Introduction to Deep Learning

Abhishek Das
Facebook AI Research

08/25/2020

@abhshkdz · abhishekdas.com · abhshkdz@fb.com

Outline

- From linear classifiers to neural networks

- Activation functions

- Loss functions

- Backpropagation

- Gradient descent

- Recurrent neural networks

- Reinforcement learning

Image Classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike writing a function to sort a list of numbers

No obvious way to hard-code the algorithm for recognizing a cat

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Supervised Learning

- Input: x
- Output: y

- Unknown target function
  - f: x —> y

- Data: $\{(x_1, y_1), \ldots, \{x_N, y_N\})$

- Loss function: how good is a model with respect to my data?

- Machine learning: finding the best model for a given dataset + loss function

Image Classification on CIFAR-10

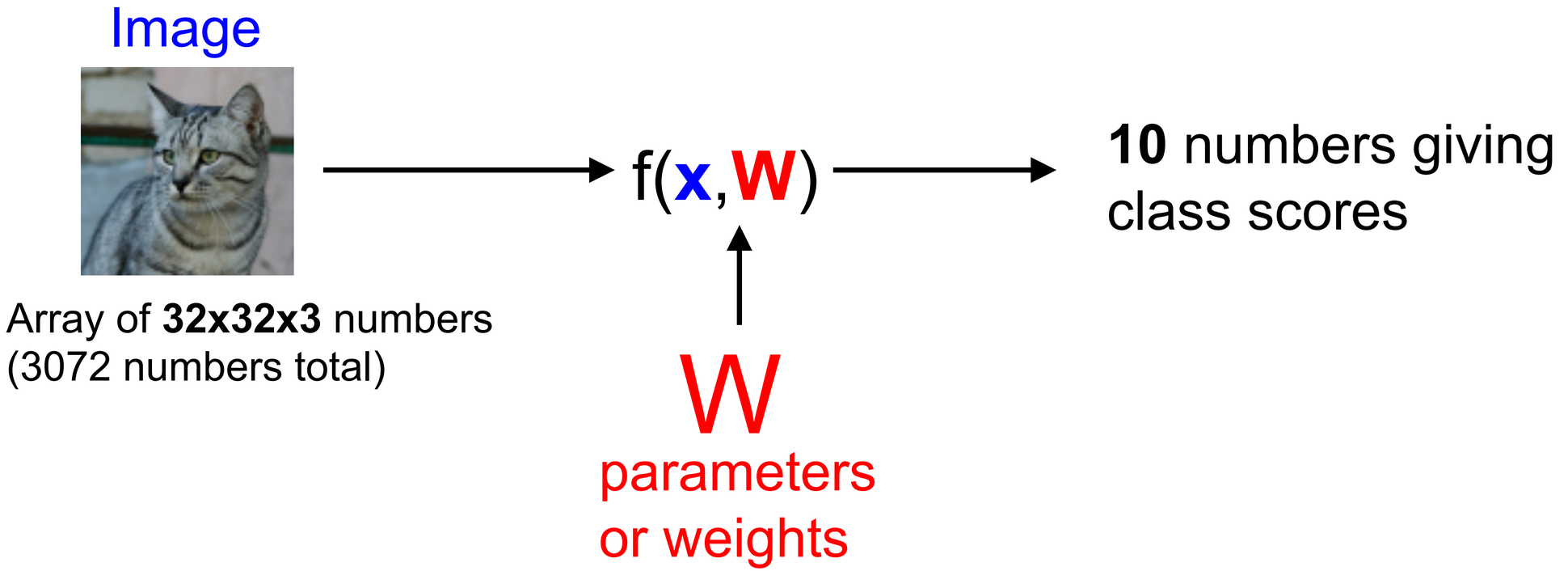
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

50000 training images
10000 test images

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification: linear classifier



Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification: linear classifier

$y' = f(x, W) = W x$

x: 1 x 3072
W: 3072 x 10
y': 1 x 10

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving class scores

W
parameters or weights

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification: neural network

$$y' = f(x, W) = W_2 \max(0, W_1 x)$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving
class scores

W
parameters
or weights

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification: neural network

$y' = f(x, W) = W_2 \max(0, W_1 x)$

Variable **no. of layers**, for example:

$f(x, W) = W_3 \max(0, W_2 \max(0, W_1 x))$

$f(x, W) = W_n \max(0, (\dots (\dots (\dots W_1 x))))$

Image Classification: neural network

$y' = f(x, W) = W_2 \max(0, W_1 x)$

Variable **no. of layers**, for example:

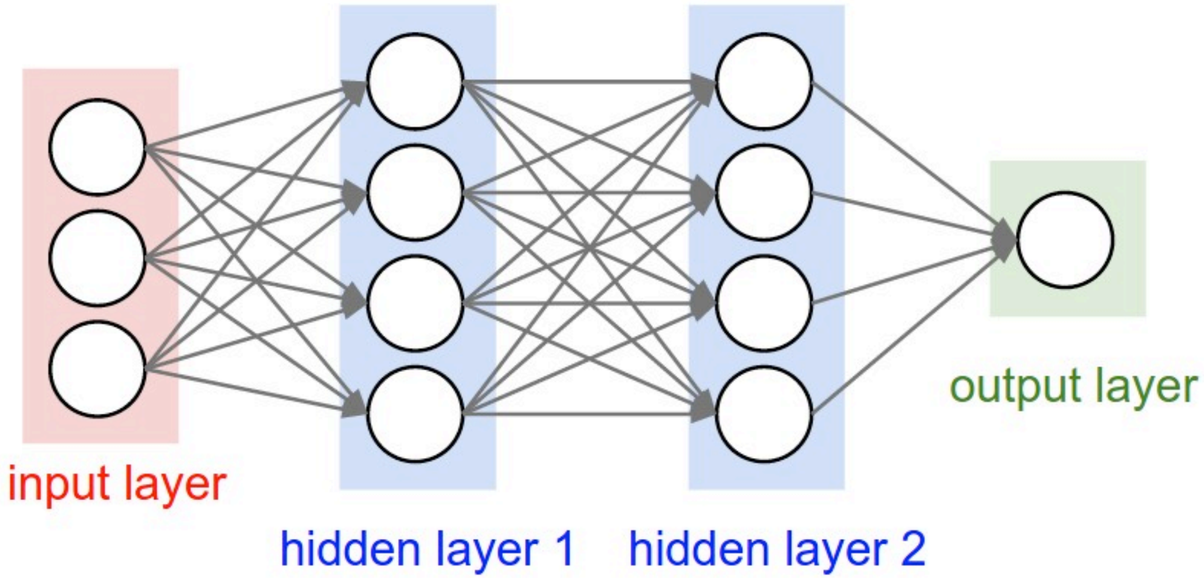


Image Classification: neural network

$y' = f(x, W) = W_2 \max(0, W_1 x)$

Variable **no. of layers**, for example Residual Network:

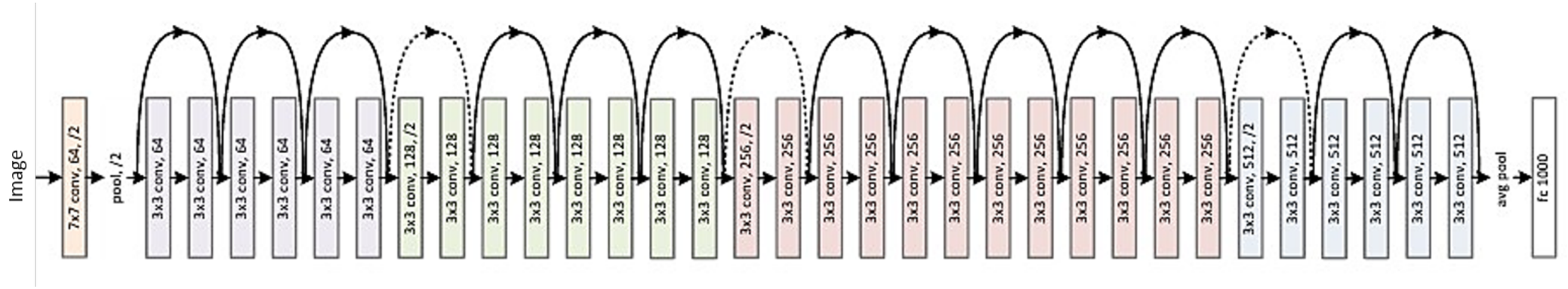


Image credit: He et al., 2016a, b

Image Classification: neural network

$y' = f(x, W) = W_2 \max(0, W_1 x)$

Different **activation functions**, for example:

$f(x, W) = W_2 \text{ReLU}(W_1 x) = W_2 \max(0, W_1 x)$ [ReLU = rectified linear unit]

$f(x, W) = W_2 \tanh(W_1 x)$

$f(x, W) = W_2 \sin(W_1 x)$

... (many more)

Image Classification: neural network

$$y' = f(x, W) = W_2 \max(0, W_1 x)$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving
class scores

W
parameters
or weights

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Image Classification: neural network

$$y' = f(x, W) = W_2 \max(0, W_1 x)$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

W
parameters
or weights

10 numbers giving
class scores

- Loss function: how good is **f** with respect to my data?

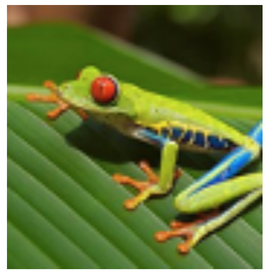
- Machine learning: finding the best **f** for a given dataset + loss function

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:





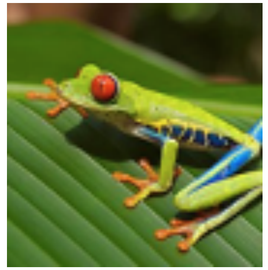
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:





cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

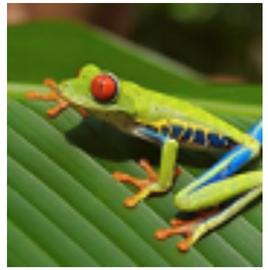
and using the shorthand for the scores vector: $s = f(x_i, W)$

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:





cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Example code: linear classifier + multiclass SVM loss

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:

cat	**3.2**
car	5.1
frog	-1.7

Cross-entropy loss

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$L_i = -\log P(Y = y_i|X = x_i)$$

$$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Loss functions

Suppose: 3 training examples, 3 output classes
With some W, the output scores f(x, W) are:

Cross-entropy loss

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

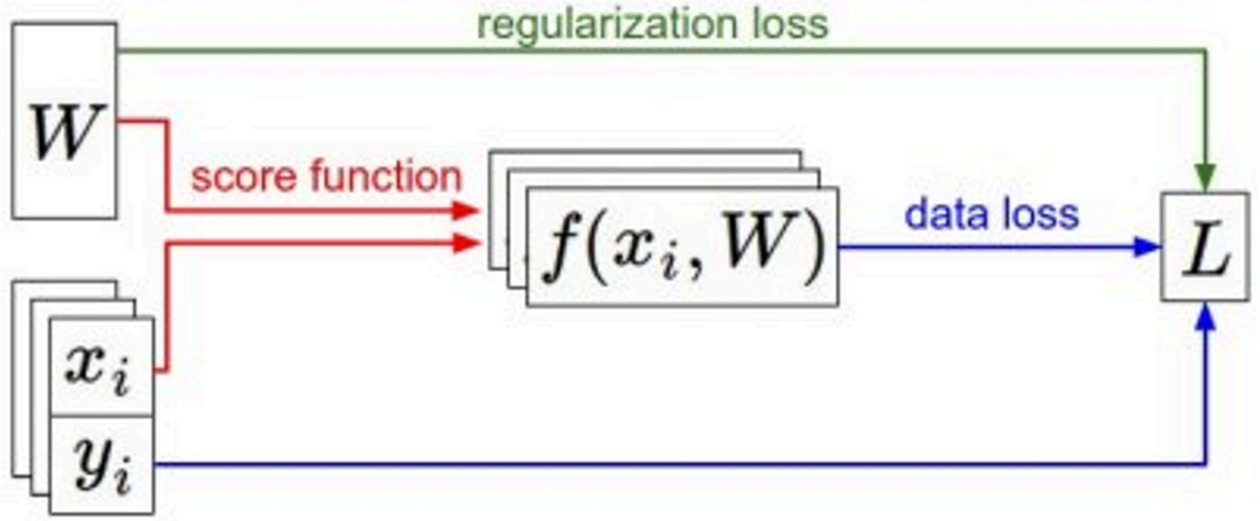
$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities	compare	Correct probs
cat	3.2		24.5		0.13		1.00
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Probabilities must be >= 0

Probabilities must sum to 1

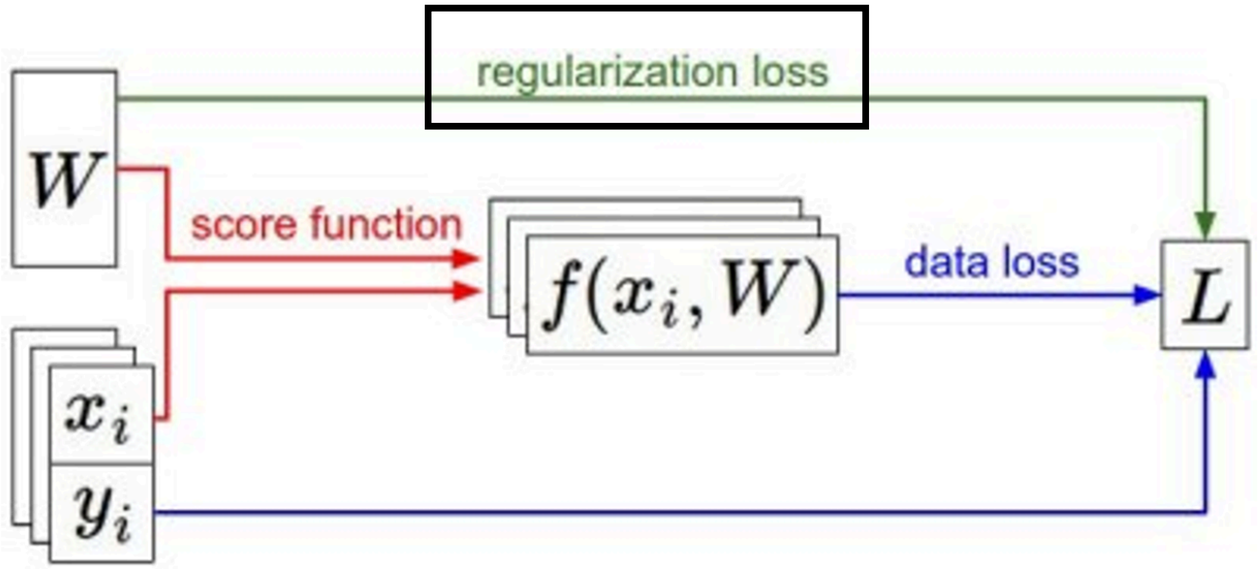
So far…



Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

So far…

Prevents overfitting
Prefers simpler models



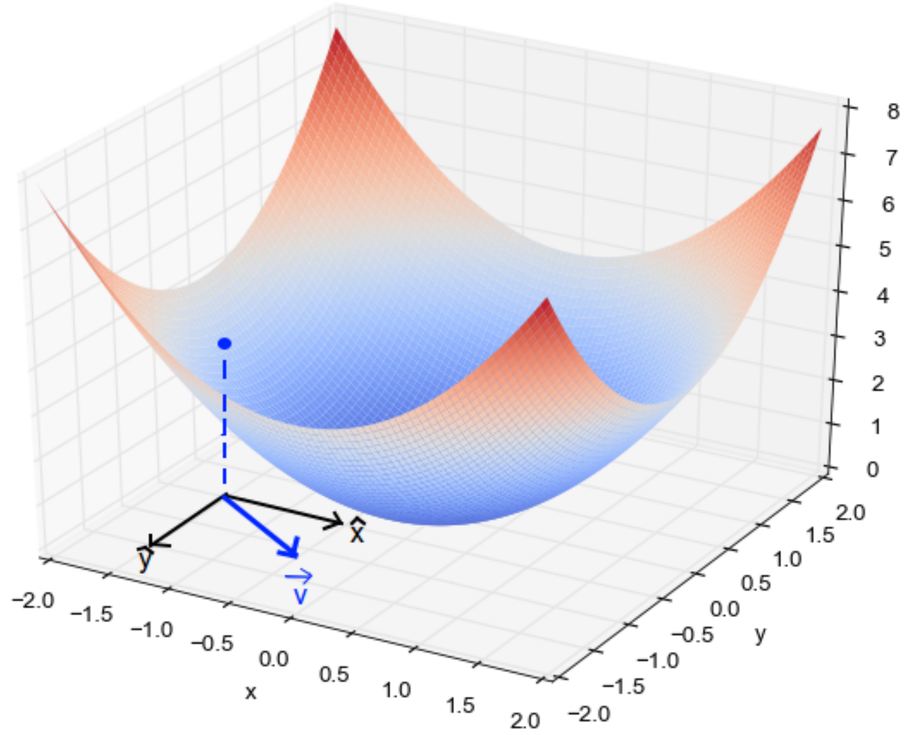
Slide credit: Fei Fei Li, Justin Johnson, Serena Young, CS 231n

Backpropagation

$$f(x, W) = W_2 \max(0, W_1 x)$$

Backpropagation

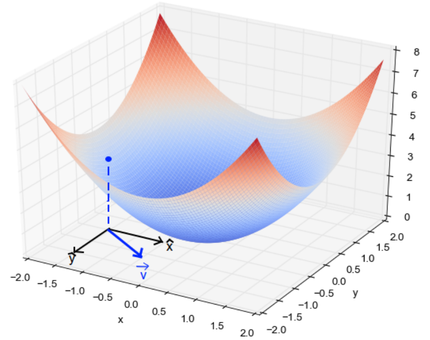
$$f(x, W) = W_2 \max(0, W_1 x)$$



Loss surface L(f(x, W))

Backpropagation

$f(x, W) = W_2 \max(0, W_1 x)$



$$\frac{\partial L}{\partial W}$$

derivative / gradient points towards slope or direction of steepest ascent

updating weights in the opposite direction of gradient should decrease loss!

Backpropagation

$f(x, W) = W_2 \max(0, W_1 x)$
$= W_2 \, g(x, W_1)$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial W}$$

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial W_1} = W_2 \frac{\partial g}{\partial W_1}$$

Backpropagation

$$f(x, W) = W_2 \max(0, W_1 x)$$
$$= W_2\, g(x, W_1)$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial W}$$

$$\frac{\partial f}{\partial W_1} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial W_1} = W_2 \frac{\partial g}{\partial W_1}$$

computing gradients with chain rule is referred to as **backpropagation**

Gradient descent

$$W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}$$

learning rate

Make an update to W in the opposite direction of $\frac{\partial L}{\partial W}$

Learning rate controls the **step size**

Gradient descent

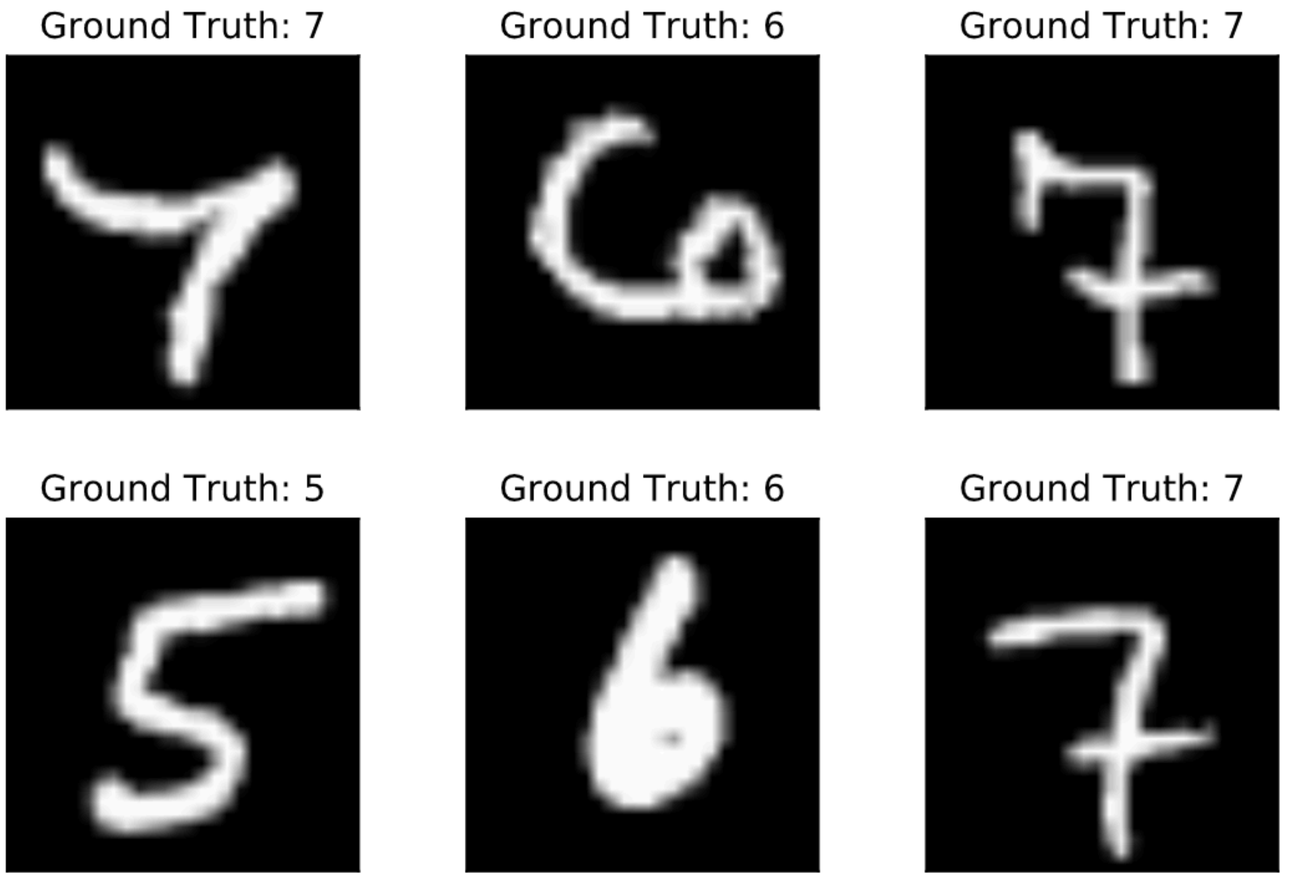
$$W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}$$

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

MNIST Tutorial in Pytorch

https://github.com/pytorch/examples/blob/master/mnist/main.py



MNIST Tutorial in Pytorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Defining layers of the neural network

Recall $f(W, x) = W_2 \max(0, W_1 x)$

MNIST Tutorial in Pytorch

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Going from images **x** to output scores

MNIST Tutorial in Pytorch

Loading the dataset

```
train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)
```

```
model = Net().to(device)
optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

MNIST Tutorial in Pytorch

```
train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

model = Net().to(device)
optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

Loading the model

MNIST Tutorial in Pytorch

```
train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

model = Net().to(device)
optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

Loading the optimizer

Recall $W_{t+1} = W_t - \eta \frac{\partial L}{\partial W}$

MNIST Tutorial in Pytorch

```
    train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Sampling subsets of the dataset

MNIST Tutorial in Pytorch

```
    train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Computing output scores from the model

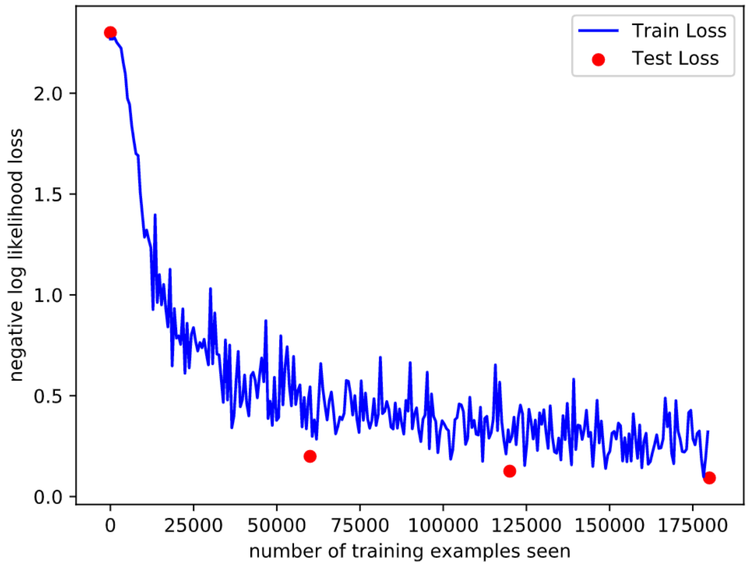
MNIST Tutorial in Pytorch

```
    train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```



Computing cross-entropy loss

MNIST Tutorial in Pytorch

```
    train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Backpropagating gradients

Recall that we did not need to specify the gradient calculation equations. It is done automatically!

MNIST Tutorial in Pytorch

```
    train_loader = torch.utils.data.DataLoader(dataset1,**kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
```

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Weight update

MNIST Tutorial in Pytorch

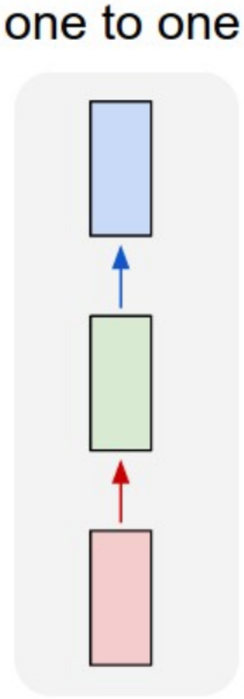
https://github.com/pytorch/examples/blob/master/mnist/main.py

Loading data, defining the model, training, testing
in ~120 lines of code!

Pytorch is currently the industry-standard deep learning library. It has great tutorials and is easy to learn.
Highly recommended! http://pytorch.org/

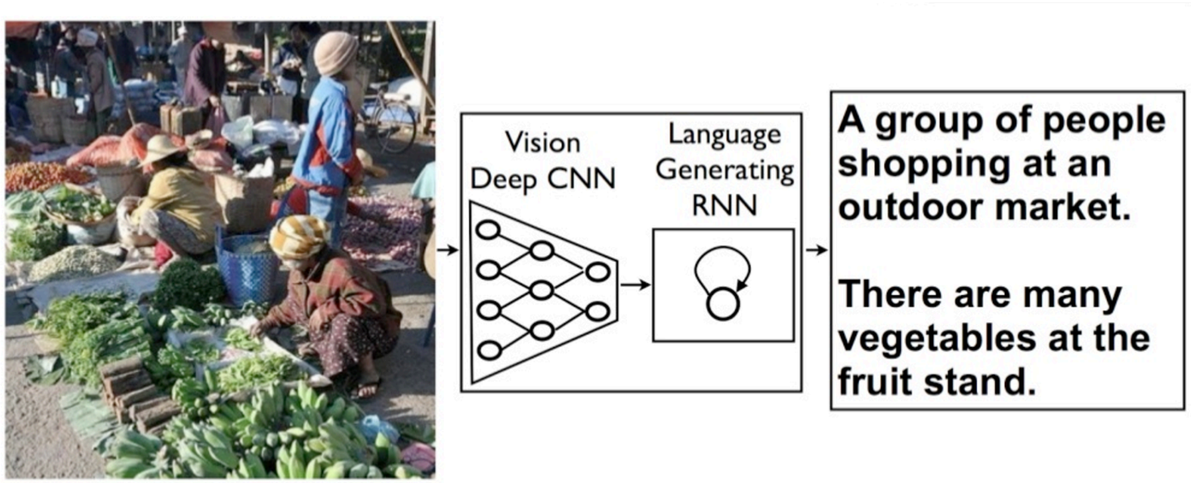
Recurrent Neural Networks

Image classification: input image to output category



Recurrent Neural Networks

Image captioning: input image to a sentence description



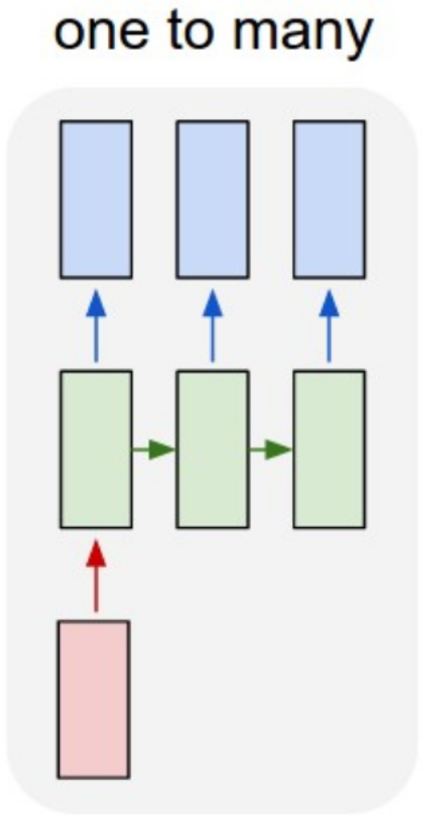
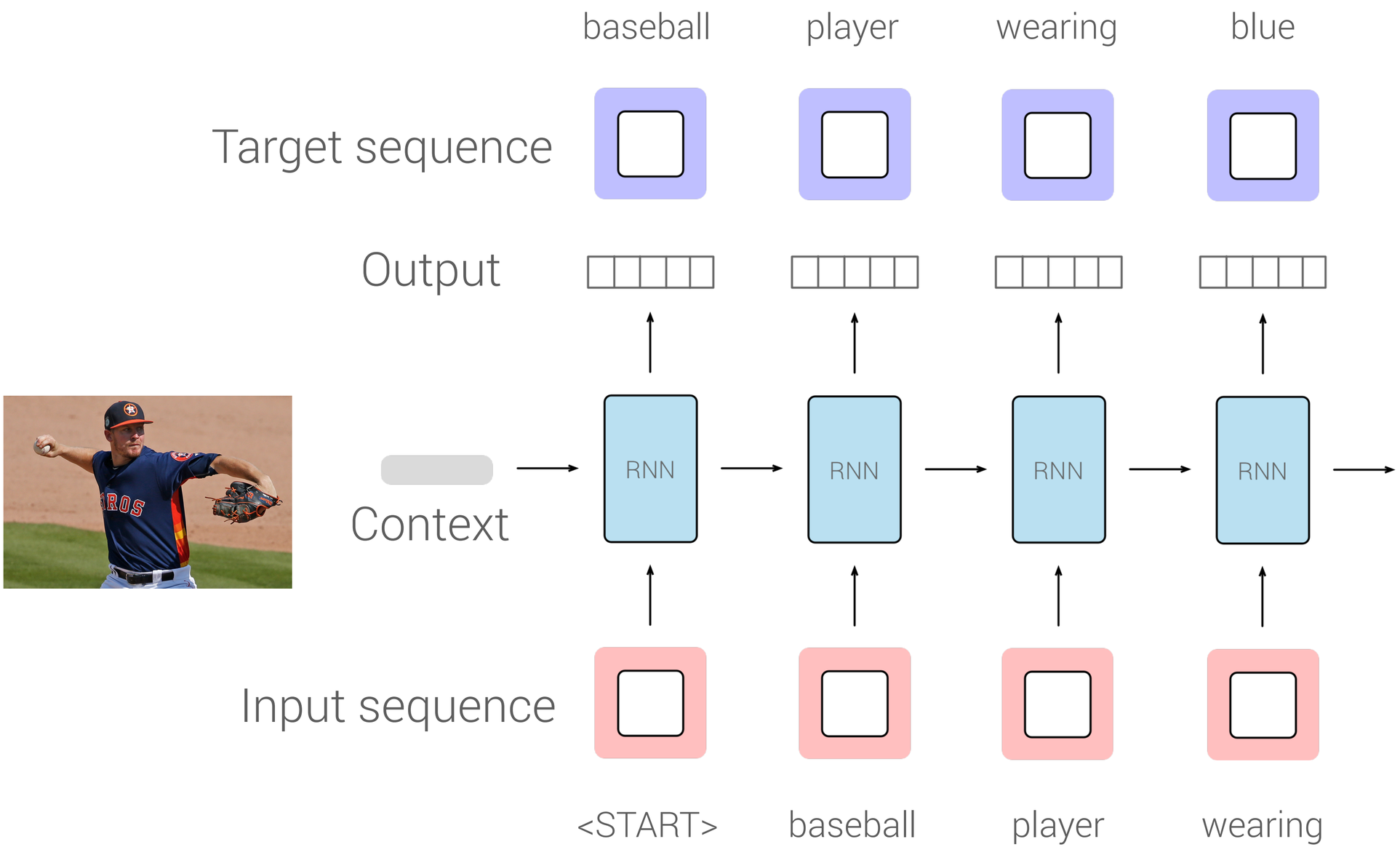
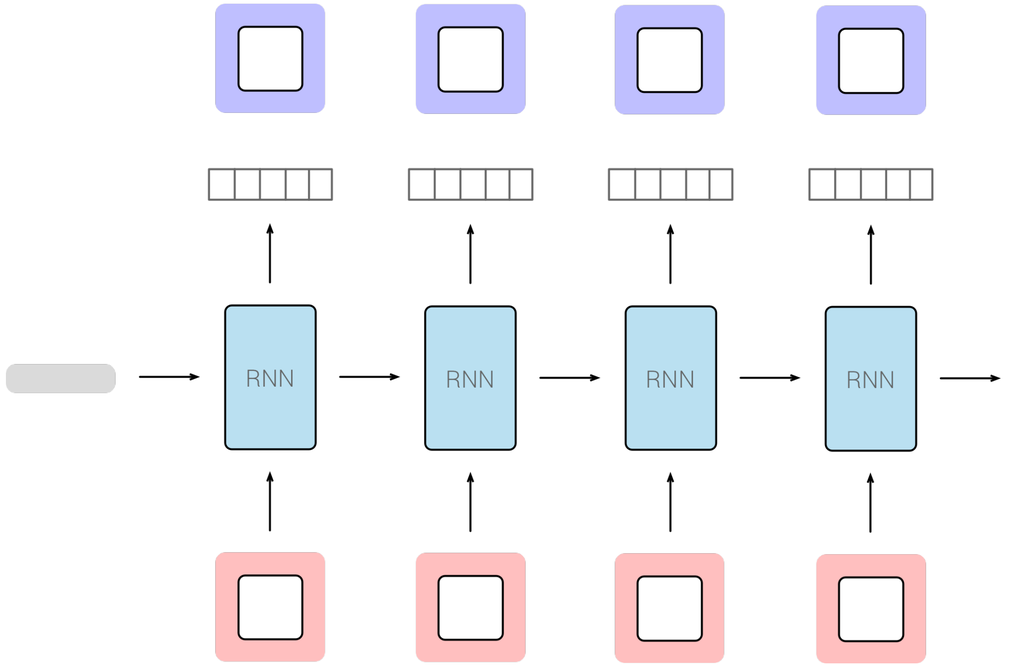


Image credit: Vinyals et al., 2015

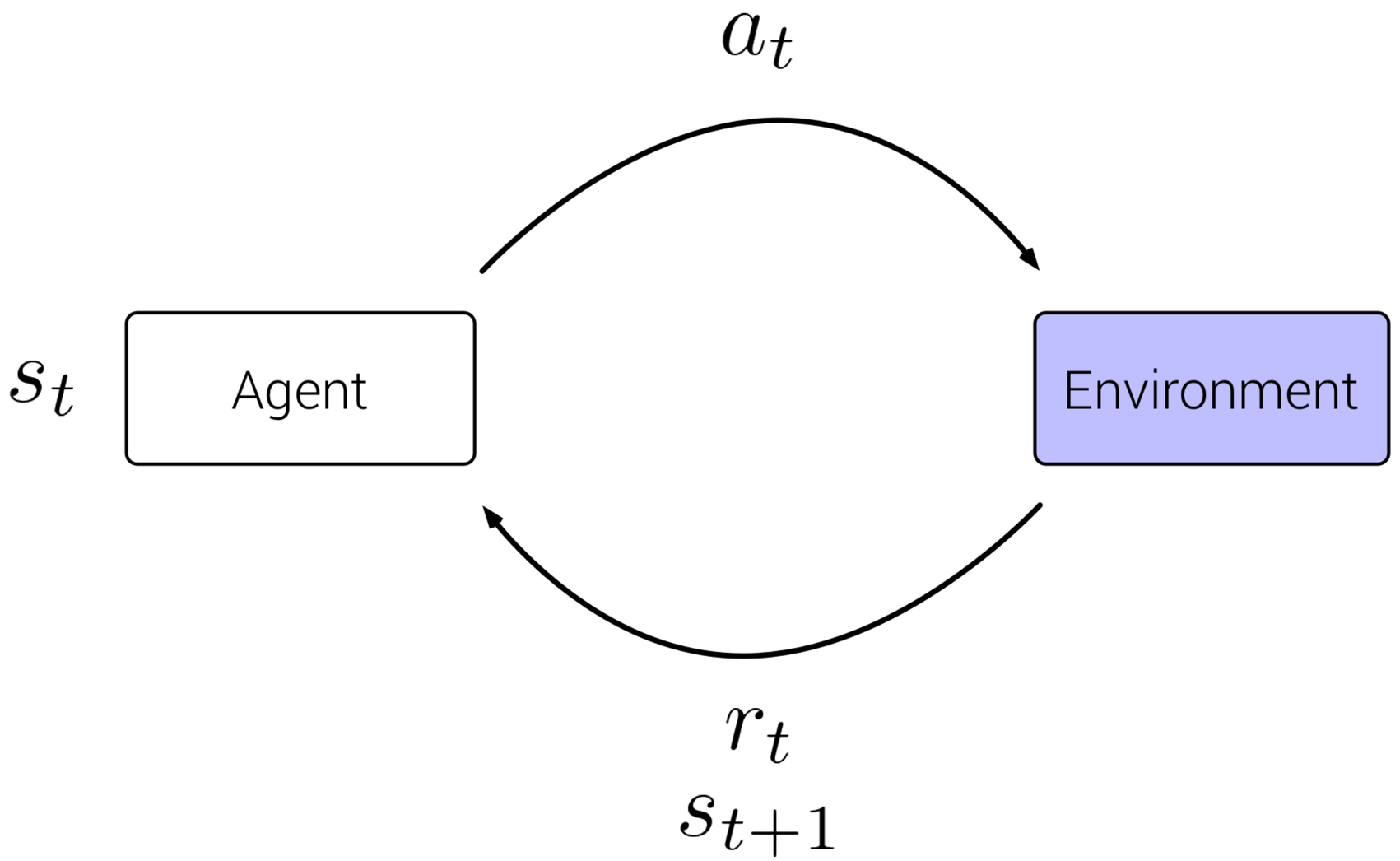

baseball
player
wearing
blue
Target sequence
Output
RNN
RNN
RNN
RNN
Context
Input sequence
<START>
baseball
player
wearing

Recurrent Neural Networks

$$h_{t+1} = W_{hh} h_t + W_{xh} x_t$$

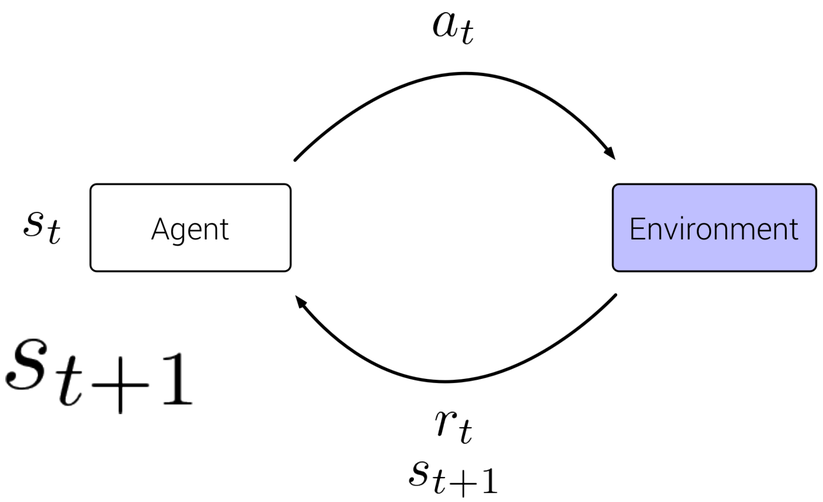


Reinforcement learning



Reinforcement learning

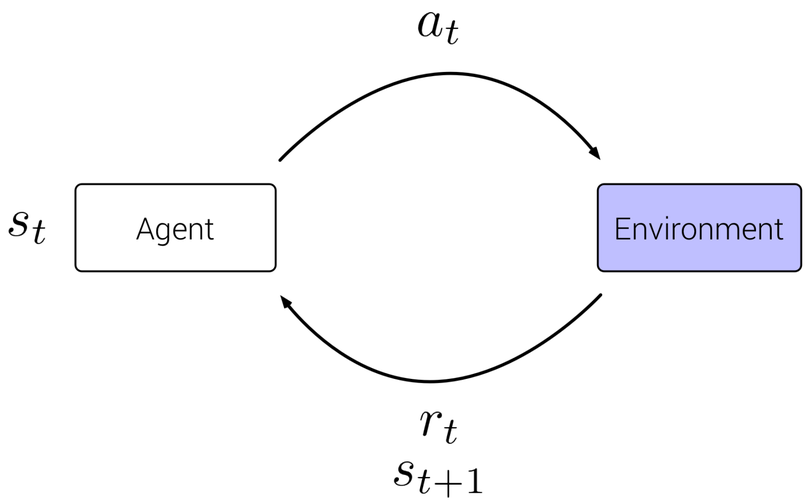
- While not ‘done’
  - Agent selects action a_t
  - Environment returns reward r_t and next state s_{t+1}



- Policy π specifies which action to take in each state
- Objective: maximizing cumulative discounted future reward $\sum_{t=0}^{\infty} \gamma^t r_t$

Reinforcement learning

- Salient features of an RL problem
  - Some notion of sequential decision making
  - Reward function is non-differentiable.
    So we cannot compute $\frac{\partial L}{\partial W}$ exactly



Reinforcement learning

- Approximating gradients in REINFORCE:

  f is the reward function, p is our neural network,
  theta are the weights of the neural network, x is the sampled action

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= E_x[f(x) \nabla_\theta \log p(x)] && \text{definition of expectation}
\end{aligned}
$$

Useful pointers

- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
  http://cs231n.stanford.edu/
- Deep Learning for Computer Vision, University of Michigan
  https://www.youtube.com/watch?v=dJYGatp4SvA&list=PL5-TkQAfAZFbzxjBHtzdVCWE0Zbhomg7r
- FastAI pytorch + neural network tutorials
  https://www.fast.ai/