Customer Ratings, Letter Grades, and Other Rankings

Using Deep Learning When Class Labels Have A Natural Order

Sebastian Raschka

Lead AI Educator @ Grid.ai

Asst. Prof. of Statistics @ University of Wisconsin

@rasbt

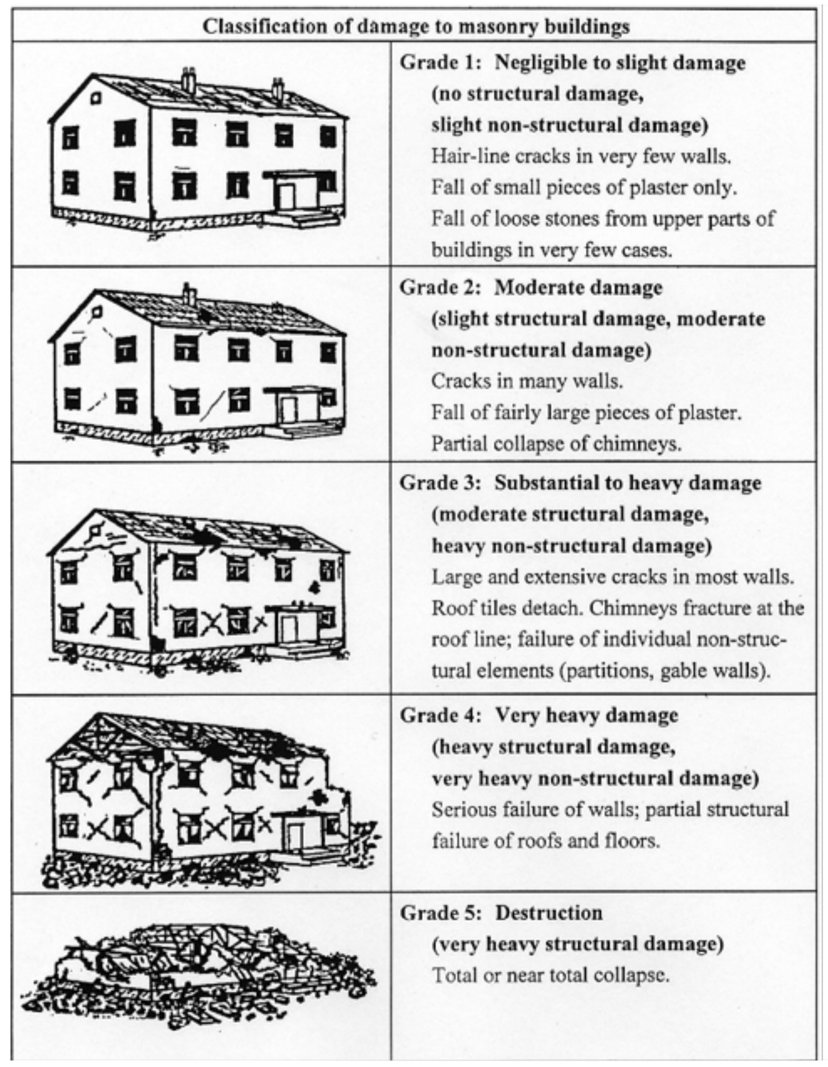
sebastian@grid.ai

https://sebastianraschka.com

RE • WORK
Deep Learning Summit
17 Feb, 2022

Many Real-World Predictions Problems Have Ordered Labels

Damage assessment



https://emergency.copernicus.eu/mapping/ems/damage-assessment

Plant disease

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

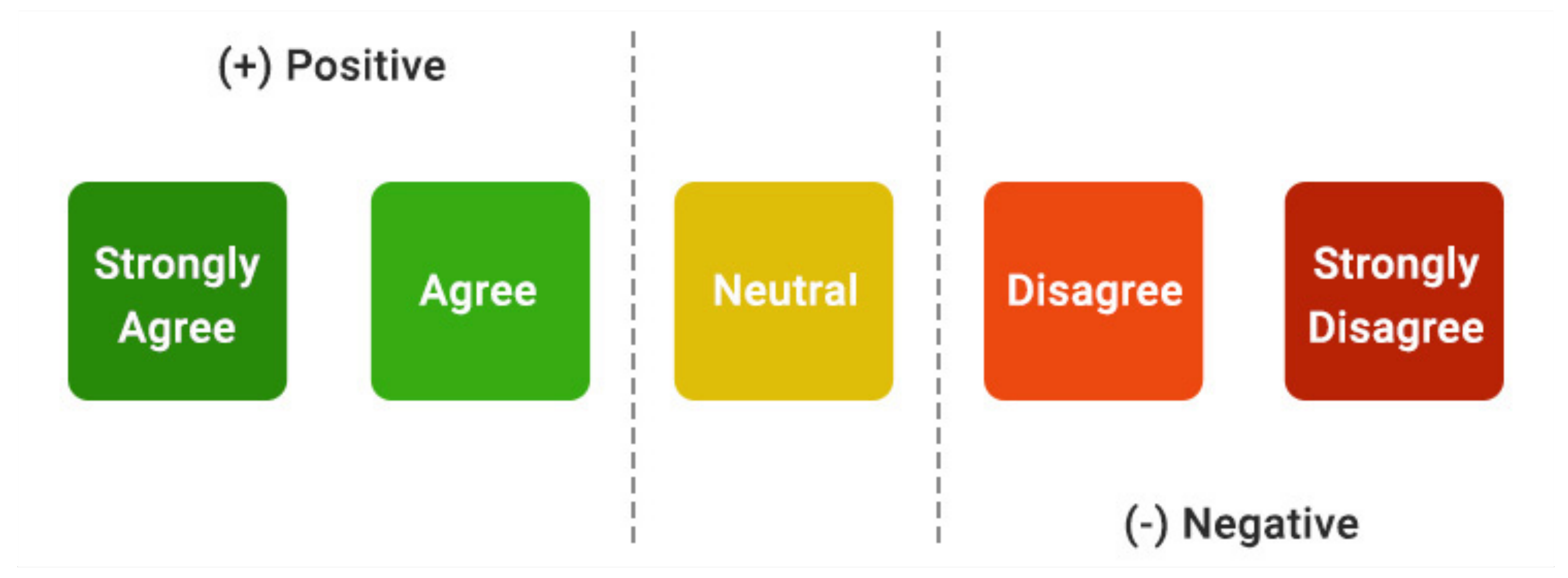
Islam, M. U., et al. "Screening of potato germplasm against RNA viruses and their identification through ELISA." J Green Physiol Genet Genom 1 (2015): 22-31.

Credit risk rating

PASS					SPECIAL MENTION	SUB-STANDARD	DOUBTFUL	LOSS
1	2	3	4	5	6	7	8	9
Largely risk free	Minimal risk	Modest risk	Bankable	Addition-al review	Criticized	Classified	Classified	Classified

https://www.abrigo.com/blog/how-to-create-a-credit-risk-rating-system/

Likert scale for customer satisfaction



https://www.questionpro.com/blog/ordinal-scale/

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

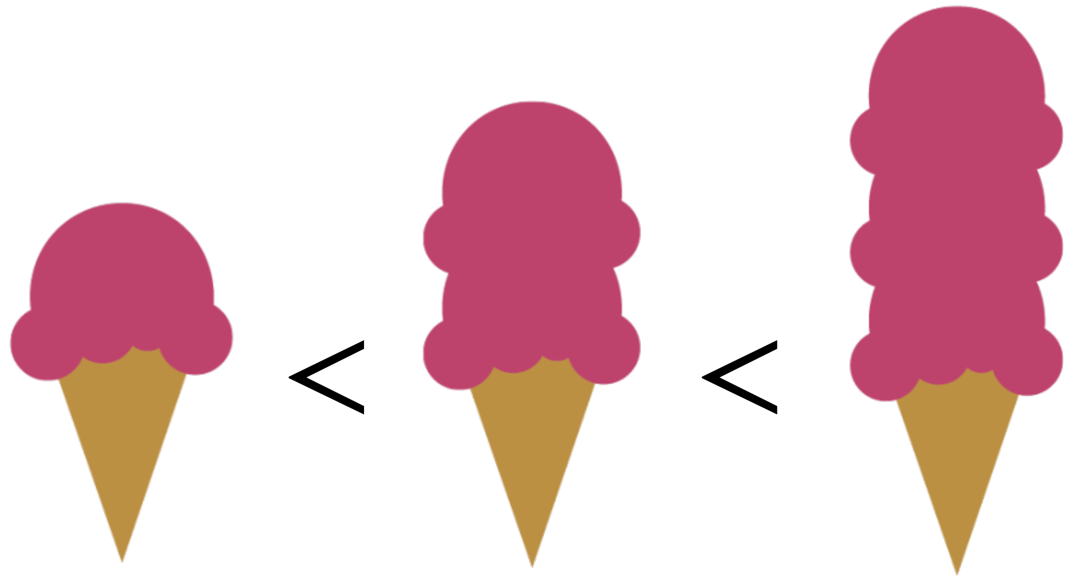
1 Setosa

2 Versicolor

3 Virginica

No ordering

Regression



1 2 3

Ordered Labels? Tell Me More!

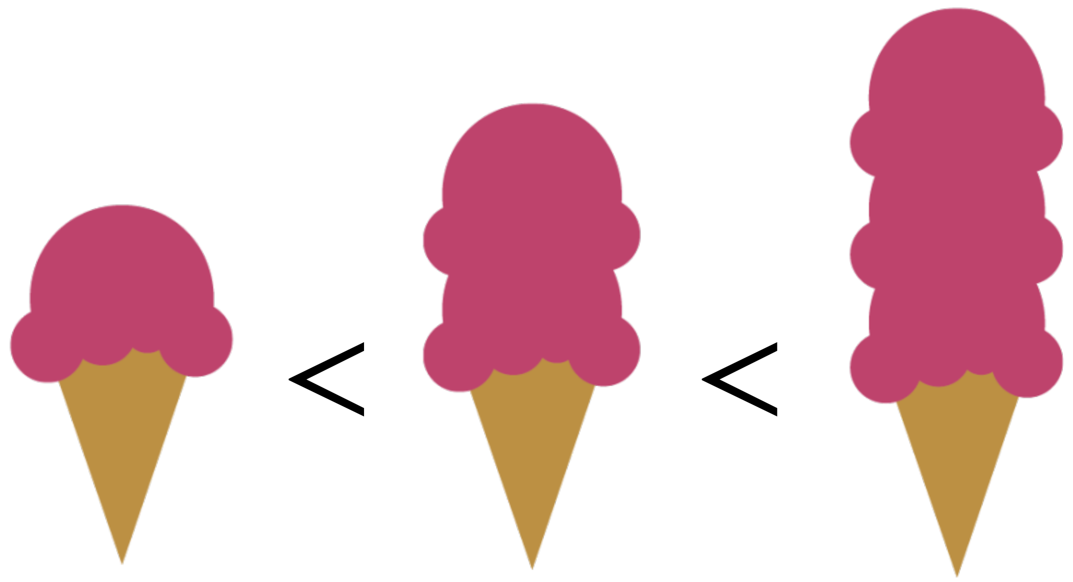
Classification



1 Setosa 2 Versicolor 3 Virginica

No ordering

Regression



1 2 3

Identical distances

Ordered Labels? Tell Me More!

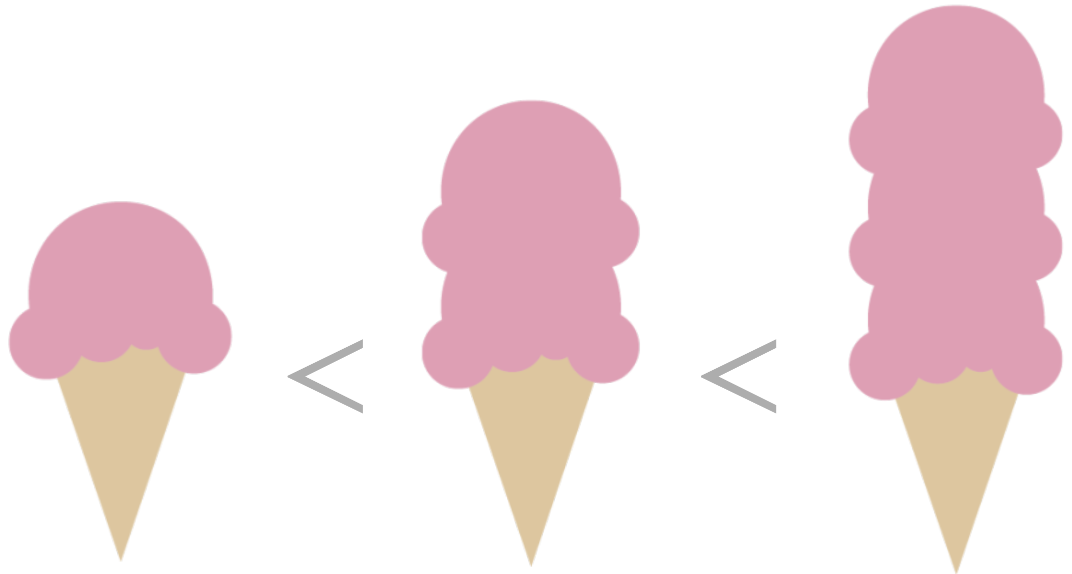
Classification

1 Setosa 2 Versicolor 3 Virginica

No ordering

Ordinal Regression / Ordinal Classification

Regression



< <

1 2 3

Identical distances

Ordered Labels? Tell Me More!

Classification

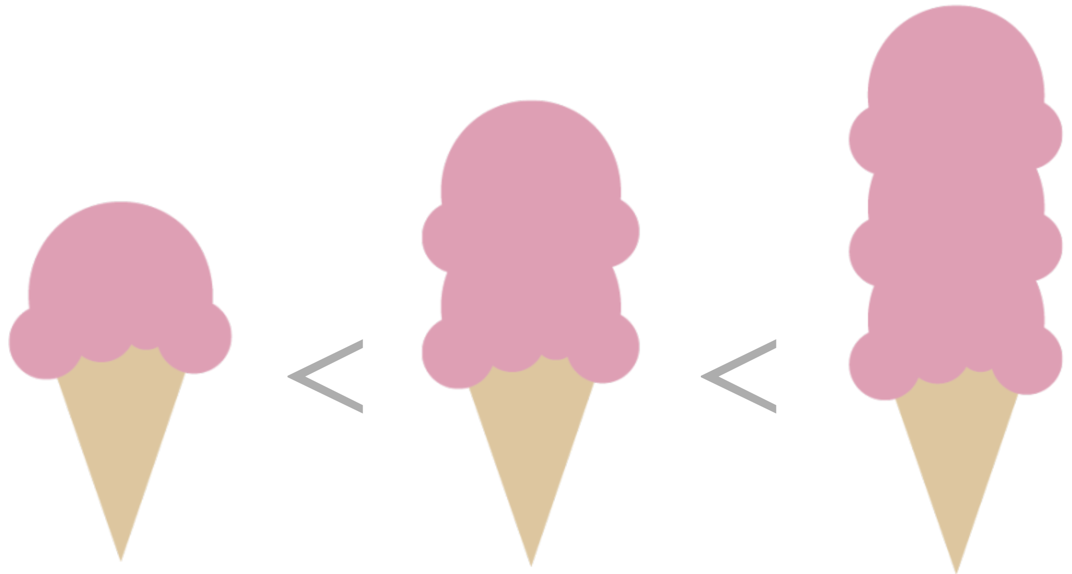
1 Setosa 2 Versicolor 3 Virginica

No ordering

Ordinal Regression / Ordinal Classification

1 2 3

Regression



1 2 3

Identical distances

Ordered Labels? Tell Me More!

Classification

1 Setosa   2 Versicolor   3 Virginica

No ordering

Ordinal Regression / Ordinal Classification

≺


≺


1    2    3 

Regression

 <  < 

1   2   3

Identical distances

Ordered Labels? Tell Me More!

Classification

1 Setosa 2 Versicolor 3 Virginica

No ordering

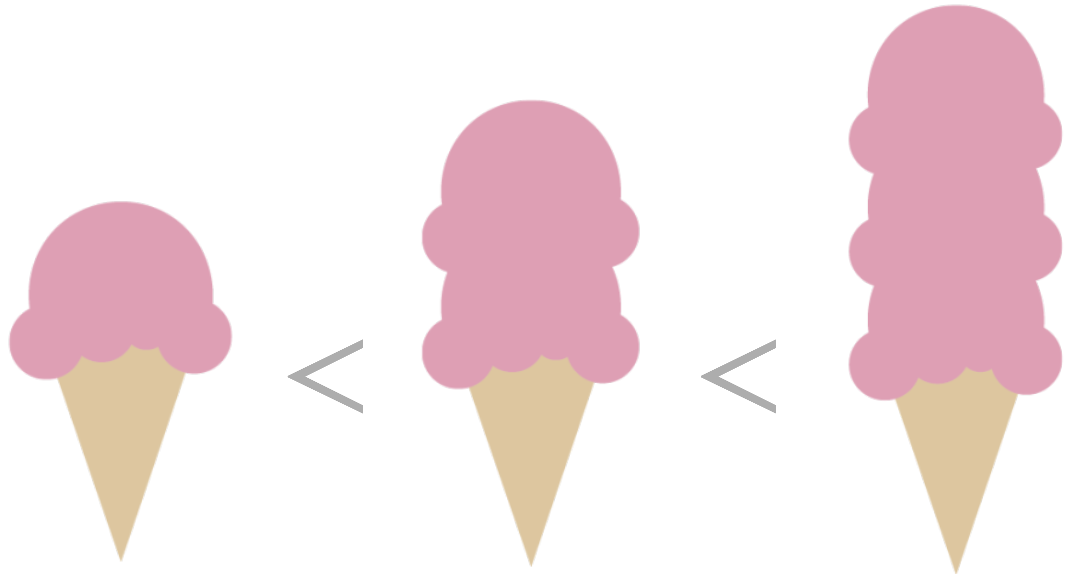
Ordinal Regression / Ordinal Classification

1 2 3

Class labels

- but with order info
- and arbitrary distances

Regression



1 2 3

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

1 

Wrong prediction

2 

Wrong prediction

3 

It is not ideal because all wrong predictions look equally wrong to a classifier

Assume this is the true label

1

Wrong prediction

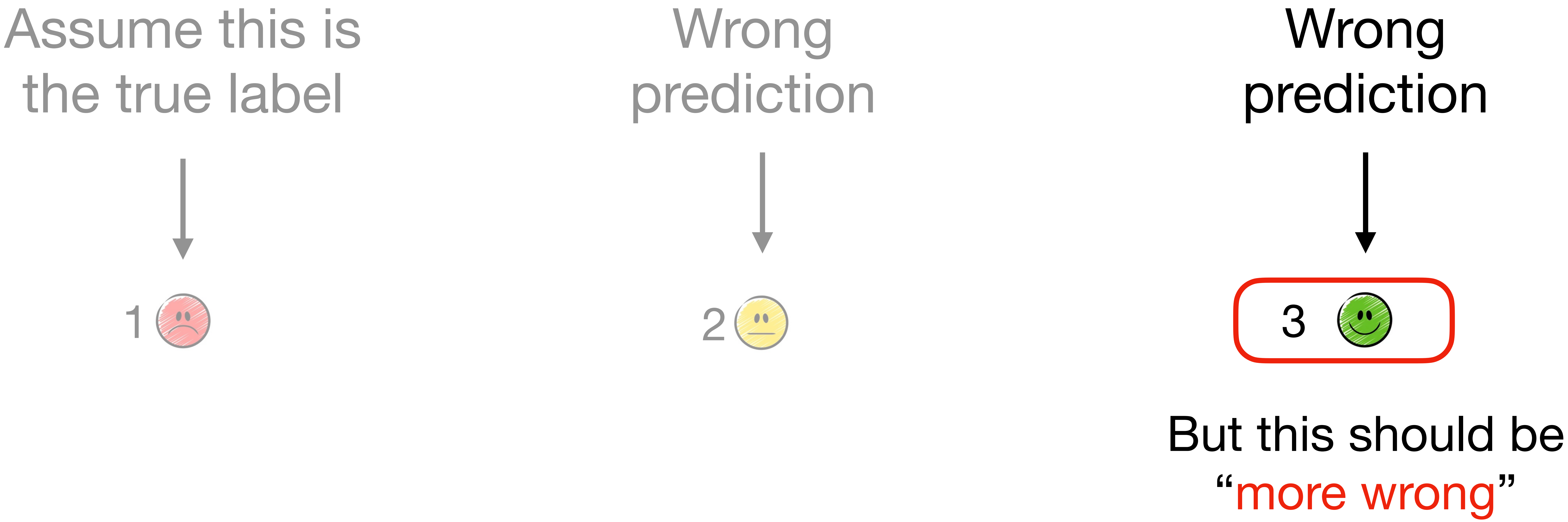
2

Wrong prediction

3

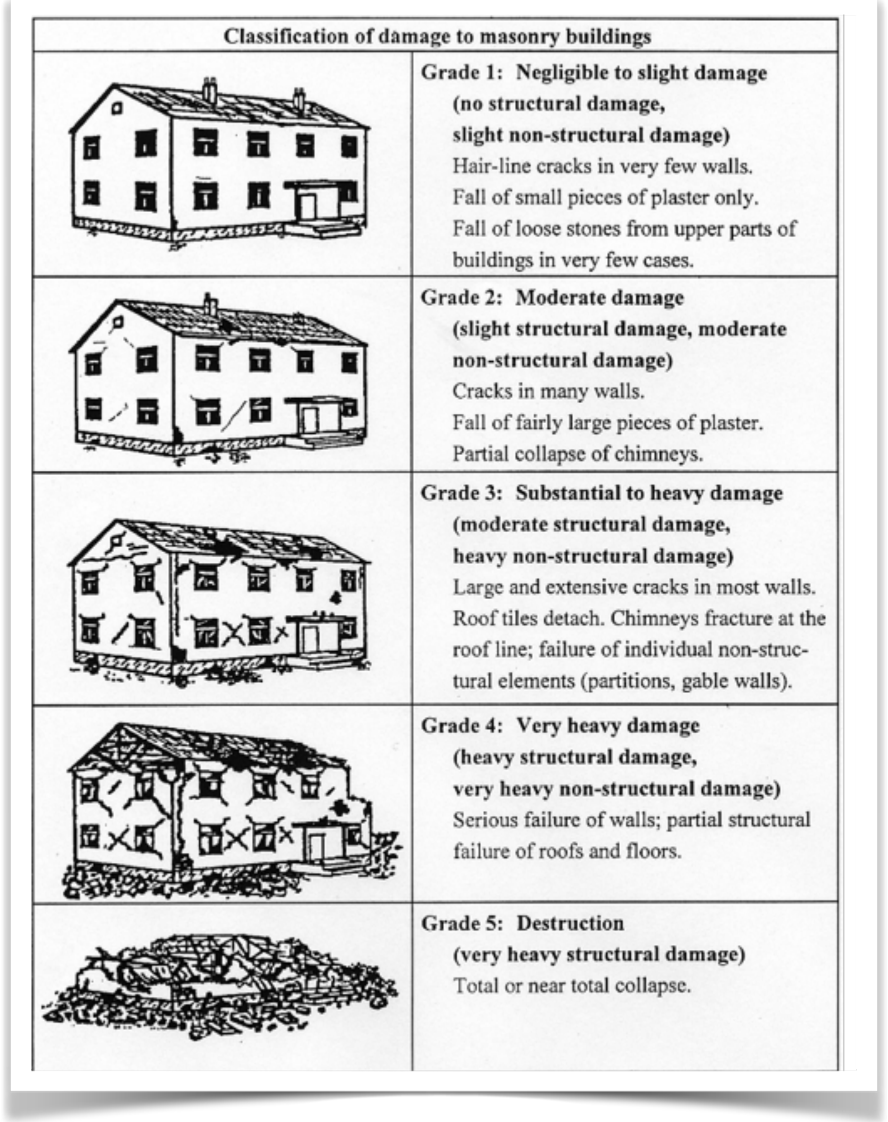
Treated equally if we compute the loss in a regular classifier

It is not ideal because all wrong predictions look equally wrong to a classifier



Many Real-World Predictions Problems Have Ordered Labels

Damage assessment


Classification of damage to masonry buildings

Grade 1: Negligible to slight damage (no structural damage, slight non-structural damage)
Hair-line cracks in very few walls.
Fall of small pieces of plaster only.
Fall of loose stones from upper parts of buildings in very few cases.

Grade 2: Moderate damage (slight structural damage, moderate non-structural damage)
Cracks in many walls.
Fall of fairly large pieces of plaster.
Partial collapse of chimneys.

Grade 3: Substantial to heavy damage (moderate structural damage, heavy non-structural damage)
Large and extensive cracks in most walls.
Roof tiles detach. Chimneys fracture at the roof line; failure of individual non-structural elements (partitions, gable walls).

Grade 4: Very heavy damage (heavy structural damage, very heavy non-structural damage)
Serious failure of walls; partial structural failure of roofs and floors.

Grade 5: Destruction (very heavy structural damage)
Total or near total collapse.

https://emergency.copernicus.eu/mapping/ems/damage-assessment

Plant disease

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

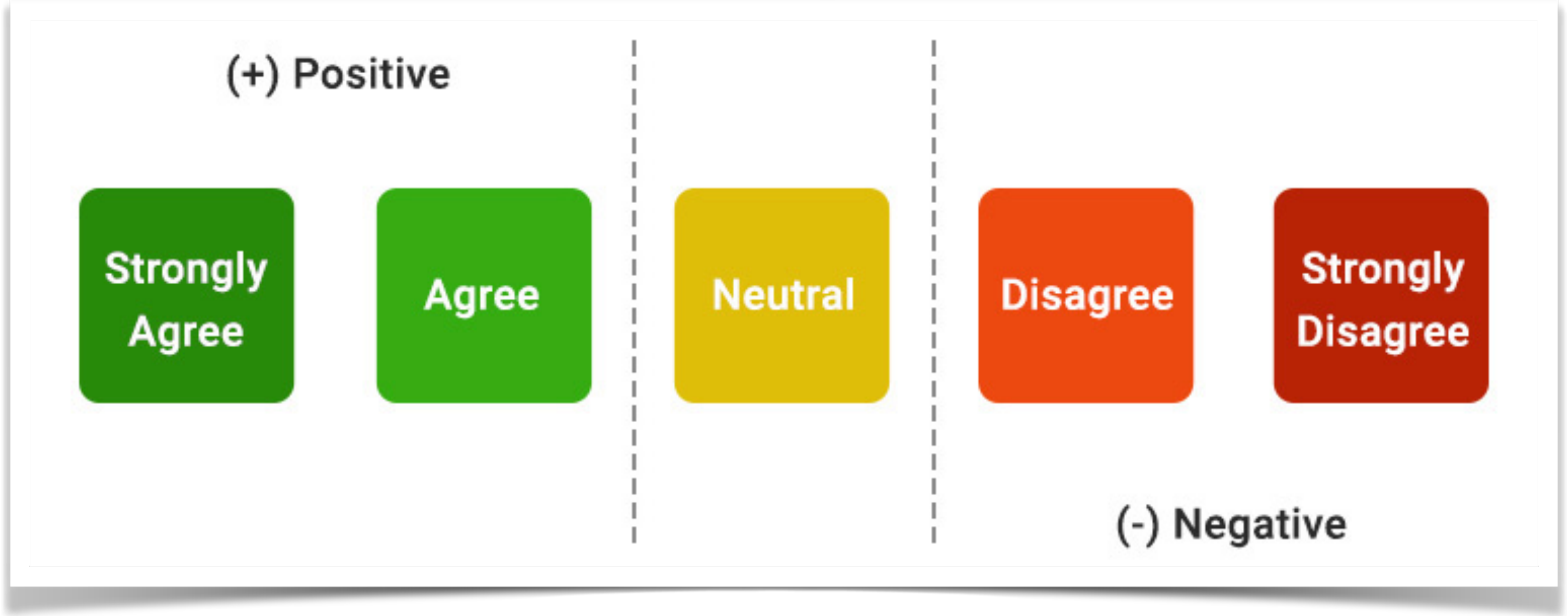
Islam, M. U., et al. "Screening of potato germplasm against RNA viruses and their identification through ELISA." J Green Physiol Genet Genom 1 (2015): 22-31.

Credit risk rating

PASS					SPECIAL MENTION	SUB-STANDARD	DOUBTFUL	LOSS
1	2	3	4	5	6	7	8	9
Largely risk free	Minimal risk	Modest risk	Bankable	Addition-al review	Criticized	Classified	Classified	Classified

https://www.abrigo.com/blog/how-to-create-a-credit-risk-rating-system/

Likert scale for customer satisfaction



https://www.questionpro.com/blog/ordinal-scale/

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)

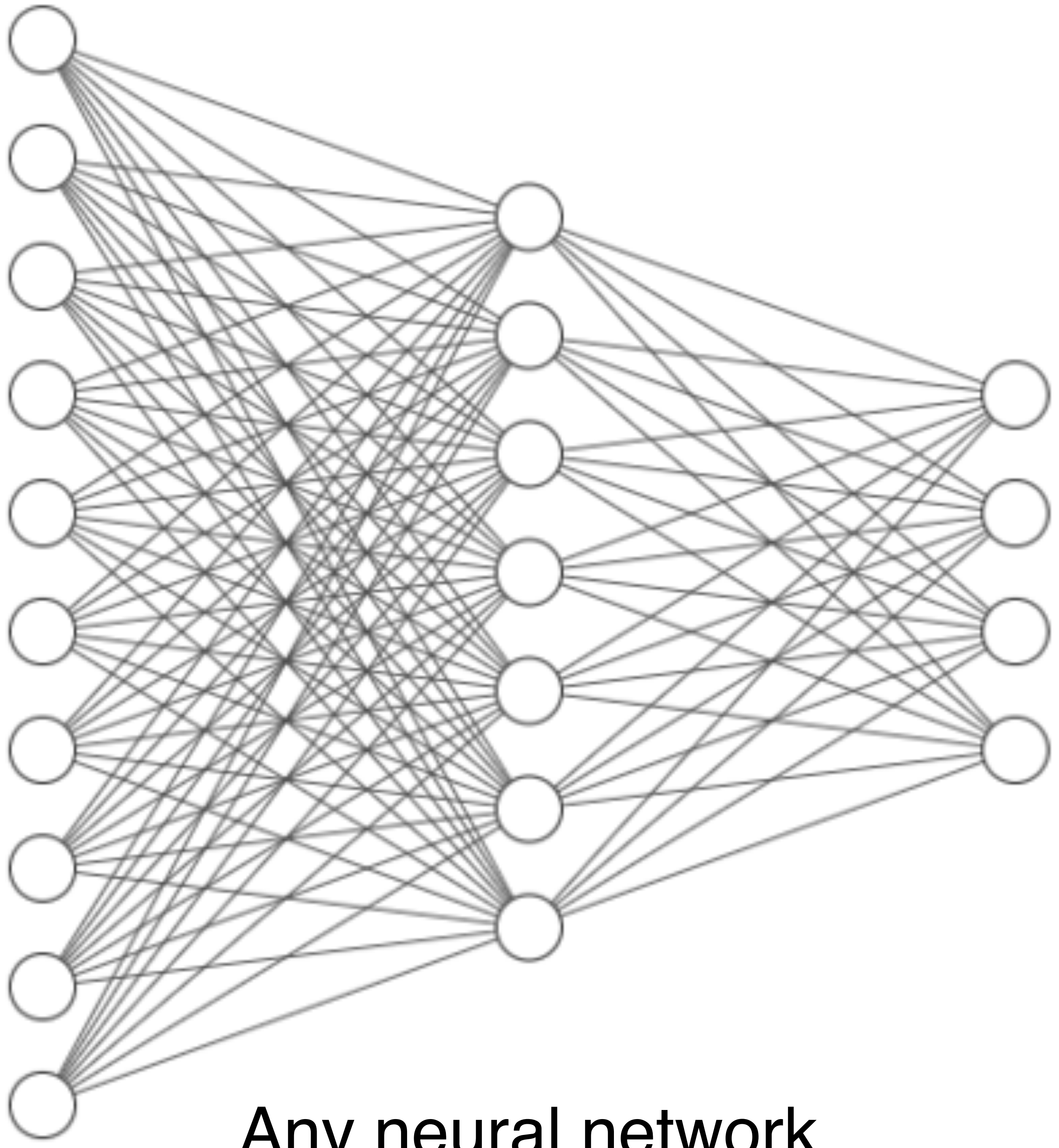
Possible labels:
Rating 1, 2, 3, 4, 5

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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



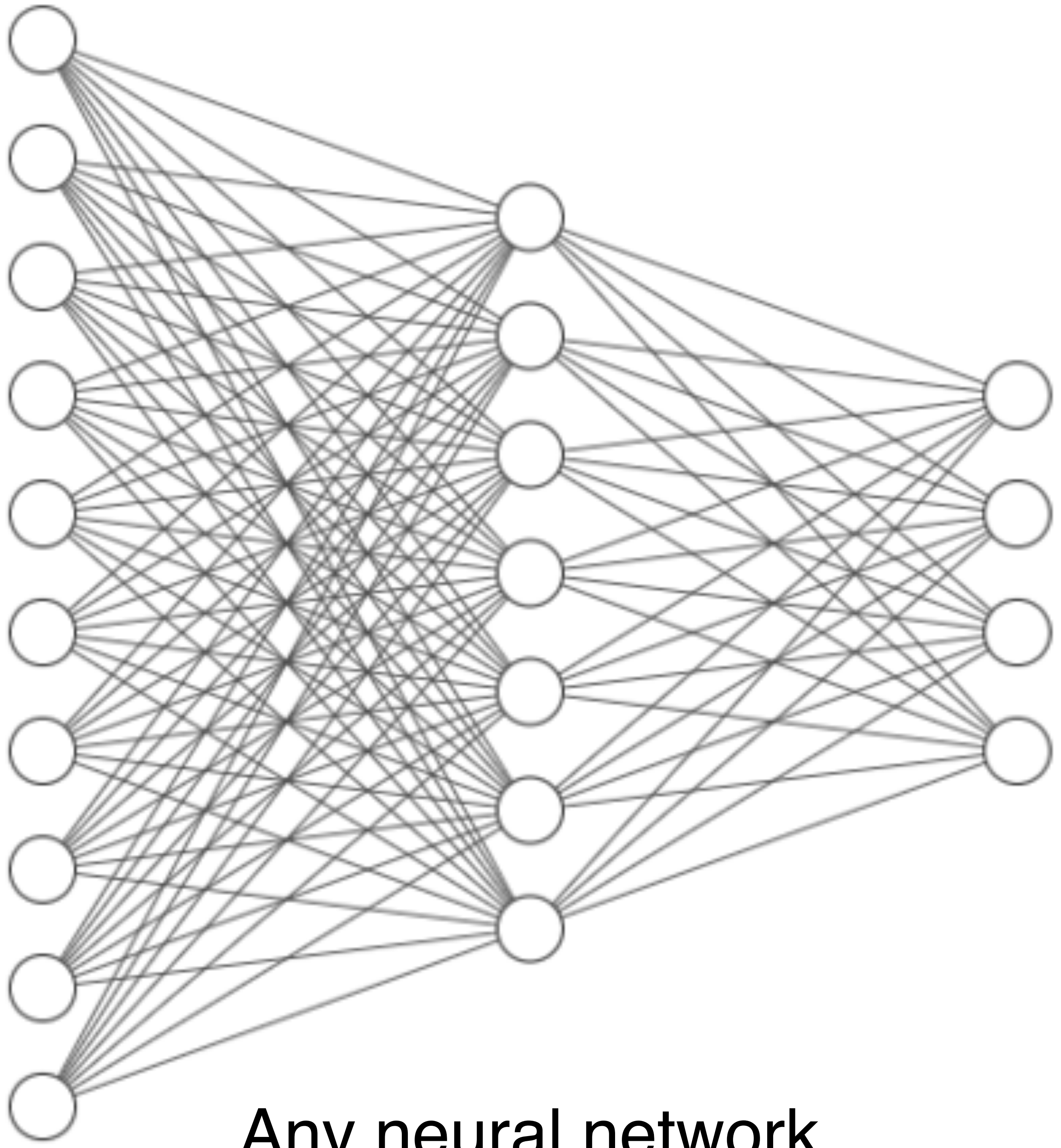
Any neural network
(CNN, RNN, MLP, ...)

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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



Any neural network
(CNN, RNN, MLP, …)

Binary classification task

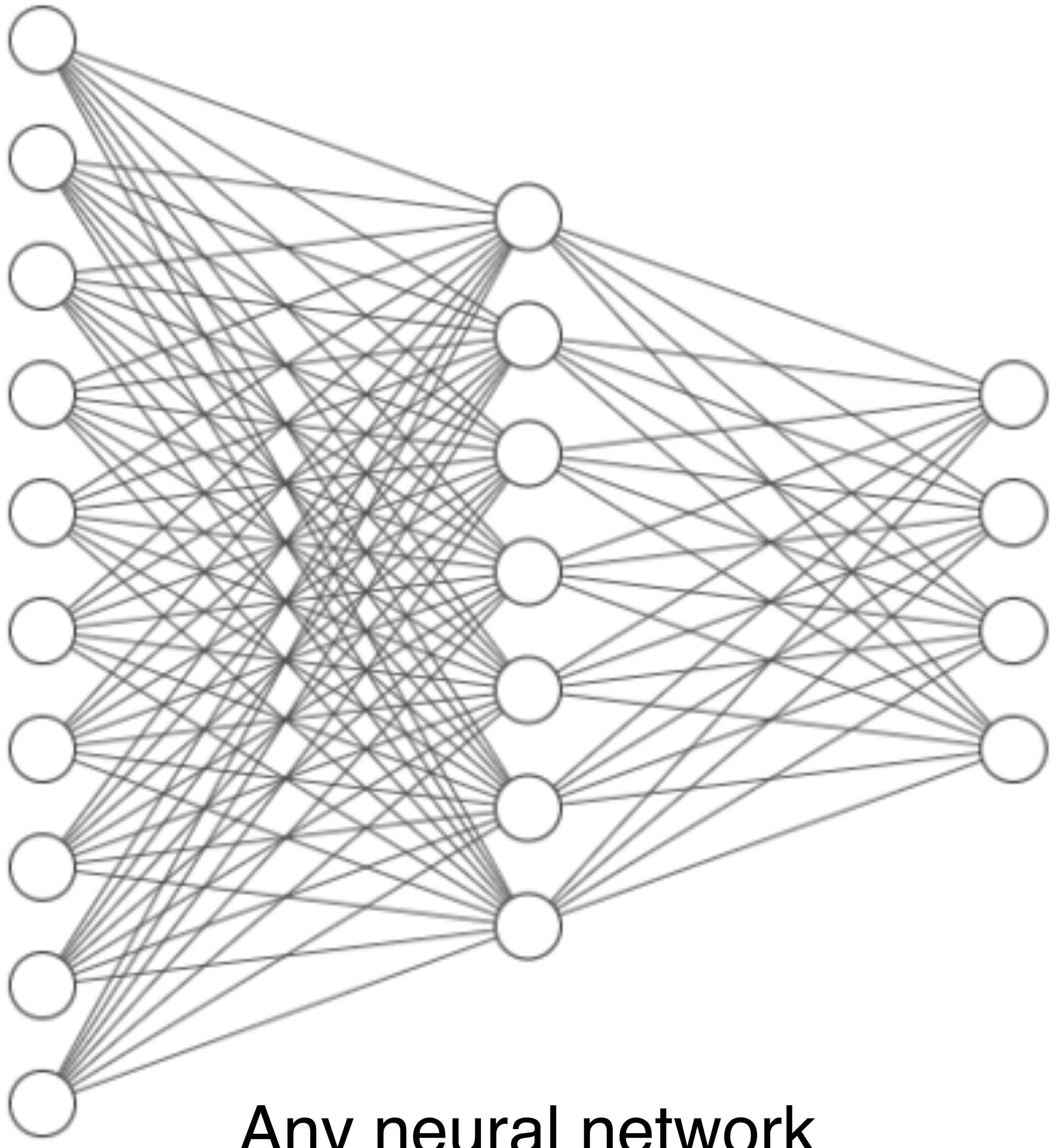
Rating > 1? → yes/no

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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



Rating > 1? → yes/no
Rating > 2? → yes/no
Rating > 3? → yes/no
Rating > 4? → yes/no

Each output node is a binary task

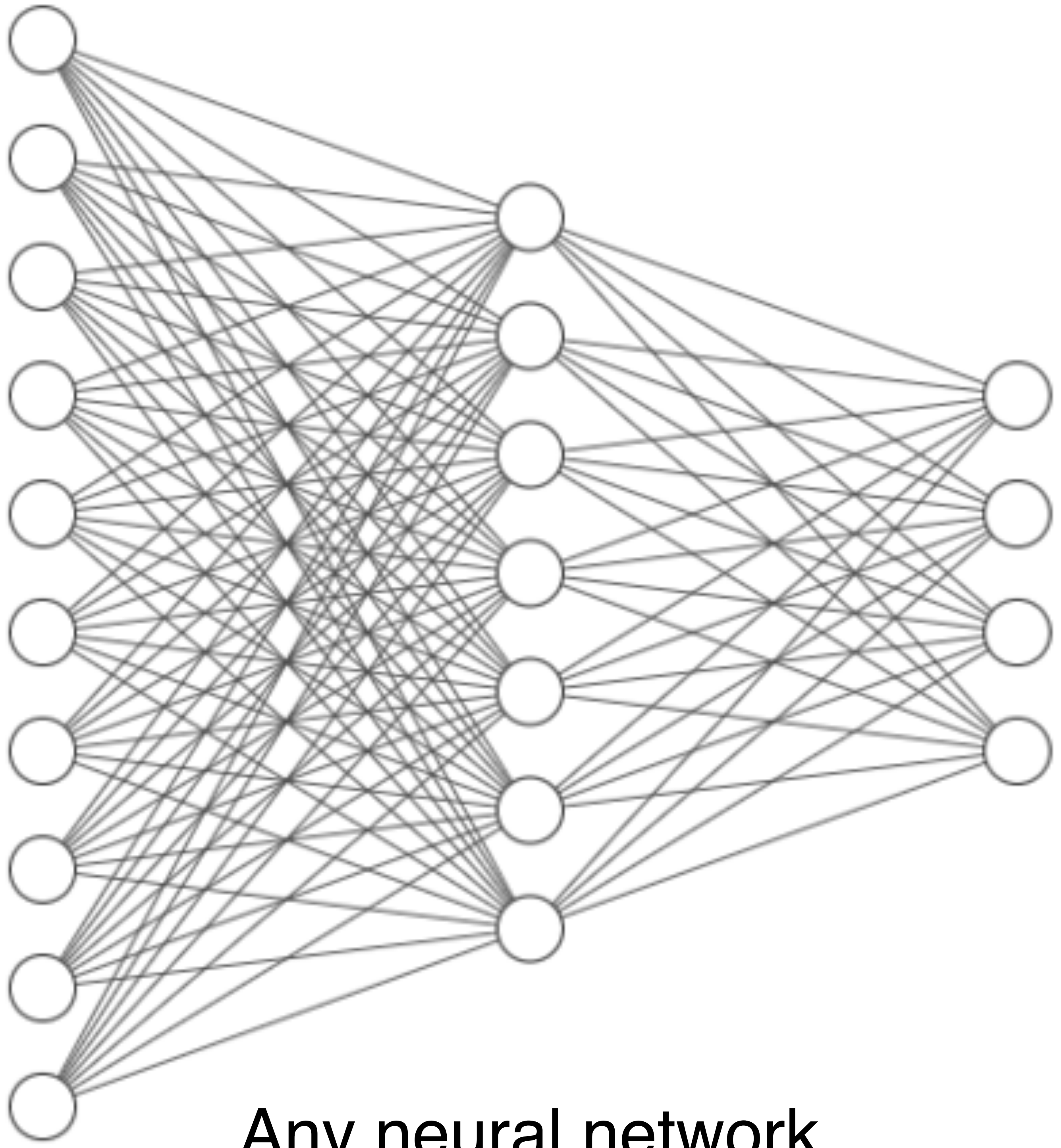
Any neural network
(CNN, RNN, MLP, …)

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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



Rating > 1? → yes/no
Rating > 2? → yes/no
Rating > 3? → yes/no
Rating > 4? → yes/no

Each output node is a binary task

Any neural network
(CNN, RNN, MLP, …)

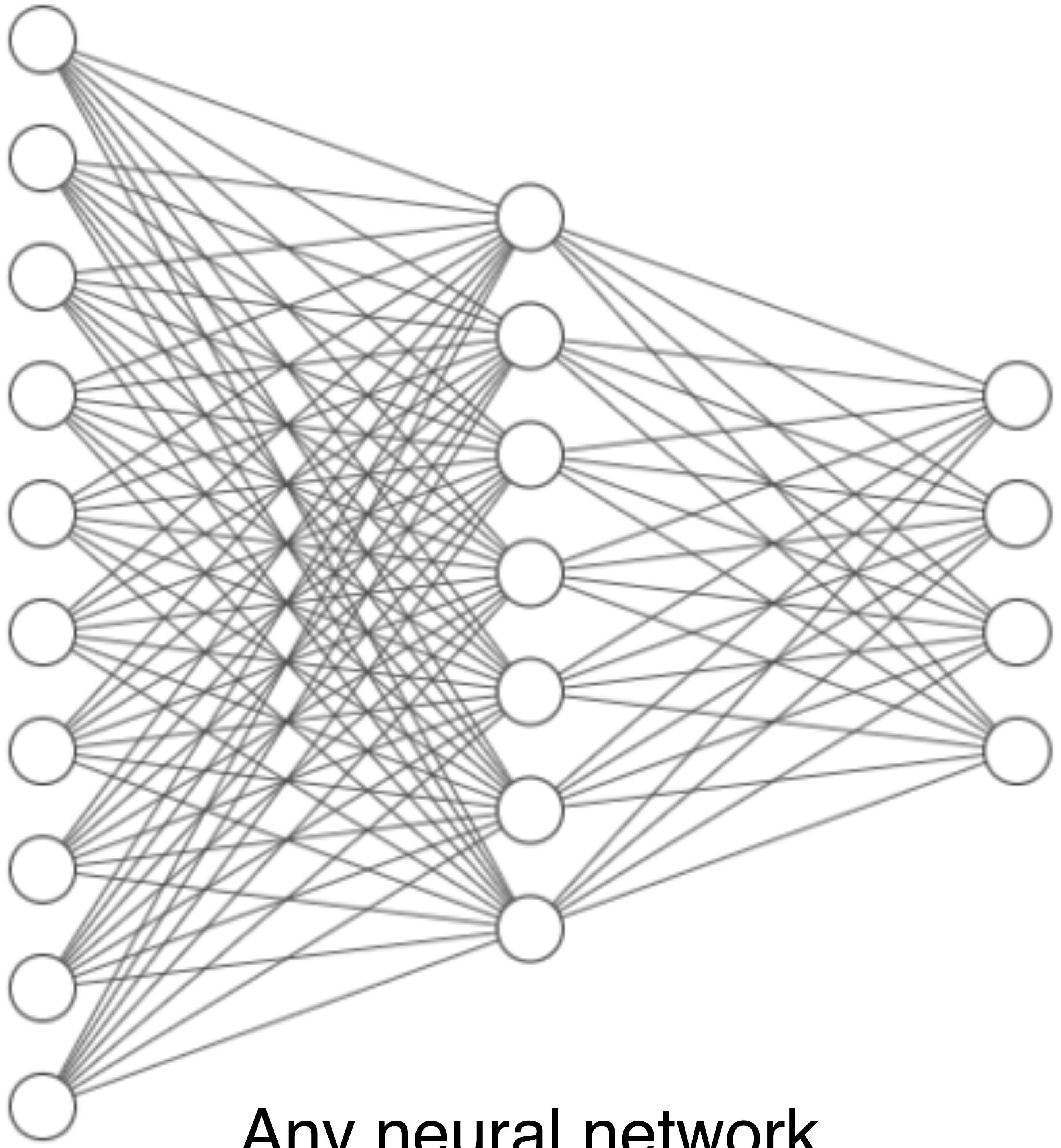
Predicted ordinal label is the sum over the <u>yes</u>es + 1

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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



Any neural network
(CNN, RNN, MLP, …)

Rating > 1? → **yes**/no
Rating > 2? → **yes**/no
Rating > 3? → yes/**no**
Rating > 4? → yes/**no**

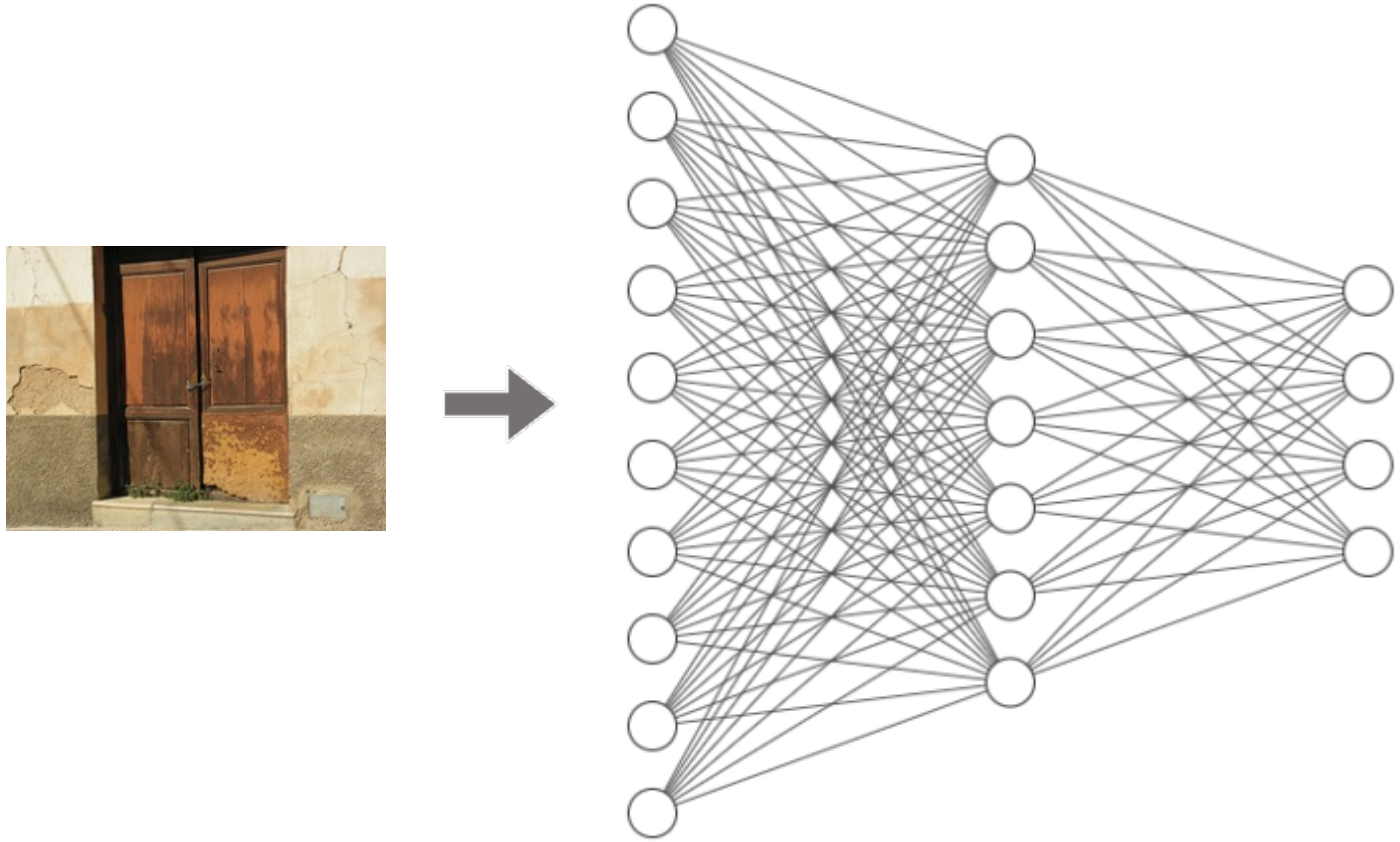
Each output node is a binary task

Predicted label:
3

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

Problem: rank inconsistency

Problem: rank inconsistency



Rating > 1? → **yes**/no

Rating > 2? → **yes**/no

Rating > 3? → yes/**no**

Rating > 4? → **yes**/no

Predicted label:

3

Greater than 4,
but not greater than 3?
That's paradoxical.

Niu Z, Zhou M, Wang L, Gao X, Hua G. Ordinal regression with multiple output CNN for age estimation. CVPR 2016

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
Pattern Recognition Letters. 140, 325-331, https://www.sciencedirect.com/science/article/pii/S016786552030413X

Shi, Cao, Raschka (2021)
Deep Neural Networks for Rank-Consistent Ordinal Regression Based On Conditional Probabilities.
Arxiv preprint, https://arxiv.org/abs/2111.08851

COnsistent RAnk Logits

Cao, Mirjalili, Raschka (2020)
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation
Pattern Recognition Letters. 140, 325-331, https://www.sciencedirect.com/science/article/pii/S016786552030413X

Conditional Ordinal Regression for Neural Networks

Shi, Cao, Raschka (2021)
Deep Neural Networks for Rank-Consistent Ordinal Regression Based On Conditional Probabilities.
Arxiv preprint, https://arxiv.org/abs/2111.08851

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

	How?	Advantages
Coral	Weight-sharing in output layer (mathematical proof in paper)	• Easy to implement • Reduced overfitting • Fast
CORN	Chain rule for probabilities & conditional training sets	

	How?	Advantages
Coral	Weight-sharing in output layer (mathematical proof in paper)	• Easy to implement • Reduced overfitting • Fast
CORN	Chain rule for probabilities & conditional training sets	• Easy to implement • Higher capacity • Better predictive performance

Skipping over the mathematical details ...
How do we use this in practice?

Converting a Classifier into a CORN Model

in 3 Lines of Code

PyTorch Lightning

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

Converting a Classifier into a CORN Model
in 3 Lines of Code

Full 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
```

PyTorch Lightning

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

Converting a Classifier into a CORN Model

in 3 Lines of Code

```
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
```

```
num_classes-1)
```

1

PyTorch Lightning

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

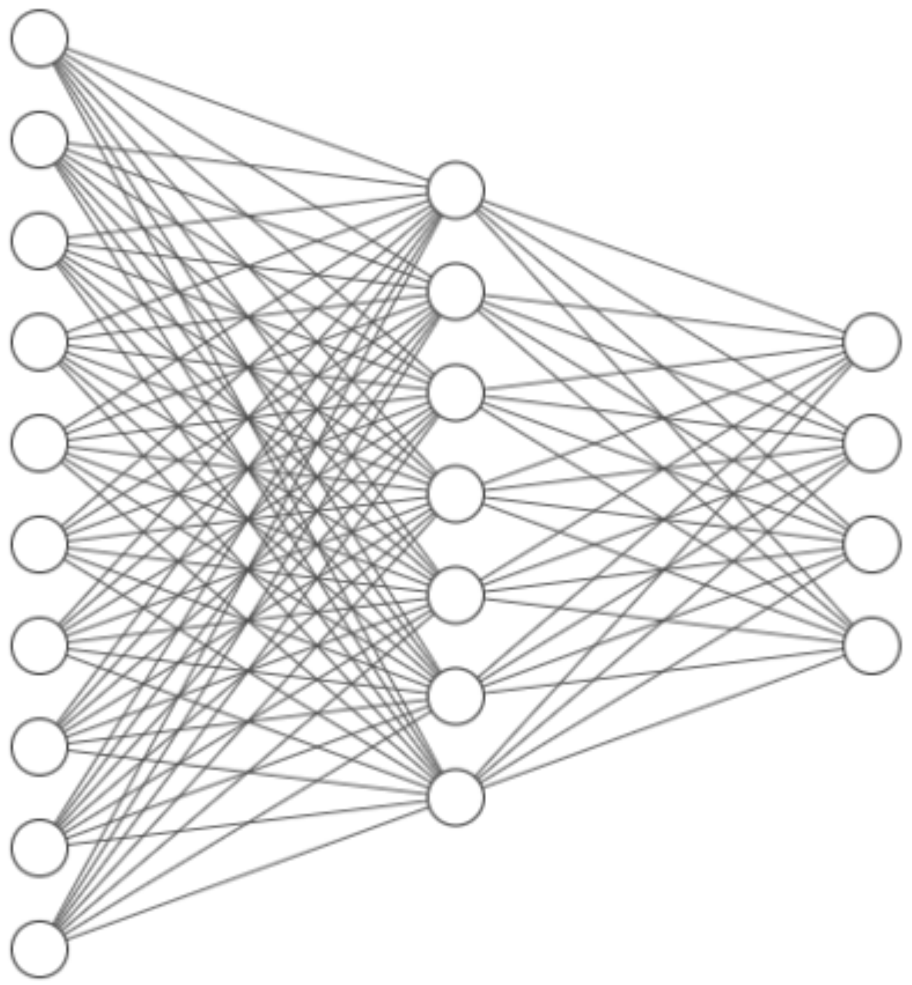
Converting a Classifier into a CORN Model

in 3 Lines of Code

```
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
```

```
num_classes-1)
```

1



PyTorch Lightning

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

Converting a Classifier into a CORN Model

in 3 Lines of Code

```
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
from coral_pytorch.dataset import corn_label_from_logits
```

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

2

PyTorch Lightning

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

Converting a Classifier into a CORN Model
in 3 Lines of Code

```
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
from coral_pytorch.dataset import corn_label_from_logits
```

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

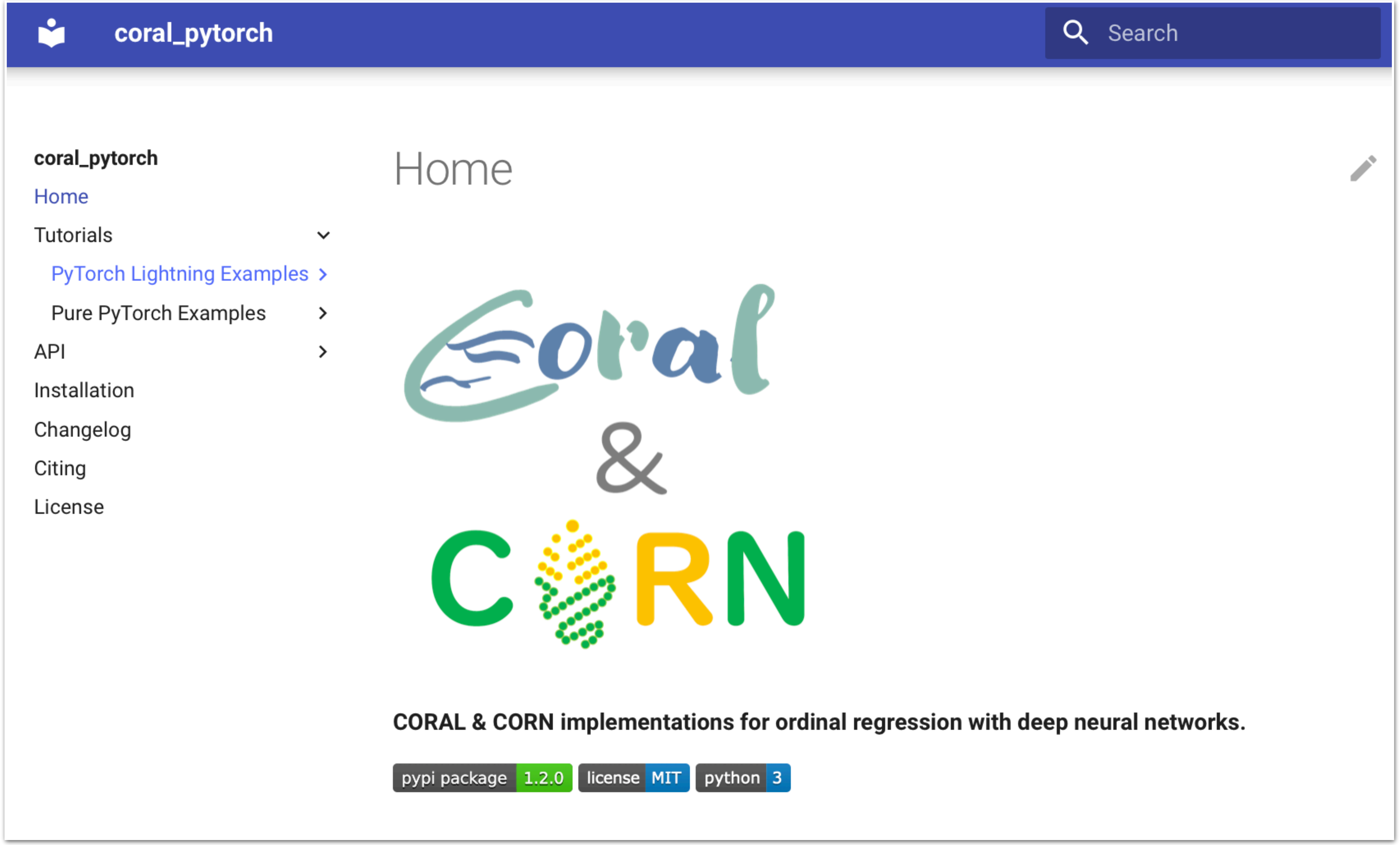
2

```
predicted_labels = corn_label_from_logits(logits)
```

3

PyTorch Lightning

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



More examples:

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

Acknowledgements

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Wenzhi Cao
Xintong Shi
Vahid Mirjalili

GRID.AI
PyTorch Lightning

William Falcon
Adrian Waechtli
Jirka Borovec
Alex Rose
Thomas Chaton
Marc Ferradou

Feb 25

https://sebastianraschka.com/books/

https://github.com/rasbt/machine-learning-book

Contact

@rasbt

sebastian@grid.ai

https://sebastianraschka.com

Additional Slides for Q&A

Converting a Classifier into a CORAL Model
in 4 Lines of Code

PyTorch Lightning

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

Converting a Classifier into a CORAL Model in 4 Lines of Code

```
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

```
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)
```

1

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
```

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

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

2

3

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
```

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

2
3

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

4

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

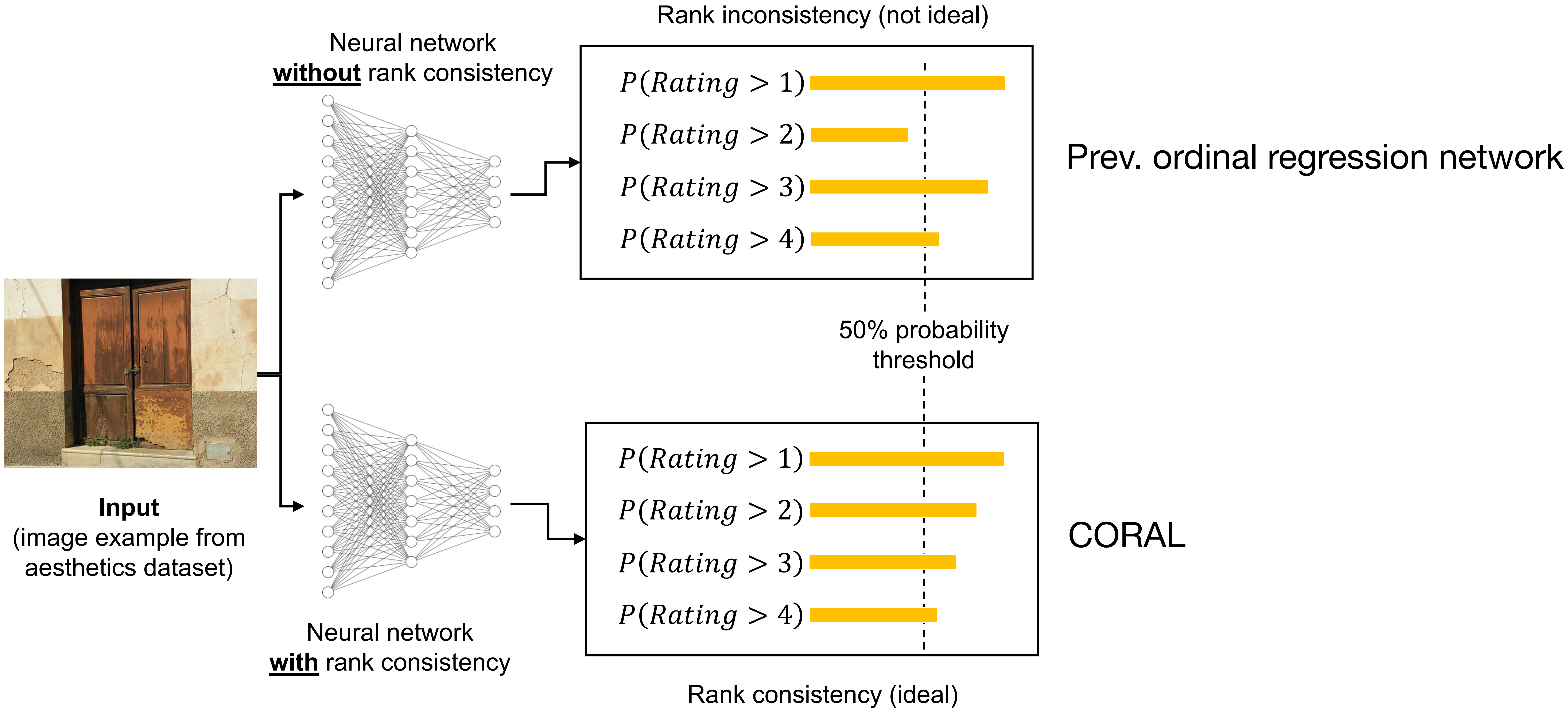
PyTorch Lightning

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

CORAL Performance

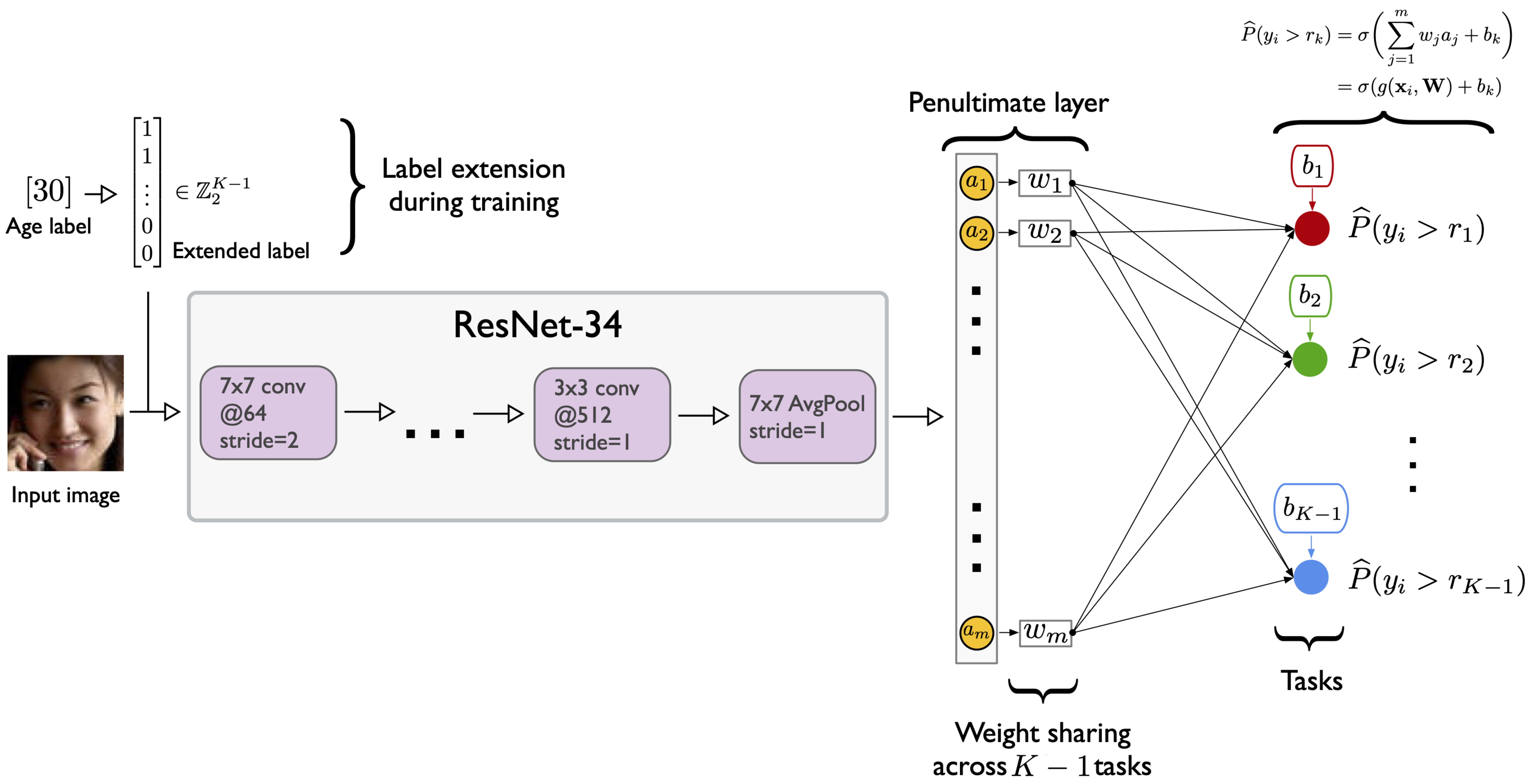
Table 1. **Age prediction errors on the test sets. All models are based on the ResNet-34 architecture.**

Method	Random Seed	MORPH-2		AFAD		CACD	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
CE-CNN	0	3.26	4.62	3.58	5.01	5.74	8.20
	1	3.36	4.77	3.58	5.01	5.68	8.09
	2	3.39	4.84	3.62	5.06	5.53	7.92
	AVG ± SD	3.34 ± 0.07	4.74 ± 0.11	3.60 ± 0.02	5.03 ± 0.03	5.65 ± 0.11	8.07 ± 0.14
OR-CNN (Niu et al., 2016)	0	2.87	4.08	3.56	4.80	5.36	7.61
	1	2.81	3.97	3.48	4.68	5.40	7.78
	2	2.82	3.87	3.50	4.78	5.37	7.70
	AVG ± SD	2.83 ± 0.03	3.97 ± 0.11	3.51 ± 0.04	4.75 ± 0.06	5.38 ± 0.02	7.70 ± 0.09
CORAL-CNN (ours)	0	2.66	3.69	3.42	4.65	5.25	7.41
	1	2.64	3.64	3.51	4.76	5.25	7.50
	2	2.62	3.62	3.48	4.73	5.24	7.52
	AVG ± SD	**2.64 ± 0.02**	**3.65 ± 0.04**	**3.47 ± 0.05**	**4.71 ± 0.06**	**5.25 ± 0.01**	**7.48 ± 0.06**



Cao, Mirjalili, Raschka (2020)
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation
Pattern Recognition Letters. 140, 325-331, https://www.sciencedirect.com/science/article/pii/S016786552030413X

CORAL Architecture



CORAL Theorem

Theorem 1 (Ordered bias units). *By minimizing the loss function defined in Eq. 4, 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: replacing b_k with b_{k+1} , or replacing b_{k+1} with b_k, decreases the objective value L. Let

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

By the ordering relationship, we have

$$A_1 \cup A_2 \cup A_3 = \{1, 2, \ldots, N\}.$$

Denote $p_n(b_k) = \sigma(g(\mathbf{x}_n, \mathbf{W}) + b_k)$ and

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

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

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

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

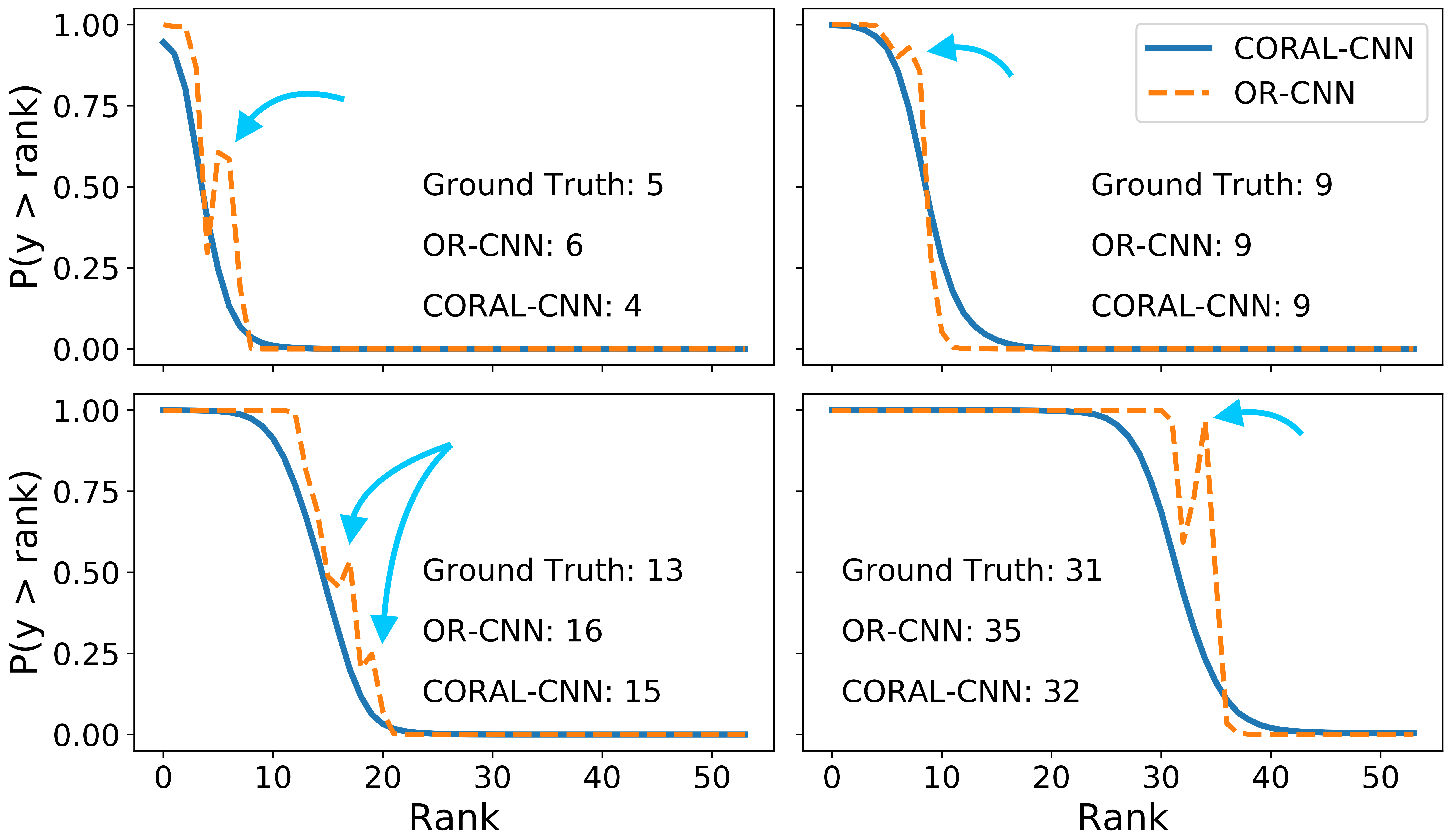
$$\frac{1}{\lambda^{(k)}}\Delta_1 L + \frac{1}{\lambda^{(k+1)}}\Delta_2 L = -\sum_{n\in A_3}(\delta_n + \delta'_n) < 0,$$

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

$$b_1^* \geq b_2^* \geq \ldots \geq b_{K-1}^*.$$

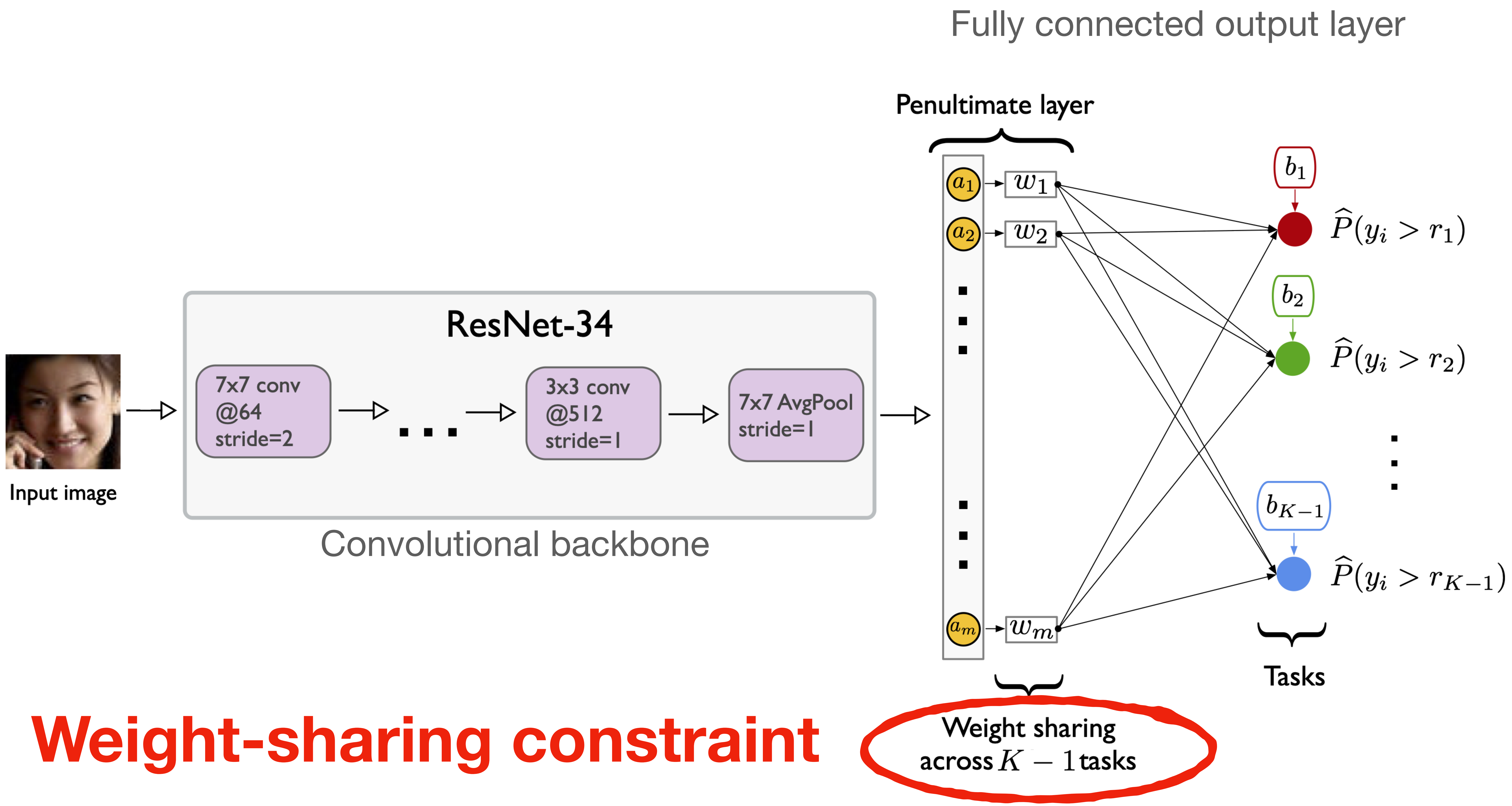
□

CORAL Rank Consistency



Fixing rank inconsistency introduced a limitation:
weight-sharing constraint restricts the network's capacity

Coral



Weight-sharing constraint

Cao, Mirjalili, Raschka (2020)
Rank Consistent Ordinal Regression for Neural Networks with Application to Age Estimation
Pattern Recognition Letters. 140, 325-331, https://www.sciencedirect.com/science/article/pii/S016786552030413X

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.
Arxiv preprint, https://arxiv.org/abs/2111.08851

CORN Method 1/3

3.3. Rank-consistent Ordinal Regression based on Conditional Probabilities

Given a training set $D = \left\{\mathbf{x}^{[i]}, y^{[i]}\right\}_{i=1}^{N}$, 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 $f_k\left(\mathbf{x}^{[i]}\right)$ represents the conditional probability[1]

$$f_k\left(\mathbf{x}^{[i]}\right) = \hat{P}\left(y^{[i]} > r_k \,|\, y^{[i]} > r_{k-1}\right), \tag{2}$$

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:

$$\hat{P}\left(y^{[i]} > r_k\right) = \prod_{j=1}^{k} f_j\left(\mathbf{x}^{[i]}\right). \tag{3}$$

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

$$\hat{P}\left(y^{[i]} > r_1\right) \geq \hat{P}\left(y^{[i]} > r_2\right) \geq ... \geq \hat{P}\left(y^{[i]} > r_{K-1}\right), \tag{4}$$

which guarantees rank consistency among the $K - 1$ binary tasks.

CORN Method 2/3

3.4. Conditional Training Subsets

Our model aims to estimate $f_1\left(\mathbf{x}^{[i]}\right)$ and the conditional probabilities $f_2\left(\mathbf{x}^{[i]}\right), ..., f_{K-1}\left(\mathbf{x}^{[i]}\right)$. Estimating $f_1\left(\mathbf{x}^{[i]}\right)$ 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 $\hat{P}\left(y^{[i]} > r_2 \mid y^{[i]} > r_1\right)$, 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[2].

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:

$$
\begin{aligned}
&S_1: \text{ all } \left\{\left(\mathbf{x}^{[i]}, y^{[i]}\right)\right\}, \text{ for } i \in \{1, ..., N\},\\
&S_2: \left\{(\mathbf{x}^{[i]}, y^{[i]}) \mid y^{[i]} > r_1\right\},\\
&\cdots\\
&S_{K-1}: \left\{(\mathbf{x}^{[i]}, y^{[i]}) \mid y^{[i]} > r_{k-2}\right\},
\end{aligned}
$$

where $N = |S_1| \geq |S_2| \geq ... \geq |S_{K-1}|$, 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 $\hat{P}\left(y^{[i]} > r_k \mid y^{[i]} > r_{k-1}\right)$ for $k \geq 2$.

CORN Method 3/3

3.5. Loss Function

Let $f_j(\mathbf{x}^{[i]})$ 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:

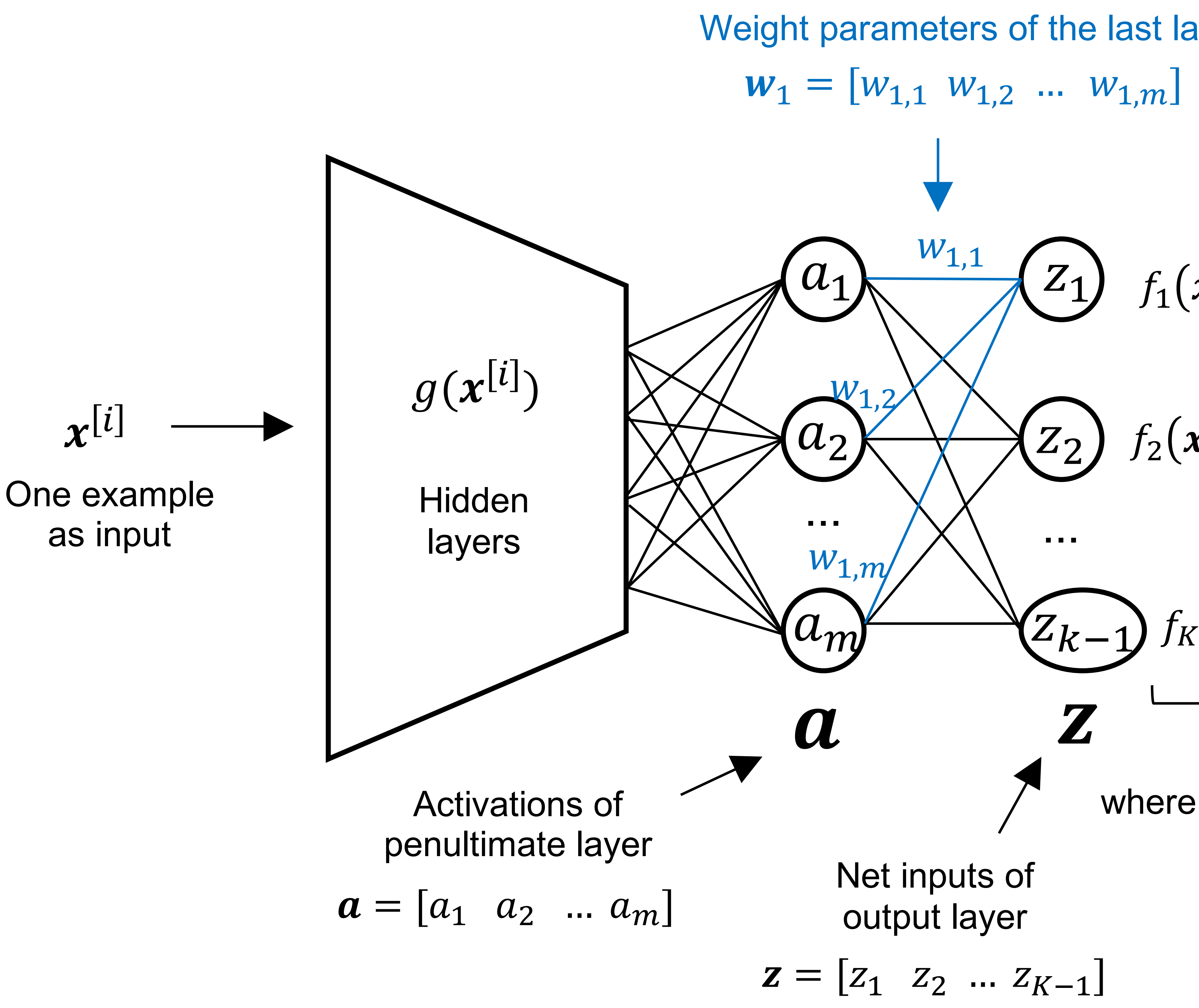
$$L(\mathbf{X}, \mathbf{y}) = -\frac{1}{\sum_{j=1}^{K-1}|S_j|} \sum_{j=1}^{K-1} \sum_{i=1}^{|S_j|} \Big[\log\left(f_j(\mathbf{x}^{[i]})\right) \cdot \mathbb{1}\left\{y^{[i]} > r_j\right\} + \log\left(1 - f_j\left(\mathbf{x}^{[i]}\right)\right) \cdot \mathbb{1}\left\{y^{[i]} \leq r_j\right\} \Big], \quad (5)$$

We note that in $f_j(\mathbf{x}^{[i]})$, $\mathbf{x}^{[i]}$ represents the i-th training example in S_j. To simplify the notation, we omit an additional index j to distinguish between $\mathbf{x}^{[i]}$ 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 $\mathbf{Z}$ are the net inputs of the last layer (aka logits), as shown in Fig. 2, and $\log\left(\sigma\left(\mathbf{z}^{[i]}\right)\right) = \log\left(f_j\left(\mathbf{x}^{[i]}\right)\right)$:

$$L(\mathbf{Z}, \mathbf{y}) = -\frac{1}{\sum_{j=1}^{K-1}|S_j|} \sum_{j=1}^{K-1} \sum_{i=1}^{|S_j|} \Big[\log\left(\sigma\left(\mathbf{z}^{[i]}\right)\right) \cdot \mathbb{1}\left\{y^{[i]} > r_j\right\} + \left(\log\left(\sigma\left(\mathbf{z}^{[i]}\right)\right) - \mathbf{z}^{[i]}\right) \cdot \mathbb{1}\left\{y^{[i]} \leq r_j\right\} \Big]. \quad (6)$$

CORN Architecture



Logistic sigmoid function

$$f_1(\mathbf{x}^{[i]}) = \sigma\left(z_1^{[i]}\right) \quad \text{where} \quad z_1^{[i]} = \mathbf{a}^{[i]}\, \mathbf{w}_1^T + b_1$$

$$f_2(\mathbf{x}^{[i]}) = \sigma\left(z_2^{[i]}\right) \quad \text{where} \quad z_2^{[i]} = \mathbf{a}^{[i]}\, \mathbf{w}_2^T + b_2$$

...

$$f_{K-1}(\mathbf{x}^{[i]}) = \sigma\left(z_{K-1}^{[i]}\right) \quad \text{where} \quad z_{K-1}^{[i]} = \mathbf{a}^{[i]}\, \mathbf{w}_{K-1}^T + b_{K-1}$$

Outputs

where $\sigma\left(z_{K-1}^{[i]}\right) = \hat{P}\left(y^{[i]} > r_{K-1} \mid y^{[i]} > r_{K-2}\right)$

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

where $\hat{P}\left(y^{[i]} > r_k\right) = \hat{P}\left(y^{[i]} > r_1\right) \cdot \hat{P}\left(y^{[i]} > r_2 \mid y^{[i]} > r_1\right) \cdots \hat{P}\left(y^{[i]} > r_k \mid y^{[i]} > r_{k-1}\right)$

CORN Performance 1/2

Table 1. Prediction errors on the test sets. Best results are highlighted in bold.

Method	Seed	MORPH-2 (Balanced)		AFAD (Balanced)		AES		FIREMAN	
		MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
CE-NN	0	3.81	5.19	3.31	4.27	0.43	0.68	0.80	1.14
	1	3.60	4.8	3.28	4.19	0.43	0.69	0.80	1.14
	2	3.61	4.84	3.32	4.22	0.45	0.71	0.79	1.13
	3	3.85	5.21	3.24	4.15	0.43	0.70	0.80	1.16
	4	3.80	5.14	3.24	4.13	0.42	0.68	0.80	1.15
	AVG±SD	3.73 ± 0.12	5.04 ± 0.20	3.28 ± 0.04	4.19 ± 0.06	**0.43 ± 0.01**	0.69 ± 0.01	0.80 ± 0.01	1.14 ± 0.01
OR-NN [11]	0	3.21	4.25	2.81	3.45	0.44	0.70	0.75	1.07
	1	3.16	4.25	2.87	3.54	0.43	0.69	0.76	1.08
	2	3.16	4.31	2.82	3.46	0.43	0.69	0.77	1.10
	3	2.98	4.05	2.89	3.49	0.44	0.70	0.76	1.08
	4	3.13	4.27	2.86	3.45	0.43	0.69	0.74	1.07
	AVG±SD	3.13 ± 0.09	4.23 ± 0.10	2.85 ± 0.03	3.48 ± 0.04	**0.43 ± 0.01**	0.69 ± 0.01	**0.76 ± 0.01**	**1.08 ± 0.01**
CORAL [1]	0	2.94	3.98	2.95	3.60	0.47	0.72	0.82	1.14
	1	2.97	4.03	2.99	3.69	0.47	0.72	0.83	1.16
	2	3.01	3.98	2.98	3.70	0.48	0.73	0.81	1.13
	3	2.98	4.01	3.00	3.78	0.44	0.70	0.82	1.16
	4	3.03	4.06	3.04	3.75	0.46	0.72	0.82	1.15
	AVG±SD	2.99 ± 0.04	4.01 ± 0.03	2.99 ± 0.03	3.70 ± 0.07	0.46 ± 0.02	0.72 ± 0.01	0.82 ± 0.01	1.15 ± 0.01
CORN (ours)	0	2.98	4	2.80	3.45	0.41	0.67	0.75	1.07
	1	2.99	4.01	2.81	3.44	0.44	0.69	0.76	1.08
	2	2.97	3.97	2.84	3.48	0.42	0.68	0.77	1.10
	3	3.00	4.06	2.80	3.48	0.43	0.69	0.76	1.08
	4	2.95	3.92	2.79	3.45	0.43	0.69	0.74	1.07
	AVG±SD	**2.98 ± 0.02**	**3.99 ± 0.05**	**2.81 ± 0.02**	**3.46 ± 0.02**	**0.43 ± 0.01**	**0.68 ± 0.01**	**0.76 ± 0.01**	**1.08 ± 0.01**

CORN Performance 2/2

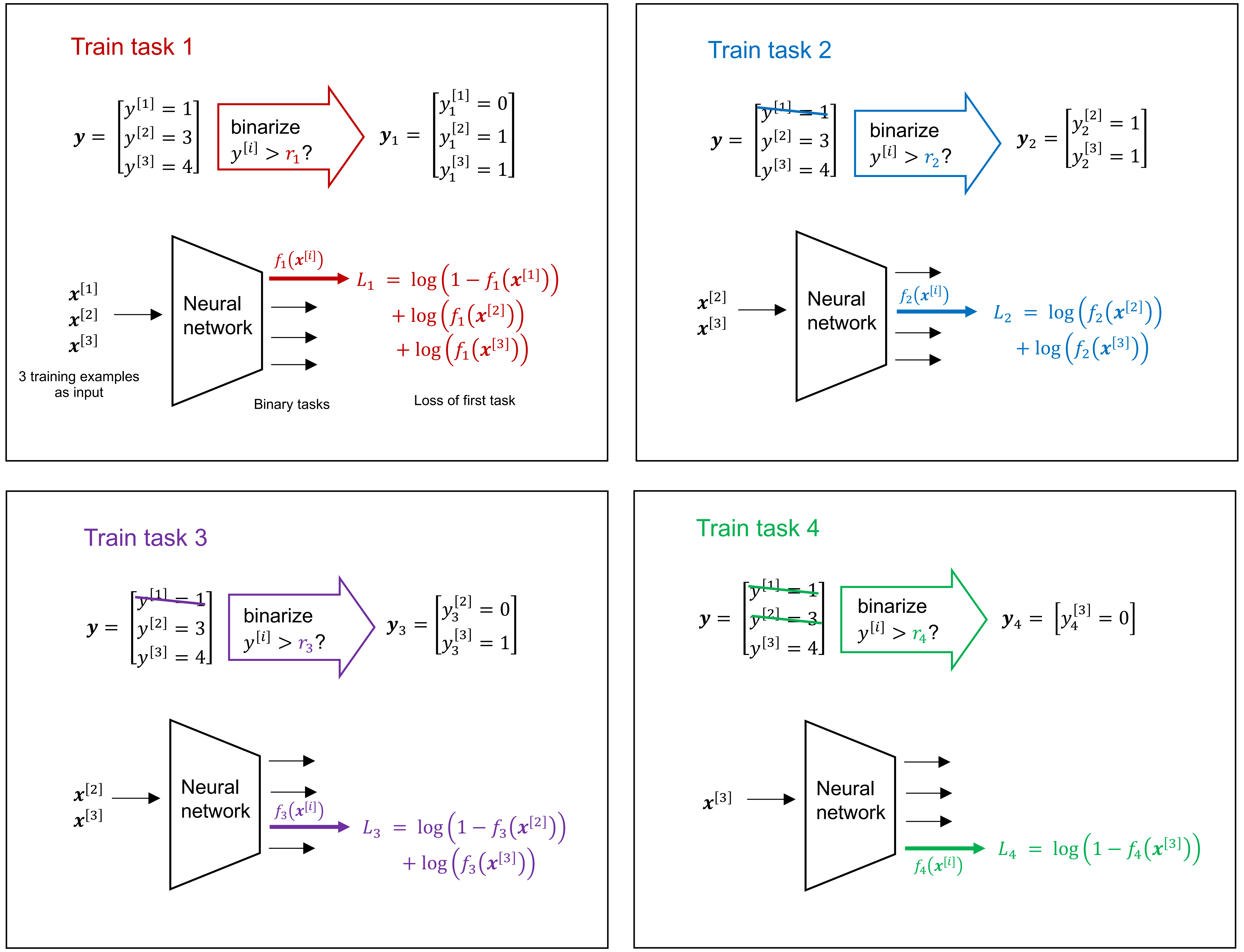
Table S1. Prediction errors on the test sets. Best results are highlighted in bold.

Method	Seed	TripAdvisor (Balanced)		Coursera (Balanced)	
		MAE	RMSE	MAE	RMSE
CE-RNN	0	1.13	1.56	1.01	1.48
	1	1.04	1.53	0.97	1.05
	2	1.05	1.54	1.12	1.65
	3	1.23	1.81	1.18	1.76
	4	1.03	1.52	0.84	1.26
	AVG±SD	1.10 ± 0.09	1.59 ± 0.12	1.02 ± 0.13	1.53 ± 0.19
OR-RNN [11]	0	1.06	1.53	0.98	1.34
	1	1.09	1.50	0.93	1.24
	2	1.11	1.53	1.12	1.47
	3	1.23	1.52	1.11	1.53
	4	1.07	1.40	0.85	1.16
	AVG±SD	1.11 ± 0.07	1.50 ± 0.06	1.00 ± 0.12	1.35 ± 0.15
CORAL [1]	0	1.15	1.58	0.99	1.29
	1	1.14	1.49	1.03	1.39
	2	1.16	1.46	1.14	1.40
	3	1.19	1.41	1.20	1.40
	4	1.13	1.47	0.82	1.11
	AVG±SD	1.15 ± 0.02	**1.48 ± 0.06**	1.04 ± 0.15	**1.33 ± 0.13**
CORN (ours)	0	1.09	1.55	0.95	1.37
	1	1.09	1.53	0.90	1.32
	2	1.01	1.45	1.07	1.49
	3	1.12	1.51	1.05	1.47
	4	1.03	1.46	0.78	1.14
	AVG±SD	**1.07 ± 0.05**	1.50 ± 0.04	**0.95 ± 0.12**	1.36 ± 0.14

CORN Loss

Assume 3 training examples $\boldsymbol{x}^{[1]}$, $\boldsymbol{x}^{[2]}$, and $\boldsymbol{x}^{[3]}$ with the following 3 rank labels:

$$\boldsymbol{y} = \begin{bmatrix} y^{[1]} = 1 \\ y^{[2]} = 3 \\ y^{[3]} = 4 \end{bmatrix}$$



Overall loss: $L(\boldsymbol{X}, \boldsymbol{y}) = \frac{1}{\sum_i |\boldsymbol{y}_i|} \sum_i L_i$

$$= \frac{1}{3 + 2 + 2 + 1} L_1 + L_2 + L_3 + L_4$$