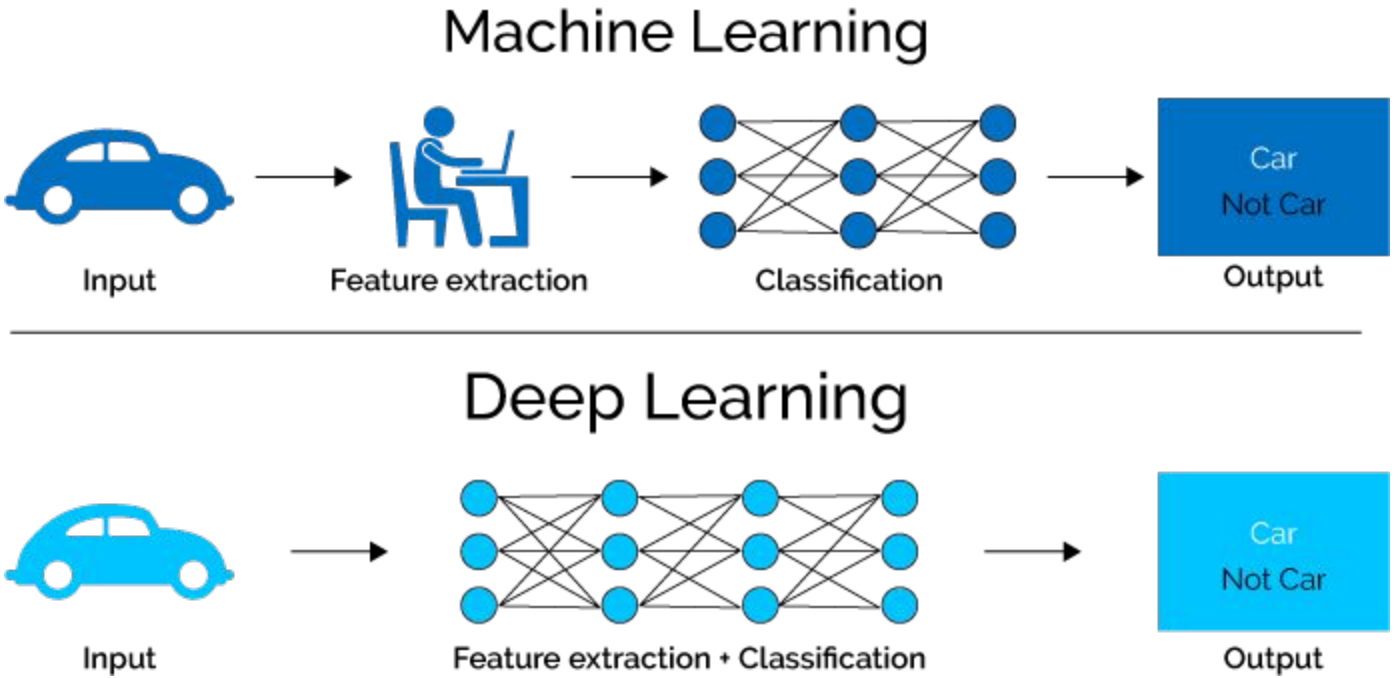
Few-shot learning

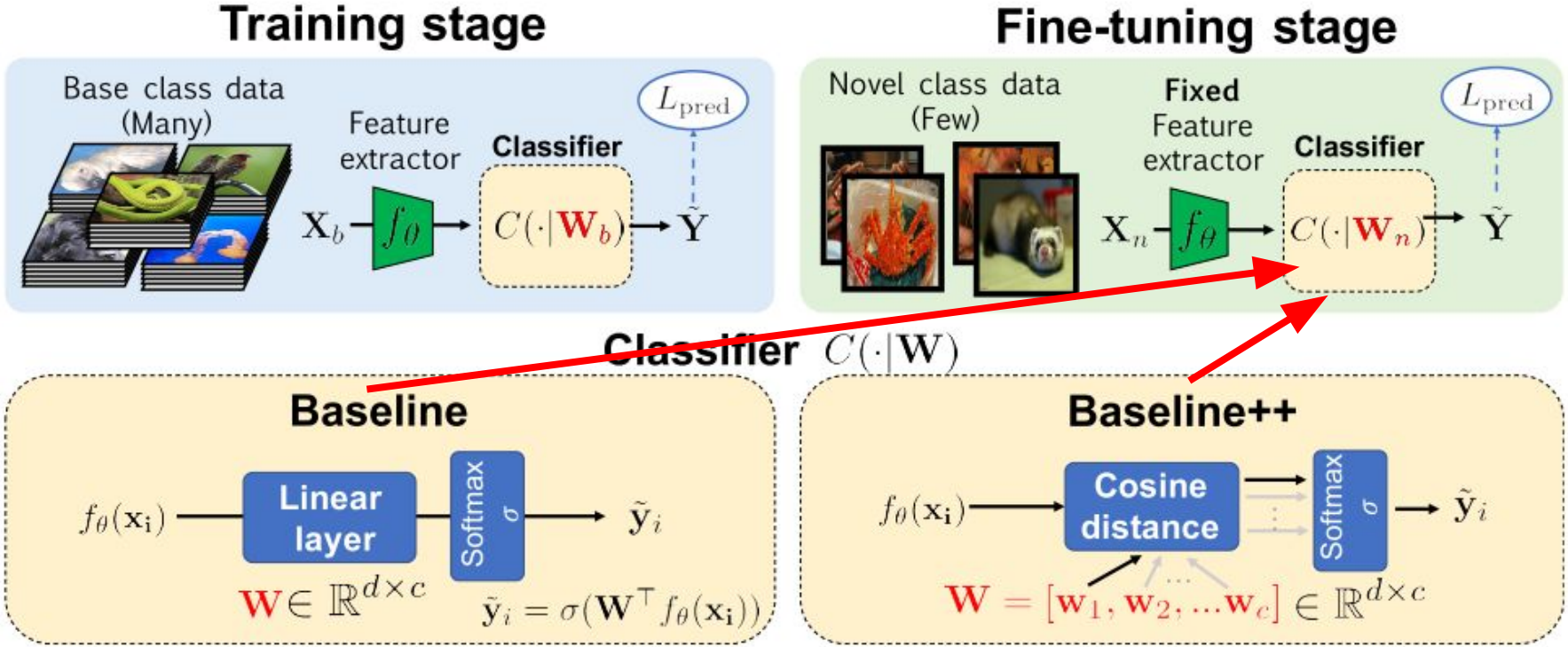
Petr Lorenc
petr.lorenc@cvut.cz

https://docs.google.com/presentation/d/17I2PSG4Q5NxoFDReuigiM6xp7aaybuvq7TbMzecqYgo/edit?usp=sharing

Definition

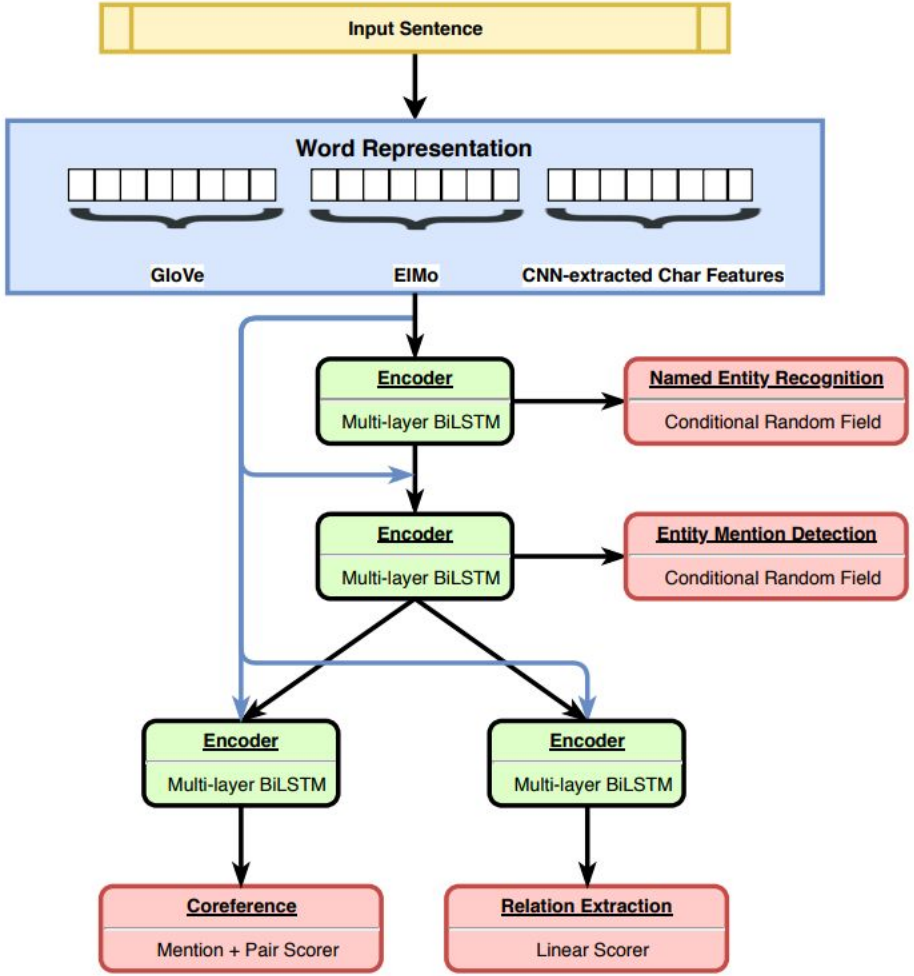

Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Definition - Transfer learning in general



Transfer learning
- multi-task

- Training with more task together or switching from one task to another per epoch
- Can substitute not enough data for only one task

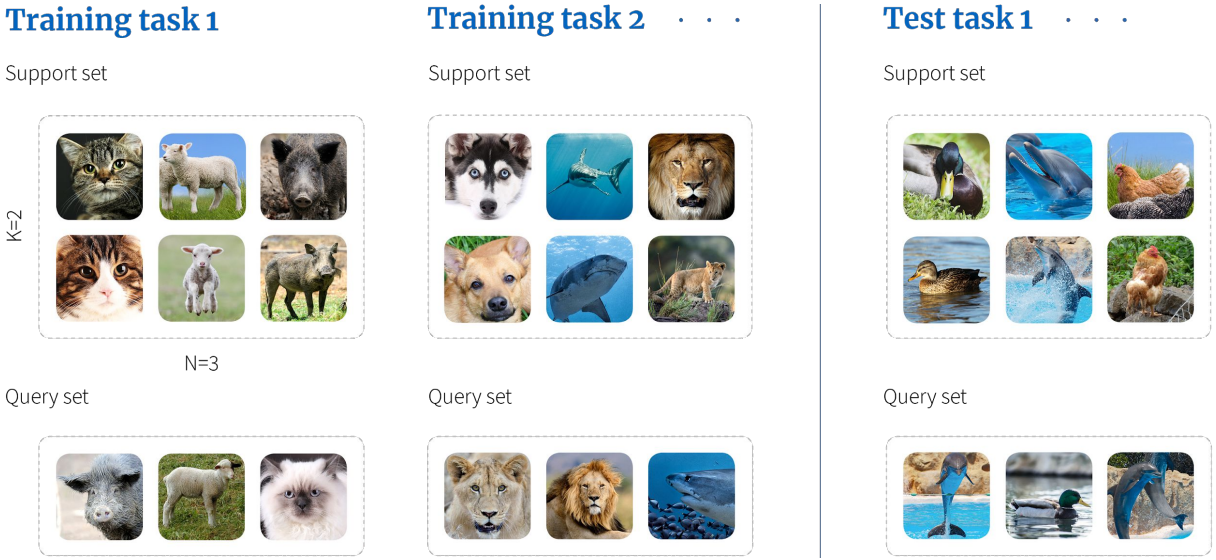


Definition

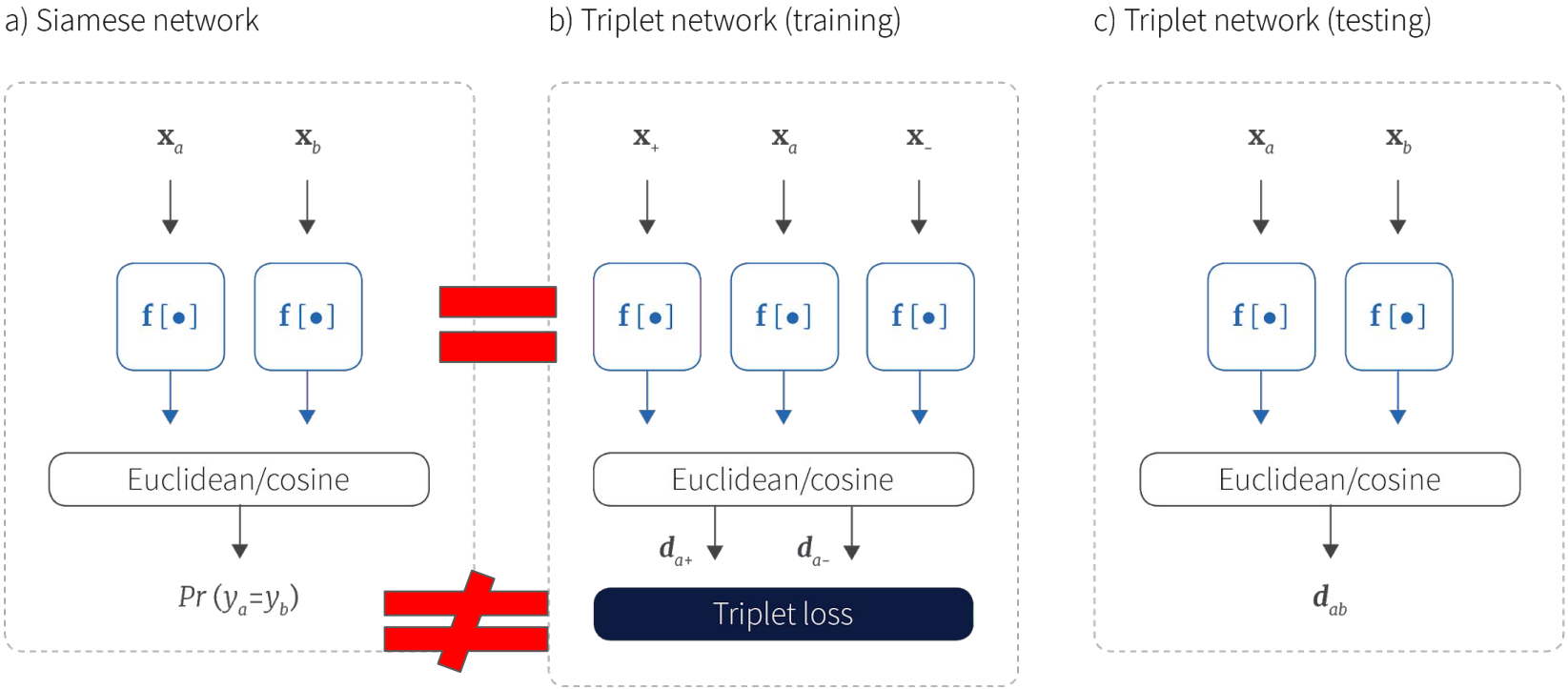
- **N-way-K-shot classification** = each task includes N classes with K examples of each class **for training**
  - few-shot learning
  - one-shot learning
  - zero-shot learning
  - “Less Than One-Shot Learning”

Definition

- N-way-K-shot classification = each task includes N classes with K examples of each class

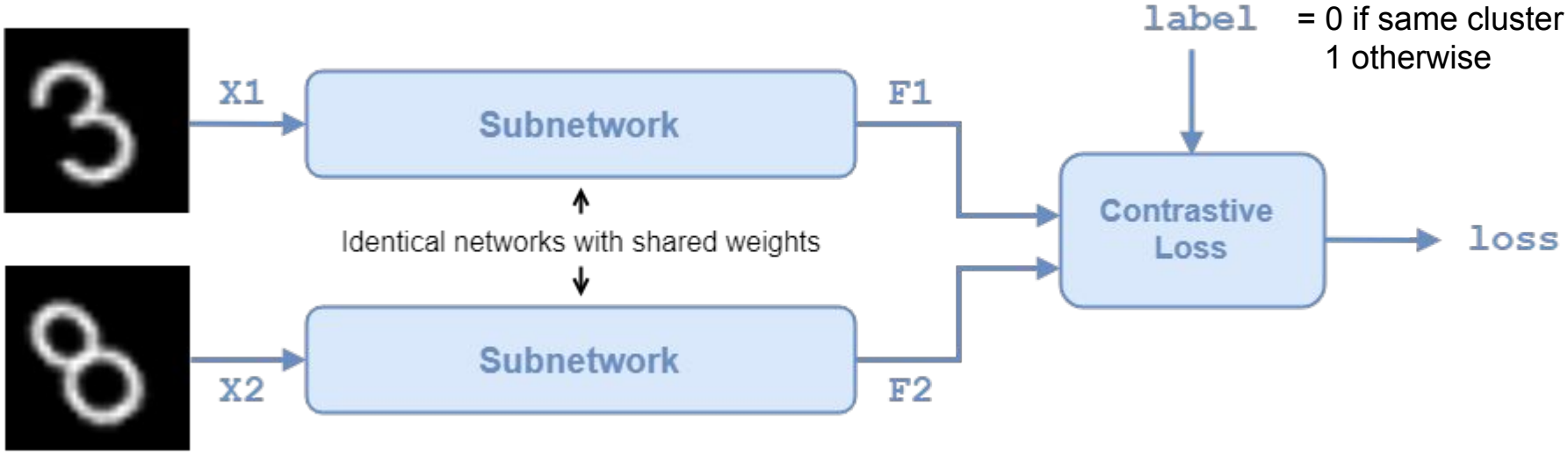


Transfer learning related to few shot learning



Siamese networks

$$\text{Binary Cross-entropy} = -\Big(\underbrace{p(x)\cdot\log q(x)}_{\text{This cancels out if the target is 0}} + \underbrace{(1-p(x))\cdot\log(1-q(x))}_{\text{This cancels out if the target is 1}}\Big)$$



Siamese networks

$$\text{Binary Cross-entropy} = -\Big(\underbrace{p(x)\cdot\log q(x)}_{\text{This cancels out if the target is 0}} + \underbrace{(1-p(x))\cdot\log(1-q(x))}_{\text{This cancels out if the target is 1}}\Big)$$

X1

Subnetwork

F1

label

= 0 if same cluster
1 otherwise

Identical networks with shared weights

Contrastive Loss

loss

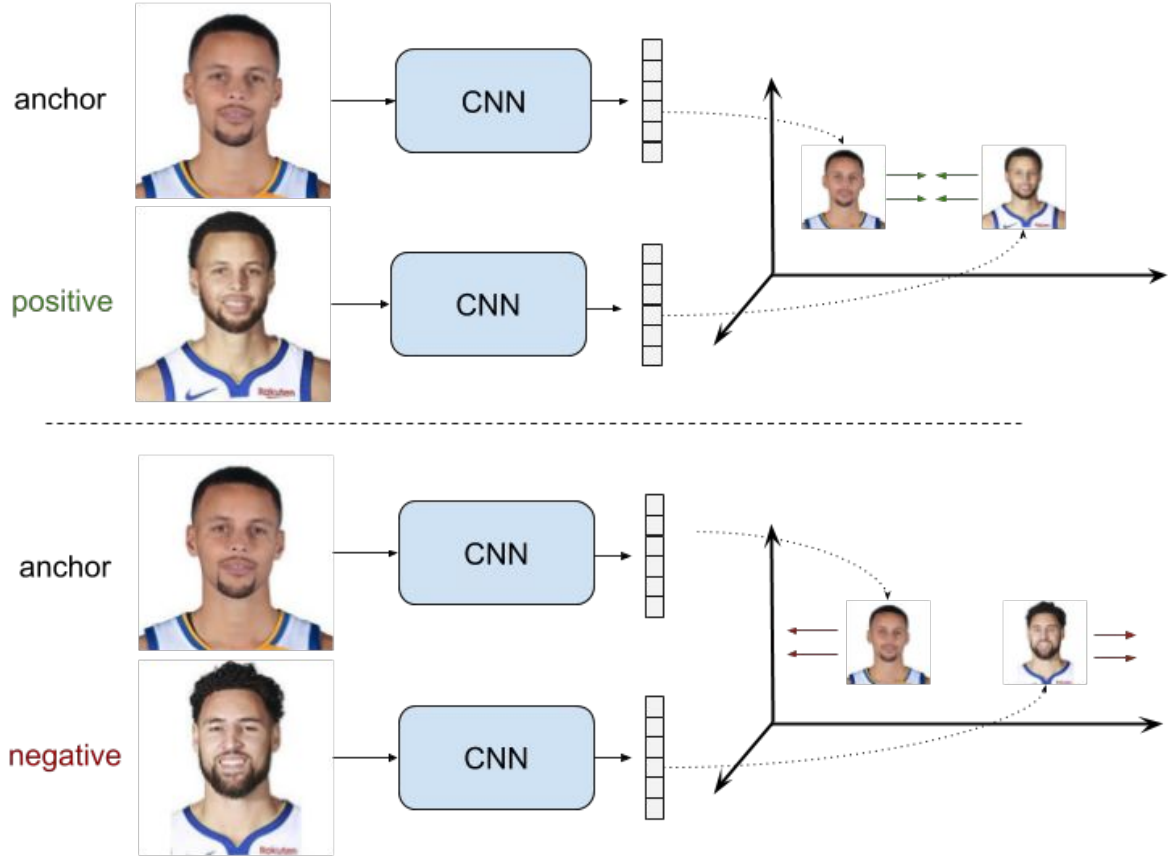
X2

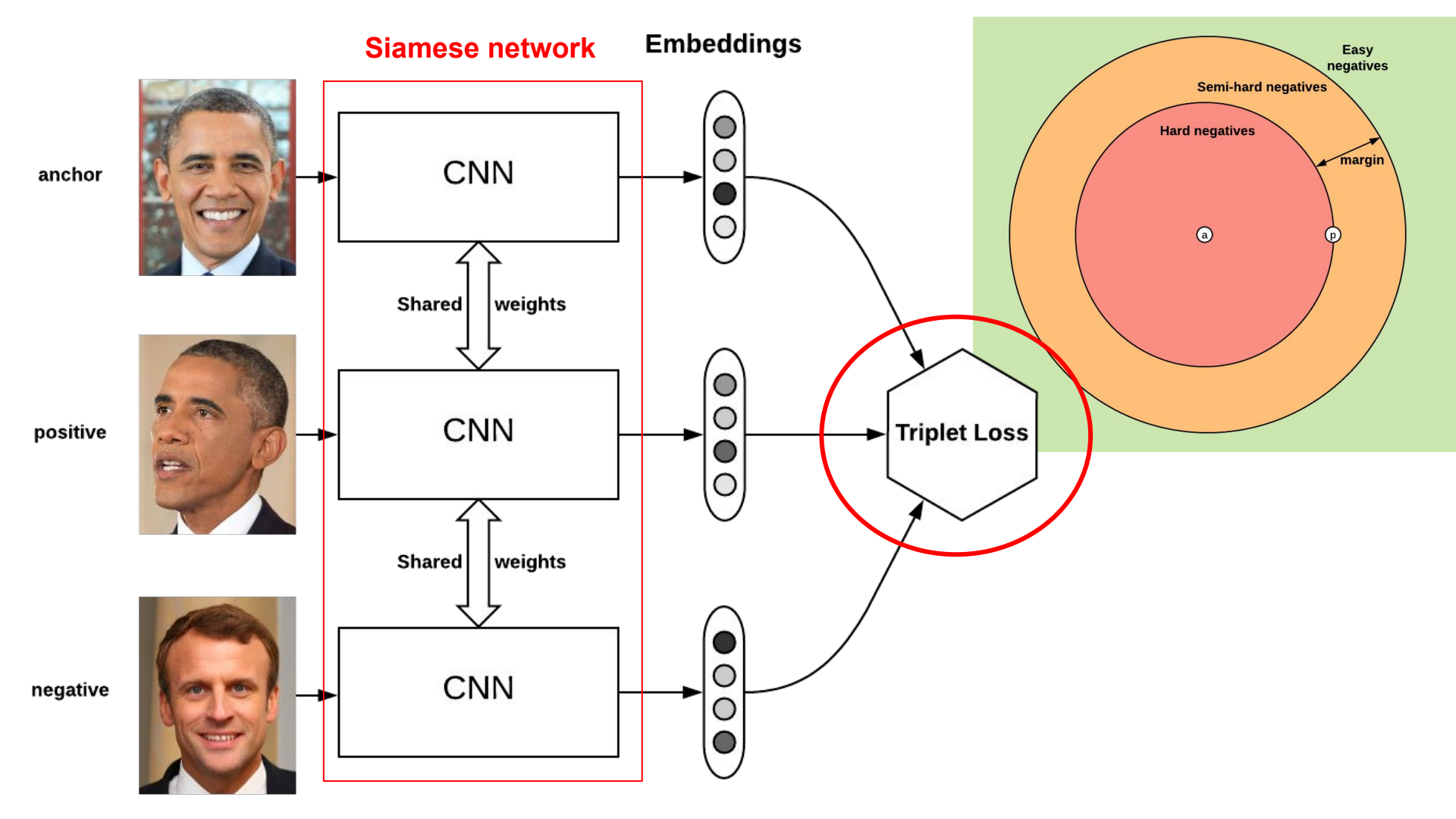
Subnetwork

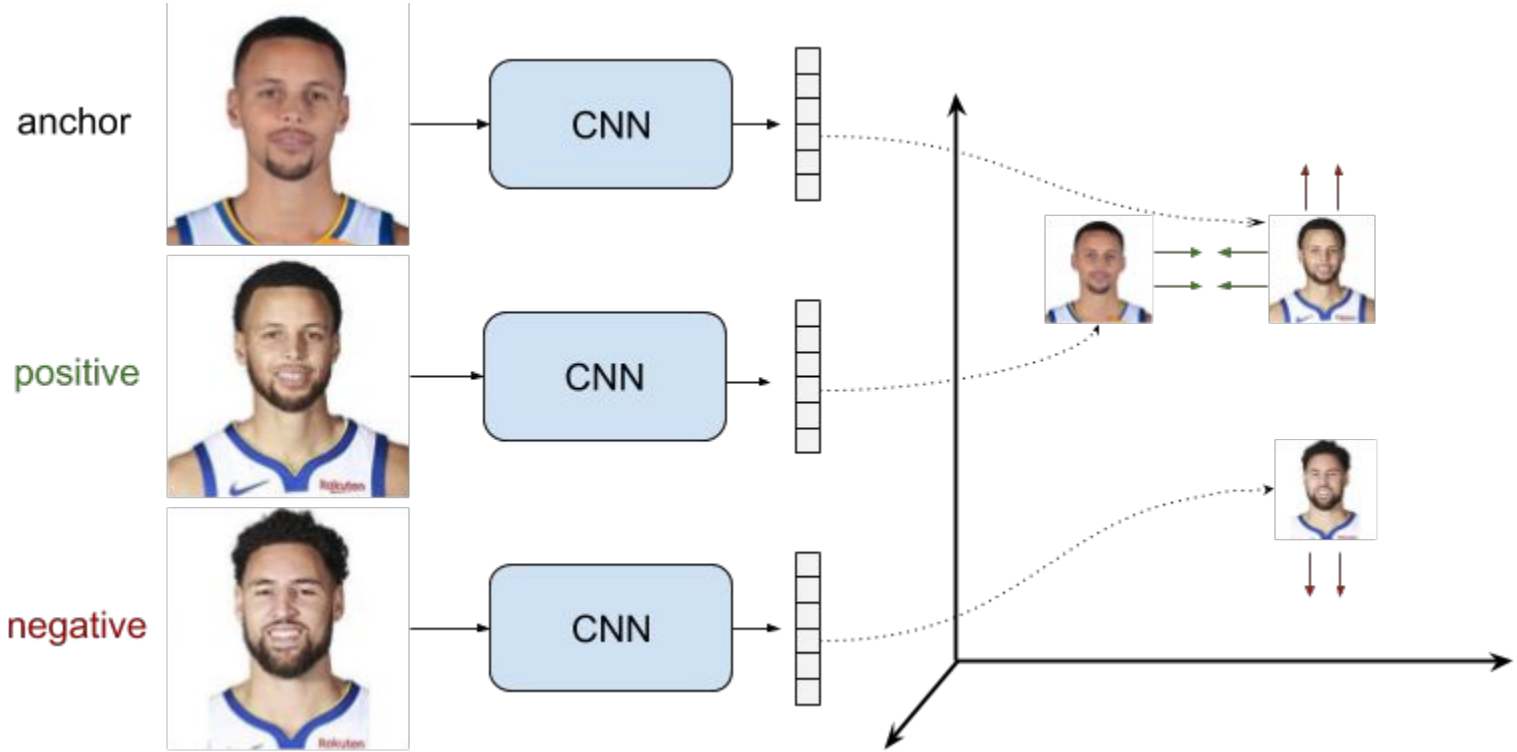
F2

$$(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2$$

Siamese networks




Siamese network
Embeddings
anchor
positive
negative
CNN
CNN
CNN
Shared weights
Shared weights
Triplet Loss
Easy negatives
Semi-hard negatives
Hard negatives
margin
a
p



$$L(r_a, r_p, r_n) = max(0, m + d(r_a, r_p) - d(r_a, r_n))$$

---> 0 ---> m

Ways to go

- **Prior knowledge about similarity**: We learn embeddings in training tasks that tend to separate different classes even when they are unseen.
  - similar to **Siamese networks**

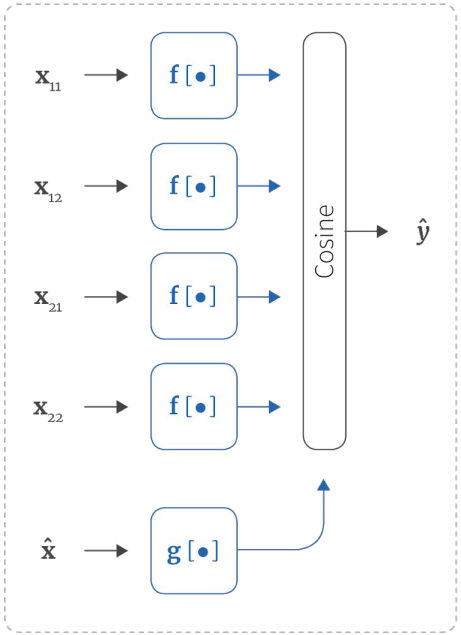
- **Prior knowledge about learning**: We use prior knowledge to constrain the learning algorithm to choose parameters that generalize well from few examples

- **Prior knowledge of data**: We exploit prior knowledge about the structure and variability of the data and this allows us to learn viable models from few examples.

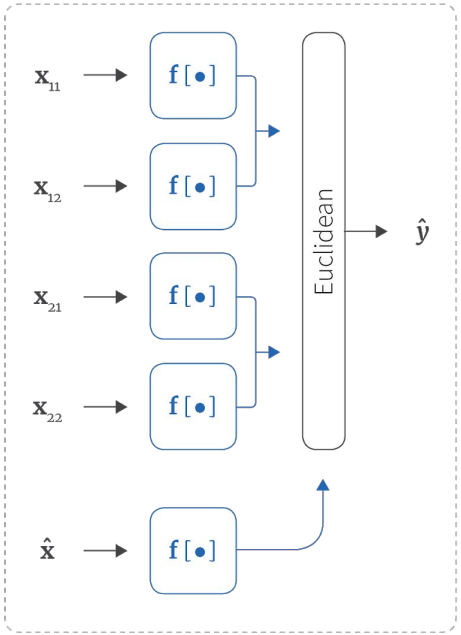
Prior knowledge about similarity

- Based on idea of Siamese/Triplet networks

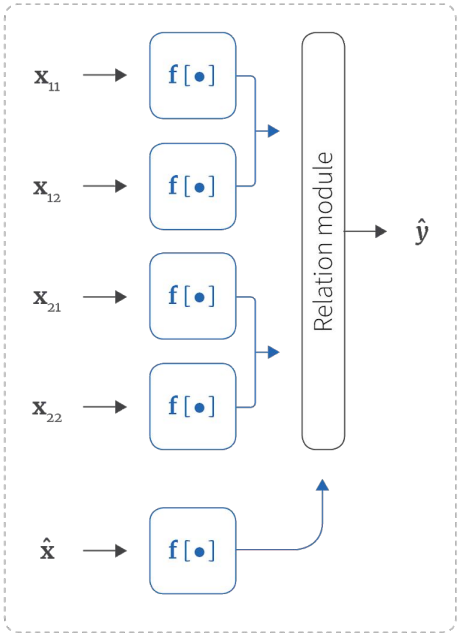
a) Matching network



b) Prototypical network

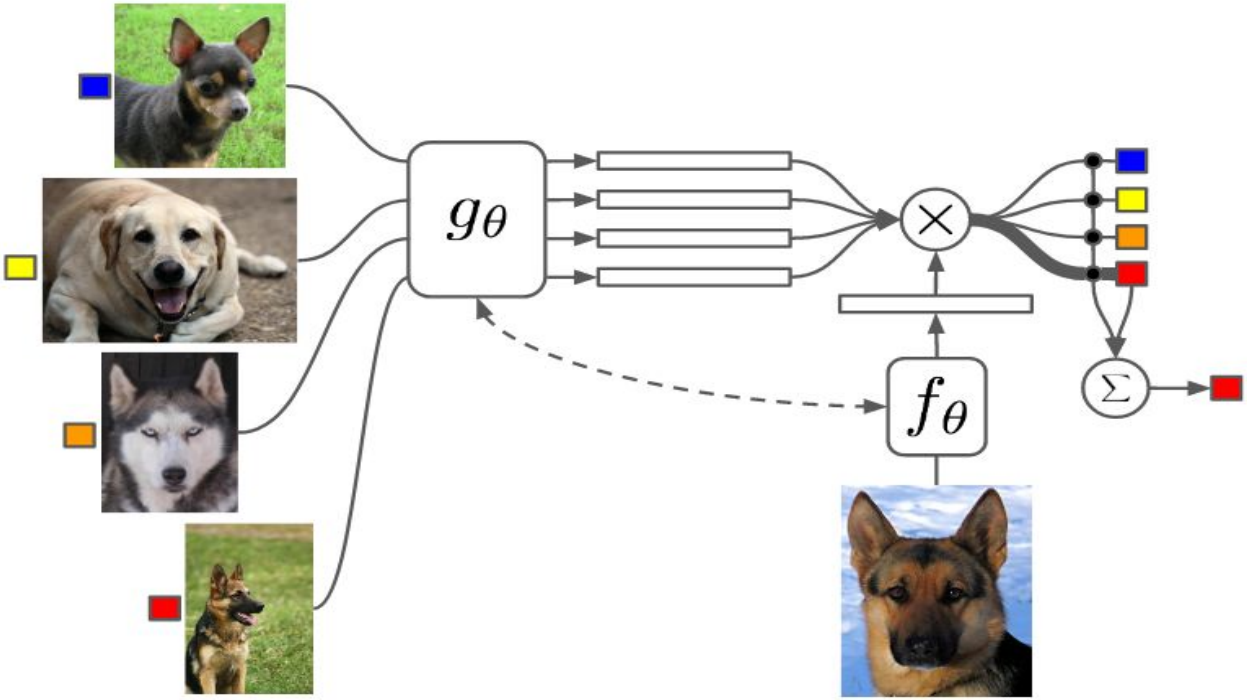


c) Relation network



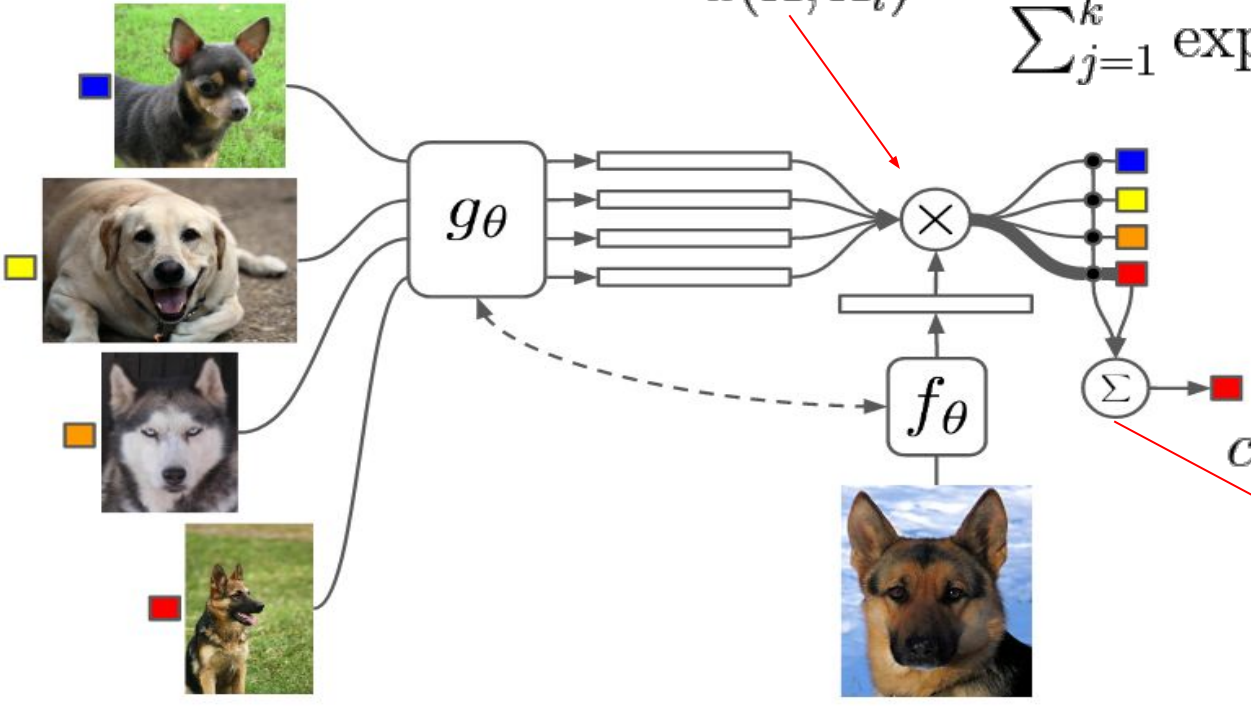
Matching network

4-way-1-shot learning



Matching network

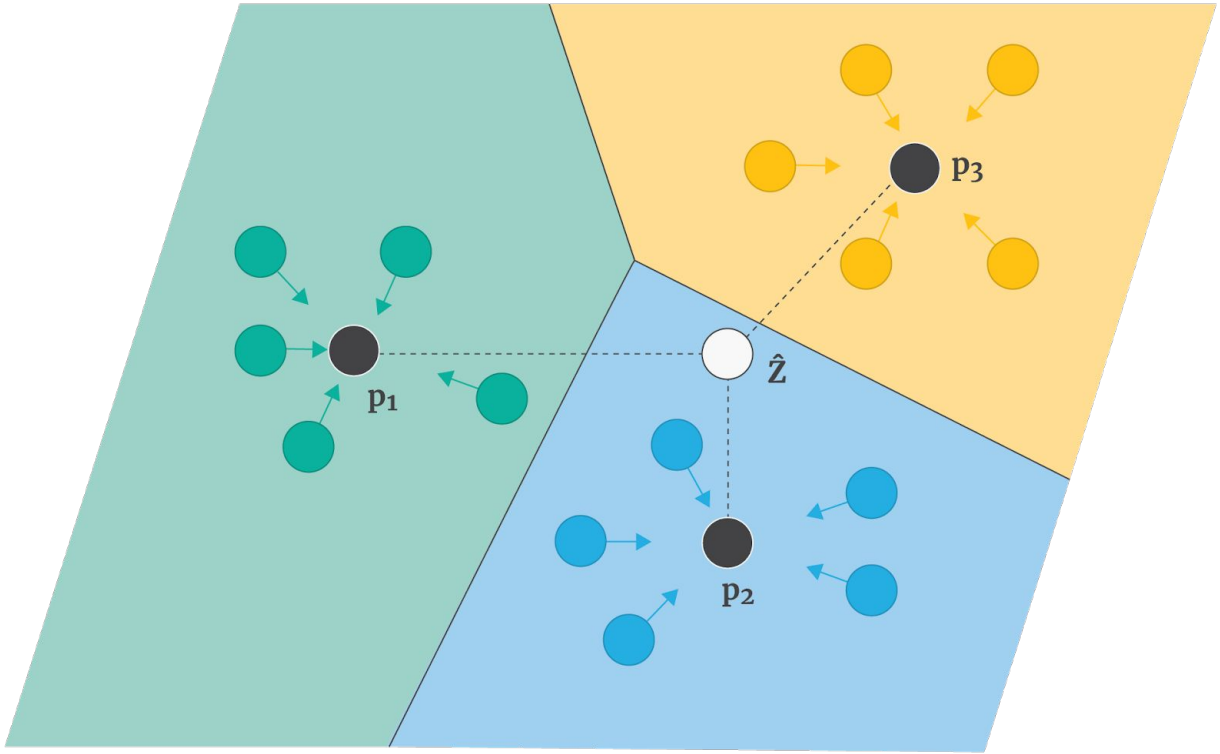
$$a(\mathbf{x}, \mathbf{x}_i) = \frac{\exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_i))}{\sum_{j=1}^{k} \exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_j))}$$



$$c_S(\mathbf{x}) = P(y|\mathbf{x}, S) = \sum_{i=1}^{k} a(\mathbf{x}, \mathbf{x}_i) y_i$$

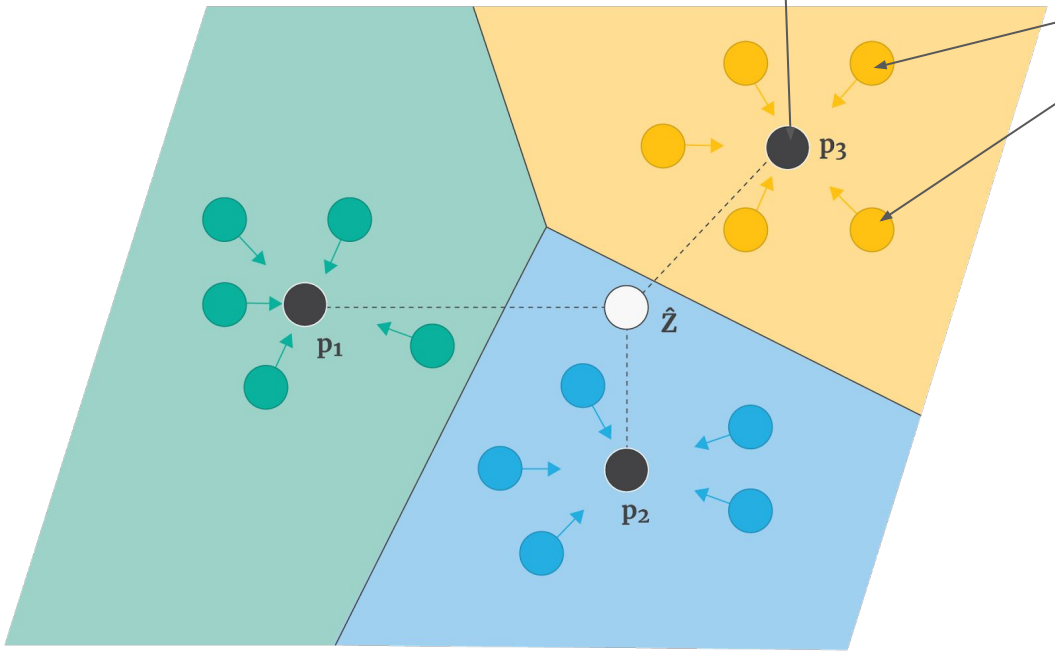
where $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{k}$

Prototypical Networks



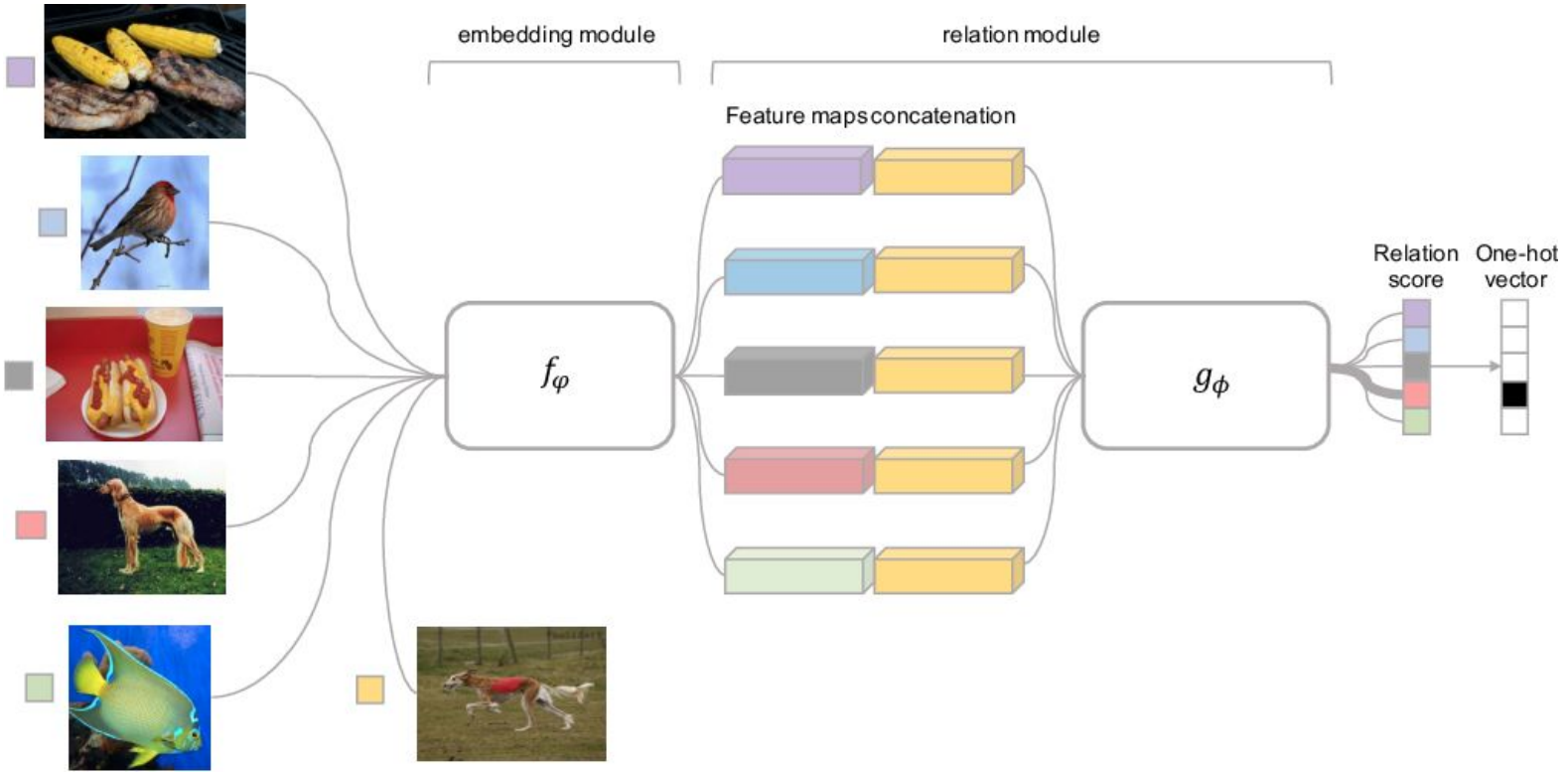
Prototypical Networks

$$\mathbf{v}_c = \frac{1}{|S_c|} \sum_{(\mathbf{x}_i, y_i) \in S_c} f_\theta(\mathbf{x}_i)$$



$$P(y = c|\mathbf{x}) = \text{softmax}(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c)) = \frac{\exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c))}{\sum_{c' \in \mathcal{C}} \exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_{c'}))}$$

Relation Networks



Relation Networks

- The relationship is not captured by a simple L1 distance in the feature space, but predicted by a **CNN classifier g**. The **relation score** between a pair of inputs, x_i and x_j, is $r_{ij}=\mathbf{g}_\phi([x_i, x_j])$ where [.,.] is concatenation.

- The **objective function is MSE loss** instead of cross-entropy, because conceptually RN focuses more on predicting relation scores which is more like regression, rather than binary classification

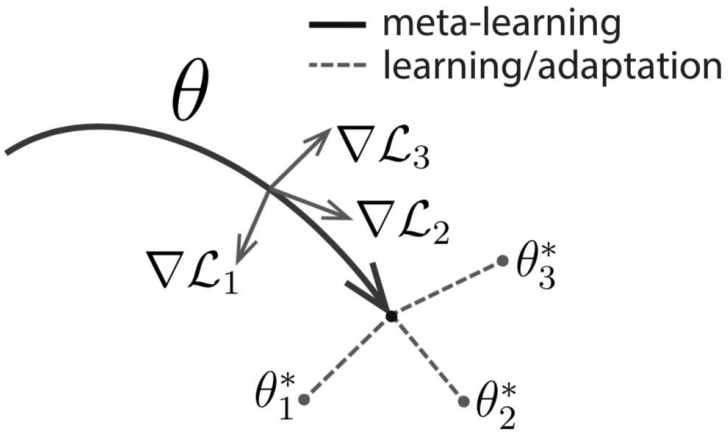
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Prior knowledge about learning

- Learning to initialize
  - MAML - Model-agnostic Meta-learning
  - learn good starting positions for optimization
- Learning to optimize
  - LSTM meta learner
  - models are able to faster optimization with only few examples

MAML

- The aim is to learn a general model that can easily be fine-tuned for many different tasks, even when the training data is scarce.
- Classic update of parameter $= \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$



Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
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**

MSE or CE

$$\text{Cross-entropy} = -\sum_x p(x) \cdot \log q(x)$$

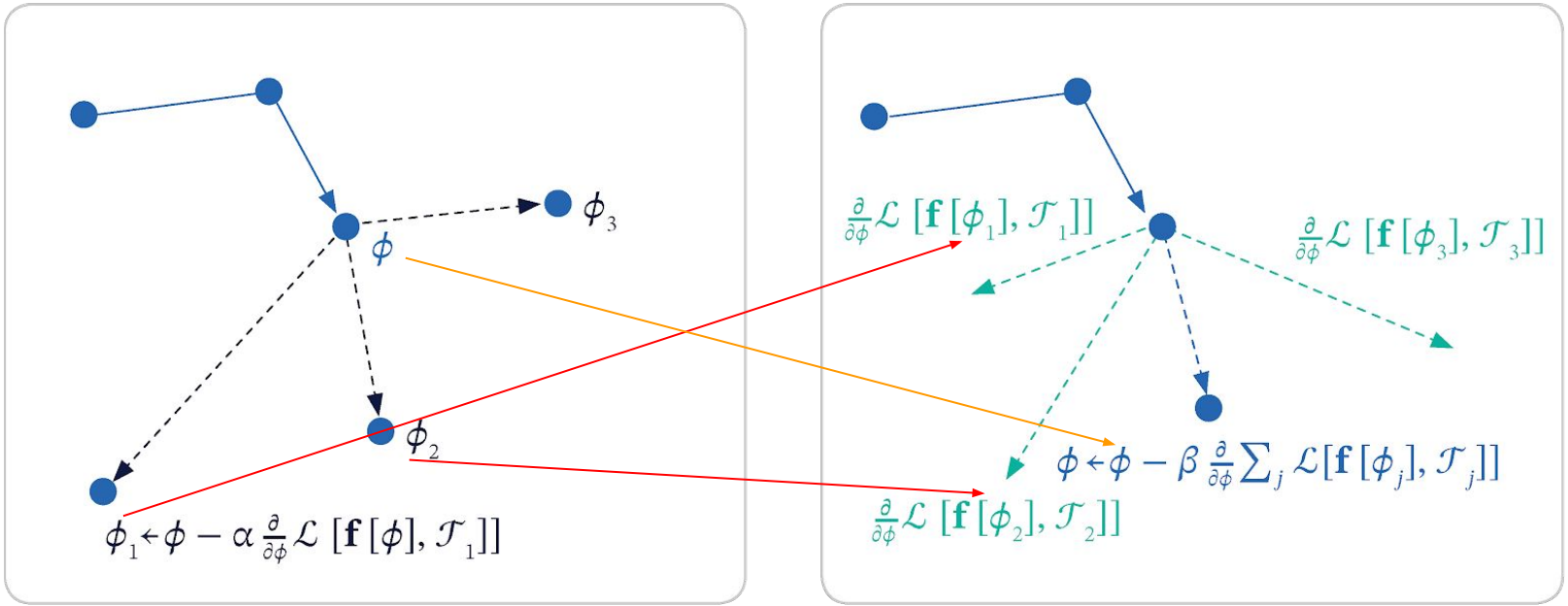
MSE or CE

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

MAML

MAML task learning

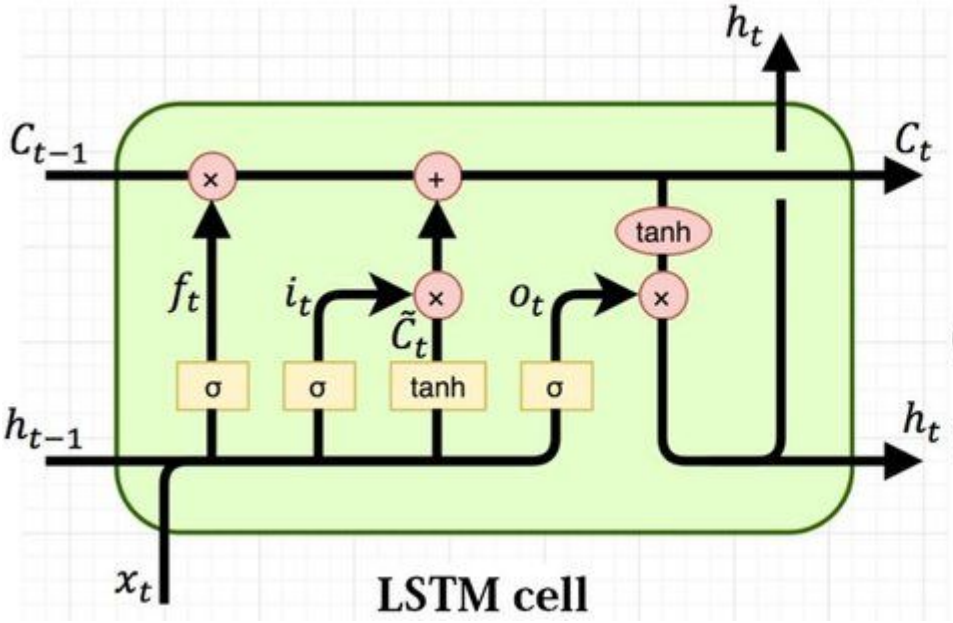
MAML meta-learning



LSTM Meta-Learner

- Classic update param of NN ->
- LSTM:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

LSTM Meta-Learner

- There is similarity between the gradient-based update in backpropagation and the cell-state update in LSTM.
  - $f_t = 1$
  - $i_t = \alpha_t$
- Knowing a history of gradients benefits the gradient update
  - f_t = Neural network output - how much to forget the old value of parameters
  - i_t = Neural network output - learning rate at timestamp t

Algorithm 1 Train Meta-Learner

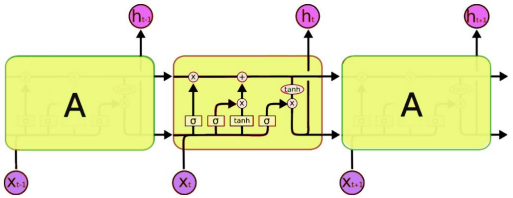
Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$= \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

$\Theta_0 \leftarrow$ random initialization

for $d = 1, n$ **do**
  $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$
  $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters
  for $t = 1, T$ **do**
    $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}
    $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch
    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2
    $\theta_t \leftarrow c_t$ ▷ Update learner parameters
  end for
  $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$
  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch
  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
end for

LSTM Meta-Learner



$(\mathbf{X}_i, \mathbf{Y}_i)$: mini-batches sampled from $D^{(d)}_{\text{train}}$

$D^{(d)}_{\text{test}}$

$(\mathbf{X}_1, \mathbf{Y}_1)$ $(\mathbf{X}_2, \mathbf{Y}_2)$ $(\mathbf{X}_3, \mathbf{Y}_3)$ $(\mathbf{X}_T, \mathbf{Y}_T)$ $(\mathbf{X}, \mathbf{Y})$

θ_0 M_{θ_0} θ_1 M_{θ_1} θ_2 M_{θ_2} θ_{T-1} $M_{\theta_{T-1}}$ θ_T M_{θ_T}

Learner $(\nabla_1, \mathcal{L}_1)$ $(\nabla_2, \mathcal{L}_2)$ $(\nabla_T, \mathcal{L}_T)$

R_{Θ_d} θ_1 R_{Θ_d} R_{Θ_d} θ_{T-1} R_{Θ_d}

$\mathcal{L}(M_{\theta_T}(\mathbf{X}), \mathbf{Y})$

$R_{\Theta_{d+1}}$

Meta-learner : outputs the learner's parameter

Repeats T steps; t = 1, …, D

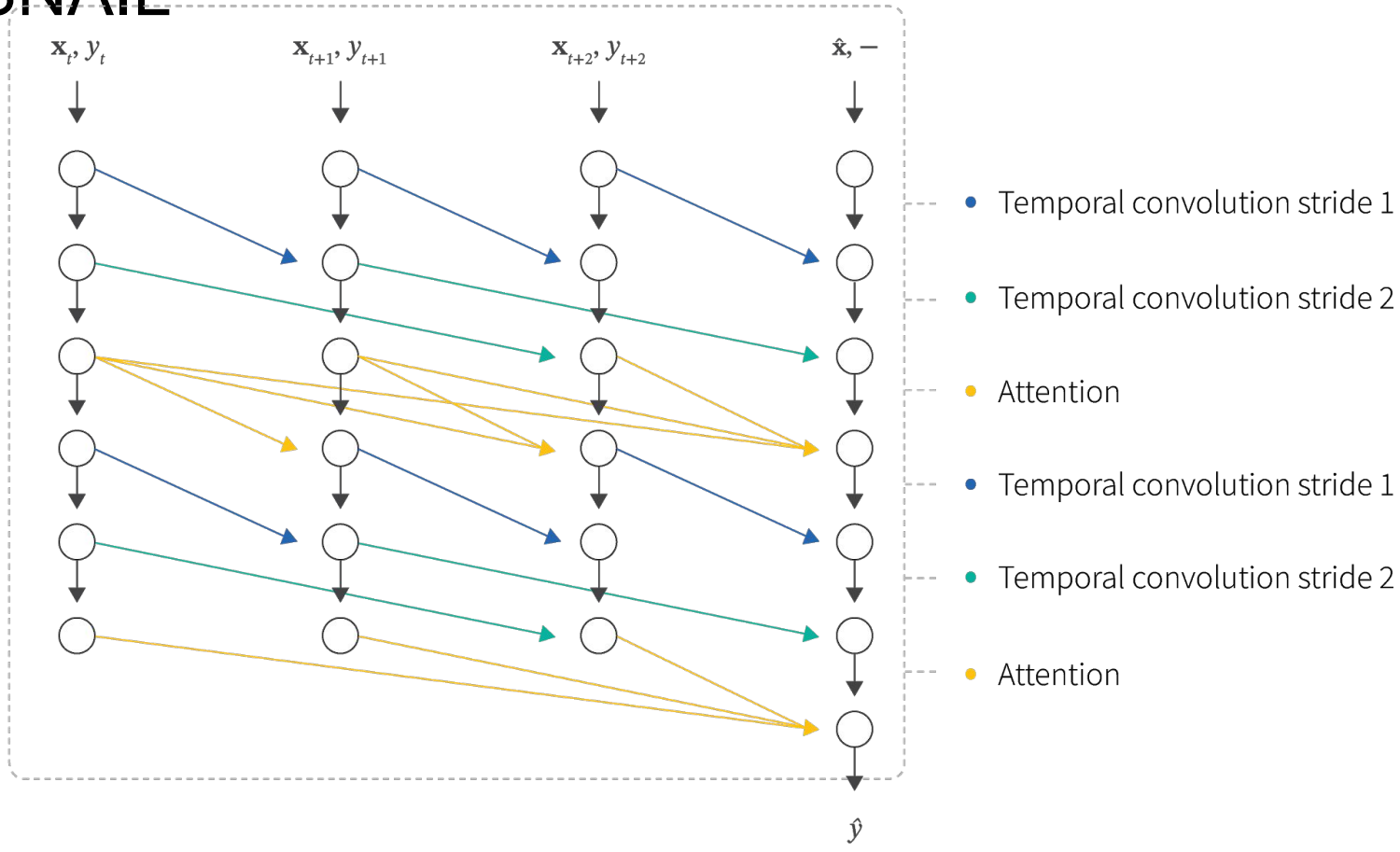
Repeats D steps; d = 1, …, D

Using reinforcement learning

- Similar to LSTM - system learns how best to update the model parameters in unseen test tasks, based on experience gained from a diverse collection of training tasks.
- We train a Recurrent Neural Network controller to choose from **a list of primitive functions and apply to update rule (b, u_1, u_2)**

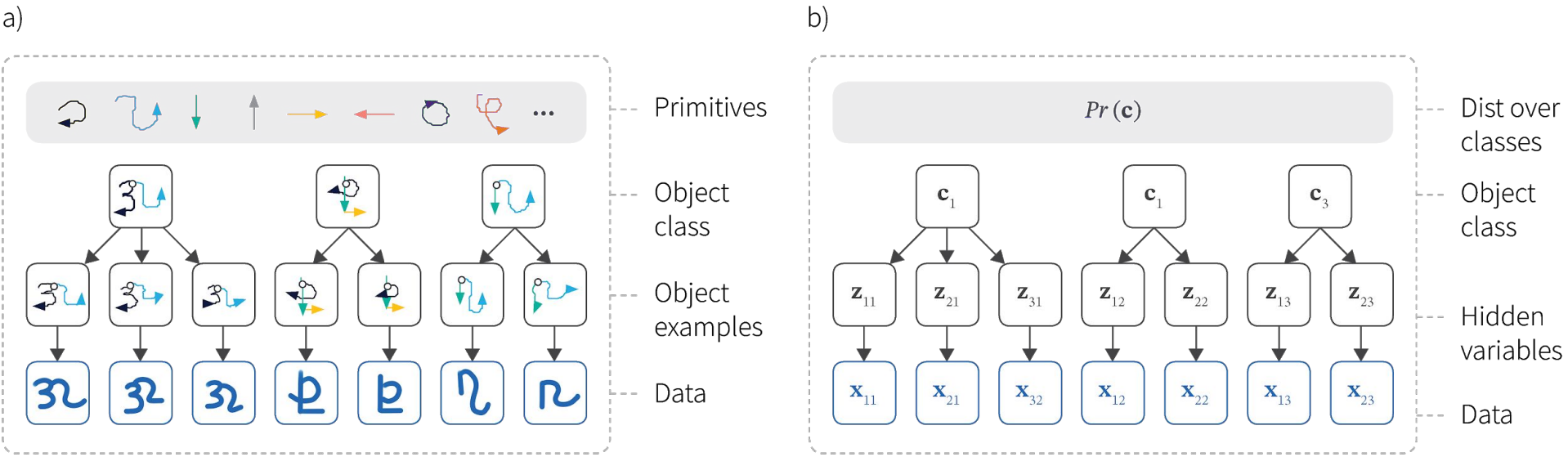
$$\phi_t \leftarrow \phi_{t-1} + \alpha \cdot \mathrm{b}\left[\mathrm{u}_1[o_1], \mathrm{u}_2[o_2]\right]$$

SNAIL


$\mathbf{x}_t, y_t$
$\mathbf{x}_{t+1}, y_{t+1}$
$\mathbf{x}_{t+2}, y_{t+2}$
$\hat{\mathbf{x}}, -$
$\hat{y}$
Temporal convolution stride 1
Temporal convolution stride 2
Attention
Temporal convolution stride 1
Temporal convolution stride 2
Attention

Prior knowledge of data

- Data augmentation



Neural statistician

- takes as input a set of vectors and outputs a vector of summary statistics used **for generative model** - a mean and variance specifying a Gaussian distribution in a latent space -> **generate data**

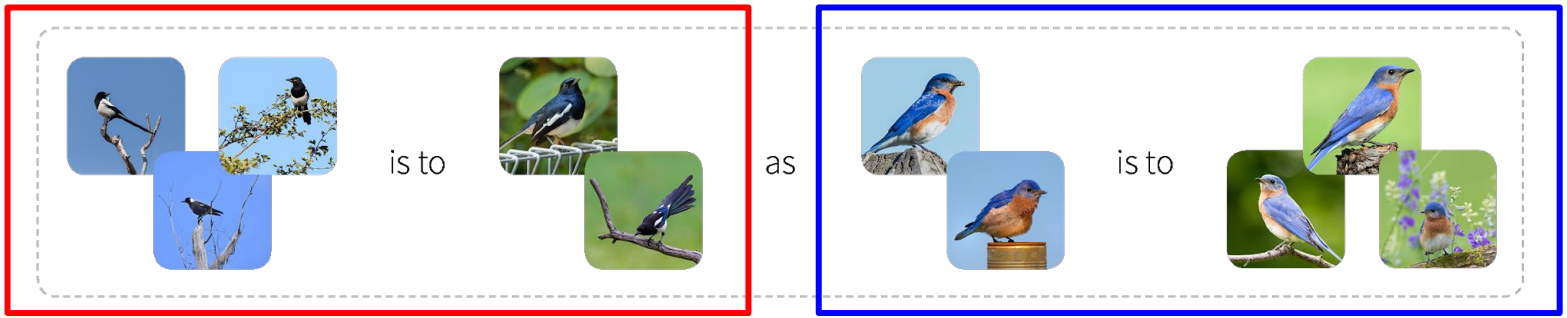
Algorithm 4 K-way few-shot classification

$D_0, \ldots, D_K \leftarrow$ sets of labelled examples for each class
$x \leftarrow$ datapoint to be classified
$N_x \leftarrow q(c|x;\phi)$ {approximate posterior over c given query point}
for $i = 1$ **to** K **do**
  $N_i \leftarrow q(c|D_i;\phi)$
end for
$\hat{y} \leftarrow argmin_i D_{KL}\left(N_i \| N_x\right)$

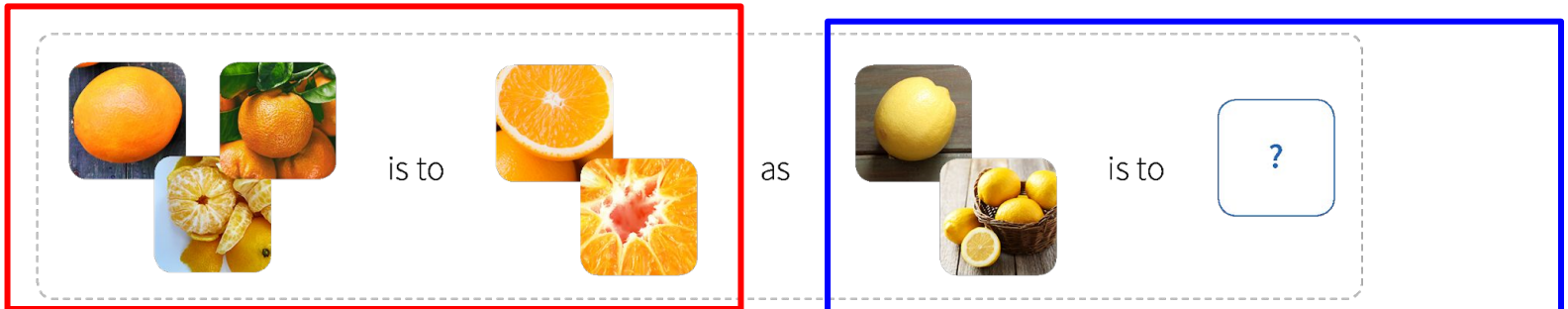
Analogies

- Data augmentation - based on analogy and intra-class variation

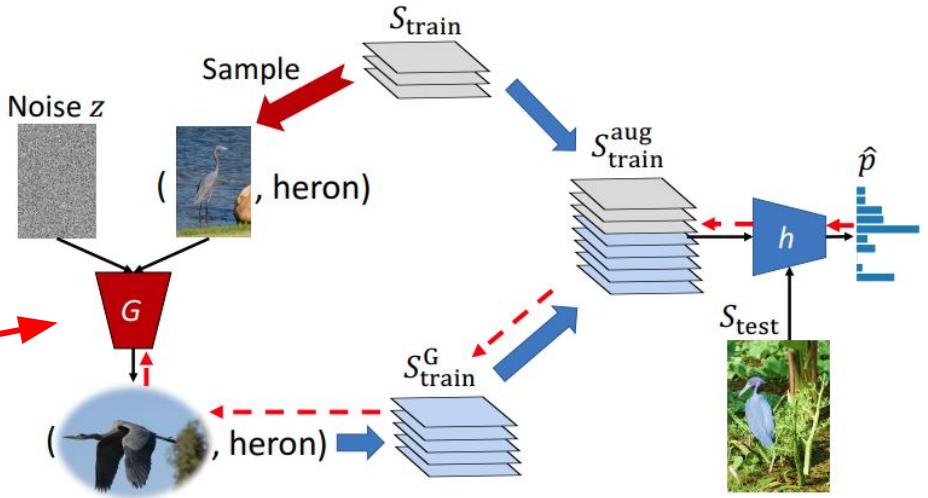
a)

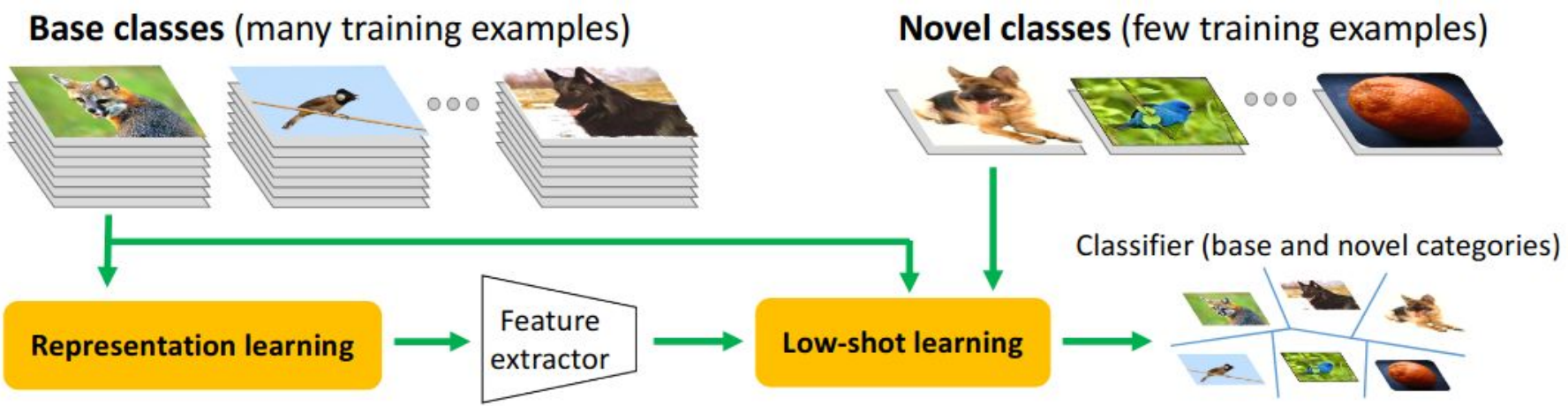


b)



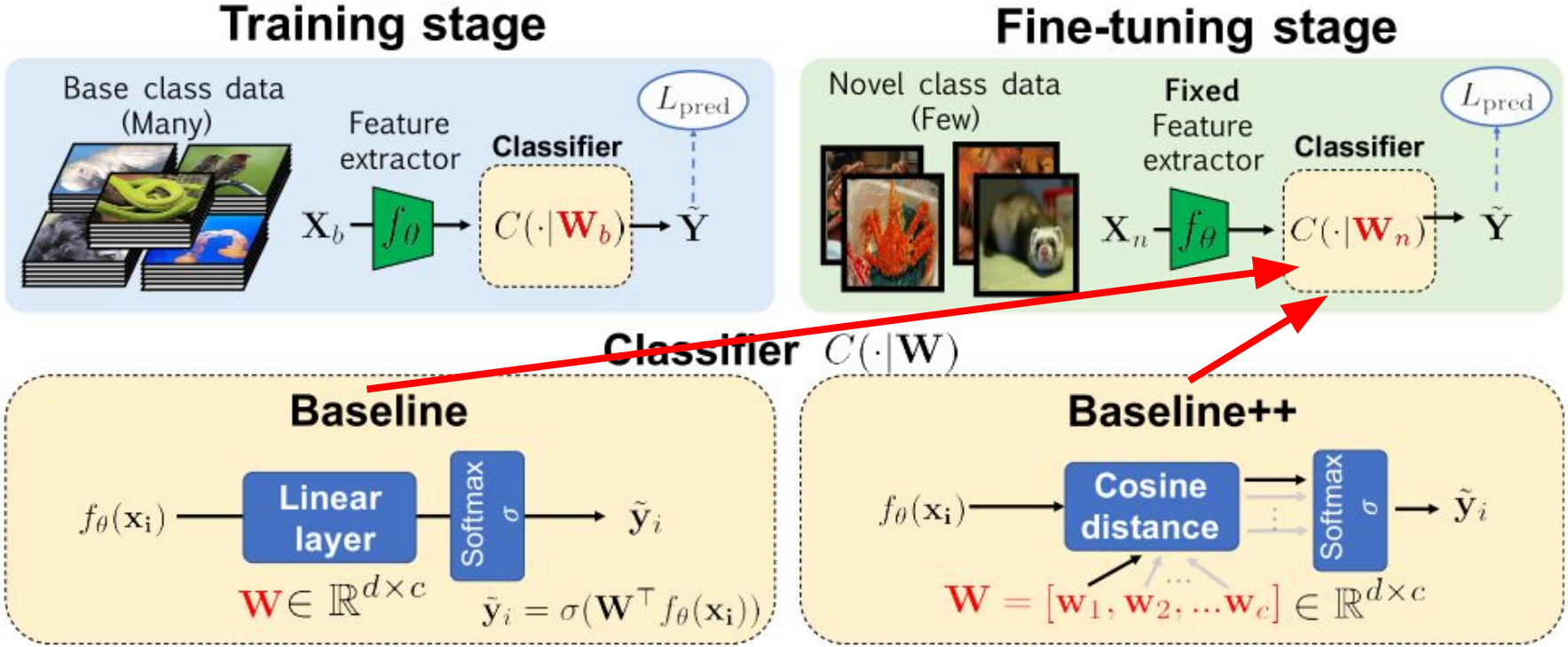
- Better loss function - update
- Hallucinating additional examples
- Standard approach





Prior knowledge about similarity	Prior knowledge about learning	Prior knowledge about the data
Learn embedding to discriminate two unseen classes. *Siamese networks (Koch et al., 2015)* *Triplet networks (Hoffer & Ailon, 2014)*	Learn model with parameters that can easily be fine-tuned to give good results. *MAML (Finn & Levine 2017)* *FOMAML (Finn & Levine, 2017)* *Reptile (Nichol et al., 2018)*	Learn generative model for family of data classes. *Pen-stroke model (Lake et al., 2015)* *Neural statistician (Edwards & Storkey., 2016)*
Learn embedding to discriminate many unseen classes. *Matching networks (Vinyals et al., 2016)* *Prototypical networks (Snell et al., 2017)* *Relation networks (Santoro et al., 2016)*	Learn update rule that encourages good performance with small datasets. *LSTMs (Ravi & Larochelle, 2016)* *Reinforcement learning (Li & Malik, 2016)* *Optimization rules (Bello et al., 2017)*	Learn to synthesize new examples and train with augmented data. *Analogies (Hariharan & Girschick, 2017)* *End-to-end (Wang et al., 2018)*
	Sequence methods. Take entire dataset and test example and predict test label. *Memory-augmented NN (Santoro et al., 2016)* *SNAIL (Mishra et al., 2017)*	

Definition - Transfer learning in general



Method	CUB		*mini*-ImageNet	
	1-shot	5-shot	1-shot	5-shot
Baseline	47.12 ± 0.74	64.16 ± 0.71	42.11 ± 0.71	62.53 ±0.69
Baseline++	60.53 ± 0.83	79.34 ± 0.61	48.24 ± 0.75	66.43 ±0.63
MatchingNet Vinyals et al. (2016)	60.52 ± 0.88	75.29 ± 0.75	48.14 ± 0.78	63.48 ±0.66
ProtoNet Snell et al. (2017)	50.46 ± 0.88	76.39 ± 0.64	44.42 ± 0.84	64.24 ±0.72
MAML Finn et al. (2017)	54.73 ± 0.97	75.75 ± 0.76	46.47 ± 0.82	62.71 ±0.71
RelationNet Sung et al. (2018)	62.34 ± 0.94	77.84 ± 0.68	49.31 ± 0.85	66.60 ±0.69

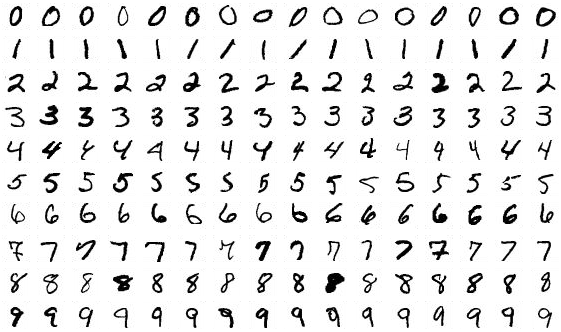
Method	CUB 1-shot	CUB 5-shot	*mini*-ImageNet 1-shot	*mini*-ImageNet 5-shot
Baseline	47.12 ± 0.74	64.16 ± 0.71	42.11 ± 0.71	62.53 ±0.69
Baseline++	60.53 ± 0.83	79.34 ± 0.61	48.24 ± 0.75	66.43 ±0.63
MatchingNet Vinyals et al. (2016)	60.52 ± 0.88	75.29 ± 0.75	48.14 ± 0.78	63.48 ±0.66
Proto		39 ± 0.64	44.42 ± 0.84	64.24 ±0.72
MAM		75 ± 0.76	46.47 ± 0.82	62.71 ±0.71
Relat		34 ± 0.68	49.31 ± 0.85	66.60 ±0.69

	mini-ImageNet →CUB
Baseline	**65.57±0.70**
Baseline++	62.04±0.76
MatchingNet	53.07±0.74
ProtoNet	62.02±0.70
MAML	51.34±0.72
RelationNet	57.71±0.73

Table 3: **5-shot accuracy under the cross-domain scenario with a ResNet-18 backbone.** Baseline outperforms all other methods under this scenario.

Bonus

- ‘Less Than One’-Shot Learning: Learning N Classes From M<N Samples



Bonus

- kNN
- Soft labels

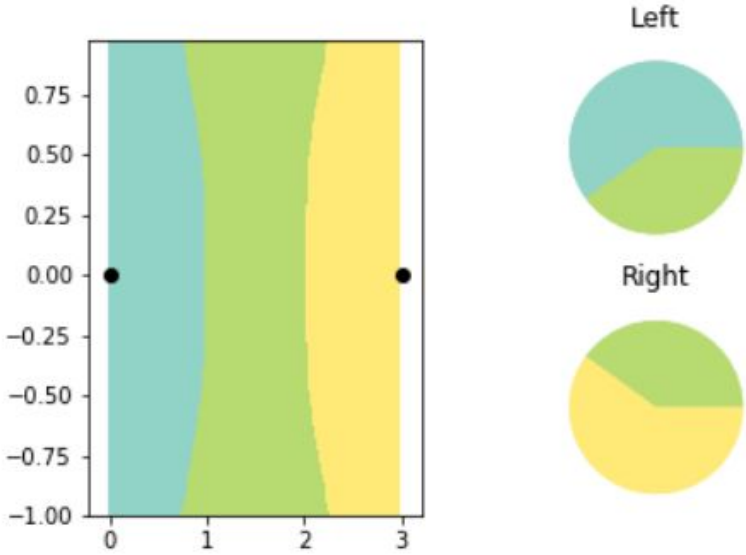


Figure 1: A SLaPkNN classifier is fitted on 2 soft-label prototypes and partitions the space into 3 classes. The soft label distribution of each prototype is illustrated by the pie charts.

Bonus

- kNN
- Soft labels