Using Deep Learning When Class Labels Have A Natural Order

Predicting Ratings And Rankings Using PyTorch Lightning

Sebastian Raschka
Lead AI Educator @ Lightning AI
Asst. Prof. of Statistics @ University of Wisconsin

@rasbt
sebastian@lightning.ai
https://sebastianraschka.com

https://lightning.ai

SciPy2022
Scientific Computing with Python
Austin, TX • July 25 - July 27
Code & slides

https://github.com/rasbt/scipy2022-talk
Many real-world predictions problems have ordered labels.

Plant disease

<table>
<thead>
<tr>
<th>Index</th>
<th>Reaction</th>
<th>PLRV</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Highly</td>
<td>Resistance</td>
</tr>
<tr>
<td>1</td>
<td>Resistance</td>
<td>Rolling of leaves in case of primary infection and lower leaves in case of secondary infection, erect growth</td>
</tr>
<tr>
<td>2</td>
<td>Moderately Resistance</td>
<td>Rolling of leaves extending, leaves become stiff and leathery, stunting of plants and erect growth</td>
</tr>
<tr>
<td>3</td>
<td>Moderately Susceptible</td>
<td>Short internodes, papery sound of leathery leaves, rolling and stunting of whole plants. Young buds are slightly yellowish and purplish</td>
</tr>
<tr>
<td>4</td>
<td>Susceptible</td>
<td>Clear rolling of leaves, severe stunting, few tubers and tuber necrosis</td>
</tr>
<tr>
<td>5</td>
<td>Highly Susceptible</td>
<td>All above symptoms and small number of small sized tubers.</td>
</tr>
</tbody>
</table>

Damage assessment

- Grade 1: Moderate damage (light structural damage, andleuro loss, slight brown necrosis, rolling leaves, few small pieces of plant tissues, and upper part of plant tissues is very few leaves)
- Grade 2: Moderate damage (slight structural damage, andleuro loss, slight brown necrosis, rolling leaves, few small pieces of plant tissues, and upper part of plant tissues is very few leaves)
- Grade 3: Severe damage (severe structural damage, brown necrosis in leaves, rolling leaves, few small pieces of plant tissues, and upper part of plant tissues is very few leaves)
- Grade 4: Very heavy damage (heavy structural damage, very black necrosis in leaves, rolling leaves, few small pieces of plant tissues, and upper part of plant tissues is very few leaves)
- Grade 5: Destruction (very heavy structural damage, total or more total collapse)

Customer reviews

4 out of 5

Credit risk rating

<table>
<thead>
<tr>
<th>PASS</th>
<th>SPECIAL MENTION</th>
<th>SUB- STANDARD</th>
<th>DOUBTFUL</th>
<th>LOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Likert scale for customer satisfaction

Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree
Ordered labels? Tell me more!
Ordered labels? Tell me more!

How do ordered (ordinal) labels differ from conventional class labels
Ordered labels? **Tell me more!**

Classification

Setosa   Versicolor   Virginica

No ordering
Ordered labels? Tell me more!

Classification

1 Setosa  2 Versicolor  3 Virginica

No ordering
Ordered labels? Tell me more!

Classification

1 Setosa  2 Versicolor  3 Virginica

No ordering

Regression

1  2  3
Ordered labels? Tell me more!

**Classification**
1 Setosa  2 Versicolor  3 Virginica
No ordering

**Regression**
1 < 2 < 3
Ordered labels?  Tell me more!

Classification

<table>
<thead>
<tr>
<th>1</th>
<th>Setosa</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Versicolor</td>
</tr>
<tr>
<td>3</td>
<td>Virginica</td>
</tr>
</tbody>
</table>

No ordering

Regression

1 < 2 < 3

Identical distances
Ordered labels? Tell me more!

Classification

1 Setosa
2 Versicolor
3 Virginica

No ordering

Ordinal regression / ordinal classification

Regression

Identical distances
Ordered labels? Tell me more!

Classification

1 Setosa 2 Versicolor 3 Virginica
No ordering

Ordinal regression / ordinal classification

1 😠 2 😐 3 😊

Regression

Identical distances
Ordered labels? Tell me more!

Classification

Ordinal regression / ordinal classification

Regression

1 Setosa
2 Versicolor
3 Virginica

No ordering

1 ☹️
2 😞
3 😊

1 < 2 < 3

Identical distances

Ordered labels?
Ordered labels? Tell me more!

Classification

Class labels
• but with ordering info
• and arbitrary distances

Ordinal regression / ordinal classification

Regression

Identical distances
Can't we just use regular classifiers for ordered labels?
Can't we just use regular classifiers for ordered labels?

Yes, but it is not ideal
It is **not ideal** because all wrong predictions look equally wrong to a classifier.
It is **not ideal** because all wrong predictions look equally wrong to a classifier

Assume this is the true label

1 😞
It is **not ideal** because all wrong predictions look equally wrong to a classifier.

Assume this is the true label

- 1 😞

Wrong prediction

- 2 😞
It is **not ideal** because all wrong predictions look equally wrong to a classifier.

Assume this is the true label

Wrong prediction

Wrong prediction
It is **not ideal** because all wrong predictions look equally wrong to a classifier.

Assume this is the true label

![1](image1)

Wrong prediction

![2](image2)

Wrong prediction

![3](image3)

Treated **equally** if we compute the **loss** in a regular classifier.
It is **not ideal** because all wrong predictions look equally wrong to a classifier.

Assume this is the true label

Wrong prediction

Wrong prediction

But this should be “more wrong”
Many real-world predictions problems have ordered labels.

After incubation the plates were washed with washing buffer 3 times. The collected leaf samples mixed with virus extraction buffer @ 1:10 with the help of mortar and pestle to extract the sap and homogenized. The six different plates for six viruses were filled with the sap @ 200ul for each well with the help of micropipette. Before adding the samples into the plates, filled 2 wells for the positive control and 2 for the negative control of the six viruses separately. The coated plate was again incubated for overnight at 4°C.

Step 3
- Washed the plates with washing buffer 3 times and then take conjugate 20ul and added conjugate buffer 20ml separately for each virus.
- Poured the plates at the rate of 200ul for each well.
- Incubated the plate at 4°C overnight.

Step 4
- Washed the plate 3 times with washing buffer and then take 20ml substrate buffer and 1 PNP tablet.
- Mixed tablet in substrate buffer.
- Poured the plates @ 200ul for each well with the help of micropipette.
- Put the plate for 30 minutes at room temperature and reaction was visually observed for the development of yellow color.
- The reaction was stopped by adding 50µl 3M NaOH to each well.
- The results were compiled by the following scale:
  - Deep Yellow= (++)= Highly Susceptible
  - Moderate Yellow= Moderate(++)= Moderately Susceptible
  - No color= Resistant

Table 1. Disease Rating Scale for PLRV, PVX and PVY

<table>
<thead>
<tr>
<th>Index</th>
<th>Reaction</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Very high resistance</td>
<td>No visible symptoms.</td>
</tr>
<tr>
<td>1</td>
<td>Resistance</td>
<td>Rolling of leaves in case of primary infection and lower leaves in case of secondary infection, erect growth</td>
</tr>
<tr>
<td>2</td>
<td>Moderately Resistance</td>
<td>Rolling of leaves extending, leaves become stiff and leathery, stunting of plants and erect growth</td>
</tr>
<tr>
<td>3</td>
<td>Moderately Susceptible</td>
<td>Short internodes, papery sound of leathery leaves, rolling and stunting of whole plants. Young buds are slightly yellowish and purplish</td>
</tr>
<tr>
<td>4</td>
<td>Susceptible</td>
<td>Clear rolling of leaves, severe stunting, few tubers and tuber necrosis</td>
</tr>
<tr>
<td>5</td>
<td>Highly Susceptible</td>
<td>All above symptoms and small number of small sized tubers.</td>
</tr>
</tbody>
</table>

Credit risk rating

<table>
<thead>
<tr>
<th>PASS</th>
<th>SPECIAL</th>
<th>STANDARD</th>
<th>DOUBTFUL</th>
<th>LOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Largely risk free</td>
<td>Minimal risk</td>
<td>Modest risk</td>
<td>Bankable</td>
<td>Additional review</td>
</tr>
</tbody>
</table>

Likert scale for customer satisfaction

(+): Strongly Positive
(-): Strongly Negative
Many real-world predictions problems have ordered labels.

And we can get much better performance using ordinal regression models rather than regular classifiers.
How? Let's (re)use what we already know:
An extended *binary classification* framework
How? Let's (re)use what we already know:
An extended **binary classification** framework

**Input**
(Aesthetics dataset)

**Possible labels:**
Rating 1, 2, 3, 4, 5
How? Let's (re)use what we already know: An extended **binary classification** framework.

Possible labels:
Rating 1, 2, 3, 4, 5

Input (Aesthetics dataset)

Any neural network (CNN, RNN, MLP, ...)

How? Let's (re)use what we already know:
An extended **binary classification** framework

Possible labels:
Rating 1, 2, 3, 4, 5

Any neural network
(CNN, RNN, MLP, ...)

Input (Aesthetics dataset)

Score between 0 and 1

Input
(Aesthetics dataset)

Possible labels:
Rating 1, 2, 3, 4, 5

Any neural network
(CNN, RNN, MLP, …)

50% probability threshold

\[ P(\text{Rating} > 1) \]
\[ P(\text{Rating} > 2) \]
\[ P(\text{Rating} > 3) \]
\[ P(\text{Rating} > 4) \]

Score between 0 and 1

Binary classification task

\[ \text{Rating} > 1? \quad \rightarrow \quad \text{yes/no} \]
Input (Aesthetics dataset)

Possible labels:
Rating 1, 2, 3, 4, 5

Any neural network (CNN, RNN, MLP, …)

Predicted ordinal label is the sum over the yeses + 1
Input
(Aesthetics dataset)

Possible labels:
Rating 1, 2, 3, 4, 5

Any neural network
(CNN, RNN, MLP, …)

50% probability threshold

\[
P(\text{Rating} > 1) \quad \text{Rating} > 1? \rightarrow \text{yes/no}
\]

\[
P(\text{Rating} > 2) \quad \text{Rating} > 2? \rightarrow \text{yes/no}
\]

\[
P(\text{Rating} > 3) \quad \text{Rating} > 3? \rightarrow \text{yes/no}
\]

\[
P(\text{Rating} > 4) \quad \text{Rating} > 4? \rightarrow \text{yes/no}
\]

Each output node is a binary task

Predicted ordinal label is
the sum over the yeses + 1

Predicted label:
3
Problem: rank inconsistency
Problem: rank inconsistency

Addressing the rank inconsistency issue leads to better predictive performance

Cao, Mirjalili, Raschka (2020)
*Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation*
Cao, Mirjalili, Raschka (2020)
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation
Fixing rank inconsistency introduced a limitation: weight-sharing constraint restricts the network's capacity
**Weight-sharing constraint**

Cao, Mirjalili, Raschka (2020)
*Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation*
Removing the weight-sharing constraint (while maintaining rank consistency) leads to even better performance

Shi, Cao, Raschka
Deep Neural Networks for Rank-Consistent Ordinal Regression Based On Conditional Probabilities.
CORN in a nutshell: chain rule for probabilities

Any neural network
(CNN, RNN, MLP, …)
CORN in a nutshell: chain rule for probabilities

Conditional probability:

\[ f_k(x[i]) = \hat{P}(y[i] > r_k \mid y[i] > r_{k-1}) \]

"Predicted probability that the label exceeds rank \( k \) given that it exceed rank \( k-1 \)"
CORN in a nutshell: chain rule for probabilities

Conditional probability:

\[ f_k(x[i]) = \hat{P}(y[i] > r_k \mid y[i] > r_{k-1}) \]

(Learned via conditional training subsets; more details in paper)

suppose \( k = 2 \)
CORN in a nutshell: chain rule for probabilities

Conditional probability:
\[ f_k (x[i]) = \hat{P} (y[i] > r_k \mid y[i] > r_{k-1}) \]

Apply chain rule for probabilities to obtain unconditional probability:
\[ \hat{P} (y[i] > r_k) = \prod_{j=1}^{k} f_j (x[i]) \]
CORN in a nutshell: chain rule for probabilities

Conditional probability:

\[ f_k(x[i]) = \hat{P}(y[i] > r_k \mid y[i] > r_{k-1}) \]

Apply chain rule for probabilities to obtain unconditional probability:

\[
\hat{P}(y[i] > r_k) = \prod_{j=1}^{k} f_j(x[i])
\]

\[
\hat{P}(y[i] > r_2) = \hat{P}(y[i] > r_2 \mid y[i] > r_1) \cdot \hat{P}(y[i] > r_1)
\]

Suppose \( k = 2 \)
CORN in a nutshell: chain rule for probabilities

\[ \hat{P}(y[i] > r_2) = \hat{P}(y[i] > r_2 | y[i] > r_1) \cdot \hat{P}(y[i] > r_1) \]

Left side guaranteed to be equal or less than right side

(Rank consistency: Rank probabilities are decreasing)
Cao, Mirjalili, Raschka (2020)
*Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation*

Shi, Cao, Raschka
*Deep Neural Networks for Rank-Consistent Ordinal Regression Based On Conditional Probabilities.*
How do these methods compare?
How?

Weight-sharing in output layer
(mathematical proof in paper)
How?

Weight-sharing in output layer
(mathematical proof in paper)

Chain rule for probabilities
& conditional training sets
<table>
<thead>
<tr>
<th>How?</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight-sharing in output layer (mathematical proof in paper)</td>
<td>• Easy to implement</td>
</tr>
<tr>
<td></td>
<td>• Reduced overfitting</td>
</tr>
<tr>
<td></td>
<td>• Fast</td>
</tr>
<tr>
<td>Chain rule for probabilities &amp; conditional training sets</td>
<td></td>
</tr>
<tr>
<td>How?</td>
<td>Advantages</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------</td>
</tr>
</tbody>
</table>
| Weight-sharing in output layer (mathematical proof in paper) | - Easy to implement  
- Reduced overfitting  
- Fast |
| Chain rule for probabilities & conditional training sets | - Easy to implement  
- Higher capacity  
- Better predictive performance |
Skipping over further mathematical details ... How do we use this *in practice*?
Converting a classifier into a **CORN** model
in 3 lines of code
Converting a classifier into a **CORN** model

in 3 lines of code

More code examples for tabular, text, and image data

PyTorch Lightning

Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a classifier into a **CORN** model in 3 lines of code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...
        output_layer = torch.nn.Linear(hidden_units[-1],
                                        num_classes)

        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

*Any* neural network (CNN, RNN, MLP, …)

PyTorch Lightning

Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a classifier into a **CORN** model

in 3 lines of code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...

        output_layer = torch.nn.Linear(hidden_units[-1],
                                        num_classes)

        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

Update the number of classes

Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a classifier into a **CORN** model in 3 lines of code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...
        output_layer = torch.nn.Linear(hidden_units[-1], num_classes)
        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

Why $-1$
Converting a classifier into a **CORN** model in 3 lines of code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...
        output_layer = torch.nn.Linear(hidden_units[-1], num_classes)

        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

"Rating > 4? Yes" implies Rating = 5

---

PyTorch Lightning

Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a classifier into a **CORN** model in 3 lines of code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

        def _shared_forward_step(self, batch, batch_idx):
            features, true_labels = batch
            logits = self(features)

            loss = torch.nn.functional.cross_entropy(logits, true_labels)
            predicted_labels = torch.argmax(logits, dim=1)
            return loss, predicted_labels
```

② Replace the standard cross entropy loss
Converting a Classifier into a **CORN** Model in 3 Lines of Code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

    def shared_forward_step(self, batch, batch_idx):
        features, true_labels = batch
        logits = self(features)

        loss = torch.nn.functional.cross_entropy(logits, true_labels)
        predicted_labels = torch.argmax(logits, dim=1)
        return loss, predicted_labels

from coral_pytorch.losses import corn_loss

loss = corn_loss(logits, true_labels,
                 num_classes=self.model.num_classes)
```

PyTorch Lightning
Full examples: https://raschka-research-group.github.io/coral-pytorch/
Converting a classifier into a **CORN** model
in 3 lines of code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

        def _shared_forward_step(self, batch, batch_idx):
            features, true_labels = batch
            logits = self(features)
            loss = torch.nn.functional.cross_entropy(logits, true_labels)

            predicted_labels = torch.argmax(logits, dim=1)

            return loss, predicted_labels
```

3. Convert logits to classes
Converting a Classifier into a **CORN** Model
in 3 Lines of Code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

    def shared_forward_step(self, batch, batch_idx):
        features, true_labels = batch
        logits = self(features)
        loss = torch.nn.functional.cross_entropy(logits, true_labels)

        predicted_labels = torch.argmax(logits, dim=1)
        return loss, predicted_labels

from coral_pytorch.dataset import corn_label_from_logits

predicted_labels = corn_label_from_logits(logits)
```

Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Code

https://github.com/rasbt/scipy2022-talk
(coral-pytorch) gridai@session:~/scipy2022-talk/src -> python main_mlp.py \
> --batch_size 16 \ 
> --data_path ../datasets/ \ 
> --learning_rate 0.005 \ 
> --mixed_precision true \ 
> --num_epochs 40 \ 
> --num_workers 3 \ 
> --output_path ./cement_strength \ 
> --loss_mode crossentropy
python main_mlp.py \
--batch_size 16 \
--data_path ../datasets/ \
--learning_rate 0.005 \
--mixed_precision true \
--num_epochs 40 \
--num_workers 3 \
--output_path ./cement_strength \
--loss_mode corn
Code

https://github.com/rasbt/scipy2022-talk
More examples (CNN, RNN, MLP):
https://raschka-research-group.github.io/coral-pytorch/
Acknowledgements

Wenzhi Cao
Xintong Shi
Vahid Mirjalili

William Falcon
Adrian Wälchli
Jirka Borovec
Marc Ferradou
Machine Learning with PyTorch and Scikit-Learn

Develop machine learning and deep learning models with Python

PyTorch book of the bestselling and widely acclaimed Python Machine Learning series

Foreword by:
Eugene Dethulal
PyTorch Care Memboller

Sebastian Raschka
YuXi (Hayden) Liu
Vahid Mirjalili

Feb 25

https://sebastianraschka.com/books/
Contact

@rasbt

sebastian@lightning.ai

https://sebastianraschka.com

Code & slides

https://github.com/rasbt/scipy2022-talk
Additional Slides for Q&A
Converting a Classifier into a CORAL Model in 4 Lines of Code

Full examples: https://raschka-research-group.github.io/coral-pytorch/
Converting a Classifier into a **CORAL** Model in 4 Lines of Code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...

        output_layer = torch.nn.Linear(hidden_units[-1],
                                        num_classes)

        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

PyTorch Lightning
Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a Classifier into a **CORAL** Model in 4 Lines of Code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...
        output_layer = torch.nn.Linear(hidden_units[-1], num_classes)
        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x
```

PyTorch Lightning

Full examples: https://raschka-research-group.github.io/coral-pytorch/
Converting a Classifier into a CORAL Model in 4 Lines of Code

```python
class NeuralNetwork(torch.nn.Module):
    def __init__(self, input_size, hidden_units, num_classes):
        super().__init__()

        # ... define hidden layers ...
        output_layer = torch.nn.Linear(hidden_units[-1], num_classes)

        all_layers.append(output_layer)
        self.model = torch.nn.Sequential(*all_layers)

    def forward(self, x):
        x = self.model(x)
        return x

from coral_pytorch.layers import CoralLayer

output_layer = CoralLayer(size_in=hidden_units[-1], num_classes=num_classes)
```

PyTorch Lightning
Full examples:
https://raschka-research-group.github.io/coral-pytorch/
Converting a Classifier into a **CORAL** Model in 4 Lines of Code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

    def _shared_forward_step(self, batch, batch_idx):
        features, true_labels = batch
        logits = self(features)

        loss = torch.nn.functional.cross_entropy(logits, true_labels)
        predicted_labels = torch.argmax(logits, dim=1)
        return loss, predicted_labels

from coral_pytorch.losses import coral_loss
from coral_pytorch.dataset import levels_from_labelbatch
from coral_pytorch.dataset import proba_to_label

levels = levels_from_labelbatch(
    true_labels, num_classes=self.model.num_classes)
loss = coral_loss(logits, levels)
```

Full examples: [https://raschka-research-group.github.io/coral-pytorch/](https://raschka-research-group.github.io/coral-pytorch/)
Converting a Classifier into a **CORAL** Model in 4 Lines of Code

```python
import pytorch_lightning as pl

class LightningMLP(pl.LightningModule):
    def __init__(self, model):
        super().__init__()

    def __shared_forward_step__(self, batch, batch_idx):
        features, true_labels = batch

        logits = self(features)

        loss = torch.nn.functional.cross_entropy(logits, true_labels)

        predicted_labels = torch.argmax(logits, dim=1)

        return loss, predicted_labels
```

```python
from coral_pytorch.losses import coral_loss
from coral_pytorch.dataset import levels_from_labelbatch
from coral_pytorch.dataset import proba_to_label

levels = levels_from_labelbatch(
    true_labels, num_classes=self.model.num_classes)
loss = coral_loss(logits, levels)

predicted_labels = proba_to_label(torch.sigmoid(logits))
```

PyTorch Lightning
Full examples:
https://raschka-research-group.github.io/coral-pytorch/
## CORAL Performance

Table 1. Age prediction errors on the test sets. All models are based on the ResNet-34 architecture.

<table>
<thead>
<tr>
<th>Method</th>
<th>Random Seed</th>
<th>MOPRH-2</th>
<th>AFAD</th>
<th>CACD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MAE</td>
<td>RMSE</td>
<td>MAE</td>
</tr>
<tr>
<td>CE-CNN</td>
<td>0</td>
<td>3.26</td>
<td>4.62</td>
<td>3.58</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3.36</td>
<td>4.77</td>
<td>3.58</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.39</td>
<td>4.84</td>
<td>3.62</td>
</tr>
<tr>
<td>Avg ± SD</td>
<td></td>
<td>3.34 ± 0.07</td>
<td>4.74 ± 0.11</td>
<td>3.60 ± 0.02</td>
</tr>
<tr>
<td>OR-CNN (Niu et al., 2016)</td>
<td>0</td>
<td>2.87</td>
<td>4.08</td>
<td>3.56</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2.81</td>
<td>3.97</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.82</td>
<td>3.87</td>
<td>3.50</td>
</tr>
<tr>
<td>Avg ± SD</td>
<td></td>
<td>2.83 ± 0.03</td>
<td>3.97 ± 0.11</td>
<td>3.51 ± 0.04</td>
</tr>
<tr>
<td>CORAL-CNN (ours)</td>
<td>0</td>
<td>2.66</td>
<td>3.69</td>
<td>3.42</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2.64</td>
<td>3.64</td>
<td>3.51</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.62</td>
<td>3.62</td>
<td>3.48</td>
</tr>
<tr>
<td>Avg ± SD</td>
<td></td>
<td>2.64 ± 0.02</td>
<td>3.65 ± 0.04</td>
<td>3.47 ± 0.05</td>
</tr>
</tbody>
</table>

Fig. 3. Graphs of the predicted probabilities for each binary classifier task on four different examples from the MORPH-2 test dataset. In all cases, OR-CNN suffers from one or more inconsistencies (indicated by arrows) in contrast to CORAL-CNN.
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation


Cao, Mirjalili, Raschka (2020)
In this section, we describe our proposed consistent rank logits model. The rank index is an integer in the range \( 1 \) to \( K \). The indicator function \( \mathbb{1}_x \) is 1 if \( x \) is true and 0 otherwise. Using the extended binary classification tasks that guarantee that the binary tasks produce consistent predictions. To achieve rank-monotonicity and guarantee binary tasks, we train a single CNN with \( K - 1 \) binary classifiers in the output layer, which is illustrated in Figure 2.

Given a training dataset \( \{x_i, y_i\}_{i=1}^n \), we consider a simplified version, that is, the linear form when computing the cost function:

\[
\hat{y}_i = \text{softmax}(W x_i + b)
\]

where \( \hat{y}_i \) is a vector of size \( K \), \( W \) is a \( K \times D \) weight matrix, \( b \) is a \( K \)-dimensional bias vector, \( D \) is the input dimension, and \( \text{softmax}(\cdot) \) is the softmax function.

To provide further intuition for the weight sharing requirement, we may consider ordering information. In ordinal regression, where the cost function is application-specific and defined by the user, for instance, the linear form is commonly defined by

\[
\hat{y}_i = \text{softmax}(W x_i + b)
\]

The weight sharing requirement ensures that the model learns consistent predictions across different ranks.

The loss function of the CORAL framework is given by

\[
\text{loss}(W, b) = \sum_{i=1}^n \sum_{k=1}^{K-1} \mathbb{1}_{y_i > k} \log \hat{y}_{ik} + \mathbb{1}_{y_i = k} \log (1 - \hat{y}_{ik})
\]

for an input \( x_i \), this approach is considered as infeasible in practice due to its high training complexity (Niu et al., 2016).

While the rank label \( y_i \) is obtained via Eq. 5 and converted to the age label via Eq. 1.
Theorem 1 (Ordered bias units). By minimizing the loss function defined in Eq. 4, the optimal solution \((W^*, b^*)\) satisfies \(b_1^* \geq b_2^* \geq \ldots \geq b_{k-1}^*\).

Proof. Suppose \((W, b)\) is an optimal solution and \(b_k < b_{k+1}\) for some \(k\). Claim: replacing \(b_k\) with \(b_{k+1}\), or replacing \(b_{k+1}\) with \(b_k\), decreases 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 \(p_n(b_k) = c(g(x_n, W) + b_k)\) and

\[
\delta_n = \log(p_n(b_{k+1})) - \log(p_n(b_k)),
\]

\[
\delta'_n = \log(1 - p_n(b_{k+1})) - \log(1 - p_n(b_k)).
\]

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 the \(k\)-th task are updated. The change of loss \(L\) (Eq. 4) is given as

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

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

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

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

\[
\frac{1}{|d^{(k)}|}\Delta_1 L + \frac{1}{|d^{(k+1)}|}\Delta_2 L = - \sum_{n \in A_1} (\delta_n + \delta'_n) < 0,
\]

and know that either \(\Delta_1 L < 0\) or \(\Delta_2 L < 0\). Thus, our claim is justified. We conclude that any optimal solution \((W^*, b^*)\) that minimizes \(L\) satisfies

\[
b_1^* \geq b_2^* \geq \ldots \geq b_{k-1}^*.
\]
CORAL Rank Consistency

Table 1. Age prediction errors on the test sets. All models are based on the ResNet-34 architecture.

<table>
<thead>
<tr>
<th>Method</th>
<th>Random</th>
<th>MORPH-2</th>
<th>AFAD</th>
<th>CACD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAE</td>
<td>RMSE</td>
<td>MAE</td>
<td>RMSE</td>
</tr>
<tr>
<td>CE-CNN</td>
<td>3.26</td>
<td>4.62</td>
<td>3.58</td>
<td>5.01</td>
</tr>
<tr>
<td>(Niu et al., 2016)</td>
<td>3.39</td>
<td>4.84</td>
<td>3.58</td>
<td>5.01</td>
</tr>
<tr>
<td>OR-CNN</td>
<td>2.87</td>
<td>4.08</td>
<td>3.46</td>
<td>4.80</td>
</tr>
<tr>
<td>(ours)</td>
<td>2.66</td>
<td>3.69</td>
<td>3.42</td>
<td>4.65</td>
</tr>
</tbody>
</table>

Fig. 3. Graphs of the predicted probabilities for each binary classifier task on four different examples from the MORPH-2 test dataset. In all cases, OR-CNN suffers from one or more inconsistencies (indicated by arrows) in contrast to CORAL-CNN.
Fixing rank inconsistency introduced a limitation: 
weight-sharing constraint restricts the network's capacity
Cao, Mirjalili, Raschka (2020)
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation
Removing the weight-sharing constraint (while maintaining rank consistency) leads to even better performance

Shi, Cao, Raschka (2021)
*Deep Neural Networks for Rank-Consistent Ordinal Regression Based On Conditional Probabilities.*
3.3. Rank-consistent Ordinal Regression based on Conditional Probabilities

Given a training set \( D = \{ \mathbf{x}^{[i]}, y^{[i]} \}_{i=1}^{N} \), CORN applies a label extension to the rank labels \( y^{[i]} \) similar to CORAL, such that the resulting binary label \( y_{k}^{[i]} \in \{0, 1\} \) indicates whether \( y^{[i]} \) exceeds rank \( r_{k} \). Similar to CORAL, CORN also uses \( K - 1 \) learning tasks associated with ranks \( r_{1}, r_{2}, ..., r_{K} \) in the output layer as illustrated in Fig. 2.

However, in contrast to CORAL, CORN estimates a series of conditional probabilities using conditional training subsets (described in Section 3.4) such that the output of the \( k \)-th binary task \( f_{k}(\mathbf{x}^{[i]}) \) represents the conditional probability

\[
f_{k}(\mathbf{x}^{[i]}) = \hat{P}(y^{[i]} > r_{k} | y^{[i]} > r_{k-1}),
\]

where the events are nested: \( \left\{ y^{[i]} > r_{k} \right\} \subseteq \left\{ y^{[i]} > r_{k-1} \right\} \).

The transformed, unconditional probabilities can then be computed by applying the chain rule for probabilities to the model outputs:

\[
\hat{P}(y^{[i]} > r_{k}) = \prod_{j=1}^{k} f_{j}(\mathbf{x}^{[i]}).
\]

Since \( \forall j, 0 \leq f_{j}(\mathbf{x}^{[i]}) \leq 1 \), we have

\[
\hat{P}(y^{[i]} > r_{1}) \geq \hat{P}(y^{[i]} > r_{2}) \geq ... \geq \hat{P}(y^{[i]} > r_{K-1}),
\]

which guarantees rank consistency among the \( K - 1 \) binary tasks.
3.4. Conditional Training Subsets

Our model aims to estimate $f_1 (\mathbf{x}^{[i]})$ and the conditional probabilities $f_2 (\mathbf{x}^{[i]}), \ldots, f_{K-1} (\mathbf{x}^{[i]})$. Estimating $f_1 (\mathbf{x}^{[i]})$ is a classic binary classification task under the extended binary classification framework with the binary labels $y_1^{[i]}$. To estimate the conditional probabilities such as $\hat{P} (y^{[i]} > r_2 | y^{[i]} > r_1)$, we focus only on the subset of the training data where $y^{[i]} > r_1$. As a result, when we minimize the binary cross-entropy loss on these conditional subsets, for each binary task, the estimated output probability has a proper conditional probability interpretation.

In order to model the conditional probabilities in Eq. 3, we construct conditional training subsets for training, which are used in the loss function (Section 3.5) that is minimized via backpropagation. The conditional training subsets are obtained from the original training set as follows:

$$
S_1 : \text{all } \left\{ \left( \mathbf{x}^{[i]}, y^{[i]} \right) \right\}, \text{ for } i \in \{1, \ldots, N\},
$$
$$S_2 : \left\{ \left( \mathbf{x}^{[i]}, y^{[i]} \right) | y^{[i]} > r_1 \right\},$$
$$\ldots$$
$$S_{K-1} : \left\{ \left( \mathbf{x}^{[i]}, y^{[i]} \right) | y^{[i]} > r_{k-2} \right\},$$

where $N = |S_1| \geq |S_2| \geq \ldots \geq |S_{K-1}|$, and $|S_k|$ denotes the size of $S_k$. Note that the labels $y^{[i]}$ are subject to the binary label extension as described in Section 3.3. Each conditional training subset $S_k$ is used for training the conditional probability prediction $\hat{P} (y^{[i]} > r_k | y^{[i]} > r_{k-1})$ for $k \geq 2$. 
3.5. Loss Function

Let $f_j(x^{[i]})$ denote the predicted value of the $j$-th node in the output layer of the network (Fig. 2), and let $|S_j|$ denote the size of the $j$-th conditional training set. To train a CORN neural network using backpropagation, we minimize the following loss function:

$$L(X, y) =$$

$$-\frac{1}{\sum_{j=1}^{K-1} |S_j|} \sum_{j=1}^{K-1} \sum_{i=1}^{|S_j|} \left[ \log \left( f_j(x^{[i]}) \right) \cdot \mathbb{1} \{ y^{[i]} > r_j \} \right]$$

$$+ \log \left( 1 - f_j(x^{[i]}) \right) \cdot \mathbb{1} \{ y^{[i]} \leq r_j \}.$$  \hspace{1cm} (5)

We note that in $f_j(x^{[i]})$, $x^{[i]}$ represents the $i$-th training example in $S_j$. To simplify the notation, we omit an additional index $j$ to distinguish between $x^{[i]}$ in different conditional training sets.

To improve the numerical stability of the loss gradients during training, we implement the following alternative formulation of the loss, where $Z$ are the net inputs of the last layer (aka logits), as shown in Fig. 2, and $\log (\sigma(z^{[i]})) = \log (f_j(x^{[i]}))$:

$$L(Z, y) =$$

$$-\frac{1}{\sum_{j=1}^{K-1} |S_j|} \sum_{j=1}^{K-1} \sum_{i=1}^{|S_j|} \left[ \log \left( \sigma(z^{[i]}) \right) \cdot \mathbb{1} \{ y^{[i]} > r_j \} \right]$$

$$+ \left( \log \left( \sigma(z^{[i]}) \right) - z^{[i]} \right) \cdot \mathbb{1} \{ y^{[i]} \leq r_j \}.$$  \hspace{1cm} (6)
CORN Architecture

Weight parameters of the last layer (bias units $b_1 ... b_{k-1}$ not shown)

$$w_1 = [w_{1,1} \ w_{1,2} \ ... \ w_{1,m}]$$

Logistic sigmoid function

$$f_1(x^{[i]}) = \sigma(z_1^{[i]}) \quad \text{where} \quad z_1^{[i]} = a_1^{[i]} w_1^T + b_1$$

$$f_2(x^{[i]}) = \sigma(z_2^{[i]}) \quad \text{where} \quad z_2^{[i]} = a_1^{[i]} w_2^T + b_2$$

$$f_{k-1}(x^{[i]}) = \sigma(z_{k-1}^{[i]}) \quad \text{where} \quad z_{k-1}^{[i]} = a_1^{[i]} w_{k-1}^T + b_{k-1}$$

Outputs

$$\sigma(z_{k-1}^{[i]}) = \hat{p}(y^{[i]} > r_{k-1} \mid y^{[i]} > r_{k-2})$$

Predicted rank

$$q^{[i]} = 1 + \sum_{k=1}^{K-1} \mathbb{1}\{\hat{p}(y^{[i]} > r_k) > 0.5\}$$

where

$$\hat{p}(y^{[i]} > r_k) = \hat{p}(y^{[i]} > r_1) \cdot \hat{p}(y^{[i]} > r_2 \mid y^{[i]} > r_1) \cdot \hat{p}(y^{[i]} > r_3 \mid y^{[i]} > r_2) \cdot \hat{p}(y^{[i]} > r_{k-1} \mid y^{[i]} > r_{k-2})$$

Fig. 1. Illustration of the difference between rank-consistent and rank-inconsistent methods.

Fig. 2. Outline of the neural network architecture used for CORN.
Table 1. Prediction errors on the test sets. Best results are highlighted in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Seed</th>
<th>MORPH-2 (Balanced)</th>
<th>AFAD (Balanced)</th>
<th>AES</th>
<th>FIREMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MAE</td>
<td>RMSE</td>
<td>MAE</td>
<td>RMSE</td>
</tr>
<tr>
<td>CE-NN</td>
<td>0</td>
<td>3.81</td>
<td>5.19</td>
<td>3.31</td>
<td>4.27</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3.60</td>
<td>4.8</td>
<td>3.28</td>
<td>4.19</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.61</td>
<td>4.84</td>
<td>3.32</td>
<td>4.22</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.85</td>
<td>5.21</td>
<td>3.24</td>
<td>4.15</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.80</td>
<td>5.14</td>
<td>3.24</td>
<td>4.15</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>3.73</td>
<td>5.04</td>
<td>3.28</td>
<td>4.19</td>
</tr>
<tr>
<td>OR-NN</td>
<td>0</td>
<td>3.21</td>
<td>4.25</td>
<td>2.81</td>
<td>3.45</td>
</tr>
<tr>
<td>[11]</td>
<td>1</td>
<td>3.16</td>
<td>4.25</td>
<td>2.87</td>
<td>3.54</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.16</td>
<td>4.31</td>
<td>2.82</td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.98</td>
<td>4.05</td>
<td>2.89</td>
<td>3.49</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.13</td>
<td>4.27</td>
<td>2.86</td>
<td>3.45</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>3.13</td>
<td>4.23</td>
<td>2.85</td>
<td>3.48</td>
</tr>
<tr>
<td>CORAL</td>
<td>0</td>
<td>2.94</td>
<td>3.98</td>
<td>2.95</td>
<td>3.60</td>
</tr>
<tr>
<td>[1]</td>
<td>1</td>
<td>2.97</td>
<td>4.03</td>
<td>2.99</td>
<td>3.69</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3.01</td>
<td>3.98</td>
<td>2.98</td>
<td>3.70</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2.98</td>
<td>4.01</td>
<td>3.00</td>
<td>3.78</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.03</td>
<td>4.06</td>
<td>3.04</td>
<td>3.75</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>2.99</td>
<td>4.01</td>
<td>2.99</td>
<td>3.70</td>
</tr>
<tr>
<td>CORN</td>
<td>0</td>
<td>2.98</td>
<td>4.00</td>
<td>2.80</td>
<td>3.45</td>
</tr>
<tr>
<td>(ours)</td>
<td>1</td>
<td>2.99</td>
<td>4.02</td>
<td>2.81</td>
<td>3.44</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2.97</td>
<td>3.97</td>
<td>2.84</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3.00</td>
<td>4.06</td>
<td>2.80</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.95</td>
<td>3.92</td>
<td>2.79</td>
<td>3.45</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>2.98</td>
<td>4.01</td>
<td>2.81</td>
<td>3.46</td>
</tr>
</tbody>
</table>
### Table S1. Prediction errors on the test sets. Best results are highlighted in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>Seed</th>
<th>TripAdvisor (Balanced)</th>
<th>Coursera (Balanced)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MAE</td>
<td>RMSE</td>
</tr>
<tr>
<td>CE-RNN</td>
<td>0</td>
<td>1.13</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.04</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.05</td>
<td>1.54</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.23</td>
<td>1.81</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.03</td>
<td>1.52</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>1.10 ± 0.09</td>
<td>1.59 ± 0.12</td>
</tr>
<tr>
<td>OR-RNN [11]</td>
<td>0</td>
<td>1.06</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.09</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.11</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.23</td>
<td>1.52</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.07</td>
<td>1.40</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>1.11 ± 0.07</td>
<td>1.50 ± 0.06</td>
</tr>
<tr>
<td>CORAL [1]</td>
<td>0</td>
<td>1.15</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.14</td>
<td>1.49</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.16</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.19</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.13</td>
<td>1.47</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>1.15 ± 0.02</td>
<td>1.48 ± 0.06</td>
</tr>
<tr>
<td>CORN (ours)</td>
<td>0</td>
<td>1.09</td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1.09</td>
<td>1.53</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.01</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.12</td>
<td>1.51</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.03</td>
<td>1.46</td>
</tr>
<tr>
<td>AVG±SD</td>
<td></td>
<td>1.07 ± 0.05</td>
<td>1.50 ± 0.04</td>
</tr>
</tbody>
</table>
Assume 3 training examples $x^{(i)}$, $x^{(j)}$, and $x^{(k)}$ with the following 3 rank labels:

$$
y = \begin{cases} 
    y^{(i)} = 3 \\
    y^{(j)} = 4 
\end{cases}
$$

3 training examples as input

Train task 1

$y = \begin{cases} 
    y^{(i)} = 1 \\
    y^{(j)} = 3 \\
    y^{(k)} = 4 
\end{cases}$

$y_1 = \begin{cases} 
    y_1^{(i)} = 0 \\
    y_1^{(j)} = 1 \\
    y_1^{(k)} = 1 
\end{cases}$

$y^{(i)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(i)})}{\rightarrow} \overset{L_1 = \log \left(1 - f_1(x^{(i)})\right)}{\text{Loss of first task}}$ $y^{(j)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(j)})}{\rightarrow} \overset{L_1 = \log \left(f_1(x^{(j)})\right)}{\text{Loss of first task}}$ $y^{(k)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(k)})}{\rightarrow} \overset{L_1 = \log \left(f_1(x^{(k)})\right)}{\text{Loss of first task}}$

Train task 2

$y = \begin{cases} 
    y^{(i)} = 1 \\
    y^{(j)} = 3 \\
    y^{(k)} = 4 
\end{cases}$

$y_2 = \begin{cases} 
    y_2^{(i)} = 1 \\
    y_2^{(j)} = 1 
\end{cases}$

$y^{(i)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(i)})}{\rightarrow} \overset{L_2 = \log \left(f_1(x^{(i)})\right) + \log \left(f_1(x^{(j)})\right)}{\text{Loss of first task}}$ $y^{(j)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(j)})}{\rightarrow} \overset{L_2 = \log \left(f_1(x^{(j)})\right) + \log \left(f_1(x^{(k)})\right)}{\text{Loss of first task}}$ $y^{(k)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(k)})}{\rightarrow} \overset{L_2 = \log \left(f_1(x^{(k)})\right) + \log \left(f_1(x^{(i)})\right)}{\text{Loss of first task}}$

Train task 3

$y = \begin{cases} 
    y^{(i)} = 1 \\
    y^{(j)} = 3 \\
    y^{(k)} = 4 
\end{cases}$

$y_3 = \begin{cases} 
    y_3^{(i)} = 0 \\
    y_3^{(j)} = 1 \\
    y_3^{(k)} = 1 
\end{cases}$

$y^{(i)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(i)})}{\rightarrow} \overset{L_3 = \log \left(1 - f_1(x^{(i)})\right) + \log \left(f_1(x^{(j)})\right)}{\text{Loss of first task}}$ $y^{(j)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(j)})}{\rightarrow} \overset{L_3 = \log \left(1 - f_1(x^{(j)})\right) + \log \left(f_1(x^{(k)})\right)}{\text{Loss of first task}}$ $y^{(k)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(k)})}{\rightarrow} \overset{L_3 = \log \left(1 - f_1(x^{(k)})\right) + \log \left(f_1(x^{(i)})\right)}{\text{Loss of first task}}$

Train task 4

$y = \begin{cases} 
    y^{(i)} = 1 \\
    y^{(j)} = 3 \\
    y^{(k)} = 4 
\end{cases}$

$y_4 = \begin{cases} 
    y_4^{(i)} = 0 \\
    y_4^{(j)} = 1 \\
    y_4^{(k)} = 1 
\end{cases}$

$y^{(i)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(i)})}{\rightarrow} \overset{L_4 = \log \left(1 - f_1(x^{(i)})\right) + \log \left(f_1(x^{(j)})\right)}{\text{Loss of first task}}$ $y^{(j)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(j)})}{\rightarrow} \overset{L_4 = \log \left(1 - f_1(x^{(j)})\right) + \log \left(f_1(x^{(k)})\right)}{\text{Loss of first task}}$ $y^{(k)} \overset{\text{binarize}}{\rightarrow} \text{Neural network} \overset{f(x^{(k)})}{\rightarrow} \overset{L_4 = \log \left(1 - f_1(x^{(k)})\right) + \log \left(f_1(x^{(i)})\right)}{\text{Loss of first task}}$

Overall loss: $L(X, y) = \frac{1}{\sum_{i=1}^{3} y_i} \sum_{i=1}^{3} L_i = \frac{1}{3 + 2 + 2 + 1} L_1 + L_2 + L_3 + L_4$