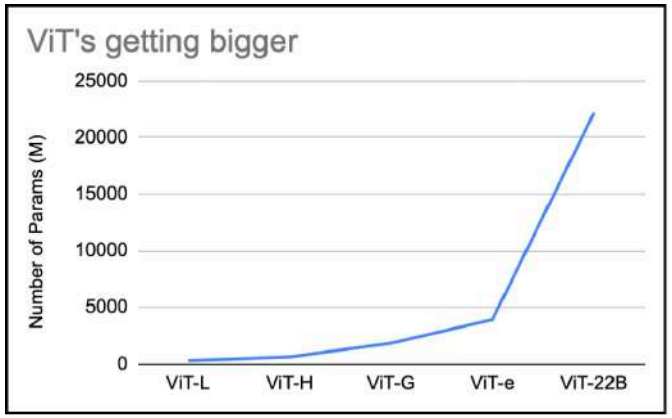
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

More weights than samples in the dataset!

Krizhevsky, Sutskever & Hinton, NeurIPS 2012

Current status:





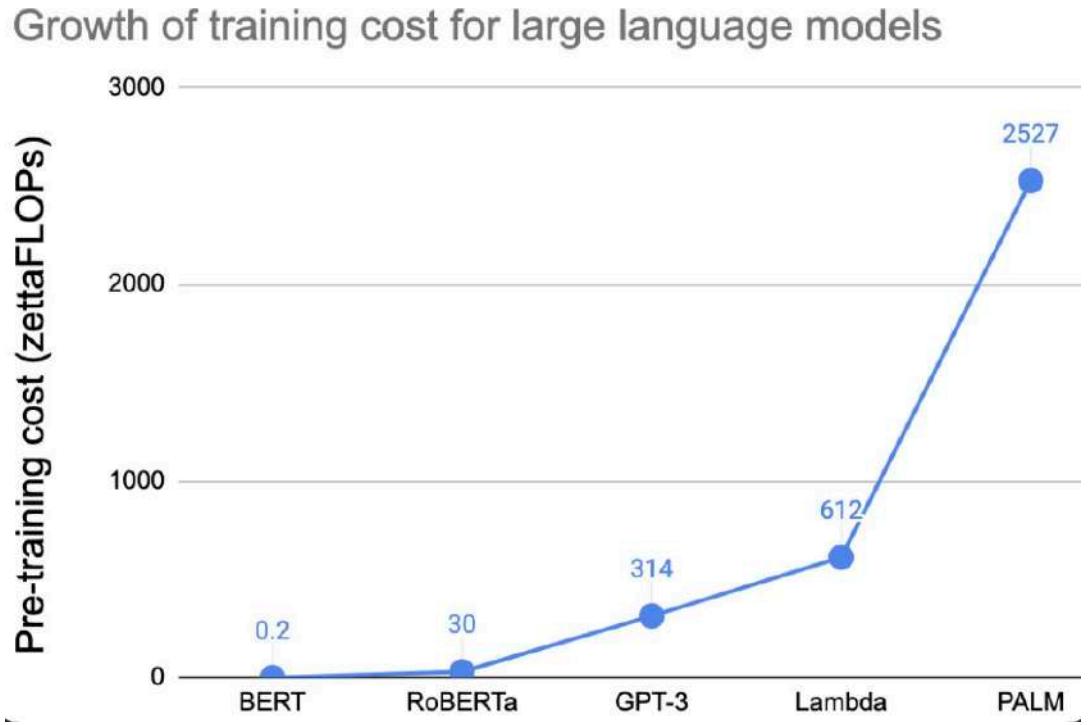
"Compute Requirements: ViT-22B was trained on 1024 TPU V4 chips [..]"

How? (slightly exaggerated)

@tipsybartender

The current scaling of models.

- BERT model (354M parameters) ~ now $2K
- RoBERTa (1000 GPUs for a week) ~ now $350K
- GPT-3 (175B parameters, 1500 GPUs for 2 months) ~ $3M
- …
- PaLM
  - 6144 TPUs, ~$25M
  - 3.2 million kWh ~~1000 Households for a year



Source: https://twitter.com/tomgoldsteincs/status/1544370726119112704

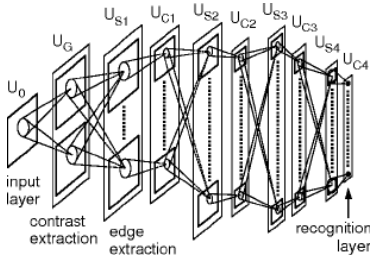
Self-supervised Representation Learning

University of Amsterdam

The field of AI has made rapid progress, the crucial fuel is data

Algorithms

Deep neural networks



Hardware

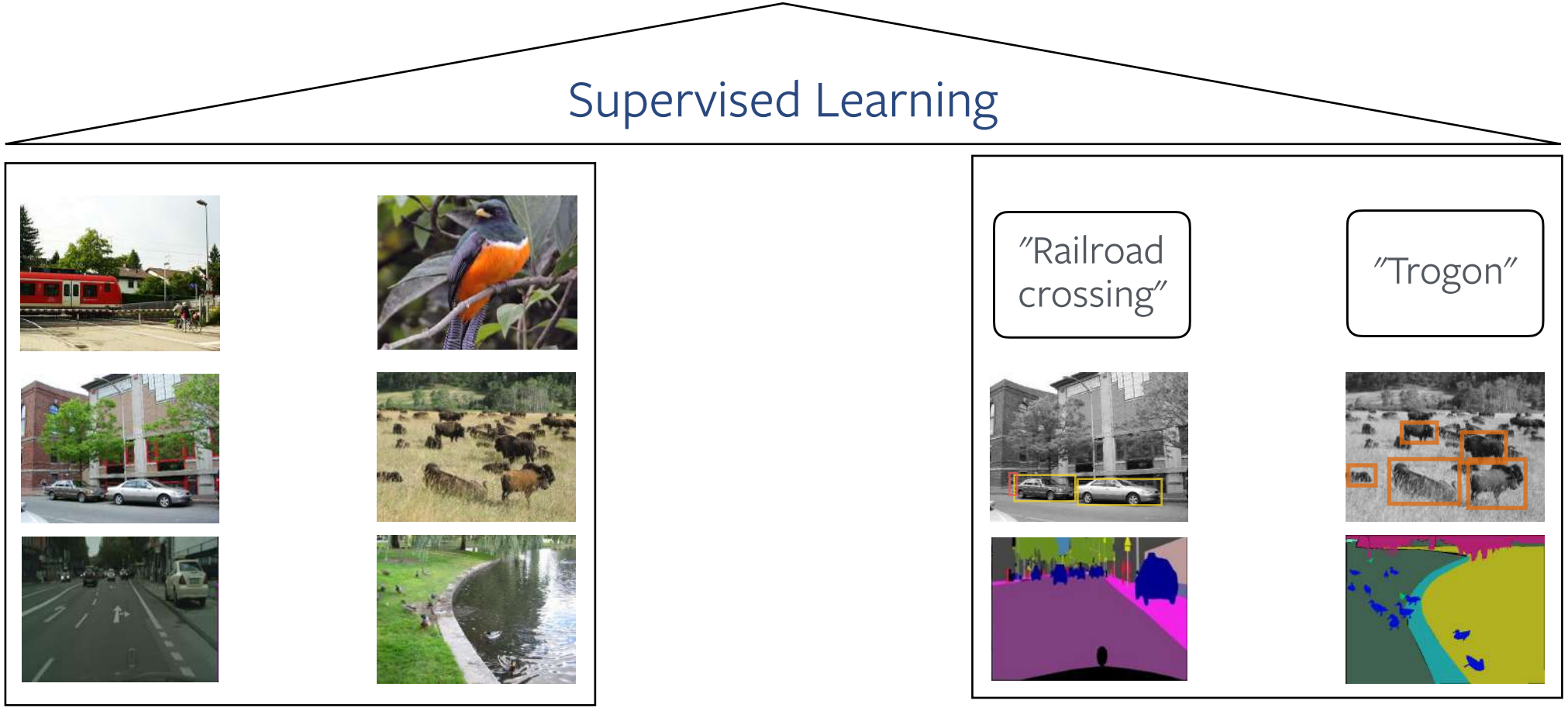
GPUs

Data

Large scale datasets

Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Fukushima, K., Biol. Cybernetics 1980
Object Recognition with Gradient-Based Learning. LeCun et al. Shape, Contour and Grouping in Computer Vision 1999
ImageNet: A Large-Scale Hierarchical Image Database. Deng, et al. CVPR, 2009.
ImageNet Classification with Deep Convolutional Neural Networks., Krizhevsky et al., NeurIPS 2012

Manual annotations for the data are limiting.

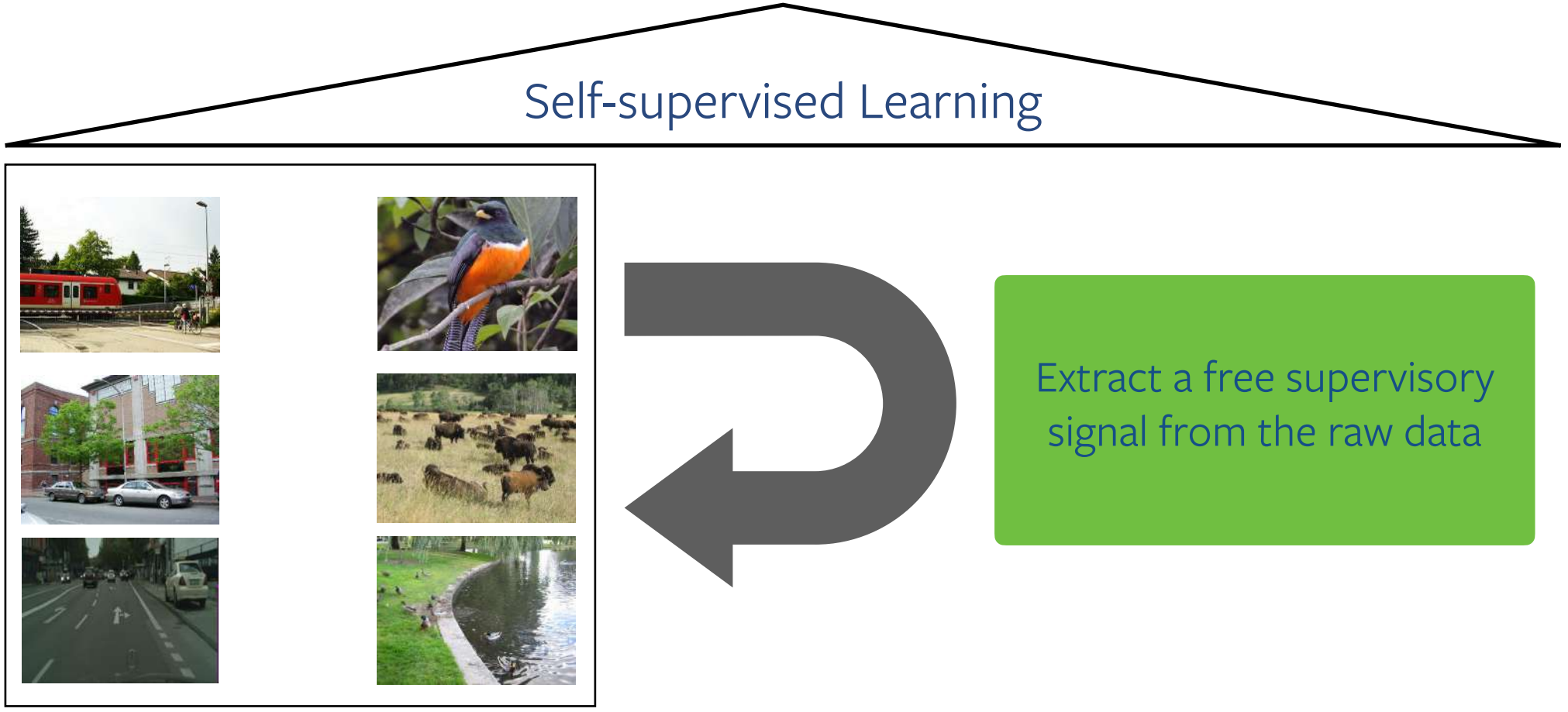


Images are often cheap

But manual annotations are expensive:
e.g. 30min per image / requiring experts

ImageNet: A Large-Scale Hierarchical Image Database. Dong et al. CVPR 2009
The Cityscapes Dataset for Semantic Urban Scene Understanding. Cordts et al. CVPR 2016
Scene parsing through ADE20K dataset. Zhou et al. CVPR 2017.

Self-supervised Learning replaces the need for labels & annotations.

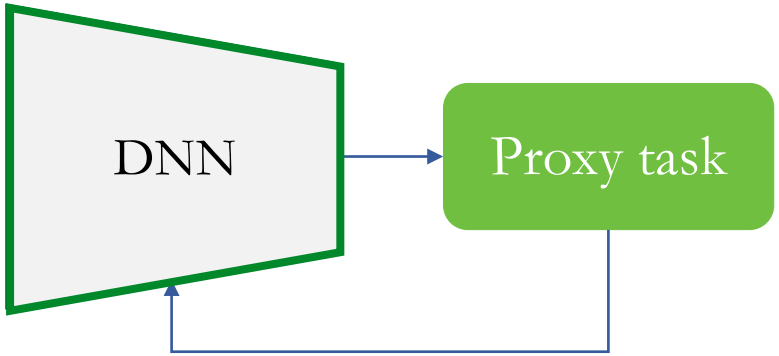


ImageNet: A Large-Scale Hierarchical Image Database. Dong et al. CVPR 2009
The Cityscapes Dataset for Semantic Urban Scene Understanding. Cordts et al. CVPR 2016
Scene parsing through ADE20K dataset. Zhou et al. CVPR 2017.

General procedure of self-supervised learning.

Phase 1: Pretraining

Unlabelled data
+ transformations



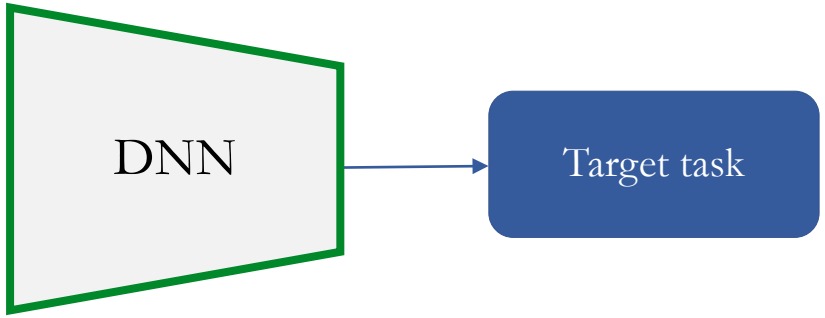
Gradient

For example:

1) rotate image by multiple of 90 degrees
2) have network predict the rotation

Phase 2: Downstream tasks

(Sparse) labeled data

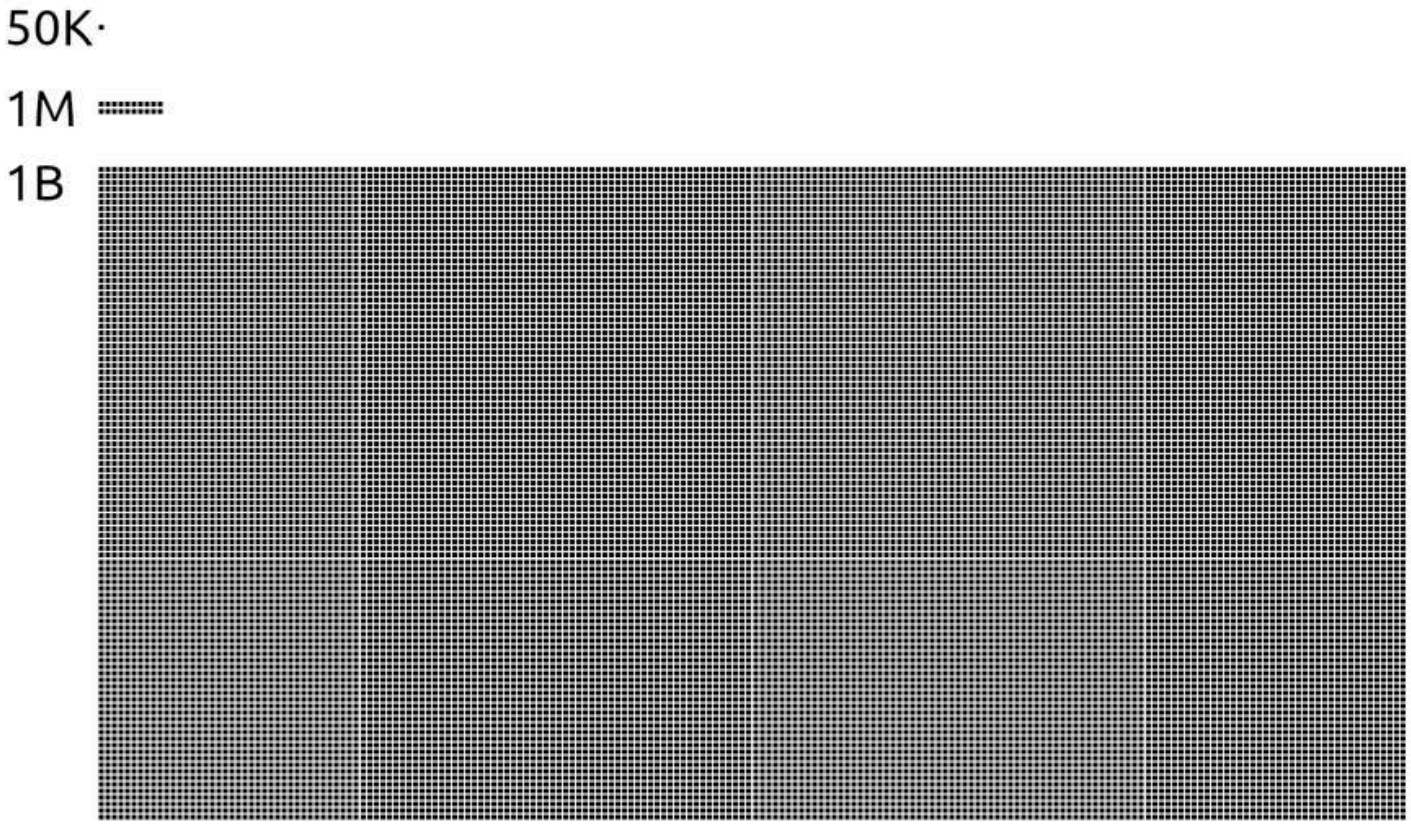


Typically has less data

Why do we want to do self-supervised learning?

Reason 1: Scalability and GPT as proof-of-concept

Instagram: >50B images



Annotation is expensive, yet datasets keep getting bigger.

Reason 2: Constantly changing domains

Unclear when & what to relabel. Again, large costs just to "keep up".

Reason 3: Ambiguity of labels and captions

"A house"?

"A boat"?

Example from Flickr30k

A hot, blond girl getting criticized by her boss.

Problematic captions

Labels are ambiguous at best, discriminating and bias-propagating at worst.
Do we really wish to provide our models with these priors?

Crawford & Paglen
Stereotyping and Bias in the Flickr30K Dataset. Miltenburg. MMC. 2016.

Especially videos open exciting new directions

Visual development for AI

"Get" physics

Embodied AI

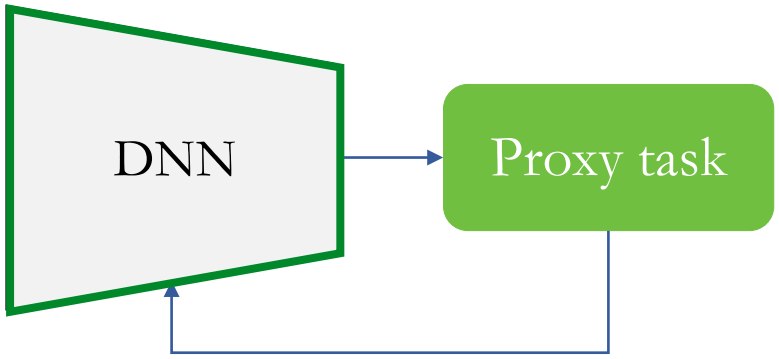
Bonus: *insane* scale:

Part: ″What″?

General procedure of self-supervised learning.

Phase 1: Pretraining

Unlabelled data
+ transformations



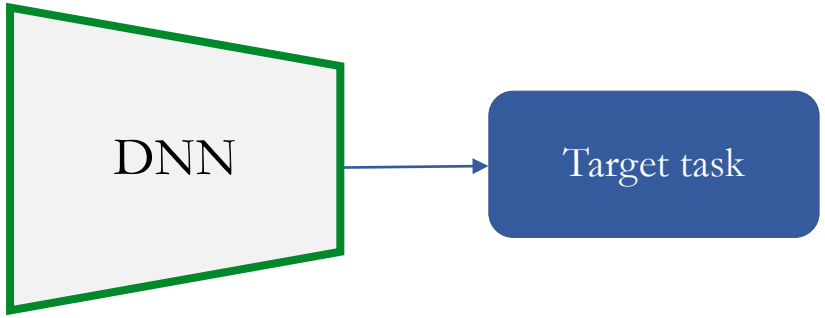
Gradient

Types:

- Geometry based
- Clustering
- Contrastive
- Generative (partial/full)
- (more)

Phase 2: Downstream tasks

(Sparse) labeled data



Types:

- Limited fine-tuning (e.g. linear layer)
- Finetuning (w/ full or fraction of dataset)

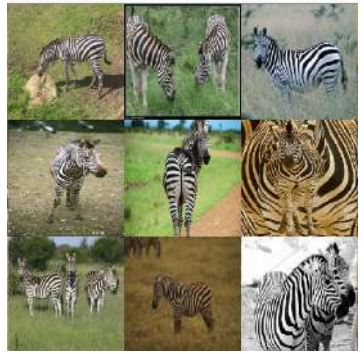
Generalizeable representations and `free labelling' of images
or
using optimal-transport for self-supervised representation learning

Self-labelling via simultaneous clustering and representation learning (ICLR'20 spotlight)

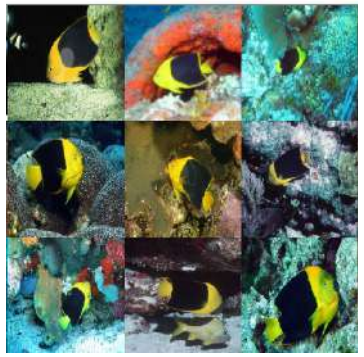
YUKI M. ASANO, CHRISTIAN RUPPRECHT, ANDREA VEDALDI

Goal: Discover visual concepts without annotations.

(above) x 50 =1.2M images



concept "A"



concept "Z"

ImageNet: A Large-Scale Hierarchical Image Database. Deng, et al. CVPR, 2009.

The key to image understanding is separating meaning from appearance.

Original

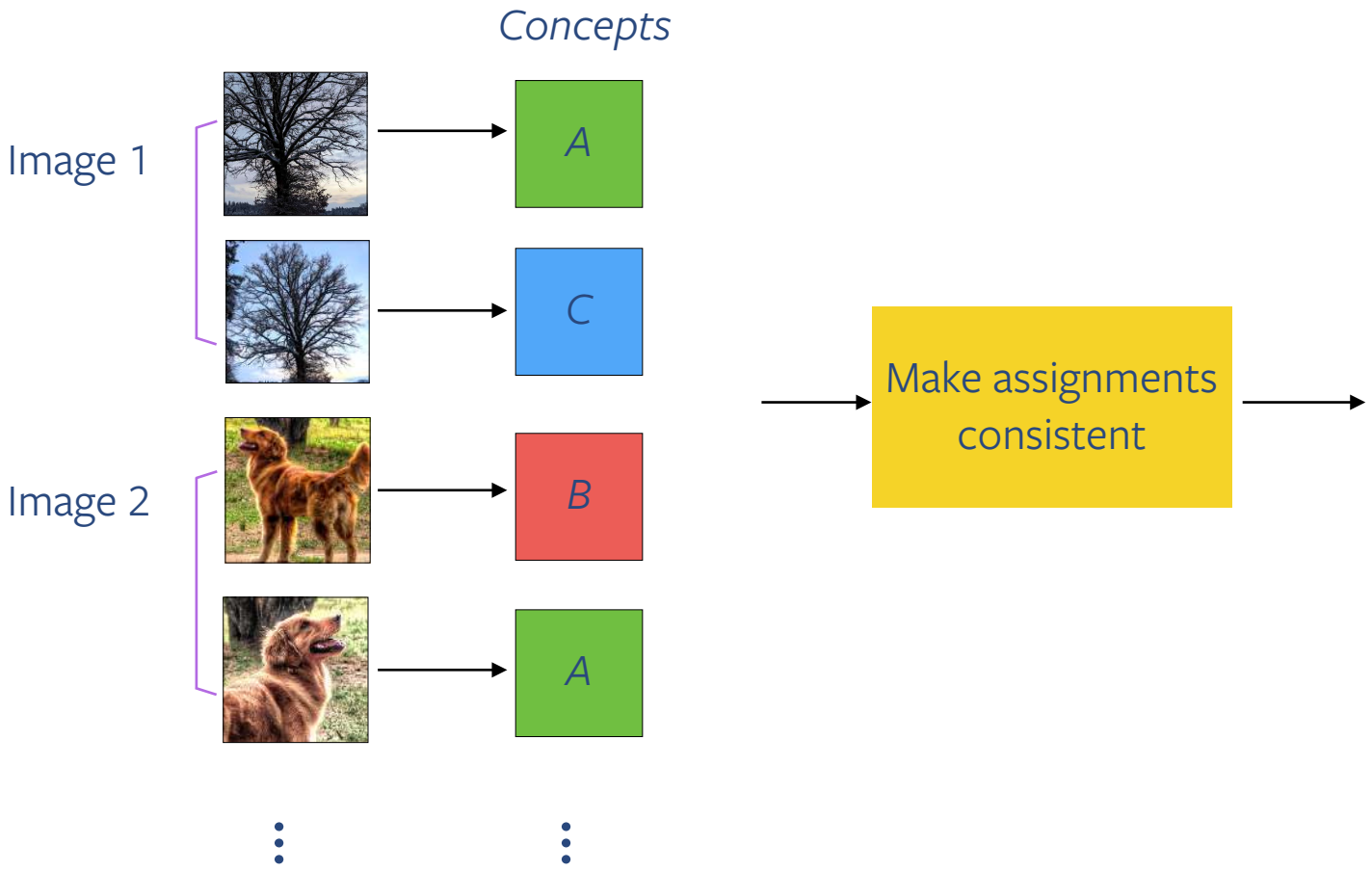
Different lighting

Mirrored

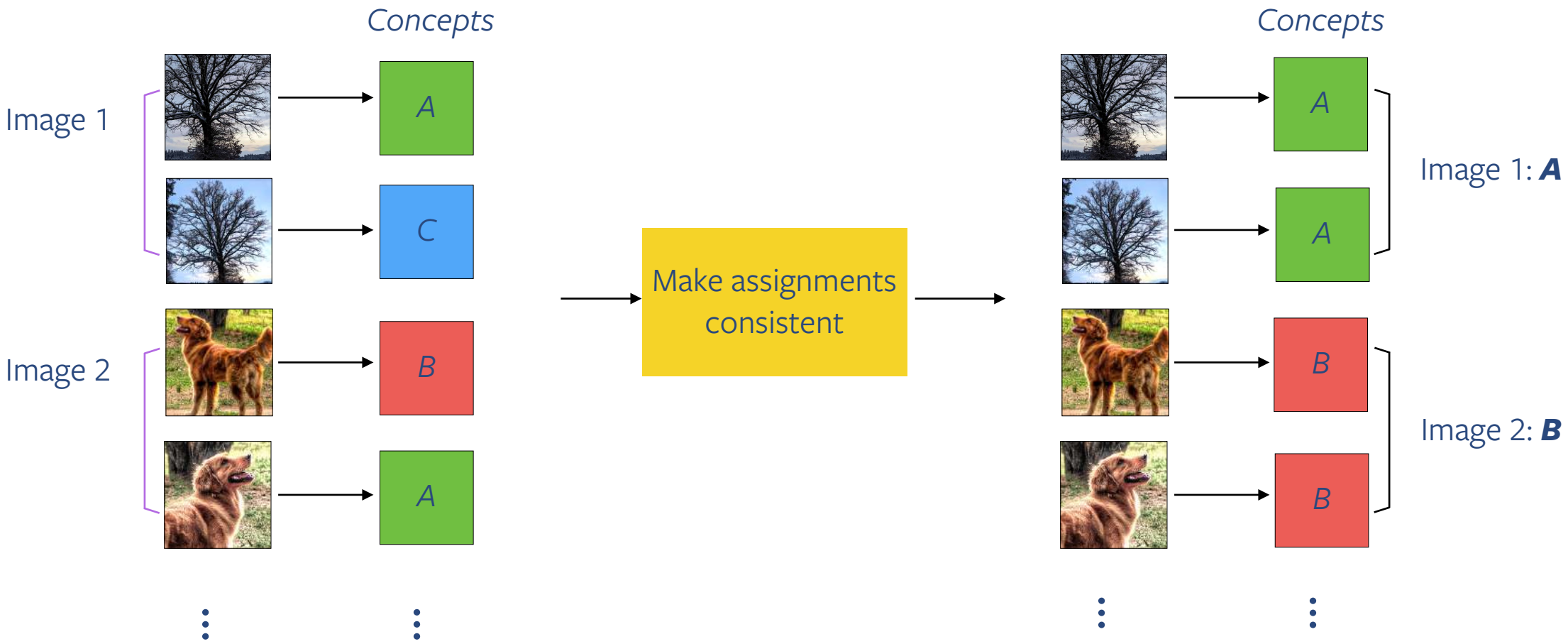
Different zoom

Augmentations

Our work applies the idea of augmentation invariance to assign concepts.



Our work applies the idea of transformation invariance to assign concepts.



How can we optimize the labels and make assignments consistent?

If we had ground-truth labels

$$\min_{\Phi} L(y, \Phi),$$

$$\text{where } L(y, \Phi) = \frac{1}{N} \sum_{i=1}^{N} \log p(y_i \,|\, \mathbf{x}_i, \Phi)$$

- L is the loss (cost) function
- Φ is the deep neural network model
- y are the labels

Our novel contribution *without* ground-truth

Solution sketch:

1. Represent via an assignment table q and optimize:

$$L(q, \Phi) = \frac{1}{N} \sum_{i=1}^{N} \sum_{y} q(y \,|\, \mathbf{x}_i) \log p(y \,|\, \mathbf{x}_i, \Phi)$$

But: The trivial solution for q is to set all labels to be the same

2. Use pseudolabels an equal number of times:

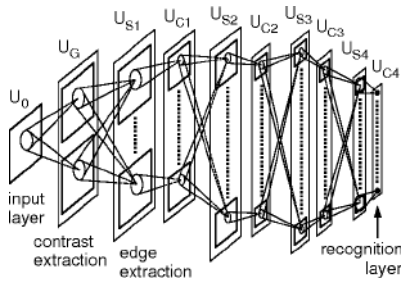
3. Pose as approximate optimal transport:

$$\min_{q,\Phi} L(q, \Phi) \quad \text{s.t.} \quad \sum_{i=1}^{N} q(y \,|\, \mathbf{x}_i) = \frac{N}{K},$$

Self-labelling via simultaneous clustering and representation learning. Asano et al. ICLR 2020
Sinkhorn distances: Lightspeed computation of optimal transport. Cuturi. NeurIPS 2013

Algorithm

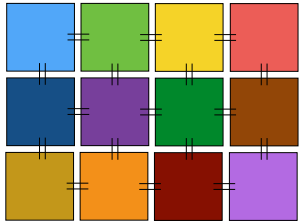
Model Φ



Optimal labelling

Cross entropy training with augmentations

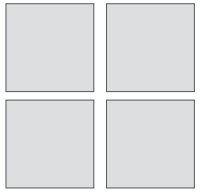
Label assignments q



Our method applied on 1.2 million images: Examples

1.2M images

Legend:

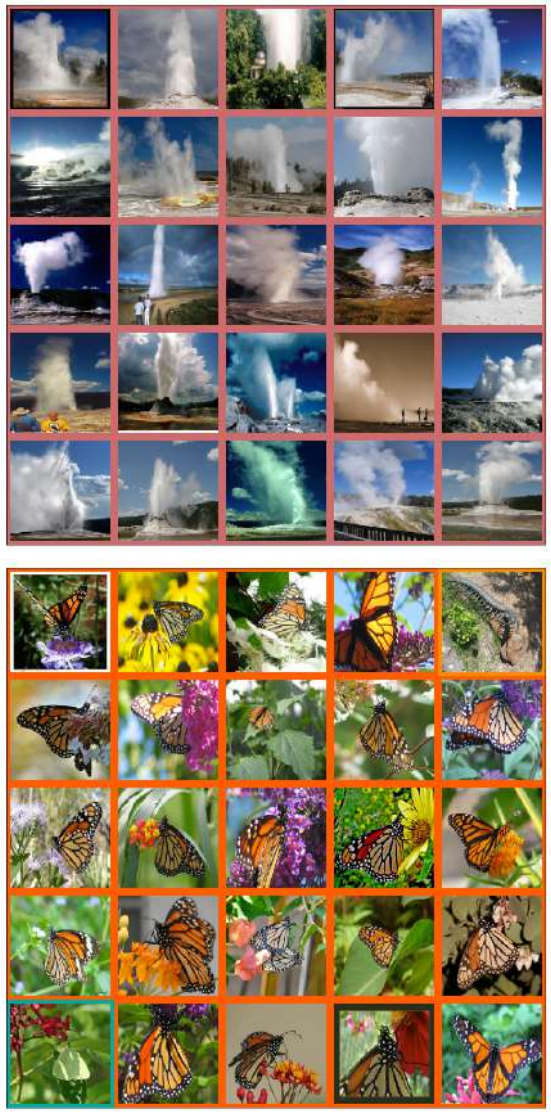


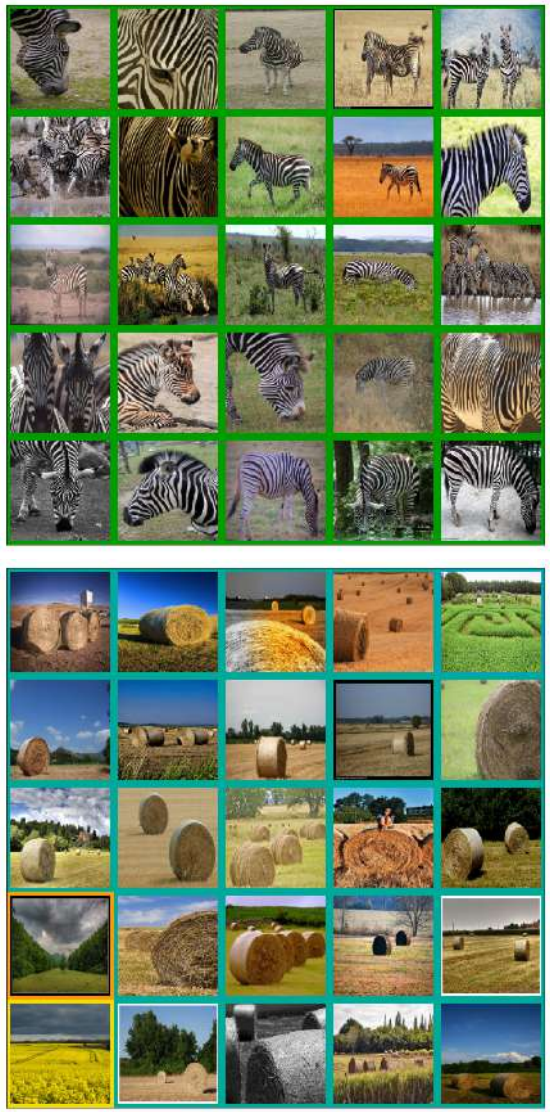
Concept

Self-labelling via simultaneous clustering and representation learning. Asano et al. ICLR 2020

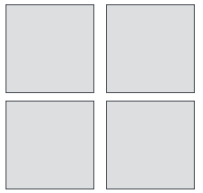
Automatically discovered concepts match manual annotation.

1.2M images





Legend:



Concept

border color

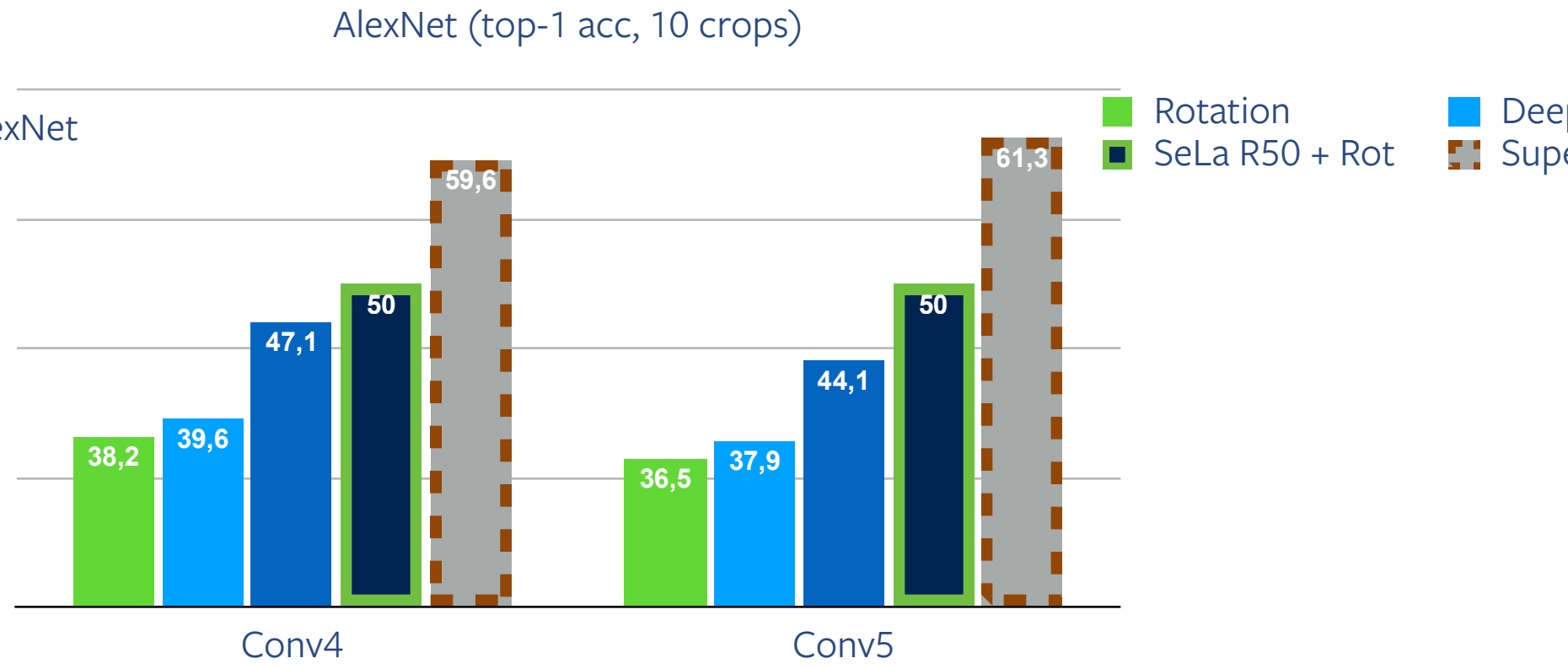
Manually annotated label (>2.5y of work)

Explore all clusters:

Self-labelling via simultaneous clustering and representation learning. Asano et al. ICLR 2020
ImageNet: A Large-Scale Hierarchical Image Database. Deng et al. CVPR, 2009.

AlexNet, ImageNet linear probes

- Big jump on DeepCluster
- SoTA or close to SoTA for AlexNet



Unsupervised representation learning by predicting image rotations. Gidaris et al. ICLR,2018.
Self-supervised representation learning by rotation feature decoupling. Feng et al. CVPR 2019.

More recently...

Method	Top-1 2x224	Top-1 2x160+4x96	Δ
Supervised	76.5	76.0	−0.5
Contrastive-instance approaches			
SimCLR	68.2	70.6	+2.4
Clustering-based approaches			
SeLa-v2	67.2	71.8	+4.6
DeepCluster-v2	70.2	74.3	+4.1
SwAV	70.1	74.1	+4.0

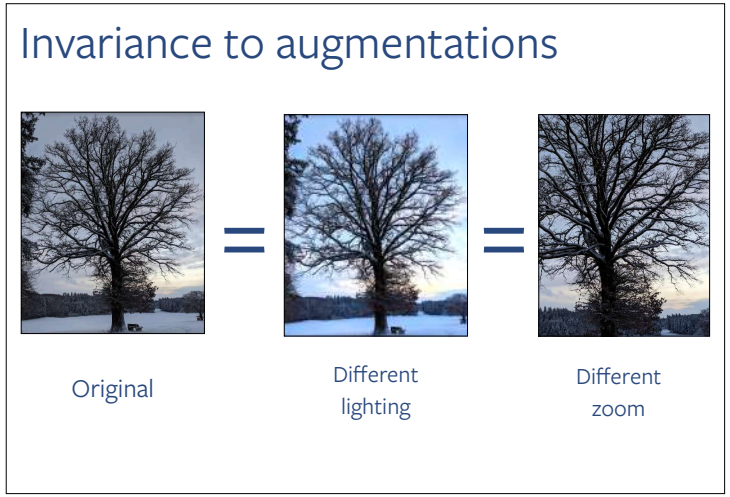
- SwAV uses SeLa's SK algo
- SeLa-v2 better than SimCLR

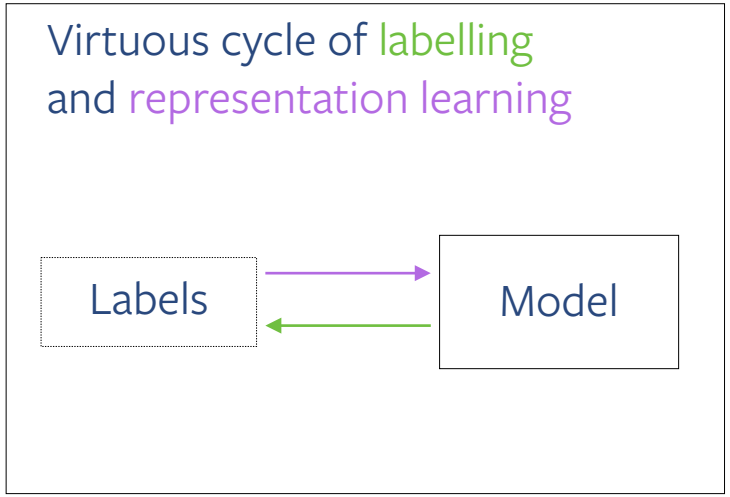
	Method	Momentum	Operation	Top-1
1	DINO	✓	`Centering`	76.1
2	–	✓	`Softmax(batch)`	75.8
3	–	✓	`Sinkhorn-Knopp`	76.0
4	–		`Centering`	0.1
5	–		`Softmax(batch)`	72.2
6	SwAV		`Sinkhorn-Knopp`	71.8

- DINO with SeLa's Sinkhorn: ~same performance.

Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. Caron et al. NeurIPS 2020.
Emerging Properties in Self-Supervised Vision Transformers. Caron et al. arXiv 2021.

Self-supervised labelling from three core ideas





(1) Transformations

- Data augmentations "infuse knowledge"

(2) Useful labels

- Labels discovered are similar to ground-truth
- Can be used to analyze how the network "sees" the data

(3) Balanced pseudo-labelling

- Well defined, fast objective
- No trivial solutions

Self-labelling via simultaneous clustering and representation learning. Asano et al., ICLR 2020