Visual AI

CPSC 532R/533R – 2019/2020 Term 2

Lecture 2. Deep learning basics and best practices

Helge Rhodin

Overview

- Course project introduction
- Compute resources
- Machine learning components in PyTorch
  - Interactive
- Best practices
  - Optimization
  - Loss functions
  - Training and evaluation

Organization

Instructor:

Helge Rhodin

rhodin@cs.ubc.ca

Office hours:

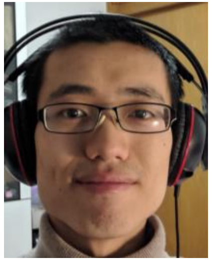
Wednesday 11 am – noon

Room ICCS X653

Teaching assistant:

Yuchi Zhang

yuchi45@cs.ubc.ca



Office hours:

Tuesday 1 pm – 2 pm room

Room ICCS X341

Course Website

Curriculum https://www.cs.ubc.ca/~rhodin/20_CPSC_532R_533R/

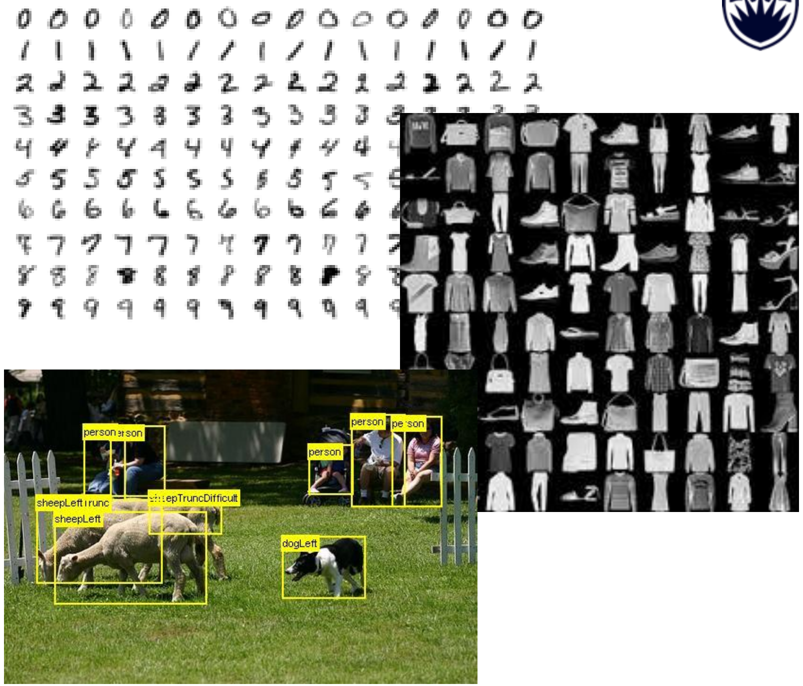
Forum https://piazza.com/ubc.ca/winterterm22019/cs532533

Assignment I

“Playing with PyTorch”

- Network architecture
- Dataloaders
- Evaluation
- Visualization
- Optional add-ons

- Posted on Piazza
- Submit solution on Canvas

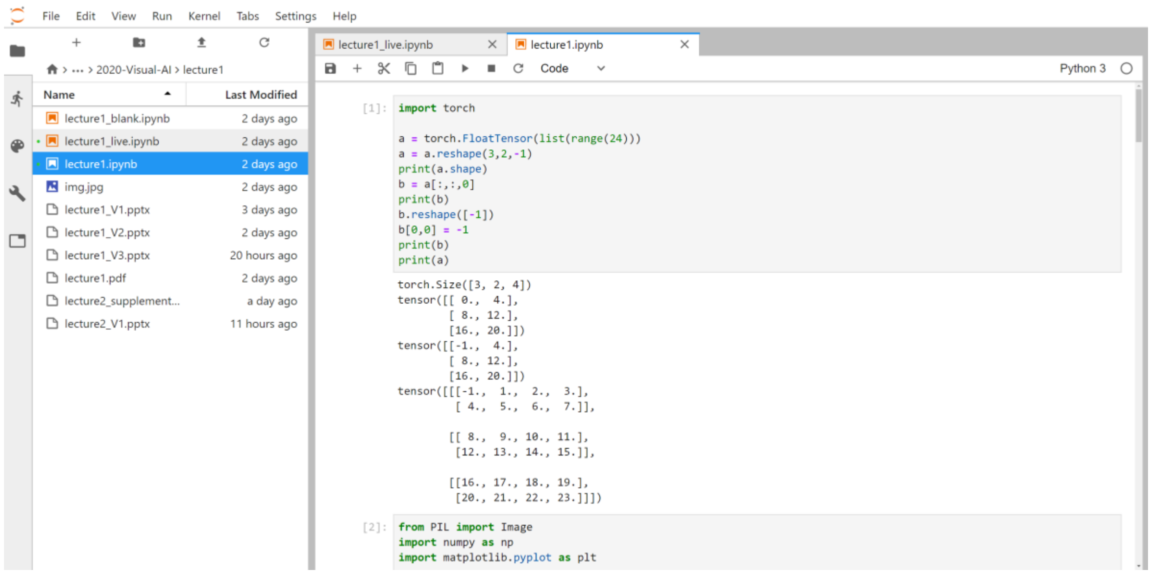


- You can choose your own problem (the task is to implement certain changes)
  - Many good PyTorch tutorials on the web!
  - self-study according to your background knowledge

Jupyter notebooks in Jupyter Lab

Browser-based editor

- easy to use
- cell-based notebooks (.ipynb)
- Good integration of plotting and interactive tools
- remote access possible (start Jupyter Lab on the server, access url on client)



Course projects

Conditions

- groups of two students
- a CV or CG topic of your choice

Project proposal

- 3-minute pitch
- written proposal (one page, 11pt font)
  - research idea
  - possible algorithmic contributions
  - outline of the planned evaluation

Project scope

- Literature review
- Development and coding
- Evaluation

Project report

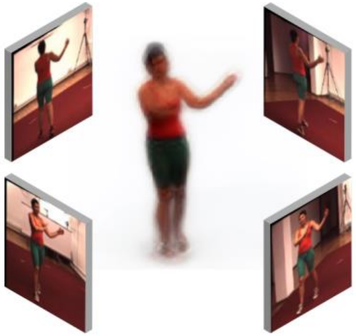
- 8 pages in CVPR double column format
- Sections: motivation, related work, method description, and evaluation

Project presentation

- 10 min talk per group

Possible project directions I

Improve visual quality



e.g., account for perspective effects

Character animation

handle mesh and skeleton sequences

Movie editing

“movie reshaping”

Possible project directions II

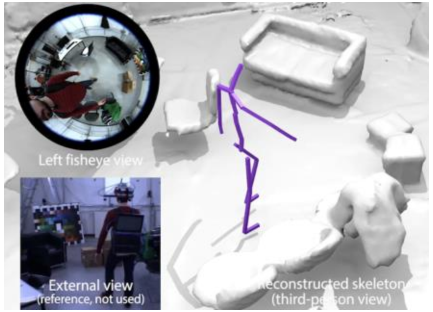
Killer whale identification

Andrew W Trites

Professor and Director

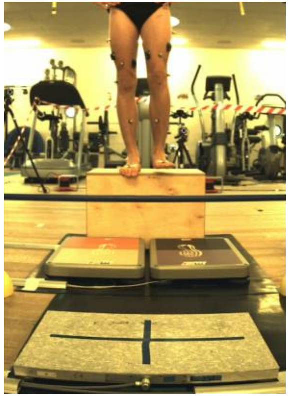
Institute for the Oceans and Fisheries UBC

Prevent foot sliding



IMU-based?

mm-accurate 3D pose estimation



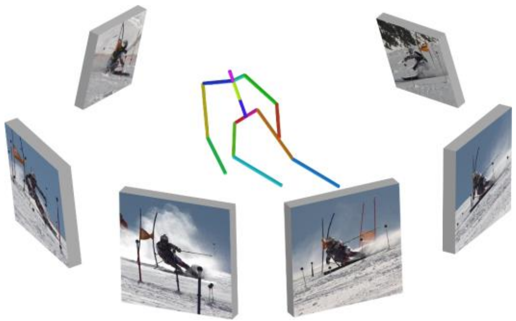
Dr. Jörg Spörri

Sport medicine head

University Hospital Balgrist

Possible project directions III

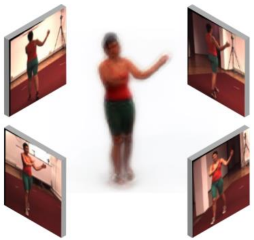
Fast motion capture



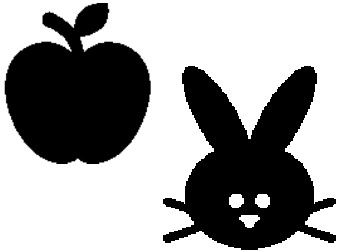
Exploit fast-moving background

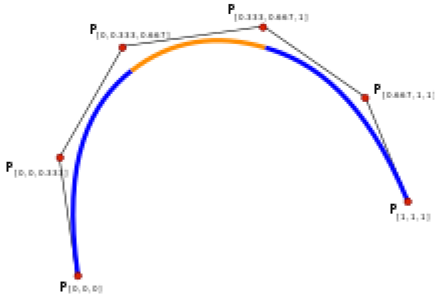
Computer graphics (simulation)

+ Computer vision (real world)



Your own idea!





Compute resources

Personal

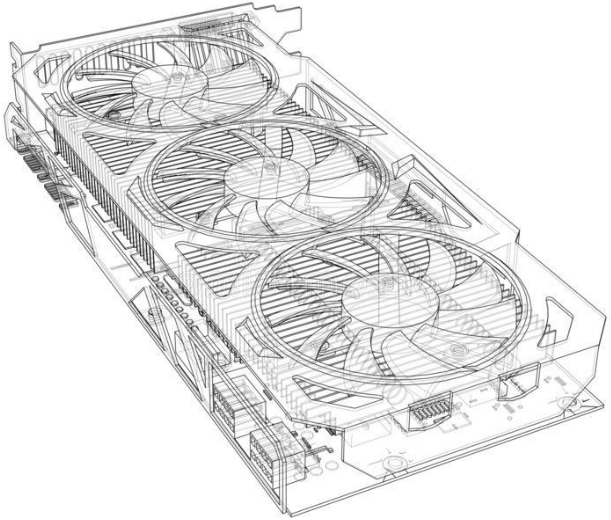
- Your laptop / desktop
  - No GPU?

UBC

- lin01.students.cs.ubc.ca to lin25.students.cs.ubc.ca
  - GTX 1060, 3GB memory
  - Will be setup with pytorch for Assignment 2

Cloud computing

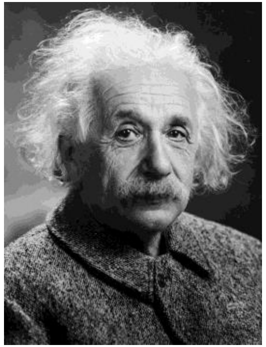
- google colab
  - Tesla K80 GPU
    (free so long you have limited workload)

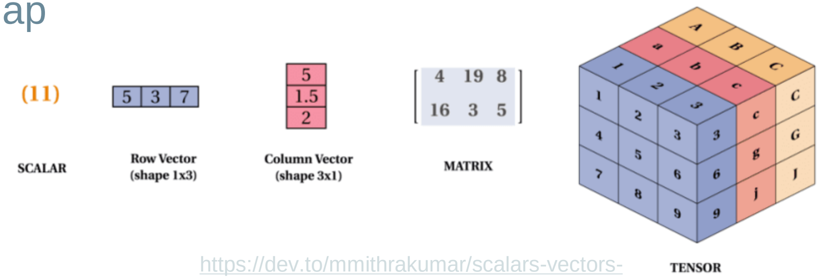


Tensors in pytorch

- Tensor: a multi-dimensional array
  - scaler, vector, matrix, … tensor
- Term hijacked by ML community (in the math/physics community a tensor is a function that can be represented by a multi-dimensional array, but not every array is a math tensor)
- Pytorch uses the NCHW convention:
  dim 0: N, the number of images in a batch
  dim 1: C, the number of channels of an image / feature map
  dim 2: H, the height of the image / feature map
  dim 3: W, the width of the image / feature map
- Different #dimensions possible, dependent on the task

- Order of dimensions matters (cache locality, parallelization)
  - TensorFlow has C in the last dimension, Nervada Neon N





https://dev.to/mmithrakumar/scalars-vectors-matrices-and-tensors-with-tensorflow-2-0-1f66

For the sake of performance…

```
from PIL import Image
pil_image = torch.FloatTensor(np.array(Image.open("img.jpg")))/256

tensor_image = pil_image.permute(2, 0, 1)

pil_image = tensor_image.permute(1, 2, 0)
plt.imshow(tensor_image.permute(1, 2, 0))

batch = torch.stack([tensor_image , tensor_image , tensor_image])
```

Datasets, preprocessing, and efficient loading

- Well-known datasets readily available
  - MNIST, KMNIST, EMNIST, QMNIST, Fashion-MNIST
  - COCO, ImageNet, CIFAR, Cityscapes, Kinetics-400
  - Many more: pytorch.org/docs/stable/torchvision/datasets.html

```
train_set = datasets.FashionMNIST(
    root = './data/FashionMNIST',
    train = True,
    download = True,
    transform = transforms.Compose([
        transforms.ToTensor(),
    ])
)
```

- Loading custom datasets
  - FakeData, ImageFolder, DatasetFolder

- Efficient data loaders
  - parallel threads
  - pinned memory

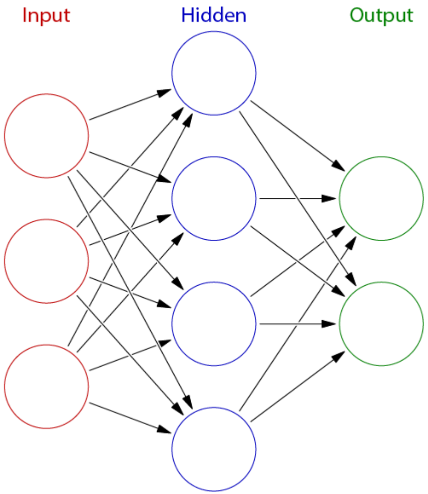
```
loader = torch.utils.data.DataLoader(
            train_set, batch_size = 8)
```

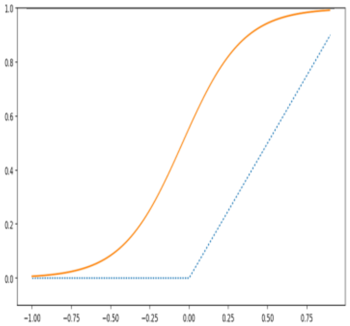
Neural network building blocks (basics)

A summary. More details are provided in supplemental slides.

Fully-connected layer

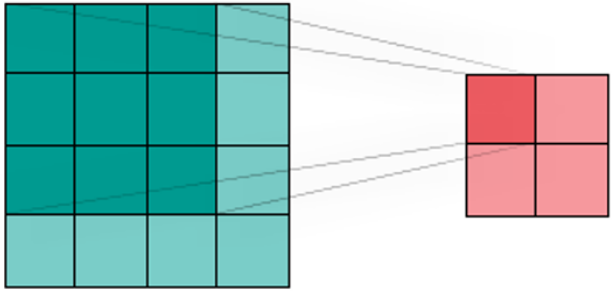
- Linear transformation + activation function (ReLU, sigmoid, tanh, exp)
- Each layer is composed of multiple neurons (same computational rule, different weights)
- Multiple fully-connected layers form a multi layer perceptron (MLP)





Convolution

- **Local** linear transformation + activation function
- Each layer is composed of multiple neurons, some of them sharing weights
- Multiple layers form a convolutional neural network (CNN)



Neural network definition in pytorch

- Standard architectures

```
network = torchvision.models.resnet18(num_classes=10).cuda()
```

- Custom designs

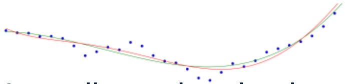
```
class Network(nn.Module):
  def __init__(self):
    super(Network, self).__init__()
    self.conv = nn.Sequential(
        nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.MLP = nn.Linear(in_features=12*4*4, out_features=10)

  def forward(self, batch):
    t = self.conv(batch)
    t = t.reshape(-1, 12*4*4)
    return self.MLP(t)
```

Deep learning – its curve fitting

Parametric curves

- Polynomial

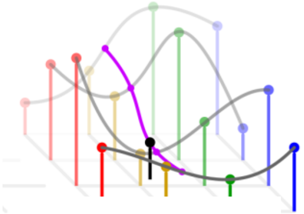


Low-dimensional poly.

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

- Spline

$$f(x) = \begin{cases} f_1(x), & \text{if } x_1 < x \leq x_2 \\ f_2(x), & \text{if } x_2 < x \leq x_3 \\ \vdots & \vdots \\ f_n(x), & \text{if } x_n < x \leq x_{n+1} \end{cases}$$



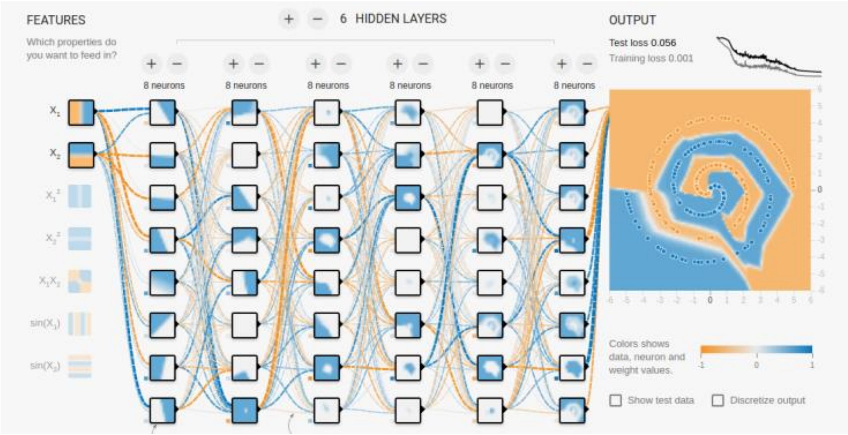
Bicubic spline

- Neural network

$$f(x) = h(\text{linear}(h(\text{linear}(x, W^{(1)})), W^{(2)}))$$

Goal: Find θ that minimizes the objective function on the dataset D

$$\arg\min_{\theta} E(D, \theta)$$



Multilayer perceptron (fully connected network)
live at playground.tensorflow.org

Objective function

General form

$$\arg\min_{\theta} E(D, \theta)$$

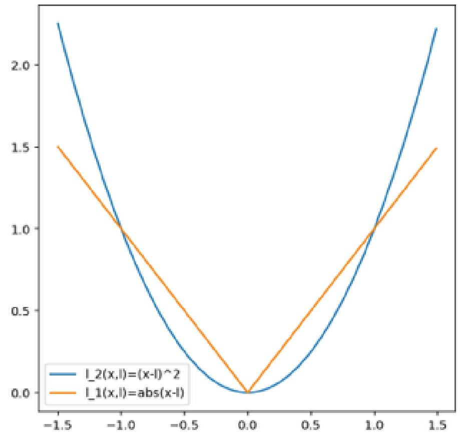
Separable form

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} l(\ f(\mathbf{x}^{(i)}, \theta), y^{(i)})$$

MNIST example

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\ f(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2$$

$=(\ f($$, \theta) - 7)^2 + (\ f($$, \theta) - 8)^2 \ldots$



Quadratic loss

$$l_2(y, l) = (y - l)^2$$

Absolute loss

$$l_1(y, l) = |y - l|$$

Note, in PyTorch, a loss is also called a criterion

Objective function in pytorch

Regression: squared loss, l1 loss, huber loss…

- nn.functional. MSELoss(pred, gt)

Classification: cross-entropy loss, hinge loss, …

- nn.functional.cross_entropy(pred_probabilities, gt_probabilities)

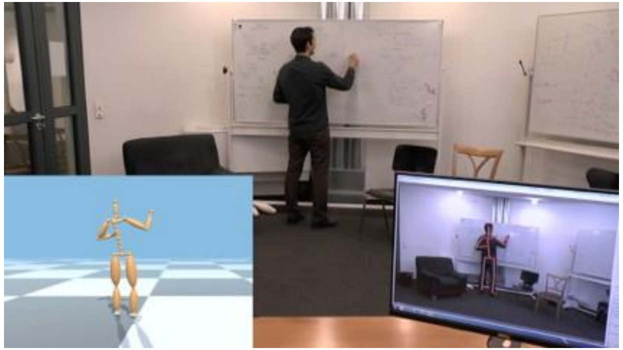
```python
class MyHingeLoss(torch.nn.Module):
  def __init__(self):
      super(MyHingeLoss, self).__init__()
  def forward(self, output, target):
      hinge_loss = 1 - torch.mul(output, target)
      hinge_loss[hinge_loss < 0] = 0
      return hinge_loss
```

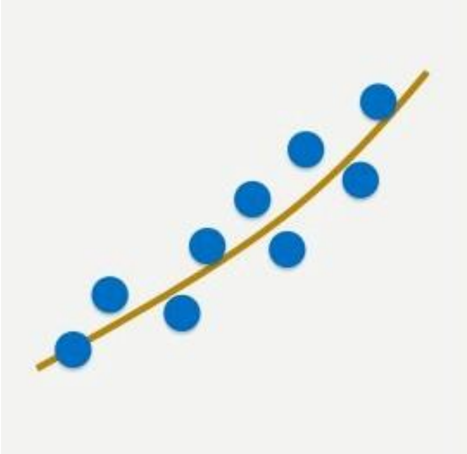

https://medium.com/bbm406f16/week-4-classification-or-regression-f2bf5072cc43

Classification vs. regression

Classification

Regression

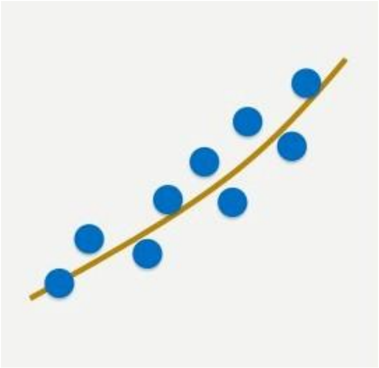




Classification and regression

- Regression
  (for continuous values)

$$\mathrm{nn}(\mathbf{x}) \rightarrow y \in \mathbb{R}$$
$$l_2(y, l) = (y - l)^2$$

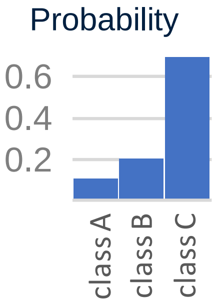


- Classification
  - discrete classes

$$\mathrm{nn}(\mathbf{x}) \rightarrow \mathbf{y} \in [0, 1]$$

  - probabilistic interpretation (probability of class)

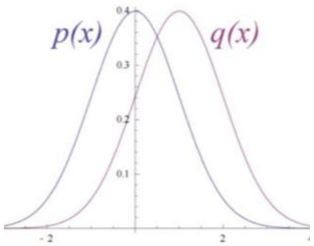
$$l_2(\mathbf{y}, \mathbf{l}) = \|\mathbf{y} - \mathbf{l}\|^2$$



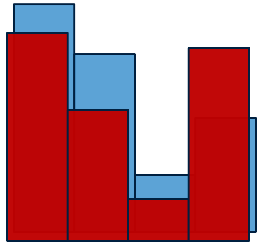
Cross-entropy loss / Cross-entropy criterion

Cross entropy definition:

$$H(p,q) \;=\; -\mathrm{E}_p[\log q] \;=\; -\sum_{c=1}^{K} p(c)\log q(c)$$



continuous case


discrete case

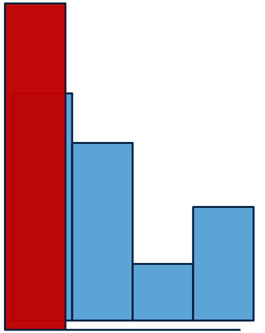
Cross-entropy loss for label y

$$l_{\text{cross entropy}}(x,y) = -\sum_{j=1}^{K} y_{[j]}\log(f_{[j]}(x))$$

j’th value of vector, j’th class

Negative Log Likelihood (NLL) formulation for a *one-hot vector,* target class c

$$l_{\text{NLL}}(x,c) = -\log(f_{[c]}(x))$$


one-hot vector case

There is a trivial solution, simply let f go to infinity for all classes!

Cross correlation with a soft-max layer

Negative log likelihood with preceding soft-max

$$l_{\text{log-likelihood}}(x, y) = -\log(\text{soft-max}(f(x), y))$$

$$= -f_{[y]}(x) + \log\left(\sum_{j=1}^{K} e^{f_{[j]}(x)}\right)$$

Soft-max

$$\text{soft-max}(z, i) = \frac{e^{z[i]}}{\sum_{j=1}^{K} e^{z[j]}}$$

log-sum-exp trick

$$\text{log-sum-exp}(z) = \log\left(\sum_{j=1}^{K} e^{z}\right)$$

$$= \bar{z} + \log\left(\sum_{j=1}^{K} e^{z-\bar{z}}\right)$$

$$\text{with } \bar{z} = \max(z)$$

Exp normalize trick

$$\text{soft-max}(z, i) = \frac{e^{z[i]-\bar{z}} \quad e^{\bar{z}}}{\sum_{j=1}^{K} e^{z[j]-\bar{z}} \quad e^{\bar{z}}}$$

$$= \frac{e^{z[i]-\bar{z}}}{\sum_{j=1}^{K} e^{z[j]-\bar{z}}}$$

shift invariance to increase numerical stability!

Cross-entropy loss in PyTorch

def def cross_entropy(input, …

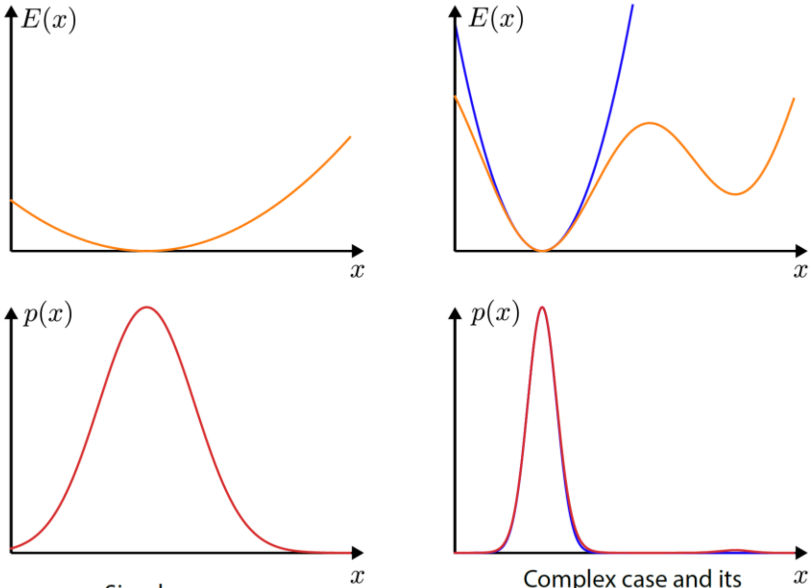
    return nll_loss(log_softmax(input …

- Includes the normalization by log-soft-max
  - numerically stable
  - fast
  - don’t normalize twice with your own soft-max layer!

Regression revisited

Error functions
$E(x)$

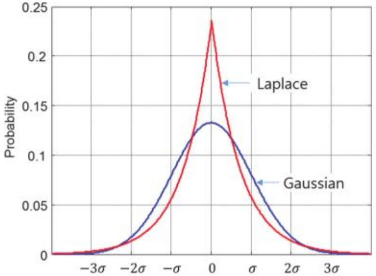
Distributions
$p(x) = \exp(-E(x))$



Simple case

Complex case and its quadratic/Gaussian approximation

x^2	$\lvert x \rvert$
Mean squared error (MSE)	Mean absolute error (MAE)
$\exp(-x^2)$	$\exp(-\lvert x \rvert)$
Gaussian distribution	Laplace distribution



3 min Break

Register on Piazza or play playground.tensorflow.org

Optimization loop in Pytorch

- Iterative local optimization (`opt`) over minibatches (`x, y`) returned by the dataloader (`loader`) using automatic differentiation of the objective and neural network (`loss.backward()`)

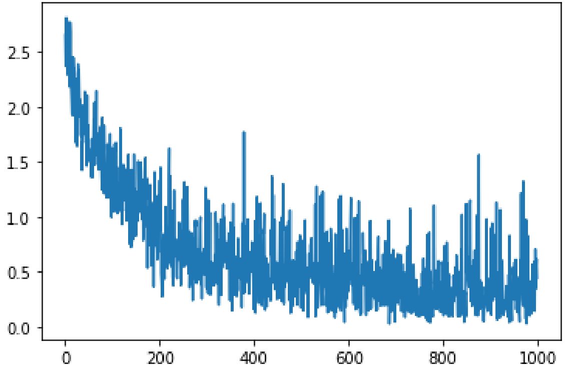
```
iterator = iter(loader)
device = "cuda"
for i in range(len(loader)):
    x, y = next(iterator)
    preds = net(x)
    loss = nn.functional.cross_entropy(preds, y)
    opt = optim.SGD(net.parameters(), lr=0.001)

    optimizer.zero_grad()
    loss.backward()
    opt.step()
```

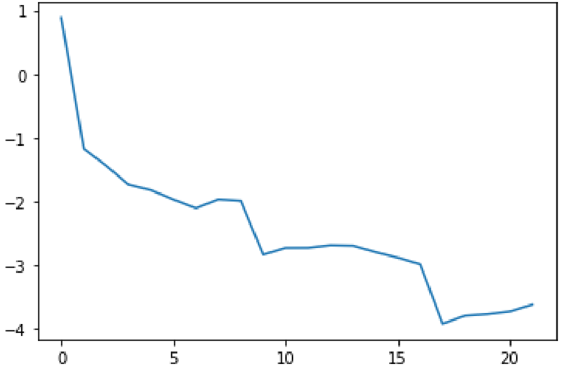
Don’t forget zero gradients, pytorch accumulates gradients by default

4. Evaluation

Training loss (per iteration)



Validation loss (per epoch)



Test accuracy (once and for all)

0.94

Golden rule of machine learning: Don’t touch the test set when building your model (including high-level design choices)!

My silver rule: Don’t use only the training set. Separate out a validation set to systematically determine hyper parameters (stopping time, network architecture, …). How else? Human intelligence?

Hidden questions

Open MSc / PhD position

Interactive FishTank VR experience

- STAIR project with Sidney Fels (ECE)

- Interested? Send me a note, rhodin@cs.ubc.ca