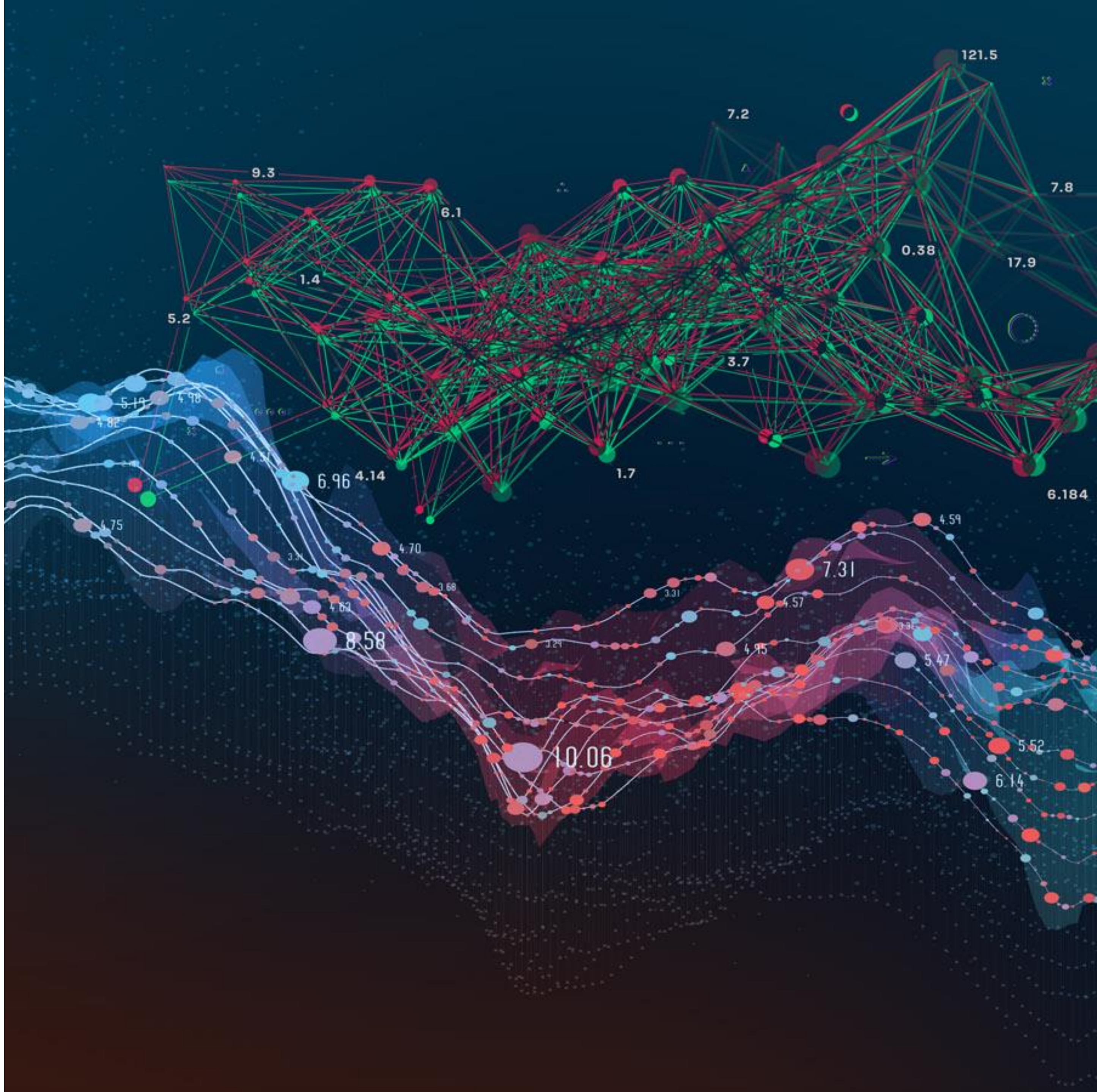
Pacific Northwest
NATIONAL LABORATORY

Day 2 – Models and Methods

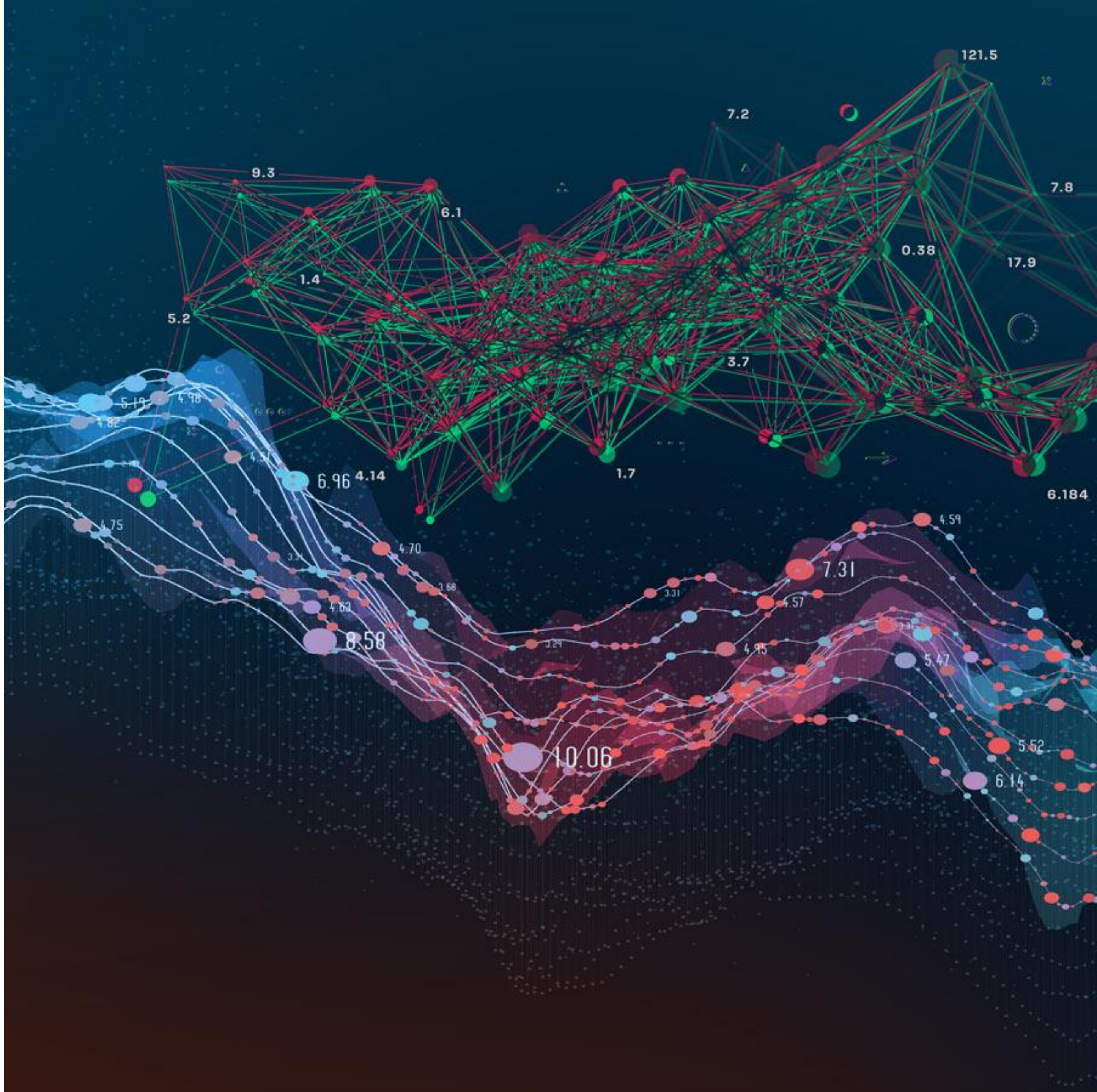


U.S. DEPARTMENT OF ENERGY BATTELLE

Day 1 reminders

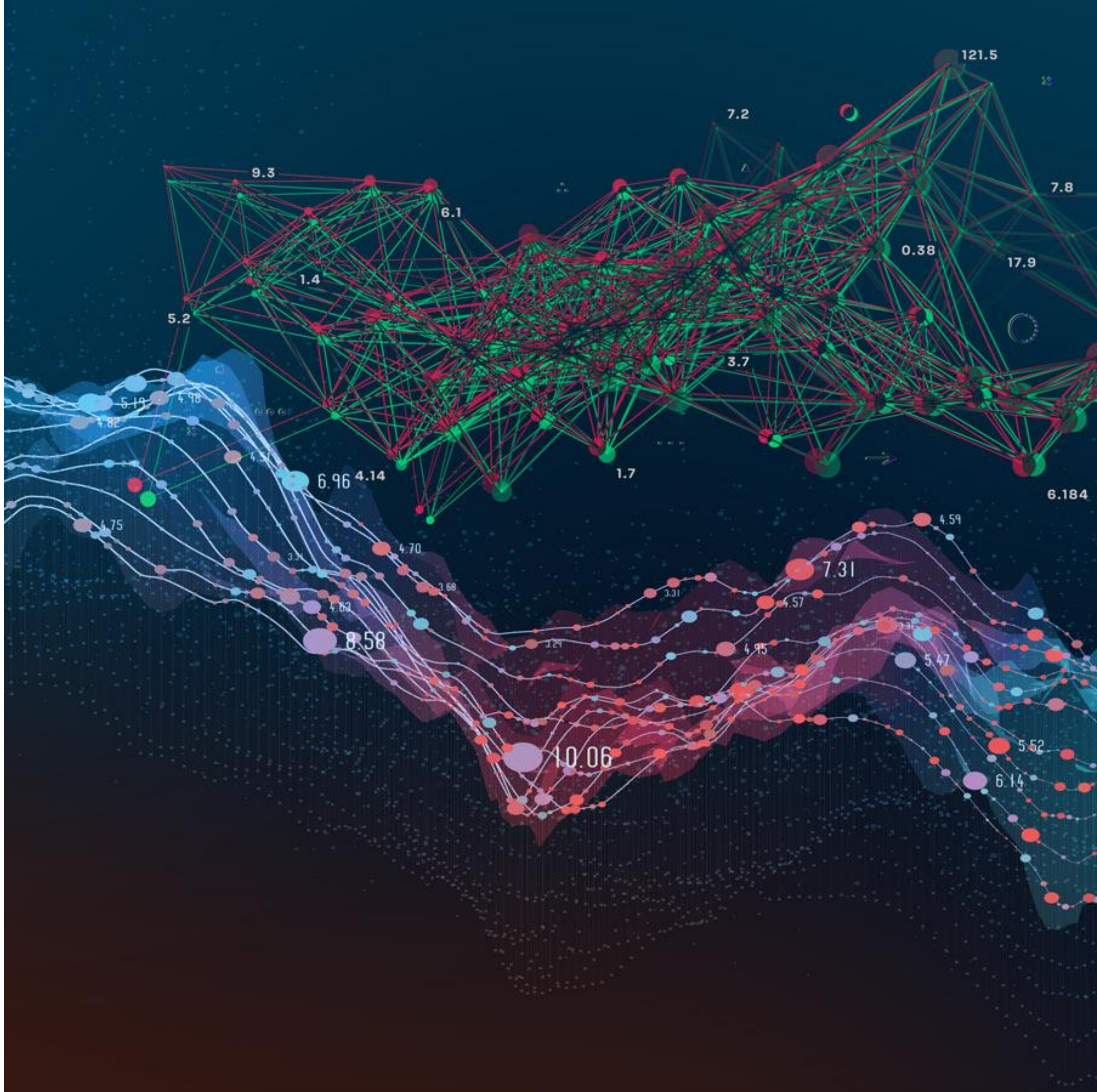
- Deep learning overview
- Few-shot learning
  - Explicitly train model to generalize to new classes
- Few-shot in action
  - Image filtering
  - Recent trends
- Practical example
  - Applying few-shot models to new data

Questions from day 1?



Pacific Northwest
NATIONAL LABORATORY

Training a Deep Learning Model



Large Data Paradigm

- When training a typical deep learning model using a sufficiently large dataset, data is split into train, test and validation splits.

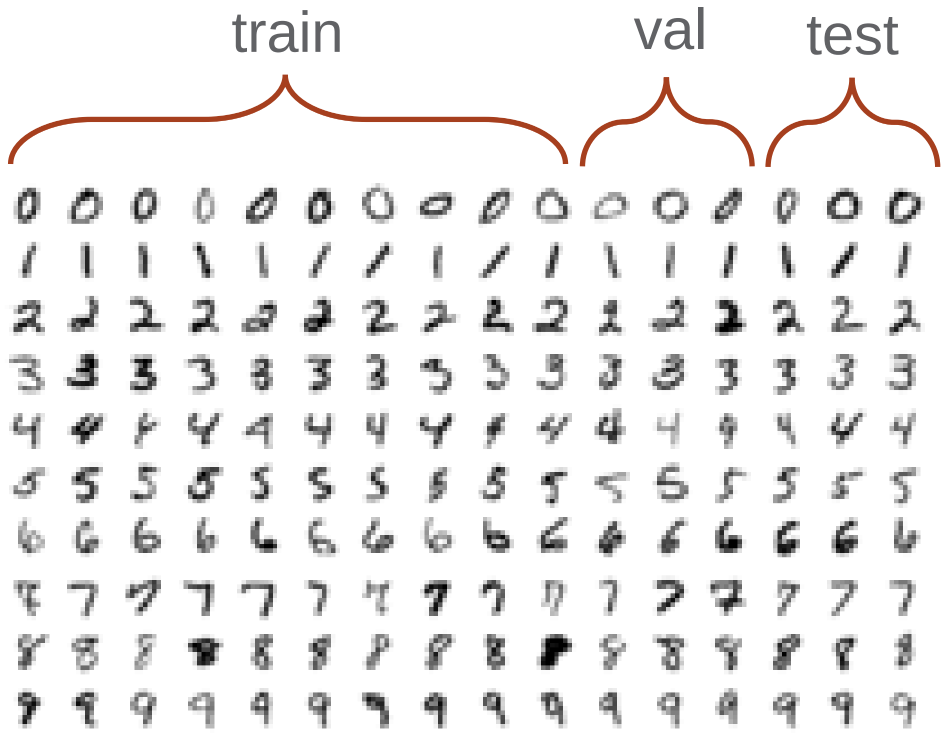
- The percentage allocated to each split varies, but anything from 60/20/20 to 80/10/10 is common

- Each split has an explicit purpose:
  - **Train** – Data used to update model weights.
  - **Validation/Development** – Data used to evaluate model during training and tune hyperparameters.
  - **Test** – Hold-out data, used for final evaluation values that are published.

Large Data Paradigm

- When using this three-way split, it is assumed that each split sufficiently represents the true distribution of each class

- For example, the MNIST handwritten digit dataset contains the numbers 0-9 and is used to train an image classifier



http://yann.lecun.com/exdb/mnist/

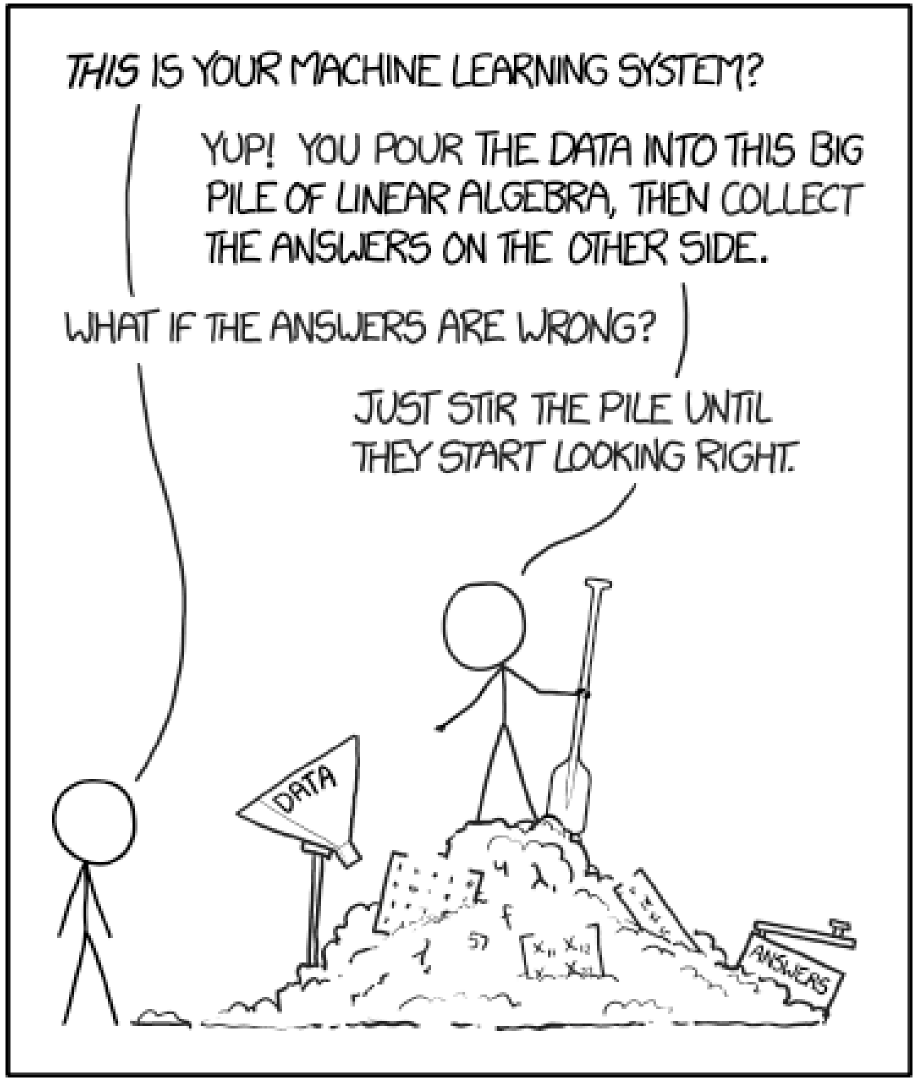
- A bad split would be to use digits 0-7 for training, digit 8 for validation, and digit 9 for testing
  - There is no hope for a model to perform inference correctly on digits 8 or 9

- Instead, any reasonable researcher would use a random sample or a stratified random sample to create the test, validation, and test sets.

Generalizability

- Generalizability is the ability of the model to perform well on unseen data points
- In the large data paradigm, generalizability is constrained by the domain of the training data
- Continuing with the MNIST example:
  - A model with good generalizability will be able to classify the digits 0-9 written in a variety of handwriting styles.
  - No model trained on MNIST would be able to classify letters

Training


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

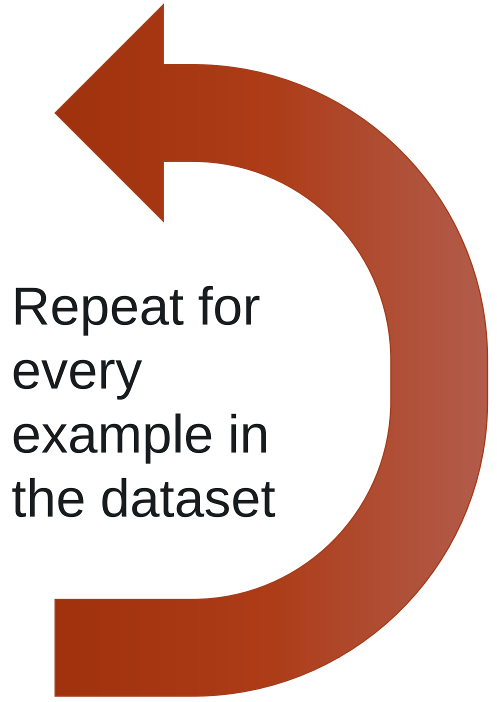
https://xkcd.com/1838/

Training

1) **Forward Pass:** Feed an datapoint (or batch of datapoint) through the model

2) **Loss Function:** Compare the model output to the ground truth target value(s) to get the amount of error

3) **Backpropagation and weights update:** Change the weights of the model to try and decrease that error



Once you have completed these steps for every example in your dataset, you have completed one **epoch** of training. Training can take dozens to thousands of epochs

Loss Functions

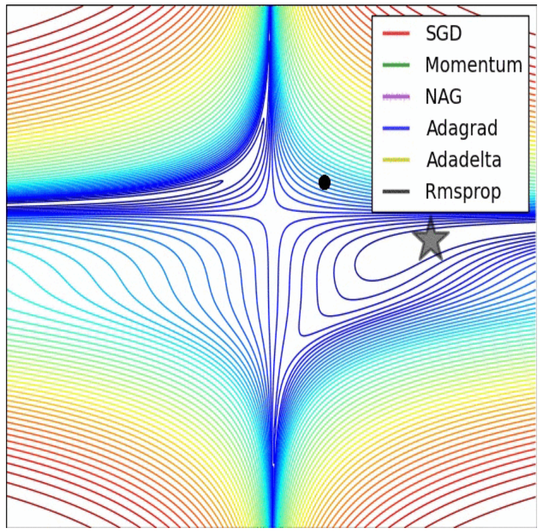
- Loss functions measure the error of a model
  - Always produce a scalar value
  - Always differentiable
  - Is a function of the weights, biases, and training data
- Regression Tasks:
  - Mean Squared Error
  - Mean Absolute Error
- Classification Tasks:
  - Binary Cross-entropy
  - Categorical Cross-entropy
- Regularization
  - L1, L2, or custom regularizing terms can be added to constrain model weights

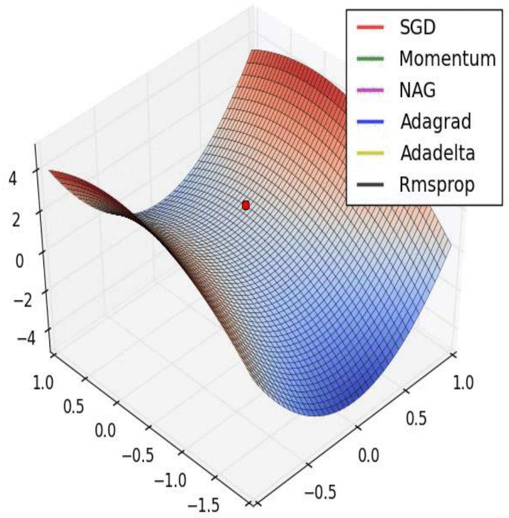
Optimizers

- We need an algorithm that will find the optimal values for the weights and biases that minimizes the loss function

- This is a high dimensional nonlinear optimization problem. The weights of the model are the parameters of the loss function

- We can use numerical methods (optimizers) to minimize the loss function to find model weights that result in low error.





https://ruder.io/optimizing-gradient-descent/

Optimizers

- Gradient Descent
  - An approximation method for finding minima for a given objective
  - w: parameter (weight/bias)
  - n: learning rate
  - Q(w): objective function (loss function)
  - $w = w - n\nabla Q(w)$

- How do we get $\nabla Q(w)$?
  - Backpropagation is an algorithm that leverages the chain rule (remember how fun calculus was?) to efficiently calculate the gradient of the loss function with respect to each weight/bias

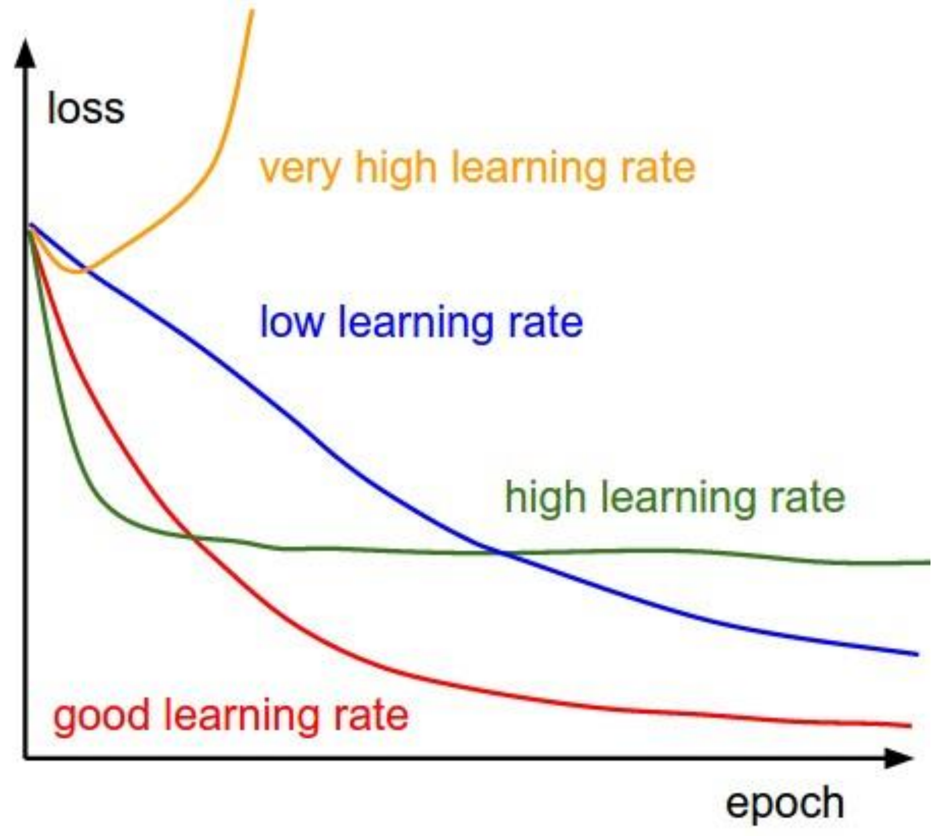
Mini-batches

- Mini-batches
  - We usually don’t calculate the loss for each example individually
  - Mini-batch loss is usually a simple average of the loss of each example in the batch
  - Instead, a *mini-batch* (often just called a *batch*) is used to:
    - Reduce computation time by parallelizing the feed forward and backpropagation steps
    - Explore the loss landscape in an efficient way

- Each mini-batch can be randomly sampled from the training data or stratified sampling can be used to control the influence of different types of data

Learning Rate

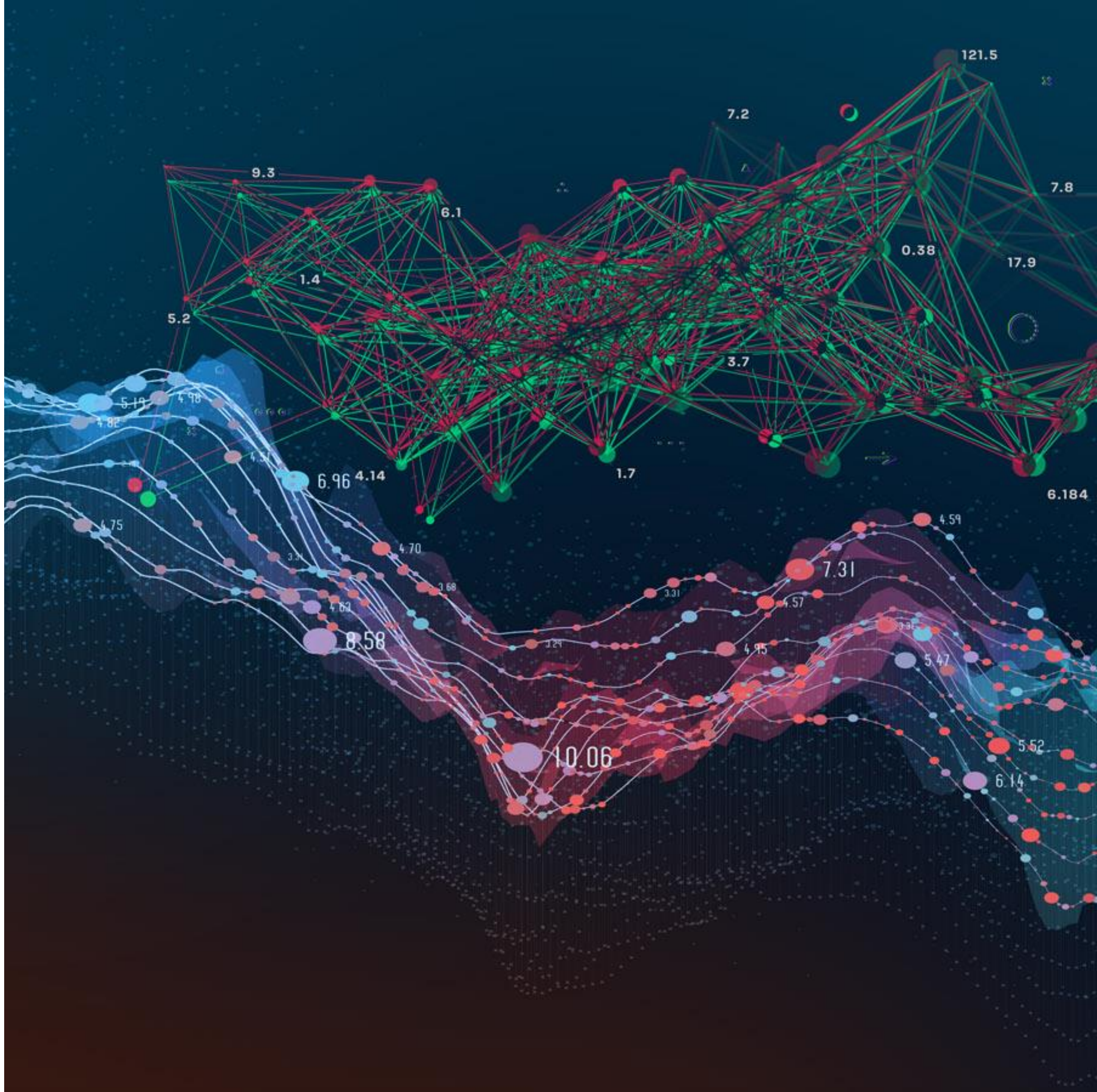
- Optimizers use a *learning rate* parameter to dictate how big of an update to make to the weights at each step
- Too high learning rates cause the model training to diverge
- Too low of learning rates produce slow model training that may not explore enough of the loss landscape



https://cs231n.github.io/neural-networks-3

Pacific Northwest
NATIONAL LABORATORY

Training a Few-shot Deep Learning Model



U.S. DEPARTMENT OF ENERGY BATTELLE

Datasets

Few-shot learning needs to generalize to unseen classes instead of unseen datapoints. This fundamentally changes how we approach splitting our data.

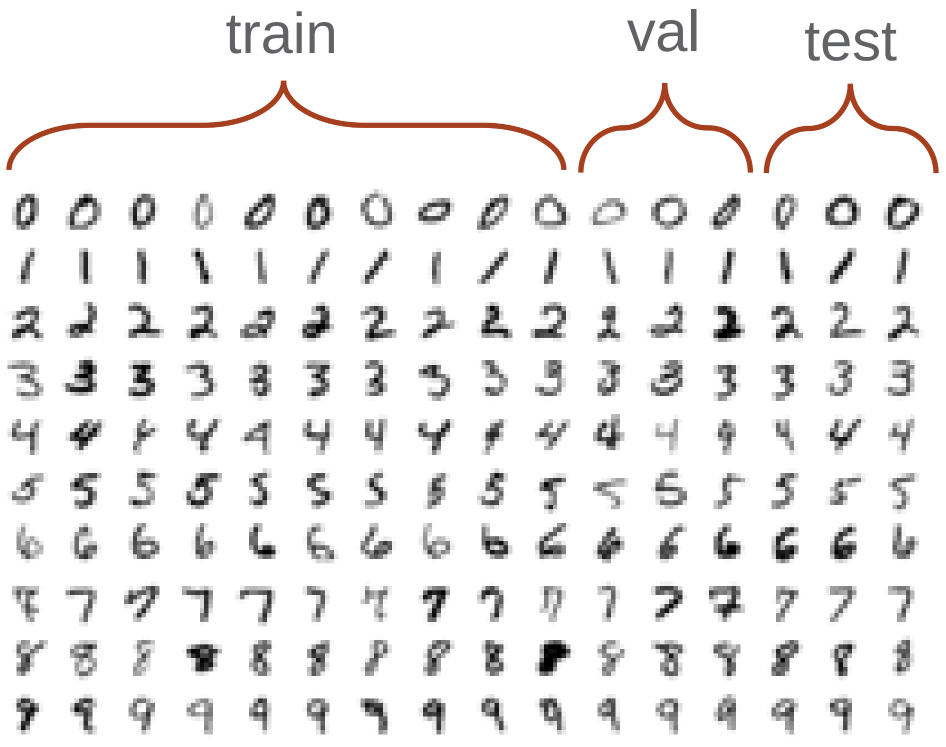
Recall:

- **Train** – Data used to update model weights.
- **Validation** – Data used to evaluate model during training and tune hyperparameters.
- **Test** – Hold-out data, used for final evaluation values that actually get published

But now we split on the classes instead of the datapoints so each split contains entirely unique classes.

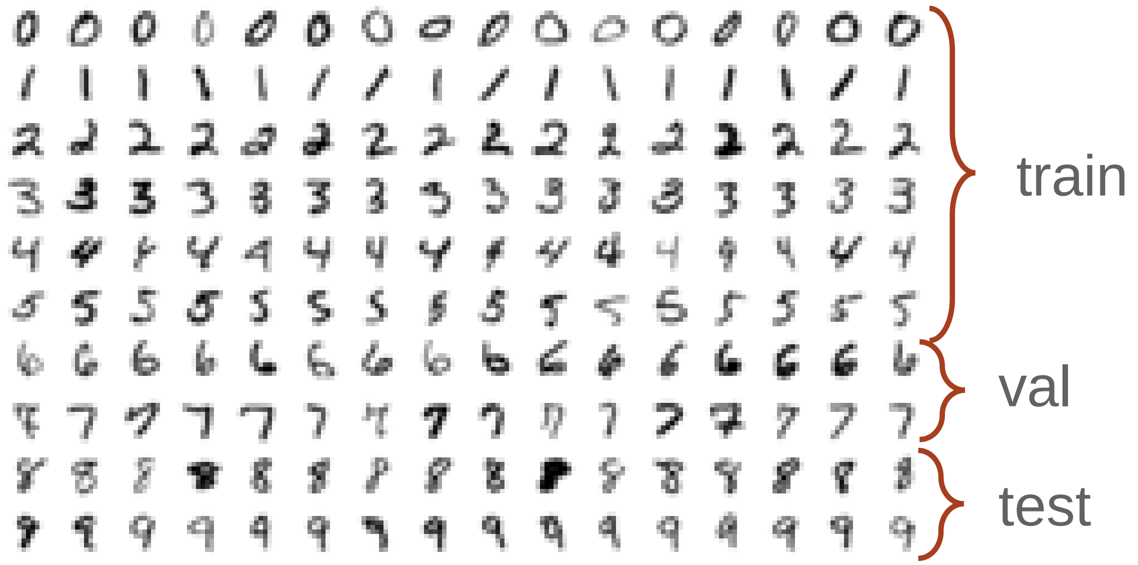
Datasets

Big data setting



http://yann.lecun.com/exdb/mnist/

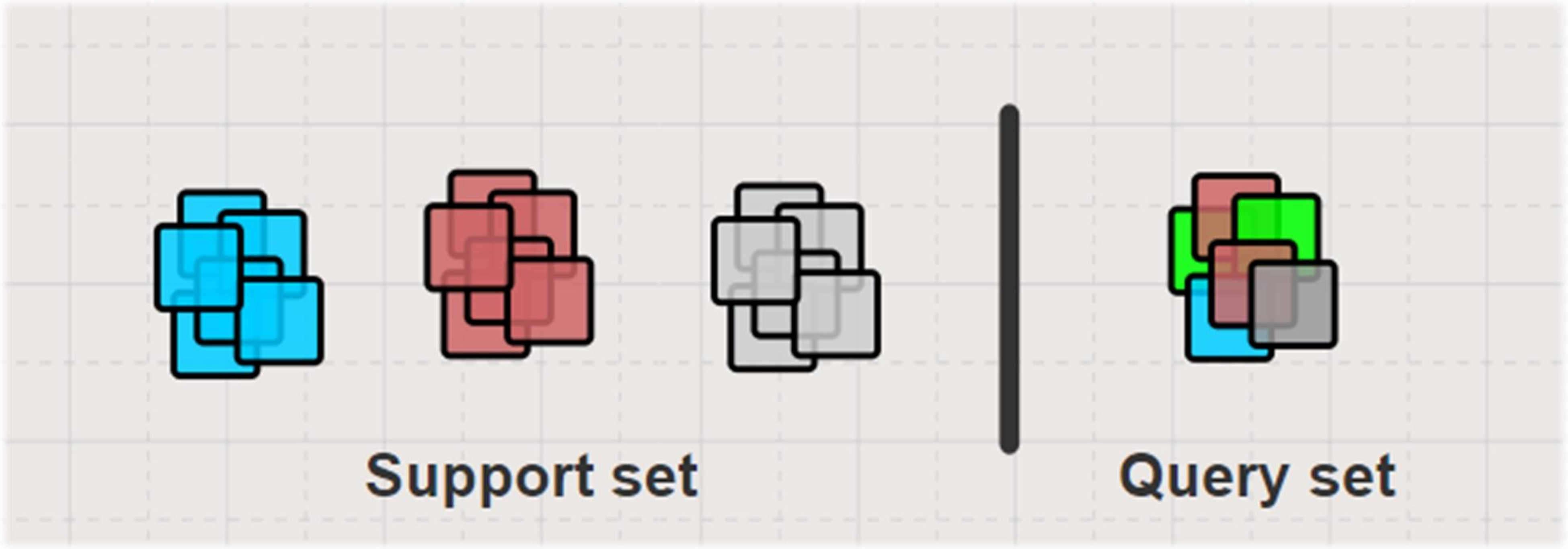
Few-shot setting



http://yann.lecun.com/exdb/mnist/

Episodes

Recall the structure of our problem from day 1:



Loss Functions

Most few-shot problems are classification problems, so we naturally lean heavily on the classic cross-entropy loss function:

$$\mathrm{loss}(\mathrm{x}, \mathrm{class}) = -\log\left(\frac{\exp(\mathrm{x}[\mathrm{class}])}{\sum_{\mathrm{j}} \exp(\mathrm{x}[\mathrm{j}])}\right) = -\mathrm{x}[\mathrm{class}] + \log\left(\sum_{\mathrm{j}} \exp(\mathrm{x}[\mathrm{j}])\right)$$

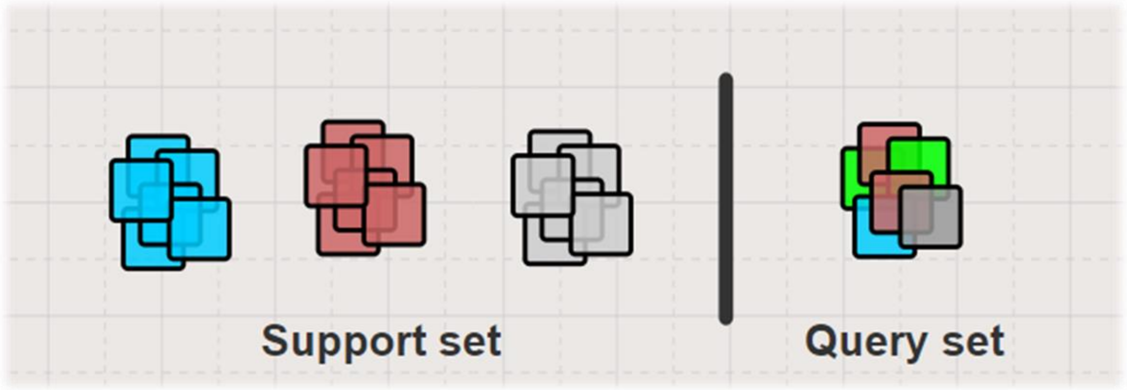
We want the model to learn how to classify arbitrary classes in the support set, not particular classes in the training data so we relabel our datapoints according to their class order in the support set.

Pacific Northwest NATIONAL LABORATORY

Episodes

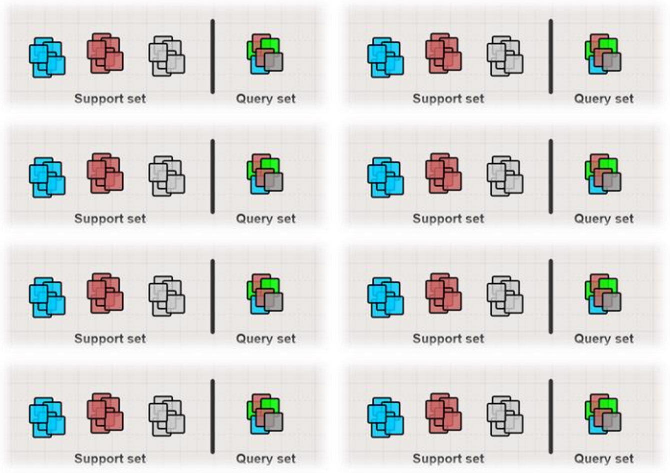
- In order to take advantage of big datasets
  - Sample whole episode from dataset
  - Run episode through the model and compute the loss
  - Compute gradients via backpropagation
  - Update model weights

- Note that under this setup we are sampling new classes in each episode

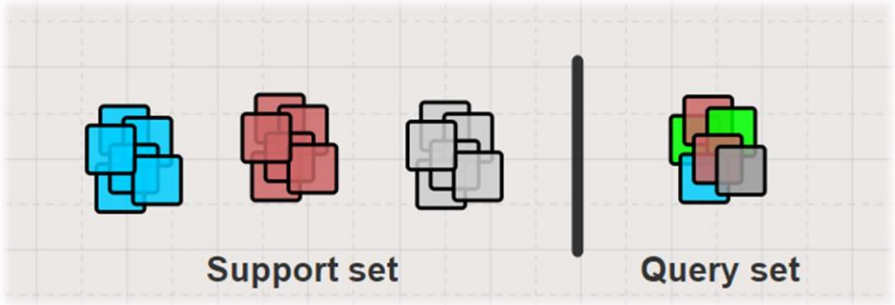


Meta-batches

- Some few-shot learning approaches apply meta-batching to the training process, sampling many episodes at a time instead of just one
  - Because of the stochasticity of random sampling, gradient updates may vary considerably from episode to episode. Averaging over multiple batches smooths this



Instead of

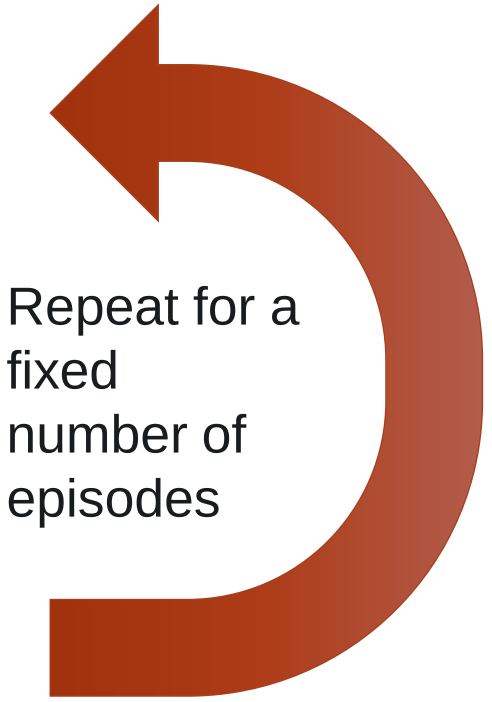


Training

1) **Forward Pass:** Feed an episode (or batch of episodes) through the model

2) **Loss Function:** Compare the model output to the ground truth target value(s) to get the amount of error

3) **Backpropagation and weights update:** Change the weights of the model to try and decrease that error



Once you have completed these steps for some fixed number of episodes, you have completed one **iteration** of training. Training can take dozens to thousands of iterations

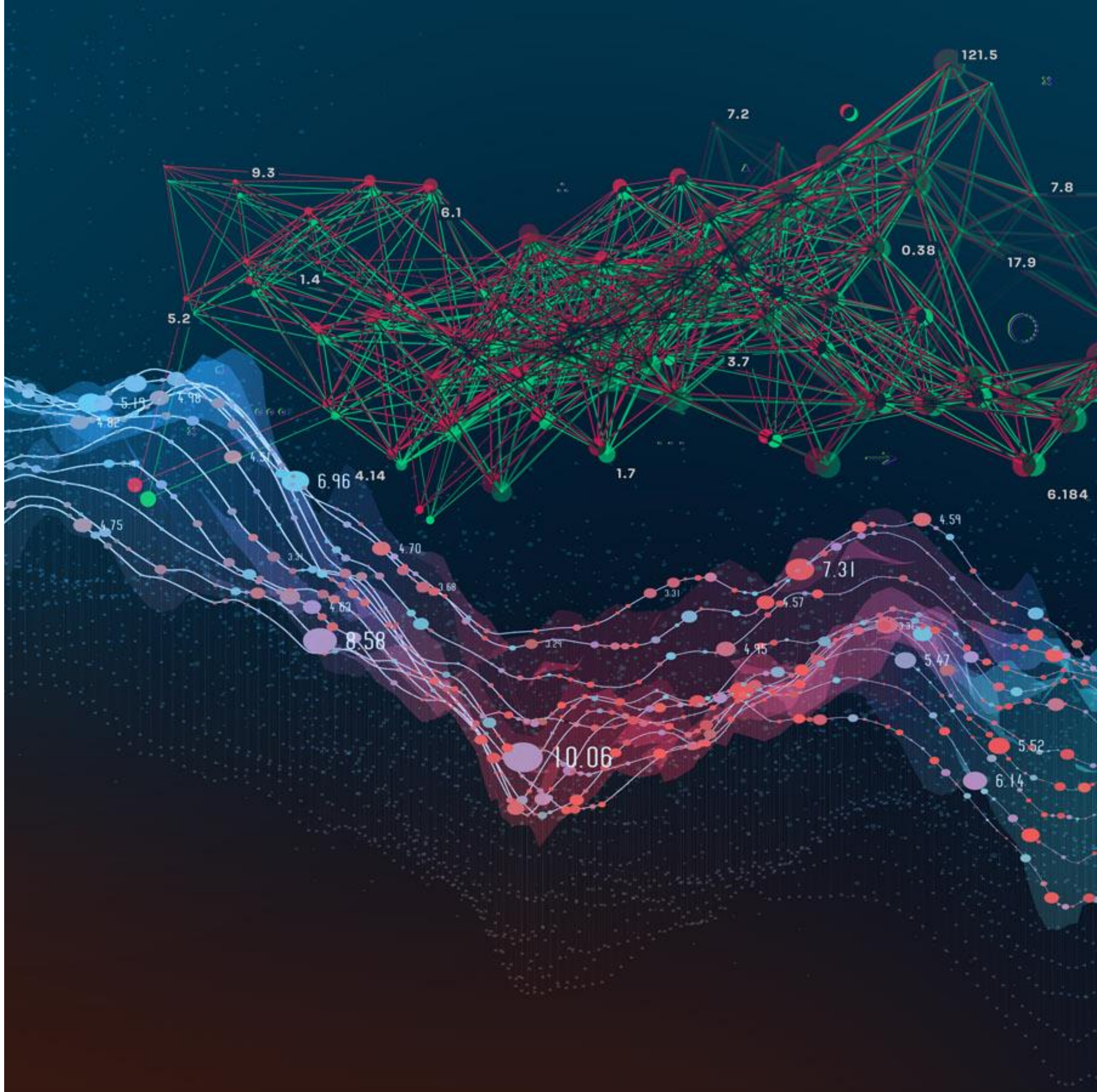
Tips and Tricks

- Learning rates are *very* important
  - Because episodes change stochastically, there may be large gradient differences episode-to-episode
  - Small learning rates & using meta-batches help to smooth gradient updates
- Training encoders from scratch seems to be difficult
  - Significantly better performance starting with a pretrained encoder
  - Can make the difference between completely untrainable and state-of-the-art performance
- When training on large datasets, it can take a long time to start overfitting
  - If it isn’t obviously overfitting, training longer might help

Pacific Northwest
NATIONAL LABORATORY

Few-shot Models & Algorithms

U.S. DEPARTMENT OF ENERGY BATTELLE



Pacific Northwest NATIONAL LABORATORY

Few-shot Models

Metric Learning

Metric learning models attempt to learn a mapping from the dataset into a metric space where proximity implies similarity.

Meta-Learning

Meta-learning models explicitly optimize to find parameters that can be easily fine-tuned on the support set

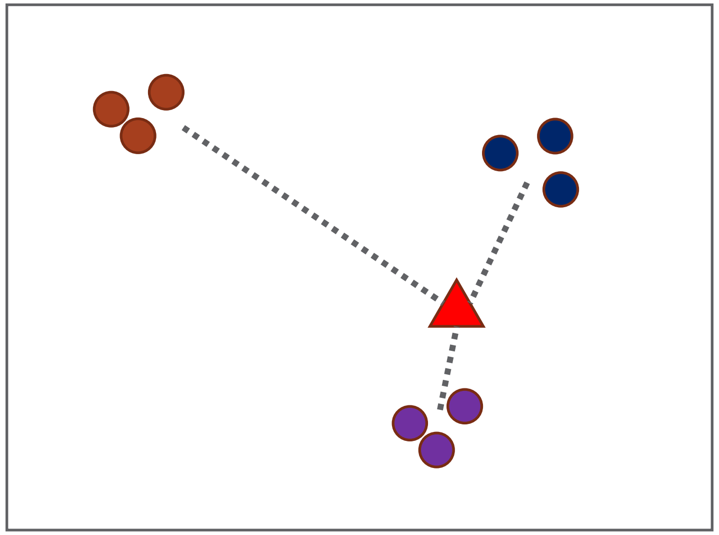
Conditioning Models

Conditioning models attempt to modify the encoder at evaluation time to better adapt to the particular support set.

Metric Learning - Prototypical Networks

- Since data is limited, our model probably needs a very simple inductive bias. (Snell et al. 2017)
  - With so few samples per class, nearest-neighbor classification might be too noisy.

1. Represent data as points in embedded space
2. Each class gets a "prototype" that is average of its members
3. New data grouped based on distance to prototypes



- No longer state of the art, but it's a very common few-shot baseline.
  - Simple to implement and straightforward to train.

Metric Learning -
Relating and Comparing Networks (RCNet)

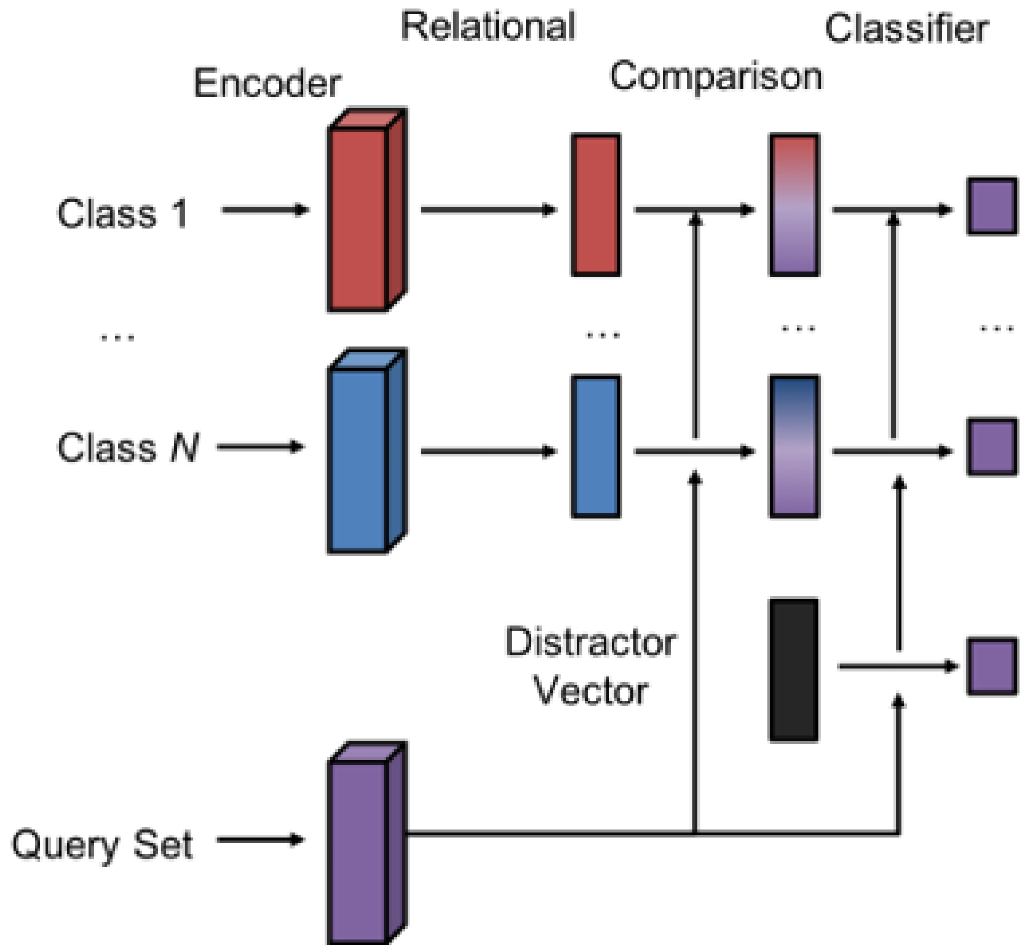
- Prototypical networks has limited flexibility. Only one way to compare classes

- Solution: Allow model to learn how to compare
  - Class prototypes based on pairwise relations between support examples
  - Query examples are compared to class prototypes, conditioning the class representation
  - MLP to predict classes instead of Euclidean distance

- Performs similarly to prototypical networks, but more flexible.
  - Can incorporate corrective user feedback and classify "none of the above" queries.

Metric Learning - RCNet

- Relational stage creates a class representation
  - Relates datapoints within a class
- Comparison stage allows queries to condition the class representation
- "Distractor" vector represents a none-of-the-above option for images that belong to no class



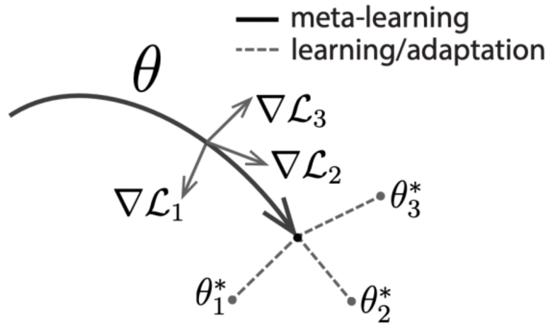
Metric Learning

- ProtoNets distance calculation enforces a strong inductive bias that is useful for few-shot problems
  - Many non-metric based approaches draw inspiration from this
- RCNet allows the model to learn how to perform comparisons
  - Flexibility to support none-of-the-above
  - Integration into a user interface allows for user feedback through “negative” examples
- Metric-based models typically require zero gradient updates during inference making them computationally efficient

Meta-Learning – MAML

- It might not be reasonable for one set of parameters to solve every problem.
  - What if we "learned to learn", so we could adapt a model to new data quickly?

- Solution: Meta-optimization! (Finn et al., 2017)
  - Starting from our initial weights θ, update using a few steps of gradient descent
  - Repeat over a few episodes to get θ* for each episode
  - Find a value for θ such that gradient descent would have found better θ* values
    - Gradient descent over gradient descent



https://arxiv.org/abs/1703.03400

- Influential idea, but computationally inefficient
  - REPTILE, FOMAML, and many other variants
  - Can be applied to non-few-shot problems as well (e.g. Reinforcement learning)

Meta Learning

- MAML and other some of its variants are among highest performers on many standard academic datasets
  - Although note that these are mostly small image datasets (84x84 pixels), so it's unclear whether this result scales to larger datasets

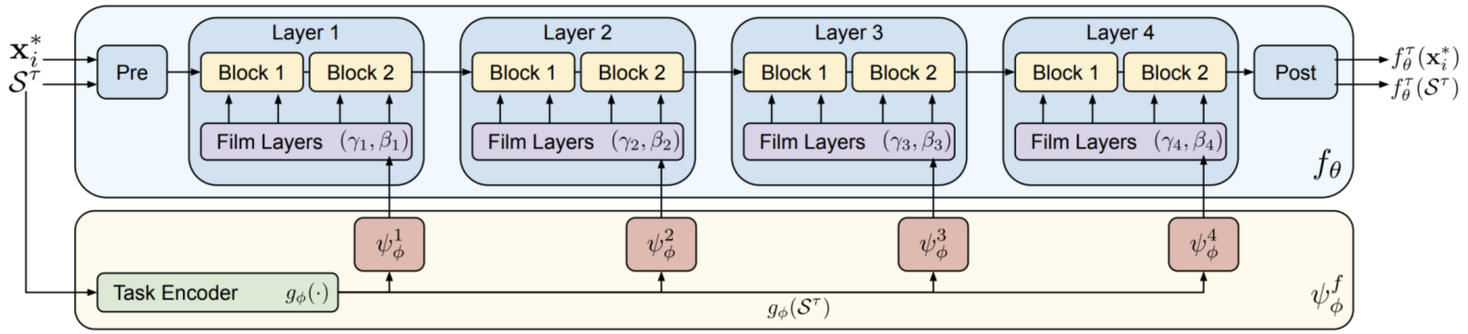
- Very flexible approach that is also useful for non-few-shot tasks

- Difficulties
  - Computational efficiency
  - Requires gradient steps during inference
  - Performs poorly as data begins to differ from training classes
    - Although incorporating aspects of ProtoNets reverses this (Triantafillou et al., 2019)

Conditioning Models – CNAPS / SCNAPS

- Meta-optimization adapts to unseen tasks, but is computationally expensive

- Solution: Train model to adapt its own parameters
  - Use the support set to “condition” the encoder, modifying how the encoder processes query datapoints (CNAPS:Requeima et al. 2019; SCNAPS: Bateni et al., 2020)
  - FiLM conditioning (Perez et al. 2017)



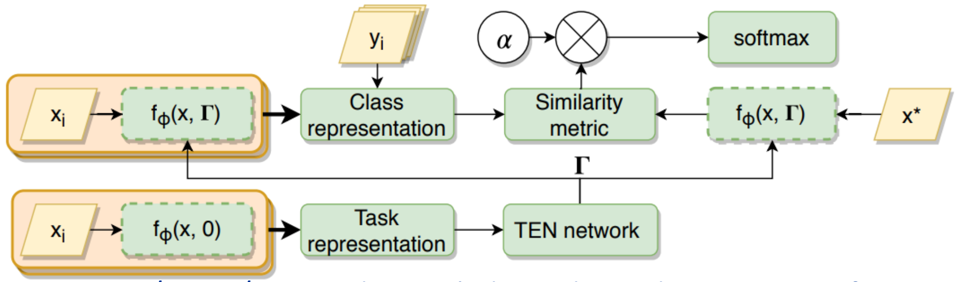
$\mathbf{x}_i^*$
$\mathcal{S}^\tau$
Pre
Layer 1
Block 1
Block 2
Film Layers (γ_1, β_1)
Layer 2
Block 1
Block 2
Film Layers (γ_2, β_2)
Layer 3
Block 1
Block 2
Film Layers (γ_3, β_3)
Layer 4
Block 1
Block 2
Film Layers (γ_4, β_4)
Post
$f_\theta^\tau(\mathbf{x}_i^*)$
$f_\theta^\tau(\mathcal{S}^\tau)$
f_θ
ψ_ϕ^1
ψ_ϕ^2
ψ_ϕ^3
ψ_ϕ^4
Task Encoder $g_\phi(\cdot)$
$g_\phi(\mathcal{S}^\tau)$
ψ_ϕ^f

FiLM conditioning structure. From fig 3: https://arxiv.org/pdf/1912.03432.pdf

- Less computationally expensive, shows good generalization performance

Conditioning Models - TADAM

- Early conditioning model, similar to SCNAPS.
  - Incorporates conditioning into a Protonets-like model (Oreshkin et al., 2018)



http://papers.nips.cc/paper/7352-tadam-task-dependent-adaptive-metric-for-improved-few-shot-learning

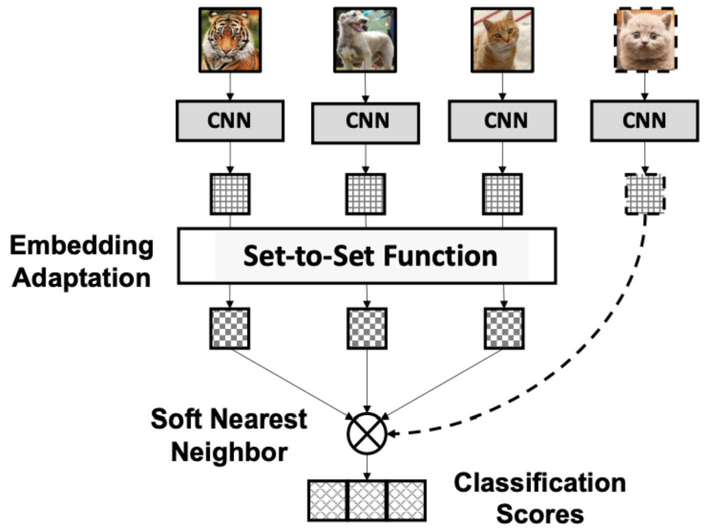
- Outperforms prototypical networks by 9+ accuracy points!
  - Still outperformed by other conditioning methods, today.

Conditioning Models - FEAT

- Previous methods run through the data twice
  - Once to determine conditioning, then to apply it
  - TADAM has to run the full encoder twice. SCNAPS uses a smaller "task network"

- Solution: Co-adapt all support features *after* encoding!
  - Uses a transformer as an efficient set-to-set function.

- Competitive with other recent approaches.
  - Transformer provides additional flexibility in model design

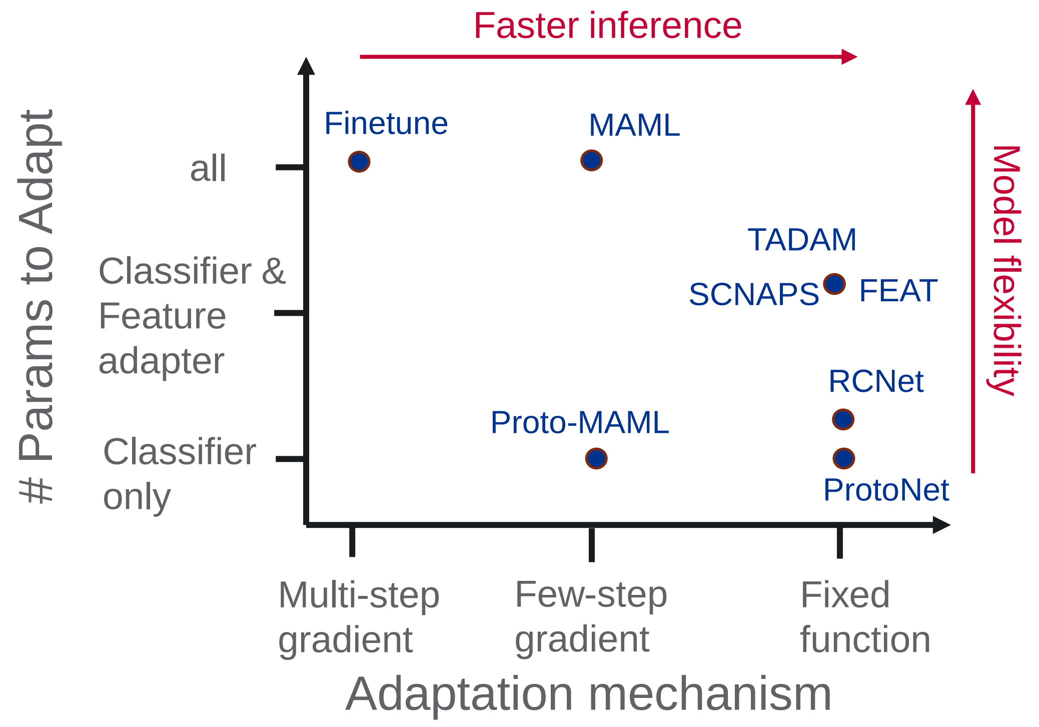


http://openaccess.thecvf.com/content_CVPR_2020/html/Ye_Few-Shot_Learning_via_Embedding_Adaptation_With_Set-to-Set_Functions_CVPR_2020_paper.html

Conditioning Models

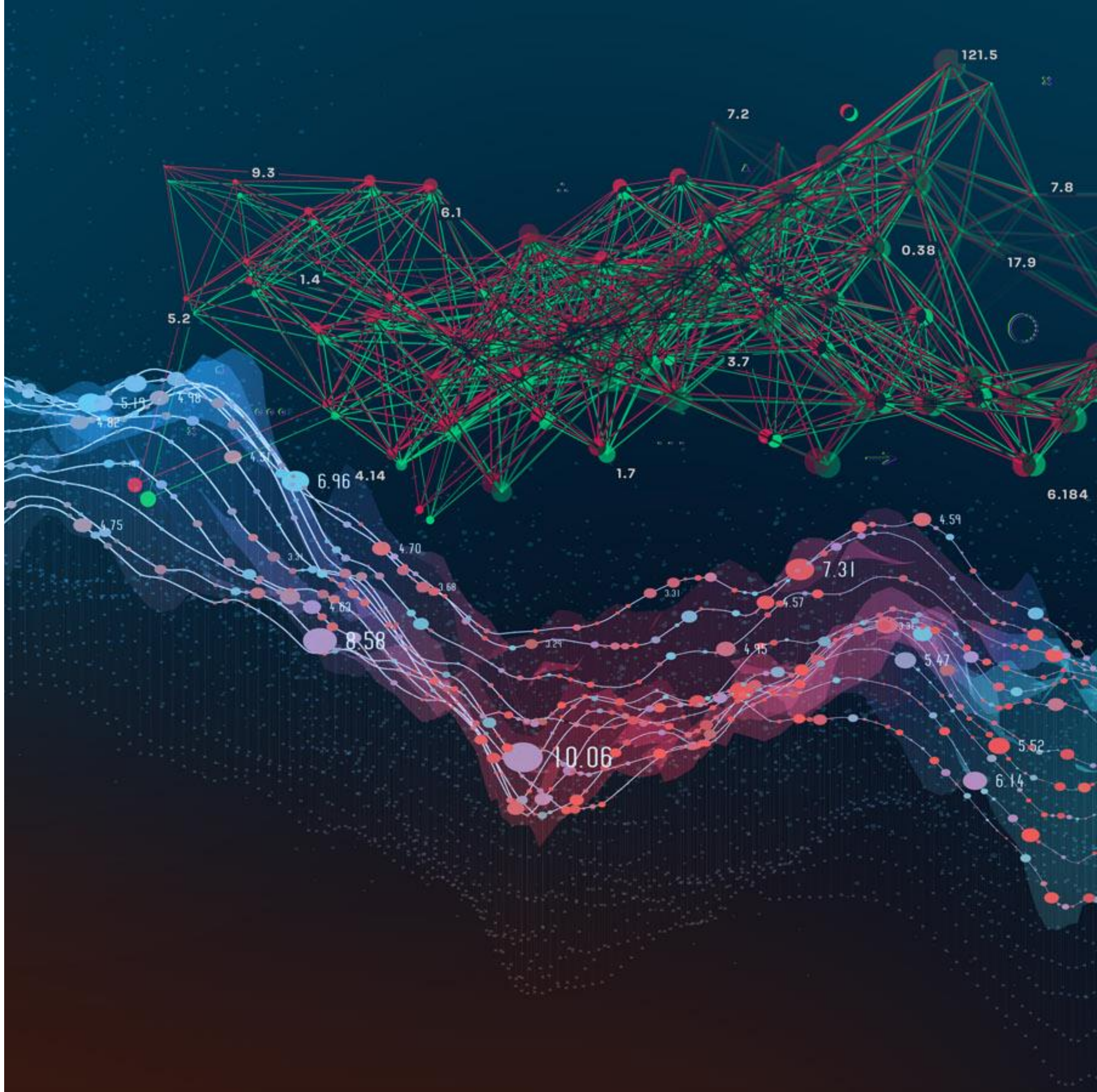
- Conditioning has been a strong approach for improving metric-based model performance
  - Best performers still seem to rely on strong ProtoNets-like inductive biases

- Lots of active research in this area
  - No systematic comparison yet



Pacific Northwest
NATIONAL LABORATORY

Model Evaluation



Comparing Models

- Many different competing approaches to few-shot learning
  - Including some different alternatives we'll touch on Day 4

- Direct comparison of models is difficult to assess from academic literature for a variety of reasons…

- Implementational details
  - How are classes/datapoints sampled for training?
  - Are you allowed to use the full test set to make individual predictions (i.e. transductive learning?)
  - Data splits are sometimes vague

MiniImageNet

- As an example, 5-shot, 5-way performance on MiniImageNet is the most reported performance metric in the academic literature
  - There are two train/test splits in common usage…
- Is fine-tuning a good approach?
  - Maybe, but it also uses a bigger encoder…
- All of these models are trained on 84x84 pixel images
  - Not useful for real-world tasks

	5-shot, 5-way	Encoder	Transductive?
MatchingNets	60.0	Conv (64)x4	No
ProtoNets	68.2	Conv (64)x4	No
MAML	63.1	Conv (32)x4	Yes* Not stated in paper
R2D2	68.4	Conv (96)x4	No
TADAM	76.7	Resnet-12	No
Fine-tuning	**78.2**	WRN-28-10	No
Transductive fine-tuning	**78.4**	WRN-28-10	Yes
LEO	77.6	WRN-28-10	Yes?

Results as reported in https://arxiv.org/pdf/1909.02729.pdf

Model Take-aways

- Most results are on small models trained on small images
  - Metric-based and conditioning approaches seem well-suited to scaling up
  - Meta-learning is a strong approach, but hasn't been tested on many large problems
- Inductive biases, e.g. fixed distance functions for classification, seem strong
  - ProtoNets, ProtoMAML, SCNAPS all perform well even when generalizing to new datasets
- As in other areas of deep learning, the field moves quickly
  - Expect to see new innovations