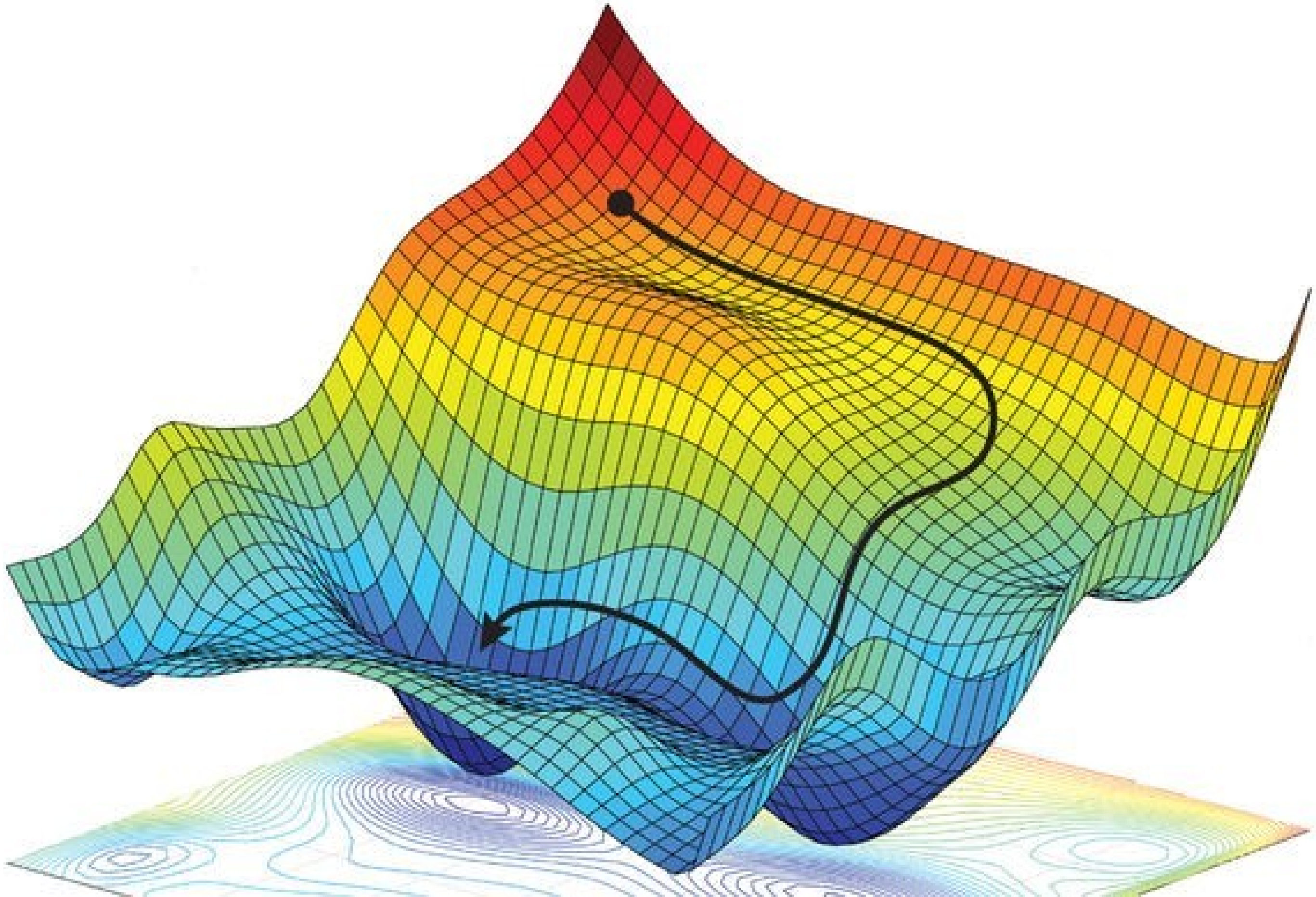
Loss functions



Loss functions

How good is your neural network?

Loss functions

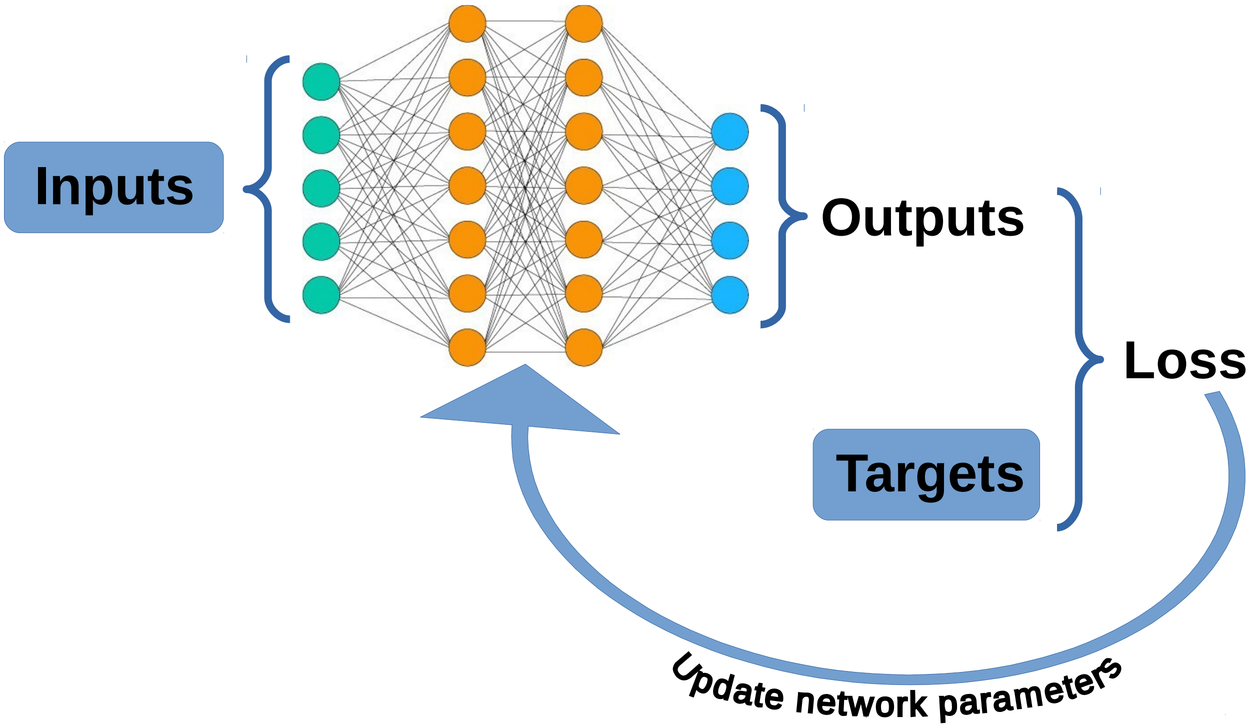
How good is your neural network?

Inputs

e.g.
- *Image (pixel values)*
- *Sentence (encoded)*
- *State (for RL)*

Targets

e.g.
- *'Cat'/'Dog' (encoded)*
- *Next word (encoded)*
- *TD(0) (for RL)*



Loss functions

Some pedantry

***Loss function** and **Cost function** are often used interchangeably but they are different:*

- *The **loss function** measures the network performance **on a single datapoint***
- *The **cost function** is the **average of the losses** over the entire training dataset*

*Our goal is to **minimize the cost function**.*

*In reality, we actually use **batch losses** as a proxy for the cost function.*

Loss/Cost functions

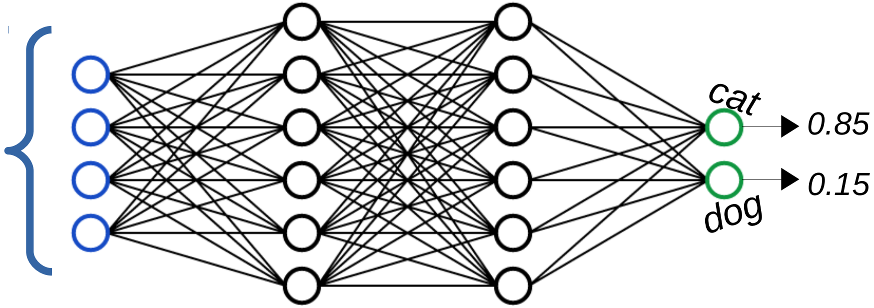
Some stuff to remember

- *The cost function distils the performance of the network down to a* ***single scalar number****.*
- *Generally (although not necessarily)* ***loss*** *≥* ***0*** *(so* ***cost*** *≥* ***0*** *also)*
  - *cost=0 ⇒ perfect performance (on the given dataset)*
- *The* ***lower the value of the cost function, the better the performance*** *of the network.*
  - *Hence gradient <u>descent</u>*
- *A cost function must* ***faithfully represent the purpose*** *of the network.*

Loss/Cost functions

Some stuff to remember

- *The cost function distils the performance of the network down to a **single scalar number**.*
- *Generally (although not necessarily) **loss** ≥ **0** (so **cost** ≥ **0** also)*
  - *cost=0 ⇒ perfect performance (on the given dataset)*
- *The **lower the value of the cost function, the better the performance** of the network.*
  - *Hence gradient <u>descent</u>*
- *A cost function must **faithfully represent the purpose** of the network.*



Accuracy: 1
Loss (CCE): *0.07*

Loss/Cost functions

Loss functions

- ***Classification***
  - *Maximum likelihood*
  - *Binary cross-entropy (aka log loss)*
  - *Categorical cross-entropy*
- ***Regression*** *(i.e. function approximation)*
  - *Mean Squared Error*
  - *Mean Absolute Error*
  - *Huber Loss*

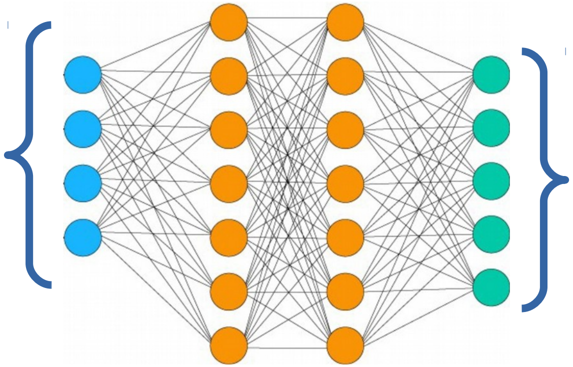
Classification loss functions

Loss functions

- ***Classification***
  - *Maximum likelihood*
  - *Binary cross-entropy (aka log loss)*
  - *Categorical cross-entropy*

Classification loss functions

Maximum likelihood



Cat - Dog - Bear - Hitler - Putin -

(.5, .9, .2, .6, .1)

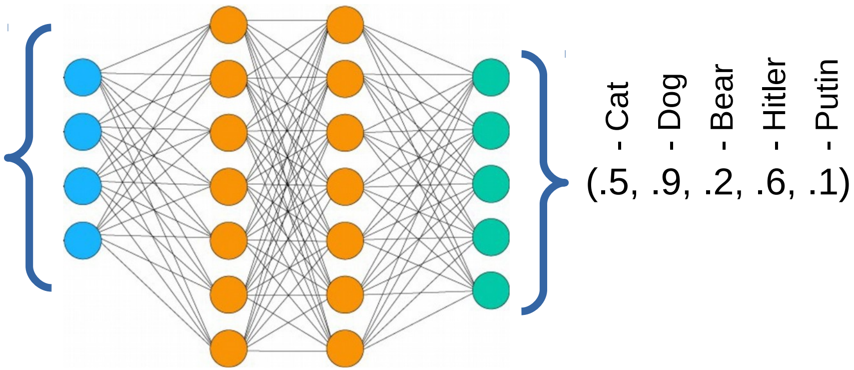
Target

(1, 1, 0, 0, 0)

Cat - Dog - Bear - Hitler - Putin -

Classification loss functions

Maximum likelihood



Target

(1, 1, 0, 0, 0)

Cat - Dog - Bear - Hitler - Putin -

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot (p(y_i)) + (1 - y_i) \cdot (1 - p(y_i))$$

$$= -{}^{1}/_{5}\,[\,(1*0.5 + 0*0.5) + (1*0.9 + 0*0.1) + (0*0.2 + 1*0.8) + (0*0.6 + 1*0.4) + (0*0.1 + 1*0.9)\,]$$

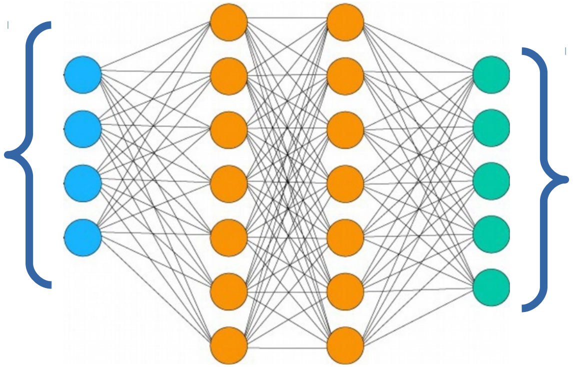
$$= -0.7$$

Classification loss functions

Binary cross-entropy (aka Log loss)

Torch.nn.BCELoss

Input



Cat - Dog - Bear - Hitler - Putin

(.2, .4, .8, .3, .9)

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Target

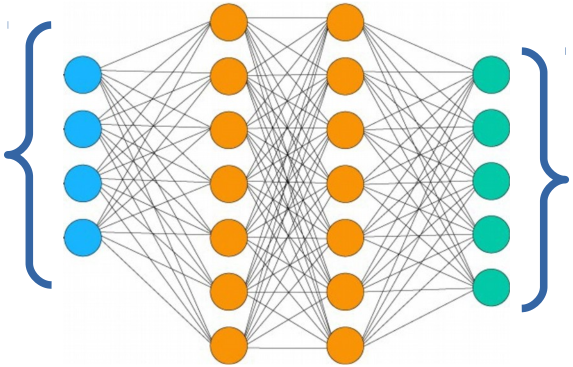
(0, 0, 1, 0, 1)

Cat - Dog - Bear - Hitler - Putin

Classification loss functions

Binary cross-entropy (aka Log loss)

Torch.nn.BCELoss



Cat, Dog, Bear, Hitler, Putin

(.2, .4, .8, .3, .9)

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Target

(0, 0, 1, 0, 1)

Cat, Dog, Bear, Hitler, Putin

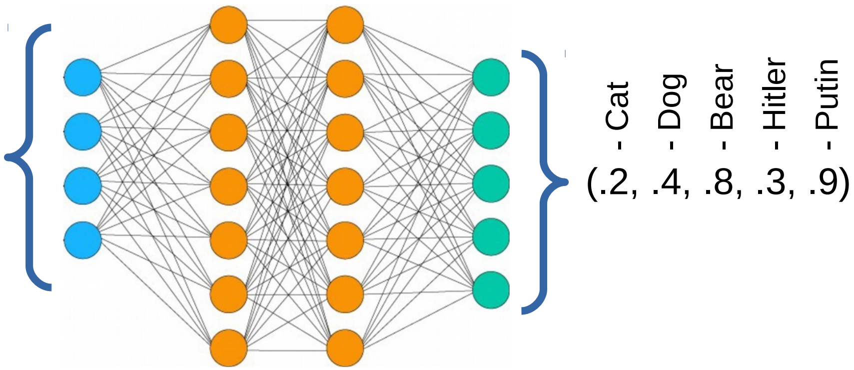
$$= -{}^{1}/_{5}\ [\ (1*log(0.8)) + (1*log(0.6)) + (1*log(0.8)) + (1*log(0.7)) + (1*log(0.9))\ ]$$

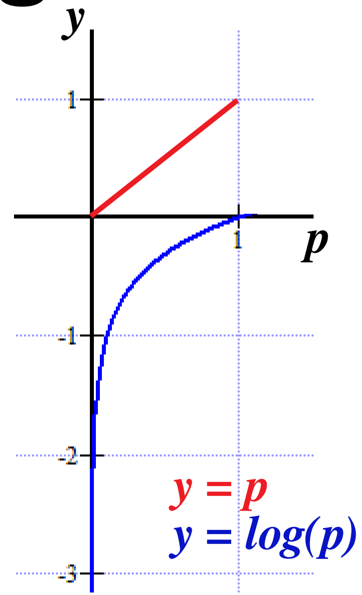
$$= 0.28$$

Classification loss functions

Binary cross-entropy (aka Log loss)

Torch.nn.BCELoss





$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Target

(0, 0, 1, 0, 1)

Cat -
Dog -
Bear -
Hitler -
Putin -

$$= -{}^{1}/_{5}\ [\ (1*log(0.8)) + (1*log(0.6)) + (1*log(0.8)) + (1*log(0.7)) + (1*log(0.9))\ ]$$

$$= 0.28$$

Classification loss functions

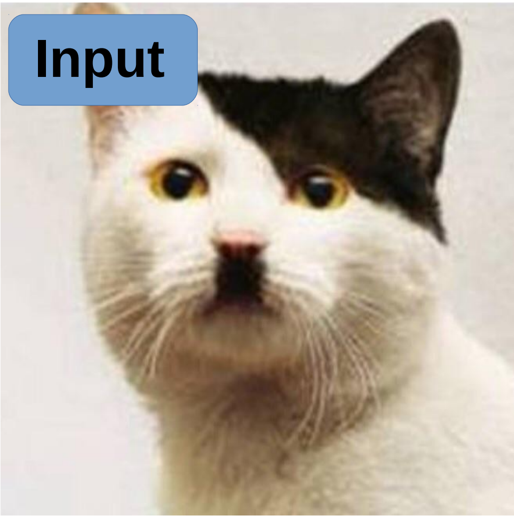
Categorical cross-entropy

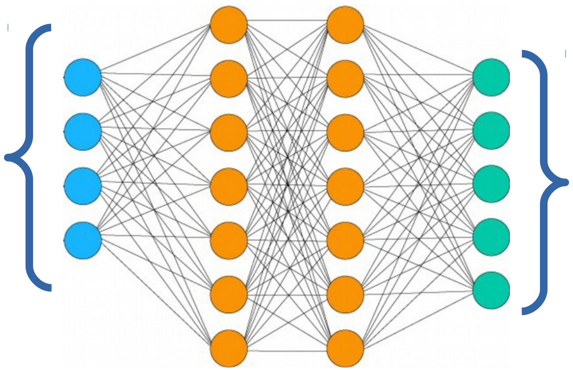
Torch.nn.NLLLoss / Torch.nn.CrossEntropyLoss

This one does the softmax for you

$$= \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right)$$

Input





- Cat - Dog - Bear - Hitler - Putin

(.8, 0, 0, .2, 0)

Must be a probability distribution

$$Loss = -\sum_{i=1}^{N} y_i \cdot log(p(y_i))$$

$$= -[\ 1*log(0.8)\ ]$$

$$= 0.22$$

Target

(1, 0, 0, 0, 0)

Cat - Dog - Bear - Hitler - Putin -

Classification loss functions

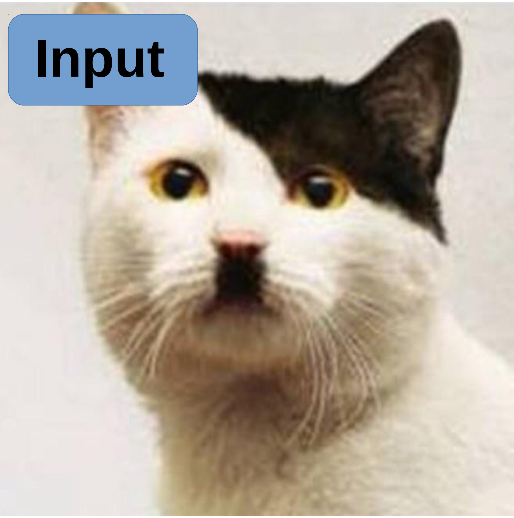
Categorical cross-entropy

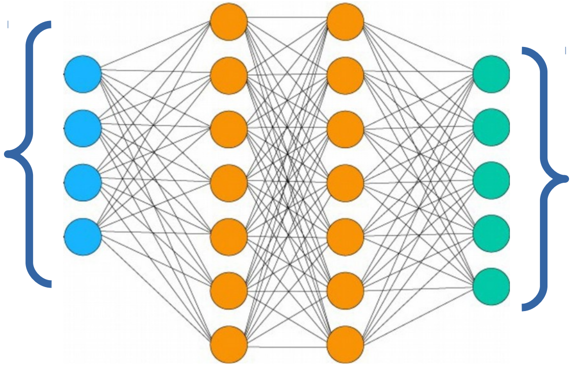
Torch.nn.NLLLoss / Torch.nn.CrossEntropyLoss

This one does the softmax for you

$$= \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right)$$

Input





(.8, 0, 0, .2, 0)

Cat - Dog - Bear - Hitler - Putin -

Must be a probability distribution

$$Loss = -\sum_{i=1}^{N} y_i \cdot log(p(y_i))$$

$$= -[\ 1*log(0.8)\ ]$$

$$= 0.22$$

Target

(1, 0, 0, 0, 0)

Cat - Dog - Bear - Hitler - Putin -

This loss function can also be used when you have a single binary (e.g. yes/no) output

Regression loss functions

Loss functions

- ***Regression*** *(i.e. function approximation)*
  - *Mean Squared Error*
  - *Mean Absolute Error*
  - *Huber Loss*

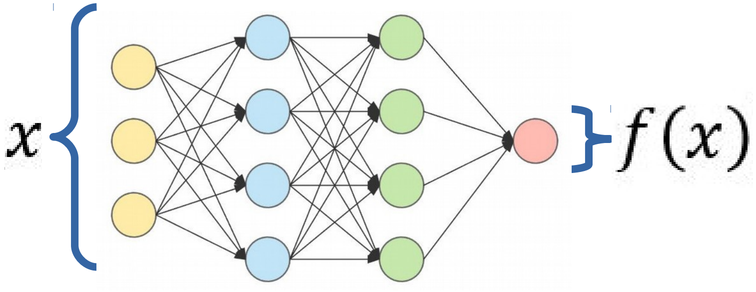
Regression loss functions

aka L2 loss

Mean squared error

Torch.nn.MSELoss

Input x



Target y

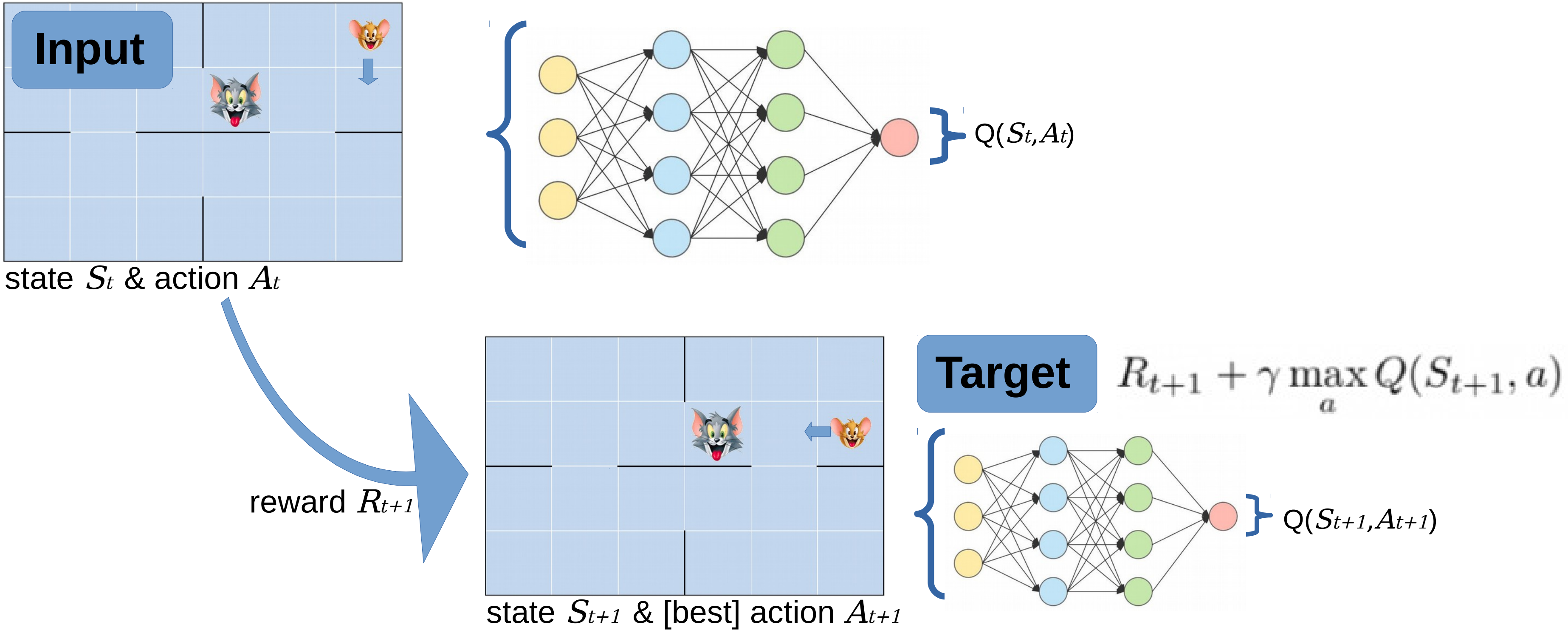
$$Loss = (y - f(x))^2$$

Regression loss functions

aka L2 loss

Mean squared error

Torch.nn.MSELoss



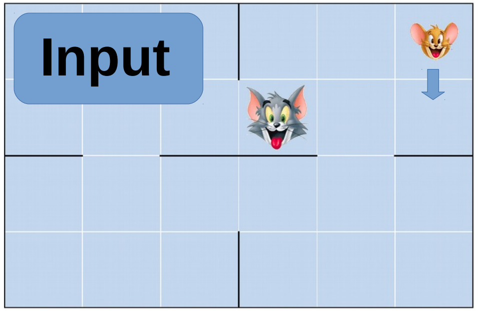
Regression loss functions

aka L2 loss

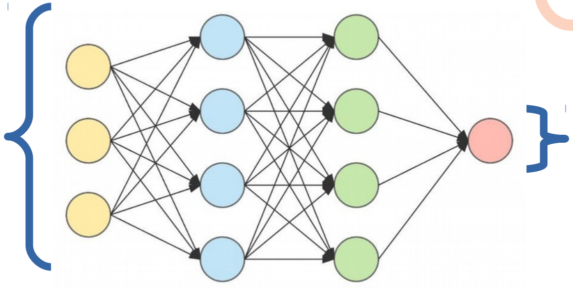
Mean squared error

Torch.nn.MSELoss

$$Loss = (y - f(x))^2$$
$$= \left(\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right) - Q(S_t, A_t) \right)^2$$

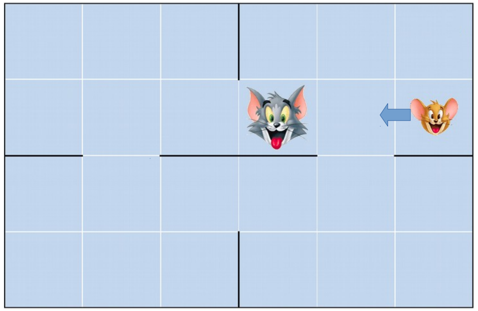


state S_t & action A_t



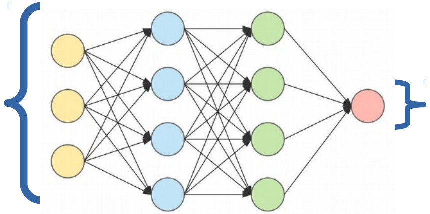
$Q(S_t, A_t)$

reward R_{t+1}



state S_{t+1} & [best] action A_{t+1}

Target $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$



$Q(S_{t+1}, A_{t+1})$

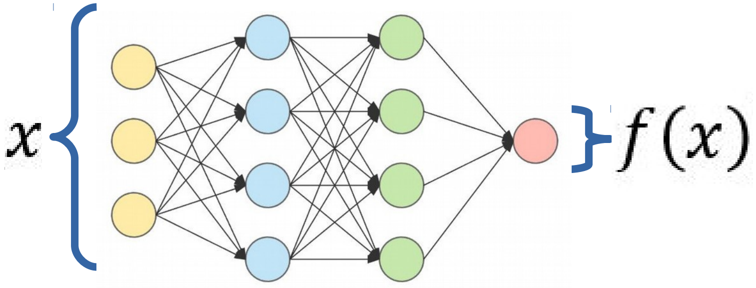
Regression loss functions

aka L1 loss

Absolute error

Torch.nn.L1Loss

Input x



Target y

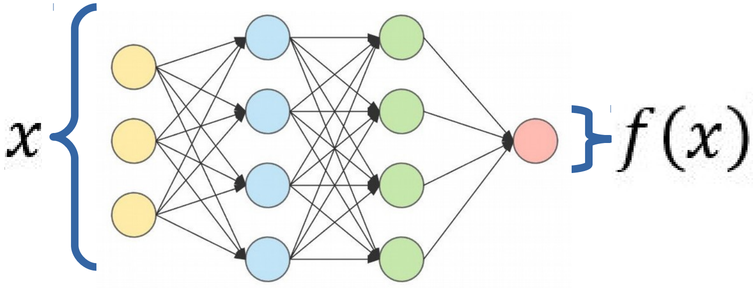
$$Loss = |y - f(x)|$$

Regression loss

Huber loss

Torch.nn.SmoothL1Loss

Input x



Target y

$$Loss = \begin{cases} \frac{1}{2}\left(y - f(x)\right)^2, & if\ |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases}$$

Huber loss combines the best features of MSE and AE:
- like MSE for small errors.
- like AE otherwise.

This avoids over-shooting:
- MSE has a **very steep gradient when the error is large** (i.e. for outliers) because of its quadratic form.
- AE has a **steep gradient when the error is small** because it doesn't have a 'flat' bottom.