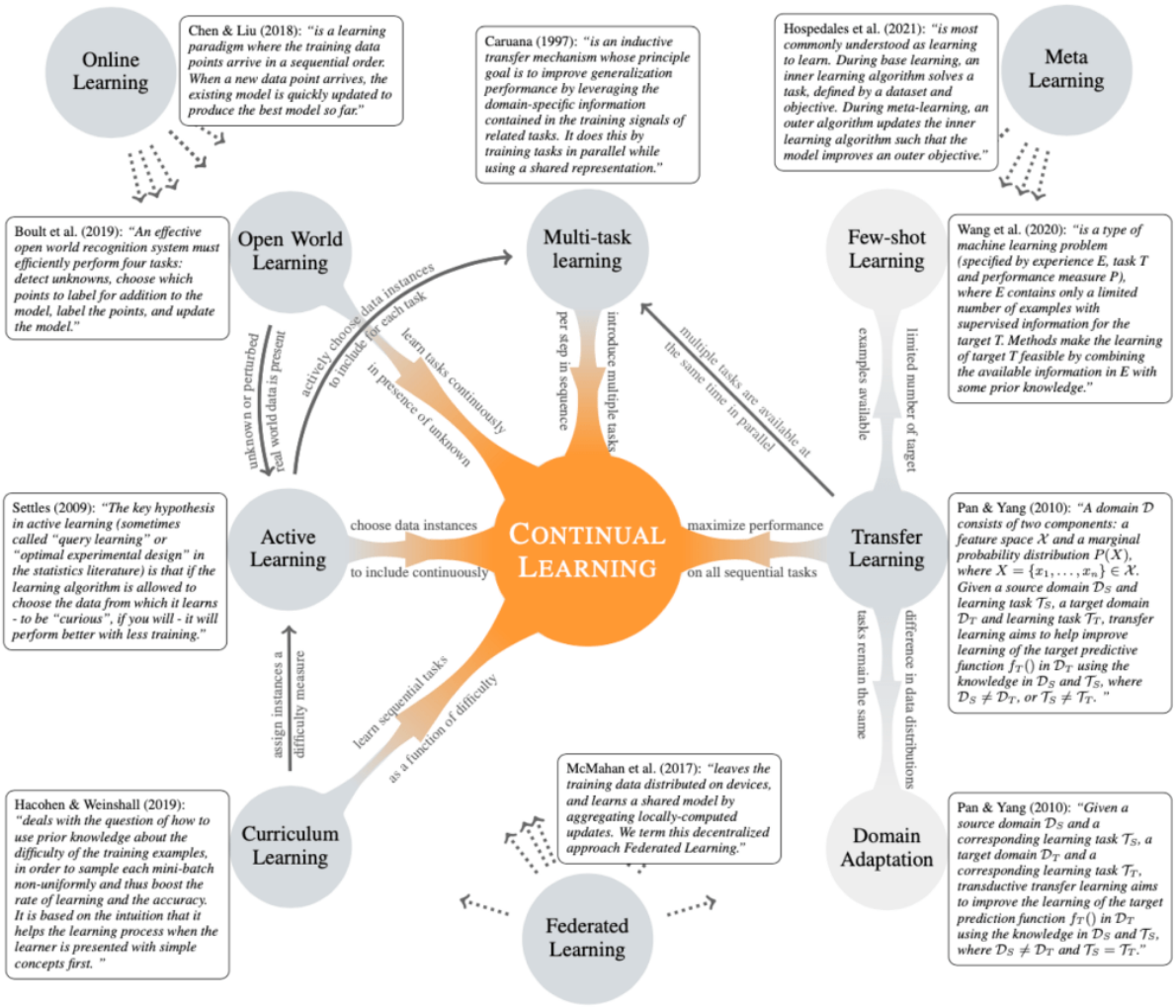
Continual Machine Learning

1. Introduction and Motivation

Machine learning studies the design of models and training algorithms in order to learn how to solve tasks from data. Whereas historically machine learning has concentrated primarily on static predefined training datasets and respective test scenarios, recent advances also take into account the fact that the world is constantly evolving. In this course, we will go beyond the train-validate-test phase and explore modern approaches to machines that can learn continually. In addition to a comprehensive overview of the breath of factors to consider in continual learning, the course will delve into techniques that span mitigation of forgetting across multiple tasks, selection of new data in continuous training, dynamic model architectures, and robustness with respect to unexpected data inputs.



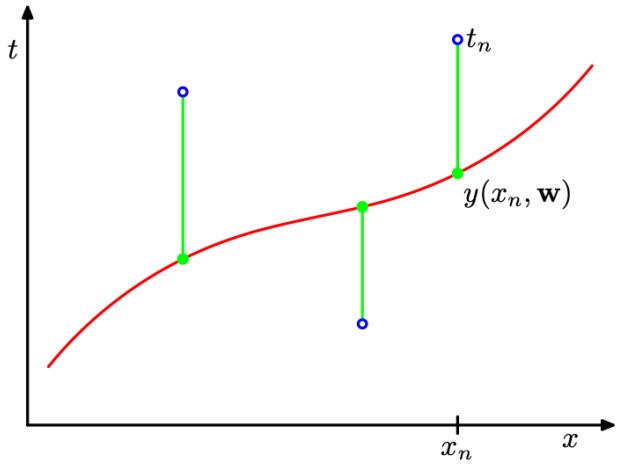
1.1. What is Machine Learning

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The result of running the machine learning algorithm can be expressed as a function. The precise form of the function is determined during the *training phase*, also known as the learning phase, based on the training data. Once the model is trained it can then determine the identity of new images, which are said to comprise a *test set*. The ability to categorize correctly new examples that differ from those used for training us known as *generalization*.

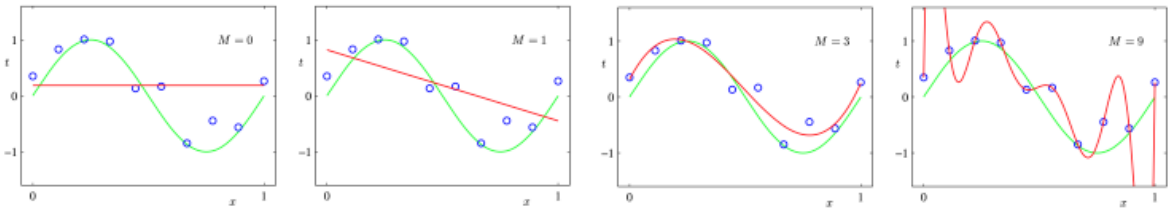
Error/Loss and Learning

The error function corresponds to (one half of) the sum of the squares of the displacements (shown by the vertical green bars) of each data point from the function $y(x, \boldsymbol{w})$. Here is an example for polynomial curve fitting.

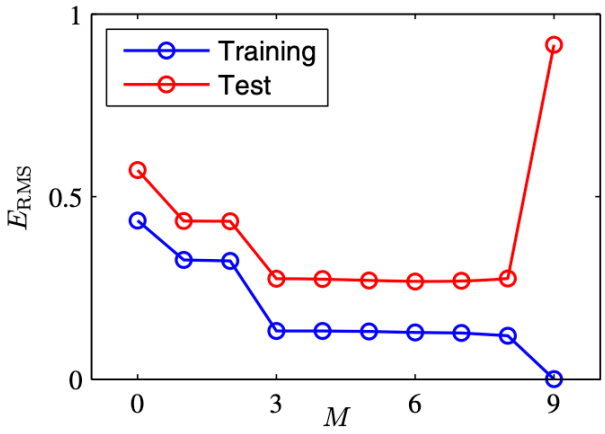


Under and Overfitting

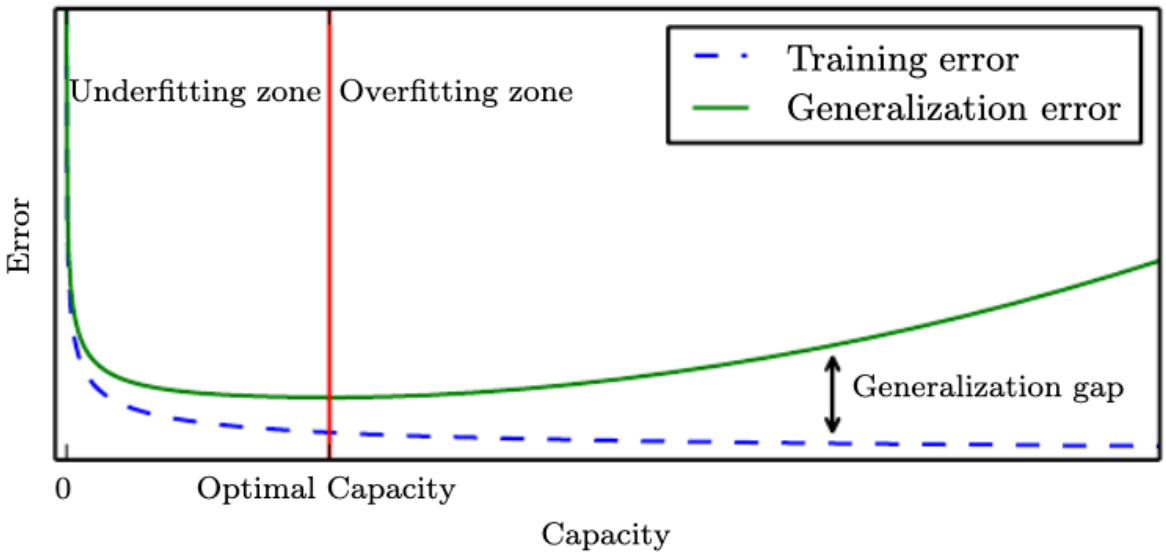
Looking at the previous example polynomials depending on the order M can lead to overfitting or underfitting as shown in the red curves:



Intuitively, what is happening is that the more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values. Here is an example for where graphs of the root-mean-square error are evaluated on the training set and on an independent test set for various values of M: At the beginning we underfit and at the end we overfit.



This picture is still very much the same in the “deep learning era”

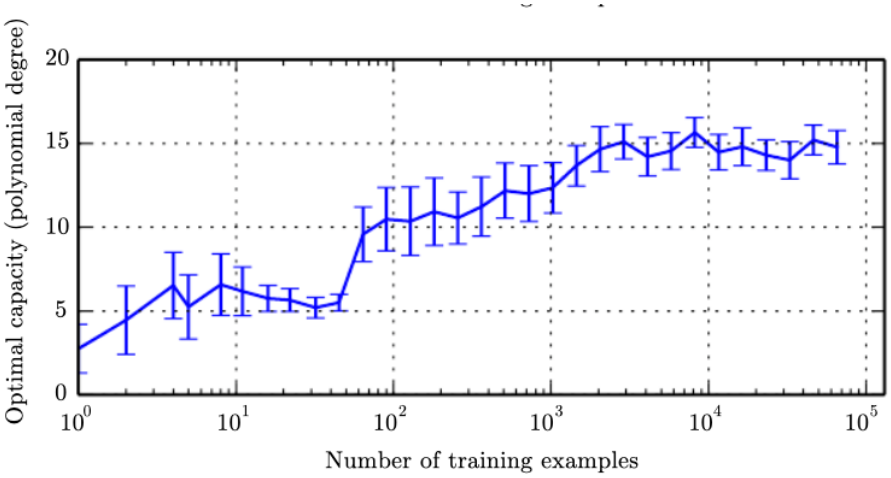


1.2. Goals of (static) ML workflow

Of course, when we use a machine learning algorithm, we do not fix the parameters ahead of time, then sample both datasets. We sample the training set, then use it to choose the parameters to reduce training set error, then sample the test set. The factors determining how well a machine learning algorithm will perform are its ability to:

a) Make the training error small.
b) Make the gap between training and test error small

So, is ML all about finding a large dataset and a right capacity model? Yes!

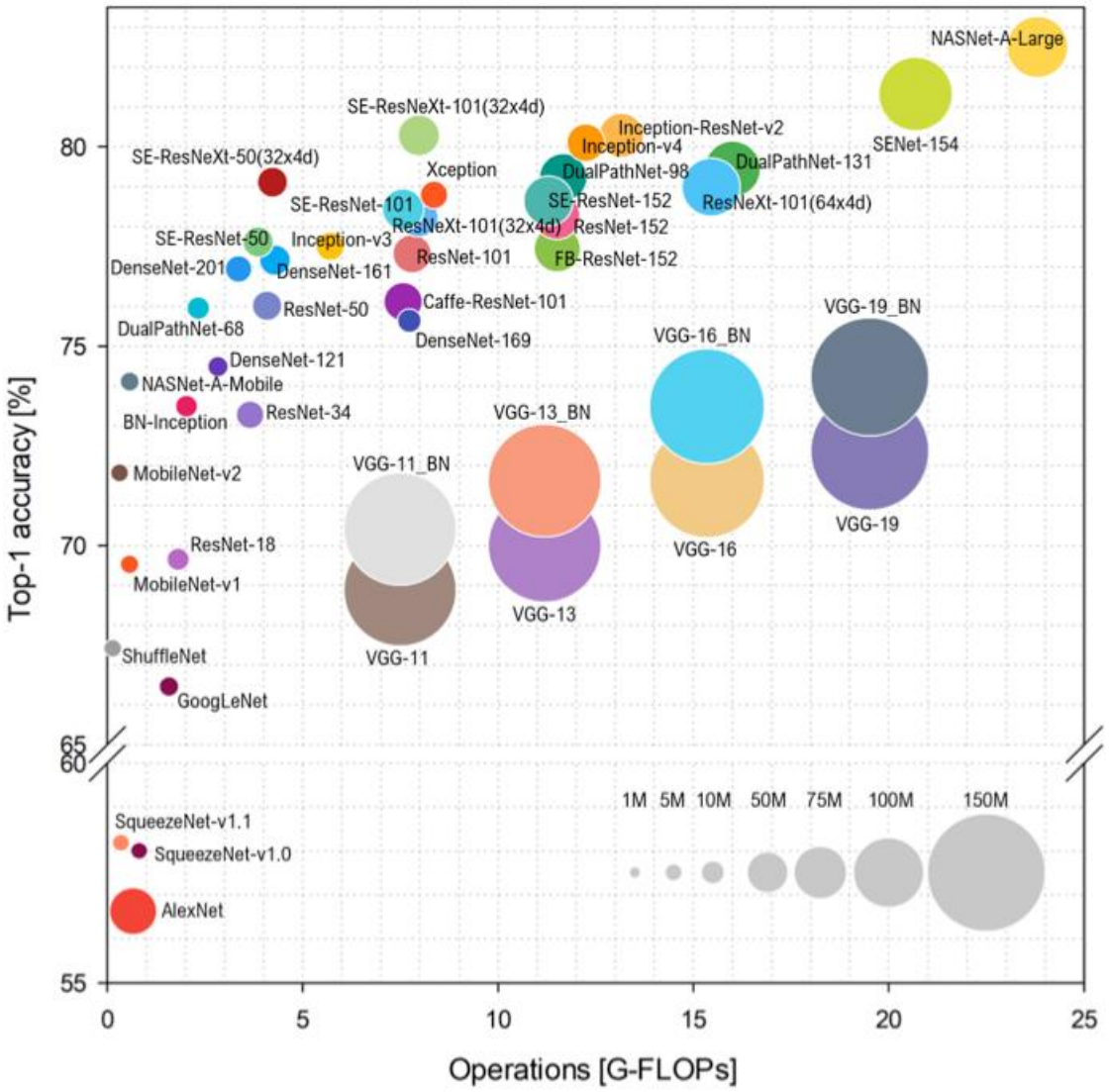


1.3. Acquiring static datasets

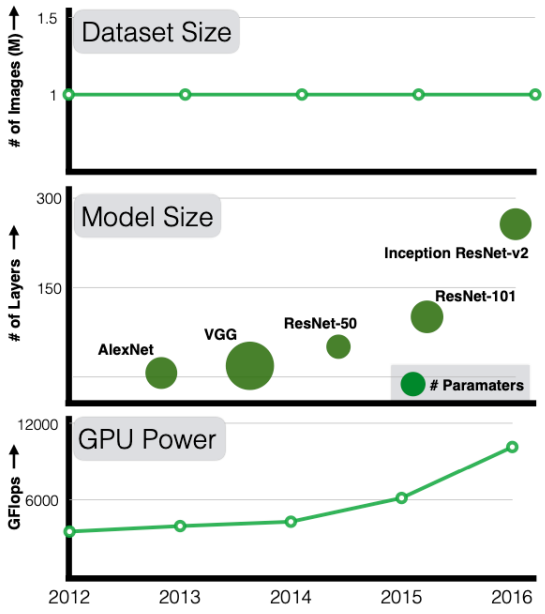
Either small scale, but (some) controlled acquisition parameters or large scale and diversity which is a big focus of modern datasets. Trying to ensure reasonable train, validation, test splits through complex collection processes for large scale static datasets is important.

1.4. Static Models and the workflow

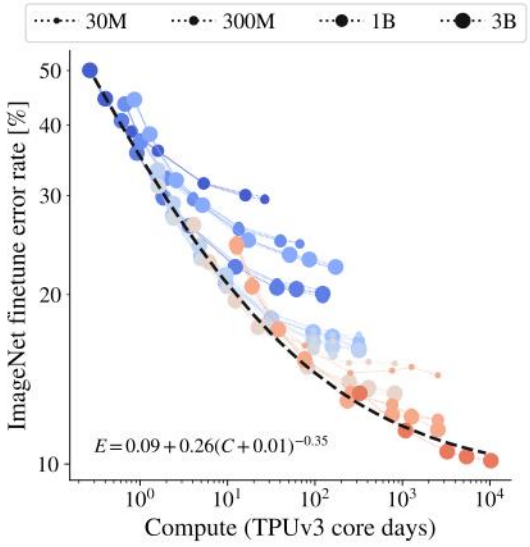
Should our primary goal be the solution to such benchmarks (meaning for large scale data)? A very big emphasis has then been on “solving” such benchmarks. ImageNet is a prime example, where models and compute got bigger and more accurate over time.



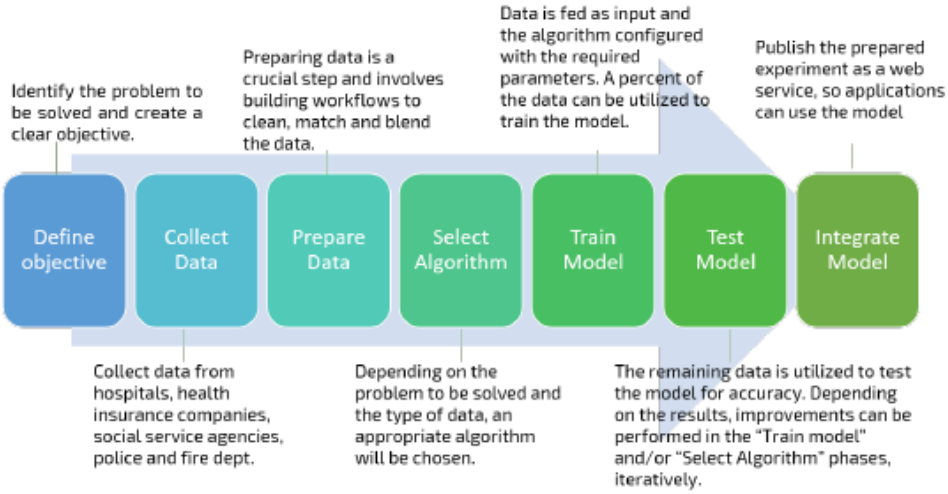
At the same time, it's often "either" models or data. For example, ImageNet has remained largely static over time (excluding some concerns over fair representation).



Or conversely, a model is picked (here a transformer) and datasets are extended

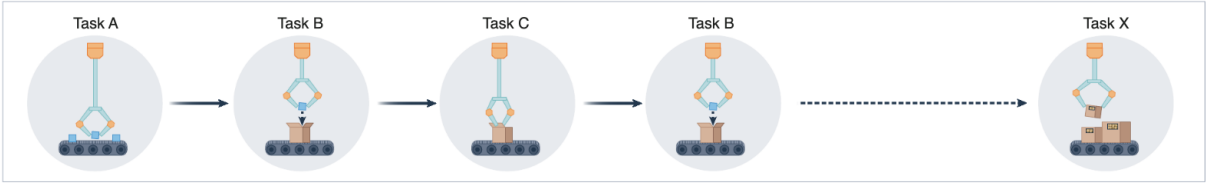


Overall, this is the static ML workflow



1.5. Continual Learning

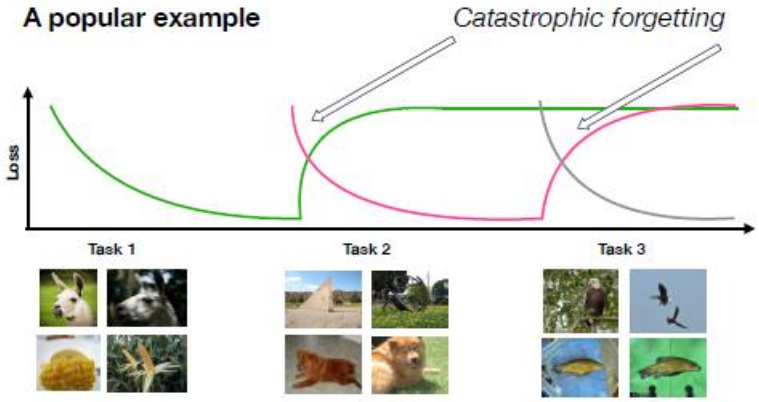
But what if we want to continue learning tasks? Or add more categories? Can we just iterate over our static Machine Learning workflow? No!



Humans seem to actively benefit from temporal correlation during "training". Machine learning typically shuffles data and performs poorly when data is ordered! Big difference! Why do we need an entire lecture? Well, there are a lot of challenges in continual machine learning!

Challenge Forgetting

In Forgetting the key assumption is that there is no access to/revisiting of prior “task” data!

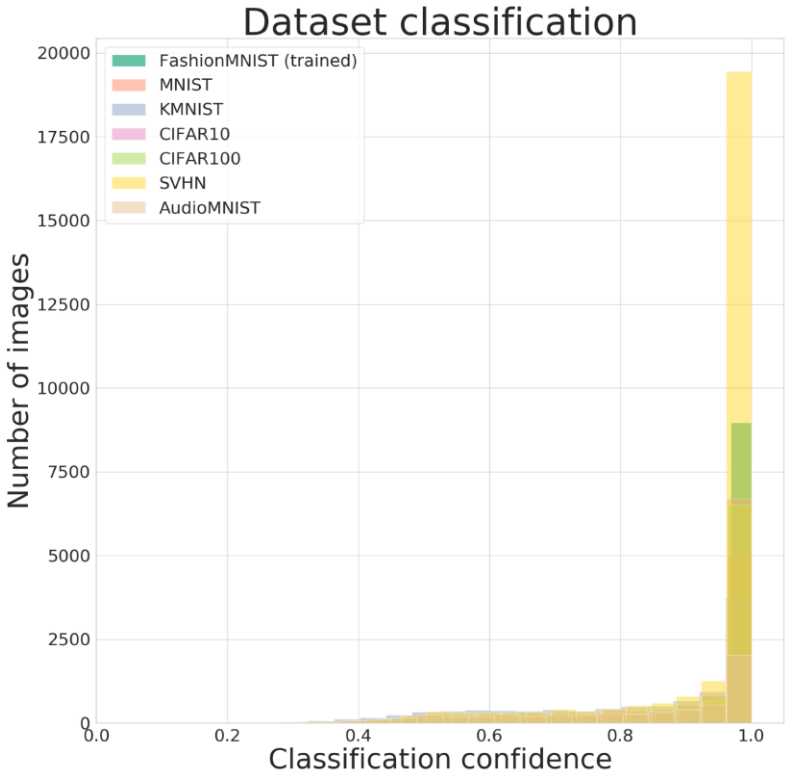


We will solve this problem with the continual learning paradigms later!

Challenge the world is open

Most ML models are overconfident! They don’t know when they don’t know. A quantitative example:

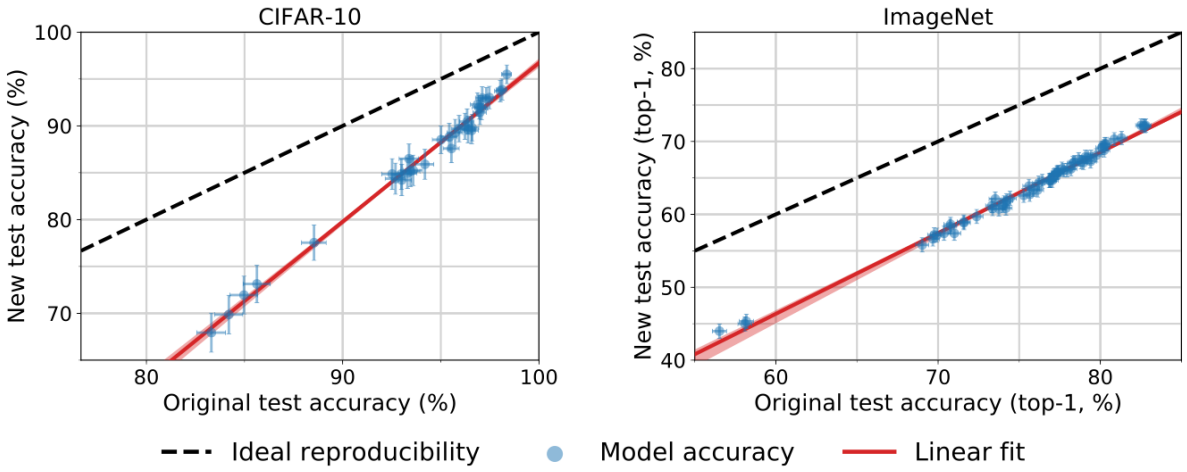
1. Train a neural network classifier on a dataset (here Fashion items)
2. Log predictions for arbitrary other datasets
3. Observe that majority of misclassifications happen with large output “probability”



But this example is unrealistic! What do you think will happen if we collect a second test set (following the same procedure) and evaluate?

Challenge distribution shifts

Natural data distributions are complex and can easily shift! Performance loss even happens if we recollect another test set with the same instructions a second time!



Challenge select and add data

What if we want to add data over time?

- How to pick data?
- Does the data belong to the task?
- How similar is the data?
- How optimize accumulated error (is this even what we want?)

We will solve this problem with active learning later!

Challenge concept difficulty

Example: Ranking language model trained with vs without curriculum on Wikipedia

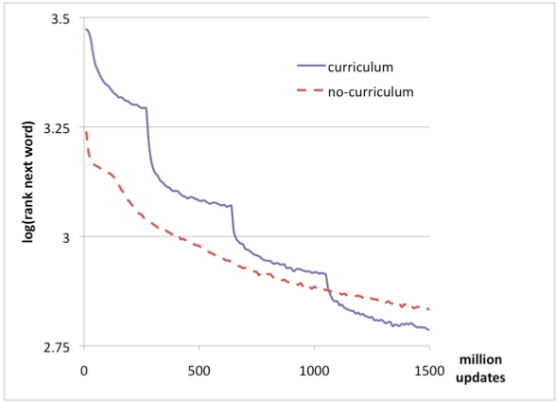
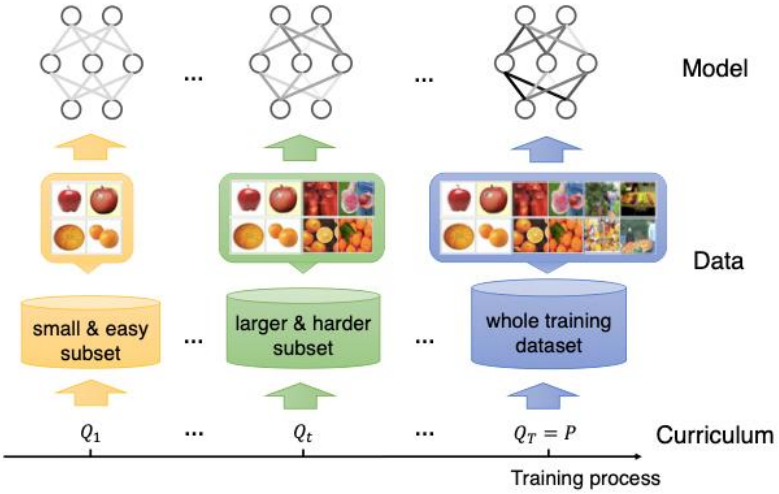


Figure 5. Ranking language model trained with vs without curriculum on Wikipedia. "Error" is log of the rank of the

"Error" is log of the rank of the next word (within 20k-word vocabulary).

1. The curriculum-trained model skips examples with words outside of 5k most frequent words
2. Then skips examples outside 10k most frequent words and so on

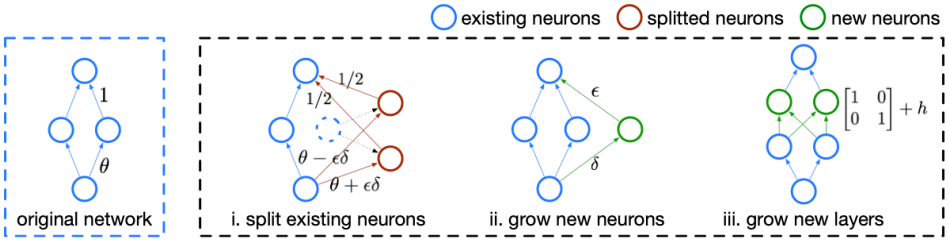
The model choice in this picture remains the same, do you think this is sufficient?



Will be solved with curriculum learning.

Challenge adapting models

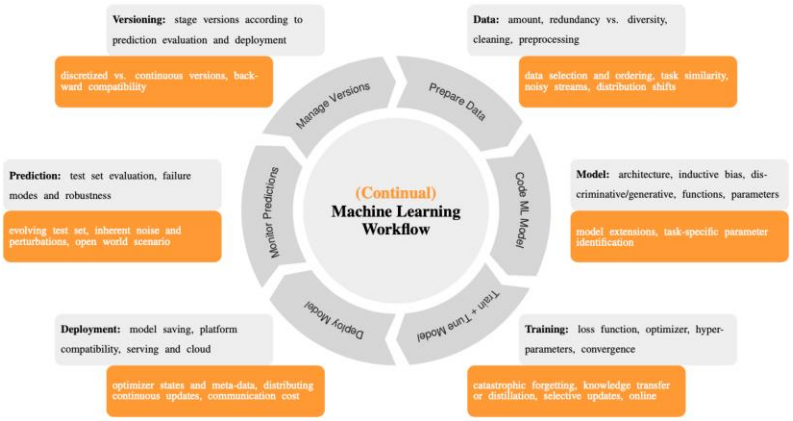
But is our initial model choice and its practical realization still good enough? What if complexity changes? Or even the inductive bias should be altered?



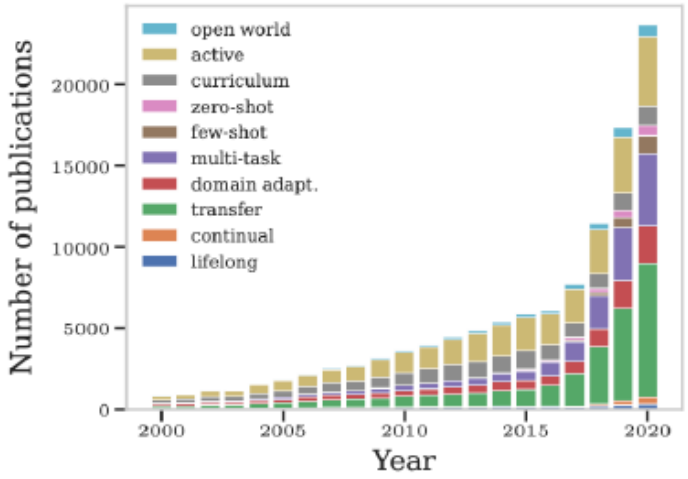
Ideally, we may want all challenges solved together, as hypothesized for biological systems!

Summary of Objectives and content

It turns out just iterating the static ML system is harder than expected! From static ML workflow to continual ML



We try to gain understanding in this course of the following:



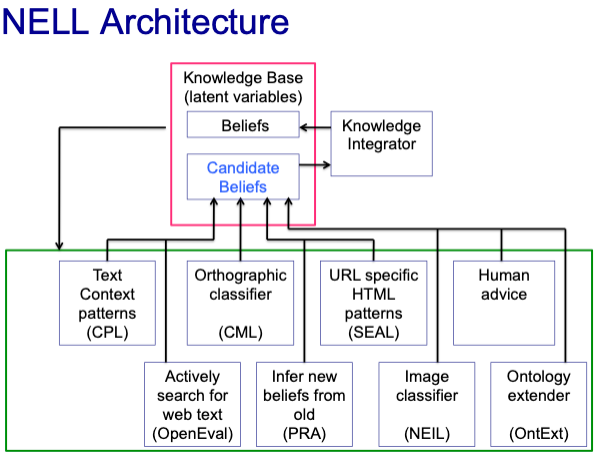
2. Transfer Learning, Domain Adaptation & Continual/Lifelong Machine Learning

2.1. Early Definition of lifelong ML

The early definition of Lifelong Machine Learning according is: 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.

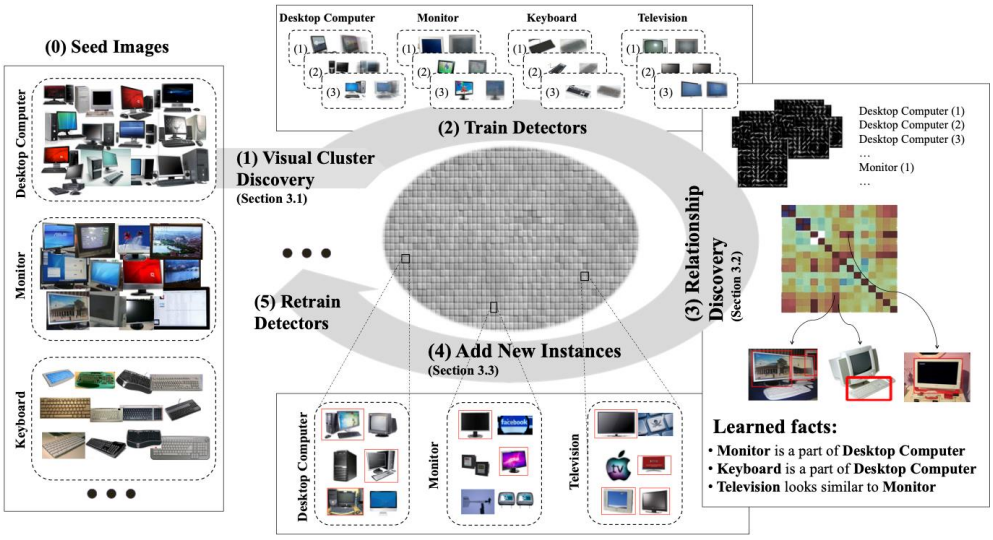
What is knowledge in a machine learning system?

Knowledge in a machine learning system is lot *more than just parameters*. Let's look at the NELL Architecture as an example, which is a never-ending language learner (or lifelong learning system):



It ran 24/7 from 2010-2018 and accumulated over 50 million candidate "beliefs" by reading the web. NELL maintained a relational database to store the "beliefs" it acquired. For instance, one of the entries in NELL's database might read "Facts: *barley is a grain*," signifying a piece of information it had gleaned from online sources. Another example might be "Beliefs: $sportUsesEquip\ (soccer, balls)$," which indicated a belief or link between the sport of soccer and the use of balls as sporting equipment.

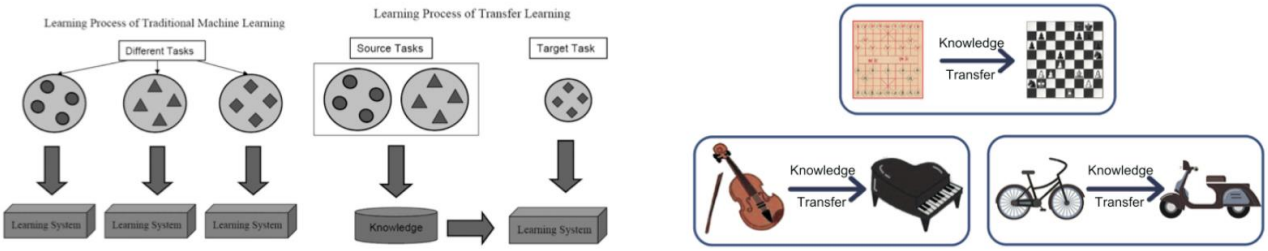
Another example is a never-ending image learner which extracts visual knowledge form Web Data and has a similar idea like NELL: (1) Visual Cluster Discovery, (2) Train Detectors, (3) Relationship Discovery, (4) Add New Instances, (5) Retrain Detectors, (6) Repeat Steps (1)-(5)



The questions to ask now is not only what is knowledge but also what is a task? How is data accumulated and stored? What are the ways to "help" the $(N+1)th$ task?

2.2. Transfer Learning

To help the $(N+1th)$ task assume that we already have "knowledge"/ a model based on initial task(s). This is the essence of transfer learning. We can then use that model on other tasks!



Data Shifts in Machine Learning

When we think about knowledge transfer, we think about how the data is shifted, because in knowledge transfer statistical characteristics of the incoming data change over time. There are three different types of data shift: Covariate Shift, Label Shift and Concept Shift

- In *covariate Shift* the probability distribution $p(x)$ of the input features (covariates) between the source (knowledge) task and the target $(N+1)th$ task.

- In *Label Shift*, the probability distribution of class labels or target variables $p(y)$ changes between the between the source (knowledge) task and the target $(N+1)th$ task.

- In *Concept Shift* the probability distribution $p(y|x)$ changes in the underlying concepts or knowledge that is being transferred from the source task (knowledge) task and the target $(N+1)th$ task.

Definition Transfer Learning

Given a source domain D_S and learning task τ_s, a target domain D_T and learning task τ_T, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and τ_S, where $D_S \neq D_T$ or $\tau_S \neq \tau_T$.

Domain	D
Task	τ
Sources	S
Target	T

Definition of Domain and Task

Given a specific domain, $D = \{\chi, p(x)\}$ a task consists of two components which are a label space Y and an objective predictive function $f()$ (denoted by $T = \{Y, f()\}$, which is not observed but can be learned from the training data, which consist of pairs $\{x^{(n)}, y^{(n)}\}$, where $x^{(n)} \in X$ and $y^{(n)} \in Y$.

To summarize a Domain D in detail is a pair of data distribution and corresponding feature space χ. A Task finds a function $f()$ to map to labels in the case of supervision! Of course, generally $\chi_S \neq \chi_T$ or $p_S(x) \neq p_T(x)$.

Definition transductive transfer Learning

Given a source domain D_S and learning task τ_s, a target domain D_T and learning task τ_T, transductive transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and τ_S, where $D_S \neq D_T$ or $\tau_S = \tau_T$.

- Feature spaces between the source and target are different $\chi_S \neq \chi_T$
- Feature spaces between source and target are the same, but $p_S(x) \neq p_T(x)$.
- Frequently encountered as domain adaptation or sample selection bias

Definition inductive transfer learning

Given a source domain D_S and learning task τ_s, a target domain D_T and learning task τ_T,inductive transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and τ_S, where $D_S \neq D_T$ or $\tau_S \neq \tau_T$.

(A few) (labeled) data points are required to “induce” the target predictive function

Questions and Goals

Central questions:

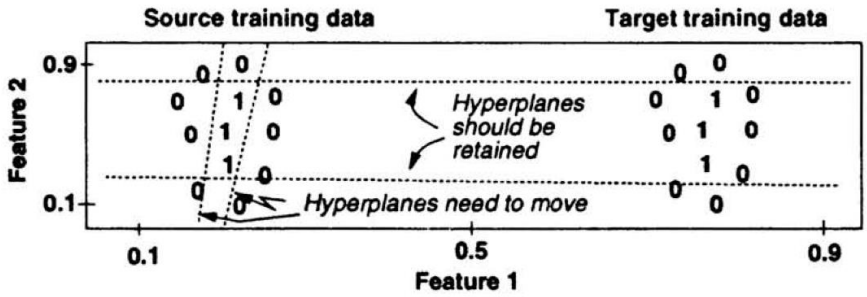
1. What to transfer: some knowledge is domain or task specific or may be more general/ transferable
2. When to transfer: when does transfer help or when does it even hurt?
3. How to transfer: algorithms to actually include, transfer/combine knowledge

Central objectives:

1. Improved loss/more accurate function in direct comparison to learning just on the target
2. Accelerate learning
3. Reduce data dependence (of target) (Some) central questions 1. What to transfer: some knowledge is domain or task specific or

Domain Adaptation for transductive transfer

Early approaches transfer by identifying the amount that a specific hyperplane helps to separate the data into different classes (& then reweighting/reinitializing).



How does transductive transfer learning work? Samples from the source domain and target domain are represented by different features. By using projection matrices P and Q we transform the heterogenous samples from two domains into an augmented feature space! This is also called domain adaptation!

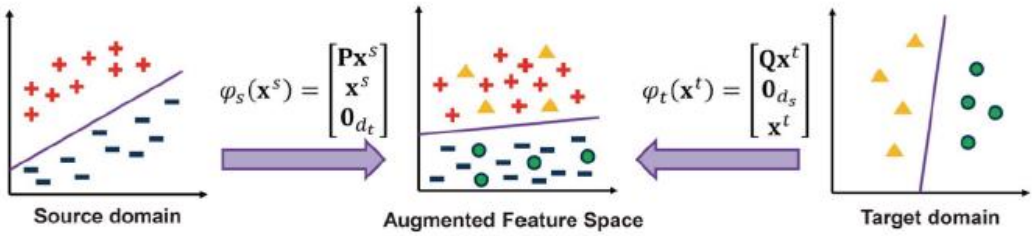
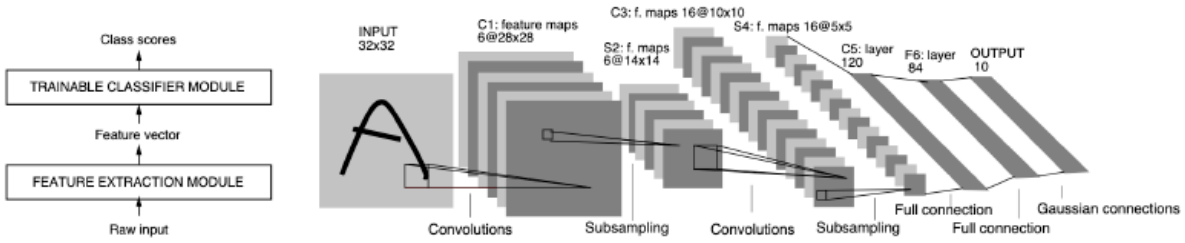
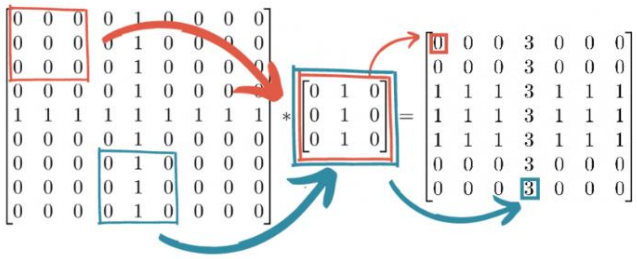


Fig. 1. Samples from different domains are represented by different features, where red crosses, blue strips, orange triangles and green circles denote source positive samples, source negative samples, target positive samples and target negative samples, respectively. By using two projection matrices **P** and **Q**, we transform the heterogenous samples from two domains into an augmented feature space.

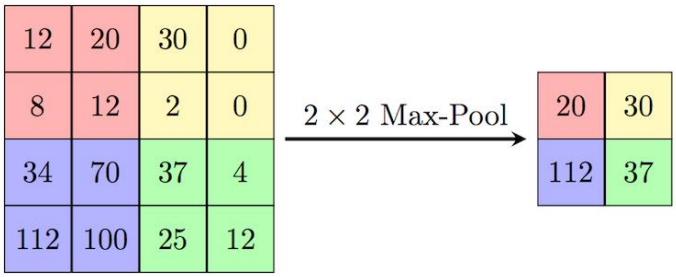
Recap: CNNs



- Convolutions: multiple learnable patterns, “weight-sharing” - sliding window



- Pooling: not learned dimensionality reduction, also e.g., local invariance



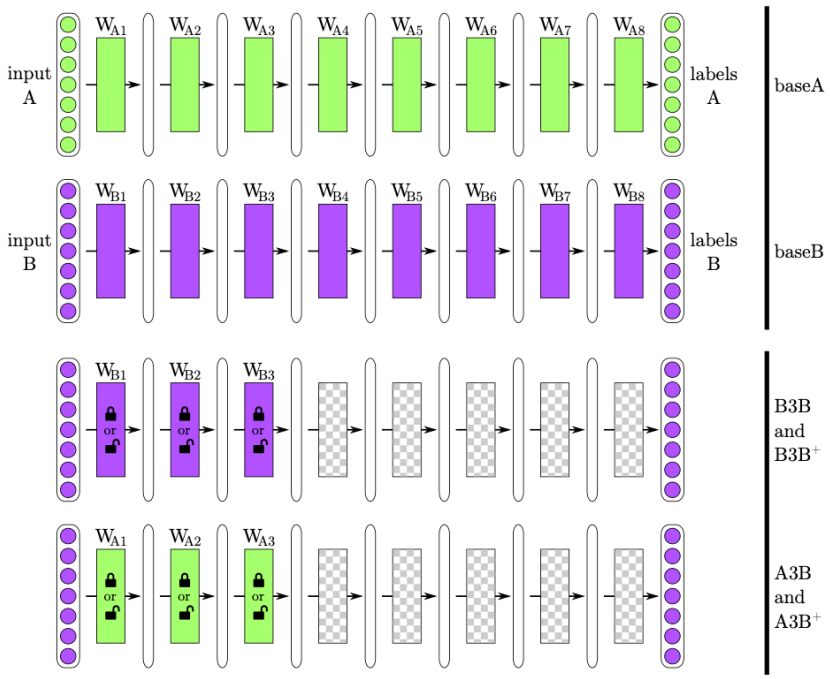
- Modern advances like dropout, batch-norm etc. of which some are learnable
- If you don’t know how learning/training works, don’t worry, we’ll visit this next week

The key hypothesis is that early layers learn generic patterns, deeper layers become increasingly more specific!

Inductive Transfer learning in deep learning

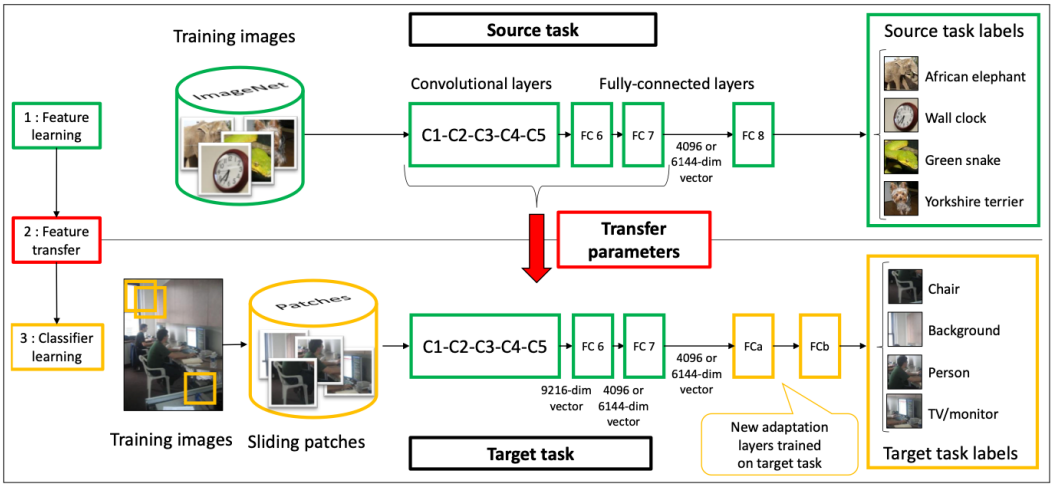
How transferable are features in deep neural networks since we did a recap of CNNs? Example Inductive ImageNet transfer:

1. Split ImageNet into 2 sets of 500 classes: A and B
2. Lock different sets of layers/representations
3. Randomly initialize upper remaining layers
4. Alternatively: continue training/fine-tuning transferred layers



Let's look at these inductive transfer learning methods in detail!

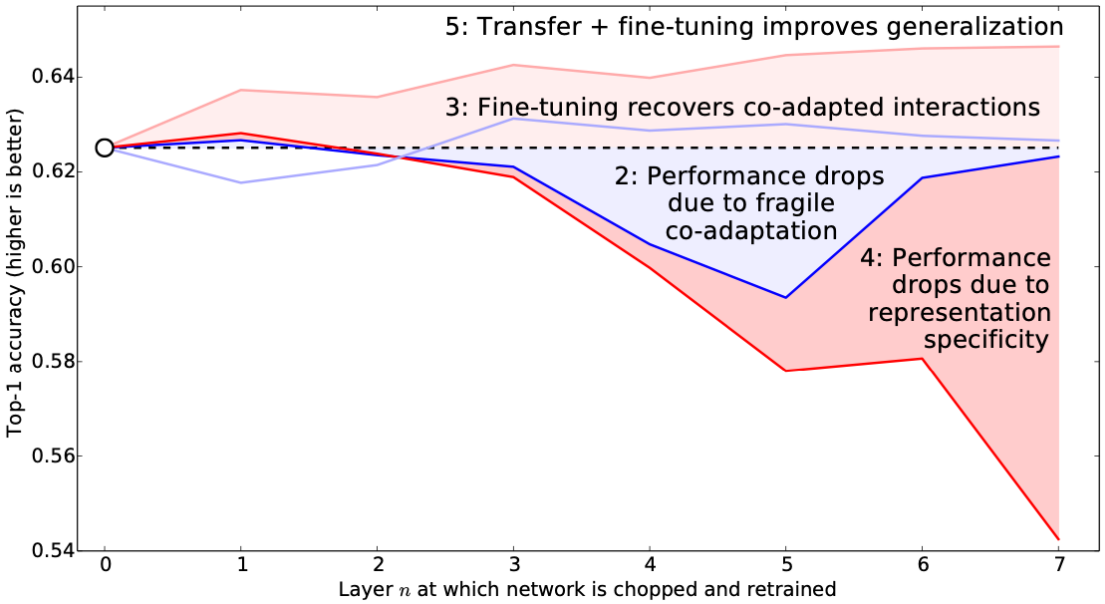
- Transfer features (knowledge) of model trained on source task A to the model for target task B with layers allowed to fine-tune on task B: This has the best accuracy and improves generalization which increases the higher the number of the networks layer where it is chopped and retrained. See graph 5.



- Transfer features (knowledge) from model trained on source task A to the model for target task B with frozen layers: The performance drops due to representation specificity (see graph 4).

- Features (knowledge) learned from task B are copied and kept frozen, with additional random layers trained on task B. The performnace drops due to fragile co-adaptation (see graph 2).

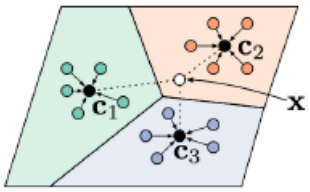
- Copied features (knowledge) from task B are allowed to fine-tune on task B, adjusting to potentially new or nuanced aspects of the same task. Fine tuning recovers co-adapted interactions (see graph 3).



The role of embeddings: few-shot, one-shot and zero-shot transfer

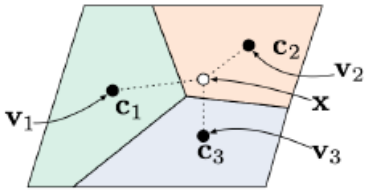
Embeddings play a crucial role in leveraging information from a source domain to assist learning in a target domain, particularly when there's scarcity of labeled data. As always, we will assume a classification problem.

In *Few shot learning* a model is trained to perform target tasks with a very small number of examples. Basically, Compute prototype c as the mean vector of each class with parametrized embedding function of a support set of labelled examples!



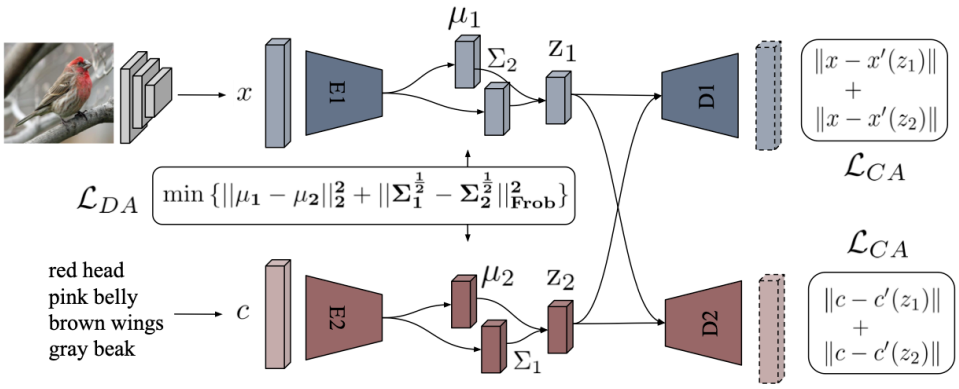
(a) Few-shot

In *zero-shot learning*, the model must perform target tasks for which it has seen no examples at all during training! Given a distance function d, classify according to softmax over distances to the prototypes in embedding space.



(b) Zero-shot

Common approach for zero or few-shot learning measures the maximum mean discrepancy or Wasserstein distance:

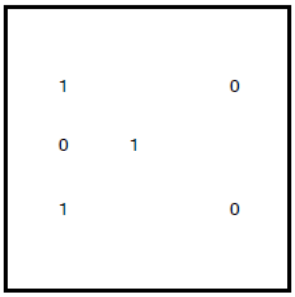


In *One-shot learning* a model is trained to perform target tasks with only one example per class.

1. Learn from extra sample a distance function d that achieves gamma separation
2. Learn a nearest neighbor classifier, where the classifier employs d

Why is transfer challenging?

How would you separate this data with a set of hyperplanes? (Try 3)



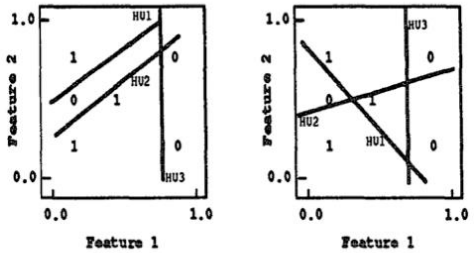


Figure 2: Two examples of hyperplane sets that separate training data in a small network.

It's not intuitive if the transfer works or not! Is selective transfer a solution for this?

Alternatively, to selecting entire layers, freezing the weights, or letting them partially adapt, we could also try to select and inject only the features that are "representative" for the new task. For instance: pick only features that have large activations.

Representations are biased in ways that we don't anticipate:

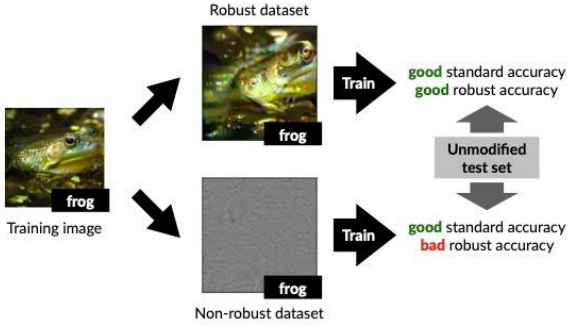
- Texture bias

(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

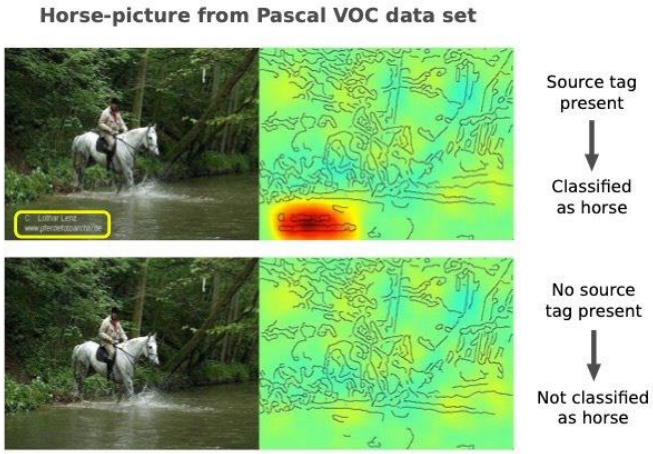
(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

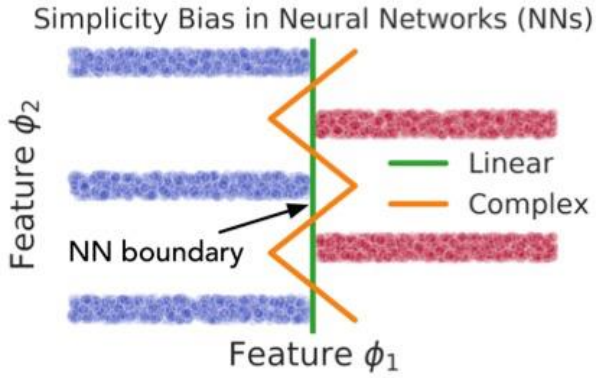
- Adversarial features



- Confounders



- Pitfalls of simplicity



Transferring knowledge with Hints

Another way to transfer knowledge is to learn from hints. A hint is any piece of information about the function f. As a matter of fact, an input-output example is a special case of a hint. A hint may take the form of a global constraint on f, such as a symmetry property or an invariance.

- Symmetry: We could directly include in- or equivariance into our representations (e.g., rotation)

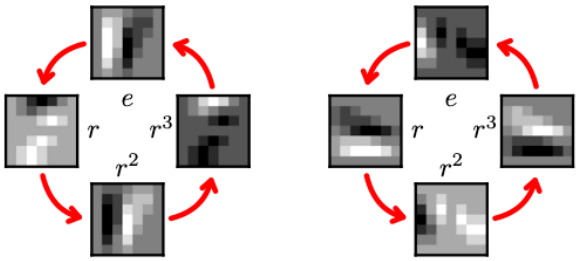


Figure 1. A p4 feature map and its rotation by r.

- Invariance:
  - We could also incorporate a degree of scale invariance, or even try to learn it.
  - We could pre-process and account for illumination changes
  - We could assume that RGB colour ratios are quasi-invariant with white illumination. Alternatively, we can use local binary patterns (simplified): mark all nearest neighbors for a pixel with 0 if greater and 1 otherwise, compute histogram over values to create features-

2.3. Back to lifelong learning

Lifelong Machine Learning is a continuous learning process. At any time, point, the learner performed a sequence of N learning tasks, .These tasks can be of the same type or different types and from the same domain or different domains. When faced with the (N+1)th task (which is called the new or current task) with its data , the learner can leverage past knowledge in the knowledge base (KB) to help learn. The objective of LML is usually to optimize the performance on the new task , but it can optimize any task by treating the rest of

the tasks as previous tasks. KB maintains the knowledge learned and accumulated from learning the previous task. After the completion of learning , KB is updated with the knowledge (e.g., intermediate as well as the final results) gained from learning . The updating can involve inconsistency checking, reasoning, and meta-mining of additional higher-level knowledge

3. Optimization & Knowledge Retention

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

3.1. Intuitive Examples for forgetting

Intuitive examples for forgetting are K-means and linear regression.

K-means

An intuitive example is *K-means*. Given an initial set of k means $m_1^{(1)}, \ldots, m_k^{(1)}$ the algorithm proceeds by alternating between two steps:

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

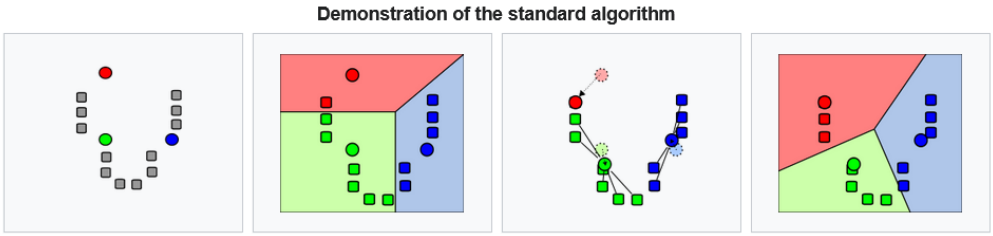
$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\},$$

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

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

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

The algorithm has converged when the assignments no longer change. The algorithm is not guaranteed to find the optimum.



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

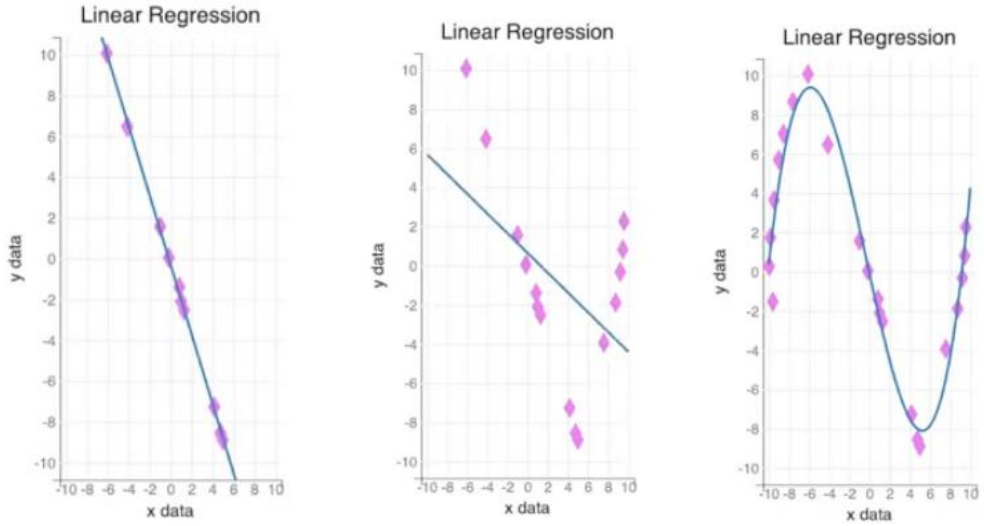
So now when will it be surprising to see that we forget if we add new data? Maybe the number of clusters could be an indication for that since data isn't accumulated but replaced, so that the mean changes abruptly.

Considerations such as exponentially moving average of the mean leads to not forgetting the source data abruptly.

Linear Regression

Another intuitive example is linear regression. If we for example add new data points the fitted line will forget the old solution and at the same time also doesn't fit the new task well (solution to that would be to use polynomials).

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



Conclusion

It's not very surprising that if we add new data points and we keep updating the current model we have, we will essentially overwrite what we had before.

3.2. Optimization

Risk & Losses

What we would like to generally do is minimize the following scenario: Find a hypothesis or decision procedure:

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

and define the risk or expected loss as:

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

where $\tilde{D}$ is data from the true distribution, represented by parameter θ^*. The challenges are we cannot compute above risk (usually don't know the distribution). Also, if we think of e.g., binary classification, i.e., a 0-1 measure, it can be hard to optimize as it is not smooth. Instead, we use:

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

We can look at the true but unknown response and our predictions $\delta(x)$ given an input x. As we still do not know the true distribution, we can also use empirical estimates:

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

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

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

or similarly an unsupervised reconstruction:

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

Gradient Descent

There are various optimization algorithms, the most popular ones are perhaps: (Stochastic) gradient descent (SGD) and expectation maximization (EM). Let us consider (S)GD here, as the "workhorse" underlying a lot of deep learning. In the simple form, a first order optimization algorithm to find a minimum of a differentiable function. This is achieved by iteratively taking (small) steps in the gradient direction of a function f in the direction in which it decreases the fastest:

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

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

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

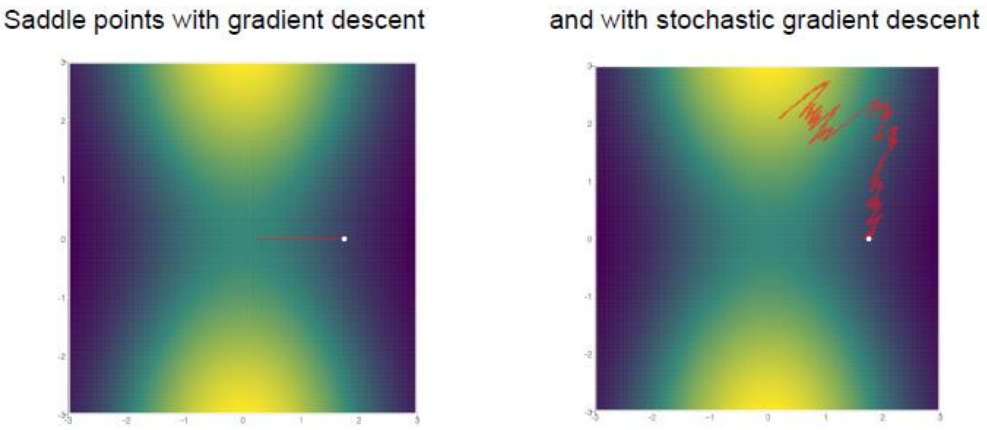
Then iterative updates become (where in neural nets we backpropagate gradients):

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

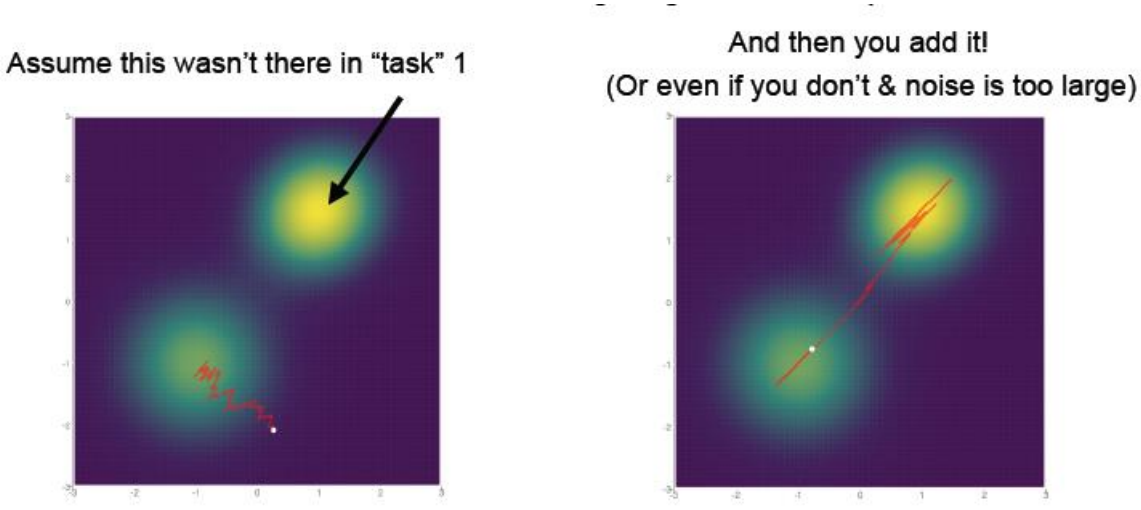
Forgetting in Gradient Descent

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

The difference between *SGD* and *GD* is that in SGD we add noise or jitter. This enables us to get out of a saddle point.

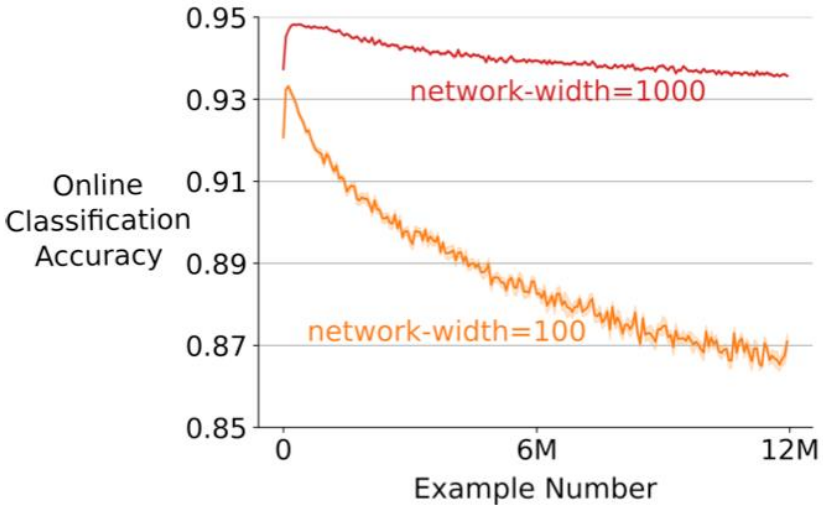


From an SGD point of view inference/forgetting can be explained like this:



The undesired trivial solution would be to use large amounts of parameters plus data accumulation. However, we are constrained by capacity, we won't learn indefinitely!

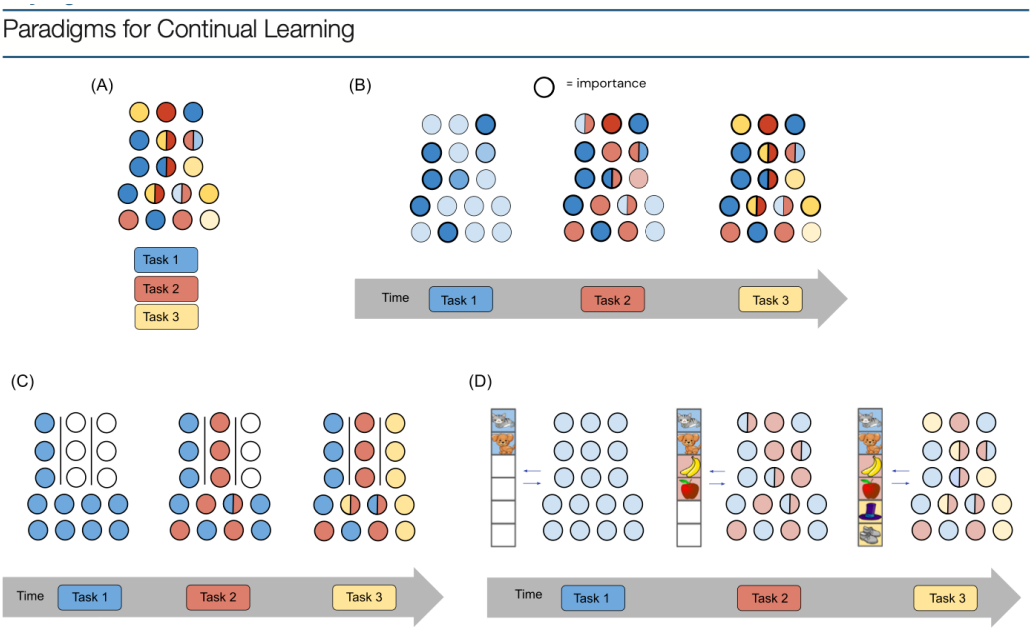
Keep in mind if the number of parameters is limited & already learned, it will become increasingly difficult to encode new concepts e.g., on the right example of permuting the data points over time. Here is an example that illustrates this.



3.3. Knowledge Retention Methods - Introduction

The Continual Learning methods/paradigms (to prevent forgetting) are

- Regularization of important parameters/Knowledge Distillation (see graph B)
- Rehearsal/Pseudo Rehearsal (see graph D)
- Modifying the architecture (see graph C and chapter 5)



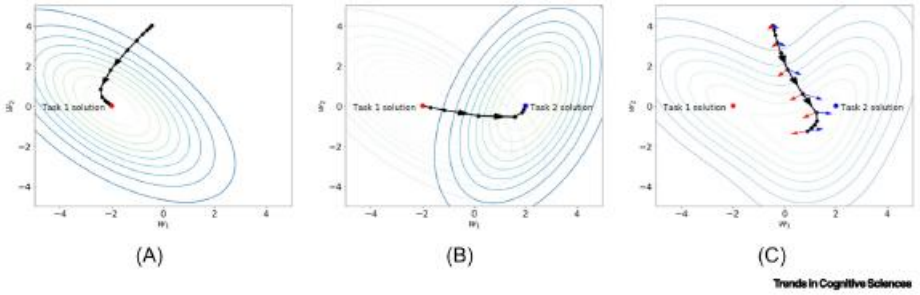
Excurse: Stability - plasticity (sensitivity) dilemma

For regularization approaches, what we are essentially interested in is the so-called *stability - plasticity (or sensitivity) dilemma*!

The Stability-Plasticity Dilemma underscores a trade-off in learning systems: maintaining previously learned knowledge (stability) versus adapting to new information (plasticity).

High stability can lead to a system's inability to learn from new data, while high plasticity can cause a system to forget previously learned information.

Let's look at illustrations of gradient descent for different tasks. In (A) we see the trajectory taken by gradient descent optimization when minimizing a loss corresponding a single task. In (B) the optimization trajectory when subsequently training the same model on a second task. In (C) the trajectory taken when using the total loss from both tasks (black) and the gradients from each individual task at multiple points during optimization (red and blue)



Now lets deep dive in the methods!

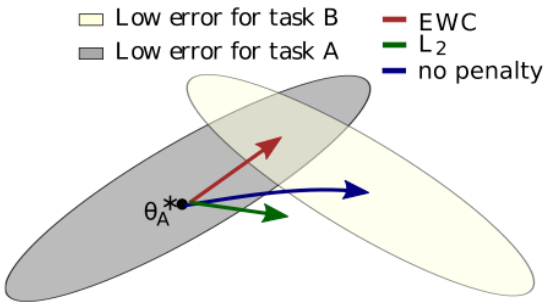
3.4. Finding and regularizing important parameters and Knowledge Destillation

Either identify relevant parameters for a task and make sure they do not change much, or make sure the input output relationship remains the same (knowledge distillation).
Gradient based approaches preserve parameters based of their importance to previously learned tasks.

Finding and regularizing important parameters

Let's start with the first part. For regularization approaches, what we are essentially interested in is the so-called *stability - plasticity (or sensitivity) dilemma* as explained above!
Techniques like Elastic Weight Consolidation and Synaptic Intelligence are proposed to tackle this dilemma by balancing the trade-off between stability and plasticity!

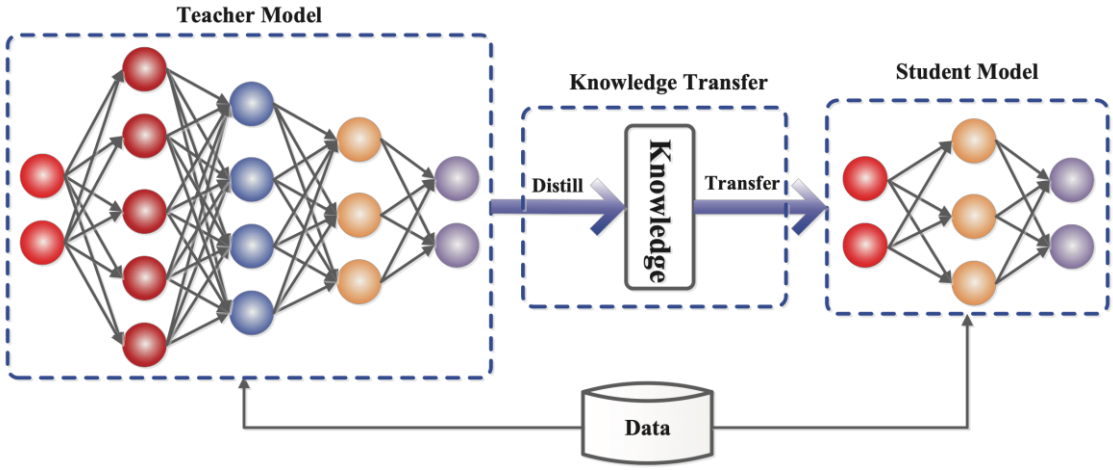
- *Elastic weight consolidation* (EWC): Instead of naively continuing to optimize task B, we can impose a penalty on previously learned parameters (assuming over-parameterization). We will need to find a matrix F that tells us which parameters are most important for task A. This matrix can for example be the Fisher information matrix.



- *Synaptic Intelligence:* Here, "synapse" synonymous with parameter. Key idea: Change (with time t) in loss is well approximated by the gradient. Each parameter change contributes an amount to the change in total loss. Assign an importance to each parameter according to the monitored trajectory and formulate a similar penalty to EWC again (with a different measure of importance).

Knowledge Destillation

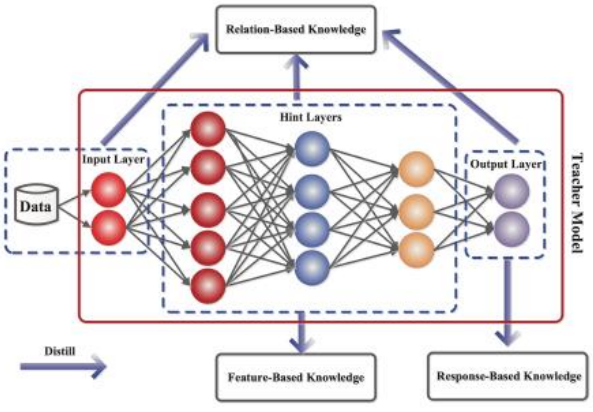
Alternatively, we know that if we have enough parameters, there are many potential solutions to produce the same input-output relationships. Key idea: Let's try to maintain a task's input-output relationship! This is called knowledge distillation.



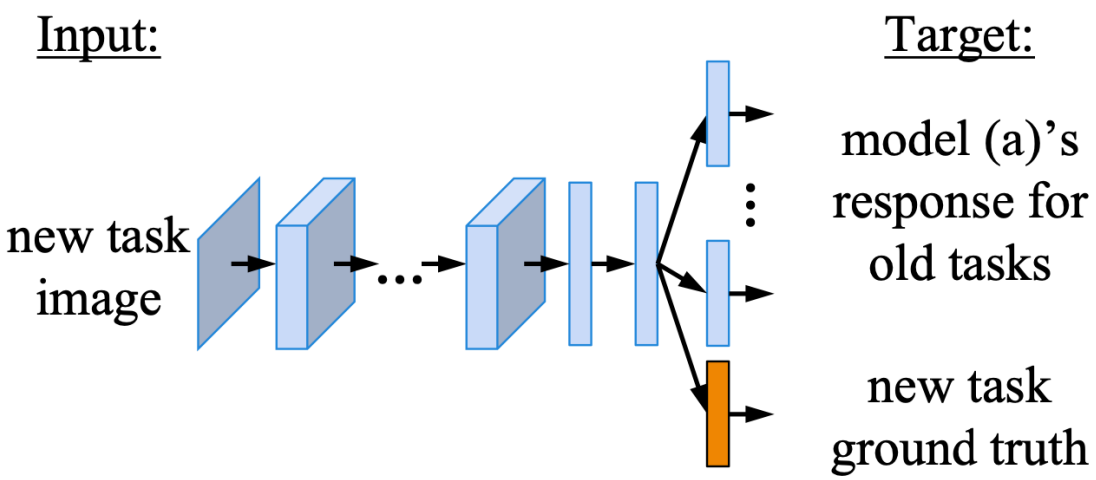
In essence we are making sure that the distance between teacher and student models is minimized, or more generally minimizing the KL divergence over the two probability distributions. Apart from continual learning why would we like to distill?

- Simplified structure
- Quantized structure
- Same structure
- Small structure (optimized/condensed)

We generally have various choices of what types of relationships we wish to distill (and how) from the teacher model to the student model: Feature-Based Knowledge, Relation-Based Knowledge, Response-Based Knowledge.

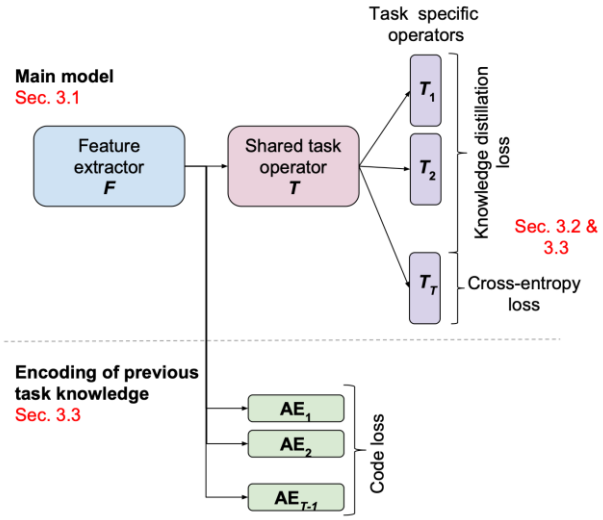


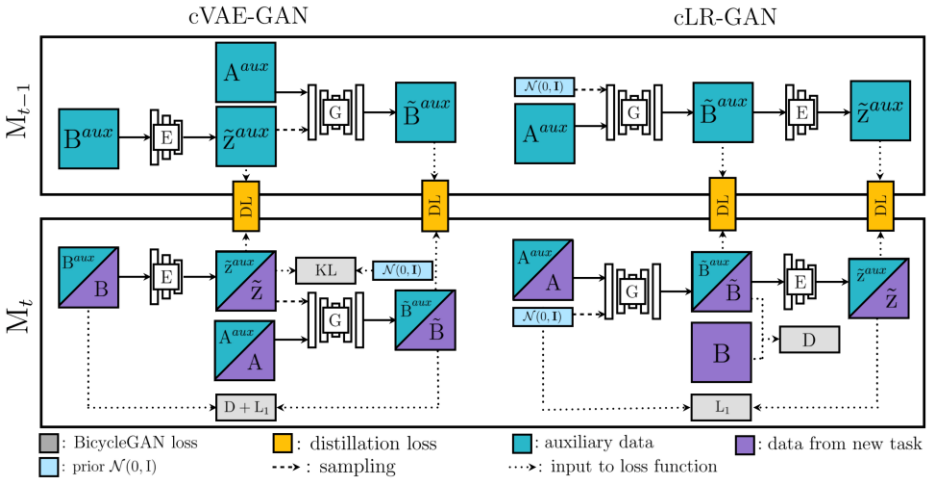
Let's now make the jump to continual knowledge distillation (learning without forgetting). Key idea: compute task "head" with new data and continue to preserver this input-output relationship, while learning a new task "head" simultaneously!



But "cross-talk" can be challenging if we don't dedicate an individual expert to each task. Especially true for e.g., Softmax layers that normalize over the entire output vector. Let's discuss if expert outputs are desirable when we learn about structure changes.

Encoder Based Lifelong Learning has become very popular in continual learning and adapted in various ways. It also has been extended to generative models, (shared feature spaces) etc. Distillation is regularly used together with other techniques, such as generative models, rehearsal, or architecture modifications/expansion.



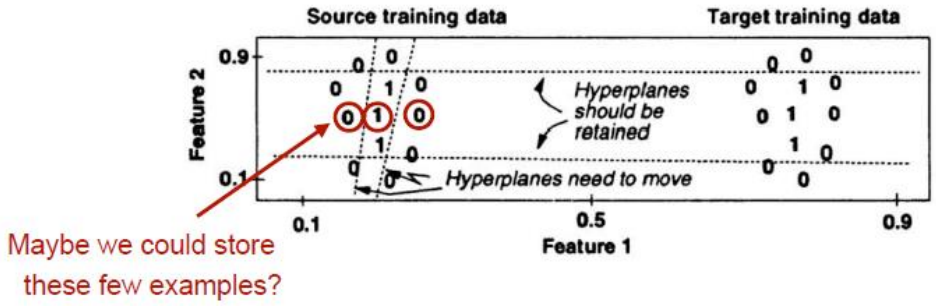


3.5. Rehearsal and Pseudo Rehearsal

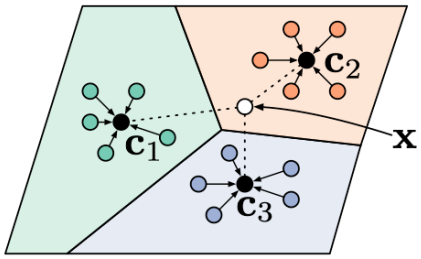
Either store a subset of old data to rehearse or make use of a generative model to generate old task data.

Rehearsal

Let's start with the first part. Rehearsal is a memory-based method that writes experience to memory to avoid forgetting. Assuming privacy is not a concern and that we have auxiliary memory: some data is more relevant than other, can we retain a subset?



Let's start with an example this time and then discuss desiderata. Recall few-shot learning and nearest mean classifier.



(a) Few-shot

1. Get prototype as mean vector of each class.

2. Given a distance function d, classify according to distances to the prototypes in embedding space.

Note, nomenclature is inconsistent: prototypes, exemplars, core sets (will be defined later). The simplest method is the *incremental classifier and representation learning (iCaRL)* algorithm.

The iCaRL Classification Algorithm is:

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

How is our memory buffer filled with specific examples?

- Iteratively: one by one, based on “herding”.
- Pick exemplars to best approximate the overall mean.
- For a size of k exemplars: loop k times!

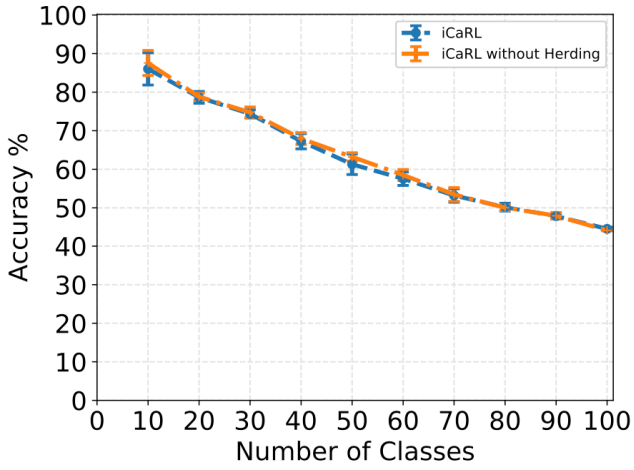
Our memory buffer is limited, how do we later remove samples or replacing exemplars with iCaRL?

- The Memory buffer is like a prioritized list
- Later picked exemplars for a task “weigh” less
- Simply cut and repopulate.

How do we train iCaRL incrementally?

1. Concatenate dataset with exemplars/ interleave exemplars into training
2. Pick new exemplars (not shown on the right) and replace existing
3. Additionally use knowledge distillation

Does the herding selection algorithm outperform random selection? Randomly picking out the data point in comparison to the herding procedure does not make a difference!



How expected are our observations? The larger our memory buffer the larger our classifier accuracy gets!

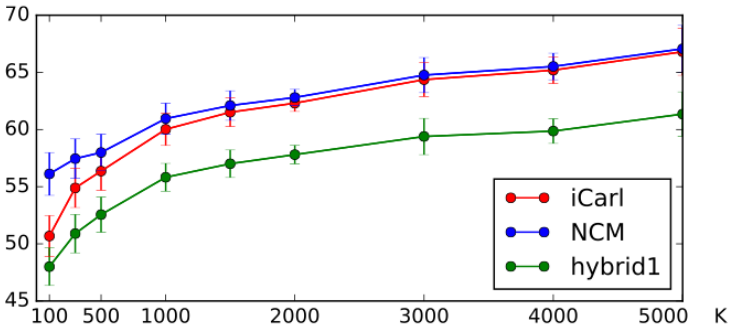
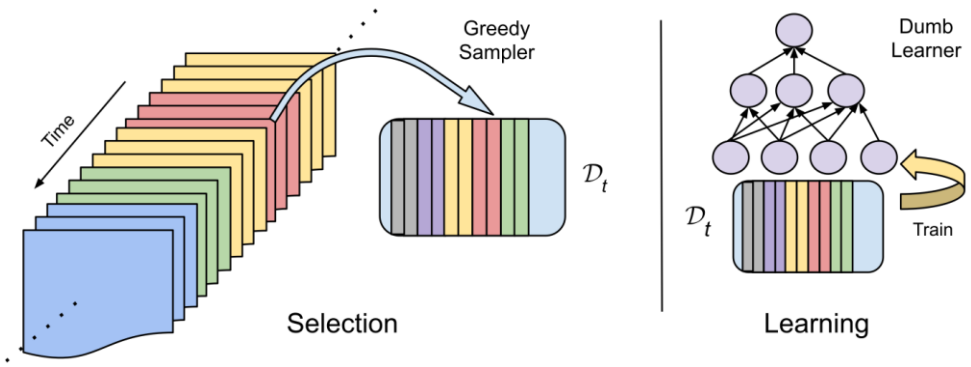


Figure 4: Average incremental accuracy on iCIFAR-100 with 10 classes per batch for different memory budgets K.

We need to question the role of memory now. Is it really just the data subset that we retain? A "dumb learner" comparison suggests that we may get similar performance if we just train on the exemplar subset!

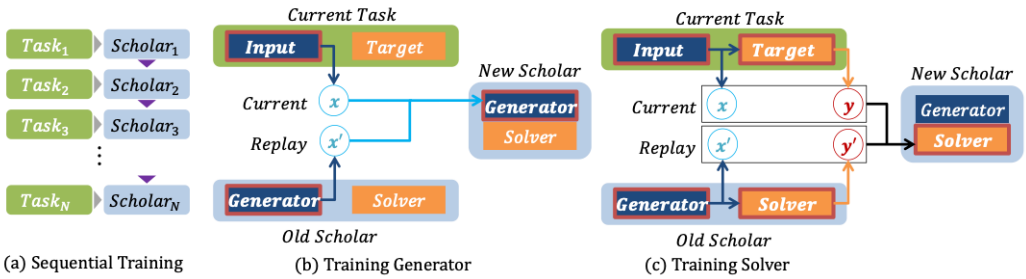


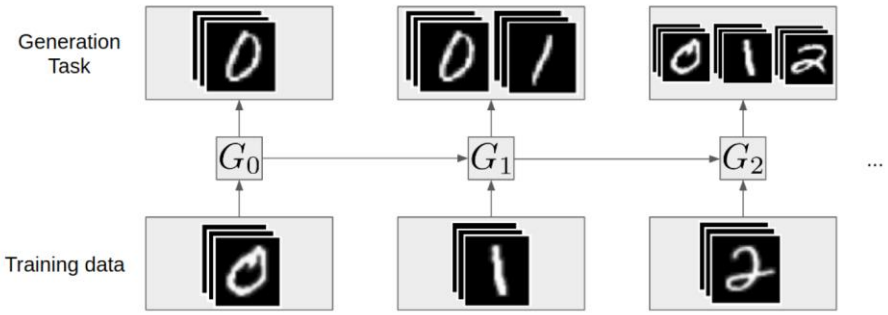
Are memory buffers of real examples the solution to continual learning? While it is an effective method in ANNs, rehearsal is unlikely to be a realistic model of biological learning mechanisms, as in this context the actual old information (accurate and complete representation of all items ever learned 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.

Pseudo-rehearsal

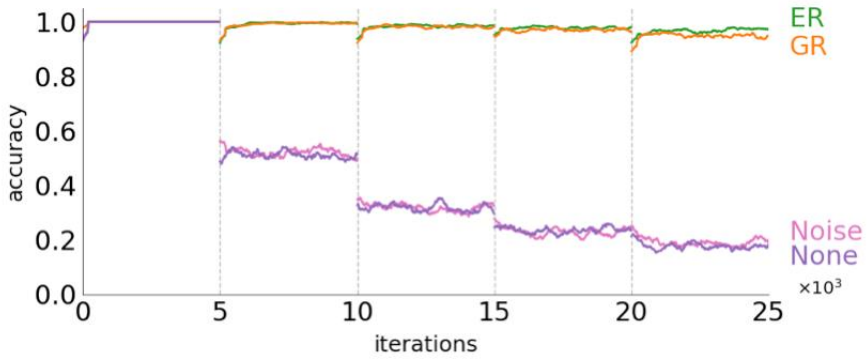
Pseudorehearsal is based on the use in the rehearsal process of artificially constructed populations of "pseudoitems" instead of the "actual" previously learned items. A pseudoitem is constructed by generating a new input vector (setting at random 50% of input elements to 0 and 50% to 1 as usual) and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

We could train two machine learning models: a generator (there are many types) and a task solver. After that alternate training!





Exemplar Rehearsal (ER) and Generative (Pseudo-)Rehearsal (GR) can work equally well if we have a powerful generator. In contrast, randomly rehearsed sampled noise patterns will no longer work on complex tasks!

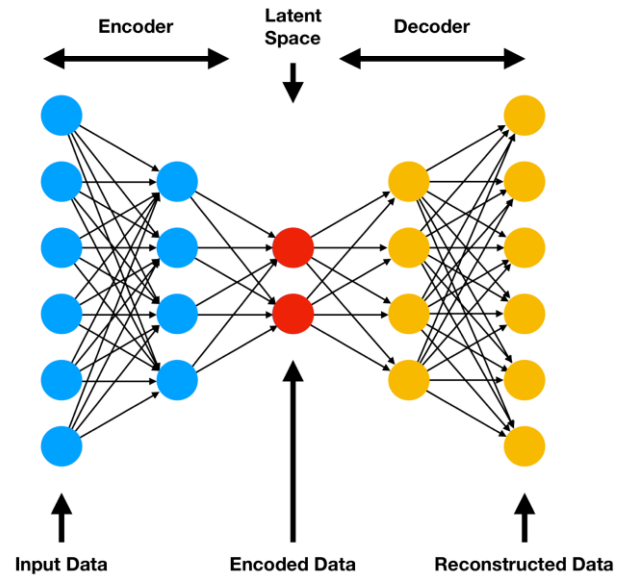


A short interlude: generative models

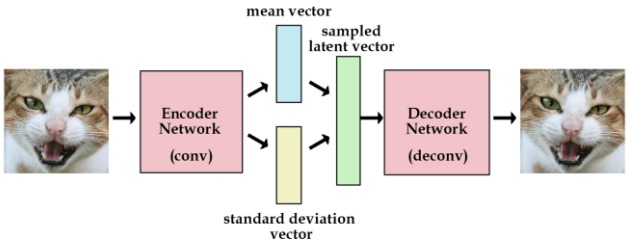
A discriminative model typically learns something like $p(y|x)$. A generative model also learns about the data distribution $p(x)$ and the process by which data is created (the generative factors). Having a generative model does not mean we cannot also solve discriminative tasks $p(x,y) = p(y|x)p(x)$!

Let's take a look at autoencoders Why? To later see that we don't necessarily need two separate models:

- Learn a representation, an "encoding" of the data
- An encoder maps to a "code": "latent variables"
- A decoder learns to reconstruct the input



The autoencoders latent embedding/variables may be difficult to grasp if unconstrained. But we could constrain the latent space to follow a specific distribution, e.g. a Variational Autoencoder



Describing this more formally, let us consider:

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

But the parameters and values of latent variables $\mathbf{z}$ are not known to us.

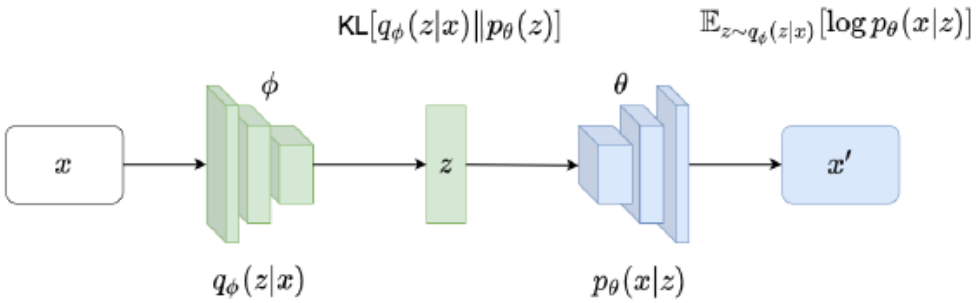
$p_\theta(x) = \int p_\theta(x,z)dz$ is intractable!

After some derivation we can use instead a variational lower-bound to $p(x)$ also called evidence lower bound (ELBO).

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

The 1. term is the expected reconstruction error given by the log-likelihood (with sampling). The 2. term is a KL divergence encouraging the approximate posterior to be close to a prior. (Reverse) Kl divergence measures the amount of information (in nets, or units of 1/log2 bits) required to “distort” p(z) into q(z)!

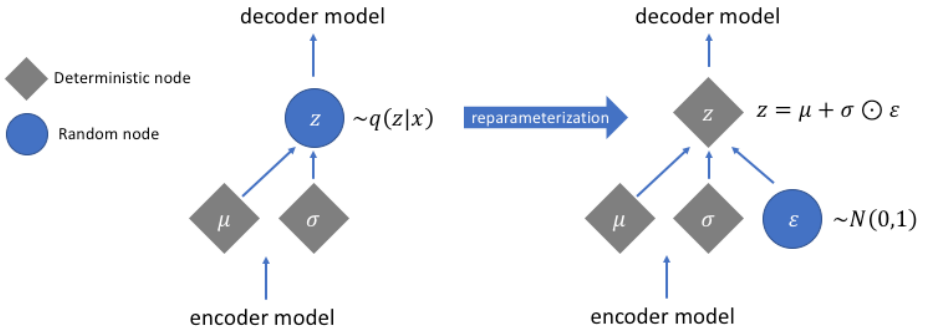
The recognition model, a probabilistic encoder $q_\phi(z|x)$, is a variational approximation to the intractable posterior $p_\theta(x|z)$. Given a datapoint x it produces a distribution over possible values of z from which it could have been generated. The probabilistic decoder, $p_\theta(x|z)$ produces a distribution over possible values of x given z!

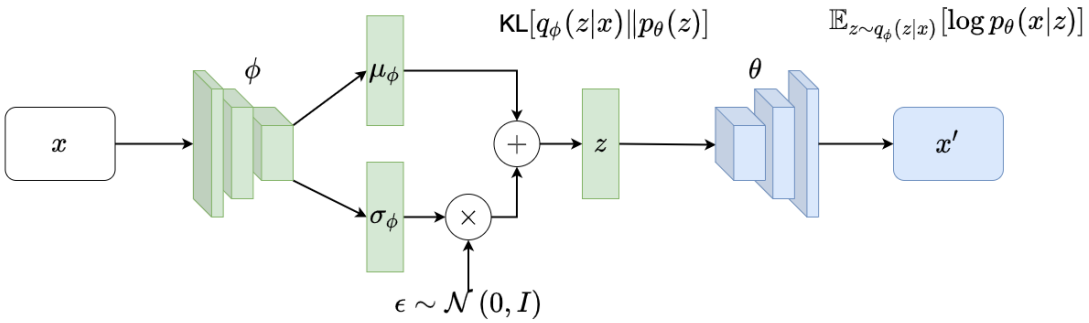


For example, we can approximate the posterior as multivariate Gaussian

$$\log q_\phi(z\,|\,x) = \log \mathcal{N}(z; \mu, \sigma I)$$

Then use reparameterization trick to generate samples from $q_\phi(z|x)$ with z as deterministic variable. E.g., sample posterior $z\sim q_\phi(z|x)$ using $z = +\mu + \sigma\varepsilon$ with samples $\varepsilon\sim N(0,1)$





Back to pseudo-Rehearsal: Why was this detour important?

Why did we go through all this math? And why is it important for continual learning? We can sample from a trained model: $z \sim p(z)$, here $N(0,1)$, and generate (decode) x. We also have the approximate posterior that we could regularize in continual learning. This leads us to variational continual learning.

Let's look at the evidence lower bound (ELBO).

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

The left term represents the "likelihood focused" perspective: generative/pseudo rehearsal. Generate old tasks' data and concatenate it with new task data. Primarily optimize "the likelihood" (left)

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

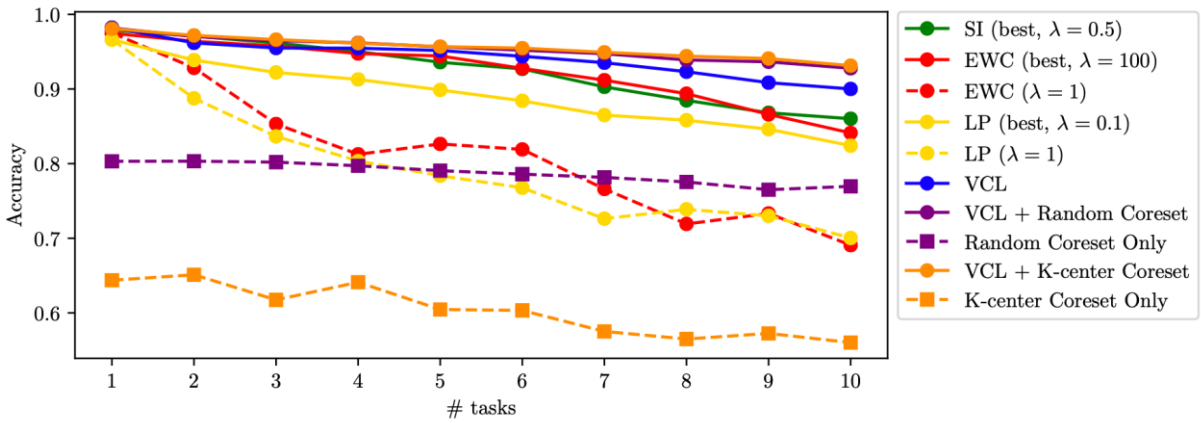
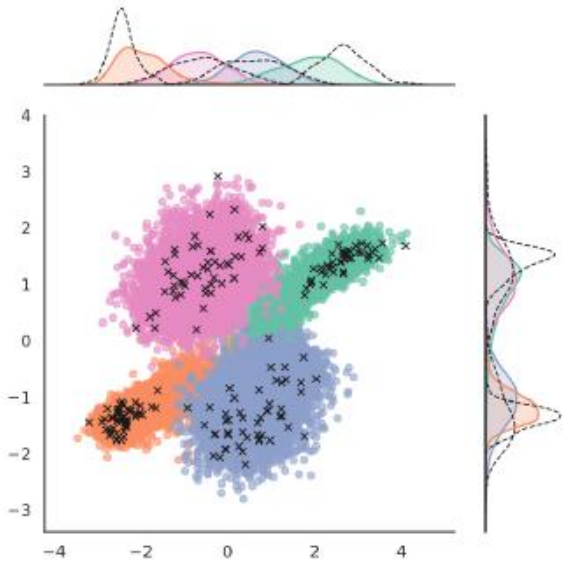


Figure 2: Average test set accuracy on all observed tasks in the Permuted MNIST experiment.

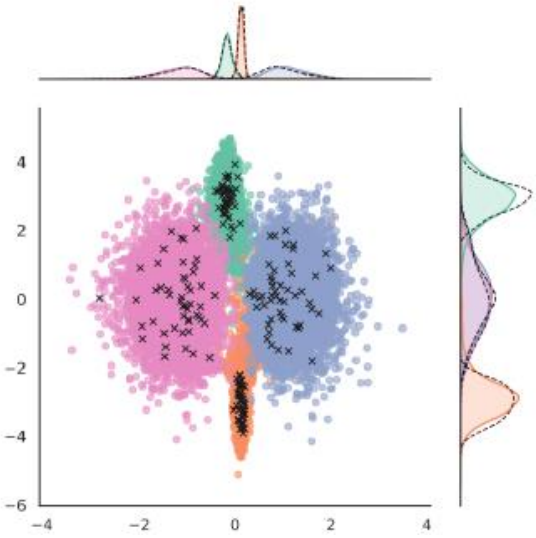
Optionally we can also store real data subsets (or a core set). What is a core set? The term core set is often loosely employed in modern literature to be synonymous to exemplars and subsets of data "coresets are small, (weighted) summaries of large data sets such that solutions found on the summary itself are provably competitive with solutions found on the full data set"

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

Note how this is specific to some data, a set of questions/queries, models and loss/cost functions. Why could we potentially pick better data subsets/exemplars now? Here is an example of a 2-D latent space with 4 classes/clusters:



Random or k-means (depending on the amount of k) may not mirror the distribution well! If we know our approx. posterior, we could sample and pick instances that lie close the samples (It's not actually that easy, for various reasons, but the intuition is that we are somewhat aware of p(x) now.



We could now also use task-specific priors more explicitly. If we use task specific Gaussians its called CURL!

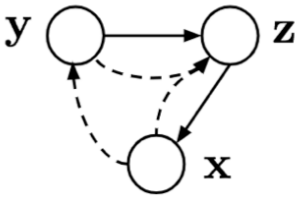


Figure 1: Graphical model for CURL. The categorical task variable $\mathbf{y}$ is used to instantiate a latent mixture-of-Gaussians $\mathbf{z}$, which is then decoded to $\mathbf{x}$.

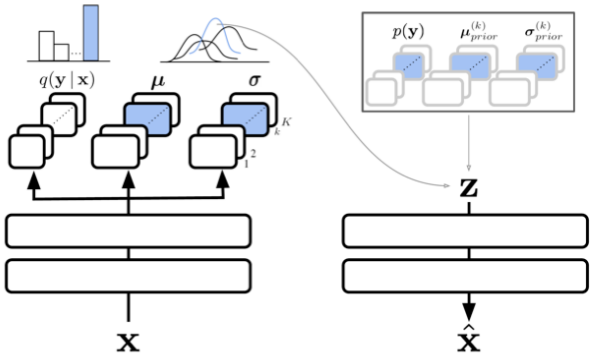


Figure 2: Diagram of the proposed approach, showing the inference procedure and architectural components used.

4. Have we solved forgetting? Why are we not done?

Why may we need these multiple perspectives? Generators also suffer from forgetting, errors can "snowball" rapidly! Fine-tuning alone forgets, regularization is a trade-off and data rehearsal can overfit. What could our expectations be, what might we desire?

- Constant memory budget?
- Pragmatically? A selection algorithm that outperforms randomly stored data points?
- A way to shrink the memory buffer to add new tasks, e.g., recursively select exemplars.
- Ideally? Knowledge of the distribution(s) and a subset with approximation guarantees?
- A natural formulation to allow (pseudo-)rehearsal, regularization…?
- …. many more …?

We can (naturally) combine ideas! In most of what we have seen so far however, the "architecture" **has remained static**. The assumption that we can continue accumulate knowledge because we have enough capacity has been prevalent. See chapter 5 for more!

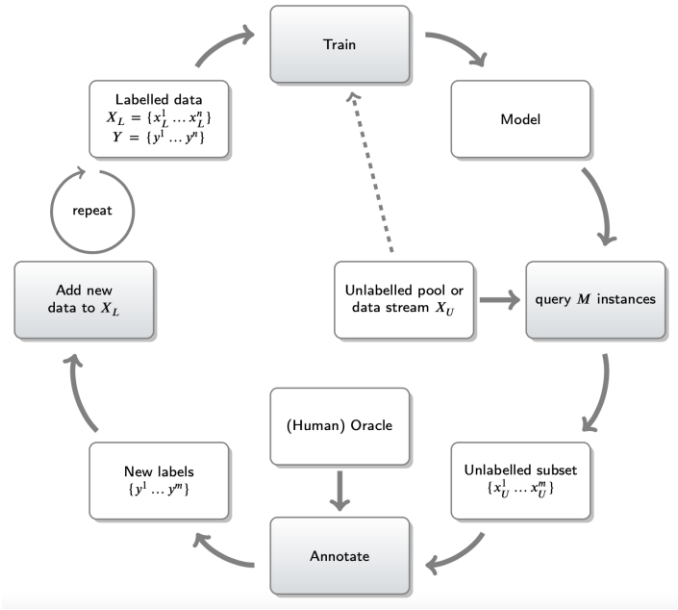
There are A LOT of works that use rehearsal, which are not mentioned here. They also come in all shapes of supervised/ unsupervised/reinforced for different modalities

5. Active Learning - Querying Future Data

Rather than asking ourselves the question what we forget when learning a new task, we will ask ourselves now: If we are in a training process on some initial data what data should we include next? Why should we be interested in this question? Let's answer this!

Active Learning chooses data instances to include continuously. Settles described it best (see introduction diagram): The key hypothesis in active learning (sometimes called "query learning") is that if the learning algorithm is allowed to choose the data from which it learns to be "curious" if you will it will perform better with less training.

A popular statement in supervised learning is that data is cheap in comparison to labels. Active Learning is also referred to as "query learning". The underlying mechanism for queries is called "acquisition function". The iterative process is shown in the next figure.



Active Learning with (unlabelled) data pools can be huge. Not every data point is equally informative. This is typically referred to as "pool based" Active Learning. We typically accumulate data (into the labelled dataset) after selection (querying M instances). Here is how Pool Based Active Learning works:

Given: Labeled set $\mathcal{L}$, unlabeled pool $\mathcal{U}$, query strategy $\phi(\cdot)$, query batch size B

repeat

  // learn a model using the current $\mathcal{L}$

  $\theta = \text{train}(\mathcal{L})$;

  for $b = 1$ **to** B **do**

    // query the most informative instance

    $\mathbf{x}_b^* = \arg\max_{\mathbf{x} \in \mathcal{U}} \phi(\mathbf{x})$;

    // move the labeled query from $\mathcal{U}$ *to* $\mathcal{L}$

    $\mathcal{L} = \mathcal{L} \cup \langle \mathbf{x}_b^*, \text{label}(\mathbf{x}_b^*) \rangle$;

    $\mathcal{U} = \mathcal{U} - \mathbf{x}_b^*$;

  end

until *some stopping criterion* ;

Algorithm 1: Pool-based active learning.

5.1. What assumptions could me make about the set-up?

Looking at the (pool-based) Active learning process from above there are many (non-exhaustive) potential assumptions we can make:

- Pool of data entirely available upfront
- Typically accumulate data after selection
- One data element at a time vs. batches
- Queries only allowed to be based on training of already available data
- Re-train model on new dataset vs. continued training?
- Oracle: infallible versus noisy

5.2. Acquisition's function: what techniques can you think of?

The acquisition function selects the data points from the unlabelled pool that is expected to provide the most valuable information. There are 3 techniques: Version Space reduction, uncertainty & heuristics, core sets& representation learning.

Version space reduction is a more formal approach where we reduce the set/space of possible hypotheses (a hypothesis is a function that maps our inputs to the outputs)

$$h: X \to Y$$

by removing the ones that are inconsistent with the data.

Uncertainty and heuristics are a perhaps intuitive approach is to use the predictions, or maybe even better, uncertainty in the predictions for the queries.

Core sets and representation learning is a distribution-based approach is maximizing distribution coverage instead of reducing the possible set of hypotheses (version space) explicitly. Basically, a subset that represents the data well!

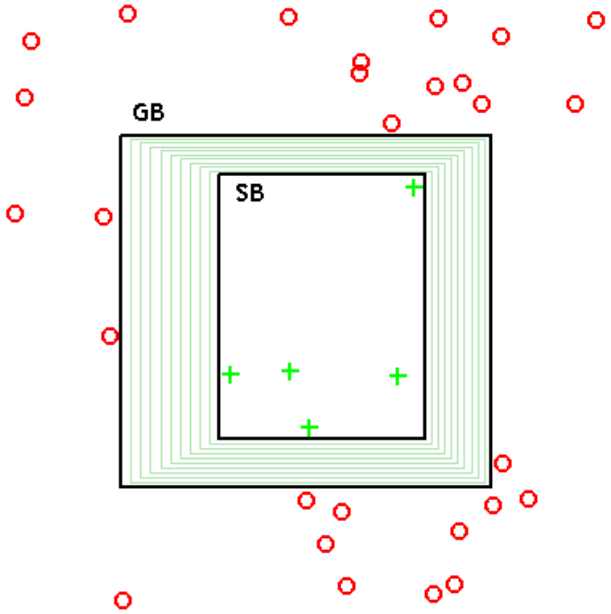
The next question to ask is if we should use discriminative or generative models?

Discriminative models (models are designed to model the conditional probability distribution of the output variable given the input variables) could allow for natural ways to assess "novelty" of a new example but be cautious about overconfidence phenomena (recall lecture 1, topic in upcoming lecture) .

Generative models (model the joint probability distribution of both the input and output variables) could allow to reason about the data distribution, but our parameters only reflect the distribution seen so far! Unless we use the entire pool! We will see that the choice between discriminative and generative models also heavily depends on the set-up assumptions!

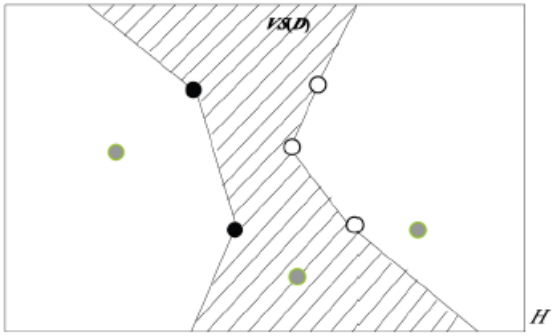
5.3. Version Space

Assume that there exist hypotheses consistent with the labelled data points $h: X \rightarrow Y$, then the version space of the data points is $VS(D) = \{h \in H | cons(h, D)\}$. h must be consistent with the dataset D. Let's look at a visual explanation of the version space:

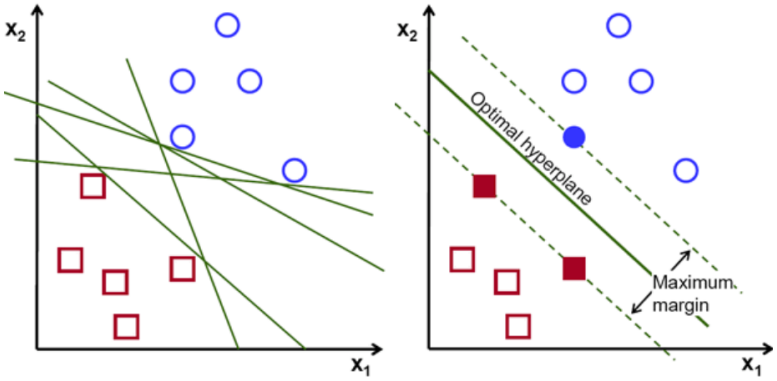


- *A specific hypothesis (SB)*: cover positive examples & as little remaining feature space as possible
- *General hypotheses (GB):* cover positive examples & as much of the remaining feature space as possible
- *Version space:* represented as green rectangles

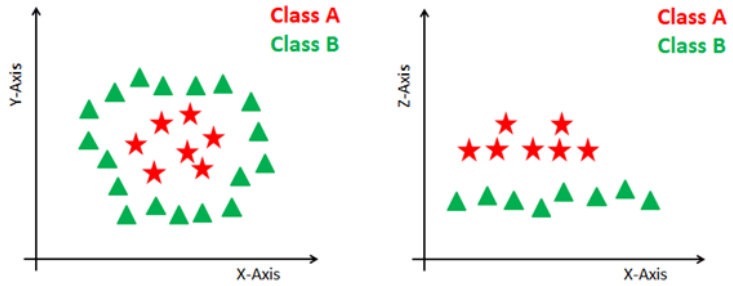
Now we could query such that the version space $VS(D) = \{h \in H | cons(h, D)\}$: ,i.e., the set of consistent hypotheses, quickly gets reduced.



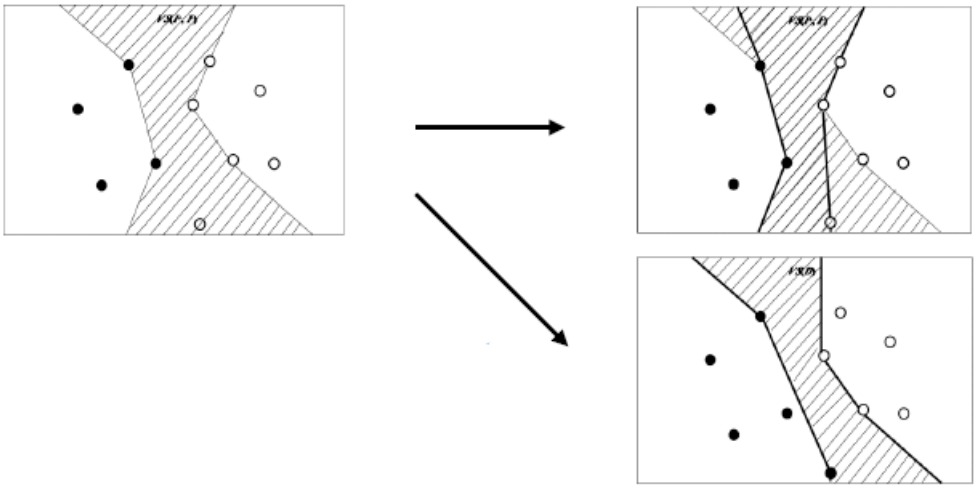
But first a quick recap to Support Vector Machines (SVMs). Choose hyperplane that maximizes the margin to closest to the instances (the support vectors) that divides data points into the two classes $(1, -1)$.



Data is not always linearly separable. We can project data to a (higher dimensional) feature space through kernel functions. An example are polar coordinates.



Now back to version spaces. The version space can be thought of a set of hyperplanes (or could be redefined through vectors W). In active learning we would query the bottom white point because its forces us to readjust our hyperplane. This allows us to rapidly reduce the version space. We intuitively choose successive queries that halve the version space. However, there are various approximations: Is version space symmetric? Estimates of the size? etc.



Version space is hard to do in practice or it is hard to get good results! Version spaces & expected error reduction can be complicated (& computationally heavy).

5.4. Uncertainty & Heuristics

Reducing the set of consistent hypotheses does not regard the evaluation metric. We could also look at the machine learning loss and query data points that would most reduce the expected error and most change the current model. This is called First-order Markov action learning: "First-order Markov active learning aims to select a query x^* such that when the query is given label y^* and added to the training set, the learner trained on the resulting set $D^+(x^*, y^*)$ has lower error than any other x".

- *Heuristic Techniques:* Minimum confidence, Maximum entropy, best versus second best
- *Uncertainty Techniques:* Model uncertainty (output variability), Ensembles/query by committee

Minimum Confidence

This technique involves selecting data points for labelling that the model is least confident about. In other words, you choose samples where the model's predicted class probability is close to 0.5, indicating uncertainty. These samples are expected to provide the most informative feedback to improve the model's accuracy.

Version spaces & expected error reduction can be complicated (& computationally heavy). Simple heuristics are thus still popular, especially in deep learning. The simplest algorithm for uncertainty sampling with a single classifier (which is the query function):

1. Create an initial classifier
2. While teacher is willing to label examples
   (a) Apply the current classifier to each unlabeled example
   (b) Find the b examples for which the classifier is least certain of class membership
   (c) Have the teacher label the subsample of b examples
   (d) Train a new classifier on all labeled examples

Maximum Entropy

Instead of pure output confidence, we could resort to more information theoretic approaches Example: maximize expected information gain by querying examples with largest entropy (as a measure of disorder, related to information gain). By selecting data points with the maximum entropy, you aim to choose samples where the model is most uncertain, as these are likely to be the most informative.

$$H(p) = -\sum_{i}^{c} p_i \log_2(p_i)$$

Example p(y|x):

- H[1.0, 0.0, 0.0, 0.0, 0.0] = 0
- H[0.2, 0.2, 0.2, 0.2, 0.2] = 1

Best versus second best

Confidence & entropy can be poor estimates when multiple classes are considered, because not all classes are equally important for our predictions. This technique involves comparing the predicted class probabilities for different classes. Data points where the model is uncertain and the margin between the best and second-best class probabilities is small are prioritized for labelling. This focuses on areas where the model is less confident in distinguishing between classes.

Exploration vs. Exploitation

When the task isn't binary classification, we also need to care about exploration versus exploitation. How much do we explore very novel classes and how much do we extend knowledge of classes we have already seen? Our measures often overemphasize "novelty"

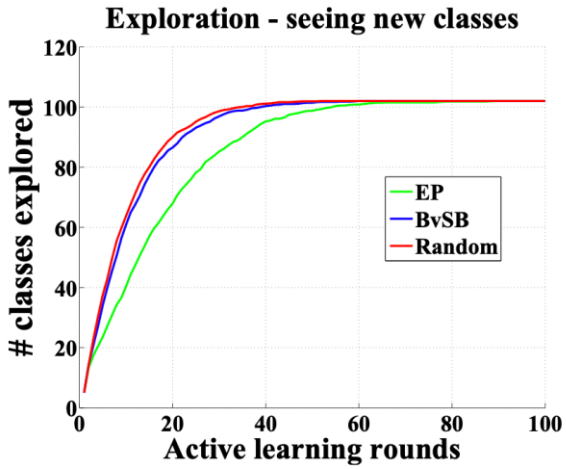


Figure 6. Space exploration of active selection – BvSB-based selection is almost as good as random exploration, while the former achieves much higher classification accuracy than random.

Can we correct entropy alone? We could weigh entropy with some measure of data similarity, to get “information density”:

$$ID(x) = -\sum_{\hat{y}} p(\hat{y}\,|\,x;\theta)\,\log p(\hat{y}\,|\,x;\theta) \cdot \frac{1}{U}\left[\sum_{u} \mathtt{sim}(x, x^{(u)})\right]^{\beta}$$

Where beta is a weighting & the similarity over all unlabelled examples U could be a distance

$$\mathtt{sim}_{cos}(x, x^{(u)}) = \frac{\vec{x} \cdot \vec{x}^{(u)}}{||\vec{x}|| \times ||\vec{x}^{(u)}||}$$

Ensembles/Query by committee

We could also maximize the information gain between two/multiple models: ensembles. Could also be interpreted as reducing the version space across models or gauging uncertainty.

In this approach, multiple machine learning models (an ensemble or committee) are trained and used. Each member of the committee may have different predictions for the same data point. Data points for which the committee members disagree the most are selected for labelling. This disagreement indicates uncertainty, and labelling such samples can help improve the model.

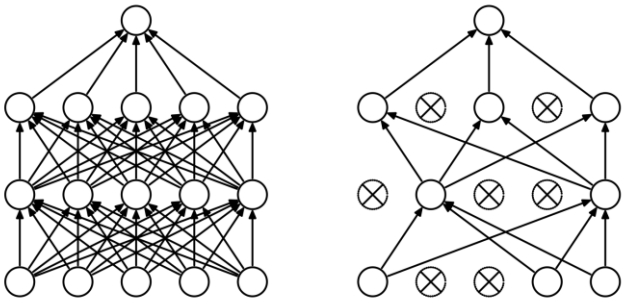
Query by a committee of two
Repeat the following until n queries have been accepted

1. Draw an unlabeled input $\boldsymbol{x} \in X$ at random from $\mathcal{D}$.
2. Select two hypotheses h_1, h_2 from the posterior distribution. In other words, pick two hypotheses that are consistent with the labeled examples seen so far.
3. If $h_1(\boldsymbol{x}) \neq h_2(\boldsymbol{x})$ then query the teacher for the label of $\boldsymbol{x}$, and add it to the training set.

Model uncertainty (output variability) – Monte Carlo Dropout

This technique considers the variability or spread in the model's output predictions. Samples with higher output variability are selected for labelling because they represent areas where the model's predictions are less consistent and, therefore, more uncertain.

Make use of dropout: randomly turning off units in a model. Bayesian interpretation: Bernoulli distribution on the parameters. Do stochastic forward passes to assess variation in predictions (model uncertainty).

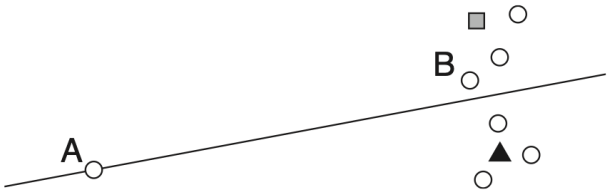


(a) Standard Neural Net (b) After applying dropout.

MCD could be useful as an approximation to using multiple model-based ensembles. The acquisition function could still be entropy, standard deviation in output confidence etc.

Limits of uncertainty sampling

Here is an illustration of when uncertainty sampling can be a poor strategy for classification. Shaded polygons represent labelled instances ($\mathcal{L}$) and circles represent unlabelled instances (U). Since A is on the decision boundary, it will be queried as the most uncertain, However, querying B is likely to result in more information about the data as a whole!

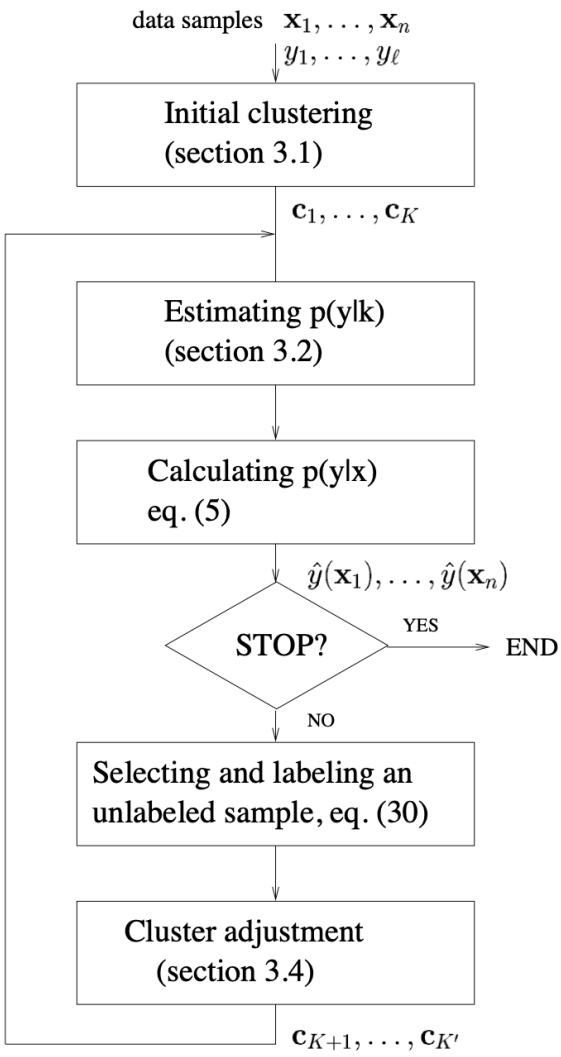


5.5. Core Sets & Representation Learning

What if we allow to use and even train on the unlabelled pool: “cover the distribution”? Assumption: a “teacher” information source is allowed, like a generative model. We wouldn’t necessarily get a lot of advantage of generative models in active learning, unless we also train on the unlabelled pool: in close relation to semi-supervised learning. We could then also make use of core sets, as discussed for rehearsal in the last lecture

We could now try to:

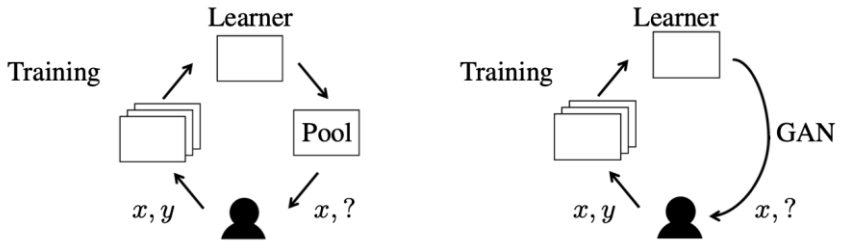
1. Pre-cluster our unlabelled data pool
2. Compute core sets of the unlabelled data pool
3. Learn a generative model & representations on the unlabelled data pool



Generative adversarial active learning

As one example of a family of approaches of how to use a generator: “query-synthesizing” •

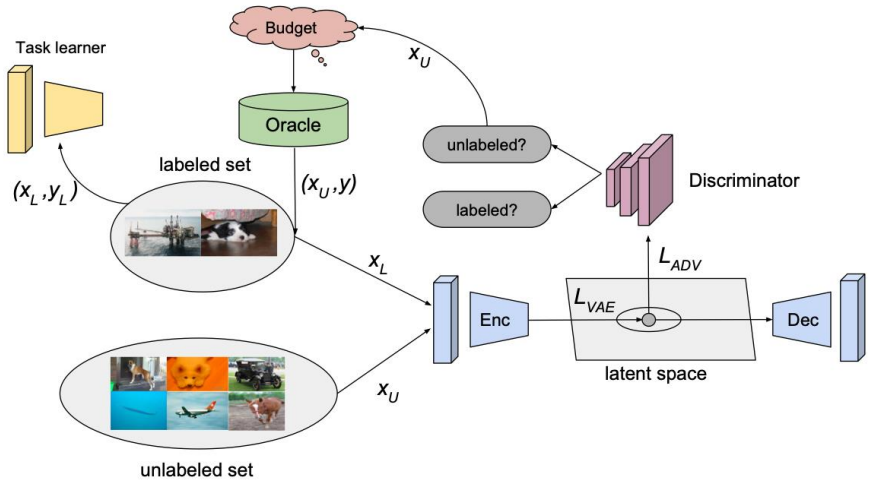
1. Let generative model interpolate/ synthesize “novel” data to label and learn actively
2. Various follow-ups



(a) Pool-based (b) GAAL

Variational adversarial active learning

Optimize on all data. Learn a discriminator on latent space to distinguish labelled/ unlabelled Adversarial: try to fool into believing everything is labelled. Query according to unlabelled/ labelled confidence.

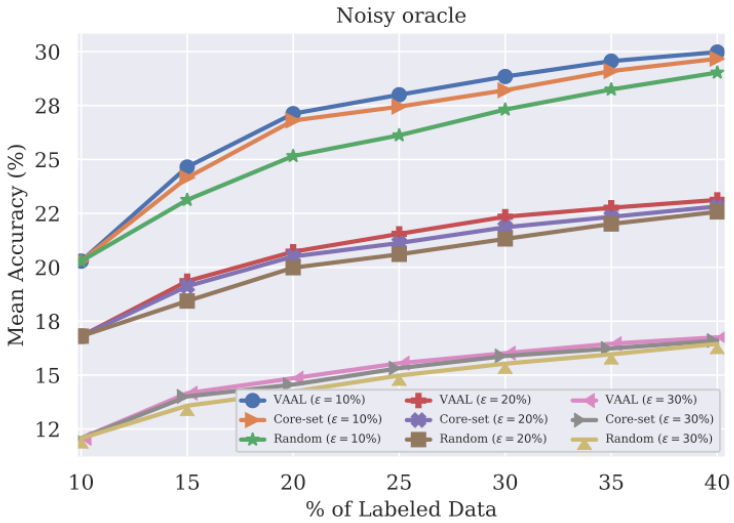


5.6. Summary

Technique	**Some of their Assumptions**
Version space reduction	Set of hypotheses is clear
Minimum confidence	No overconfidence phenomenon and out-of-distribution/ task data
Maximum entropy	
Best versus second best	
Model “uncertainty” (output variability)	Accurate uncertainty everywhere
Ensembles/query by committee	Training of multiple models
Representation learning on the pool	Upfront training on entire pool (access + computational expense)
Core sets	

More general assumptions

- *Oracle is infallible:* the teacher/labeler does not make mistakes!
- *Data is accumulated*: no “continual active learning”
- *Pool belongs to task*: we will cover this in our lecture on “learning and the unknown”



6. Dynamic/Modular Neural Architectures

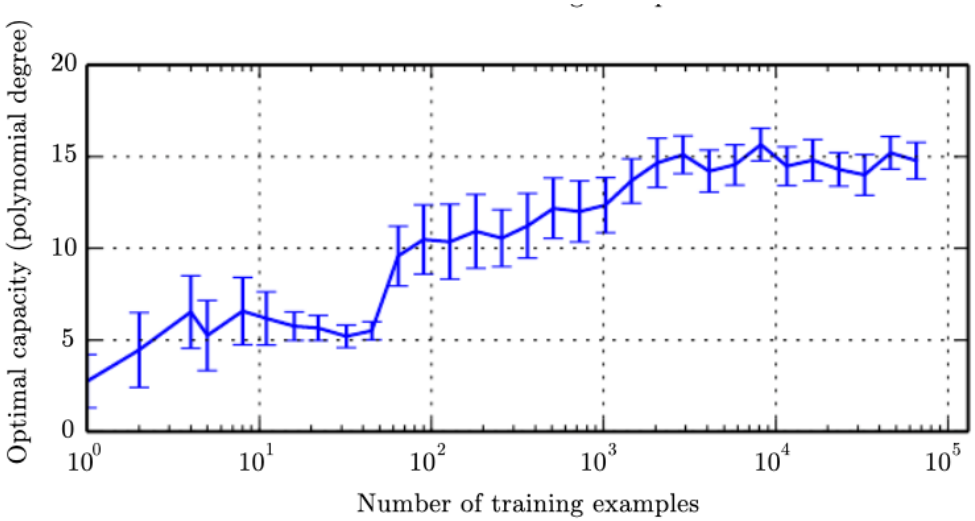
We have investigated ways to mitigate (catastrophic) forgetting but haven't talked about modularity-based methods which define hard boundaries to separate task specific parameters (often accompanied by shared parameters to allow transfer)!

We basically either use task specific masks in an overparameterized model or grow/expand the architecture. Disclaimer: we will focus primarily on neural networks today!

6.1. Why dynamic architecture?

Why are we talking about dynamic/modular architectures at this point? Catastrophic forgetting is a direct consequence of the overlap of distributed representations and can be reduced by reducing this overlap! 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!

But it's not only about catastrophic forgetting: it's also finding suitable capacity!



There are two common ways to think about modular architectures:

- Implicit: over-parametrized and try to create specific sub-modules
- Explicit: add actual parameters/capacity over time

6.2. The implicit perspective

Recall the regularization perspective: identify important parameters and constrain those. We could assume over-parametrization and try to "sparsify" our parameters. We create *sub-models* that are primarily responsible for a specific task.

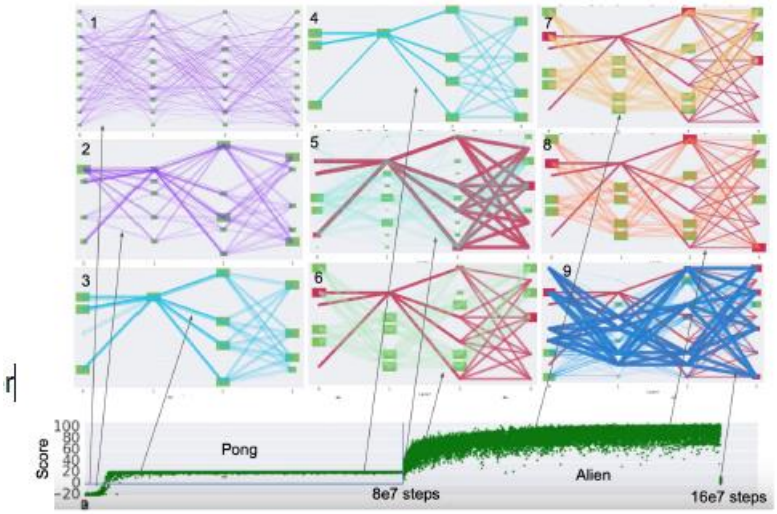
Activation Sharpening

Increase activation of some k nodes, decrease that of others . Suggestion, overlap as a sum of the smaller activations, the "shared" activation as a measure of inference! Example:

Four hidden unit example: (0.2, 0.1, 0.9, 0.1) & (0.2, 0.0, 1.0, 0.2)
Activation overlap: (0.2 + 0.0 + 0.9 + 0.1) / 4 = 0.3
A non interfering example: (1, 0, 0, 0) & (0, 0, 1, 0) have 0 overlap

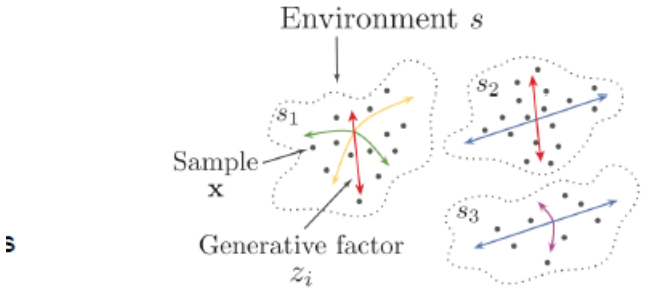
Pathways/PathNets

1. Start with an over-parametrized model
2. Constrain a task to use a subset of parameters
3. Enforce a small/fixed number of active modules/"paths"



Variational Autoencoder with Shared Embeddings (VASE)

1. Keep (over-parametrized) encoder/decoder fixed in terms of number of parameters
2. Progressively increase latent space capacity in continual learning



$$\underbrace{\mathbb{E}_{\mathbf{z}^s \sim q_\phi(\cdot|\mathbf{x}^s)}[-\log p_\theta(\mathbf{x} \mid \mathbf{z}^s, s)]}_{\text{Reconstruction error}} + \gamma \Big| \underbrace{\mathbb{KL}(q_\phi(\mathbf{z}^s|\mathbf{x}^s)||p(\mathbf{z}))}_{\text{Representation capacity}} - \underbrace{C}_{\text{Target}} \Big|^2$$

Other methods

There are many ways to go about task specific subsets of parameters/modules:

- Activation overlap
- Parameter sparsity (e.g., through L1 regularization)
- “Attention” masks

Surely interesting & useful, but what if we don’t want to start large/over-parametrized?

6.3. The explicit perspective

Recall lecture 2 on transfer learning: some features are more transferable than others. The “experts” approach: We could share parts and add individual experts on top!

Positive thing is that this is a solid and somewhat “safe” approach. The “Backbone” is static, and experts don’t share all knowledge (is this a + or -?). The negative thing is it can be tough to determine which expert to use!

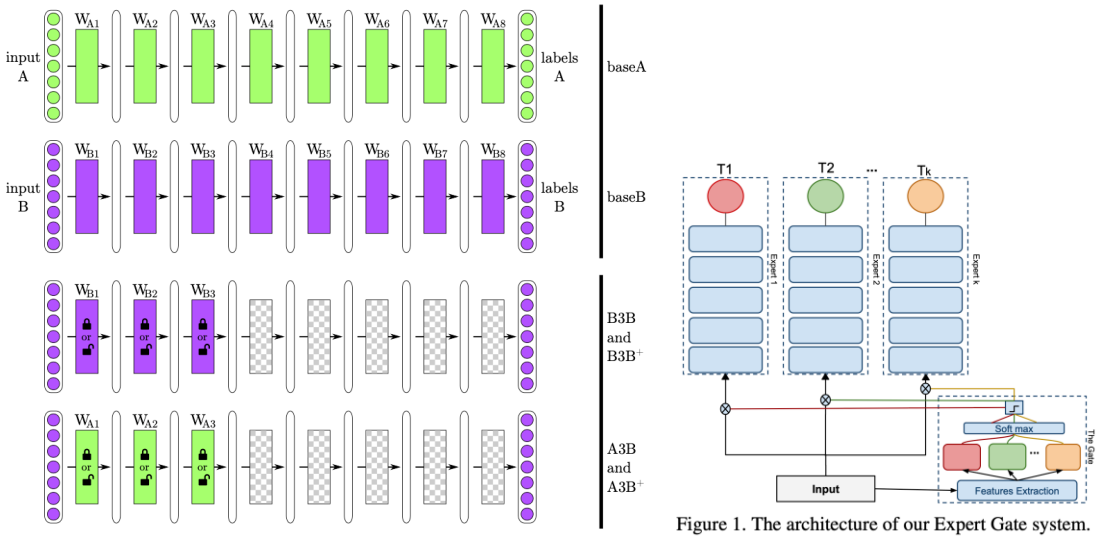
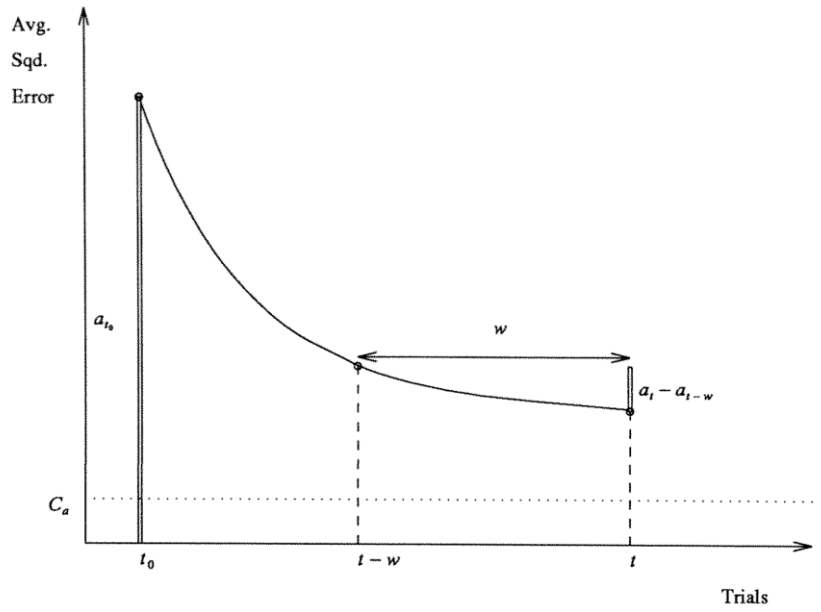


Figure 1. The architecture of our Expert Gate system.

Dynamic Node Creation

Small initial number of parameters.



- First crucial question: When should we add?
  - Assumes decaying exponential for error
  - Add node when error plateaus

- Second crucial question: when do we stop?
  - Calculate the ratio over the drop in average (squared) error (a) across some window (w) of time (t)
  - Stop when relative improvement becomes too small:

    $$\frac{a_t - a_{t-w}}{a_{t_0}} < \Delta_T$$

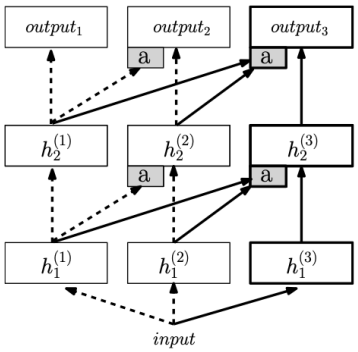
  - Stop when acceptable performance/cutoff (C) is reached:

    $$a_t \leq C_a$$

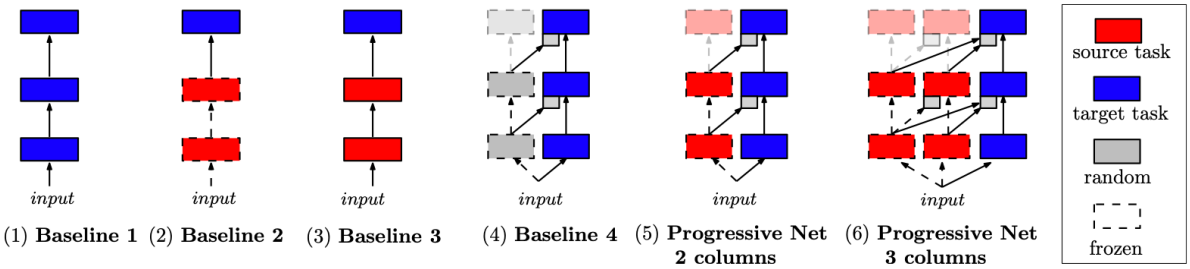
- Technically, third crucial question (not taken into account here): how/what do we add?
  - Do we add one parameter or many?
  - A neural network layer?
  - Do we add a whole new function?
  - A different output head if our tasks change?

Progressive Networks

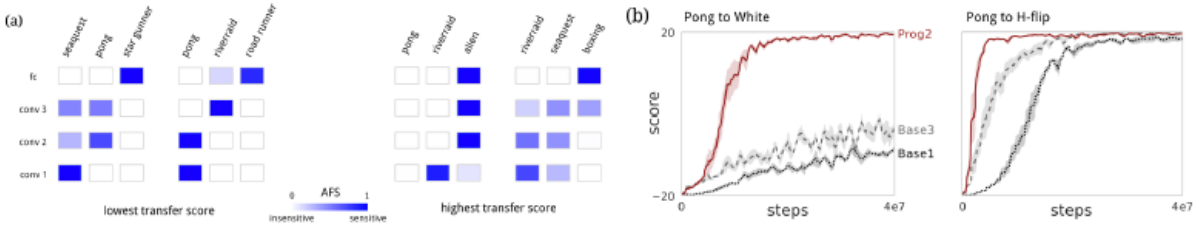
1. Start with a single “column” of parameters.
2. Add “column” for new task and freeze old column.
3. New columns receive lateral connections from old ones
4. Avoid forgetting and allow transfer where possible



We can evaluate and analyse similarly to what we have already seen in lecture 2, when we talked about knowledge transfer.



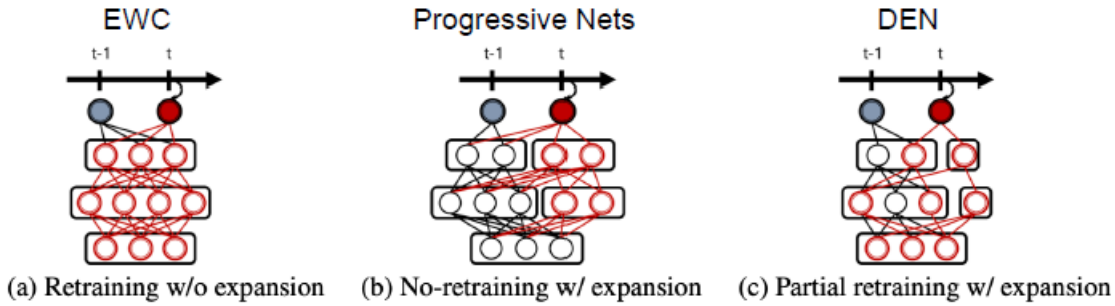
(1) **Baseline 1** (2) **Baseline 2** (3) **Baseline 3** (4) **Baseline 4** (5) **Progressive Net 2 columns** (6) **Progressive Net 3 columns**



Aren't some of these solutions "obvious"? While many of the individual ingredients used in progressive nets can be found in the literature, their combination and use in solving complex sequences of tasks is novel. Recall questions: what to start with, when to add/remove, what, how, how much, when to stop etc. Developing concrete algorithms and applications is challenging.

Dynamically Expandable Nets

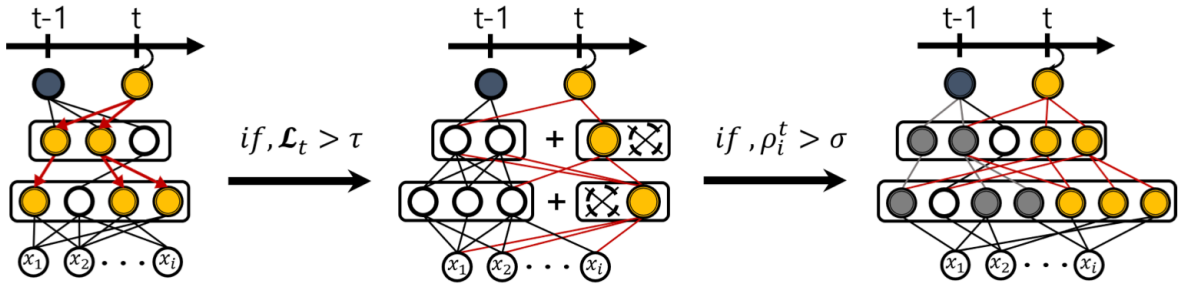
Various combinations with partial re-training with expansion



Three key steps:

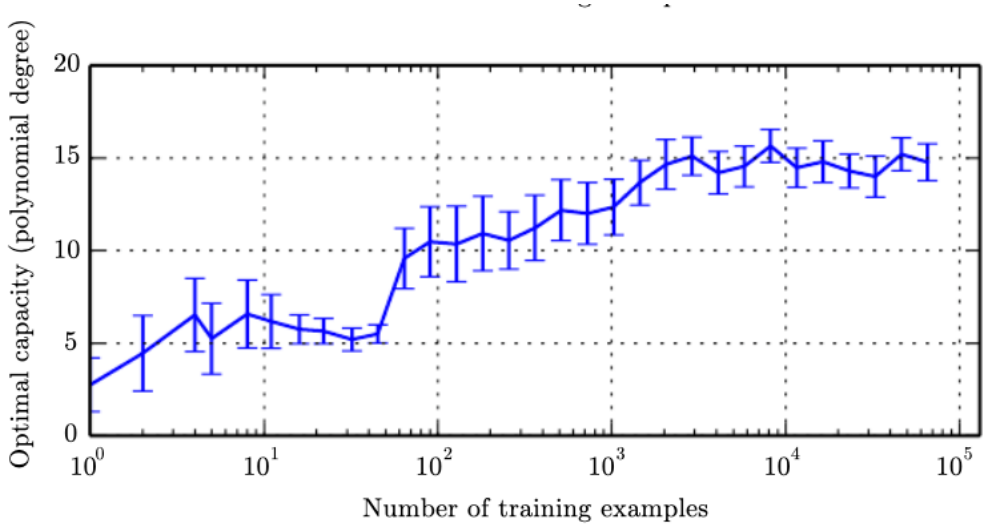
1. Selective retraining
2. Dynamic expansion
3. Split & duplicate units

Perhaps sidelines the question of how much to add by removing again!



6.4. Why is the efficacy of these approaches hard to interpret? Beyond measuring (catastrophic) forgetting

Recall lecture 1 on static ML, but it's not only about catastrophic forgetting: it's also finding suitable capacity

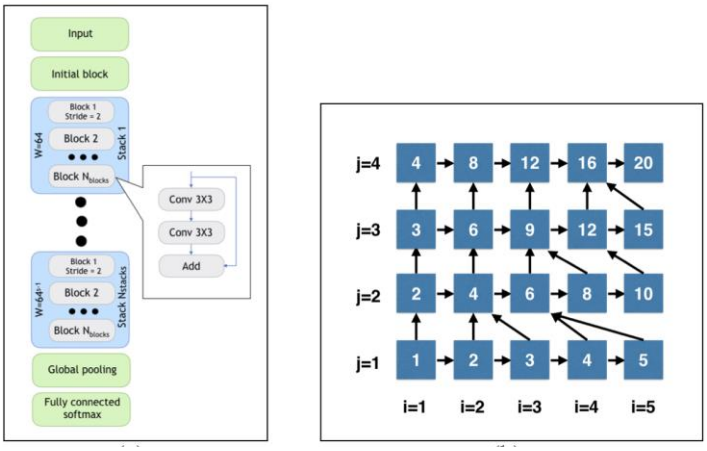


Architecture and active learning

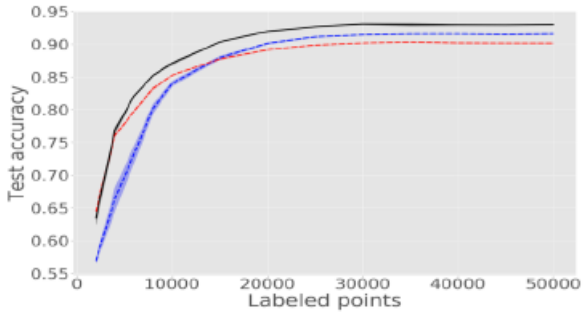
The active learning perspective. Incremental architecture approach: For every query, evaluate three architecture choices:

1. The present architecture
2. One with expanded width
3. One that also adds layers

Greedily select the best candidate in terms of a validation dataset



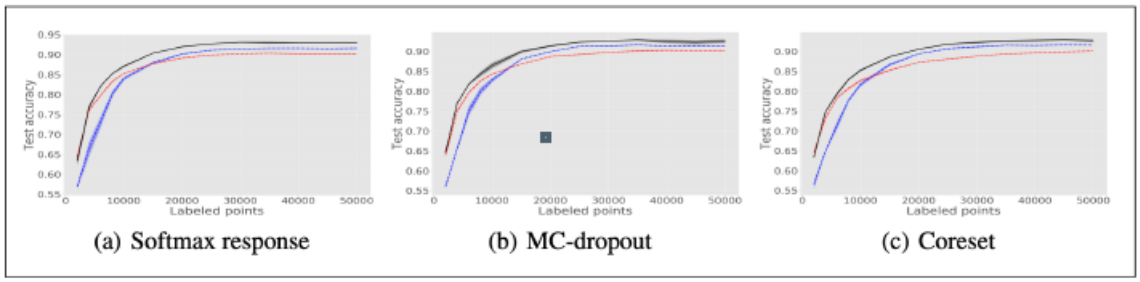
What kind of architecture do you think is depicted in the 3 curves?



(a) Softmax response

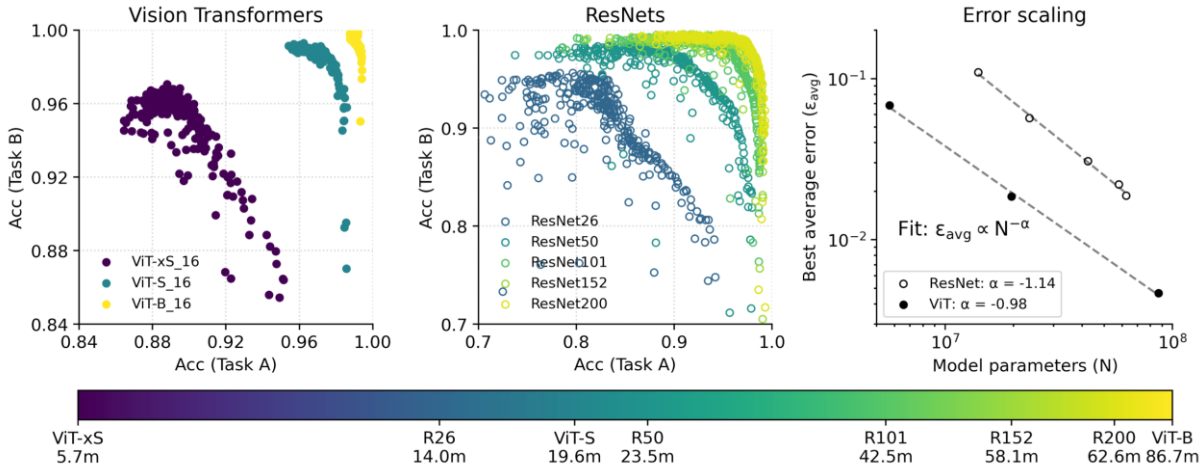
- Black line: incremental architecture
- Blue line: fixed Resnet (large)
- Red line: fixed small architecture (start of the incremental one

Consistent for different active learning acquisition functions

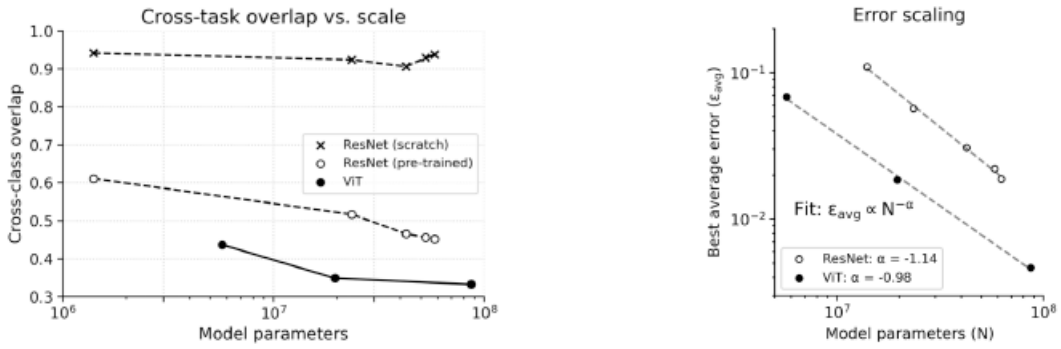


Choice of model and scale

As always: it's likely even more complicated. We don't have a solid idea of representation overlap in deep learning yet!



Some models may be more suitable than others: orthogonal representations?



Meta Learning

There are other ways to think about suitable architecture configurations. The meta-learning perspective: learning to learn

- Learning to choose a suitable model variant
- Learning to grow
- Architecture search
- Learning loss functions

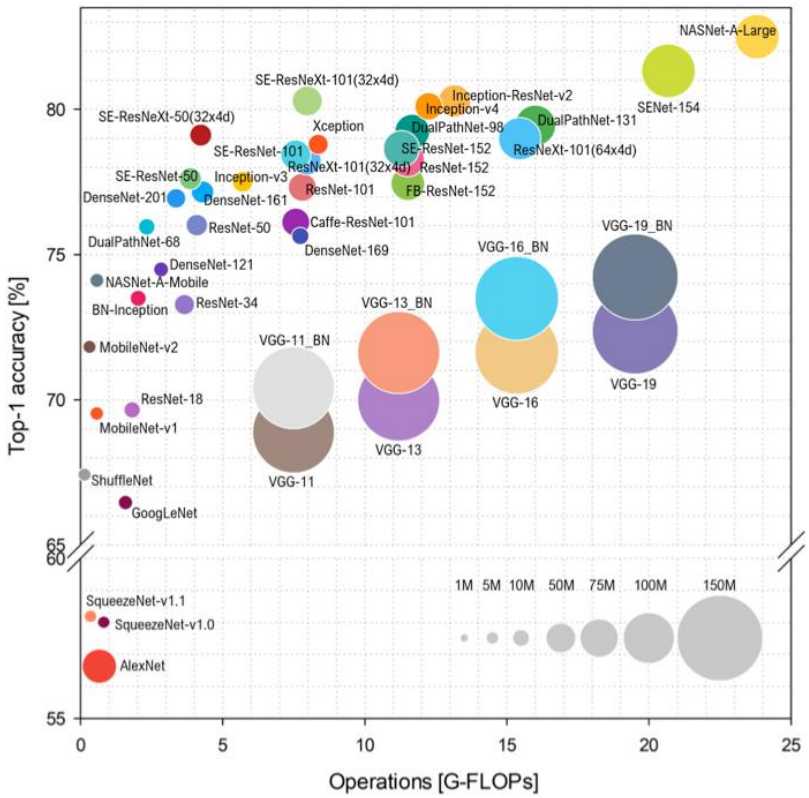
- Learning optimizers

7. Evaluation

7.1. Why is evaluation challenging in machine learning?

There is a reproducibility crisis in Machine Learning. For example, in deep RL both intrinsic (e.g., environment properties) and extrinsic sources (e.g., hyperparameters) on non - determinism can contribute to difficulties in reproducing baseline algorithms.

Even in “static” scenarios (model/data) we have many aspects of variation/interest like fair comparisons, statistical significance, exhaustive and factual reporting. Research incentives (misaligned?), code, data, assets, accessibility etc. also play a role in evaluation. For example, here is a benchmark Analysis of Representative Deep Neural Network Architectures:



7.2. Dimensions of evaluation in continual/lifelong learning

What where some of the sequences of tasks we have seen so far?

- A sequence of datasets
- Sequences of classes (from known datasets)
- Sequentially querying the instances of datasets
- Sequences of games (in RL), or languages etc.
- Sequences of the same task with shifting distribution

Benchmarks are commonly based on popular vision datasets, language datasets, or reinforcement tasks (such as games). For now, let’s assume that we know the sequence of tasks, i.e., a dedicated test set for each “experience/task” exists. So, the boundary of these

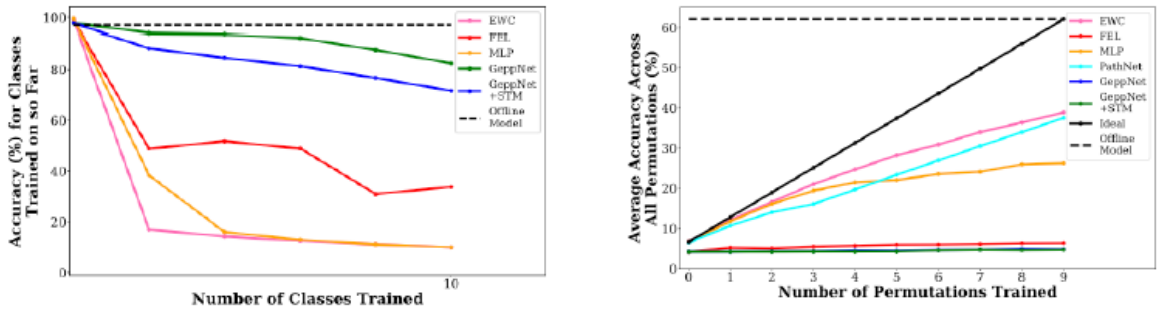
task is clear! Depending on the choice of the continual learning method we will likely be interested in different mechanism.

- *Rehearsal methods:* Original data amount, generated data, (constant?) memory size, computational expense etc.

- *Regularization methods:* Regularization strength (hyper-parameters), memory expense, computational expense etc.

- *Architecture/parameter methods*: Number of parameters, number of models, expert heads, memory expense, computational expense etc.

The important thing is now that the final average losses seem insufficient. Let's look at some further suggestions. Do we care about the overall performance? Or the one up to the current point in time? We care about both! Here is an example for measuring catastrophic forgetting in Neural Networks:



Per task measures

Okay here are some measures that can be used:

- "Base" loss: the initial (an old) task after i new experiences → Measures retention/forgetting

$$\Omega_{base} = \frac{1}{T-1}\sum_{i=2}^{T}\frac{\alpha_{base,i}}{\alpha_{ideal}}$$

- "New" loss: the newest task only → Measure ability to encode new tasks

$$\Omega_{new} = \frac{1}{T-1}\sum_{i=2}^{T}\alpha_{new,i}$$

- "All" loss: average up to the present point in time → Measure present overall performance

$$\Omega_{all} = \frac{1}{T-1}\sum_{i=2}^{T}\frac{\alpha_{all,i}}{\alpha_{ideal}}$$

- "Ideal" loss: offline value trained at once → Measure achievable baseline

Definition Forgetting

We can define forgetting for a particular task (or label) as the difference between the maximum knowledge gained about the task throughout the learning process in the past and the knowledge the model currently has about it.

Definition Intransigence

We can also define *intransigence* as the inability of a model to learn new tasks. Since we wish to quantify the *inability* to learn, we compare to the standard classification model which has access to all the datasets at all times.

Measures: Forward and Backward Transfer, B-Shot Performance and LCA

- (Avg.) Forward transfer (with random baseline b) measures the influence of a learning task on future tasks.

- (Avg.) Backward transfer measures the influence of a task on previous tasks (negative = forgetting, positive = retrospective improvement)

- (Avg.) b-shot performance (b = mini-batch number) measures the performance after the model has been trained on all tasks T

- Learning Curve Area (LCA) at beta is the area of the convergence curve Z as a function of b in [0, beta]. Beta = 0 is zero-shot performance == Forward transfer. It measures both the speed and stability of learning.

Memory, Size, and Compute

We can construct similar measures for memory, size and compute:

- Computational Efficiency: Quantifies add/multiply ops (inference & updates)
- Model Size Efficiency: Quantifies parameter growth
- Sample Storage Size Efficiency: Quantifies stored amount of data (for rehearsal)

However, there are plenty of other interesting ideas of what to measure!

7.3. Why evaluation is even more challenging in continual/lifelong learning

What should we report now? How do we compare and draw conclusions with various metrics and set-ups we have seen in the last section? Let's look at some examples:

Model	Dataset	Data Permutation			Incremental Class			Multi-Modal			Memory Constraints	Model Size (MB)
		Ω_{base}	Ω_{new}	Ω_{all}	Ω_{base}	Ω_{new}	Ω_{all}	Ω_{base}	Ω_{new}	Ω_{all}		
MLP	**MNIST**	0.434	0.996	0.702	0.060	1.000	0.181	N/A	N/A	N/A	Fixed-size	1.91
	CUB	0.488	0.917	0.635	0.020	1.000	0.031	0.327	0.412	0.610		4.24
	AS	0.186	0.957	0.446	0.016	1.000	0.044	0.197	0.609	0.589		2.85
EWC	**MNIST**	0.437	0.992	0.746	0.001	1.000	0.133	N/A	N/A	N/A	Fixed-size	3.83
	CUB	0.765	0.869	0.762	0.105	0.000	0.094	0.944	0.369	0.872		8.48
	AS	0.129	0.687	0.251	0.021	0.580	0.034	1.000	0.588	0.984		5.70
PathNet	**MNIST**	0.687	0.887	0.848	N/A	N/A	N/A	N/A	N/A	N/A	New output layer for each task	2.80
	CUB	0.538	0.701	0.655	N/A	N/A	N/A	0.908	0.376	0.862		7.46
	AS	0.414	0.750	0.615	N/A	N/A	N/A	0.069	0.540	0.469		4.68
GeppNet	**MNIST**	0.912	0.242	0.364	0.960	0.824	0.922	N/A	N/A	N/A	Stores all training data	190.08
	CUB	0.606	0.029	0.145	0.628	0.640	0.585	0.156	0.010	0.089		53.48
	AS	0.897	0.170	0.343	0.984	0.458	0.947	0.913	0.005	0.461		150.38
GeppNet+STM	**MNIST**	0.892	0.212	0.326	0.919	0.599	0.824	N/A	N/A	N/A	Stores all training data	191.02
	CUB	0.615	0.020	0.142	0.727	0.232	0.626	0.031	0.329	0.026		55.94
	AS	0.820	0.041	0.219	1.007	0.355	0.920	0.829	0.005	0.418		151.92
FEL	**MNIST**	0.117	0.990	0.279	0.451	1.000	0.439	N/A	N/A	N/A	Fixed-size	4.54
	CUB	0.043	0.764	0.184	0.316	1.000	0.361	0.110	0.329	0.412		6.16
	AS	0.081	0.848	0.239	0.283	1.000	0.240	0.473	0.320	0.494		6.06

As expected, the MLP suffers from forgetting displayed by the base loss and of course is very good at the new task as displayed. If we were only looking at final/all loss it would not be clear why it has such a performance. But still, we can't compare every measure you see!

Unfortunately, it's not just about what to measure! It's about assumptions, trade-offs, benchmarks etc. Should we strive for specific benchmarks and overall consensus or transparency? So, is the crisis worse in lifelong ML? Yes

The challenge to define a task - 3 scenarios of continual learning

It's not just challenging to compare across multiple metrics, it's also challenging to agree on what tasks should be! We have 3 incremental learning types (updating a machine learning model with new data incrementally (schrittweise), rather than training it from scratch each time):

- Task-IL: Solve task so far task-ID provided (probably the best scenario)
- Domain-IL: Solve task so far task-ID not provided
- Class-IL: Solve task so far and infer task-ID provided

The challenge of expert heads

Recall the "experts" approach: We could share parts and add individual experts on top. Expert heads often evaluated from a "forgetting only" perspective. Not only test set for each experience/task, but also the task id is provided!

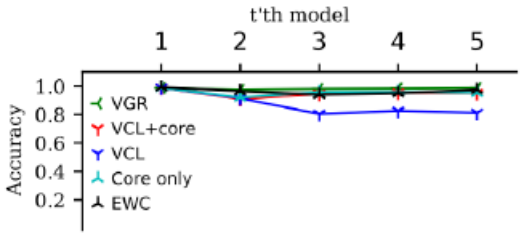


Figure 5. **Multi-headed Split FashionMNIST.**

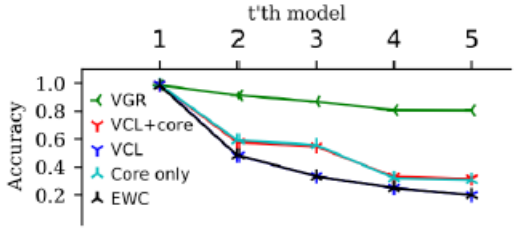


Figure 3. **Single-headed Split Fashion MNIST.**

The challenge of hyper-parameters in continual learning

There are more set-up assumptions: how do we select the continual hyper-parameters? Recall the plasticity - sensitivity trade-off and algorithms like Elastic Weight Consolidation and Synaptic Intelligence which can solve it.

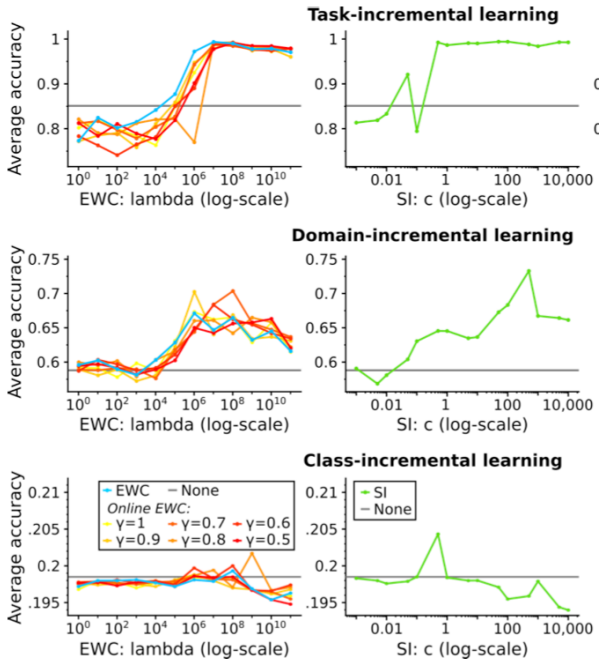


Figure D.1: Grid searches for the split MNIST task protocol. Shown

The challenge of formulating desiderata: consensus

The challenge of consensus. Is it possible to postulate general desiderata? Some suggestions:

- Cross-task resemblance
- Shared output head
- No test time task labels
- No unconstrained re-training on old tasks
- More than two tasks

And also questions like unclear task boundaries, continuous tasks, overlapping vs. disjoint tasks, long task sequences, time/compute/memory constraints, strict privacy guarantees etc. are rising.

Maybe we can use this as the desiderate of continual learning?

Property	*Definition*
Knowledge retention	The model is not prone to catastrophic forgetting.
Forward transfer	The model learns a new task while reusing knowledge acquired from previous tasks.
Backward transfer	The model achieves improved performance on previous tasks after learning a new task.
On-line learning	The model learns from a continuous data stream.
No task boundaries	The model learns without requiring neither clear task nor data boundaries.
Fixed model capacity	Memory size is constant regardless of the number of tasks and the length of a data stream.

As you can see its assumptions, assumptions, assumptions! Why are there so many possible assumptions and things to measure? Let's remind ourselves where they come from and the reason why we have waited to discuss evaluation for 7 weeks. The differences between machine learning paradigms with continuous components can be nuances. Key aspects often reside in how we evaluate. Each paradigm seems to have a particular preference (potentially neglecting other important factors). In all honesty, it is presently challenging to assess continual/lifelong learning systems.

7.4. How can we move forward?

Whether a crisis or not, there is much room for general improvement! on the incentives and presentation part, on the empirical experimentation parts and on many other fronts: assets, data, ethics etc.

Dataset sheets & Model cards

In dataset sheets we can specify motivation, composition, collection process, pre-processing, cleaning, labelling, distribution, maintenance, ethical considerations etc.

In model cards we can specify:

- model details, intended use, human-centric application intent, organization developing the model
- considerations on deployment, limits, and ethics
- descriptions of metrics, model version, license etc.

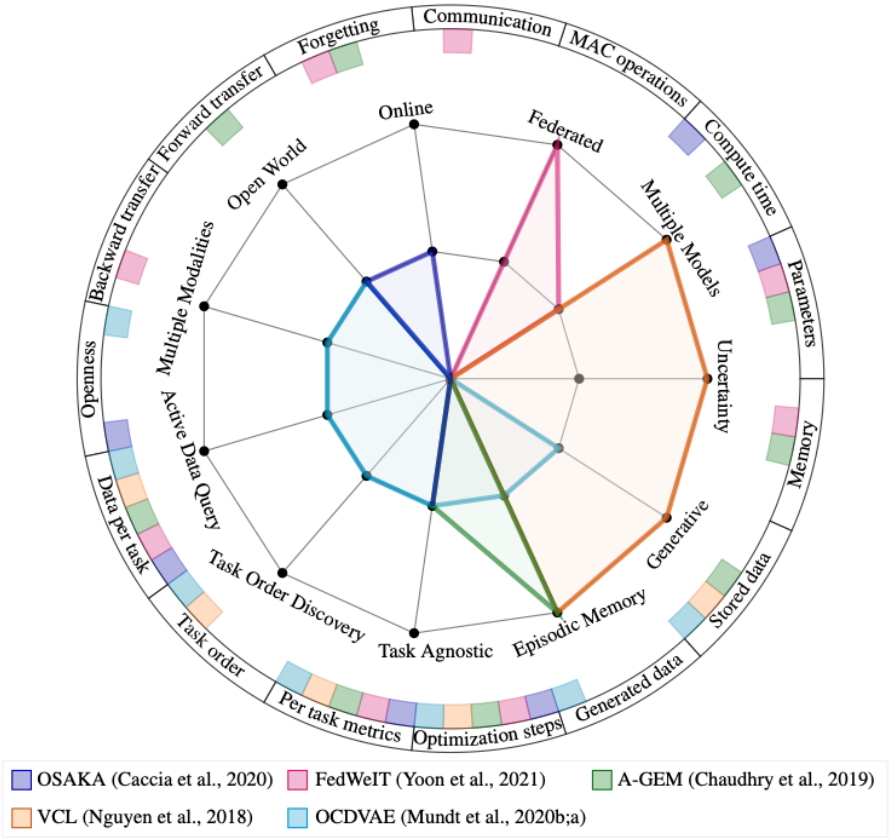
Types of Limitations

Are limitations a sign of bad research or an exercise of self-reflection? Here are some types of limitations:

- *Fidelity:* training data is label but in the real world it's not
- *Generalizability:* model does not apply to other scenarios/contexts
- *Robustness:* noise in data reduces accuracy drastically
- *Reproducibility*: researchers provide parameter settings but cannot share code or data
- Resource requirements: Specialzed hardware for computational efficiency, scale etc.
- *Value tension:* model has high accuracy on test set but it's hard to interpret (black box)
- *Vulnerability to mistakes and misuse:* system operators are liable to misinterpret results with no sufficient training

CLEVA Compass

Important note: previous efforts are largely yet to develop for continual/lifelong learning! Do distinct applications warrant the existence of numerous scenarios? Make inspiration in set-up transparent and promote comparability with CLEVA! It encourages transparency, summarizes incentives, and promotes comparability in a compact visual form:



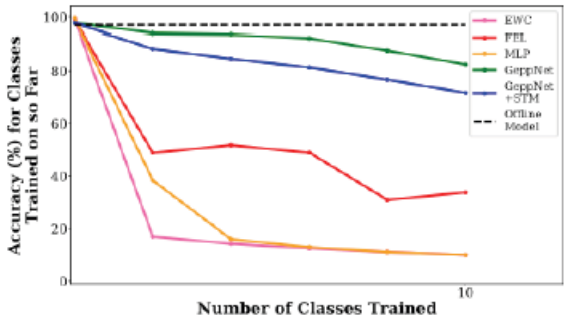
- **Inner compass level (star plot):** indicates related paradigm inspiration & continual setting configuration (assumptions)

- **Inner compass level of supervision:** “rings” on the star plot indicate presence of supervision. Importantly: supervision is individual to each dimension!

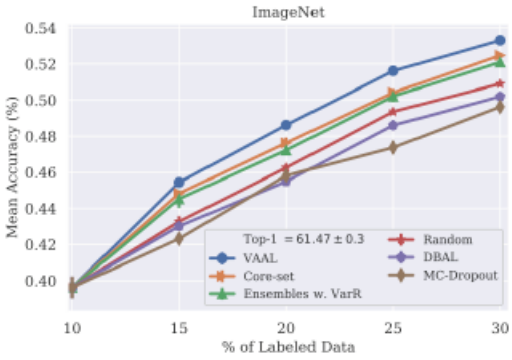
- **Outer compass level:** Contains a comprehensive set of practically reported measures. Encourages transparency, summarizes incentives, & promotes comparability in a compact visual form

8. Open world learning - learning & prediction in the presence of the unknown

We've discussed various ways to measure and many assumptions, but so far it was always clear what to test on. What if we don't know what to test on?



Kemker et al, "Measuring Catastrophic Forgetting in Neural Networks", AAAI 2018

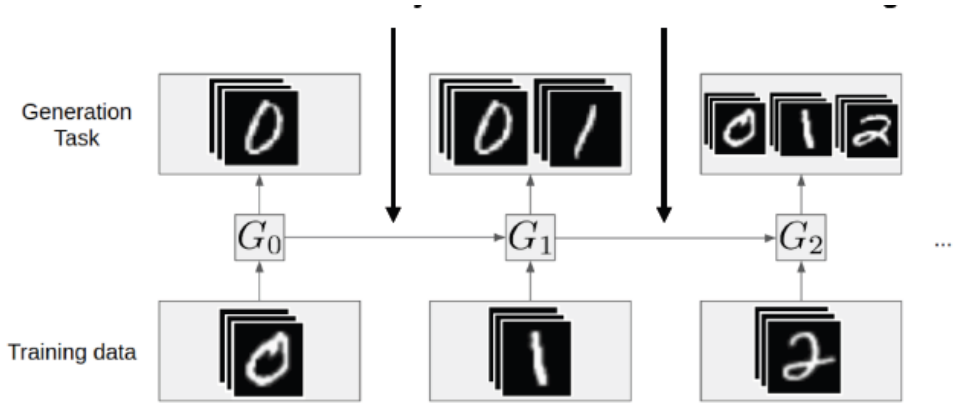


Sinha et al, "Variational Adversarial Active Learning", ICCV 2019

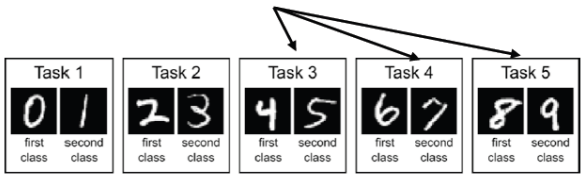
8.1. Recall the challenges

Tasks

What if we don't know the boundary and aren't constrained on our testing examples?

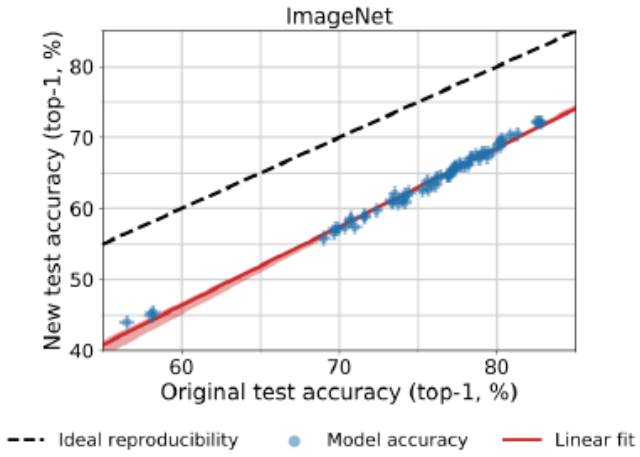


What if future or unrelated data is in the test set?



Distribution Shifts

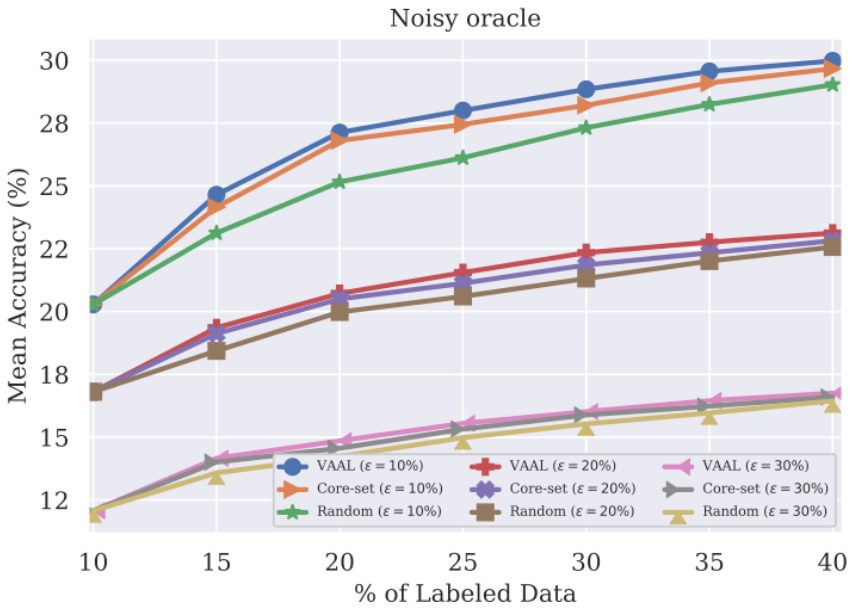
Natural data distributions are complex and can easily shift! Performance loss even happens if we recollect another "test set" with the same instructions a second time!



Noisy Oracles

Our active learning assumptions were:

- *Oracle is infallible:* the teacher/labeller does not make mistakes! Noisy Oracles are thus a problem!
- *Pool belongs to task*: we will cover this in our lecture on "learning and the unknown"



8.2. Perspectives to address these challenges

Its more than known vs unknown. Let's address some perspectives:

- *Known knowns (or simply knowns):* Examples belong to the distribution from training set was drawn. Assumption of an accurate and confident prediction.
    - Example: Model trained to identify dog breeds correctly labels a Labrador.

- *Known unknowns*: Unknown examples where models are not confident, or uncertainty is high. Can be optionally "negatively" labelled examples used in training.
    - Example: Model encounters an exotic bird and returns low confidence.

- *Unknown unknowns:* Unseen instances belonging to unexplored and unknown data distributions. Predictions generally overconfident and by definition false.
    - Example: Model confidently labels a drawing of a mythical creature as an actual animal.

- Unknown knowns: Usually not considered! We know the concept but choose to treat it as unknown (wilful ignorance?) or our ML system cannot represent the concept plus structure altogether.
    - Example: Model can't identify handwritten digits despite understanding numerical concepts.

8.3. Three types of approaches to solve the challenges

What do you think: how can we solve our challenge? We have 3 types of approaches:

- *Anomalies in predictions:* The unsuspecting angle, where out-of-distribution are hopefully separable through anomalous output values.

- *Incorporating prior knowledge*: The intuitive idea to include "background" or "non-example" data population explicitly.

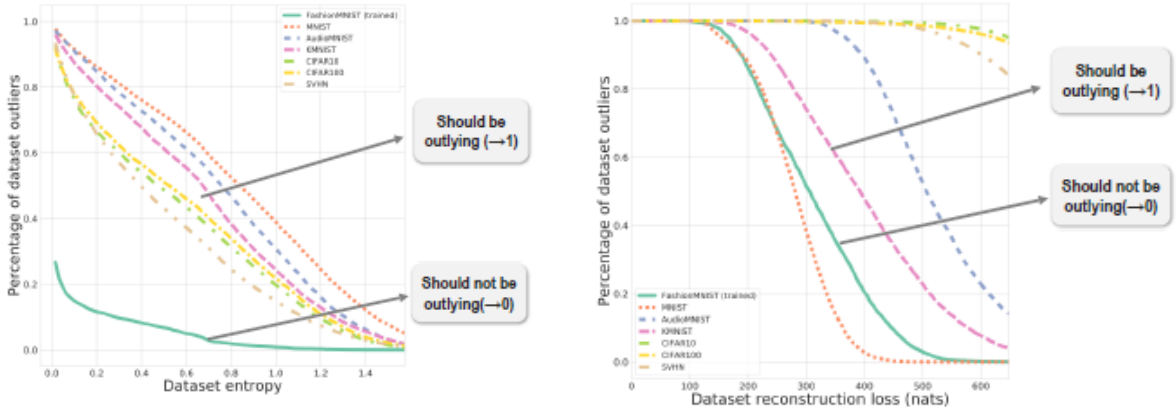
- *Open Set recognition:* The more formal approach ensures that we only rely on predictions from our "covered space"; we create bounds.

Predictive anomalies: the unfortunate part of the story

Recall lecture 1 about *overconfidence* and the quantitative example:

1. Train a neural network classifier on a dataset (here fashion items)
2. Log predictions for arbitrary other datasets
3. Observe that majority of misclassifications happen with large output "probability"

Overconfidence in machine learning is when a model thinks it's more accurate than it is! Can we fix this with uncertainty? Unfortunately, uncertainty is not a necessarily a "fix" and it gets even harder when we try to select a threshold. Overconfidence is not exclusive to discriminative models but also for generative models!



This approach handles only known unknows!

Including prior knowledge: an alternative?

An intuitive idea is to incorporate everything we know that does not belong to our task(s). In essence we include background class/"non-examples" that aren't of interest. The key questions to ask are:

- How to implement the loss? (Disclaimer: possibly uncountable amount of works)
- What part of the universum is useful?
- What are we expected to see during prediction later"? (Noise? Other concepts? Etc.)

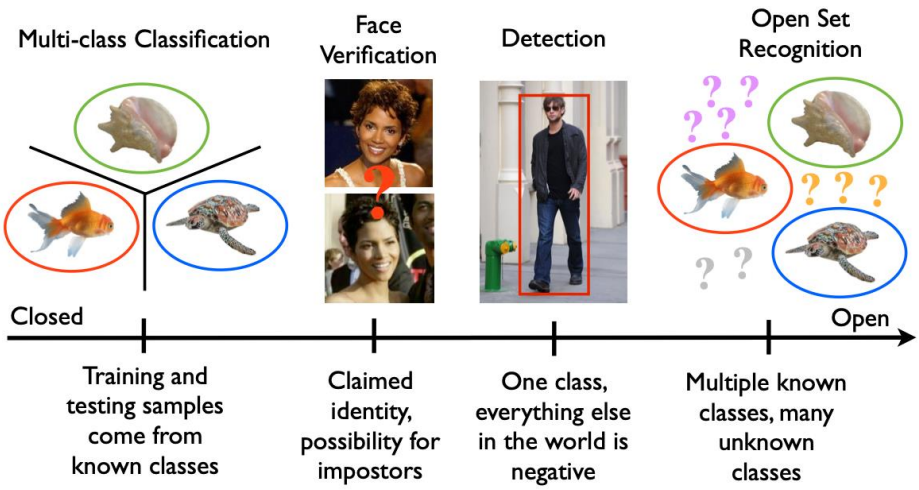
So how can we do this now?

- We could let our predictions (classifier) explicitly follow a uniform distribution for "out" data
- We could calibrate our outputs, e.g., by scaling a temperature parameter later
- We could also think about encouraging features to be zero for OOD data

This approach handles only known unknows!

Open set recognition and explicit bounds

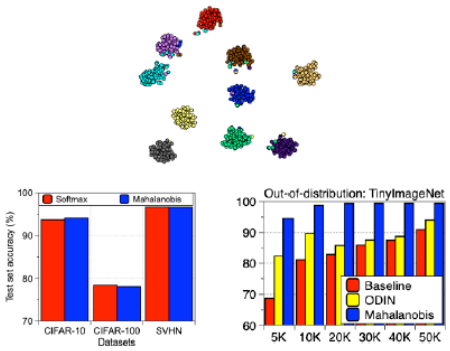
What do you think are the up and downsides so far? We may need a different approach: as the world grows more "open" we move from known unknowns to unknown unknowns. Our two perspectives only handle the former!



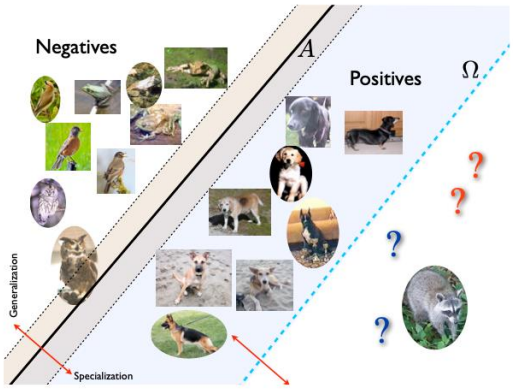
Let's start with some intuitions what open space might be:

We could take into account distances from the known data points:

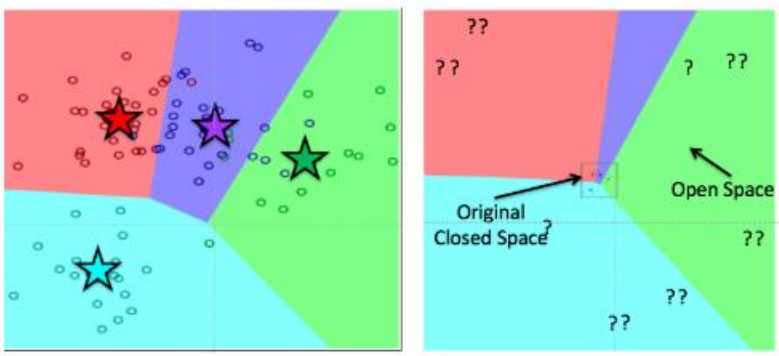
- *Example 1*: We could make assumptions like every class being Normal distributed and then calculate distances to our existing data points, e.g., Mahalanobis distance.



- *Example 2*: We could fit another parallel plane in an SVM, for a reject option, based on the support set with large distances.



We could also look at the open space which is what we have not covered with known data!

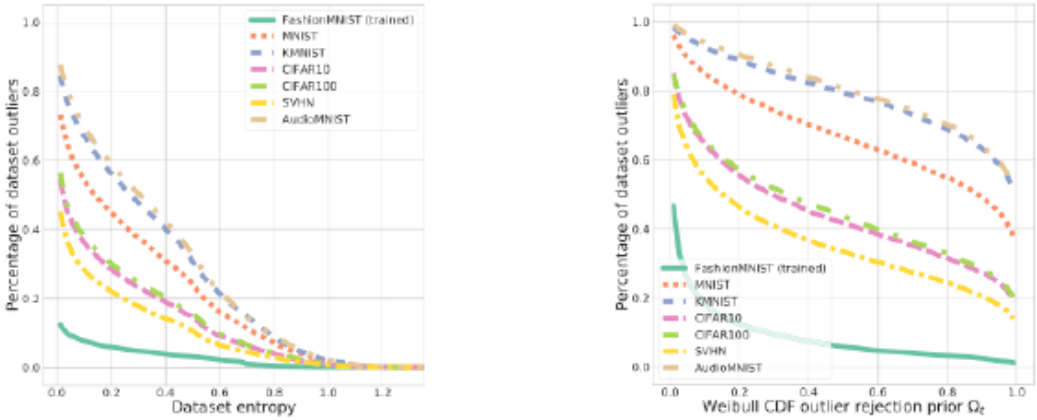


The idea is to create a radius around each training example. Within this “ball”, the model is expected to make reliable predictions. Anything outside these balls falls into the open space, where the model's predictions are less trustworthy.

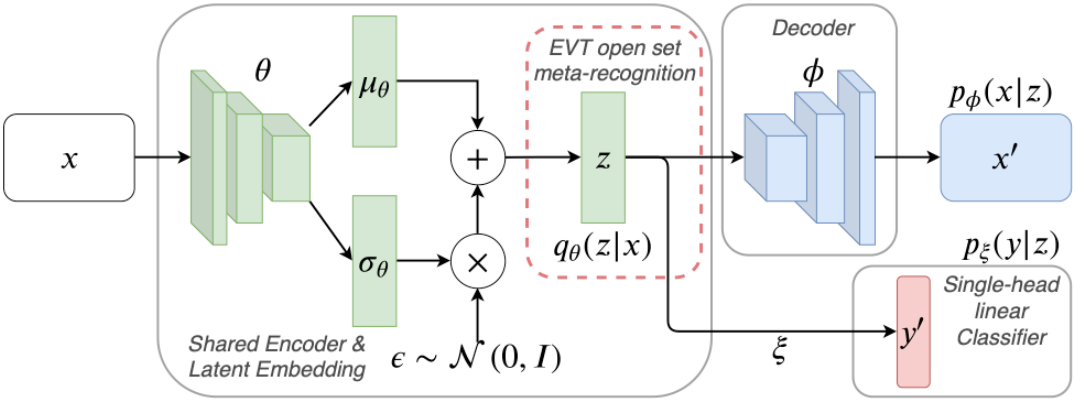
In other words, we could fit a distance-based model (following the radius idea), e.g., here based on the mean activations of training data in a deep net. But which distribution should we choose? We are mainly interested in the extreme distances, as we want to make a decision of when to reject. *Extreme value theory* may provide an answer for us.

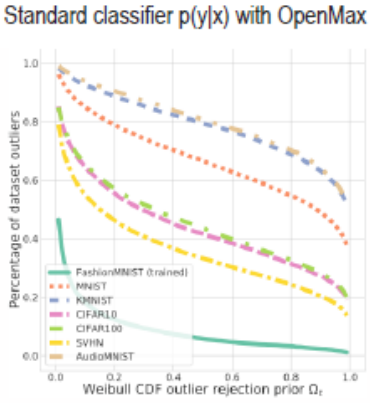
Extreme value theory is interested in the probability of events that are more extreme than any previously observed. Regardless of the overall distribution, if the data is bounded, EVT tells us that sampling the tail/the extrema away from the median of our distribution results in an EVT distribution: Weibull, Gumbel or Fréchet.

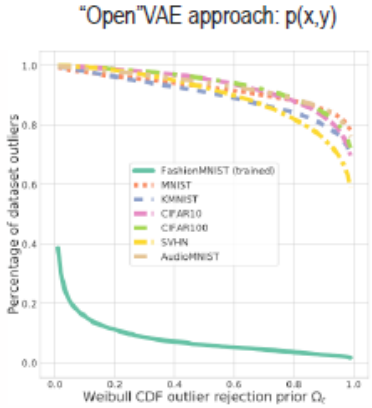
We can use the cumulative distribution function (CDF) to either reject right away, because we exceed our extremely observed distances, or use the value to modify our prediction score also referred to as OpenMax. OpenMax seem to improve a lot! But why is there still so much room for improvement?



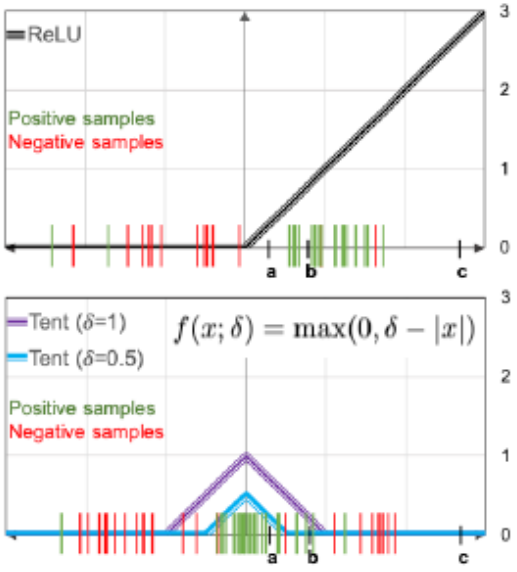
So, do we need generative models to improve? Recall earlier that overconfidence is not exclusive to discriminative models, but what if it’s only about predictive values again? We could formulate an OpenMax variant based on a VAE, based on generative factors. It may indeed be a question of the learned representations!





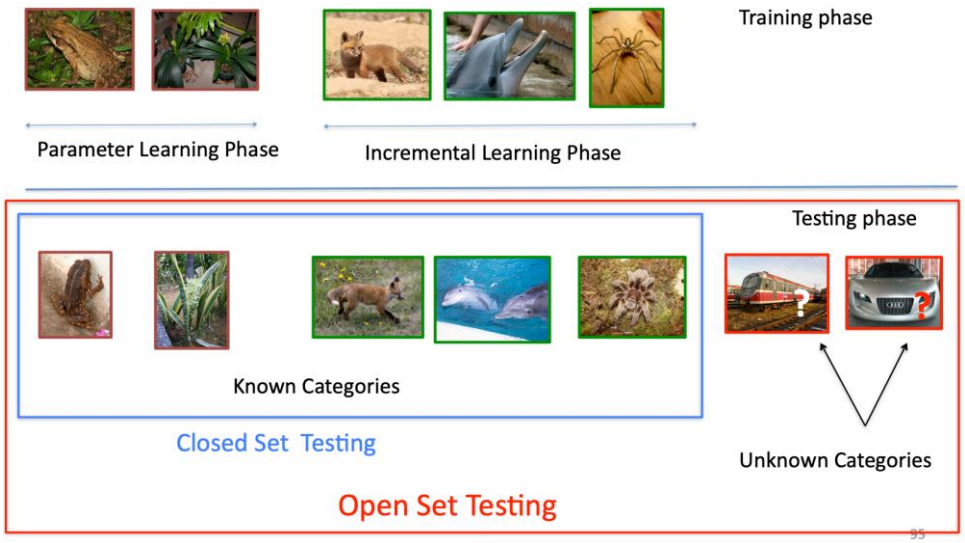


As an alternative/auxiliary approach, we could also take a direct look at the functions that we use in our model! Hypothesis: Specific functions in our ML models, like ReLU in NNs are (at least in parts) the culprit - they always produce high confidence far away from the data. Alternative idea: use functions that are bounded and try to determine their "extent" based on the observed data



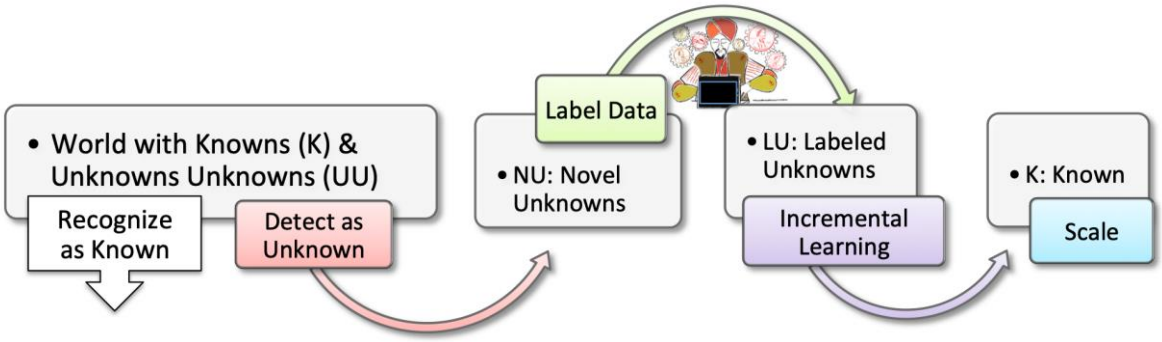
8.4. Open world learning: Combining ideas

In retrospect: although there have been increments, the types of continual learning we have seen so far were indeed in a closed world!



Open world learning tries to "puzzle together" the pieces we have seen so far**. An effective open world recognition system must efficiently perform four tasks**:

1. detect unknown
2. Choose which points to label for addition to the model (active learning)
3. label the points
4. update the model



We can try to puzzle the pieces together now. As it is very much a cutting-edge research frontier, let's talk about it more in the "frontiers" lecture.

Ending on some open questions & a disclaimer:

- Note the "towards" in many of the paper titles
- There is much to be done still: what about avoiding forgetting in addition now?
- Naturally, evaluation gets even more complicated now!
- It's no longer a question of ML algorithms, perhaps it already was systems question beforehand, but now it definitely is
- What about natural corruptions, adversarial attacks etc.?

9. Curriculum Learning

What if we don't know the boundary and aren't constrained on our testing examples? What if future or unrelated data is in the test set? In retrospect: although there have been increments, the types of continual learning we have seen so far were indeed in a closed world! Open world learning tries to "puzzle together" the pieces we have seen so far. An effective open world recognition system must efficiently perform four tasks:

1. detect unknown
2. Choose which points to label for addition to the model (active learning)
3. label the points
4. update the model

What about the order in which we learn? If we have the choice, which (identified unknown) data should we start with/include next?

9.1. Definition of Curriculum Learning

A Curriculum is a sequence of training criteria over T training steps. Each criterion Q_t includes the design for all the elements in training a machine learning model, e.g., data/tasks, model capacity, learning objective etc. Curriculum learning is the strategy that trains a model with such a curriculum.

Curriculum learning deals with the question of how to use prior knowledge about the difficulty of the training examples in order to sample each mini batch non-uniformly and thus boost the rate of learning and the accuracy.

It is based on the intuition that it boosts the rate of learning and the accuracy when the machine learning model is presented with simple concepts first. In summary, curriculum learning helps to determine the sequence in which tasks should be trained, starting with simpler tasks and progressing to more complex ones.

Let's look at an example: Ranking language model trained with vs without curriculum on Wikipedia.

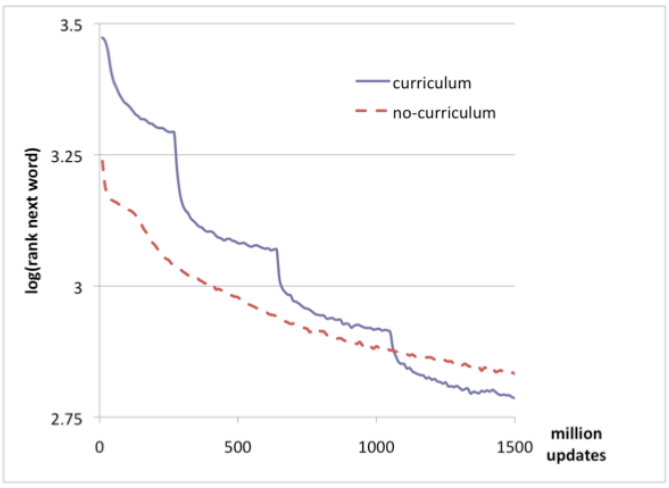


Figure 5. Ranking language model trained with vs without curriculum on Wikipedia. "Error" is log of the rank of the

The “Error” is log of the rank of the next word (within 20k-word vocabulary). The curriculum-trained model skips examples with words outside of 5k most frequent words. It then skips examples outside 10k most frequent words and so on!

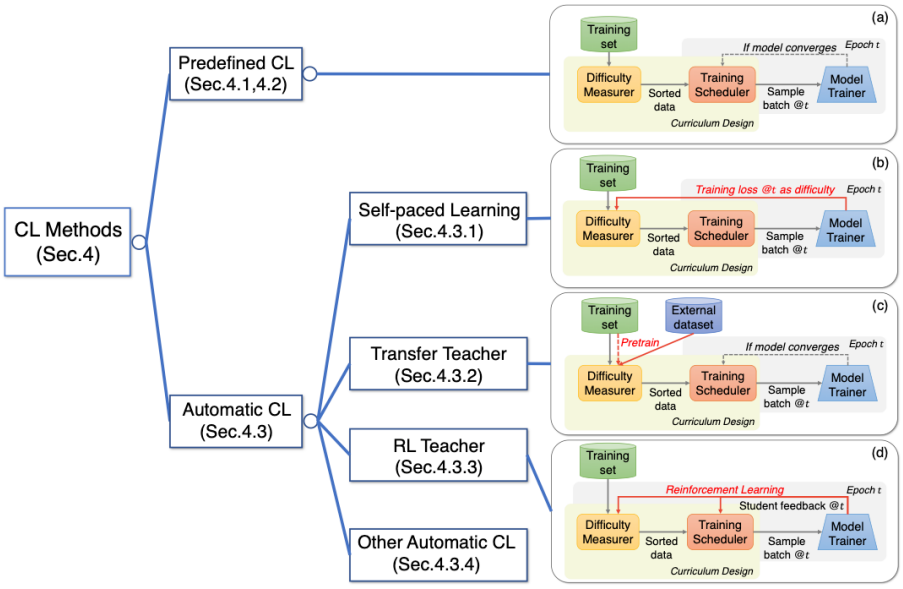
9.2. What are central questions in curriculum learning?

There are 2 key challenges:

- *Scoring function* (difficulty measurer): Any function that provides us with an estimate of the difficulty of the instances in our dataset(s).

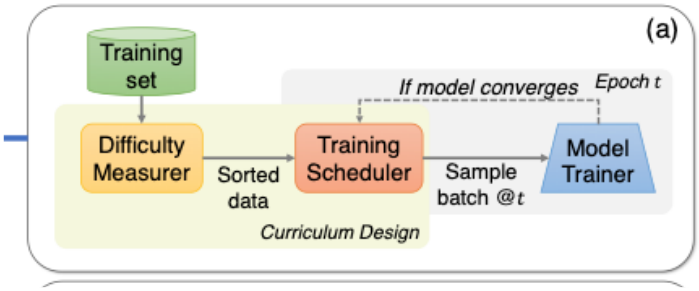
- *Pacing function* (training scheduler): sometimes also called competence, as we’ll see later. The function that tells us how to interleave samples into the training process over time.

We will now cover some curriculm methods to adress these challenges:

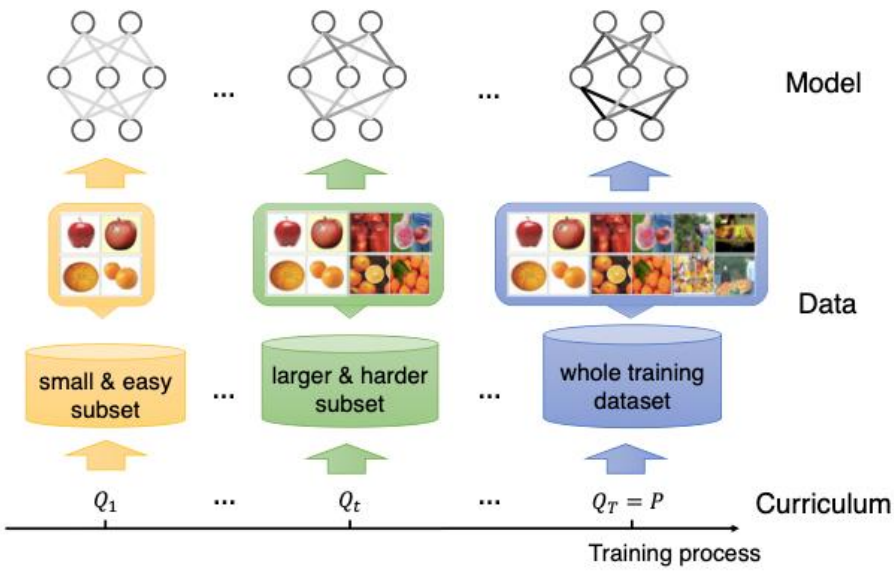


9.3. Pre-defined curriculum

Let’s start by considering a pre-defined curriculum, inspired by learning from “textbook style” content as seen in the next picture:



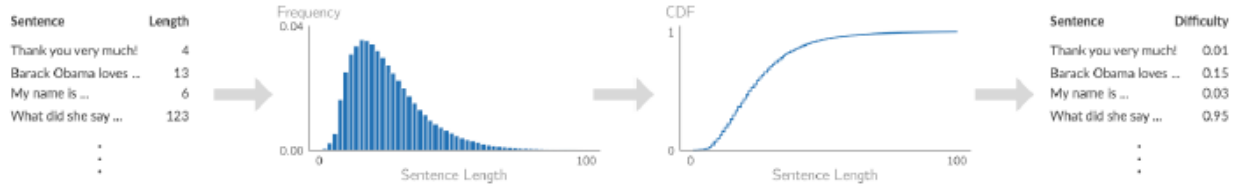
If we want to define the curriculum up-front, according to prior knowledge, then what is an easy and what is a “harder” subset/dataset?



Defining Difficulty (Scoring function)

Is difficulty task and model specific? We have already seen that *specific tasks* allow for specific definitions of difficulty. Examples:

- Natural language translation (sentence length)



- Image segmentation (entropy/clutter)

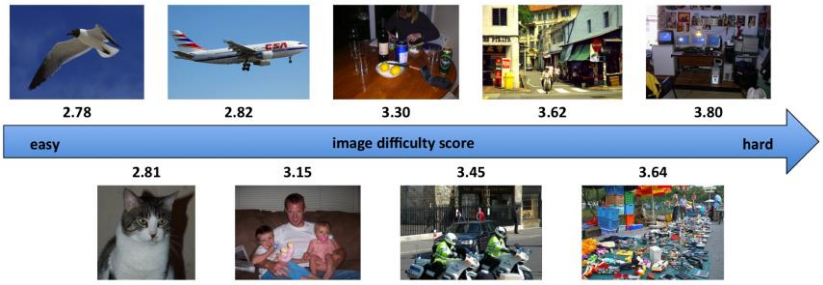


Figure 1. Images with difficulty scores predicted by our system in increasing order of their difficulty.

There are various dimensions to difficulty, not just (basic) data statistics. Especially if we think about factors that relate to what humans may find difficult:

- compositional factors (size, location)
- semantic factors (object type, scene type and depiction strength)
- context factors (unusable object-scene pair)!

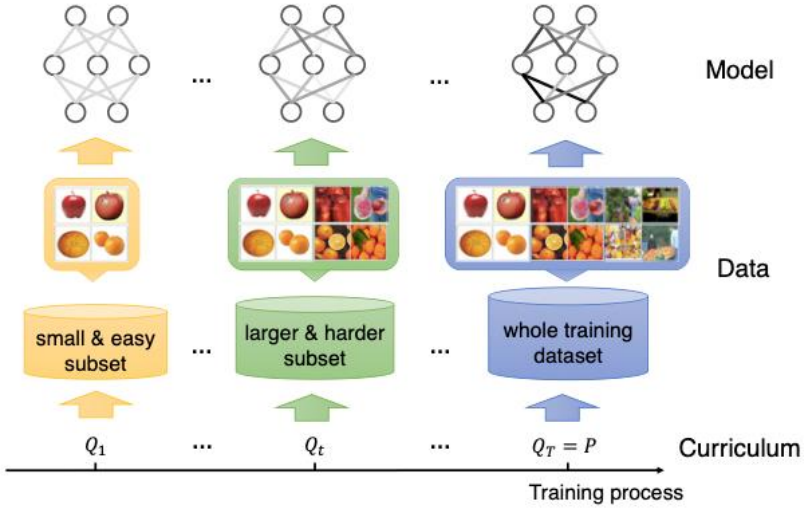
But what is difficulty for ML models and is this related to human perception? Example: human memorability and image statistics, human response times. Human difficulty and model difficulty are not necessarily the same. Various factors come into play in ML models:

- Regression example: Edge strength, number of segments, image file size
- Shallow embeddable examples seem to be learned first
- invariance to certain discriminative factors (e.g., frequencies) may exist

Assessing difficulty of data instances is interesting beyond curriculum learning. For example estimating the difficulty with respect to annotation cost!

Pacing: How to schedule the training

If we want to define the curriculum up-front, according to prior knowledge, then: When do we introduce more difficult examples?



Various options and heuristics are conceivable:

- One-Pass Curriculum: Training progresses in a single sequence from easier to more difficult tasks or data.
- Baby Steps Curriculum: Training starts with very simple tasks and *incrementally* adds complexity in small steps.
- Competence based Curriculum: The training sequence adapts based on the model's performance, focusing on areas where the model needs improvement.

However, it's not straightforward to choose, especially due to model/task dependency. Let's move away from pre-defined curriculum!

9.4. Transfer Teacher

Instead of defining the curriculum ourselves, we could use a pre-trained teacher model (based on a different related dataset) based difficulty measure.

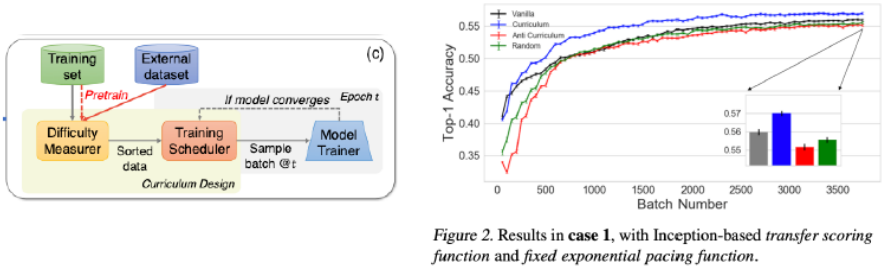
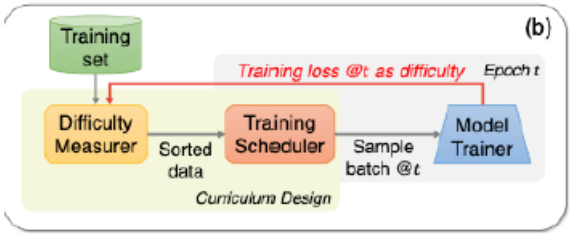


Figure 2. Results in **case 1**, with Inception-based *transfer scoring function* and *fixed exponential pacing function*.

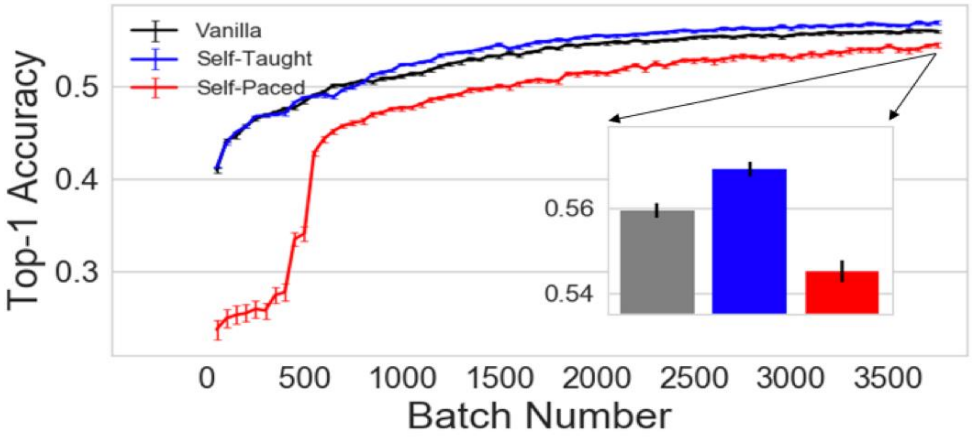
9.5. Self-paced learning

Using a teacher is still a form of pre-defined curriculum however, what if we want to have an adaptive measure of difficulty, based on our current model? Moving away from a pre-defined curriculum towards model "competence". Often this is called self-paced learning.

We now rely on the model's current hypothesis at each point in time to assign difficulty to the training instances, rather than ranking according to the target hypothesis.



Somewhat related to what we've already seen:

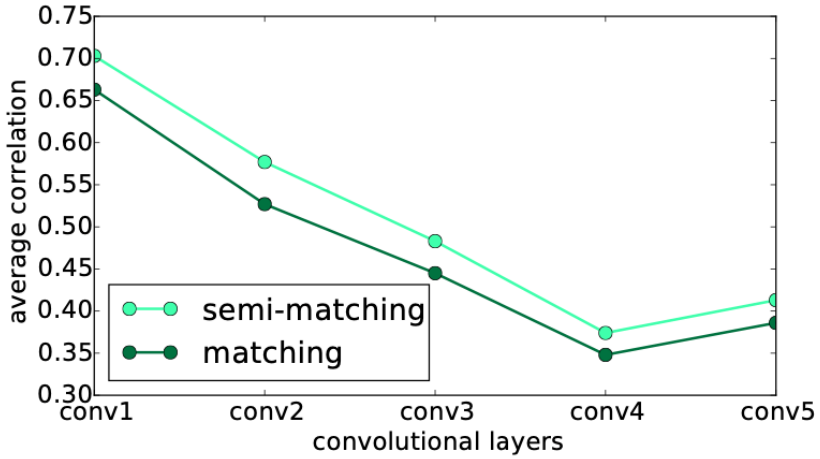


- Self-paced learning: Measure the difficulty of an instance according to current loss/predictions etc. (related to the ideas in *active learning*)

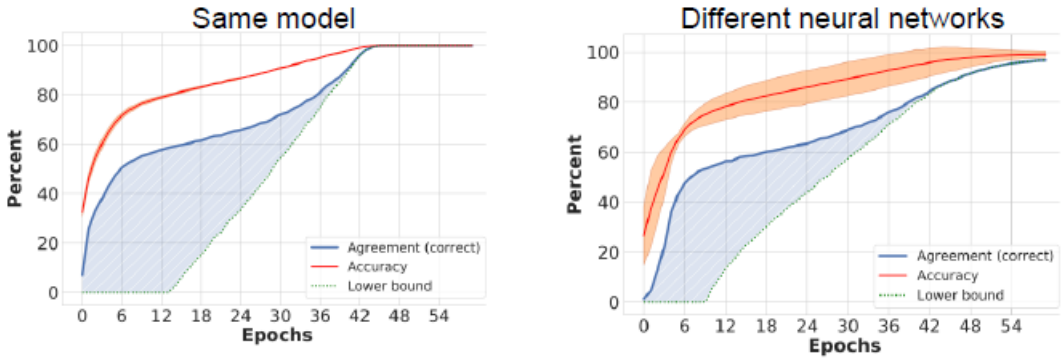
- Self-taught learning: Train a model fully, measure each instance according to final model, assign difficulty score and start over with curriculum → repeat (related to the ideas in *boosting*)

Does intrinsic ordering/pacing exist? If we can use the loss of a model as a measure of difficulty, does this perhaps mean that models "intrinsically order" examples during regular training to some degree as well? An experiment: let's train multiple models and check how similar representations are. Why is this interesting? Recall that we typically use mini batches plus stochastic gradient descent, where data is shuffled differently in every "epoch".

If we try to do a bi-partite matching of the representations in each neural network layer of different networks, there seem to exist strong correlations, especially in early, "generic" features.



Let's go a step further and train multiple models and check how much they agree on instances. Different neural networks seem to classify the same instances correctly at similar points in training: they "agree to agree"!



10. The influence and role of soft- and hardware

It is perhaps underappreciated how much machine learning frameworks shape ML research. They don't just enable machine learning research. They enable and restrict the ideas that researchers can easily explore. How many nascent ideas are crushed simply because there is no easy way to express them in a framework?

10.1. AI & ML Software Frameworks

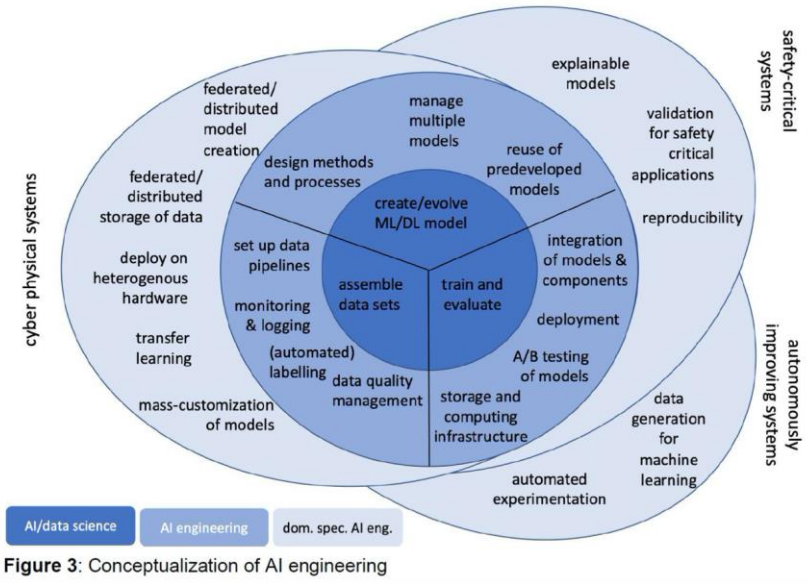


Figure 3: Conceptualization of AI engineering

Inner to outer circles are reflected in/ driven by development of software tools & hardware advances Software requirements are constantly being reshaped. Some well-known long-term ideas or key examples:

- Automatic differentiation
- Numerical optimization in natural sciences & algorithmic techniques at the heart of machine learning: expectation maximization or backpropagation
- Specific models such as neural networks decision trees and random forests date back at least as much, if not even further.
- Symbolic Programming

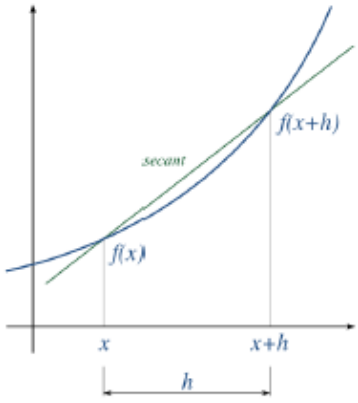
What are the enablers for the current wave? Availability of data, computational power, hardware, software tools.

10.2. Torch, Theano and the "core"

- Make differentiation easy: Theano through symbolic programming, Torch through reverse mode accumulation. Significantly facilitates numerical optimization.
- Started including code building blocks for common models such as neural network layers, logistic regression, random forests, support vector machines.
- Build on strong matrix computation backend (in C), starting to abstract away parallelization and hardware specific code from the developer to large degree. Integration with higher level programming languages such as Python or Lua.

Numerical Optimization

Pick two points and compute slope of nearby secant line through points $[x, f(x)]$ and $[x + h, f(x + h)]$. The derivative of f at x is the limit of the value of the difference quotient as the secant lines get closer to being a tangent.

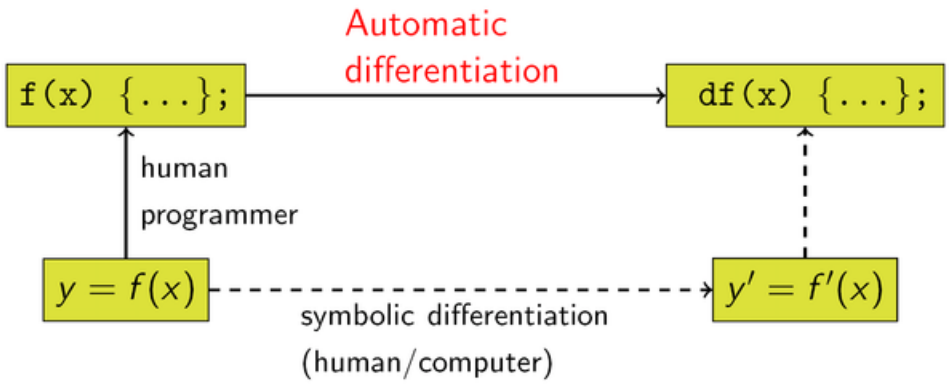


$$\frac{f(x+h) - f(x)}{h}$$

If h is too small: subtraction yields large rounding error. If h is too large: estimate of the secant becomes more accurate but estimate for slope of the tangent gets worse.

Automatic Differentiation and building blocks

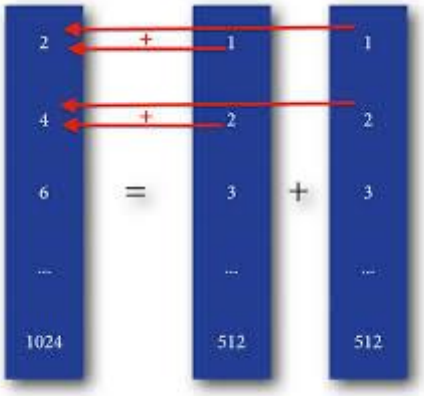
The key idea of automatic differentiation is called “forward” and “reverse mode” accumulation. Automatic differentiation makes use of the fact that *every complicated operation is built* from a small set of primitive operations such as addition, multiplication, or trigonometric functions. Automatic differentiation tracks operations and makes use of the chain rule of differentiation.



Combing code building blocks for (deep) models with automatic differentiation is a big game changer in ML.

Importance of Hardware (GPUs)

Let’s do a small tour de force in parallelization to fully appreciate the presented software frameworks. Think of vector addition or the respective Hadamard product. Ideally, we could calculate them all at the same time, in parallel!



This is fairly straightforward in C code executed on a CPU. When we use a GPU we now also need to worry about: Managing the GPU memory as a separate device, transferring arrays , Writing the code to parallelize on GPU, The GPU memory layout.

(ML) software abstracts such hardware acceleration away. We now get automatic differentiation, model blocks and hardware acceleration together!

10.3. The ML frameworks competition

Many frameworks appear (we won't go in detail today, see reference below). The core remains: CUDA + automatic differentiation. More layers for "ease of use" on top. More than just model optimization: data pipelines, reuse of models, monitoring, logging convenience …

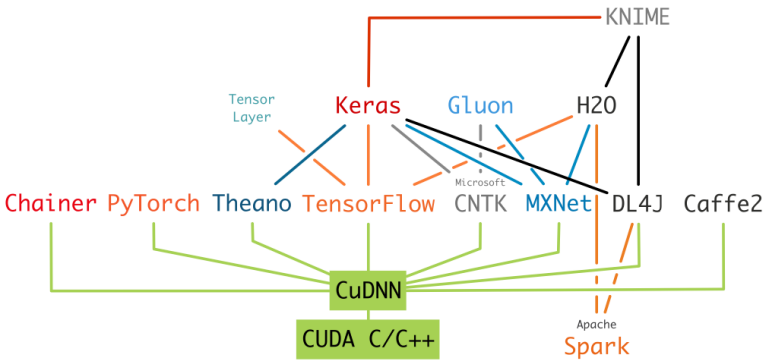


Fig. 3 The most popular Deep Learning frameworks and libraries layering in various abstraction implementation levels

There is a large focus on improving ease of use and accessibility! Frameworks keep growing, but are perhaps losing uniqueness? "Core" convergence. We are starting to emphasize sharing, transferring & reproducing