[An Overview of Multi-Task Learning in Deep Neural Networks]

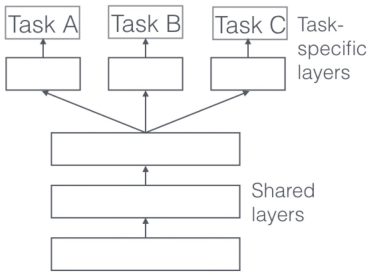
Multi-Task Learning (MTL)

: By sharing representations between related tasks, we can enable our model to generalize better on our original task.

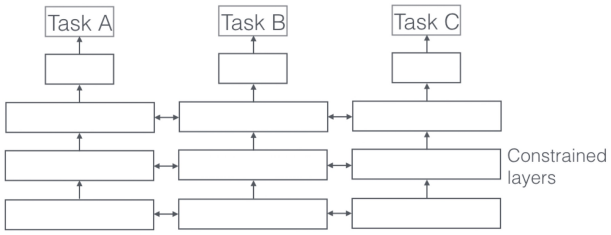
⊜ Joint learning, learning to learn, learning with auxiliary task
(optimizing more than one loss function)

Two MTL Methods for Deep Learning:

① Hard parameter sharing of hidden layers



② Soft parameter sharing of hidden layers



Each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar.

Why does MTL work?

Assume: two related tasks A and B rely on a common hidden layer representation F.

[1] Implicit data augmentation

MTL aims to learn a good representation that ignores the data-dependent noise and generalizes well.

[2] Attention focusing

MTL can help the model focus its attention on those features that actually matter as other tasks will provide additional evidence for the relevance or irrelevance of those features.

[3] Eavesdropping

Some features G are easy to learn for some task B, while being difficult to learn for another task A.

[4] Representation bias

MTL biases the model to prefer representations that other tasks also prefer.
MTL also helps the model to also perform well for learning novel tasks as long as they are from the same environment.

[5] Regularization

MTL reduces the risk of overfitting as well as the Rademachar complexity of the model
(= its ability to fit random noise)

< MTL in non-neural models >

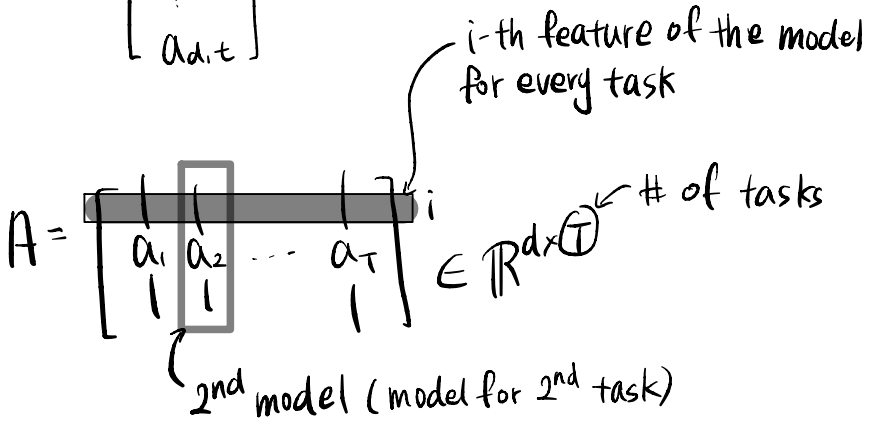
Existing literature on MTL
- linear models
- Kernel methods
- Bayesian algorithms

Past two pervasive main ideas

- enforcing sparsity across tasks through norm regularization
- modelling the relationships between tasks.

▨ Block-sparse regularization

a model (parameters) for task t : $a_t = \begin{bmatrix} a_{1,t} \\ a_{2,t} \\ \vdots \\ a_{d,t} \end{bmatrix} \in \mathbb{R}^{d}$ ← # of features



Sparsity assumption: all models share a small set of features
⇔ only a few features being used across all tasks
⇔ generalize ℓ_1 norm to the MTL setting

ℓ_1 norm: a constraint on the sum of the parameters, which enforces all but a few parameters to be exactly 0

step ① $\ell_q = \|\ \|_q$ $\quad b = \begin{bmatrix} \|a_1\|_q \\ \|a_2\|_q \\ \vdots \\ \|a_T\|_q \end{bmatrix}$ step ② $\ell_1 = \|b\|_1$ ← forces all but a few entries of b (a few rows in A) to be 0

Learning task relationships

a model (parameters) for task t : $a_t = \begin{bmatrix} a_{1,t} \\ a_{2,t} \\ \vdots \\ a_{d,t} \end{bmatrix} \in \mathbb{R}^{d}$ ← # of features

$$A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_T \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d \times T}$$

- i-th feature of the model for every task
- # of tasks
- 2^{nd} model (model for 2^{nd} task)

In some cases, sharing information with an unrelated task might actually hurt performance, a phenomenon known as negative transfer.

∴ Leverage prior knowledge indicating that some tasks are related while others are not.

→ a constraint that enforces a clustering of tasks might be more appropriate.

ex) $\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T} \sum_{t=1}^{T} \|a_{\cdot t} - \bar{a}\|^2$

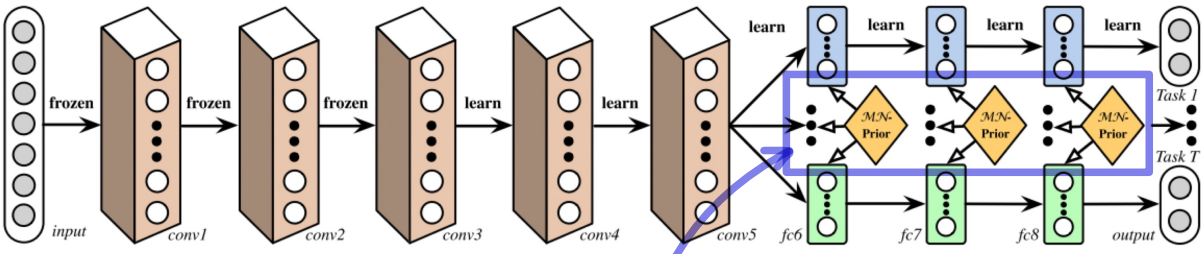
$\bar{a} = \frac{(\sum_{t=1}^{T} a_{\cdot t})}{T}$

This penalty enforces a clustering of the task parameter vectors $a_{\cdot 1}, \ldots a_{\cdot t}$ towards their mean that is controlled by λ

< Recent work on MTL for Deep Learning >

While many recent Deep Learning approaches have used multi-task learning (either explicitly or implicitly) as part of their model,
only a few papers have looked at developing better mechanisms for MTL in deep neural networks.

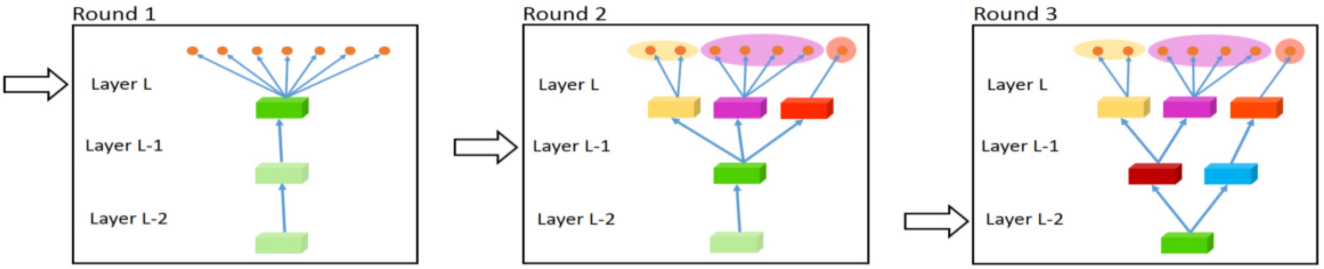
▨ Deep Relationship Networks



Shared CNN ⊕ task-specific FC

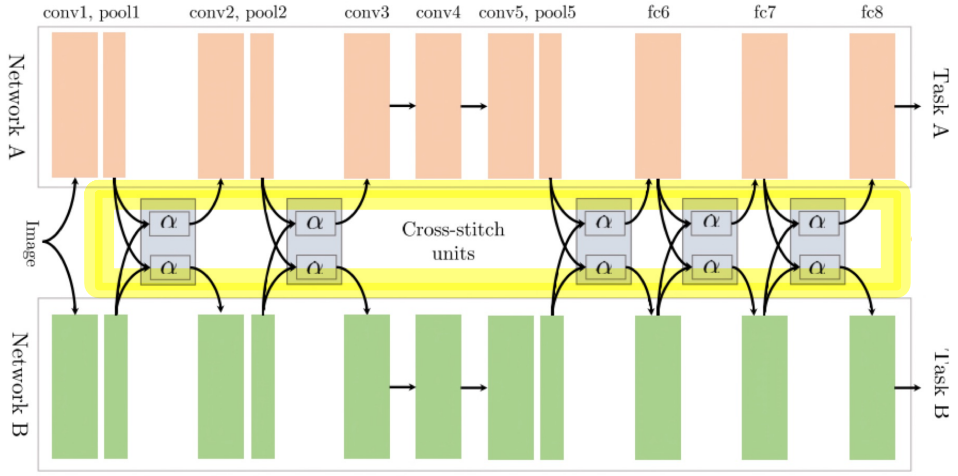
matrix priors
: allow the model to learn the relationship between tasks

▨ Fully-adaptive Feature Sharing



a bottom-up approach that starts with a thin network and dynamically widens it greedily during training using a criterion that promotes grouping of similar tasks.

Cross-stitch Networks

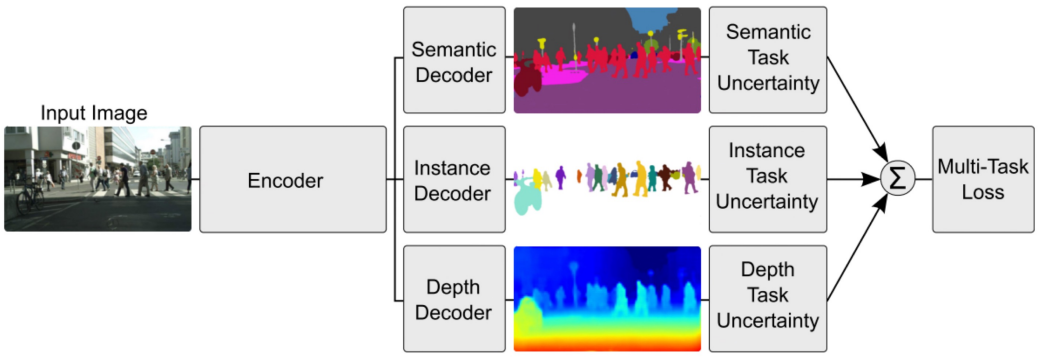


- Two separate model architectures
- Cross-stitch units: allow the model to determine in what way the task-specific networks leverage the knowledge of the other task by learning a linear combination of the previous layers. (placed after pooling or FC layers only)

Low Supervision (NLP)

A Joint Many-Task Model (NLP)

Weighting losses with Uncertainty

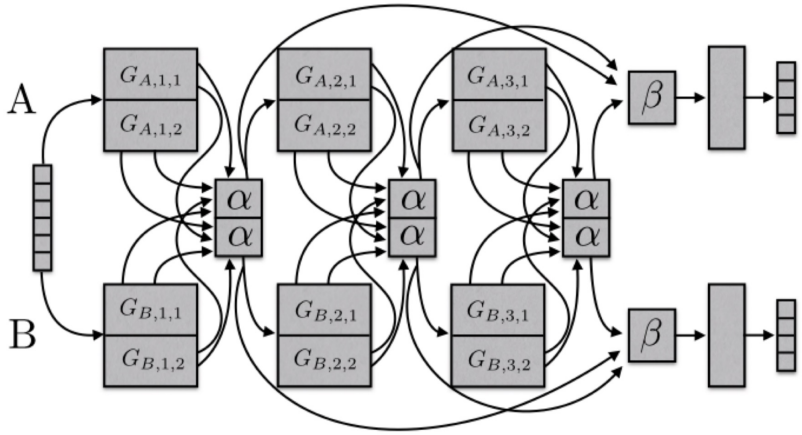


- Orthogonal approach by considering the uncertainty of each task.
- Adjust each task's relative weight in the cost function by deriving a multi-task loss function based on maximizing the Gaussian likelihood with task-dependant uncertainty.

Tensor factorization for MTL

- split the model parameters into shared and task-specific parameters for every layer

Sluice Networks



- allows to learn what layers and subspaces should be shared, as well as as what layers the network has learned the best representations of the input sequences.

What should I share in my model?

In order to develop robust models for MTL, we thus have to be able to deal with unrelated or only loosely related tasks.

Hard sharing quickly breaks down if tasks are not closely related or require reasoning on different levels.

⟨Auxiliary tasks⟩

In most situations, we only care about the performance on one task.
→ find a suitable auxiliary task in order to still reap the benefits of multi-task learning

Related task

ex) jointly predicts the class and the coordinates of an object in an image.

Adversarial

- Have access to a task that is opposite of what we want to achieve.
- Beneficial for the main task as it forces the model to learn representations that cannot distinguish between domains.

Hints

- Predict the features as an auxiliary task.
  ex) predict whether a name is present in a sentence as auxiliary task for name error detection.

Focusing Attention

- Focus attention on parts of the image that a network might normally ignore.

Quantization Smoothing

- Typically, labels are available as a discrete set. (ex. positive / neutral / negative)
  Using less quantized auxiliary tasks might help in these cases, as they might be learned more easily due to their objective being smoother.

Predicting Inputs

- The features that are unhelpful for predicting the desired objective can be used as outputs rather than inputs to guide the learning of the task.

Using the future to predict the present

- Some features only become available after the predictions are supposed to be made.

  ex) for self-driving cars, more accurate measurements of obstacles and lane markings can be made once the car is passing them.

  The additional data can be used as an auxiliary task to impart additional to the model during training

Representation Learning

- A more explicit modelling by, for instance, employing a task that is known to enable a model to learn transferable representations.

  ex) autoencoder objective

- Still do not know what auxiliary task will be useful in practice.
- Still do not know when two tasks should be considered similar or related
- Still, hard parameter sharing paradigm is pervasive.