Reinforcement Learning
Intelligent Systems Series
Lecture 2

Georg Martius

MPI for Intelligent Systems, Tübingen, Germany

November 2, 2018

Details on Exercises

- You need to come to the recitations (Übungen)
- You will mutually work through the exercises with your class-mates
- Tutor will clarify questions and work through solutions
- Hand it in at the end

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

Supervised learning: $\{x, y\}$ pairs to function $f(x) \mapsto y$

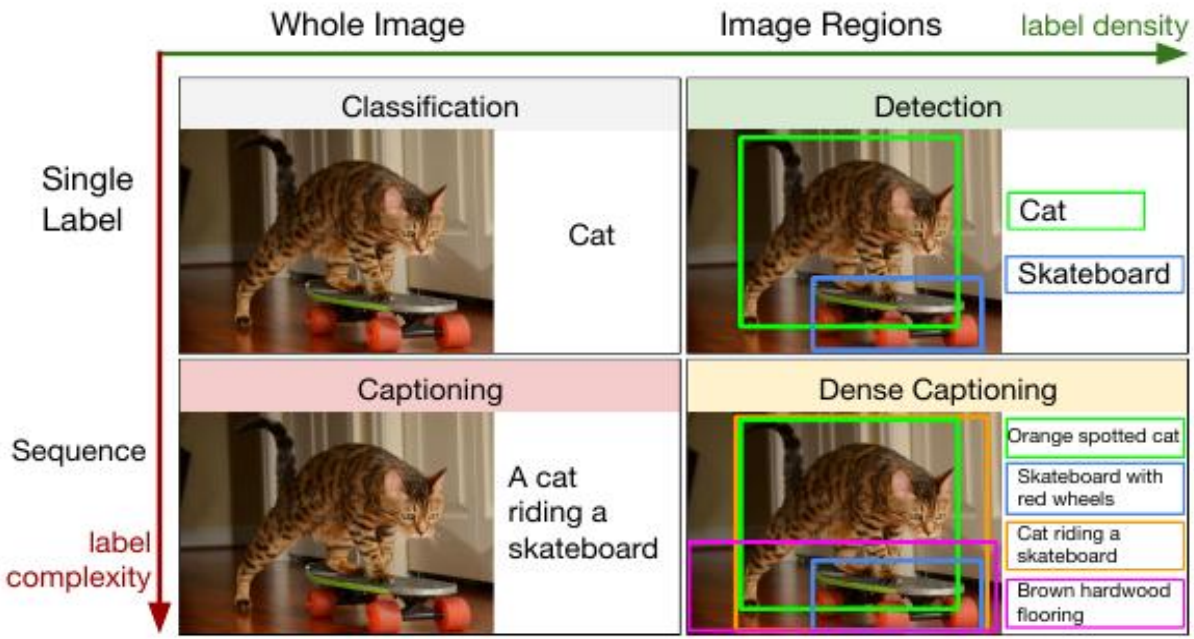
Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

Supervised learning: $\{x, y\}$ pairs to function $f(x) \mapsto y$

Example: images and captions creates automatic image captioning function



DenseCap, Johnson et al 2015

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

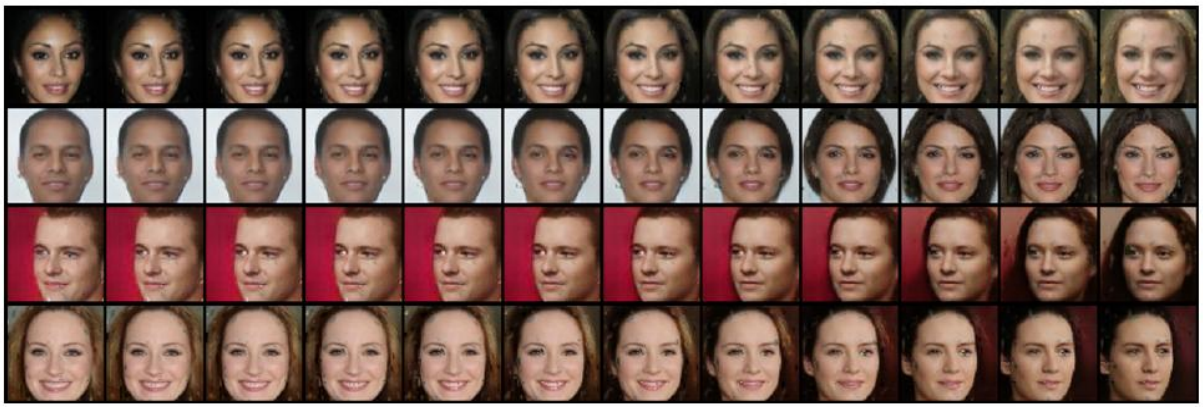
Unsupervised learning: $\{x\}$ to low-dimensional representation y: $f(x) \mapsto y$

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

Unsupervised learning: $\{x\}$ to low-dimensional representation y: $f(x) \mapsto y$
Example: Take images of faces and find few discriptive variables: can also generate new images



Khan et al 2018

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

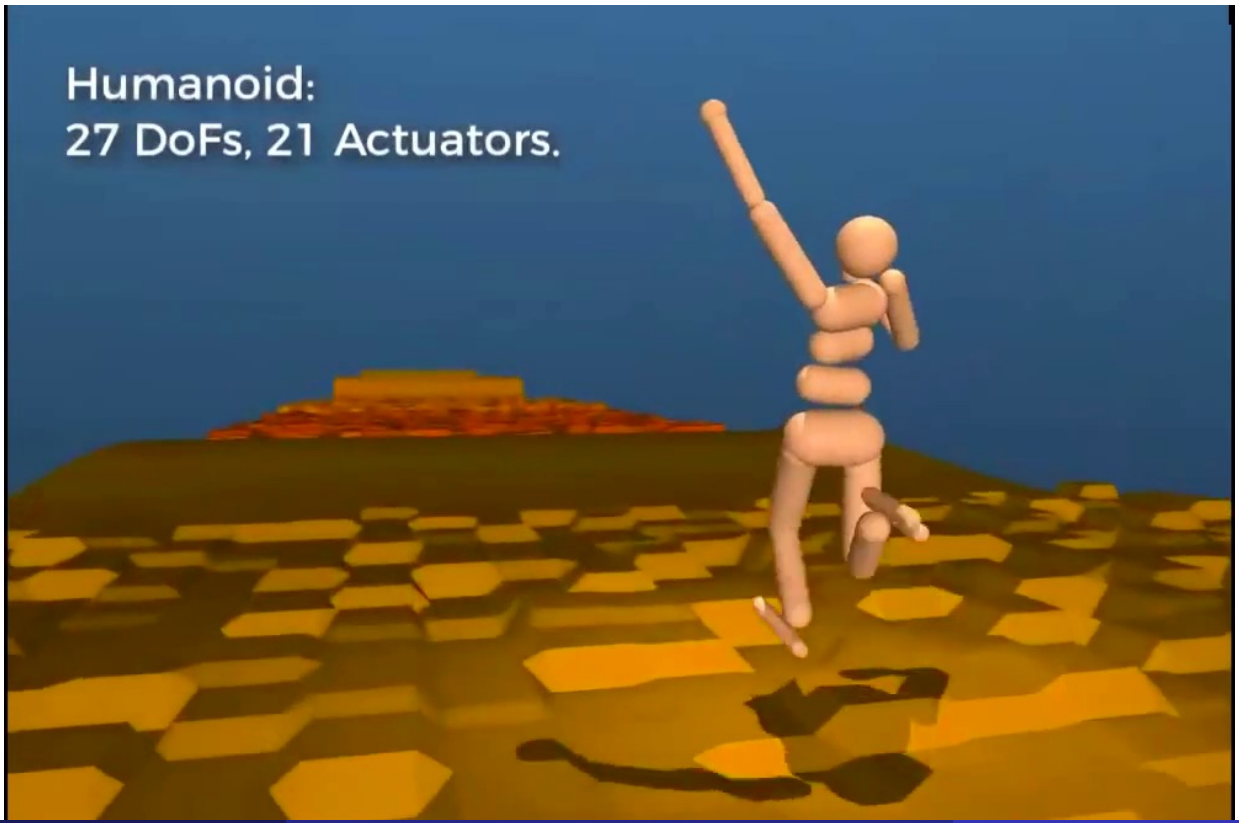
Reinforcement learning: System S: policy $\pi(s) \mapsto a$ (sensor to action)

Reminder

Machine learning has three branches:

- supervised learning
- unsupervised learning
- reinforcement learning

Reinforcement learning: System S: policy $\pi(s) \mapsto a$ (sensor to action)
Example: learning to locomote



Reminder

Linear Regression: $f(x) = w^\top x$

Reminder

Linear Regression: $f(x) = w^\top x$

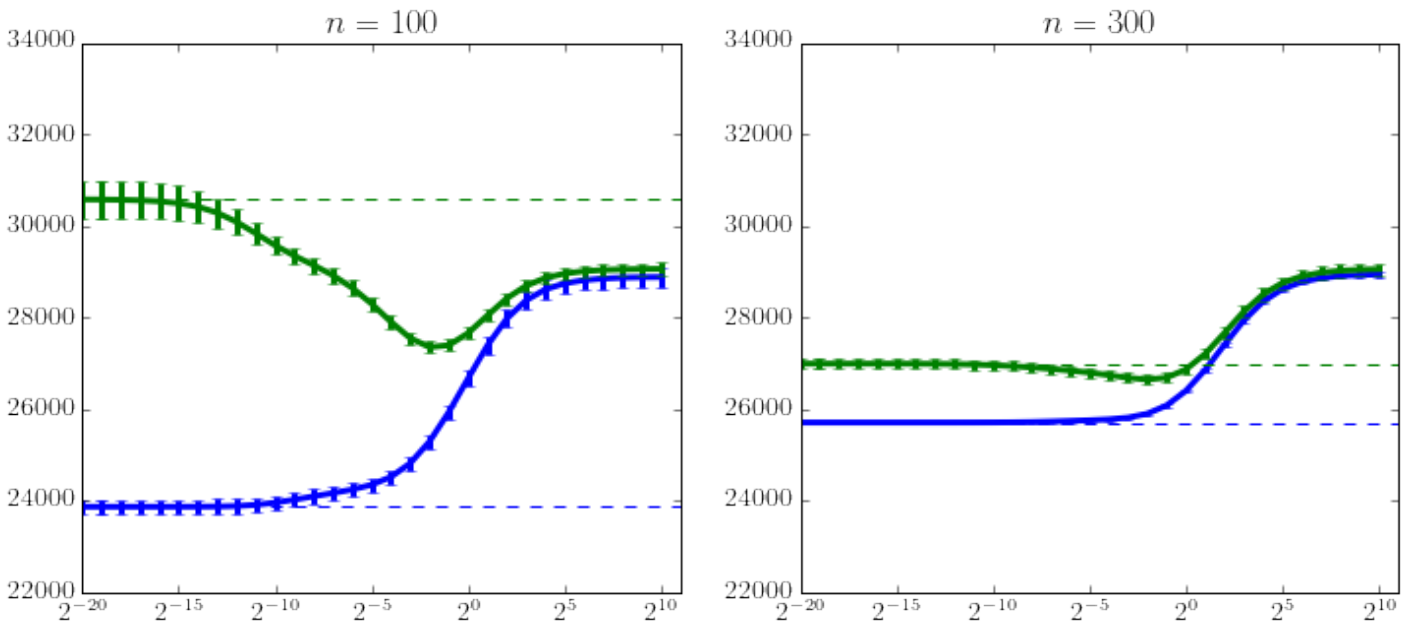
Cost function: Squared error: $\mathcal{L}(w) = \frac{1}{n} \sum_i (w^\top x_i - y_i)^2$

Reminder

Linear Regression: $f(x) = w^\top x$

Cost function: Squared error: $\mathcal{L}(w) = \frac{1}{n} \sum_i (w^\top x_i - y_i)^2$

To avoid overfitting:



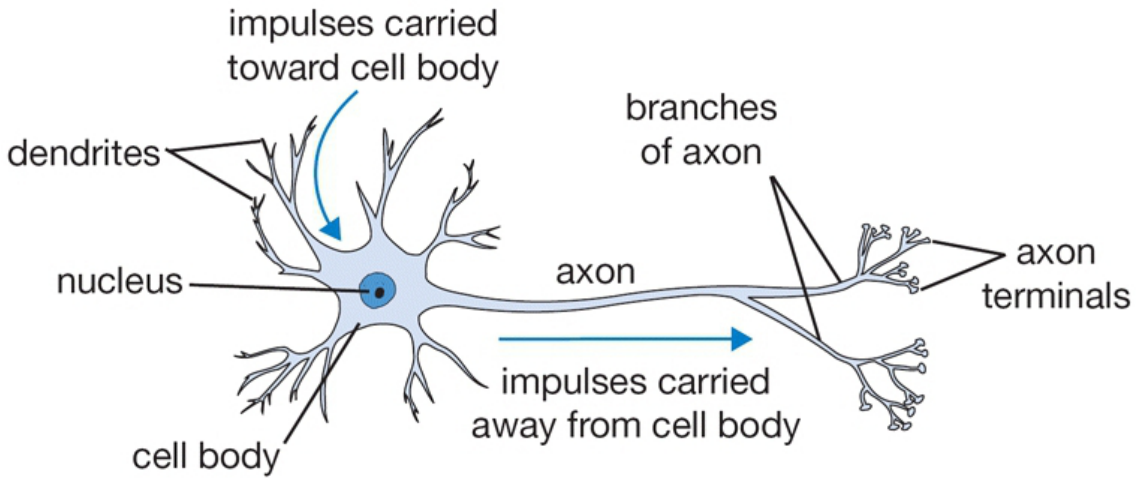
Regularization: $\mathcal{L}(w) = \frac{1}{n} \sum_i (w^\top x_i - y_i)^2 + \lambda \|w\|^2$

Today

- Supervised learning with Neural Networks
- Imitation learning for behavior generation

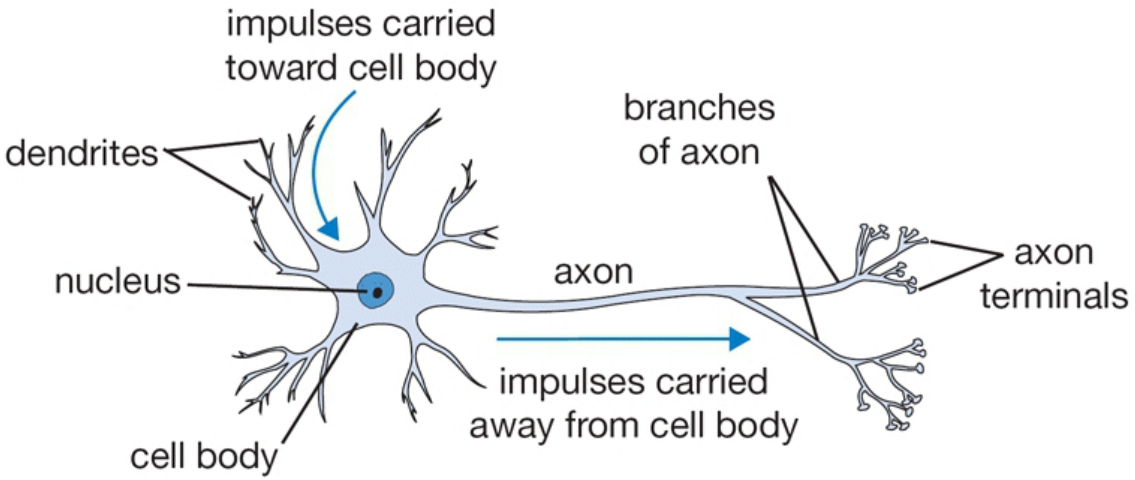
Artificial Neural Networks – a short introduction

Inspired by biological neurons:

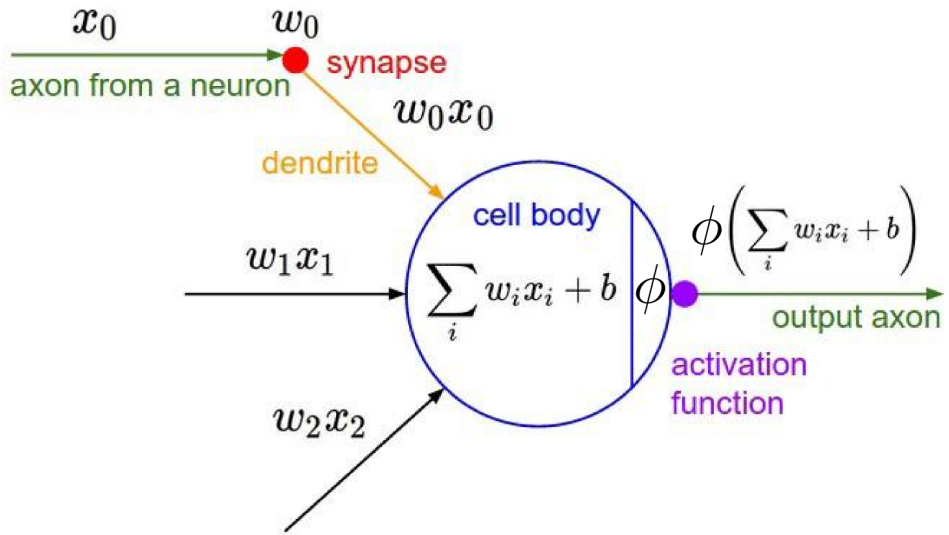


Artificial Neural Networks – a short introduction

Inspired by biological neurons:



Math model:

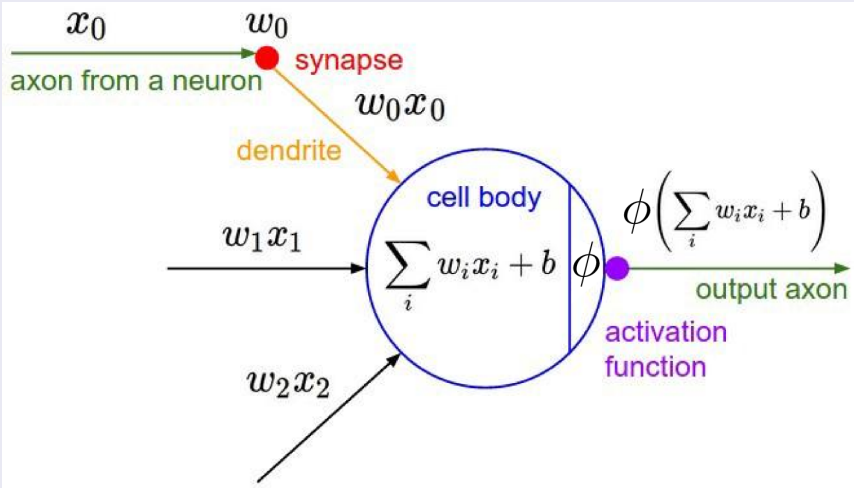


[hackernoon.com by D. Jeffries]

Artificial Neural Networks – a short introduction

Inspired by biological neurons, but extremely simplified:

Simple artificial Neuron

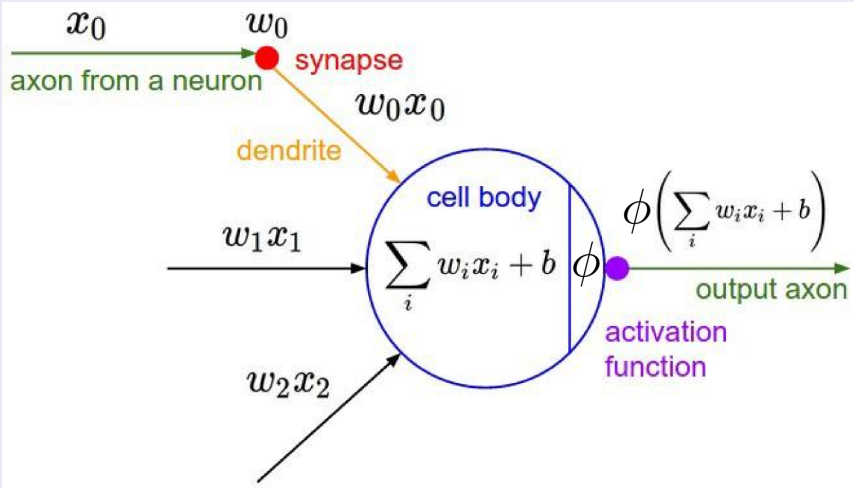


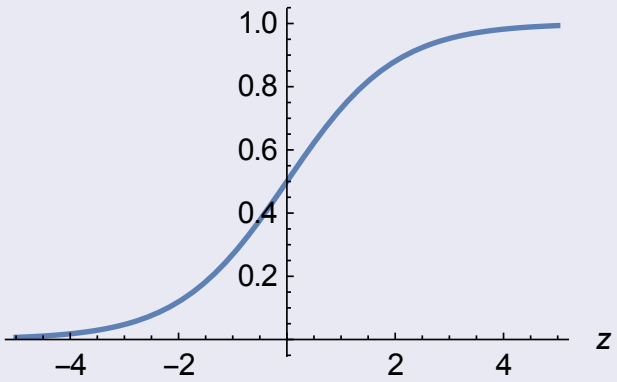
$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

Artificial Neural Networks – a short introduction

Inspired by biological neurons, but extremely simplified:

Simple artificial Neuron





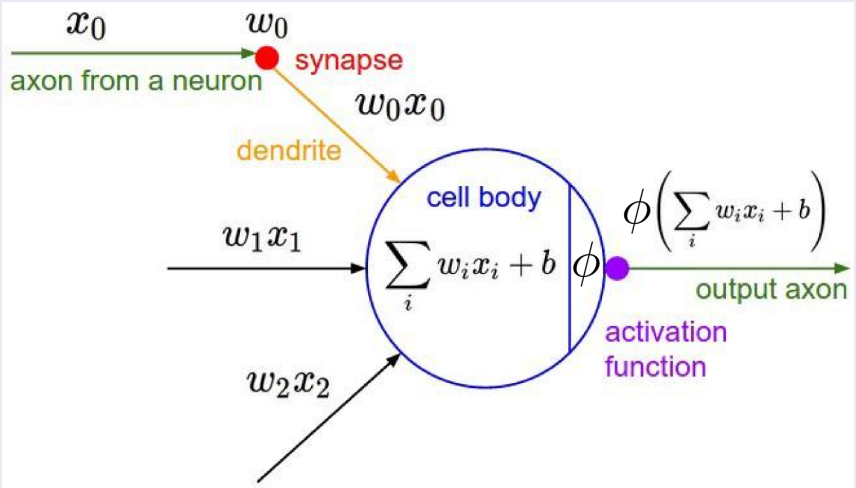
$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

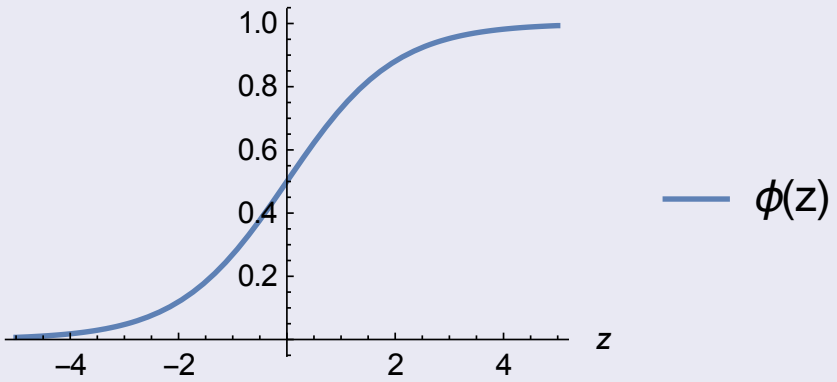
$$\phi(z) = \frac{1}{1 + e^{-z}} \qquad \text{sigmoid}$$

Artificial Neural Networks – a short introduction

Inspired by biological neurons, but extremely simplified:

Simple artificial Neuron





$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

$$\phi(z) = \frac{1}{1+e^{-z}} \qquad \text{sigmoid}$$

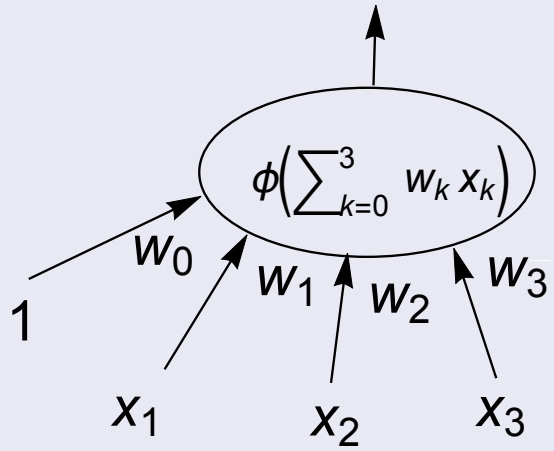
Like in regression problems:
can use squared error:

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

(plus regularization)

Delta Rule

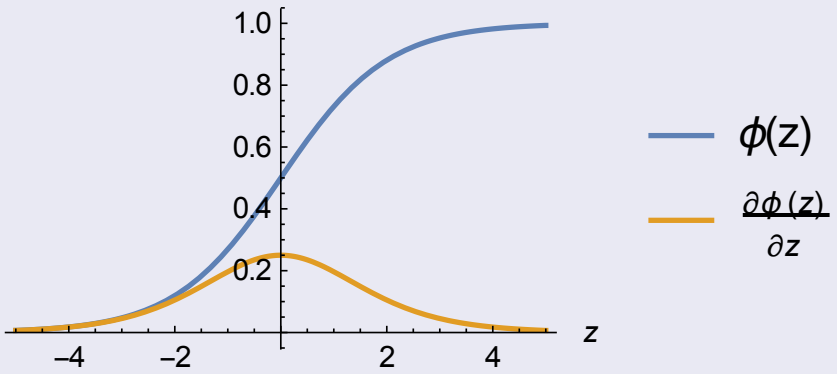
Perform gradient descent on $\mathcal{L}$: $w^t = w^{t-1} - \epsilon \frac{\partial \mathcal{L}(w)}{\partial w}$



$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

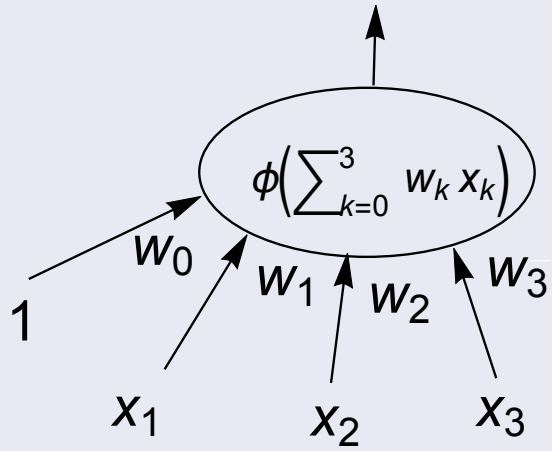
$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Sigmoid ϕ:



Delta Rule

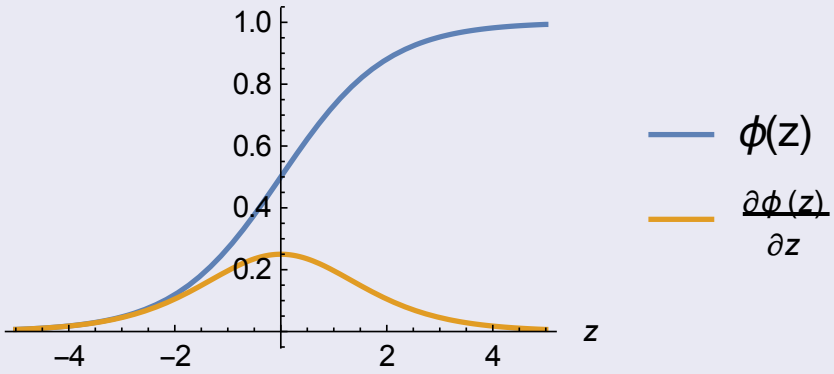
Perform gradient descent on $\mathcal{L}$: $w^t = w^{t-1} - \epsilon \frac{\partial \mathcal{L}(w)}{\partial w}$

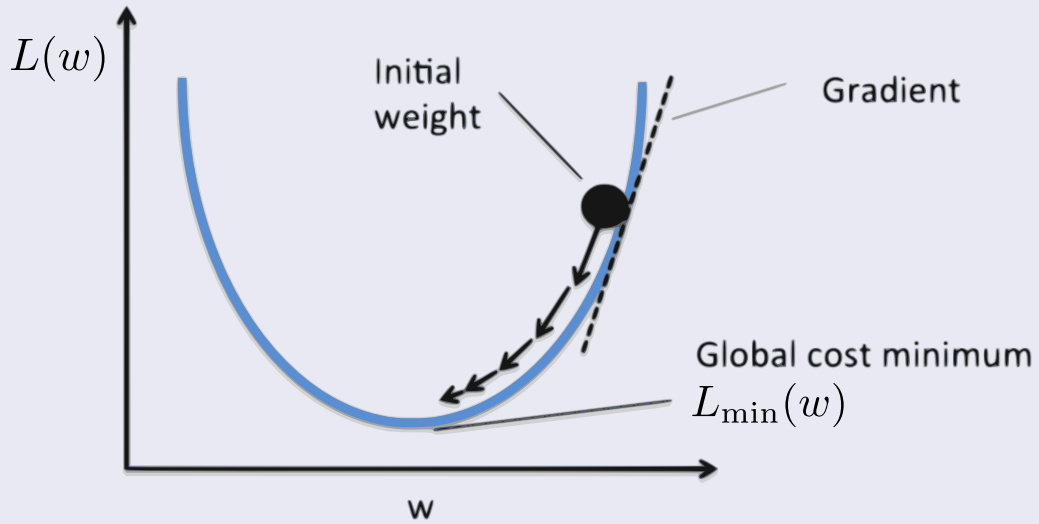


$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

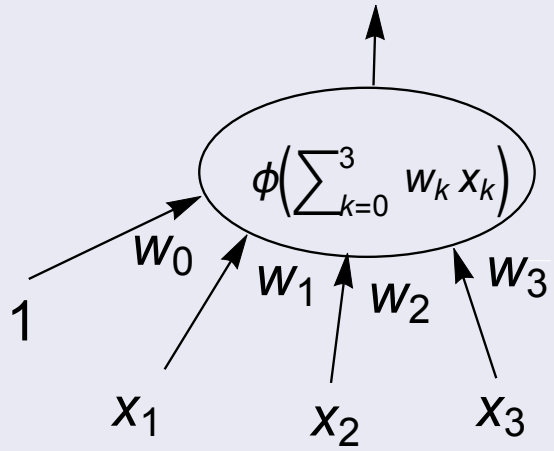
Sigmoid ϕ:





Delta Rule

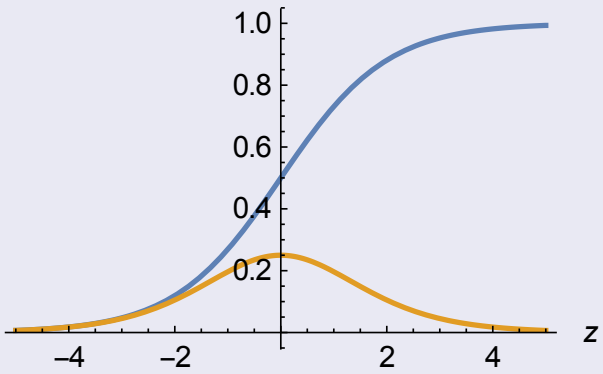
Perform gradient descent on $\mathcal{L}$: $w^t = w^{t-1} - \epsilon \frac{\partial \mathcal{L}(w)}{\partial w}$



$$\hat{y}_i = \phi\left(\sum_{j=1}^{d} w_{ij} x_j\right)$$

$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Sigmoid ϕ:



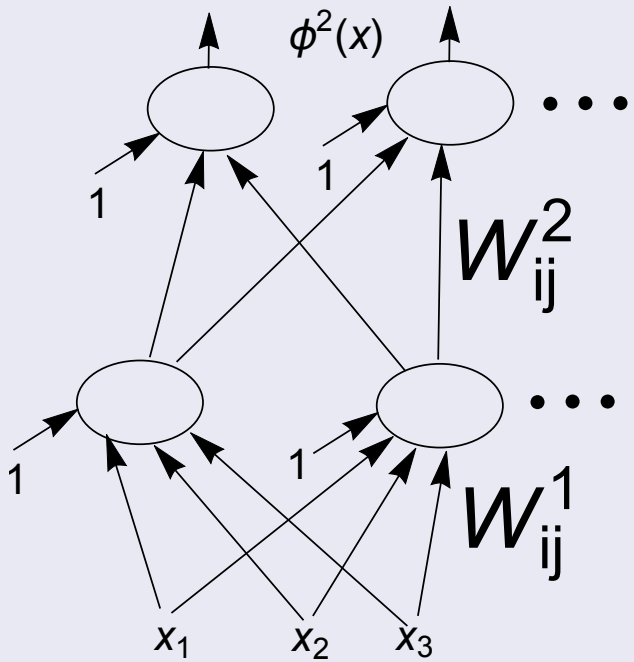
$$\Delta w = -\epsilon \frac{\partial \mathcal{L}(w)}{\partial w}$$

$$w := w + \Delta w$$

complete update equation: homework

Multilayer Networks – backpropagation

Stack layers of neurons on top of each other.

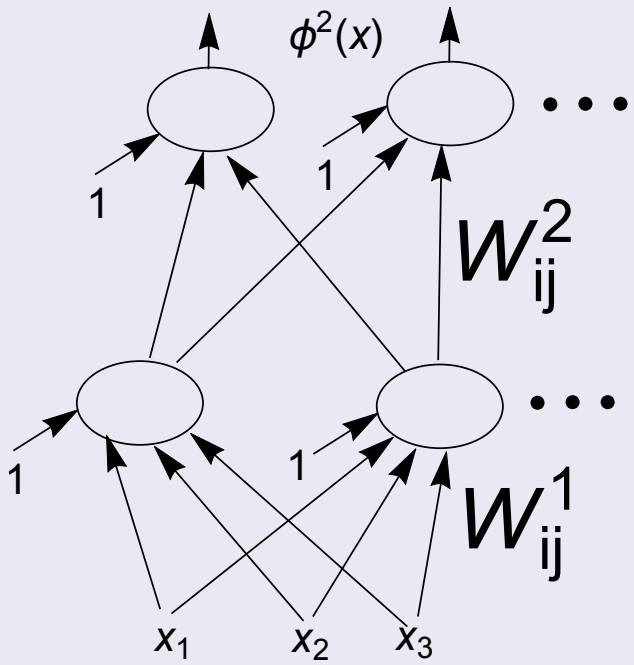


$$\hat{y} = \ldots \phi^2(W^2 \phi^1(W^1 x))$$

$$\mathcal{L}(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

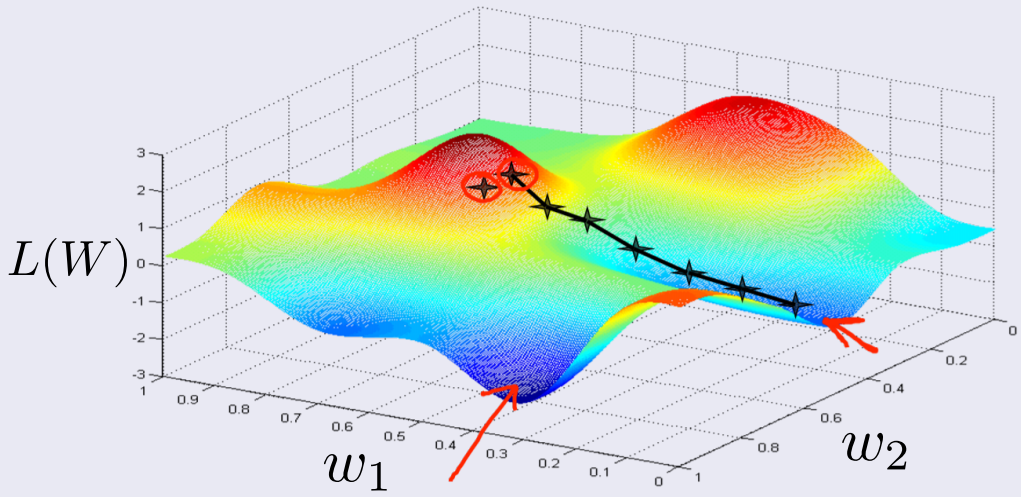
Multilayer Networks – backpropagation

Stack layers of neurons on top of each other.



$$\hat{y} = \ldots \phi^2(W^2 \phi^1(W^1 x))$$

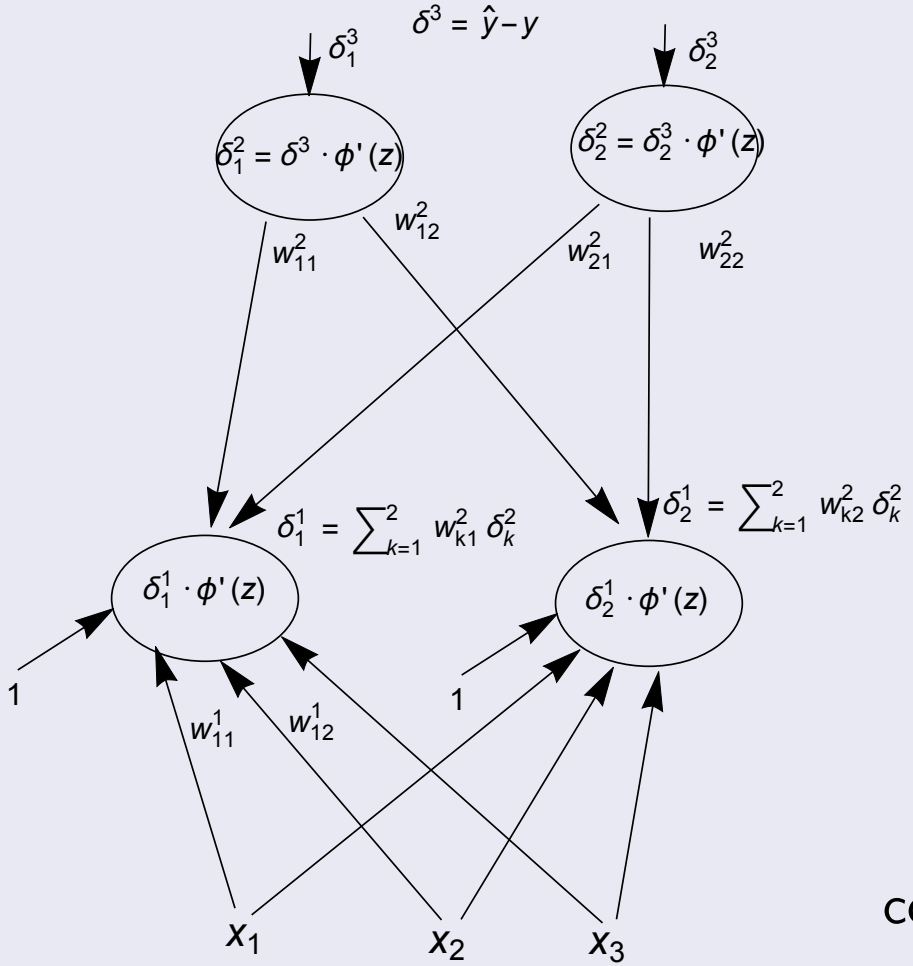
$$\mathcal{L}(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$



hackernoon.com

Multilayer Networks – backpropagation

Stack layers of neurons on top of each other.



$$\hat{y} = \ldots \phi^2(W^2 \phi^1(W^1 x))$$

$$\mathcal{L}(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$\Delta W^l = -\epsilon \frac{\partial \mathcal{L}(W)}{\partial W^l}$$

Backpropagation of the error signal:
$\delta^l = (W^{l+1})^\top \delta^{l+1}$
complete update equation: homework

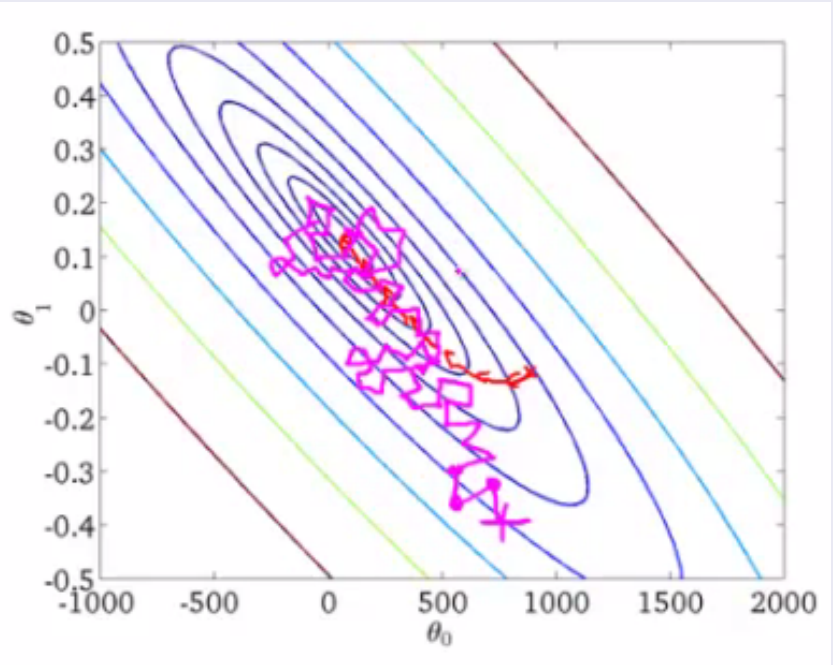
Stochastic gradient descent (SGD)

- Loss/Error is expected empirical error: sum over examples (batch)

- SGD: update parameters on every example:

$$\Delta W = -\epsilon \sum_{i}^{N} \frac{\partial(\hat{y}_i - y_i)^2}{\partial W}$$

- Minibatches: average gradient over a small # of examples



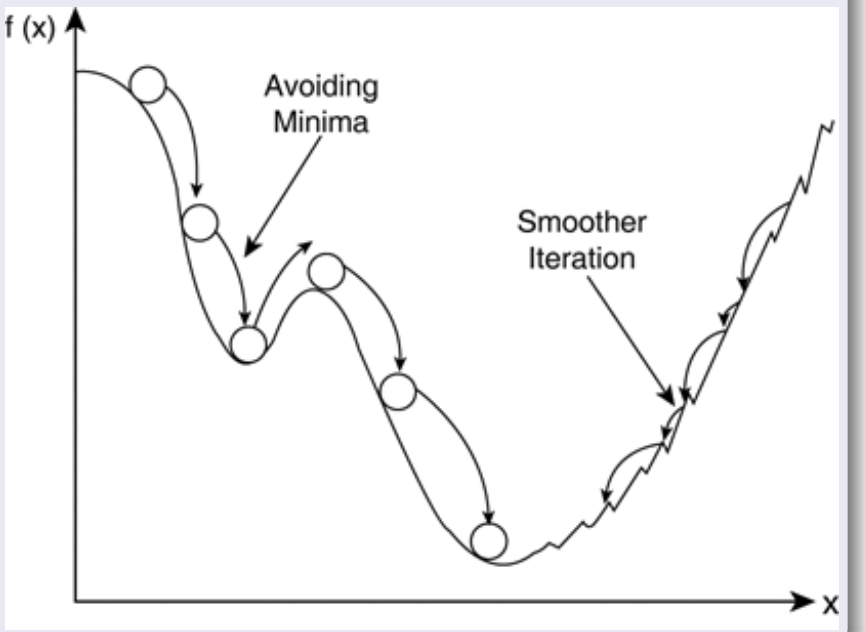
Advantages: many updates of parameters, noisier search helps to avoid flat regions

Momentum

Speed up gradient descent

- Momentum: add a virtual mass to the parameter-particle

$$\Delta W_t = -\epsilon \frac{\partial L(x_t)}{\partial W} + \alpha \Delta W_{t-1}$$

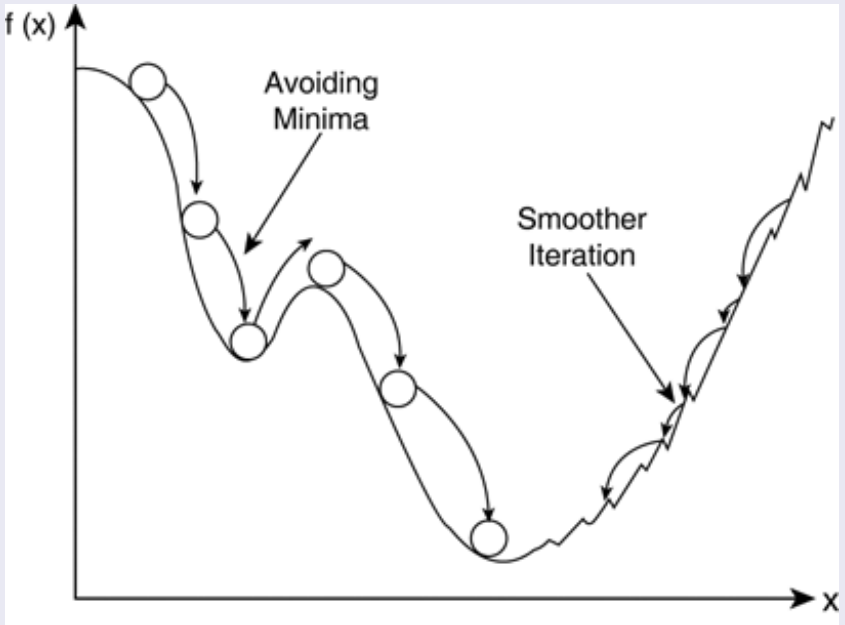


Momentum

Speed up gradient descent

- Momentum: add a virtual mass to the parameter-particle

$$\Delta W_t = -\epsilon \frac{\partial L(x_t)}{\partial W} + \alpha \Delta W_{t-1}$$



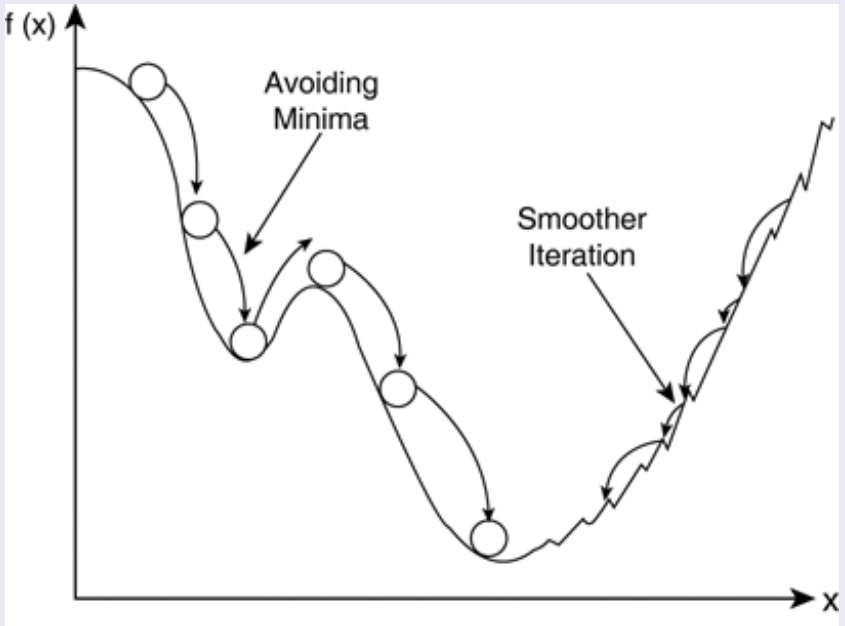
Advantages: may avoids some local minima, faster on ragged surfaces
Disadvantages: another hyperparameter, may overshoot

Momentum

Speed up gradient descent

- Momentum: add a virtual mass to the parameter-particle

$$\Delta W_t = -\epsilon \frac{\partial L(x_t)}{\partial W} + \alpha \Delta W_{t-1}$$



Advantages: may avoids some local minima, faster on ragged surfaces
Disadvantages: another hyperparameter, may overshoot

Adam (2014)

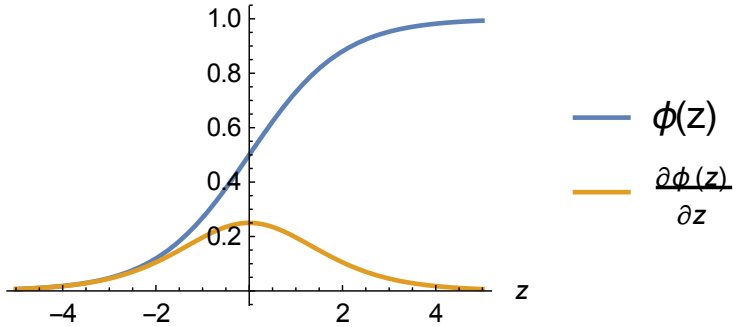
Rescale gradient for each parameter to unit size:

$$W_t = W_{t-1} - \epsilon \frac{\langle \nabla W \rangle_{\beta_1}}{\sqrt{\langle (\nabla W)^2 \rangle_{\beta_2}} + \lambda}$$ with moving averages: $\langle \cdot \rangle_\beta$

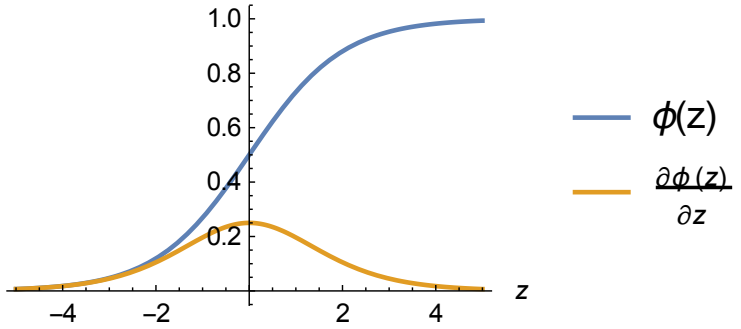
Artificial Neural Networks – a short introduction

Training: old and new tricks

- Derivative of sigmoid vanished for large absolute input (saturation)
- For deep networks (many layers)
  - gradient vanishes



- Derivative of sigmoid vanished for large absolute input (saturation)
- For deep networks (many layers)
  - ➪ gradient vanishes



ReLU

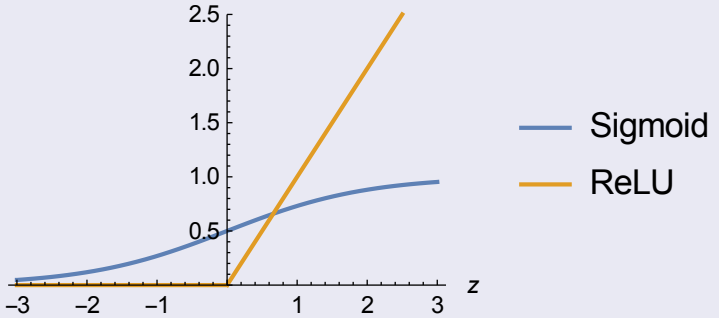
Use a simpler non-linearity:

$$\phi(z) = \max(0, z)$$

CRelu: concatenate positive and negative

$$\phi(z) = (\max(0, z), -\max(0, -z))$$

Unit-derivative everywhere

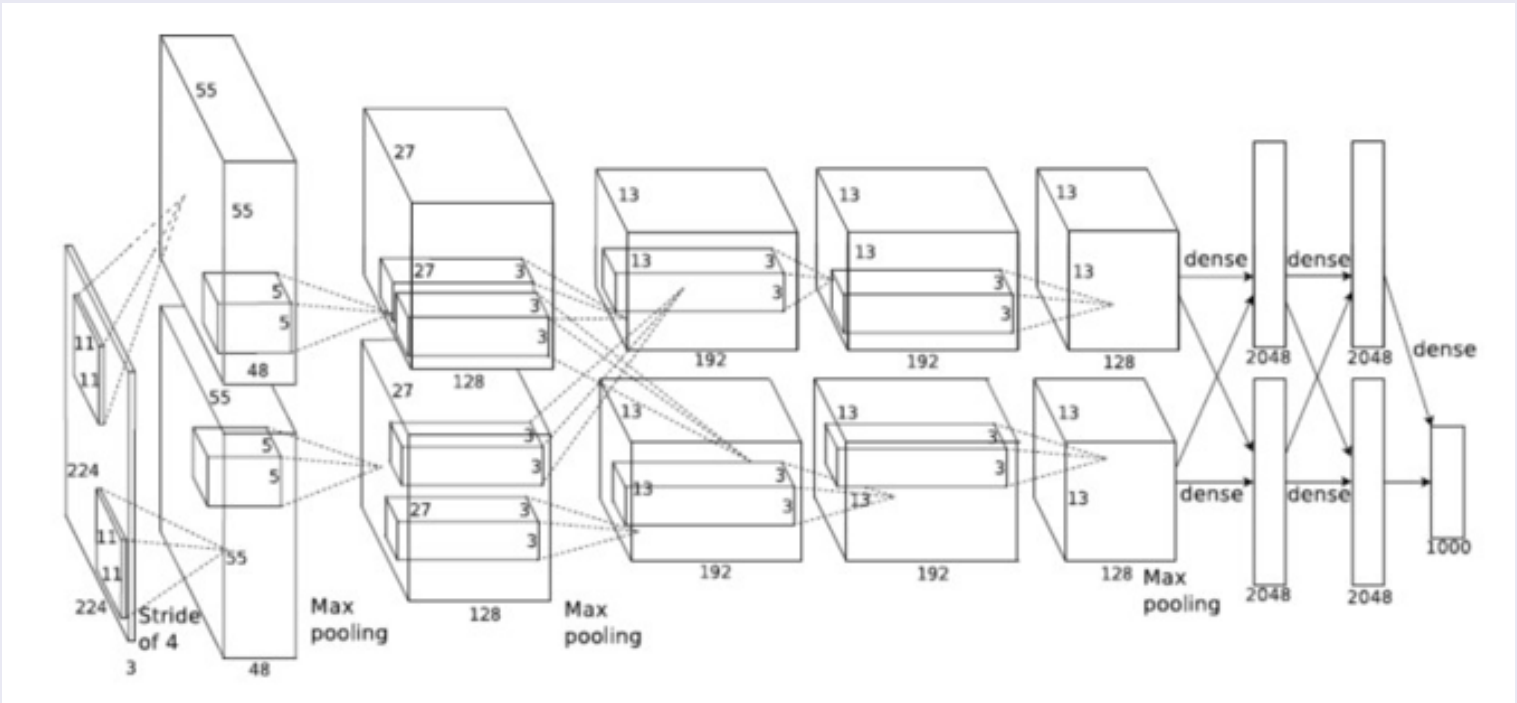


Try: `http://playground.tensorflow.org/`

Artificial Neural Networks – a short introduction

- Trainability and more computer power
  - ➡ larger and deeper networks (¿6 layers)
- Breakthrough in performance in many ML applications
  Vision, NLP, Speech,...

Convolutionary Network (CNN) – for vision

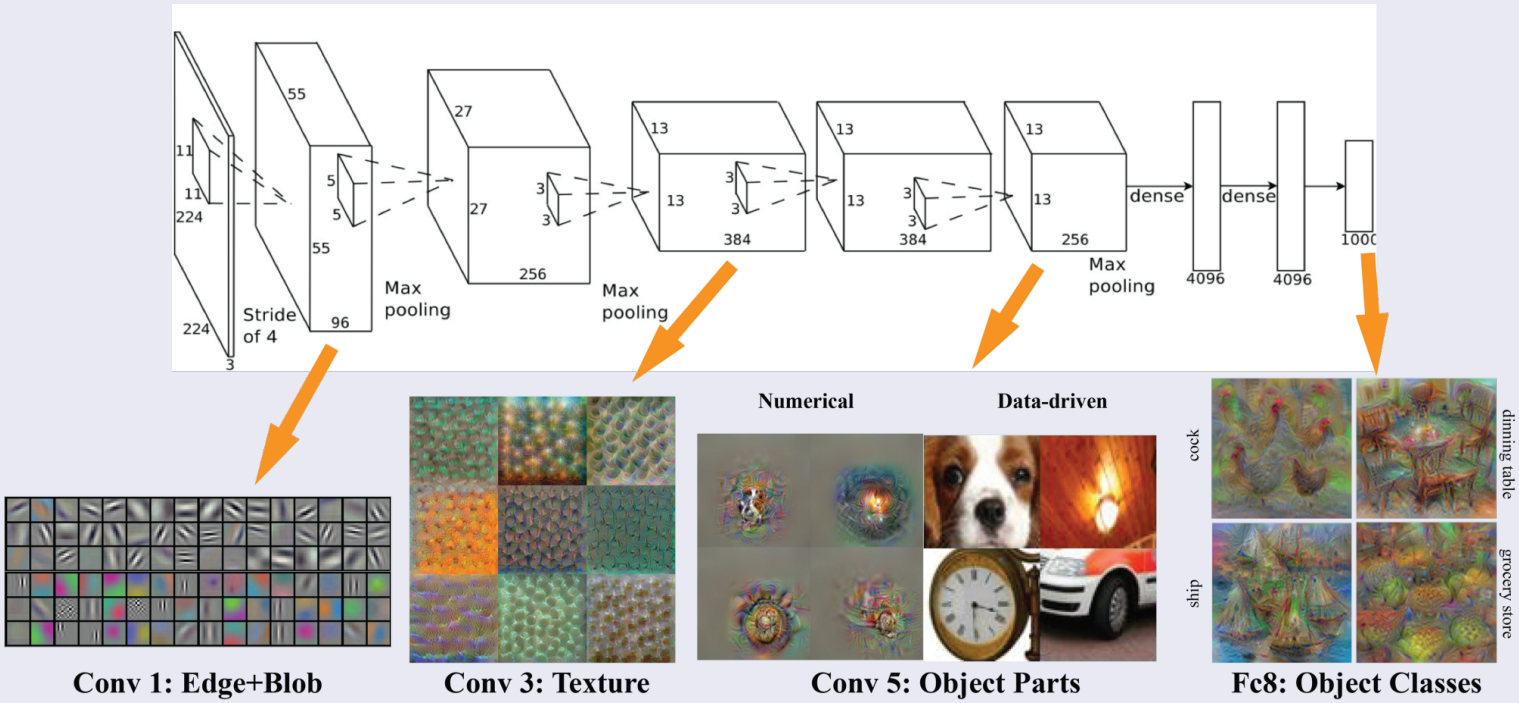


[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

Artificial Neural Networks – a short introduction

- Trainability and more computer power
  - ➡ larger and deeper networks (¿6 layers)
- Breakthrough in performance in many ML applications
  Vision, NLP, Speech,. . .

Convolutionary Network (CNN) – for vision



[http://vision03.csail.mit.edu/cnn_art/data/single_layer.png]

Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Solution: Let the computer do the work for us!

Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Solution: Let the computer do the work for us! Given code:

```
y = tanh( dot(W,x) + b)
```

Treat as a symbolic definition, not an imperative command.

Automatic differentiation (AutoDiv)

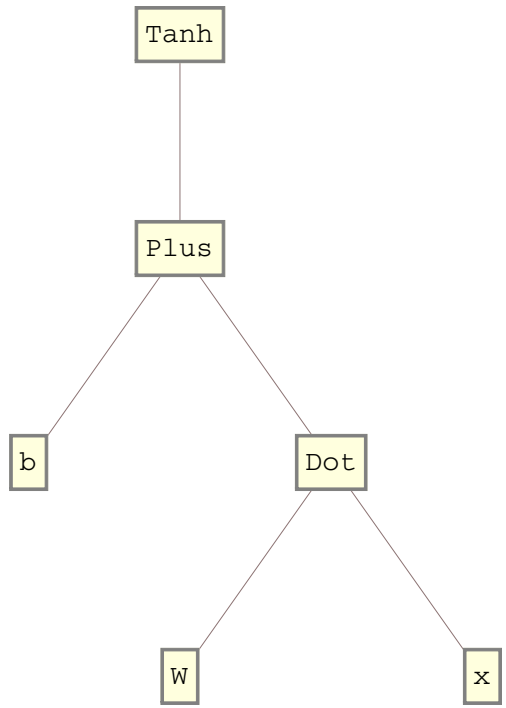
- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Solution: Let the computer do the work for us! Given code:

```
y = tanh( dot(W,x) + b)
```

Treat as a symbolic definition, not an imperative command.

Create computational graph:



Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

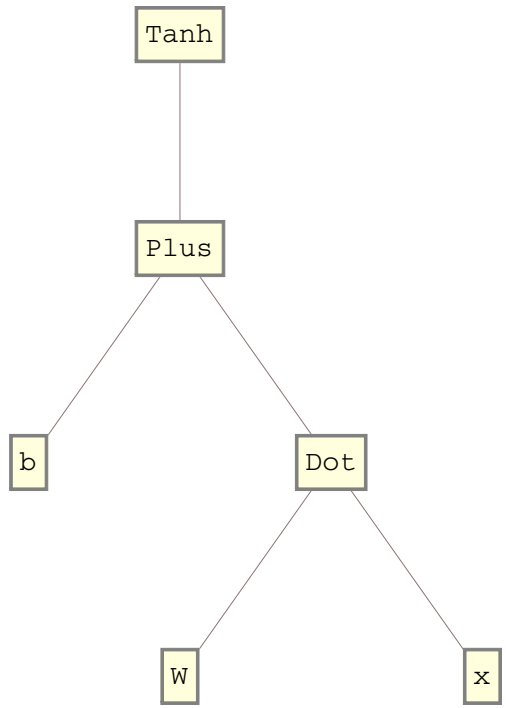
Solution: Let the computer do the work for us! Given code:

```
y = tanh( dot(W,x) + b)
```

Treat as a symbolic definition, not an imperative command.

Create computational graph:

- generate function to evaluate expression



Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

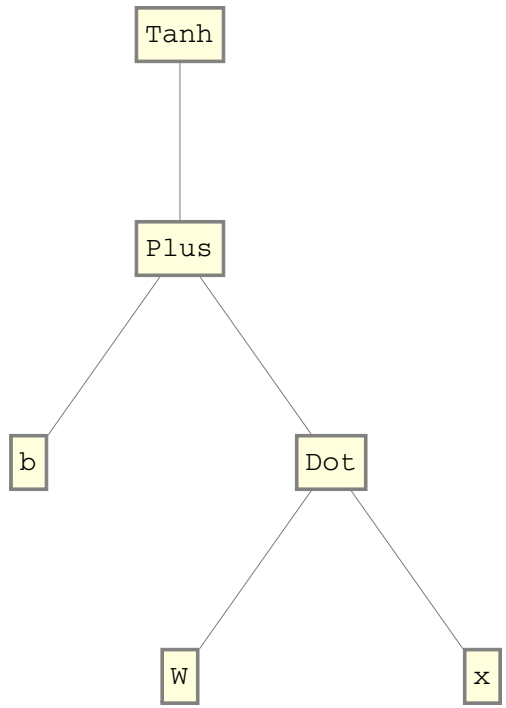
Solution: Let the computer do the work for us! Given code:

```
y = tanh( dot(W,x) + b)
```

Treat as a symbolic definition, not an imperative command.

Create computational graph:

- generate function to evaluate expression
- generate function for derivative (know deriv. of individual terms, and apply chain rule etc)



Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Solution: Let the computer do the work for us! Given code:

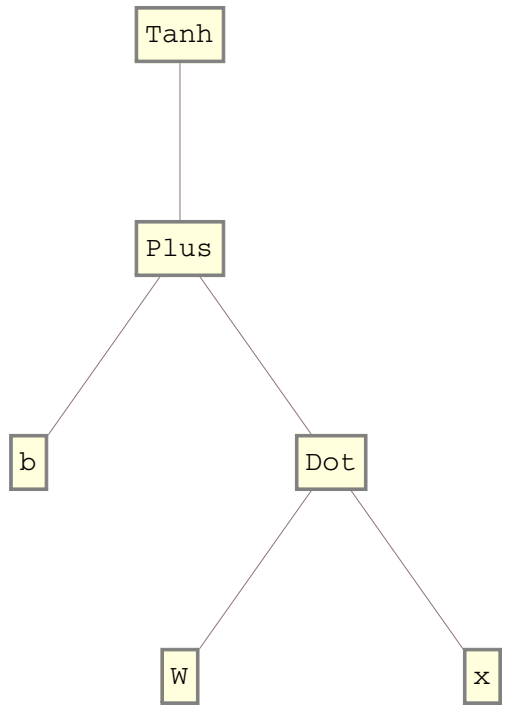
```
y = tanh( dot(W,x) + b)
```

Treat as a symbolic definition, not an imperative command.

Create computational graph:

- generate function to evaluate expression
- generate function for derivative (know deriv. of individual terms, and apply chain rule etc)

Common frameworks: Tensorflow, PyTorch, Theano (Python) but exist also for Matlab, C++ etc.

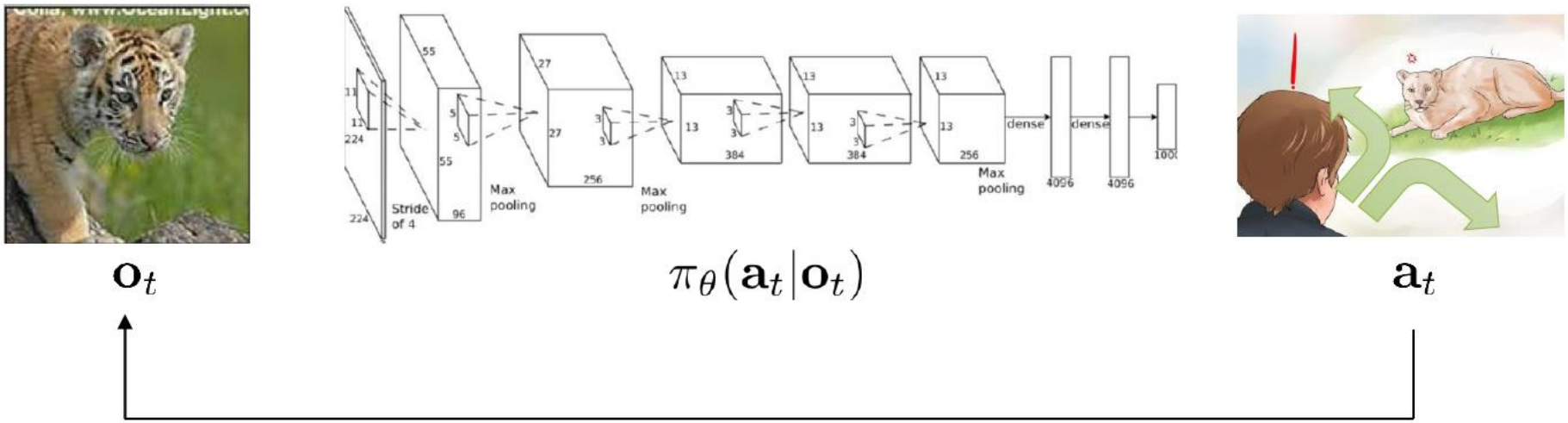


Break

Imitation learning: Supervised learning for decision making

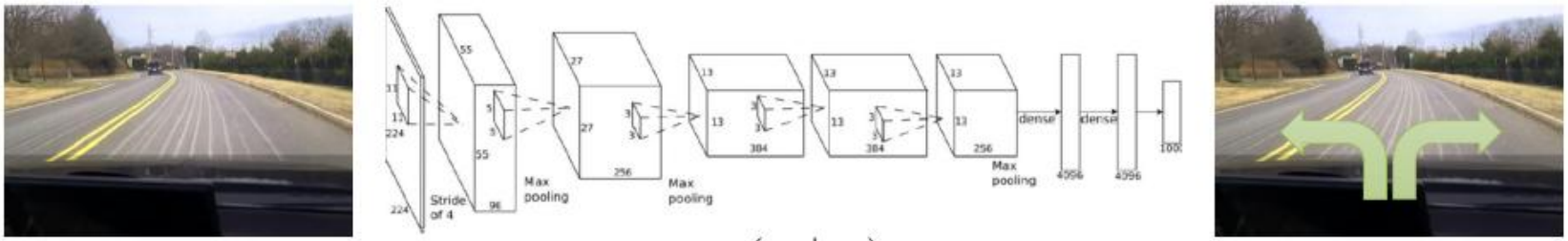
Setting

Sequential decision making: state s_t ➡ observation o_t ➡ action: $a_t \sim \pi(a|o_t)$



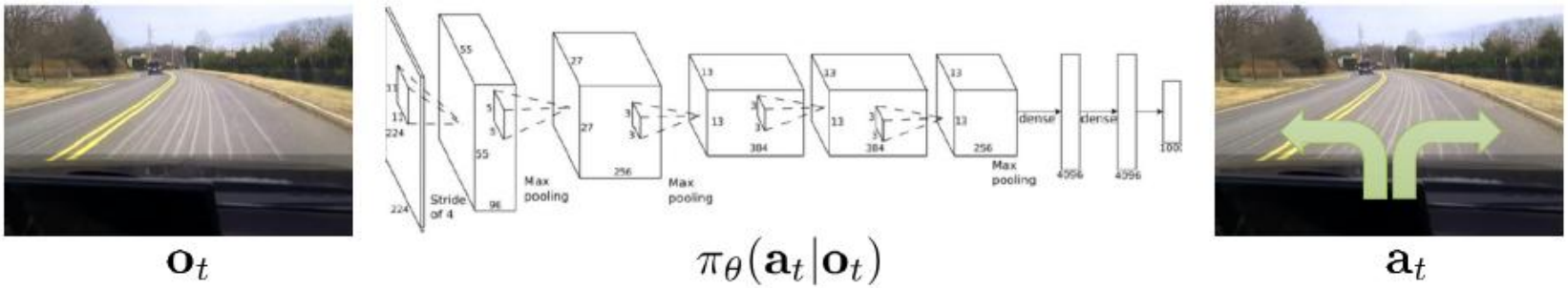
if $o_t = s_t$: fully observable

Imitation Learning



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

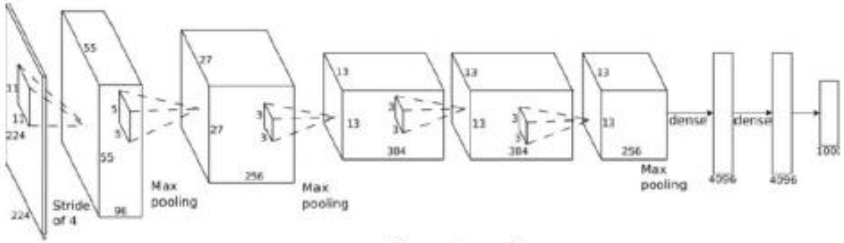
Imitation Learning



Behavioral Cloning: after collecting data of $\{o_t, a_t\}$ pairs: train policy function to "replicate behaviour"

How does it perform?

$\mathbf{o}_t$

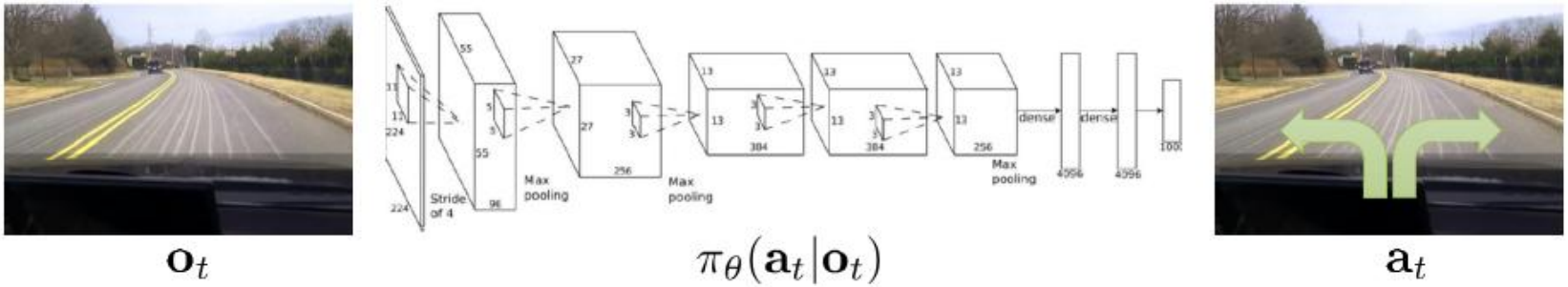


$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

[Video: Bojarski et al. 16, NVIDIA]

How does it perform?



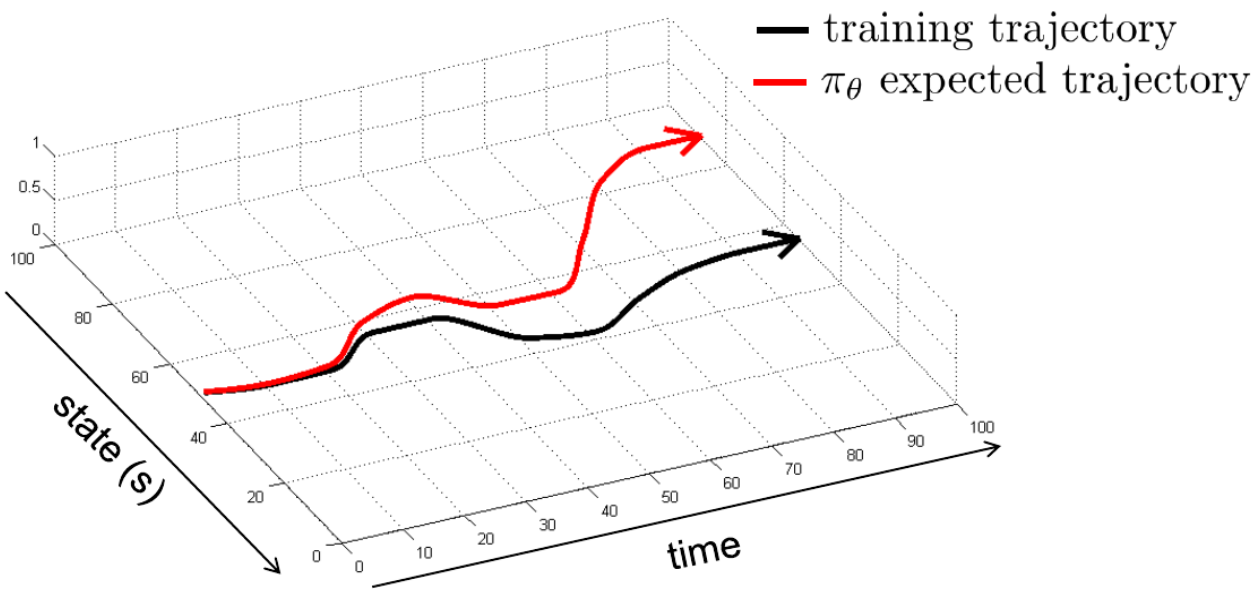
$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

[Video: Bojarski et al. 16, NVIDIA] badly!

Why does it not work?

Why does it not work?

Data collected during training deviates quickly from training-data



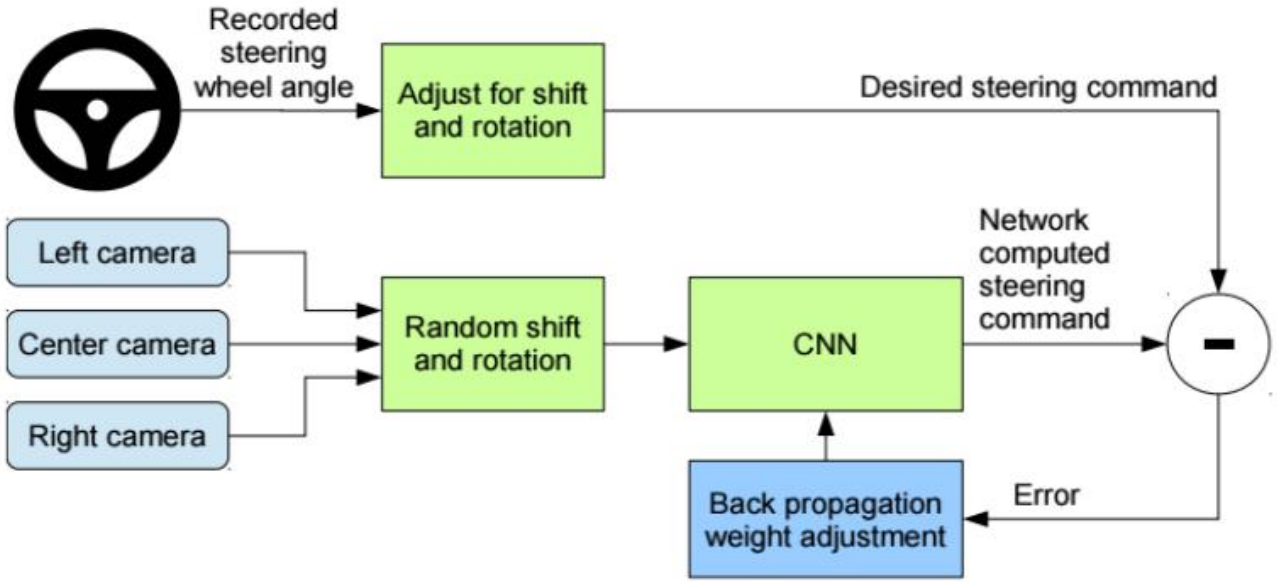
How about this?

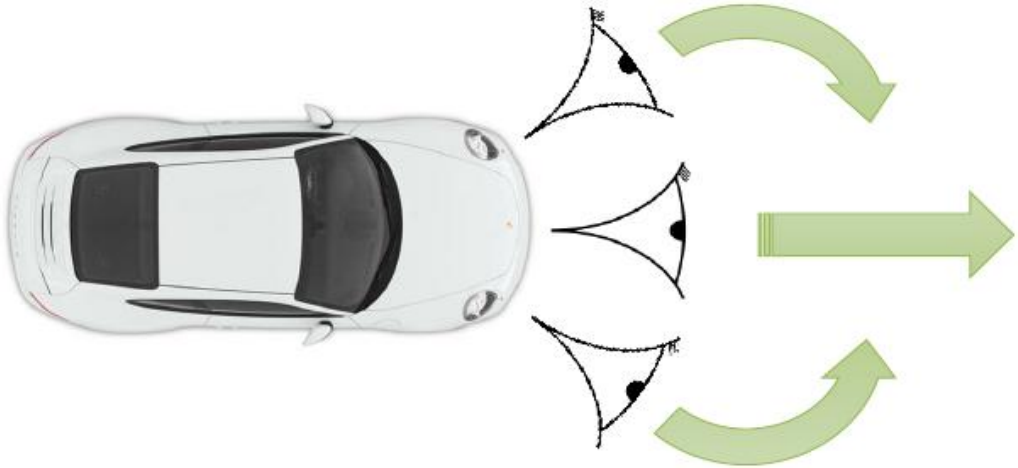
[Video: Bojarski et al. 16, NVIDIA]

How about this?

[Video: Bojarski et al. 16, NVIDIA]

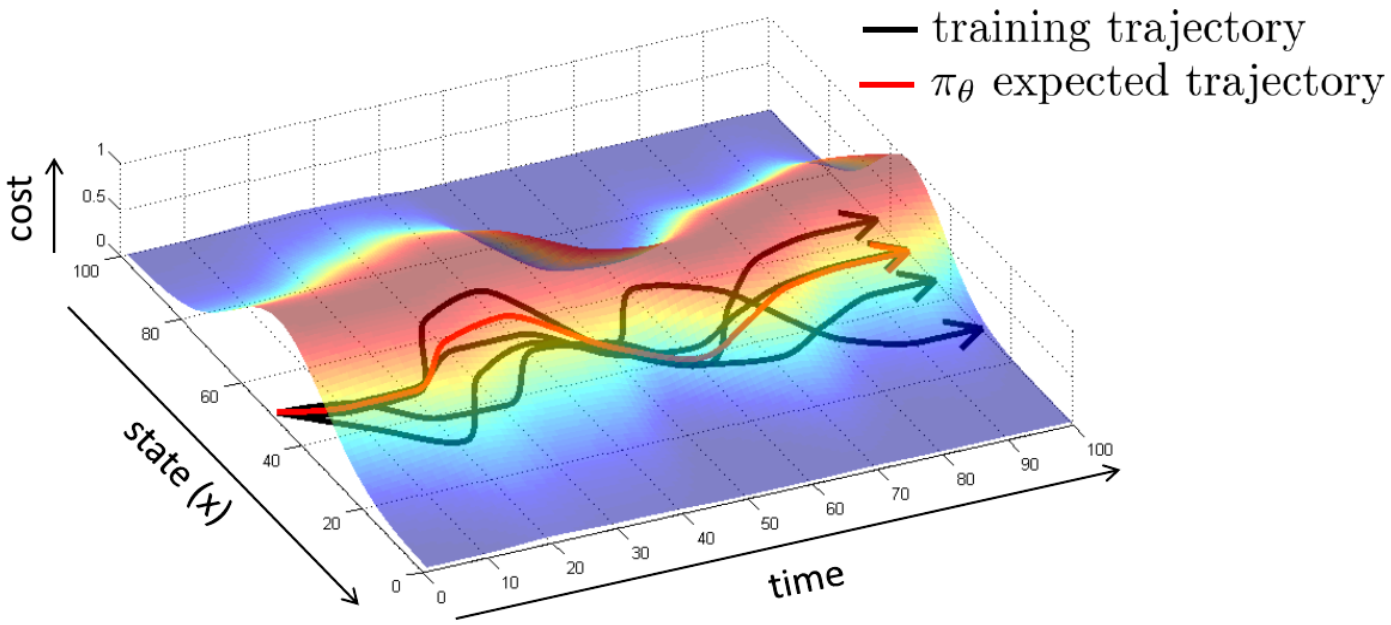
Why did that work?





side cameras, more data and data augentation

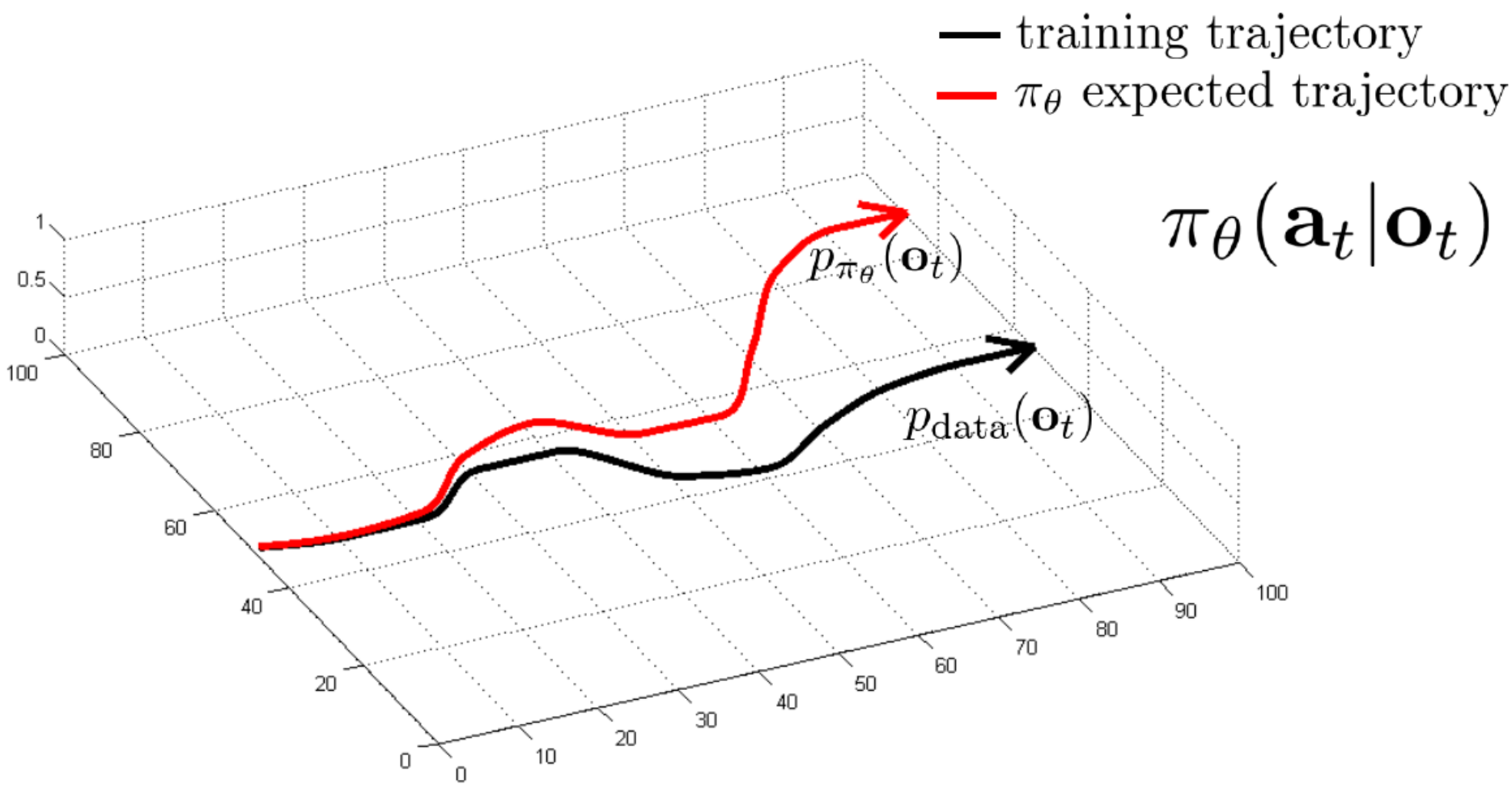
Making it work more often



Expected trajectory should be inside of training data.
Options:

- use some stabilizing controller
- collect data in a different manner

Making it work more often



can we make $p_{\text{data}}(\mathbf{o}_t) = p_{\pi_\theta}(\mathbf{o}_t)$?

make $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$

Dataset Aggretation

want: $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$
instead of improving $p_{\pi_\theta}(o_t)$, improve $p_{\text{data}}(o_t)$!

Dataset Aggretation

want: $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$
instead of improving $p_{\pi_\theta}(o_t)$, improve $p_{\text{data}}(o_t)$!
DAgger: Dataset Aggregation
Goal: collect training data from $p_{\pi_\theta}(o_t)$ (instead of $p_{\text{data}}(o_t)$)
How?

Dataset Aggretation

want: $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$
instead of improving $p_{\pi_\theta}(o_t)$, improve $p_{\text{data}}(o_t)$!
DAgger: Dataset Aggregation
Goal: collect training data from $p_{\pi_\theta}(o_t)$ (instead of $p_{\text{data}}(o_t)$)
How? We can run $\pi_\theta(a_t|o_t)$,

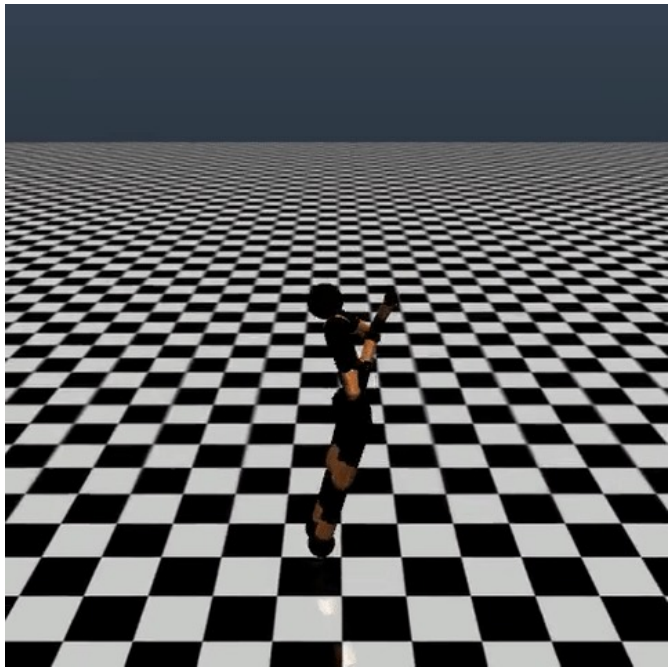
Dataset Aggretation

want: $p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t)$
instead of improving $p_{\pi_\theta}(o_t)$, improve $p_{\text{data}}(o_t)$!
DAgger: Dataset Aggregation
Goal: collect training data from $p_{\pi_\theta}(o_t)$ (instead of $p_{\text{data}}(o_t)$)
How? We can run $\pi_\theta(a_t|o_t)$,
but we need the correct actions? (labels)!
Algorithm:

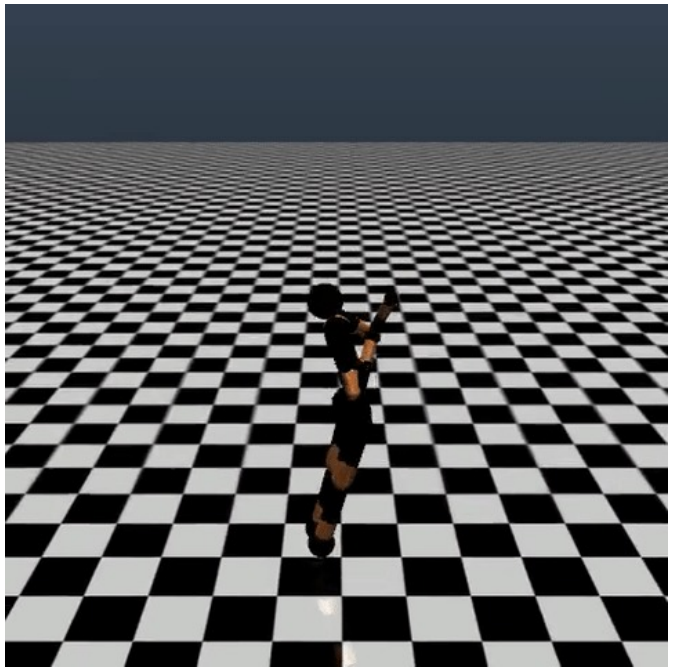
1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

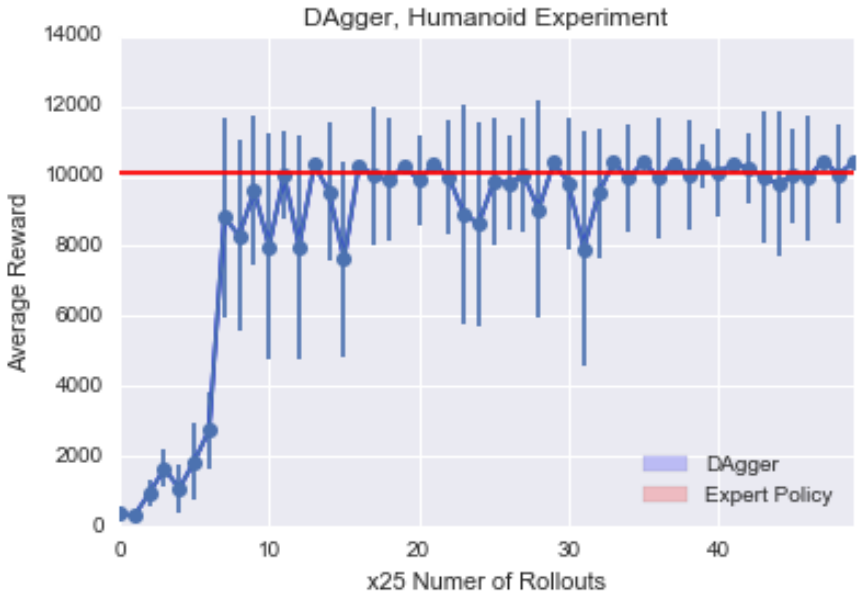
and iterate [Ross et al. '11]

Dataset Aggretation in action



Dataset Aggretation in action





Why is that not the ultimate solution?

Why is that not the ultimate solution?

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Why is that not the ultimate solution?

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

- Human is expensive and sometimes also does not know the right control command (see Humanoid)

Why is that not the ultimate solution?

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

- Human is expensive and sometimes also does not know the right control command (see Humanoid)
- If we have already a machine-expert what is the point?

Why is that not the ultimate solution?

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

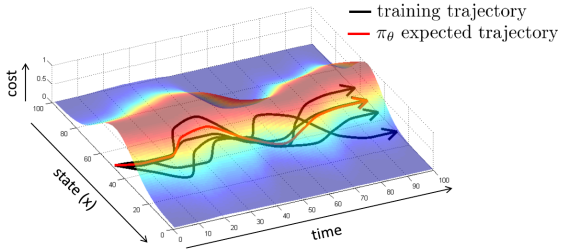
- Human is expensive and sometimes also does not know the right control command (see Humanoid)
- If we have already a machine-expert what is the point?
- ➡ distilling a faster policy

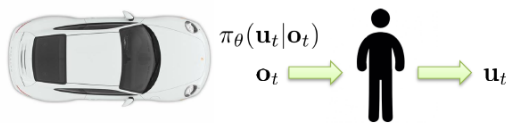
Summary: Imitation Learning with supervised learning from Humans

$\mathbf{o}_t$
$\mathbf{a}_t$

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

- Typically insufficient by itself
  - Distribution mismatch between training and test data
- It is simple and can sometimes work well
  - Hacks (left and right images)
  - Samples from a stable trajectory distribution
  - Add more on-policy data, (DAgger)





Can we do DAgger without human labeling?

Yes, but only if we know something about the system (can automatically label data).

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3.
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Can we do DAgger without human labeling?

Yes, but only if we know something about the system (can automatically label data).

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask human/expert to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Can we do DAgger without human labeling?

Yes, but only if we know something about the system (can automatically label data).

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Can we do DAgger without human labeling?

But there is still a problem:

Can we do DAgger without human labeling?

But there is still a problem:

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\pi_\theta(a_t|o_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$

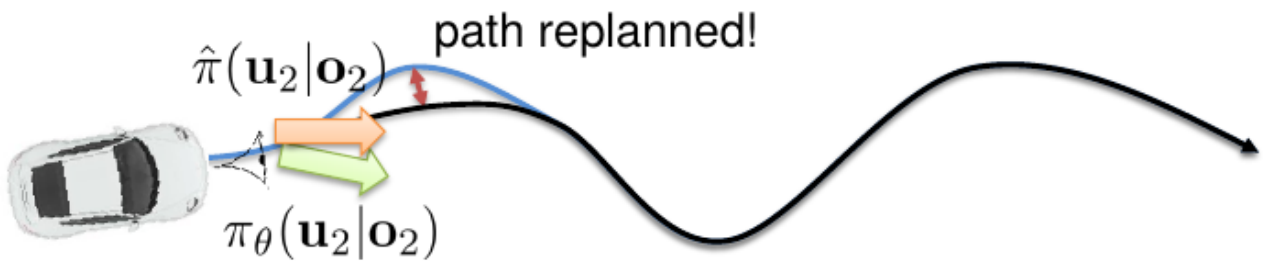
[Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$



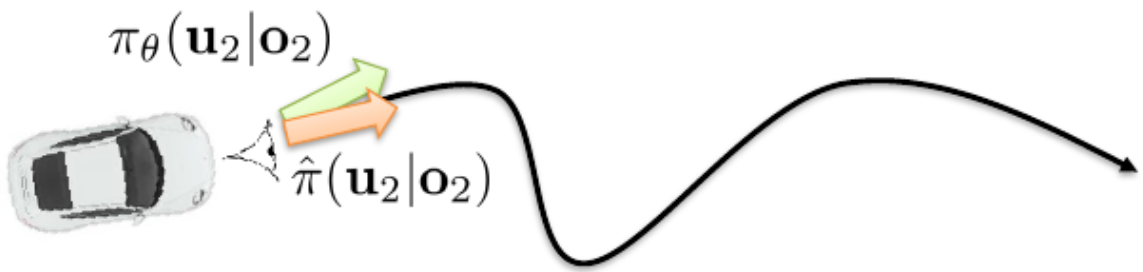
[Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\not{o}_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$



[Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$

replanning = Model predictive Control (later lecture)
Need to get the state s (not only the camera) while training.

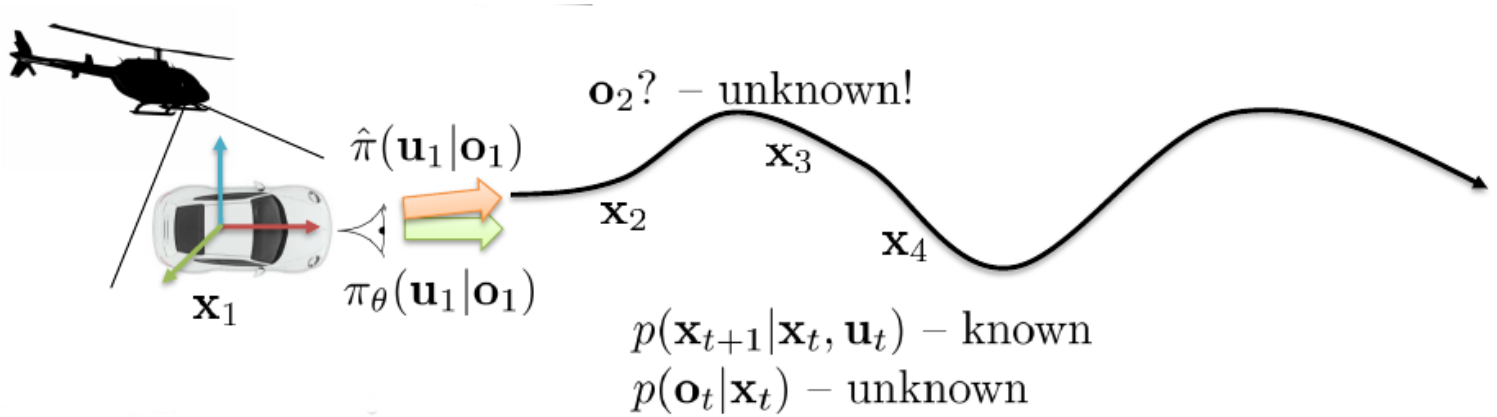
[Kahn, Zhang, Abbeel'16, Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$



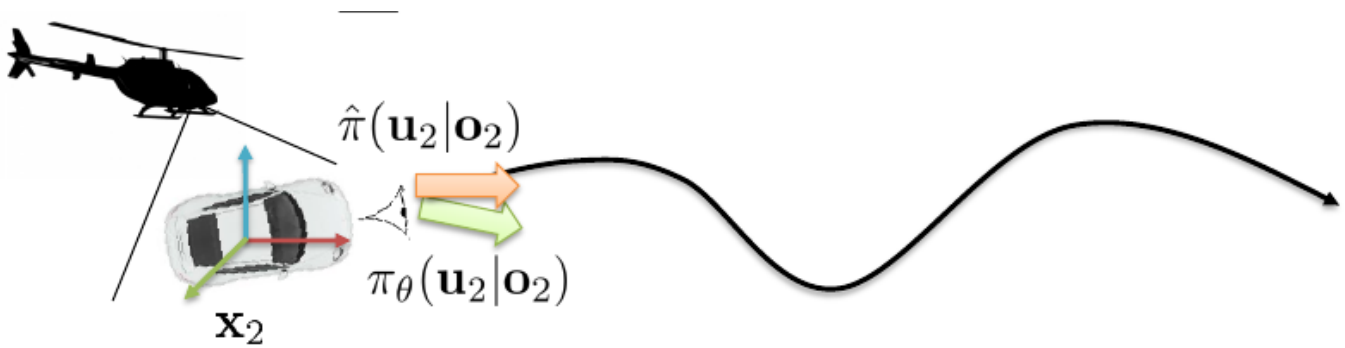
[Kahn, Zhang, Abbeel'16, Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

PLATO: Policy Learning with Adaptive Trajectory Optimization

1. train $\pi_\theta(a_t|o_t)$ from (human) data $\mathcal{D} = \{o_1, a_1, \ldots, o_N, a_N\}$
2. run $\hat{\pi}(a_t|\emptyset_t)$ and collect new observations $\mathcal{D}_\pi = o_1, \ldots, o_M$
3. Ask computer to provide actions a_t (labels) for $\mathcal{D}_\pi$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

$\hat{\pi}(a_t|s_t) = \mathcal{N} K_t s_t + k_t, \Sigma_{a_t}$ is a stochastic policy that can control robot

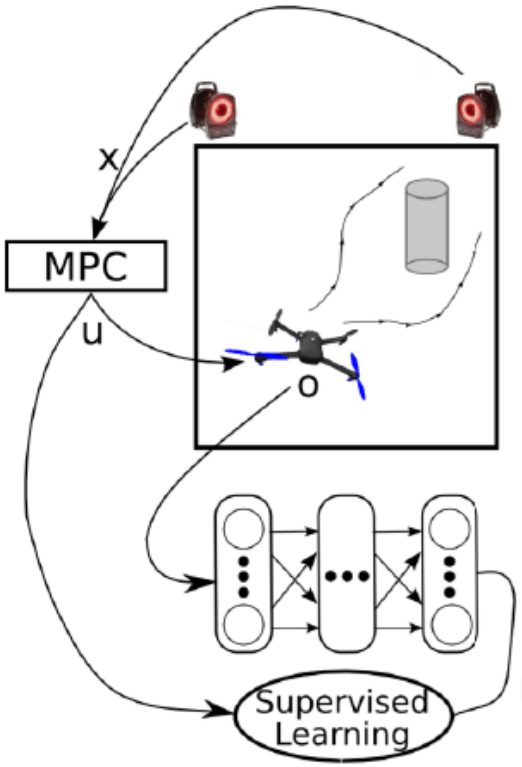
$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$

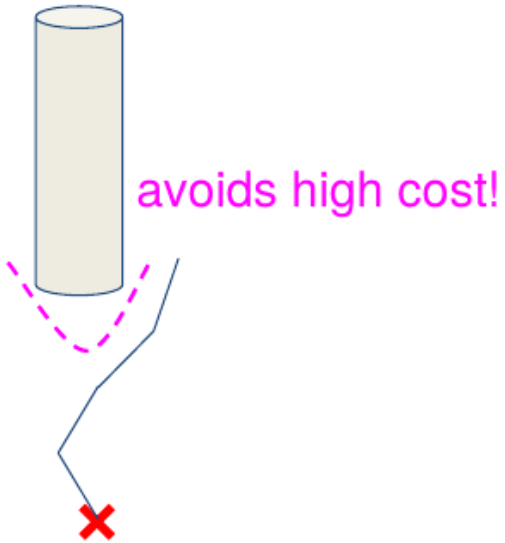


[Kahn, Zhang, Abbeel'16, Pictures from Sergey Levine. Notation: $x = s$ and $u = a$]

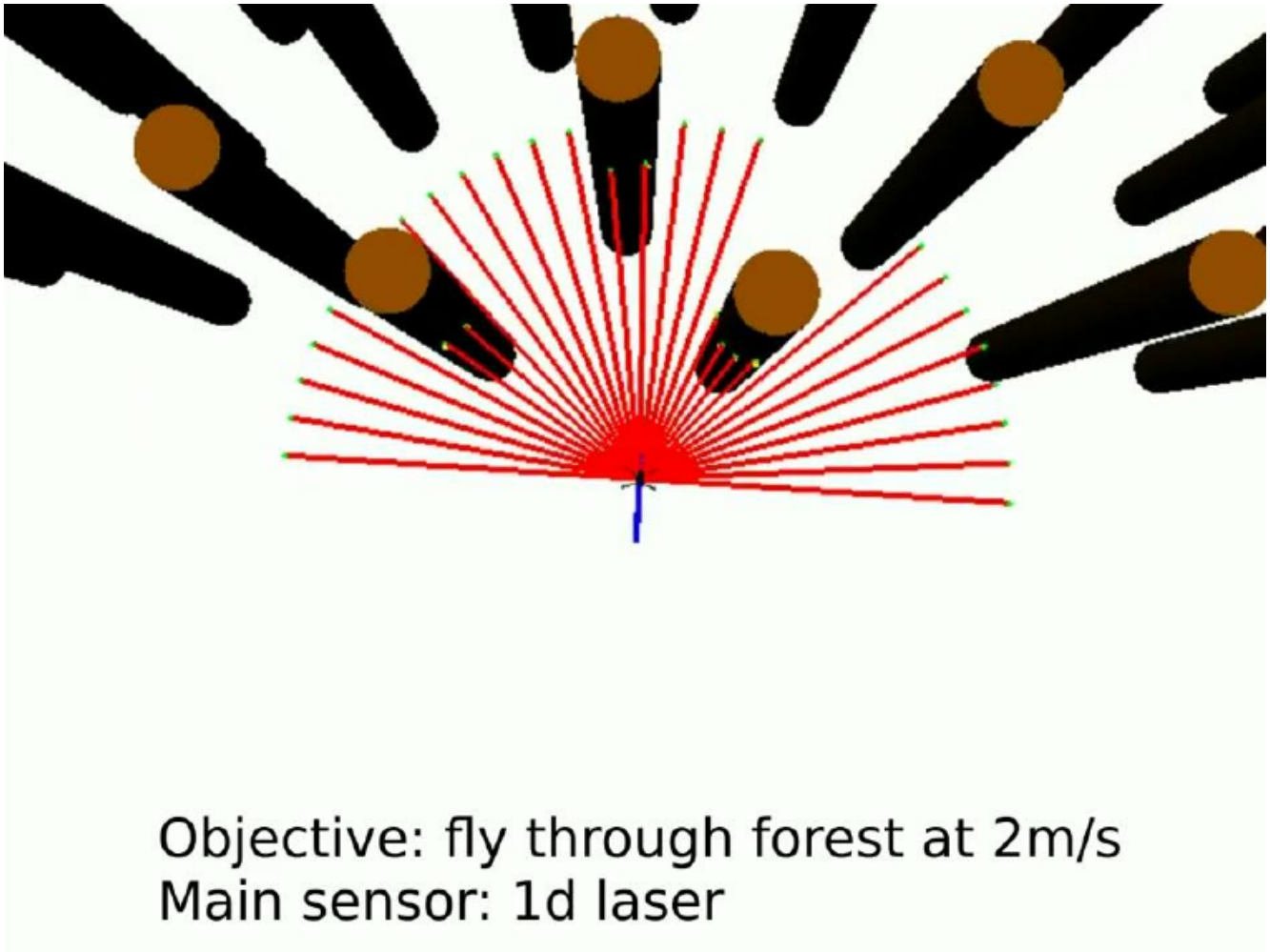
PLATO: Policy Learning with Adaptive Trajectory Optimization

$$\hat{\pi}(a_t|s_t) = \arg\min_{\hat{\pi}} \sum_{t'=t}^{T} \underbrace{c(s_{t'}, a_{t'})}_{\text{cost}} + \lambda \underbrace{D_{\mathrm{KL}}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))}_{\text{distance to learned policy}}$$





input substitution trick
need state at training time
but not at test time!



Flight in forest

Summary

- PLATO is a way to perform policy learning from known controller
- Avoids unsafe rollouts
- Don't need full state-space at test-time (because policy uses camera images)