Continual Machine Learning

Summer 2023

Teacher

Dr. Martin Mundt,

hessian.AI-DEPTH junior research group leader on Open World Lifelong Learning (OWLL)
& researcher in the Artificial Intelligence and Machine Learning (AIML) group at TU Darmstadt

Time

Every Tuesday 17:30 - 19:00 CEST

Course Homepage

http://owll-lab.com/teaching/cl_lecture_23

https://www.youtube.com/playlist?list=PLm6QXeaB-XkA5-lVBB-h7XeYzFzgSh6sk

Recall: lifelong ML

Definition - **Lifelong Machine Learning** - Chen & Liu 2017:

"Lifelong Machine Learning is a continuous learning process. At any time point, the learner performed a sequence of N learning tasks, $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$ .These tasks can be of the same type or different types and from the same domain or different domains. When faced with the (N+1)th task $\mathcal{T}_{N+1}$ (which is called the new or current task) with its data D_{N+1}, the learner can leverage past knowledge in the knowledge base (KB) to help learn $\mathcal{T}_{N+1}$.

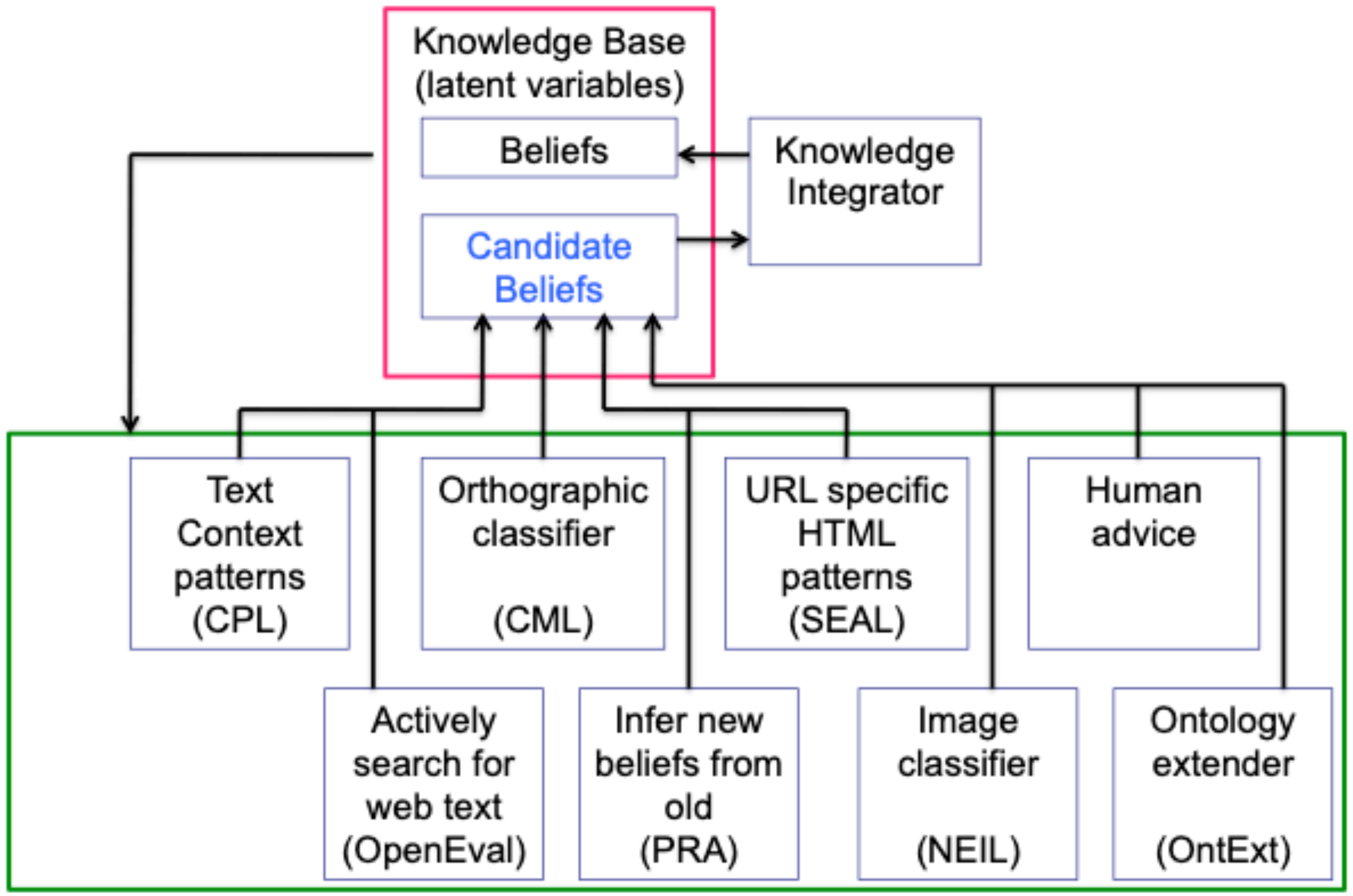
The objective of LML is usually to optimize the performance on the new task $\mathcal{T}_{N+1}$, but it can optimize any task by treating the rest of the tasks as previous tasks. KB maintains the knowledge learned and accumulated from learning the previous task. After the completion of learning $\mathcal{T}_{N+1}$, KB is updated with the knowledge (e.g. intermediate as well as the final results) gained from learning $\mathcal{T}_{N+1}$. The updating can involve inconsistency checking, reasoning, and meta-mining of additional higher-level knowledge."

"Lifelong Machine Learning", Chen & Liu, Morgan Claypool, 2017

Recall: knowledge in NELL

Knowledge is a lot more than just parameters

NELL Architecture



Knowledge Base (latent variables)

Beliefs

Candidate Beliefs

Knowledge Integrator

Text Context patterns (CPL)

Orthographic classifier (CML)

URL specific HTML patterns (SEAL)

Human advice

Actively search for web text (OpenEval)

Infer new beliefs from old (PRA)

Image classifier (NEIL)

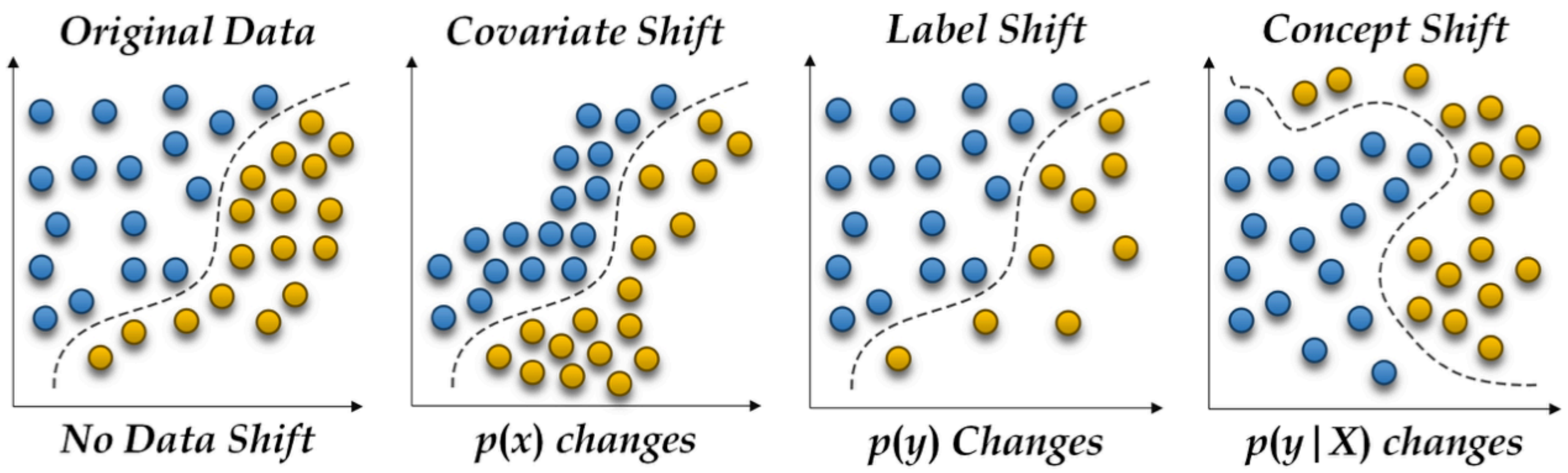
Ontology extender (OntExt)

- Ran 24/7 from 2010-2018
- Accumulated over 50 million candidate “beliefs” by reading the web
- Relational database
- Facts: barley is a grain
- Beliefs: sportUsesEquip (soccer, balls)

“Towards an Architecture for Never-Ending Language Learning”, Carlson et al, AAAI 2010

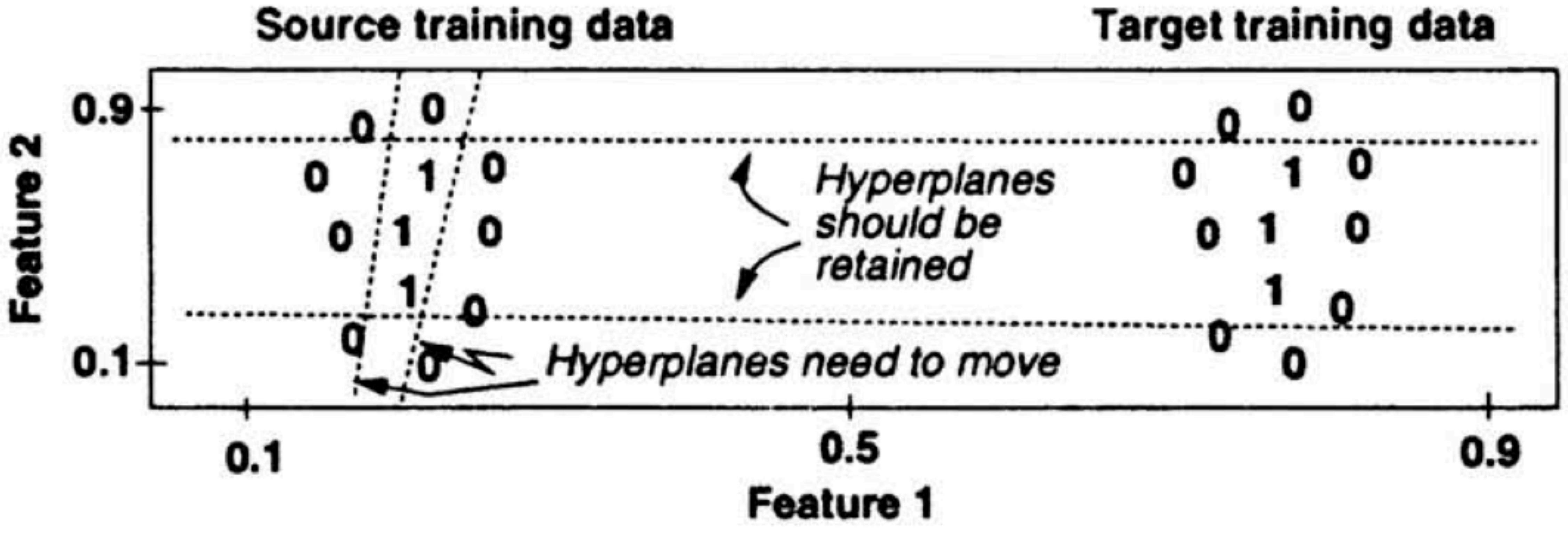
“Never-Ending Learning”, T. Mitchell et al, AAAI 2015

Recall: shifts & transfer



(a) Original data (b) Covariate shift (c) Label shift (d) Concept shift

Figure from "Understanding Dataset Shift and Potential Remedies", Vector Institute Technical Report, 2021

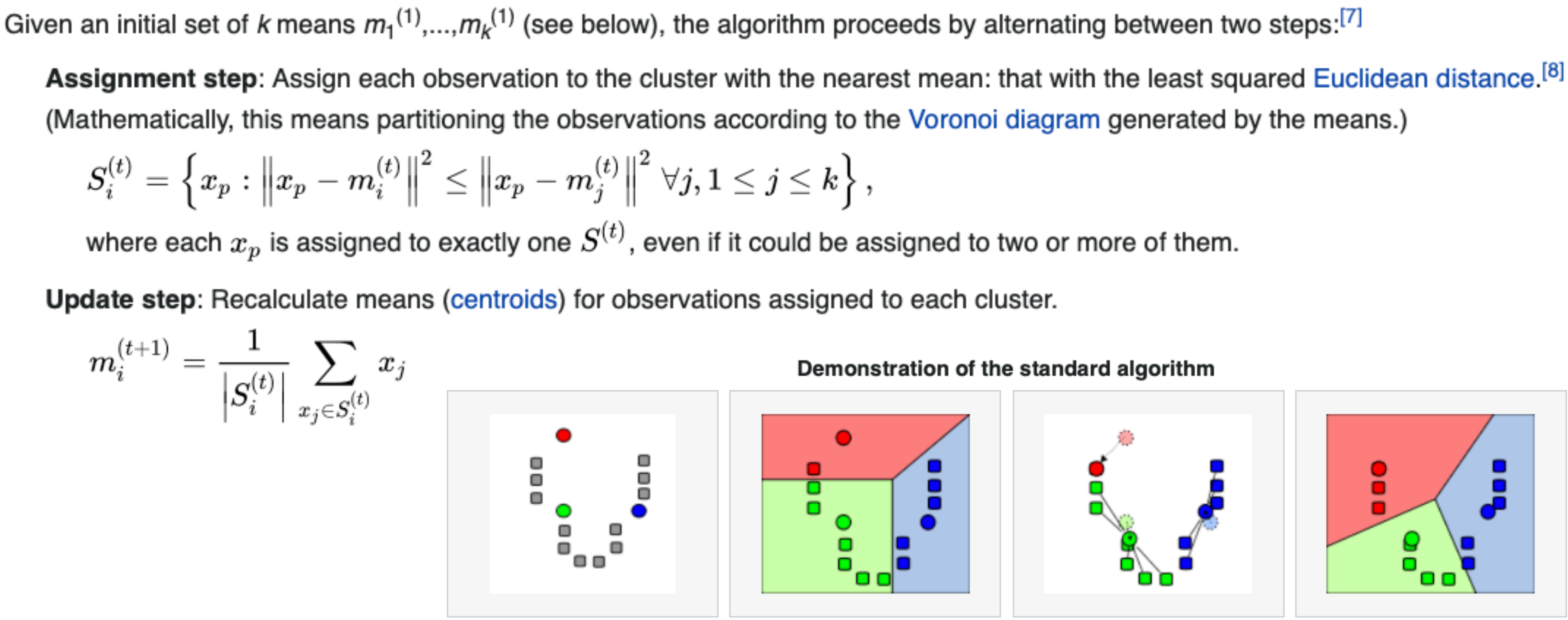


"Discriminability-Based Transfer between Neural Networks", L. Y. Pratt, NeurIPS 1992

In transfer learning, if we equate knowledge with learned parameters, we will very likely have some degree of forgetting of how to perform on the source task

Week 3: Optimization & Knowledge Retention

An intuitive example: K-means



Given an initial set of *k* means $m_1^{(1)},\dots,m_k^{(1)}$ (see below), the algorithm proceeds by alternating between two steps:[7]

Assignment step: Assign each observation to the cluster with the nearest mean: that with the least squared Euclidean distance.[8] (Mathematically, this means partitioning the observations according to the Voronoi diagram generated by the means.)

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \le \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \le j \le k \right\},$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Recalculate means (centroids) for observations assigned to each cluster.

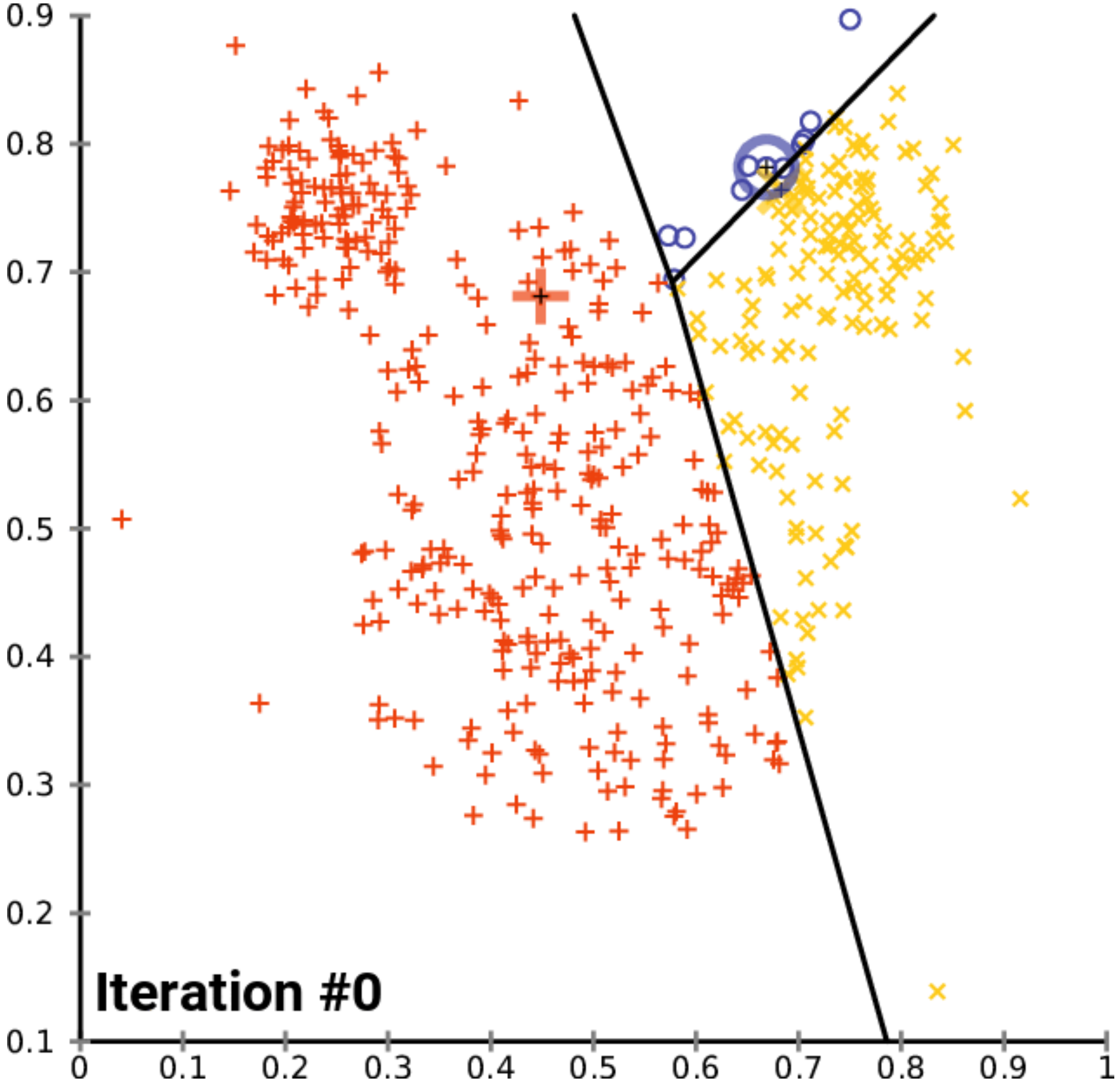
$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

Demonstration of the standard algorithm

https://en.wikipedia.org/wiki/K-means_clustering , shared under Creative Commons license

Abrupt & gradual forgetting

When will it be surprising to see that we forget if we add new data?



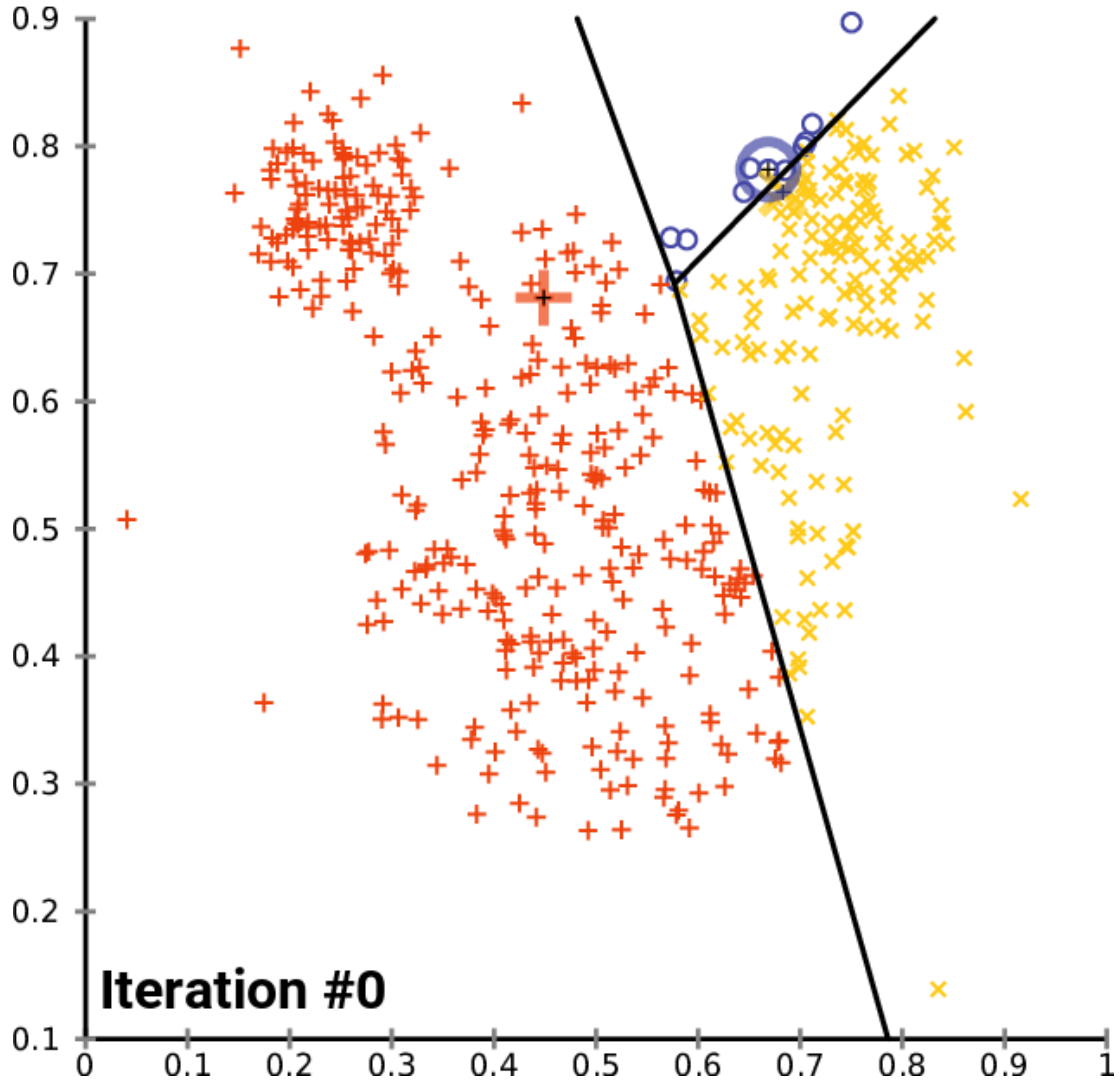
https://en.wikipedia.org/wiki/File:K-means_convergence.gif
Shared under Creative Commons license

Abrupt & gradual forgetting

When will it be surprising to see that we forget if we add new data?

- Number of clusters?
- Data isn't accumulated but replaced, mean changes abruptly
- Considerations such as exponentially moving average of the mean?

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

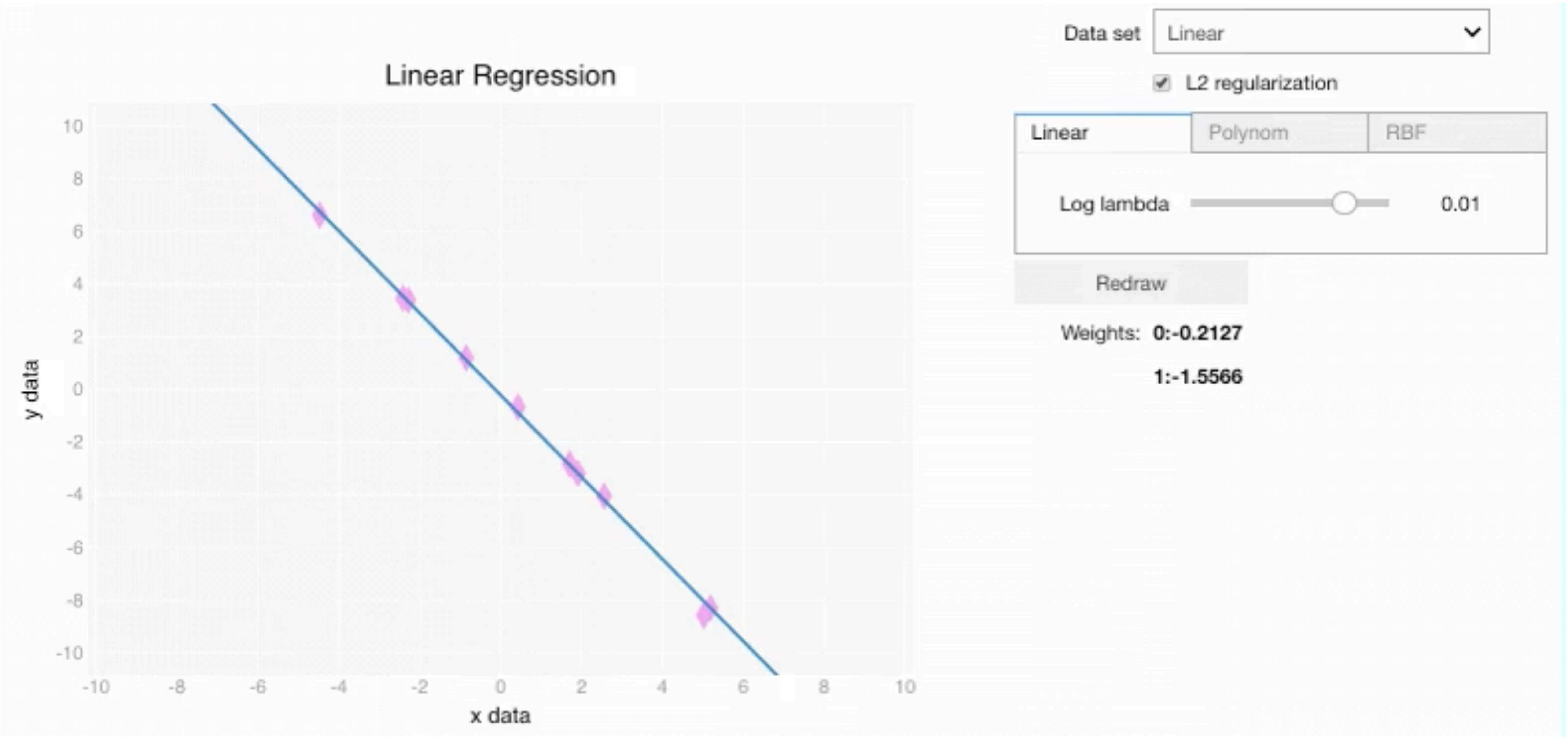


https://en.wikipedia.org/wiki/File:K-means_convergence.gif
Shared under Creative Commons license

Linear regression

Another intuitive example https://github.com/PyMLVizard/PyMLViz

$$y(x) = \boldsymbol{w}^T \boldsymbol{x} + \epsilon = \sum_{i=1}^{D} w_i x_i + \epsilon$$



Data set: Linear

☑ L2 regularization

Log lambda 0.01

Weights: 0:-0.2127

1:-1.5566

Optimization: risk & losses

What we would like to generally do is minimize the following scenario:

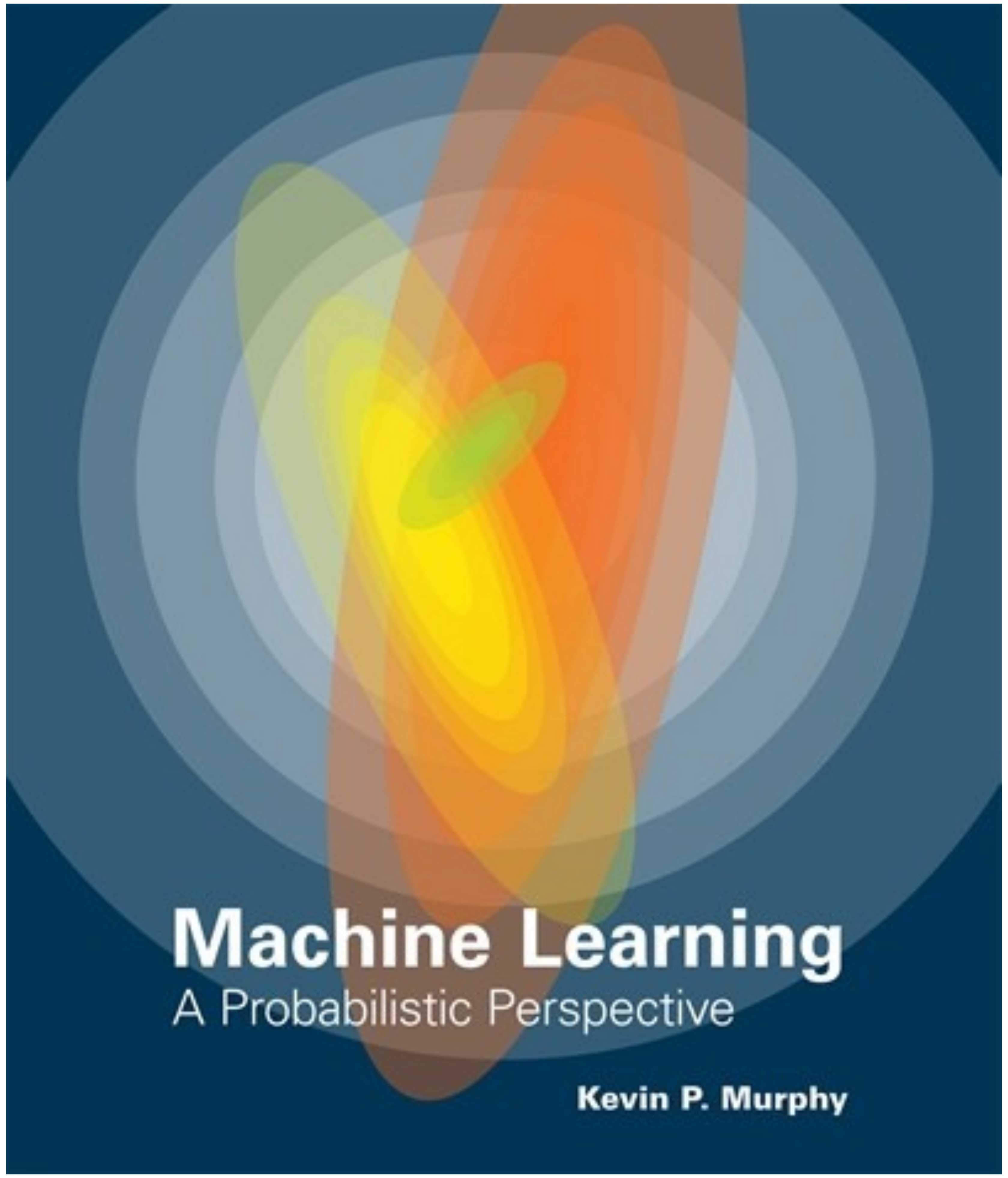
Find a hypothesis or decision procedure:

$$\delta : \mathscr{X} \rightarrow \mathscr{A}$$

and define the risk or expected loss as:

$$R(\theta^*, \delta) = \mathbb{E}_{p(\tilde{D}|\theta^*)}\left[L(\theta^*, \delta(\tilde{D}))\right]$$

Where $\tilde{D}$ is data from the true distribution, represented by parameter θ^*



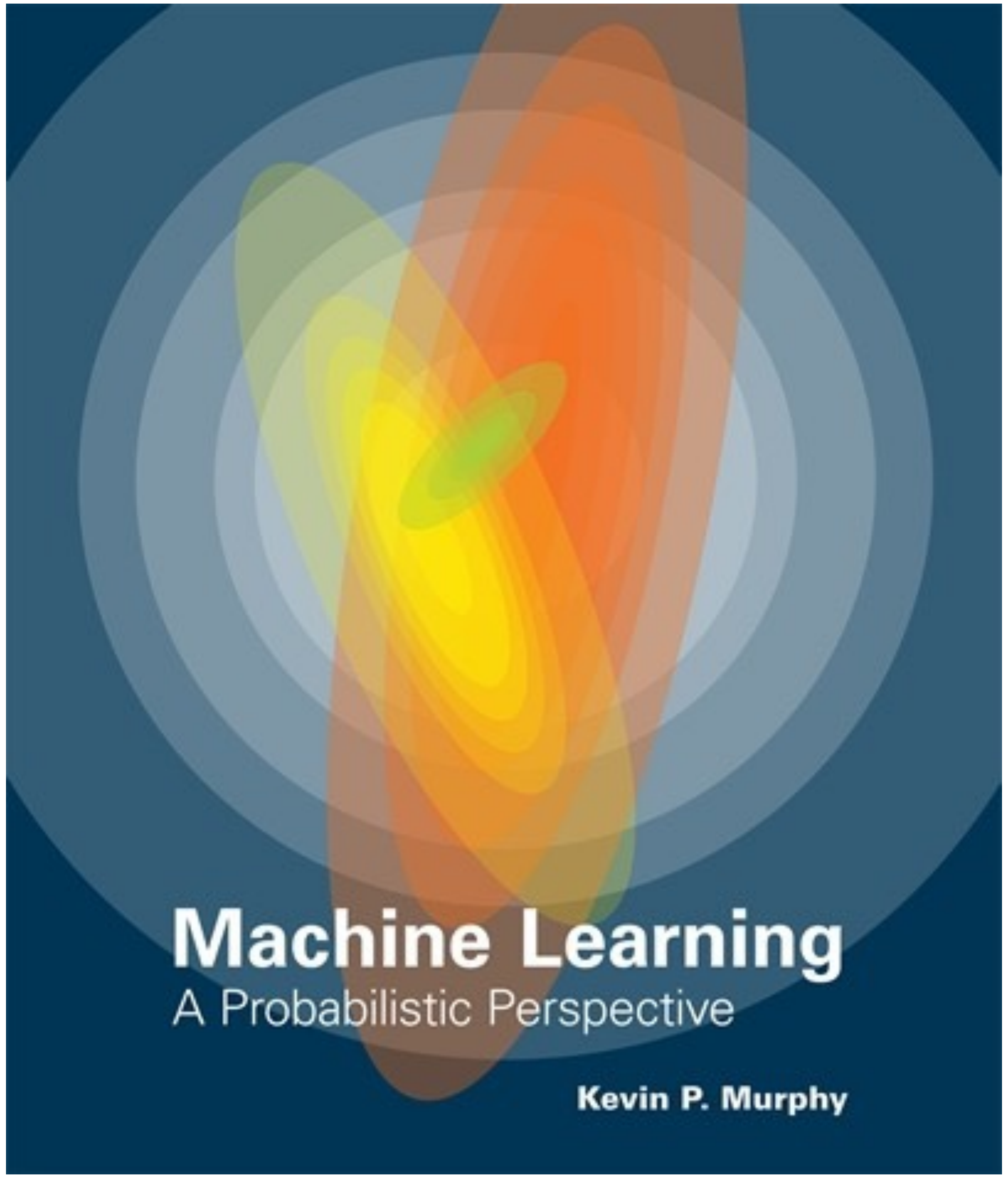
Pages 197-209

Optimization: risk & losses

$$R(\theta^*, \delta) = \mathbb{E}_{p(\tilde{D}|\theta^*)}\left[L(\theta^*, \delta(\tilde{D}))\right]$$

The challenges:

- Cannot actually compute above risk (usually don’t know the distribution)
- Besides: if we think of e.g. binary classification, i.e. a 0-1 measure, it can be hard to optimize as it is not smooth



Pages 197-209

Optimization: risk & losses

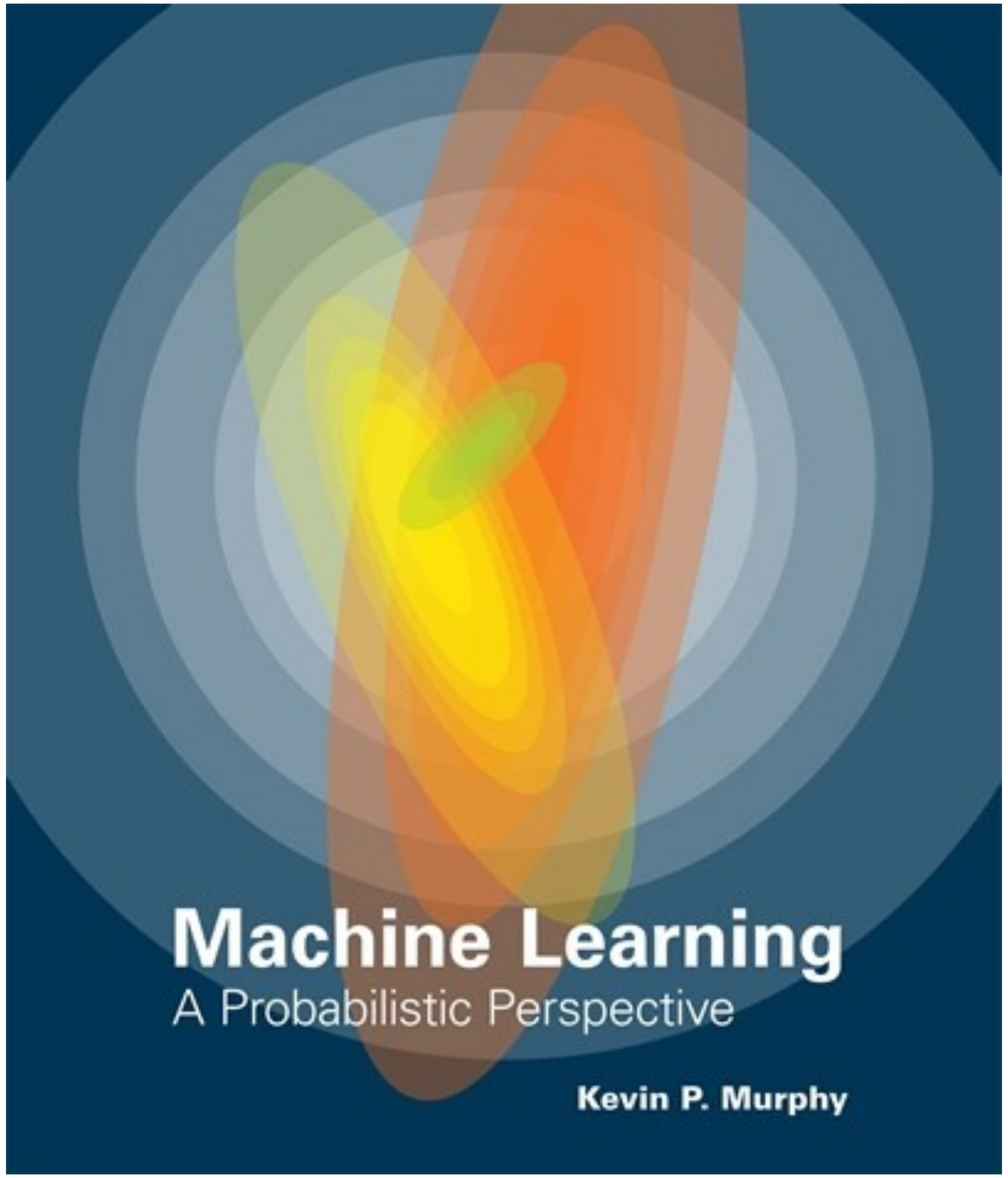
$$R(\theta^*, \delta) = \mathbb{E}_{p(\tilde{D}|\theta^*)}\left[L(\theta^*, \delta(\tilde{D}))\right]$$

instead: $R(p^*, \delta) = \mathbb{E}_{(x,y)\sim p^*}\left[L(y, \delta(x))\right]$

But can look at the true but unknown response and our predictions $\delta(x)$ given an input x.

As we still do not know the true distribution, we can also use empirical estimates:

$$R_{emp}(D, \delta) = 1/N \sum_{i=1}^{N} L(y_i, \delta(x_i))$$



Pages 197-209

Optimization: risk & losses

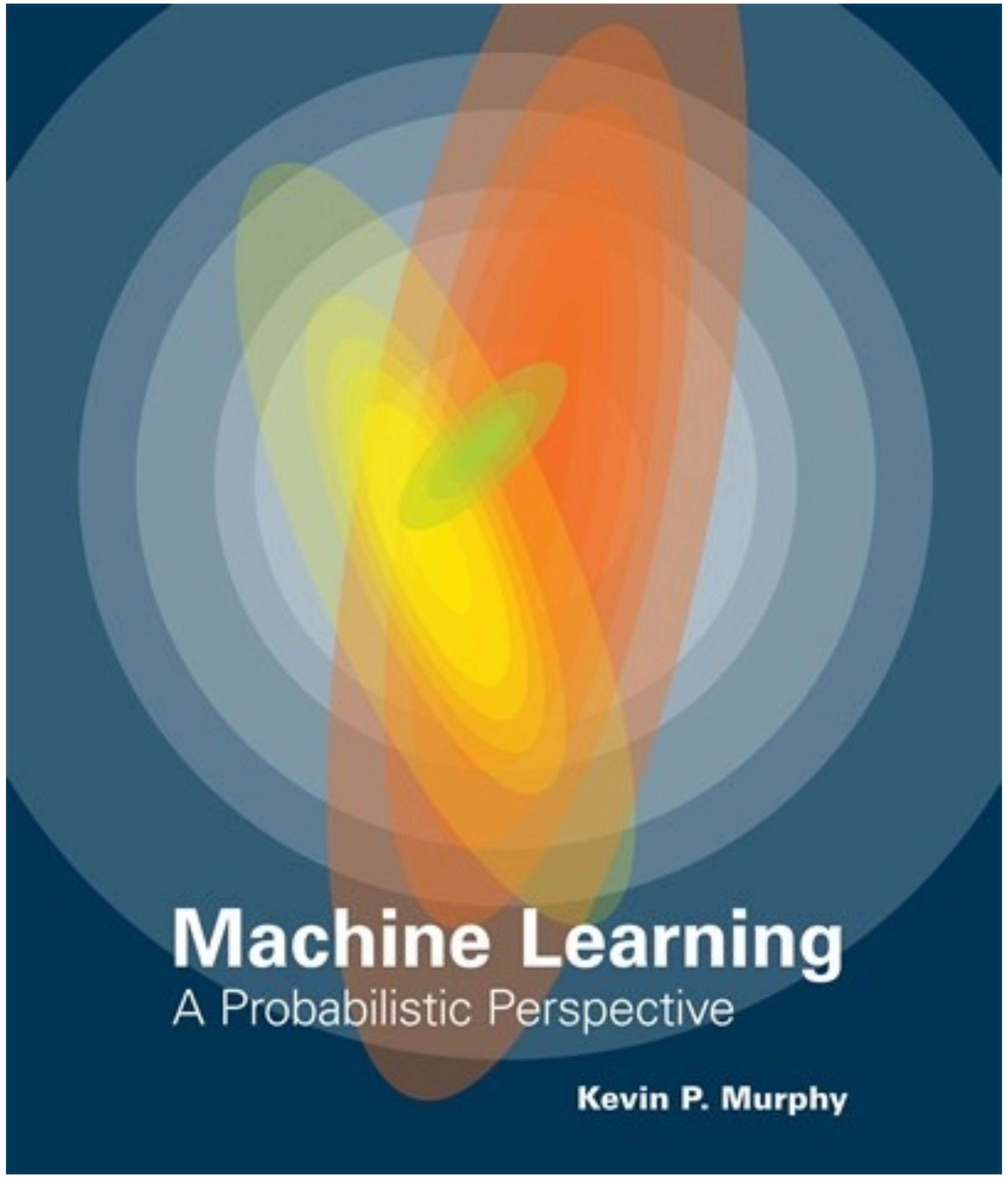
$$R_{emp}(D, \delta) = 1/N \sum_{i=1}^{N} L(y_i, \delta(x_i))$$

We then usually chose a loss function, e.g. the mean squared error (supervised):

$$L(y, \delta(x)) = (y - \delta(x))^2$$

or similarly an unsupervised reconstruction:

$$L(y, \delta(x)) = ||x - \delta(x)||_2^2$$



Pages 197-209

Optimization: gradient descent

There are various optimization algorithms, the most popular ones are perhaps: (Stochastic) gradient descent - SGD and expectation maximization (EM)

Let us consider (S)GD here, as the “workhorse” underlying a lot of deep learning:

- In the simple form, a first order optimization algorithm to find a minimum of a differentiable function
- Achieved by iteratively taking (small) steps in the gradient direction of a function f in the direction in which it decreases the fastest:

$$x_{n+1} = x_n - \lambda \nabla f(x_n) \quad where \quad f(x_0) \geq f(x_1) \geq \ldots \geq f(x_n)$$

Optimization: gradient descent

We can easily transfer this concept to the idea of parameters and losses:

$$L(\theta) = 1/N \sum_{i=1}^{N} L_i(\theta))$$

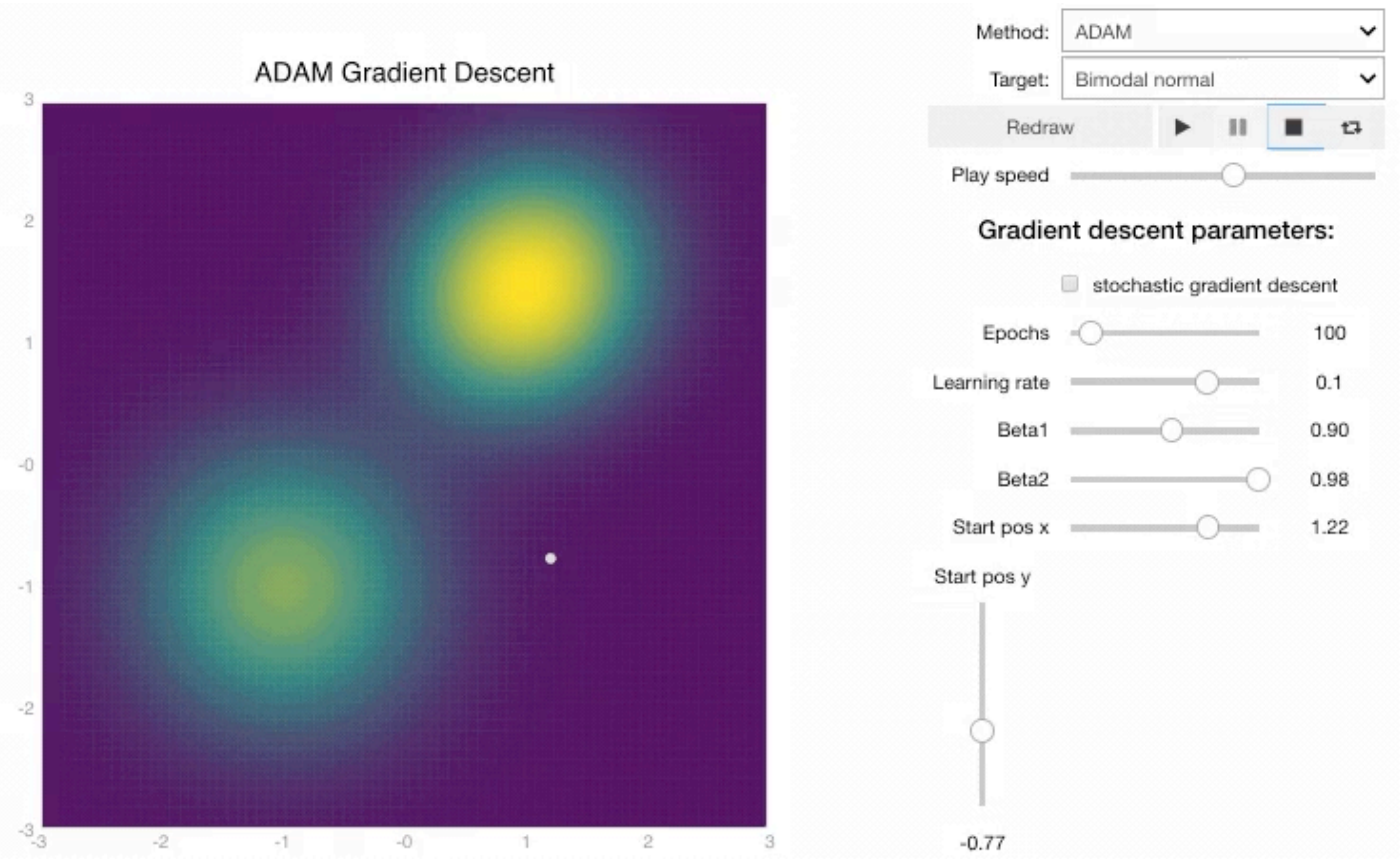
Then iterative updates become (where in neural nets we backpropagate gradients):

$$\theta \leftarrow \theta - \lambda \nabla L(\theta) = \theta - \lambda/N \sum_{i}^{N} \nabla L_i(\theta)$$

Let us talk about gradient estimates, stochasticity, step sizes, and ultimately the idea of forgetting with interactive examples

Stochastic & gradient descent

Another interactive detour https://github.com/PyMLVizard/PyMLViz



Method: ADAM

Target: Bimodal normal

Gradient descent parameters:

□ stochastic gradient descent

Epochs 100

Learning rate 0.1

Beta1 0.90

Beta2 0.98

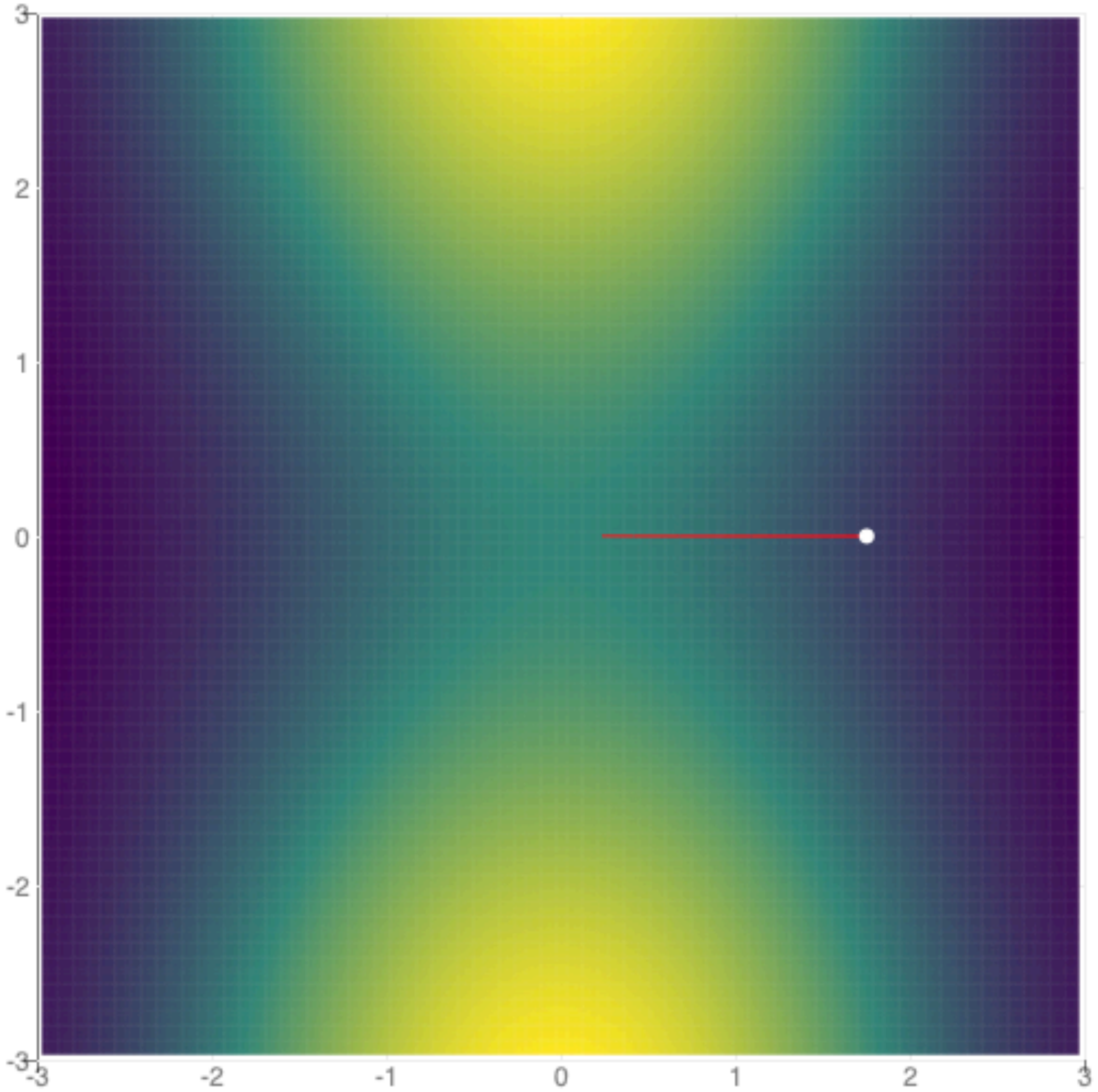
Start pos x 1.22

Start pos y -0.77

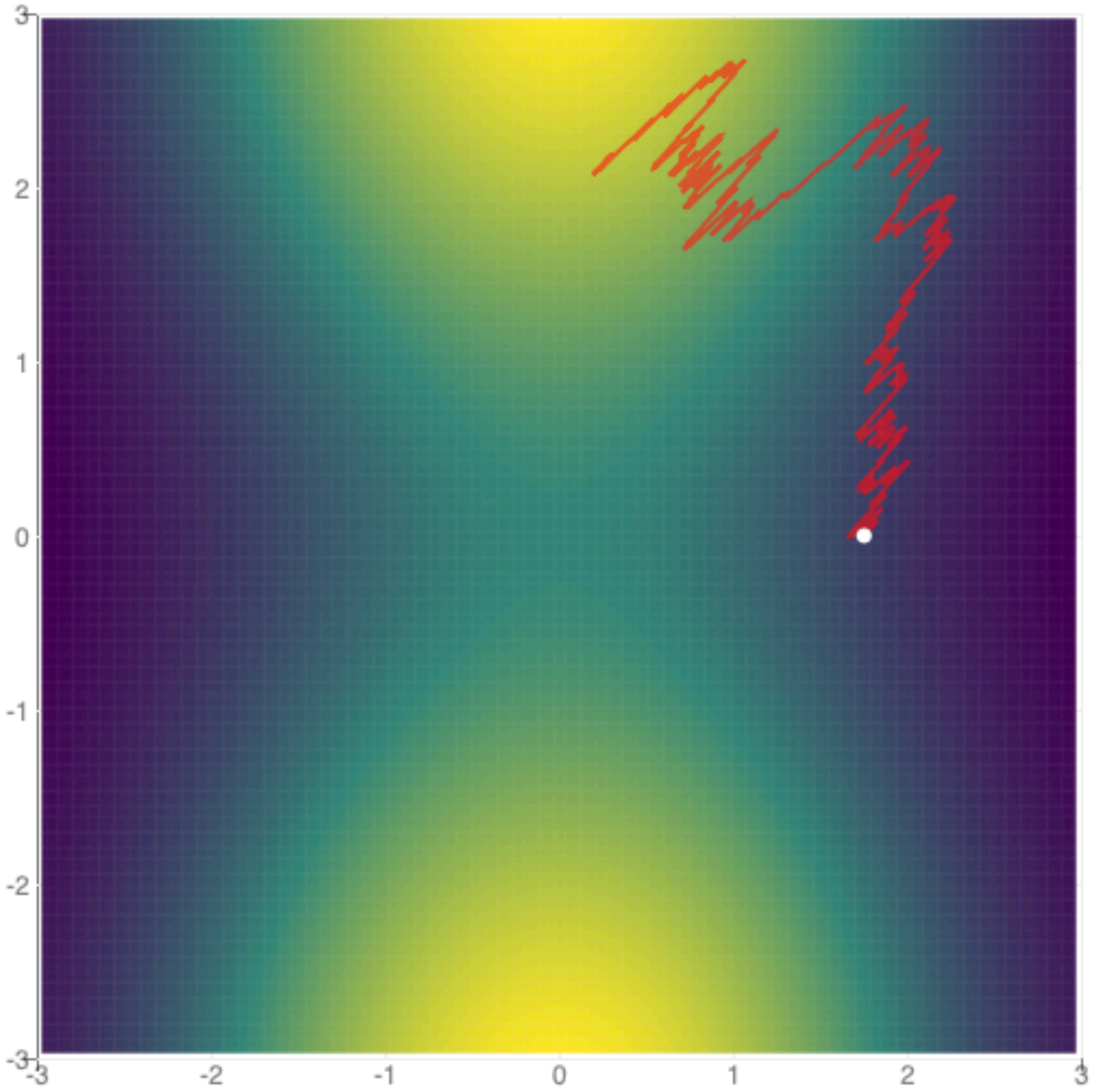
Demo: for pdf completeness

We have visited gradient descent vs stochastic gradient descent

Saddle points with gradient descent



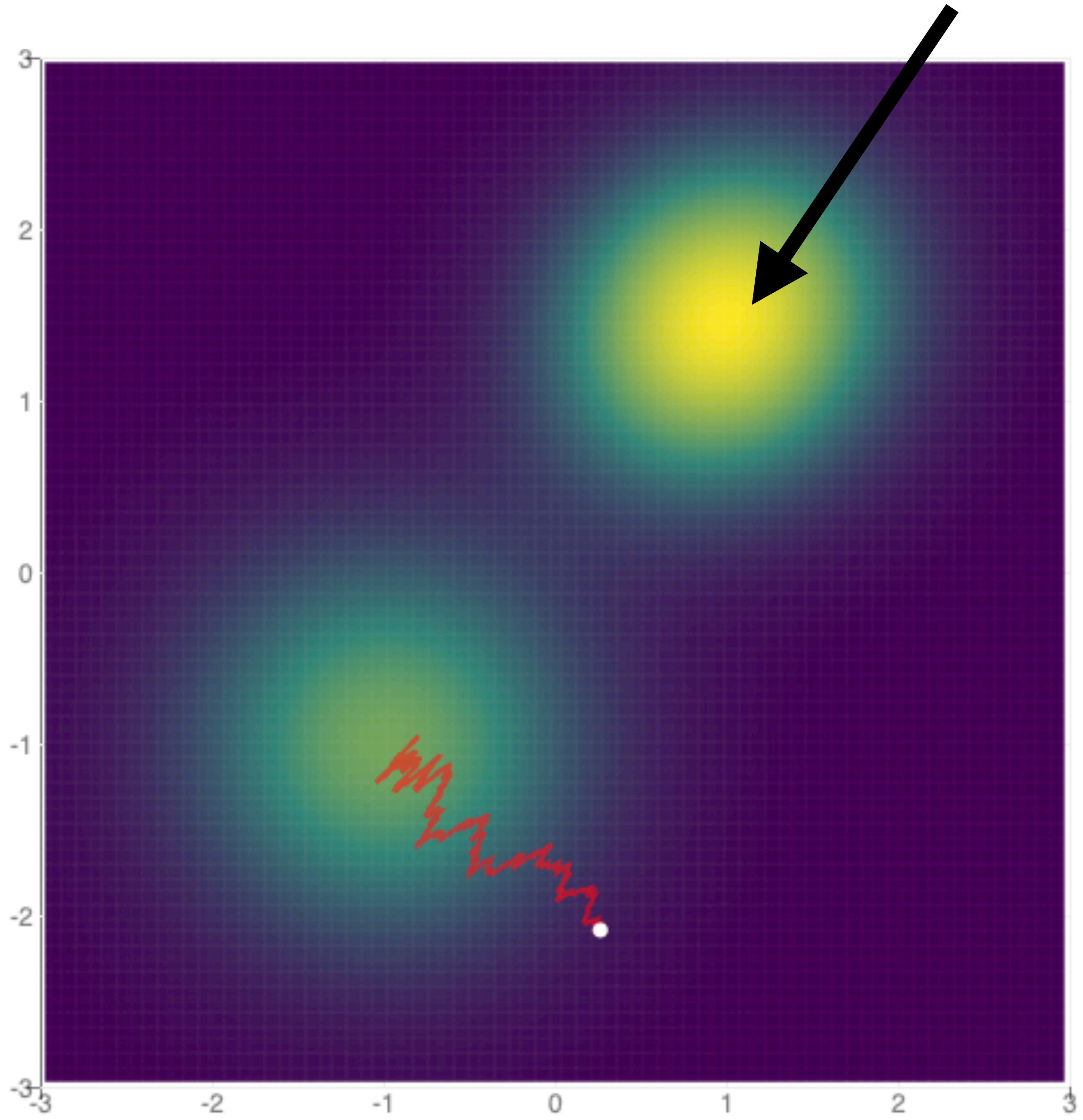
and with stochastic gradient descent



Demo: for pdf completeness

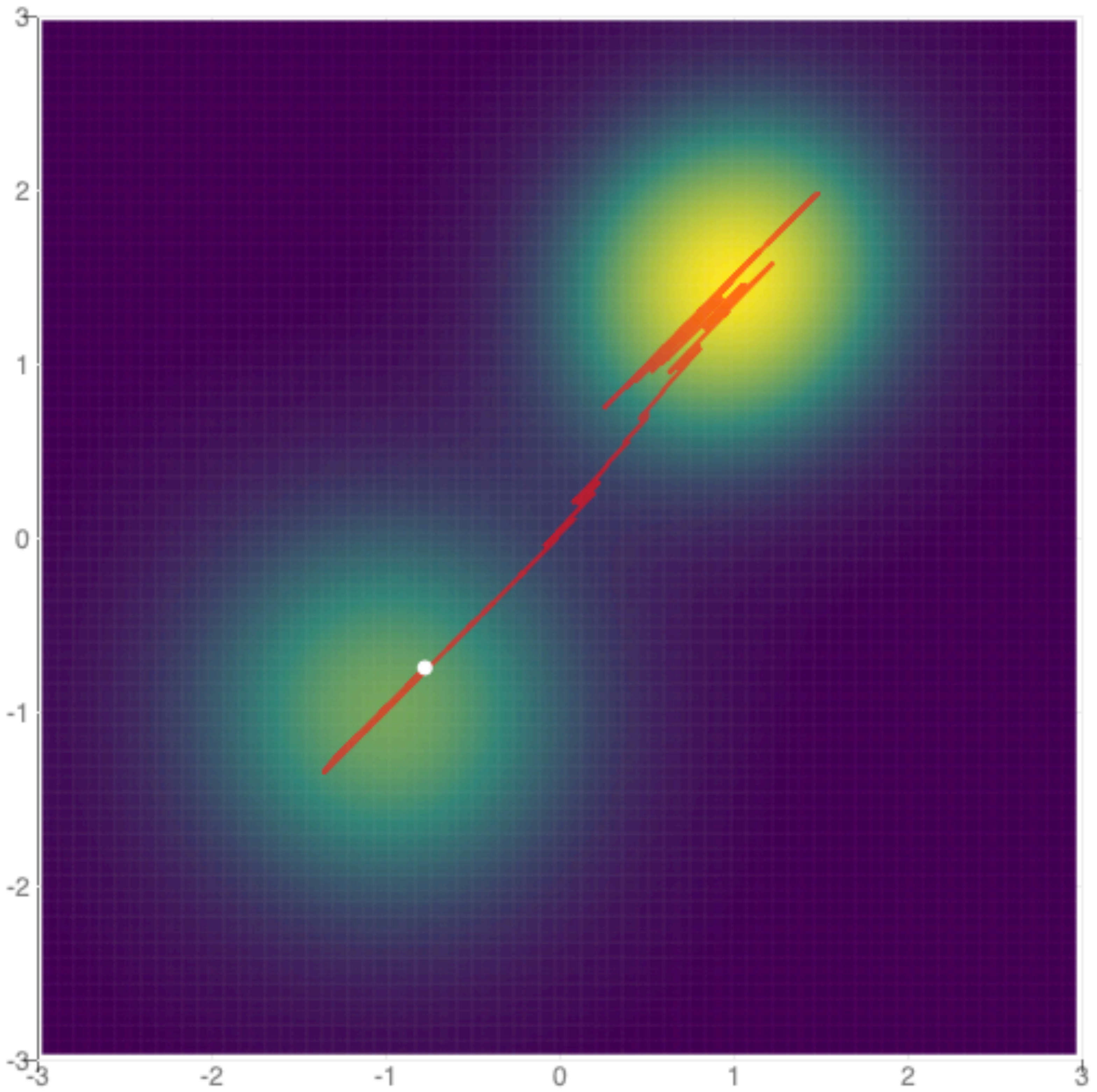
We have motivated interference/forgetting from an SGD point of view

Assume this wasn't there in "task" 1



And then you add it!

(Or even if you don't & noise is too large)



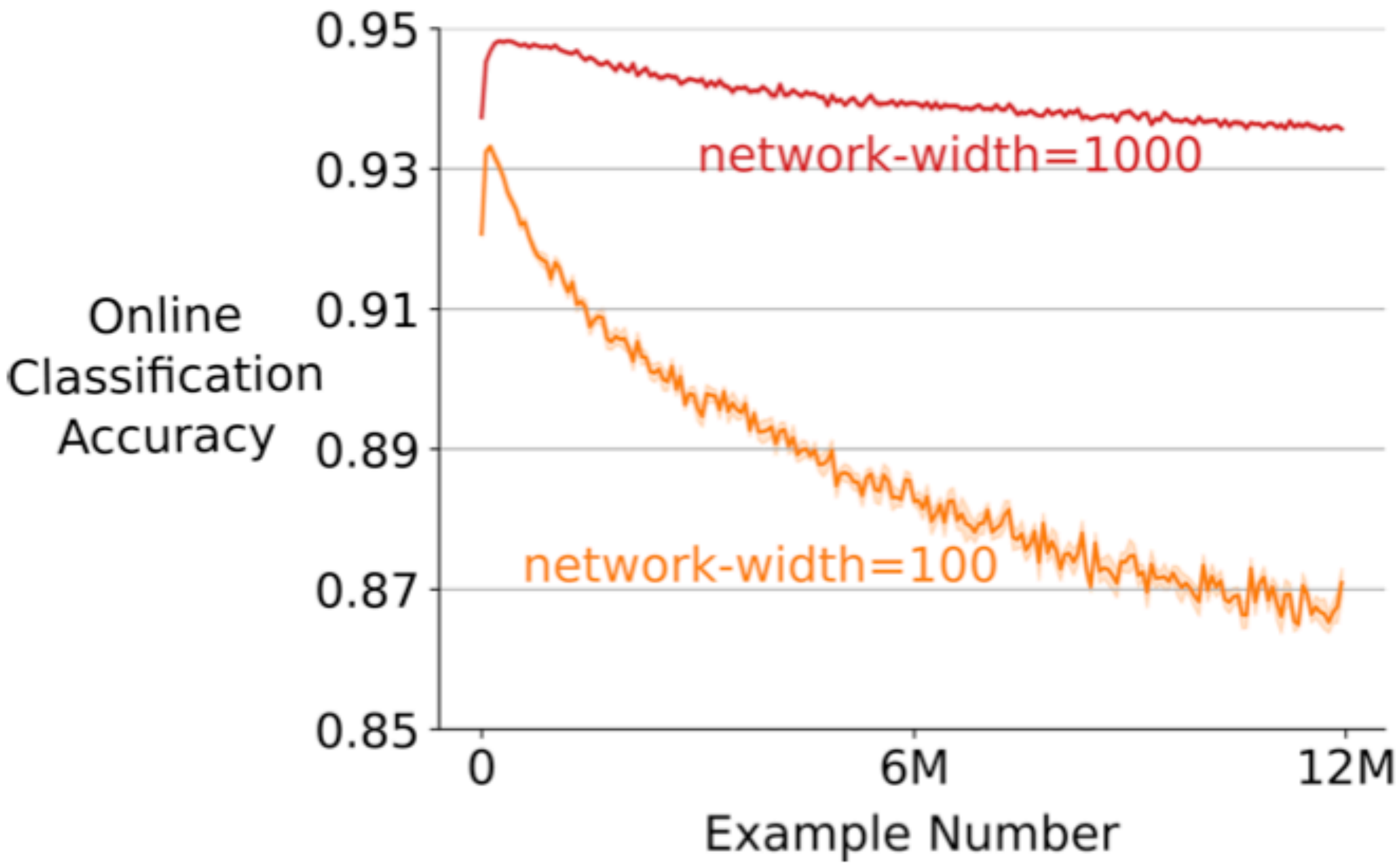
How do we prevent forgetting?

The undesired trivial solution: large amounts of parameters + data accumulation

But also, caution, if we are constrained by capacity, we won't learn indefinitely!

Keep in mind however (transfer):

if the number of parameters is limited & already learned, it will become increasingly difficult to encode new concepts (e.g. on the right example of permuting the data points over time)



Continual Backprop: Stochastic Gradient Descent with Persistent Randomness, Dohare et al, arXiv preprint:2108:06325

How can we alleviate forgetting?

How do we alleviate forgetting?

Regularize important parameters (today):

Either identify relevant parameters for a task and make sure they do not change much, or make sure the input output relationship remains the same

Rehearsal:

Either store a subset of old data to rehearse or make use of a generative model to generate old task data

Modify the architecture:

Either use task specific masks in an overparameterized model or grow/expand the architecture

Categorization found in several recent reviews, e.g. Parisi 2019, DeLange 2019, Biesialska 2020, Hadsell 2020…, but outlined mostly already in McCloskey & Cohen 1989, Ratcliff 1990, French 1999 and many more

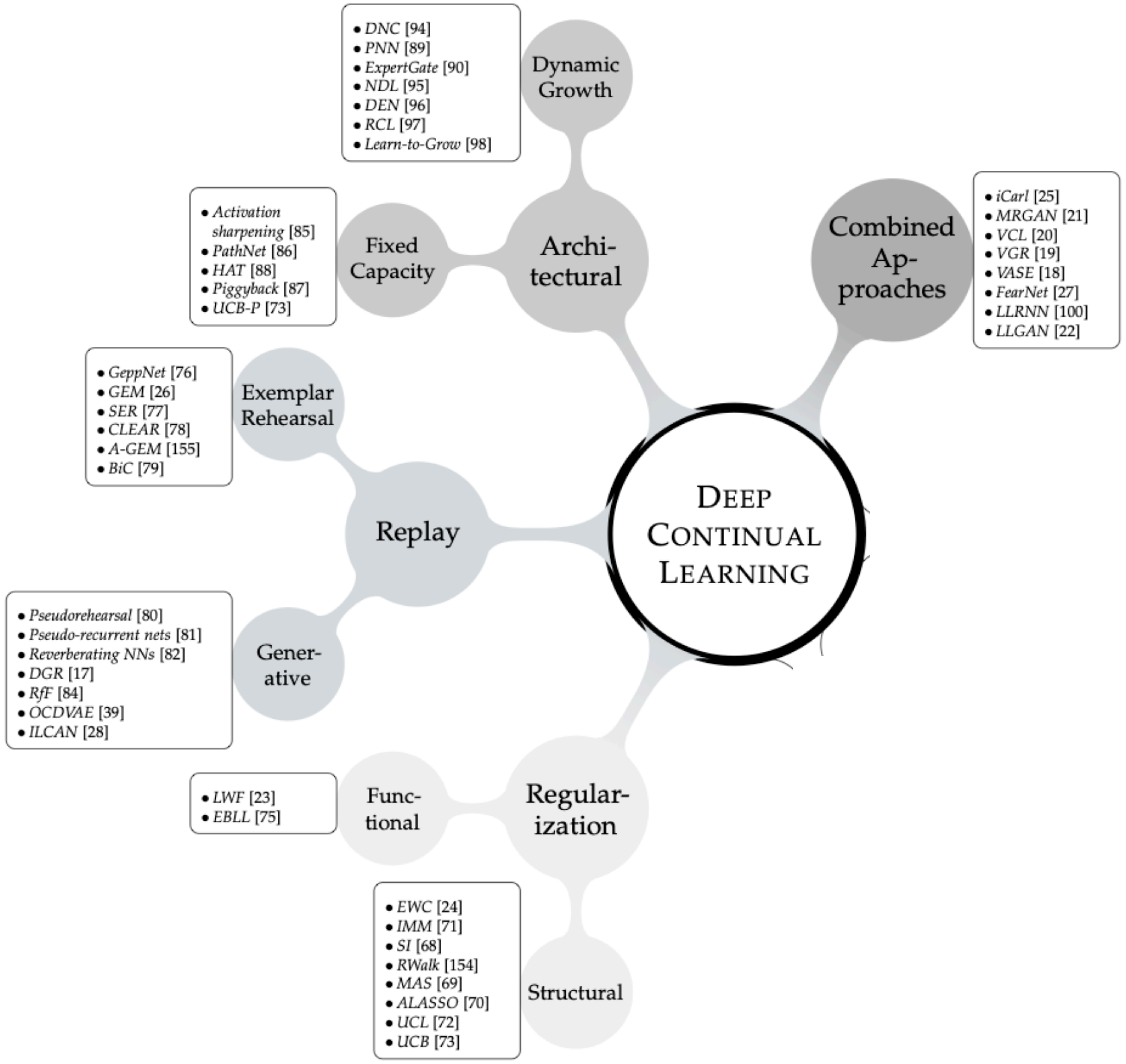


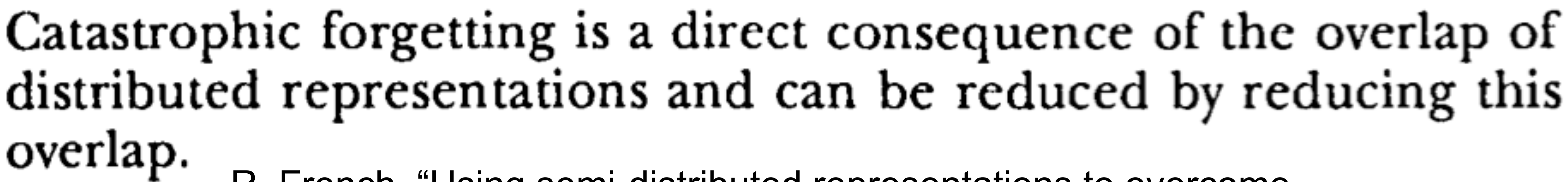
Figure from "A Wholistic View of Deep Neural Networks: Forgotten Lessons and the Bridge to Active and Open World Learning", Mundt et al, Neural Networks 2023, (arxiv 2020)

Some early thoughts

Most definitely not the earliest, but very intuitive examples!

Ideas date back to at least the 70s, even the 50s.

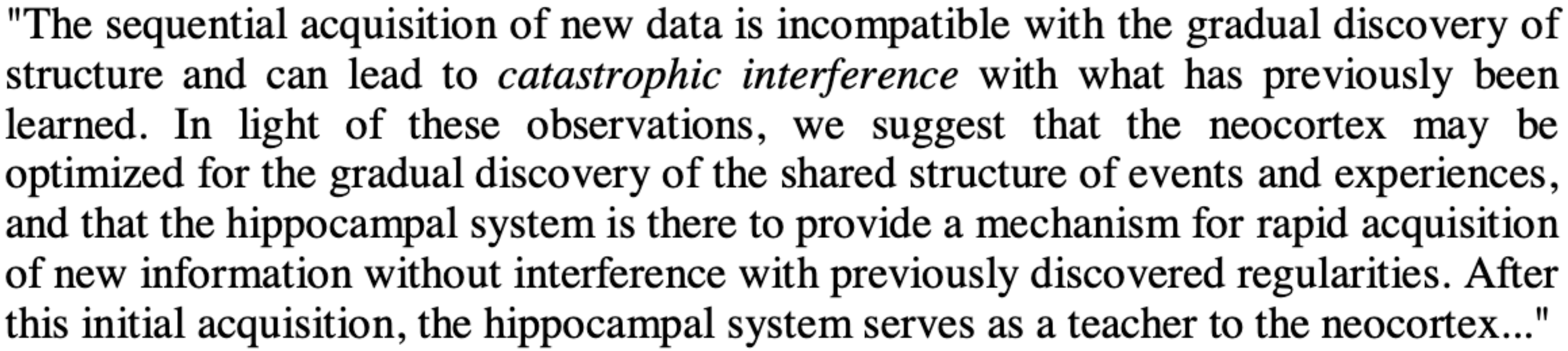
Modifying the model



Catastrophic forgetting is a direct consequence of the overlap of distributed representations and can be reduced by reducing this overlap.

R. French, “Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks”, AAAI 1993

Rehearsal

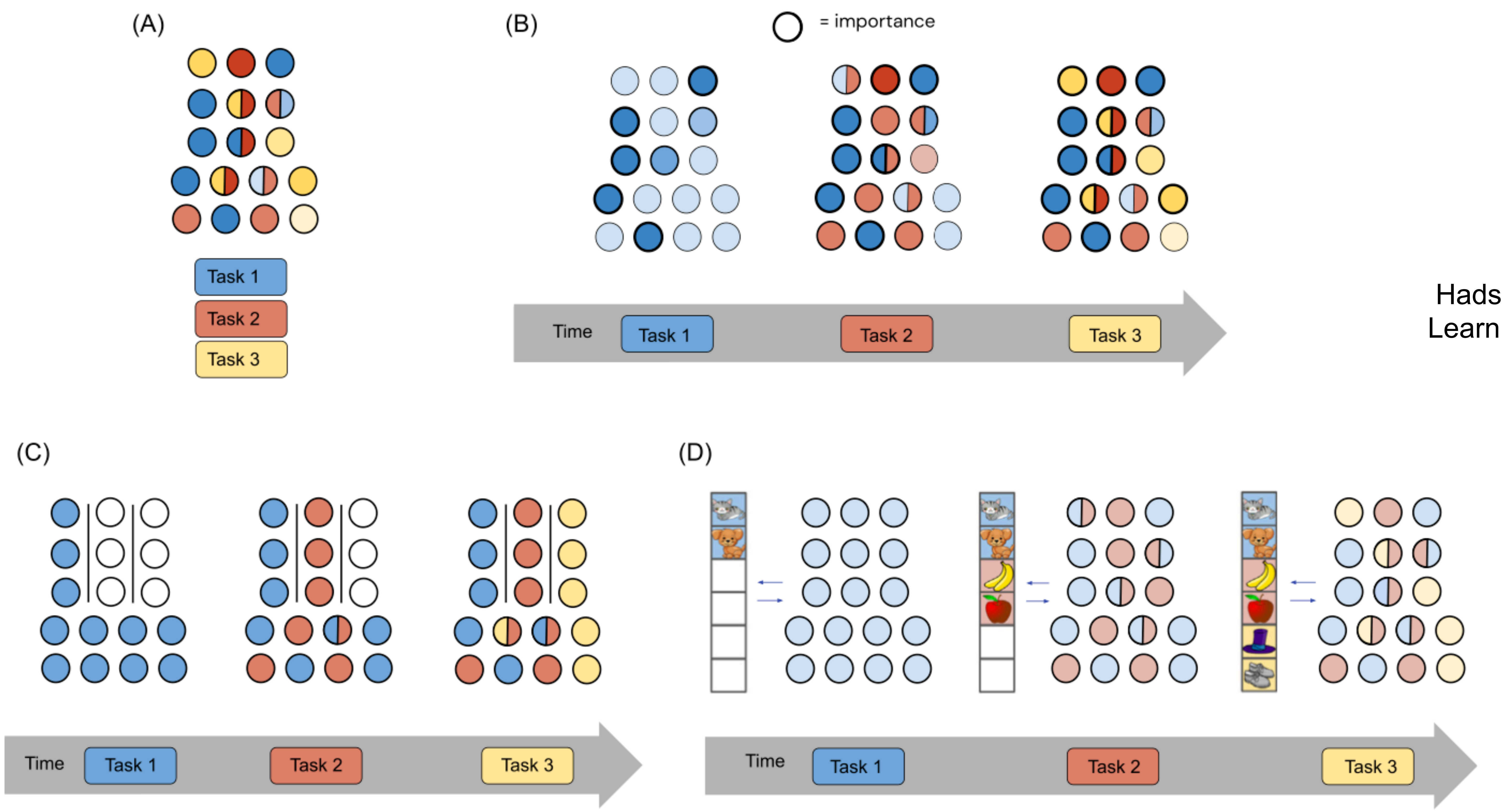


"The sequential acquisition of new data is incompatible with the gradual discovery of structure and can lead to *catastrophic interference* with what has previously been learned. In light of these observations, we suggest that the neocortex may be optimized for the gradual discovery of the shared structure of events and experiences, and that the hippocampal system is there to provide a mechanism for rapid acquisition of new information without interference with previously discovered regularities. After this initial acquisition, the hippocampal system serves as a teacher to the neocortex..."

McClelland et al, “Why there are complementary learning systems in the hippocampus and neocortex”, Psychological Review 102, 1995 (see also Robins 1995)

How do we prevent forgetting?

Paradigms for Continual Learning

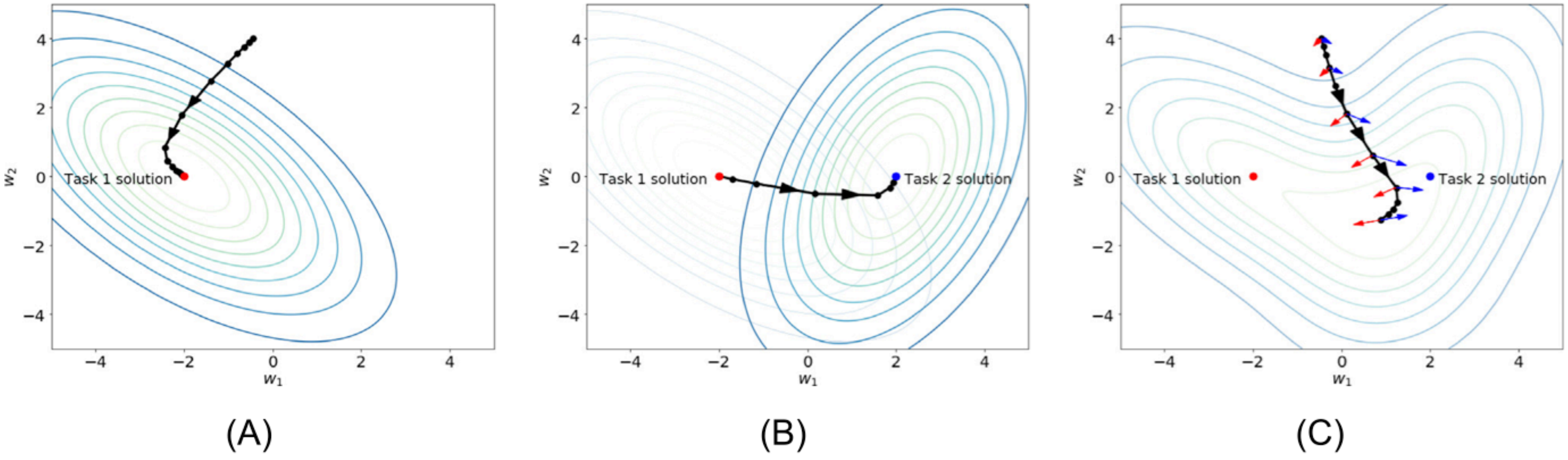


Hadsell et al, “Embracing Change: Continual Learning in Deep Neural Networks”, Trends in Cognitive Sciences 24:12, 2020

Figure 1. (A) Independent and identically distributed learning methods are standard for nonsequential, multitask learning. In this regime, tasks are learned simultaneously to avoid forgetting and instability. (B) Gradient-based approaches preserve parameters based on their importance to previously learned tasks. (C) Modularity-based methods define hard boundaries to separate task-specific parameters (often accompanied by shared parameters to allow transfer). (D) Memory-based methods write experience to memory to avoid forgetting.

Stability - plasticity (sensitivity)

For “regularization approaches”, what we are essentially interested in is the so called stability - plasticity (or sensitivity) dilemma (Hebb, “The organization of behavior”, 1949).



Trends in Cognitive Sciences

Figure 3. Illustrations of Gradient Descent Optimization for Different Tasks. (A) The trajectory taken by gradient descent optimization when minimizing a loss corresponding to a single task. (B) The optimization trajectory when subsequently training the same model on a second task. (C) The trajectory taken when using the total loss from both tasks (black) and the gradients from each individual task at multiple points during optimization (red and blue). See Box 2 for more detailed discussion.

Hadsell et al, “Embracing Change: Continual Learning in Deep Neural Networks”, Trends in Cognitive Sciences 24:12, 2020

1. Finding and regularizing important parameters

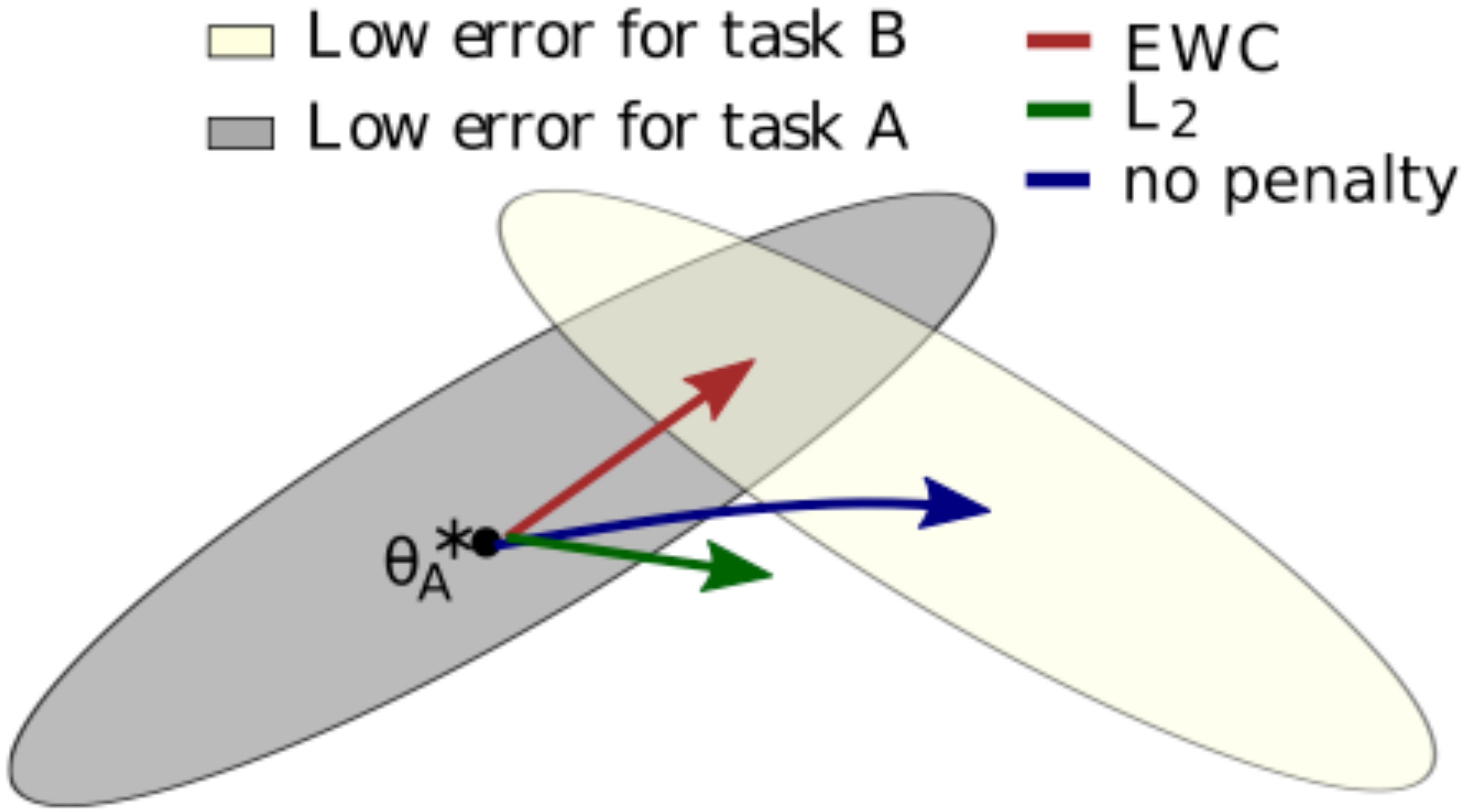
Elastic weight consolidation

$$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

Instead of naively continuing to optimize task B, we can impose a penalty on previously learned parameters (assuming over-parameterization).

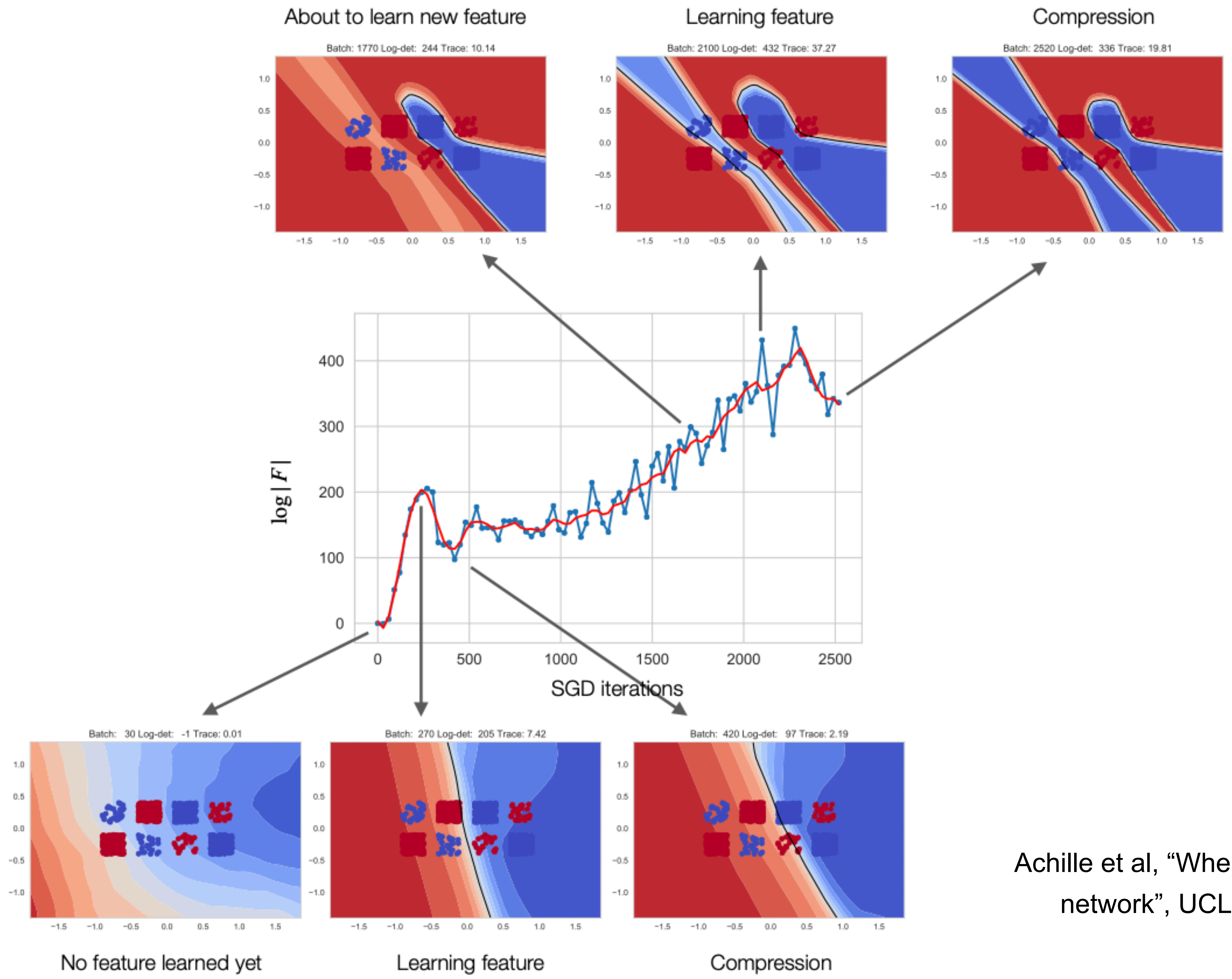
We will need to find a matrix F that tells us which parameters are most important for task A.

Example: Fisher information (related to natural gradients, the second derivative of the loss near a minimum, can be approximated). We will skip the details here for simplicity. (https://agustinus.kristia.de/techblog/2018/03/11/fisher-information/ provides a nice summary)



Kirkpatrick et al, “Overcoming catastrophic forgetting in neural networks”, PNAS 114(13), 2017

Parameter importance intuition



Achille et al, “Where is the information in a deep neural network”, UCLA-TR:190005, 2019

Synaptic intelligence

Here, “synapse” synonymous with parameter.

Key idea: change (with time t) in loss is well approximated by the gradient (g):

$$L(\theta(t) + \delta(t)) - L(\theta(t)) \approx \sum_k g_k(t)\delta_k(t)$$

Each parameter change $\delta_k(t) = \theta_k^{'}(t)$ contributes amount $g_k(t)\delta_k(t)$ to the change in total loss.

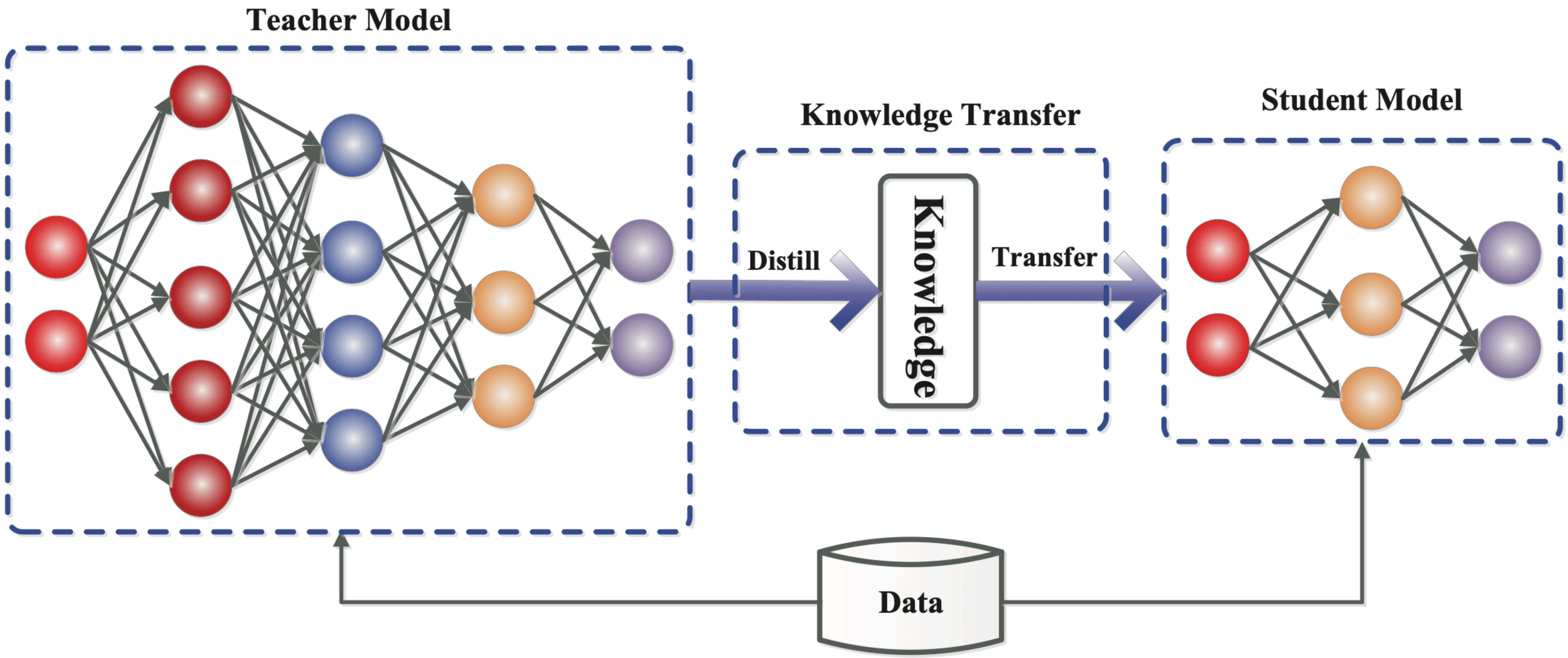
Assign an importance to each parameter according to the monitored trajectory and formulate a similar penalty to EWC again (with a different measure of importance).

2. Maintaining (input-output) relationships

Knowledge distillation

Alternatively, we know that if we have enough parameters, there are many potential solutions to produce the same input-output relationships.

Key idea: Let’s try to maintain a task’s input-output relationship



Gou et al, “Knowledge Distillation: A survey”, International Journal of Computer Vision 129, 2021

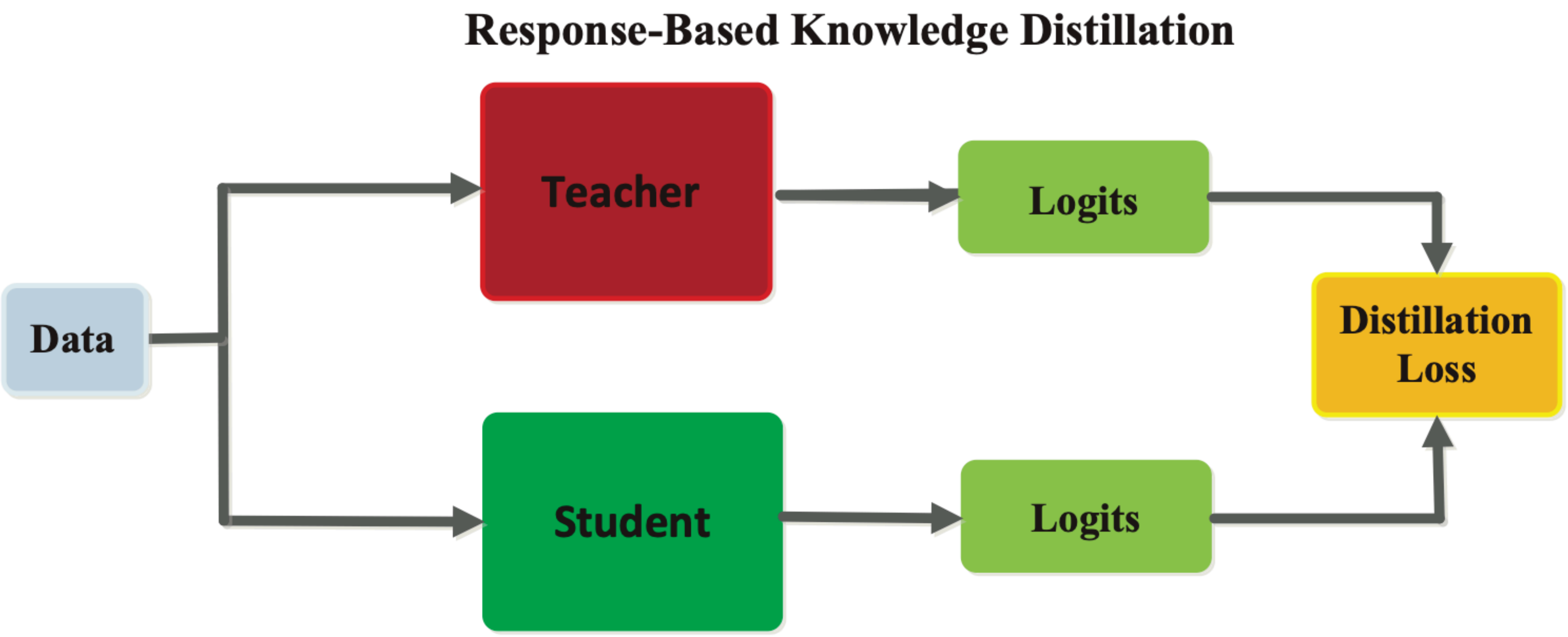
Knowledge distillation

Special case: classifier logits (Hinton et al, "Distilling the Knowledge in A Neural Network", NeurIPS 2014 Deep Learning Workshop):

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

Normally T=1, for higher T softer probability distributions are produced.

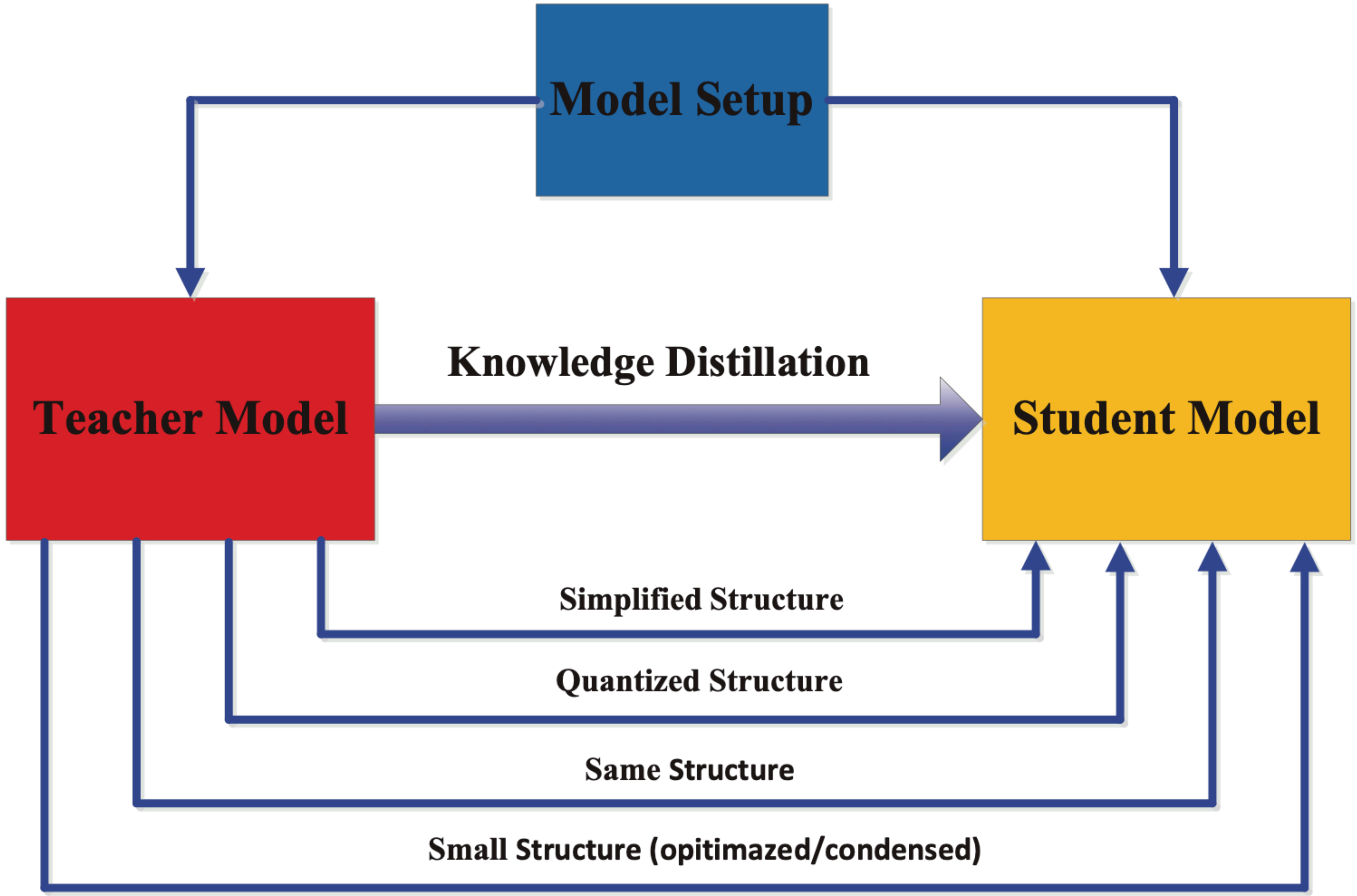
In essence we are making sure that the distance between z and v of two models is minimized, or more generally minimizing the KL divergence over the two probability distributions.



$$\frac{1}{T}(q_i - p_i) = \frac{1}{T}\left(\frac{exp(z_i/T)}{\sum_j exp(z_j/T)} - \frac{exp(v_i/T)}{\sum_j exp(v_j/T)}\right)$$

Knowledge distillation

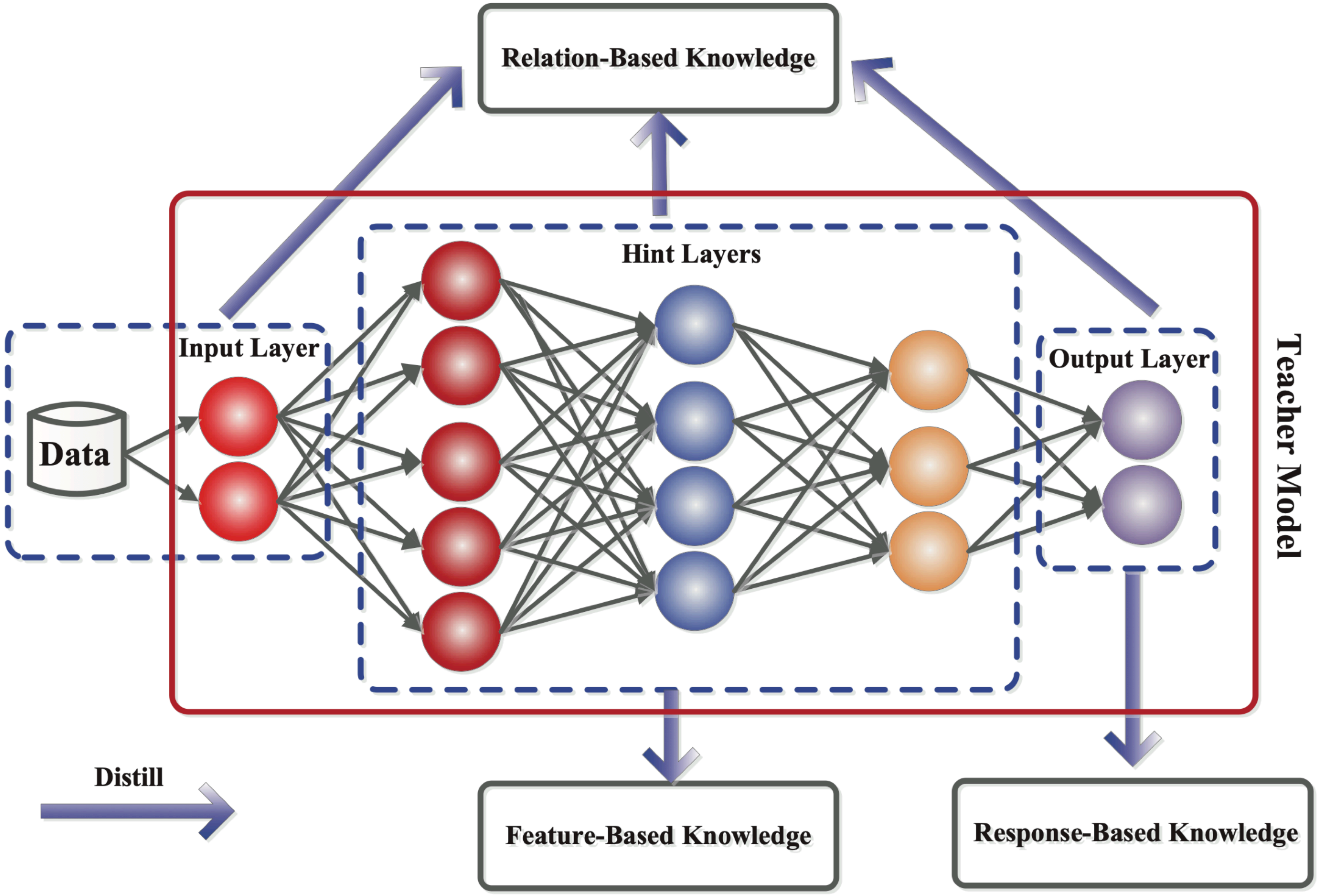
Apart from continual learning (on the next slides), why would we like to distill?



Gou et al, “Knowledge Distillation: A survey”, International Journal of Computer Vision 129, 2021

Knowledge distillation

We generally have various choices of what types of relationships we wish to distill (and how)



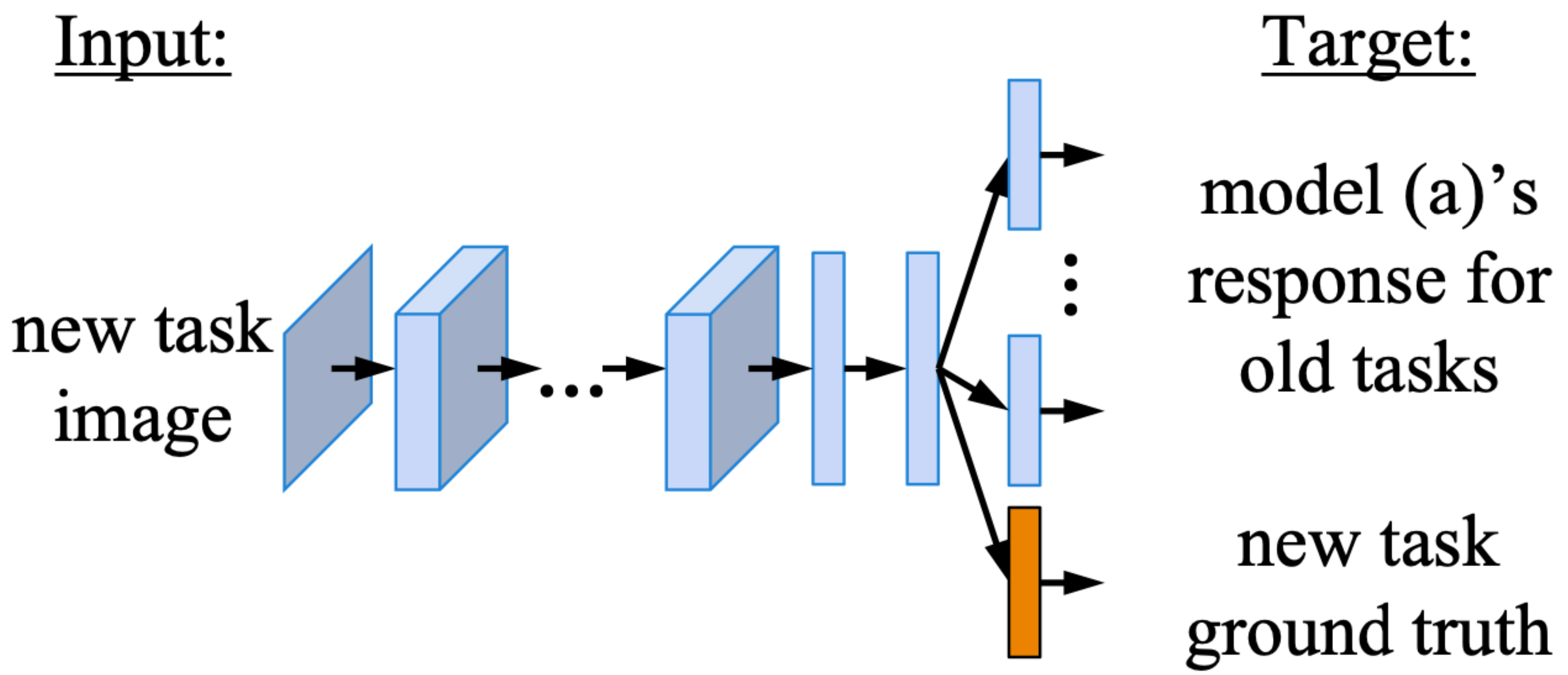
Gou et al, “Knowledge Distillation: A survey”, International Journal of Computer Vision 129, 2021

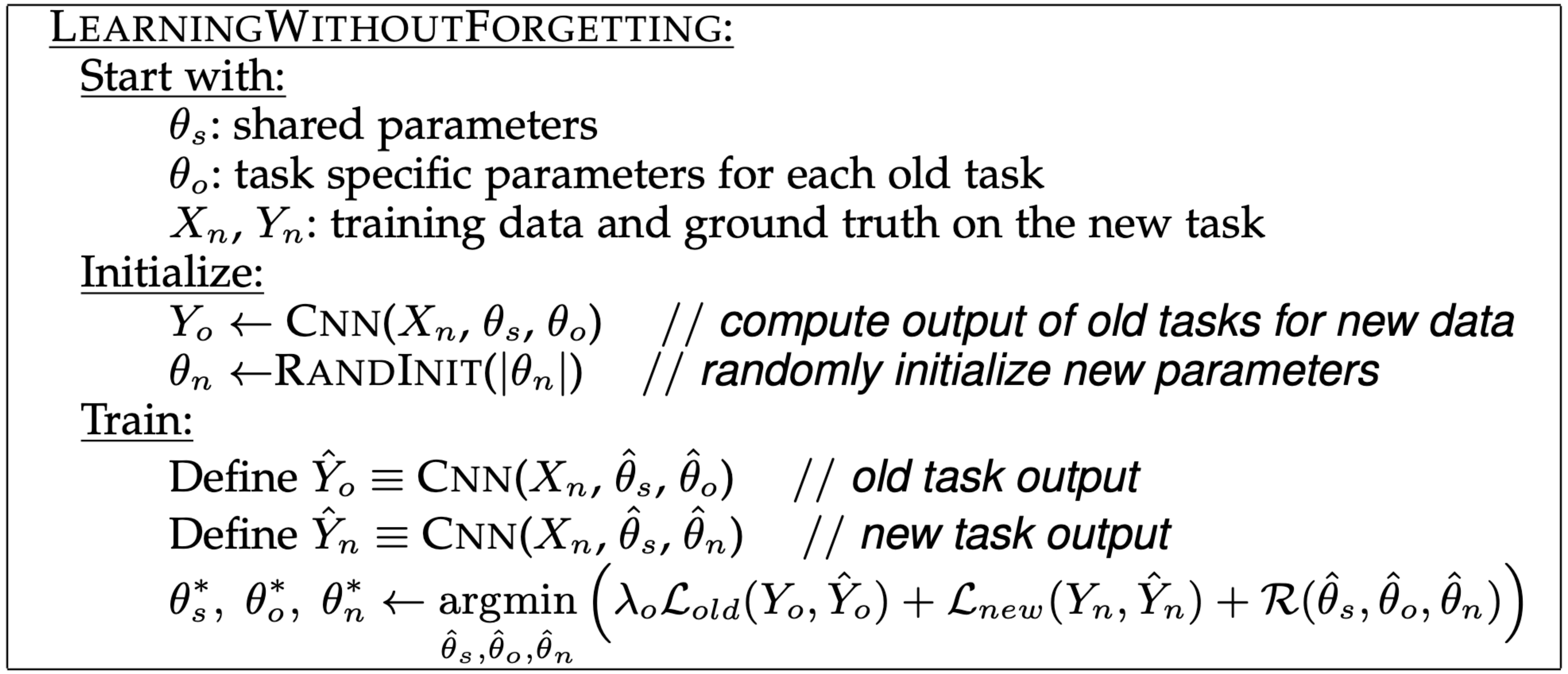
Continual knowledge distillation

Learning without forgetting

(Li & Hoiem, “Learning without Forgetting”, ECCV 2016)

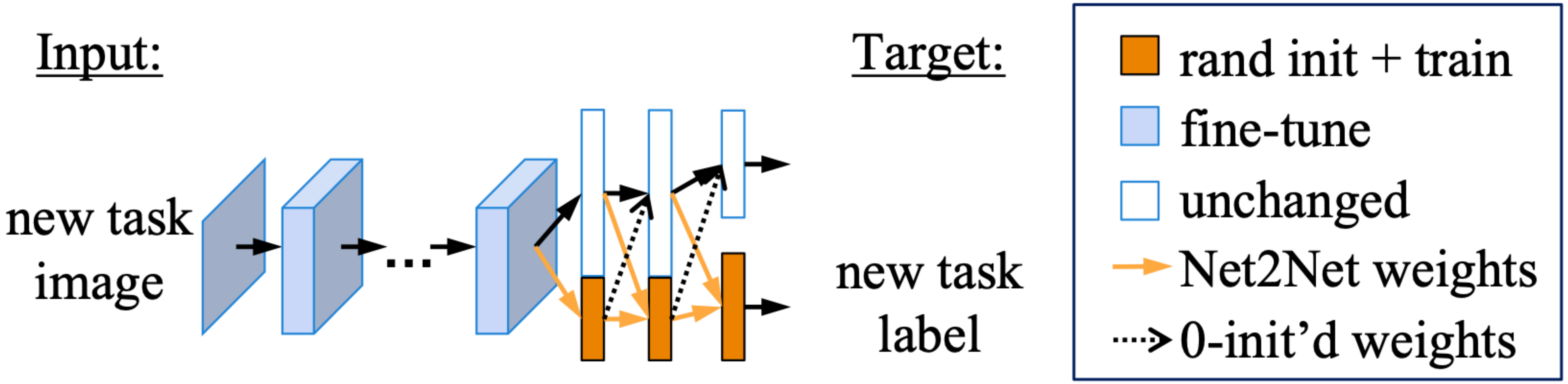
Key idea: compute task “head” with new data and continue to preserver this input-output relationship, while learning a new task “head” simultaneously





```
LEARNINGWITHOUTFORGETTING:
  Start with:
      $\theta_s$: shared parameters
      $\theta_o$: task specific parameters for each old task
      $X_n, Y_n$: training data and ground truth on the new task
  Initialize:
      $Y_o \leftarrow \textsc{Cnn}(X_n, \theta_s, \theta_o)$    // compute output of old tasks for new data
      $\theta_n \leftarrow \textsc{RandInit}(|\theta_n|)$    // randomly initialize new parameters
  Train:
      Define $\hat{Y}_o \equiv \textsc{Cnn}(X_n, \hat{\theta}_s, \hat{\theta}_o)$    // old task output
      Define $\hat{Y}_n \equiv \textsc{Cnn}(X_n, \hat{\theta}_s, \hat{\theta}_n)$    // new task output
      $\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left( \lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$
```

Continual knowledge distillation



Li & Hoiem, "Learning without Forgetting", ECCV 2016

But "cross-talk" can be challenging, if we don't dedicate an individual expert to each task

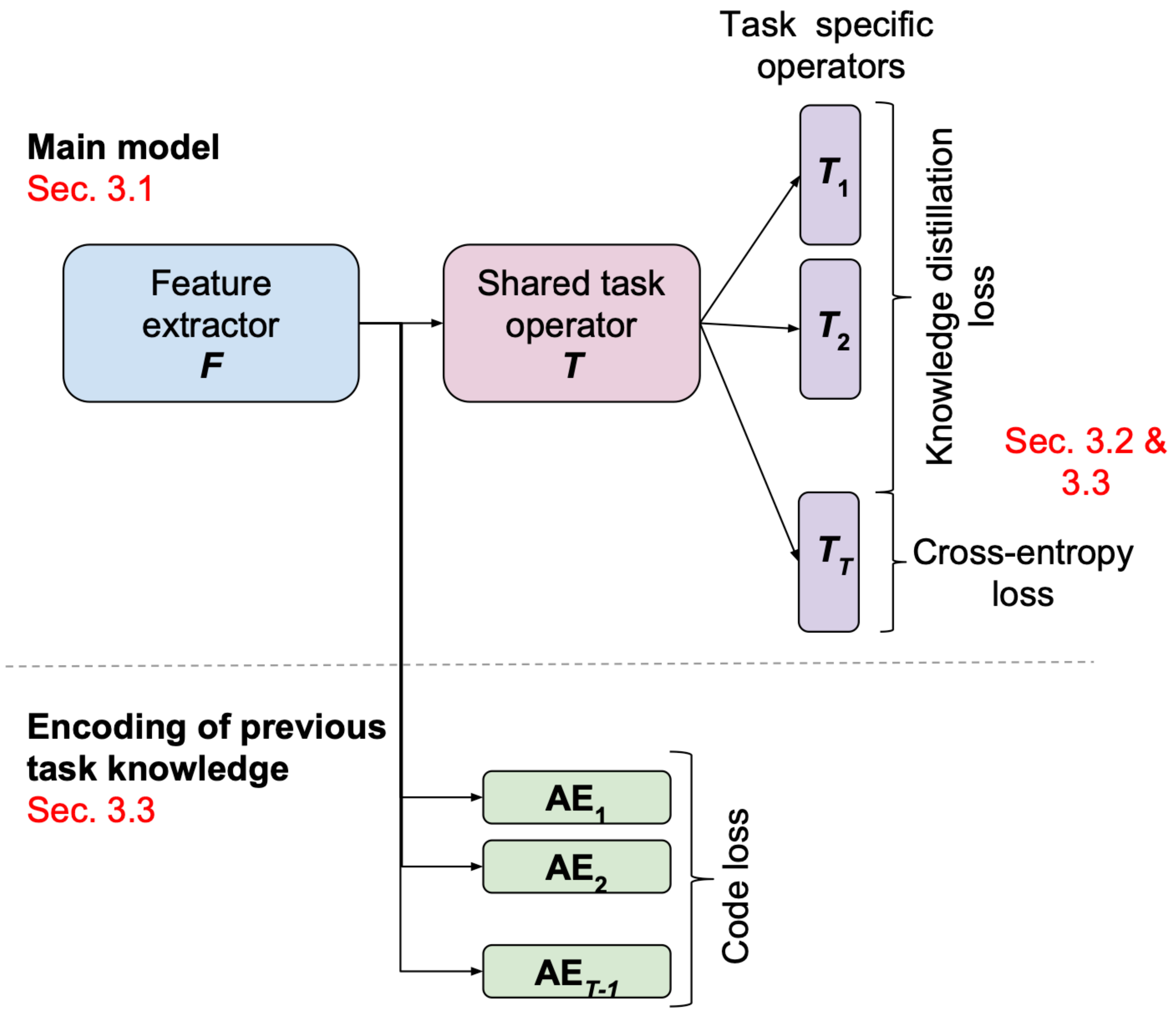
Especially true for e.g. Softmax layers that normalize over the entire output vector

Let's discuss if expert outputs are desirable when we learn about structure changes

Continual knowledge distillation

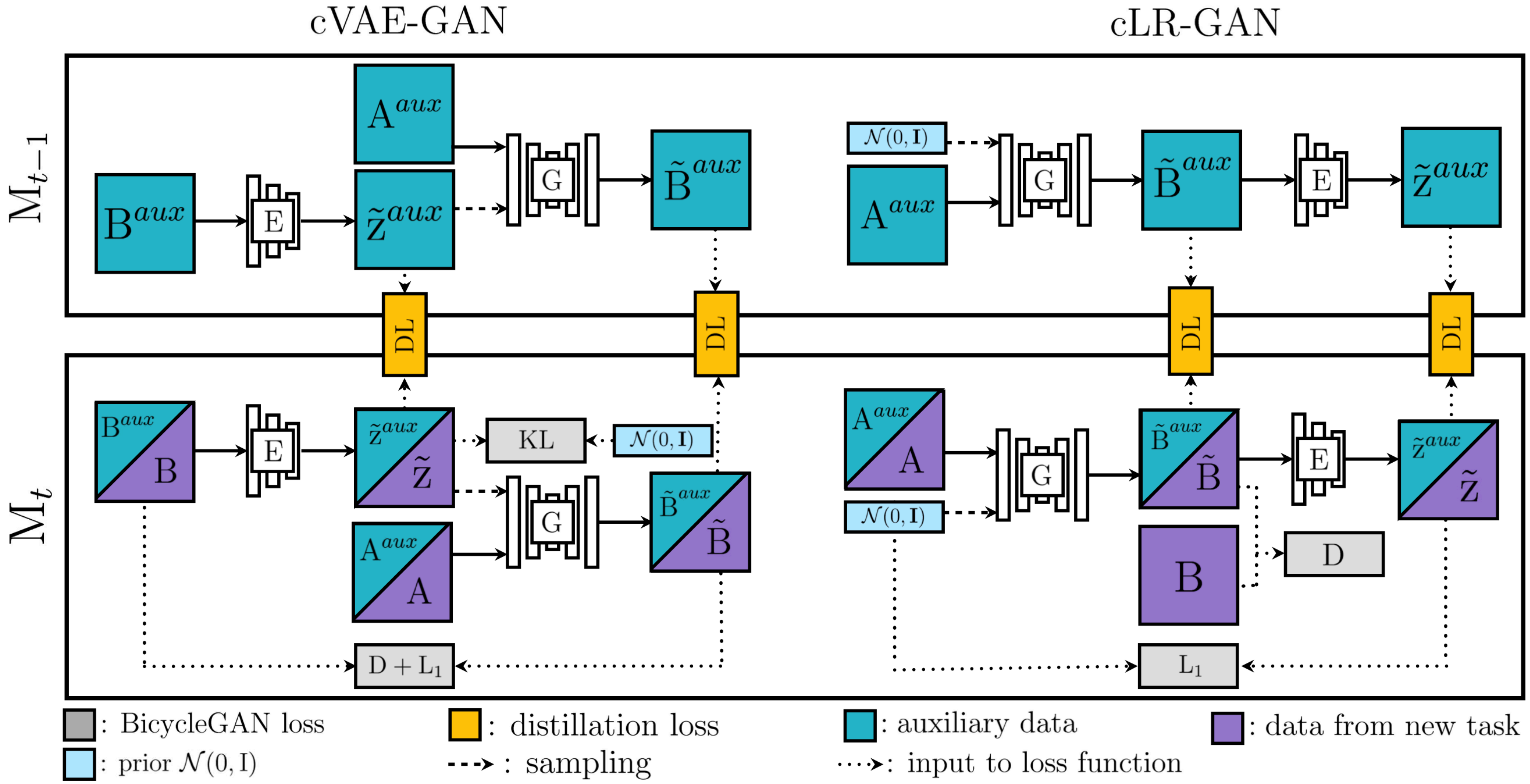
Has become very popular in continual learning & adapted in various ways

Has been extended to generative models, (shared feature spaces) etc.



Rannen & Aljundi et al, "Encoder Based Lifelong Learning", ICCV 2017

A larger scale example



Zhao & Chen et al, “Lifelong GAN: Continual Learning for Conditional Image Generation”, ICCV 2019

Distillation is regularly used together with other techniques, such as generative models, rehearsal, or architecture modifications/expansion -> we will discuss those next week