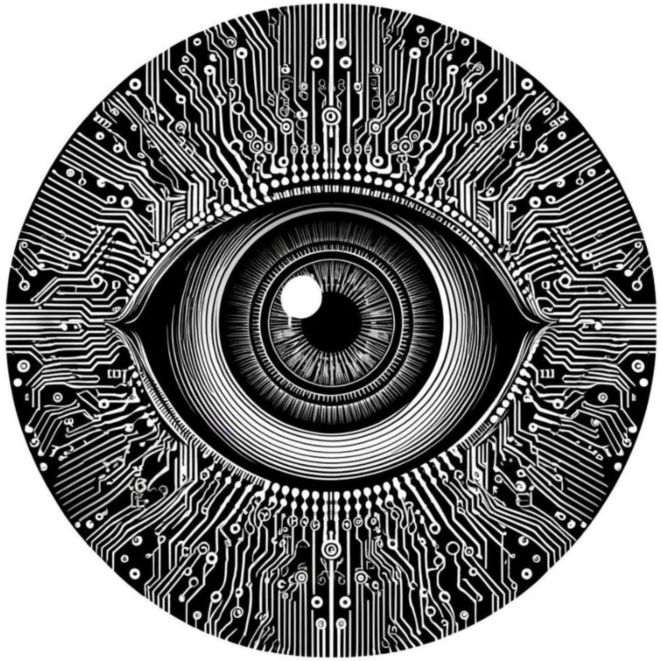
Building a Feedforward Network for Classification in PyTorch



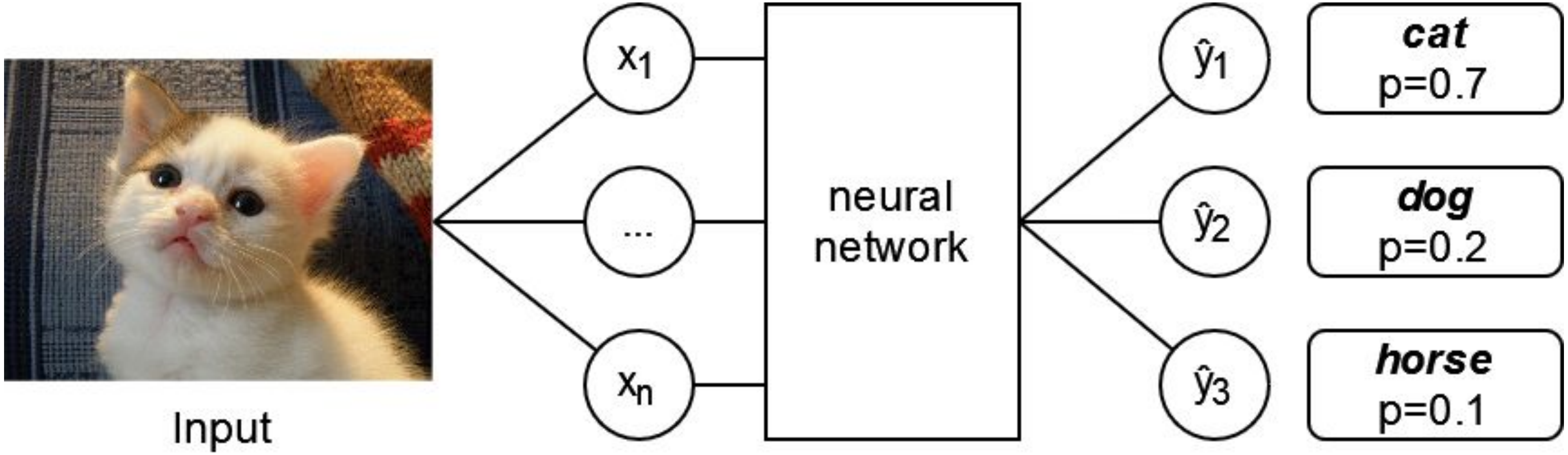
Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry of Education and Research

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



$$i = \begin{cases} 1 & \text{cat} \\ 2 & \text{dog} \\ 3 & \text{horse} \end{cases}$$

Classification

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

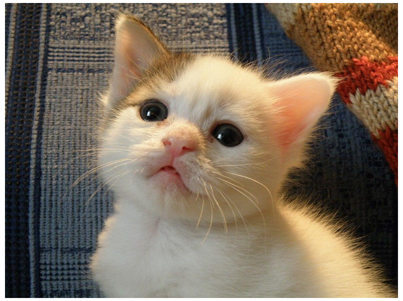
Cross entropy loss

$\boxed{y = 1.0}$ $\boxed{\hat{y} = 0.75}$

$$H(p,q) = -\sum_i p(i) \log q(i)$$

Labels y_i map to p(i), predictions y_i map to q(i). Suppose that we show the network, only the following image (n=1).

“one hot encoding” 🔥 = only one true label

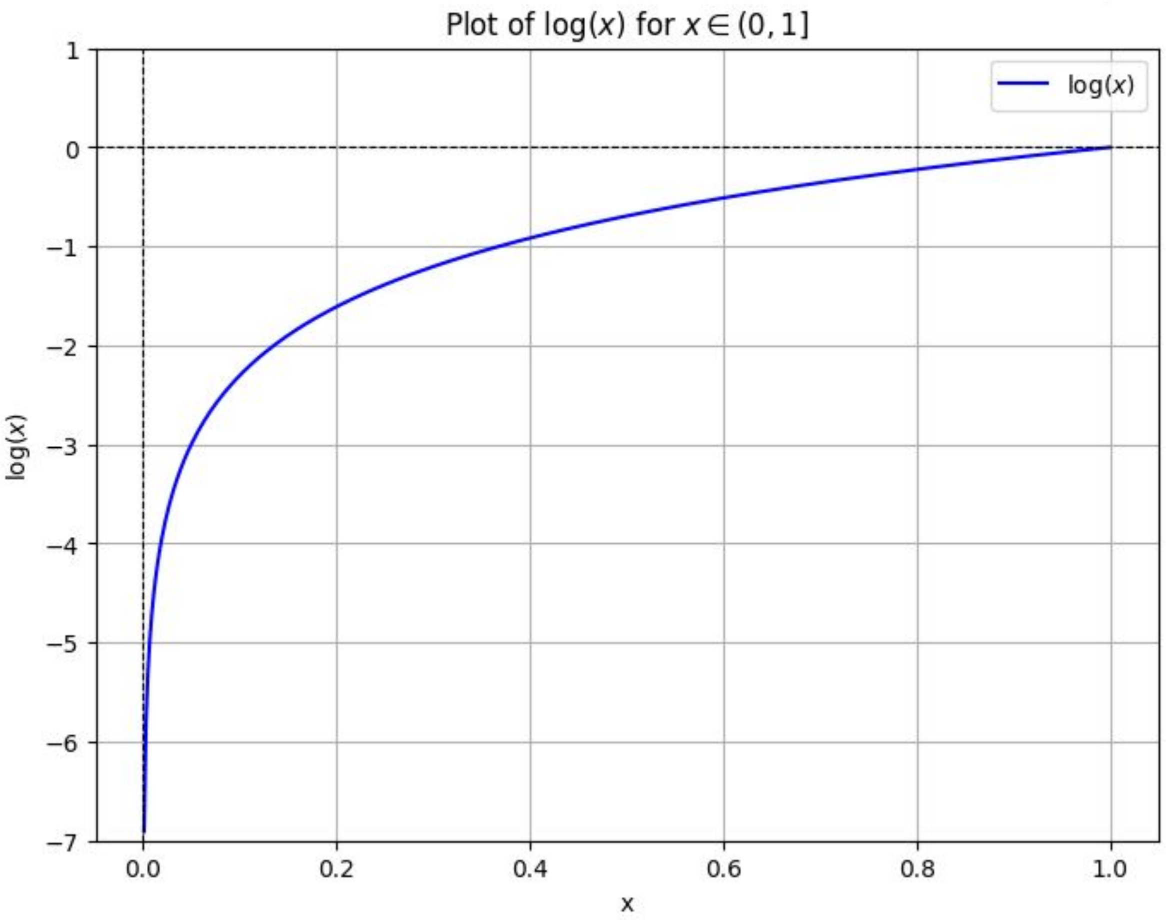


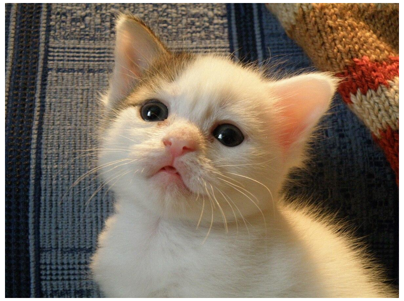
The label for the image is encoded as y = [1, 0, 0] (i = 0 = cat, i = 1 = dog, i = 2 = horse)

- First the network outputs **ŷ = [0.75, 0.25, 0.0]**
- Then the network outputs **ŷ = [0.99, 0.01, 0.0]** **<u>(if trained correctly)</u>**

Intuition for the log function in cross entropy loss

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





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

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

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

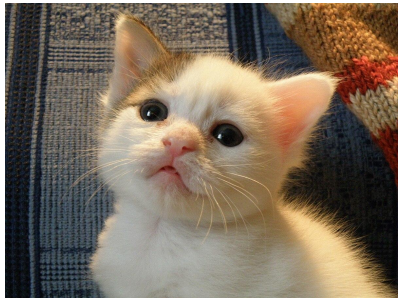
$$y_0 = 1 \quad \text{and} \quad \hat{y}_0 = 0.01$$

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

Note that we use indexing starting at 0 for the labels here, as in Python

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
```

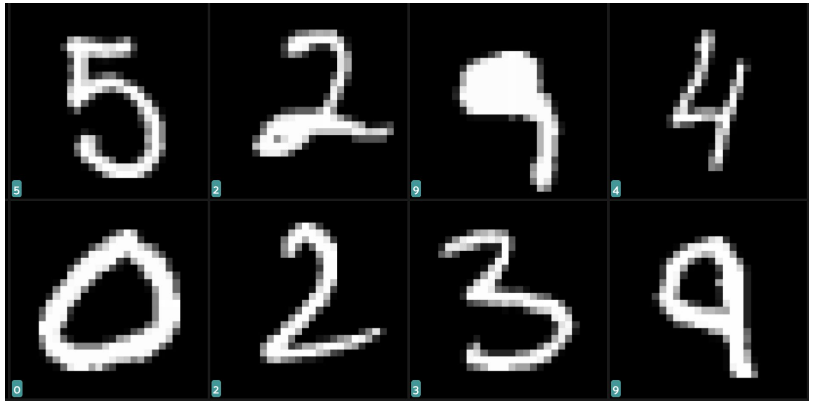


`y = [1, 0, 0]` `y_hat = [0.75, 0.25, 0.0`

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

A minimal network for classification

```python
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)
)
```



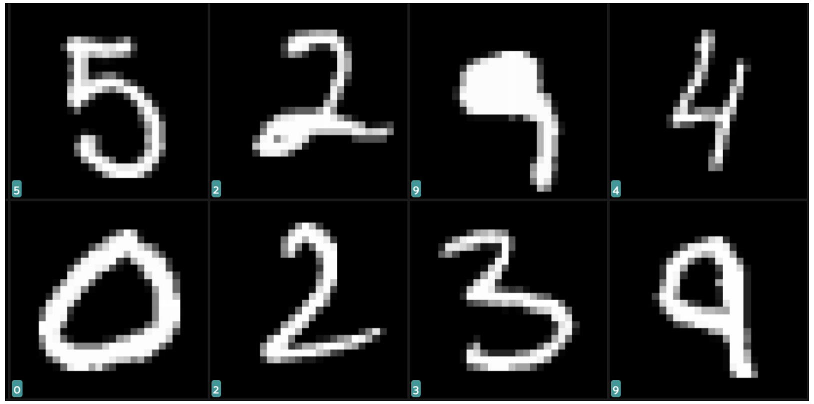
Tip: do <u>not</u> confuse these “logits” with the function described on https://en.wikipedia.org/wiki/Logit

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

these “logits” are **not** this ^

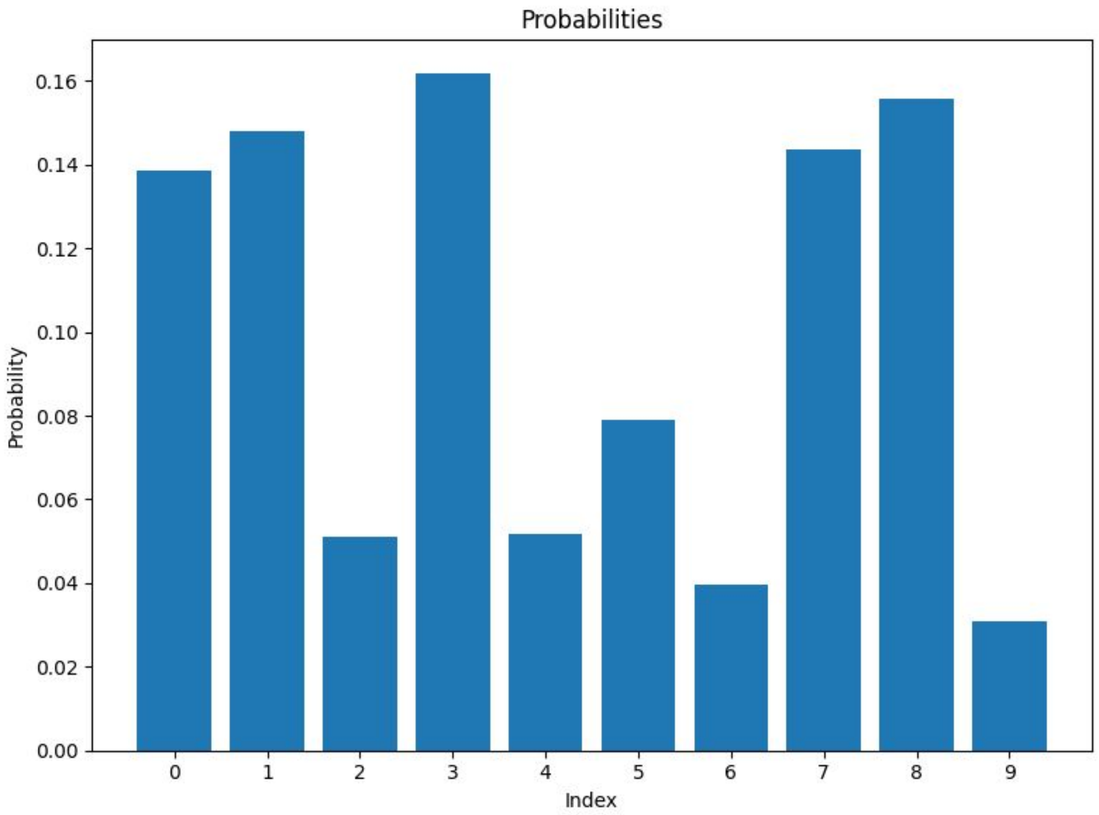
Using the nn.Module syntax

```python
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()
```

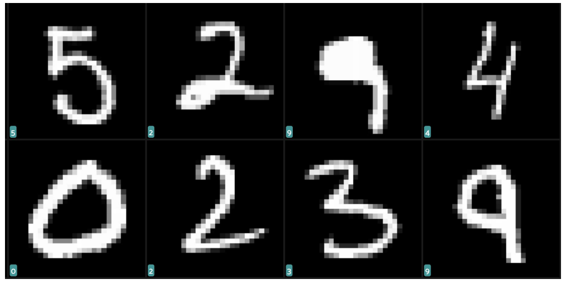


The softmax function

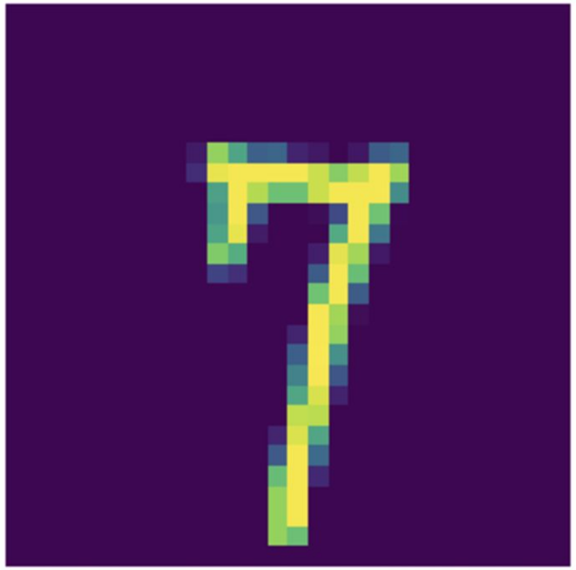
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

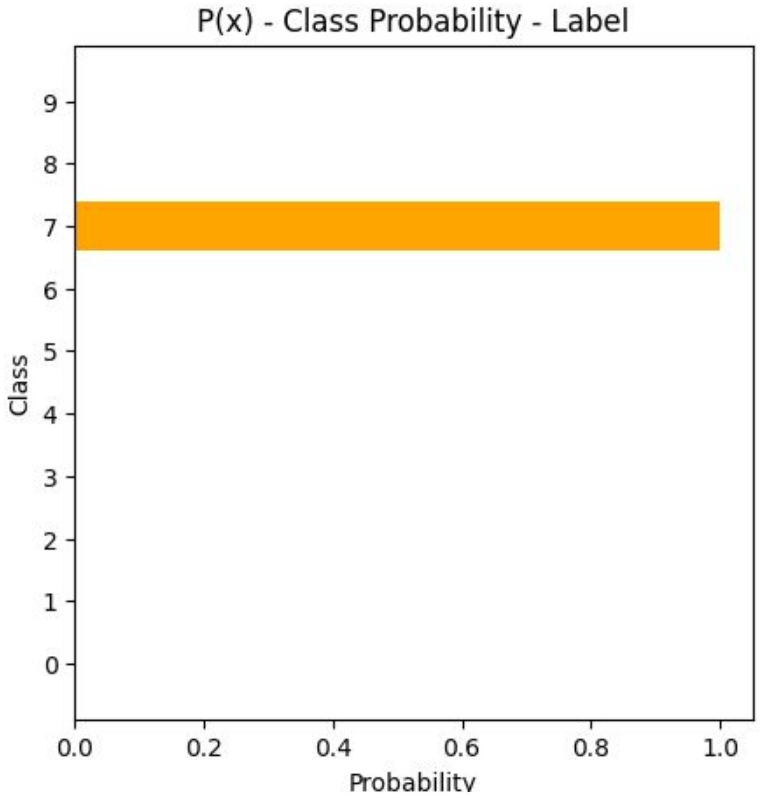


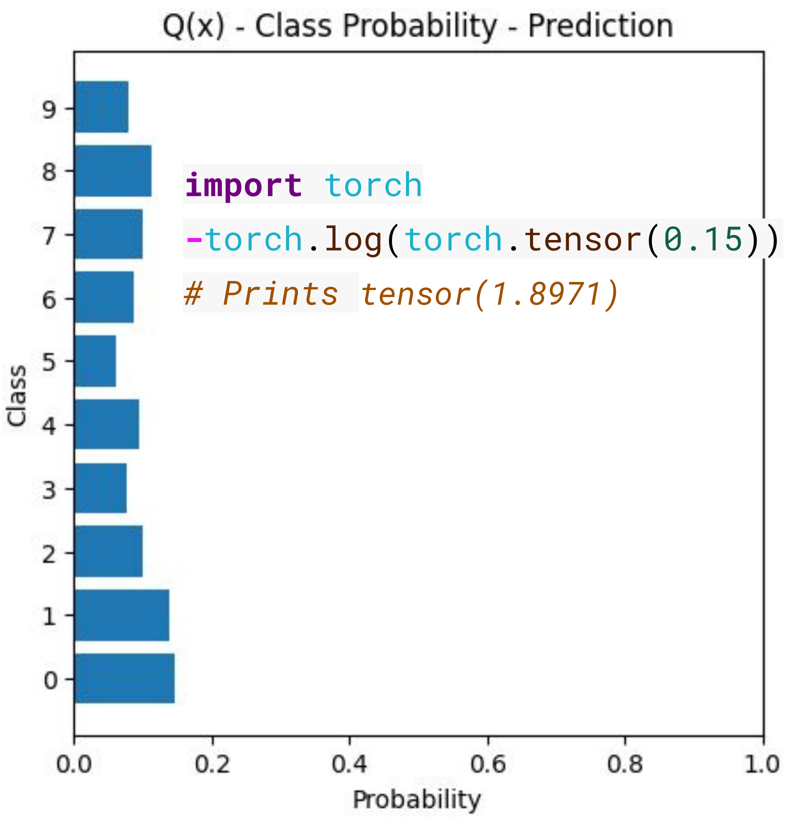

```python
import torch.nn.functional as F
logits = torch.tensor([ 0.7645,  0.8300, -0.2343,  0.9186, -0.2191,  0.2018, -0.4869,  0.8000, 0.8815, -0.7336])
probabilities = F.softmax(logits, dim=0)
print(probabilities.sum()) # prints tensor(1.00)
```



High cross entropy loss - high disagreement between P(x) = y and Q(x) = ŷ



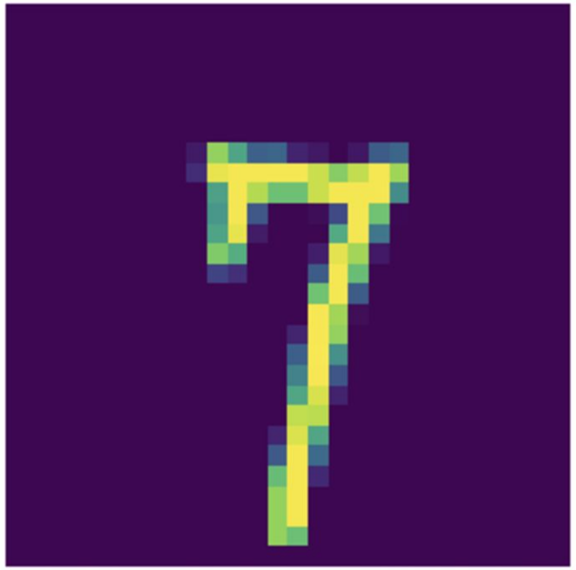


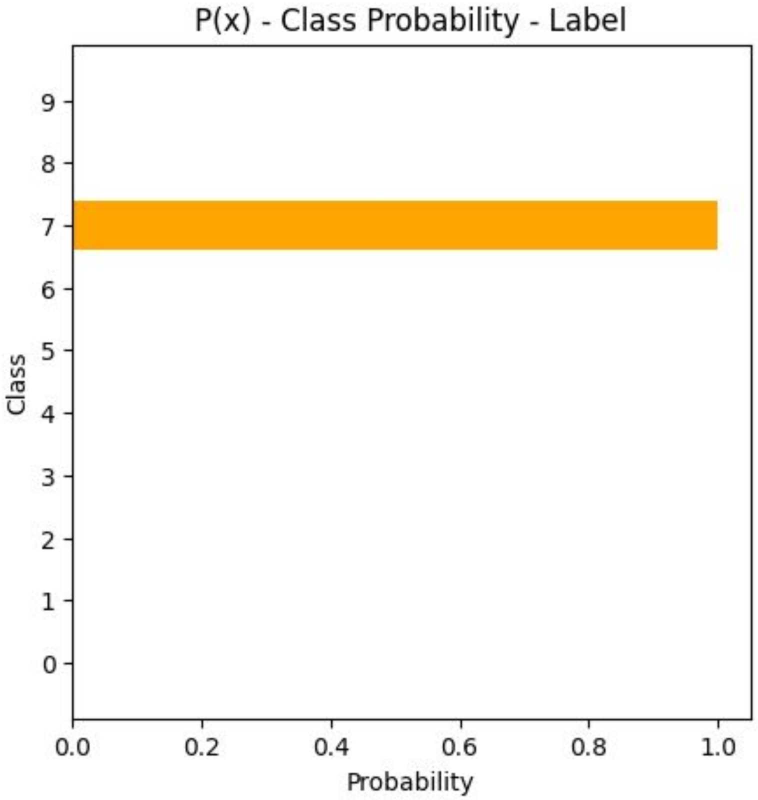


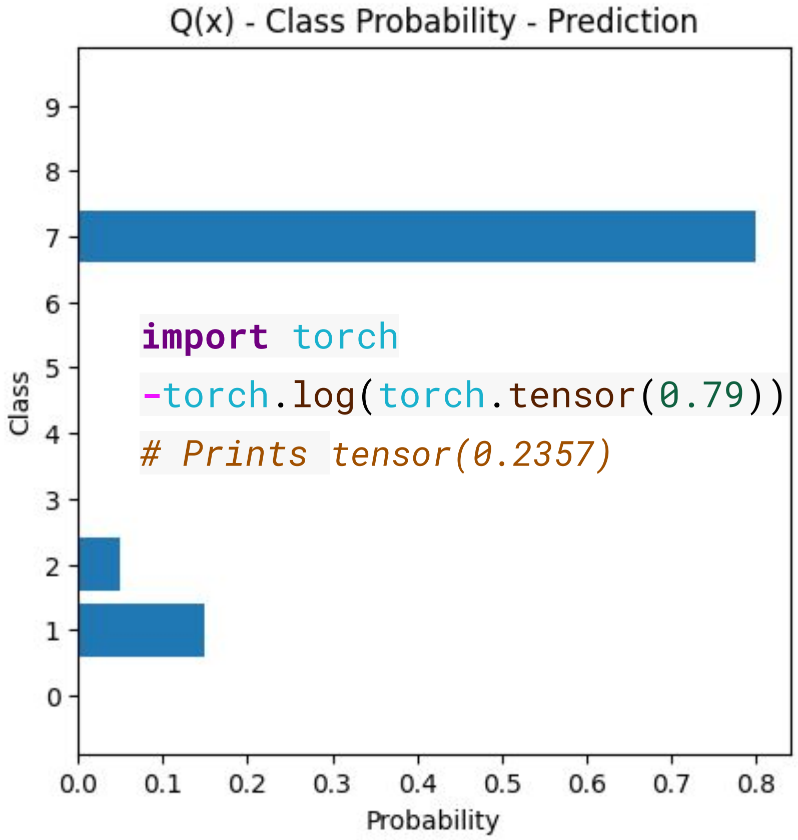

```python
import torch
-torch.log(torch.tensor(0.15))
# Prints tensor(1.8971)
```

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

Low cross entropy loss - low disagreement between y and ŷ







$$H(p,q) = -\sum_i p(i) \log q(i)$$

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

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)
```

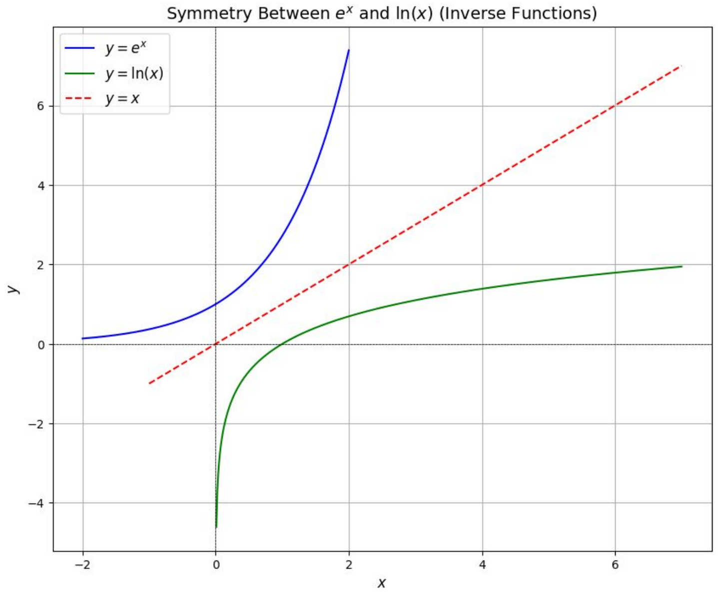
$$\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$$

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

Turning LogSoftmax into interpretable probabilities

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

```python
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)
```



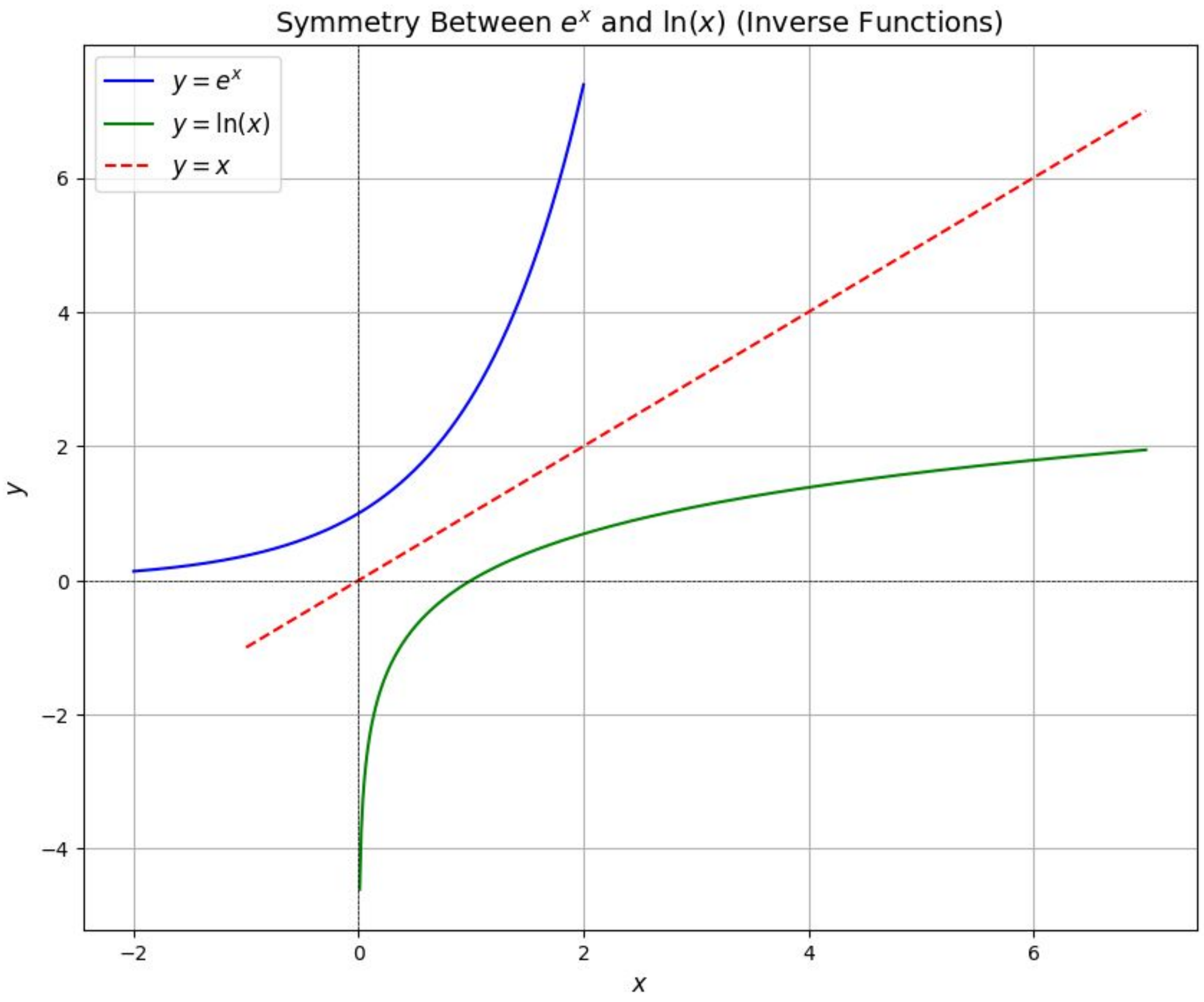
$$\text{LogSoftmax}(x)_i = \log\left(\frac{\exp(x_i)}{\sum_{j=1}^{n}\exp(x_j)}\right) = x_i - \log\left(\sum_{j=1}^{n}\exp(x_j)\right)$$

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

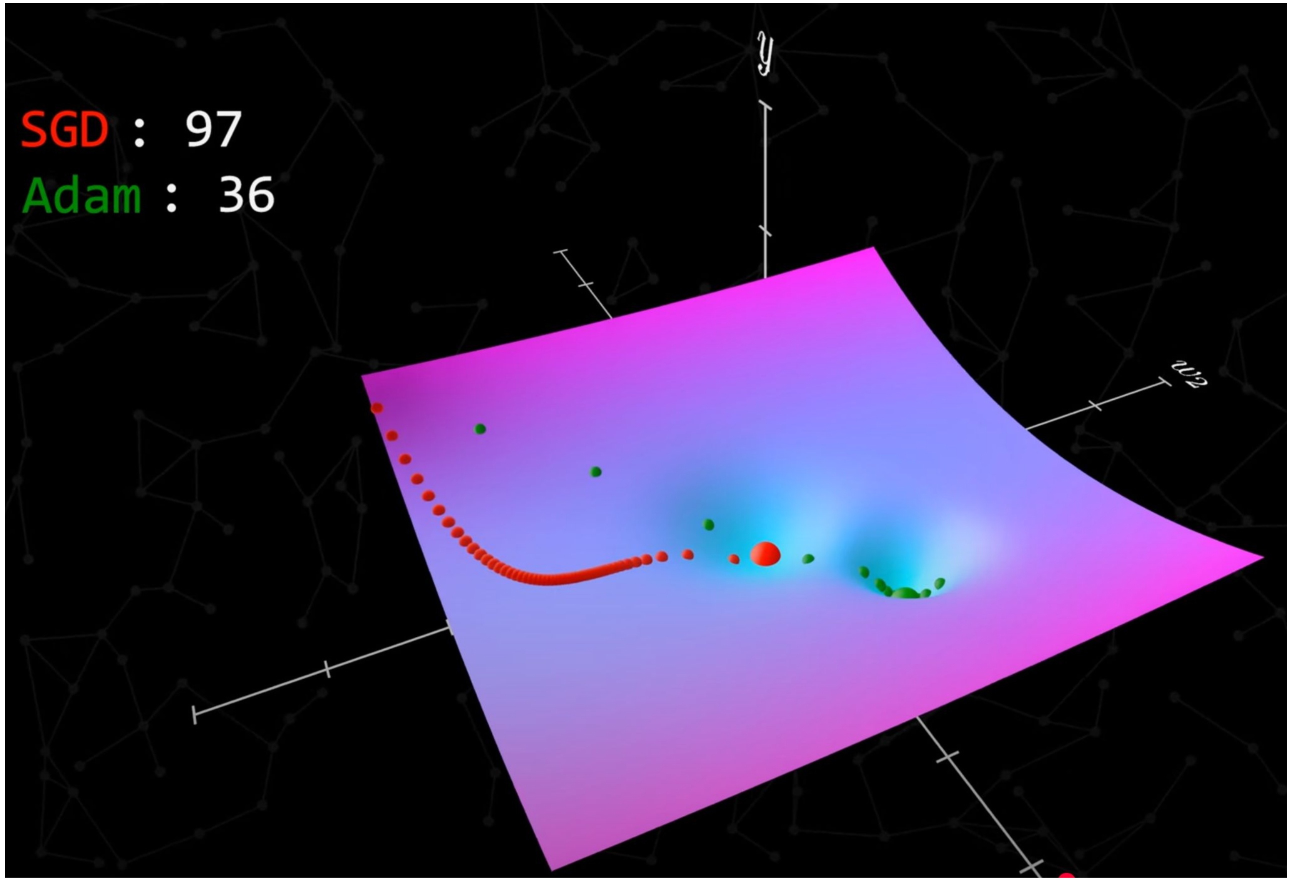
```python
import torch
import torch.nn.functional as F

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)

    return loss
```

"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
variant of stochastic gradient descent that per
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?"

KI Service Zentrum by Hasso-Plattner-Institut HPI

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 y hat and y)
- Used with log-softmax in PyTorch for numerical stability

SPONSORED BY THE
Federal Ministry of Education and Research

Further reading

PyTorch’s CrossEntropyLoss

- https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Who is Adam and what is he optimizing?

- https://www.youtube.com/watch?v=MD2fYip6QsQ

SPONSORED BY THE
Federal Ministry
of Education
and Research