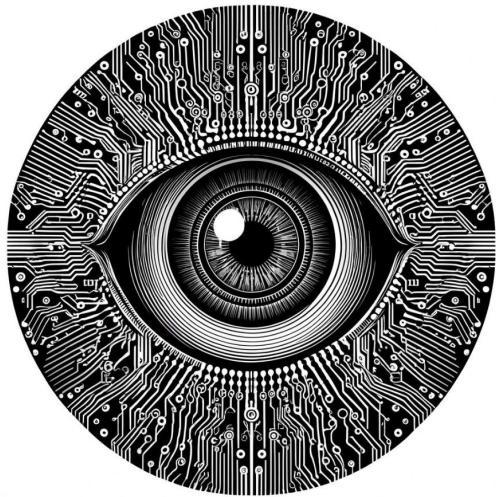


Workshop 6 - Multi-label Classification and Binary Cross Entropy Loss



Antonio Rueda-Toicen

About me

- AI Researcher at [Hasso Plattner Institute](#), AI Engineer & DevRel for [Voxel51](#)
- Organizer of the [Berlin Computer Vision Group](#)
- Instructor at [Nvidia's Deep Learning Institute](#) and Berlin's [Data Science Retreat](#)
- Preparing a [MOOC for OpenHPI](#) (to be published in May)



[LinkedIn](#)

Agenda

- Building a FeedForward Network for Classification (review)
- Usage of Binary Cross Entropy Loss for multilabel classification

Notebooks

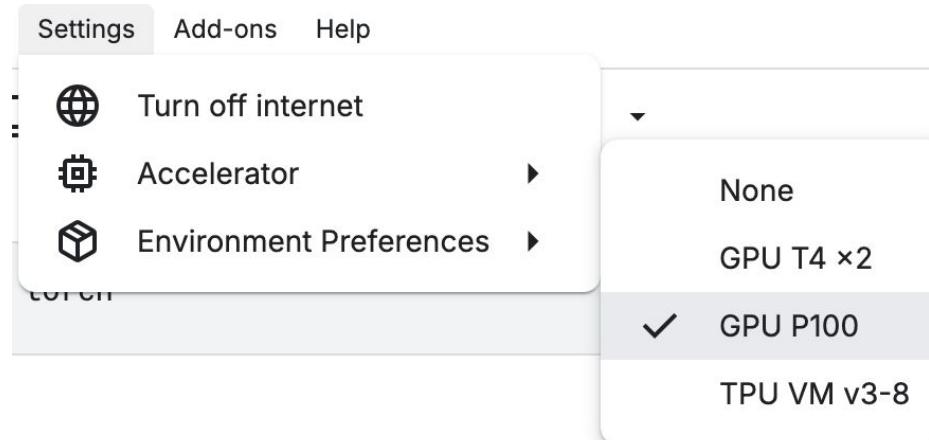
- Single label classification on pet images (review)
- Multilabel classification with binary cross entropy on satellite images

Setup for today

- [Setting up a Weights and Biases API token](#)
- [Setting up free GPU and TPU access in Kaggle Notebooks](#)

Kaggle Runtime Setup for Training

- [Follow the Kaggle GPU Access Guide on our repository](#)
- Settings -> Accelerator -> GPU P100



Google Colab Setup for Kaggle API keys

- Go to <https://www.kaggle.com/settings> and “Create New Token” on your API key
- Download and open the kaggle.json file

```
{} kaggle.json ×  
Users > antonio > Downloads > {} kaggle.json > abc key  
1 {"username": "andandand", "key": "5d5e0493f2a}
```

- In Google Colab, go to Secrets -> “Add New Secret”
- Create and enable KAGGLE_USERNAME and KAGGLE_KEY variables with the content of the JSON file



Residual block in Resnet18

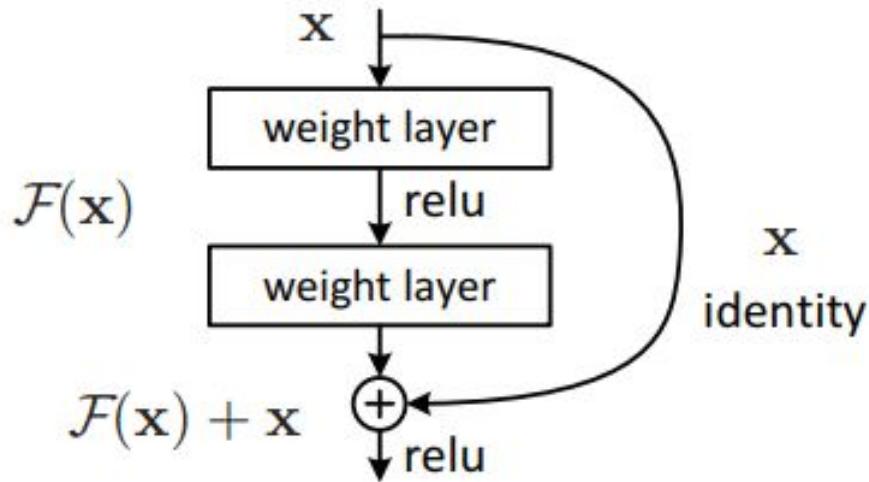


Diagram from [Deep Residual Learning for Image Recognition](#)

Understanding the Amazon from space



[Planet: understanding the Amazon](#)

The Planet satellite constellation



Tiny satellites show us the Earth as it changes in near-real-time | TED Talk

Planet's Amazon Dataset on Kaggle



PLANET · FEATURED PREDICTION COMPETITION · 8 YEARS AGO

Late Submission

...

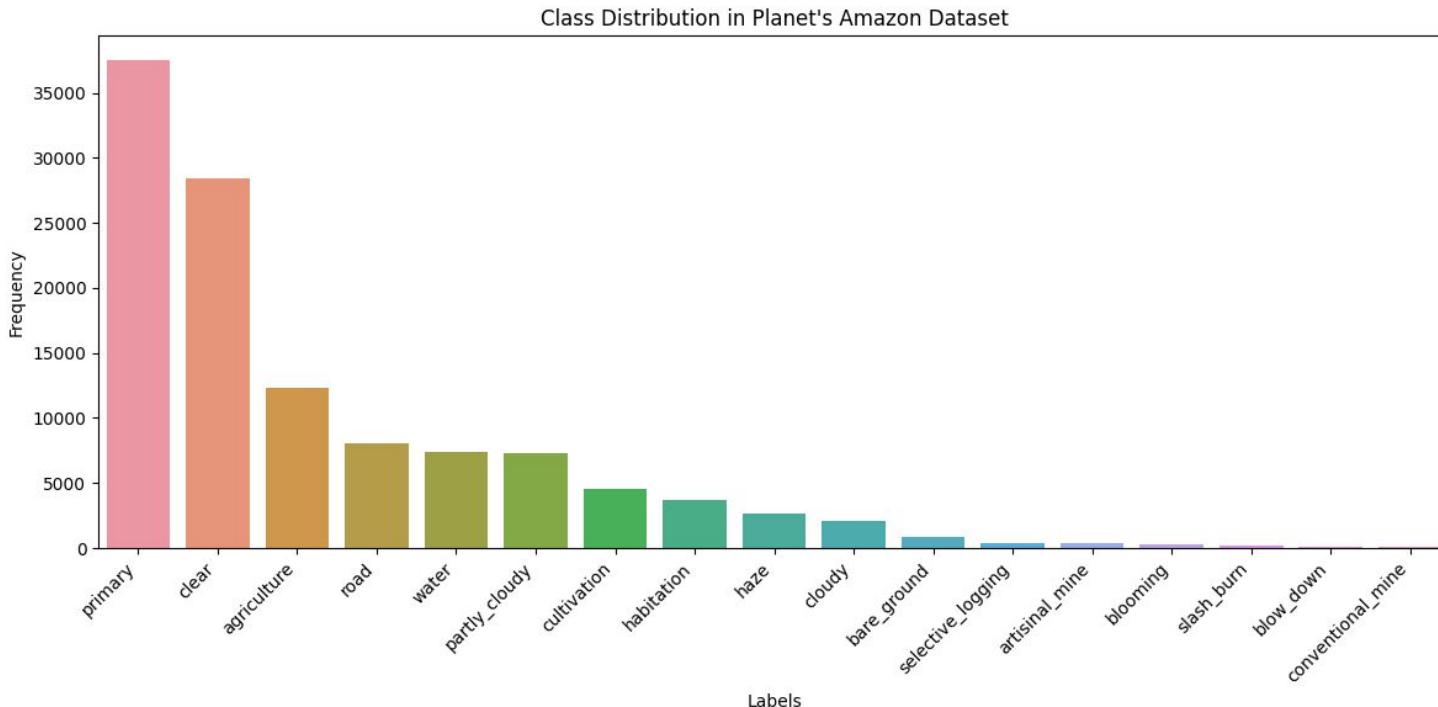
Planet: Understanding the Amazon from Space

Use satellite data to track the human footprint in the Amazon rainforest



[Competition page on Kaggle](#)

Our class distribution is unbalanced

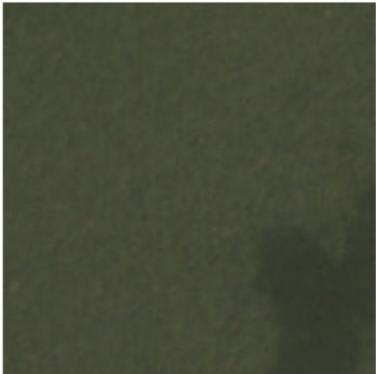


Our images have multiple labels

haze, primary



clear, primary



Sample Images with Multiple Labels
clear, conventional_mine, primary, road



clear, primary



clear, primary



partly_cloudy, primary





DATASET

Tags Names Classes Description ...

SAMPLE 0

ID Filepath Tags Label Field

SAMPLE 1

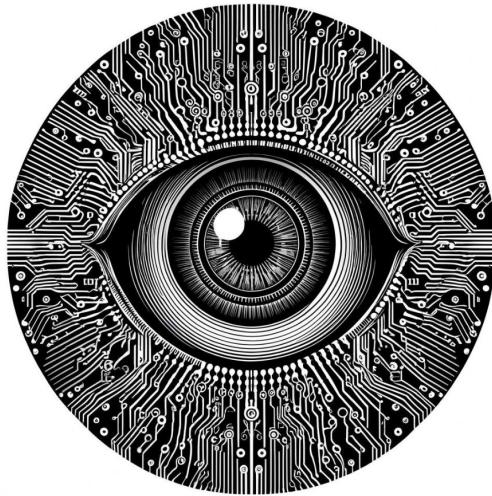
ID Filepath Tags Label Field

SAMPLE 2

ID Filepath Tags Label Field

How we create a
FiftyOne dataset

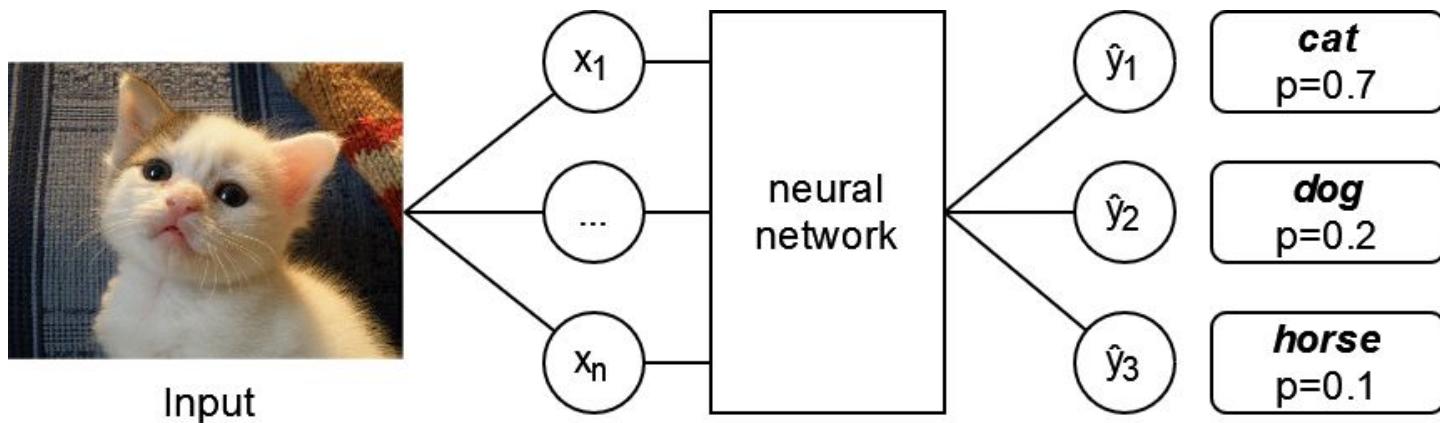
Building a Feedforward Network for Classification in PyTorch



Learning goals

- Create a multilayer neural network for classification in PyTorch
- Gain familiarity with the nn.Module syntax for network creation
- Understand usage of the softmax activation function
- Develop intuitions on Categorical Cross Entropy
- Use the Adam variant of stochastic gradient descent

Multiclass classification: n probabilities for n classes



Classification $P(\hat{y}_{\text{cat}}) + P(\hat{y}_{\text{dog}}) + P(\hat{y}_{\text{horse}}) = 1$

Cross entropy loss

$$y = 1.0$$



$$\hat{y} = 0.75$$

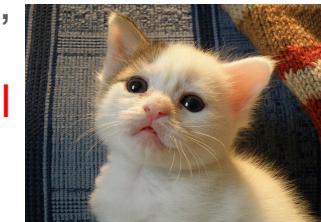


$$H(p, q) = -\frac{1}{n} \sum_{i=1}^n p(x_i) \log q(x_i)$$

Labels y_i map to $p(x_i)$, predictions \hat{y}_i map to $q(x_i)$. Suppose that we show the network, only the following image ($n=1$).

“one hot encoding” 🔥 = only one true label

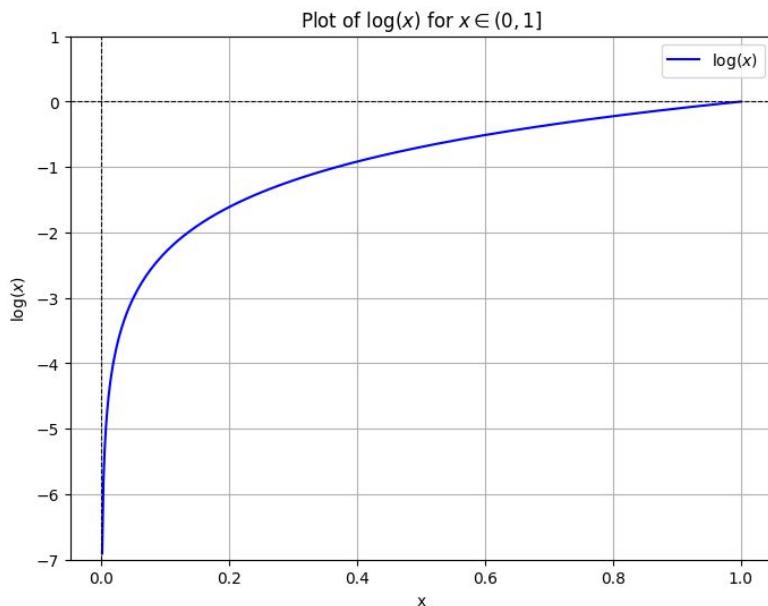
The label for the image is encoded as $y = [1, 0, 0]$ ($i = 0 = \text{cat}$, $i = 1 = \text{dog}$, $i = 2 = \text{horse}$)



- First the network outputs $\hat{y} = [0.75, 0.25, 0.0]$
- Then the network outputs $\hat{y} = [0.99, 0.01, 0.0]$ (if trained correctly)

Intuition for the log function in cross entropy loss

$$y = [1, 0, 0] \quad \text{where } i = 0 \text{ (cat), } i = 1 \text{ (dog), } i = 2 \text{ (horse)}$$



$$\ln(1) = \log(1) = 0$$



$$y_0 = 1 \quad \text{and} \quad \hat{y}_0 = 1$$

Cross Entropy Loss = $-1 \cdot \log(1) = 0$ (perfect prediction, no loss)

$$y_0 = 1 \quad \text{and} \quad \hat{y}_0 = 0.01$$

Cross Entropy Loss = $-(1 \cdot \log(0.01)) = 4.65$ (bad prediction, high loss)

Implementing cross entropy loss from probabilities for a single sample

```
import torch

# Example predicted probabilities from model
y_hat = torch.tensor([0.75, 0.25, 0.0])

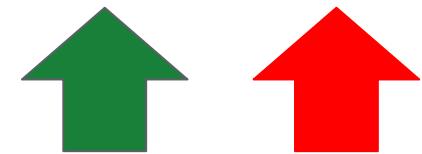
# True label: y = [1, 0, 0]
y = torch.tensor([1, 0, 0], dtype=torch.float32)

def cross_entropy(y, y_hat):
    epsilon = 1e-15 # equal to 1 times 10^-15 (very small number)
    # Adding epsilon prevents log(0) which is undefined
    log_probs = torch.log(y_hat + epsilon)
    # Notice that the sum would only give a value different
    # than zero on the index of true_label
    # we are computing a single sample
    # so n = 1 (batch dimension)
    return -torch.sum(y * log_probs)

loss = cross_entropy(y, y_hat)
# Gives us tensor(0.287), not perfect, but not the worst possible
```



$y = [1, 0, 0]$ $y_{\text{hat}} = [0.75, 0.25, 0.0]$

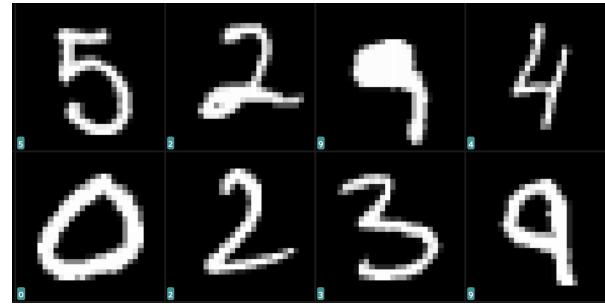


$$H(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n y_i \log \hat{y}_i$$

A minimal network for classification

```
import torch.nn as nn

# Model
model = nn.Sequential(
    # Flatten the input image
    nn.Flatten(),
    # The number of input features is the number of pixels in the image
    nn.Linear(in_features=28 * 28, out_features = 128),
    # We add a non-linearity
    nn.ReLU(),
    # We create 10 scores, aka 'logits', one for each class that we have
    # Notice that there is no ReLU after nn.Linear
    nn.Linear(in_features=128, out_features = 10))
```



Tip: do not confuse these “logits” with the function described on
<https://en.wikipedia.org/wiki/Logit>

$$f(x) = \ln\left(\frac{x}{1-x}\right)$$

✗ these “logits” are **not** this ^

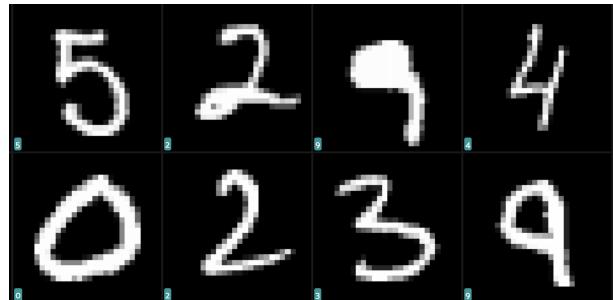
Using the nn.Module syntax

```
import torch.nn as nn

class MNISTClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        # Instead of Sequential, we define each module as a class attribute
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(in_features=28 * 28, out_features=128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(in_features=128, out_features=10)

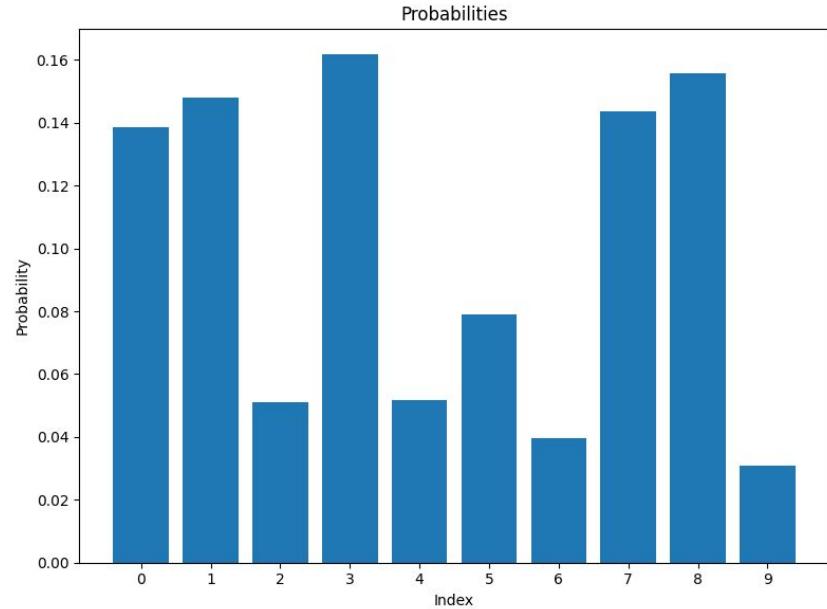
    def forward(self, x):
        # Define the forward pass and set a breakpoint
        import pdb; pdb.set_trace()
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x) # notice no relu here
        return x

# Create an instance of the model
model = MNISTClassifier()
```

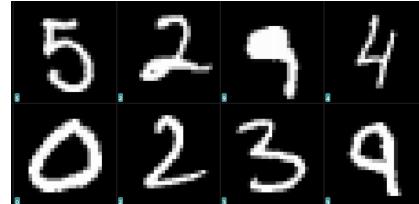


The softmax function

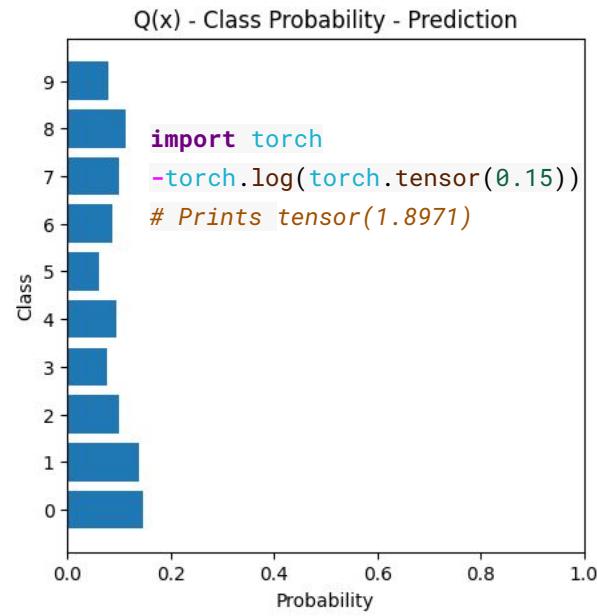
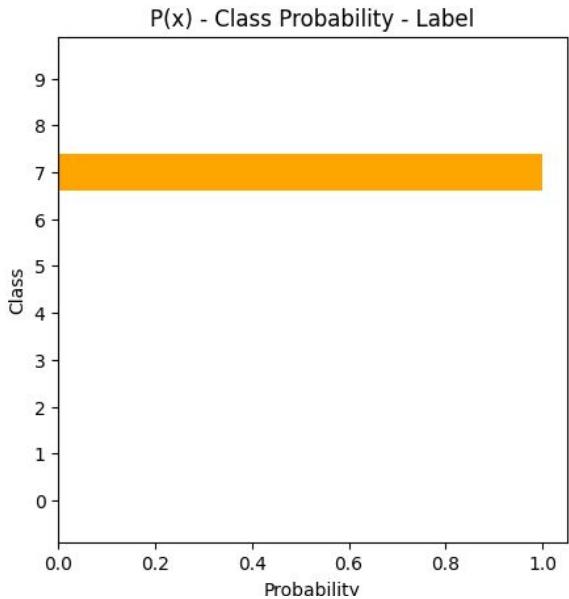
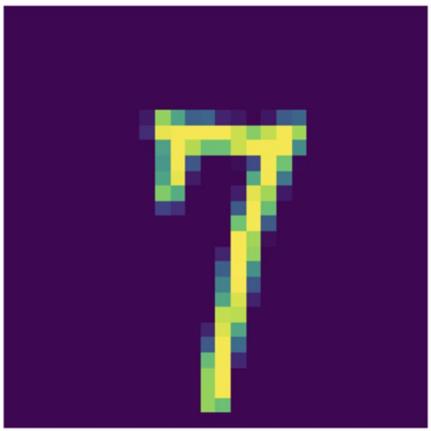
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



```
import torch.nn.functional as F
logits = torch.tensor([ 0.7645,  0.8300, -0.2343,  0.9186, -0.2191,  0.2018, -0.4869,  0.8000,  0.8815, -0.7336])
probabilities = F.softmax(logits, dim=0)
print(probabilities.sum()) # prints tensor(1.00)
```

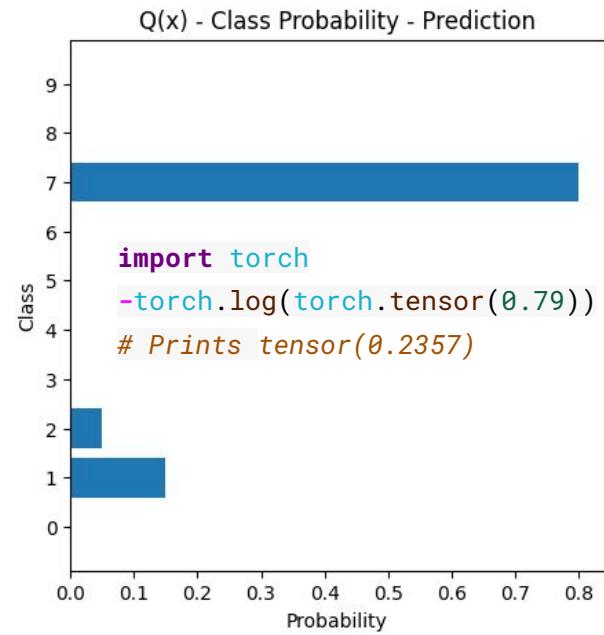
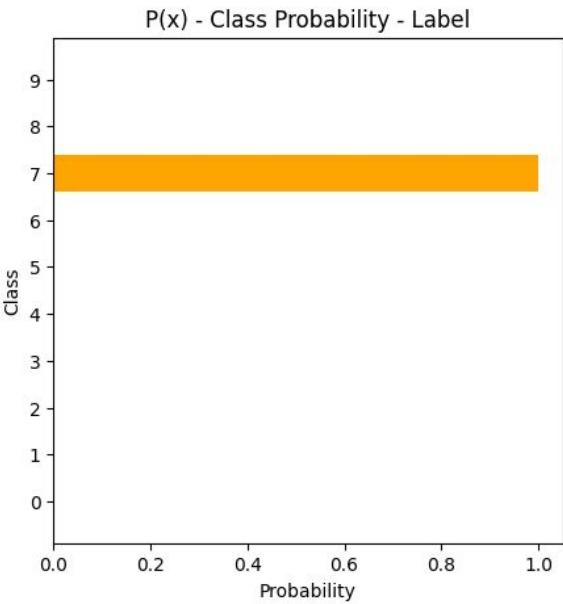
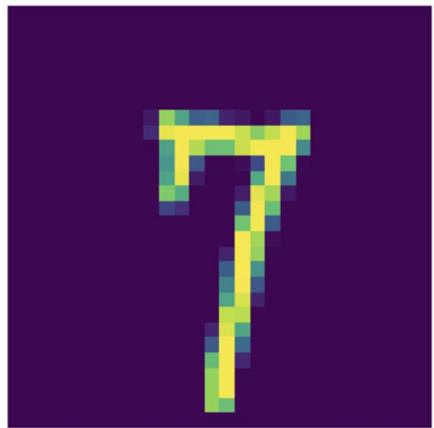


High cross entropy loss - high disagreement between $P(x) = y$ and $Q(x) = \hat{y}$



$$H(p, q) = -\frac{1}{n} \sum_{i=1}^n p(x_i) \log q(x_i) = H(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n y_i \log \hat{y}_i$$

Low cross entropy loss - low disagreement between y and \hat{y}



$$H(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n y_i \log \hat{y}_i$$

Cross Entropy = $-\log(\hat{y}_i)$
with index i being the one true class

Quirks of nn.CrossEntropyLoss

```
# Compute output
logits = model(input_image)

# Turn output into probabilities, useful for interpretation
probabilities = F.softmax(logits)

# CrossEntropyLoss will compute log-softmax on the raw logits
# because it is more numerically stable
ce_loss = nn.CrossEntropyLoss()
ce_loss(logits, labels)
```

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

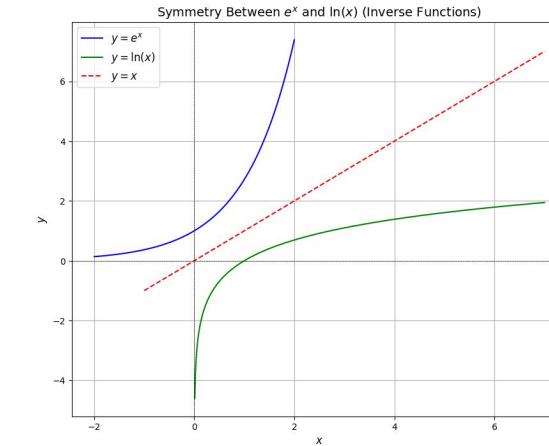
$$\text{LogSoftmax}(\mathbf{x})_i = x_i - \log\left(\sum_{j=1}^n \exp(x_j)\right)$$

Turning LogSoftmax into interpretable probabilities

$$p_i = \exp(\log p_i) = e^{\ln p_i}$$

```
import torch.nn.functional as F
logits = torch.tensor([ 0.7645,  0.8300, -0.2343,
                      0.9186, -0.2191,  0.2018,
                     -0.4869,  0.8000,  0.8815,
                     -0.7336])
log_probabilities = F.log_softmax(logits, dim=0)
print(log_probabilities.sum()) # prints tensor (-24.6857)
probabilities = torch.exp(log_probabilities)
print(probabilities.sum()) # prints tensor(1.00)
```

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



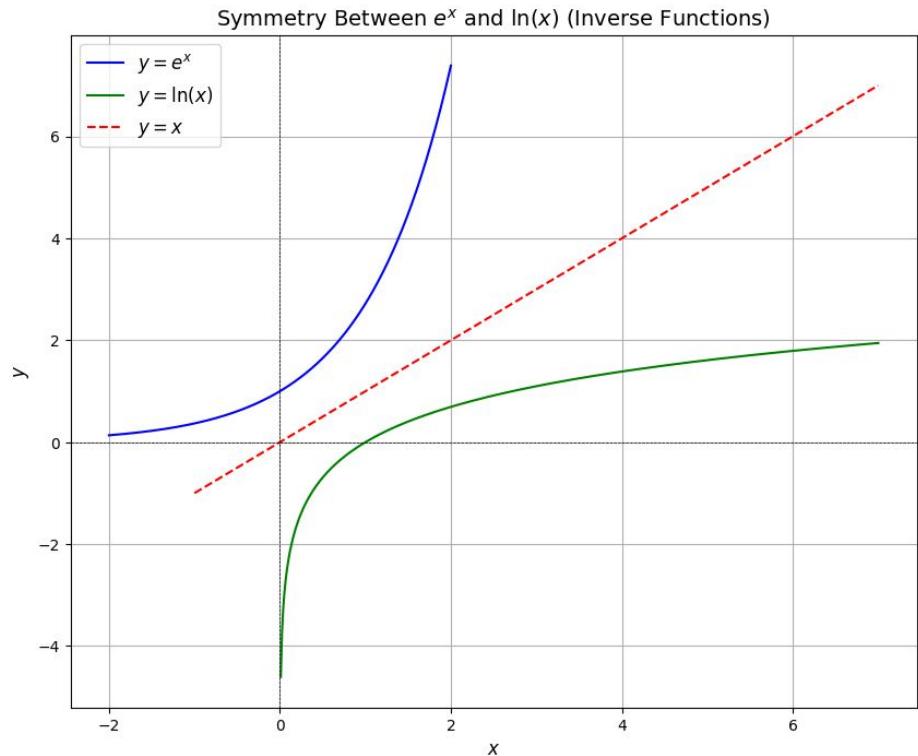
$$\text{LogSoftmax}(\mathbf{x})_i = x_i - \log\left(\sum_j^n \exp(x_j)\right)$$

Why do we use $\log(x) = \ln(x)$?

$$p_i = \exp(\log p_i) = e^{\ln p_i}$$

$$\frac{d}{dx} e^x = e^x$$

$$\frac{d}{dx} \ln(x) = \frac{1}{x}$$



Cross entropy loss from logits

```
import torch
import torch.nn.functional as F

def cross_entropy_loss(logits, labels):
    """
    Calculate cross entropy loss from logits using log softmax

    Args:
        logits: Tensor of shape (batch_size, num_classes) containing raw model
        outputs
        labels: Tensor of shape (batch_size,) containing class indices (0-9 for
        MNIST)

    Returns:
        loss: Scalar tensor with the mean loss
    """
    # Apply log softmax to get log probabilities
    log_probs = F.log_softmax(logits, dim=1)

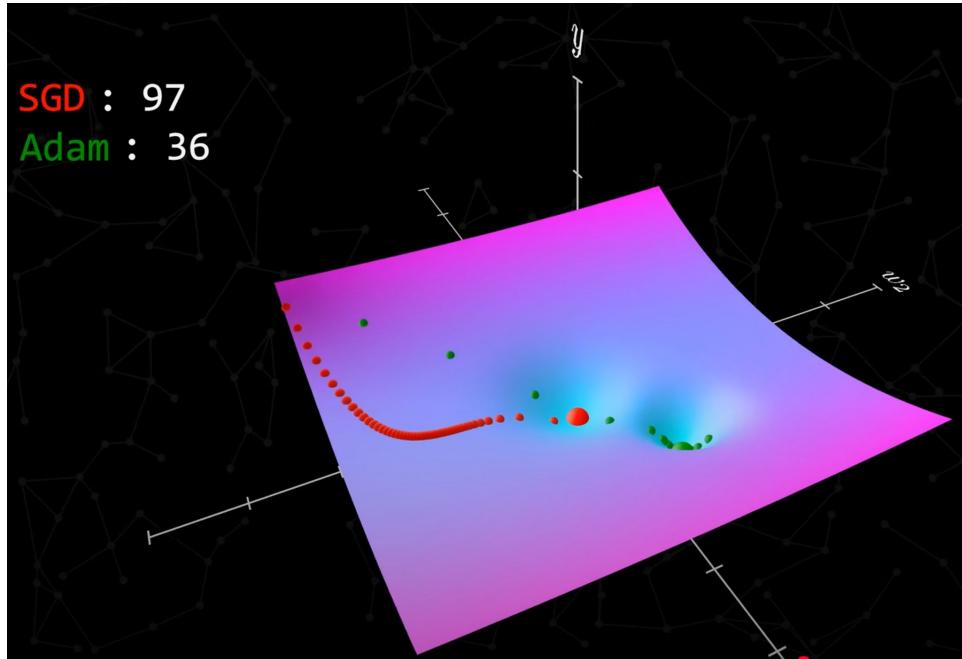
    # Calculate negative log likelihood loss
    # This efficiently computes cross entropy without explicitly creating one-hot
    vectors
    loss = F.nll_loss(log_probs, labels)
```

“negative of the log of the likelihood (of the correct class” = `F.nll_loss`

$$\text{Cross Entropy} = -\log(\hat{y}_i)$$

with index i being the one true class

Extra: Adam vs Stochastic Gradient Descent



Adaptive Gradients with Momentum (ADAM) is a variant of stochastic gradient descent that performs well in many problems.

```
from torch.optim import Adam
optimizer = Adam(
    model.parameters(),
    lr=0.001)

# Compute gradients and update weights
ce_loss.backward()
optimizer.step()
```

Image from "[Who is Adam and what is he optimizing?](#)"

Summary

Feedforward neural networks for image classification

- Convert pixel inputs into class probabilities
- The number of output units determines the number of scores that we can produce
- The Adam optimizer is a variant of SGD that works well in practice

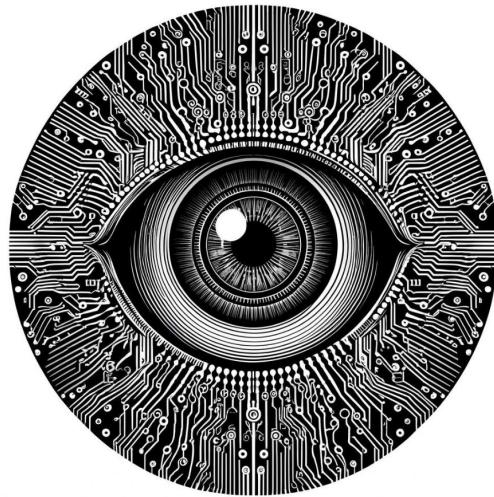
Classification pipeline

- We use one-hot encoding when we want to predict a single label per image
- We convert the raw output scores (logits) to probabilities using softmax

Cross entropy loss

- Measures classification prediction quality (agreement of probabilities in \hat{y} and y)
- Used with log-softmax in PyTorch for numerical stability

Usage of Binary Cross Entropy Loss

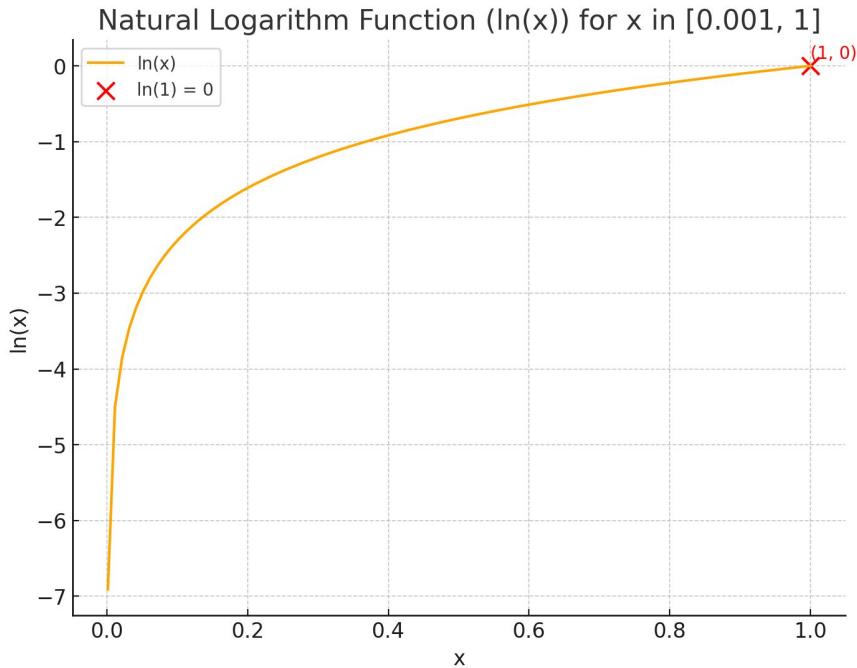


Antonio Rueda-Toicen

Learning goals

- Encode class labels as probability distributions
- Understand differences between categorical and binary cross entropy loss
- Describe differences between sigmoid and softmax activation functions
- Design network architectures with agreement between the classification task, output layer, activation function, and loss function

The logarithm function and cross entropy



$$H(y, \hat{y}) = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

Predicting a single label from multiple classes with a neural network

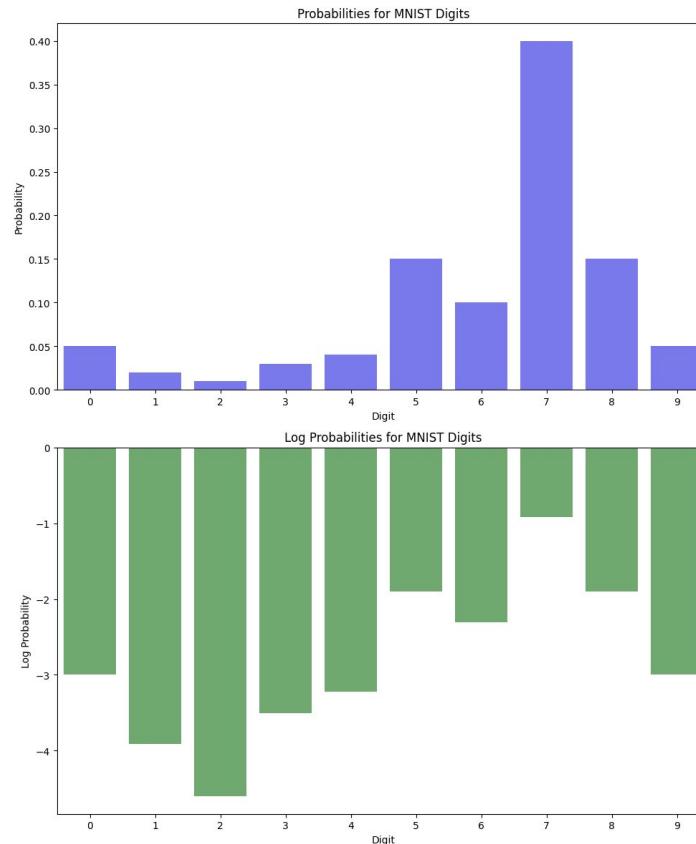
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$



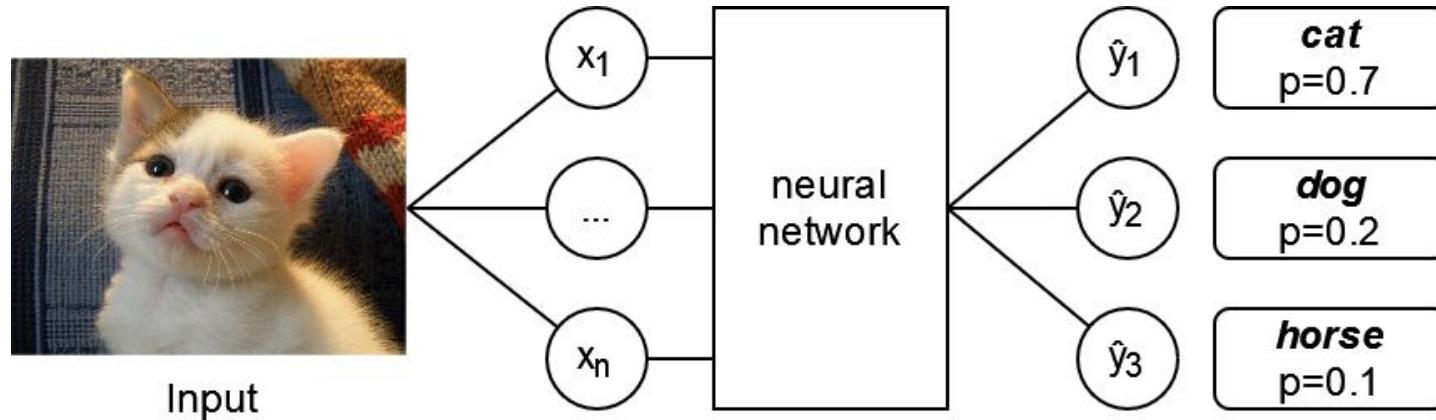
MNIST has a single label per image

$$\log \text{softmax}(x_i) = x_i - \log \sum_{j=1}^n \exp(x_j)$$

$$H(y, \hat{y}) = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$



Single label classification: n probabilities for n classes



```
model = nn.Sequential(  
    # We get raw logits as output  
    nn.Linear(input_size, 3),  
)  
  
criterion = nn.CrossEntropyLoss()  
loss = criterion(predictions, targets)
```

$$P(\hat{y}_{\text{cat}}) + P(\hat{y}_{\text{dog}}) + P(\hat{y}_{\text{horse}}) = 1$$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

Binary Cross Entropy

Used to model events such as

$$P(A) = 1 - P(B)$$

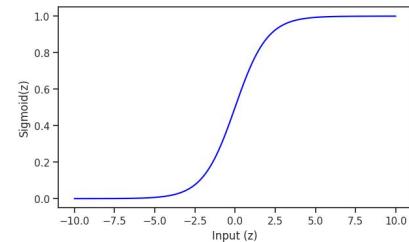
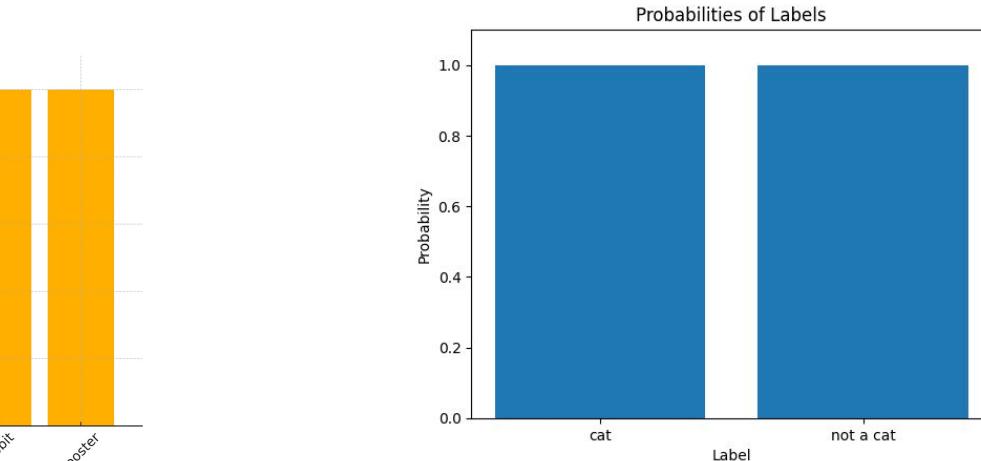
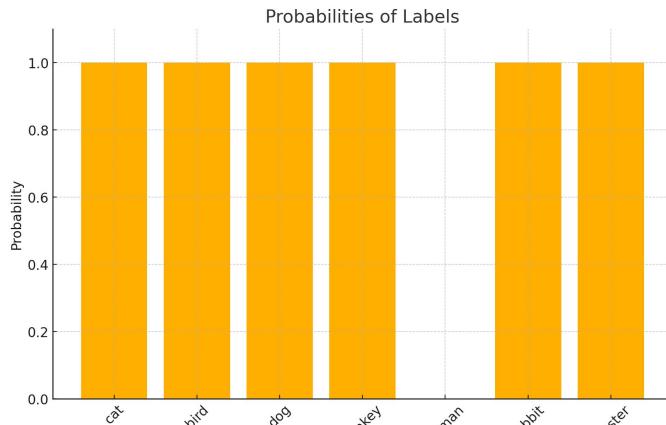


$$\text{BinaryCrossEntropy}(y, \hat{y}) = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

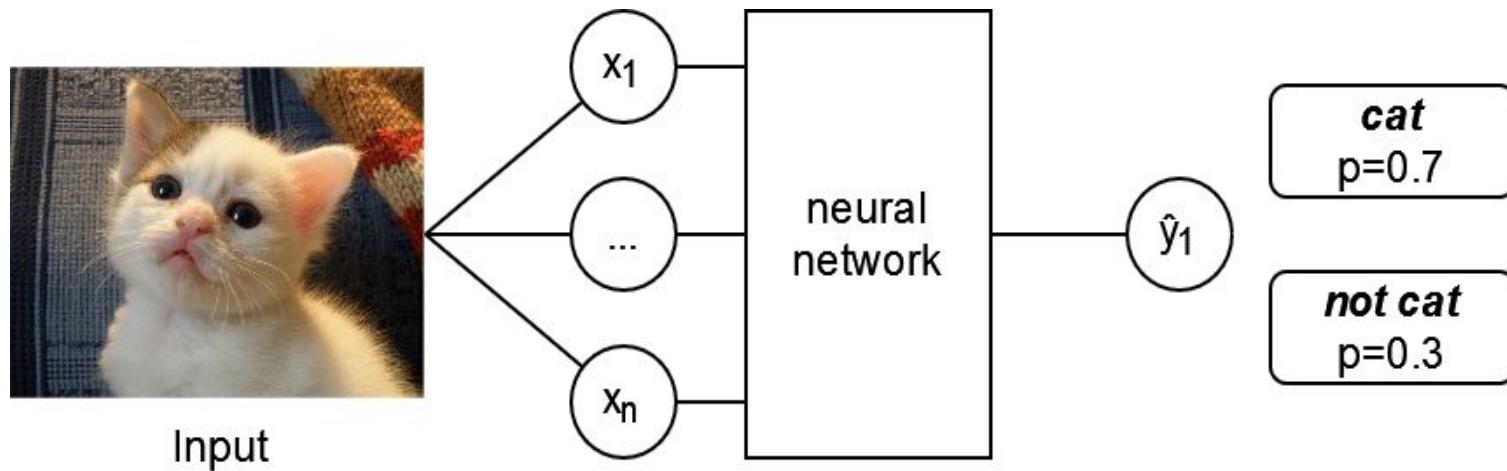
Compare with categorical cross entropy
when having a single class and a single data point

$$H(y, \hat{y}) = -y \log(\hat{y})$$

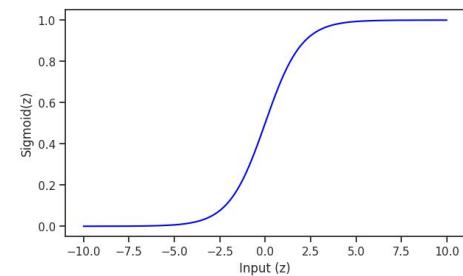
Encoding class labels as probability distributions for binary and multilabel classification



Single label binary classifiers



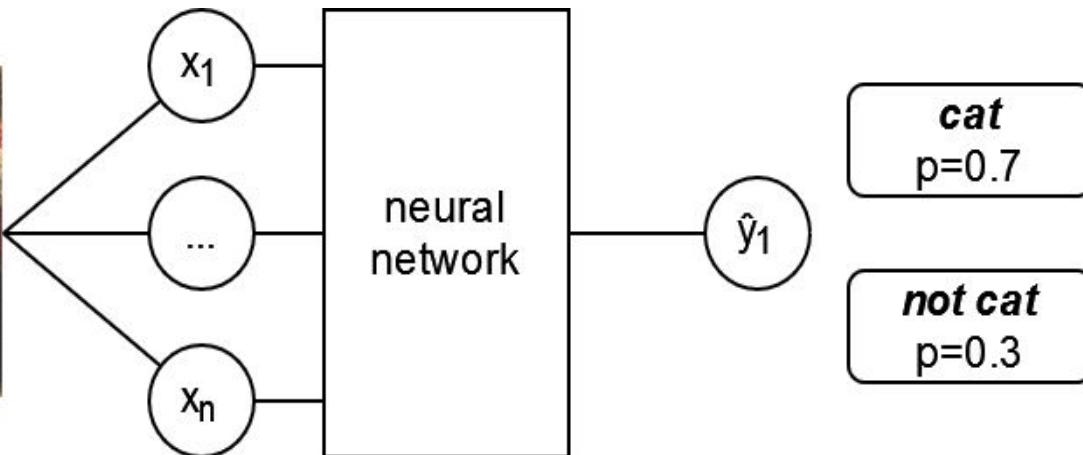
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = w^T x + b$$



Single label binary classifiers in PyTorch



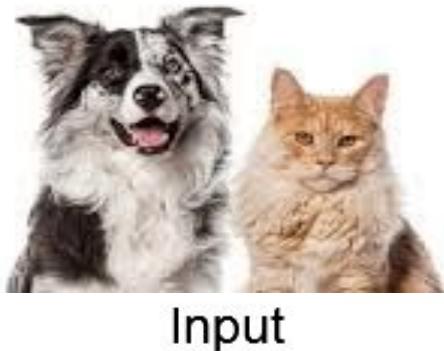
Input



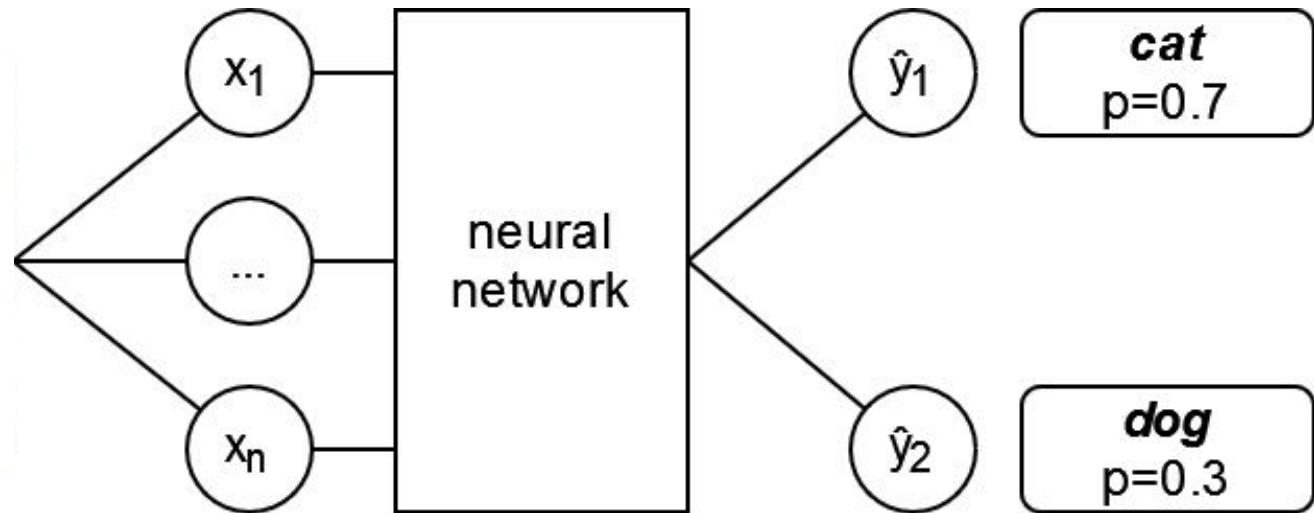
```
model = nn.Sequential(  
    # Raw logits output  
    nn.Linear(input_size, 1) )  
  
criterion = nn.BCEWithLogitsLoss()  
loss = criterion(logits, targets)
```

```
model = nn.Sequential(  
    nn.Linear(input_size, 1),  
    # Convert to probabilities  
    nn.Sigmoid()  
)  
  
criterion = nn.BCELoss()  
loss = criterion(predictions, targets)
```

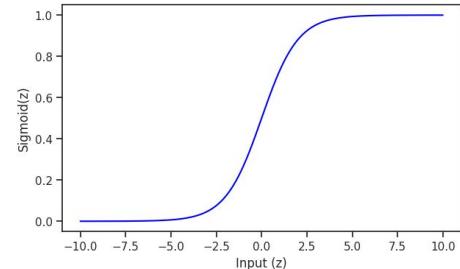
Multilabel classifiers



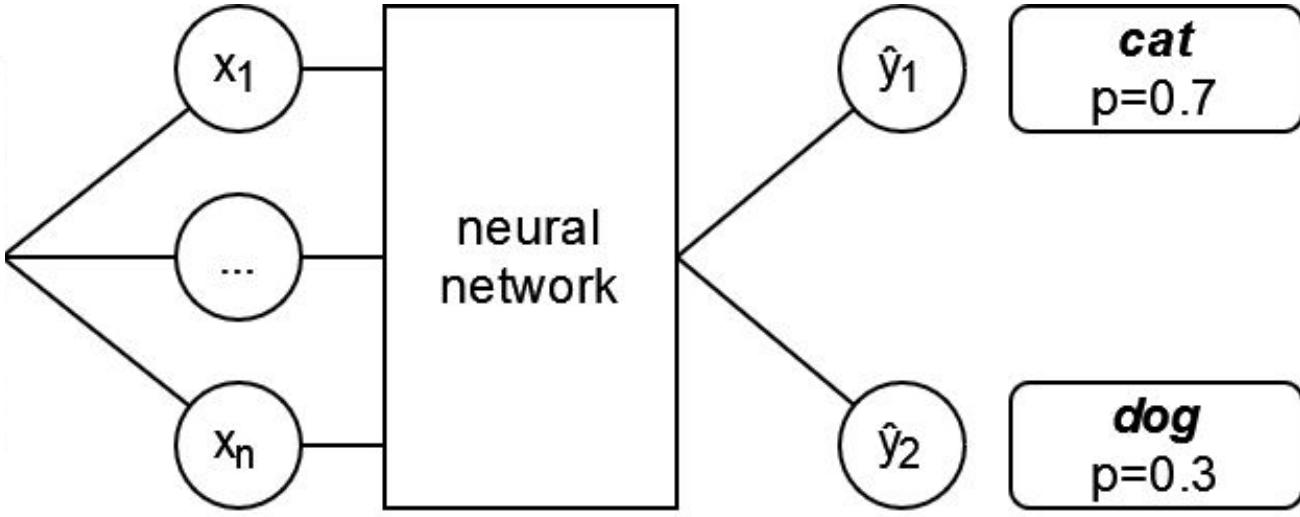
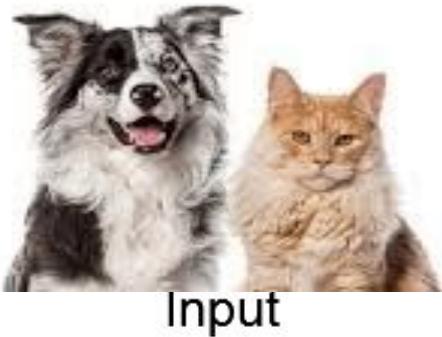
Input



$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = w^T x + b$$



Multilabel classifiers in PyTorch



```
model = nn.Sequential(  
    nn.Linear(input_size, num_classes),  
    # Convert to probabilities  
    nn.Sigmoid()  
)  
  
criterion = nn.BCELoss()  
loss = criterion(predictions, targets)
```

```
model = nn.Sequential(  
    # Raw logits  
    nn.Linear(input_size, num_classes),  
  
    criterion = nn.BCEWithLogitsLoss()  
    loss = criterion(predictions, targets)
```

Multilabel classification on satellite images



PLANET · FEATURED PREDICTION COMPETITION · 7 YEARS AGO

Planet: Understanding the Amazon from Space

Use satellite data to track the human footprint in the Amazon rainforest

<https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/>

Multilabel classification on satellite images



<https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/>

Summary

Cross entropy and the final activation function

- Cross entropy measures the mismatch between predicted and true distributions
- Softmax handles single-label problems with multiple classes
- Sigmoid handles both binary and multi-label cases

Network design

- Match output layer size to number of classes
- Consider single-label vs multi-label requirements for final layer
- Choose activation function based on classification type (e.g. single label vs multi-label)

Different tasks require different variants of cross entropy loss

- Use `nn.BCELoss` or `nn.BCEWithLogitsLoss` for binary or multi-label classification tasks
- Use `nn.CrossEntropyLoss` for single label tasks

Further reading

LogSoftmax vs Softmax

- <https://discuss.pytorch.org/t/logsoftmax-vs-softmax/21386>

Resources

- [Github Repository](#)
- [YouTube playlist](#)
- [Discord channel](#)

#practical-computer-vision-workshops