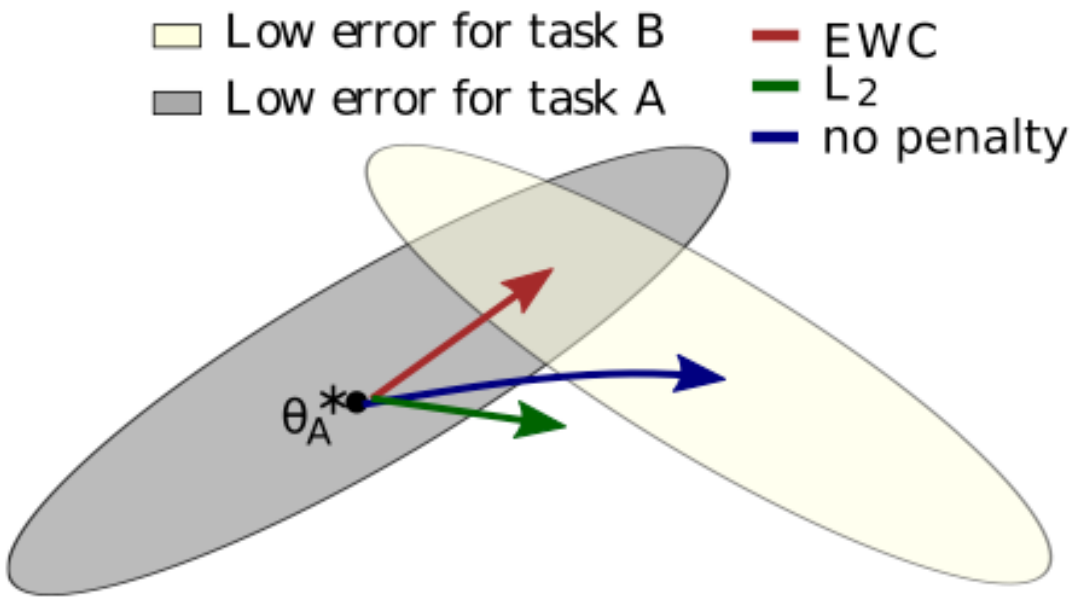
Universität Bremen



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

Part 2 - Retaining the Past

Optimization & Forgetting

Lifelong Machine Learning
Summer 2025

Prof. Dr. Martin Mundt

Early Definition

Definition - ~~Lifelong Machine Learning~~ - Thrun 1996:

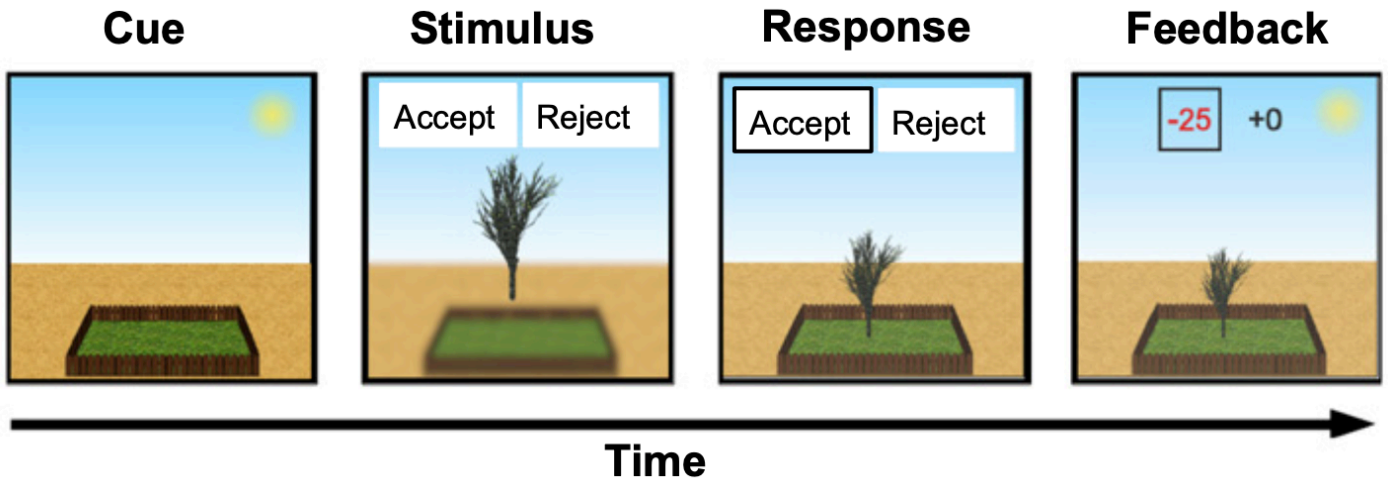
"*The system has performed N tasks. When faced with the (N+1)th task, it uses the knowledge gained from the N tasks to help the (N+1)th task.*"

"Is Learning The n-th Thing Any Easier Than Learning the First?" (NeurIPS 1996) & "Explanation based Neural Network Learning A Lifelong Learning Approach", Springer US, 1996

- The definition we've seen was actually called "lifelong learning"
- In practice, it corresponds to a narrow one for transfer learning
- It discounts what happens to the first N tasks (as is typical of transfer learning). However, there is also positive/negative backward transfer, which is critical for continual learning!

A study on how humans learn continually

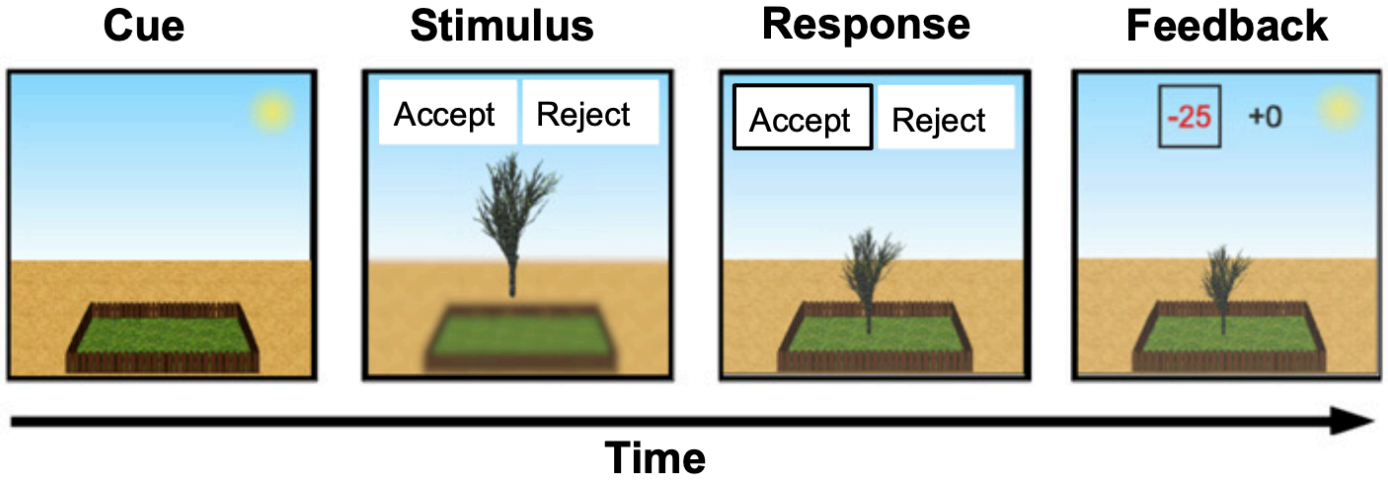
- Virtual gardening task with two different gardens (north & south)
- Via trial & error learn which type of tree grows best - based on an initial cue, a stimulus with blurred context, response + feedback

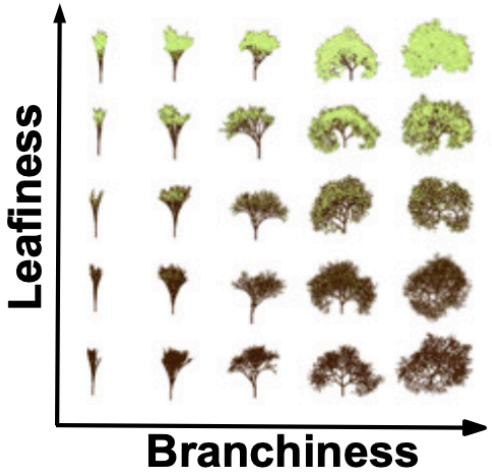


Flesch et al, “Comparing continual task learning in minds and machines”, PNAS 115, 2018

A study on how humans learn continually

- Virtual gardening task with two different gardens (north & south)
- Goal: via trial & error learn which type of tree grows best
- Initial cue, stimulus with blurred context, response -> feedback
- Trees varied along 2 dimensions & grow/shrink based on feedback

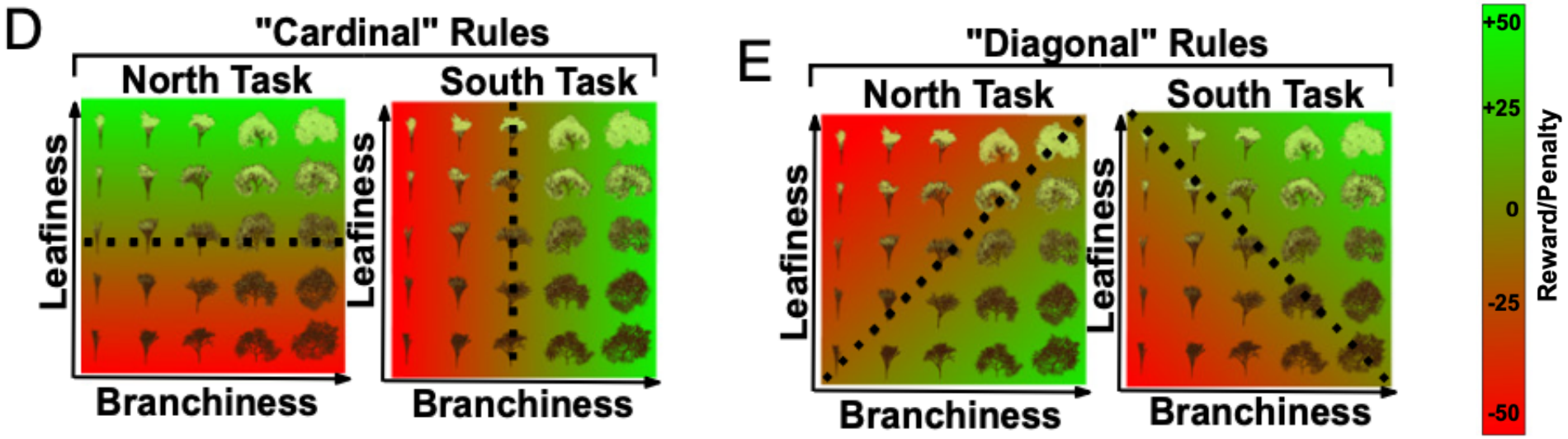




Flesch et al, “Comparing continual task learning in minds and machines”, PNAS 115, 2018

A study on how humans learn continually

- Unknown to participants, feature dimensions are mapped to rewards



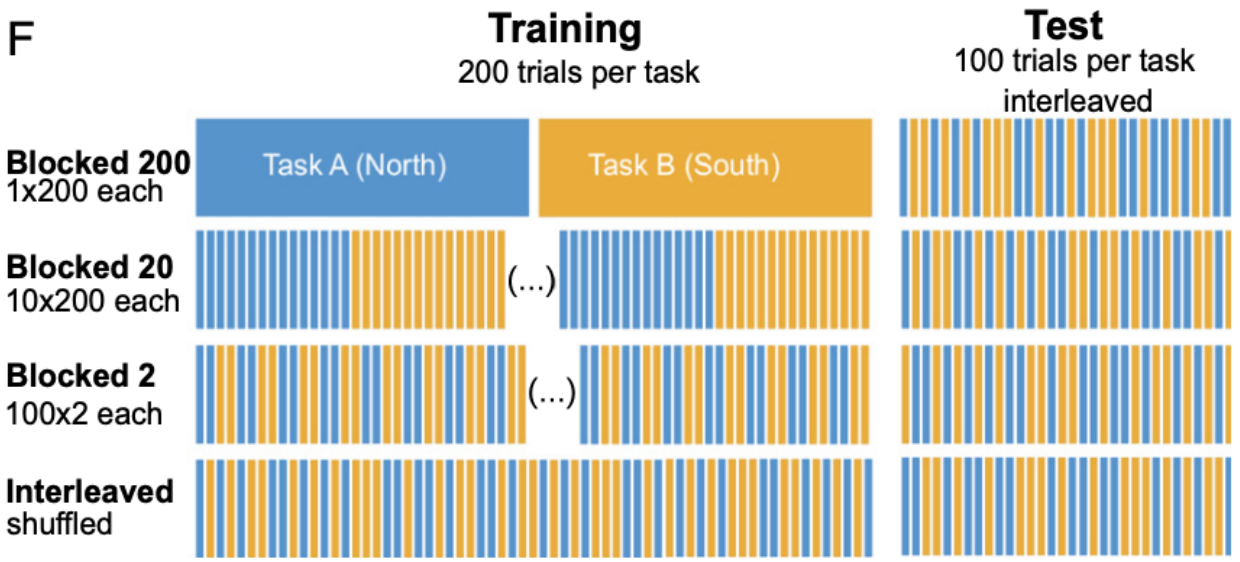
Flesch et al, “Comparing continual task learning in minds and machines”, PNAS 115, 2018

A study on how humans learn continually

- Unknown to participants, feature dimensions are mapped to rewards

Most importantly:

- Participant groups were “trained” on different temporal correlation
- All were tested on interleaved tasks



Flesch et al, “Comparing continual task learning in minds and machines”, PNAS 115, 2018

Universität Bremen

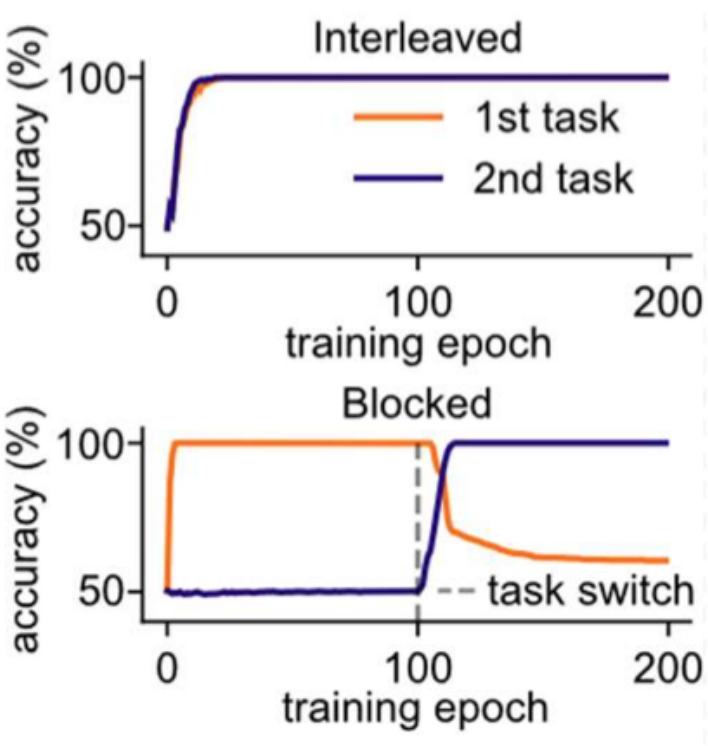
Summary of the study's findings

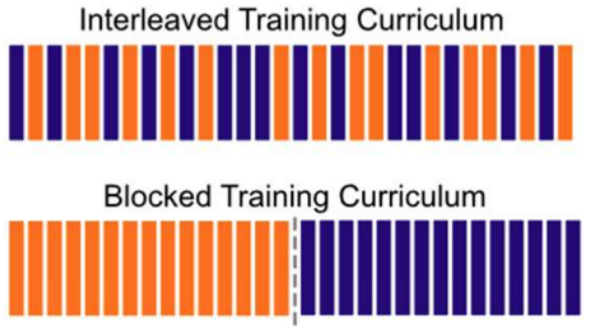
"Humans, but not machines, seem to benefit from training regimes that blocked one task at a time, especially when they had a prior bias to represent stimuli in a way that encouraged task separation"

Flesch et al, "Comparing continual task learning in minds and machines", PNAS 115, 2018

Summary of the study's findings

"Humans, but not machines, seem to benefit from training regimes that blocked one task at a time, especially when they had a prior bias to represent stimuli in a way that encouraged task separation"

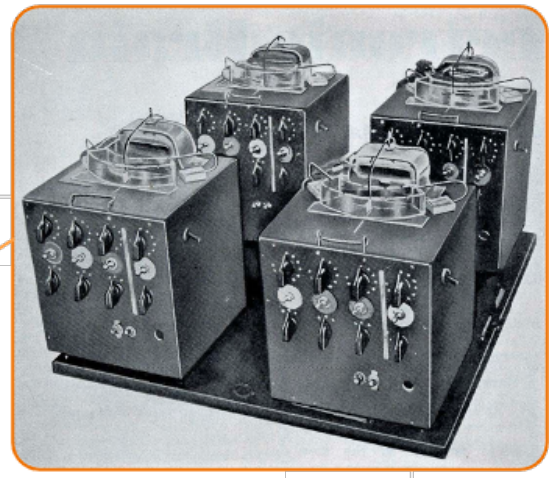


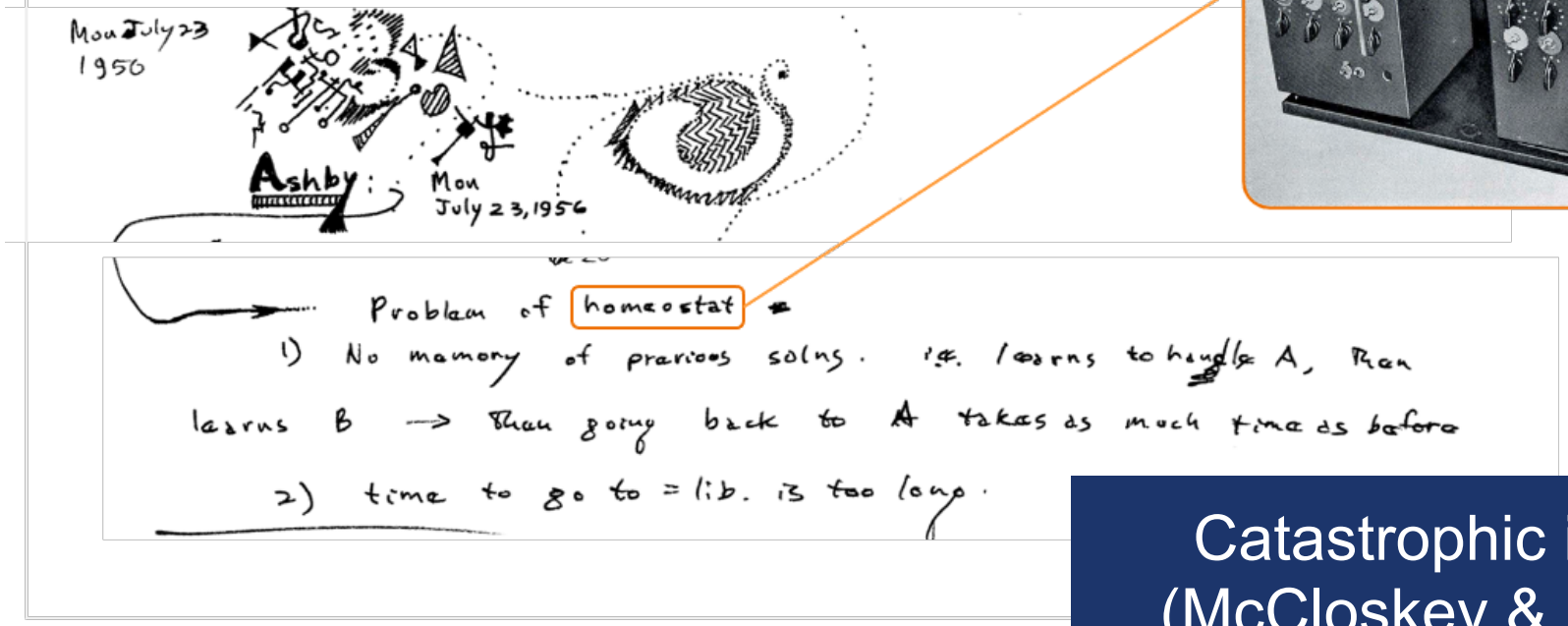


Flesch et al, "Comparing continual task learning in minds and machines", PNAS 115, 2018

The forgetting phenomenon in ML is well known

Ray Solomonoff's notes on Ross Ashby's talk, Dartmouth 1956





Catastrophic interference (McCloskey & Cohen 1989)

Universität Bremen

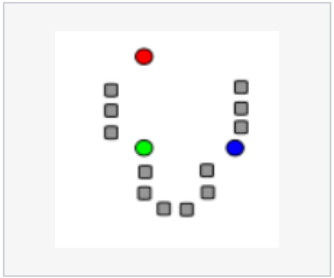
OWL ML

Question time

Didn't we learn that transfer learning is beneficial?
Why does ML model forget & is it surprising?

An intuitive example you know: K-means

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



Demonstration of the standard algorithm

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

An intuitive example you know: K-means

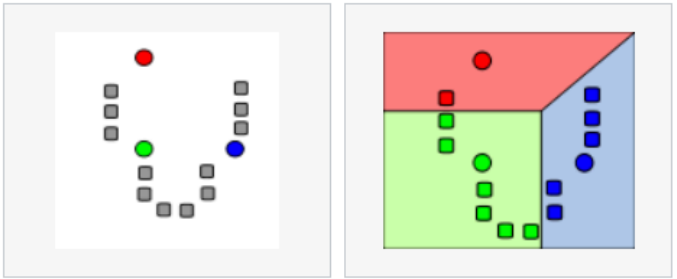
Given an initial set of k means $m_1^{(1)},\ldots,m_k^{(1)}$ (see below), the algorithm proceeds by alternating between two steps:[7]

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.)

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

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

Demonstration of the standard algorithm



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

An intuitive example you know: K-means

Given an initial set of k means $m_1^{(1)},\ldots,m_k^{(1)}$ (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.)

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

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.

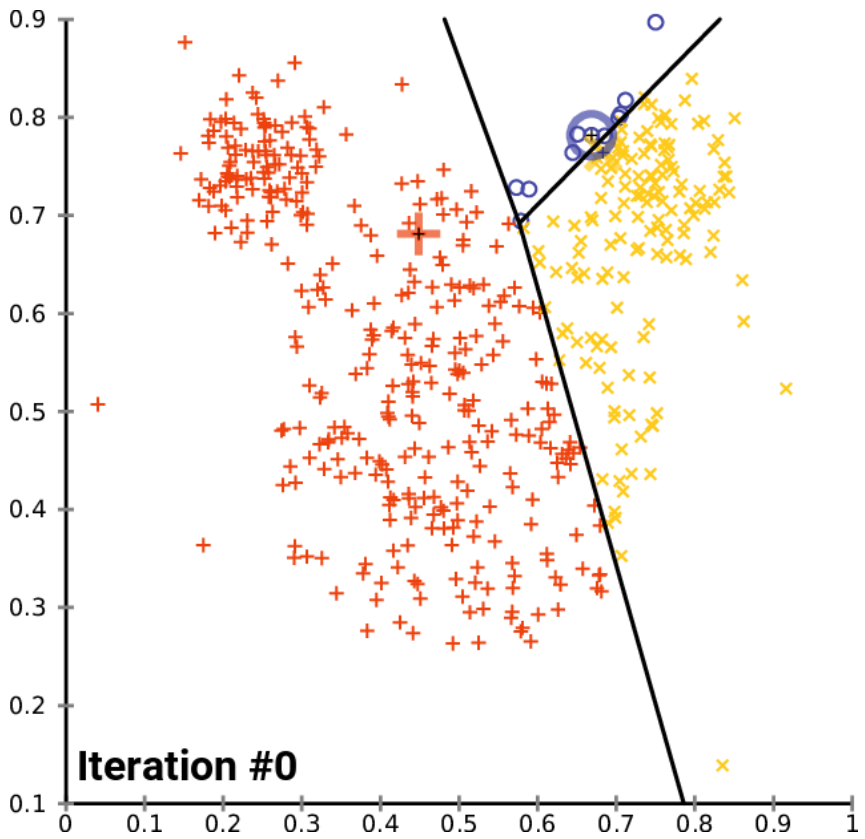
$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

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

Universität Bremen

Is forgetting surprising?

Imagine you “blocked” training like in our study. What would happen?



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

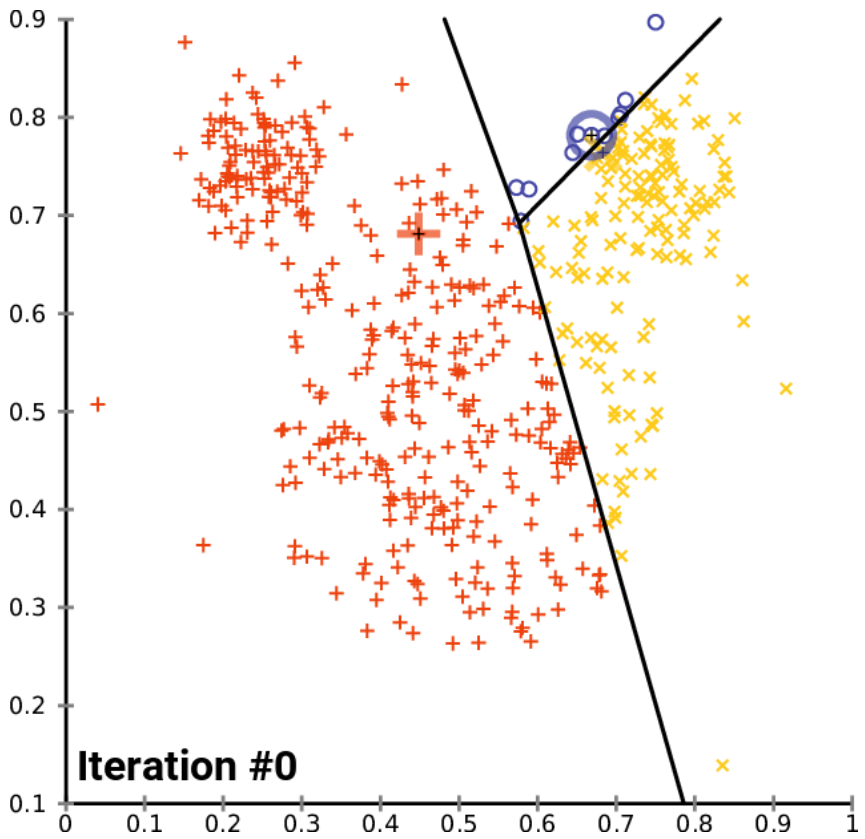
Is forgetting surprising?

Imagine you “blocked” training like in our study. What would happen?

If data isn’t accumulated but replaced, the mean changes abruptly!

If the number of clusters changes over time, it will also be problematic

We could consider an exponential moving average, but it remains affected



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

Universität Bremen

OWL ML

Question time

The study trained artificial neural networks.
What are the origins of forgetting in NNs
(and similarly functioning ML models)?

Recall intro lecture: optimization, risk & loss

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

We couldn't directly solve this

- Cannot actually compute above risk (don't know the distribution)

Recall intro lecture: optimization, risk & loss

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

We couldn't directly solve this

- Cannot actually compute above risk (don't know the distribution)

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

So we resorted to empirical samples

-> look at true & unknown response & predictions $\delta(x)$ given input x

Recall intro lecture: optimization, risk & loss

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

We couldn't directly solve this

- Cannot actually compute above risk (don't know the distribution)

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

So we resorted to empirical samples

-> look at true & unknown response & predictions $\delta(x)$ given input x

We still do not know the true distribution -> estimate:

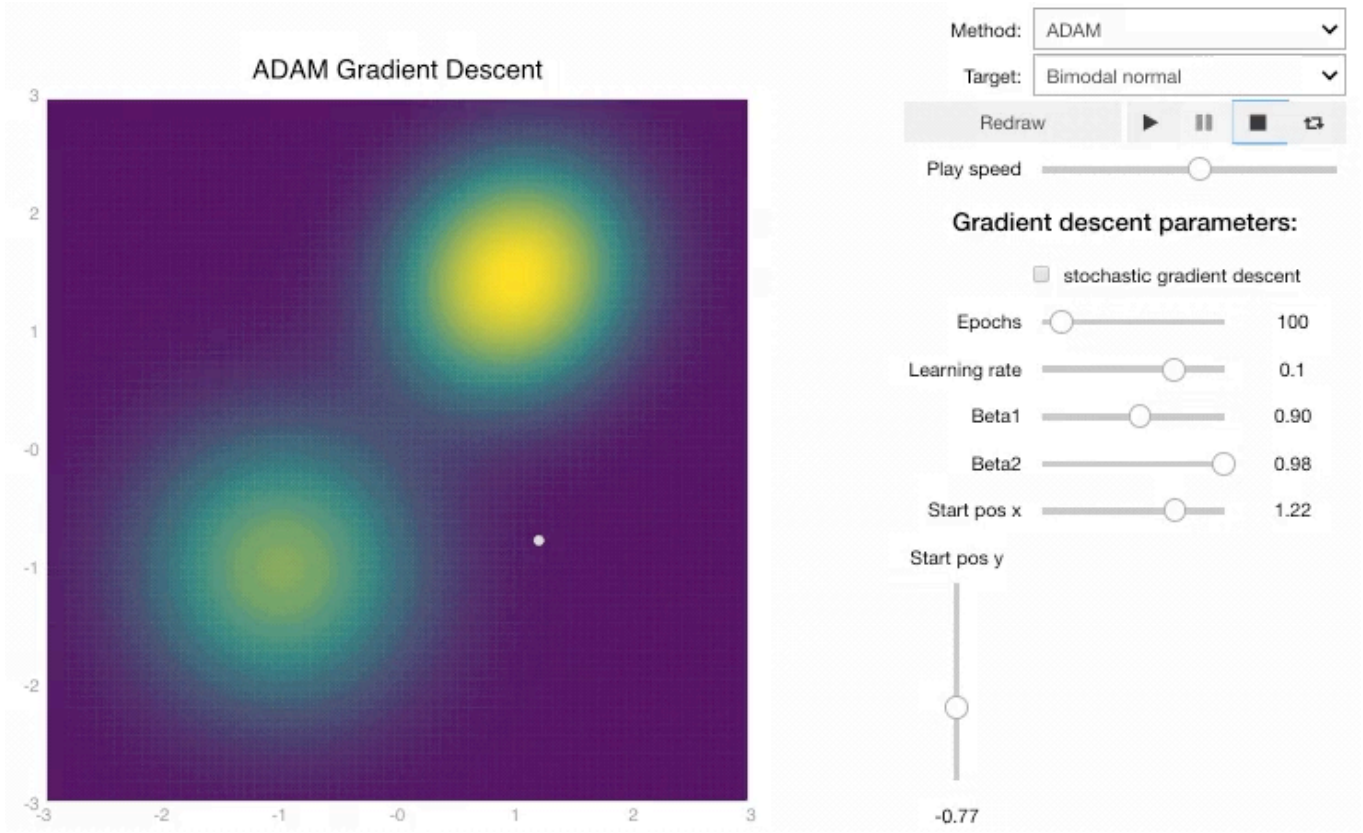
$$R_{emp}(D, \delta) = 1/N \sum_{i=1}^{N} L(y_i, \delta(x_i))$$

And finally average over predictions

Universität Bremen

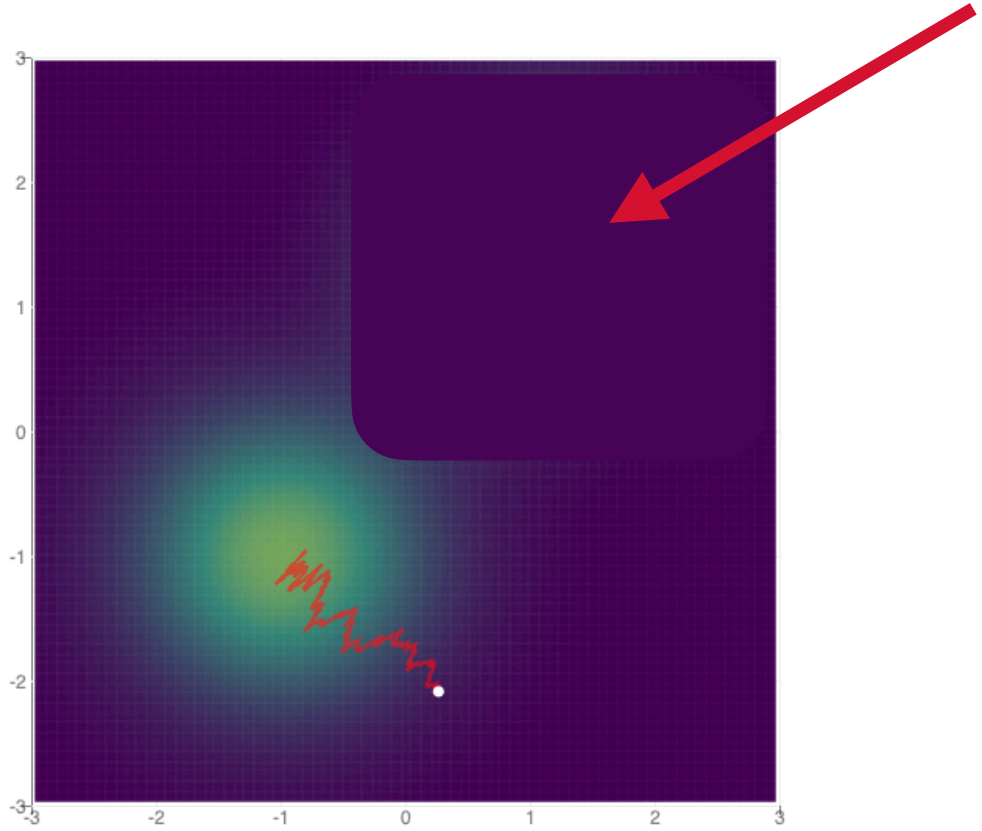
(Stochastic) gradient descent at play

An example with two extrema



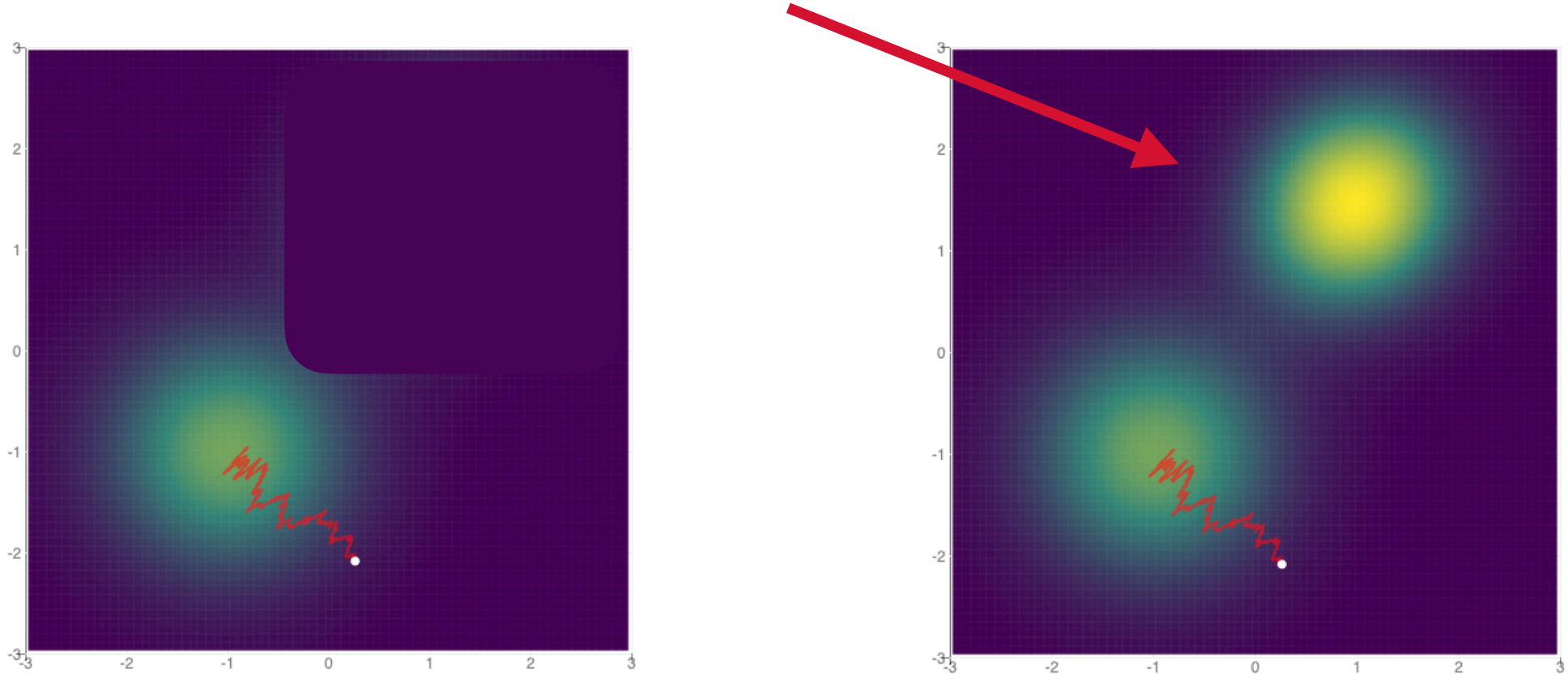
(Stochastic) gradient descent at play

Imagine one extremum disappeared, because of a "blocked task 2”



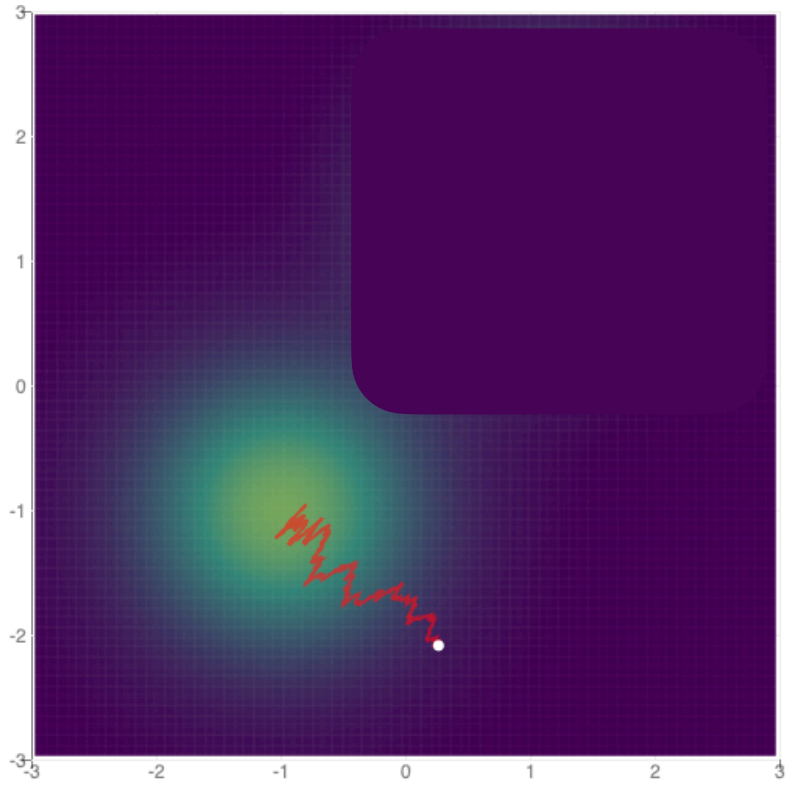
(Stochastic) gradient descent at play

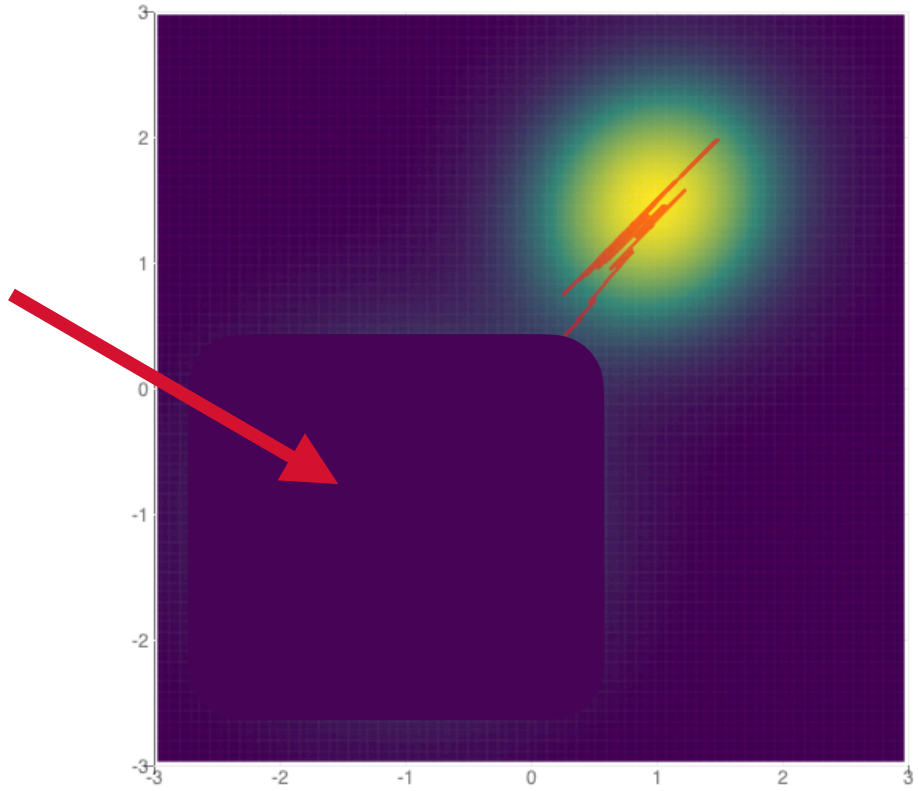
But eventually it gets observed when the “task 2 block” arrives



(Stochastic) gradient descent at play

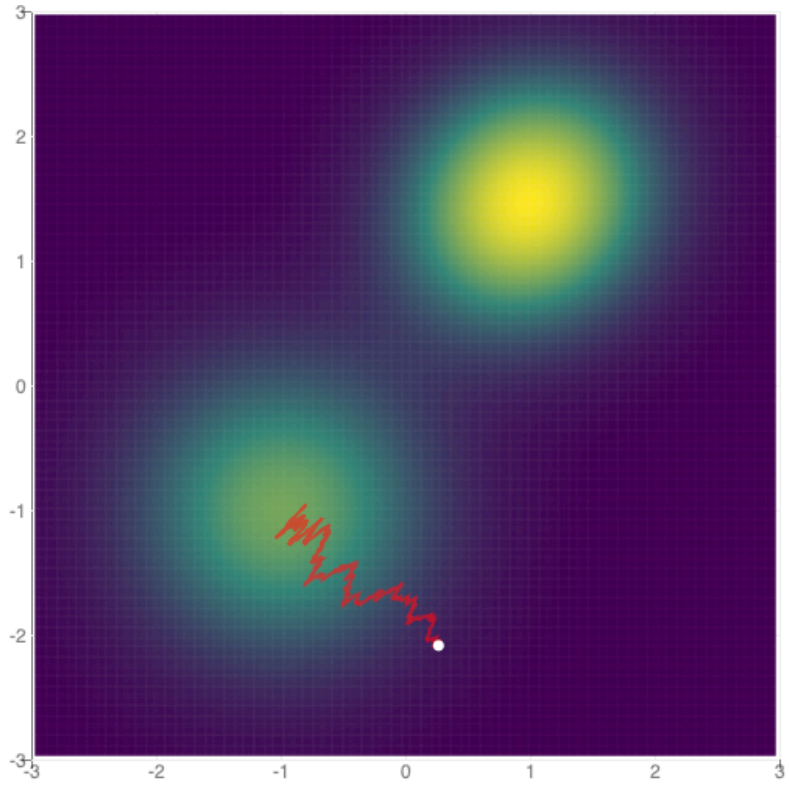
Which also means we don’t actually see the “task 1 block” anymore

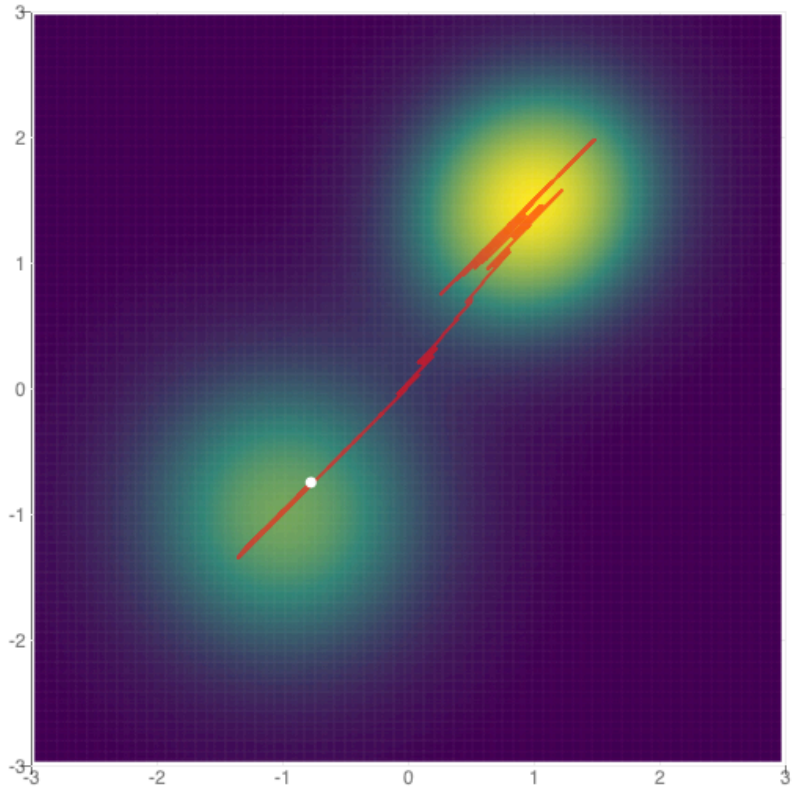




(Stochastic) gradient descent at play

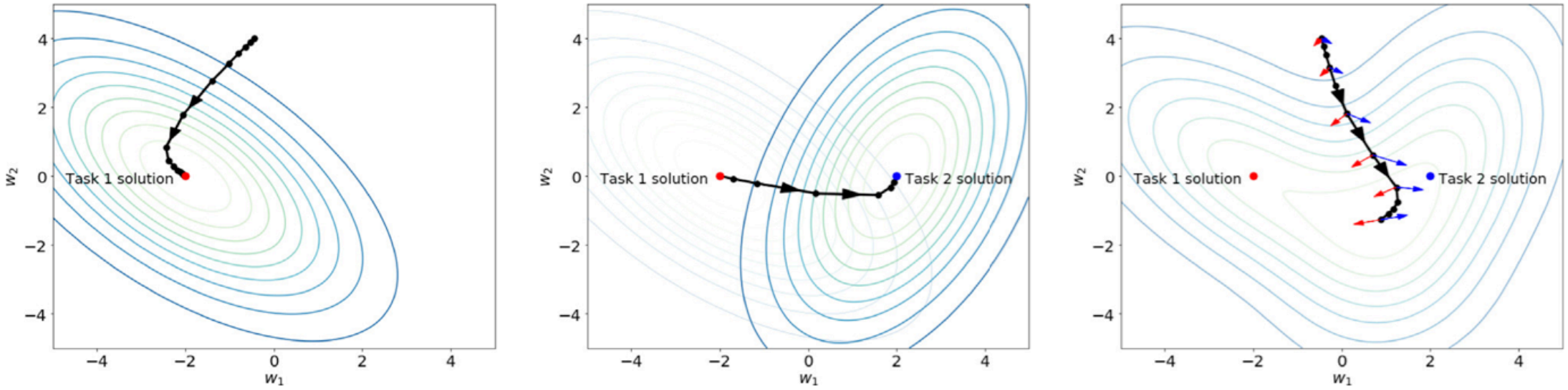
If data got accumulated (curriculum), a solution will depend on noise





Typically called: stability - plasticity dilemma

We have to “balance” stability with plasticity (also called sensitivity)
(already described in Hebb 1949, “the organization of behavior)



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

Universität Bremen

OWL ML

Question time

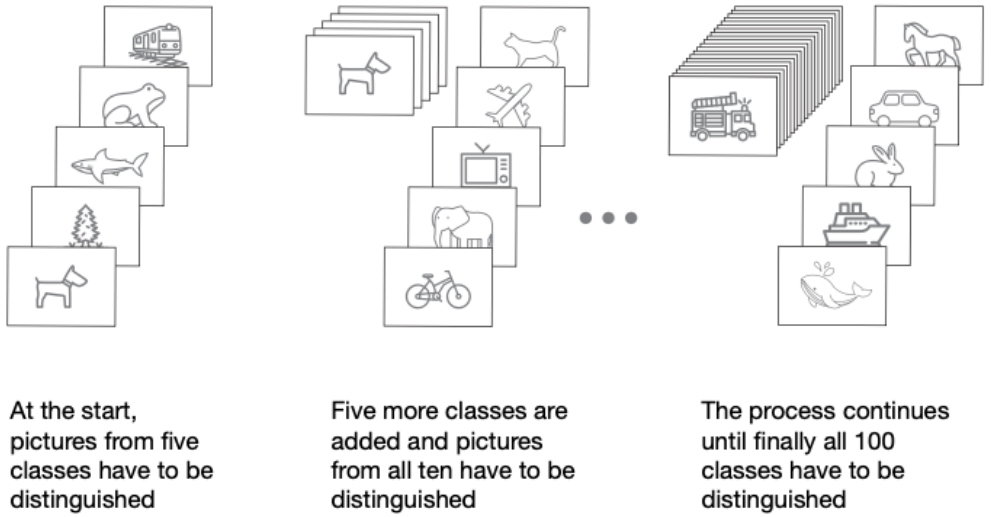
How could we alleviate forgetting? How do we overcome the stability-plasticity dilemma?

The trivial solution?

Large amount of parameters + accumulate data (grow the dataset)

The trivial solution? It's not so simple

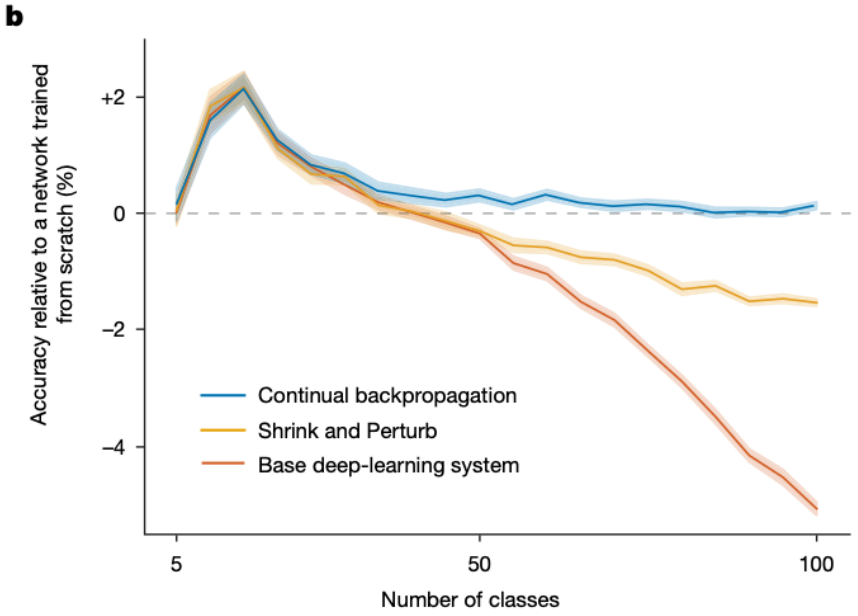
A note on maintaining *only plasticity* - without considering stability
Experiment: run a class-based curriculum while growing the dataset

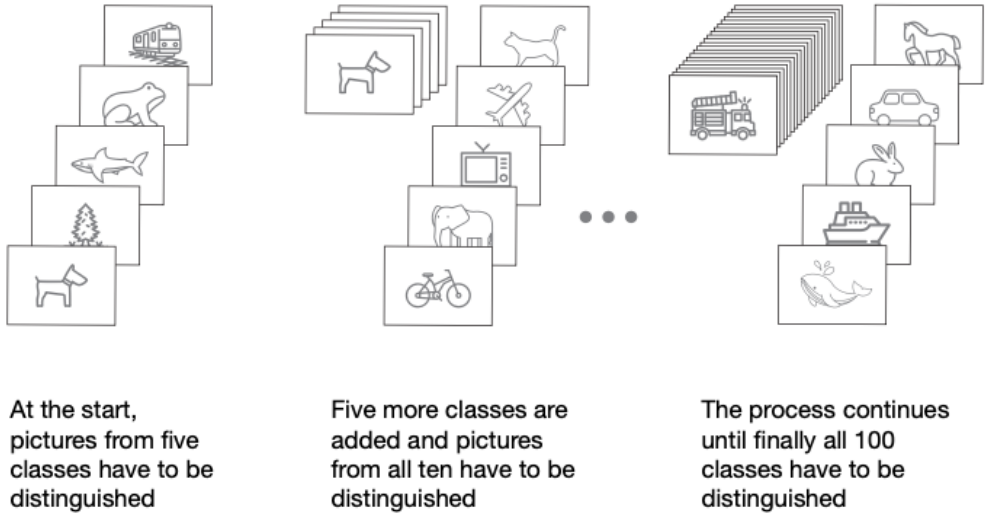


Dohare et al, "Loss of plasticity in deep continual learning", Nature 632, 2024

The trivial solution? It's not so simple

A note on maintaining *only plasticity* - without considering stability
Experiment: run a class-based curriculum while growing the dataset

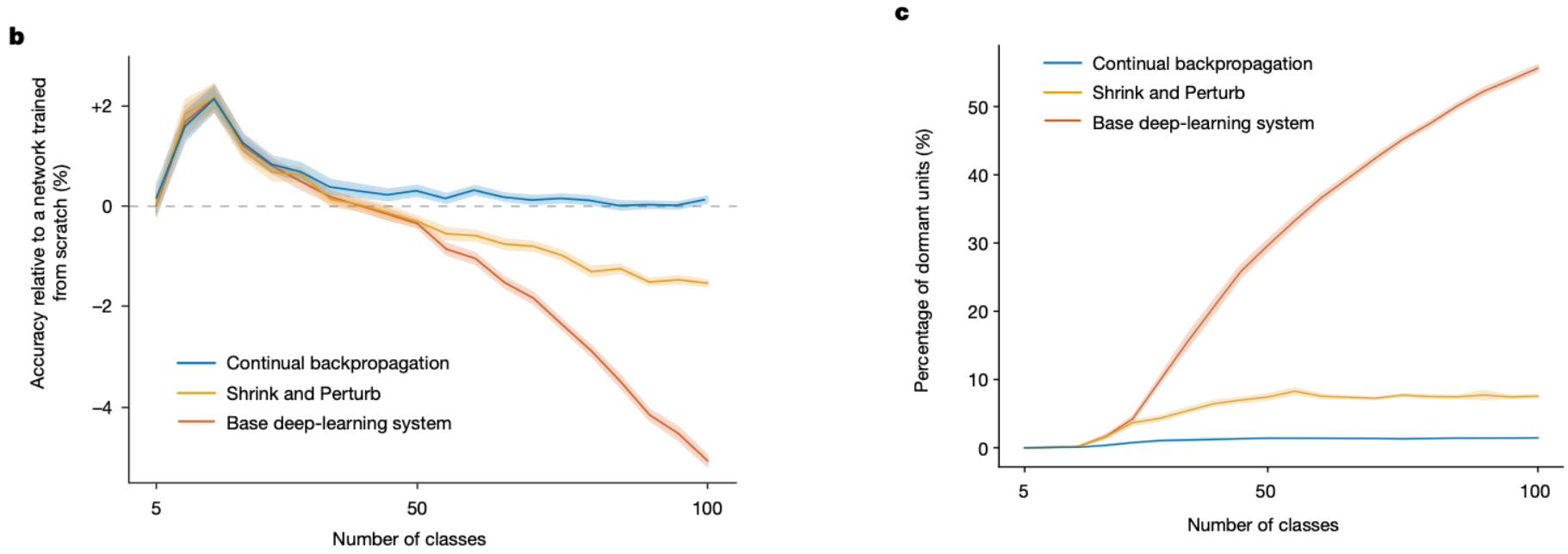




Dohare et al, "Loss of plasticity in deep continual learning", Nature 632, 2024

Loss of plasticity

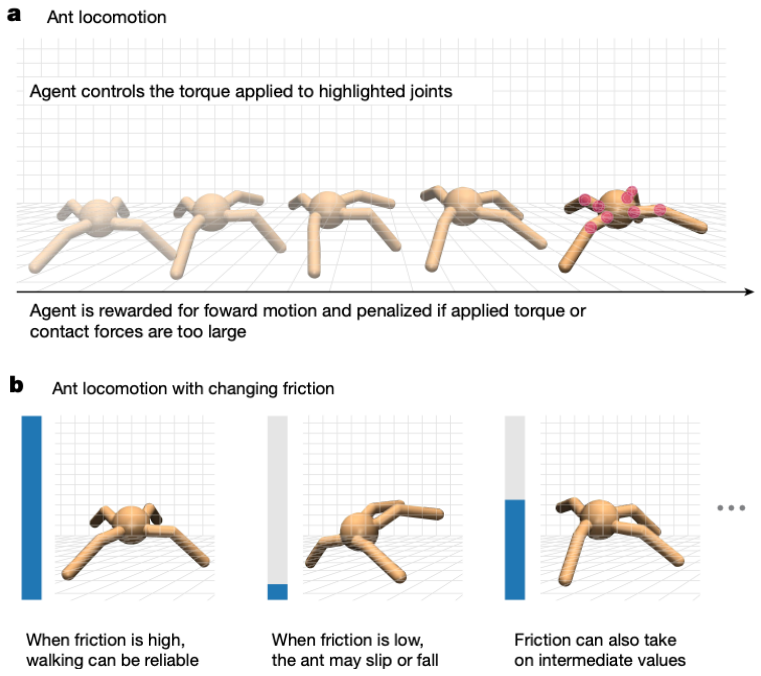
Models “get stuck”. Specifically, the number of units that are active less than 1% of the time (“dormant” units) increases rapidly

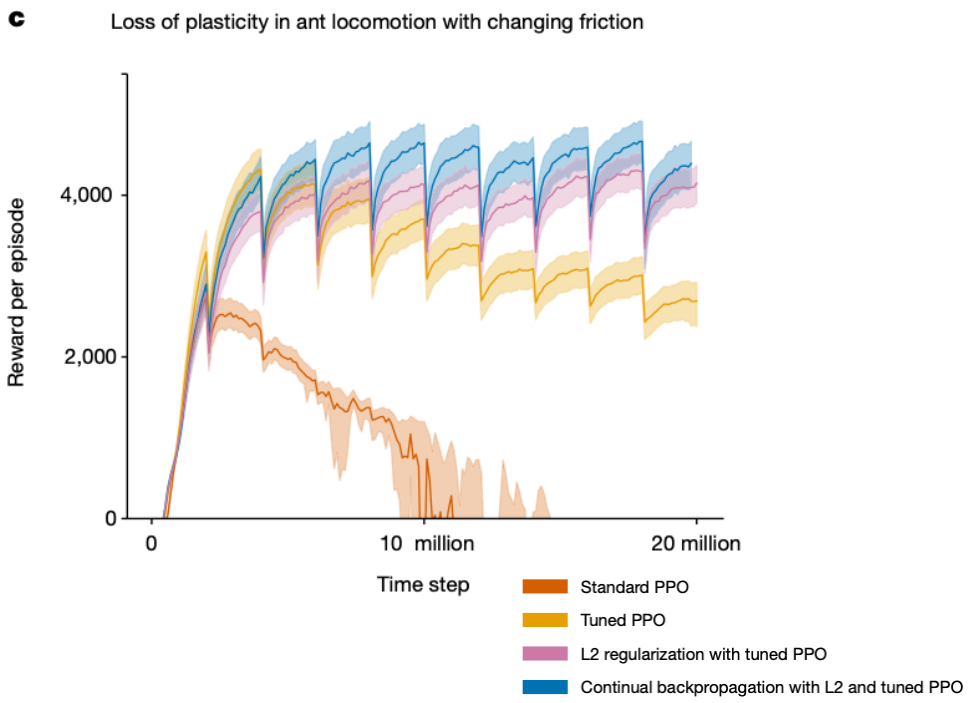


Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

Loss of plasticity

With analogous empirical observations in reinforcement learning





Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

What does “continual backprop” change?

Continual backprop selectively reinitializes “low-utility” units

A unit i’s contribution utility u_i relies on the magnitude of the product of units’ activations (h) and outgoing weights ($w_{i,k}$) at time t:

$$u_i = \eta u_i + (1 - \eta) \left| h_{i,t} \right| \sum_{k=1}^{K} \left| w_{i,k,t} \right| \text{ (eq. 1 in the algorithm)}$$

Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

What does “continual backprop” change?

Continual backprop selectively reinitializes “low-utility” units

A unit i’s contribution utility u_i relies on the magnitude of the product of units’ activations (h) and outgoing weights ($w_{i,k}$) at time t:

$$u_i = \eta u_i + (1 - \eta) \, |h_{i,t}| \sum_{k=1}^{K} |w_{i,k,t}| \quad \text{(eq. 1 in the algorithm)}$$

Re-initialization means setting weights to zero, avoiding affecting the already learned function

Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

What does “continual backprop” change?

To avoid direct reinitialization, units are protected given their “age”

Finally, units with the smallest utility (e.g. dormant) are replaced

Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

What does “continual backprop” change?

To avoid direct reinitialization, units are protected given their “age”

Finally, units with the smallest utility (e.g. dormant) are replaced

Algorithm 1. Continual backpropagation for a feed-forward network with L layers

Set replacement rate ρ, decay rate η and maturity threshold m
Initialize the weights $\mathbf{w}_0, \ldots, \mathbf{w}_{L-1}$, in which $\mathbf{w}_l$ is sampled from distribution d_l
Initialize utilities $\mathbf{u}_1, \ldots, \mathbf{u}_{L-1}$, number of units to replace $c_1, \ldots, c_{L-1}$, and ages $\mathbf{a}_1, \ldots, \mathbf{a}_{L-1}$ to 0
For each input $\mathbf{x}_t$ **do**
Forward pass: pass $\mathbf{x}_t$ through the network to get the prediction $\hat{\mathbf{y}}_t$
Evaluate: receive loss $l(\mathbf{x}_t, \hat{\mathbf{y}}_t)$
Backward pass: update the weights using SGD or one of its variants
For layer l in $1: L-1$ **do**
Update age: $\mathbf{a}_l = \mathbf{a}_l + 1$
Update unit utility: see equation (1)
Find eligible units: n_{eligible} = number of units with age greater than m
Update number of units to replace: $c_l = c_l + n_{\text{eligible}} \times \rho$
If $c_l > 1$
Find the unit with smallest utility and record its index as r
Reinitialize input weights: resample $\mathbf{w}_{l-1}[:, r]$ from distribution d_l
Reinitialize output weights: set $\mathbf{w}_l[r, :]$ to 0
Reinitialize utility and age: set $\mathbf{u}_l[r] = 0$ and $\mathbf{a}_l[r] = 0$
Update number of units to replace: $c_l = c_l - 1$
End For
End For

Dohare et al, “Loss of plasticity in deep continual learning”, Nature 632, 2024

Question time

This definition of utility is crude & we still need to consider stability. How should we measure utility?

Universität Bremen

The stability - plasticity dilemma

*“There exists in the mind of man a block of wax … harder, moister, and having more or less of purity in one than another… **the soft are good at learning**, **but apt to forget**; and **the hard are the reverse**”*

Plato - Theaetetus, circa 369 BCE

The stability - plasticity dilemma

*“There exists in the mind of man a block of wax … harder, moister, and having more or less of purity in one than another… **the soft are good at learning**, **but apt to forget**; and **the hard are the reverse**”*

Plato - Theaetetus, circa 369 BCE

In addition to worrying about plasticity, let us find a way to quantify which units *should be constrained from future learning*

Elastic Weight Consolidation (EWC)

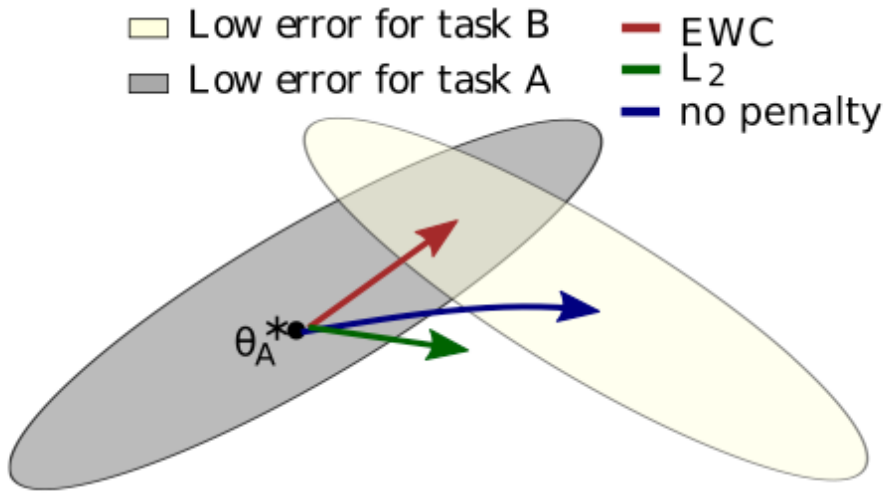
Motivation: many configs of θ result in the same performance and over-parametrization makes it likely that there is a solution for a task B, θ^*_B, that is close to a previously found solution for task A, θ^*_A.

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

Elastic Weight Consolidation (EWC)

Motivation: many configs of θ result in the same performance and over-parametrization makes it likely that there is a solution for a task B, θ^*_B, that is close to a previously found solution for task A, θ^*_A.

Intuition: While learning task B, EWC protects performance of task A by constraining the parameters to stay in a region of low error centered around θ^*_A



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

Elastic Weight Consolidation (EWC)

In practice: implement constraint (regularization term) as a quadratic loss penalty to “spring anchor” (hence “elastic”) the parameters to the previous solution -> identify the strength of the anchoring for every individual parameter based on its importance to task A

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

Elastic Weight Consolidation (EWC)

In practice: implement constraint (regularization term) as a quadratic loss penalty to “spring anchor” (hence “elastic”) the parameters to the previous solution -> identify the strength of the anchoring for every individual parameter based on its importance to task A

Formally: start with Bayesian optimization framing to finding the most probable value of parameters given the data $p(\theta \mid D)$.

Given a prior probability & Bayes rule: $p(\theta \mid D) = \frac{p(D \mid \theta)p(\theta)}{p(D)}$

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

Derivation of EWC

Given a prior probability & Bayes rule: $p(\theta \mid D) = \frac{p(D \mid \theta) p(\theta)}{p(D)}$

Using logarithms: $\log p(\theta \mid D) = \log p(D \mid \theta) + \log p(\theta) - \log p(D)$

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

Derivation of EWC

Given a prior probability & Bayes rule: $p(\theta \mid D) = \frac{p(D \mid \theta) p(\theta)}{p(D)}$

Using logarithms: $\log p(\theta \mid D) = \log p(D \mid \theta) + \log p(\theta) - \log p(D)$

We can assume the data is split into two independent parts for the tasks A & B in sequence and can then re-arrange:

$$\begin{aligned} \log p(\theta \mid D) &= \log p(D_B \mid D_A, \theta) + \log p(\theta \mid D_A) - \log p(D_B \mid D_A) \\ &= \log p(D_B \mid \theta) + \log p(\theta \mid D_A) - \log p(D_B) \end{aligned}$$

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

Derivation of EWC

$$\begin{aligned}\log p(\theta \mid D) &= \log p(D_B \mid D_A, \theta) + \log p(\theta \mid D_A) - \log p(D_B \mid D_A) \\ &= \log p(D_B \mid \theta) + \log p(\theta \mid D_A) - \log p(D_B)\end{aligned}$$

We now have a loss for task B, a likelihood for D_B and the posterior for A, $p(\theta \mid D_A)$, has become the prior for task B.

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

Derivation of EWC

$$\begin{aligned}\log p(\theta \mid D) &= \log p(D_B \mid D_A, \theta) + \log p(\theta \mid D_A) - \log p(D_B \mid D_A) \\ &= \log p(D_B \mid \theta) + \log p(\theta \mid D_A) - \log p(D_B)\end{aligned}$$

We now have a loss for task B, a likelihood for D_B and the posterior for A, $p(\theta \mid D_A)$, has become the prior for task B.

The posterior for A - with information about the parameters to explain task A - is intractable and we need to approximate it.

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

Derivation of EWC: approximations

Let us look at the EWC approximations:
First, the quadratic penalty in EWC originates from a second order Taylor expansion of $\mathcal{L}(\theta)$ around θ_A^*:

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta_A^*) + \frac{\delta \mathcal{L}(\theta)}{\delta \theta}|_{\theta_A^*} + \frac{1}{2}(\theta - \theta_A^*)^T(\frac{\delta^2 \mathcal{L}(\theta)}{\delta^2 \theta}|_{\theta_A^*})(\theta - \theta_A^*) + \ldots$$

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

Derivation of EWC: approximations

Let us look at the EWC approximations:
First, the quadratic penalty in EWC originates from a second order Taylor expansion of $\mathscr{L}(\theta)$ around θ^*_A:

$$\mathscr{L}(\theta) \approx \mathscr{L}(\theta^*_A) + \frac{\delta \mathscr{L}(\theta)}{\delta \theta}\Big|_{\theta^*_A} + \frac{1}{2}(\theta - \theta^*_A)^T \left(\frac{\delta^2 \mathscr{L}(\theta)}{\delta^2 \theta}\Big|_{\theta^*_A}\right)(\theta - \theta^*_A) + \ldots$$

together with a *Laplace approximation*: assuming $p(\theta \mid A)$ is peaked around θ^*_A and approximating it with a normal distribution with mean θ^*_A and variance $[\mathbb{I}_A]^{-1}$.

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

Derivation of EWC: approximations

We can now set $\frac{\delta \mathcal{L}(\theta)}{\delta \theta}|_{\theta_A^*} = 0$ (slope at the peak) and plug back in:

$$\log p(\theta|D_A) = \log p(\theta_A^*|D_A) + \frac{1}{2}(\theta - \theta_A^*)^T(\frac{\delta^2 \mathcal{L}(p(\theta|D_A))}{\delta^2 \theta}|_{\theta_A^*})(\theta - \theta_A^*)$$

$$= \frac{1}{2}(\theta - \theta_A^*)^T(\frac{\delta^2 \mathcal{L}(p(\theta|D_A))}{\delta^2 \theta}|_{\theta_A^*})(\theta - \theta_A^*) + \Delta$$

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

Derivation of EWC: approximations

Skipping some steps for brevity, we can rewrite the second derivate to express the standard form of a normal distribution:

$$p(\theta \mid D_A) \sim \mathcal{N}(\theta_A^*, (\frac{\delta^2 \mathcal{L}(p(\theta \mid D_A))}{\delta^2 \theta} \mid_{\theta_A^*})^{-1})$$

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

Derivation of EWC: approximations

Skipping some steps for brevity, we can rewrite the second derivate to express the standard form of a normal distribution:

$$p(\theta \mid D_A) \sim \mathcal{N}(\theta_A^*, (\frac{\delta^2 \mathcal{L}(p(\theta \mid D_A))}{\delta^2 \theta} \mid_{\theta_A^*})^{-1})$$

Using our Laplace approximation of the posterior pdf (mean θ_A^*, variance $[\mathbb{I}_A]^{-1}$), we can return back to finally write out EWC

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

Derivation of EWC: approximations

$$\log p(\theta|D) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B)$$

$$= \log p(D_B|\theta) + \frac{1}{2}(\theta - \theta_A^*)^T(\frac{\delta^2 \mathscr{L}(p(\theta|D_A))}{\delta^2 \theta}|_{\theta_A^*})(\theta - \theta_A^*) + \texttt{const}$$

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

Derivation of EWC: approximations

$$\log p(\theta|D) = \log p(D_B|\theta) + \log p(\theta|D_A) - \log p(D_B)$$

$$= \log p(D_B|\theta) + \frac{1}{2}(\theta - \theta_A^*)^T(\frac{\delta^2 \mathscr{L}(p(\theta|D_A))}{\delta^2 \theta}|_{\theta_A^*})(\theta - \theta_A^*) + \texttt{const}$$

In “simple form”: $\mathscr{L}(\theta) = \mathscr{L}_B - \frac{1}{2}(\theta - \theta_A^*)^T \mathbb{I}_A (\theta - \theta_A^*) + \texttt{const}$

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

In a nutshell: Elastic Weight Consolidation

In "simple form": $\mathscr{L}(\theta) = \mathscr{L}_B(\theta) - \frac{1}{2}(\theta - \theta^*_A)^T \mathbb{I}_A (\theta - \theta^*_A) + \texttt{const}$

In practice, we use the Fisher information (an approximation to the inverse Hessian/second order derivative), ignore constants, and introduce a hyper parameter λ to weigh the strength of the regularizer for each parameter θ_i

$$\mathscr{L}(\theta) = \mathscr{L}_B(\theta) - \frac{\lambda}{2} F_i (\theta - \theta^*_A)^2$$

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

In a nutshell: Elastic Weight Consolidation

In "simple form": $\mathscr{L}(\theta) = \mathscr{L}_B(\theta) - \frac{1}{2}(\theta - \theta^*_A)^T \mathbb{I}_A (\theta - \theta^*_A) + \texttt{const}$

In practice, we use the Fisher information (an approximation to the inverse Hessian/second order derivative), ignore constants, and introduce a hyper parameter λ to weigh the strength of the regularizer for each parameter θ_i

$$\mathscr{L}(\theta) = \mathscr{L}_B(\theta) - \frac{\lambda}{2} F_i (\theta - \theta^*_A)^2$$

You likely saw this coming: EWC is our next tutorial :)

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

Universität Bremen

How EWC works: parameter importance intuition

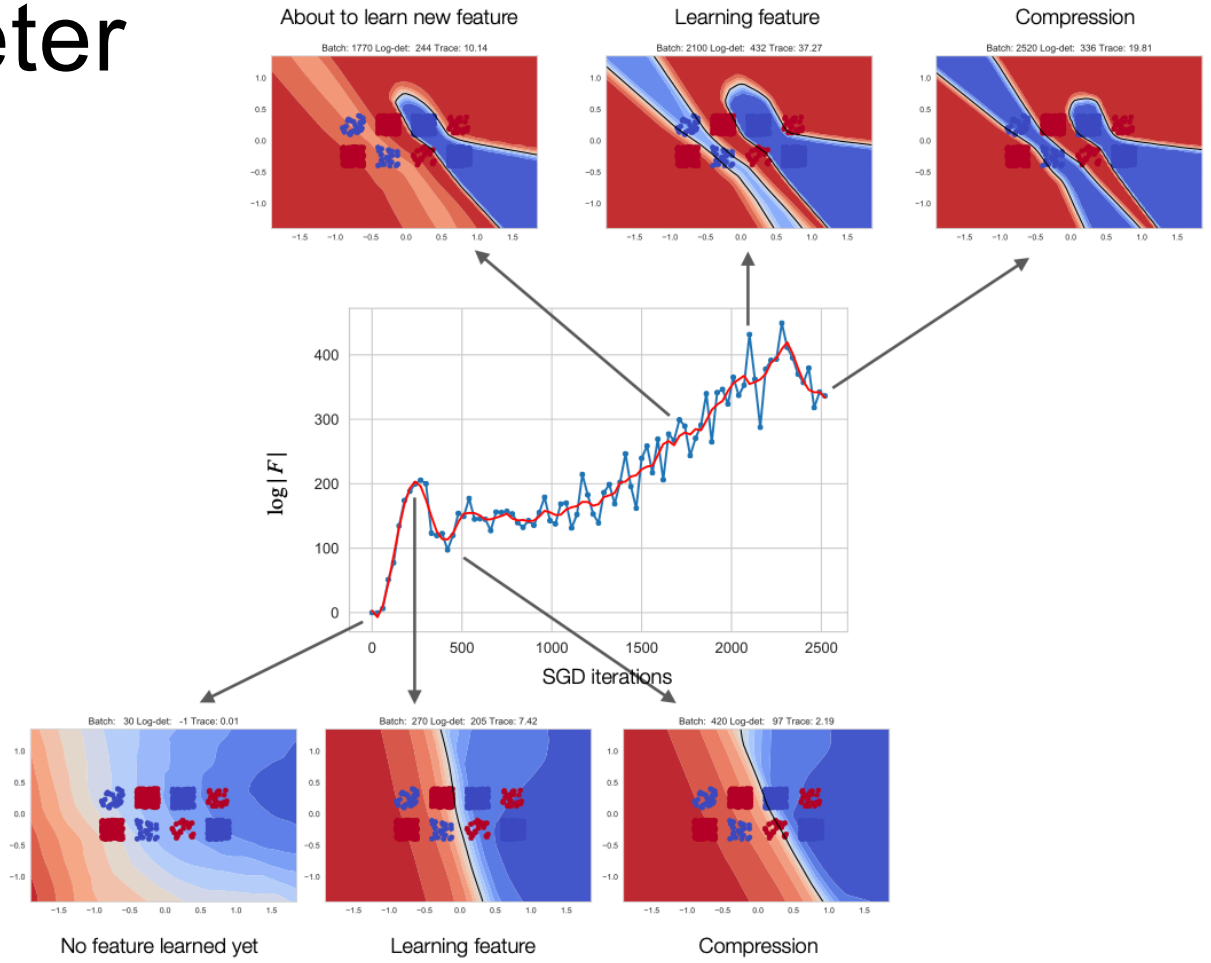
The Fisher information quantifies parameter “importance”, how much parameters contribute to decisions

Universität Bremen

How EWC works: parameter importance intuition

The Fisher information quantifies parameter “importance”, how much parameters contribute to decisions

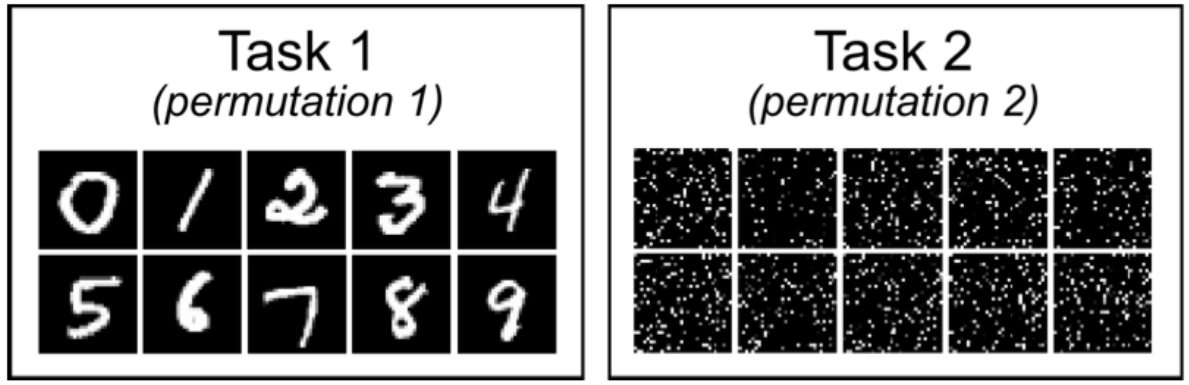
As a network learns an increasingly complex function, new features intuitively correspond to increases in the (log-determinant) of the Fisher



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

EWC in continual learning tasks

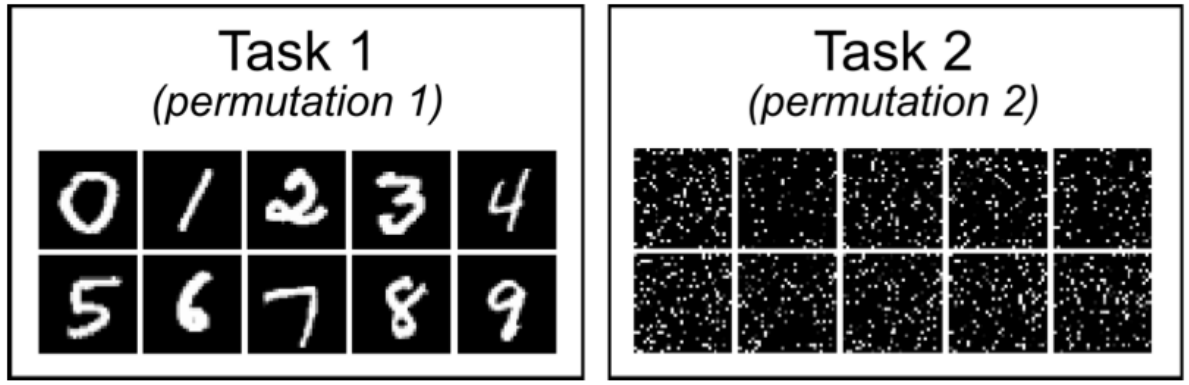
Evaluation on a continual learning task based on permuting pixels in MNIST images of handwritten digits

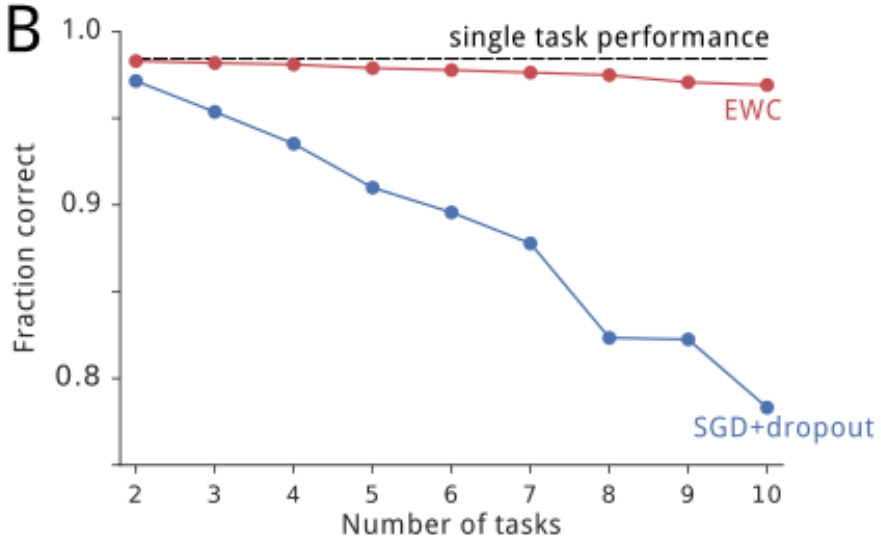


Kirkpatrick et al, “Overcoming catastrophic forgetting in neural networks”, PNAS 114(13), 2017
Permuted MNIST: Goodfellow et al, “An empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks”, ICLR 2014,
Permuted MNIST Image from van de Ven et al, “Three scenarios for continual learning”, 2019

EWC in continual learning tasks

Evaluation on a continual learning task based on permuting pixels in MNIST images of handwritten digits

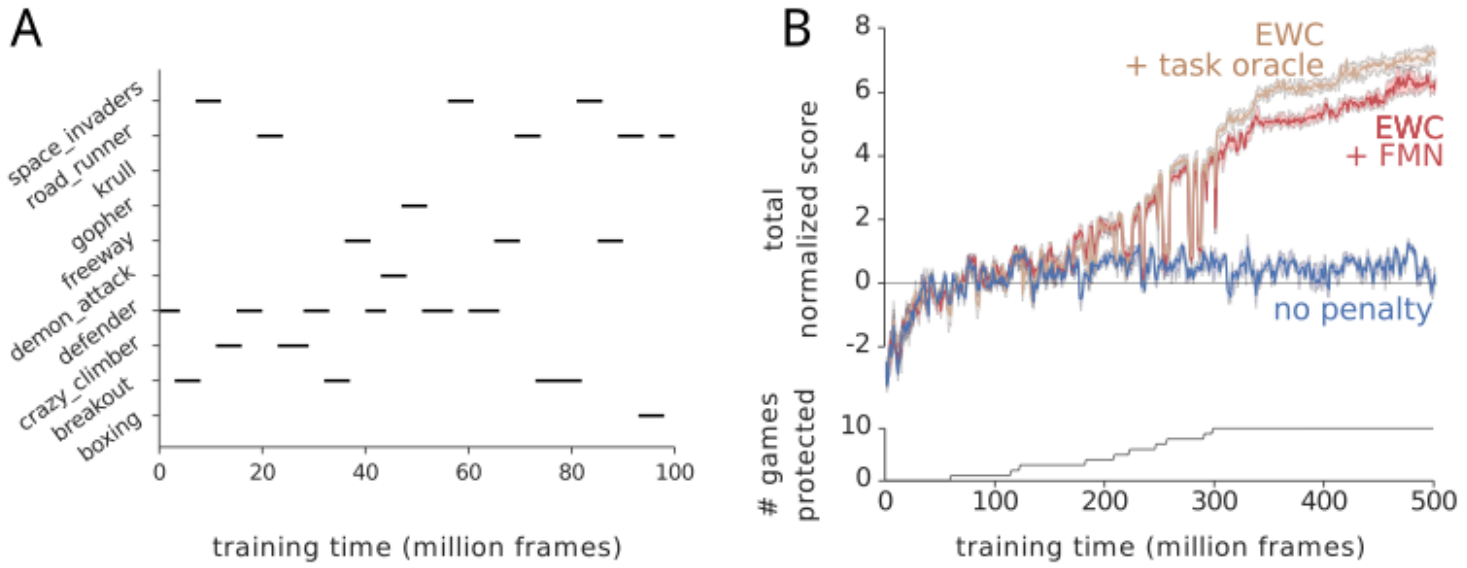




Kirkpatrick et al, “Overcoming catastrophic forgetting in neural networks”, PNAS 114(13), 2017
Permuted MNIST: Goodfellow et al, “An empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks”, ICLR 2014,
Permuted MNIST Image from van de Ven et al, “Three scenarios for continual learning”, 2019

EWC in continual learning tasks

Evaluation on a sequence of different Atari games (black bars indicate the sequential training periods of the played game)



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

Universität Bremen

OWL ML

Question time

Do you foresee any challenges with EWC?

Some EWC caveats and limitations

- Hessians and the Fisher information for neural networks are challenging. The matrix dimensionality is params times params

Some EWC caveats and limitations

- Hessians and the Fisher information for neural networks are challenging. The matrix dimensionality is params times params

- Many ways to compute the Fisher. In continual ML, the empirical Fisher is mostly used: computing a squared gradient only for each sample's ground truth class (avoiding an expectation over all data). See below references for more details

G. van de Ven, "On the Computation of the Fisher Information in Continual Learning", ICLR 2025 Blogpost, Kunstner et al, "Limitations of the Empirical Fisher Approximation for Natural Gradient Descent", NeurIPS 2019

Some EWC caveats and limitations

- Hessians and the Fisher information for neural networks are challenging. The matrix dimensionality is params times params
- Many ways to compute the Fisher. In continual ML, the empirical Fisher is mostly used: computing a squared gradient only for each sample's ground truth class (avoiding an expectation over all data). See below references for more details
- Selecting λ, the regularization strength, is challenging in practice without "looking into the future"

G. van de Ven, "On the Computation of the Fisher Information in Continual Learning", ICLR 2025 Blogpost, Kunstner et al, "Limitations of the Empirical Fisher Approximation for Natural Gradient Descent", NeurIPS 2019

Some EWC caveats and limitations

Illustration of empirical impact of some of these choices for MNIST

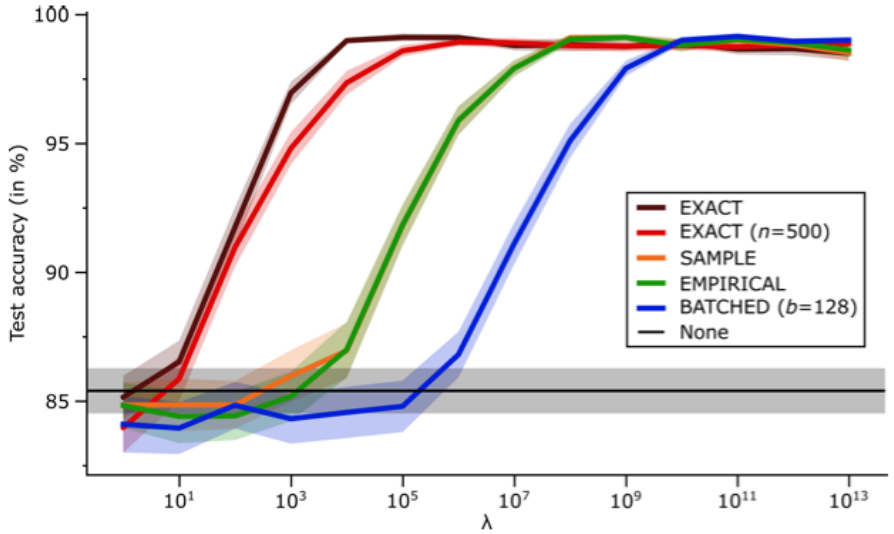


Figure 1: **Split MNIST.** Performance of EWC with different ways of computing the Fisher Information for a wide range of hyperparameter values.

G. van de Ven, “On the Computation of the Fisher Information in Continual Learning”, ICLR 2025 Blogpost

Some EWC caveats and limitations

Illustration of empirical impact of some of these choices for MNIST

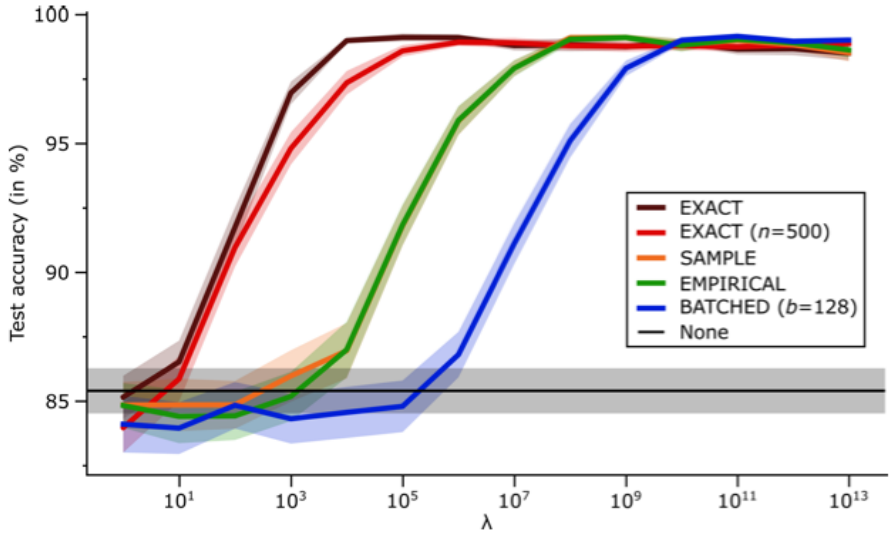


Figure 1: **Split MNIST.** Performance of EWC with different ways of computing the Fisher Information for a wide range of hyperparameter values.

	Accuracy	Train time
EXACT	99.12 (± 0.16)	121 (± 1)
EXACT (n=500)	98.93 (± 0.15)	58 (± 0)
SAMPLE	99.12 (± 0.12)	101 (± 1)
EMPIRICAL	99.12 (± 0.12)	100 (± 1)
BATCHED (b=128)	99.11 (± 0.16)	58 (± 1)
None	*85.41 (± 0.88)*	*53 (± 0)*

Table 1: **Split MNIST.** The average final test accuracy (in %) for the best performing hyperparameter value of each variant, and the total training time (in seconds) on an NVIDIA RTX 2000 Ada Generation GPU.

G. van de Ven, “On the Computation of the Fisher Information in Continual Learning”, ICLR 2025 Blogpost

Recursive Laplace approx. for 3+ tasks in EWC

In the original EWC (as shown before also on the examples)

- Compute a separate Fisher per task
- Use “an anchoring point” per task given current value of parameters
- Impose a separate penalty with a separate lambda per task as well

F. Huszár, “Note on the Quadratic Penalties in Elastic Weight Consolidation”, PNAS 115(11), 2018

Recursive Laplace approx. for 3+ tasks in EWC

In the original EWC (as shown before also on the examples)

- Compute a separate Fisher per task
- Use “an anchoring point” per task given current value of parameters
- Impose a separate penalty with a separate lambda per task as well

Intuitively, one penalty based on the last task should suffice, but we would have to recursively apply the (diagonal) Laplace approximation

$$\log p(\theta \mid D_A, D_B, D_C) = \log p(D_C \mid \theta) + \log p(\theta \mid D_A, D_B) + const$$

F. Huszár, “Note on the Quadratic Penalties in Elastic Weight Consolidation”, PNAS 115(11), 2018

Recursive Laplace approx. for 3+ tasks in EWC

$$\log p(\theta \mid D_A, D_B, D_C) = \log p(D_C \mid \theta) + \log p(\theta \mid D_A, D_B) + const$$

After learning on task A we lost access to D_A, and we only have an approximation to the posterior.

However, we could again recursively apply a second order Taylor approximation on the RHS around θ^*_B.

F. Huszár, “Note on the Quadratic Penalties in Elastic Weight Consolidation”, PNAS 115(11), 2018

Recursive Laplace approx. for 3+ tasks in EWC

Let us not go through this now & just briefly look at an intuitive figure

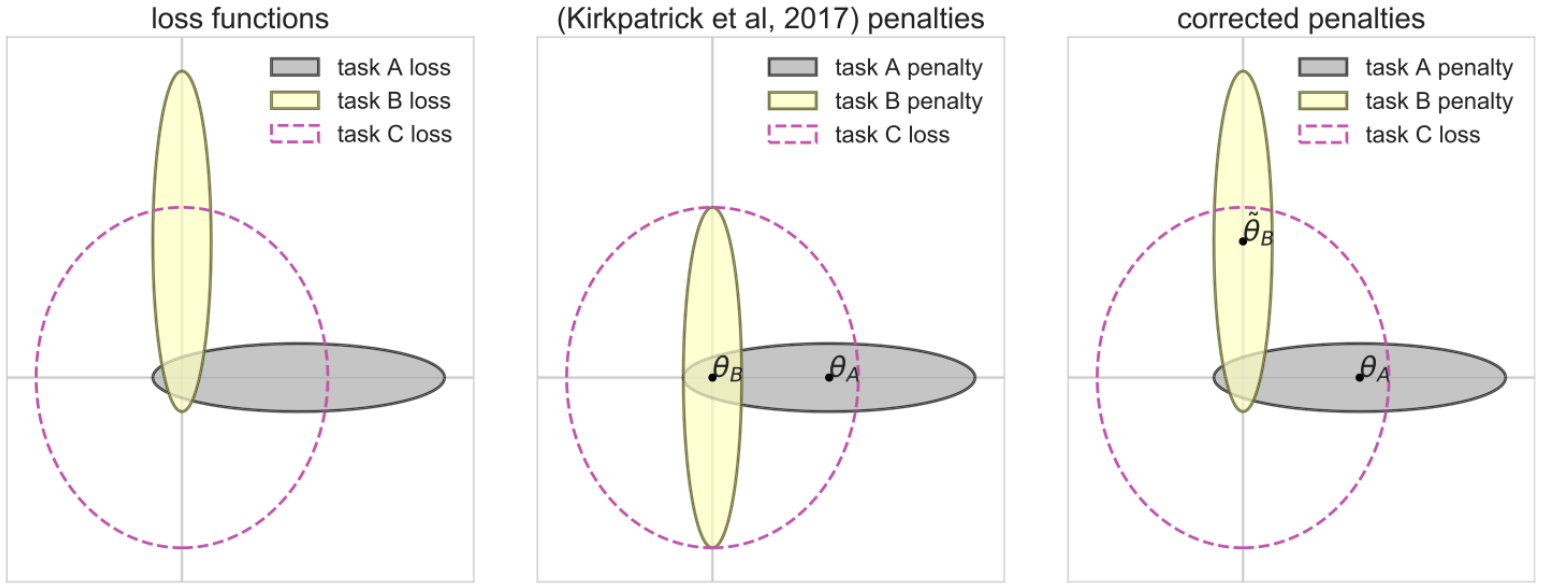


Fig. 1. (*Left*) Elliptical level sets of quadratic loss functions for tasks A, B, and C also used in Table 1. (*Center*) When learning task C via EWC, losses for tasks A and B are replaced by quadratic penalties around θ_A^* and θ_B^*. (*Right*) Losses are approximated perfectly by the correct quadratic penalties around $\tilde{\theta}_A = \theta_A^*$ and $\tilde{\theta}_B$.

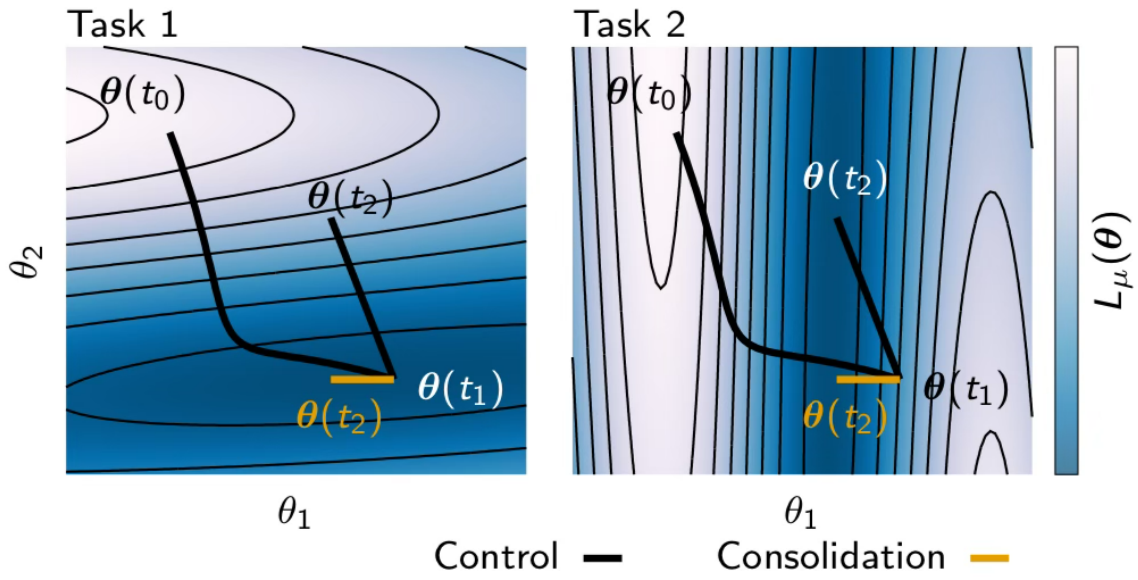
F. Huszár, “Note on the Quadratic Penalties in Elastic Weight Consolidation”, PNAS 115(11), 2018

Of course: many other ways to penalize

Example: Synaptic Intelligence
(concurrent timing to EWC)

Account for the trajectory by approximating importance as summation over what each parameter's change contributes to change in total loss:

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



Zenke et al, "Continual Learning Through Synaptic Intelligence", ICML 2017

Question time

Recall below EWC motivation.

If many configurations of θ result in the same performance, could there be a different approach than regularizing the (important) parameters?

EWC motivation: many configs of θ result in the same performance and over-parametrization makes it likely that there is a solution for a task B, θ^*_B, that is close to a previously found solution for task A, θ^*_A.

Weight-space & function-space regularization

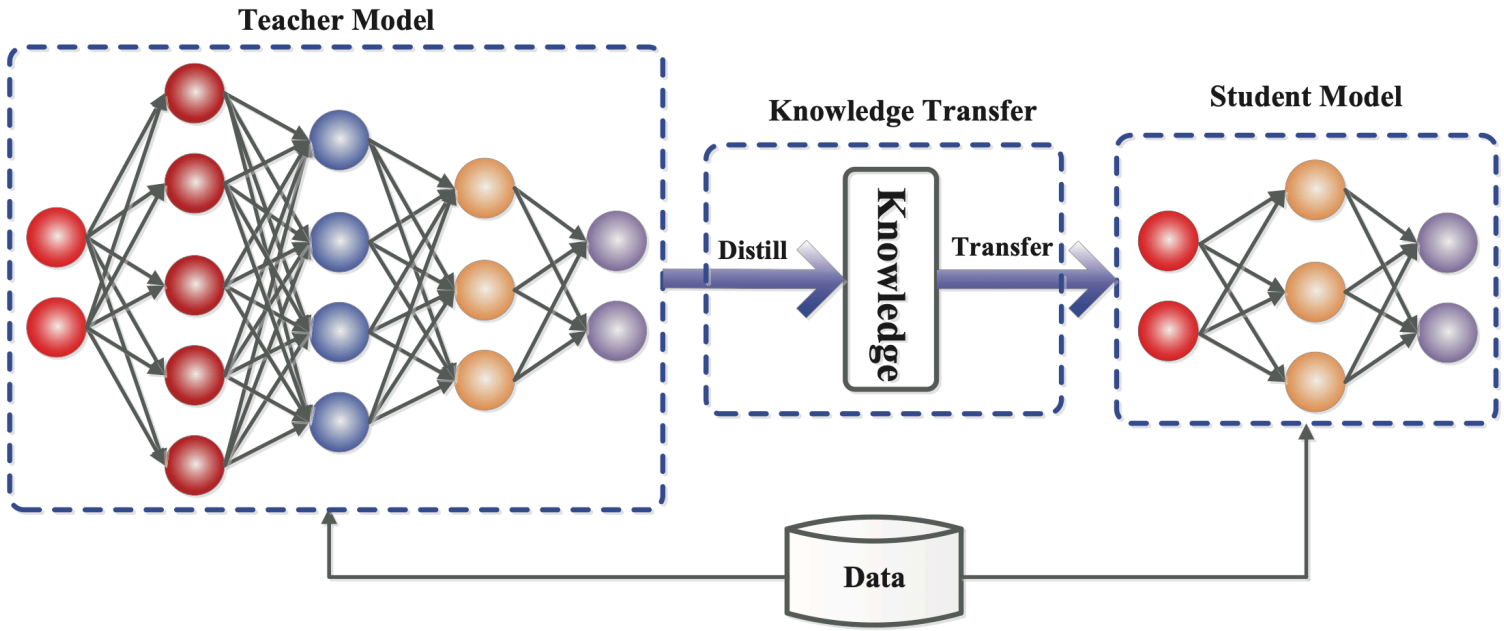
Algorithms like EWC and Synaptic Intelligence regularize the loss function on the basis of the found parameters.

But if many different θ can lead to the same quality solution, why constrain the parameters instead of the input-output mappings?

This approach has become equally popular in continual learning, and could be viewed as “functional” regularization.

Function regularization basics: distillation

A popular algorithm is called “knowledge distillation”, which attempts to have a student learn to mimic a teacher’s predictions given input



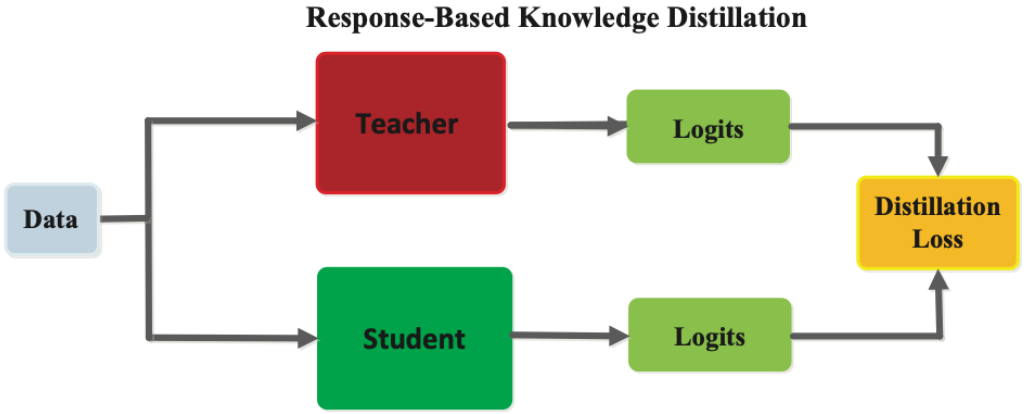
Hinton et al, “Distilling the Knowledge in A Neural Network”, NeurIPS Deep Learning Workshop, 2014. Images from Gou et al, “Knowledge Distillation: A survey”, IJCV 129, 2021

A special distillation case: class logit matching

Match a teacher's soft-targets

$$p(z^T, \tau) = \frac{\exp(z_c/\tau)}{\sum_j \exp(z_j/\tau)} \text{ with}$$

logit z_c, class c & temperature τ



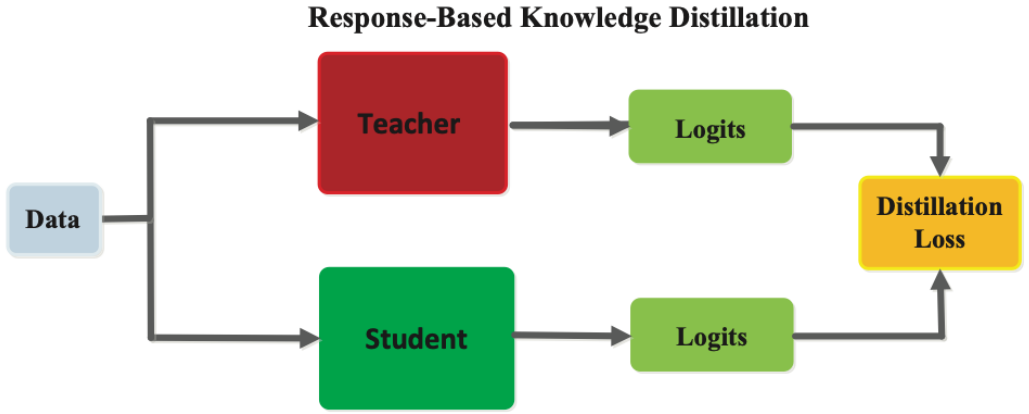
Hinton et al, "Distilling the Knowledge in A Neural Network", NeurIPS Deep Learning Workshop, 2014. Image from Gou et al, "Knowledge Distillation: A survey", IJCV 129, 2021

A special distillation case: class logit matching

Match a teacher's soft-targets

$$p(z^T, \tau) = \frac{\exp(z_c/\tau)}{\sum_j \exp(z_j/\tau)} \text{ with}$$

logit z_c, class c & temperature τ



In essence, we ensure that the distance between "z" of two models is minimized & generally, the divergence between their distributions p & q:

$$\frac{1}{\tau}(q_i - p_i) \approx \frac{1}{\tau}\left(\frac{\exp(z_c^T/\tau)}{\sum_j \exp(z_j^T/\tau)} - \frac{\exp(z_c^S/\tau)}{\sum_j \exp(z_j^S/\tau)}\right)$$

Hinton et al, "Distilling the Knowledge in A Neural Network", NeurIPS Deep Learning Workshop, 2014. Image from Gou et al, "Knowledge Distillation: A survey", IJCV 129, 2021

A special distillation case: class logit matching

We can control the weights of soft/real targets through a λ

$$\mathcal{L}(\boldsymbol{x}_i, y_i) = \lambda_1 \underbrace{\mathrm{CE}\left(y_i, p\left(\boldsymbol{z}_i^{\mathrm{S}}, 1\right)\right)}_{\text{match true labels}} + \lambda_2 \underbrace{\mathrm{CE}\left(p\left(\boldsymbol{z}_i^{\mathrm{T}}, \tau\right), p\left(\boldsymbol{z}_i^{\mathrm{S}}, \tau\right)\right)}_{\text{match teacher soft labels}}$$

Hinton et al, “Distilling the Knowledge in A Neural Network”, NeurIPS Deep Learning Workshop, 2014. Image from Gou et al, “Knowledge Distillation: A survey”, IJCV 129, 2021

A special distillation case: class logit matching

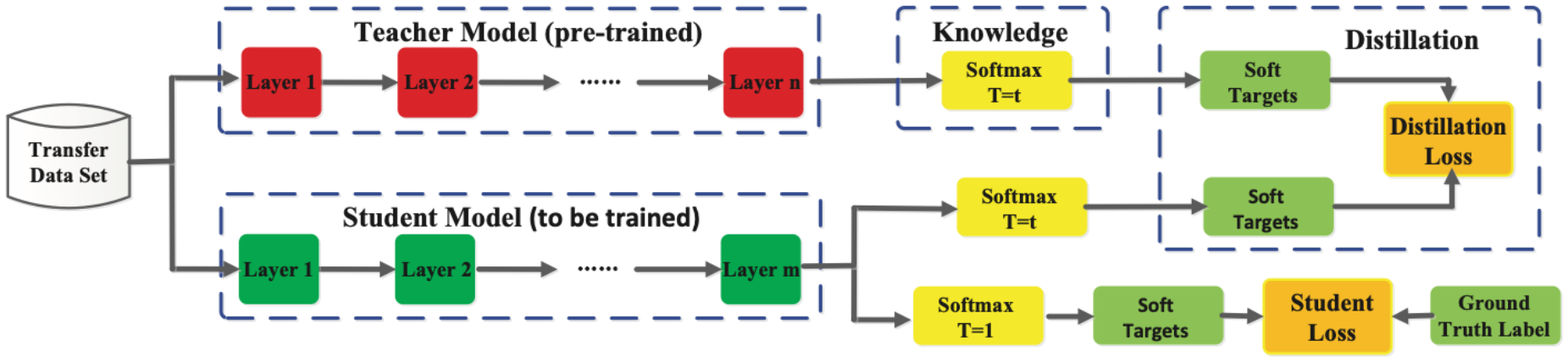
We can control the weights of soft/real targets through a λ

Higher τ produce softer probability distributions

$$\mathcal{L}(\boldsymbol{x}_i, y_i) = \lambda_1 \, \mathrm{CE}\left(y_i, p\left(\boldsymbol{z}_i^{\mathrm{S}}, 1\right)\right) + \lambda_2 \, \mathrm{CE}\left(p\left(\boldsymbol{z}_i^{\mathrm{T}}, \tau\right), p\left(\boldsymbol{z}_i^{\mathrm{S}}, \tau\right)\right)$$

match true labels

match teacher soft labels

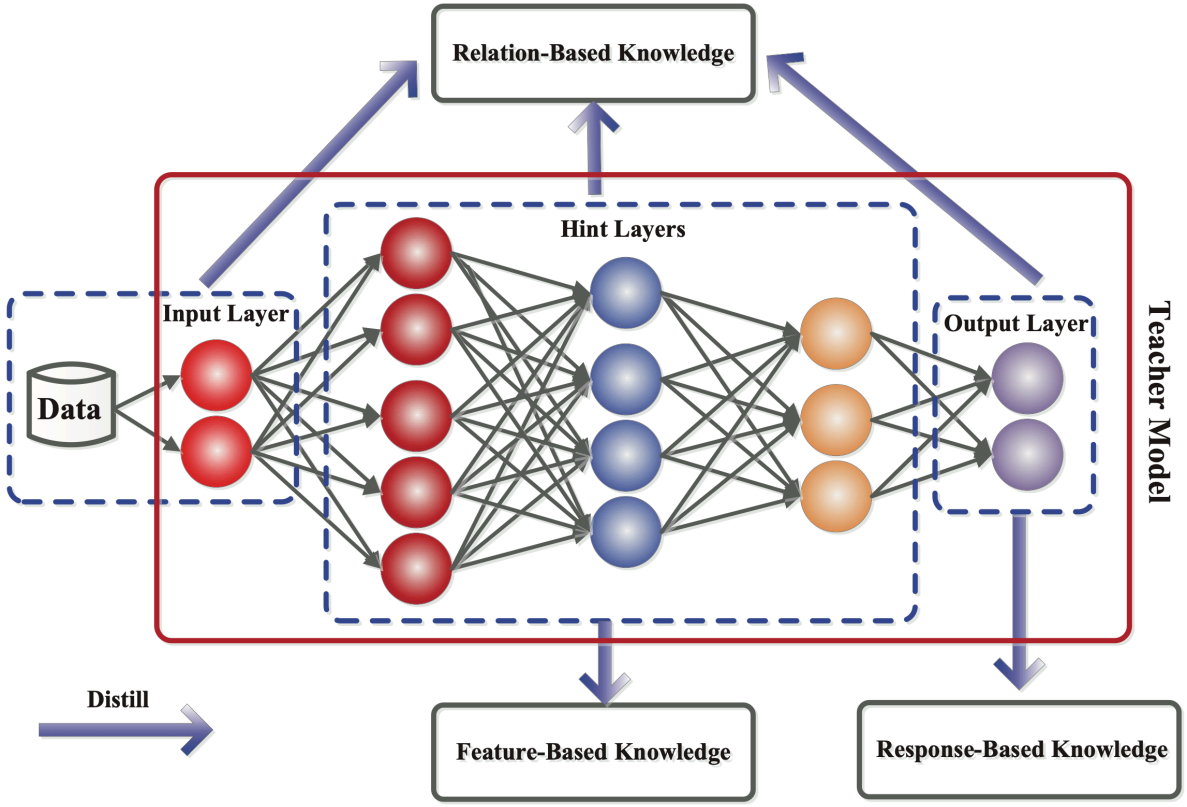


Hinton et al, “Distilling the Knowledge in A Neural Network”, NeurIPS Deep Learning Workshop, 2014. Image from Gou et al, “Knowledge Distillation: A survey”, IJCV 129, 2021

Knowledge distillation beyond the special case

"Knowledge" and mappings are diverse, which we can leverage in deciding what to distill:

- Response distillation
- Feature distillation
- Relational distillation
- Self-distillation
- Data-free distillation
- ….

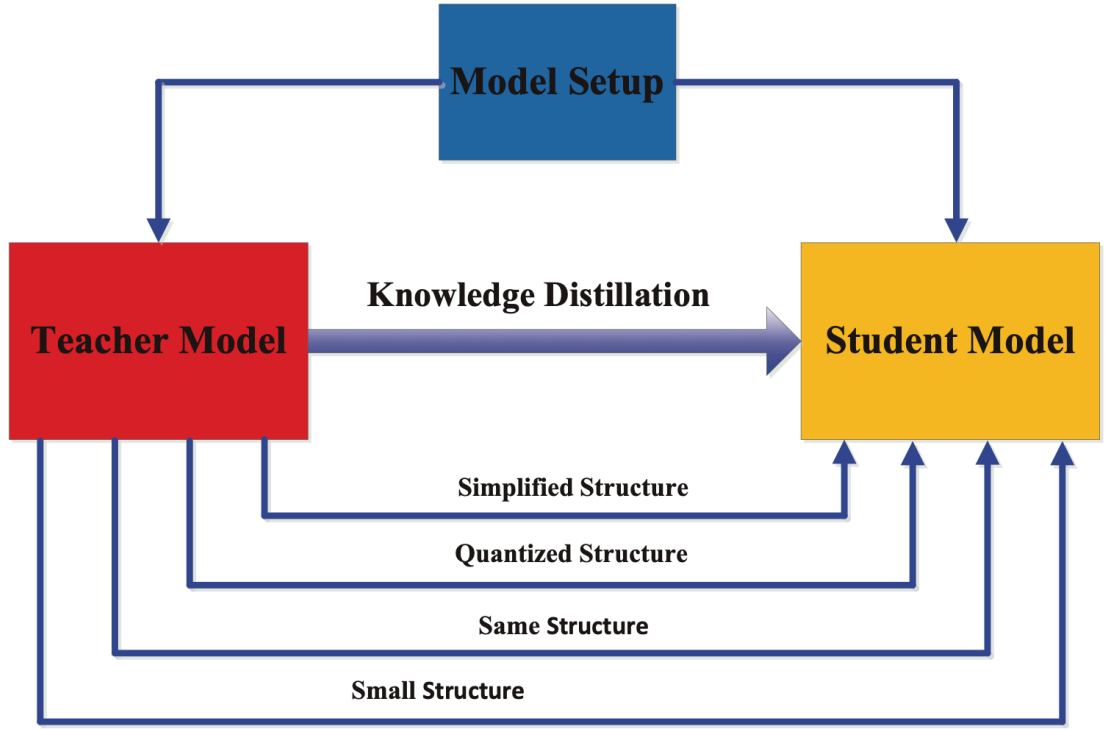


Gou et al, "Knowledge Distillation: A survey", IJCV 129, 2021

Original goals of knowledge distillation

At a very similar performance/ accuracy, student models could:

- Use a different type of model
- Employ quantization for computational efficiency
- Improve performance (think back to curriculum learning)
- Use less parameters (learning can be easier with guidance)
- ….



Gou et al, “Knowledge Distillation: A survey”, IJCV 129, 2021

Universität Bremen

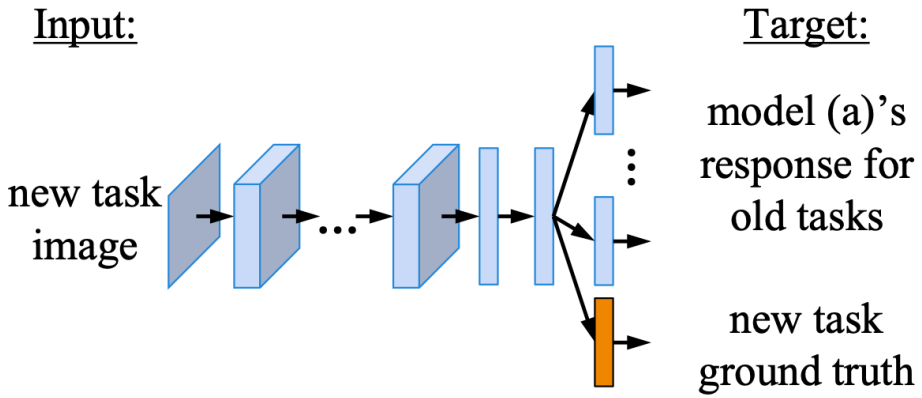
OWL ML

Question time

How can knowledge distillation help us with continual learning & avoiding forgetting?

“Learning without forgetting”: lifelong KD

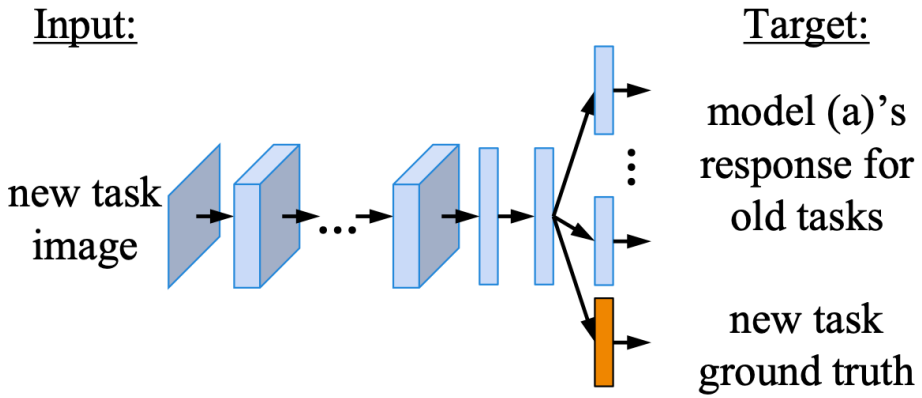
Key idea: let the teacher and students be models at time t and $t+1$ when learning two tasks & add a new output layer (a “task head”)



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

“Learning without forgetting”: lifelong KD

Key idea: let the teacher and students be models at time t and $t+1$ when learning two tasks & add a new output layer (a “task head”)



LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

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

“Learning without forgetting”: lifelong KD

If we don’t have access to old task’s data anymore because we are in a sequential set-up, then what is “old-task output” & why should KD work?

LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

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

“Learning without forgetting”: lifelong KD

If we don’t have access to old task’s data anymore because we are in a sequential set-up, then what is “old-task output” & why should KD work?

Key hypothesis in LwF:
Task A outputs are computed with task B data! Even if the output is wrong, the *function* of the network should be preserved, while weights of a shared encoder can adapt, provided task B data has some similarity (e.g. natural images)

LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

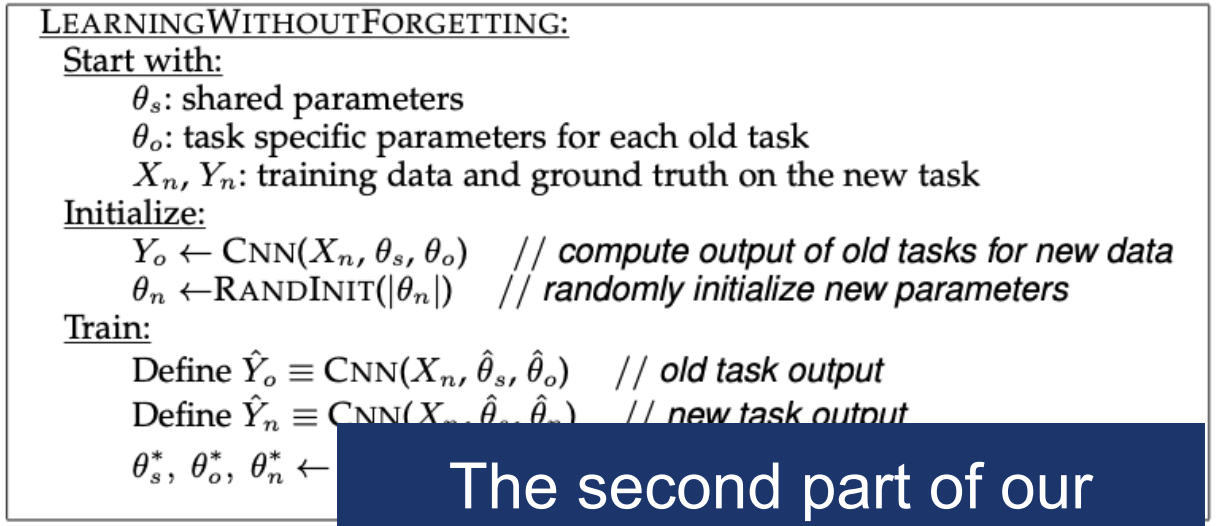
$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

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

“Learning without forgetting”: lifelong KD

If we don’t have access to old task’s data anymore because we are in a sequential set-up, then what is “old-task output” & why should KD work?

Key hypothesis in LwF:
Task A outputs are computed with task B data! Even if the output is wrong, the *function* of the network should be preserved, while weights of a shared encoder can adapt, provided task B data has some similarity (e.g. natural images)



LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow$

The second part of our “regularization” tutorial

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

Universität Bremen

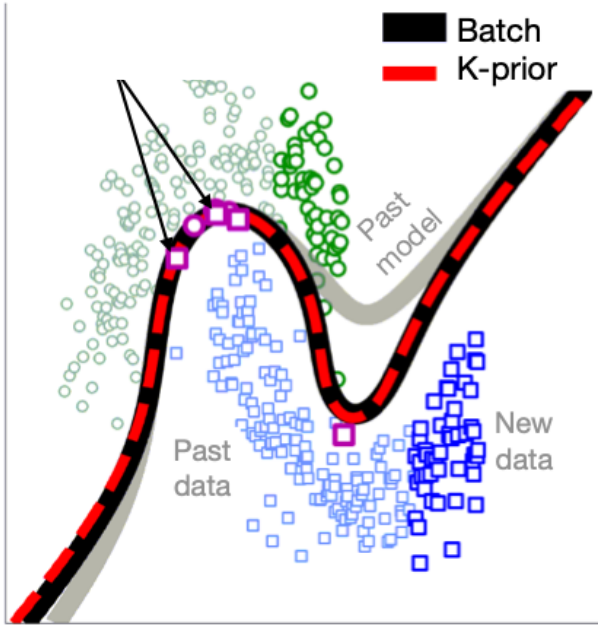
OWL ML

Question time

What are limitations of this lifelong KD approach?

Lifelong knowledge distillation caveat 1

Assessing similarity is hard. When task B is “too dissimilar” from A, we may preserve the original function of task A “head”, but in an unimportant part of space

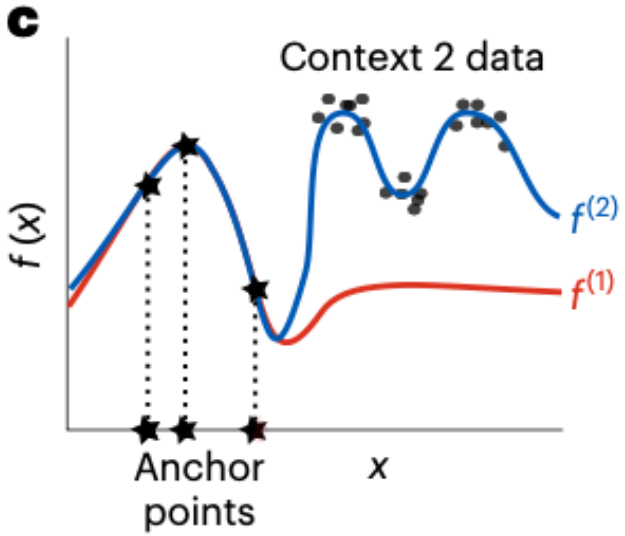


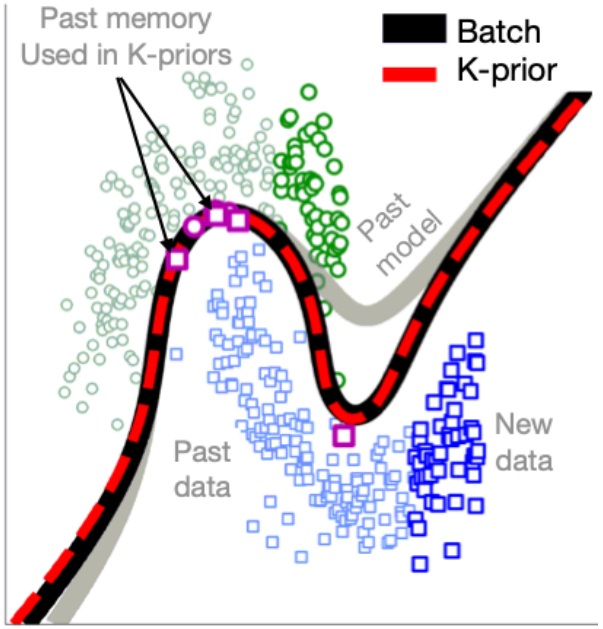
Khan & Swaroop, “Knowledge-Adaptation Priors”, NeurIPS 2021

Lifelong knowledge distillation caveat 1

Assessing similarity is hard. When task B is “too dissimilar” from A, we may preserve the original function of task A “head”, but in an unimportant part of space

A popular solution is to include “memory” “anchors”





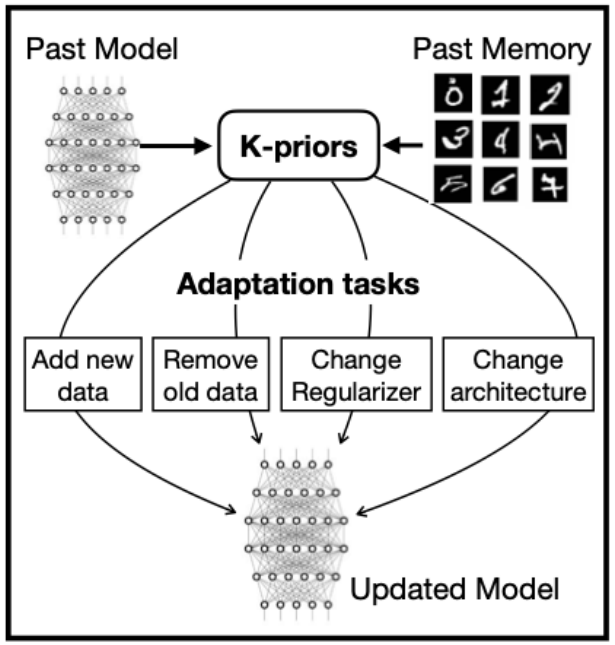
Khan & Swaroop, “Knowledge-Adaptation Priors”, NeurIPS 2021
van de Ven et al, “Three types of incremental learning”, Nature Machine Intelligence 4, 2022

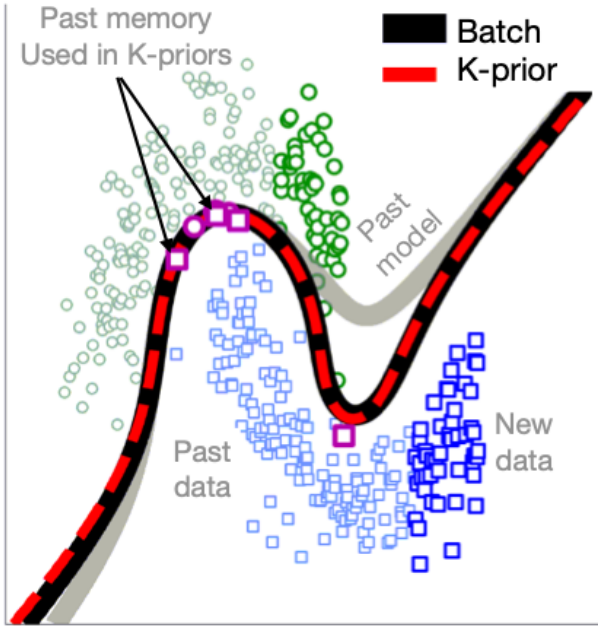
Lifelong knowledge distillation caveat 1

Assessing similarity is hard. When task B is “too dissimilar” from A, we may preserve the original function of task A “head”, but in an unimportant part of space

A popular solution is to include “memory” “anchors”

We’ll continue with memory & how to combine functional + weight regularization next, but before: another caveat

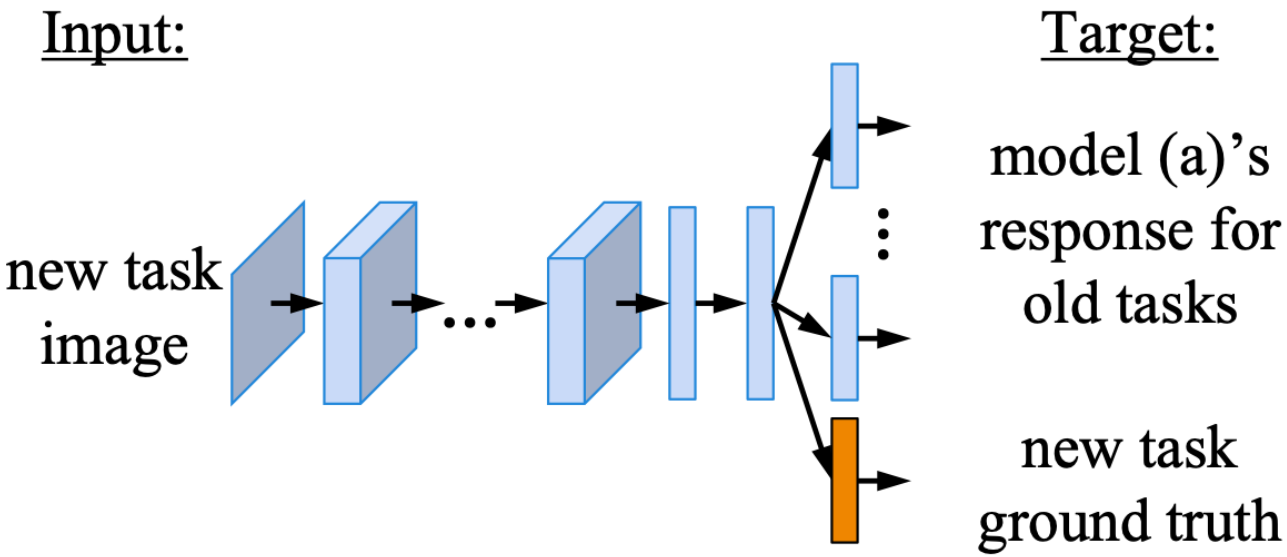




Khan & Swaroop, “Knowledge-Adaptation Priors”, NeurIPS 2021

Lifelong knowledge distillation caveat 2

If we have separate output heads, we need to know during test-time/deployment which output head to use for prediction

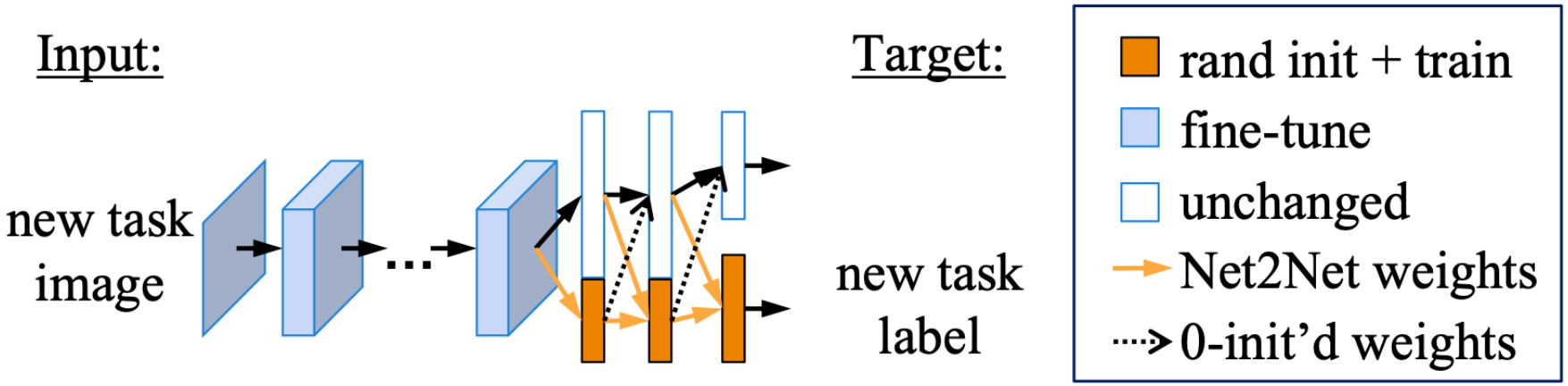


Li & Hoiem, "Learning without Forgetting", ECCV 2016

Lifelong knowledge distillation caveat 2

If we have separate output heads, we need to know during test-time/deployment which output head to use for prediction

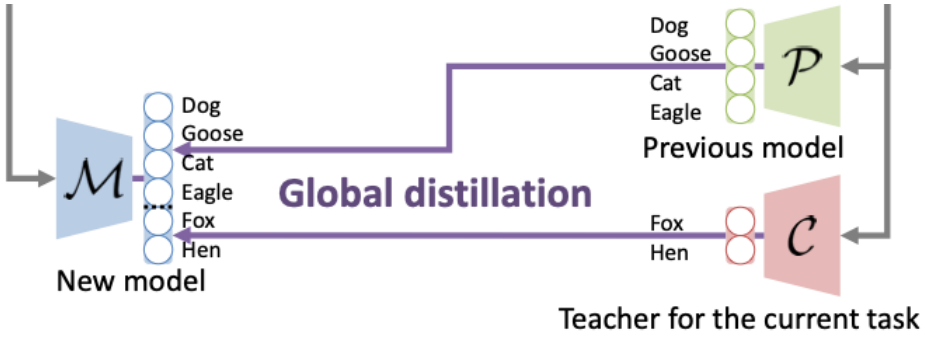
On the contrary, if we have a single output head that “grows”, we have to worry about “cross-talk” & e.g. Softmax layer normalization



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

Lifelong knowledge distillation caveat 2

We could proceed similarly to EWC and define “anchoring points”, store models with θ_A^* and θ_B^* and recursively distill them into a new joint model

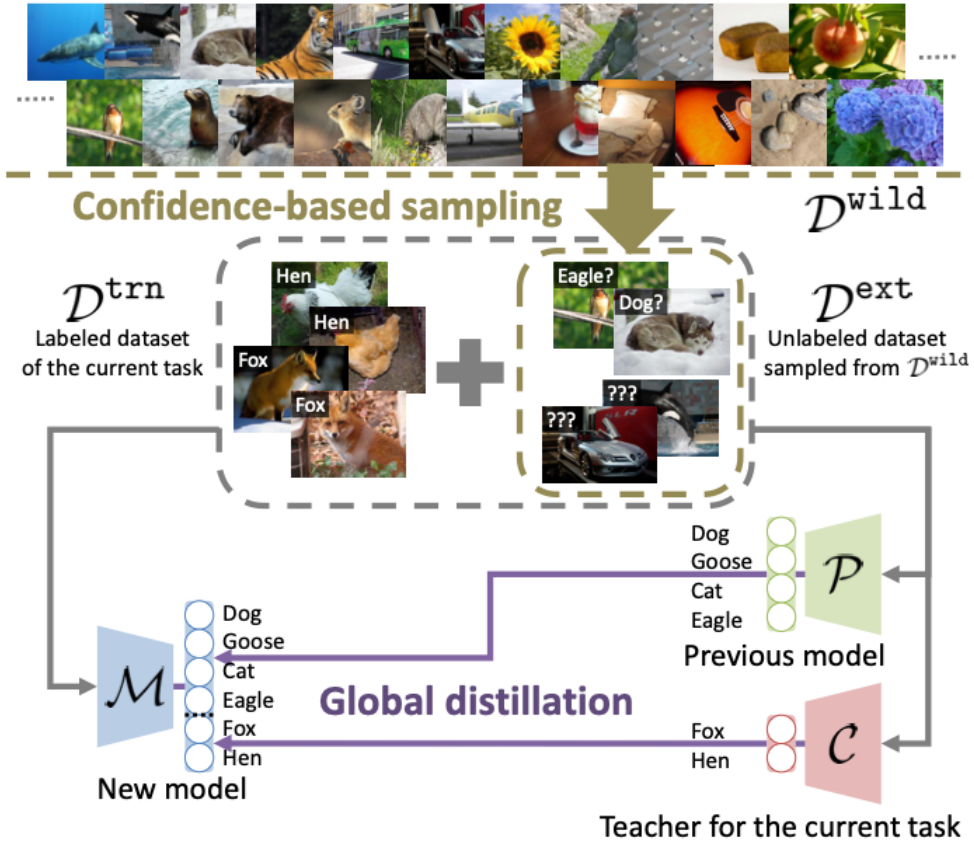


K. Lee et al, “Overcoming Catastrophic Forgetting with Unlabeled Data in the Wild”, ICCV 2019

Lifelong knowledge distillation caveat 2

We could proceed similarly to EWC and define “anchoring points”, store models with θ_A^* and θ_B^* and recursively distill them into a new joint model

We could also use external unlabelled “in-the-wild” data that we expect to be helpful for distillation



K. Lee et al, “Overcoming Catastrophic Forgetting with Unlabeled Data in the Wild”, ICCV 2019

Universität Bremen

Incremental learning has many distinct set-ups

Such considerations, especially on task-similarity & availability of a “task-id” have sparked numerous set-ups in “continual ML”, similar to and inspired by inductive/transductive transfer learning definitions

van de Ven et al, “Three types of incremental learning”, Nature Machine Intelligence 4, 2022

Incremental learning has many distinct set-ups

Such considerations, especially on task-similarity & availability of a "task-id" have sparked numerous set-ups in "continual ML", similar to and inspired by inductive/transductive transfer learning definitions

Table 1 | Overview of the three continual learning scenarios

Scenario	Intuitive description	Mapping to learn
Task-incremental learning	Sequentially learn to solve a number of distinct tasks	$f : x \times c \rightarrow y$
Domain-incremental learning	Learn to solve the same problem in different contexts	$f : x \rightarrow y$
Class-incremental learning	Discriminate between incrementally observed classes	$f : x \rightarrow c \times y$

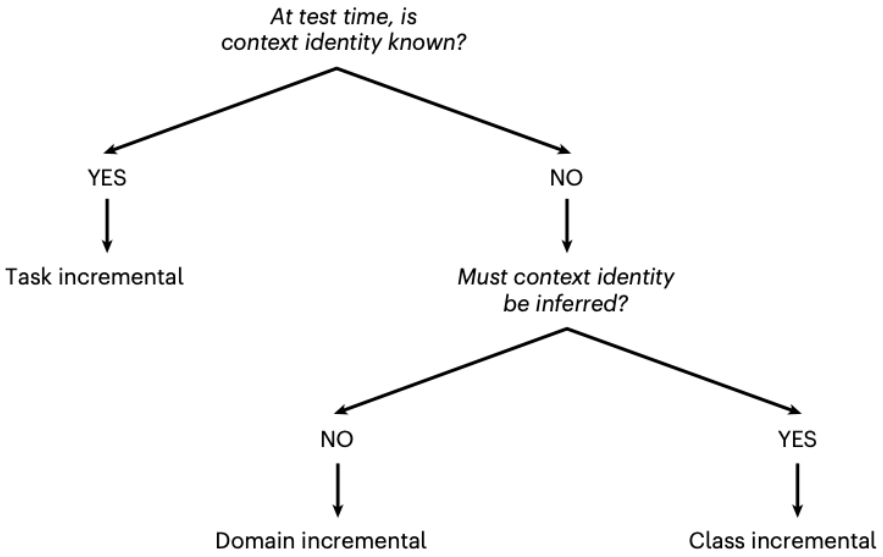
van de Ven et al, "Three types of incremental learning", Nature Machine Intelligence 4, 2022

Incremental learning has many distinct set-ups

Such considerations, especially on task-similarity & availability of a "task-id" have sparked numerous set-ups in "continual ML", similar to and inspired by inductive/transductive transfer learning definitions

Table 1 | Overview of the three continual learning scenarios

Scenario	Intuitive description	Mapping to learn
Task-incremental learning	Sequentially learn to solve a number of distinct tasks	$f : x \times c \rightarrow y$
Domain-incremental learning	Learn to solve the same problem in different contexts	$f : x \rightarrow y$
Class-incremental learning	Discriminate between incrementally observed classes	$f : x \rightarrow c \times y$

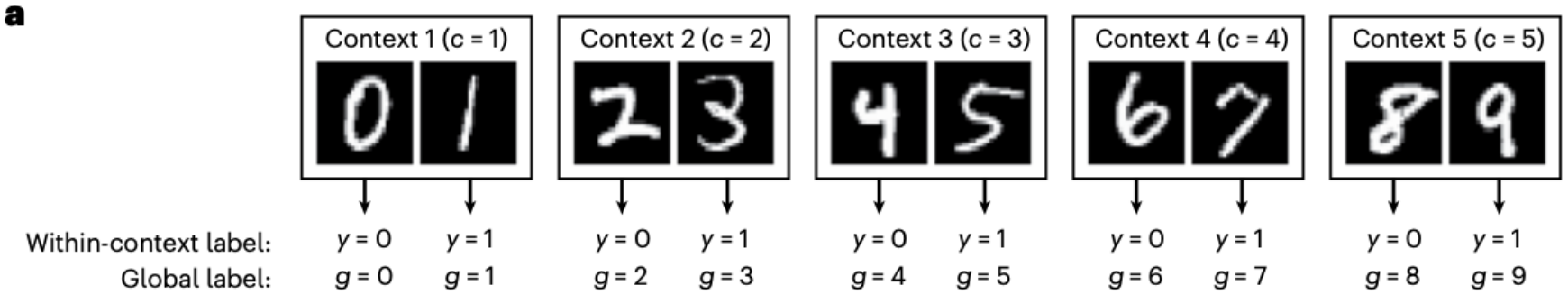


van de Ven et al, "Three types of incremental learning", Nature Machine Intelligence 4, 2022

Set-ups exemplified with MNIST digits

Unfortunately, this introduces a lot of complexity that we will keep discussing in the course. Here's an MNIST example

a



b

	Input (at test time)	Expected output	Intuitive description
Task-incremental learning	Image + context label	Within-context label[a]	Choice between two digits of same context (e.g. 0 or 1)
Domain-incremental learning	Image	Within-context label	Is the digit odd or even?
Class-incremental learning	Image	Global label	Choice between all ten digits

van de Ven et al, "Three types of incremental learning", Nature Machine Intelligence 4, 2022

Set-ups exemplified with MNIST digits

Unfortunately, this introduces a lot of complexity that we will keep discussing in the course. Here's a (less intuitive?) MNIST example

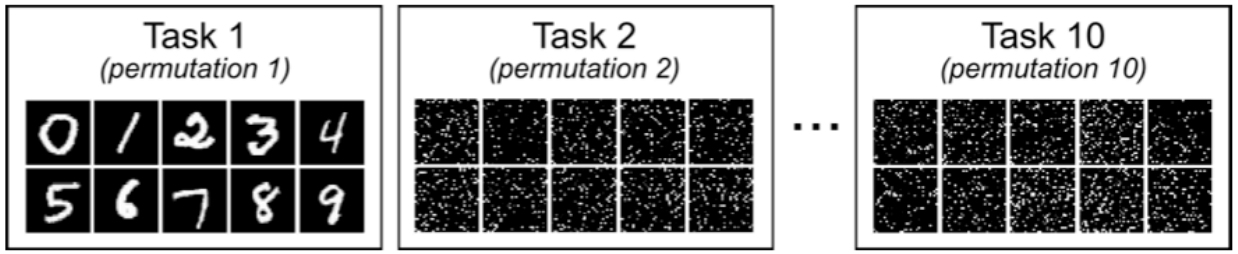


Figure 2: Schematic of permuted MNIST task protocol.

Table 3: Permuted MNIST according to each scenario.

Task-IL	Given permutation *X*, which digit?
Domain-IL	With permutation unknown, which digit?
Class-IL	Which digit *and* which permutation?

van de Ven et al, "Three types of incremental learning", Nature Machine Intelligence 4, 2022

Set-ups exemplified with MNIST digits

Many of the discussed techniques are technically agnostic, but will be heavily influenced by set-up in terms of expected performance!

Strategy	Method	Budget	GM	Task-IL	Domain-IL	Class-IL
Baselines	None – lower target			84.32 (±0.99)	60.13 (±1.66)	19.89 (±0.02)
	Joint – upper target			99.67 (±0.03)	98.59 (±0.05)	98.17 (±0.04)
Context-specific components	Separate Networks	-	-	99.57 (±0.03)	-	-
	XdG	-	-	99.10 (±0.10)	-	-
Parameter regularization	EWC	-	-	99.06 (±0.15)	63.03 (±1.58)	20.64 (±0.52)
	SI	-	-	99.20 (±0.11)	66.94 (±1.13)	21.20 (±0.57)
Functional regularization	LwF	-	-	99.60 (±0.03)	71.18 (±1.42)	21.89 (±0.32)
	FROMP	100	-	99.12 (±0.13)	84.86 (±1.02)	77.38 (±0.64)

van de Ven et al, “Three types of incremental learning”, Nature Machine Intelligence 4, 2022

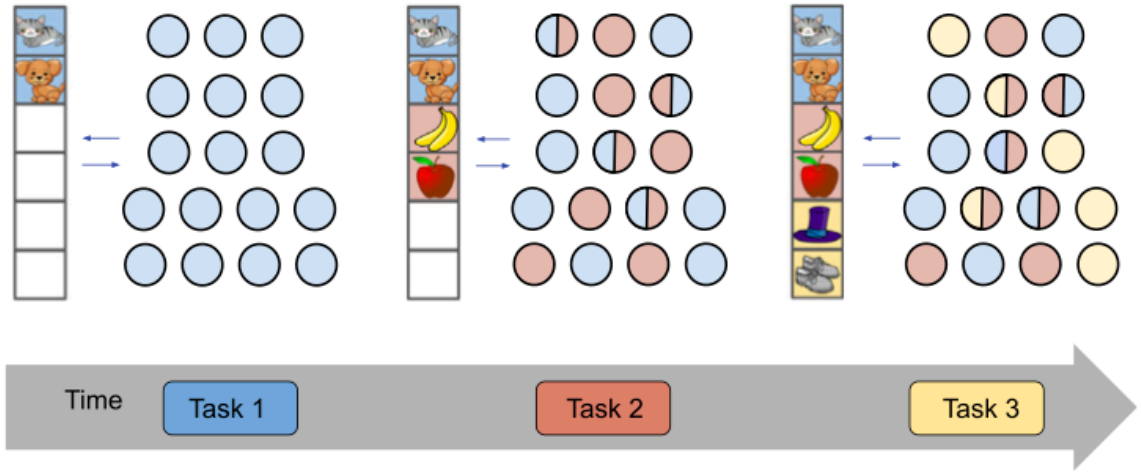
Set-ups exemplified with MNIST digits

In practice, “simple solutions” like storing a sub-set of relevant data as a “memory” have become vastly popular. Let’s look at these next

Strategy	Method	Budget	GM	Task-IL	Domain-IL	Class-IL
Baselines	None – lower target			84.32 (±0.99)	60.13 (±1.66)	19.89 (±0.02)
	Joint – upper target			99.67 (±0.03)	98.59 (±0.05)	98.17 (±0.04)
Context-specific components	Separate Networks	-	-	99.57 (±0.03)	-	-
	XdG	-	-	99.10 (±0.10)	-	-
Parameter regularization	EWC	-	-	99.06 (±0.15)	63.03 (±1.58)	20.64 (±0.52)
	SI	-	-	99.20 (±0.11)	66.94 (±1.13)	21.20 (±0.57)
Functional regularization	LwF	-	-	99.60 (±0.03)	71.18 (±1.42)	21.89 (±0.32)
	FROMP	100	-	99.12 (±0.13)	84.86 (±1.02)	77.38 (±0.64)
Replay	DGR	-	Yes	99.50 (±0.03)	95.57 (±0.30)	90.35 (±0.24)
	BI-R	-	Yes	99.61 (±0.03)	97.26 (±0.15)	94.41 (±0.15)
	ER	100	-	98.98 (±0.07)	93.75 (±0.24)	88.79 (±0.20)
	A-GEM	100	-	98.54 (±0.10)	87.67 (±1.33)	65.10 (±3.64)

van de Ven et al, “Three types of incremental learning”, Nature Machine Intelligence 4, 2022

Universität Bremen



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

Part 2 cont. - Retaining the Past

Memory: Rehearsal & Generative Modeling

Lifelong Machine Learning
Summer 2025

Prof. Dr. Martin Mundt

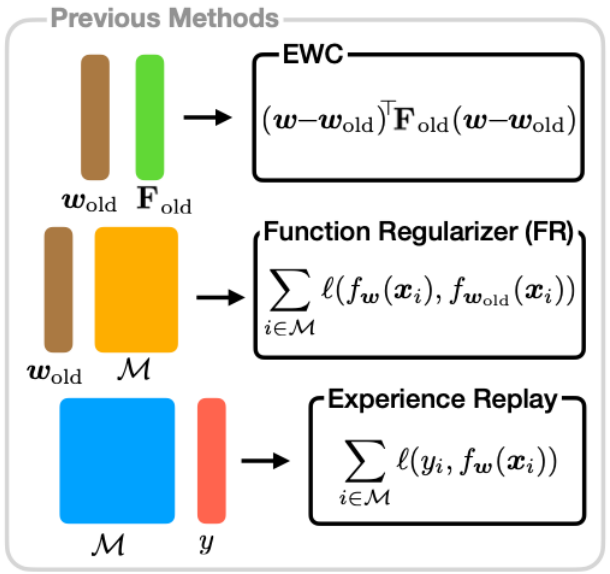
Intuitively simple: experience replay

The most straightforward “memory” is storing some random old data.

Daxberger et al, “Improving Continual Learning by Accurate Gradient Reconstructions of the Past”, TMLR 11/2023

Intuitively simple: experience replay

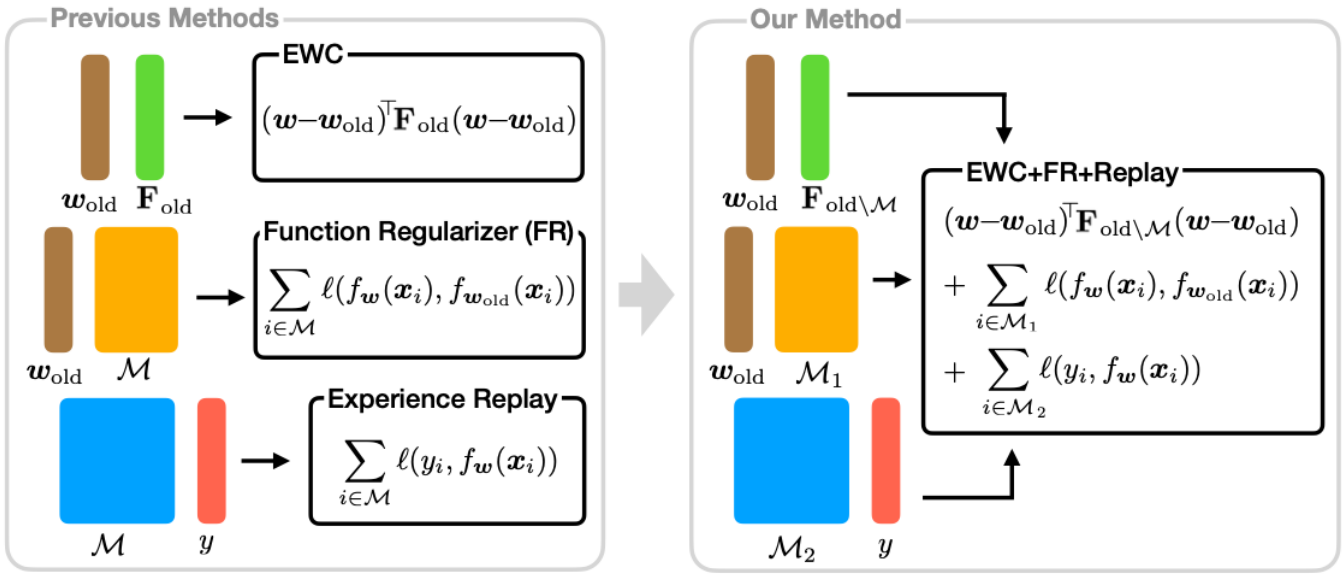
The most straightforward “memory” is storing some random old data.
This is very popular, and often taken as a “method itself”.



Daxberger et al, “Improving Continual Learning by Accurate Gradient Reconstructions of the Past”, TMLR 11/2023

Intuitively simple: experience replay

The most straightforward “memory” is storing some random old data.
This is very popular, and often taken as a “method itself”.
However, parameter & functional regularization benefit “anchors”!



Daxberger et al, “Improving Continual Learning by Accurate Gradient Reconstructions of the Past”, TMLR 11/2023

Universität Bremen

OWL ML

Question time

Assuming we can store part of the previous data, how would you find the most meaningful subset?

Universität Bremen

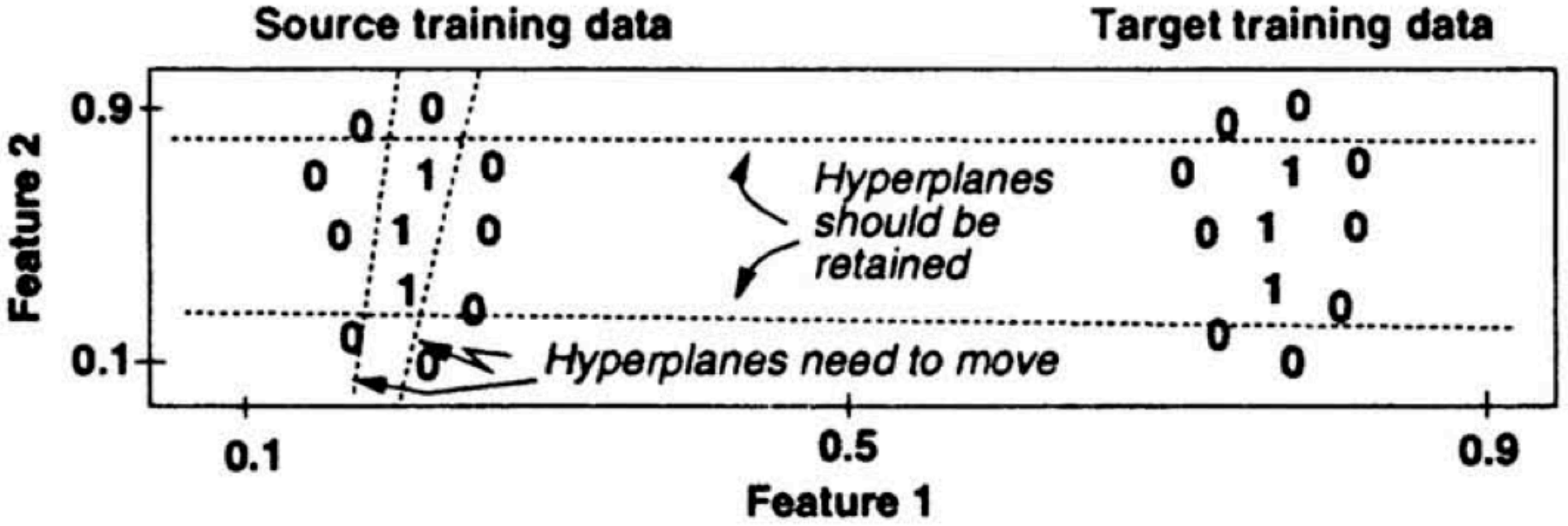
OWL/ML

Finding a meaningful subset of observed data

Let’s start with the intuition, then methods & this time, formalize last.

Finding a meaningful subset of observed data

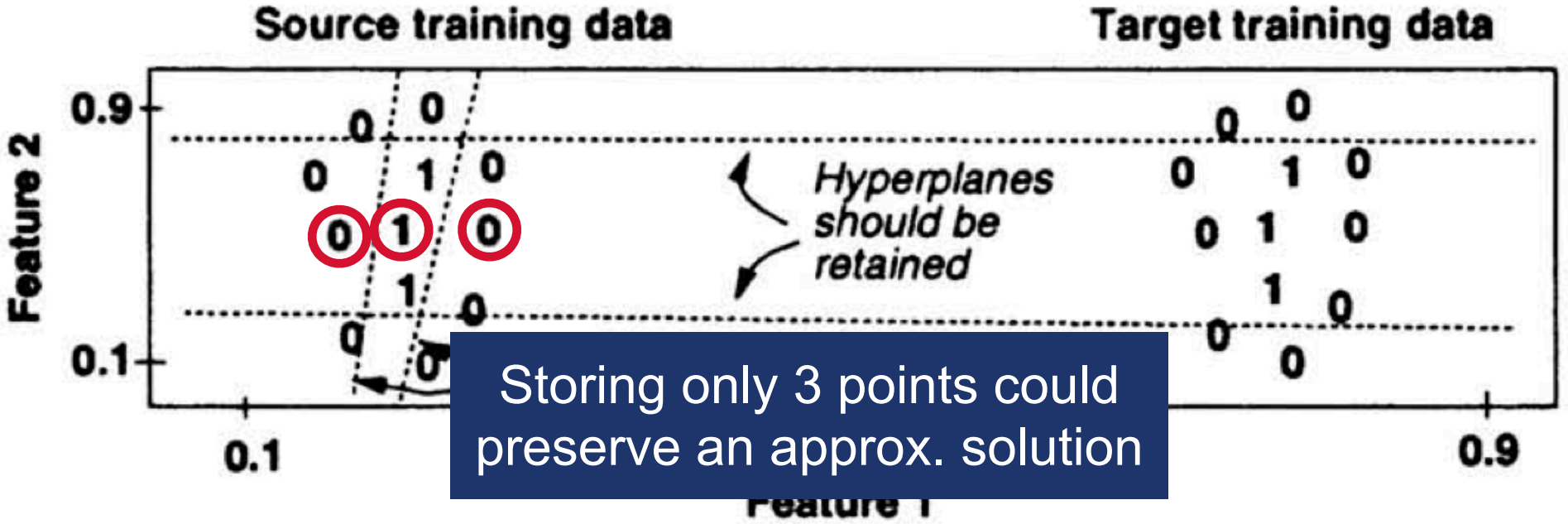
Let's start with the intuition, then methods & this time, formalize last.
Recall our transfer example: what would you store to avoid forgetting?



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

Finding a meaningful subset of observed data

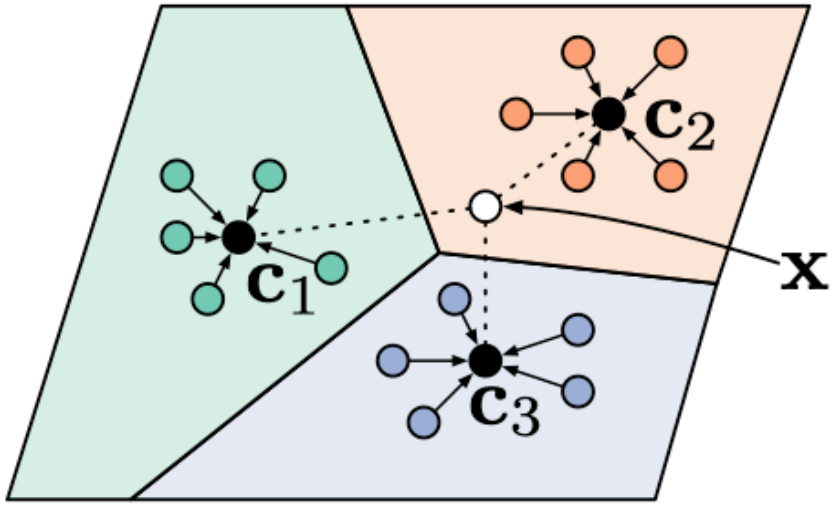
Let's start with the intuition, then methods & this time, formalize last.
Recall our transfer example: what would you store to avoid forgetting?



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

Finding a meaningful subset of observed data

We've also already encountered a second popular choice: storing data points that represent a "mean", e.g. in terms of class embeddings. Nomenclature is inconsistent, but often called "exemplar" or "prototype"



(a) Few-shot

"Prototypical Networks for Few-shot Learning", Snell et al, NeurIPS 2017

Universität Bremen

The benefit of nearest-mean classification

iCaRL: Incremental Classifier and Representation Learning

1. Stores a subset of data in a fixed size memory buffer
2. Classifies based on nearest class means
3. Consecutively replaces parts of memory buffer with new examples

Algorithm 1 iCaRL CLASSIFY

input x // image to be classified
require $\mathcal{P} = (P_1, \ldots, P_t)$ // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$ // feature map
  for $y = 1, \ldots, t$ **do**
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ // mean-of-exemplars
  end for
  $y^* \leftarrow \underset{y=1,\ldots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|$ // nearest prototype
output class label y^*

Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

Universität Bremen

Picking “exemplars” based on means

iCaRL: Incremental Classifier and Representation Learning

- Iteratively: one by one, based on “herding” (Welling ICML 2009)
- Pick exemplars to best approximate the overall mean
- For a size of k exemplars: loop k times

Algorithm 4 iCaRL CONSTRUCTEXEMPLARSET

input image set $X = \{x_1, \ldots, x_n\}$ of class y
input m target number of exemplars
require current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$
  $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ // current class mean
  for $k = 1, \ldots, m$ **do**
    $p_k \leftarrow \underset{x \in X}{\operatorname{argmin}} \left\| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right\|$
  end for
  $P \leftarrow (p_1, \ldots, p_m)$
output exemplar set P

Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

Universität Bremen

OWL ML

Question time

Our memory buffer is limited, how do we remove exemplars when new tasks arrive?

Universität Bremen

Replacing exemplars based on herding

iCaRL: Incremental Classifier and Representation Learning

- Memory buffer is like a prioritized list
- Later picked exemplars for a task "weigh" less
- Simply cut and repopulate

Algorithm 5 iCaRL REDUCEEXEMPLARSET

input m // target number of exemplars
input $P = (p_1, \ldots, p_{|P|})$ // current exemplar set
$P \leftarrow (p_1, \ldots, p_m)$ // *i.e.* keep only first m
output exemplar set P

Rebuffi et al, "iCaRL: Incremental Classifier and Representation Learning", CVPR 2017

Incremental training

iCaRL: Incremental Classifier and Representation Learning

- Update representations, reduce the exemplar set & add new ones

Algorithm 2 iCaRL INCREMENTALTRAIN

input $X^s, \ldots, X^t$ // training examples in per-class sets
input K // memory size
require Θ // current model parameters
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // current exemplar sets
$\Theta \leftarrow$ UPDATEREPRESENTATION$(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
$m \leftarrow K/t$ // number of exemplars per class
for $y = 1, \ldots, s-1$ **do**
$P_y \leftarrow$ REDUCEEXEMPLARSET(P_y, m)
end for
for $y = s, \ldots, t$ **do**
$P_y \leftarrow$ CONSTRUCTEXEMPLARSET(X_y, m, Θ)
end for
$\mathcal{P} \leftarrow (P_1, \ldots, P_t)$ // new exemplar sets

Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

Incremental training

iCaRL: Incremental Classifier and Representation Learning

- Update representations, reduce the exemplar set & add new ones
- Updating is based on
  - Concatenating dataset with exemplars (or interleaving them)
  - Using them as anchors for knowledge distillation

Algorithm 3 iCaRL UPDATEREPRESENTATION

input $X^s, \ldots, X^t$ // training images of classes $s, \ldots, t$
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // exemplar sets
require Θ // current model parameters
// form combined training set:

$$\mathcal{D} \leftarrow \bigcup_{y=s,\ldots,t} \{(x,y) : x \in X^y\} \cup \bigcup_{y=1,\ldots,s-1} \{(x,y) : x \in P^y\}$$

// store network outputs with pre-update parameters:
for $y = 1, \ldots, s-1$ **do**
$q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}$
end for
run network training (*e.g.* BackProp) with loss function

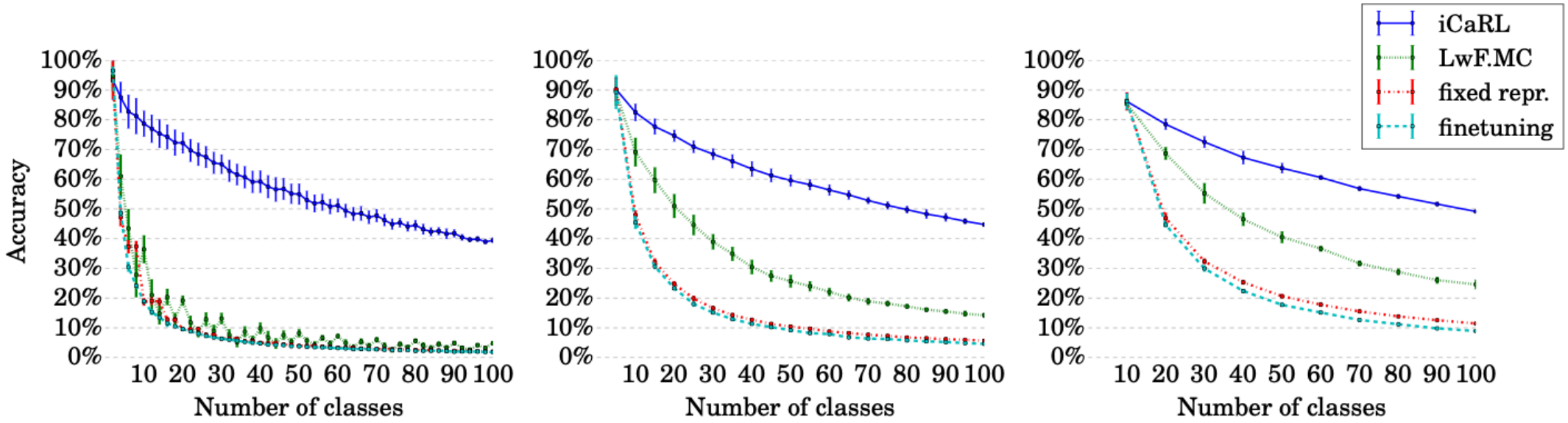
$$\ell(\Theta) = -\sum_{(x_i,y_i)\in\mathcal{D}} \Big[\sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y\neq y_i} \log(1-g_y(x_i)) + \sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1-q_i^y) \log(1-g_y(x_i)) \Big]$$

that consists of *classification* and *distillation* terms.

Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

iCaRL in practice

As expected, the stored exemplars boost knowledge distillation, here on an incremental CIFAR-100 image classification task.

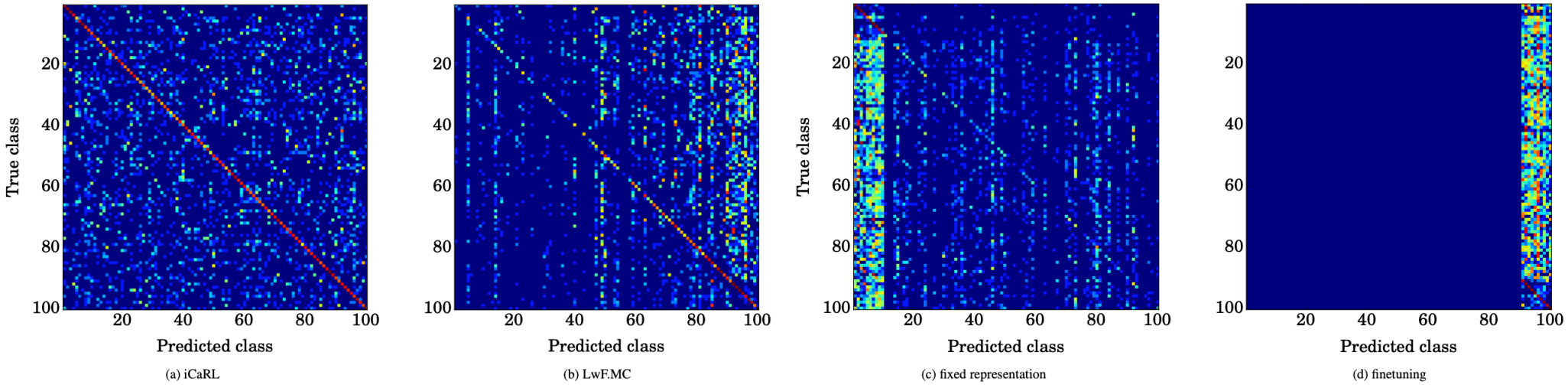


Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

iCaRL in practice

As expected, the stored exemplars boost knowledge distillation, here on an incremental CIFAR-100 image classification task.

Confusion matrices empirically confirm our intuition



Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

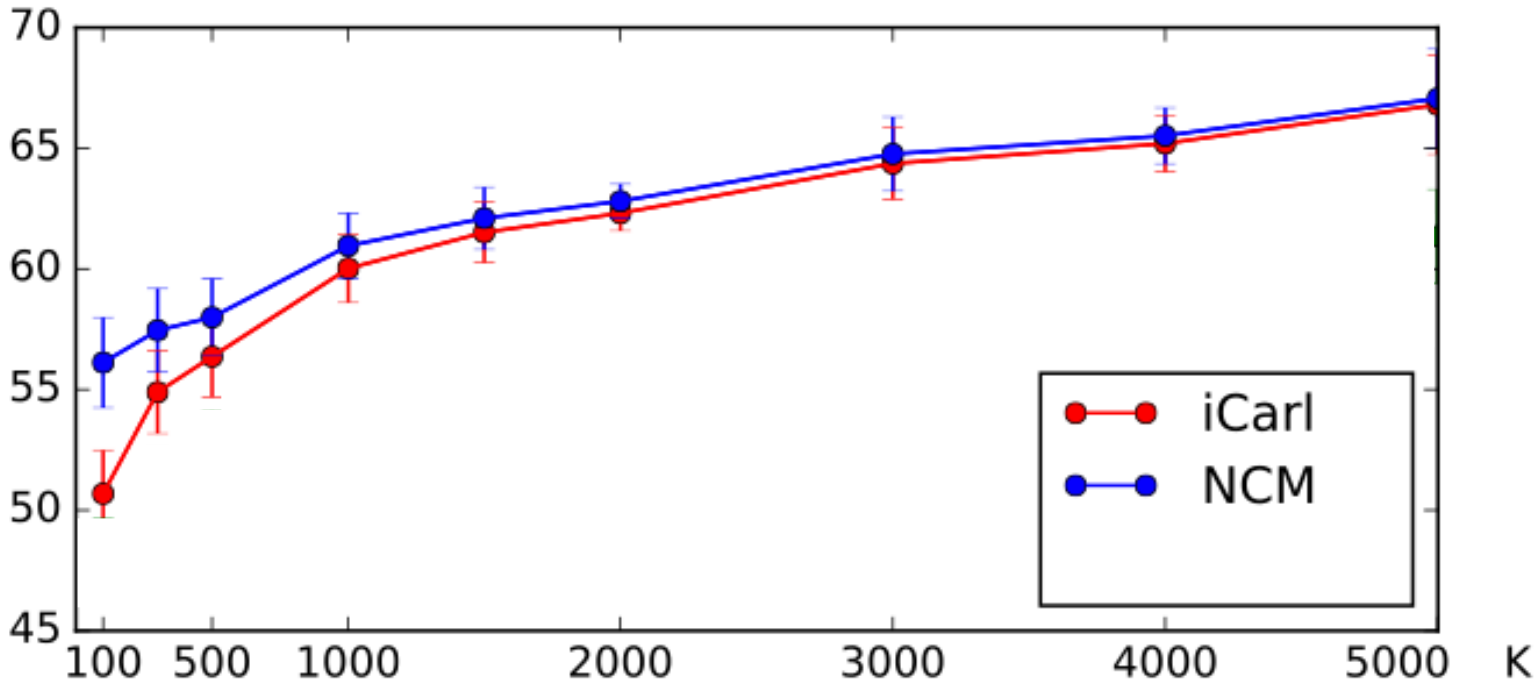
Universität Bremen

Question time

How do you expect iCaRL to be influenced by the number of stored exemplars/memory buffer size?

iCaRL in practice: the memory budget

Intuitively, more stored exemplars (K) also lead to less forgetting



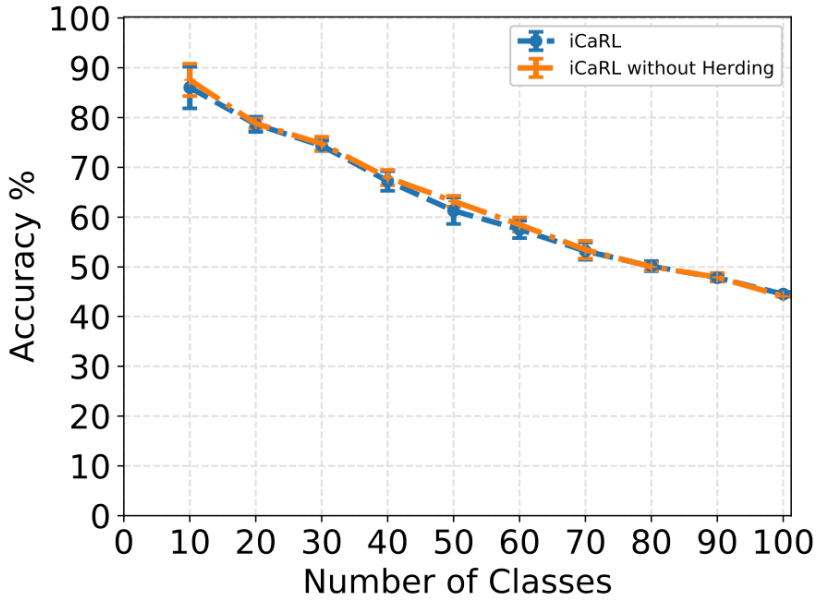
Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

iCaRL in practice: the impact of herding

Does herding outperform random data selection in incremental learning?

Algorithm 1: Herding Algorithm for Instance Selection

Input: Trained Model M, $C_i^j \in$ Images of class i , Size k;
Output: Set containing k instances of class C_i;
$\forall C_i^j \in C_i$, use M to get the feature map F_i^j;
Let S be a null set;
Compute the mean of all F_i^j. Let this be F_i^{mean};
Select F_i^j and add it in S such that mean of selected set is closest to F_i^{mean};
If $|S| < k$, repeat step 5. Else, return S.



Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

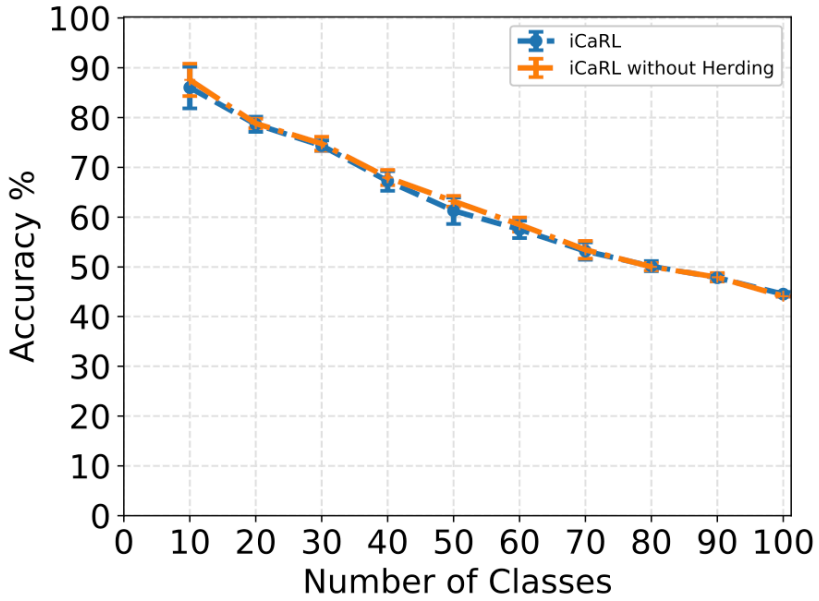
iCaRL in practice: the impact of herding

Does herding outperform random data selection in incremental learning? Unfortunately not, at least not in these kinds of tasks.

Algorithm 1: Herding Algorithm for Instance Selection

Input: Trained Model M, $C_i^j \in$ Images of class i , Size k;
Output: Set containing k instances of class C_i;
$\forall C_i^j \in C_i$, use M to get the feature map F_i^j;
Let S be a null set;
Compute the mean of all F_i^j. Let this be F_i^{mean};
Select F_i^j and add it in S such that mean of selected set is closest to F_i^{mean};
If $|S| < k$, repeat step 5. Else, return S.

Question time: Why?
What are potential issues?



Rebuffi et al, “iCaRL: Incremental Classifier and Representation Learning”, CVPR 2017

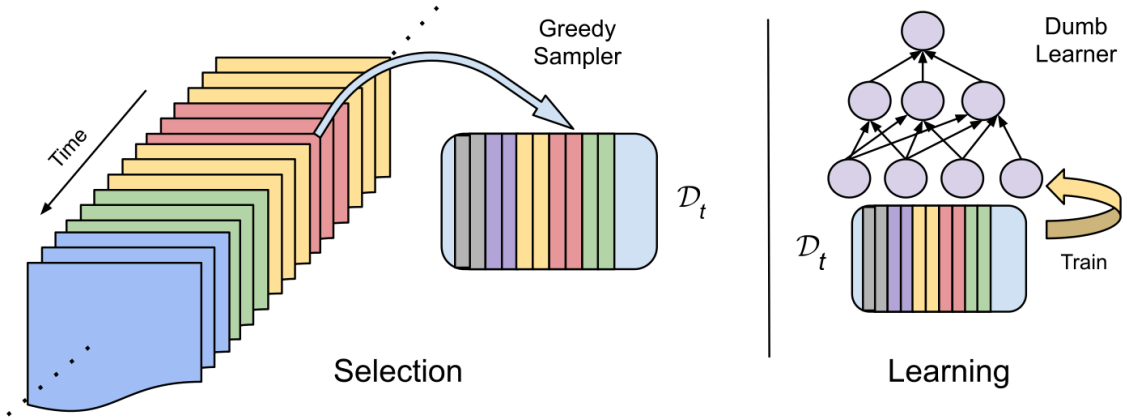
Universität Bremen

The role of memory & how to select it

Choosing to store a data memory is very helpful! Selecting an appropriate one that’s better than random, seems much harder.

The role of memory & how to select it

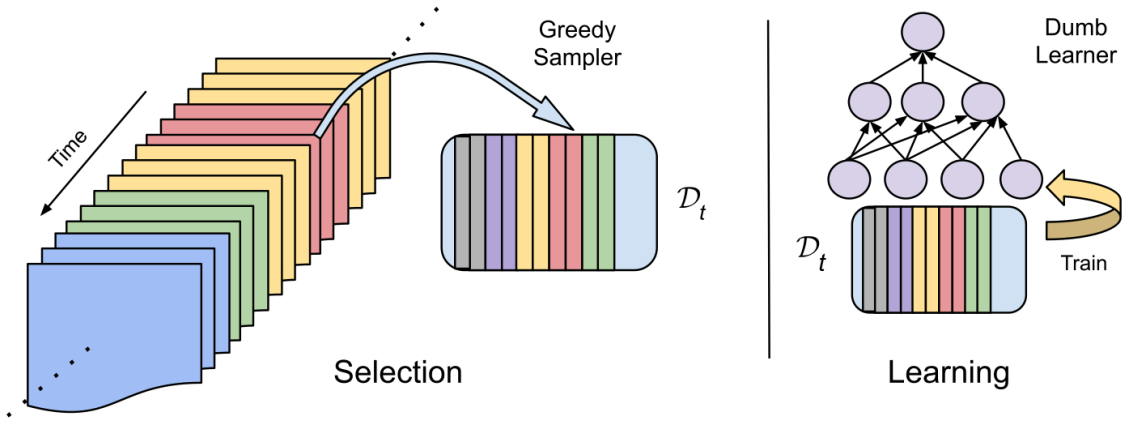
Choosing to store a data memory is very helpful! Selecting an appropriate one that's better than random, seems much harder. The GDumb baselines makes this clear



Prabu et al, "GDumb: A Simple Approach that Questions our Approach to Continual Learning", ECCV 2020

The role of memory & how to select it

Choosing to store a data memory is very helpful! Selecting an appropriate one that's better than random, seems much harder. The GDumb baselines makes this clear



Method (k)	CIFAR100 (1105)
RWalk [8]	40.9 ± 3.97
EWC [6]	42.4 ± 3.02
Base	42.9 ± 2.07
MAS [32]	44.2 ± 2.39
SI [5]	47.1 ± 4.41
iCARL [4]	50.1
GDumb	60.3 ± 0.85

Prabu et al, "GDumb: A Simple Approach that Questions our Approach to Continual Learning", ECCV 2020

Universität Bremen

OWL-ML

Question time

Surely, there must exist a better and more formal way to select an appropriate subset: any ideas?

Formalizing the problem: coresets

"*coresets are small, (weighted) summaries of large data sets such that solutions found on the summary itself are* ***provably competitive*** *with solutions found on the full data set*"

$$|\texttt{cost}(P, Q) - \texttt{cost}(C, Q)| \leq \varepsilon \cdot \texttt{cost}(P, Q)$$

Good introductions are: Bachem et al, "Practical Coreset Constructions for Machine Learning" (2017) or Jubran et al, "Introduction to Coresets: Accurate Coresets" (2019)

Formalizing the problem: coresets

"*coresets are small, (weighted) summaries of large data sets such that solutions found on the summary itself are* ***provably competitive*** *with solutions found on the full data set*"

$$|\texttt{cost}(P, Q) - \texttt{cost}(C, Q)| \leq \varepsilon \cdot \texttt{cost}(P, Q)$$

Note how this is specific to some data (here called P), a set of queries (here called Q), and thus also the choice of model, and the cost/loss function!

Good introductions are: Bachem et al, "Practical Coreset Constructions for Machine Learning" (2017) or Jubran et al, "Introduction to Coresets: Accurate Coresets" (2019)

Formalizing the problem: coresets

"*coresets are small, (weighted) summaries of large data sets such that solutions found on the summary itself are* ***provably competitive*** *with solutions found on the full data set*"

$$|\texttt{cost}(P, Q) - \texttt{cost}(C, Q)| \leq \varepsilon \cdot \texttt{cost}(P, Q)$$

Note how this is specific to some data (here called P), a set of queries (here called Q), and thus also the choice of model, and the cost/loss function!

We will refer to a core set as "*accurate*" if it does not introduce approximation errors when compressing the original data

Good introductions are: Bachem et al, "Practical Coreset Constructions for Machine Learning" (2017) or Jubran et al, "Introduction to Coresets: Accurate Coresets" (2019)

Universität Bremen

Many *accurate* coresets are known, but here, we are interested in approximate scenarios

Name	Input Weighted Set (P,w) of size $\lvert P\rvert = n$	Query Set $\mathcal{X}$	cost function $f: P \times \mathcal{X}$	loss for $f(p,x)$ over $p \in P$	Coreset C	Coreset Weights	Const. time	Query time	Section
1-Center	$P \subseteq \ell \subseteq \mathbb{R}^d$ $w \equiv 1$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = \lVert p - x\rVert$	$\lVert\cdot\rVert_\infty$	$C \subseteq P$ $\lvert C\rvert = 2$	$u \equiv 1$	$O(n)$	$O(d)$	3.1
Monotonic function	$P \subseteq \mathbb{R}$ $w \equiv 1$	$\mathcal{X} = \{g \mid g$ is monotonic decreasing/increasing or increasing and then decreasing function$\}$	$f(p,g) = g(p)$	$\lVert\cdot\rVert_\infty$	$C \subseteq P$ $\lvert C\rvert = 2$	$u \equiv 1$	$O(n)$	$O(1)$	3.2
Vectors sum (1)	$P \subseteq \mathbb{R}^d$ $w : P \to \mathbb{R}$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = p - x$	Σ	$C \subseteq \mathbb{R}^d$ $\lvert C\rvert = 1$	$u \equiv \sum_{p\in P} w(p)$	$O(n)$	$O(d)$	3.3
Vectors sum (2)	$P \subseteq \mathbb{R}^d$ $w : P \to \mathbb{R}$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = p - x$	Σ	$C \subseteq P$ $\lvert C\rvert \le d+1$	$u : C \to \mathbb{R}$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$	$O(nd^2)$	$O(d^2)$	3.3.1
Vectors sum (3)	$P \subseteq \mathbb{R}^d$ $w : P \to [0,\infty)$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = p - x$	Σ	$C \subseteq P$ $\lvert C\rvert \le d+2$	$u : C \to \left[0, \sum_{p\in P} w(p)\right]$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$	$O\big(\min\{n^2d^2,$ $nd + d^4 log n\}\big)$	$O(d^2)$	3.3.2
1-Mean (1)	$P \subseteq \mathbb{R}^d$ $w : P \to \mathbb{R}$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = w(p)\lVert p - x\rVert^2$	$\lVert\cdot\rVert_1$	$C \subseteq \mathbb{R}^d \times \mathbb{Z} \times \mathbb{R}$ $\lvert C\rvert = 3$ Different loss	*unweighted*	$O(nd)$	$O(d)$	3.4
1-Mean (2)	$P \subseteq \mathbb{R}^d$ $w : P \to \mathbb{R}$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = w(p)\lVert p - x\rVert^2$	$\lVert\cdot\rVert_1$	$C \subseteq P$ $\lvert C\rvert \le d+2$	$u : C \to \mathbb{R}$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$ $\sum_{p\in C} u(p)\lVert p\rVert^2 = \sum_{p\in P} w(p)\lVert p\rVert^2$	$O(nd^2)$	$O(d^2)$	3.4.1
1-Mean (3)	$P \subseteq \mathbb{R}^d$ $w : P \to [0,\infty)$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = w(p)\lVert p - x\rVert^2$	$\lVert\cdot\rVert_1$	$C \subseteq P$ $\lvert C\rvert \le d+3$	$u : C \to \left[0, \sum_{p\in P} w(p)\right]$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$ $\sum_{p\in C} u(p)\lVert p\rVert^2 = \sum_{p\in P} w(p)\lVert p\rVert^2$	$O\big(\min\{n^2d^2,$ $nd + d^4 log n\}\big)$	$O(d^2)$	3.4.2
1-Segment	$P = \{(t_i \mid p_i)\}_{i=1}^n \subseteq \mathbb{R}^{d+1}$ $w : P \to [0,\infty)$	$\mathcal{X} = \{g \mid g : \mathbb{R} \to \mathbb{R}^d\}$	$f((t,p),g) = \lVert p - g(t)\rVert^2$	$\lVert\cdot\rVert_1$	$C \subseteq \mathbb{R}^{d+1}$ $\lvert C\rvert = d+2$	$u \equiv 1$	$O(nd^2)$	$O(d^2)$	3.5
Matrix 2-norm (1)	$P \subseteq \mathbb{R}^d$ $w : P \to [0,\infty)$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = (p^T x)^2$	$\lVert\cdot\rVert_1$	$C \subseteq \mathbb{R}^d$ $\lvert C\rvert = d$	$u \equiv 1$	$O(nd^2)$	$O(d^2)$	3.6
Matrix 2-norm (2)	$P \subseteq \mathbb{R}^d$ $w : P \to [0,\infty)$	$\mathcal{X} = \mathbb{R}^d$	$f(p,x) = (p^T x)^2$	$\lVert\cdot\rVert_1$	$C \subseteq P$ $\lvert C\rvert \le d^2+1$	$u : C \to \left[0, \sum_{p\in P} w(p)\right]$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$	$O\big(\min\{n^2d^4,$ $nd^2 + d^8 log n\}\big)$	$O(d^3)$	3.6.1
Least Mean Squares	$P = \{(a_i^T \mid b_i)\}_{i=1}^n \subseteq \mathbb{R}^{d+1}$ $w : P \to [0,\infty)$	$\mathcal{X} = \mathbb{R}^d$	$f((a^T \mid b),x) = (a^T x - b)^2$	$\lVert\cdot\rVert_1$	$C \subseteq P$ $\lvert C\rvert \le (d+1)^2+1$	$u : C \to \left[0, \sum_{p\in P} w(p)\right]$ $\sum_{p\in C} u(p) = \sum_{p\in P} w(p)$	$O\big(\min\{n^2d^4,$ $nd^2 + d^8 log n\}\big)$	$O(d^3)$	3.7

Jubran et al, “Introduction to Coresets: Accurate Coresets” (2019)

Grad-Match: Data Subset Selection for DNNs

A popular relaxation for (deep) neural networks is to find the subset that closely matches the gradients, motivated by the use of SGD.

$$||\sum_{i \in C} w_i \nabla_\theta \mathscr{L}_i(\theta) - \nabla_\theta \mathscr{L}(\theta)|| + \lambda ||w||^2$$

In other words, minimize above equation to find weighing & subset.

Killamsetty et al, “GRAD-MATCH: Gradient Matching based Data Subset Selection for Efficient Deep Model Training”, ICML 2021

Grad-Match: Data Subset Selection for DNNs

A popular relaxation for (deep) neural networks is to find the subset that closely matches the gradients, motivated by the use of SGD.

$$||\sum_{i \in C} w_i \nabla_\theta \mathscr{L}_i(\theta) - \nabla_\theta \mathscr{L}(\theta)|| + \lambda ||w||^2$$

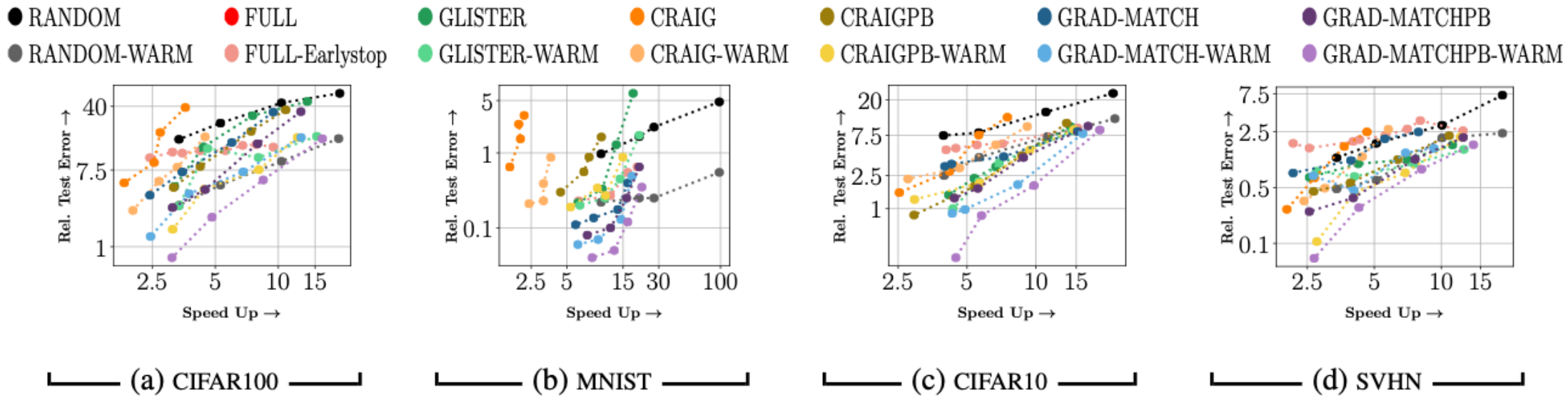
In other words, minimize above equation to find weighing & subset.

In practice, a greedy algorithm like matching pursuit can be used (trying to find the best matching projections). Ultimately, our choices will influence the approximation guarantees & bounds we receive.

Killamsetty et al, “GRAD-MATCH: Gradient Matching based Data Subset Selection for Efficient Deep Model Training”, ICML 2021

Grad-Match: Data Subset Selection for DNNs

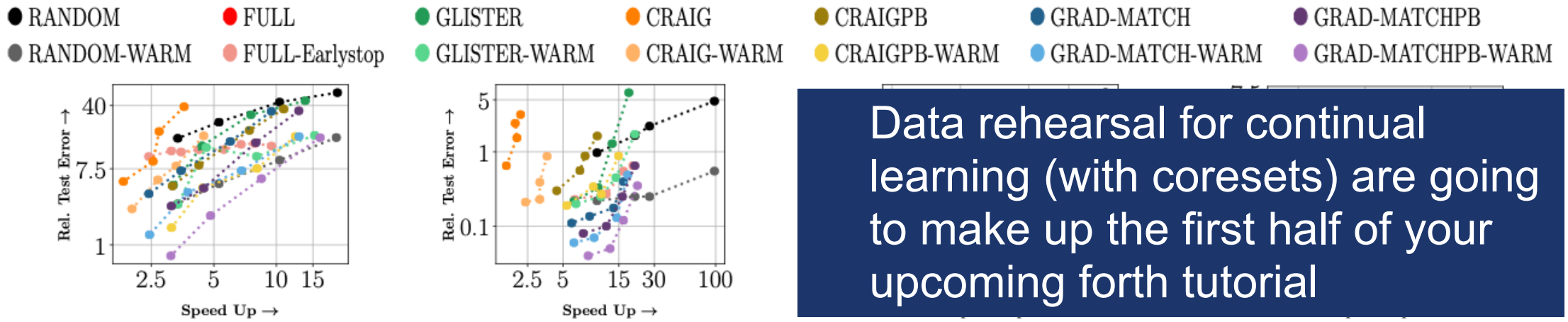
Respective coreset selection algorithms perform much better than random selection (here relative to full training & obtained speed-up)



Killamsetty et al, “GRAD-MATCH: Gradient Matching based Data Subset Selection for Efficient Deep Model Training”, ICML 2021

Grad-Match: Data Subset Selection for DNNs

Respective coreset selection algorithms perform much better than random selection (here relative to full training & obtained speed-up)



Data rehearsal for continual learning (with coresets) are going to make up the first half of your upcoming forth tutorial

A word of caution: the literature (especially continual ML), trends to be very loose in calling an approach a “coreset” - often relying on extreme heuristics

Killamsetty et al, “GRAD-MATCH: Gradient Matching based Data Subset Selection for Efficient Deep Model Training”, ICML 2021

Universität Bremen

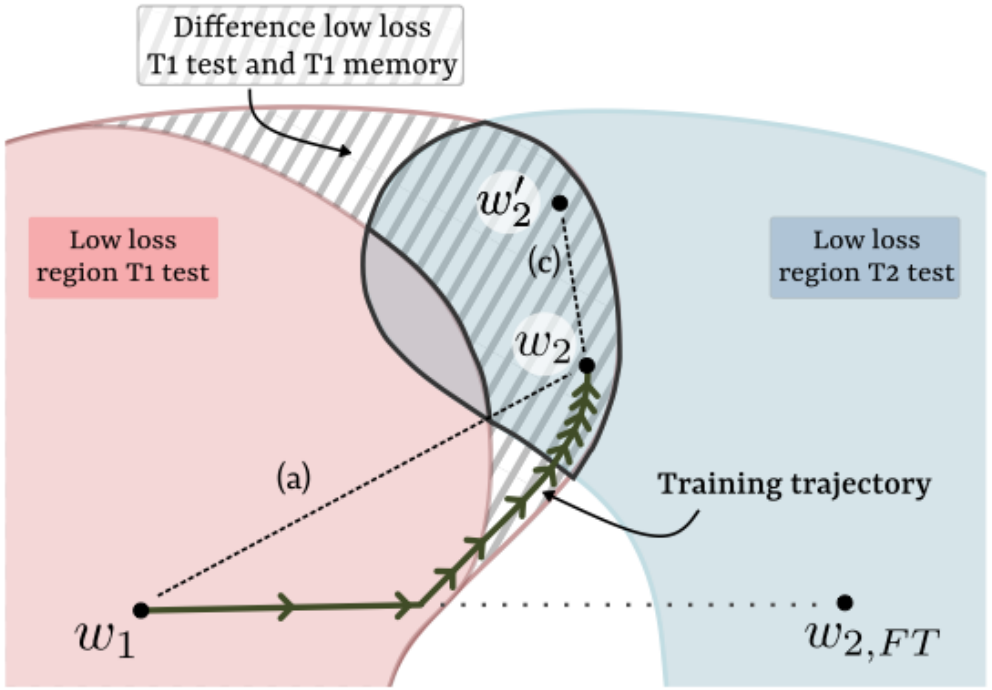
In the continual set-up, a key additional challenge is overfitting to the task A memory

In continual learning, coresets get more complicated, because we are not just approximating the task A loss, but also optimizing B,C…

In the continual set-up, a key additional challenge is overfitting to the task A memory

In continual learning, coresets get more complicated, because we are not just approximating the task A loss, but also optimizing B,C…

It has been empirically observed, that rehearsal “deflects” the learning trajectory at a high-loss ridge of T1’s low loss region, where the test loss becomes high again



Verwimp et al, “Rehearsal revealed: the limits and merits of revisiting samples in continual learning”, ICCV 2021

Question time

Sub-sets and core sets are great, but there are concerns, also including so far neglected privacy. Can you think of other ways to construct “memory”?

Biologically plausible memory?

“While it is an effective method in ANNs, rehearsal is unlikely to be a realistic model of biological learning mechanisms, as in this context the actual old information (accurate and complete representation of all items ever learned by the organism) is not available.”

R. French, “Pseudo-recurrent Connectionist Networks: An Approach to the Sensitivity-Plasticity Dilemma”, Connection Science 9:4, 1997

Biologically plausible memory?

"While it is an effective method in ANNs, rehearsal is unlikely to be a realistic model of biological learning mechanisms, as in this context the actual old information (accurate and complete representation of all items ever learned by the organism) is not available. Pseudorehearsal is significantly more likely to be a mechanism which could actually be employed by organisms as it does not require access to this old information, it just requires a way of approximating it."

R. French, "Pseudo-recurrent Connectionist Networks: An Approach to the Sensitivity-Plasticity Dilemma", Connection Science 9:4, 1997

Biologically plausible memory?

“Pseudorehearsal is based on the use in the rehearsal process of artificially constructed populations of “pseudoitems” instead of the “actual” previously learned items.”

A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal”, Journal of Neural Computing 7: 123-146, 1995

Biologically plausible memory?

“Pseudorehearsal is based on the use in the rehearsal process of artificially constructed populations of “pseudoitems” instead of the “actual” previously learned items. A pseudoitem is constructed by generating a new input vector (setting at random 50% of input elements to 0 and 50% to 1 as usual), and passing it forward through the network in the standard way. Whatever output vector this input generates becomes the associated target output”

A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal”, Journal of Neural Computing 7: 123-146, 1995

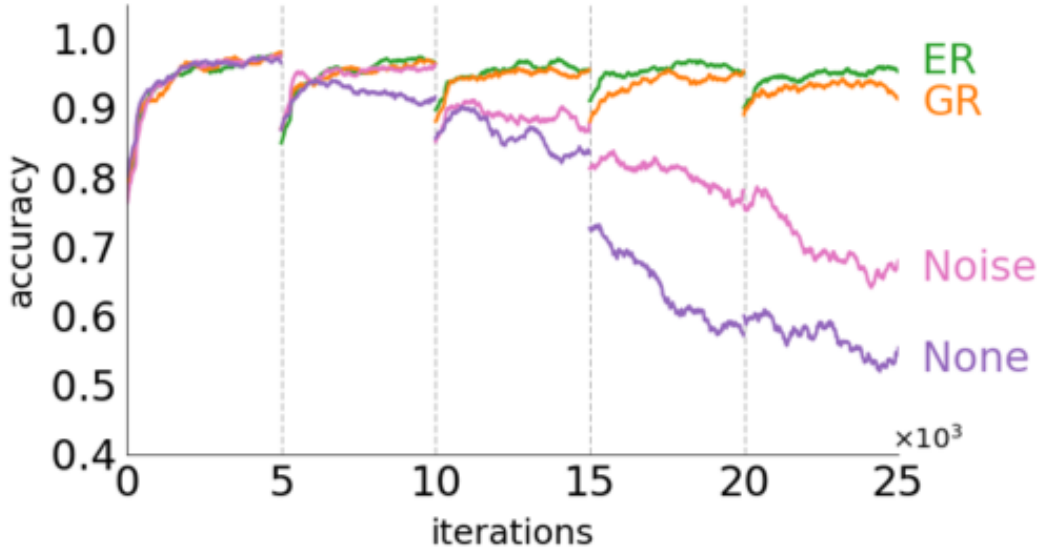
Biologically plausible memory?

“Pseudorehearsal is based on the use in the rehearsal process of artificially constructed populations of “pseudoitems” instead of the “actual” previously learned items. A pseudoitem is constructed by generating a new input vector (setting at random 50% of input elements to 0 and 50% to 1 as usual), and passing it forward through the network in the standard way. Whatever output vector this input generates becomes the associated target output”

In practice, random noise as pseudo patterns is not going to suffice

A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal”, Journal of Neural Computing 7: 123-146, 1995

Biologically plausible memory?

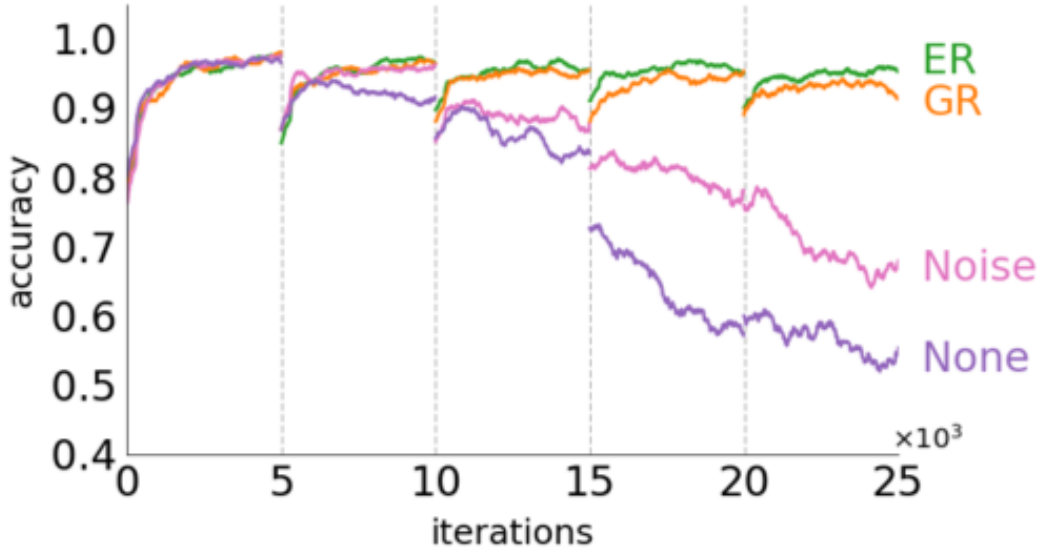


Example: despite permuted MNIST digits looking like “noise”, replaying noise deteriorates, much in contrast to experience replay.

In practice, random noise as pseudo patterns is not going to suffice

A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal”, Journal of Neural Computing 7: 123-146, 1995. Figure from Shin et al, “Continual Learning with Deep Generative Replay”, NeurIPS 2017

Biologically plausible memory?



Example: despite permuted MNIST digits looking like “noise”, replaying noise deteriorates, much in contrast to experience replay. What does the GR line do?

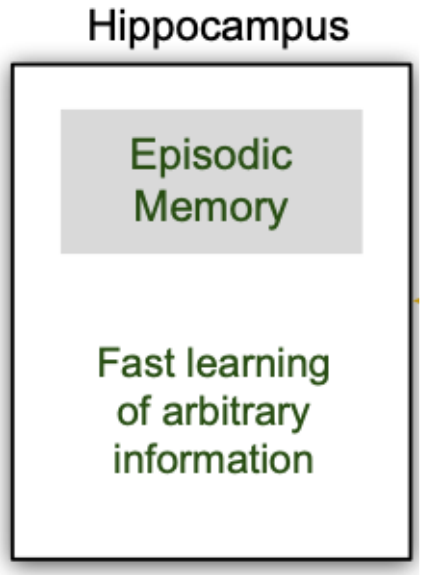
In practice, random noise as pseudo patterns is not going to suffice

A. Robins, “Catastrophic forgetting, rehearsal and pseudorehearsal”, Journal of Neural Computing 7: 123-146, 1995. Figure from Shin et al, “Continual Learning with Deep Generative Replay”, NeurIPS 2017

Drawing inspiration from the brain

Complementary learning systems theory, simplified:

- Hippocampus: short-term adaptation & rapid learning of novel information

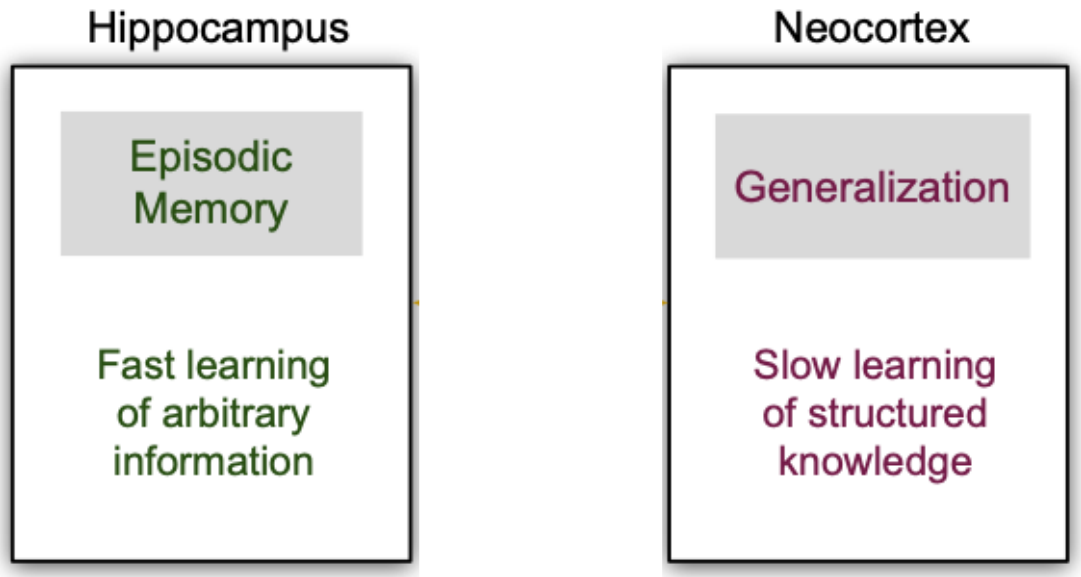


McClelland et al, "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex", Psychological Review Vol 101, 1995 Figure from review by Parisi et al, "Continual Lifelong Learning with Neural Networks: A Review", Neural Networks 113, 2019

Drawing inspiration from the brain

Complementary learning systems theory, simplified:

- Hippocampus: short-term adaptation & rapid learning of novel information
- Neocortex: slow learning, to consolidate & build up overlapping representations

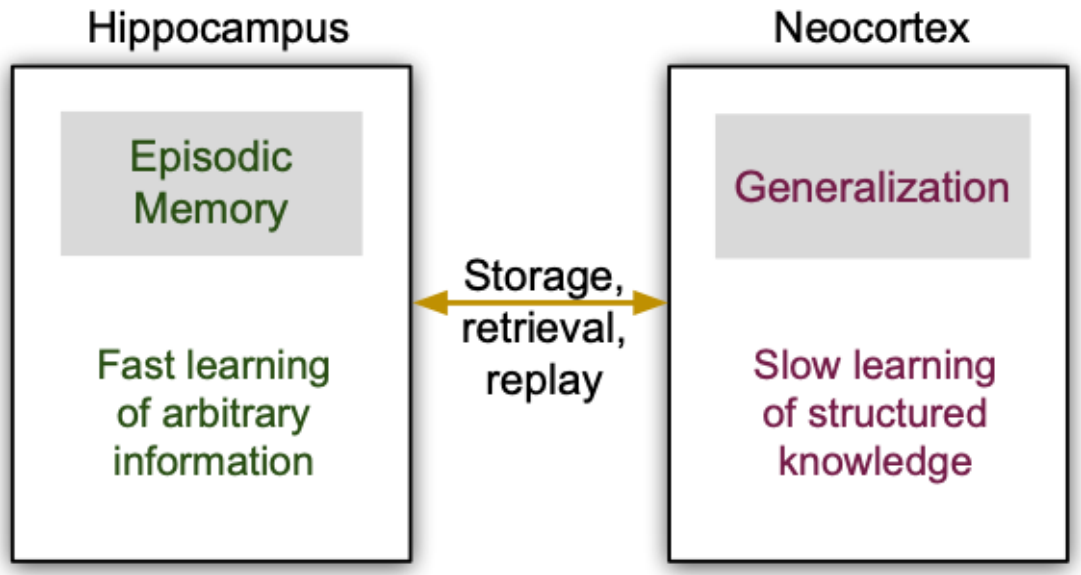


McClelland et al, "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex", Psychological Review Vol 101, 1995 Figure from review by Parisi et al, "Continual Lifelong Learning with Neural Networks: A Review", Neural Networks 113, 2019

Drawing inspiration from the brain

Complementary learning systems theory, simplified:

- Hippocampus: short-term adaptation & rapid learning of novel information
- Neocortex: slow learning, to consolidate & build up overlapping representations
- Hippocampus "*plays back*" over time to neocortex



McClelland et al, "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex", Psychological Review Vol 101, 1995 Figure from review by Parisi et al, "Continual Lifelong Learning with Neural Networks: A Review", Neural Networks 113, 2019

Universität Bremen

OWL ML

Question time

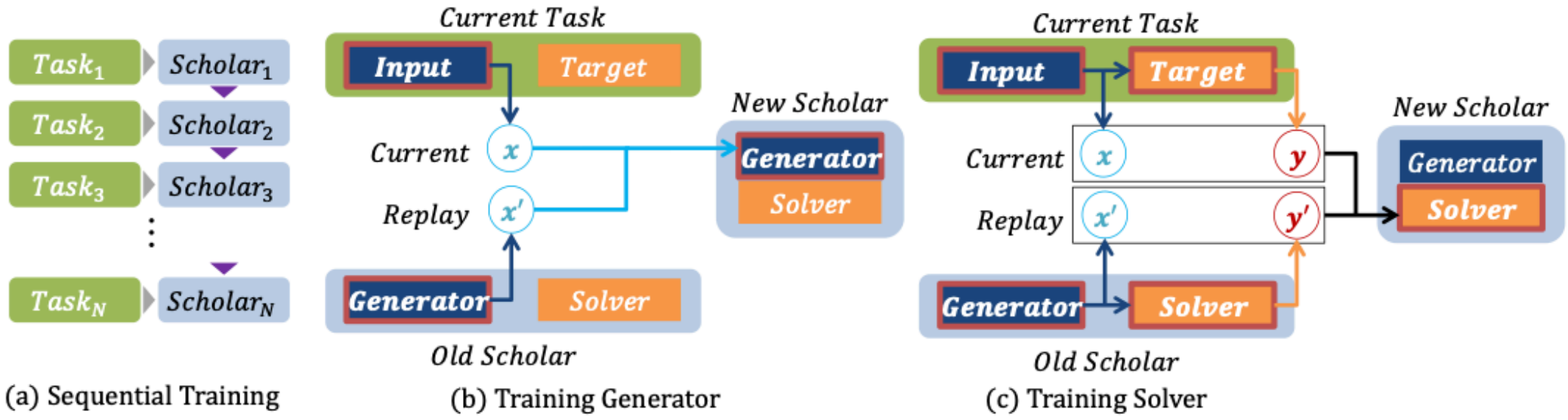
Can you think of a straightforward implementation of the key aspect of CLS in machine learning?

Pseudo-rehearsal -> (deep) generative replay

We can train two models: a “solver” and a “generator”, or more formally a discriminative and a generative model. In a straightforward set-up, we would alternate their training

Pseudo-rehearsal -> (deep) generative replay

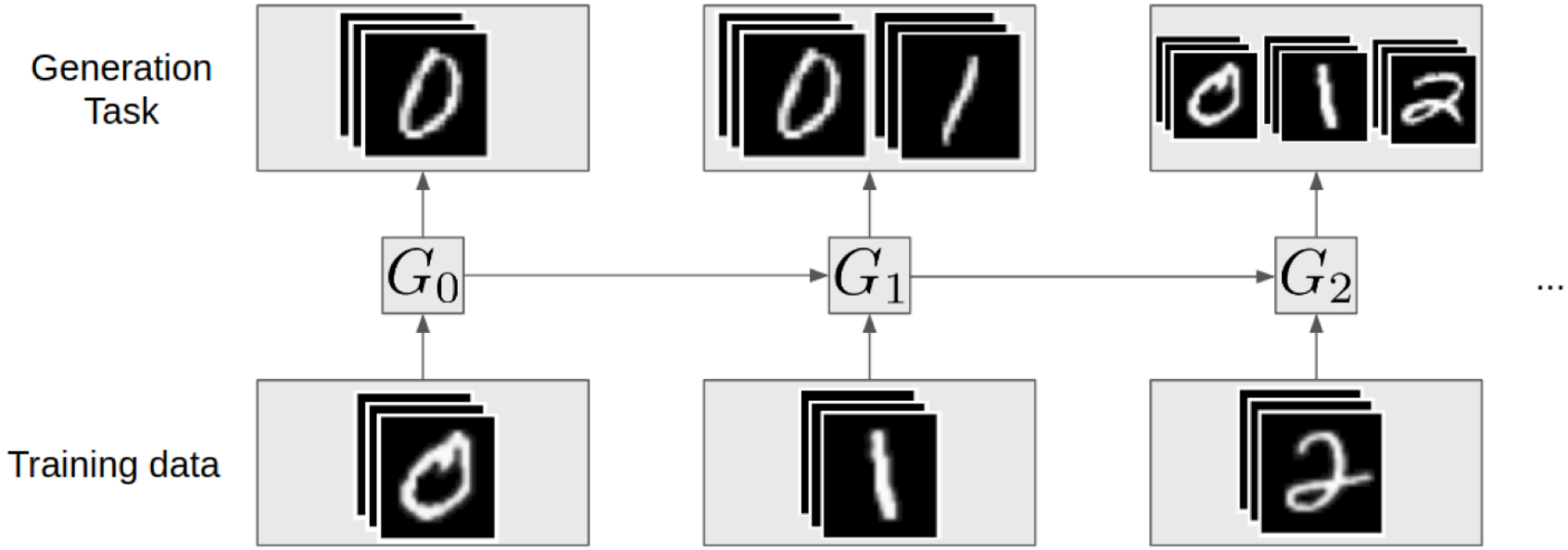
We can train two models: a “solver” and a “generator”, or more formally a discriminative and a generative model. In a straightforward set-up, we would alternate their training



Shin et al, “Continual Learning with Deep Generative Replay”, NeurIPS 2017

Generative replay in practice: incremental digits

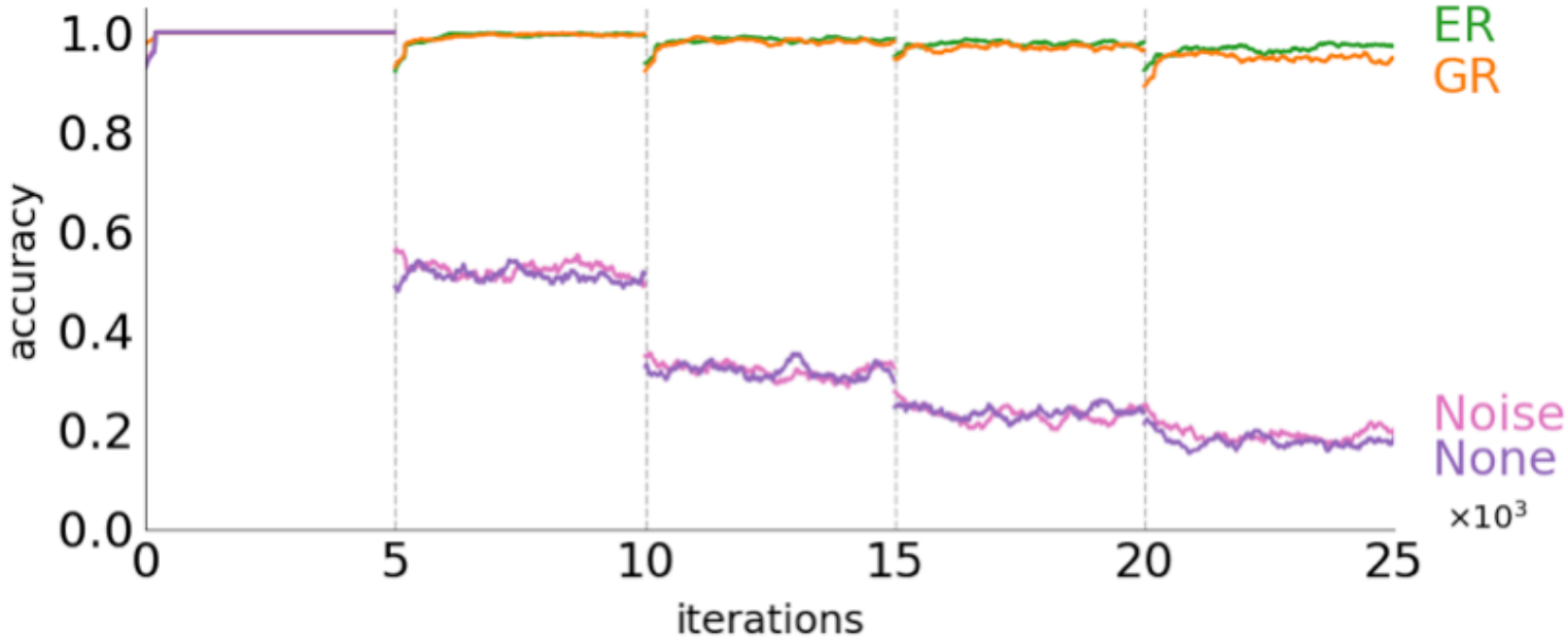
Of course, we will either need to store a generator per task, or have the generator train on its own past samples (more on this later)



Lesort et al, “Generative Models from the perspective of Continual Learning”, IJCNN 2019

Generative replay in practice: incremental digits

Naturally, there are many choices of generative models to choose from. In general: we will need a generator that is sufficiently powerful



Shin et al, “Continual Learning with Deep Generative Replay”, NeurIPS 2017

Universität Bremen

OWL ML

Question time

What is a generative model (formally) and how do they work? What approaches do you know?

Generative models: the briefest summary

A **discriminative** model typically learns something like p(y|x)

A **generative** model learns to model the data distribution p(x) and the process by which the data is created (the generative factors)

Generative models: the briefest summary

A **discriminative** model typically learns something like p(y|x)

A **generative** model learns to model the data distribution p(x) and the process by which the data is created (the generative factors)

Starting based on a generative approach does not mean we cannot also solve discriminative tasks like classification:

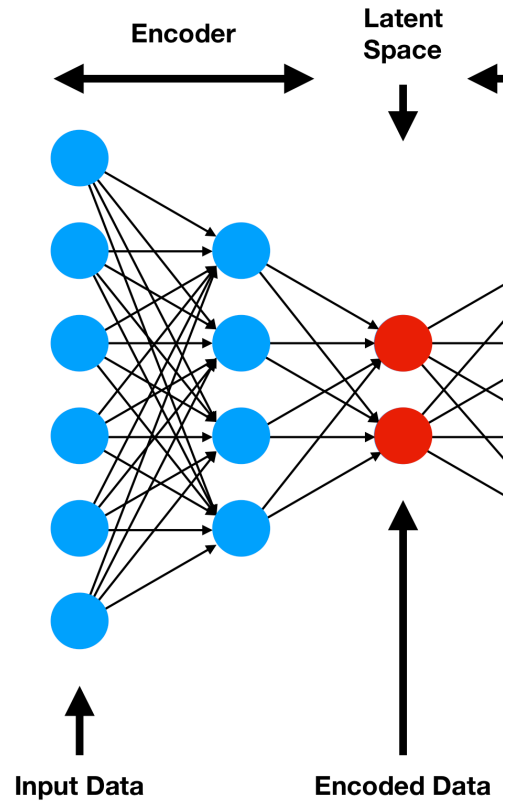
$$p(x,y) = p(y|x)p(x)$$

Let us look at a concrete example of a flexible (deep) generative model based on the auto-encoding principle

From autoencoding to variational autoencoders

How do autoencoders work?

- Learn a representation, an “encoding” of the data
- An **encoder** maps to a “code”, typically referred to as “latent representation” or “latent space”

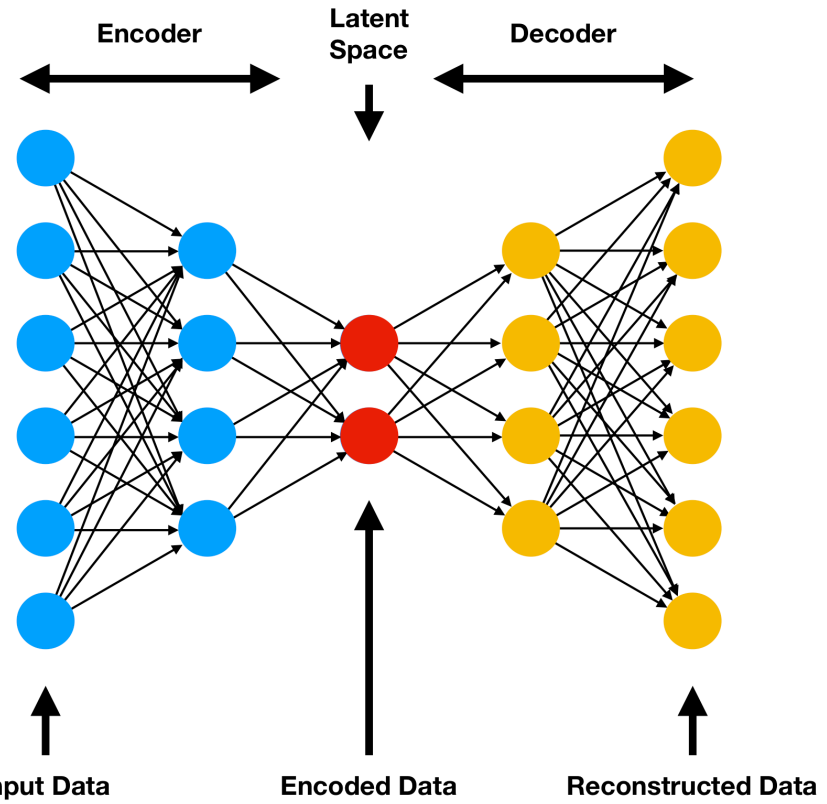


https://www.compthree.com/blog/autoencoder/

From autoencoding to variational autoencoders

How do autoencoders work?

- Learn a representation, an “encoding” of the data
- An **encoder** maps to a “code”, typically referred to as “latent representation” or “latent space”
- We tune this generally compressed representation by having a **decoder** learn how to reconstruct the input



https://www.compthree.com/blog/autoencoder/

From autoencoding to variational autoencoders

The “code”/latent embeddings of an autoencoder are however difficult to grasp, we do not know how they are distributed

In consequence, for our purpose of generative replay, it will be hard for us to generate because we don’t know how to sample from the autoencoder.

From autoencoding to variational autoencoders

The “code”/latent embeddings of an autoencoder are however difficult to grasp, we do not know how they are distributed

In consequence, for our purpose of generative replay, it will be hard for us to generate because we don’t know how to sample from the autoencoder.

A very popular approach (which won the test of time award at ICLR 2024) is thus to constrain the latent space to follow a specific distribution and employ variational inference for learning: a VAE (“Auto-encoding Variational Bayes”, Kingma & Welling, ICLR 2014)

From autoencoding to variational autoencoders

The idea behind VAEs simplified in a picture based on a Gaussian prior

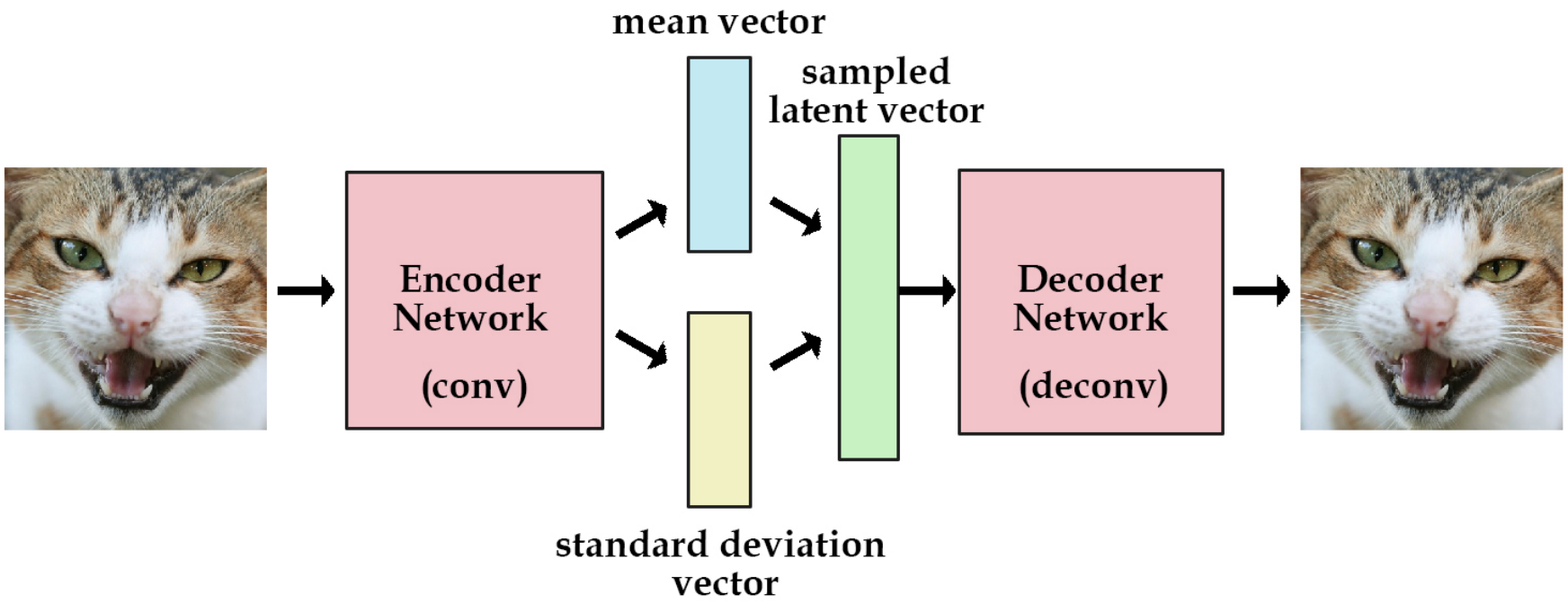


Image from: https://kvfrans.com/content/images/2016/08/vae.jpg

VAEs formally: why gen. models make sense

- A dataset with variable x
- Data is generated by a random process involving unobserved random variable z
- z is generated from some prior distribution $p_\theta(z)$
- A value x is generated from some conditional distribution $p_\theta(x \mid z)$

VAEs formally: why gen. models make sense

- A dataset with variable x
- Data is generated by a random process involving unobserved random variable z
- z is generated from some prior distribution $p_\theta(z)$
- A value x is generated from some conditional distribution $p_\theta(x \mid z)$

But, the parameters and values of latent variables z are not known to us.

$$p_\theta(x) = \int p_\theta(x, z) dz \text{ is intractable}$$

Let us try to find a way to approximately solve this problem

VAE derivation in a nutshell

Here is what we'll do conceptually:

- We will make use of Bayes rule to describe the marginal prob. p(x)
- We will make use of Variational Inference to approximate the unknown real posterior p(z|x) by something we will call q(z|x)

VAE derivation in a nutshell

Here is what we'll do conceptually:

- We will make use of Bayes rule to describe the marginal prob. p(x)
- We will make use of Variational Inference to approximate the unknown real posterior p(z|x) by something we will call q(z|x)
- We will make use of the concept of divergences between distributions (which will always be positive) to bound quantities we cannot compute
- We will arrive at a loss that we can optimize by easily mapping the objectives to auto-encoding

VAE derivation: Evidence Lower Bound (ELBO)

1. The densities of the marginal & joint are related through **Bayes rule**:

$$p_\theta(z \mid x) = \frac{p_\theta(x, z)}{p_\theta(x)} = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)}$$

VAE derivation: Evidence Lower Bound (ELBO)

1. The densities of the marginal & joint are related through **Bayes rule**:

$$p_\theta(z \mid x) = \frac{p_\theta(x, z)}{p_\theta(x)} = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)}$$

- p(z|x): posterior (updated probability after considering evidence)
- p(x|z): likelihood (probability of the evidence, given belief is true)
- p(z): prior (probability before the evidence is considered)
- p(x): marginal (probability of the evidence under any circumstance)
- Quotient p(x|z)/p(x): the support x provides for z

VAE derivation: Evidence Lower Bound (ELBO)

1. The densities of the marginal & joint are related through **Bayes rule**:

$$p_\theta(z \mid x) = \frac{p_\theta(x, z)}{p_\theta(x)} = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)}$$

- p(z|x): posterior (updated probability after considering evidence)
- p(x|z): likelihood (probability of the evidence, given belief is true)
- p(z): prior (probability before the evidence is considered)
- p(x): marginal (probability of the evidence under any circumstance)
- Quotient p(x|z)/p(x): the support x provides for z

2. Make use of the **logarithm** on both sides to write as a sum:

$$\log p_\theta(x) = \log p_\theta(x \mid z) + \log p_\theta(z) - \log p_\theta(z \mid x)$$

VAE derivation: Evidence Lower Bound (ELBO)

3. We do not know our real posterior $p_\theta(z \mid x)$. We will make use of **variational inference** and introduce an **approximation** $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \int q_\phi(z \mid x) \left[\log p_\theta(x \mid z) + \log p_\theta(z) - \log p_\theta(z \mid x) \right] dz$$

VAE derivation: Evidence Lower Bound (ELBO)

3. We do not know our real posterior $p_\theta(z|x)$. We will make use of **variational inference** and introduce an **approximation** $q_\phi(z|x)$:

$$\log p_\theta(x) = \int q_\phi(z|x)\left[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x)\right] dz$$

4. We will do a trick that will be useful later: **add and subtract** $q_\theta(z|x)$:

$$\log p_\theta(x) = \int q_\phi(z|x)\left[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x) + \log q_\phi(z|x) - \log q_\phi(z|x)\right] dz$$

VAE derivation: Evidence Lower Bound (ELBO)

3. We do not know our real posterior $p_\theta(z|x)$. We will make use of **variational inference** and introduce an **approximation** $q_\phi(z|x)$:

$$\log p_\theta(x) = \int q_\phi(z|x)\left[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x)\right] dz$$

4. We will do a trick that will be useful later: **add and subtract** $q_\theta(z|x)$:

$$\log p_\theta(x) = \int q_\phi(z|x)\left[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x) + \log q_\phi(z|x) - \log q_\phi(z|x)\right] dz$$

Note that these tricks don't change anything, step 4 just adds/subtracts 0 and the term introduced in step 3 integrates to 1

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = \int q_\phi(z|x)\Big[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x) + \log q_\phi(z|x) - \log q_\phi(z|x)\Big]\, dz$$

5. Make use of the **Kullback Leibler divergence** between distributions:

$$KL(Q||P) = \int_{\infty}^{\infty} Q(x) \log \frac{Q(x)}{P(x)}$$

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = \int q_\phi(z|x)\Big[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x) + \log q_\phi(z|x) - \log q_\phi(z|x)\Big]\, dz$$

5. Make use of the **Kullback Leibler divergence** between distributions:

$$KL(Q||P) = \int_{\infty}^{\infty} Q(x)\log\frac{Q(x)}{P(x)}$$

Question time: we can formulate 2 KLD, which?

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = \int q_\phi(z|x)\Big[\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(z|x) + \log q_\phi(z|x) - \log q_\phi(z|x)\Big]\, dz$$

5. Make use of the **Kullback Leibler divergence** between distributions:

$$KL(Q||P) = \int_{\infty}^{\infty} Q(x) \log \frac{Q(x)}{P(x)}$$

We get a KLD wrt the likelihood & prior + one for true & approx. posterior

To approximate the integral, we take the **expectation over samples**:

$$\log p_\theta(x) = KL\left[q_\theta(z|x)||p_\theta(z|x)\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z|x)||p_\theta(z)\right]$$

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = KL\left[q_\theta(z\,|\,x)\,||\,p_\theta(z\,|\,x)\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x\,|\,z)\right] - KL\left[q_\phi(z\,|\,x)\,||\,p_\theta(z)\right]$$

6. We still **cannot evaluate** the first term on the right hand side. As from the beginning, we do not know the **true posterior** (why we introduced q)

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = KL\left[q_\theta(z|x)||p_\theta(z|x)\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z|x)||p_\theta(z)\right]$$

6. We still **cannot evaluate** the first term on the right hand side. As from the beginning, we do not know the **true posterior** (why we introduced q)

-> But, **KLD is strictly positive**! So at best, we know that we are “only off” by this posit factor, if we **optimize the remaining terms:**

$$\log p_\theta(x) = KL\left[q_\theta(z|x)||p_\theta(z|x)\right] + \mathcal{L}(\theta, \phi; x)$$

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) = KL\left[q_\theta(z|x)||p_\theta(z|x)\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z|x)||p_\theta(z)\right]$$

6. We still **cannot evaluate** the first term on the right hand side. As from the beginning, we do not know the **true posterior** (why we introduced q)

-> But, **KLD is strictly positive**! So at best, we know that we are “only off” by this posit factor, if we **optimize the remaining terms:**

$$\log p_\theta(x) = KL\left[q_\theta(z|x)||p_\theta(z|x)\right] + \mathscr{L}(\theta,\phi;x)$$

7. We now have a **variational lower-bound to p(x)** (the ELBO)

$$\log p_\theta(x) \geq \mathscr{L}(\theta,\phi;x) = \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z|x)||p_\theta(z)\right]$$

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \mid z) \right] - KL \left[q_\phi(z \mid x) \mid\mid p_\theta(z) \right]$$

What does this loss tell us?

- We can think of the first term on the RHS as the **expected reconstruction error** given by the log-likelihood, based on *samples from our posterior approximation*

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \mid z) \right] - KL \left[q_\phi(z \mid x) \mid\mid p_\theta(z) \right]$$

What does this loss tell us?

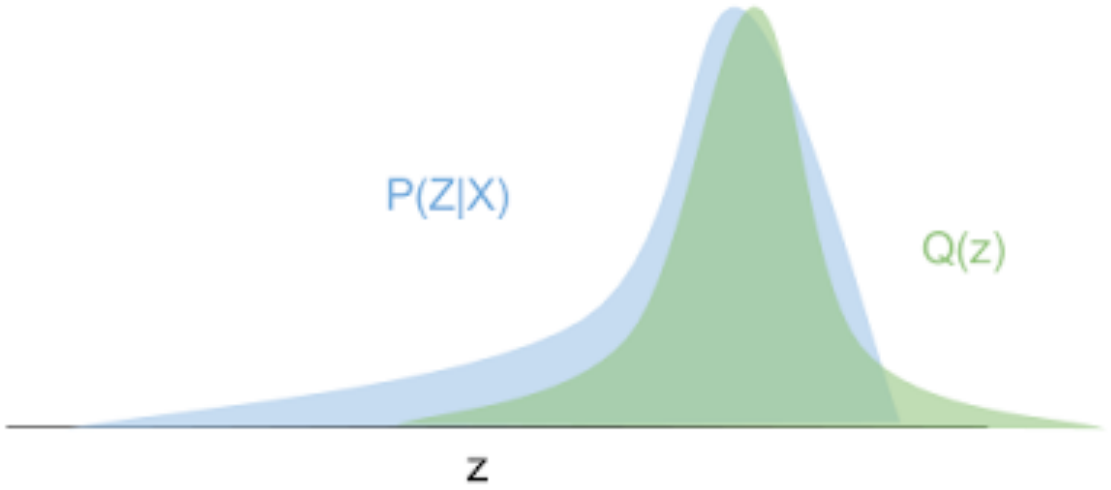
- We can think of the first term on the RHS as the **expected reconstruction error** given by the log-likelihood, based on *samples from our posterior approximation*
- We can hink of the second term on the RHS as the **KL divergence** which encourages the *approximate posterior to be close to the prior*

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \mid z) \right] - KL \left[q_\phi(z \mid x) \,||\, p_\theta(z) \right]$$

What does the KLD part tell us?

(Reverse) KLD has an intuitive interpretation: it measures the amount of information (in natural units, or units of 1/log2 bits) required to “distort” p(z) into q(z)



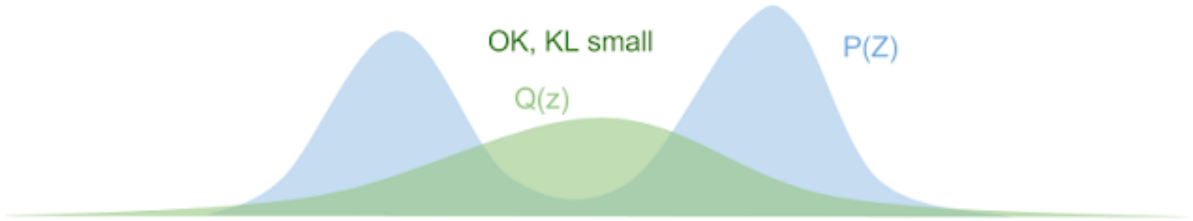
https://blog.evjang.com/2016/08/variational-bayes.html

VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \mid z) \right] - KL \left[q_\phi(z \mid x) \,||\, p_\theta(z) \right]$$

A side-note: KLD is not symmetric. Which is important when dealing with multiple modes

Forward: “stretches” the q(z) to cover **over** the entire p(z)



https://blog.evjang.com/2016/08/variational-bayes.html

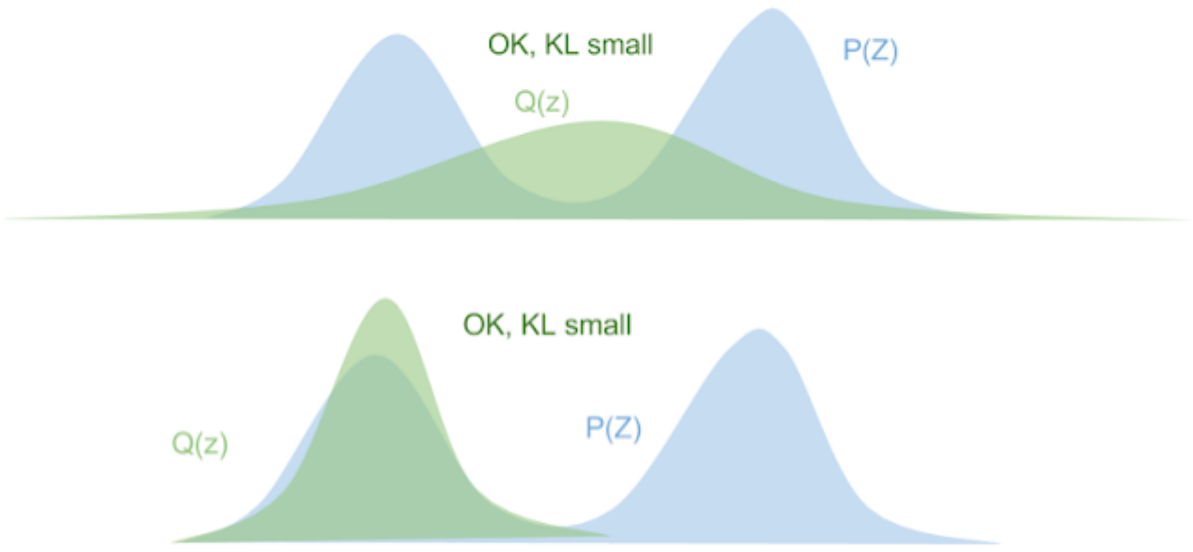
VAE derivation: Evidence Lower Bound (ELBO)

$$\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \mid z) \right] - KL \left[q_\phi(z \mid x) \,||\, p_\theta(z) \right]$$

A side-note: KLD is not symmetric. Which is important when dealing with multiple modes

Forward: “stretches” the q(z) to cover **over** the entire p(z)

Reverse: “squeezes” the q(z) **under** p(z)

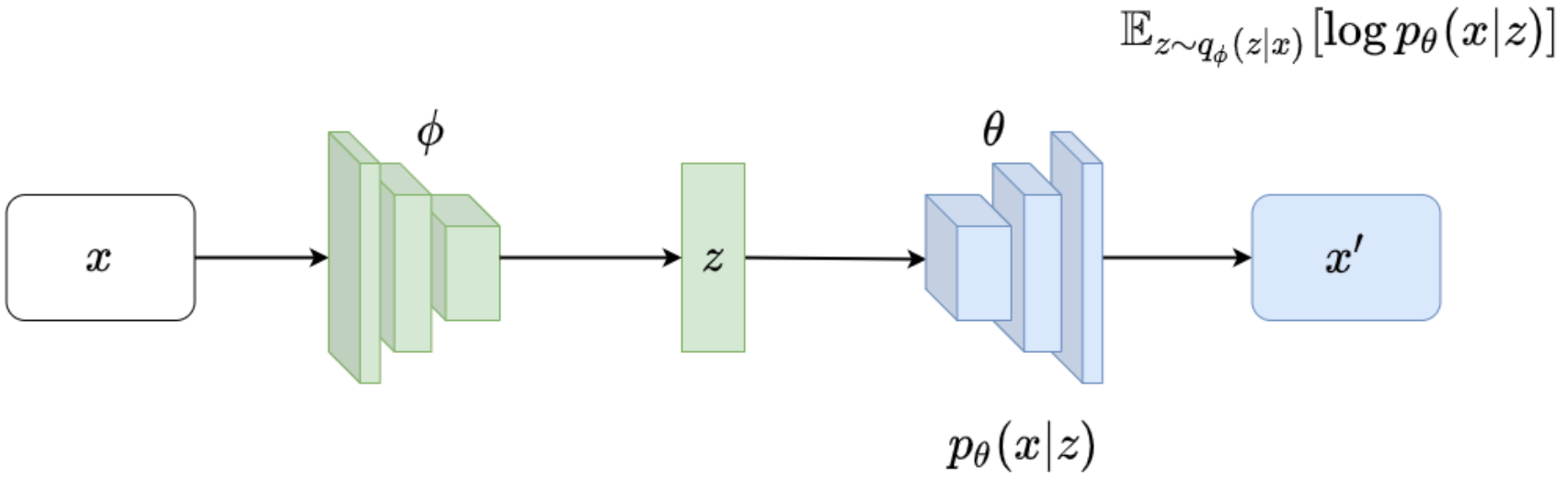


https://blog.evjang.com/2016/08/variational-bayes.html

Variational auto-encoding with neural nets

In a nutshell:

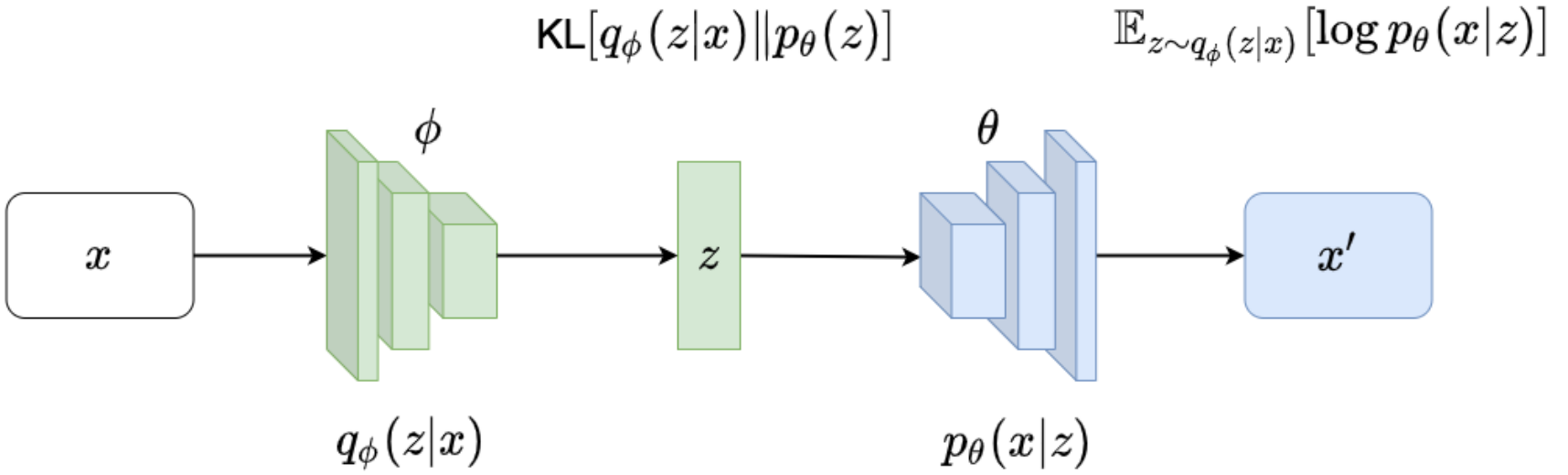
- A probabilistic decoder reconstructs based on latent factors: $p_\theta(x \mid z)$



Variational auto-encoding with neural nets

In a nutshell:

- A probabilistic decoder reconstructs based on latent factors: $p_\theta(x \mid z)$
- A probabilistic encoder, the “recognition” model, $q_\phi(z \mid x)$ is a variational approximation to the intractable posterior $p_\theta(z \mid x)$



Variational auto-encoding with neural nets

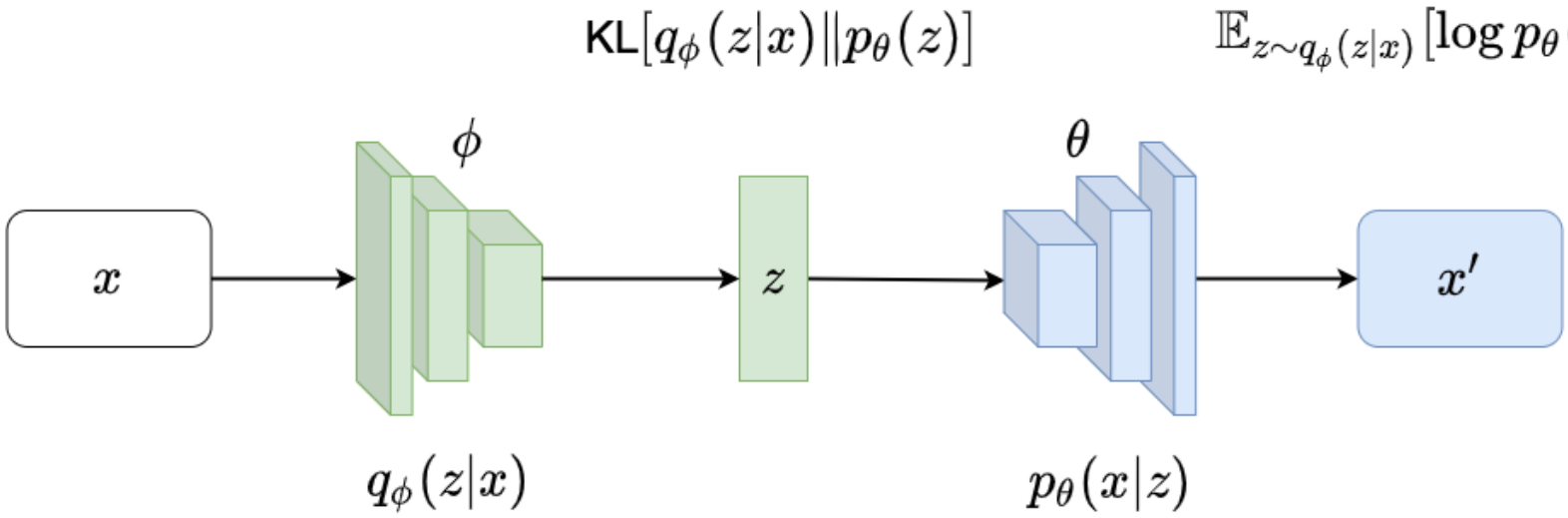
More formally:

- A probabilistic decoder reconstructs based on latent factors: $p_\theta(x \mid z)$
  -> for samples from the approximate posterior, the decoder's parameters are trained to produce a distribution over possible values of x given z
- A "recognition" model, a probabilistic encoder, $q_\phi(z \mid x)$ is a variational approximation to the intractable posterior $p_\theta(z \mid x)$
  -> given a datapoint x, the encoder's parameters are trained to produce a distribution over possible values of z from which it could have been generated

Variational auto-encoding with neural nets

Intuitively:

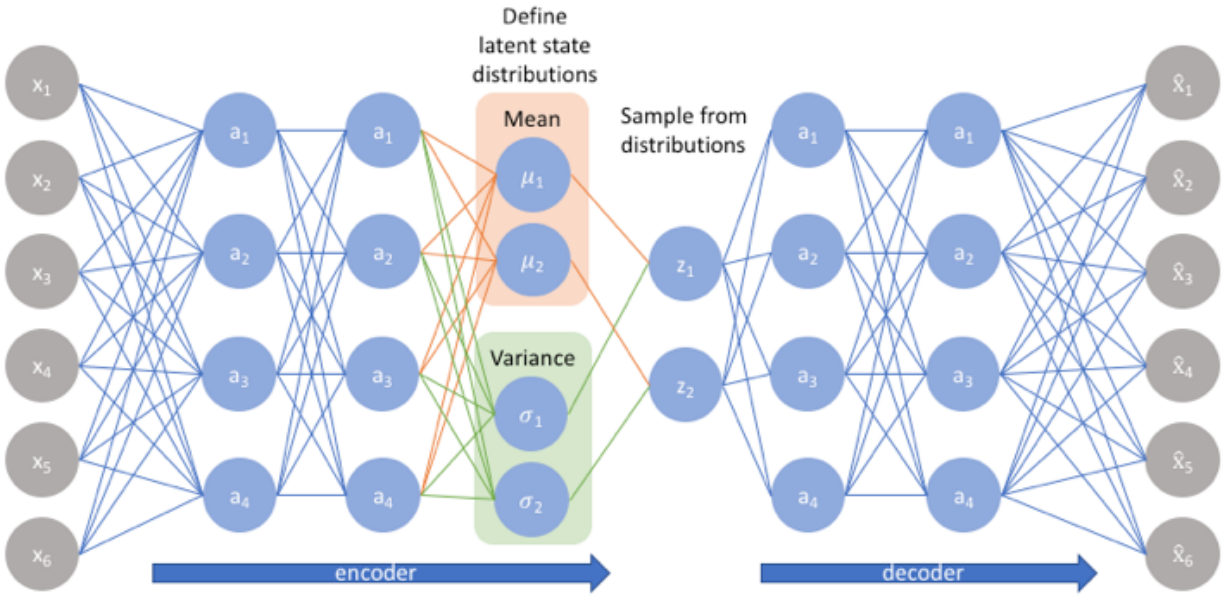
- The decoder ensures learning of latent factors that are able to describe the data
- The encoder is regularized to map to a latent space that follows the distribution of the prior (which is subject to our choice)



Variational auto-encoding with neural nets

The only two changes we need to make to our “standard” NN:

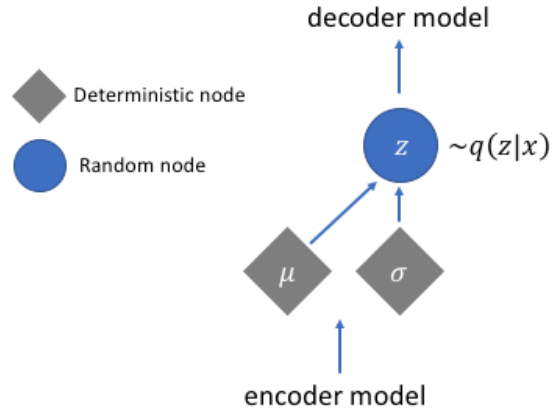
1. Change the loss to include the KLD in the Autoencoder
2. Modify encoder to output params of a distribution (usually Gaussian)



https://www.jeremyjordan.me/variational-autoencoders/

Variational auto-encoding: training

Unfortunately, it is not easy to run a sampling process with back-propagation to train such a network with a stochastic node (it's a bit more involved, see https://gregorygundersen.com/blog/2018/04/29/reparameterization/)

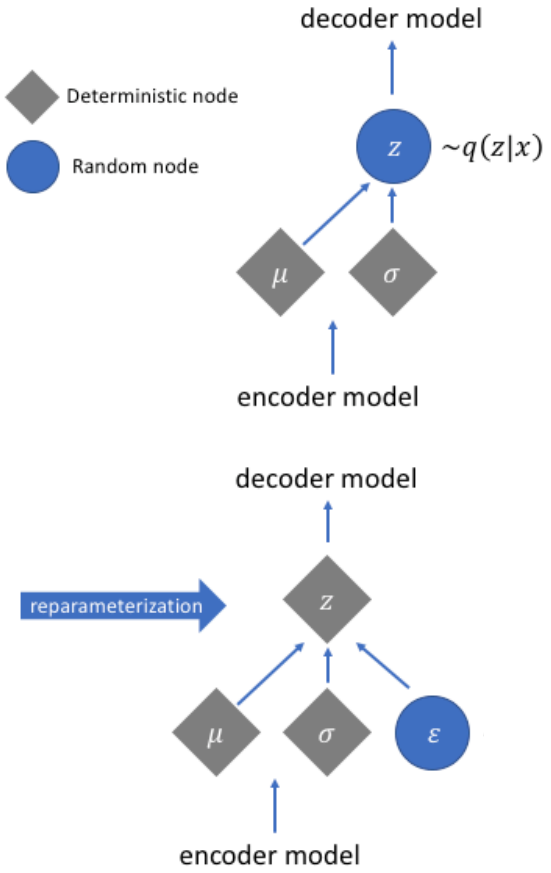


https://www.jeremyjordan.me/variational-autoencoders/

Variational auto-encoding: training

Unfortunately, it is not easy to run a sampling process with back-propagation to train such a network with a stochastic node (it's a bit more involved, see https://gregorygundersen.com/blog/2018/04/29/reparameterization/)

Instead, we can use what is called a "reparameterization trick" to treat z as if it was a deterministic variable, by sampling and reparameterizing noise

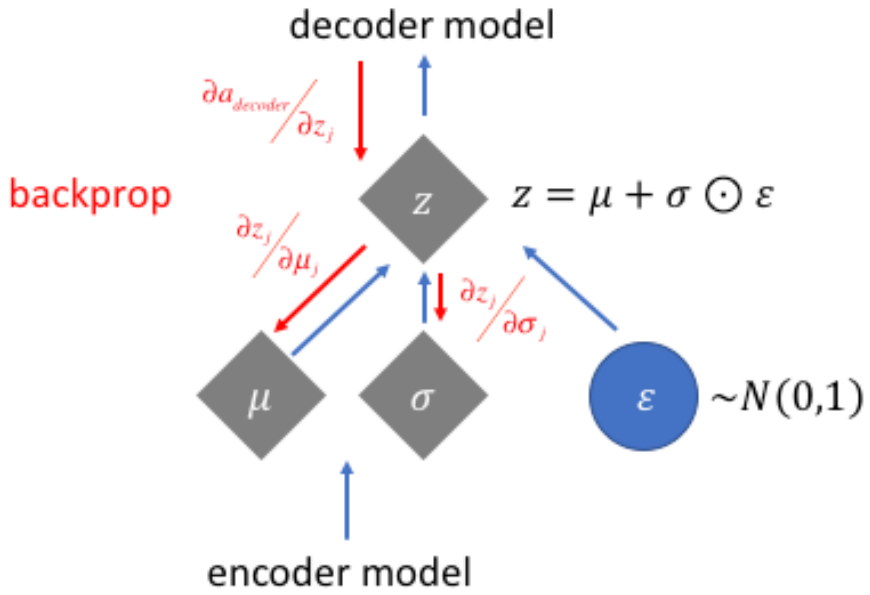


https://www.jeremyjordan.me/variational-autoencoders/

Variational auto-encoding: training

Example - approx. posterior as multivariate Gaussian: $\log q_\phi(z|x) = \log \mathcal{N}(z; \mu, \sigma I)$

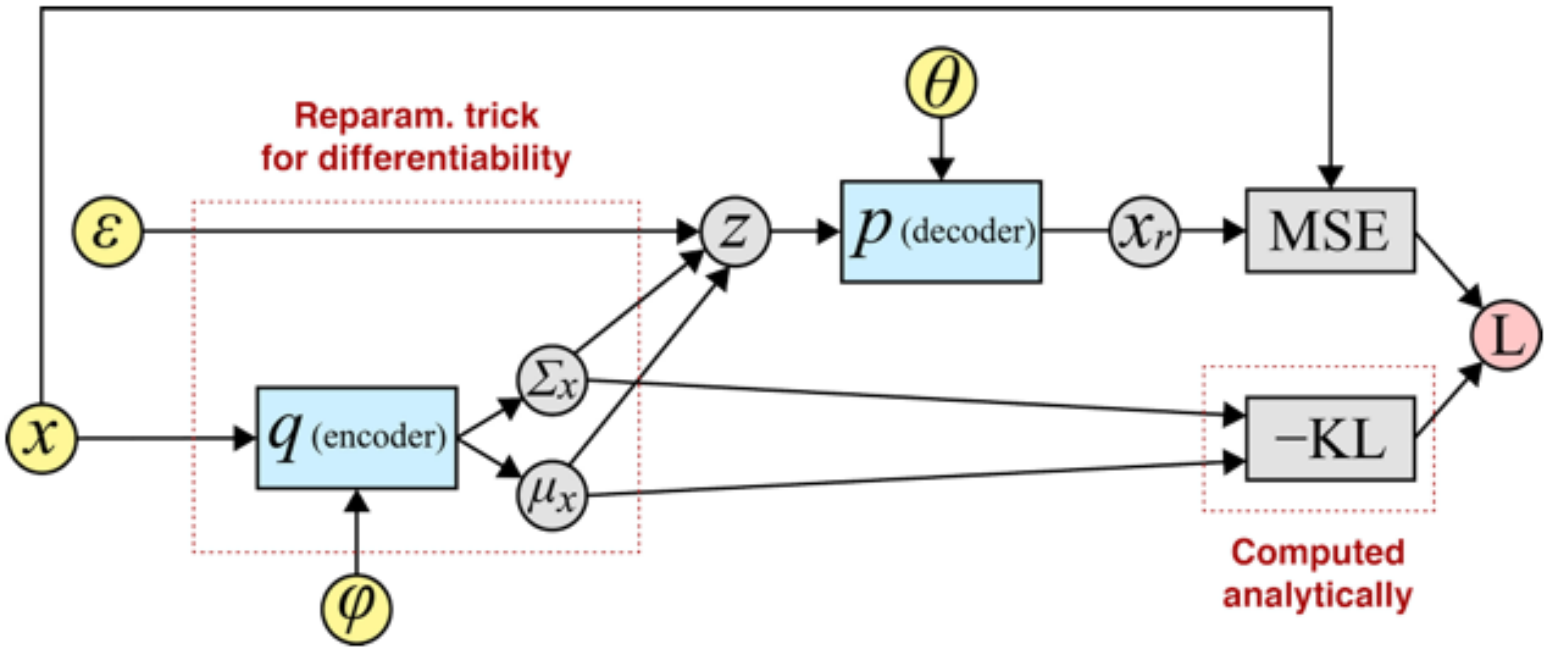
- Use **reparameterization trick** to generate samples from $q_\phi(z|x)$ with z as deterministic variable
- (Monte-Carlo) sample posterior $z \sim q_\phi(z|x)$ using $z = \mu + \sigma \circ \varepsilon$ with **samples** $\varepsilon \sim \mathcal{N}(0,I)$ (choice of prior)



https://www.jeremyjordan.me/variational-autoencoders/

Variational auto-encoding: training

Expressed as a neural network diagram, our computing looks like:



https://gregorygundersen.com/blog/2018/04/29/reparameterization/

Universität Bremen

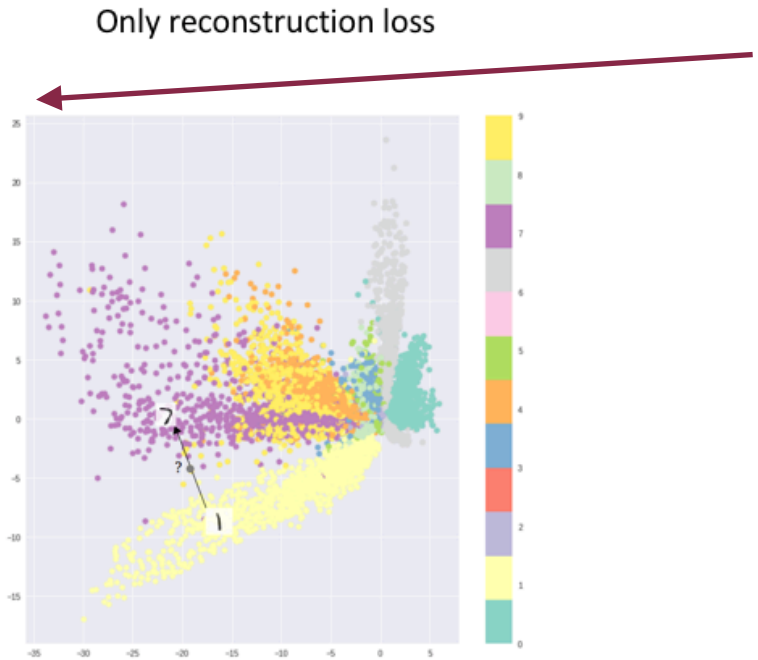
OWL ML

Question time

You may have gotten lost. Don’t worry, we’ll learn to understand what happens & the importance for continual learning. Does anyone already have any idea why what we just derived is helpful?

What have we gained with VAEs?

Why bother about the math? Let's compare AE & VAE representations



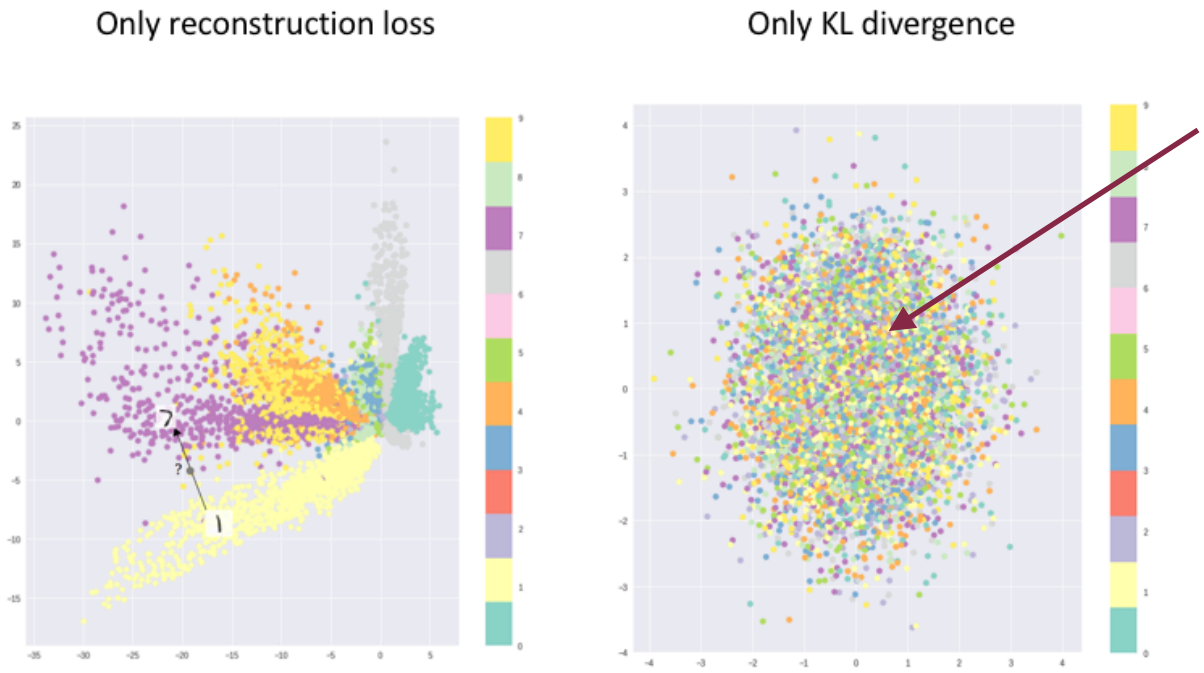
Hard to read in the diagram, but with only an autoencoder, the range of the latent space is rather arbitrary

(In this MNIST example -20 to +25 on the y-axis and -25 to 5 on the x-axis)

https://www.jeremyjordan.me/variational-autoencoders/ and https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

What have we gained with VAEs?

Why bother about the math? Let's compare AE & VAE representations



On the contrary: optimizing only KLD contains no data structure

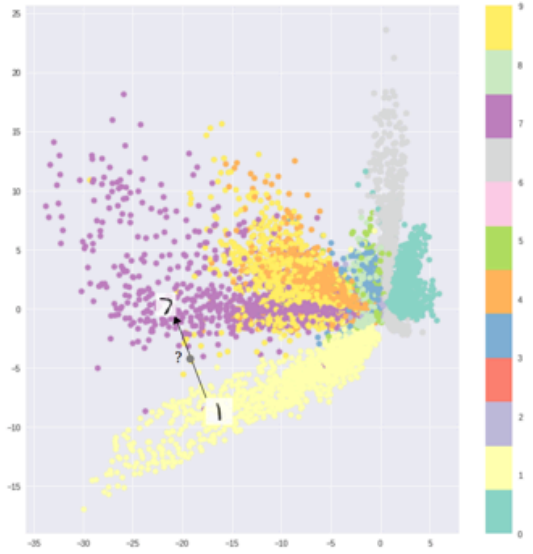
Anything can simply be collapsed to a Gaussian

https://www.jeremyjordan.me/variational-autoencoders/ and https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

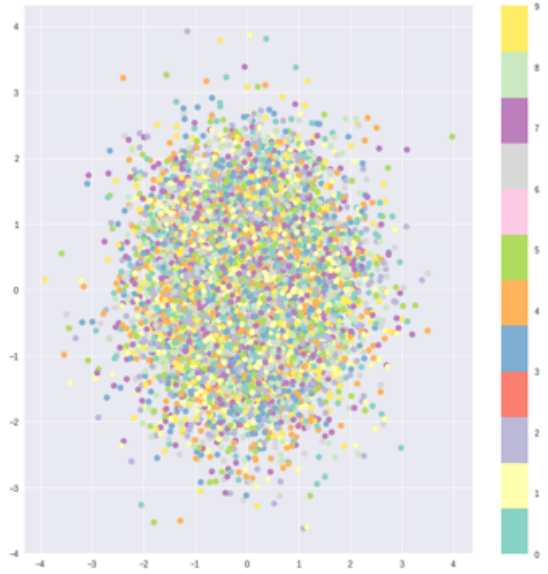
What have we gained with VAEs?

When we regularize the reconstruction with the KLD term, we retain data structure AND ensure the latent space follows a Gaussian distribution

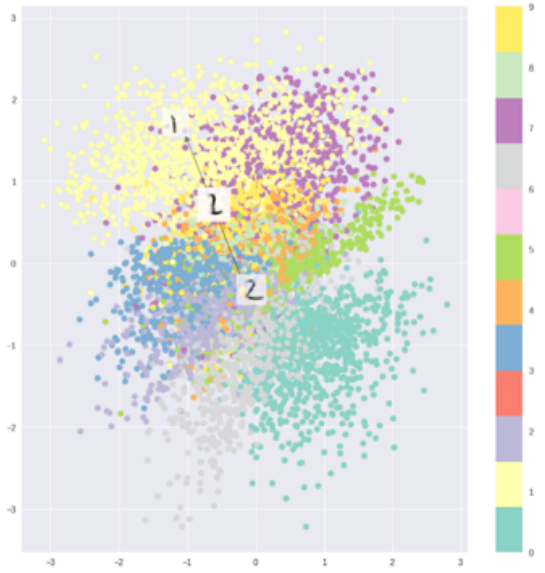
Only reconstruction loss



Only KL divergence



Combination



https://www.jeremyjordan.me/variational-autoencoders/ and https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Universität Bremen

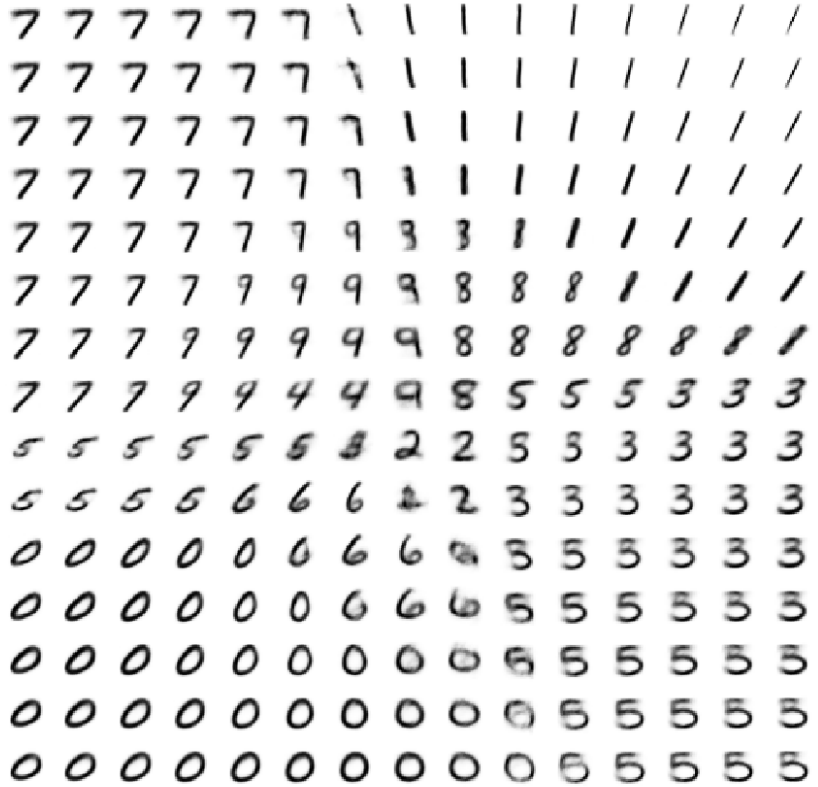
OWL ML

Question time

Once more: do you have an idea how this is helpful for continual learning? Recall we started from generative replay, but also talked about regularization to avoid forgetting beforehand

Benefits of VI & VAEs for continual learning

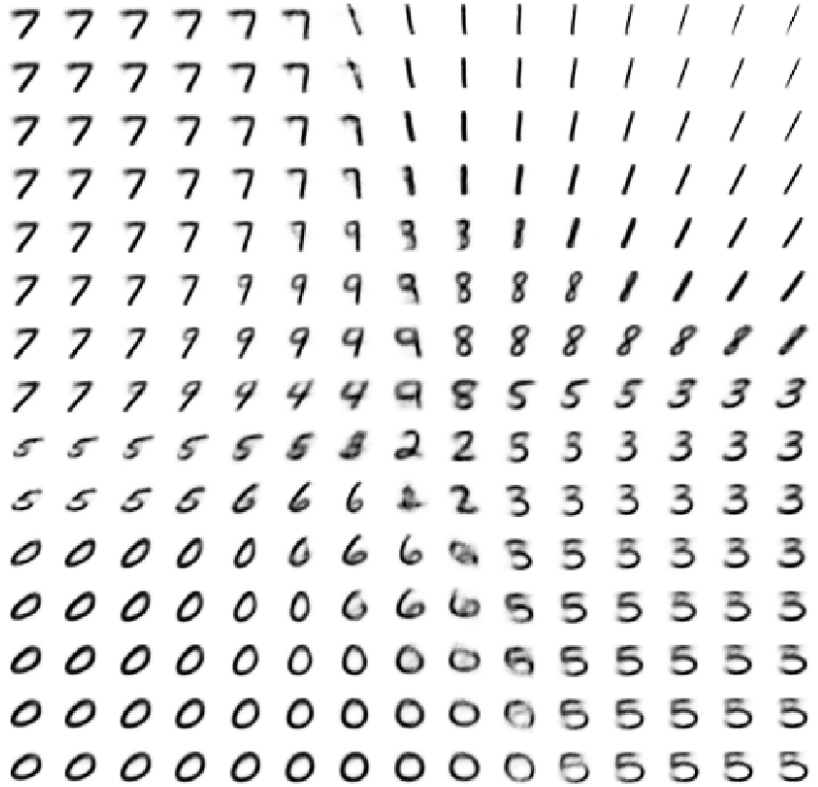
- We can sample from a trained model: $z \sim p(z)$, here $\mathcal{N}(0,I)$, and generate (decode) x



https://www.compthree.com/blog/autoencoder/

Benefits of VI & VAEs for continual learning

- We can sample from a trained model: $z \sim p(z)$, here $\mathcal{N}(0,I)$, and generate (decode) x
- We also have the approximate posterior $q(z \mid x)$ that we could regularize in continual learning



https://www.compthree.com/blog/autoencoder/

Variational Continual Learning (VCL)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \,|\, z) \right] - KL \left[q_\phi(z \,|\, x) \,||\, p_\theta(z) \right]$$

The "likelihood focused" perspective: useful for generative/ pseudo rehearsal

- Generate old tasks' data and concatenate or interleave it with new task data
- Primarily driven by the likelihood

See Nguyen et al, "Variational Continual Learning" ICLR 2018 & follow-ups like Farquhar et al "A Unifying Bayesian View of Continual Learning", NeurIPS workshops 2018

Variational Continual Learning (VCL)

$$\log p_\theta(x) \geq \mathscr{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \,|\, z) \right] - KL \left[q_\phi(z \,|\, x) \,||\, p_\theta(z) \right]$$

The "likelihood focused" perspective: useful for generative/ pseudo rehearsal

- Generate old tasks' data and concatenate or interleave it with new task data
- Primarily driven by the likelihood

The "prior focused" perspective: useful for regularization/distillation

- Only use new task data
- Use the posterior of an old task as the new task's prior $KL \left[q_t(z) \,||\, q_{t-1}(z) \right]$

See Nguyen et al, "Variational Continual Learning" ICLR 2018 & follow-ups like Farquhar et al "A Unifying Bayesian View of Continual Learning", NeurIPS workshops 2018

Variational Continual Learning (VCL)

$$\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \,|\, z) \right] - KL \left[q_\phi(z \,|\, x) \,||\, p_\theta(z) \right]$$

The "likelihood focused" perspective: useful for generative/ pseudo rehearsal

- Generate old tasks' data and concatenate or interleave it with new task data
- Primarily driven by the likelihood

VCL, or at least the "likelihood" perspective based on generative replay in VAEs, will be the second half of your fourth tutorial

- Only use new task data
- Use the posterior of an old task as the new task's prior $KL \left[q_t(z) \,||\, q_{t-1}(z) \right]$

See Nguyen et al, "Variational Continual Learning" ICLR 2018 & follow-ups like Farquhar et al "A Unifying Bayesian View of Continual Learning", NeurIPS workshops 2018

Variational Continual Learning (VCL)

Through a combined perspective VCL improves upon e.g. pure EWC

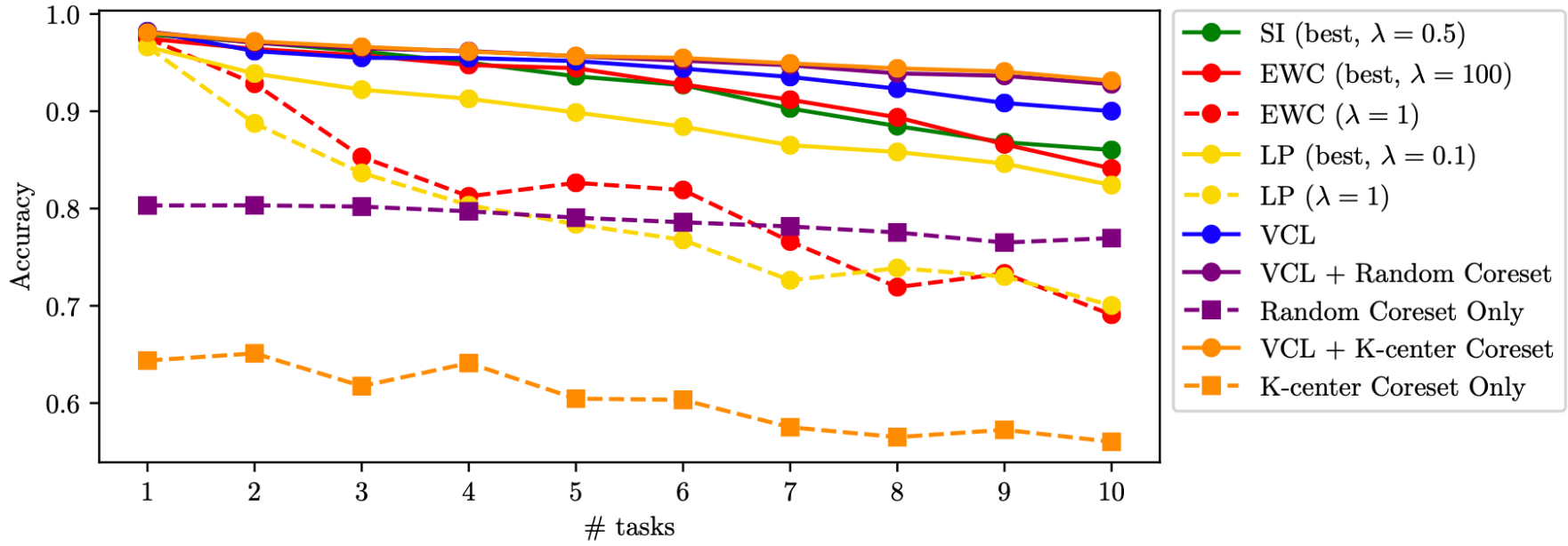


Figure 2: Average test set accuracy on all observed tasks in the Permuted MNIST experiment.

Nguyen et al, “Variational Continual Learning” ICLR 2018

Variational Continual Learning (VCL)

And naturally, data rehearsal (core sets) are always complementary

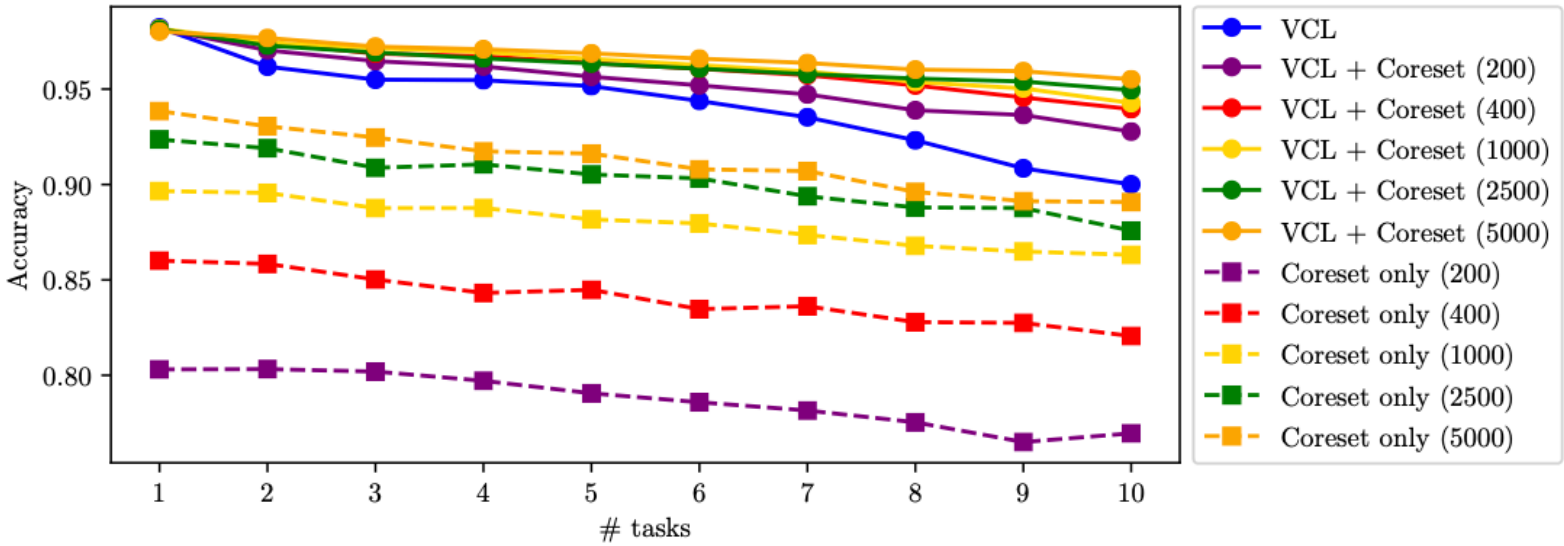


Figure 3: Comparison of the effect of coreset sizes in the Permuted MNIST experiment.

Nguyen et al, “Variational Continual Learning” ICLR 2018

A short summary: what we've learned so far

- Data subsets and exemplars are often used for rehearsal
- Core sets formalize the question how to pick an adequate subset

A short summary: what we've learned so far

- Data subsets and exemplars are often used for rehearsal
- Core sets formalize the question how to pick an adequate subset
- Biological evidence suggest that "raw rehearsal" may not be fully realistic. Even in ML practice there are various privacy concerns
- Complementary learning systems theory suggest the use of two models, where one is of generative nature for "pseudo-rehearsal"

A short summary: what we've learned so far

- Data subsets and exemplars are often used for rehearsal
- Core sets formalize the question how to pick an adequate subset
- Biological evidence suggest that "raw rehearsal" may not be fully realistic. Even in ML practice there are various privacy concerns
- Complementary learning systems theory suggest the use of two models, where one is of generative nature for "pseudo-rehearsal"
- Generative ML models allow us to "replay" by sampling from the already observed distribution & continuing to train to avoid forgetting
- We can in principle use any generative approach (GAN, Normalizing Flow, Diffusion etc.), but variational approaches also give us a nice formalism to combine with regularization perspectives

Universität Bremen

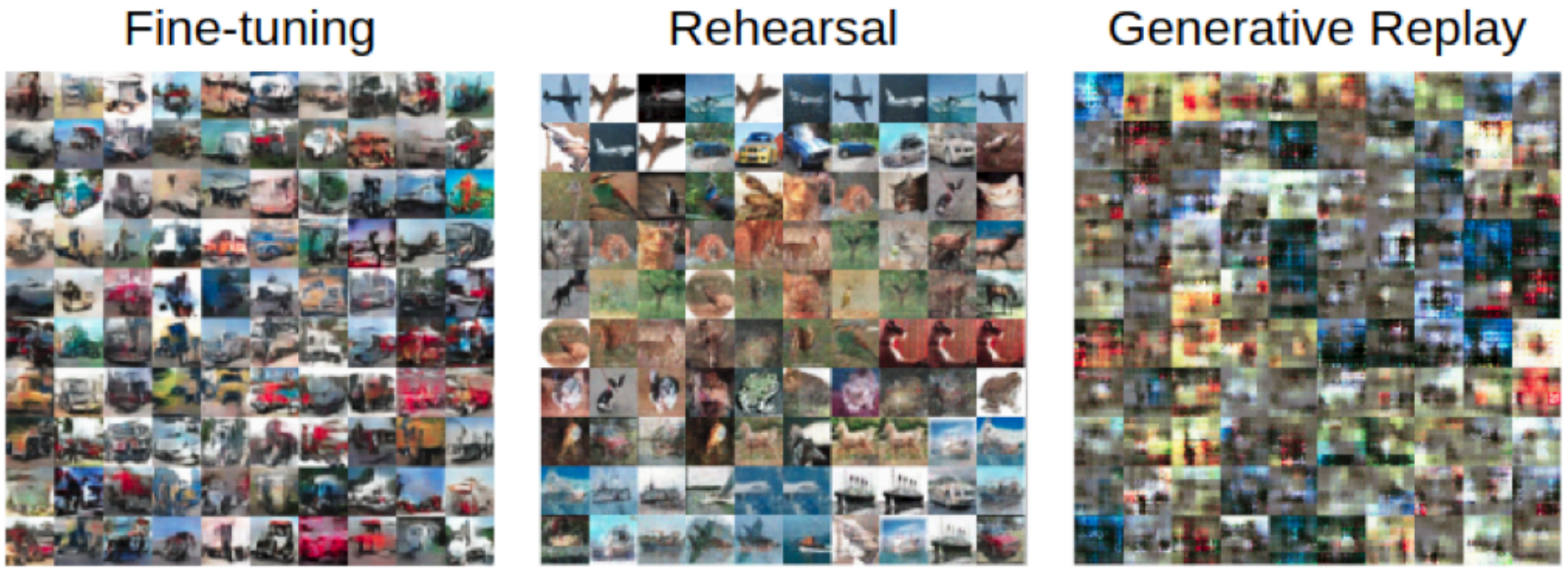
OWL ML

Question time

Have we "solved" continual learning? Or at least, have we fixed the phenomenon of forgetting?

Why we may need to combine perspectives

Fine-tuning forgets, regularization has the stability-plasticity trade-off, data rehearsal can overfit, generative models need to be powerful



Lesort et al, “Generative Models from the perspective of Continual Learning”, IJCNN 2019

Why we may need to combine perspectives

Generative models also suffer from forgetting. In particular, errors “snowball” because we need to train on prior generated content

Lesort et al, “Generative Models from the perspective of Continual Learning”, IJCNN 2019

Universität Bremen

Why we may need to combine perspectives

And finally, as before, we have many more hyper-parameters. What we accept or expect may depend very highly on application context

Why we may need to combine perspectives

And finally, as before, we have many more hyper-parameters. What we accept or expect may depend very highly on application context

- Size of the memory buffer?
- Constant memory budget with recursion or growing memory?
- Amount of generated examples?
- Weight of interleaving sub sets/generated examples with new ones?
- When combining param regularization, functional regularization, data rehearsal, and generative replay, how to weigh individual terms?

- … quite a lot of other considerations

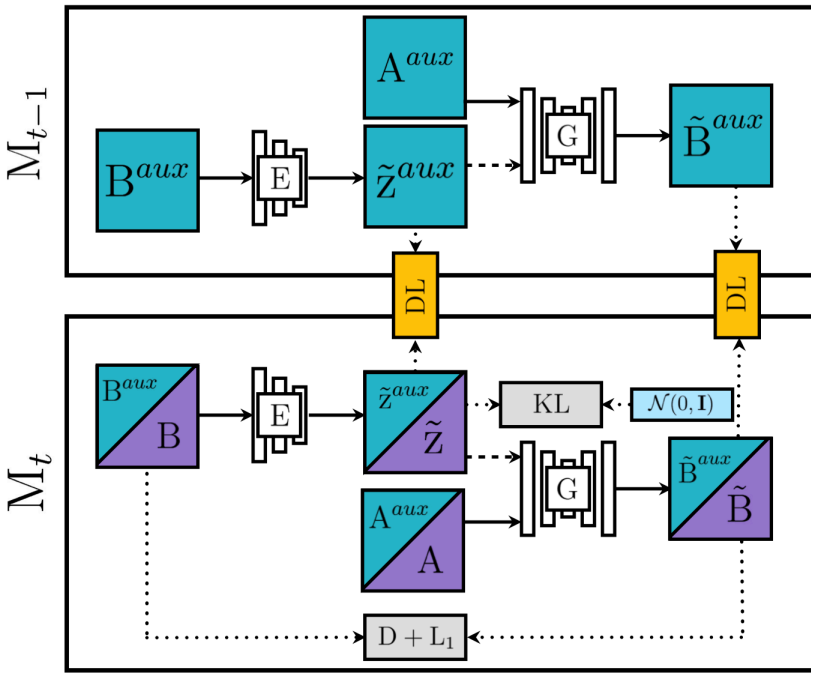
Why we may need to combine perspectives

There are MANY MANY continual ML works that combine ideas to address forgetting. Some are more theoretically grounded, like VCL, but some may work “better” in practice, despite being very ad-hoc.

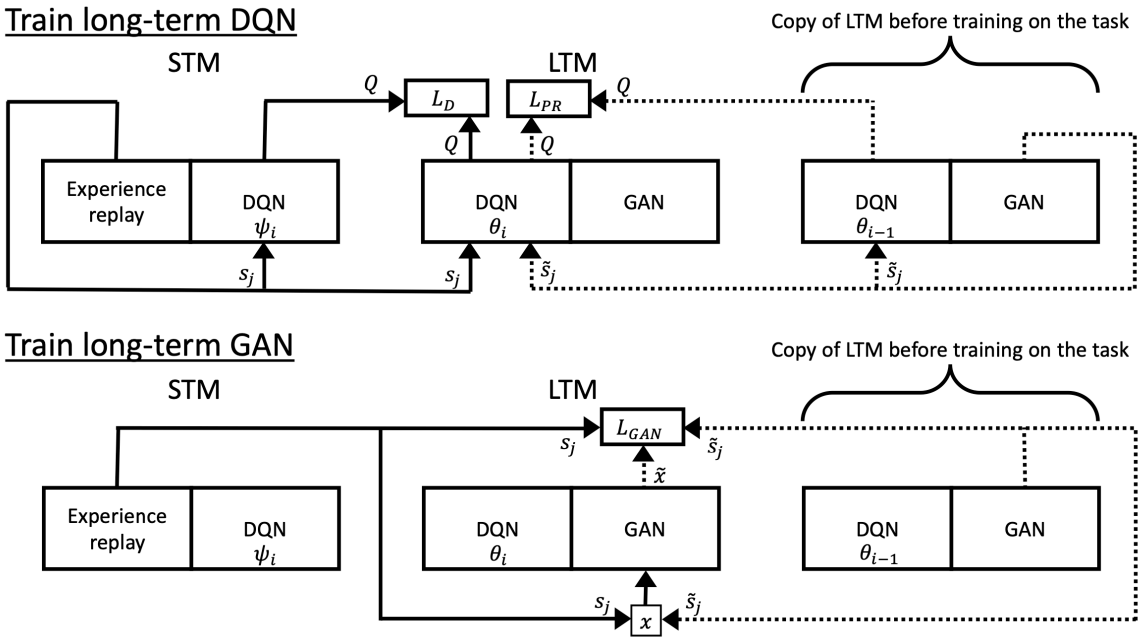
It will be impossible to list all examples, especially considering applications across vision, language, reinforcement learning, etc.

Here are two examples, to simply get an intuition for how “easy” it is to arbitrarily combine what we have learned so far to deal with catastrophic forgetting in a much better way than simply accept it

Why we may need to combine perspectives



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



Atkinson, “Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting”, Neurocomputing 428:7, 2021

Question time

We learned about regularization & rehearsal. When you recall transfer learning, there however was a third option of transfer by adding “experts”. VCL can tackle all three dimensions: do you already have any ideas for the last one?

Universität Bremen

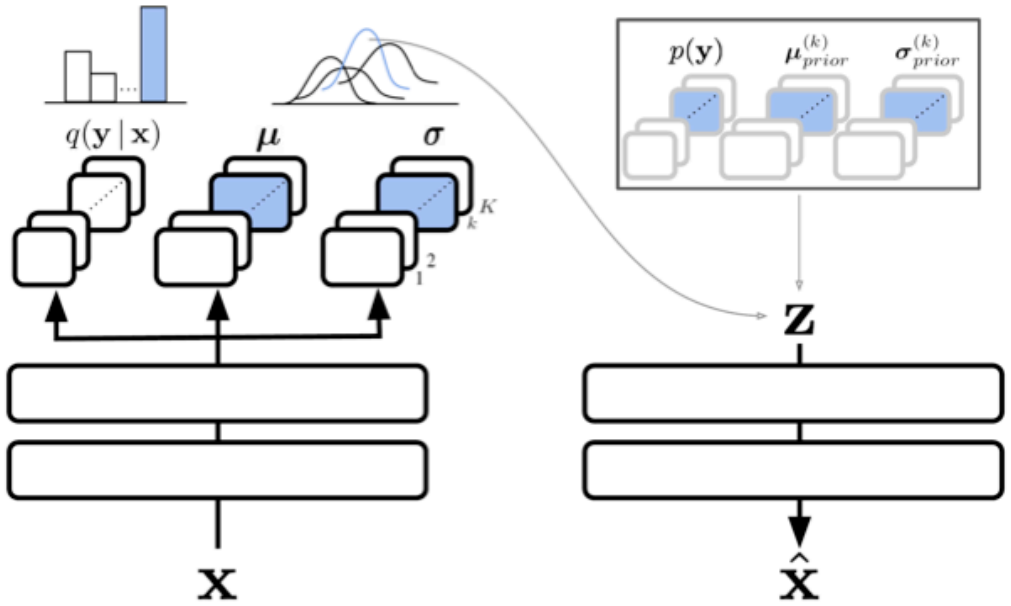
From VCL to CURL: Adding task specific architectural components

In VAEs, one way in which we can add a task “expert” is by controlling where tasks are mapped to in latent space.

From VCL to CURL: Adding task specific architectural components

In VAEs, one way in which we can add a task “expert” is by controlling where tasks are mapped to in latent space.

In other words, we can assign & optimize an entirely different prior per task. That is, we create a mixture of tasks



Rao et al, “Continual Unsupervised Representation Learning”, NeurIPS 2019

Universität Bremen

From VCL to CURL: Adding task specific architectural components

The caveat is that we will need to infer which task-specific Gaussian to use after training.

Rao et al, “Continual Unsupervised Representation Learning”, NeurIPS 2019

From VCL to CURL: Adding task specific architectural components

The caveat is that we will need to infer which task-specific Gaussian to use after training.

We factorize the joint probability: $p(x, y, z) = p(y)p(z \mid y)p(x \mid z)$
where y indicates the current task & we assume a uniform task prior (without having external knowledge about tasks)

Rao et al, “Continual Unsupervised Representation Learning”, NeurIPS 2019

From VCL to CURL: Adding task specific architectural components

The caveat is that we will need to infer which task-specific Gaussian to use after training.

We factorize the joint probability: $p(x, y, z) = p(y)p(z \mid y)p(x \mid z)$
where y indicates the current task & we assume a uniform task prior (without having external knowledge about tasks)

The approx. variational posterior is $p(y, z \mid x) = q(y \mid x)q(z \mid x, z)$

Rao et al, “Continual Unsupervised Representation Learning”, NeurIPS 2019

From VCL to CURL: Adding task specific architectural components

Following the derivation from before, but skipping it for brevity, we gain another evidence lower bound to optimize as follows:

$$\log p(x) \geq \mathscr{L} = \mathbb{E}_{q(y,z|x)}[\log p(x,y,z) - \log q(y,z|x)]$$
$$= \mathbb{E}_{q(y|x)q(z|x,y)}[\log p(x|z)] - \mathbb{E}_{q(y|x)}[KL(q(z|x,y)||p(z|y))]$$
$$-KL(q(zy|x)||p(y))$$

The last KL term can be thought of as a categorical regularizer

Rao et al, “Continual Unsupervised Representation Learning”, NeurIPS 2019

From VCL to CURL: Adding task specific architectural components

Such a framing allows us to think about a key question that we will now encounter when thinking about addressing catastrophic forgetting from an “architectural perspective”.

From VCL to CURL: Adding task specific architectural components

Such a framing allows us to think about a key question that we will now encounter when thinking about addressing catastrophic forgetting from an “architectural perspective”.

Option 1: Implicit: we can assume a very large over-parametrized model and try to create specific sub-modules (note the encoder is likely heavily over-parametrized and very powerful in CURL/VCL)

From VCL to CURL: Adding task specific architectural components

Such a framing allows us to think about a key question that we will now encounter when thinking about addressing catastrophic forgetting from an “architectural perspective”.

Option 1: Implicit: we can assume a very large over-parametrized model and try to create specific sub-modules (note the encoder is likely heavily over-parametrized and very powerful in CURL/VCL)

Option 2: we can add actual parameters and capacity over time (in CURL, we “dynamically expand” by adding mixture components)

Universität Bremen

How is forgetting linked to the architecture? A short intuitive and motivational recap

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

Robert French, “Using Semi-Distributed Representations to Overcome Catastrophic Forgetting in Connectionist Networks”, AAAI 1993

Universität Bremen

How is forgetting linked to the architecture? A short intuitive and motivational recap

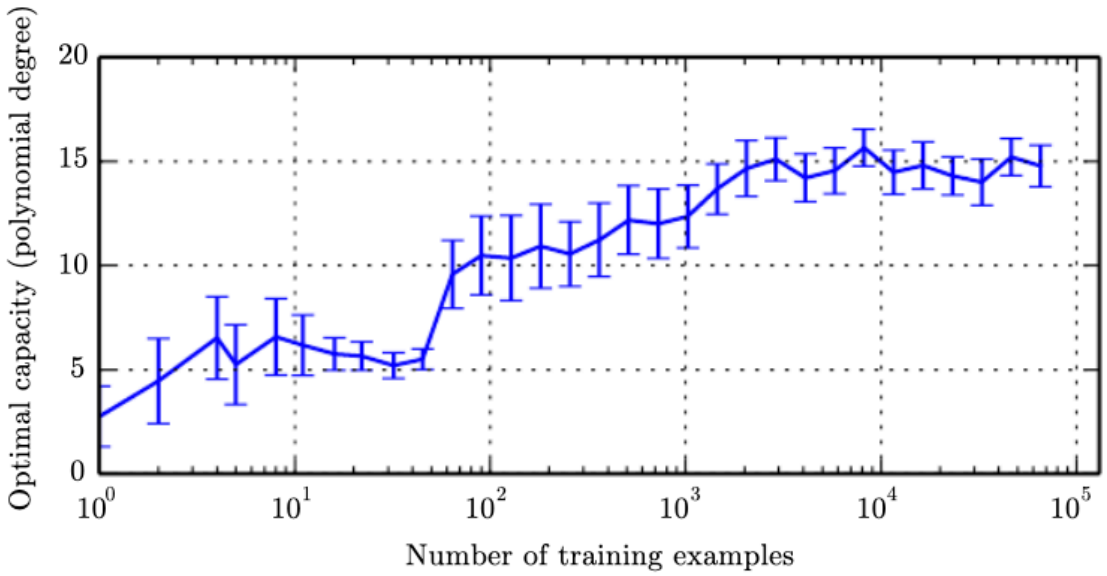
“Catastrophic forgetting is a direct consequence of the overlap of distributed representations and can be reduced by reducing this overlap.”

Robert French, “Using Semi-Distributed Representations to Overcome Catastrophic Forgetting in Connectionist Networks”, AAAI 1993

”Very local representations will not exhibit catastrophic forgetting because there is little interaction among representations. However, a look-up table lacks the all-important ability to generalize.The moral of the story is that you can’t have it both ways.”

How is forgetting linked to the architecture? A short intuitive and motivational recap

And also recall that we need to find a suitable model capacity, especially if the model is “too small” and we will be bottlenecked by lack of plasticity



Deep Learning, Goodfellow, Bengio, Courville, MIT Press 2016