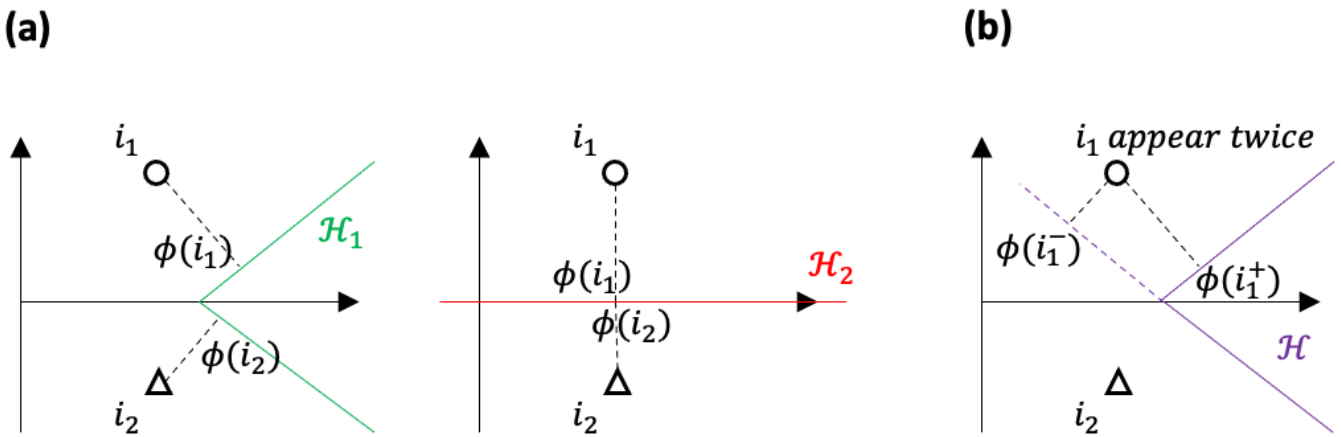
Pre-trained representations

Two mainstream pre-training methods in IR:

1. Contrastive-learning-based (unsupervised, construct similar and dissimilar pairs, inner product)
2. Same-structure-based (supervised, use the same structure for both pre-training and downstream task)

Many use pre-trained representations as numerical feature vectors and feed them into downstream task. Is it truly valid?

For instance, Xu et al.[7] shows that **suprious dimension** and **imperfect negative sampling** contrastive-learning-based embedding not suitable to use as numerical feature vector.



[7] Rethinking Pre-trained Embedding for E-commerce ML, preprint'22

Pre-trained representations

Formally, model design will have critical impact on generalization:

Theorem (i.i.d generalization)

Let $\mathcal{R}_n(\Phi)$ and $\mathcal{G}_n(\Phi)$ be the empirical Rademarcher and Gaussian complexity. Let R^*_{task} be the best risk that can be achieved in the downstream task.

***Same-structure pre-training**.*

$$R_{task}(\hat{\phi}) - R^*_{task} \lesssim \frac{\mathcal{G}_n(\Phi)}{\sqrt{n}} + \frac{\tilde{\mathcal{G}}(\mathcal{F})}{n} + slack$$

where $\tilde{\mathcal{G}}(\mathcal{F})$ is a complexity measure of $\mathcal{F}$.

***constrastive pre-training**.*

$$R_{task}(\hat{\phi}) - R^*_{task} \lesssim \frac{R\mathcal{R}_n(\Phi)}{n} + R \cdot \mathbb{E}_y \left\| cov_{P_X^{(y)}}(\phi^*) \right\|_2 + slack,$$

where $cov_{P_X^{(y)}}(\cdot)$ represents the uncertainty of constrative learning.

Pre-trained representations

Same-structure pre-training has better theoretical guarantee, but contrastive pre-training is obviously more versatile. How do we understand its cross-domain generalization?
Again, we resort to the **NTK analysis**. Recall that:

$$f\left(\theta;\phi(x)\right) \approx \left\langle \theta-\theta_0, \nabla f\left(\theta^{(0)};\phi(x)\right)\right\rangle,$$

and the NTK under pre-trained embedding is:

$$K_{\mathrm{NTK}}\Big(\phi(x),\phi(x')\Big) = \left\langle \nabla f\left(\theta^{(0)};\phi(x)\right), \nabla f\left(\theta^{(0)};\phi(x')\right)\right\rangle.$$

Also recall that the optimization and generalization properties of the downstream model $f(\theta;\cdot)$ is characterized by the NTK. So how does K_{NTK} relate to ϕ? Note that in contrastive pre-training, the algorithm itself defines an embedding kernel:$K_\phi(x,x') = \left\langle \phi(x),\phi(x')\right\rangle$.

Pre-trained representations

Using the two-layer MLP for example, it is shown that:

$$\mathrm{NTK}\Big(\phi(x), \phi(x')\Big) = 1 - \frac{1}{\pi} \cos^{-1} \frac{\big\langle \phi(x), \phi(x') \big\rangle}{\|\phi(x)\| \cdot \|\phi(x')\|},$$

which essentially means the NTK is a composition of the arc-cosine kernel and the **embedding kernel** K_ϕ!
We reach a novel interpretation for the role of pre-trained embeddings in cross-domain downstream task:

- the pre-trained embeddings influence the performance of downstream task by determining the input of the NTK, i.e. via $\mathrm{NTK}\Big(\phi(x), \phi(x')\Big) = \sigma\Big(K_\phi(x, x')\Big)$;
- if K_ϕ already aligns well with the downstream task, then even a simple $\sigma(\cdot)$, e.g. $\sigma(t) = c \cdot t$ as induced by a linear predictor f, can achieve good performance;
- otherwise, it can be difficult to find a model that has the optimal mapping in terms of $\sigma(\cdot)$, such that $\sigma\big(K_\phi\big)$ aligns with the downstream task.

Pre-trained representations

Does the same analysis hold for sequential model?

Example (Sequential task)

Suppose the downstream task is to recover $y \in \mathbb{R}^+$ according the fixed-length sequences of $s = (x_1, \ldots, x_k)$, and the downstream model is a linear function of the concatenated pre-trained embeddings, i.e.

$$f\left(\theta; s, \phi\right) = \theta^\intercal \left[\phi(x_1), \ldots, \phi(x_k)\right],$$

We use $\Phi \in \mathbb{R}^{n \times kd}$ to denote the matrix of concatenated embeddings. Given the outcome $\mathrm{y} \in \mathbb{R}^n$, the least-square prediction is: $\Phi\left(\Phi\Phi^\intercal\right)^{-1}\Phi^\intercal \mathrm{y}$. Notice that the entries in $\Phi\Phi^\intercal$ – which determines the dual solution of the linear model – corresponds to a new sequence kernel composed entirely by K_ϕ:

$$K_\phi\left(s, s'\right) = \sum_{i=1}^{k} K_\phi\left(x, x'\right), \text{ for } s' = \left(x'_1, \ldots, x'_k\right).$$

Pre-trained representations

As a matter of fact, it is again the kernel alignment that determines the cross-domain generalization.
W.l.o.g, we use the kernel predictor for the downstream task, given by:

$$f_\phi(x) = \frac{E_{x'}\left[y' k_\phi(x, x')\right]}{\sqrt{\mathbb{E}[k_\phi^2]}}.$$

It holds with probability at least $1 - \delta$ that:

$$R^{\mathrm{OOD}}(f_\phi) \leq 1 - \frac{\mathbb{E}\left[K_Y(y, y') K_\phi(x, x')\right] \cdot \sqrt{\delta}}{\sqrt{\mathbb{E}[k_\phi^2]}}, \tag{2}$$

where $K_Y(y, y')$ is the downstream task kernel (given by $1[y = y']$) that represents its label generating mechanism. R^{OOD} means the risk is taken w.r.t. some other domain.
The result rigorously reveal the path way of **representation** $\rightarrow$ **kernel alignment** $\rightarrow$ **cross-domain generalization**.

Spectral analysis and Bellman equation

It is somewhat unorthodox to put these two topics together, but for the purpose of our tutorial, they both serve to analyze the **graph neural network**.

- **Spectral filtering** is an analogy to the time series filtering we are familiar with: it removes the high-frequency component which are likely to be noises. (why?)
- However, time series use the Fourier transformation to map signal from temporal to frequency domain. On the graph domain, Fourier transform is done through the lens of **Laplacian operator**.
- On the real line, the Laplace operator is the second derivative:

$$\Delta f = \lim_{\delta \to 0} \frac{f(x+\delta) - 2f(x) + f(x-\delta)}{\delta^2}.$$

- For functions f defined on the nodes of the graph, we use the discretized version for the above definition, which simply reduces to the difference of $f(i)$ and f evaluated at all the neighbors of node i:

$$\Delta f(i) = \sum_{j \in N(i)} f(i) - f(j).$$

Spectral analysis and Bellman equation

Fourier transform of a function f is the expansion of f in terms of the eigenfunctions of the Laplace operator. Laplacian matrix admits:

$$\tilde{L} = U\Lambda U^T, \quad U = [U_1, \ldots, U_{|\mathcal{V}|}] \text{ forms a set of orthogonal basis,}$$

and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_{|\mathcal{V}|}))$ consists of the eigenvalues.
Graph Fourier transform for graph function f, is the expansion of f in terms the eigenfunctions:

$$\hat{f}(\lambda_k) = \sum_{i=1}^{|\mathcal{V}|} f(i) U_k[i] = \langle \vec{f}, U_k \rangle,$$

where $\vec{f} = [f(1), \ldots, f(|\mathcal{V}|)]$. Using the shorthand, we have: $\vec{\tilde{f}} = U^T \vec{f}$.
The inverse Fourier transform is given by: $\vec{f} = U\vec{\tilde{f}}$.

Spectral analysis and Bellman equation

Graph convolution of f under g, which can be thought of as applying operations defined by g on the spectrum of the graph, can be formulated as:

$$g \star f(i) = \sum_{k=1}^{|\mathcal{V}|} \hat{g}(\lambda_k) \cdot \hat{f}(\lambda_k) U_k[i].$$

When we want g to serve as **filtering**, e.g. preserve and remove certain elements in the spectrum of $[\lambda_1, \ldots, \lambda_{|\mathcal{V}|}]$, we may directly parameterize g_θ.
if X is the node feature matrix, then the spectral filtering of X under g_θ is given by:

$$g_\theta \star X = U \mathrm{diag}\big(\hat{g}_\theta(\lambda_1), \ldots, \hat{g}_\theta(\lambda_{|\mathcal{V}|})\big) U^T X.$$

Look similar to graph convolutional network?

- **Bellman equation** often refers to the necessary condition for optimality of dynamic programming – a classical algorithmic solution for many ML problems!
- Many **message-passing-type** of ML algorithms have roots in dynamic programming: graphical model inference, reinforcement learning (backup computation), graph neural network (local aggregation).

In general, Bellman equation recursively defines the value of a state by other states (bootstrap).
For instance, when finding the optimal action-value function, one proceeds with:

$$Q(s,a) = \max_{a'} \sum_{s'} p(s'|s,a)\big\{r(s,a) + \gamma Q(s',a')\big\}.$$

When evaluating the action-value function w.r.t. a particular policy such that $a \sim \pi(s)$, one proceeds with:

$$Q_\pi(s,a) = \mathbb{E}_{a' \sim \pi(s)} \sum_{s'} p(s'|s,a)\big\{r(s,a) + \gamma Q(s',a')\big\}.$$

Domain topic: graph neural network for IR

Why graph neural networks are often found helpful in practice, especially for IR tasks that are often cross-domain? Our investigation leads to two possible explanations:

1. via applying invariant filtering mechanism
2. via implicitly solving bellman equation

Firstly, we mention that spectral filtering under a parameterized Chebyshev polynomial G_θ is:

$$g_\theta \star X = \theta(I + \tilde{L})X.$$

By stacking multiple graph convolution layers, a typical *graph convolutional network* is constructued as:

$$\ldots \sigma\Big(\theta_2(I + \tilde{L})\sigma\big(\theta_1(I + \tilde{L})X\big)\Big)\ldots.$$

Domain topic: graph neural network for IR

By simple algebra, we show that[8]:

$$(I + \tilde{L})^k X\theta_0 = U\theta_0 \text{diag}\big((2-\lambda_1)^k, \ldots, (2-\lambda_{|\mathcal{V}|})^k\big) U^T X,$$

so by doing $(I + \tilde{L})^k X\theta_0$, we are filtering the spectrum using the filter function $\theta_0(2-\lambda)^k$.
Note that $2 \geq \lambda_1, \ldots, \lambda_{|\mathcal{V}|} \geq 1$ due to the Laplacian matrix. As a consequence, the components that correspond to a large λ will be shrunken more heavily as we increase k since $2 - \lambda \leq 1$!
If this filtering mechanism is invariant across domains, then the mechanism works as follow:
shared filtering $\rightarrow$ **better aligned representation kernel after filtering** $\rightarrow$ **improved domain generalization**.

[8] The Mystery of Graph Neural Network for Cross-domain Generalization, coming soon

Domain topic: graph neural network for IR

From the Bellman equation side, note that many graph neural networks have local aggregation updates as follow:

$$h^{(k)}(u) = \min_{v \in \mathcal{N}(u)} NN^{(k)}\Big(h^{(k)}(u), h^{(k)}(v), w(u,v)\Big).$$

Recall the Bellman-Ford equation for finding the **shortest-path distance**:

$$d[k][u] = \min_{v \in \mathcal{N}(u)} d[k-1][v] + w(u,v).$$

Notice the similarity? Espeically that shortest-path distance also defines a kernel that measure **graph-based similarity**.
Therefore, the updates of graph neural network also implicitly approximates a *graph-based kernel*, by transforming the node features by its neighboring information. The mechanism thus works as follow:
message-passing GNN $\rightarrow$ implcitly approximate a Bellman equation that defines a particular graph kernel $\rightarrow$ the graph kernels are aligned across domains $\rightarrow$ improved generalization.

Model diagnostic for IR

The modern information retrieval systems are often complicated but also business critical. Desires to analyze and diagnose the models arise from the many common scenarios.

1. Help debug and trouble shooting
   "Why I see cantaloupe when I search for honeydew?"
2. Find improvement opportunities
   "I need to improve the search conversion by 1%. Where should I start?"
3. Help other stack holders understand the system
   "Please improve the taxonomy quality on this part, and we proved it"

Model diagnostic for IR

There are various ways of classifying the methods of diagnosing and analyzing the models:

1. global / local diagnostics methods
   Global methods explain model's average behavior while the local methods explain the model's output for specific examples.
   Sometimes local methods can be generalized to global methods.
2. model dependent / agnostic methods
   Models such as linear models and tree-based models can be analyzed with specialized methods which don't work for other models. While model agnostics treat the model as black-box predictor.

Attribution algorithm

Attribution methods find the contribution of each input features.

1. Perturbation-based
   Perturbation-based methods works on features space directly using operations including but not limited to removing, masking and permuting the features and then observe the impacts on the model outputs.
2. Backpropagation-based
   Backpropagation-based methods utilizes the gradients or outputs of the neurons during forward and backward propagation of the networks.

Problems of perturbation-based attribution for IR models

1. Feature correlations
   Correlated features might compete for attributions while features in IR model, such as CTR at different levels, do have lots of correlations.
2. Scalability
   IR models have hundreds of thousands of features. Not efficient to look one feature at a time.
3. Missing or noisy ground truth
   The ground-truth is rare and biased heavily by exposure mechanism while perturbation based method relies on the notion of impacts of correctness of the model.

Backpropagation-based attribution

The ideal attribution methods allows us to only use one or a few fixed mounts of forward and backward passes of the network to get the attributions for all the features of the target example.
We could apply the following generic backpropagation-based attribution methods to find explanations of the deep learning models used in IR system.

1. Gradient * input
2. *Integrated gradients*
3. *DeepLIFT*

There are other architecture specific methods such as *Grad-CAM*, *Deconvolutional network* and *Guided Backpropagation* whose targeting network structures are not commonly used in information retrieval system.

Gradient * input

Consider the attribution of the variable x_i as $R_i^c(x)$, the gradient * input is defined as:

$$x_i \cdot \frac{\partial S_c(x)}{\partial x_i}$$

where $S_c(x)$ is the c-th output of the network.
This is commonly used in computer vision models to generate the saliency map in which the graident * input is computed at very pixels of the images.
Different from CV, the IR system heavily relies on the embedding to model discrete objects. We would consider the gradients at the embedding level.

Integrated Gradients

The *Integrated Gradients* considers the path integral of the gradient from the baseline input $\bar{x}$ and the input x:

$$(x_i - \bar{x}_i) \cdot \int_{\alpha=0}^{1} \frac{\partial S_c(\tilde{x})}{\partial(\tilde{x}_i)}\Big|_{\tilde{x}=\bar{x}+\alpha(x-\bar{x})} d\alpha$$

It is recommended to pick a baseline $\bar{x}$ that leads to near-zero scores or the scores which represents absence of information. [9]
The *Integrated Gradients* satisfies the **Completeness** which means the attributions from each feature sum up to the difference of the output.

[9] For IR models whose inputs are mostly embeddings and well standardized numerical features, **zero vector** is a good baseline.

DeepLIFT

DeepLIFT computes the backward pass on the network. It assigns the relevance from the output neurons to the input layers recursively. Starting from the L layer:

$$r_i^L(x) = \begin{cases} S_i(x) - S_i(\bar{x}), & \text{if unit } i \text{ is target neuron} \\ 0, & \text{otherwise} \end{cases}$$

, and for other layer:

$$r_i^{(l)} = \sum_i \frac{z_{ji} - \bar{z}_{ji}}{\sum_i' z_{ji} - \sum_i' \bar{z}_{ji}} r_j^{(l+1)}$$

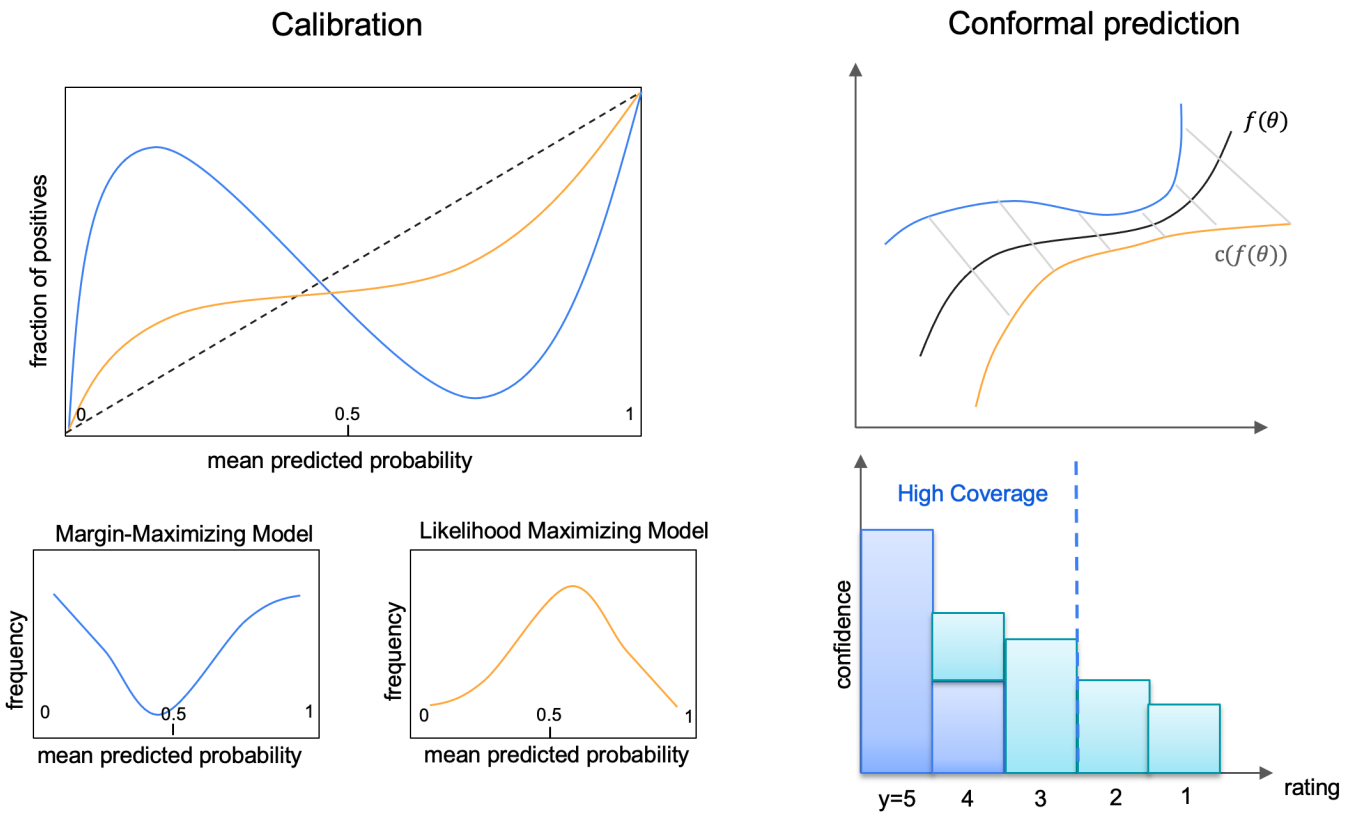
$\bar{z}_{ji} = w_{ji}^{(l+1,l)} \bar{x}_i^{(l)}$ is weighted activation of the neuron i onto neuron j given baseline. DeepLIFT proposes a few rules to determine the weights.

Conformal Prediction and Calibration

- Recall from previous slides that over-parameterized models all **maximizes margins** – they will push the score (logits) to extreme (0 and 1 in binary classification)!
- Their scores do not have **probabilistic interpretation** – they only represent margins, not the confidence or uncertainty in their prediction.
- Therefore, recommending e_1 over e_2 based on score is not appropriate: we have no idea how confident this outcome is.

We need to either **calibrate** the output scores into appropriate probability distribution (for binary task), or derive a **conformal prediction** that contains the true outcome with high confidence (for ranking).

Conformal Prediction and Calibration

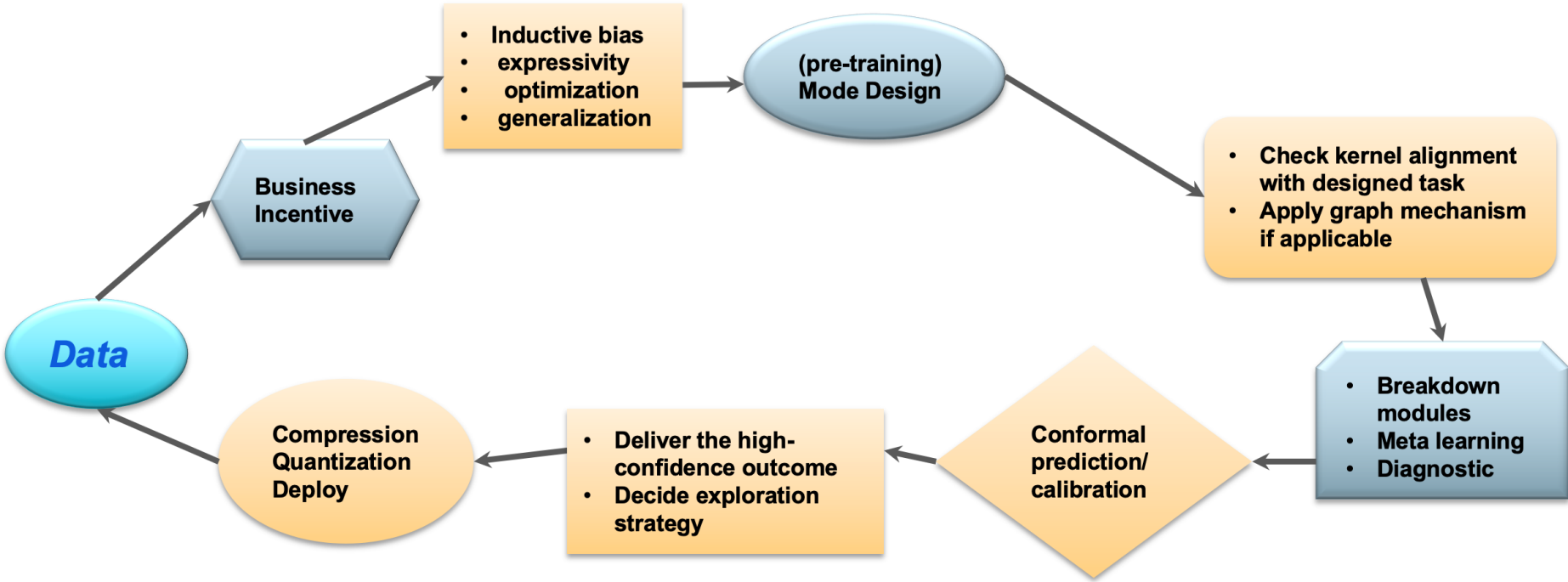


As you may imagine, the key implementation for both tasks is to find the quantile functions, and use their statistical properties to rearrange and rescale the outputs[10].

[10] Paying attention to the scores of your recommendation model coming soon

Domain topic: learning system design

Knowing how things work → making things happen!



Not your typical engineering cycle.
But can help you draft PPT that looks fancy :)

Domain topic: Learning to learn

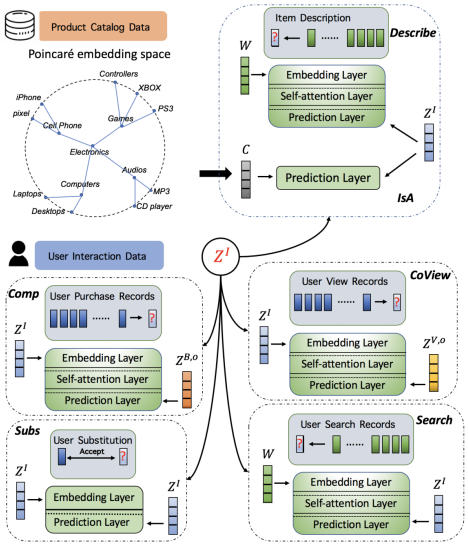
The application of LTL (learning to learn) is ubiquitous in industrial IR systems.
Common approaches used in LTL tasks are:

1. Model-based such as Memory-Augmented network [11]
2. Metric-based such as Siamese network [12]
3. Optimization-based such as Model-Agnostic Meta-Learning [13]

[11] Meta-Learning with Memory-Augmented Neural Networks, Adam et al., ICML'16
[12] Siamese Neural Networks for One-shot Image Recognition, Gregory et al., ICML'15
[13] Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, Chelsea et al., ICML'17

Shared embeddings

The straightforward application of shared representations between previous and new tasks is also very useful for IR system. Although it has practical success, little do we know why it works from theoretical point of view.



The application of shared product knowledge embeddings for e-Commerce IR system [14]

[14] Product Knowledge Graph Embedding for E-commerce, Da et al., WSDM'20

Provable LTL with linear representation

Good news is the effectiveness can be answered under the linear model setting.
Assume we have unobserved feature matrix $B = (b_1, ..., b_r) \in \mathbb{R}^{d \times r}$ with orthonormal columns. Define he statistical model for pairs (x_i, y_i)

$$y_i = x_i^\top B\alpha_{t(i)} + \epsilon_i; \quad \beta_{t(i)} = B\alpha_{t(i)}$$

,
$\alpha_{t(i)}$ is the unobserved task parameters, $_i \in \mathbb{R}^d$ is the random covariate. It is proved that [15].

$$\mathbb{E}_{x_*}[< x_*, \hat{B}\hat{\alpha} - B\alpha_{t+1} >^2] \leq \tilde{O}(\frac{dr^2}{n_1} + \frac{r}{n_2})$$

,
$\beta_{t+1} = B\alpha_{t+1}$ is the parameters of the new task, and is the best estimate learned from all other tasks. n_1 is the number of samples used for feature learning and n_2 is the number of sample used for the unseen tasks.

[15] Provable Meta-Learning of Linear Representations, Nilesh et al., ICML'2021