Multi-Task & Meta-Learning Basics

CS 330

Logistics

Homework 1 posted today, due **Wednesday, October 9**

Fill out **paper preferences** by tomorrow.

TensorFlow review session **tomorrow, 4:30 pm in Gates B03**

Plan for Today

Multi-Task Learning

- Models & training
- Challenges
- Case study of real-world multi-task learning

— *short break* —

Meta-Learning

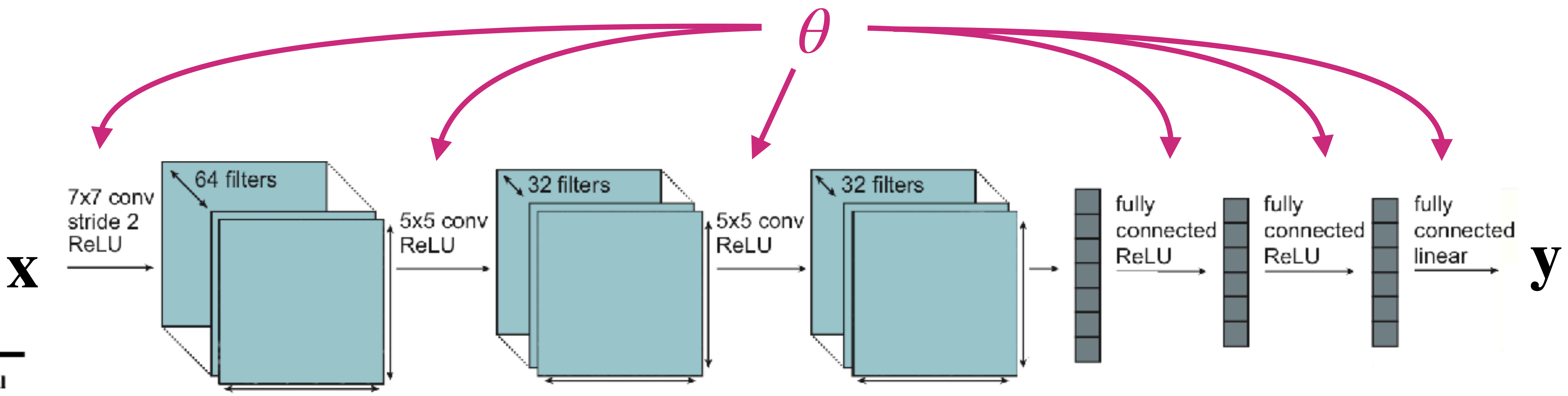
- Problem formulation
- General recipe of meta-learning algorithms
- Black-box adaptation approaches

} **Topic of Homework 1!**

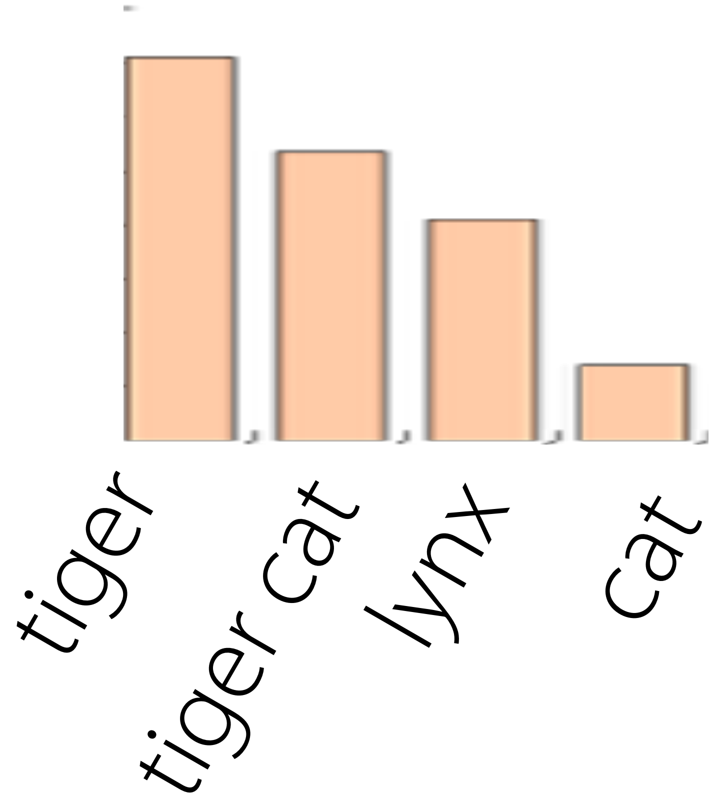
Multi-Task Learning Basics

Some notation

ImageNet Classification with Deep Convolutional Neural Networks



$f_\theta(\mathbf{y} \mid \mathbf{x})$



length of paper

Single-task learning: $\mathscr{D} = \{(\mathbf{x}, \mathbf{y})_k\}$

[supervised] $\min_\theta \mathscr{L}(\theta, \mathscr{D})$

Typical loss: negative log likelihood

$$\mathscr{L}(\theta, \mathscr{D}) = -\mathbb{E}_{(x,y)\sim\mathscr{D}}[\log f_\theta(\mathbf{y} \mid \mathbf{x})]$$

What is a task? (more formally this time)

A task: $\mathscr{T}_i \triangleq \{p_i(\mathbf{x}), p_i(\mathbf{y} \mid \mathbf{x}), \mathscr{L}_i\}$

data generating distributions

Corresponding datasets: $\mathscr{D}_i^{tr}$ $\mathscr{D}_i^{test}$

will use $\mathscr{D}_i$ as shorthand for $\mathscr{D}_i^{tr}$:

Examples of Tasks

A task: $\mathscr{T}_i \triangleq \{p_i(\mathbf{x}), p_i(\mathbf{y} \mid \mathbf{x}), \mathscr{L}_i\}$

data generating distributions

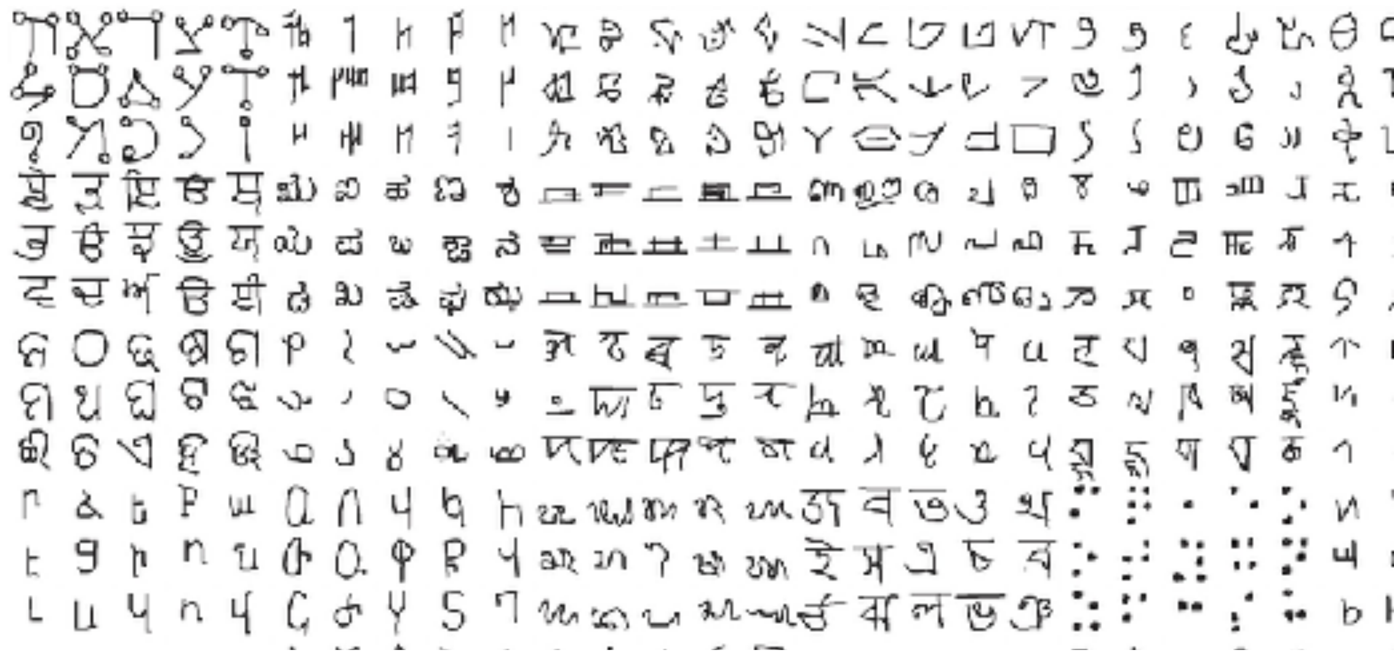
Corresponding datasets: $\mathscr{D}_i^{tr}$ $\mathscr{D}_i^{test}$

will use $\mathscr{D}_i$ as shorthand for $\mathscr{D}_i^{tr}$:

Multi-task classification: $\mathscr{L}_i$ same across all tasks

e.g. per-language handwriting recognition

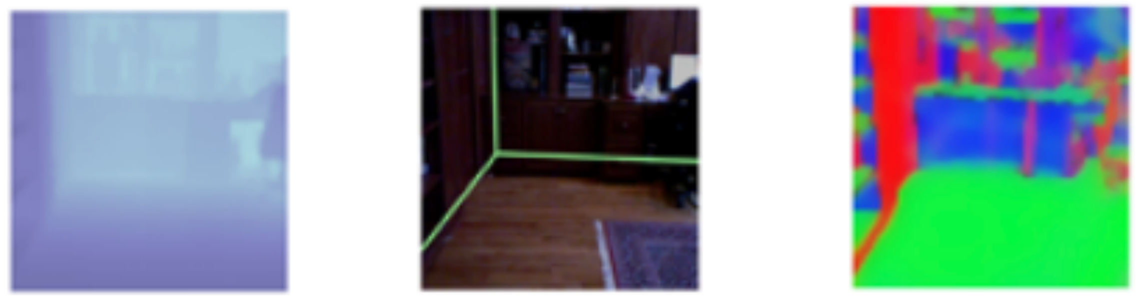
e.g. personalized spam filter



Multi-label learning: $\mathscr{L}_i, p_i(\mathbf{x})$ same across all tasks

e.g. CelebA attribute recognition

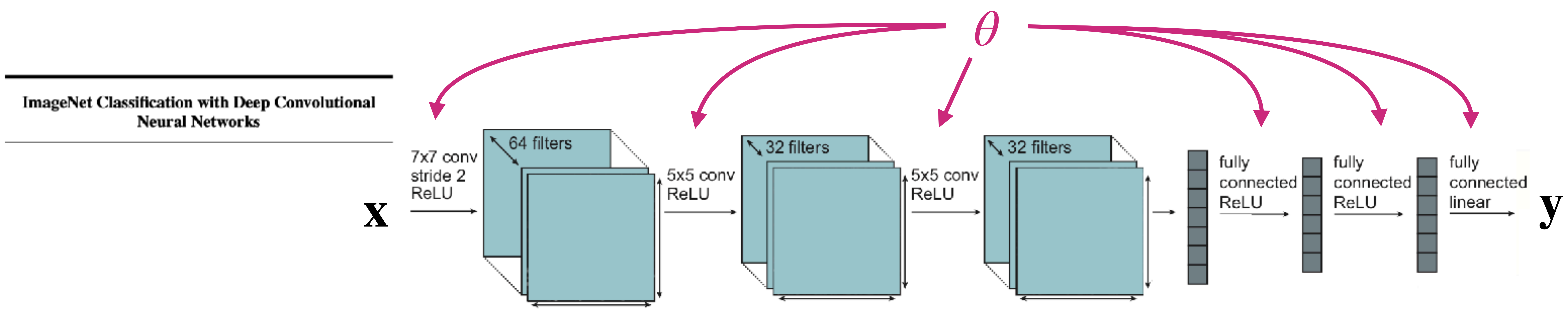
e.g. scene understanding



$$L_{\text{tot}} = w_{\text{depth}} L_{\text{depth}} + w_{\text{kpt}} L_{\text{kpt}} + w_{\text{normals}} L_{\text{normals}}$$

When might $\mathscr{L}_i$ vary across tasks?

- mixed discrete, continuous labels across tasks
- if you care more about one task than another



length of paper

summary of paper

paper review

$\mathbf{z}_i$

task descriptor

$\cancel{f_\theta(\mathbf{y} \mid \mathbf{x})}$ $f_\theta(\mathbf{y} \mid \mathbf{x}, \mathbf{z}_i)$

e.g. one-hot encoding of the task index

or, whatever meta-data you have

- personalization: user features/attributes
- language description of the task
- formal specifications of the task

Objective: $\min_\theta \sum_{i=1}^{T} \mathscr{L}_i(\theta, \mathscr{D}_i)$

A model decision and an algorithm decision:

How should we condition on $\mathbf{z}_i$?

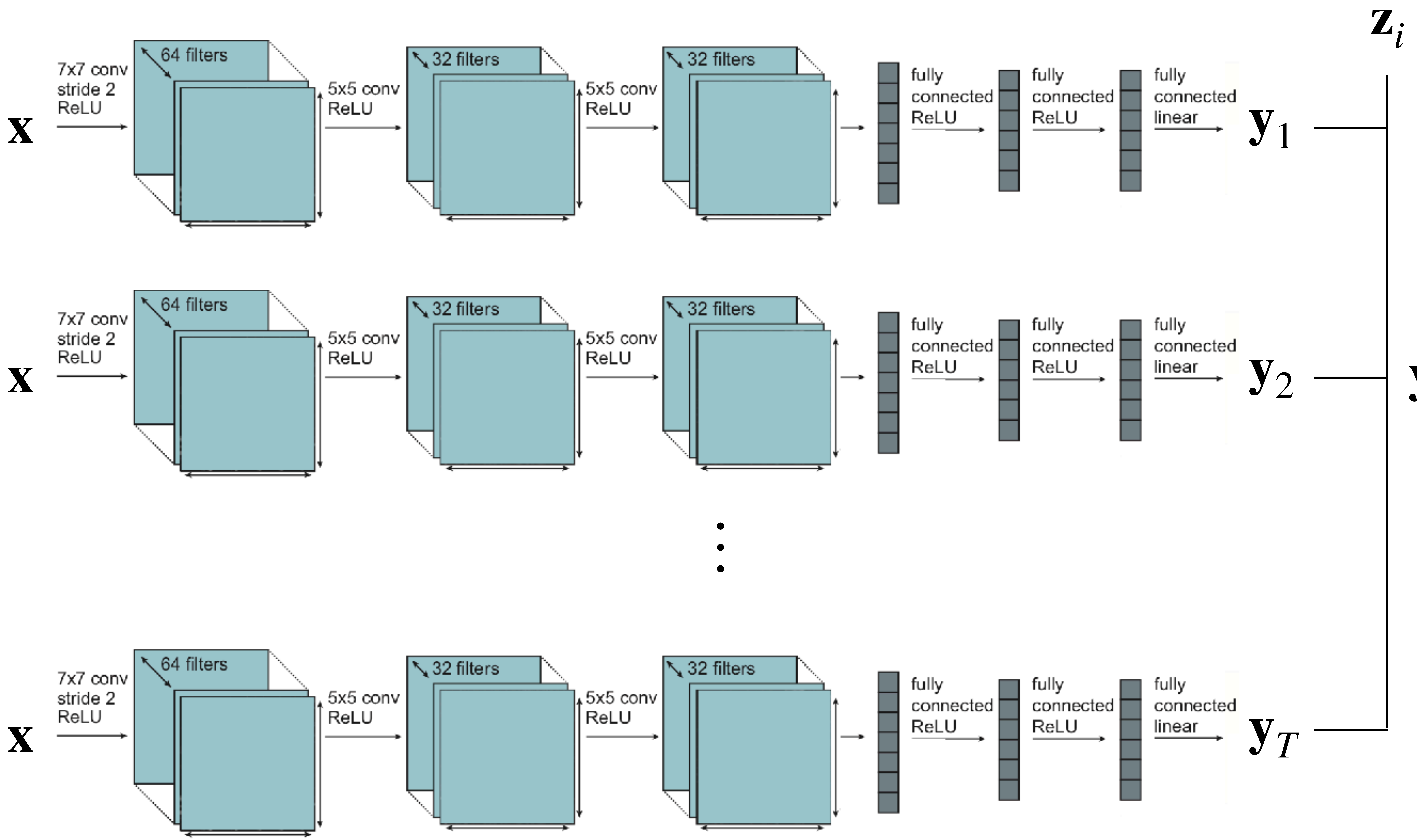
How to optimize our objective?

Conditioning on the task

Let's assume $\mathbf{z}_i$ is the task index.

Question: How should you condition on the task in order to share as little as possible?

Conditioning on the task



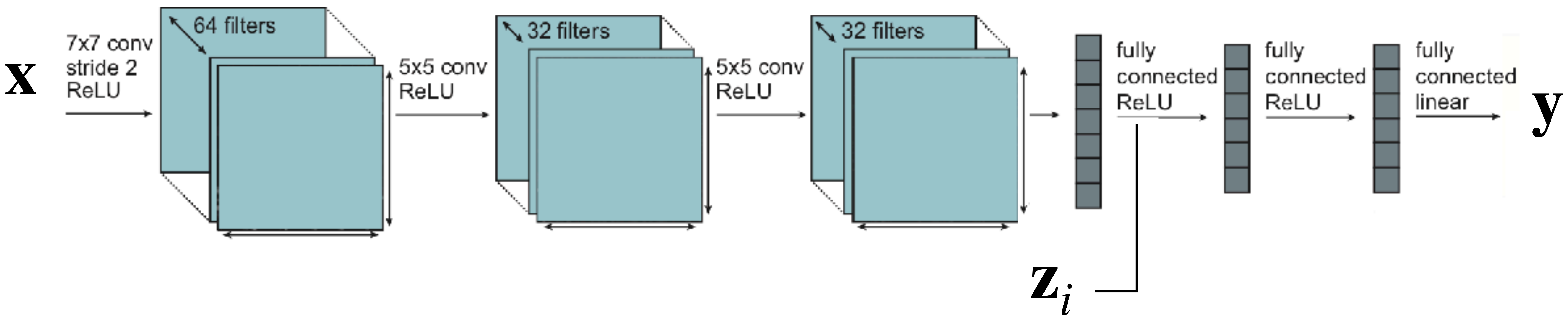
multiplicative gating

$$\mathbf{y} = \sum_j \mathbf{1}(\mathbf{z}_i = j)\mathbf{y}_j$$

—> independent training within a single network!

with no shared parameters

The other extreme



Concatenate $\mathbf{z}_i$ with input and/or activations

all parameters are shared
except
the parameters directly following $\mathbf{z}_i$

An Alternative View on the Multi-Task Objective

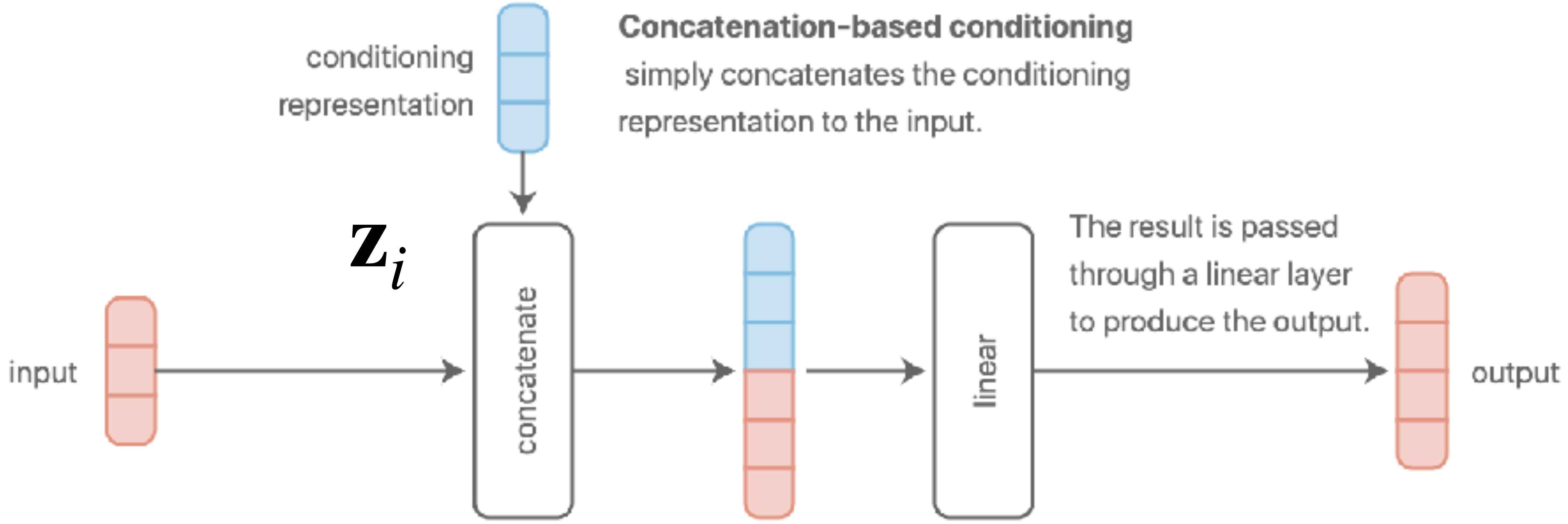
Split θ into shared parameters θ^{sh} and task-specific parameters θ^i

Then, our objective is: $$\min_{\theta^{sh},\theta^1,\ldots,\theta^T} \sum_{i=1}^{T} \mathscr{L}_i(\{\theta^{sh}, \theta^i\}, \mathscr{D}_i)$$

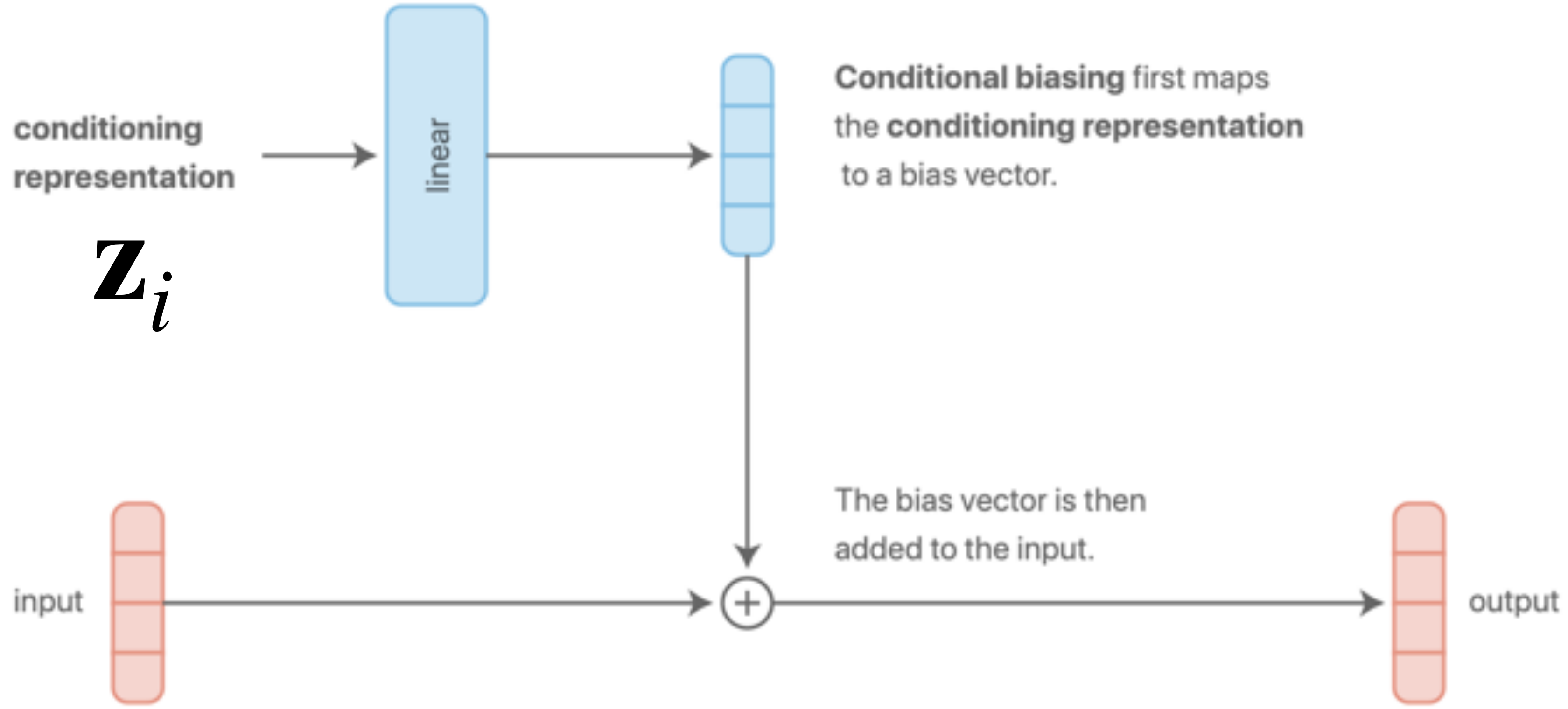
Choosing how to condition on $\mathbf{z}_i$ equivalent to Choosing how & where to share parameters

Conditioning: Some Common Choices

1. **Concatenation-based** conditioning



2. **Additive** conditioning



These are actually the same!

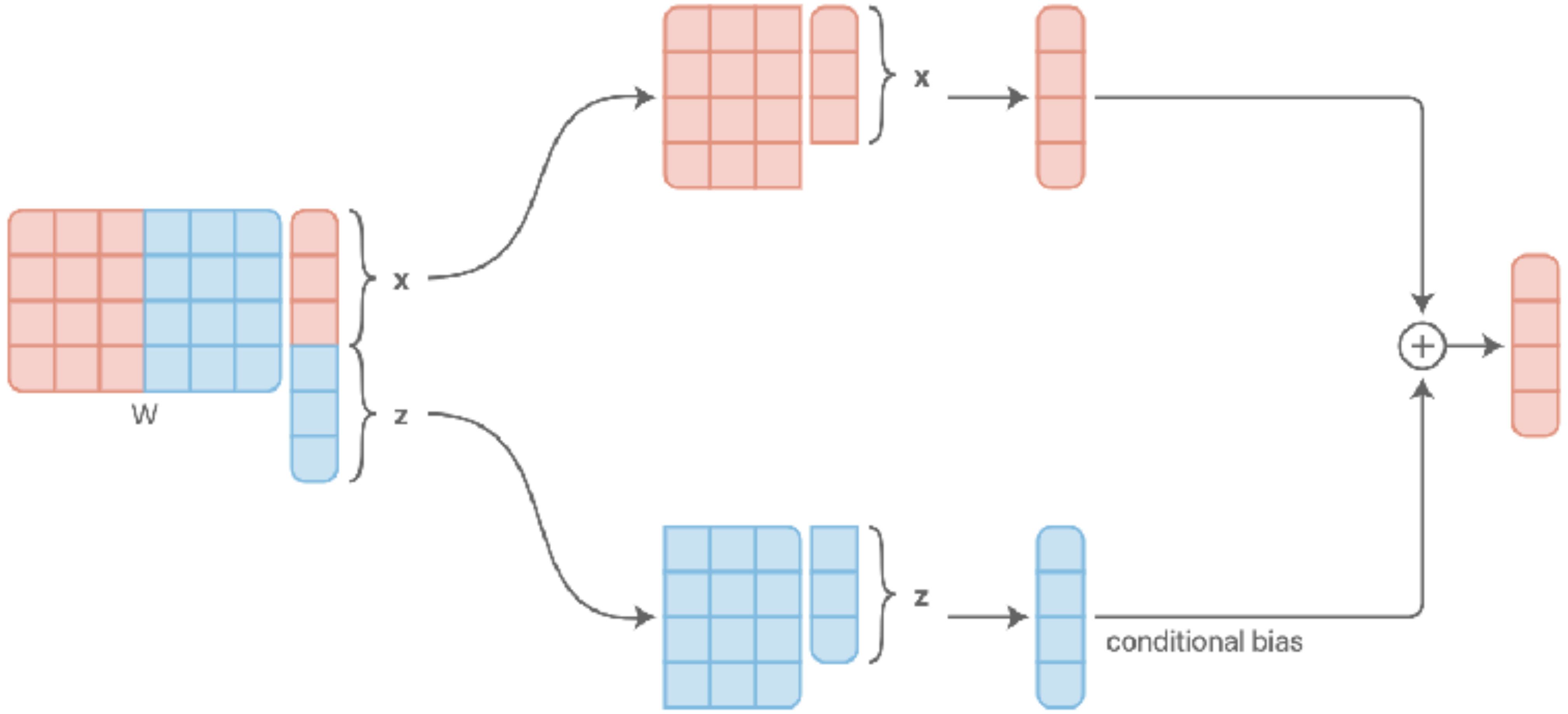
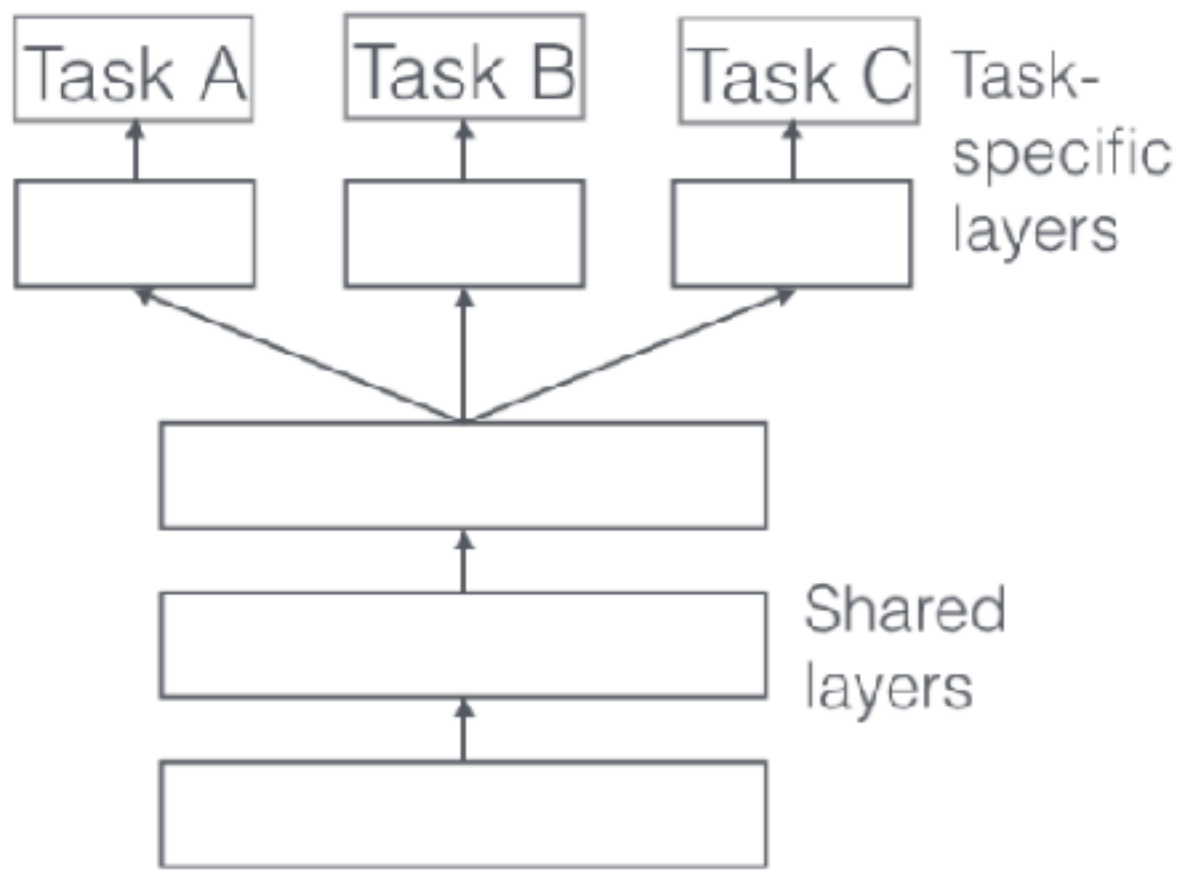


Diagram sources: distill.pub/2018/feature-wise-transformations/

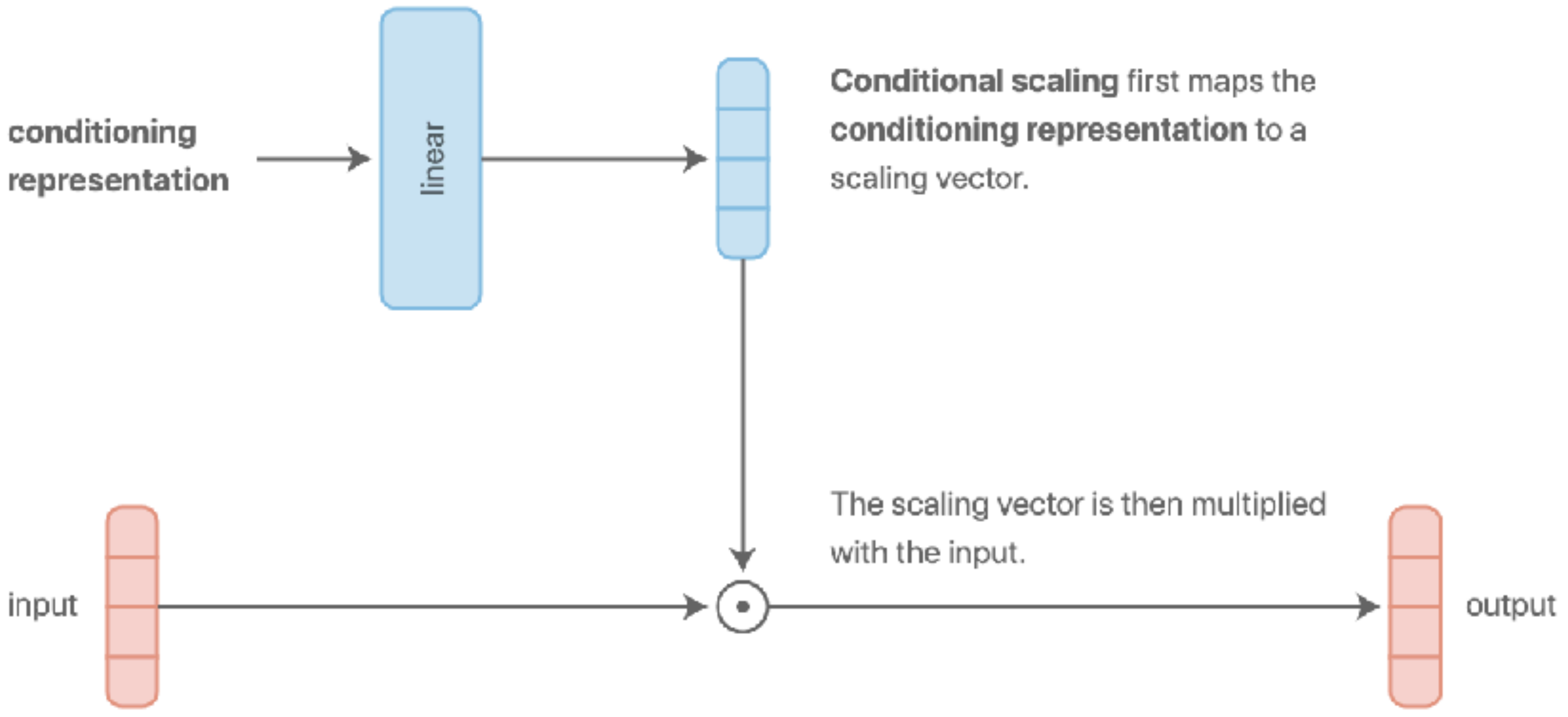
Conditioning: Some Common Choices

3. **Multi-head architecture**



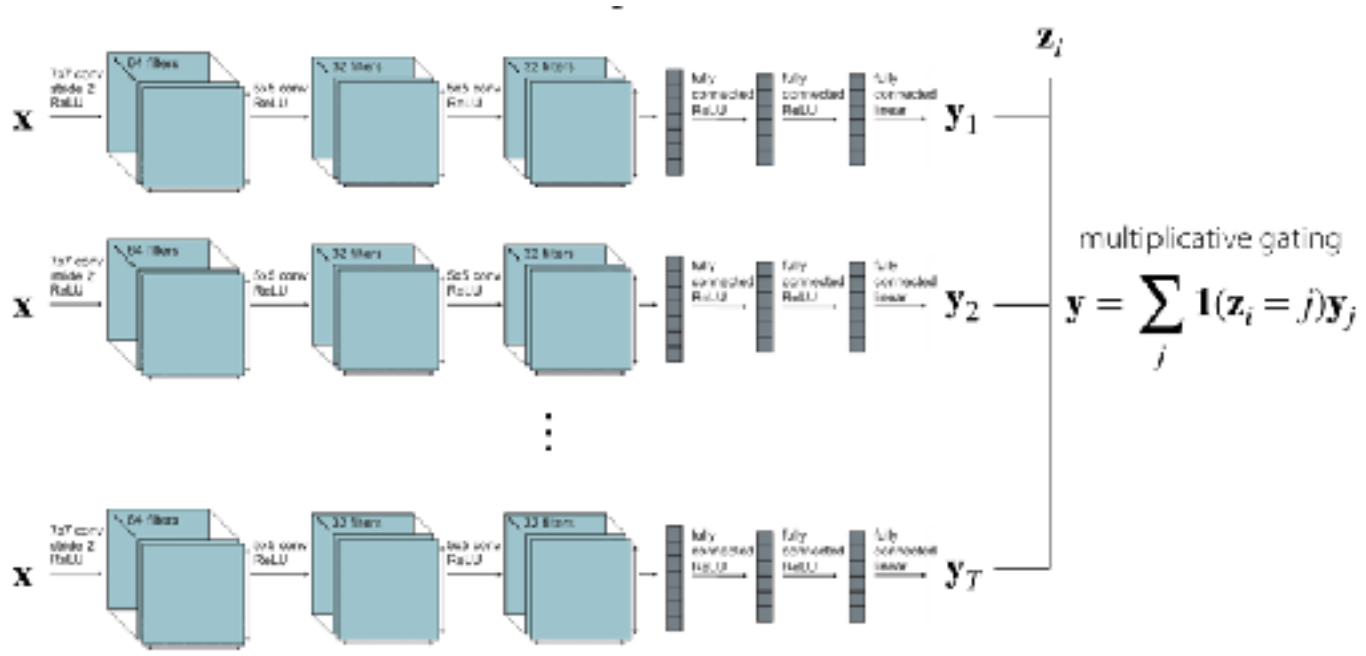
Ruder '17

4. **Multiplicative** conditioning



Why might multiplicative conditioning be a good idea?

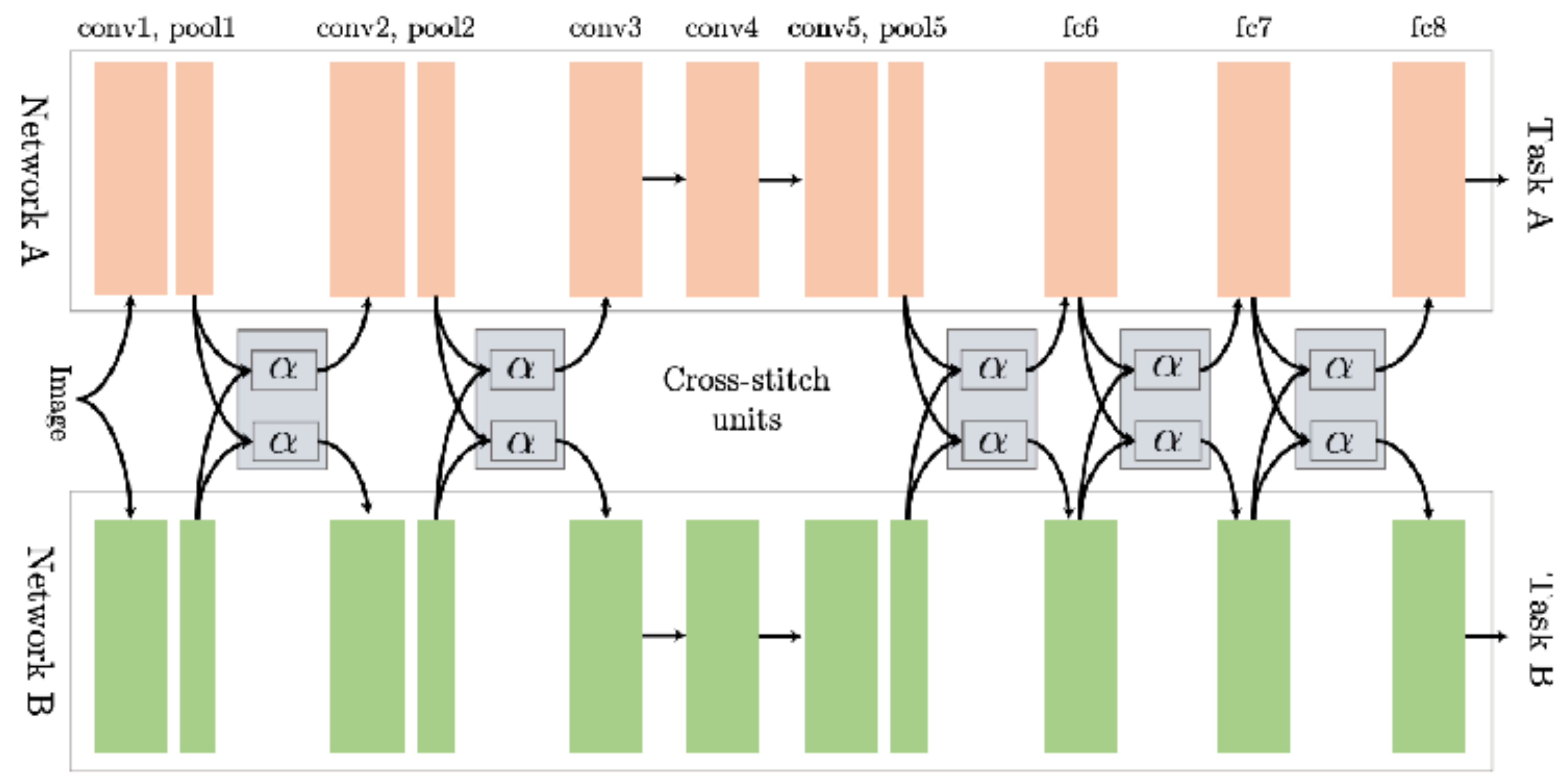
- more expressive
- recall: multiplicative gating



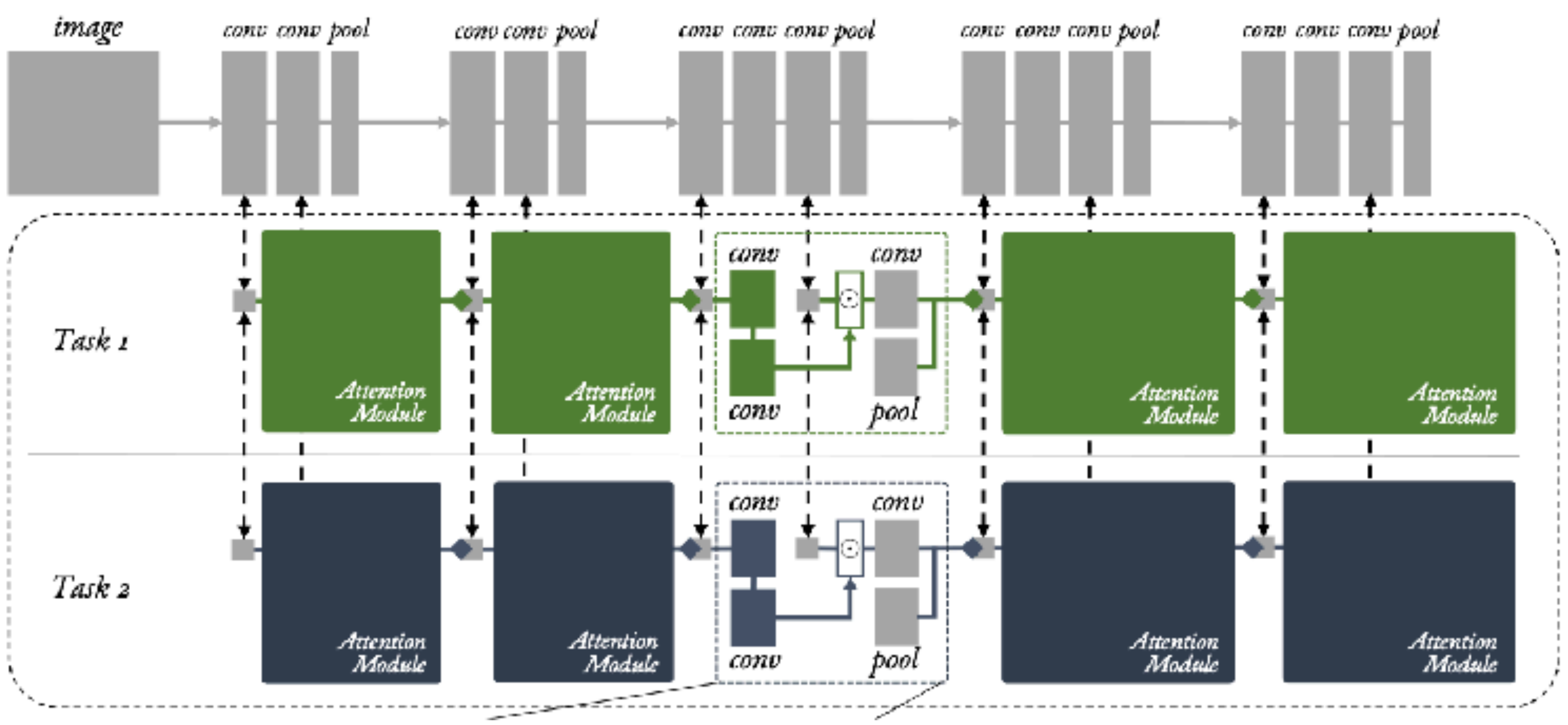
Multiplicative conditioning **generalizes** independent networks and independent heads.

Diagram sources: distill.pub/2018/feature-wise-transformations/

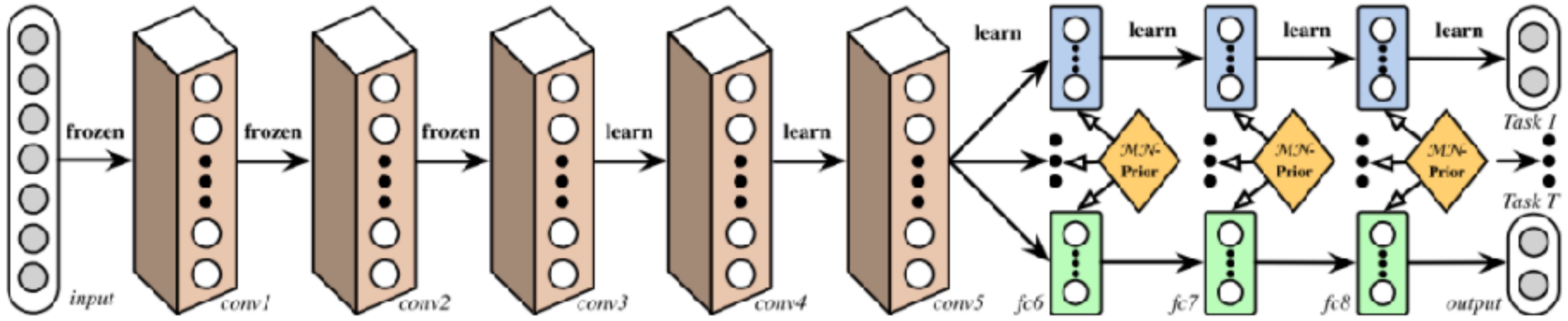
Conditioning: More Complex Choices



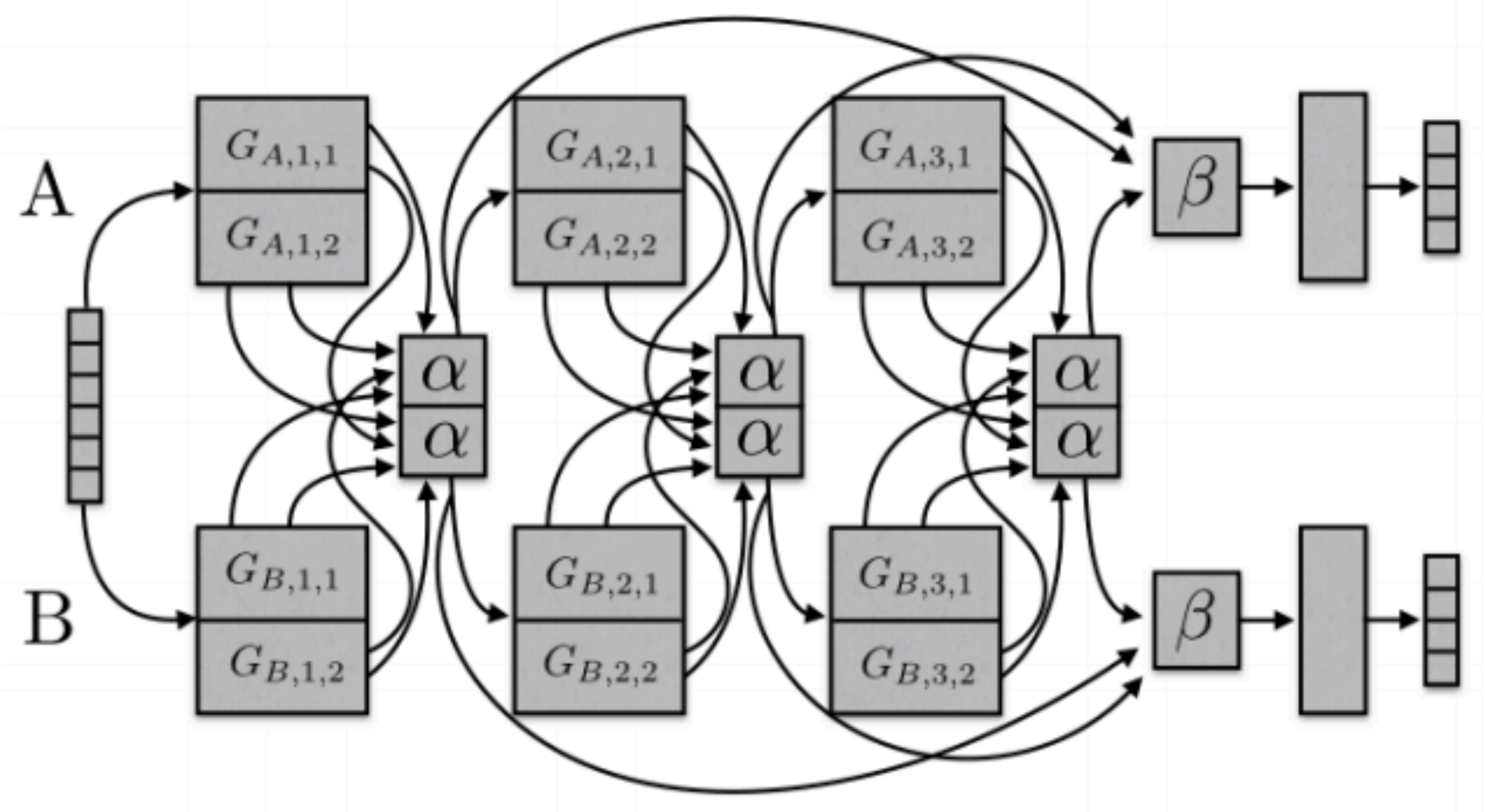
Cross-Stitch Networks. Misra, Shrivastava, Gupta, Hebert '16



Multi-Task Attention Network. Liu, Johns, Davison '18



Deep Relation Networks. Long, Wang '15



Sluice Networks. Ruder, Bingel, Augenstein, Sogaard '17

Conditioning Choices

Unfortunately, these design decisions are like neural network architecture tuning:

- **problem dependent**
- largely guided by **intuition** or **knowledge** of the problem
- currently more of an **art** than a science

Optimizing the objective

Objective: $\min_{\theta} \sum_{i=1}^{T} \mathscr{L}_i(\theta, \mathscr{D}_i)$

Basic Version:

1. Sample mini-batch of tasks $\mathscr{B} \sim \{\mathscr{T}_i\}$
2. Sample mini-batch datapoints for each task $\mathscr{D}_i^b \sim \mathscr{D}_i$
3. Compute loss on the mini-batch: $\hat{\mathscr{L}}(\theta, \mathscr{B}) = \sum_{\mathscr{T}_k \in \mathscr{B}} \mathscr{L}_k(\theta, \mathscr{D}_k^b)$
4. Backpropagate loss to compute gradient $\nabla_\theta \hat{\mathscr{L}}$
5. Apply gradient with your favorite neural net optimizer (e.g. Adam)

Note: This ensures that tasks are sampled uniformly, regardless of data quantities.

Tip: For regression problems, make sure your task labels are on the same scale!

Challenges

Challenge #1: Negative transfer

Negative transfer: Sometimes independent networks work the best.

Multi-Task CIFAR-100

state-of-the-art approaches

	% accuracy
task specific-1-fc (Rosenbaum et al., 2018)	42
task specific-all-fc (Rosenbaum et al., 2018)	49
cross stitch-all-fc (Misra et al., 2016b)	53
routing-all-fc + WPL (Rosenbaum et al., 2019)	64.1
independent	64.3

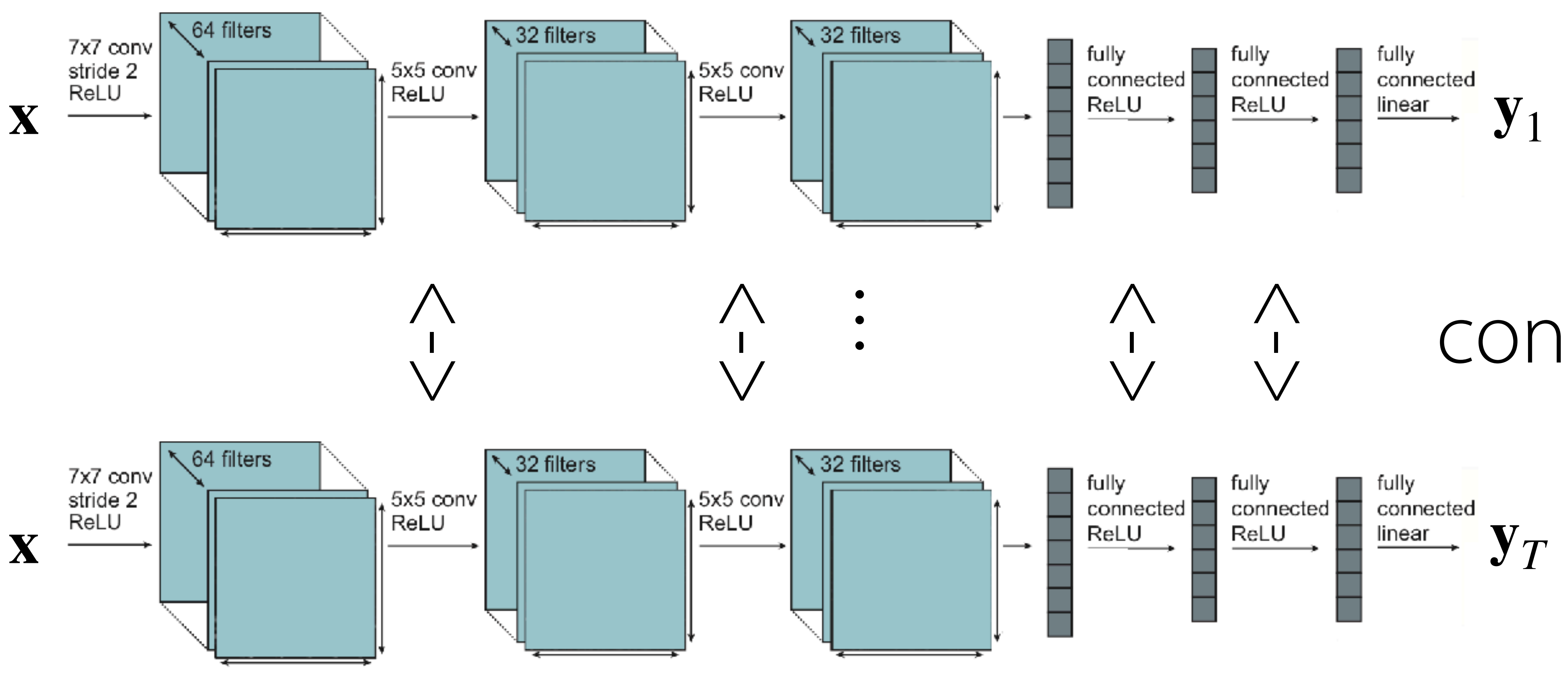
Why?

- **optimization challenges**
  - caused by cross-task interference
  - tasks may learn at different rates
- **limited representational capacity**
  - multi-task networks often need to be *much larger* than their single-task counterparts

If you have negative transfer, **share less** across tasks.

It's not just a binary decision!

$$\min_{\theta^{sh},\theta^1,\ldots,\theta^T} \sum_{i=1}^{T} \mathscr{L}_i(\{\theta^{sh},\theta^i\},\mathscr{D}_i) + \underbrace{\sum_{t'=1}^{T} \|\theta^t - \theta^{t'}\|}_{\text{“soft parameter sharing”}}$$



constrained weights

+ allows for more fluid degrees of parameter sharing

- yet another set of design decisions / hyperparameters

Challenge #2: Overfitting

You may not be sharing enough!

Multi-task learning <-> a form of regularization

Solution: Share more.

Case study

Recommending What Video to Watch Next: A Multitask Ranking System

Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, Ed Chi
Google, Inc.
{zhezhao,lichan,liwei,jilinc,aniruddhnath,shawnandrews,aditeek,nlogn,xinyang,edchi}@google.com

Goal: Make recommendations for YouTube

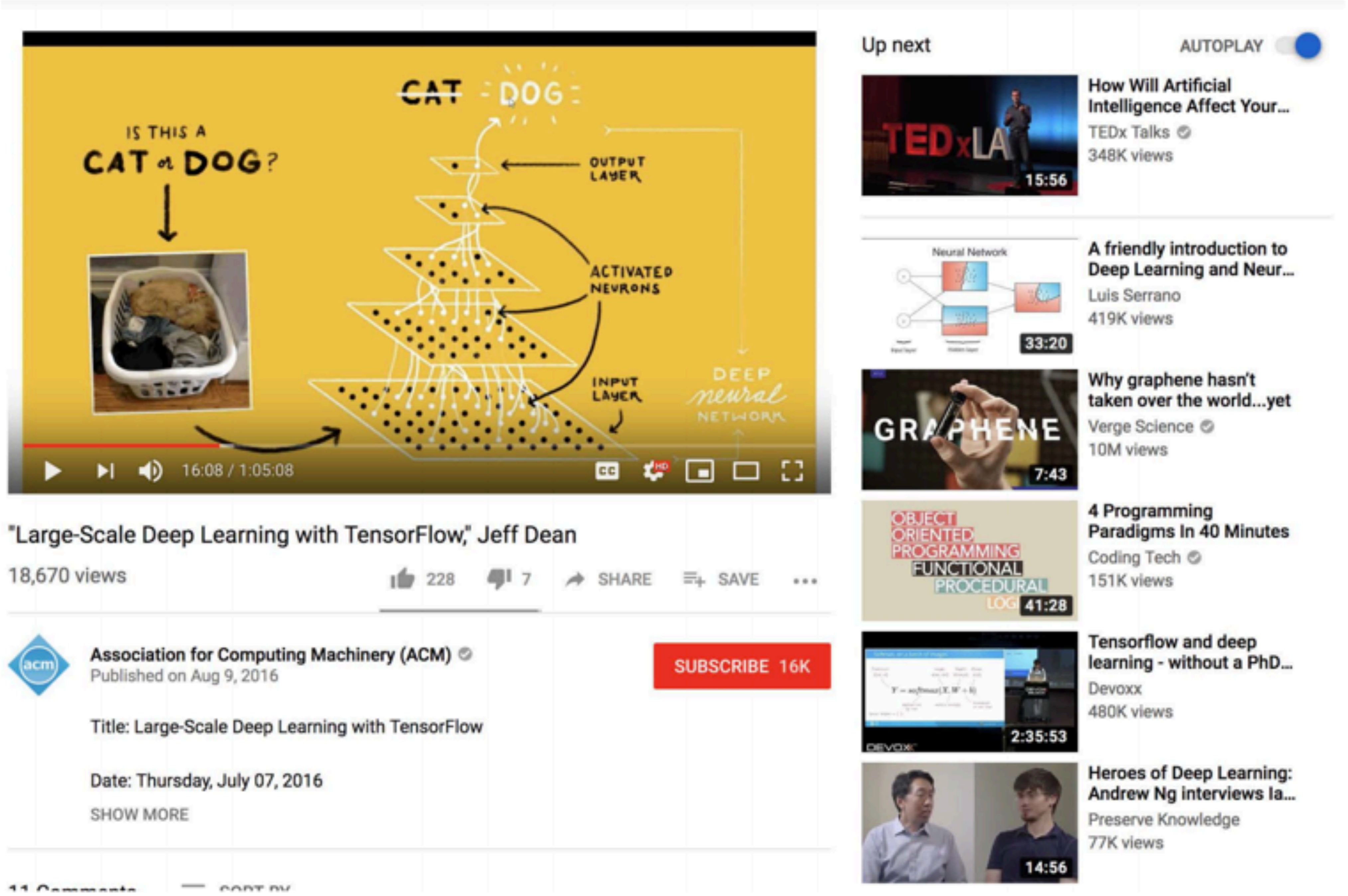


Figure 4: Recommending what to watch next on YouTube.

Case study

Recommending What Video to Watch Next: A Multitask Ranking System

Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar,
Maheswaran Sathiamoorthy, Xinyang Yi, Ed Chi
Google, Inc.
{zhezhao,lichan,liwei,jilinc,aniruddhnath,shawnandrews,aditeek,nlogn,xinyang,edchi}@google.com

Goal: Make recommendations for YouTube

Conflicting objectives:

- videos that users will rate highly
- videos that users they will share
- videos that user will watch

implicit bias caused by *feedback:*
user may have watched it because it was recommended!

Framework Set-Up

Input: what the user is currently watching (query video) + user features

1. Generate a few hundred of candidate videos
2. Rank candidates
3. Serve top ranking videos to the user

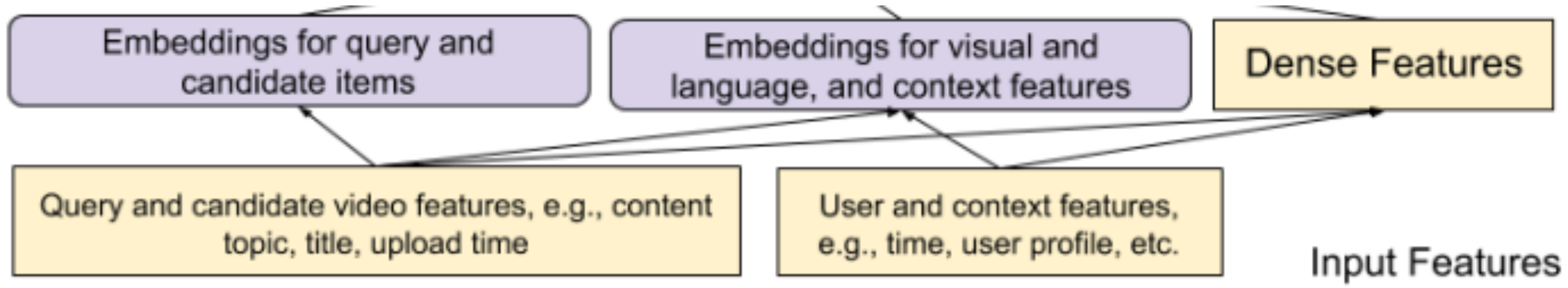
Candidate videos: pool videos from multiple candidate generation algorithms

- matching topics of query video
- videos most frequently watched with query video
- And others

Ranking: central topic of this paper

The Ranking Problem

Input: query video, candidate video, user & context features



Model output: engagement and satisfaction with candidate video

Engagement:

- binary classification tasks like **clicks**
- regression tasks for tasks related to **time spent**

Satisfaction:

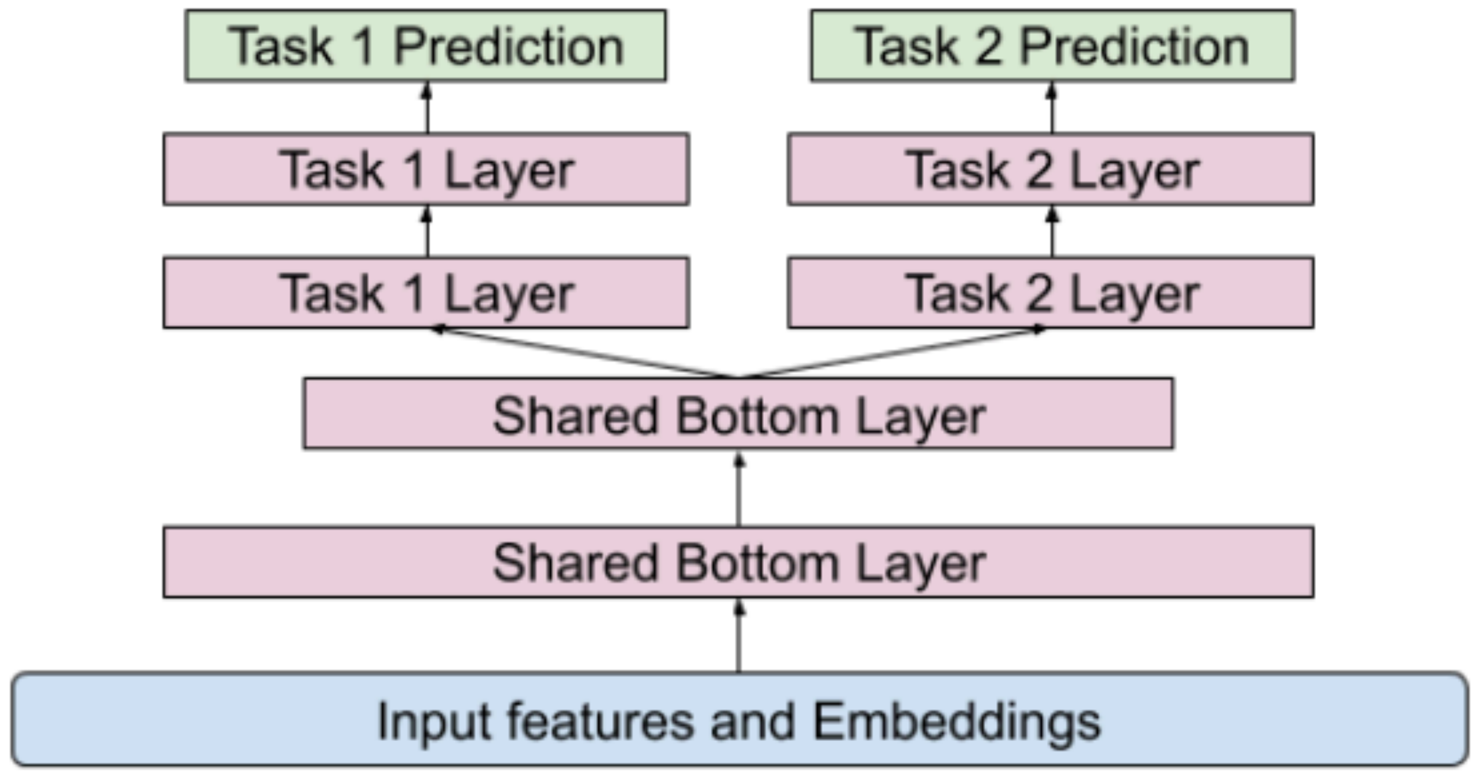
- binary classification tasks like **clicking "like"**
- regression tasks for tasks such as **rating**

Weighted combination of **engagement** & **satisfaction** predictions -> **ranking score**

score weights manually tuned

The Architecture

Basic option: "Shared-Bottom Model"
(i.e. multi-head architecture)

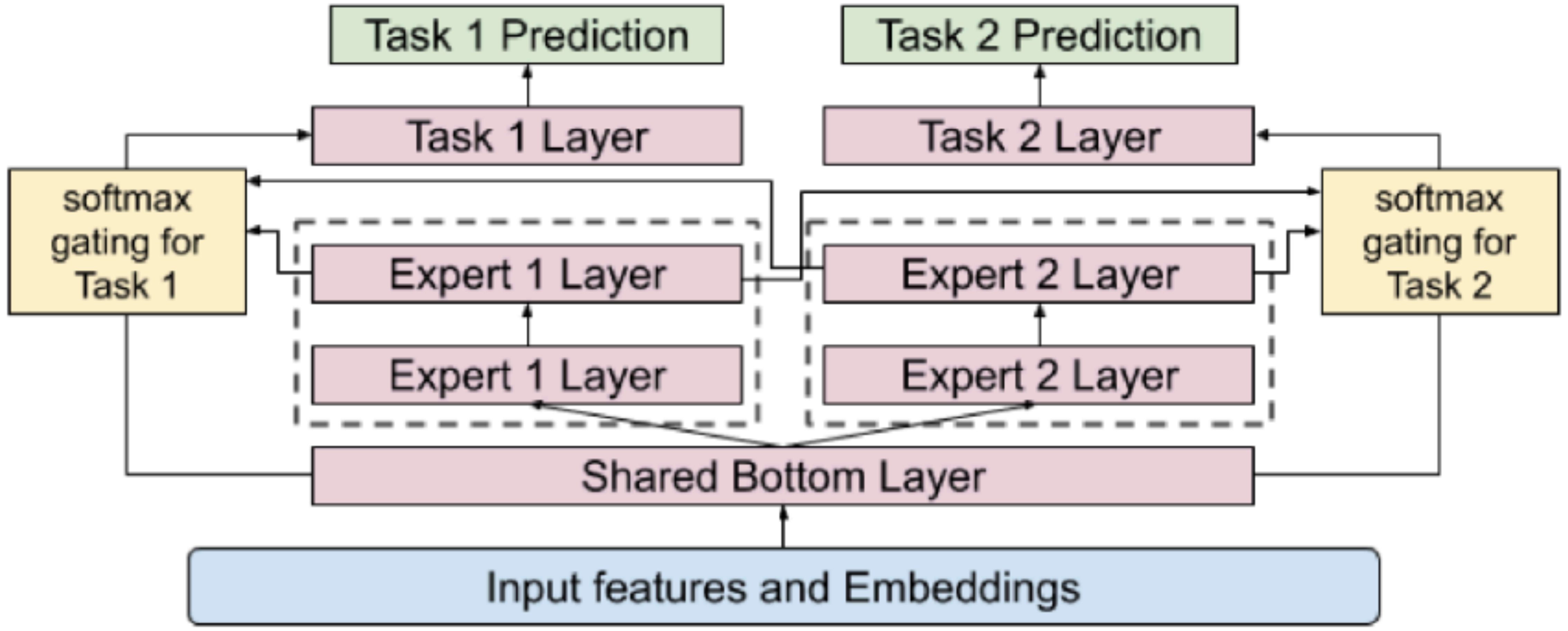


(a) Shared-Bottom Model with shared bottom hidden layers and separate towers for two tasks.

-> harm learning when correlation between tasks is low

The Architecture

Instead: use a form of soft-parameter sharing
“**Multi-gate Mixture-of-Experts (MMoE)**"



(b) Multi-gate Mixture-of-Expert Model with one shared bottom layer and separate hidden layers for two tasks.

Allow different parts of the network to “specialize"

expert neural networks $f_i(x)$

Decide which expert to use for input x, task k:

$$g^k(x) = \text{softmax}(W_{g^k} x)$$

Compute features from selected expert:

$$f^k(x) = \sum_{i=1}^{n} g^k_{(i)}(x) f_i(x)$$

Compute output: $y_k = h^k(f^k(x)),$

Experiments

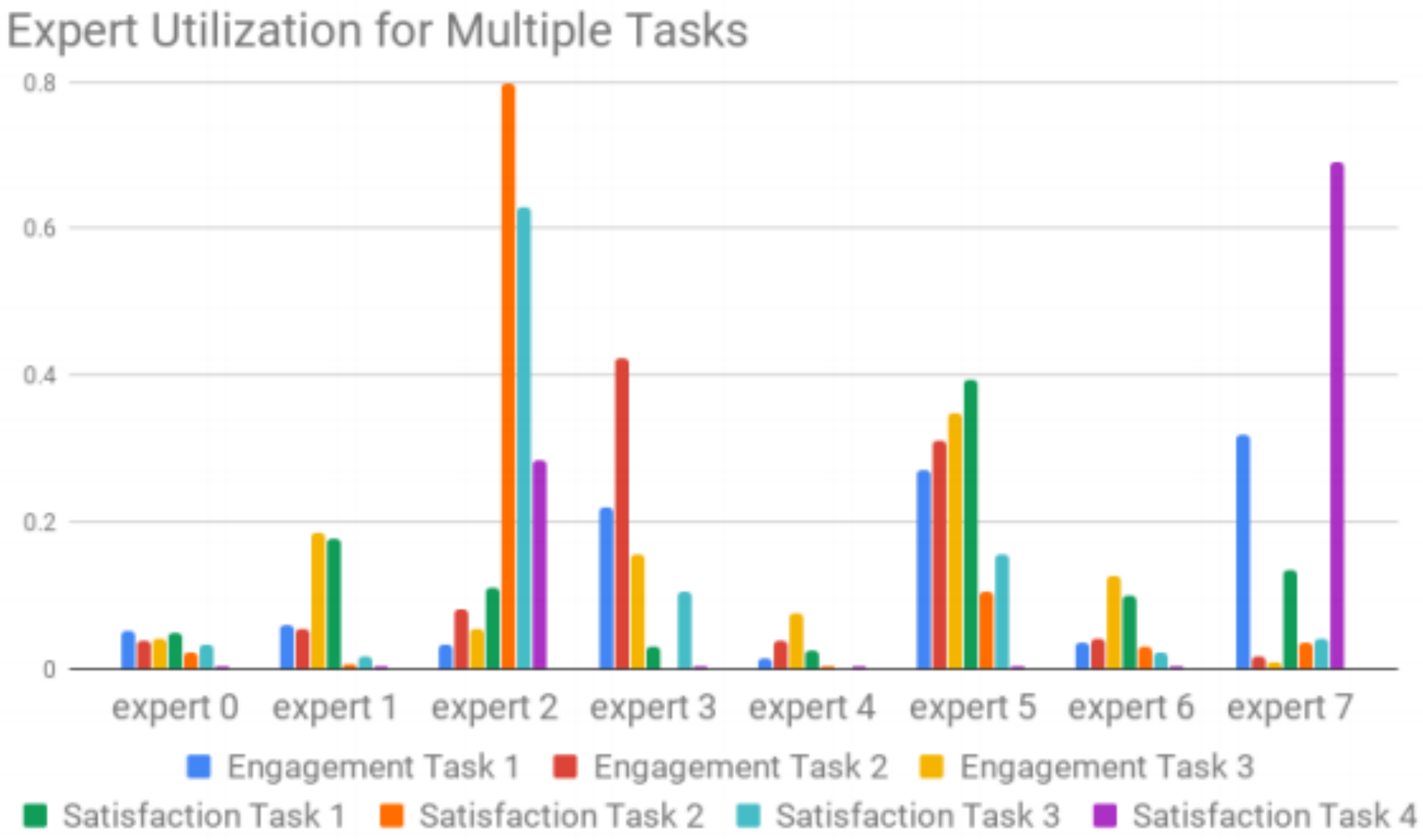
Set-Up

- Implementation in TensorFlow, TPUs
- Train in ***temporal order***, running training continuously to consume newly arriving data
- **Offline** AUC & squared error metrics
- **Online A/B testing** in comparison to production system
    - live metrics based on time spent, survey responses, rate of dismissals
- Model **computational efficiency** matters

Results

Model Architecture	Number of Multiplications	Engagement Metric	Satisfaction Metric
Shared-Bottom	3.7M	/	/
Shared-Bottom	6.1M	+0.1%	+ 1.89%
MMoE (4 experts)	3.7M	+0.20%	+ 1.22%
MMoE (8 Experts)	6.1M	+0.45%	+ 3.07%

Table 1: YouTube live experiment results for MMoE.



Found 20% chance of gating polarization during distributed training -> use drop-out on experts

Plan for Today

Multi-Task Learning
- Models & training
- Challenges
- Case study of real-world multi-task learning

— short break —

Meta-Learning
- Problem formulation
- General recipe of meta-learning algorithms
- Black-box adaptation approaches

} Topic of Homework 1!

Meta-Learning Basics

Two ways to view meta-learning algorithms

Mechanistic view

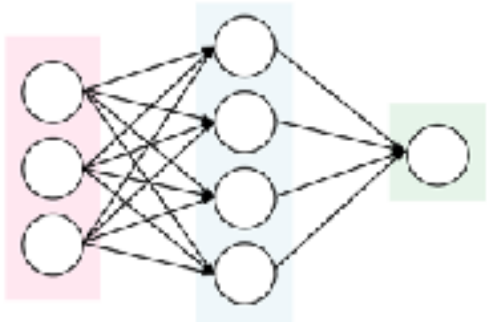
- Deep neural network model that can read in an entire dataset and make predictions for new datapoints
- Training this network uses a meta-dataset, which itself consists of many datasets, each for a different task
- This view makes it easier to implement meta-learning algorithms

Probabilistic view

- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks
- Learning a new task uses this prior and (small) training set to infer most likely posterior parameters
- This view makes it easier to understand meta-learning algorithms

Problem definitions

supervised learning:



$$\arg\max_{\phi} \log p(\phi|\mathcal{D})$$

model parameters

training data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

input (e.g., image)

label

$$= \arg\max_{\phi} \log p(\mathcal{D}|\phi) + \log p(\phi)$$

data likelihood

regularizer (e.g., weight decay)

$$= \arg\max_{\phi} \sum_i \log p(y_i|x_i, \phi) + \log p(\phi)$$

What is wrong with this?

- ➢ The most powerful models typically require large amounts of labeled data
- ➢ Labeled data for some tasks may be very limited

Problem definitions

supervised learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D})$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

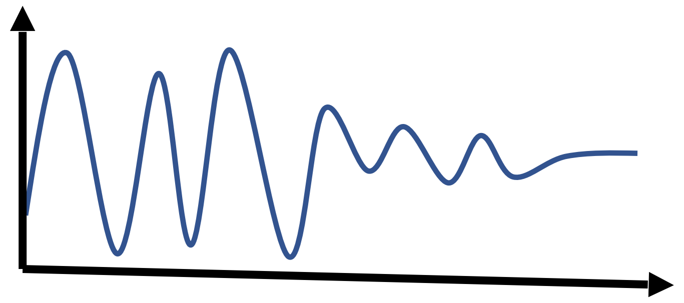
can we incorporate *additional* data?

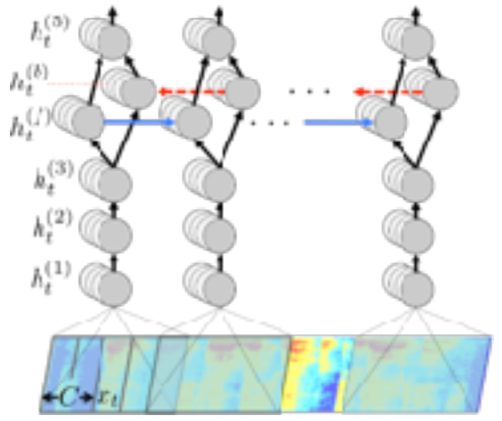
$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

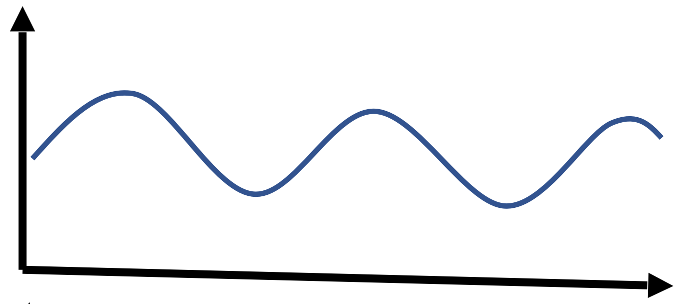
$\mathcal{D}$



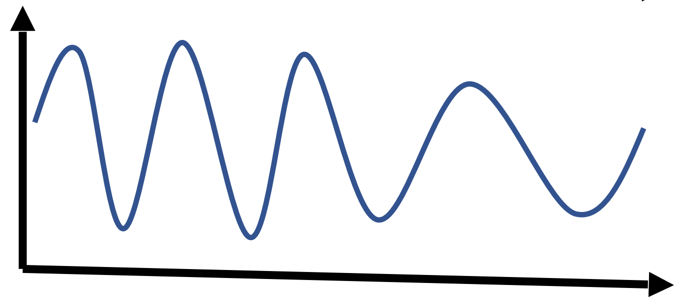


$\mathcal{D}_{\text{meta-train}}$

$\mathcal{D}_1$



$\mathcal{D}_2$



⋮

Image adapted from Ravi & Larochelle

The meta-learning problem

meta-learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

what if we don't want to keep $\mathcal{D}_{\text{meta-train}}$ around forever?

learn *meta-parameters* θ: $p(\theta|\mathcal{D}_{\text{meta-train}})$

whatever we need to know about $\mathcal{D}_{\text{meta-train}}$ to solve new tasks

assume $\phi \perp\!\!\!\perp \mathcal{D}_{\text{meta-train}}|\theta$

$$\log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) = \log \int_{\Theta} p(\phi|\mathcal{D}, \theta) p(\theta|\mathcal{D}_{\text{meta-train}}) d\theta$$

$$\approx \log p(\phi|\mathcal{D}, \theta^\star) + \log p(\theta^\star|\mathcal{D}_{\text{meta-train}})$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \approx \arg\max_{\phi} \log p(\phi|\mathcal{D}, \theta^\star)$$

this is the meta-learning problem

$$\theta^\star = \arg\max_{\theta} \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

A Quick Example

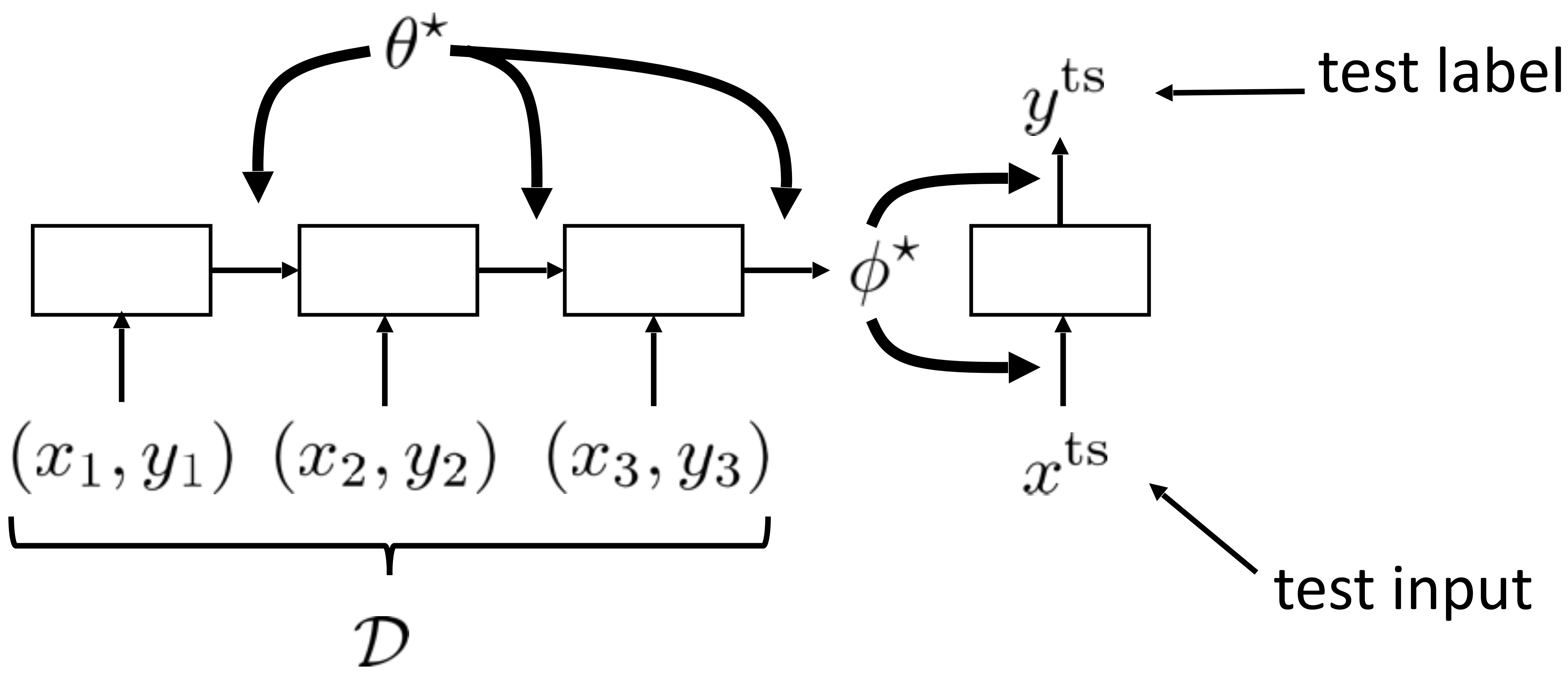
$$\text{meta-learning:}\;\; \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

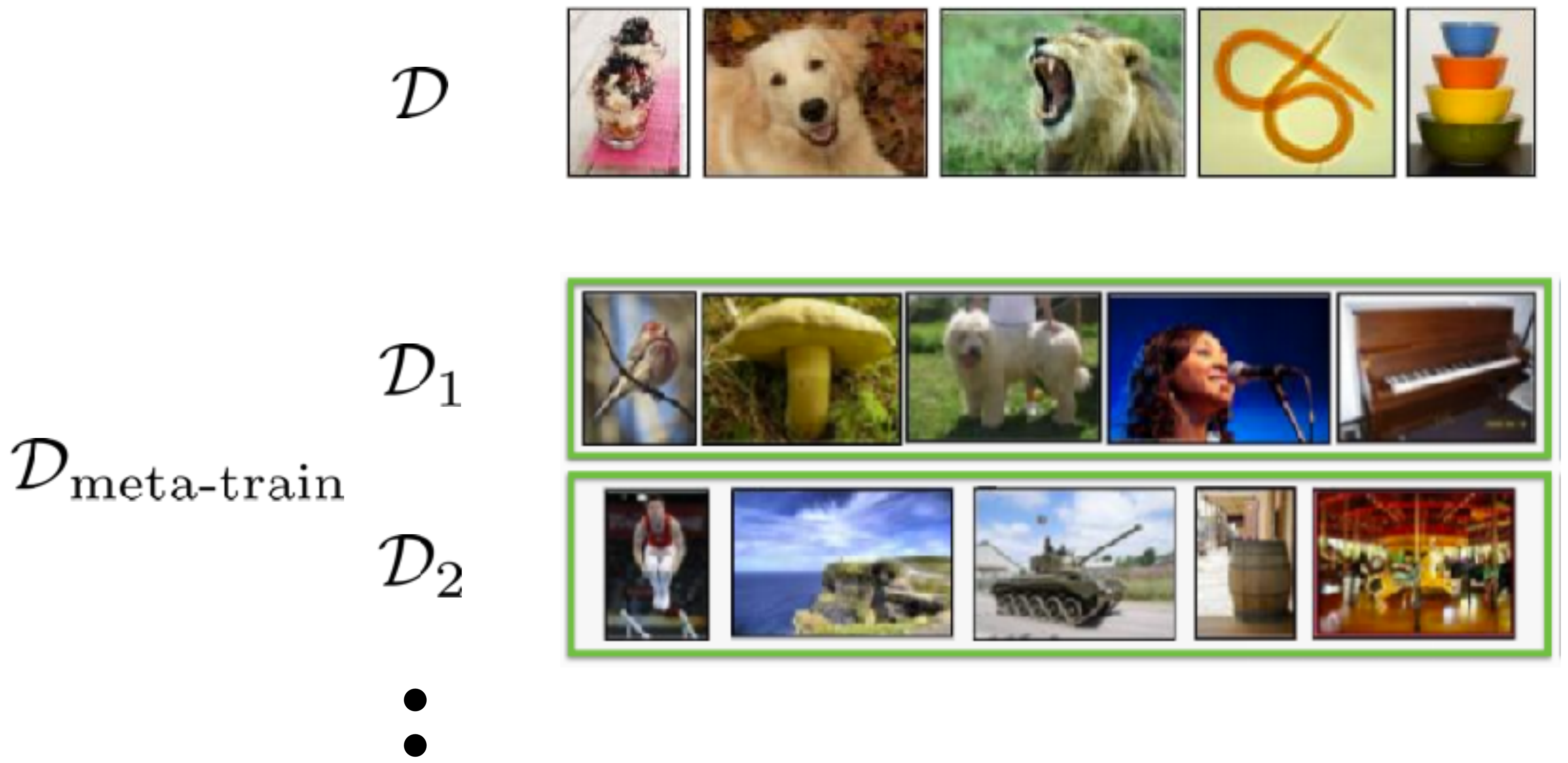
$$\boxed{\text{adaptation:}\;\; \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)}$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$





How do we train this thing?

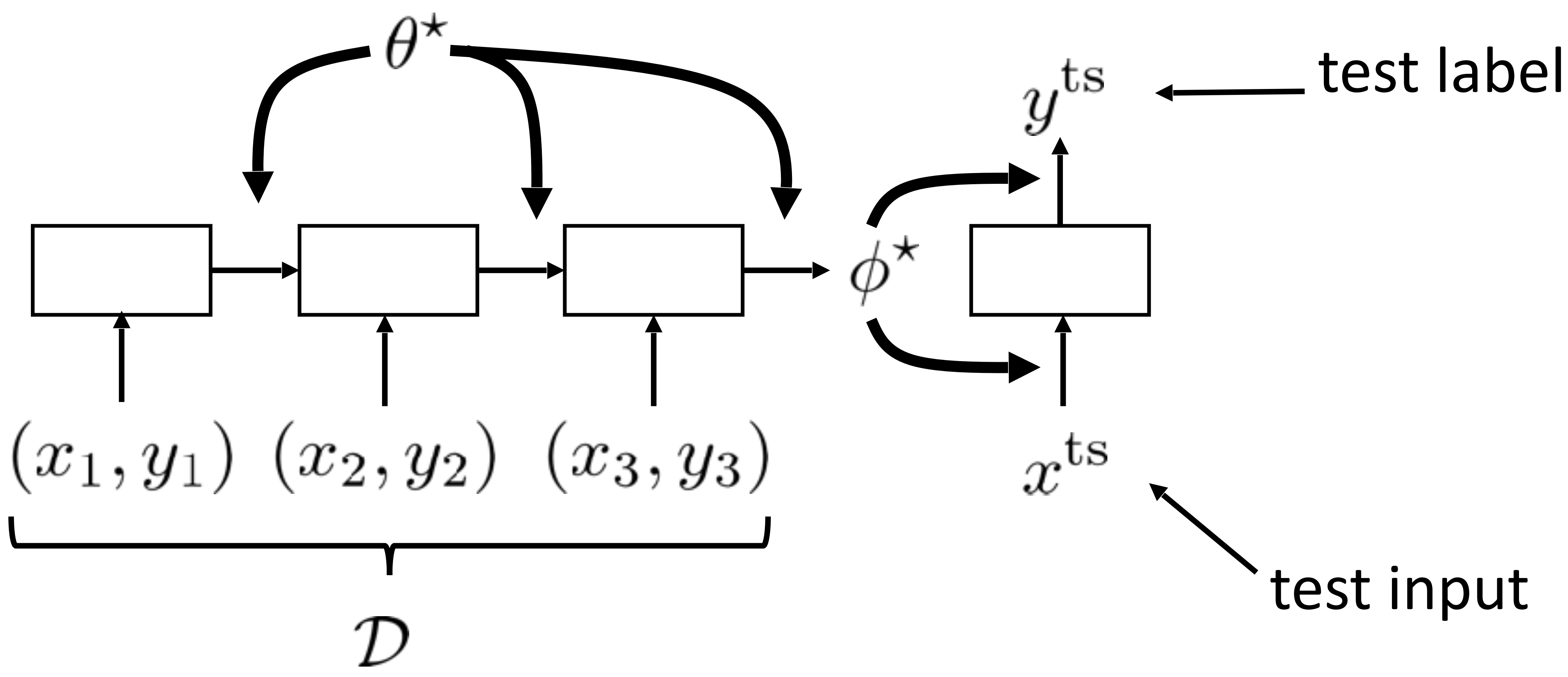
$$\boxed{\text{meta-learning: } \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})}$$

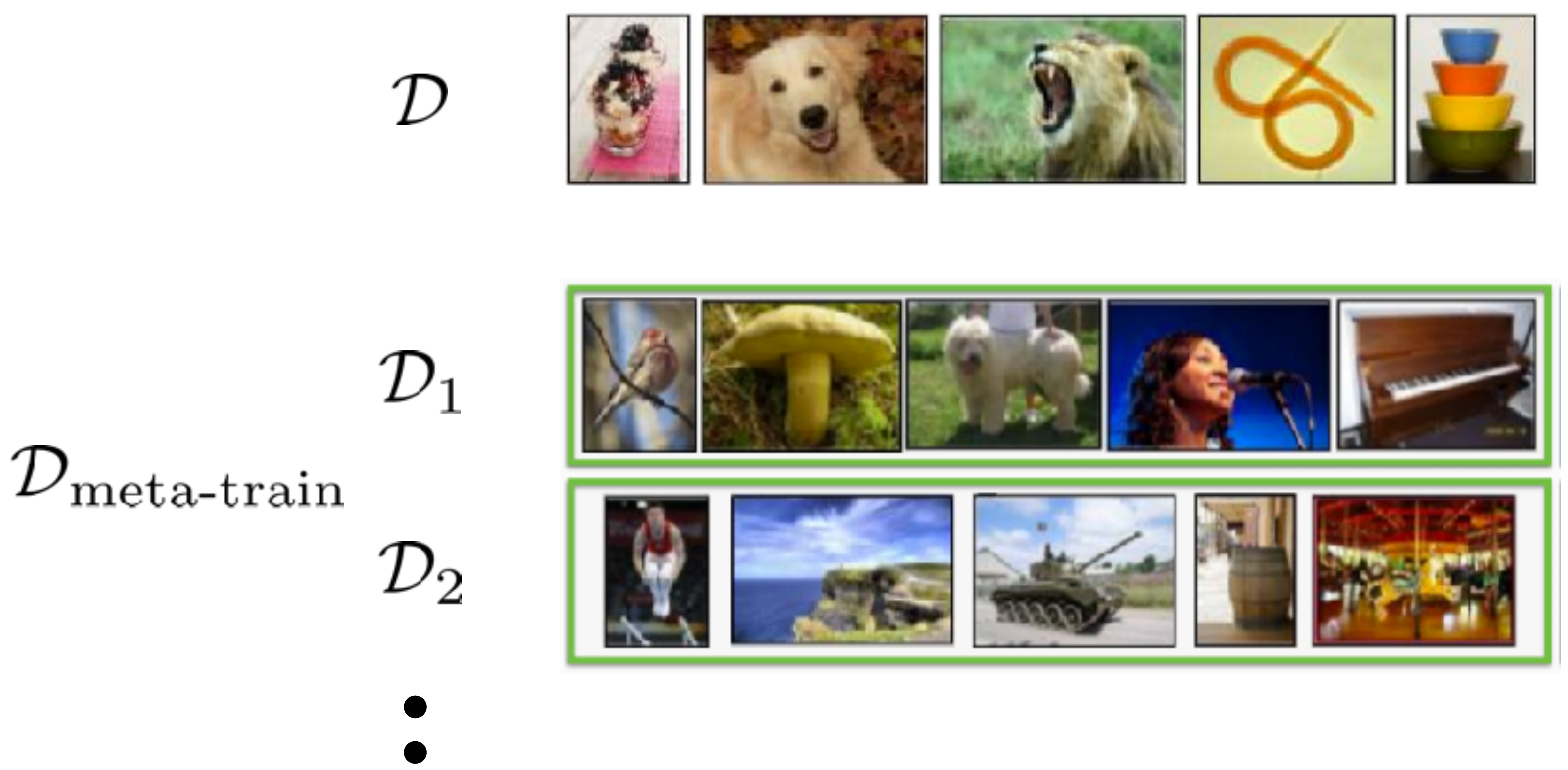
$$\text{adaptation: } \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$





Key idea:
“our training procedure is based on a simple machine learning principle: test and train conditions must match”
Vinyals et al., Matching Networks for One-Shot Learning

How do we train this thing?

$$\boxed{\text{meta-learning: } \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})}$$

$$\text{adaptation: } \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)$$

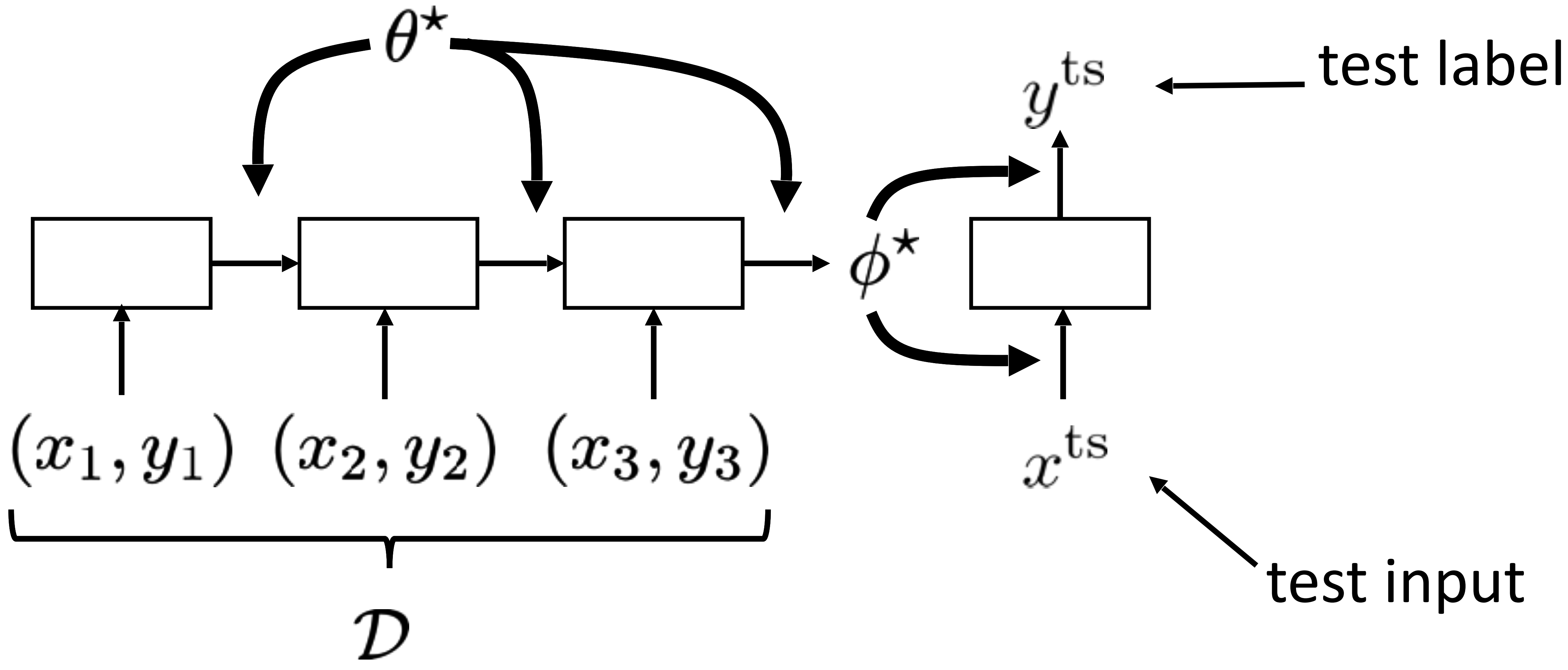
(meta) test-time

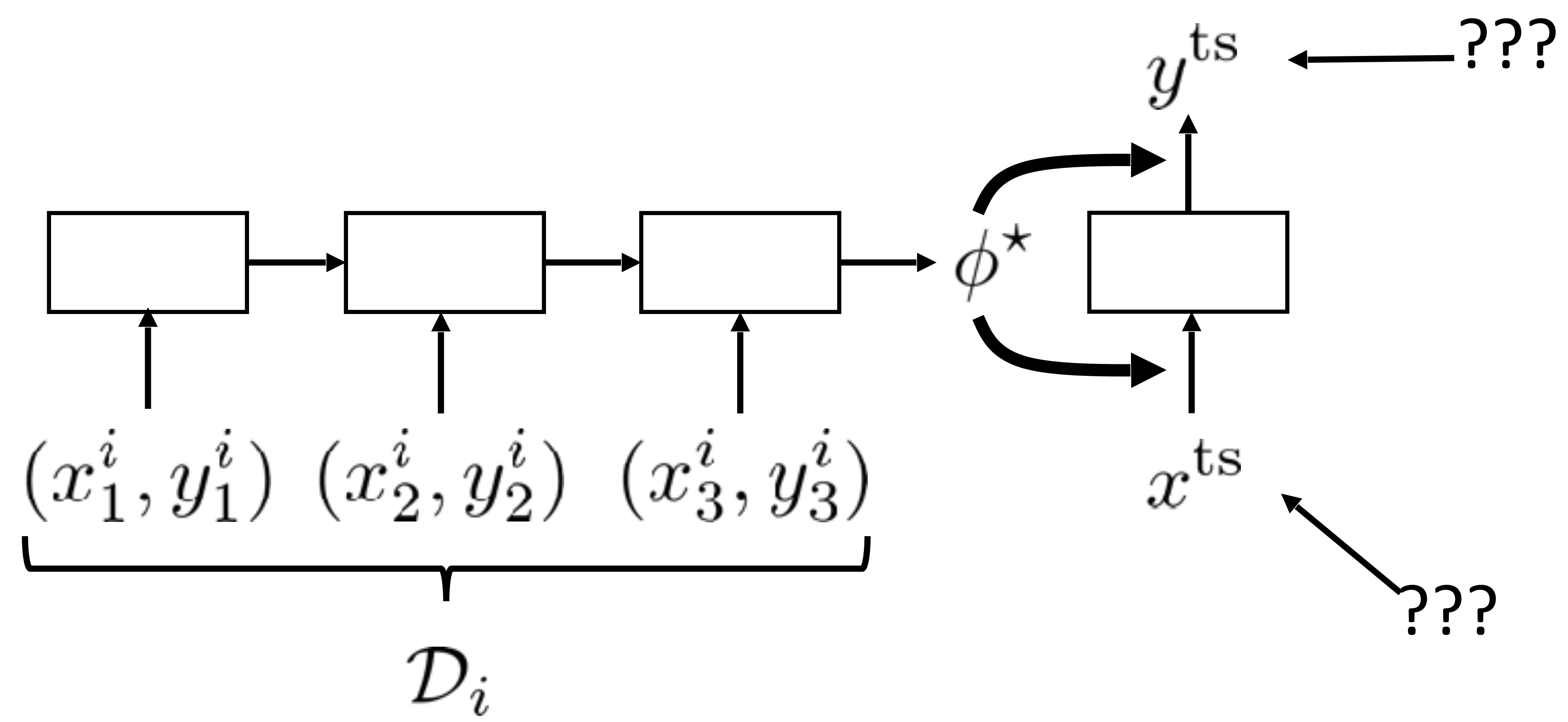
$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

(meta) training-time





Key idea:

"our training procedure is based on a simple machine learning principle: test and train conditions must match"

Vinyals et al., Matching Networks for One-Shot Learning

Reserve a test set for each task!

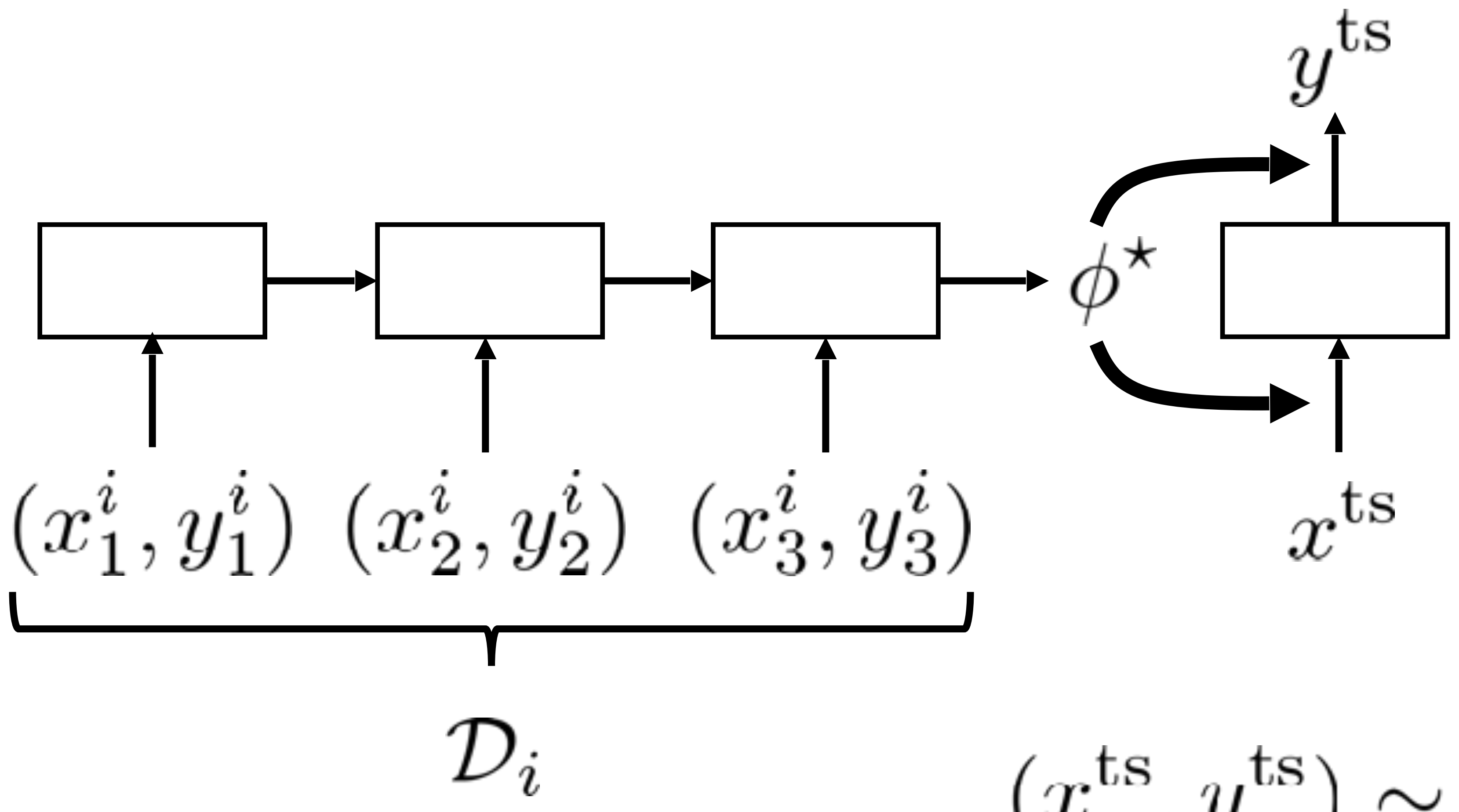
$\mathcal{D}_{\text{meta-train}}$

$\mathcal{D}_1$

$\mathcal{D}_2$

$\vdots$

(meta) training-time



$(x^{\text{ts}}, y^{\text{ts}}) \sim \mathcal{D}_i^{\text{ts}}$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \dots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \dots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \dots, (x_l^i, y_l^i)\}$$

Key idea:

"our training procedure is based on a simple machine learning principle: test and train conditions must match"

Vinyals et al., Matching Networks for One-Shot Learning

The complete meta-learning optimization

meta-learning: $\theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$

adaptation: $\phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}^{\text{tr}}, \theta^\star)$

$\Updownarrow$

$\phi^\star = f_{\theta^\star}(\mathcal{D}^{\text{tr}})$

learn θ such that $\phi = f_\theta(\mathcal{D}_i^{\text{tr}})$ is good for $\mathcal{D}_i^{\text{ts}}$

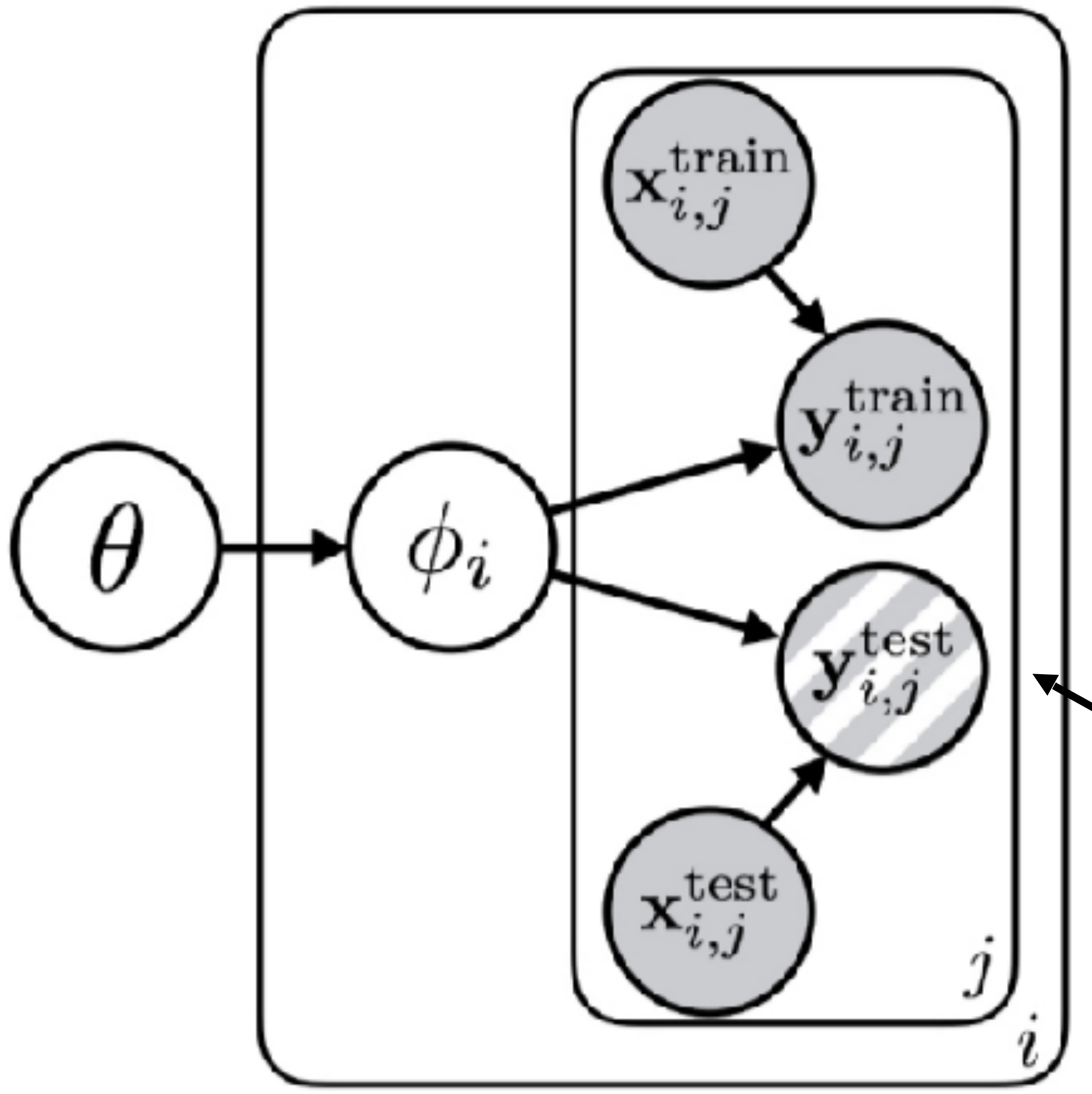
$$\theta^\star = \max_\theta \sum_{i=1}^{n} \log p(\phi_i|\mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$

$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$

$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$



unobserved at meta-test time

Some meta-learning terminology

learn θ such that $\phi_i = f_\theta(\mathcal{D}_i^{\mathrm{tr}})$ is good for $\mathcal{D}_i^{\mathrm{ts}}$

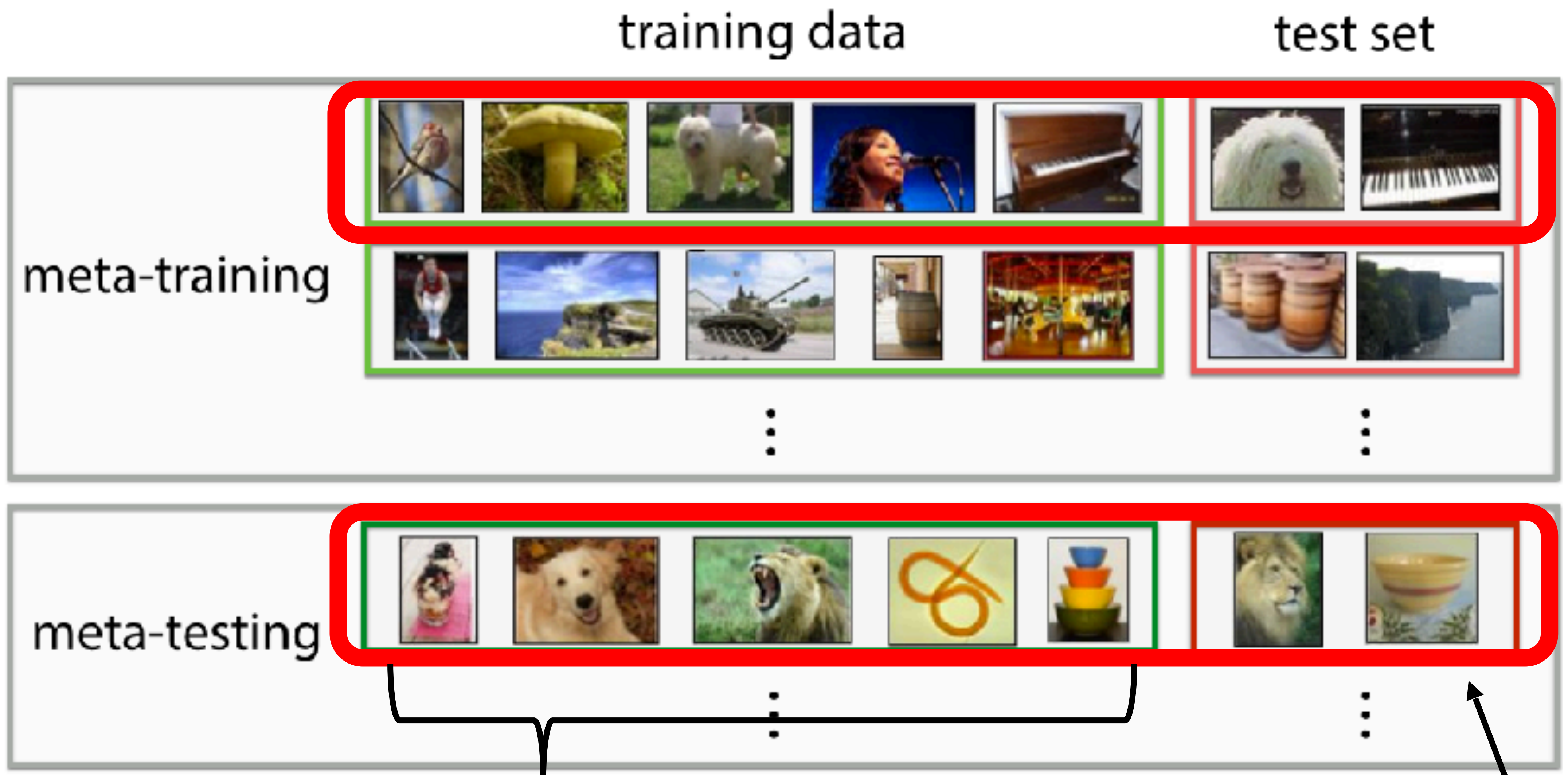
$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\mathrm{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\mathrm{tr}})$$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\mathrm{tr}}, \mathcal{D}_1^{\mathrm{ts}}), \ldots, (\mathcal{D}_n^{\mathrm{tr}}, \mathcal{D}_n^{\mathrm{ts}})\}$$

$$\mathcal{T}_i \begin{cases} \mathcal{D}_i^{\mathrm{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\} \\ \mathcal{D}_i^{\mathrm{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\} \end{cases}$$

shot
(i.e., k-shot, 5-shot)



(meta-training) task $\mathcal{T}_i$

$\mathcal{D}_1$

$\mathcal{D}_2$

$\mathcal{D}_{\text{meta-train}}$

(meta-test) task

support (set)

query

image credit: Ravi & Larochelle '17

Closely related problem settings

meta-learning:

$$\theta^\star = \max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$

multi-task learning: learn model with parameters $\theta^\star$ that solves multiple tasks $\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\theta | \mathcal{D}_i)$

can be seen as special case where $\phi_i = \theta$ (i.e., $f_\theta(\mathcal{D}_i) = \theta$)

hyperparameter optimization & auto-ML: can be cast as meta-learning

hyperparameter optimization: θ = hyperparameters, ϕ = network weights

architecture search: θ = architecture, ϕ = network weights

very active area of research! but outside the scope of this course

Plan for Today

Multi-Task Learning

- Models & training
- Challenges
- Case study of real-world multi-task learning

— *short break* —

Meta-Learning

- Problem formulation
- **General recipe of meta-learning algorithms**
- Black-box adaptation approaches

} Topic of Homework 1!

Rest will be covered next time.

Reminders

Homework 1 posted today, due **Wednesday, October 9**

Fill out **paper preferences** by tomorrow.

TensorFlow review session **tomorrow, 4:30 pm in Gates B03**