International Summer School on Deep Learning

July 1st-5th, 2019 in Gdansk, Poland

www.2019.dl-lab.eu

Deep Learning for Ordinal Regression (in PyTorch)

Sebastian Raschka

Assistant Professor of Statistics at

University of Wisconsin-Madison

http://pages.stat.wisc.edu/~sraschka/

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

BIOMEDICAL ENGINEERING DEPARTMENT

GDAŃSK UNIVERSITY OF TECHNOLOGY

Tutorial Overview

Intro: Different approaches for ordinal regression

Part 1: Warm-up
- Implementing a nominal classifier in PyTorch for age prediction

Part 2: Dataset
- Preparing a custom dataset for model training in PyTorch
- MNIST DataLoader example
- UTKFace image dataset for age prediction

Part 3: Nominal Classifier
- Implementing a convolutional neural network classifier for age prediction (VGG-16 with cross entropy loss trained on UTKFace)

Tutorial Overview

Part 4: Extended Binary Classifier
- Ordinal regression via extended binary classification for age prediction
- Implementing Niu et al.'s ordinal regression approach
- Implementing the CORAL (Consistent Rank Logits) approach

Part 5: Other Approaches for Ordinal Regression
- Ensemble methods
- Beckham et al.'s metric
- Threshold approaches
- Pair-wise (as opposed to point-wise) methods

Tutorial Code

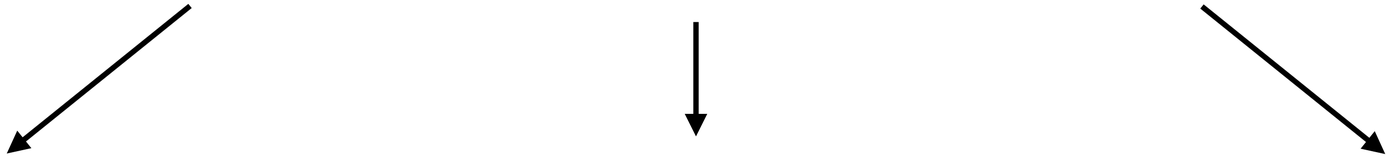
Tutorial code should be available to you via the Jupyter Hub server.
If not:

https://github.com/rasbt/DeepLearning-Gdansk2019-tutorial

```
git clone https://github.com/rasbt/DeepLearning-Gdansk2019-tutorial.git
```

Types of Labels in Supervised Learning Tasks

Color	Size	Price
green	M	10.1
red	L	13.5
blue	XXL	15.3



Nominal type
Task: classification

Ordinal type
Task: ordinal regression

Continuous
Task: metric regression

Supervised Learning: Ordinal Regression

Ordinal regression, also called *ordinal classification* or *ranking*
(although ranking is a bit different)

Order dependence like in metric regression,
but no metric distance

discrete values like in classification,
but order dependence

$$r_K \succ r_{K-1} \succ \ldots \succ r_1$$

E.g., movie ratings: *great* $\succ$ *good* $\succ$ *okay* $\succ$ *for genre fans* $\succ$ *bad*

Supervised Learning: Ordinal Regression

- Ranking: Correct order matters
  (0 loss if order is correct, e.g., rank a collection of movies by "goodness")

- Ordinal Regression: Correct label matters (as well)
  (E.g., age of a person in years; here, regard aging as a non-stationary process)

18 ≻ 29 ≻ 41

Excerpt from the UTKFace dataset
https://susanqq.github.io/UTKFace/

We will work with this dataset
in the hands-on tutorial this afternoon!

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

1. Basic/naive
2. Binary decomposition
3. Threshold approaches

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

1. Basic/naive

- Nominal classification (e.g., cross-entropy loss)
- Metric regression (+ conversion to integer/rounding)

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

2. Binary decomposition

- Convert target labels to binary labels
- Then, use single or multiple models on binary labels

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

3. Threshold approaches

- Learn function mapping labels to Real
- Categories are thresholds/intervals on the Real number line

Part 1: Warm-up

Implementing a nominal classifier in PyTorch for age prediction

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

1. Basic/naive

- Nominal classification (e.g., cross-entropy loss)
- Metric regression (+ conversion to integer/rounding)

Regression loss:

$$\text{MSE} = \frac{1}{n}\sum_{i}(a^{[i]} - y^{[i]})^2$$

last-layer activation/ predicted label

true label

Multi-class classification loss (cross-entropy):

$$\mathcal{H} = \sum_{i=1}^{n}\sum_{j=1}^{k} -y_j^{[i]} \log\left(a_j^{[i]}\right)$$

Class-Membership Probabilities via Softmax

$$a^{(i)} = P(y = t \mid z_t^{(i)}) = \sigma_{\text{softmax}}(z_t^{(i)}) = \frac{e^{z_t^{(i)}}}{\sum_{j=1}^{k} e^{z_j^{(i)}}}$$

$$t \in \{j...k\}$$

k is the number of class labels

(Basically, softmax is just an exponential function that normalizes the activations so that they sum up to 1)

Loss Functions for Logistic Regression & Multi-layer Neural Network Classifiers

$$\mathcal{L}_{\text{binary}} = -\sum_{i=1}^{n} \Big(y^{[i]} \log(a^{[i]}) + (1 - y^{[i]}) \log(1 - a^{[i]}) \Big)$$

Negative log-likelihood loss / binary Cross Entropy

$$\mathcal{L}_{\text{multiclass}} = \sum_{i=1}^{n} \sum_{j=1}^{k} -y_j^{[i]} \log \left(a_j^{[i]} \right)$$

(Multi-category) Cross Entropy for k different class labels

This one assumes one-hot encoded labels!

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

(4 training examples, 3 classes)

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

1 training example

$$\mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

(4 training examples, 3 classes)

$$\mathcal{L}_{\text{multiclass}} = \sum_{i=1}^{n} \sum_{j=1}^{k} -y_j^{[i]} \log\left(a_j^{[i]}\right)$$

$$\begin{aligned} \mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &= 0.969692... \end{aligned}$$

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

$$\begin{aligned} \mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &\quad + [(-0) \cdot \log(0.3104)] \\ &= 0.969692... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[2]} &= [(-0) \cdot \log(0.3072)] \\ &\quad + [(-1) \cdot \log(0.4147)] \\ &\quad + [(-0) \cdot \log(0.2780)] \\ &= 0.880200... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[3]} &= [(-0) \cdot \log(0.4263)] \\ &\quad + [(-0) \cdot \log(0.2248)] \\ &\quad + [(-1) \cdot \log(0.3490)] \\ &= 1.05268... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[4]} &= [(-0) \cdot \log(0.2668)] \\ &\quad + [(-0) \cdot \log(0.2978)] \\ &\quad + [(-1) \cdot \log(0.4354)] \\ &= 0.831490... \end{aligned}$$

Cross Entropy Loss Function Example

$$\mathbf{Y}_{\text{onehot}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{\text{softmax outputs}} = \begin{bmatrix} 0.3792 & 0.3104 & 0.3104 \\ 0.3072 & 0.4147 & 0.2780 \\ 0.4263 & 0.2248 & 0.3490 \\ 0.2668 & 0.2978 & 0.4354 \end{bmatrix}$$

$$\begin{aligned} \mathcal{L}^{[1]} &= [(-1) \cdot \log(0.3792)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &+ [(-0) \cdot \log(0.3104)] \\ &= 0.969692... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[2]} &= [(-0) \cdot \log(0.3072)] \\ &+ [(-1) \cdot \log(0.4147)] \\ &+ [(-0) \cdot \log(0.2780)] \\ &= 0.880200... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[3]} &= [(-0) \cdot \log(0.4263)] \\ &+ [(-0) \cdot \log(0.2248)] \\ &+ [(-1) \cdot \log(0.3490)] \\ &= 1.05268... \end{aligned}$$

$$\begin{aligned} \mathcal{L}^{[4]} &= [(-0) \cdot \log(0.2668)] \\ &+ [(-0) \cdot \log(0.2978)] \\ &+ [(-1) \cdot \log(0.4354)] \\ &= 0.831490... \end{aligned}$$

$$\rightarrow \mathcal{L}_{\text{multiclass}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} -y_j^{[i]} \log\left(a_j^{[i]}\right)$$

$$\approx 0.9335$$

PyTorch already takes care of one-hot encoding

```
##########################
### MODEL
##########################

class LogisticRegression(torch.nn.Module):
    """Multinomial Logistic Regression model"""

    def __init__(self, num_features, num_classes):
        super(LogisticRegression, self).__init__()

        ### Output layer
        self.linear_out = torch.nn.Linear(NUM_FEATURES, NUM_CLASSES)

        # the following two lines are optional
        self.linear_out.weight.detach().normal_(0.0, 0.1)
        self.linear_out.bias.detach().zero_()

    def forward(self, x):
        logits = self.linear_out(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

Instead of doing things "manually" via F.nll_loss, use F.cross_entropy -- it is numerically more stable

```
[24]: print(targets)

      tensor([6, 1, 7, 8, 3], device='cuda:0')

[25]: print(logits)

      tensor([[-3.5890, -1.9838,  0.4137, -2.3949,  1.5565, -2.9962,  3.4829, -1.6488,
                0.9492,  1.2827],
              [-3.0150,  5.4310,  2.3293,  1.1424, -2.1697, -1.4378, -1.5242, -2.4102,
                2.0650, -2.0856],
              [-3.7686, -0.6936, -2.3394, -0.0395, -0.4688, -0.8471, -4.5556,  8.0915,
                0.5525,  3.4793],
              [-1.6363,  1.2846,  1.4050,  0.5723, -4.4089, -0.1598, -0.3495, -2.3774,
                5.4604, -1.4640],
              [ 0.9780, -2.3383,  0.4330,  6.4788, -4.3987,  3.1476, -0.0177, -4.1507,
                0.6837, -2.8333]], device='cuda:0', grad_fn=<SliceBackward>)

[26]: F.cross_entropy(logits, targets)

[26]: tensor(0.1065, device='cuda:0', grad_fn=<NllLossBackward>)

[27]: F.nll_loss(torch.log_softmax(logits, dim=1), targets)

[27]: tensor(0.1065, device='cuda:0', grad_fn=<NllLossBackward>)
```

PyTorch Loss-Input Cheatsheet

- `torch.nn.functional.binary_cross_entropy` takes logistic sigmoid values as inputs
- `torch.nn.functional.binary_cross_entropy_with_logits` takes logits as inputs
- `torch.nn.functional.cross_entropy` takes logits as inputs (performs `log_softmax` internally)
- `torch.nn.functional.nll_loss` is like `cross_entropy` but takes log-probabilities (log-softmax) values as inputs

Code Examples:

00_check-env.ipynb

01_logistic-regression-mnist-warmup.ipynb

Part 2: Dataset

Preparing a custom dataset for model training in PyTorch

Zhang, Z., Song, Y., & Qi, H. (2017). Age progression/regression by conditional adversarial autoencoder. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5810-5818).

https://susanqq.github.io/UTKFace/

Custom DataLoader based on MNIST:

02.1_dataloader-mnist.ipynb

Custom DataLoader for the UTKFace dataset (age prediction):

02.2_dataloader-UTKFace.ipynb

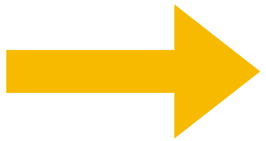
Part 3: Nominal Classifier

Implementing a convolutional neural network classifier for age prediction

Part 3: Nominal Classifier

Implementing a convolutional neural network classifier for age prediction

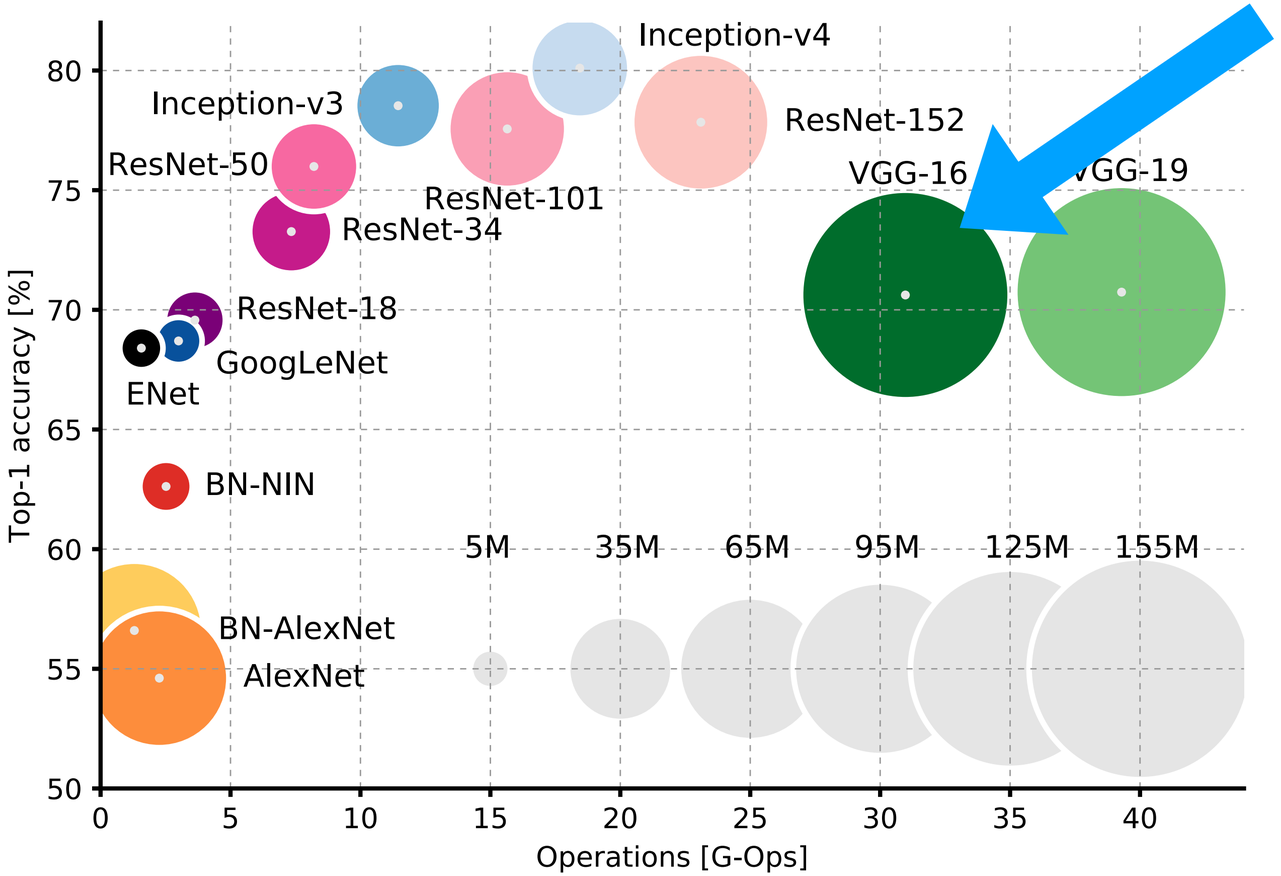
Ordinal regression approaches can be grouped into 3 categories:



1. Basic/naive

2. Binary decomposition

3. Threshold approaches



Model backbone for the rest of this tutorial

Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

VGG-16

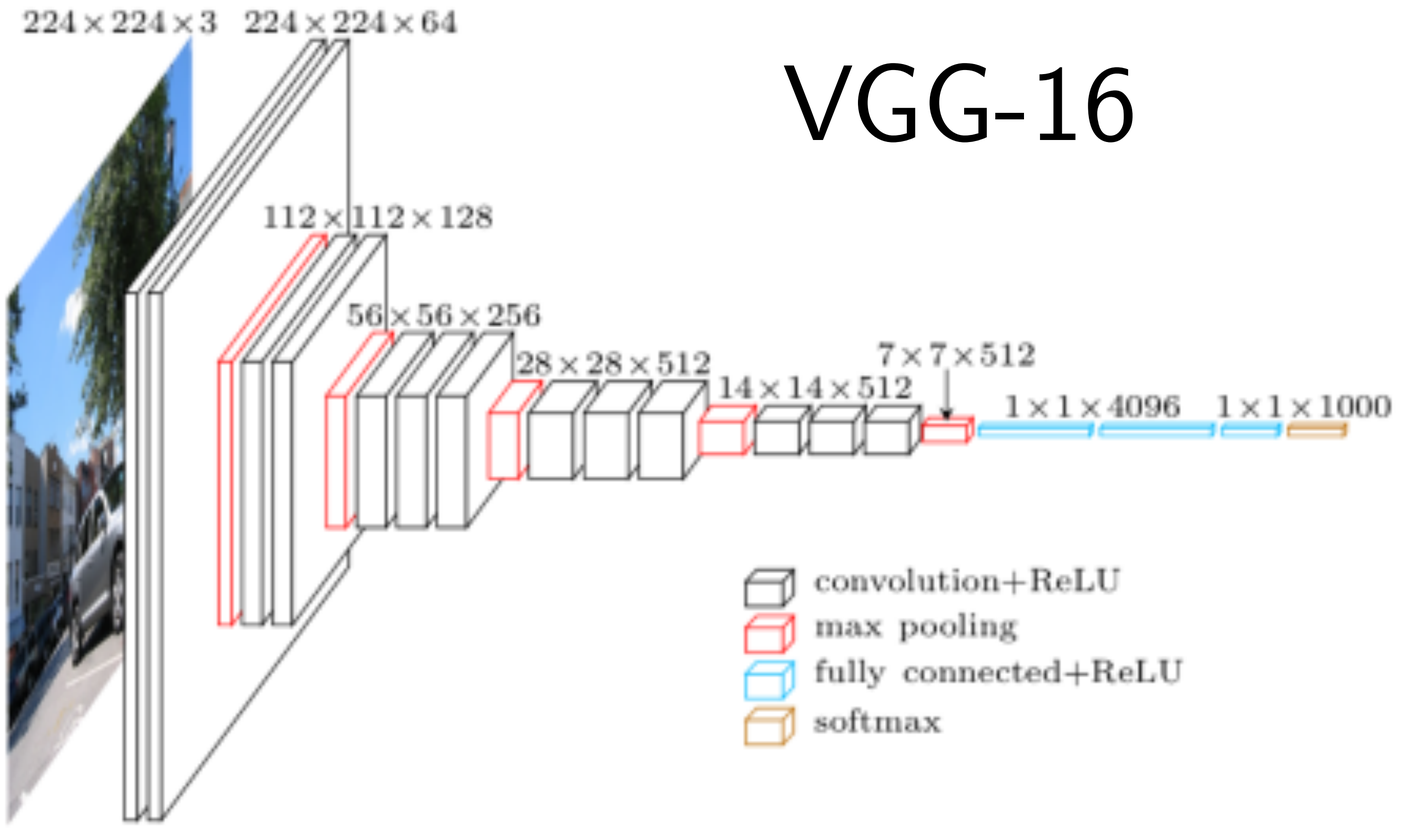


Image Source: http://www.cs.toronto.edu/~frossard/post/vgg16/

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

CNN (VGG-16) on UTKFace

03.1_CrossEntropy.ipynb

Question: Why is Cross Entropy not a good choice for ordinal tasks?

Part 4: Extended Binary Classifier

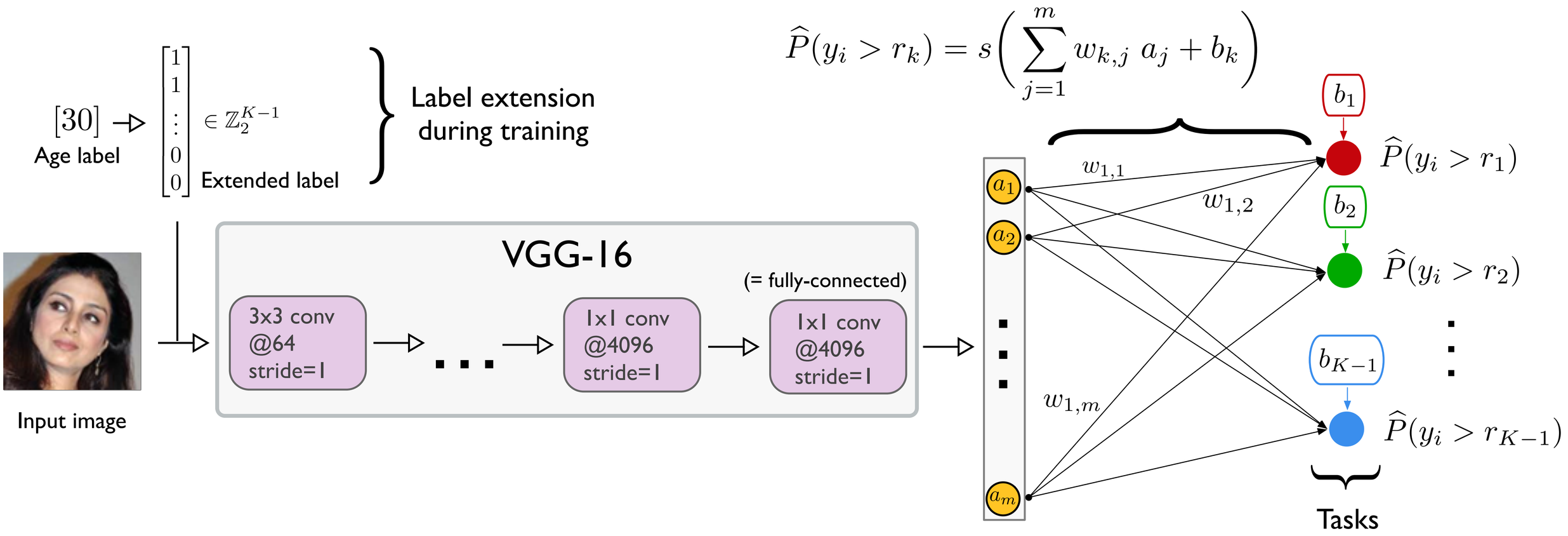
Ordinal regression via extended binary classification for age prediction

1. Basic/naive

2. Binary decomposition

3. Threshold approaches

Binary Decomposition



Binary Decomposition

the first class is $[0,\ldots,0]^\top$, the second class is $[1,\ldots,0]^\top$, the third class is $[1,1,\ldots,0]^\top$ and so on.

$$\widehat{P}(y_i > r_k) = s\left(\sum_{j=1}^{m} w_{k,j}\, a_j + b_k\right)$$

$[30]$ Age label

$$\begin{bmatrix}1\\1\\\vdots\\0\\0\end{bmatrix} \in \mathbb{Z}_2^{K-1}$$

Extended label

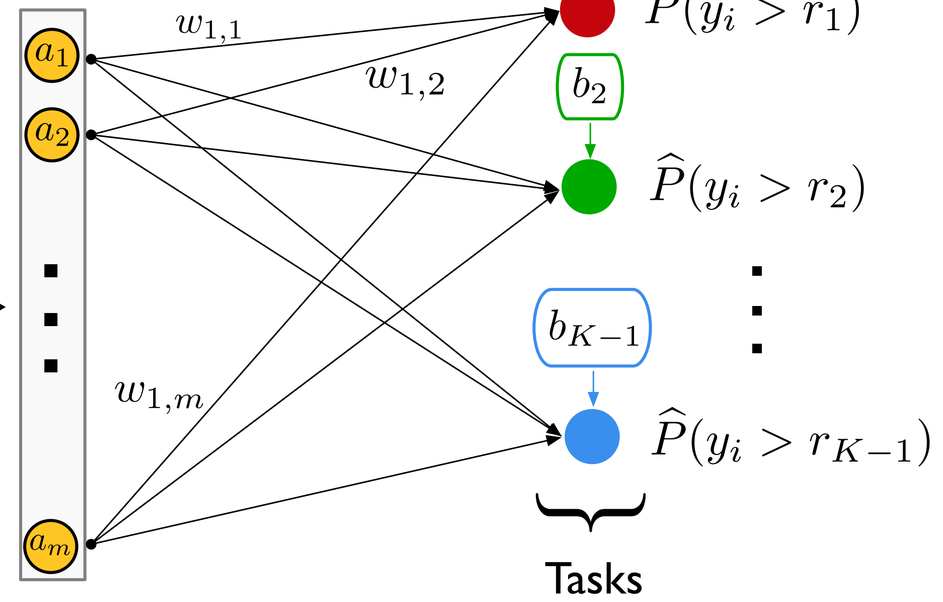
Label extension during training

Input image

VGG-16

(= fully-connected)

3x3 conv @64 stride=1 → . . . → 1x1 conv @4096 stride=1 → 1x1 conv @4096 stride=1



Age label

for each k:

$$1\{\widehat{P}\,(y_i > r_k)\} = f_k\,(\mathbf{x}_i) \in \{0, 1\}$$

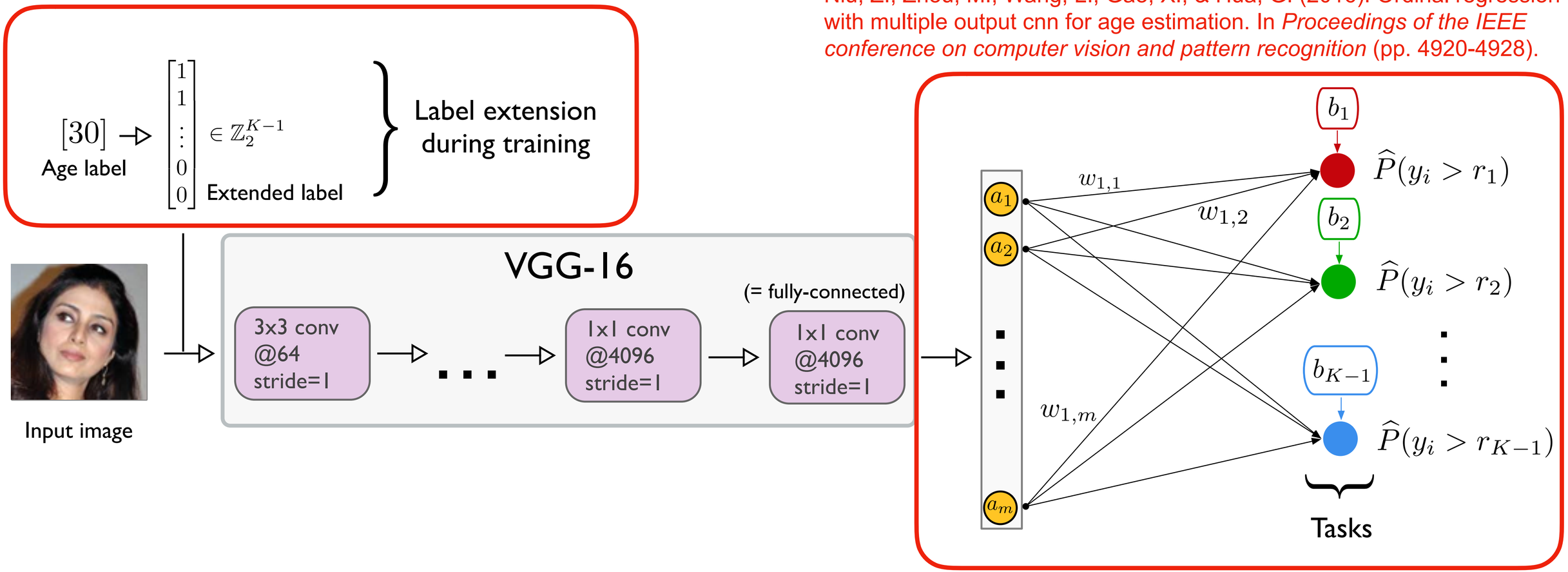
Then, compute rank label:

$$h\,(\mathbf{x}_i) = r_q, \quad q = 1 + \sum_{k=1}^{K-1} f_k\,(\mathbf{x}_i)$$

Binary Decomposition

Li, L., & Lin, H. T. (2007). Ordinal regression by extended binary classification. In *Advances in neural information processing systems* (pp. 865-872).

Niu, Z., Zhou, M., Wang, L., Gao, X., & Hua, G. (2016). Ordinal regression with multiple output cnn for age estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4920-4928).



Implementing Niu et al.'s approach with a VGG-16 backbone

03.2_DataLoader-for-binary-tasks.ipynb

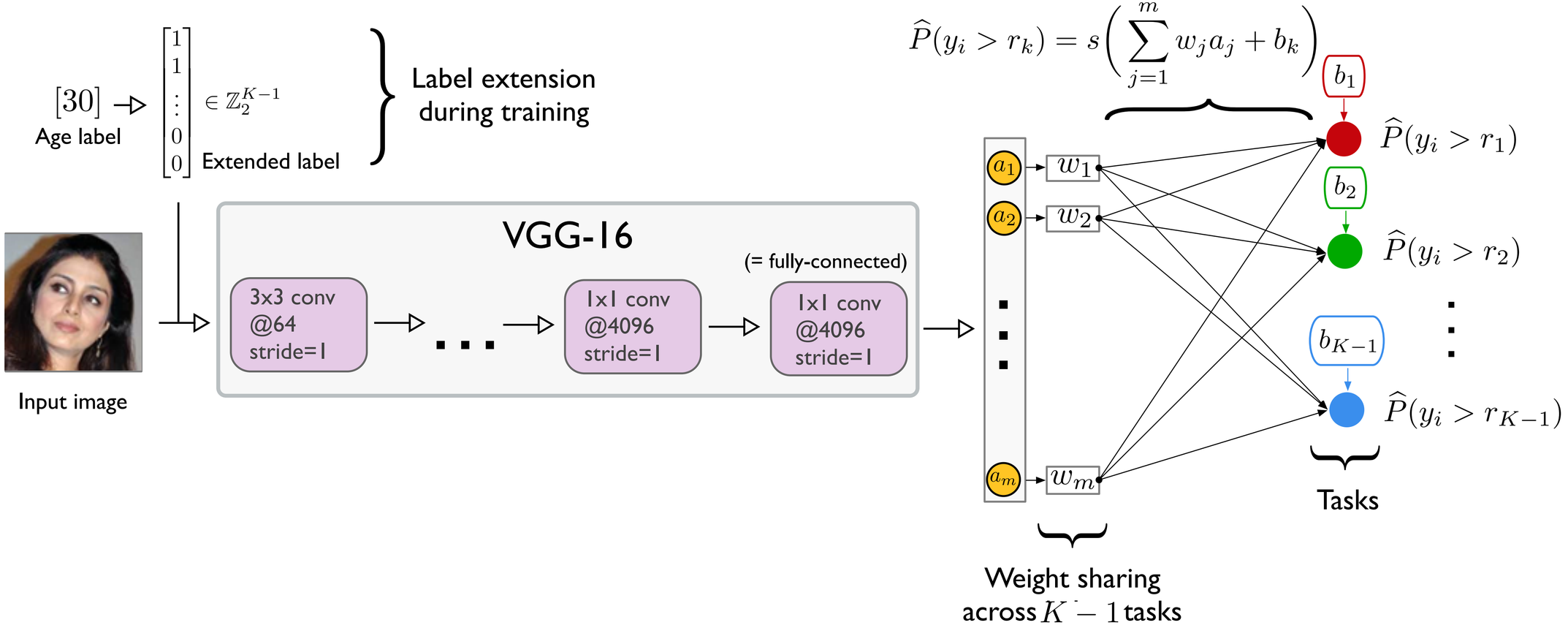
03.3_Niu-et-al.ipynb

Niu et al.'s approach does not guarantee rank consistency

For example, in an age estimation setting, it would be contradictory if the kth binary task predicted that the age of a person was larger than 30, but a previous task predicted it was smaller than 20, which is suboptimal when the $K - 1$ task predictions are combined to obtain the estimated age

CORAL (Consistent Rank Logits)

Wenzhi Cao, Vahid Mirjalili, Sebastian Raschka. Rank-consistent Ordinal Regression for Neural Networks. arXiv:1901.07884v3. https://arxiv.org/abs/1901.07884v3



Theorem 1 (ordered biases)**.** *By minimizing loss function defined in Eq.* (3)*, the optimal solution* $(\mathbf{W}^*, \mathbf{b}^*)$ *satisfies* $b_1^* \geq b_2^* \geq \ldots \geq b_{K-1}^*$.

Proof. Suppose $(\mathbf{W}, b)$ is an optimal solution and $b_k < b_{k+1}$ for some k. Claim: by either replacing b_k with b_{k+1} or replacing b_{k+1} with b_k, we can decrease the objective value L. Let

$$A_1 = \{n : y_n^{(k)} = y_n^{(k+1)} = 1\},\ A_2 = \{n : y_n^{(k)} = y_n^{(k+1)} = 0\},\ A_3 = \{n : y_n^{(k)} = 1,\ y_n^{(k+1)} = 0\}.$$

By the ordering relationship we have $A_1 \cup A_2 \cup A_3 = \{1, 2, \ldots, N\}$. Denote $p_n(b_k) = s(g(\mathbf{x}_n, \mathbf{W}) + b_k)$ and

$$\delta_n = \log(p_n(b_{k+1})) - \log(p_n(b_k)), \quad \delta'_n = \log(1 - p_n(b_k)) - \log(1 - p_n(b_{k+1})).$$

Since $p_n(b_k)$ is increasing in b_k, we have $\delta_n > 0$ and $\delta'_n > 0$.

If we replace b_k with b_{k+1}, the loss terms related to kth task are updated. The change of loss L (Eq. 3) is given as

$$\Delta_1 L = \lambda^{(k)} \Big[- \sum_{n \in A_1} \delta_n + \sum_{n \in A_2} \delta'_n - \sum_{n \in A_3} \delta_n \Big].$$

Accordingly, if we replace b_{k+1} with b_k, the change of L is given as

$$\Delta_2 L = \lambda^{(k+1)} \Big[\sum_{n \in A_1} \delta_n - \sum_{n \in A_2} \delta'_n - \sum_{n \in A_3} \delta'_n \Big].$$

By adding $\frac{1}{\lambda^{(k)}} \Delta_1 L$ and $\frac{1}{\lambda^{(k+1)}} \Delta_2 L$, we have

$$\frac{1}{\lambda^{(k)}} \Delta_1 L + \frac{1}{\lambda^{(k+1)}} \Delta_2 L = - \sum_{n \in A_3} (\delta_n + \delta'_n) < 0,$$

and know that either $\Delta_1 L < 0$ or $\Delta_2 L < 0$. Thus, our claim is justified, and we conclude that any optimal solution $(\mathbf{W}^*, b^*)$ that minimizes L satisfies $b_1^* \geq b_2^* \geq \ldots \geq b_{K-1}^*$. □

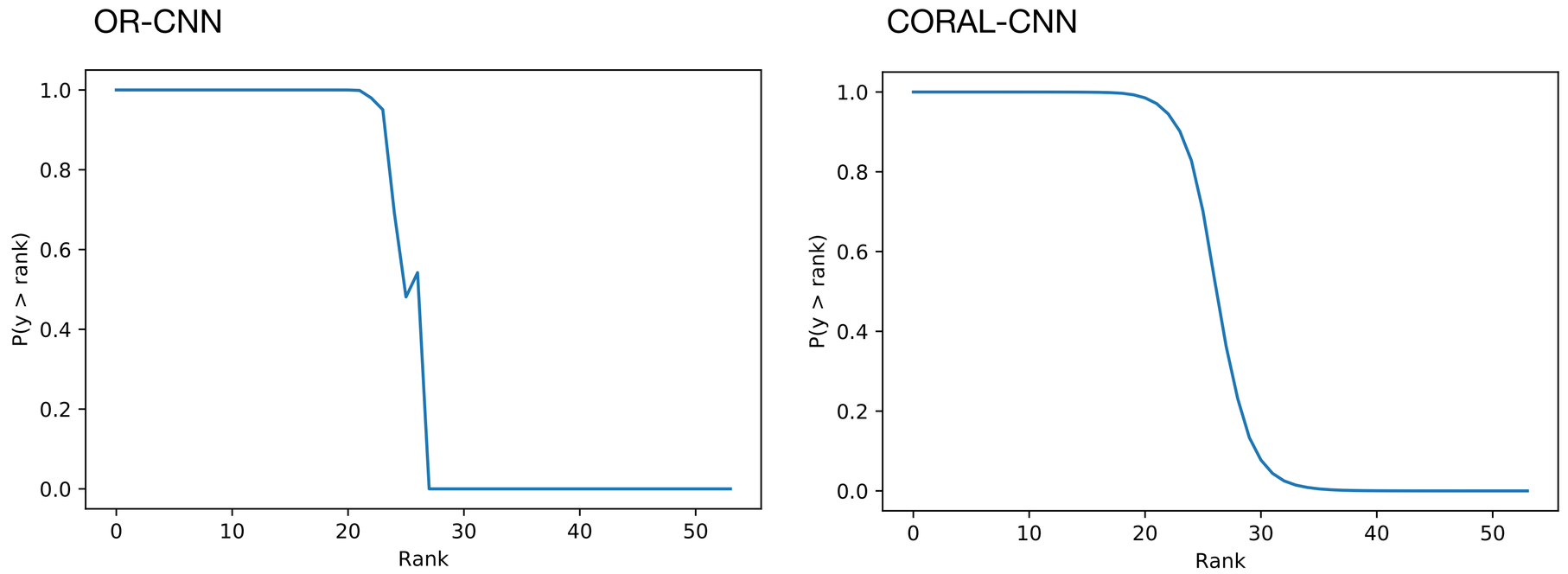


Figure S2: Plots show graphs of the predicted probabilities for each binary classifier task on one test data point in the MORPH dataset by OR-CNN (left subpanel) and CORAL-CNN (right subpanel). In this example, the ordinal regression CNN has an inconsistency at rank 26. The CORAL-CNN does not suffer from inconsistencies such that the rank prediction is a cumulative distribution function.

Wenzhi Cao, Vahid Mirjalili, Sebastian Raschka. Rank-consistent Ordinal Regression for Neural Networks. arXiv:1901.07884v3. https://arxiv.org/abs/1901.07884v3

Implementing CORAL with a VGG-16 backbone

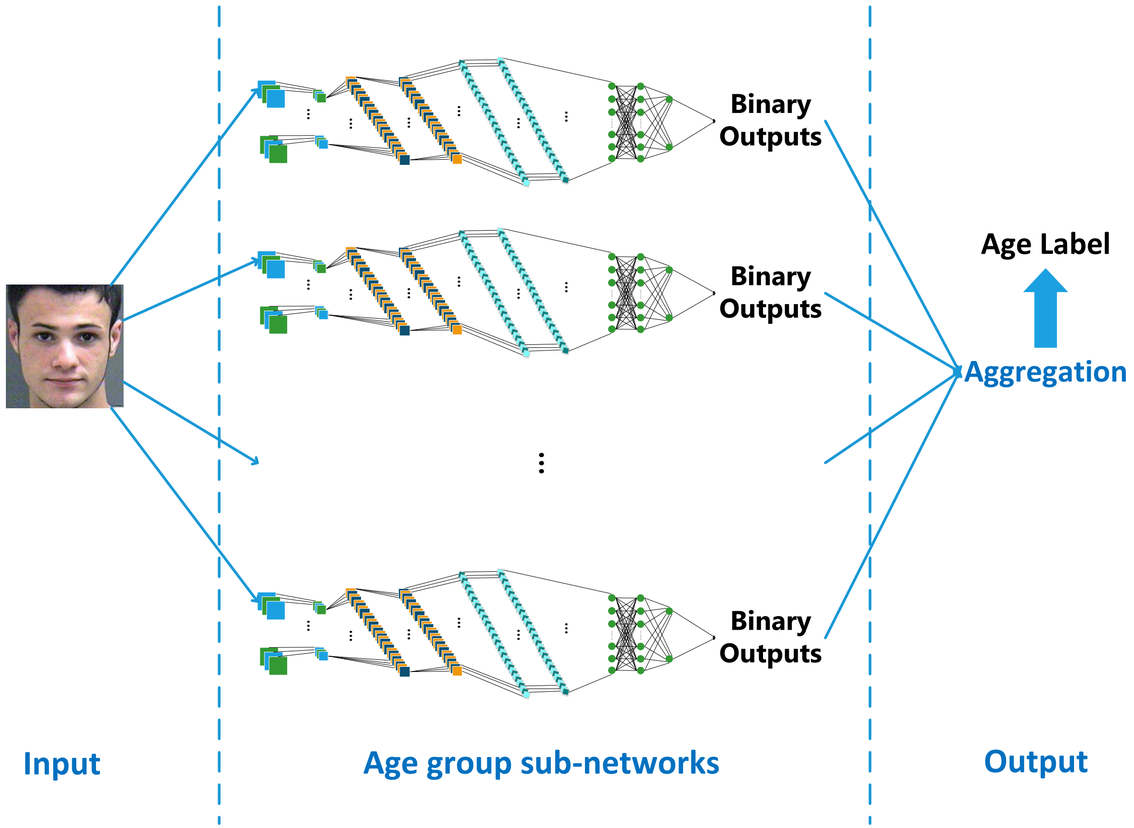
4.1_CORAL.ipynb

Part 5: Other Approaches

Other approaches for ordinal regression

Ensemble CNN for Ordinal Regression

Chen, S., Zhang, C., Dong, M., Le, J., & Rao, M. (2017, July). Using Ranking-CNN for age estimation. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).



- Extended binary classification (similar to Niu et al. 2017 and CORAL)
- However, 1 CNN per binary task (very expensive!)
- According to their paper, it slightly outperforms Niu et al.'s method (but k times more expensive, where k is the number of tasks)

Squared Error Reformulation

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

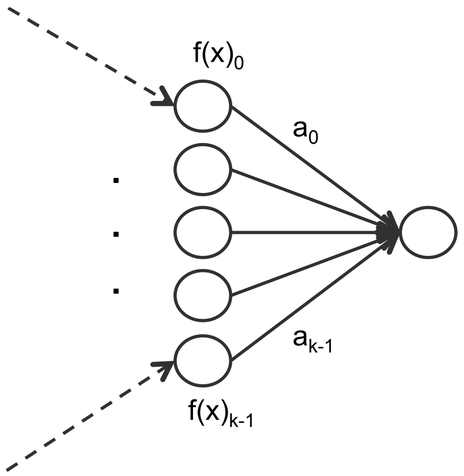
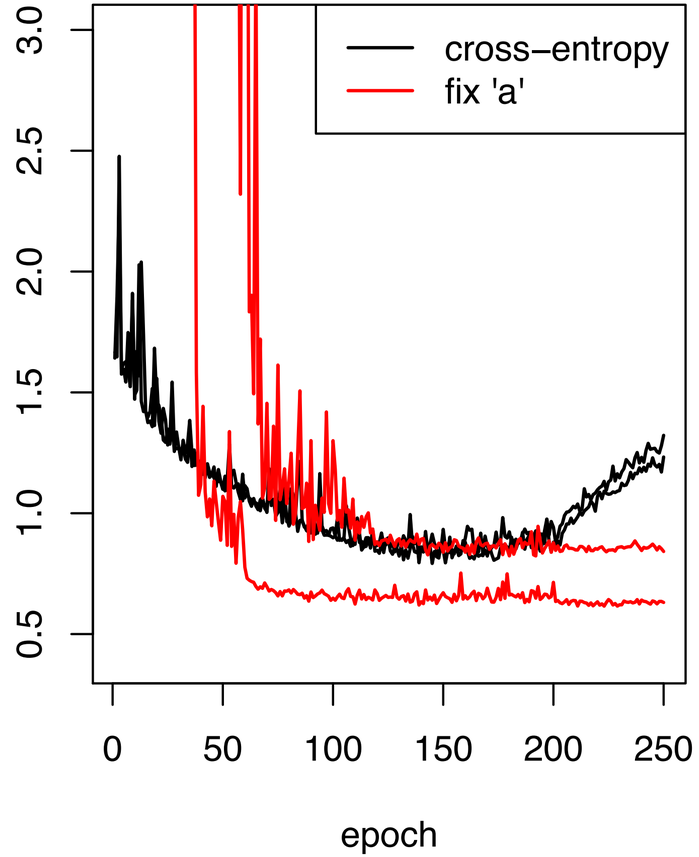


Figure 1: The final layers of an arbitrary neural network. The penultimate layer $f(\mathbf{x})$ in this example has $k = 5$ units. The final layer (consisting of one hidden unit) computes $p(c|\mathbf{x})$ and is parameterized by weight vector $\mathbf{a}$.



Squared Error Reformulation

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

Conventional classifier:

$$\sigma_{\text{softmax}}(\text{logits})$$

Basically, like an additional layer

Beckham et al.:

$$(c-1) * \sigma_{\text{sigmoid}}\left(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\right)$$

```
logits = self.fc(x)
probas = torch.softmax(logits, dim=1)
predictions = ((self.num_classes-1)
                        * torch.sigmoid(probas.mm(self.a).view(-1)))
return logits, probas, predictions
```

Squared Error Reformulation

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

$$\text{pred} = (c-1) * \sigma_{\text{sigmoid}}\Big(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\Big)$$

$$\text{loss} = \frac{1}{n}\sum_{i=1}^{n}\Big(\text{pred}^{(i)} - \text{target}^{(i)}\Big)^2$$

Assume true label 3, and you predict

a)

$$\text{proba} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.7 \\ 0.1 \end{bmatrix}$$

label 2

b)

$$\text{proba'} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

label 0

proba, f(x)

$$\text{pred} = (c-1) * \sigma_{\text{sigmoid}}\left(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\right)$$

Assume true label 3, and you predict

a)

$$\text{proba} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.7 \\ 0.1 \end{bmatrix}$$

label 2

b)

$$\text{proba'} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

label 0

if $\mathbf{a} = [0, 1, ..., c-1]^\top$

then the penalty for b) is higher

$$\text{loss} = \frac{1}{n} \sum_{i=1}^{n} \left(\text{pred}^{(i)} - \text{target}^{(i)} \right)^2$$

proba, f(x)

$$\text{pred} = (c-1) * \sigma_{\text{sigmoid}} \left(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a} \right)$$

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

"In this paper we present a simple ordinal classification technique whose formulation is a Gaussian distribution whose mean is based on a softmax hidden layer. When we use this model in conjunction with a squared error loss, we not only obtain a loss that is sensitive to class ordering (due to the utilization of squared error), we can still obtain a discrete probability distribution over the class through f(x)."

Incorrect; there is no sensitivity towards class ordering, can you spot the issue?

$$(c-1) * \sigma_{\text{sigmoid}}\left(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\right)$$

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

Incorrect; there is no sensitivity towards class ordering, can you spot the issue?

$$(c-1) * \sigma_{\text{sigmoid}}\big(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\big)$$

dot product is symmetric, hence the order in a does not matter -> you can simply reorder a relative to f(x); thus, there is no relationship between class order and the predictions. I.e., you can just choose an arbitrary ordering for

$$\mathbf{a} = [0, 1, ..., c-1]^\top$$

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

Incorrect; there is no sensitivity towards class ordering, can you spot the issue?

$$(c-1) * \sigma_{\text{sigmoid}}\left(\sigma_{\text{softmax}}(\text{logits}) \cdot \mathbf{a}\right)$$

1) a = (2,2,2,2) f = (0.25, 0.25, 0.25, 0.25) a.f = 2
2) a = (0,0,0,2) f = (0, 0, 0, 1) a.f = 2
3) a = (0,1,2 ,3) f = (0.8, 0.7, 0.6, 1/30) a.f = 2

Beckham, C., & Pal, C. (2016). A simple squared-error reformulation for ordinal classification. arXiv preprint arXiv:1612.00775.

My implementation with ResNet-34 backbone on AFAD-Lite:

- https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/ordinal/ordinal-cnn-beckham2016-afadlite.ipynb

  MAE: 3.40

- https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/ordinal/ordinal-cnn-niu-afadlite.ipynb

  Niu et al. MAE: 3.91

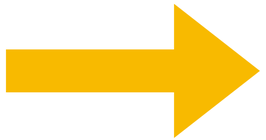
- https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/ordinal/ordinal-cnn-coral-afadlite.ipynb

  CORAL MAE: 3.38

Threshold approaches

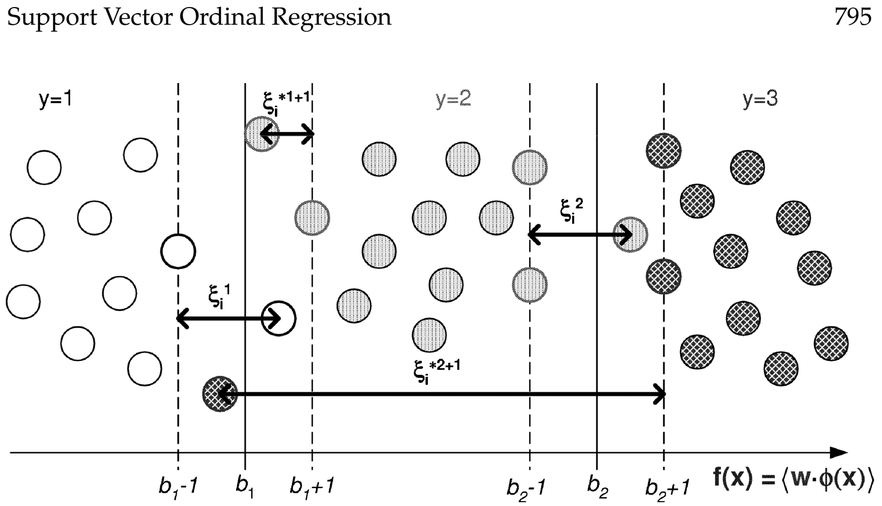
Ordinal regression approaches can be grouped into 3 categories:

1. Basic/naive
2. Binary decomposition
3. Threshold approaches



Threshold Approaches

- assume that there is a latent function mapping the instances to a real line
- the categories of the instances are intervals on the line
- the natural order of interval boundaries on the real line represents the ordinal relationship between categories



Chu, W., & Keerthi, S. S. (2007). Support vector ordinal regression. *Neural computation*, *19*(3), 792-815.

Ordinal Regression Categories

Ordinal regression approaches can be grouped into 3 categories:

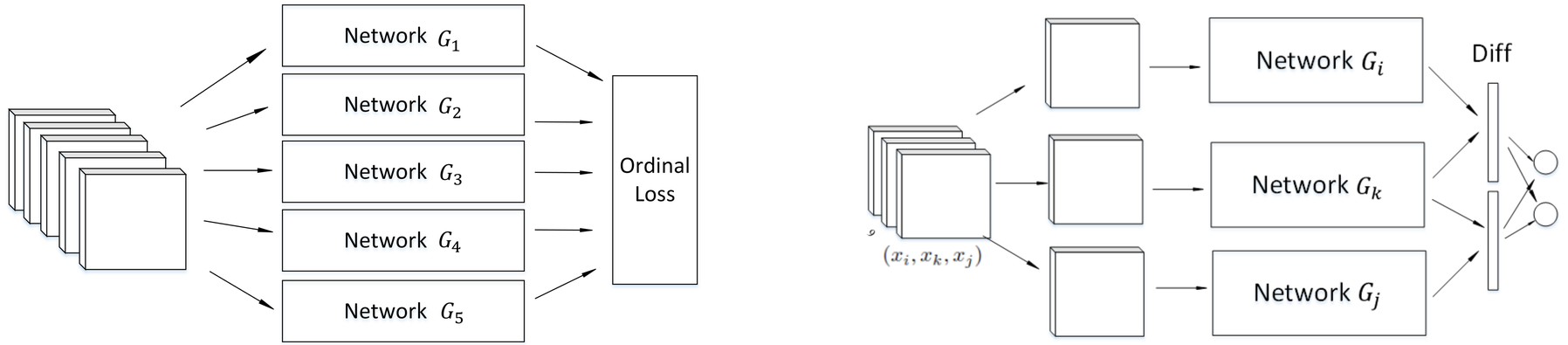
1. Basic/naive
2. Binary decomposition
3. Threshold approaches

] "Point-wise" methods

In all above point-wise methods, the parameters are learned from individual data points, and therefore, the relationship between instances is not explored explicitly.

4th Category: Pair-wise methods

- typically based on triplets
- i.e., answering the questions: "Is $k - 1 < y_t < k+1$ true?"



(a) The architecture of CNN_{reg} (b) The architecture of CNN_k

Figure 2: The architecture of CNNOR for a 5-rank problem.

Liu, Y., Kong, A. W. K., & Goh, C. K. (2017, August). Deep Ordinal Regression Based on Data Relationship for Small Datasets. In *IJCAI* (pp. 2372-2378).

Thank you for attending!