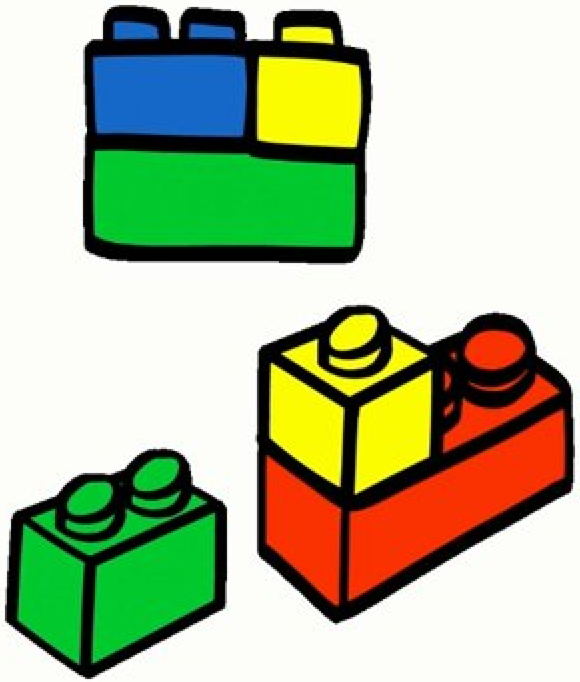
IMPORTANT ELEMENTS OF NEURAL NETWORK

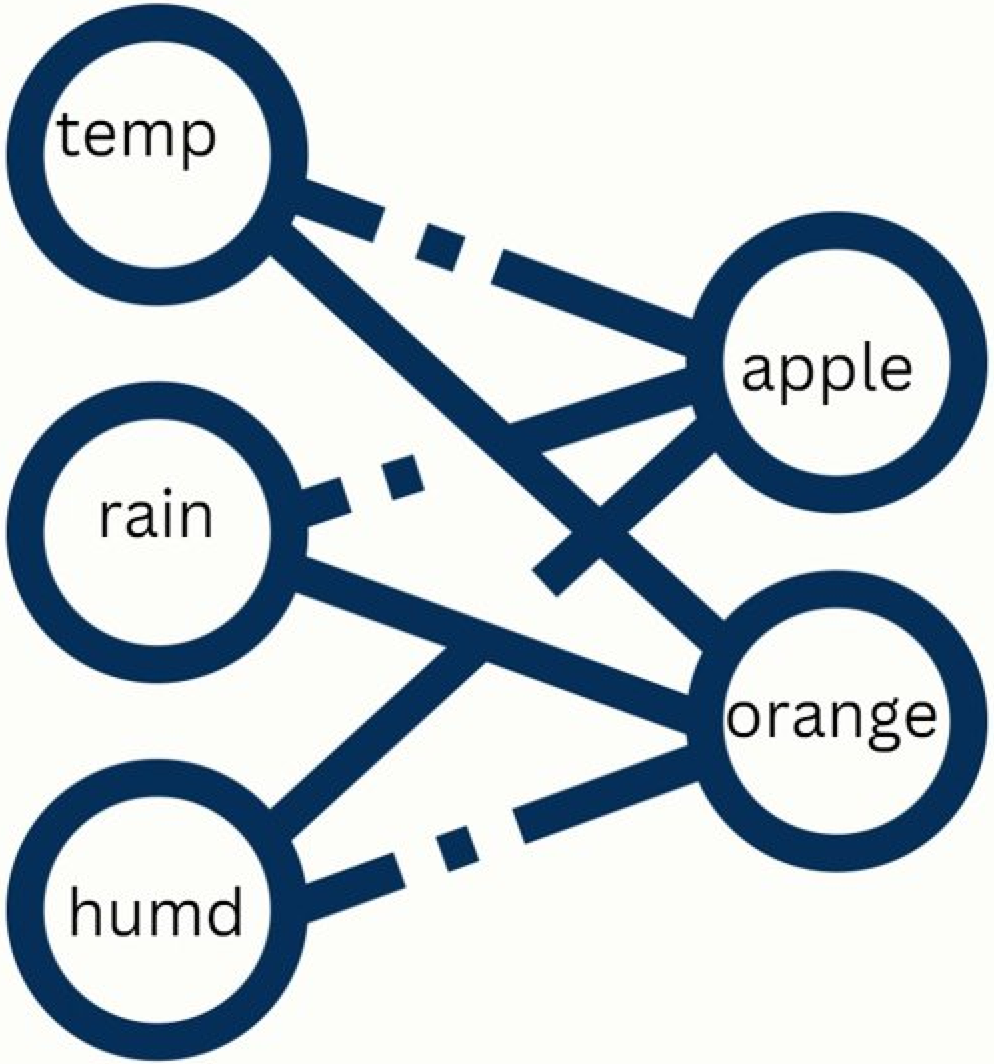
- **Loss Function**
- **Optimizer**
- **Activation Function**

IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function
- Optimizer
- Activation Function



SIMPLE NEURAL NETWORK



$$X * W + b$$

Pytorch.ipynb - Colab

Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=jIF12AOVj1Kt

+ Code + Text All changes saved

```
[1]                          [119, 133],
                             [22, 37],
                             [103, 119],
                             [98, 110],
                             [88, 95],
                             [115, 140],
                             [76, 85],
                             [65, 75]], dtype='float32')
```

```
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)
print(inputs)
print(targets)
```

```
tensor([[ 73.,  67.,  43.],
        [ 91.,  88.,  64.],
        [ 87., 134.,  58.],
        [102.,  43.,  37.],
        [ 69.,  96.,  70.],
        [ 85., 100.,  60.],
        [ 95.,  80.,  55.],
        [105., 120.,  75.],
        [ 78.,  90.,  50.],
        [ 82.,  70.,  45.]])
tensor([[ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
        [103., 119.],
        [ 98., 110.],
        [ 88.,  95.],
```


Connected to Python 3 Google Compute Engine backend (GPU)

Pytorch.ipynb - Colab | Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=6DG1zS7iKBCP

+ Code + Text All changes saved

[2]

```
        [105., 120.,  75.],
        [ 78.,  90.,  50.],
        [ 82.,  70.,  45.]])
tensor([[ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
        [103., 119.],
        [ 98., 110.],
        [ 88.,  95.],
        [115., 140.],
        [ 76.,  85.],
        [ 65.,  75.]])
```

```python
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)
print(w)
print(b)
```

```
tensor([[ 0.1674,  0.2380],
        [ 0.8814, -0.4406],
        [ 0.9254,  0.4304]], requires_grad=True)
tensor([ 1.2801, -0.7225], requires_grad=True)
```

[9]

```python
# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
```

Connected to Python 3 Google Compute Engine backend (GPU)

Pytorch.ipynb - Colab
Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=ecfwVB8-W3JY

+ Code + Text All changes saved

```
[ 0.9234,  0.4304]], requires_grad=True)
tensor([ 1.2801, -0.7225], requires_grad=True)
```

```
# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
def mse(p1, p2):
  diff = p1-p2
  return torch.sum(diff**2)/ diff.numel()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
loss = mse(preds, targets)
print(loss)
```

Connected to Python 3 Google Compute Engine backend (GPU)

```
167.897216796875
97.26914978027344
63.02490234375
46.403343200683594
38.319190979003906
34.37253952026367
32.43240737915039
31.466655731201172
30.975189208984375
30.71561050415039
tensor(30.7137, grad_fn=<DivBackward0>)
```

Basic Neural Network with Built-in functions

> Using Pytorch

↳ 37 cells hidden

Pytorch.ipynb - Colab

Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=Oor1qUGUqzSO

+ Code + Text All changes saved

```python
import torch
import numpy as np

# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43],
                   [91, 88, 64],
                   [87, 134, 58],
                   [102, 43, 37],
                   [69, 96, 70],
                   [85, 100, 60],
                   [95, 80, 55],
                   [105, 120, 75],
                   [78, 90, 50],
                   [82, 70, 45]], dtype='float32')

# Targets (apples, oranges)
targets = np.array([[56, 70],
                    [81, 101],
                    [119, 133],
                    [22, 37],
                    [103, 119],
                    [98, 110],
                    [88, 95],
                    [115, 140],
                    [76, 85],
                    [65, 75]], dtype='float32')
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)
print(inputs)
print(targets)
```

Connected to Python 3 Google Compute Engine backend (GPU)

```
[ 65.,  75.]])
```

```python
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
def mse(p1, p2):
  diff = p1-p2
  return torch.sum(diff**2)/ diff.numel()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()
```

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Pytorch.ipynb - Colab
Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text All changes saved

```
[ 65.,  75.]])
```

```python
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
def mse(p1, p2):
  diff = p1-p2
  return torch.sum(diff**2)/ diff.numel()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()
```


0s completed at 3:02 PM

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Pytorch.ipynb - Colab
Loss Functions.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text All changes saved

```
[ 65.,  75.]])
```

```python
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
def mse(p1, p2):
  diff = p1-p2
  return torch.sum(diff**2)/ diff.numel()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()
```


0s completed at 3:02 PM

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Loss Functions

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical	Multiclass Classification with	Efficient for large number of classes, avoids one-hot	Similar issues as categorical cross-entropy

Loss Function	Use Case	Pros	Cons
Mean Squared Error (MSE)	Regression	Simple to compute and differentiate.	Sensitive to outliers, can lead to slow convergence.
Mean Absolute Error (MAE)	Regression	Robust to outliers.	Less smooth gradients compared to MSE, can be slower to converge.
Huber Loss	Regression (robust to outliers)	Balances sensitivity and robustness to outliers.	Requires tuning of the hyperparameter δ\deltaδ.
Cross-Entropy Loss	Classification (binary and multiclass)	Effective for classification tasks, especially with softmax output.	Can be sensitive to class imbalance.
Binary Cross-Entropy	Binary Classification	Suitable for binary classification, handles probabilities well.	Can suffer from vanishing gradients for extreme predictions.
Categorical Cross-Entropy	Multiclass Classification	Standard for multiclass classification with one-hot encoded labels.	Assumes mutually exclusive classes, not suitable for multi-label classification.
Sparse Categorical Cross-Entropy	Multiclass Classification with integer labels	Efficient for large number of classes, avoids one-hot encoding.	Similar issues as categorical cross-entropy with class imbalance.
Hinge Loss	Support Vector Machines (SVMs)	Good for maximum margin classifiers,	Not differentiable at the margin, less commonly used in deep learning.

Google

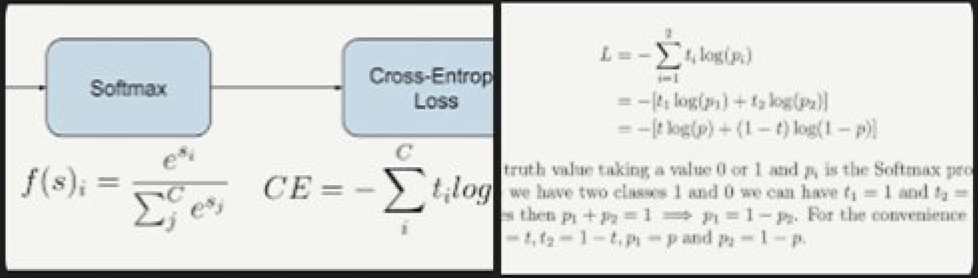
Cross-EntropyLoss

All Images Videos Shopping News Books Web More Tools

Showing results for Cross-*Entropy Loss*
Search instead for Cross-EntropyLoss

اردو میں | In English

Cross-entropy, also known as logarithmic loss or log loss, is a popular loss function used in machine learning to measure the performance of a classification model. Namely, it measures the difference between the discovered probability distribution of a classification model and the predicted values.



DataCamp
https://www.datacamp.com › ... › Machine Learning

Cross-Entropy Loss Function in Machine Learning - DataCamp

About featured snippets • Feedback

People also ask

Cross-entropy

• Mean Absolute Error (MAE): Measures the average absolute differences between predicted and actual values. Less sensitive to outliers compared to MSE.

- PyTorch Syntax: `torch.nn.L1Loss()`

• Huber Loss: A combination of MSE and MAE, providing robustness to outliers and smooth gradients.

- PyTorch Syntax: `torch.nn.SmoothL1Loss()`

• Cross-Entropy Loss: Measures the difference between two probability distributions, commonly used for classification tasks.

- PyTorch Syntax: `torch.nn.CrossEntropyLoss()`

• Binary Cross-Entropy: A special case of cross-entropy for binary classification problems.

- PyTorch Syntax: `torch.nn.BCELoss()`

• Categorical Cross-Entropy: Used for multiclass classification where each output class is one-hot encoded.

- PyTorch Syntax: `torch.nn.CrossEntropyLoss()`

• Sparse Categorical Cross-Entropy: Similar to categorical cross-entropy but uses integer labels for classes.

- PyTorch Syntax: `torch.nn.CrossEntropyLoss() (with integer labels)`

• Hinge Loss: Used primarily for training Support Vector Machines (SVMs)

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=B6Ymnu54qzSb

+ Code + Text All changes saved

```
[ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#MSE Loss function
def mse(p1, p2):
  diff = p1-p2
  return torch.sum(diff**2)/ diff.numel()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
```


0s completed at 3:02 PM

	(imbalanced data)	hard examples.	y\gammay, requires tuning.

- Mean Squared Error (MSE): Measures the average squared differences between predicted and actual values. Widely used in regression tasks.
  - PyTorch Syntax: `torch.nn.MSELoss()`
- Mean Absolute Error (MAE): Measures the average absolute differences between predicted and actual values. Less sensitive to outliers compared to MSE.
  - PyTorch Syntax: `torch.nn.L1Loss()`
- Huber Loss: A combination of MSE and MAE, providing robustness to outliers and smooth gradients.

Pytorch.ipynb - Colab
Loss Functions.pdf
cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text

```
       [ 65.,  75.]])
```

```python
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in func


# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
loss = mse(preds, targets)
print(loss)
```


✓ 0s completed at 3:02 PM

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=B6Ymnu54qzSb

+ Code + Text All changes saved

Basic Neura

Using Pytorch

torch

The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serialization of Tensors and arbitrary types, and other useful utilities.

It has a CUDA counterpart, that enables you to run your tensor computations
on an NVIDIA GPU with compute capability >= 3.0.

```
import torch
import numpy as
import torch.nn.

# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43],
                   [91, 88, 64],
                   [87, 134, 58],
                   [102, 43, 37],
                   [69, 96, 70],
                   [85, 100, 60],
                   [95, 80, 55],
                   [105, 120, 75],
                   [78, 90, 50],
                   [82, 70, 45]], dtype='float32')

# Targets (apples, oranges)
targets = np.array([[56, 70],
                    [81, 101],
                    [119, 133],
                    [22, 37],
```


0s completed at 3:02 PM

Basic Neural Network

Using Pytorch

torch

The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serialization of Tensors and arbitrary types, and other useful utilities.

It has a CUDA counterpart, that enables you to run your tensor computations
on an NVIDIA GPU with compute capability >= 3.0.

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

# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43],
                   [91, 88, 64],
                   [87, 134, 58],
                   [102, 43, 37],
                   [69, 96, 70],
                   [85, 100, 60],
                   [95, 80, 55],
                   [105, 120, 75],
                   [78, 90, 50],
                   [82, 70, 45]], dtype='float32')

# Targets (apples, oranges)
targets = np.array([[56, 70],
                    [81, 101],
                    [119, 133],
                    [22, 37],
```

0s completed at 3:02 PM

		(e.g., embeddings).	value prediction.
Focal Loss	Object detection and classification (imbalanced data)	Addresses class imbalance by focusing on hard examples.	Introduces an additional hyperparameter γ\gammaγ, requires tuning.

• Mean Squared Error (MSE): Measures the average squared differences between predicted and actual values. Widely used in regression tasks.

- PyTorch Syntax: `torch.nn.MSELoss()`

• Mean Absolute Error (MAE): Measures the average absolute differences between predicted and actual values. Less sensitive to outliers compared to MSE.

- PyTorch Syntax: `torch.nn.L1Loss()`

• Huber Loss: A combination of MSE and MAE, providing robustness to outliers and smooth gradients.

- PyTorch Syntax: `torch.nn.SmoothL1Loss()`

• Cross-Entropy Loss: Measures the difference between two probability

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text All changes saved

[11]

```
       [ 88.,  95.],
       [115., 140.],
       [ 76.,  85.],
       [ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
mse = F.MSELoss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()
```


0s completed at 3:07 PM

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text All changes saved

T4 RAM Disk

```
         [ 76.,  85.],
         [ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
mse = F.MSELoss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
```


0s completed at 3:09 PM

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text

```
        [ 76.,  85.],
        [ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
mse = torch.nn.MSELoss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
```

✓ 0s completed at 3:09 PM

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text

```
        [ 76.,  85.],
        [ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
mse = torch.nn.MSELoss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
```


0s completed at 3:09 PM

Pytorch.ipynb - Colab | Loss Functions.pdf | cross-entropy loss formula - Go

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text

```
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
loss = mse(preds, targets)
print(loss)
```

```
24[illegible].0968017578125
183.68312072753906
148.4185028076172
127.21119689941406
113.1971664428711
103.01649475097656
95.0095443725586
88.34284973144531
82.5859146118164
77.50598907470703
tensor(77.4581, grad_fn=<MseLossBackward0>)
```

[ ]

✓ 0s completed at 3:10 PM

```
[103., 119.],
[ 98., 110.],
[ 88.,  95.],
[115., 140.],
[ 76.,  85.],
[ 65.,  75.]])
```

```
w = torch.randn(3,2, requires_grad=True)
b = torch.randn(2, requires_grad=True)

# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
mse = torch.nn.MSELoss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
```

0s completed at 3:10 PM

```python
# Defining Model
def model(x):
  # return (x @ w) + b
  return torch.matmul(x,w)+b

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

# Training step
for i in range(1000):
  preds = model(inputs)
  loss = mse(preds, targets)
  loss.backward()
  if i%100 == 99:
    print(loss.item())

  with torch.no_grad():
    w -= w.grad * 0.00001
    b -= b.grad * 0.00001
    w.grad.zero_()
    b.grad.zero_()

preds = model(inputs)
loss = mse(preds, targets)
print(loss)
```

```
247.0968017578125
183.68312072753906
```


0s completed at 3:10 PM