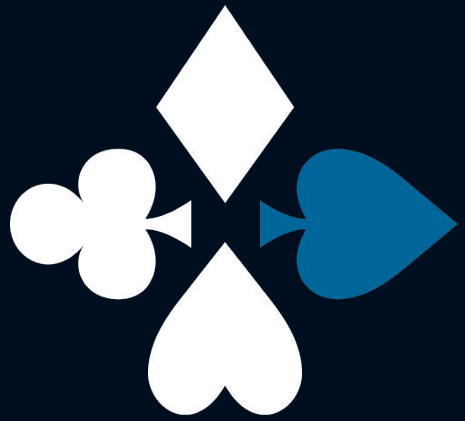


Pokerbots 2025

Lecture 7: Neural Networks and Optimization

Sponsors

CITADEL | CITADEL Securities

Announcements

Hackathon

Thanks for coming!

Prizes awarded:

- Prediction Game (Sony XM4):
  wcs312
- Most improved team before/after ($200):
  STATosphere

Week 2 Mini Tournament Results

WINNER [$1000]:

Encore 1/2 Regs

BIGGEST UPSET [$500]:

zen_fold

Tournament ELO 942, won majority matches against “free lunch lovers” (ELO 2048)

MOST IMPROVED [$750]:

long delta cfd

Tournament ELO 722 → 1841

Make-up Progress Reports due Tonight

- Required if your team missed the Week 2 bot submission deadline
- Reports are half a page and must list team name and members
- Submit via email to pokerbots@mit.edu by 11:59pm Tonight

Giveaways!

First Rare Submission: **pkr.bot/rare**

- Choose an integer in 0-9 and a color in {red, green, blue}
- *Rare submission*: minimal amount of peers with identical choices
- Winner is first person to make a rare submission
- Prize: JBL Flip 5

Clock Game: **pkr.bot/clock**

- Submit a timestamp in the form of minutes and seconds past noon EST
- Winner is the closest to the mean response time of first 40 submissions
- Prize: 24" Dell Monitor S2425HS

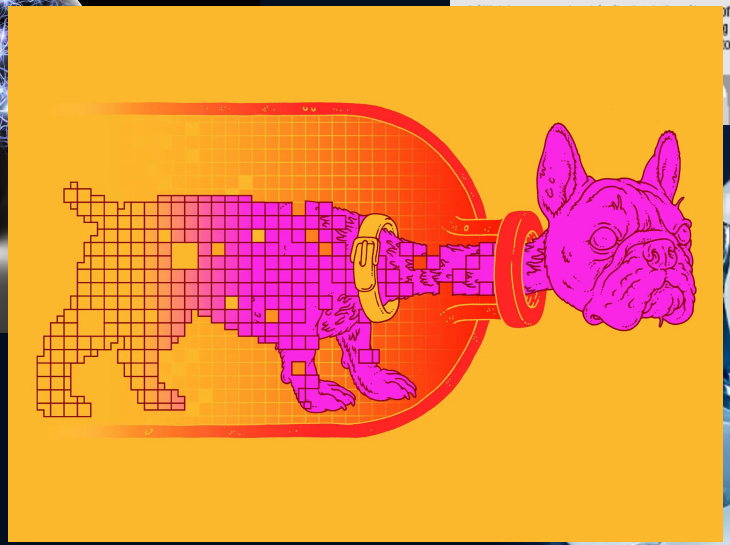
Neural Networks & Optimization

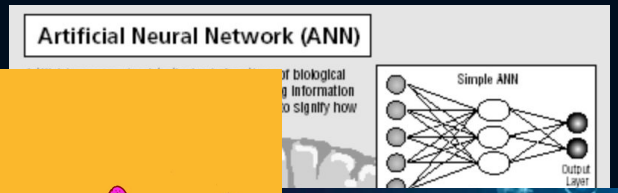
Agenda

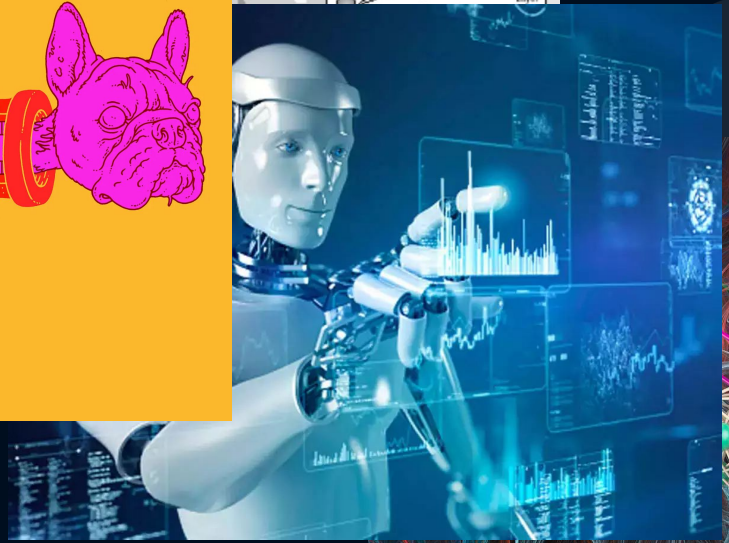
- Overview and Architecture of Neural Networks
- ML Optimization and Training Algorithms
- Neural Network Applications and Considerations
- PyTorch Coding Demo

Overview and Architecture of Neural Networks

Neural Networks seem to be all the hype…





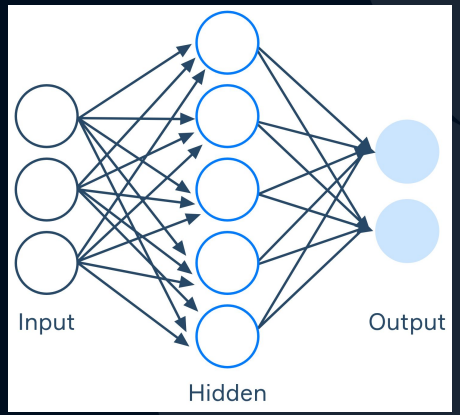


So what exactly do they do?

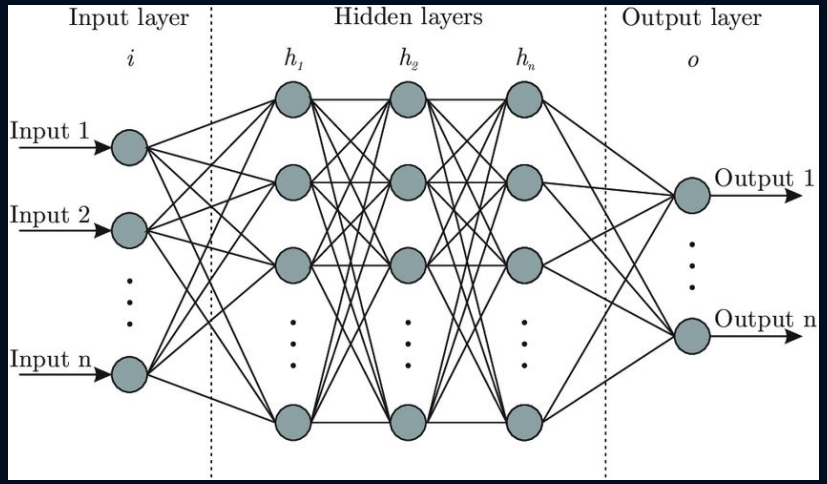
Another parameterized model class

Recall: “template for a machine learning algorithm”

- We specify such a template using *hyperparameters*
- then tweak settings within the template called *parameters*
- they modify the behavior of our *algorithm*
- which is just a function that takes in input and gives us output
- ideally our outputs are good



Neural Network Anatomy

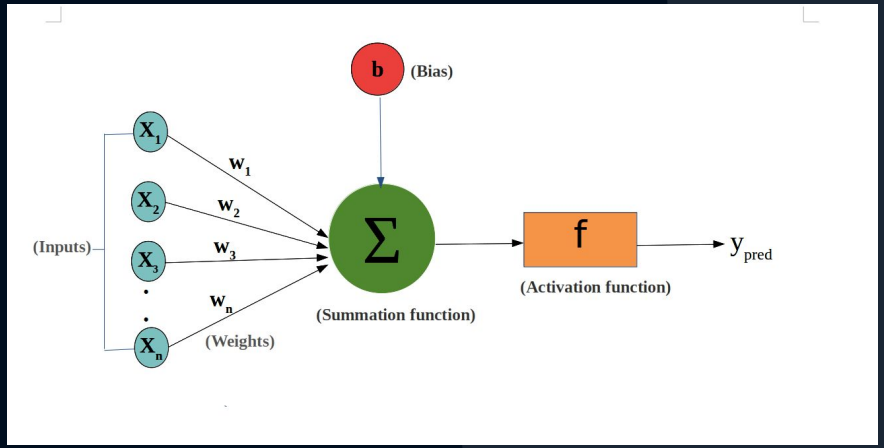


- Comprised of Layers
  - Input layer and output layer
  - “Hidden layers”: every layer in between
    - How many is up to us!
- Each layer is a set of values, or “neurons”
  - Input layer size = dimension of input
  - Output layer size = dimension of output
  - Hidden layer sizes are up to us!
- Evaluation starts at input layer
- The neurons in each layer are calculated using those from the previous layer
- Resulting values in the output layer is the output of the model

Evaluation of Layers

Each neuron is a transformed weighted sum of the previous layer, i.e.

- All the neurons of the previous layer are assigned *weights*.
- They are multiplied by their weights and summed, often with a *bias* value.
- This linear combination is then passed through an *activation function*



Weights and Biases

There is a unique set of weights and biases for every pair of consecutive layers:

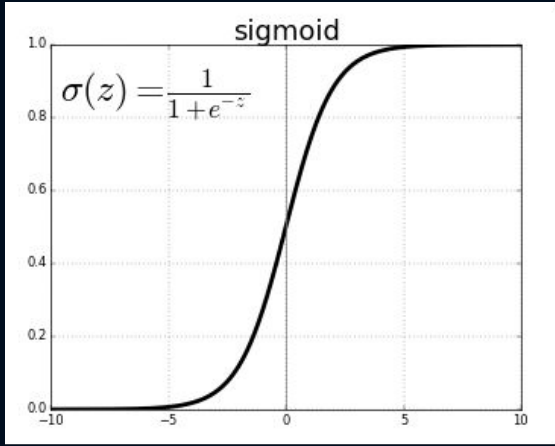
- One weight for each pair of neurons across the two layers
- One bias for each neuron in the next layer

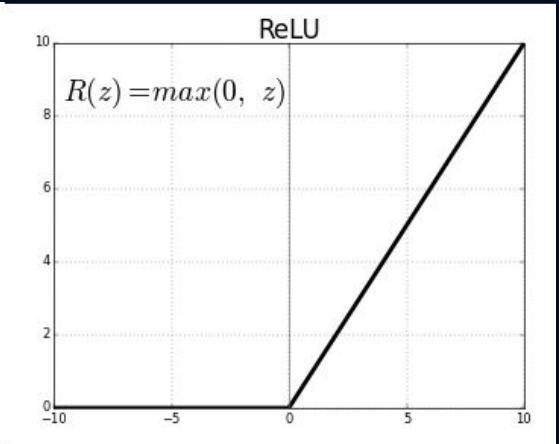
These values comprise the *parameters* of the neural network.

Our usual goal (e.g. supervised learning context) is to find settings for these parameters such that the neural network behaves well and produces useful outputs

Activation function

- Applied to linear combinations to produce new neuron values.
- The function itself is another thing that's up to us!
- Biggest rule is that it cannot be a linear function
- We also generally want it to be monotonic (ex. Always increasing)
- Common choices: sigmoid, ReLU (Rectified Linear Unit)





All together

First step: choose *hyperparameters*, aka design neural network template

- Number of layers and neurons in each
- Activation function for each layer

This first step usually involves the task at hand (dataset size, input/output types, etc.)

Next step: choose *parameters*, aka fill in the template

- weights and biases

How do we find the right parameters?

ML Optimization and Training

General Parameterization of Functions

- A neural network is an example of a general parameterized function, which can be expressed as $f(x; \theta)$, where θ represents the current setting of parameters, and x is the input.
- In a machine learning setting we would like to choose θ such that the loss resulting from the output $y = f(x; \theta)$ is as small as possible
- Finding a good θ will involve trying different values and evaluating their loss on our training data to see if it worked

Naive Idea: parameter tweaking

- Start with some parameter values
- Make a bunch of copies and make different very small changes to each copy
- Evaluate the original and each copy by computing the total loss and keep the best one
- Repeat

General idea here: find the best “direction” to shift the parameters in…

Gradient Descent

Use calculus to find the best direction!

This direction is a vector of updates to our parameters, given by the gradient of total loss with respect to the parameters.

Compute the gradient and take a step scaled by some factor

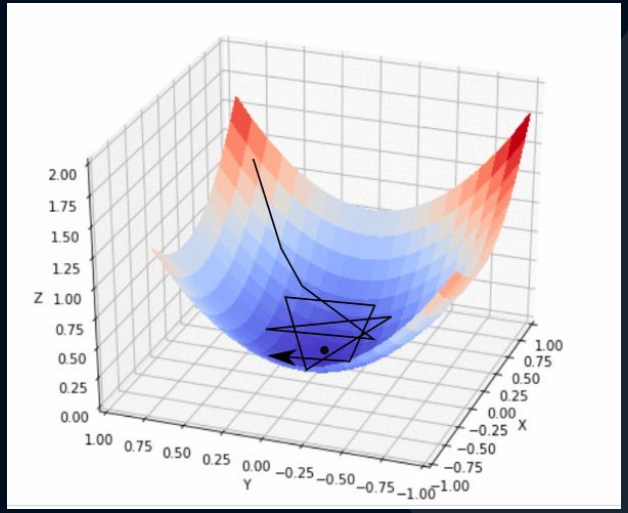
$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta)$$

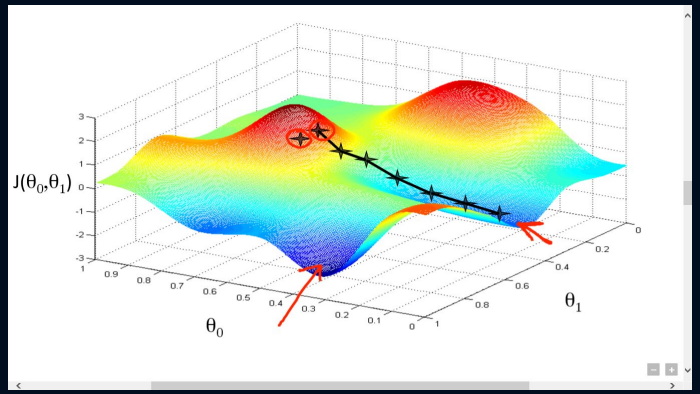
η: Learning rate (step size).
$L(\theta)$: Gradient of the loss function w.r.t. parameters.

Convex Optimization

For convex functions and carefully chosen step size, this will find the very best set of parameters (global optima)

However landscape on real datasets is rarely nice and convex





Making Gradient Descent Better

- Stochastic Gradient Descent (SGD)
- Batch Optimization
- Sizing step size
- Adam

Neural Network Considerations and Applications

Lunch 😋

Leave any type of feedback at **pkr.bot/feedback**!

Saloniki

GREEK

Neural Network Considerations and Applications

Backpropagation

An algorithm for computing gradients in a neural network using the chain rule of calculus.

- Takes advantage of the sequential computation of neural networks
- Makes use of linear algebra to simplify everything into matrix operations
- Modern libraries like PyTorch can do all the calculus for us!
- GPUs are good at matrix operations and are very helpful in training and evaluating neural networks

NNs in the wild

- Supervised Learning
  - Classification
  - Regression
  - Everything!
- Unsupervised
  - Representation Learning
  - Autoencoders
  - Language Modeling
- Reinforcement Learning
  - Approximate Value Iteration
  - Deep Q Learning
  - Policy Gradient

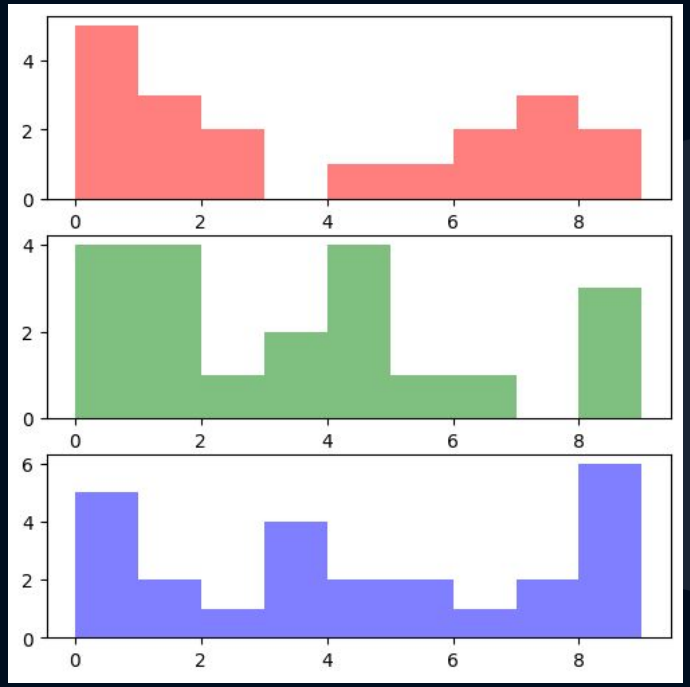
Why are they so useful?

- Can be modified to work in a number of settings
- “Universal function approximators”
- Extremely expressive due to combination of linear steps with nonlinear activation functions.
- Getting easier and easier to train with modern tools

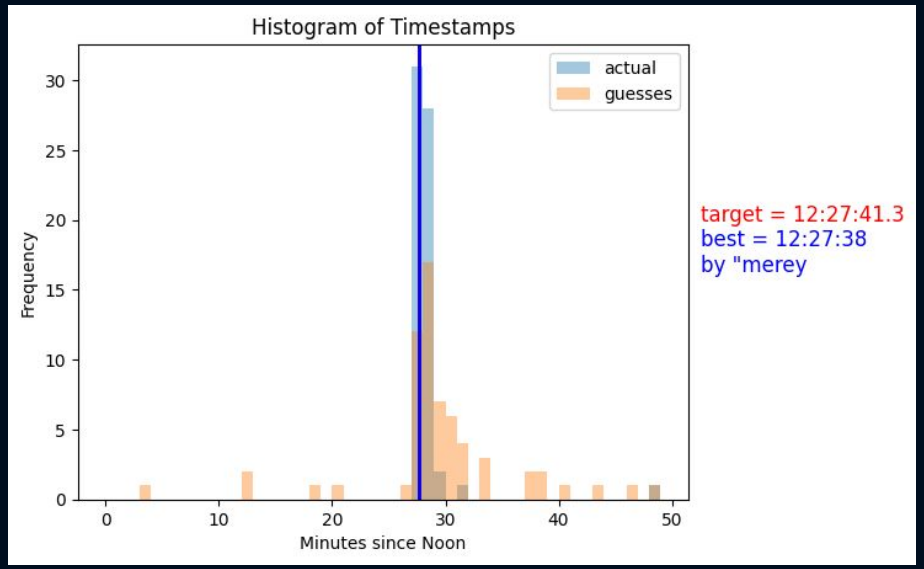
Live Coding: reference-7-2025

Giveaway Winners

First Rare Submission: kerb “brendonj” @ (5, green)



Clock Game: kerb “merey”



Thanks for watching!

Slides will be posted on **pkr.bot/resources**

Repo will be pushed to **pkr.bot/github**

Make sure to check **pkr.bot/piazza** for updates

Lecture recordings at **pkr.bot/panopto**

Leave feedback at **pkr.bot/feedback**!