複数のモデルを扱う場合のあれこれ :わりと実務的な話

Masahiro Morinaga

The University of Tokyo
(ICEPP ,Beyond AI)

ML in LHC

LHCでの機械学習

- 古の時代よりLHCでは機械学習を使ってきた, とくに粒子同定や事象選別でそのパワーを発揮していた
- なぜ機械学習なのか?? → 答えを導くための理論/方法が不明 or 複雑な場合に有効
  - 例えば, トラッキングの様に確立された手法がある場合はあまり旨味がない
  - しかし, 処理時間などでどうにもならん場合は有効である可能性はある(精度はそこそこで処理時間がセーブできるモデルとか)
- 粒子同定などの場合は, 理論的な計算が複雑で検出器からの情報が多次元であるので人間に手でできることはたかが知れている
  - 3-4変数くらいなら頑張れるけど, 10変数越えてくると相関とかわからんしめんどい → MLでブラックボックスでもいいのでうまいことやってほしい

深層学習の台頭

- 機械学習, とくに深層学習は魔法のツールではなく, うまく使いこなすには結構な労力が必要 →
  - あくまでもMLは研究を推し進めるための補助的なものとして考えるべきだと個人的には考えている
- 深層学習の発展/開発で, 複数のモデルをつないだり, 複雑なことが可能になった → いままでいろんな理由で分けて学習していたものを一纏めにできるように.
- そんなときにきになることもいろいろある
  - 損失関数のつなぎ方は?
  - どのモデルがベストなの??
  - ハイパーパラメータは??
- など, 実際にモデルを組んで試すときに決めるべきことがたくさんある → 基本的には試して最も良いものを採用でよい
  - しかし, 時間や計算機資源があまりない場合はハイパラ全探索とかできない → かしこいアルゴリズムを使って効率よく研究する

実際のデータに適用する場合に気になること

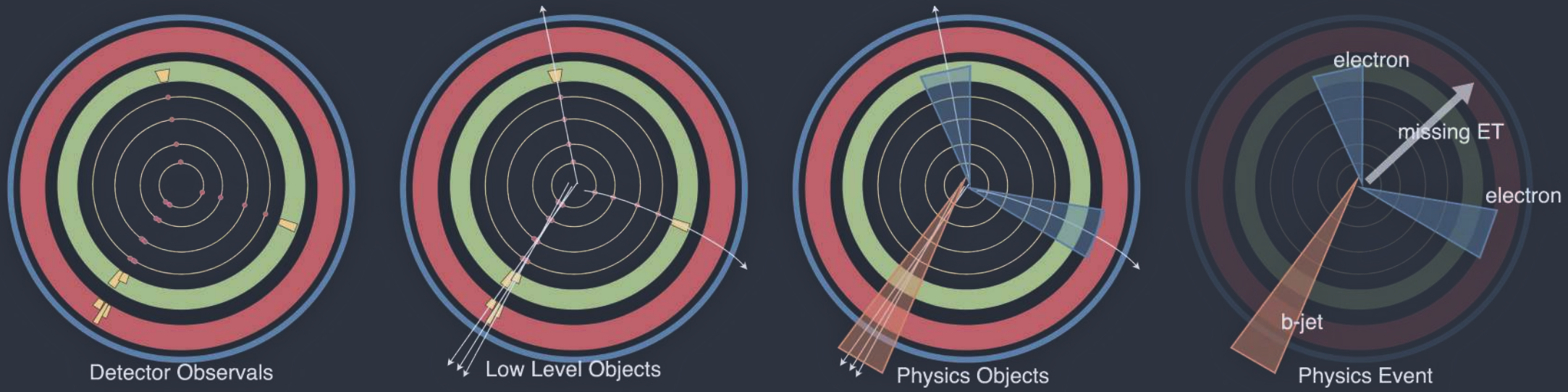
- 実際に機械学習を使ってデータ解析を行って, 例えば発見をクレイムする場合に気になったこと($H \to \tau\tau$発見時に苦労したこと)
  - シミュレーションとデータの違いをどうするの?
  - 系統誤差どうするの?
  - ネットワークの構成は?
  - ハイパーパラメータは??
  - 背景事象と信号事象のウェイトは??
  - サンプルどうやって分けるの??
  - 得られたスコアをどう使うの??
- ぱっと思い出せただけで色々と気になることがある

今回の話

- 今回は機械学習の専門家ではない物理屋さんが実際にツールとして使う場合に有益かなと思うことをできる限り紹介します
  - 細かいネットワークの話はあまりしないかも, どちらかといえばネットワーク同士をつないだり, 最適化に関するアーキテクチャの話です

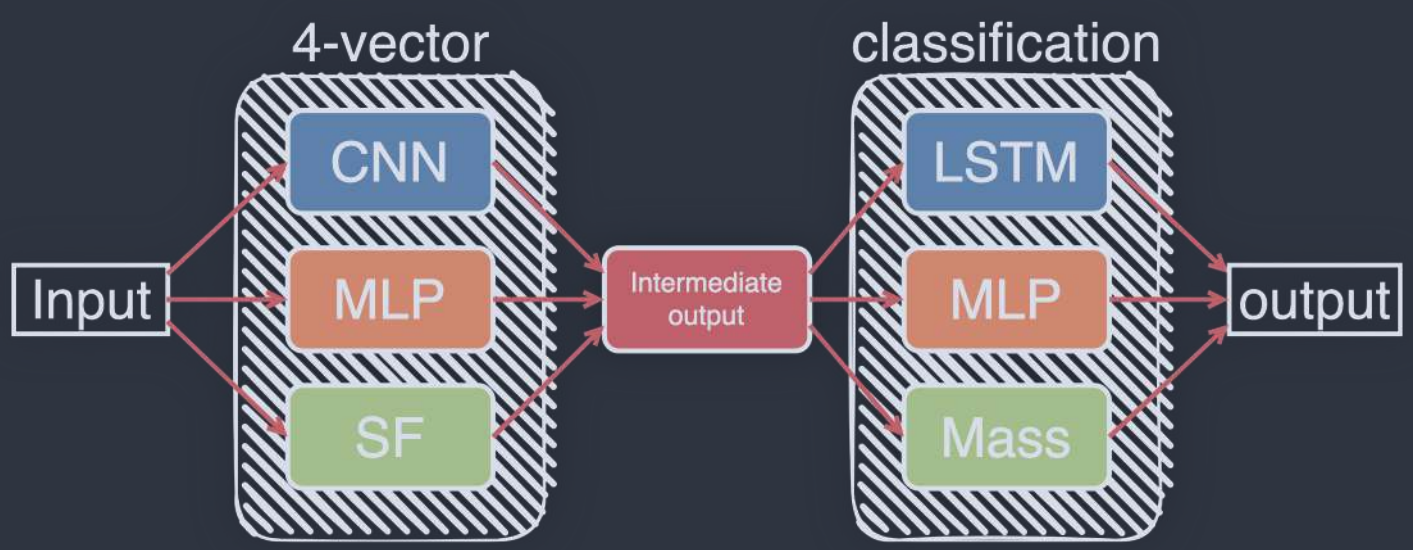
Typical Workflow to Get Results at LHC

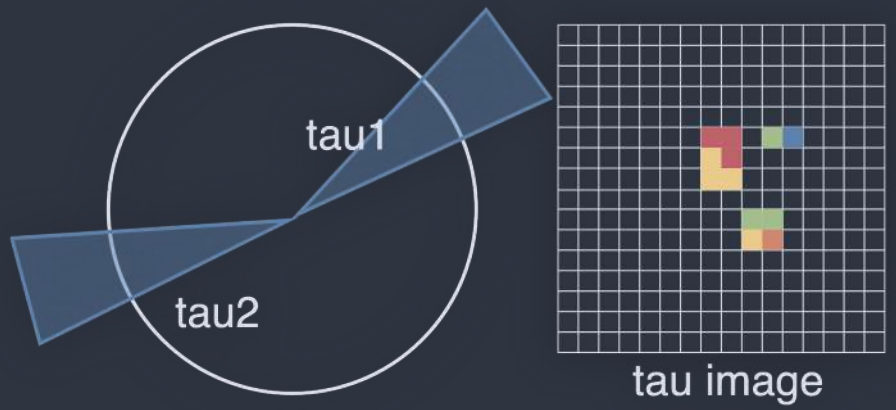
典型的なワークフロー



- 検出器の情報 → トラック/クラスター → 物理オブジェクト → イベント → セレクション or 分類
  - 流れは, 検出器の基本的な情報から段階的に物理的な情報に変換していく → 機械学習で代用可能
- トラッキングやクラスタリングはすでに精度の良いアルゴリズムで再構成可能 → 無理に機械学習に焼き直す必要はない??
- 大事なのは物理オブジェクトからイベント全体の再構成へといく流れで, 現状のLHCでは(いろいろな事情で)それぞれ個別に再構成を行う
  - まず物理オブジェクト単位での機械学習が走っていて, 粒子同定や運動量再構成されたオブジェクトを**手で**定義してイベントを再構成する
- 興味のある事象に電子が2つついるようなものだったとすると, electronひとつずつ杓子定規にID(粒子同定)するのでなく, 2つのelectronの情報を使ったほうが(背景事象との兼ね合いもあるが)典型的には感度が上がる
- これを行うには, 物理オブジェクトとイベントの再構成をつなげて最適化するようなアルゴリズム(機械学習)を設計すればよい.

Model Connection Example





モデルをつないで学習してみる

- 今回はわかり易さ等を考慮して, $H \to \tau\tau$と$Z \to \tau\tau$をそれぞれ信号,背景事象として考えてみる
- サンプルは適当にDelphesで作って, インプットとしてタウジェットのイメージ(channel=3, トラック, EMカロリメータ, ハドロンカロリメータでのアクティビティ)とジェットの4-memontumとした.
- イベント中には理想的にタウ以外のジェットはいないと思って, 以下の2ステップのモデルを考える
  - Task1 : タウジェット4-vector, 運動量のキャリブレーション to true tau(produced)
  - Task2 : Z/H の分離, Task1の出力(4-momentum)を入力として, 最終的なZ or Hを出力
- いろいろなモデルで各ステップを最適化したい → 今回は3つずつ異なるモデルを試した.

損失関数Lossについて

- 今回のように, 中間教師信号(今回はtau 4-momentum)がある場合, Lossは $\mathcal{L}_{\text{total}} = w_1\mathcal{L}_1 + w_2\mathcal{L}_2$ の様になり重み$w_{1,2}$のハイパーパラメータが発生
- 一般的に, multi-objectiveな最適化でもLossはこの様に書かれるため, Lossの重みを変えて結果を見る必要あり.

Neural Architecture Search

NAS(Neural Architecture Search)

- NASとは, モデルアーキテクチャやハイパラ等をニューラルネットワークを使って最適化しようという考え方, AutoMLのひとつ
- 典型的なものはモデルを決定するコントローラーがあり, 最適化したいモデルアーキテクチャをアップデートしながら学習を進める.
- 代表的なものとしては, ENASなどがある
    - NASの学習自体が時間がかかる, ENASですらかなり掛かる

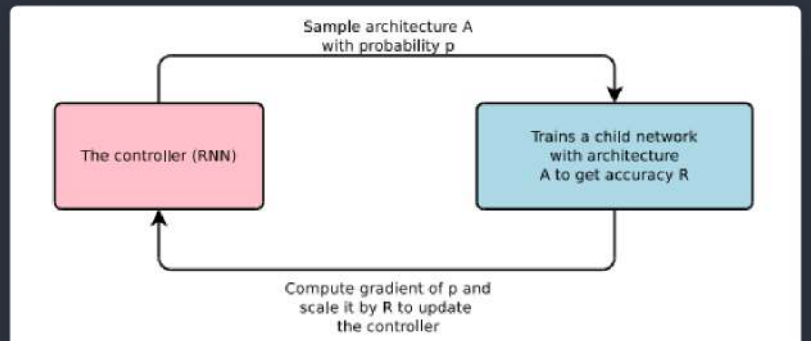
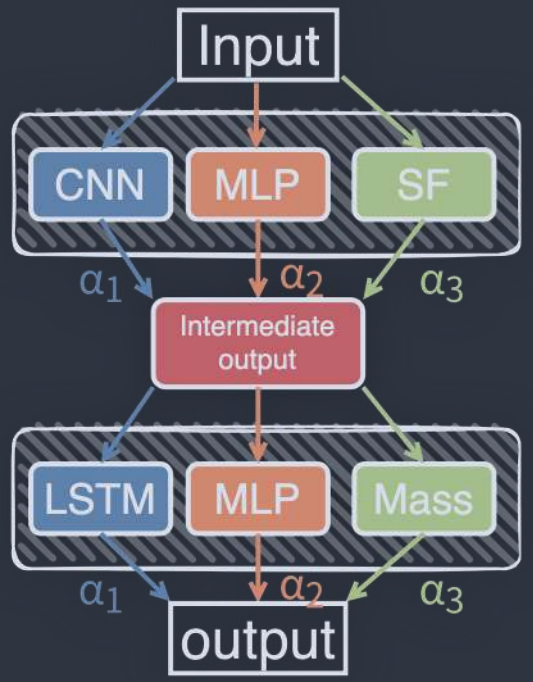


Figure 1: An overview of Neural Architecture Search.

シンプルにNASをつかう

- 前ページのモデル選択にNASを使うことを考える, 試したアルゴリズムは4つ
- 全探索 : 3x3を全通り試す
- DARTS : 探索空間をDAQとして, 探索対象を連続的な空間に緩和し微分可能だと考える
- SPOS-NAS : 選択肢をランダムに選んで探索と学習を進める
- ASNG-NAS : 自然勾配を使って効率的に探索と学習を行う

NAS : Algorithm



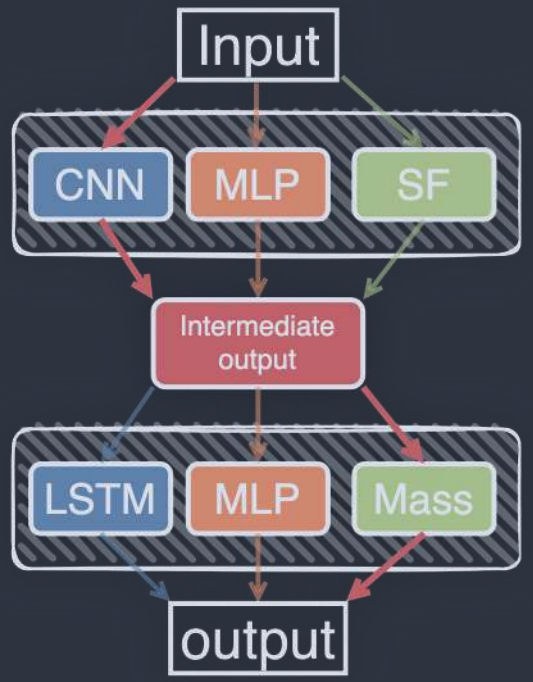
Differentiable Architecture Search(DARTS) :

- 重み α_i をモデルごとに考える, 出力は

$$y_t = \sum_{i \in \text{model}} \text{softmax}(\alpha_i) \cdot y_{t,i}$$

- とモデルごとの出力を重み付きで足し上げることで微分可能な形で学習を進める
  - トレーニングサンプルでモデル自体(CNNとかMLPとか)の重みを勾配更新
  - バリデーションサンプルで α_i の更新を行う
- 最終的なモデル選択として, 最も大きな α_i を持つモデルを選択する

NAS : Algorithm



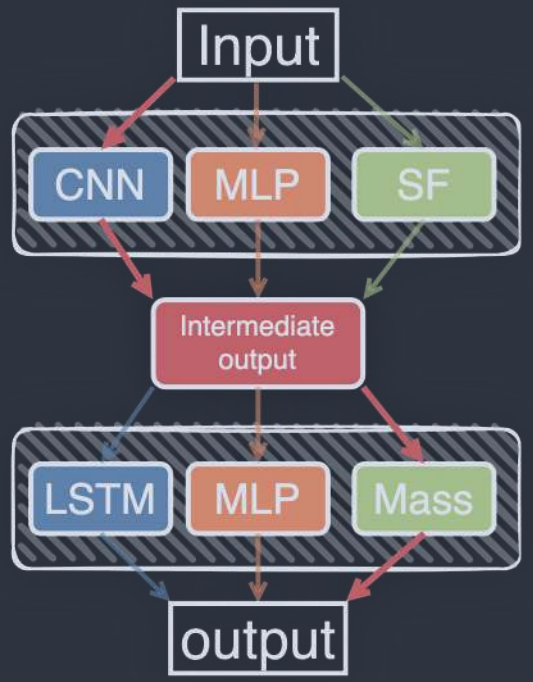
Single Path One-Shot NAS(SPOS-NAS):

- 一回の試行で一つの**PATH**, 通り道を選んで勾配を更新する

$$y_t = y_{t,i}(i \sim \mathrm{Uniform}[m_1, m_2, m_3])$$

- バッチごとに一つのモデルをランダムに選ぶ
- 最終的に, Evolutionaly algorithmを用いてベストなモデルを選択する
  - 今回は3x3=9通りと少ないので全探索してベストモデルを選択

NAS : Algorithm



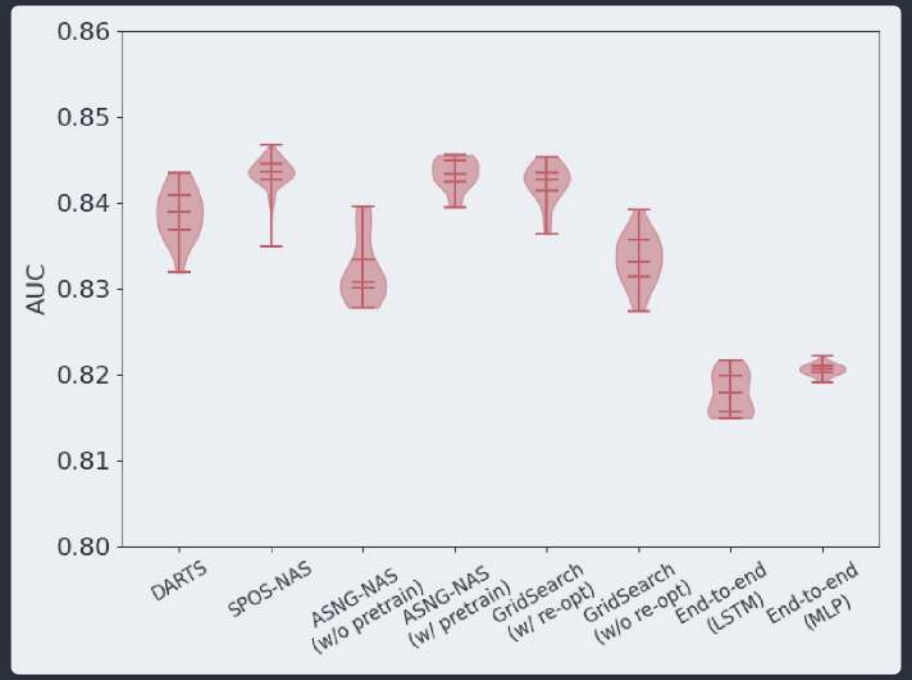
Adaptive Stochastic Natural Gradient NAS(ASNG-NAS):

- 確率緩和したカテゴリカル分布からサンプリングし, 自然勾配を使って勾配を更新する

$$y_t = y_{t,i}(i \sim \mathrm{Categorical}[m_1, m_2, m_3])$$

- カテゴリカル分布を自然勾配を使って更新, ただし自然勾配の計算は一般に困難なためMCサンプリングで代用
- 1バッチにつき, 2回のサンプリングで更新が可能 → 勾配の学習率を工夫することで担保
  - Training : 2つのモデルを選択しモデルの勾配を更新
  - Validation : 2つのモデルを選択肢, カテゴリカル分布を更新
- 最終的に, カテゴリカル分布の最も高い確率のモデルをベストモデルとして選択する

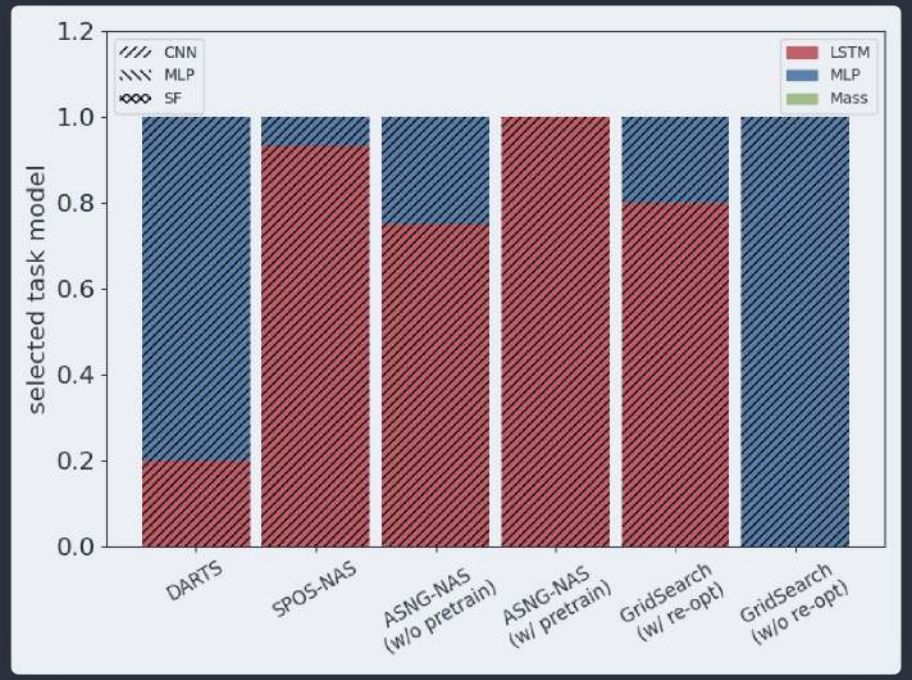
NAS : Results



結果 : 様々なアルゴリズムでの最終的なAUCの比較

- 複数のseedでの結果をバイオリンプロットしたもの
- Grid Search : 全探索, ベストモデルを選択した後に再度トレーニング
- End-to-end : 中間Lossを経ずに入力から最終的な出力まで一つのモデルで行った場合
  - パラメータ数が少ない可能性がありあまりフェアではない可能性あり
- SPOS-NAS, ASNG-NASはGridSearchと同程度のパフォーマンス
- DRATSはアルゴリズムの性質上少し弱い
- 事前学習（pretrain）は, タスクにtrue情報を入力しトレーニングしたもの
  - pretrainありで結構パフォーマンスが変わることがわかる

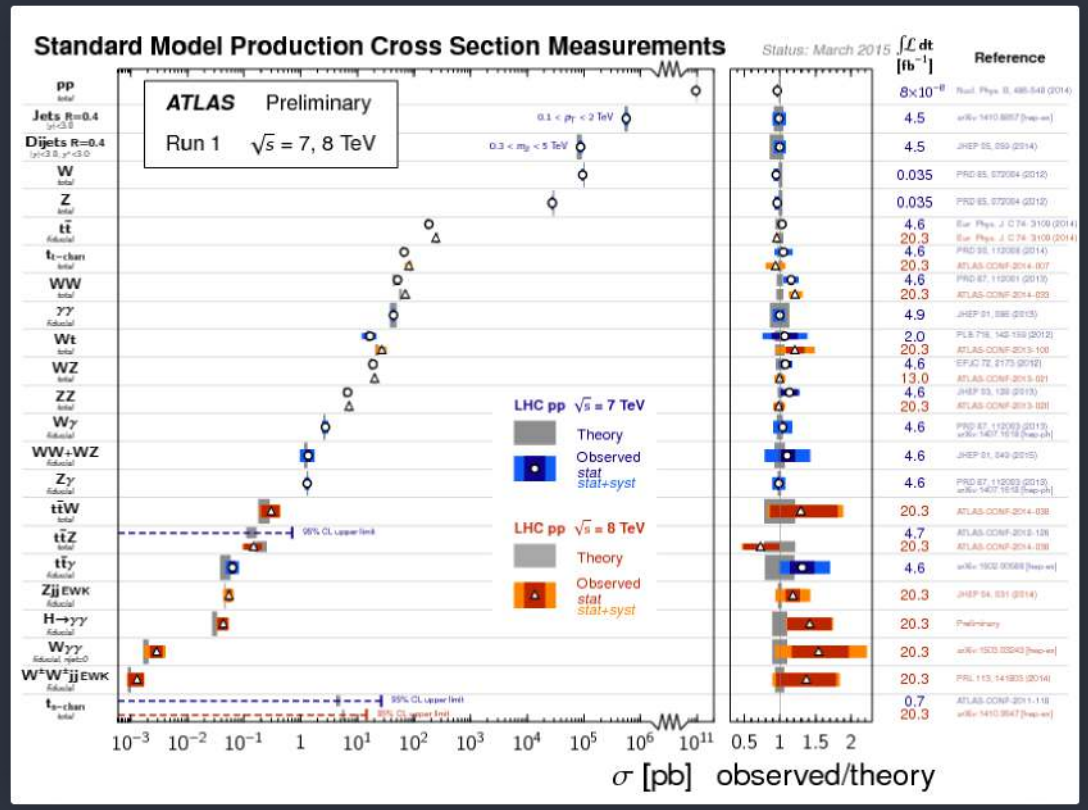
NAS : Results



どのモデルが選ばれているか

- 左レジェンドがTask1, 右レジェンドがTask2
- 基本的には, CNNとLSTMが最も良いパフォーマンス
- SPOS-NAS, ASNG-NASは似たような結果となるが, DARTSはアルゴリズムの性質上異なるモデルを選ぶ傾向がある
- 今回は大きなモデル単位での選択だったが, もっと細かい選択肢でもおｋ
    - 例えば, CNNの畳み込み層のchannel数とかkernel sizeなど, activation層の種類(ReLU or GELU or Mishとか)
    - 今回は見せませんでしたが, 動くことは把握済み

Training by Using Physics Sample



物理過程のトレーニングについて

- われわれの探索している信号はだいたい断面積が低くてサンプル生成時間がかかる
- もしくは, データを使って直接トレーニングする場合, サンプル数は不均衡になる
- 結果, サンプル(ラベル or クラス)の間でデータ数がだいたい揃わない → 合わせるようにMCサンプルをたくさん作る or 工夫する
  - 少ない方に合わせる → せっかくたくさん作ったのにもったいない
  - 多い方に合わせる → 少ない方を水増し, 過学習が怖い
- なんとかLossとかをいじって対応できないか??ということを考える

Exaple using Tau or Jet classification

- 先程と同様に, Delphesで作ったtau/jetサンプルを使って, いくつか実験をしてみる
- 入力はjet image(#of channel = 3, それぞれtrack, EMcalo, Hcaloでのactivity)
- 擬似的にデータをjet 90%, tau 10%と不均衡にして学習を行う

Class-Balanced Loss on Effective Number of Samples

CBLoss

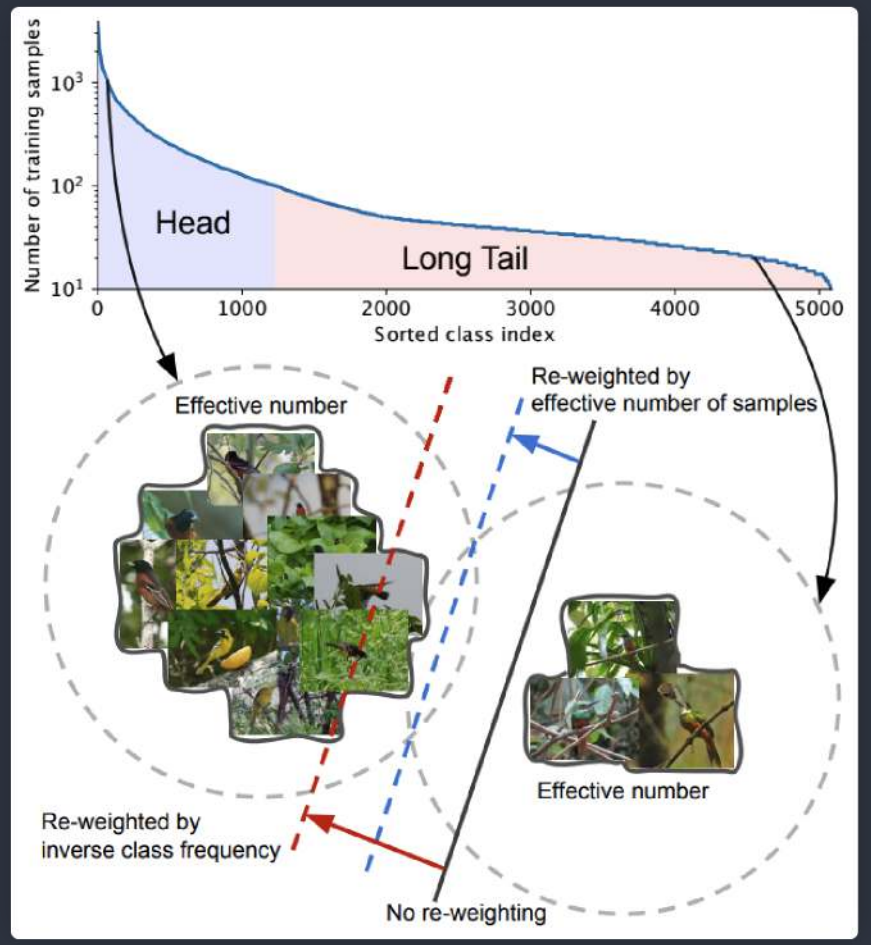
- 学習サンプル数に偏りがある場合, lossをサンプル数でweightをかけてトレーニングを行うとminor(サンプル数が少ない)サンプルの性能が上がる.
- 通常のSoftmax Cross Entropy Lossは, 下記のようにi:batch index, m_iはiの正解ラベル, p_{m_i}はラベルmに対するSoftmax output
- 単純にサンプル数で補正してみると,

$$ICE = \sum_{i=1}^{B} \frac{1}{n_{m_i}} \log(p_{m_i})$$

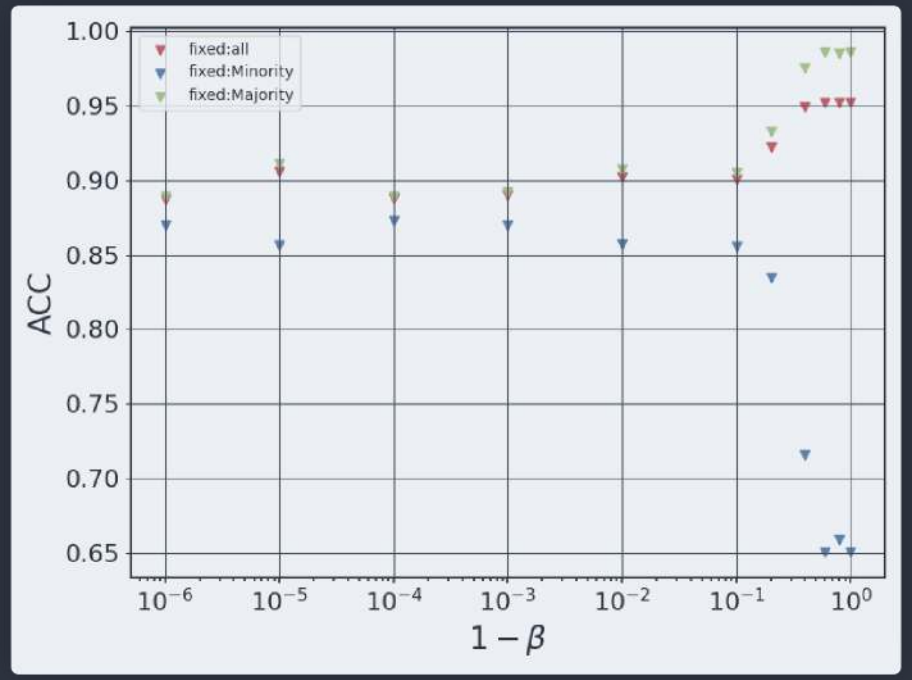
- Class Balanced Lossはデータの本質的なサンプル数を考慮したICEで,

$$CBCE = \sum_{i=1}^{B} \frac{1}{E_{m_i}} \log(p_{m_i}), \quad E_{m_i} = \frac{1 - \beta^{n_m}}{1 - \beta}$$

- とβ(ハイパラ)で表されて, $\beta = 0$のときCE, $\beta = 1$のときICEと一致する.
  - $\beta = 0$の場合は不均衡を考慮せず, $\beta = 1$のときはminor classを重点的に学習する.



Class-Balanced Loss on Effective Number of Samples



Tau or Jet Classification Result

- $1-\beta$が1のとき, なにも考慮しない結果, $1-\beta$が小さいほどMinorサンプルを重視する
- アーキテクチャは適当なResNETとDense層を重ねたもの
- $1-\beta$が1に近いとき, やはりMajorサンプルの精度が上がって, Miorサンプルの精度は下がっている
- $1-\beta$を小さくしていくと, Minorサンプルの精度が上がりMajorは下がっている
  - → Majorに引っ張られて全体の精度は下がる
- 単純にMinorが20%ゲインするのは嬉しい
- ただしβ(ハイパラ)でスキャンする必要あり

YOTO : You Only Train Once

ハイパーパラメータ問題

- 普通の人なら, なんかハイパーパラメータばっかり出てきてめんどうだなあ、と思う
- ハイパラを固定せずにどうにかして, 学習後に与えてやれないかと考える...
    - GPU資源と時間がたくさんあるならこんなことせずに脳筋プレイをかませる

Sampling Loss Weight

- YOTO : You Only Traning Once: Loss-Conditional Training of Deep Networks
- Loss weightをなんらかの分布からサンプリングして, それをモデルに反映させてトレーニング, 評価する場合は見たいLoss weightを自分で入力する
- 数式での説明 : トレーニングデータ$\boldsymbol{x}, \boldsymbol{y} \sim P_{x,y}$ with $\boldsymbol{x} \in \mathbb{X}, \boldsymbol{y} \in \mathbb{Y}$, 損失関数$\mathcal{L}(\cdot, \cdot)\ \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}$, モデル$F : \mathbb{X} \to \mathbb{Y}$がパラメタ$\boldsymbol{\theta}$をもっている
- ここでは$\hat{y} = F(x, \theta)$で, $\mathcal{L}(y, \hat{y})$を最小化することを考える, $\boldsymbol{\lambda}$がloss weightを記述するパラメタとして

$$\boldsymbol{\theta}^*_{\boldsymbol{\lambda}} = \arg\min \mathbb{E}_{x,y \sim P_{x,y}} \mathcal{L}(y, F(x, \theta), \lambda)$$

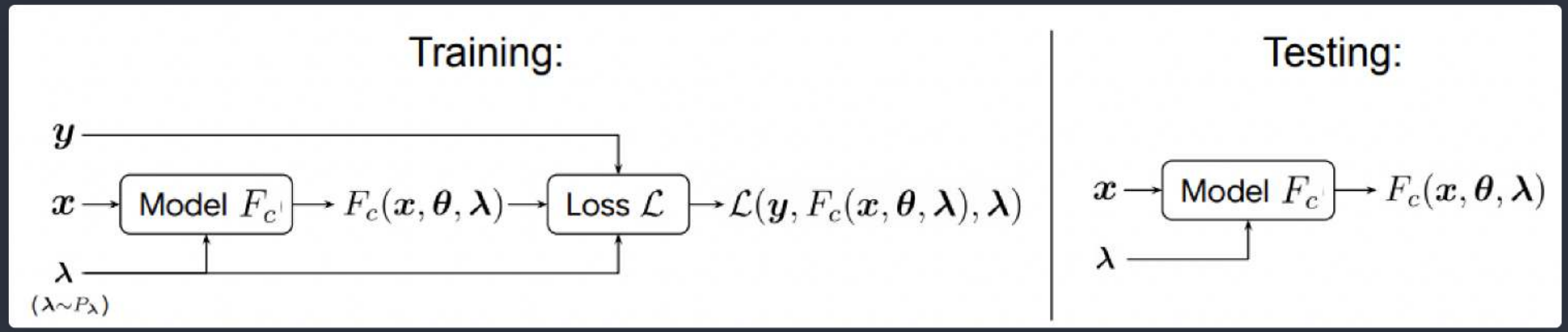
- がいま考えるべき問題となるが, ここでは$\boldsymbol{\lambda}$を固定せずにP_λからサンプリングすることを考える.

$$\boldsymbol{\theta}^*_{\boldsymbol{\lambda}} = \arg\min \mathbb{E}_{\lambda \sim P_\lambda} \mathbb{E}_{x,y \sim P_{x,y}} \mathcal{L}(y, F_c(x, \theta, \lambda), \lambda)$$

- で, $\boldsymbol{\lambda}$(ハイパラ)を使ってモデルを作って, conditionalなネットワークでトレーニングを行うことで, $\boldsymbol{\lambda}$をも考慮したネットワークを作る.

YOTO : You Only Train Once

Sampling Loss Weight



- 実際のアーキテクチャはこんな感じになっている, トレーニング時はλを任意の分布からサンプリングする, 推論時は好きな値で評価できる
- 要するに, 一度のトレーニングでいろんなloss weightでトレーニングした結果を再現できうるというのがこの論文の主張.
- では, 実際どういうふうに実装すればよいかというと, Feature-wise Linear Modulation(FiLM)を使う
- FiLMレイヤーをネットワークに組み込んでトレーニングを行う.
  - 与えたλからγ, βの2つの変数を出し, それを使って特徴量マップをアフィン変換する

FiLM : Feature-wise Linear Modulation

FiLM Layer

- FiLMへのインプット$\boldsymbol{\lambda}$から,

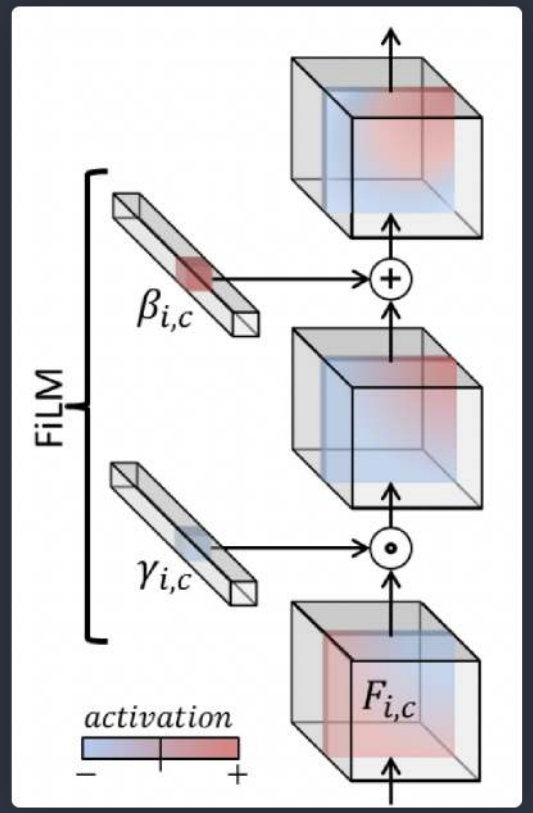
$$\gamma_{i,c} = f_c(\lambda_i)$$

$$\beta_{i,c} = h_c(\lambda_i)$$

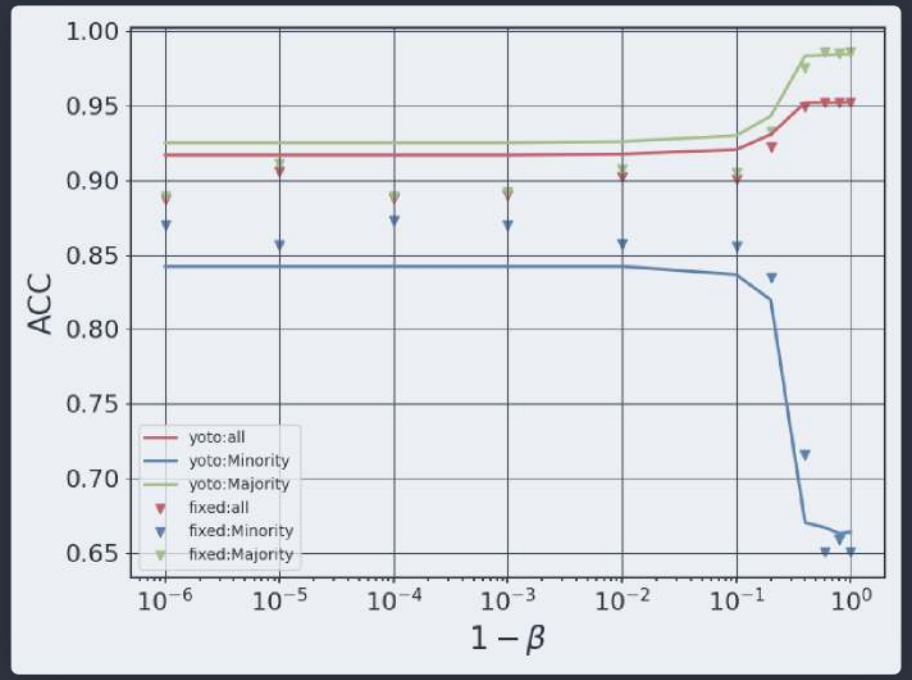
- とγ, βに変換して, ここで添字i, cはi-th入力のc-th特徴量という意味で, FiLM layerは

$$\mathrm{FiLM}(F_{i,c}|\gamma_{i,c}, \beta_{i,c}) = \gamma_{i,c} \odot F_{i,c} + \beta_{i,c}$$

- とアフィン変換を行う(⊙はHadamard product)
- f, hは何でもよくて単にNNで求める.
- 右のCNNの場合だと,
  - 特徴量マップ$F_{i,c}$について, γでスケールをして, βだけ移動させる変換
- このFiLM layerを組み込んだブロックを重ねたりする.



Yoto Results



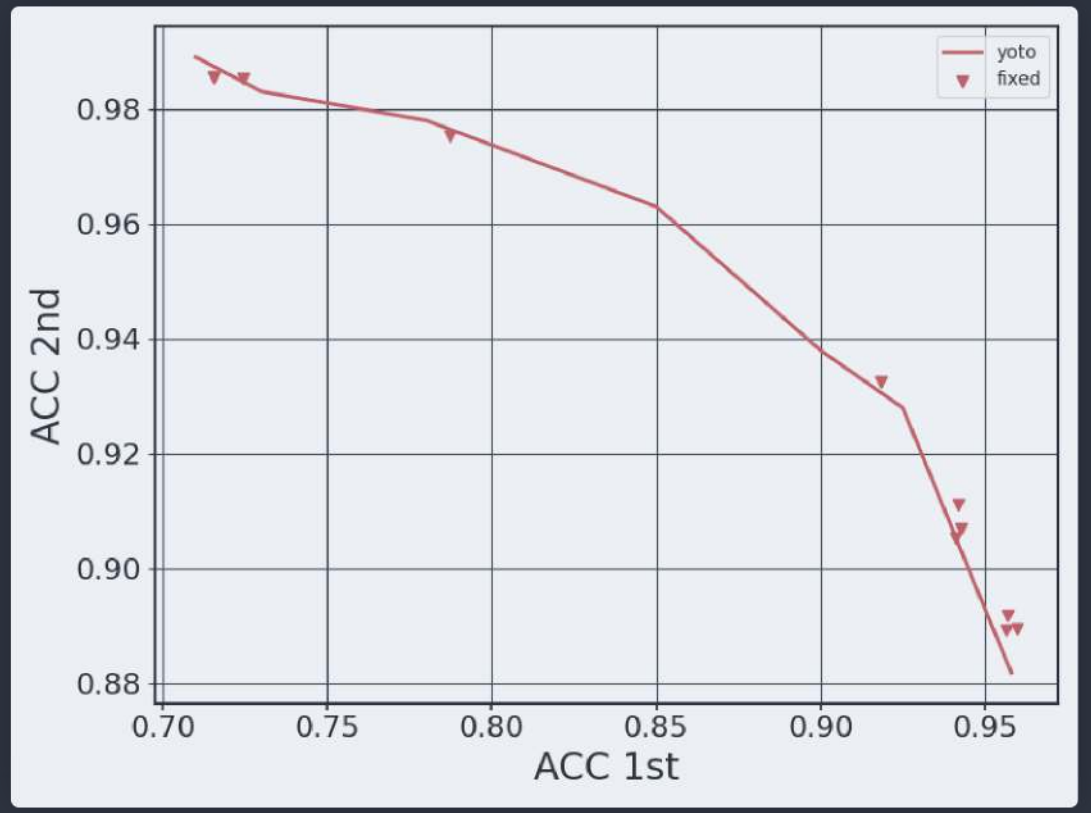
Yoto を使った結果

- CBLossのβ(ややこしくてすんません)をYotoの入力としてハイパラも含めて学習
- ほとんど遜色のない学習が達成できる
  - 実線がYotoで1回だけ学習し, 様々な値で評価した結果
- Fixedの場合がパフォーマンス良い場合がある
  - λのサンプリング分布の良し悪しや, (論文中の実験では)モデルが浅すぎる可能性ありと指摘されている
- 逆によくなる場合もある, もう少しチューニングをすれば良くなりそう

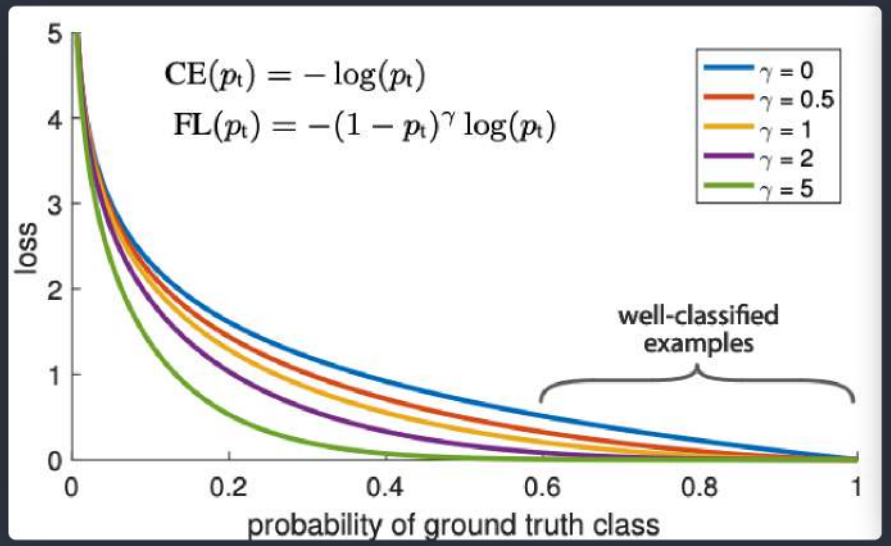
Multi Loss Function

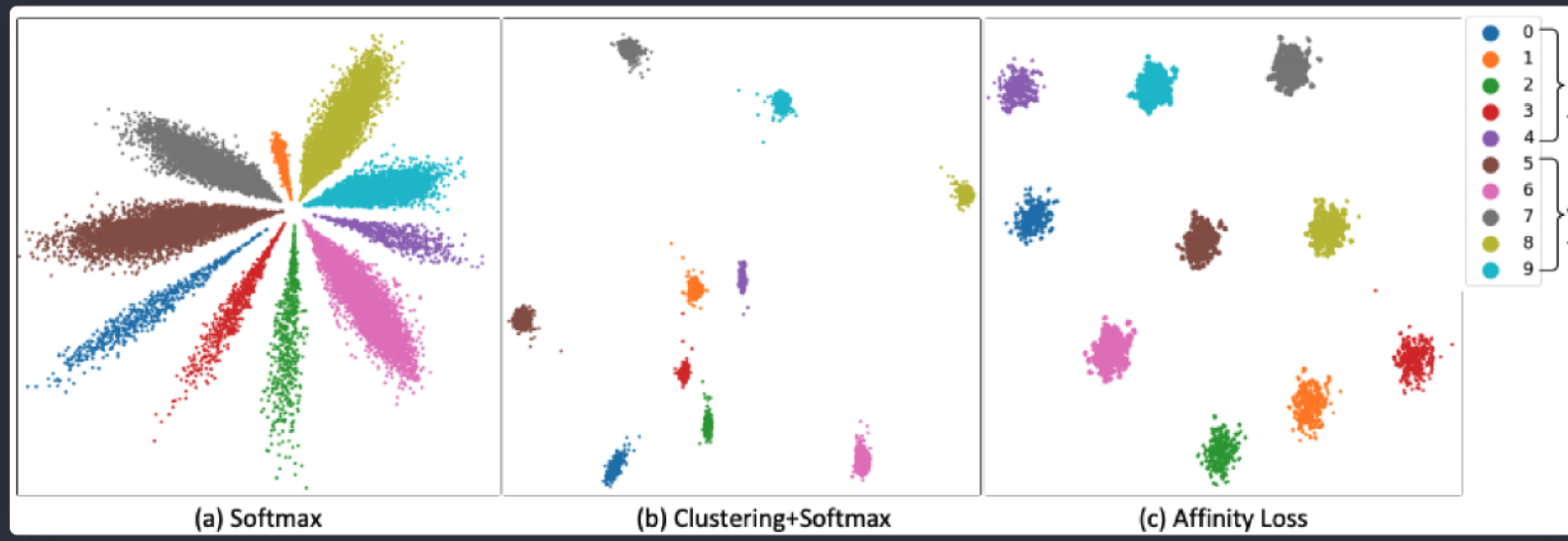
Multi Taskに対するLoss

- 例えば2つのloss(l_1, l_2)がある場合, $L = w_1 l_1 + w_2 l_2$ みたいにすると, w_1, w_2はどうやって設定しますか?
- なんも考えない場合 $\rightarrow w_1 = w_2 = 0.5$とかでトレーニングしてみる
  - その後$w_1 = [0.0, 0.1, 0.2, ..., 1.0] (w_2 = 1 - w_1)$とかってやってスキャンしてみるとか
- もうちょっと頑張りたいなら, 色々とアルゴリズムが存在して, 2つの目的がパレート最適な場合はMGDAの様なアルゴリズムが便利
  - 上のようなアルゴリズムは, loss weight(w_1, w_2)を動かしつつパレート最適な点に持っていく
- 結局, 後でweightを動かしたい場合はハイパラになる
- Yotoで行ってみた : さっきのTau or Jet 分類に4-momentum calibrationを加えたもの(shared layerを介してつなげた)
  - loss weightをYotoでサンプリングした結果, 若干怪しいところもあるが基本的にはworkしている



Data Argumentation





その他の手法

- 他にもたくさんデータ自体の構造に注目した考え方が提案されている
  - Focal Loss : シグナルっぽいバックグラウンド(逆も)に注目したLoss, $L = (1-p_t)^{\gamma}\log(p_t)$として重点的に学習する($\gamma$はハイパラ)
  - Affinity Loss : 特徴量空間でのクラスタリング＋マージン最大化を行うことでSoftmax出力を強化したもの
- これらの手法はData argumentationの効果をブーストするようなもので, データ自体の性質をより学習しやすい形に持っていくのが大事
  - 不均衡データのあつかいもここに含まれるし,
- データの本質的な特徴量をどう扱うかを考察することが極めて重要

Conclusion

まとめ

- 物理解析で機械学習を使う場合に気になる点をあげた
- いくつかの(小さな)タスクをつなげて学習を行う手法が魅力的
- NAS を使ってタスクごとにベストなモデルを選択することができた
- モデルアーキテクチャのチューニングも大切だけど, 損失関数に少し手を加えるだけで感度が上がることを示した
- Conditionなハイパーパラメータのチューニング問題を解決できる手法(FiLMレイヤーとYOTO)を試した
- Yotoを使うとハイパラを後から決定できるのでいろいろとラク
- モデルアーキテクチャの最適化もいいけど, データ自体の考察を忘れてはいけない（戒め）
- multimlというパッケージで今日紹介したものはほとんど利用可能です, ぜひforkしてください.

Backup

2nd Topic ResNet Model Architecture

```yaml
cnn1:
  - key: ConvResBlock1
    args:
      in_plane: 3
      out_plane: 256
      activation: {key: GELU, args: {}}
      conv_args:
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
  - key: ConvResBlock2
    args:
      in_plane: 256
      out_plane: 256
      activation: {key: GELU, args: {}}
      conv_args:
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
  - key: ConvResBlock3
    args:
      in_plane: 256
      out_plane: 256
      activation: {key: GELU, args: {}}
      conv_args:
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
  - key: ConvResBlock4
    args:
      in_plane: 256
      out_plane: 256
      activation: {key: GELU, args: {}}
      conv_args:
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
  - key: ConvResBlock5
    args:
      in_plane: 256
      out_plane: 256
      activation: {key: GELU, args: {}}
      conv_args:
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
        - {kernel_size: 3, stride: 1, padding: 1, bias: False }
  - key: activation
    args: {name: AdaptiveAvgPool2d, output_size: 1}
mlp1:
  - key: FiLMed_MLPBlock1
    args: {n_input: 256, n_output: 256, activation: {key: GELU, args: {}}, dropout_rate : 0.1 }
  - key: FiLMed_MLPBlock2
    args: {n_input: 256, n_output: 256, activation: {key: GELU, args: {}}, dropout_rate : 0.1 }
  - key: Linear1
    args: {in_features: 256, out_features: 256, bias: False}
  - key: activation
    args: {name: GELU}
```

2nd Topic MLP Model Architecture

```yaml
mlp:
  - key: ResBlock1
    args :
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
  - key: ResBlock2
    args :
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
  - key: ResBlock3
    args :
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
      - {n_input: 256, n_output: 256, activation: {key: GELU, args:{}}, dropout_rate : 0.1}
  - key: FiLMed_MLPBlock1
    args: {n_input: 256, n_output: 256, activation: {key: GELU, args: {} }, dropout_rate : 0.1 }
  - key: LinearLast
    args: {in_features: 256, out_features: 2, bias: True}
```