


**Deep Learning for Computer Vision**  
**Professor. Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture No. 74**  
**Few Shot and Zero- Shot Learning – Part 02**

(Refer Slide Time: 00:14)



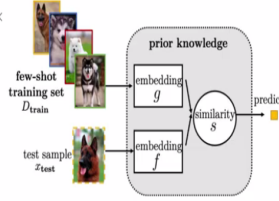
### Embedding Learning Methods

**Intuition:**


- Address few-shot learning by “learning to compare”
- If model can determine similarity of two images (and perhaps corresponding semantics of classes), it can classify unseen input in relation to labeled instance seen during training

**Method:**

- Learn separate embedding functions for training samples  $D_{train}$  and test samples  $D_{test}$
- Train sophisticated comparison models end-to-end via **meta-learning**
- At test time, predict based on comparing distance between  $x_{test}$  feature and training set features from each class



Credit: Wang et al, Generalizing from a Few Examples: A Survey on Few-Shot Learning, ACM Computing Surveys'20



Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 12 / 33


The first kind of methods that we will talk about are Embedding Learning methods or Model Based methods, as we just said. The key intuition of these methods is to address the few-shot learning problem by learning to compare. If a model can determine similarity between two images, or the similarity between the semantics of the class labels, which is required for zero-shot learning.

One can classify unseen input in comparison, or relation to labeled instances seen during training. What is the overall idea? One learns separate embedding functions, embedding functions are the output representations of a neural network in this context. For training samples,  $D_{train}$  and test samples  $D_{test}$ . And these comparison models are trained end to end using an approach known as meta learning which will describe soon.

At test time the prediction is based on is made based on comparing distances between the x test feature and the training set features from each class. So, here is the overall schematic. So, you have a Few shot training set  $D_{train}$ , and a test sample  $D_{test}$ , you have an embedding for the train

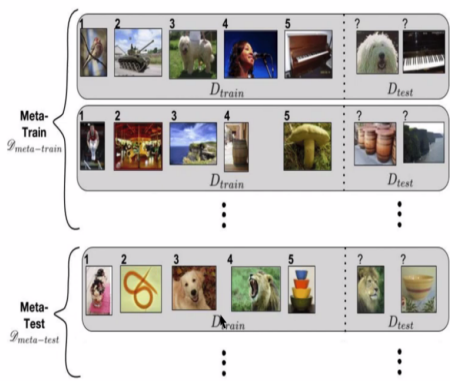
set  $g$ , and embedding for the test sample  $f$ . The similarity is computed to make the final prediction on the class label of interest. What is Meta Learning here? Meta Learning is known as learning to learn. And we will describe that in more detail.

(Refer Slide Time: 02:14)




### Problem Setup: Meta-Learning

- **N-way-K-shot:** N different classes in  $D_{train}$  with K samples per class
- $D_{meta-train} = (D_{train}, D_{test})$  set  $\rightarrow$  one task/episode
- Ensure  $D_{meta-train}$  and  $D_{meta-test}$  have disjoint/different classes



Credit: Hugo Larochelle



Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
13 / 33

If we considered the N-way-K-shot setting of Few shot learning, where N of the total number of classes have only K examples. In meta learning, in each episode, you have a different set of training classes, and a set of test examples. And these set of training classes may not be exhaustive, and complete the full training dataset. These are loosely sampled as some of the classes from your dataset. And the goal in this episode is to train a learner, which can make a prediction on this test set based on this train set. That would complete one episode of meta learning in which you get one learner as the output.


Now, in the second episode of meta learning, a different set of classes is again sampled to form your training dataset, you have a test set, and once again, the meta learner refines the learner to be able to solve the problem in this episode. This is repeated over multiple episodes, where in each episode, a different set of training classes may be sampled. And at meta test, you finally have your setting of your training classes, and your test samples where you would like to deploy your final model.

The goal here is  $D_{meta-train}$  and  $D_{meta-test}$  which are these episodes have disjoint classes. Which means the 5 classes here that you have in meta train may be different from the 5 classes here. But in meta test, but the goal is you have learned to learn a model.

(Refer Slide Time: 04:12)



**Problem Setup: Meta-Learning**




NPTEL

**Learning Algorithm A**

- **Input:** Training set  $D_{train} = (x_i, y_i)_{i=1}^I$
- **Output:** Parameter  $\theta$  model  $M$  (the learner)
- **Objective:** Good performance on  $D_{test} = (x'_i, y'_i)$

**Meta-Learning Algorithm**

- **Input:** Meta-training set  $D_{meta-train} = (D_{train}^{(n)}, D_{test}^{(n)})_{n=1}^N$  of tasks/episodes
- **Output:** Parameter  $\Theta$  algorithm  $A$  (the meta-learner)
- **Objective:** Good performance on meta-test set  $D_{meta-test} = (D_{train}^{(n)}, D_{test}^{(n)})_{n=1}^N$



Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
14 / 33

So, another way of differentiating in a classical machine learning algorithm setting, you have a training set, you learn a model with certain number of parameters with an objective to get a good performance on a test set. In meta learning, you have a meta training set, which has its own train and test split, which corresponds to one episode of meta learning. You learn a set of meta parameters theta, which is used to train learners on a new meta test set.

(Refer Slide Time: 04:49)

**Training Setup: Meta-Learning**

- **Training:** Repeat meta-loop for each task/mini-batch of tasks in  $D_{meta-train}$  as shown
- **Inference:** On tasks/episodes in  $D_{meta-test}$



Problem setup matches training and inference to enable generalization to new classes at test-time

dit: Hugo Larochelle  
Vineeth N B (IT-H) §12.1 Zero-shot and Few-shot Learning 15 / 33

Here is another way of understanding this. You have your training set in each episode of meta learning that is used to learn a meta learner which teaches a learner to learn a model, and there is a loss induced because of learning across these episodes in each round of meta learning. If you see here, between the meta train and the meta test settings, the problem set up matches, so that once you have trained a Meta-learner, the Meta-learner knows how to provide a model for the new set of classes that may come in the training part of the Meta test episode.

Remember that in Meta learning, each episode can have a training and test. So in the Meta test, you have a set of training classes. The learnt Meta-learner knows how to produce a model for the set of classes to be able to solve these test samples.

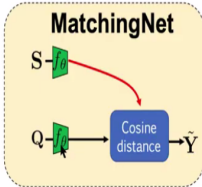
(Refer Slide Time: 05:55)



### Matching Networks<sup>1</sup>

- **Parametric models:** Slowly learn model parameters from training examples; Require large datasets to avoid overfitting
- **Non-parametric models:** Allow novel examples to be rapidly assimilated; Robust to **catastrophic forgetting**

#### MatchingNet



**Idea:** Combine best of both worlds

- **Training Phase:** Learn cosine similarity-based embedding models (parametric meta-learners)
- **Test Phase:** Use Nearest Neighbors (non-parametric) in learned embedding space

Vinyals et al, Matching Networks for One Shot Learning, NeurIPS 2016  
Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 16 / 33

Let us see a concrete example of this idea of meta learning. This is known as Matching Networks, a method that was developed and published in NeurIPS of 2016. A matching networks is premised on bringing together Parametric models and Non-parametric models. Recall that parametric models learn model parameters from training samples slowly, you ideally require large datasets to avoid over fitting.

Unfortunately, in this setting, we are talking about few-shot and zero-shot, you may not have large datasets. On the other hand non-parametric models like K Nearest Neighbors, allow novel examples to be assimilated quickly. And they are robust to a phenomenon called Catastrophic Forgetting. Catastrophic Forgetting refers to a phenomenon where if you have a set of classes for which a model is trained, if you took MNIST, and you trained a model on the classes 0 to 6.

Let us assume that the classes 7 and 8 occur a bit later. And when you train your model, or refine your model, on the classes 7 and 8, the model forgets what a 0 or a 1 look like. This is known as catastrophic forgetting. And neural network models are known to be prone to this phenomenon. If you do not retrain on the complete dataset, which may not be possible in all settings. Non-parametric models automatically are robust to such a phenomenon. As an example, as I said, could be K Nearest Neighbors.

So, in this method called matching networks, the proposers proposed to combine the best of both worlds. So you have a training phase, where you learn cosine similarity based Embedding models. So, you see here that you have a set of samples S and a query sample Q. So, S is like your train set in a Meta learning episode and Q is like your test set in that Meta learning episode. You learn Embedding function F, which gives you corresponding embeddings for the train set and the test set and then you use cosine distance to measure the similarity. That is what happens in the train face to learn the F's. And a test time once you get these embeddings a Nearest Neighbors approach based on the cosine distance is used to classify the test sample.

(Refer Slide Time: 08:43)

**Matching Networks**

- $$\hat{y}_{ts} = \sum_{i=1}^k a(\hat{x}, x_i) * y_i$$

where  $a(\hat{x}, x_i)$  denotes the attention mechanism over examples
- Simplest form of attention mechanism  $\Rightarrow$  softmax over cosine distances  $c(\cdot, \cdot)$

$$a(\hat{x}, x_i) = \frac{e^{c(h(\hat{x}), g(x_i))}}{\sum_{j=1}^k e^{c(h(\hat{x}), g(x_j))}}$$

Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
17 / 33

Here is the architecture that was used in Matching Networks. So, you can see here that you have this is one meta learning episode, you have a set of samples from a given set of training classes, you have a test sample in that meta learning episode, remember that you repeat these meta learning episodes with a different set of training classes and test sample in each Meta learning episode.

These training samples as are given as a sequential set of inputs to an LSTM. And the LSTM outputs a representation for each of these input training samples. And you also get a representation of the test sample. And then, there is a comparison module that looks at the similarity between the test sample and each of these training samples. And that comparison is



given in the equation here as  $a(x, \hat{x}_i)$ . Where  $x_i$ 's are the different training samples. How do you measure how do you compute this a function?

In this approach, it is a very simple computation. It gets a softmax over the cosine distances between  $h(\hat{x})$  and  $g(x_i)$ . So, you take a cosine similarity between the test image and each of the training images, and then do a softmax over these cosine similarities, to show to get a distribution over which of these training samples, this test sample is closest to. And multiplying that with the class label gives us the final class label prediction for this test sample.

One detail here is if you notice, the LSTM here is a Bi-directional LSTM. Why is that so? We do not want this LSTM to actually depend on the sequence of these training samples that are given as input. And by going Bi-directional, by doing a forward and reverse direction in the LSTM. You are trying to counter that dependence on the sequence.

(Refer Slide Time: 11:01)

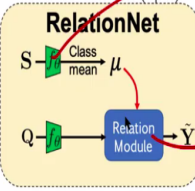
### Relation Networks<sup>2</sup>

**Idea:**

- Train data-driven learnable non-linear metric end-to-end instead of manually picking one
- Enables model to extract complex non-linear relationship among data samples  
⇒ generalize better to novel classes
- Easily extensible to tackle more difficult zero-shot setting

**RelationNet**



- **Training Phase:** Meta-learn both embedding module (for feature representation) and relation module (learned transferable deep metric) end-to-end
- **Test Phase:** Use relation scores in embedding space to classify new samples

<sup>2</sup>Sung et al, Feature Generating Networks for Zero-Shot Learning, CVPR 2018



Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 18 / 33

An improved idea was called Relation Networks, which was published in CVPR of 2018. In this approach, the author's question the use of a cosine embedding, instead, the method proposes to train and learn a data-driven nonlinear metric, instead of using the cosine distance metric. This enables the model to extract complex nonlinear relationships among the data samples, and thus generalize better to novel classes.

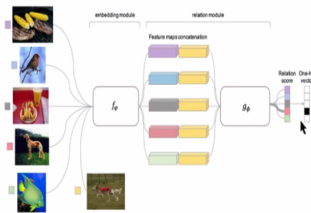
And in fact, relation networks are also extensible easily to the more challenging zero-shot setting, where you have no samples for the unseen classes. So, here is the overall schematic. So, if you see here, you still have S and Q, your train samples and your query sample. They go through a learned embedding, which is given by  $f_\phi$ , the parameters of that network. And now you take the class mean  $\mu$  for a set of samples belonging to a class, and you compare the query sample to that mean representation of a class.

How do you compare? That comparison is also learned by the model, instead of using the cosine distance. So, in the training phase, you Meta-learn both the Embedding module which is  $f$ , and the relation module, which is the one that learns the relationship, or the distance metric between the representations. And at test phase given a query sample, you use the relation scores in the embedding space to the mean of each of your training classes to finally predict the class for the query sample.

(Refer Slide Time: 13:00)

### Relation Networks




**One-shot Learning:**

- Samples  $x_j \in Q$  and  $x_i \in S$  are fed into embedding module  $f_\phi$
- Feature maps  $f_\phi(x_j)$  and  $f_\phi(x_i)$  combined with concatenation operator  $C(f_\phi(x_j), f_\phi(x_i))$  to extract relations
- Combined feature map fed into relation module  $g_\phi$ ,  $\Rightarrow$  produces a scalar  $\in (0, 1)$   

$$r_{ij} = g_\phi(C(f_\phi(x_j), f_\phi(x_i)))$$
- **Objective:**  

$$\varphi, \phi \leftarrow \arg \min_{\varphi, \phi} \sum_{i=1}^m \sum_{j=1}^n (r_{ij} - 1(y_i == y_j))^2$$



Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
19 / 33

Here is the overall architecture, you have your set of training set in a given meta learning episode, you have a query sample here, that comes in as a separate test sample. All of these go through the same Embedding module  $f_\phi$ . You get feature maps corresponding to this  $x_j$ , and all the  $x_i$ 's from S. And now different from matching networks, relation networks concatenate these feature maps. The yellow bar here corresponds to the features of  $x_j$ .




And each of these other colored bars are features from each of the other  $x_i$ 's in your train set of that Meta learning episode. What happens after you concatenate? This is then fed to a relation module  $g$ , whose parameters are again learned, which finally produces a scalar score for the similarity between the  $x_j$  and each of the  $x_i$ 's which is given as a vector of relation scores. And finally, by doing a softmax, or an arg max over the relation scores a final one hot vector can be predicted.

How is this network trained? The relation scores  $r_{ij}$  are critical for training the network. The loss

function is given by  $arg\ min_{\phi, \phi} \sum_{i=1}^m \sum_{i=1}^n (C(f_{\phi}(x_j), f_{\phi}(x_i)))$ . Rather this says that whenever  $y_i$  is equal to  $y_j$ , or the query sample's label matches one of the train samples label, we want  $r_{ij}$  to be 1, remember this is squared. So, this overall quantity has to be positive. We want  $r_{ij}$  to be 1 in that scenario, because similar similarity is high and whenever the labels do not match,  $r_{ij}$  will be forced to go lesser than 1.

(Refer Slide Time: 15:18)


Relation Networks: Application to Few/Zero-shot Learning


**Few-shot Learning** ( $K$  shot;  $k > 1$ )

- Use element-wise sum over embedding module outputs of samples from each training class (class feature map)
- Combine pooled class-level feature maps with query image feature map  
Number of relation scores for one query is always  $C$  (both one-shot or few-shot setting)

**Zero-shot Learning:**

- Support set contains semantic class embeddings  $\implies$  vector  $v_c$ ; instead of one-shot image for each class
- In addition to  $f_{\phi_1}$  (for query images), introduce embedding module  $f_{\phi_2}$  to handle semantic attributes

$$r_{ij} = g_{\theta}(C(f_{\phi_2}(v_c), f_{\phi_1}(x_j)))$$



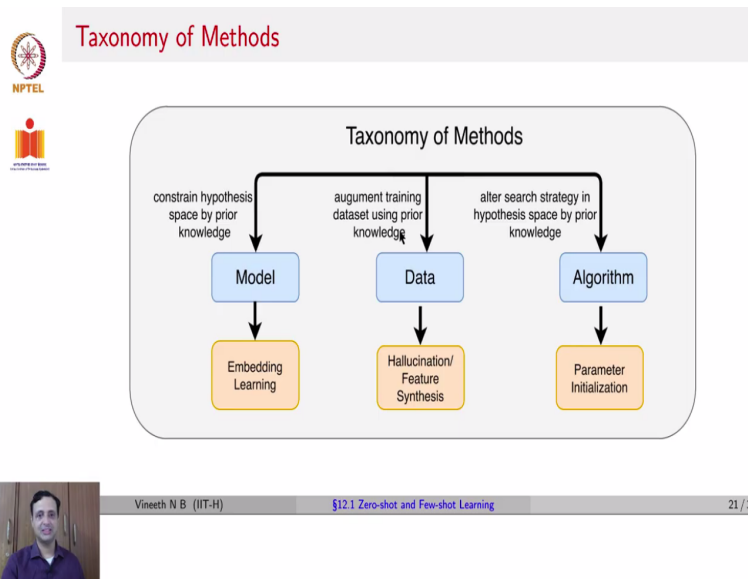
Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
20 / 33

How is this used for few-shot or zero-shot learning? In the few-shot learning scenario, what we described is exactly what is done, you have an element wise sum over the embedding module, which gets your class feature map, we combine the class feature map with the query feature map,

we just talked about the concatenation, and then we get the relation scores to finally get the predicted label.

For Zero-shot, because you may not have any samples at all, for some of your classes. You need what are known as semantic class embeddings. So, you get some Meta information about a class, could be a set of attributes, could be any other textual description of a class label, or simply take a Word2Vec representation of the class label itself or any representation, word level representation of the class label itself. Now, the concatenation, and the embedding operator compares the embedding of the class label with the embedding of each of those trained samples. And that is used to get a similarity and be able to give the final outcome.

(Refer Slide Time: 16:39)



That is about Embedding Learning methods. Let us move now to the second kind of methods data based methods, which are Hallucination or Feature Synthesis methods.

(Refer Slide Time: 16:52)

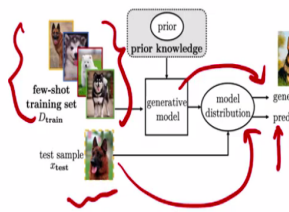


## Hallucination/Feature Synthesis Methods



### Intuition:

- Directly deal with data deficiency by "learning to augment"
- Learn a generative model to hallucinate new novel class data for data augmentation
- Reduce few/zero-shot problem to a standard supervised learning problem



### Method:

- Learn a generator conditioned on meta-information using data in base classes
- Generate novel class features conditioned on unseen class meta-information
- Train a classifier on base class samples and generated novel class samples

Credit: Wang et al, Generalizing from a Few Examples: A Survey on Few-Shot Learning, ACM Computing Surveys'20



Vineeth N B (IIT-H)

§12.1 Zero-shot and Few-shot Learning

22 / 33

The intuition with these methods is learning to augment. We saw learning to compare, now we talk about learning to augment. Here, we learn a generative model to hallucinate new novel class data for data augmentation. And by generating more data, we reduce few or the zero-shot learning problem to a standard supervised learning problem. How do we do this? A standard approach is to learn a generator that is conditioned on some Meta information using the data in the base classes. And then we generate novel class features conditioned on unseen class Meta information. And finally, we train a classifier on the base class samples and the generated novel class samples.

So just to show this, you have your set of training samples in your meta learning episode, a test sample, so using some meta information, and using the train samples themselves, you train a generative model, which is then finally given to a classifier to be able to say, of course, you have since its generative model, you may have an adversarial component there to decide whether the generated features are real or fake. But you also have a classification component, which tells how to classify these final generated features. Let us see a more concrete example.

(Refer Slide Time: 18:32)

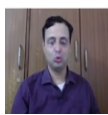
**Feature Generating Networks (f-CLSWGAN)<sup>3</sup>**

**Method:**

- Given train set  $S$  of seen classes, learn a conditional generator  $G : Z \times C \rightarrow X$ , which takes random Gaussian noise  $z \in Z \subset \mathbb{R}^{d_z}$  and class embedding  $c(y) \in C$ , and outputs image feature  $\tilde{x} \in X$
- To ensure  $\tilde{x}$  are well-suited to train a discriminative classifier, minimize classification loss over generated features  $\tilde{x}$

**Extension to Few-shot Learning:**

- For FSL, along with seen classes data set  $S$ , the training data also includes few labeled samples for each unseen class as well



<sup>3</sup>Xian et al, Feature Generating Networks for Zero-Shot Learning, CVPR 2018

Vineeth N B (IIT-H)

§12.1 Zero-shot and Few-shot Learning

23 / 33

A popular method in this space is f-CLSWGAN. Here the goal is given a train set of seen classes, we learn a conditional generator  $G$ , which receives as input  $Z$ , the random standard noise that you give to a GAN and a class embedding corresponding to each class. Using these, the generator learns to generate image features. We do not need to generate images here, because our goal is not really to generate pretty looking images, our goal is to generate image representations which can be classified and this makes it a more feasible problem.


To ensure that the features generated by this approach are good. There is also a discrimination module, which minimizes classification loss over the generated features. So, you see here that f-CLSWGAN tries to take some conditional, some attributes of belonging to different classes, which are conditioned on to generate image features. And what kind of image features are generated? These are image features that are outputs of CNN models on standard images.

So if you used a VGG or a ResNet as a CNN, you would get a certain representation of the image. And we are now asking the GAN to generate similar vector features instead of generating images. How do you extend this to Few shot learning? So, this would be the approach for Zero shot learning, where there are no samples for certain classes, which is why we need the class embedding.

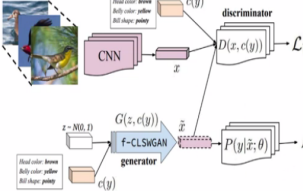
However, in few-shot learning, you already have samples from some of the classes and they can also be used to help with generation. So, you may not need the class embeddings for few-shot,

you may instead use a generator with just noise itself as the input also, when you extend such an approach to few-shot learning.

(Refer Slide Time: 20:53)



### Feature Generating Networks (f-CLSWGAN)



**Loss Formulation:**

- **GAN Loss:**  

$$\mathcal{L}_{WGAN} = E[D(x, c(y))] - E[D(\tilde{x}, c(y))] - \lambda E[(\|\nabla_{\tilde{x}} D(\tilde{x}, c(y))\|_2 - 1)^2]$$
- **Classification Loss:**  

$$\mathcal{L}_{CLS} = -E_{\tilde{x} \sim p_z} [\log P(y|\tilde{x}; \theta)]$$
- **Final loss:**  

$$L_{total} = \min_G \max_D \mathcal{L}_{WGAN} + \beta \mathcal{L}_{CLS}$$

**Training Classifier:**

- Use pre-trained generator to generate samples novel/unseen class samples conditioned on class embeddings
- Train softmax classifier on train set and generated unseen class image features



Vineeth N B (IIT-H)

§12.1 Zero-shot and Few-shot Learning

24 / 33

How is such a GAN trained? So, this method uses two components CLS and WGAN and that is the reason for its name. The WGAN component uses a standard GAN loss, but uses a variant of a GAN known as WGAN, or Wasserstein GAN, which is a variant that tries to mitigate the mode collapse issue in GANs. That is the GAN loss that is used here. And there is a classification loss that is used to classify the generated features. So, you can see here that you have some images in your dataset, you have a CNN, whose output gives you some real features  $x$ .

On the other hand, you have your generator, which takes in a noise vector, and some class information, and also generates some image features  $\tilde{x}$ . A discriminator then takes an  $x$  and  $\tilde{x}$  and the conditional information class and says whether these  $x$  and  $\tilde{x}$ , or are real or fake. But there is also another loss, which takes the generated features and tries to classify them into one of the classes in your class universe. The final loss is a combination of the GAN loss and a classification loss, which is weighted suitably using a coefficient beta.

So just to repeat, to train the classifier, you use a pre trained generator to generate samples of novel and unseen classes conditioned on class embeddings. And the classifier is often a softmax classifier that is trained on the train set, which gives you  $x$ 's and the generated unseen class

image features. So, both  $x$  and  $\tilde{x}$  are together used to train this classifier module in this framework.

Hope that gave you an understanding of Hallucination or Feature Synthesis methods. Now let us move to the third category, which are Parameter Initialization methods, which try to use prior knowledge to alter the search strategy.

(Refer Slide Time: 23:23)

**Parameter Initialization Methods**

**Intuition:**

- Tackle the few-shot learning problem by “learning to fine-tune”
- Learn parameters that transfer via few-gradient steps (fine-tuning) to novel tasks

**Method:**

- For each task/episode  $(D_{train}^i, D_{test}^i)$ , update task-specific parameters  $\Phi_i$  to minimize  $L(\theta, D_{train}^i)$
- Update meta parameter  $\theta$  to minimize  $\sum L(\Phi_i, D_{test}^i)$
- At test time, use few gradient steps to adapt classify novel classes

Credit: Wang et al. Generalizing from a Few Examples: A Survey on Few-Shot Learning, ACM Computing Surveys'20

Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 26 / 33

The key intuition here is learning to fine-tune. We saw learning to compare, we saw learning to augment and now we talk about addressing few-shot using learning to fine-tune. What does that mean? We learn a set of meta model parameters which are common to all classes. In this context, we say classes are tasks, each class is a task. So, we learn a set of meta parameters that can be applied to all classes or tasks in such a way that with very few samples for a given class, you can fine tune those meta model parameters to learn the parameters for that class or task.



So, here is an overall intuition or method of how this works. For each meta learning episode, you have  $D_{train}$  and  $D_{test}$ . You update some task specific parameters  $\Phi_i$  to minimize the loss of the overall parameters for that class. Remember, you have  $\theta$ , which is an overall set of model parameters, which you initialize your model with, then you fine tune that to solve only one particular class in your current meta learning episode. Then, across all other tasks in a given meta

learning episode, you now update the meta parameter  $\theta$  to minimize the loss across all of those classes.

So, you see here,  $\Phi_i$  is also here, you can see that  $\Phi_i$  is also here in this diagram, which is the model parameter for a specific task.  $\phi_3$  is the model parameter for another specific task.  $\phi_2$  is the model parameter for another specific task, we want to learn a  $\theta$ , which is here, which can easily be fine tuned to get  $\phi_1$ ,  $\phi_2$  or  $\phi_3$ . Once you learn  $\phi_1$ ,  $\phi_2$ ,  $\phi_3$  in a given meta learning episode, you now update  $\theta$  as a weighted sum of each of these losses corresponding to each of those tasks.

What happens at test time, if you have a few samples from one of those few-shot classes, you have already learned  $\theta$ , which are your meta model parameters, you can now fine tune  $\theta$  using those few samples to get the model for that particular few-shot class at test time.

(Refer Slide Time: 26:04)

### MAML<sup>4</sup>

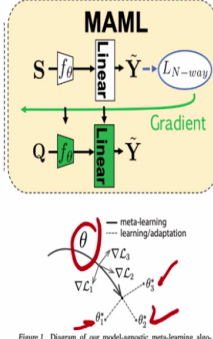


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation  $\theta$  that can quickly adapt to new tasks.

**Key Idea:** Acquire task-specific parameters ( $\Phi_i$ ) through optimization

**Formulation**

- Learn prior such that model can adapt to new tasks  
One form of prior knowledge: **Parameter Initialization**
- **Task-specific Update:**  
 $\theta'_i = \theta - \alpha \nabla_{\theta} L(\theta, D_{train}^i)$  (Single or multiple SGD updates)
- **Meta Fine-tuning:**  
 $\theta = \theta - \beta \nabla_{\theta} \sum_i L(\theta'_i, D_{test}^i)$   
 $\theta = \theta - \beta \nabla_{\theta} \sum_i L(\theta - \alpha \nabla_{\theta} L(\theta, D_{train}^i), D_{test}^i)$  (Second-order derivatives)

<sup>4</sup>Finn et al, Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, ICML 2017

Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 27 / 33

Let us see this in more detail. The most popular approach in this category is called MAML, which stands for Model Agnostic Meta Learning. There have been several improvements of MAML over these years. But we will talk about the basic method where the idea is very similar. The task specific parameters, say  $\Phi_i$  are obtained through optimization. The overall formulation goes like this. Remember, the goal of this category of methods is to learn a prior such that the model can easily adapt to new tasks. Remember, we are talking about this in the context of few-shot learning, where we expect a few samples to be there for those few-shot classes.

And by prior here, we mean a good parameter initialization, which is what  $\theta$  or the meta model parameters try to achieve. So, your task specific update is you start by initializing  $\theta$ , which with any initialization, and in each meta learning episode, you update the task specific parameters, which are  $\theta'_i$ , which are given by  $\theta - \alpha \nabla_{\theta} L(\theta, D_{train}^i)$ . So, that gives you the parameters for that specific class in that Meta learning episode.


Once this is done, the  $\theta$  is then updated using  $\theta - \beta \nabla_{\theta} \sum_i L(\theta'_i, D_{test}^i)$ . In the, of course, now you are getting your images from the test set. Visually, you can look at it as we just explained on the previous slide, you are trying to learn a  $\theta$ , which if fine tuned, can easily get you  $\theta_1^*$ ,  $\theta_2^*$  and  $\theta_3^*$ , which are the ideal model parameters for class 1, class 2 and class 3.



So, by meta model parameters, we mean the theta, which can be easily fine tuned. So, if you observe the task specific update, and the meta update, you would notice that you can now replace  $\theta_i'$ , which is inside this loss, as  $\theta - \alpha \nabla_{\theta} L(\theta, D_{train}^*)$ . And when you substitute it that way, you now see that you have  $\theta = \theta - \beta \nabla_{\theta} \sum_i L(\theta - \alpha \nabla_{\theta} L(\theta, D_{train}^i), D_{test}^i)$ .

So in a sense, this becomes equivalent to almost doing second order derivatives on  $\theta$ . However, this gives us a efficient and effective way to be able to compute the updates to theta using this meta learning approach.

(Refer Slide Time: 29:10)




### MAML: Few-shot Algorithm

**Algorithm 2** MAML for Few-Shot Supervised Learning

**Require:**  $p(\mathcal{T})$ : distribution over tasks  
**Require:**  $\alpha, \beta$ : step size hyperparameters

- 1: randomly initialize  $\theta$
- 2: **while** not done **do**
- 3: Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$
- 4: **for all**  $\mathcal{T}_i$  **do**
- 5: Sample  $K$  datapoints  $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$  from  $\mathcal{T}_i$
- 6: Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  using  $\mathcal{D}$  and  $\mathcal{L}_{\mathcal{T}_i}$  in Equation (2) or (3)
- 7: Compute adapted parameters with gradient descent:  
 $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
- 8: Sample datapoints  $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$  from  $\mathcal{T}_i$  for the meta-update
- 9: **end for**
- 10: Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$  using each  $\mathcal{D}'_i$  and  $\mathcal{L}_{\mathcal{T}_i}$  in Equation 2 or 3
- 11: **end while**




Vineeth N B (IIT-H)
§12.1 Zero-shot and Few-shot Learning
28 / 33

Here is an Algorithmic view of the same methodology. You randomly initialize  $\theta$ , you sample a set of tasks in a given meta learning episode. For the tasks in that meta learning episode, you sample a set of data points from each of those classes in that remember classes and tasks here mean the same thing. You sample a set of data points from each of those classes in that meta learning episode.

You evaluate the gradient of the loss with respect to each of those classes, and update the task specific parameters  $\theta_i'$ , and then you sample a set of data points from the same classes for the meta update. And then you do the meta update based on these samples by combining the gradients for all of the tasks specific losses, remember, each task is a class here. And that gives

you your meta model parameter update. That completes one Meta learning episode. And then you again repeat and take your next set of tasks, not here, here the next set of tasks and then you repeat this process.

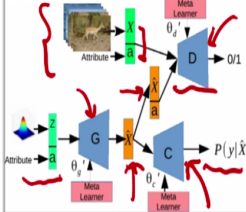
(Refer Slide Time: 30:29)



### Meta-Learning for Generalized Zero-Shot Learning<sup>5</sup>


**Idea:**

- Learn a GAN conditioned on class attributes and train using meta-learning framework (MAML) to facilitate generalization to novel classes
- Mimic ZSL setup during training  $\Rightarrow$  for each task/episode  $T_i = T_{tr}, T_{val}$ , classes of  $T_{tr}$  and  $T_{val}$  are disjoint



**Method:**

- Consider a GAN coupled with classifier module (to classify examples generated by generator) and three meta-learners (Generator  $G$ , Discriminator  $D$ , and Classifier  $C$ ) as shown
- Let  $\theta_d$ ,  $\theta_g$  and  $\theta_c$  be parameters of  $D$ ,  $G$  and  $C$  respectively, and  $\theta_{gc} = [\theta_g, \theta_c]$



<sup>5</sup>Verma et al, Meta-Learning for Generalized Zero-Shot Learning, AAAI 2020

Vineth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 29 / 33

We said that MAML was for few-shot learning, can this be used for Zero-shot learning? A more recent approach shows how this can be done. This is called Meta ZSL, or Meta Learning for generalized Zero-shot learning. Here, the idea is to learn a GAN conditioned on class attributes, and then train using the same Meta learning framework like MAML and then facilitate the generalization to normal classes.

So, the key idea here is you use the meta learning framework where in each episode, you have a set of training classes and test classes. But in each episode, we are going to simulate a Zero-shot learning kind of a setup. Let us see how this is done. So, there is a GAN coupled with a classifier module, which leads to 3 meta learners, you have a generator  $G$ , which is a meta learner learns meta model parameters, you have a discriminator  $D$ , which again learns a meta model parameter, and then you have a classifier, which checks the goodness of the generated features.

So, you can look at this as a combination of feature synthesis methods to do Zero-shot learning and MAML for the meta learning based approach to achieve this achieve this approach. So, let

the parameters for each of these modules D, G and C be  $\theta_d$ ,  $\theta_g$  and  $\theta_c$ . And we denote  $\theta_{gc}$  as the set of parameters combined for  $[\theta_g, \theta_c]$ . Now, how is this learning done?

You have given image and attributes for seen classes, you get X and a by passing the image through a CNN, you would get a set of image features X. You have the corresponding attributes a. Then given noise and attributes, you ask the generator to generate features  $\tilde{X}$ . Given  $\tilde{X}$  and a, the discriminator tries to say whether this is 0 or 1, real or fake. And then you have a classifier, which then says whether the generated features belong to a particular class. So, this is similar to the features synthesis framework.

(Refer Slide Time: 33:01)

**Objective**

$$\max_{\theta_d} \mathbb{E}[D(\mathbf{x}, \mathbf{a}_c | \theta_d)] - \mathbb{E}_{\mathbf{a}_c, \tilde{\mathbf{x}} \sim P_{\theta_g}} [D(\tilde{\mathbf{x}}, \mathbf{a}_c | \theta_d)]$$

$$\min_{\theta_{gc}} -\mathbb{E}_{\mathbf{a}_c, \mathbf{z} \sim \mathcal{N}(0, I)} [D(G(\mathbf{a}_c, \mathbf{z} | \theta_g), \mathbf{a}_c | \theta_d)] + C(y | \tilde{\mathbf{x}}, \theta_c)$$

**Updates**

$$\theta'_d = \theta_d + \eta_1 \nabla_{\theta_d} l_{\in}^D(\theta_d)$$



$$\theta'_{gc} = \theta_{gc} - \eta_2 \nabla_{\theta_{gc}} l_{\in}^{GC}(\theta_{gc})$$

Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 30 / 33

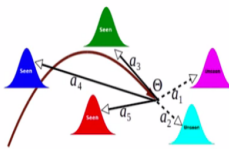
But there is a key component here, which is the meta learning framework which differentiates this and f-CLSWGAN, this is done by the objective now is to for the discriminator to maximize the log likelihood of generating x and  $a_c$  which are the true image features and the corresponding attributes. And minimize the generated image features or the corresponding attributes. On the other hand, the generator tries to fool the discriminator by taking the attributes and noise and generating an  $\tilde{X}$  and we want the discriminator to call this real and you have the classification loss. Beyond this, this objective, you have the meta learning episodes now.

The basic updates, now, given a meta model parameter  $\theta_d$ , you now get  $\theta_d'$  for each of your tasks or classes in a meta learning episode, very similar to MAML.

(Refer Slide Time: 34:08)

### Objective




**Meta-Update**

$$\max_{\theta_d} \sum_{\epsilon \sim p(\mathcal{T})} l^D(\theta'_d) = \max_{\theta_d} \sum_{\epsilon \sim p(\mathcal{T})} l^D(\theta_d + \eta_1 \nabla_{\theta} l^D(\theta_d))$$

$$\theta_d \leftarrow \theta_d + \beta_1 \nabla_{\theta_d} \sum_{\epsilon \sim p(\mathcal{T})} l^D(\theta'_d)$$
  

$$\min_{\theta_{gc}} \sum_{\epsilon \sim p(\mathcal{T})} l^{GC}(\theta'_{gc})$$



$$\theta_{gc} \leftarrow \theta_{gc} - \beta_2 \nabla_{\theta_{gc}} \sum_{\epsilon \sim p(\mathcal{T})} l^{GC}(\theta'_{gc})$$



Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 31 / 33

And then you perform a meta update at the end of that meta learning episode, which combines the losses for all of the tasks in that meta learning episode and updates  $\theta_d$ ,  $\theta_{gc}$  and so on. So, you have you put the f-CLSWGAN in a meta learning framework and you get meta ZSL.

(Refer Slide Time: 34:36)

### Generation and ZSL Classification

**Generation**


- After training, we can generate unseen class examples given class-attribute vectors as:

$$\hat{x} = G_{\theta_g}(z, \mathbf{a}) : \mathbf{a} \in \mathbb{R}^d$$

where  $\mathbf{a}$  denotes class-attributes of unseen classes and  $z \sim \mathcal{N}(0, I)$  and  $\mathbf{z} \in \mathbb{R}^k$

**Classification**

- Once unseen class samples are generated  $\implies$  we train any classifier (e.g., SVM or softmax classifier) with these samples as labeled training data

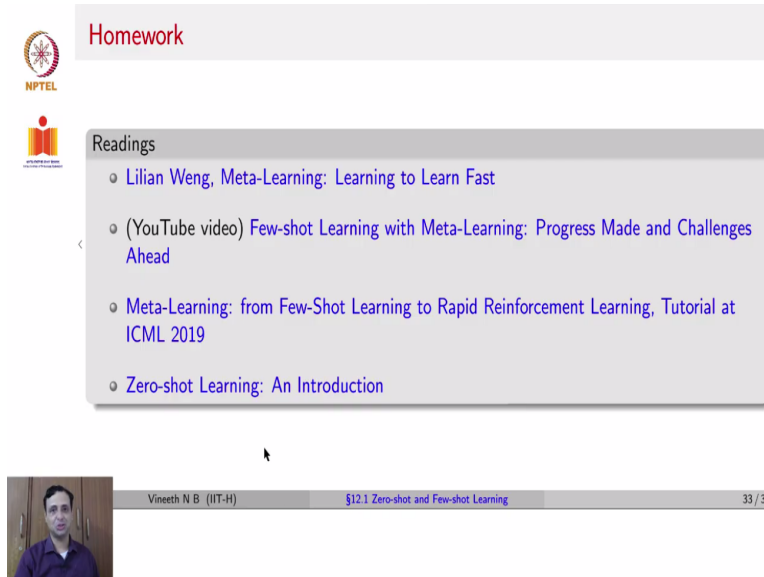


Vineeth N B (IIT-H) §12.1 Zero-shot and Few-shot Learning 32 / 33

Once this is done, the way inference is done is to generate unseen class samples. Using the learned generator you get  $\hat{x}$  assuming that you can give the attributes of the unseen class as input to condition the generator. And once the unseen class samples or the features of the unseen class

samples are generated, you can use the classifier. You can also train any other classifier to predict the class label for a Zero-shot class or any other class in that problem being considered.

(Refer Slide Time: 35:13)



The screenshot shows a presentation slide with a light gray background. At the top left is the NPTEL logo. The word "Homework" is written in red at the top. Below it, the word "Readings" is written in black. A list of four items follows, each preceded by a small blue circle:

- [Lilian Weng, Meta-Learning: Learning to Learn Fast](#)
- (YouTube video) [Few-shot Learning with Meta-Learning: Progress Made and Challenges Ahead](#)
- [Meta-Learning: from Few-Shot Learning to Rapid Reinforcement Learning, Tutorial at ICML 2019](#)
- [Zero-shot Learning: An Introduction](#)

At the bottom left is a small video thumbnail of a man. At the bottom right, a footer bar contains the text "Vineeth N B (IIT-H)", "§12.1 Zero-shot and Few-shot Learning", and "33 / 33".

Hope that gave you an overview of different kinds of methods for few-shot and zero-shot learning. Although there are far more, if you would like to know more and understand more, please go through this excellent tutorial, blog article on Meta Learning, Learning to Learn Fast by Lillian Weng, a nice YouTube video on Few-shot learning with Meta Learning, a tutorial at ICML 2019 on Meta Learning and a very nice introduction to Zero-shot learning.