Multi-Task Learning

Zhiping (Patricia) Xiao
Tuesday Paper-Reading Group

January 21, 2020
Overview
   List of Papers
   Background

Architecture
   Adaptive Feature Sharing
   Soft Parameter-Sharing

Optimization
   Optimal Task Weights
   Multi-Objective Optimization

Application
   Examples
Overview
1. An Overview of Multi-Task Learning in Deep Neural Networks (ArXiv’17, cited by 521) ¹

2. Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification (CVPR’16)

3. Training Complex Models with Multi-Task Weak Supervision (AAAI’19)

4. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics (CVPR’17)

5. (*) Multi-Task Learning as Multi-Objective Optimization (NeurIPS’18)

¹Originally a blog.
Multi-Task Learning (MTL) is often referred to as joint learning, learning with auxiliary tasks, etc.

It could be applied to either neural models or non-neural models.

The relationship among the tasks matter a lot.

Multiple tasks working together should ideally perform better on each task than they work individually.
Assume that we have $T$ tasks, each corresponding to a model $m_t$ where $t = 1, 2, \ldots, T$, and $d$ parameters, written as a column vector $a_{.,t}$:

$$a_{.,t} = \begin{bmatrix} a_{1,t} \\ \vdots \\ a_{d,t} \end{bmatrix}$$

Put $T$ columns together we have a matrix $A \in \mathbb{R}^{d \times T}$. $a_i$ is the $i^{th}$ row of $A$ containing the $i^{th}$ feature across all tasks.

**Assumption:** All tasks share the same important dimensions.

**Sparsity Assumption:** all models share a small set of features, which means that all but a few rows of $A$ are zero.
Recall that $\ell_1$ norm is also called LASSO (Least Absolute Shrinkage and Selection Operator). It is a constraint on the sum of $\beta$ and thus forces all but a few parameters to be exactly 0. (Explanation comes from here.)

Considering a simple model $y = \beta x$ and $\beta \in \mathbb{R}$, we have a loss function:

$$\mathcal{L} = \min \|y - \beta x\|_2^2$$

$$= \min \left( y^T y - 2\beta y^T x + \|\beta x\|_2^2 \right)$$

$$= \min \left( \beta^2 x^T x - 2\beta y^T x \right)$$

Assuming the least-square solution $\hat{\beta} \geq 0$
To which we could add a norm to regularize $\beta$. If it is an $\ell_1$ norm we have:

$$\mathcal{L} = \min \left( \beta^2 x^T x - 2\beta y^T x + \lambda |\beta| \right)$$

Taking $\mathcal{L}$’s derivative on $\beta$, we have:

$$2x^T x \hat{\beta} - 2y^T x + \lambda \text{sign}(\beta) = 0$$

When $\hat{\beta} > 0$ we might be able to reach the optimal point, when it is smaller than zero, since we have an $\hat{\beta} > 0$, we know that $y^T x > 0$, and $\lambda > 0$ by definition; the curve is decreasing (the optimal point $y^T x + \lambda/2$ is positive will be on the positive side).
Explaining the use of LASSO III

.

When using $\ell_1$ norm we have:

$$
\hat{\beta} = \begin{cases} y^T x - \lambda/2 & \beta > 0, 
\quad y^T x > \lambda/2 \\
0 & \beta \leq 0
\end{cases}
$$

If it is an $\ell_2$ norm instead of $\ell_1$, then:

$$2x^T x \hat{\beta} - 2y^T x + \lambda \hat{\beta} = 0$$

$$\hat{\beta} = \frac{y^T x}{x^T x + \lambda/2}$$

Since $\lambda > 0$ and $x^T x \geq 0$, by definition we have $y^T x > 0$, thus $\hat{\beta} > 0$. 
Revisit the definition of $A \in \mathbb{R}^{d \times T}$, whose $t^{th}$ column is $a_{.,t}$, containing all parameters from task $t$:

$$
a_{.,t} = \begin{bmatrix}
a_{1,t} \\
\vdots \\
a_{d,t}
\end{bmatrix}
$$

$i^{th}$ row of $A$ is $a_i$, containing the $i^{th}$ feature across all tasks.

Constraints are therefore applied to $A$ to make only a few features non-zero, regularizing by $\ell_1/\ell_q$ norm:

$$
\left\| \begin{bmatrix}
\|a_1\|_q \\
\|a_2\|_q \\
\vdots \\
\|a_d\|_q
\end{bmatrix} \right\|_1
$$

\hspace{1cm} 2^* Distributed Multi-Task Relationship Learning (KDD'17)
Adding another constraint: \(^3\)

\[
\Omega = \|\bar{a}\|^2 + \frac{\lambda}{T} \sum_{t=1}^{T} \|a_{.,t} - \bar{a}\|^2
\]

where \(\bar{a} = \sum_{t=1}^{T} a_{.,t}/T\).

It enforces the task parameters to be optimized towards their mean, so as to make them more related.
Figure: Hard parameter sharing. \(^4\)

\(^4\)More popular than soft-parameter sharing.
**Figure:** Soft parameter sharing.
Architecture
Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification (Yongxi Lu et al.)

- Base model: CNN.
- Tasks are pre-defined, the network architecture is automatically-determined.
- Hard-parameter-sharing.

Person attributes: face, cloth, etc.
Figure: Splitting of the layers only applies to one layer at a time, a greedy top-down manner starting from the top layer.
How to split the current layer?

By measuring two tasks' \((i,j)\) affinity (similarity) \(A(i,j)\), using the votes from data samples. Denote the prediction of the task \(i\) for sample \(n\) as \(s_{ni}\), and the ground-truth label \(t_{ni}\), then the error margin is \(m_{ni} = |t_{ni} - s_{ni}|\). \(m_i \in \mathbb{R}^N\). If for task \(i\), a sample \(n\) makes \(m_{ni} \geq E\{m_i\}\), we say \(n\) is a difficult example for task \(i\). Labels are binary in this case. With this definition of possible \(n\)'s, we define the affinity of two task \(A(i,j)\) as:

\[
A(i,j) = P(e_{ni} = 1, e_{nj} = 1) + P(e_{ni} = 0, e_{nj} = 0) = E\{e_{ni}e_{nj}\} + (1 - e_{ni})(1 - e_{nj})
\]
How to split the current layer?

By measuring two tasks’ \((i, j)\) affinity (similarity) \(A(i, j)\), using the votes from data samples.

Denote the prediction of the task \(i\) for sample \(n\) as \(s^n_i\), and the ground-truth label \(t^n_i\), then the error margin is \(m^n_i = |t^n_i - s^n_i|\). \(m_i \in \mathbb{R}^N\). If for task \(i\), a sample \(n\) makes \(m^n_i \geq \mathbb{E}\{m_i\}\), we say \(n\) is a difficult example for task \(i\). Labels are binary in this case.
How to split the current layer?

By measuring two tasks’ \((i, j)\) affinity (similarity) \(A(i, j)\), using the votes from data samples.

Denote the prediction of the task \(i\) for sample \(n\) as \(s_i^n\), and the ground-truth label \(t_i^n\), then the error margin is \(m_i^n = |t_i^n - s_i^n|\). \(m_i \in \mathbb{R}^N\). If for task \(i\), a sample \(n\) makes \(m_i^n \geq \mathbb{E}\{m_i\}\), we say \(n\) is a **difficult example** for task \(i\). Labels are **binary** in this case.

With this definition of possible \(n\)’s, we define the affinity of two task \(A(i, j)\) as:

\[
A(i, j) = \mathbb{P}(e_i^n = 1, e_j^n = 1) + \mathbb{P}(e_i^n = 0, e_j^n = 0)
= \mathbb{E}\{e_i^n e_j^n + (1 - e_i^n)(1 - e_j^n)\}
\]
The affinity of two branches \((k, l)\) is \(A_b(k, l)\):

\[
\tilde{A}_b(k, l) = \text{mean}_{i_k} \left( \min_{j_l} A(i_k, j_l) \right)
\]

\[
\tilde{A}_b(l, k) = \text{mean}_{i_l} \left( \min_{j_k} A(i_l, j_k) \right)
\]

\[
A_b(k, l) = \frac{\tilde{A}_b(k, l) + \tilde{A}_b(l, k)}{2}
\]

Determined by the largest distance between their associated tasks.
There’s always a maximum value of clusters $c$ (originally the same as the number of tasks, will be decreased to the number of clusters in the above layer).

The number of clusters selected is $d$, $1 \leq d \leq c$. $g_d : [d] \rightarrow [c]$ is the temporary layer’s grouping function, associating the current layer with the above one.

The loss function at layer $l$ is (optimized over $g_d$):

$$L^l(g_d) = (d - 1)L_02^{p_l} + \alpha L_S(g_d)$$

where $L_0$ is the unit cost for branch-creation, $p_l$ is the number of pooling layers above the layer $l$, larger branching factor $\alpha$ encourages more branches, $L_S(g_d)$ is a penalty for separation.
For each newly-created branch $i \in [d]$, its separation cost is:

$$L^i_s(g_d) = 1 - \text{mean}_{k \in g^{-1}(i)} \left( \min_{l \in g^{-1}(i)} A_b(k, l) \right)$$

where $k, l$ are branches from the above layer. It measures the maximum distances (minimum affinity) between the tasks within the same group. It penalizes dissimilar tasks grouped into the same branch.

$$L_s(g_d) = \frac{1}{d} \sum_{i \in [d]} L^i_s(g_d)$$
Soft Parameter-Sharing at a glance \(^6\)

**Figure:** The shared representations is learned by utilizing a linear combination of input activation maps from both tasks, at each layer. Cross-stitch units model shared representations as a linear combination of input activation maps.

\(^6\)(* Cross-stitch Networks for MTL (CVPR’16)
Given two activation maps $x_A$, $x_B$ from layer $l$ from tasks $A$ and $B$, we learn linear combinations $\tilde{x}_A$, $\tilde{x}_B$ of them parameterized by $\alpha$, and feed those linear combinations into the next layer. Specifically, at location $(i, j)$:

$$
\begin{bmatrix}
\tilde{x}^{ij}_A \\
\tilde{x}^{ij}_B \\
\end{bmatrix} =
\begin{bmatrix}
\alpha_{AA} & \alpha_{AB} \\
\alpha_{BA} & \alpha_{BB} \\
\end{bmatrix}
\begin{bmatrix}
x^{ij}_A \\
x^{ij}_B \\
\end{bmatrix}
$$

The above is called stitching operation.
Figure: Applying the above-mentioned cross-stitching to two AlexNet, the units applied only after pooling layers and fully connected layers.
Optimization
Normally MTL’s objective is a linear combination of multiple objectives.

Alex Kendall et al. provides a more interpretable estimation by using task-dependent uncertainty $^7$, estimated by Bayesian modeling, to weigh the losses. We start the computation of $\mathcal{L}(W, \sigma_1, \ldots, \sigma_K)$ from defining multi-task likelihood:

$$p(y|f^W(x)) = \mathcal{N}(y; f^W(x), \sigma^2)$$
$$p(y_1, \ldots, y_K|f^W(x)) = p(y_1|f^W(x)) \cdots p(y_K|f^W(x))$$
$$= \mathcal{N}(y_1; f^W(x), \sigma^2_1) \cdots \mathcal{N}(y_K; f^W(x), \sigma^2_2)$$

where the distribution $\mathcal{N}(y; f^W(x), \sigma^2)$ is estimated by sampling from $\text{Softmax}(f^W(x))$.

$^7$Claimed to be captured by $\sigma$, while data uncertainty by $\mu$. 
The probability density of observing a single data point $x$ is:

$$p(x; \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( - \frac{(x - \mu)^2}{2\sigma^2} \right)$$

The laws of logarithms:

$$\log(ab) = \log(a) + \log(b)$$
$$\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$$
$$\log(a^n) = n \log(a)$$

Therefore $(− \log(\sqrt{2\pi})$ is constant and could be removed latter):

$$\log p(x; \mu, \sigma) = − \frac{(x - \mu)^2}{2\sigma^2} − \log(\sigma) − \log(\sqrt{2\pi})$$
For a single-task case, the log-likelihood (to be maximized) is:

\[ \log p(y|f^W(x)) \propto -\frac{1}{2\sigma^2} \| y - f^W(x) \|^2 - \log \sigma \]

and for multi-task (where there’re only 2 tasks), we take the **negative** of the log-likelihood to compute our objective \( L(W, \sigma_1, \sigma_2) \):

\[ L(W, \sigma_1, \sigma_2) \propto \frac{1}{2\sigma_1^2} \| y_1 - f^W(x) \|^2 + \frac{1}{2\sigma_2^2} \| y_2 - f^W(x) \|^2 + \log \sigma_1 \sigma_2 \]
Let's define the loss for the two outputs \((y_1 \text{ and } y_2)\) respectively:

\[
\mathcal{L}_1(W) = \|y_1 - f^W(x)\|^2
\]

\[
\mathcal{L}_2(W) = \|y_2 - f^W(x)\|^2
\]

Then we have:

\[
\mathcal{L}(W, \sigma_1, \sigma_2) \approx \frac{1}{2\sigma_1^2} \mathcal{L}_1(W) + \frac{1}{2\sigma_2^2} \mathcal{L}_2(W) + \log \sigma_1 \sigma_2
\]

Interpret: when \(\sigma_i\) goes up, the relative weight of task \(i\)
\(\lambda_i = \frac{1}{2\sigma_i^2}\) goes down.

(*) There’s an extension from Gaussian distribution to Boltzmann (Gibbs) distribution for classification likelihoods’ computation in the paper (p 5).
Training: In practice, they train the network to predict the log variance \( s = \log \sigma^2 \), because:

- the loss avoids any division by zero, thus it is more numerically stable than regressing the variance, \( \sigma^2 \) directly.
- we can regress unconstrained scalar values, since \( \exp(\cdot) > 0 \).
Ozan Sener & Vladlen Koltun work on solving the problem of multiple tasks might conflict.

They used the multiple-gradient-descent algorithm (MGDA) optimizer, together with a definition of the Pareto optimality for MTL (in brief, no other solutions dominants the current solution), they solve the problem by solving the KKT (Karush-Kuhn-Tucker) conditions.

- Basic Idea: Use multi-objective Karush-Kuhn-Tucker KKT conditions and find a descent direction that decreases all objectives.

- Can applicable to any problem that uses optimization based on gradient descent.
Application
Figure: The goal is to apply a set of weak supervision sources $S = \{s_1, s_2, \ldots, s_m\}$ to an unlabeled dataset $\mathcal{X}$ consisting of $n$ data points, and finally apply the label to an end-model $f_w : \mathcal{X} \rightarrow \mathcal{Y}$. 
Training Complex Models with Multi-Task Weak Supervision

- Feasible Set of input: $X$, feasible set of labels: $Y$, $t$ tasks in total, $m$ multi-task weak supervision sources; $X$ and $Y$ are independent.

- $X \in X$: a data point

- $Y \in Y$: unobservable ground-truth task label, $Y = [Y_1, Y_2, \ldots, Y_t]^T$, where for each $Y_i$, $Y_i \in \{1, 2, \ldots, k_i\}$.

- **Examples** of tasks: (1) Person v.s. Organization; (2) Doctor v.s. Lawyer (or N/A); (3) Hospital v.s. Office (or N/A).

- **Tasks’ labels** are logically subsumption. e.g. $Y_1 = PERSON$, $Y_2 = DOCTOR$, $Y_3 = N/A$ makes a valid $Y \in Y$. ($Y_2 = DOCTOR$ implies $Y_3$ and $Y_1$.)
Y is not directly observable, instead, we have access to m multi-task weak supervision sources $s_i \in S, i = 1, 2, \ldots, m$; which emits label vectors $\lambda_i$.

Each $\lambda_i$ contains labels for some of the tasks, but not for all of them. For the null / abstaining labels, we use 0 to denote. The coverage set $\tau_i \subseteq \{1, 2, \ldots, t\}$ is fixed for each $s_i$, indicating the tasks that this source could emit non-zero labels for.

Still consider the previous example, assume that for a certain $X$ the corresponding $Y$ is $[PERSON, DOCTOR, N/A]^T$, we’ll probably see $\lambda_i = [PERSON, 0, N/A]^T$, where the coverage set $\tau_i = \{1, 3\}$. 
There’s a non-directional $G_{source} = \{V, E\}$ that is supposed to be given by user who runs this model, where $V = \{Y, \lambda_1, \lambda_2, \ldots, \lambda_m\}$. If $(\lambda_i, \lambda_j)$ doesn’t exists, it means that the two sources are independent of each other. All $(\lambda_i, Y)$ always exists.

We define the set of cliques (complete subgraph) of $G_{source}$ as $\mathcal{C}$. Recall that $Y$ is not observable, therefore we define observable cliques set $\mathcal{O} \subset \mathcal{C}$, s.t. we can use it to help with a matrix completion task.

(*) The users also need to provide $G_{task}$, whose structure expresses logical relationships between tasks.
Figure: For example, in this case, \( C = \{Y, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \{Y, \lambda_1\}, \{Y, \lambda_2\}, \{Y, \lambda_3\}, \{Y, \lambda_4\}, \{\lambda_1, \lambda_2\}, \{Y, \lambda_1, \lambda_2\}\} \), and \( O = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \{\lambda_1, \lambda_2\}\} \)
For $C \in \mathcal{C}$, defining $\psi(C)$ as the vector of indicator random variables for all combinations of all but one of the labels emitted by each variable ($G_{source}$ vertices) in clique $C$, a minimal set of statistics. For $\psi(C)$, it is a collection of all $\psi(C)$ where $C \in \mathcal{C}$.

$$\psi(C, y_C) = 1\{\cap_{i \in C} V_i = (y_C)_i \mid (y_C)_i \in \mathcal{Y}_{\tau_i}\}$$

$$\psi(C) \in \{0, 1\} \prod_{i \in C} (|\mathcal{Y}_{\tau_i}| - 1)$$

The separator set cliques of the junction tree (assumed to be unique in a simplification condition) is annotated as $S \subseteq \mathcal{C}$. 
What to do next: to learn the **label model** $P_\mu(Y|\lambda)$, which is parameterized by $\mu$, a vector of source correlations and accuracies, and outputs a single probabilistic label vector $\tilde{Y}$.

- $\mu = \mathbb{E}[\psi(C)]$
- $\text{Cov}[\psi(O), \psi(S)]$ is a function of $\mu$, and it is **unobserved**.
Recall that we said: $\Sigma_{OS} = \text{Cov}[\psi(O), \psi(S)]$ is a function of $\mu$, and it is unobserved. We also have:

- $\Sigma_O = \text{Cov}[\psi(O)]$ be observed.
- $\Sigma_S = \text{Cov}[\psi(S)] = \text{Cov}[\psi(Y)]$ is estimable, it is a function of $P(Y)$.
- The mathematical calculation is based on:

$$\text{Cov}[\psi(O \cup S)] \equiv \Sigma = \begin{bmatrix} \Sigma_O & \Sigma_{OS} \\ \Sigma_T & \Sigma_S \end{bmatrix}$$

All the remaining is matrix computation details, working on matrix completion. For details, please visit the longer version of this paper on ArXiv.