Multiple Things

a bit of everything

Thorsten Beier

February 14, 2019

Outline

Outline

- Deep Learning / Neural Network Frameworks

Outline

- Deep Learning / Neural Network Frameworks
    - Autoencoders

Outline

- Deep Learning / Neural Network Frameworks
    - Autoencoders
    - Variational Autoencoders

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno

Outline

- Deep Learning / Neural Network Frameworks
    - Autoencoders
    - Variational Autoencoders
    - Convolutional Autoencoders
    - PyTorch
    - Inferno
- Modern C++

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why
  - Improve {Python, R, Julia} with C++

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why
  - Improve {Python, R, Julia} with C++
  - Rapid Development with C++

Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why
  - Improve {Python, R, Julia} with C++
  - Rapid Development with C++
- Discrete Optimization

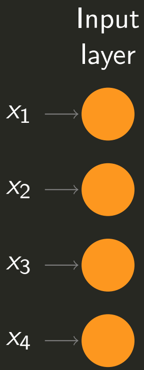
Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why
  - Improve {Python, R, Julia} with C++
  - Rapid Development with C++
- Discrete Optimization
  - Energy Minimization

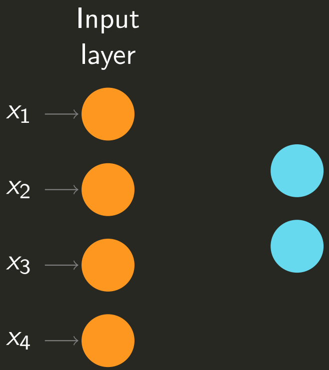
Outline

- Deep Learning / Neural Network Frameworks
  - Autoencoders
  - Variational Autoencoders
  - Convolutional Autoencoders
  - PyTorch
  - Inferno
- Modern C++
  - Why
  - Improve {Python, R, Julia} with C++
  - Rapid Development with C++
- Discrete Optimization
  - Energy Minimization
  - Graphical Models

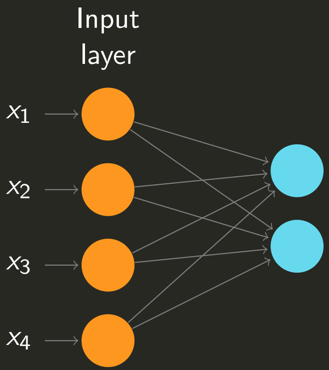
Autoencoders



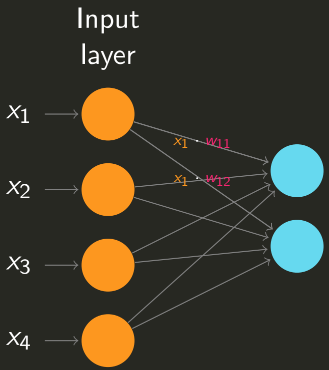
Autoencoders



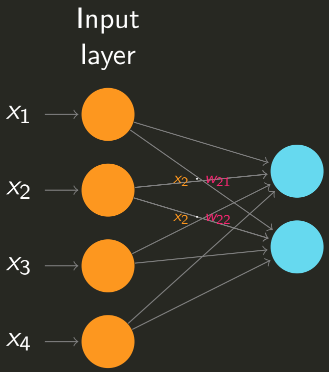
Autoencoders



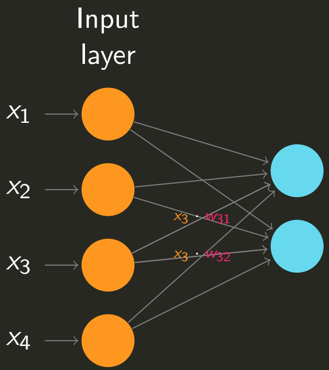
Autoencoders



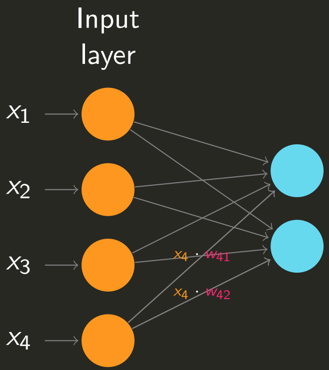
Autoencoders



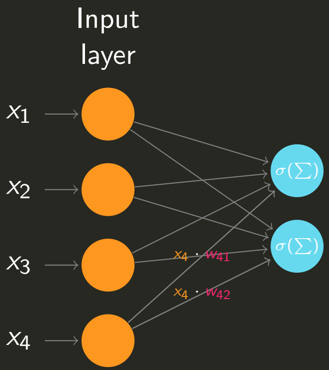
Autoencoders



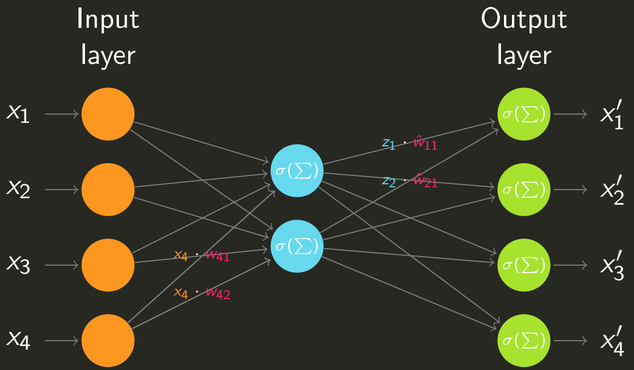
Autoencoders



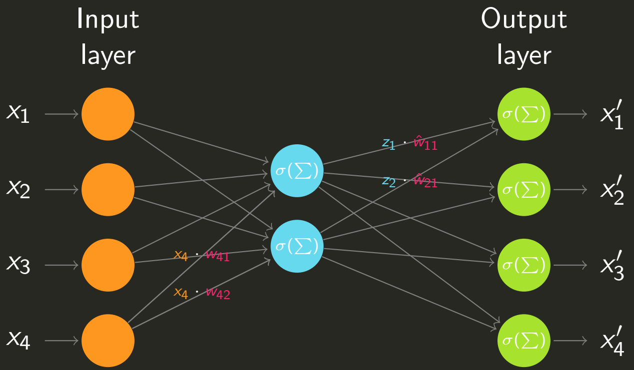
Autoencoders



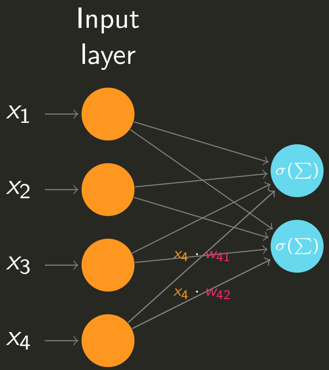
Autoencoders



Autoencoders

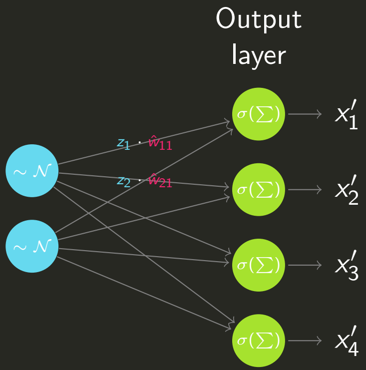


Training: Minimize $||x - x'||^2$



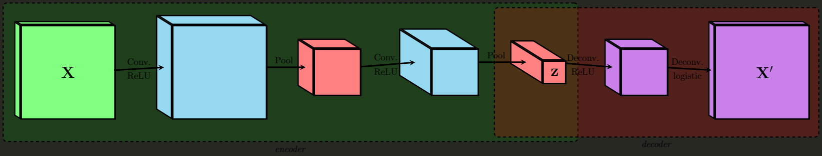
Test Time: get embeddings for unknown x

Autoencoders



Decode from random embeddings

Convolutional Autoencoder



```
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()


















#
```

Deep Learning with PyTorch

```
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(True),
            nn.Linear(256, 32),
            nn.ReLU(True))











#
```

Deep Learning with PyTorch

```
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(True),
            nn.Linear(256, 32),
            nn.ReLU(True))

        self.decoder = nn.Sequential(
            nn.Linear(32, 256),
            nn.ReLU(True),
            nn.Linear(256, 1024),
            nn.Sigmoid())





#
```

Deep Learning with PyTorch

```python
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(True),
            nn.Linear(256, 32),
            nn.ReLU(True))

        self.decoder = nn.Sequential(
            nn.Linear(32, 256),
            nn.ReLU(True),
            nn.Linear(256, 1024),
            nn.Sigmoid())

    def forward(self, x):
        x = self.encoder(x)



#
```

Deep Learning with PyTorch

```python
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(True),
            nn.Linear(256, 32),
            nn.ReLU(True))

        self.decoder = nn.Sequential(
            nn.Linear(32, 256),
            nn.ReLU(True),
            nn.Linear(256, 1024),
            nn.Sigmoid())

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)

#
```

Deep Learning with PyTorch

```python
from pytorch import nn
class Autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(True),
            nn.Linear(256, 32),
            nn.ReLU(True))

        self.decoder = nn.Sequential(
            nn.Linear(32, 256),
            nn.ReLU(True),
            nn.Linear(256, 1024),
            nn.Sigmoid())

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
#
```

```python
class VariationalAutoencoder(nn.Module):
    def __init__(self):
        super(VariationalAutoencoder, self).__init__()

        self.fc1  = nn.Linear(784, 400)
        self.fc21 = nn.Linear(400, 20)
        self.fc22 = nn.Linear(400, 20)
        self.fc3  = nn.Linear(20, 400)
        self.fc4  = nn.Linear(400, 784)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return eps.mul(std).add_(mu)

    def decode(self, z):
        h3 = F.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

```
class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=3, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(16, 8, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=1)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(8, 16, 3, stride=2),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 8, 5, stride=3, padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(8, 1, 2, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

```
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)





















#
```

```
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(500):

  # Forward pass:
  y_pred = model(x)
















#
```

```
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(500):

  # Forward pass:
  y_pred = model(x)

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())











#
```

```
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(500):

  # Forward pass:
  y_pred = model(x)

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())

  # zero all gradients
  optimizer.zero_grad()

  # Backward pass: compute gradient of
  # the loss wrt model parameters
  loss.backward()




#
```

```python
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(500):

  # Forward pass:
  y_pred = model(x)

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())

  # zero all gradients
  optimizer.zero_grad()

  # Backward pass: compute gradient of
  # the loss wrt model parameters
  loss.backward()

  # Calling the step function on an
  # Optimizer makes an update to its parameters
  optimizer.step()
#
```

```python
learning_rate = 1e-4         # <- fixed learning rate
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(500):         # <- to much / to little

  # Forward pass:
  y_pred = model(x)          # <- no prober data sampling

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())      # <- bad practise for logging

  # zero all gradients
  optimizer.zero_grad()      # <- boilerplate

  # Backward pass: compute gradient of
  # the loss wrt model parameters
  loss.backward()            # <- boilerplate

  # Calling the step function on an
  # Optimizer makes an update to its parameters
  optimizer.step()           # <- boilerplate
#
```

Inferno

convenience functions/classes around PyTorch

INFERNO

GitHub

https://github.com/inferno-pytorch/inferno

Inferno
convenience functions/classes around PyTorch

```python
from inferno.trainers.basic import Trainer
from inferno.trainers.callbacks.logging.tensorboard import TensorboardLogger
from inferno.trainers.callbacks.scheduling import AutoLR
trainer = Trainer(model)
trainer.build_criterion('NLLLoss')
trainer.build_metric('CategoricalError')
trainer.build_optimizer('Adam')
trainer.validate_every((2, 'epochs'))
trainer.save_every((5, 'epochs'))
trainer.save_to_directory('some/dir')
trainer.set_max_num_epochs(10)
trainer.register_callback(AutoLR(factor=0.9, patience=(5, 'epochs')))
trainer.build_logger(TensorboardLogger(log_scalars_every=(1, 'iteration'),
                log_directory='some/dir'))

# bind loader
trainer.bind_loader('train', train_loader)
trainer.bind_loader('validate', validate_loader)

if USE_CUDA:
  trainer.cuda()

trainer.fit()
```

Tensorboard

Visualize the training procedure *while training*

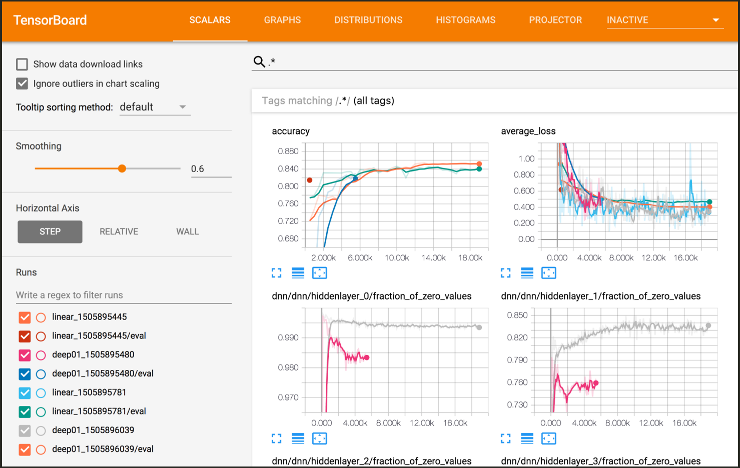


Image Source: `https://towardsdatascience.com/visualizing-your-model-using-tensorboard-796ebb73e98d`

Tensorboard

Look at embeddings *while training*

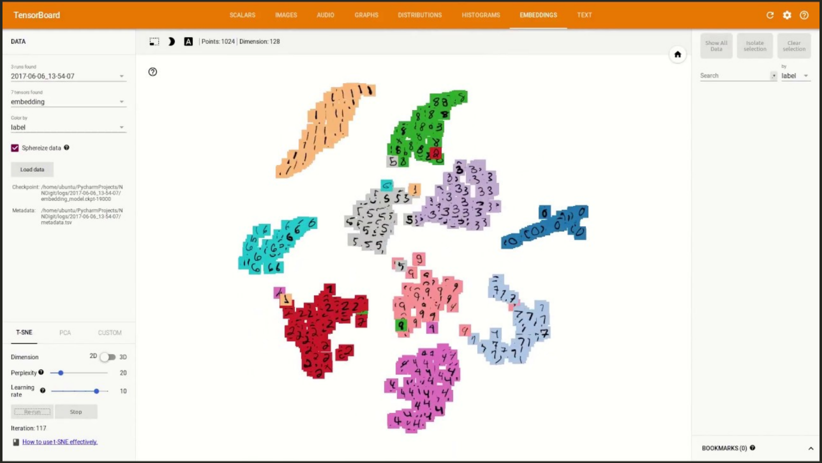


Image Source: https://www.youtube.com/watch?v=EL0wVFz0gsc

Why C++

Interpreted Languages are often slow

Interpreted Languages are often slow

```python
def strange_function(n):
    s = 0
    for x in range(n):
        for y in range(n):
            for z in range(n):
                s += x + 2 * y + z
    return s
```

Interpreted Languages are often slow

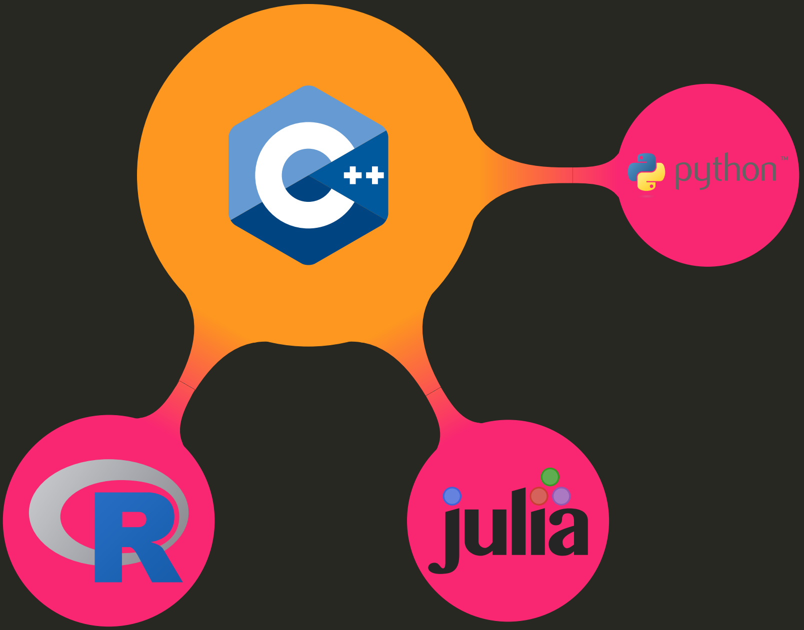
```python
def strange_function(n):
    s = 0
    for x in range(n):
        for y in range(n):
            for z in range(n):
                s += x + 2 * y + z
    return s
```

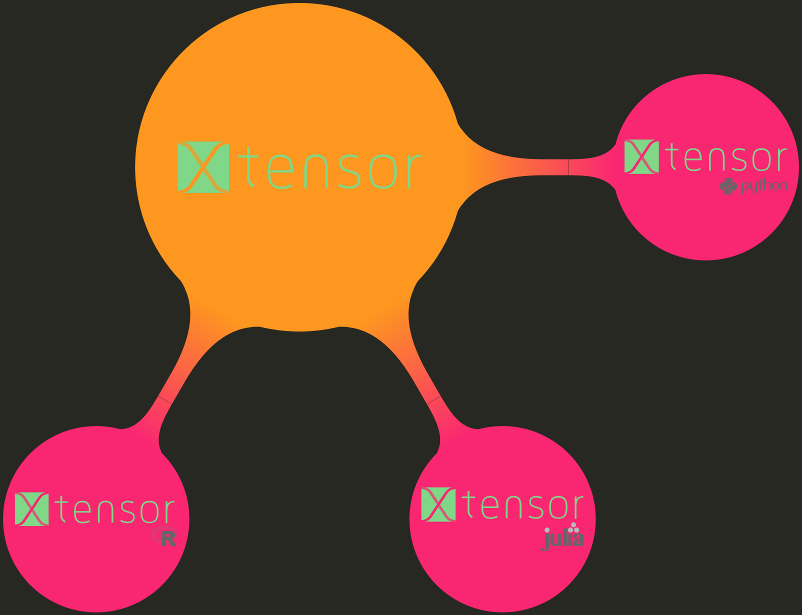
```python
strange_function(100)   #     0.1 sec
strange_function(1000)  #     100 sec
strange_function(2000)  # >>1000 sec
```

C++ is fast

```
auto strange_function(int n)
{
    auto s = 0;
    for(auto x = 0; x<n; ++x)
    for(auto y = 0; y<n; ++y)
    for(auto z = 0; z<n; ++z)
    {
        s += x + 2 * y + z;
    }
    return s;
}
```

```
strange_function(100)   # ~0.0 sec
strange_function(1000)  #  2.0 sec
strange_function(2000)  # 20.0 sec
```


C++
python
R
julia


xtensor
xtensor
python
xtensor
R
xtensor
julia

xtensor nd-arrays for C++

Python:

```python
np.array([[3, 4], [5, 6]])

arr.reshape([3, 4])
arr.astype(np.float64)

np.stack([a, b, c], axis=1)
np.concatenate([a, b, c], axis=1)
np.squeeze(a)
np.expand_dims(a, 1)
np.atleast_3d(a)
np.split(a, 4, axis=0)

a[:, np.newaxis]
a[:5, 1:]
a[5:1:-1, :]
a[..., 3]

np.broadcast(a, [4, 5, 7])
np.vectorize(f)
a[a > 5]
a[[0, 1], [0, 0]]
```

C++:

```cpp
xt::xarray<double>({{3, 4}, {5, 6}})
xt::xtensor<double, 2>({{3, 4}, {5, 6}})
arr.reshape({3, 4})
xt::cast<double>(arr)

xt::stack(xt::xtuple(a, b, c), 1)
xt::concatenate(xt::xtuple(a, b, c), 1)
xt::squeeze(a)
xt::expand_dims(a ,1)
xt::atleast_3d(a)
xt::split(a, 4, 0)

xt::view(a, xt::all(), xt::newaxis())
xt::view(a, xt::range(_, 5), xt::range(1, _))
xt::view(a, xt::range(5, 1, -1), xt::all())
xt::strided_view(a, {xt::ellipsis, 3})

xt::broadcast(a, {4, 5, 7})
xt::vectorize(f)
xt::filter(a, a > 5)
xt::index_view(a, {{0, 0}, {1, 0}})
```

C++ Code:

```cpp
double sum_of_sines(xt::pyarray<double>& m)
{
    auto sines = xt::sin(m);
    return std::accumulate(sines.begin(), sines.end(), 0.0);
}

PYBIND11_MODULE(xtensor_python_test, m)
{
    xt::import_numpy();
    m.def("sum_of_sines", sum_of_sines, "Sum the sines of the input values");
}
```

Python Code:

```python
import numpy as np
import xtensor_python_test as xt

v = np.arange(15).reshape(3, 5)
s = xt.sum_of_sines(v)
```

C++ Code:

```cpp
using namespace Rcpp;

// [[Rcpp::plugins(cpp14)]]

// [[Rcpp::export]]
double sum_of_sines(xt::rarray<double>& m)
{
    auto sines = xt::sin(m);
    return std::accumulate(sines.cbegin(), sines.cend(), 0.0);
}
```

R Code:

```r
v <- matrix(0:14, nrow=3, ncol=5)
s <- sum_of_sines(v)
```

C++ Code:

```
double sum_of_sines(xt::jlarray<double> m)
{
    auto sines = xt::sin(m);  // sines does not actually hold values.
    return std::accumulate(sines.cbegin(), sines.cend(), 0.0);
}

JLCXX_MODULE define_julia_module(jlcxx::Module& mod)
{
    mod.method("sum_of_sines", sum_of_sines);
}
```

Julia Code:

```
using xtensor_julia_test
arr = [[1.0 2.0]
       [3.0 4.0]]
s = sum_of_sines(arr)
```

C++ Notebooks

jupyter nbviewer   JUPYTER   FAQ

Initialize a 2-D array and compute the sum of one of its rows and a 1-D array

In [ ]:
```
#include <iostream>

#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"
#include "xtensor/xview.hpp"
```

In [ ]:
```
xt::xarray<double> arr1
  {{1.0, 2.0, 3.0},
   {2.0, 5.0, 7.0},
   {2.0, 5.0, 7.0}};

xt::xarray<double> arr2
  {5.0, 6.0, 7.0};

xt::view(arr1, 1) + arr2
```

In [ ]:
```
arr1
```

Initialize a 1-D array and reshape it inplace

In [ ]:
```
#include <iostream>
#include "xtensor/xarray.hpp"
#include "xtensor/xio.hpp"
```

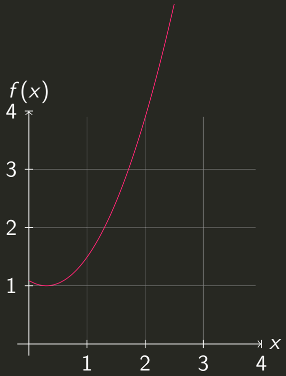
In [ ]:
```
xt::xarray<int> arr
  {1, 2, 3, 4, 5, 6, 7, 8, 9};

arr.reshape({3, 3});
```

Try it out: https://github.com/QuantStack/xtensor
https://mybinder.org/v2/gh/QuantStack/xtensor/stable?filepath=notebooks/xtensor.ipynb

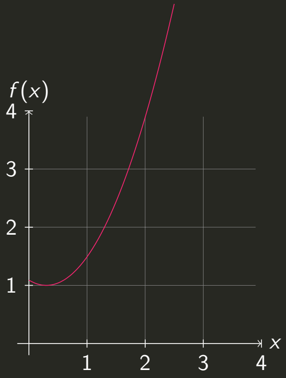
$$f(x) = (x - 0.3)^2 + 1$$

$$f(x) = (x - 0.3)^2 + 1$$



Discrete Optimization

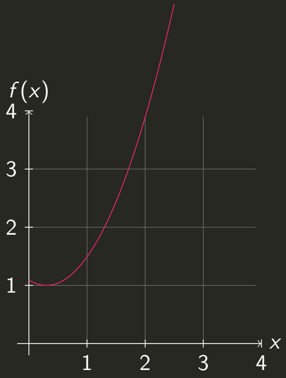
$f(x) = (x - 0.3)^2 + 1$



Find $x \in \mathbb{R}$ which minimizes $f(x)$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$

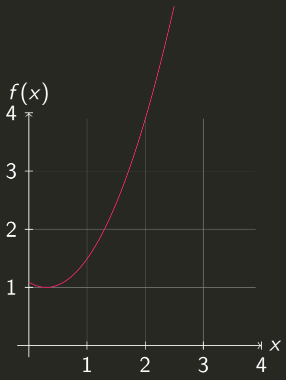


Find $x \in \mathbb{R}$ which minimizes $f(x)$

$$\hat{x} = \operatorname*{arg\,min}_{x \in \mathbb{R}} f(x)$$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$



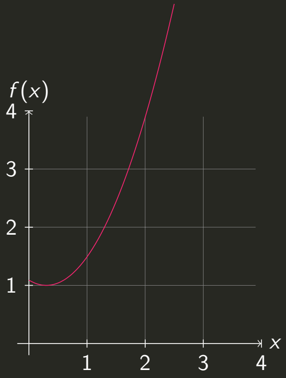
Find $x \in \mathbb{R}$ which minimizes $f(x)$

$$\hat{x} = \arg\min_{x \in \mathbb{R}} f(x)$$

$$\hat{x} = 0.3$$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$



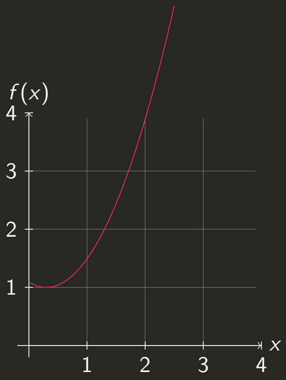
Find $x \in \mathbb{R}$ which minimizes $f(x)$

$$\begin{aligned} \hat{x} &= \arg\min_{x \in \mathbb{R}} \quad f(x) \\ \hat{x} &= 0.3 \end{aligned}$$

Find integral $x \in \mathbb{N}$ which minimizes $f(x)$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$



Find $x \in \mathbb{R}$ which minimizes $f(x)$

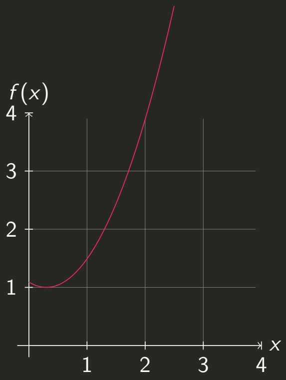
$$\hat{x} = \arg\min_{x \in \mathbb{R}} f(x)$$
$$\hat{x} = 0.3$$

Find integral $x \in \mathbb{N}$ which minimizes $f(x)$

$$\tilde{x} = \arg\min_{x \in \mathbb{N}} f(x)$$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$



Find $x \in \mathbb{R}$ which minimizes $f(x)$

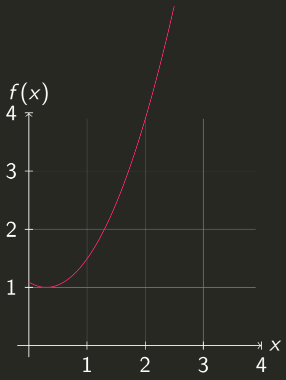
$$\hat{x} = \arg\min_{x \in \mathbb{R}} f(x)$$
$$\hat{x} = 0.3$$

Find integral $x \in \mathbb{N}$ which minimizes $f(x)$

$$\tilde{x} = \arg\min_{x \in \mathbb{N}} f(x)$$
$$\tilde{x} = 0$$

Discrete Optimization

$$f(x) = (x - 0.3)^2 + 1$$



Find $x \in \mathbb{R}$ which minimizes $f(x)$

$$\hat{x} = \arg\min_{x \in \mathbb{R}} f(x)$$
$$\hat{x} = 0.3$$

Find integral $x \in \{0, 1, 2, 3, 4\}$ which minimizes $f(x)$

$$\tilde{x} = \arg\min_{x \in \mathbb{N}} f(x)$$
$$\tilde{x} = 0$$

Discrete Optimization

$x_i \in \{0, 1\}$

$$E(x_1, x_2, x_3, \ldots, x_{N-1}, x_N)$$

Discrete Optimization

$x_i \in \{0, 1\}$

$$E(\vec{x})$$

Discrete Optimization

$x_i \in \{0, 1\}$

$$E(\vec{x})$$

$$\tilde{\vec{x}} \quad = \quad \arg\min_{\vec{x} \in \{0,1\}^N} \quad E(\vec{x})$$

Discrete Optimization

$x_i \in \{0, 1\}$

$$\tilde{\vec{x}} = \underset{\vec{x} \in \{0,1\}^N}{\arg\min} \; E(\vec{x})$$

Enumerating 2^N combinations is impractical

$x_i \in \{0, 1\}$

$$E(\vec{x}) = \phi_0(x_0) + \phi_1(x_2) + \phi_1(x_0, x_1) + \phi_2(x_1, x_2) + \phi_3(x_2, x_3)$$

$$\tilde{\vec{x}} = \operatorname*{arg\,min}_{\vec{x} \in \{0,1\}^N} E(\vec{x})$$

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*

Discrete Optimization
Graphical Models

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*

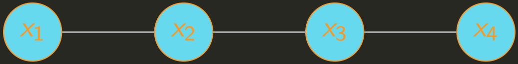
$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*

Discrete Optimization
Graphical Models

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

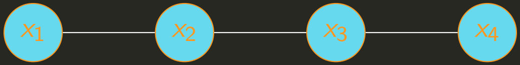
Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*



- Powerful tool orthogonal to neural networks
- Combinatorical Problems

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*



- Powerful tool orthogonal to neural networks
- Combinatorical Problems

Examples:

- Find Optimal seating arrangements for a Table
- Image Segmentation
- Protenin Folding

Discrete Optimization

Graphical Models

$$E(\vec{x}) = \phi_1(x_1) + \phi_2(x_2) + \phi_3(x_1, x_2) + \phi_4(x_2, x_3) + \phi_5(x_3, x_4)$$

Functions as $E(\vec{x})$ are also called *Discrete Graphical Model*

- Powerful tool orthogonal to neural networks
- Combinatorical Problems

Examples:

- Find Optimal seating arrangements for a Table
- Image Segmentation
- Protenin Folding

GitHub `https://github.com/opengm/opengm`