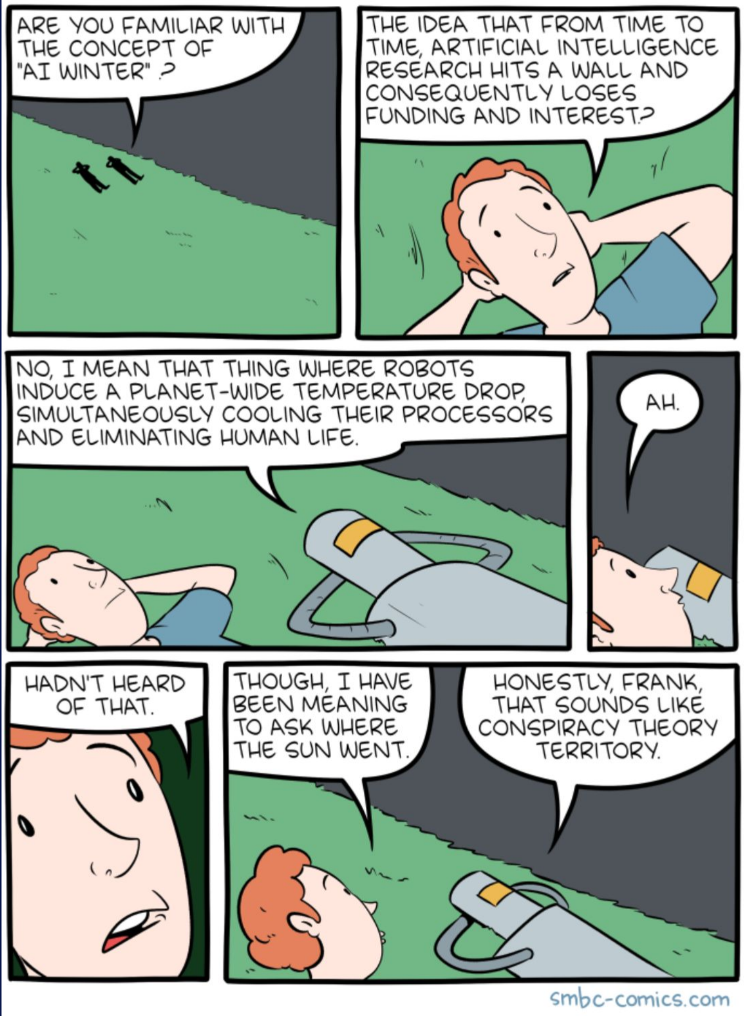

ARE YOU FAMILIAR WITH THE CONCEPT OF "AI WINTER"?
THE IDEA THAT FROM TIME TO TIME, ARTIFICIAL INTELLIGENCE RESEARCH HITS A WALL AND CONSEQUENTLY LOSES FUNDING AND INTEREST?
NO, I MEAN THAT THING WHERE ROBOTS INDUCE A PLANET-WIDE TEMPERATURE DROP, SIMULTANEOUSLY COOLING THEIR PROCESSORS AND ELIMINATING HUMAN LIFE.
AH.
HADN'T HEARD OF THAT.
THOUGH, I HAVE BEEN MEANING TO ASK WHERE THE SUN WENT.
HONESTLY, FRANK, THAT SOUNDS LIKE CONSPIRACY THEORY TERRITORY.
smbc-comics.com

# CIS 522: Lecture 2T

Introduction to the process of building networks and PyTorch

1/21/20

# Course Announcements

- HW 0 + HW 1 have both been released
- Please direct any questions about the course to Piazza.
- Office hours are on the course website.

# Today

First, review the workflow of a DL data scientist

Second, review PyTorch

# The workflow of DL designer

- Get a dataset
  - Exploratory data analysis, normalization strategies, Class imbalance/Bayes
- Write a data loader
  - And do data augmentation
- Define a neural network
  - This is where Pytorch forward/backwards is awesome
- Define a loss/ objective function
  - Consider regularization
- Optimize the neural network
  - Babysit optimization
- Test performance, Save model
- Deploy model

# Have a real question

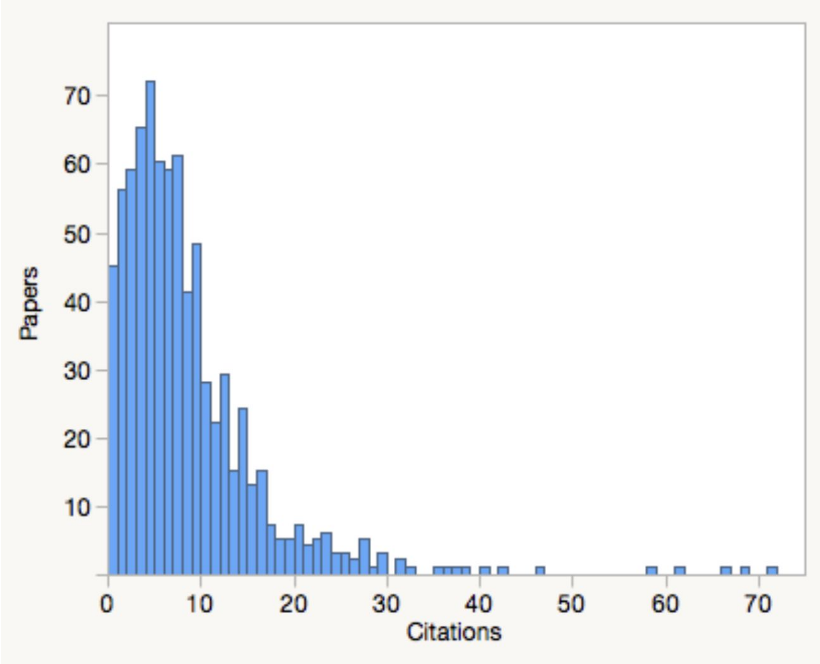
Most papers are boring

Most questions are irrelevant

In academia as in industry

Start abstract first

Review the abstracts




eLife paper cite distribution

From: scholarly kitchen

# Define the data science project you would most love to do

# Get data to ask your questions

Private dataset

Kaggle datasets

Scientific datasets (NSF, NIH etc)

# Explore and Process the Data

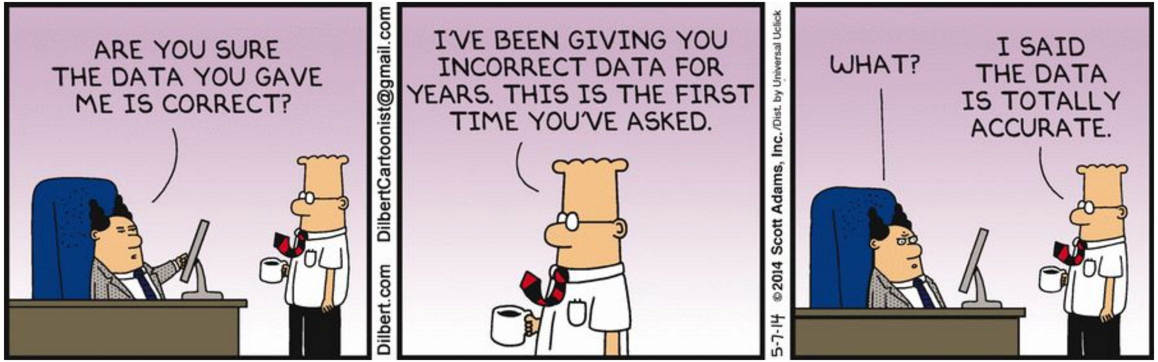
- Explore available data points and features in the data
- Create a dataset class
- Preprocessing
  - Feature Scaling
  - Normalization
- Split into train-validation-test sets
- Create dataloaders - return shuffled data batches

# When you don't have a validation set

- You will always overfit
- You can not have hyperparameters
- Your fits will be too bad
- None of the above

# Why data exploration is important?

- Data is biased
- Data is imbalanced
  - Medical datasets - Very few examples with diseases vis-a-vis without.
  - Class imbalance in multiclass datasets (Example: Cats/Dogs more common than Tigers/Leopards)
- Data is incorrect

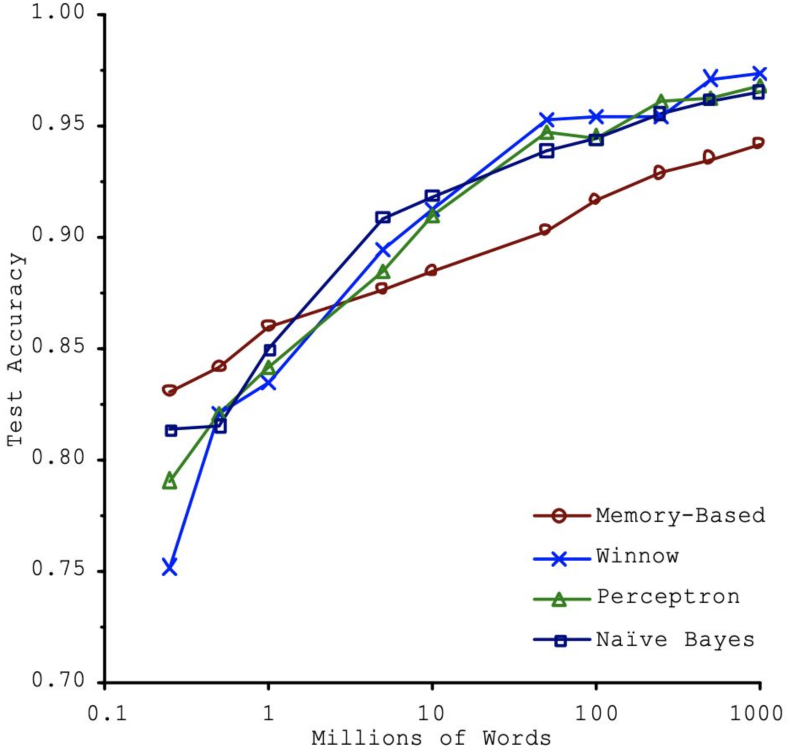


# The effect of class imbalance

Bayes rule on whiteboard

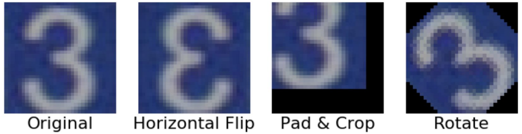
Which question do we want to ask?

# More data = key to performance

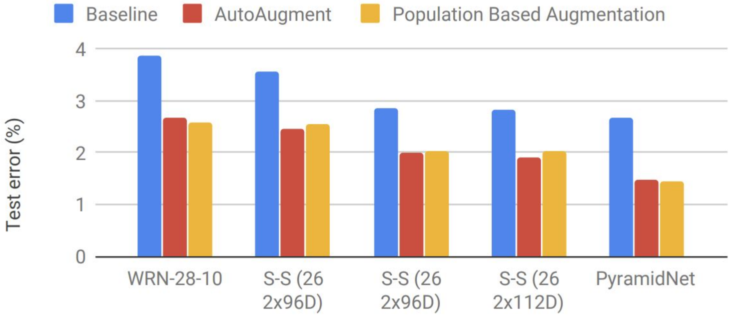



Banko and Brill 2001

# Data augmentation



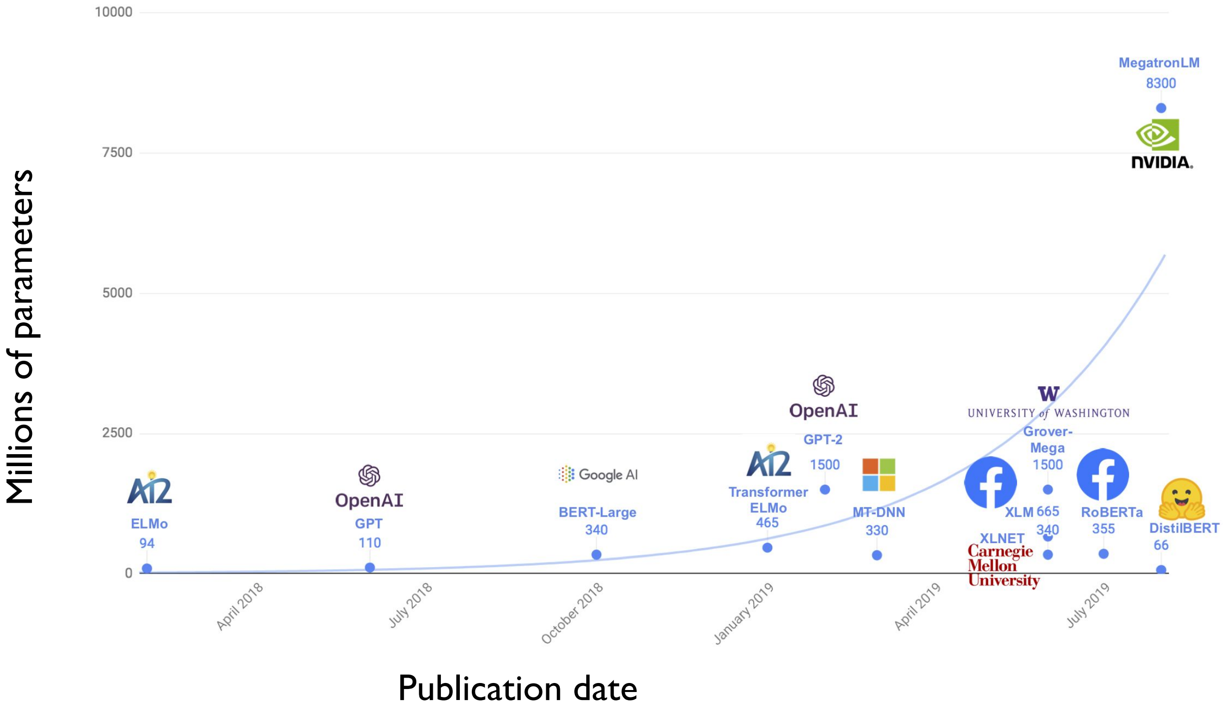





Various DL models on CIFAR 10

**Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules**

# But making models big is also good




Making them good matters too ;) - but often good just means "fast enough so bigger works"

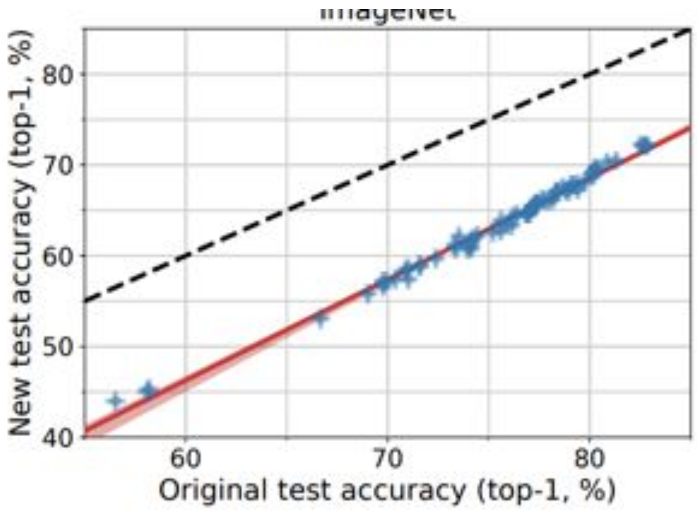
# Validation set

Don’t trust yourself

Overfitting is massive for smaller datasets

Ideally have a part of the dataset you don’t have access to

Even some signs for *huge* datasets (Do ImageNet Classifiers Generalize to ImageNet?, Recht et al, 2019)



# Define a neural network

- Intuitively choose an architecture for the neural network
- Define the components of the model - Fully Connected Layers, Non Linearities, Convolution layers etc.
- Define the forward pass - sequential data flowing through the model components
- Define the backward pass - oh wait, you don’t need to! Pytorch will do that for you using the magical autograd :)

# How to define a network

Start with a network that is known to solve your problem

Alternatively with one that is good at solving a similar problem

Find ways of making it bigger

Try one of the ideas that others tried on similar problems

Come up with a new idea

Is it science/ engineering? Is it evolution? Let's ask at the end of the course.

# Define a loss function

Optimize the function that matters

Choose a function that actually measures what matters

Choose a function whose success you can interpret

What if you can't (e.g. not differentiable)?

Train with the wrong loss function. Then optimize some more with the right one

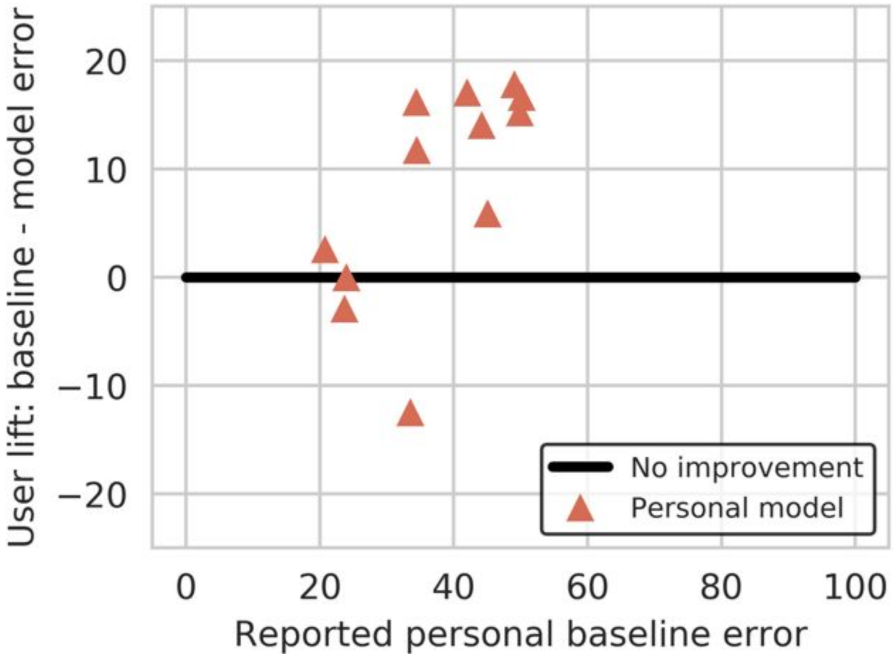
# Example: estimate stress from phone data

Estimate stress=f(phone data) individually for each subjects

R2=1-var(stressPred-stress)/var(stress-popMeanStress)

R2=.8

# ML is sometimes **worse** than no ML




DeMasi Kording Recht, 2017

# Be mindful of regularization

How can we prevent overfitting?

(1) Clever regularization
   (a) Drop-out
   (b) Weight decay
   (c) Early stopping

(2) More data

# Now optimize the Neural network

The optimizers find a good solution on the training data

But optimizers also have all kinds of inductive biases,
 their choice affects generalization performance

Pytorch makes this often quite simple

# Babysitting optimization

You write a new code. We may look for:

Performance does not improve

Improves first then goes down the drain

...

# Test performance

Testing the performance of the model is always important. Why?

- Validation set and training set features might be similar and performance on validation set might not represent performance on unseen data
- It's important to test on unseen data with varied data features so as to estimate the performance of the model when deployed in the real world

# Save and deploy model

Put into production environment

Scale up

Now let us talk about automatic differentiation and how it works in PyTorch.

# Why does it matter?

# ML code from scratch is verbose

```
    w2 = w2_init;
    b2 = b2_init;

    for iter = 1: 5000
        z = x_train * w1 + b1;
        a = 1./(1 + exp(-z));
        eta = 1./(1 + exp(-(a*w2+b2)));
        eps = eta - y_train;
        dgz = a .* (1 - a);

        dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
        db1 = (w2 .* (dgz' * eps))'./m_train;
        dw2 = (eps' * a)'./m_train;
        db2 = sum(eps)/m_train;

        w1 = w1 - step .* dw1;
        b1 = b1 - step .* db1;
        w2 = w2 - step .* dw2;
        b2 = b2 - step .* db2;
    end

    pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
    pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
    pred_y_train = sign(pred_eta_train .* 2 - 1);
    pred_y_train = (pred_y_train + 1) ./ 2;
    err_train = classification_error(pred_y_train, y_train);
    err_train_arr(1, i) = err_train;

    pred_a_test = 1./(1 + exp(-(x_test * w1 + b1)));
    pred_eta_test = 1./(1 + exp(-(pred_a_test * w2 + b2)));
    pred_y_test = sign(pred_eta_test .* 2 - 1);
    pred_y_test = (pred_y_test + 1) ./ 2;
    err_test = classification_error(pred_y_test, y_test);
    err_test_arr(1, i) = err_test;

end

%%
plot(d1_arr, err_train_arr, 'color', 'b'); hold on;
plot(d1_arr, err_test_arr, 'color', 'r'); hold on;
plot(d1_arr, err_cv_arr, 'color', 'g');


%%
disp(err_train_arr); hold on; %0.3960    0.1440    0.1120    0.0760    0.0640    0.0360
disp(err_test_arr); hold on; %0.3939    0.2066    0.1726    0.1563    0.1538    0.1544
disp(err_cv_arr); %0.3960    0.1840    0.1640    0.1000    0.1520    0.1440



%%
w1_init = load(strcat(path_init, "w1_", num2str(5), ".mat"));
```

```

b1_init = load(strcat(path_init, "b1_", num2str(5), ".mat"));
b1_init = b1_init.b1;

w2_init = load(strcat(path_init, "w2_", num2str(5), ".mat"));
w2_init = w2_init.w_2;

b2_init = load(strcat(path_init, "b2_", num2str(5), ".mat"));
b2_init = b2_init.b2;

%%
x_train = trainData;
x_train(:, d) = [];
y_train = trainData(:, d);
y_train = (y_train + 1)./2;

%%
x_test = testData;
x_test(:, d) = [];
y_test = testData(:, d);
y_test = (y_test + 1)./2;

%%
step = 0.1;
%%
w1 = w1_init;
b1 = b1_init;
w2 = w2_init;
b2 = b2_init;
for iter = 1: 5000
    z = x_train * w1 + b1;
    a = 1./(1 + exp(-z));
    eta = 1./(1 + exp(-(a*w2+b2)));
    eps = eta - y_train;
    dgz = a .* (1 - a);

    dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
    db1 = (w2 .* (dgz' * eps))'./m_train;
    dw2 = (eps' * a)'./m_train;
    db2 = sum(eps)/m_train;

    w1 = w1 - step .* dw1;
    b1 = b1 - step .* db1;
    w2 = w2 - step .* dw2;
    b2 = b2 - step .* db2;
end

pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
pred_y_train = sign(pred_eta_train .* 2 - 1);
pred_y_train = (pred_y_train + 1) ./ 2;
```

# Contrast with PyTorch

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```python
model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

# Automated Differentiation...

- Is not Symbolic Differentiation!
- Is not Numerical Differentiation!
- Instead, it relies on a specific quirk of scientific computing to make gradient computation really easy on computers and their users.

# Motivating computational graphs

# Computation for common functions

# Example 1: logarithms

$$\ln(z) = 2 \cdot \operatorname{artanh} \frac{z-1}{z+1} = 2 \left( \frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \cdots \right),$$

# Which game is which?

# Example 2: Inverse Square Roots

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

Observation: every computation in a program boils down to elementary binary functions (+, -, *, /,<<)

But how can we construct derivatives from elementary operations?

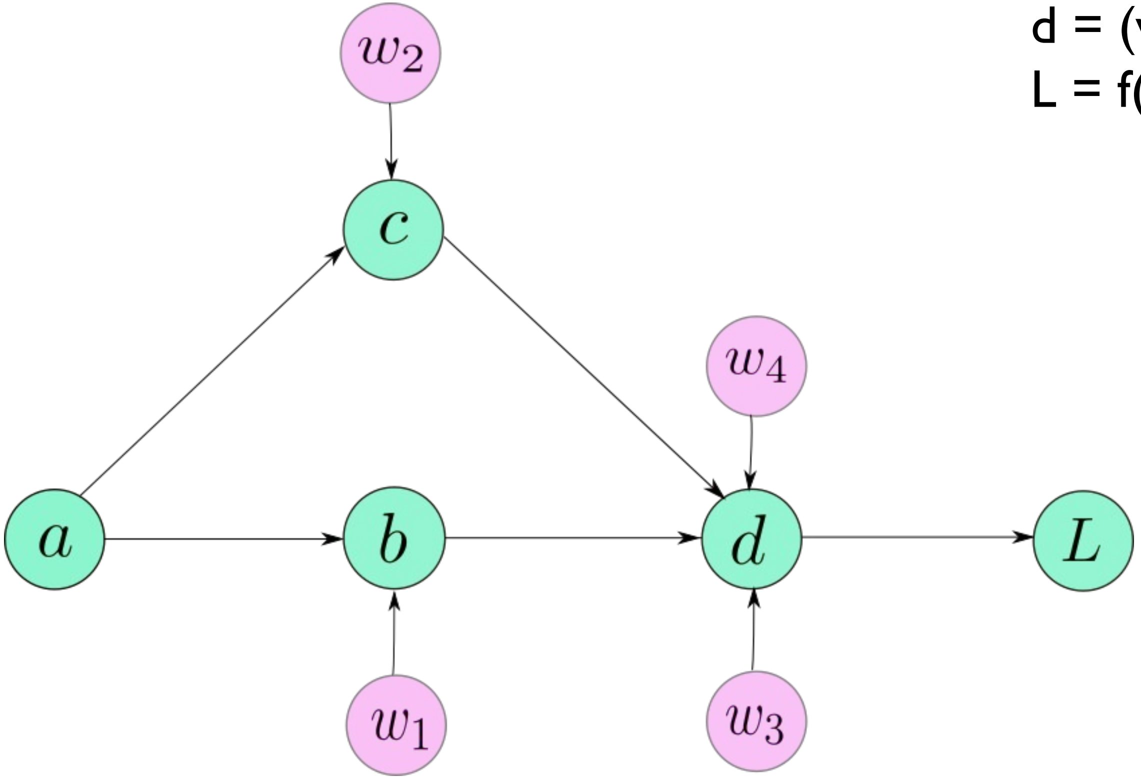
# Computational Graph

Definition: a data structure for storing gradients of variables used in computations.

- Node $v$ represents variable
  - Stores value
  - Gradient
  - The function that created the node
- Directed edge $(u,v)$ represents the partial derivative of $u$ w.r.t. $v$
- To compute the gradient $\partial L/\partial v$, find the unique path from $L$ to $v$ and multiply the edge weights, where L is the overall loss.

# Computational Graph (forward)

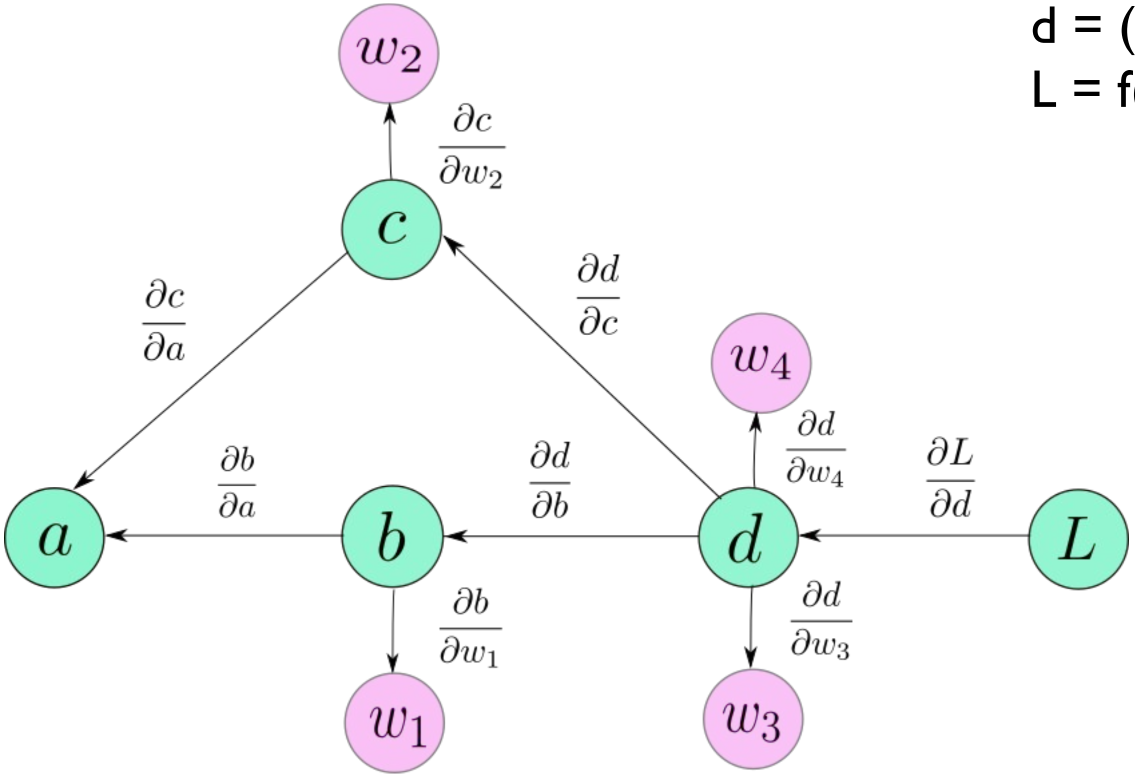
$b = w_1 * a$
$c = w_2 * a$
$d = (w_3 * b) + (w_4 * c)$
$L = f(d)$




https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

# Computational Graph, (backward)

$b = w_1 * a$
$c = w_2 * a$
$d = (w_3 * b) + (w_4 * c)$
$L = f(d)$




Source: https://towardsdatascience.com/getting-started-with-pytorch-part-1-understanding-how-automatic-differentiation-works-5008282073ec

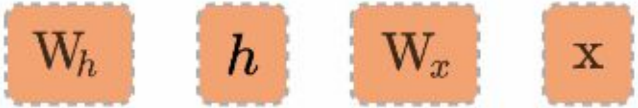
# Why computational graphs are useful

- For a single neuron with *n* inputs, we need to keep track of *O(n)* gradients.
- For a standard 784x800x10 vanilla feedforward neural net for MNIST, we need:
  - 785*800 + 801x10 = 636010 gradients per training example
  - 60,000 training examples
- So gotta be fast at gradients and consider space

- Also, calculating gradients is a compute graph - usable for learning to learn

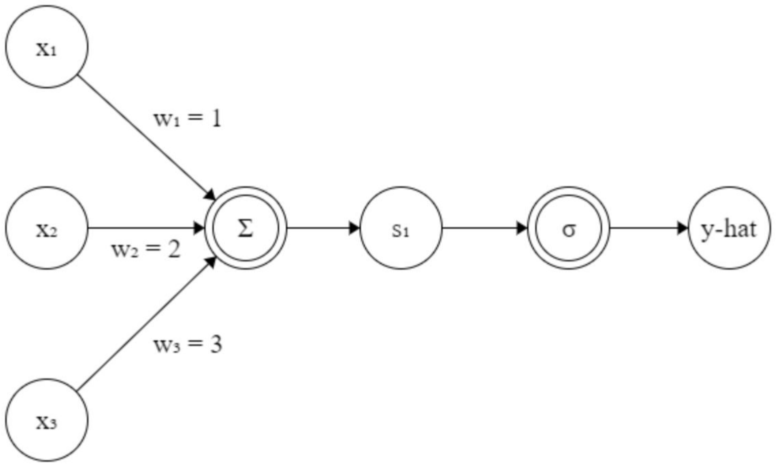
# Computational Graph/ Backpropagation Examples

# A graph is created on the fly



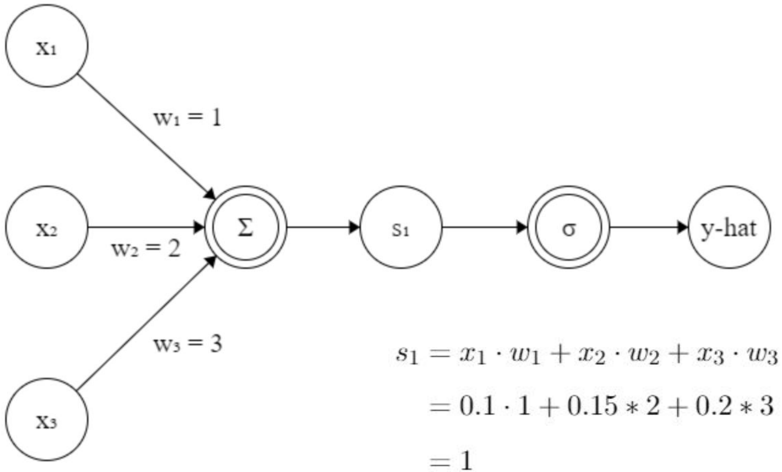

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

# Backpropagation for neural nets: forward pass




$$
\begin{aligned}
s_1 &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \\
&= 0.1 \cdot 1 + 0.15 * 2 + 0.2 * 3 \\
&= 1 \\
\hat{y} &= \sigma(s_1) \\
&= \sigma(1) \\
&= \boxed{0.73}
\end{aligned}
$$

# Backpropagation for neural nets: backward pass

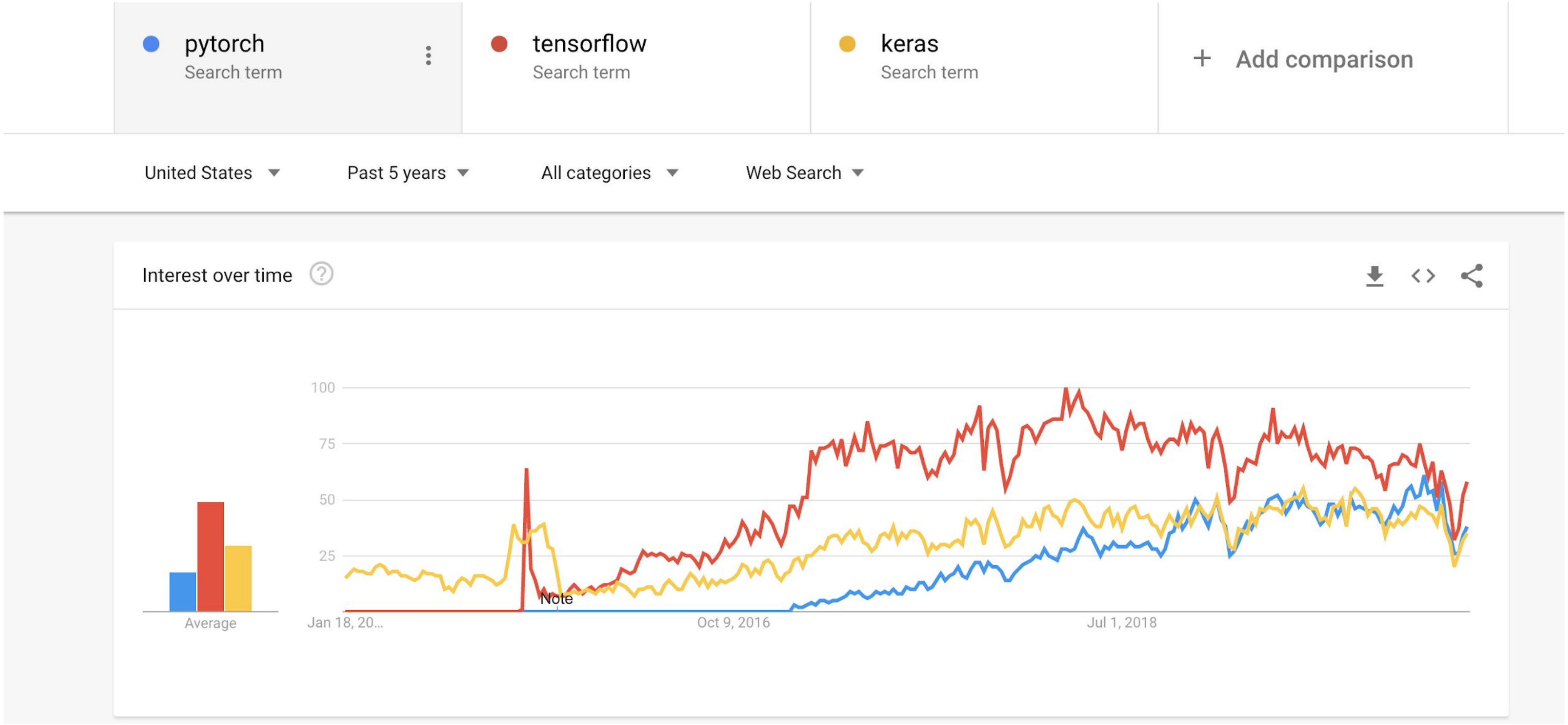



$$
\begin{aligned}
s_1 &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \\
&= 0.1 \cdot 1 + 0.15 * 2 + 0.2 * 3 \\
&= 1 \\
\hat{y} &= \sigma(s_1) \\
&= \sigma(1) \\
&= \boxed{0.73}
\end{aligned}
$$

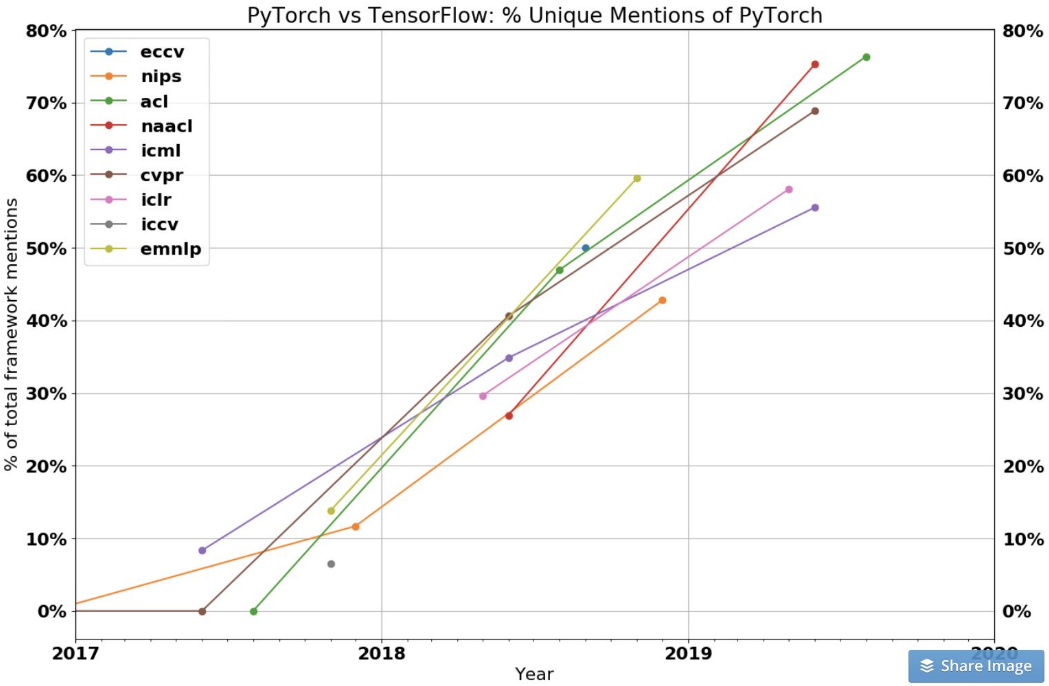
$$
\begin{aligned}
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_1} \\
&= -2 \times (y - \hat{y}) \times \sigma'(1) \times x_1 \\
&= -2 \times (1 - \sigma(1)) \times \sigma(1) \times (1 - \sigma(1)) \times 0.1 \\
&= \boxed{\text{-0.0106}}
\end{aligned}
$$

# Neural Network Packages

# Timetable of packages



# Within academia




thegradient.pub

# Comparison of packages

## TENSORFLOW PROS:

- Simple built-in high-level API.
- Visualizing training with Tensorboard.
- Production-ready thanks to TensorFlow serving.
- Easy mobile support.
- Open source.
- Good documentation and community support.

## TENSORFLOW CONS:

- Static graph.
- Debugging method.
- Hard to make quick changes.

## PYTORCH PROS:

- Python-like coding.
- Dynamic graph.
- Easy & quick editing.
- Good documentation and community support.
- Open source.
- Plenty of projects out there using PyTorch.

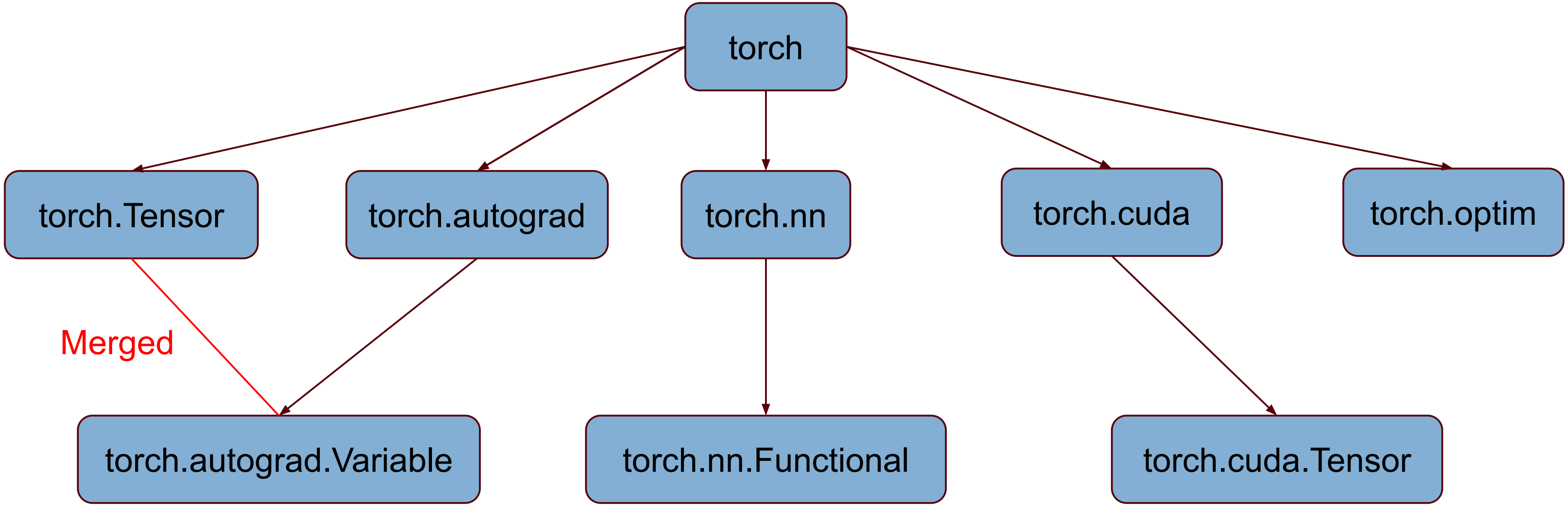
## PYTORCH CONS:

- Third-party needed for visualization.
- API server needed for production.

# PyTorch

- Based on Torch, a scientific computing library for Lua
- Developed by FAIR
- Main features are the built-in computational graph and built-in GPU acceleration

# Structure of PyTorch library



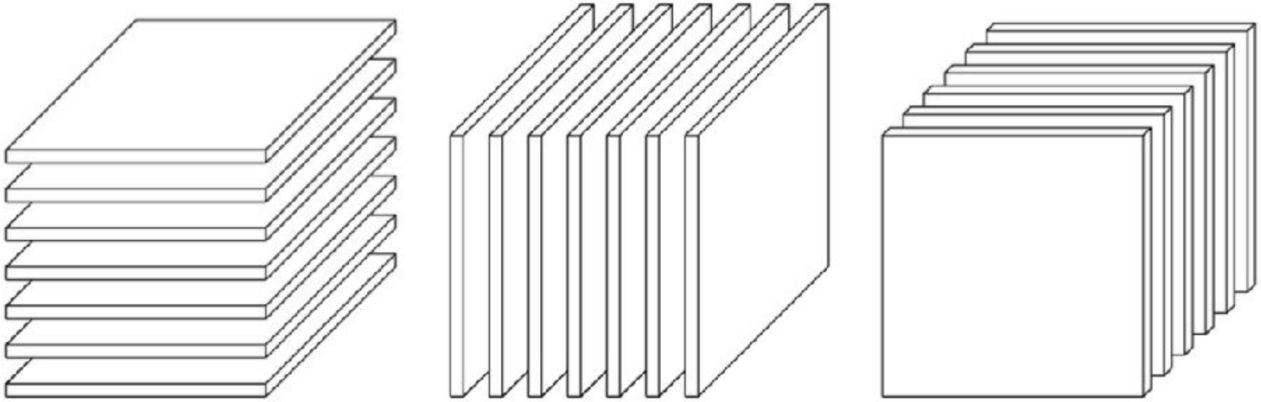
# How do we store numbers?

# torch.Tensor

```
a = torch.rand(10, 10, 5)
print a[0, :, :]
```

# torch.Tensor

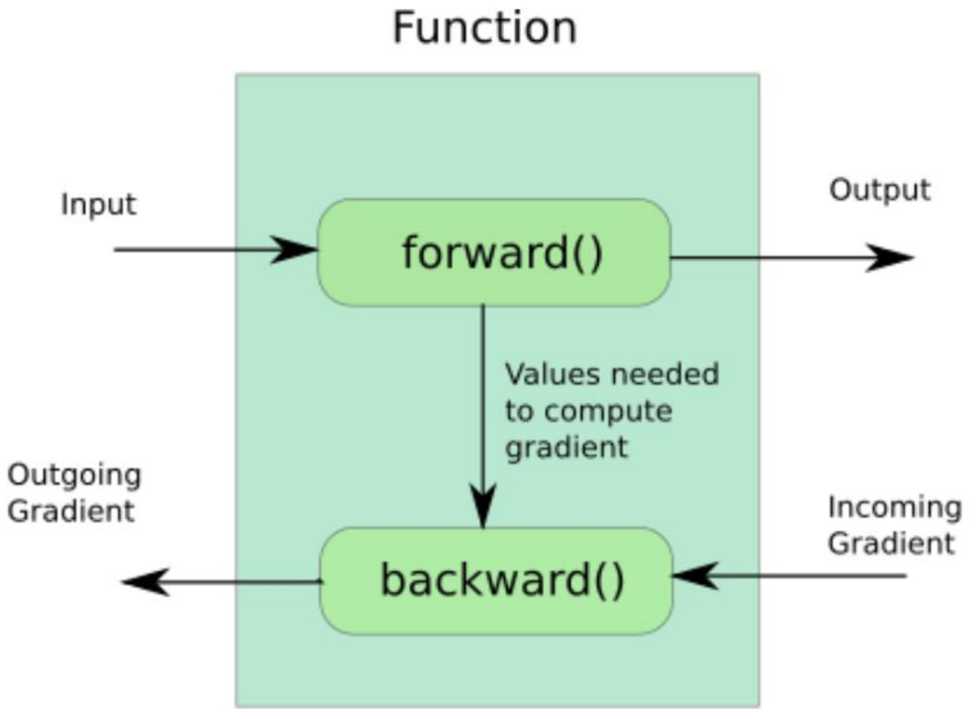
```
a = torch.rand(10, 10, 5)
print a[0, :, :]
```



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

How do we store numbers? Tensors.
Given tensors, how do we track their gradients?

# Functions


Function
Input
forward()
Output
Values needed
to compute
gradient
Outgoing
Gradient
backward()
Incoming
Gradient

How do we store numbers? Tensors.
Given tensors, how do we track their gradients?
Tensors.
Given tensors and their gradients, how do we actually update the parameter values during training?

# torch.nn.optim

- An optimizer is constructed with a model and hyperparameters.

E.g. optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)

```
for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

How do we store numbers? Tensors.

Given tensors, how do we track their gradients? Tensors.

Given tensors and their gradients, how do we actually update the parameter values during training? Optimizers.

How do we do all this on a GPU?

# How PyTorch hides the computational graph


Example:

- PyTorch masks their special built-in addition function in the __add__ method of the class Variable.
- So a+b is really:
- torch.autograd.Variable.__add__(a,b)

How do we store numbers? Tensors.

Given tensors, how do we track their gradients? Variables.

Given tensors and their gradients, how do we actually update the parameter values during training? Optimizers.

How do we do all this on a GPU? CUDA bindings.

**Andrej Karpathy**
@karpathy

Follow

I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.

11:56 AM - 26 May 2017

**399** Retweets **1,532** Likes

# Looking forward

- HW0 is due today. Congratulations to the many good submissions we got already
- Office hours schedule is on the website. Come!
- Recitations on Friday at 11:00.
- Next lecture (2R), will focus on nonlinearities and loss functions.

# Please briefly give feedback on how this lecture could be better. Or how it sucks. Or how you love it.

# Slides for Recitation

# CUDA integration

- For a variable *x*, we can simply write:
  - x = x.cuda() # or
  - x = x.to(device) # if we have a previously defined device
- To accelerate computations on *x* via GPU!
- This casts *x.data* to an object of type torch.cuda.FloatTensor() and changes the magic methods associated with *x*, which are now written in Nvidia's CUDA API.

# torch.Tensor

```
In [1]: import torch

In [2]: import numpy as np

In [3]: arr = np.random.randn((3,5))

In [4]: arr
Out[4]:

array([[-1.00034281, -0.07042071,  0.81870386],
       [-0.86401346, -1.4290267 , -1.12398822],
       [-1.14619856,  0.39963316, -1.11038695],
       [ 0.00215314,  0.68790149, -0.55967659]])

In [5]: tens = torch.from_numpy(arr)

In [6]: tens
Out[6]:

-1.0003 -0.0704  0.8187
-0.8640 -1.4290 -1.1240
-1.1462  0.3996 -1.1104
0.0022  0.6879 -0.5597
[torch.DoubleTensor of size 4x3]
```

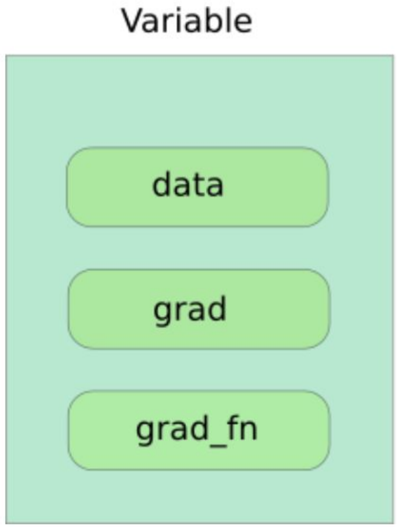
```
In [7]: another_tensor = torch.LongTensor([[2,4],[5,6]])

In [7]: another_tensor
Out[13]:

 2  4
 5  6
[torch.LongTensor of size 2x2]

In [8]: random_tensor = torch.randn((4,3))

In [9]: random_tensor
Out[9]:

1.0070 -0.6404  1.2707
-0.7767  0.1075  0.4539
-0.1782 -0.0091 -1.0463
 0.4164 -1.1172 -0.2888
[torch.FloatTensor of size 4x3]
```

# Tensors: common manipulations

- torch.cat(tensors, dim=0, out=None) → Tensor
  - Concatenates a list of Tensors along an existing dimension
- torch.reshape(input, shape) → Tensor
  - Reforms the dimensions of a Tensor
- torch.squeeze(input, dim=None, out=None) → Tensor
  - Removes a dimension from a Tensor
- torch.stack(seq, dim=0, out=None) → Tensor
  - Concatenates a list of Tensors along a new dimension
- torch.unsqueeze(input, dim, out=None) → Tensor
  - Creates a dimension

# Variables

This is the class in PyTorch that corresponds to nodes in the computational graph.




Tensor

Float

Function object

# torch.nn.Functional

Many utility functions for specific architectures of neural nets.

Example utility functions for vanilla neural nets:

- torch.nn.functional.linear(input, weight, bias=None)
- torch.nn.functional.dropout(input, p=0.5, training=True, inplace=False)

# torch.nn.Functional

Many utility functions for specific architectures of neural nets.

Example activation functions:

- torch.nn.functional.relu_(input) → Tensor
- torch.nn.functional.hardtanh_(input, min_val=-1., max_val=1.) → Tensor
- torch.nn.functional.leaky_relu(input, negative_slope=0.01, inplace=False) → Tensor
- torch.nn.functional.softmax(input, dim=None, _stacklevel=3, dtype=None)

# torch.nn.Functional

Many utility functions for specific architectures of neural nets.

Example functions for CNNs:

- torch.nn.functional.conv1d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1) → Tensor
- torch.nn.functional.conv_transpose2d(input, weight, bias=None, stride=1, padding=0, output_padding=0, groups=1, dilation=1) → Tensor
- torch.nn.functional.max_pool2d(*args, **kwargs)

# torch.nn.Functional

Many utility functions for specific architectures of neural nets.

Example normalization functions:

- torch.nn.functional.batch_norm(input, running_mean, running_var, weight=None, bias=None, training=False, momentum=0.1, eps=1e-05)
- torch.nn.functional.normalize(input, p=2, dim=1, eps=1e-12, out=None)
- torch.nn.functional.instance_norm(input, running_mean=None, running_var=None, weight=None, bias=None, use_input_stats=True, momentum=0.1, eps=1e-05)

# torch.nn.Functional

Many utility functions for specific architectures of neural nets.

Example loss functions:

- torch.nn.functional.cosine_similarity(x1, x2, dim=1, eps=1e-8) → Tensor
- torch.nn.functional.binary_cross_entropy(input, target, weight=None, size_average=None, reduce=None, reduction='mean')
- torch.nn.functional.hinge_embedding_loss(input, target, margin=1.0, size_average=None, reduce=None, reduction='mean') → Tensor
- torch.nn.functional.kl_div(input, target, size_average=None, reduce=None, reduction='mean')

# Feedforward Network in PyTorch

# Defining a Neural Net in PyTorch

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

# Training a Neural Net in PyTorch

```python
model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```