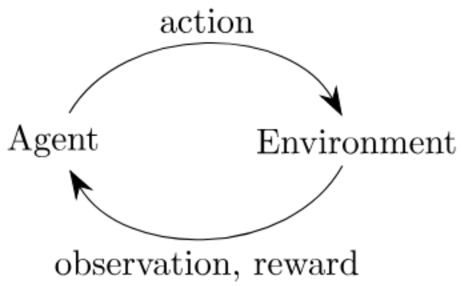
DEEP REINFORCEMENT LEARNING

GYM FRAMEWORK

- Gym is a toolkit for developing and comparing reinforcement learning algorithms
- Collection of test problems (Environments) to implement reinforcement learning algorithms
- https://gym.openai.com/envs/#classic_control

AGENT – ENVIRONMENT LOOP



Retrieved from https://gym.openai.com/docs/

- Each timestep the agent chooses an action
- Environment returns observation and reward

- Gym Environment returns:
  - Observation: Environment-specific object (pixel data from a camera, observations of a robot, …)
  - Reward: Amount of reward achieved by the previous action
  - Done: The episode has terminated
  - Info: Diagnostic information useful for debuggin

IMAGE TRANSFORMATIONS

- Based on the publication „Playing Atari with Deep Reinforcement Learning“ (https://arxiv.org/pdf/1312.5602v1.pdf)

```python
class ImageProcessor:

    def __init__(self, size, device):
        self.transformations = T.Compose([T.ToPILImage(),
                                          T.Grayscale(),
                                          T.Resize(size),
                                          T.ToTensor()])
        self.device = device

    def process(self, screen):
        # crop the image to a square image
        screen = screen[34:-16, :, :]
        return self.transformations(screen).squeeze(0).to(self.device)
```

TEMPORAL DIFFERENCE LEARNING

- Q-Learning
- Goal: learn a policy, which tells an agent which action to take under which circumstances
- Exploration: Random pick of an action
- Eventually finds an optimal policy

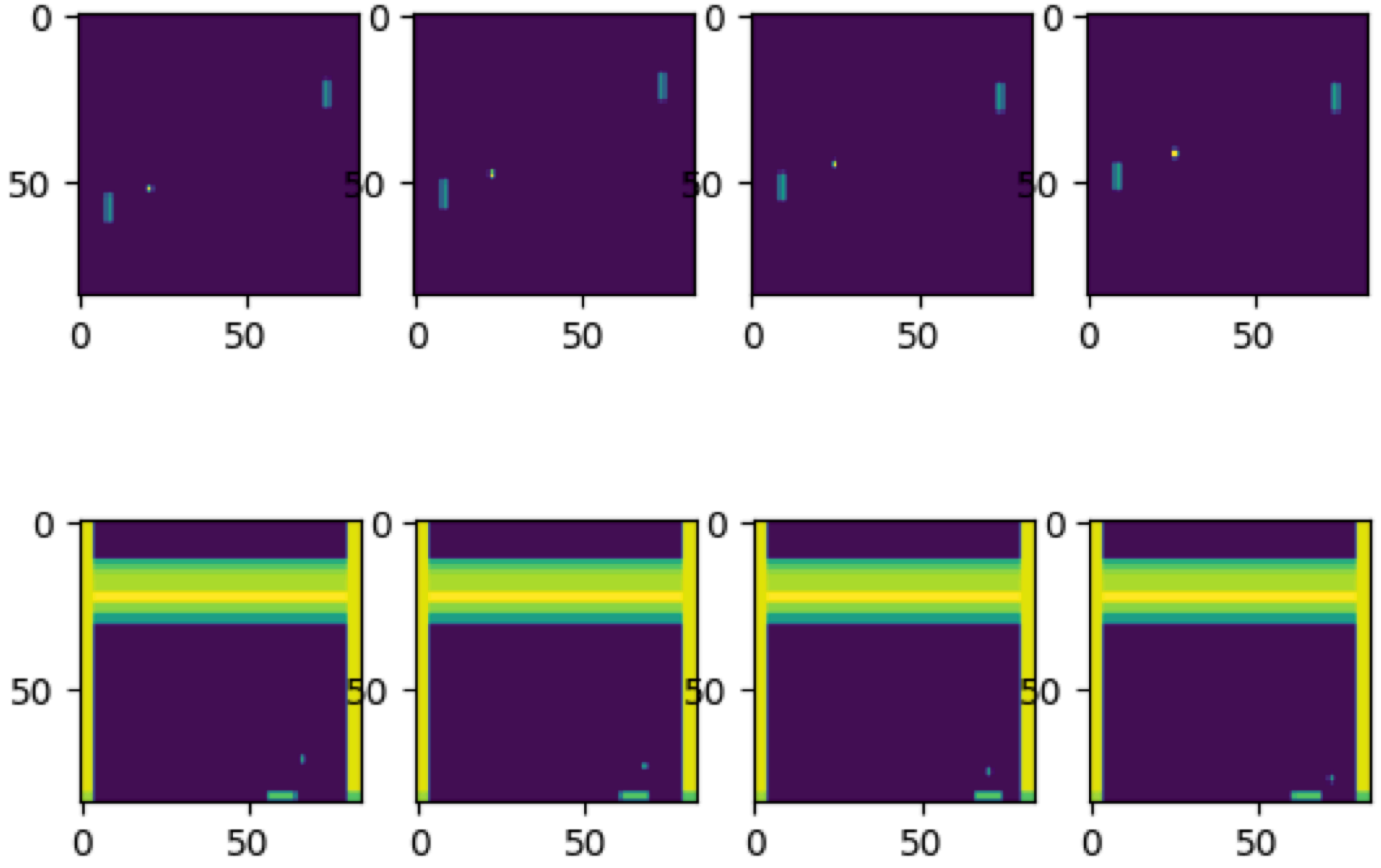
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Retrieved from https://en.wikipedia.org/wiki/Q-learning

DQN

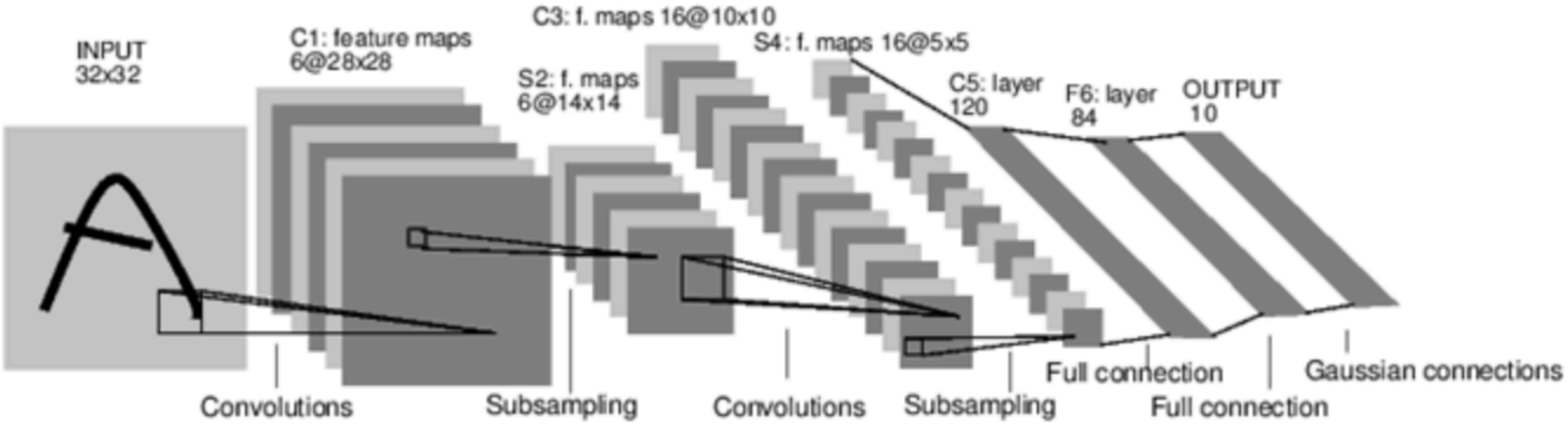
- 4 Input Channels => 4 Input Images
- Image Sequence of the last 4 images
- Network can detect Location, Direction and Velocity of the Input
- Output: Number of available Actions
- Batch Training

INPUT



CONVOLUTION

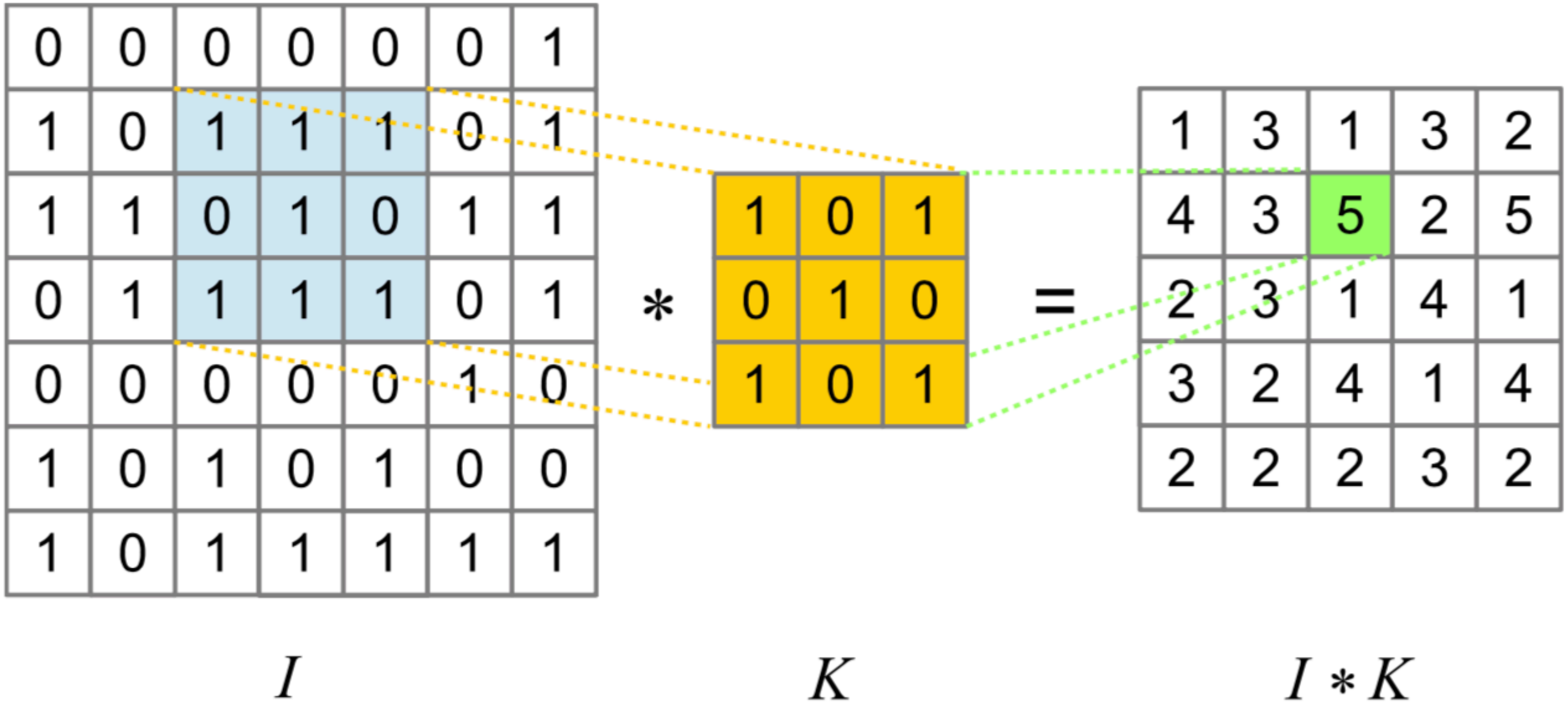
- Motivation:
  - Shared weigths => reduces the complexity and size of the network
  - Equivariance => convolutions are equivariant to many data transformation operations which means that we can predict how specific changes in the input will be reflected in the output



A Full Convolutional Neural Network (LeNet)

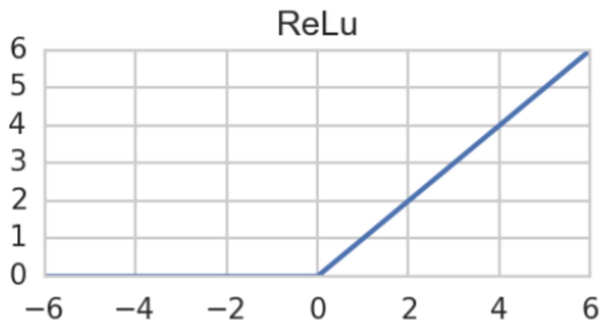
Retrieved from http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

CONVOLUTION



ACTIVATION FUNCTION

- Determines the output of the neutron
- Rectified Linear Unit:



Retrieved from http://www.cbcity.de/tag/neural-net

DQN

```python
class DQN(nn.Module):

    def __init__(self, in_channels, num_actions):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, 32, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
        self.fc4 = nn.Linear(3136, 512)
        self.fc5 = nn.Linear(512, num_actions)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.fc4(x.view(x.size(0), -1)))
        return self.fc5(x)
```

ERROR FUNCTION

- Bellmann Error: $$\varepsilon = r + \gamma P\hat{v} - \hat{v} = r + \gamma P\Phi\theta - \Phi\theta.$$

Retrieved from https://en.wikipedia.org/wiki/Automatic_basis_function_construction#Bellman_error_basis

```
# Compute Bellman error
# r + gamma * Q(s',a', theta_i_frozen) - Q(s, a, theta_i)
error = reward_batch + GAMMA * q_s_a_prime - q_s_a
```

OPTIMIERUNGSALGORITHMUS

- Adam (Adaptive Moment Estimation):
  - Combination of AdaGrad (Adaptive Gradient Algorithm) and RMSProp
  - Running averages of both the gradients and the second moments of the gradients are used

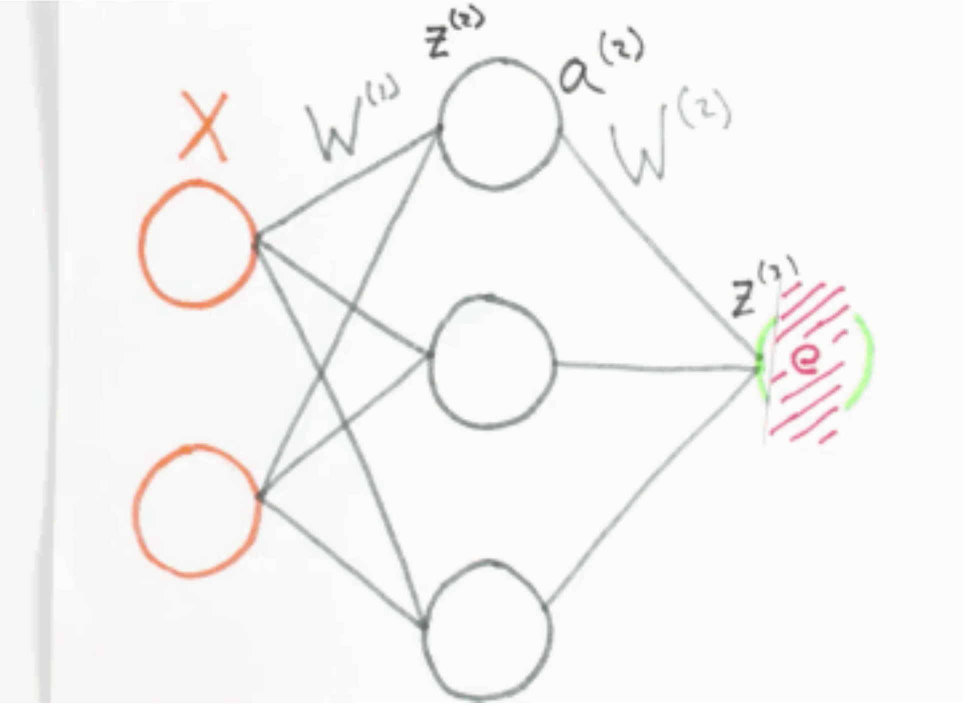
```
class torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)    [source]
```

```
optimizer = optim.Adam(policy_net.parameters())
```

Retrieved from https://pytorch.org/docs/stable/optim.html

- RMSProp (Root Mean Square Propagation):
  - Learning rate is adapted for each of the parameter
  - The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight

BACKPROPAGATION



Retrieved from http://www.cbcity.de/tag/neural-net

```
# clip the error and flip
clipped_error = -1.0 * error.clamp(-1, 1)
# Optimize the model
optimizer.zero_grad()
q_s_a.backward(clipped_error.data)

optimizer.step()
```

REPLAY MEMORY

```python
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward', 'done'))


class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

BASIC ALGORITHM

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Retrieved from https://arxiv.org/pdf/1312.5602v1.pdf

CODE

IMPROVEMENTS

- Dueling DQN
  - dueling network represents two separate estimators
- Double DQN
  - Normal Q-learning algorithm is known to overestimate action values
  - Two Q-functions
  - One function is used to determine the maximizing action and second to estimate its value

LESSONS LEARNED

- Research on the topic first
- Better data parallelization on the GPU

QUESTIONS?