**OPTIMIZER Adam:**

<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

**In our model:**

**Learning Rate** = 1e-3 // **epsilon** = eps=1e-8 // **weight decay** (L2 penalty) = 0 might change to 0.01

**1. Learning rate** (0≤ lr ≤ 1)**:** a large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to learn a more optimal or even globally optimal set of weights but may take significantly longer to train. This is the most important hyperparameter, we always fine tune it with trial and error

**2. epsilon:** variable <- variable - lr\_t \* m\_t / (sqrt(v\_t) + epsilon)

Small number that doesn’t allow denominator become 0.

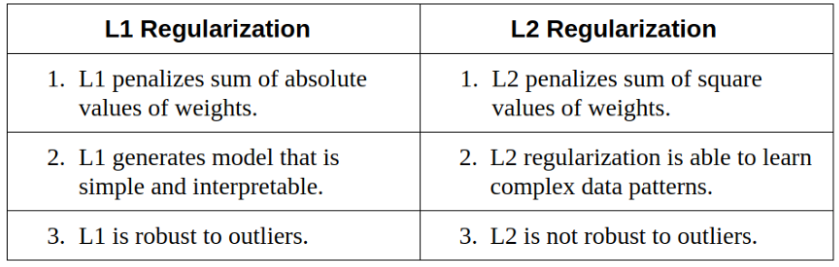
training with small epsilon the optimizer becomes unstable.

the bigger the epsilon (and the denominator), the smaller the weight updates are and thus slower the training progress will be.

**3. weight decay** (0≤ L2 ≤ 1)**:** reduces overfitting by keeping the values of the weights and biases small

**4. decay:** the learning rates adapt themselves during training steps, so we don’t add decay manually. We have Beta1 = 0.9 and Beta2 = 0.999, the default decays from Adam.

L2 (ridge regression) vs L1 (Lasso Regression):



\* We don’t use L1, because it removes features of the data (sets their weights to 0)

**Generally:** more exploitation will go faster to a local optimum, but will skip many other optimum values. More exploration will traverse many local optima and find the best, but in more time

**Loss functions:**

Graph explanations: <https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2>

**Pi\_loss:** The mean magnitude of policy loss function. Correlates to how much the policy (process for deciding actions) is changing. The magnitude of this should decrease during a successful training session. These values will oscillate during training. Generally, they should be less than 1.0.

**Value loss:** The mean loss of the value function update. Correlates to how well the model is able to predict the value of each state. This should increase while the agent is learning, and then decrease once the reward stabilizes. These values will increase as the reward increases, and then should decrease once reward becomes stable.

**Entropy Loss:** How random the decisions of the model are. Should slowly decrease during a successful training process. If it decreases too quickly, the beta hyperparameter should be increased. This corresponds to how random the decisions of a Brain are. This should consistently decrease during training. If it decreases too soon or not at all, beta should be adjusted (when using discrete action space)

**PPO Implementation:**

PPO walkthrough: <https://adventuresinmachinelearning.com/proximal-policy-optimization-ppo-tensorflow/>

general theory: <https://towardsdatascience.com/a-graphic-guide-to-implementing-ppo-for-atari-games-5740ccbe3fbc>

**Gradient descent:** probability to chose next step (calculates trajectories and then standardizes weights, so it doesn’t chose always the same direction)