Title Page - 1 page

review publications by members of my lab - sankar ganesh

abstract - one page - 200 words

introduction - 1 page - background, hypothesis, general approaches

materials and methods - 2-3 pages -> subsections of relative degree

* gradient descent (explanatory)

results: 2 to 3 pages -> describe both positive and negative results

discussion and conclusions

subsections:

**Welcome to the IDA**

The Image and Data Archive (IDA) is a **secure** online resource for **archiving**, **exploring** and **sharing** neuroscience data.

MNIST DATA

Magnetic resonance imaging

Diffusion Weighted (DW) imaging is a form of magnetic resonance imaging (MRI) technique that is widely used in tumor identification and classification in modern clinical radiology practices [6, 7]. This technology is based on measurements of random Brownian motion of water molecules within a voxel of a biological tissue [8,9,10]. The technique allows to visualize the net direction of diffusion of water molecules or collective flow of water molecules in a live tissue. Hence, it has the ability to provide information on the microscopic behaviour of living biological tissues (such as the presence and permeability of membranes and the presence of macro-molecules and intracellular–extracellular water equilibrium) by measuring and imaging the transitional mobility of water molecules [11,12,13,14]. Due to the characteristic features of DW images, they are appreciated as an indispensable tool for investigating CNS diseases, such as brain neoplasms, brain and spinal cord injuries, degenerative brain diseases, etc.

Texture features

The generated ADC images reflect the magnitude of diffusion of water molecules within tissues and these images are rich in texture allowing the analysis of image in terms of these features. The texture of an image can be defined as a constant repetition of an element or pattern on the surface of an image which represents its structure [18, 19]. Texture analysis focuses on finding a specific way of representing the hidden characteristics of textures and express them in a simplified and unique form.

Machine learning algorithms

Logistic Regression, K-Nearest Neighbors (KNN), Linear Discriminant Analysis, Naïve Bayes, Decision Trees and Random Forest are few of the most common supervised learning algorithms frequently used to solve classification problems.

Logistic regression is a ML algorithm that is designed to solve classification problems by mapping functions from attributes of a data set to its targets. The developed functions are introduced to new examples and predict the probability that the new example belongs to one of the target classes.

K-Nearest Neighbors algorithm is a supervised machine learning algorithm that can be applied on both regression and classification problems. The KNN algorithm assumes that similar data points in a data set exist nearby.

Linear Discriminant Analysis is a ML algorithm that was developed to find a linear combination of features of a data set that separates the data set into two or several classes.

Naïve Bayes is a learning algorithm that is based on the Bayes’ rule to solve classification problems. To apply the Naïve Bayes algorithm, it is crucial to assume that the attributes are conditionally independent in each class. In practice, the above assumption is frequently violated and yet provides competitive classification accuracy [34].

Decision tree is a simple algorithm that often applied to solve both classification and regression problems. It represents the decision workflow to identify the class of an instant within the data set. The algorithm creates the most suitable decision tree models for a training data set by placing the name of the class and specific tests that partitions the space of instances on each node in the process of learning simple decision rules.

Random Forest is a meta estimator that generates a cluster of decision trees on various sub-samples of the provided data set aiming at improving the prediction accuracy of the model while controlling over-fitting by averaging. The variables and thresholds that control the number of decision trees create during the learning process, the maximum number of features considered in splitting a node, maximum number of levels included in each decision tree within the algorithm, the minimum number of data points contain in a node prior to split the node, the minimum number of data points allowed to remain in a leaf node, sampling method of the data points (without or with replacement) are optimized as the model returns highest accuracy level avoiding over-fitting.

**background** (PICTURES)

* my lab background -> pamphlet handout -> how it led into my research focus

describe AI - chipsets and AI associative memory in terms of important components of Neural and spatial networks -> how does this compare to chatgpt’s structure?

* hypothesis: Find efficient and accurate techniques with various gradient descent optimizations for CNNs to establish ADD system with high accuracy with diff pooling tech and classifications as well as fuzzy decision theory to make information fusion to different classifiers.

CNN intro:(MNIST data -> MRI data)

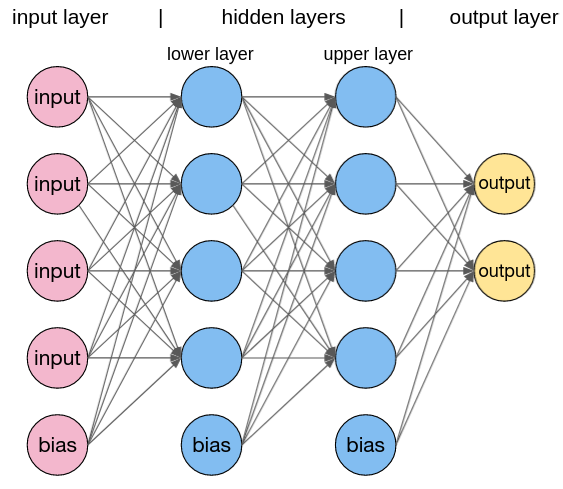
On the surface, a machine recognizing handwritten digits may not seem particularly impressive. But what if we could write a program that mimics the structure of your brain? That’s the idea behind neural networks. The hope is that by writing brain-inspired software, we might be able to create programs that tackle the kinds of fuzzy and difficult-to-reason-about problems that your mind is so good at solving.

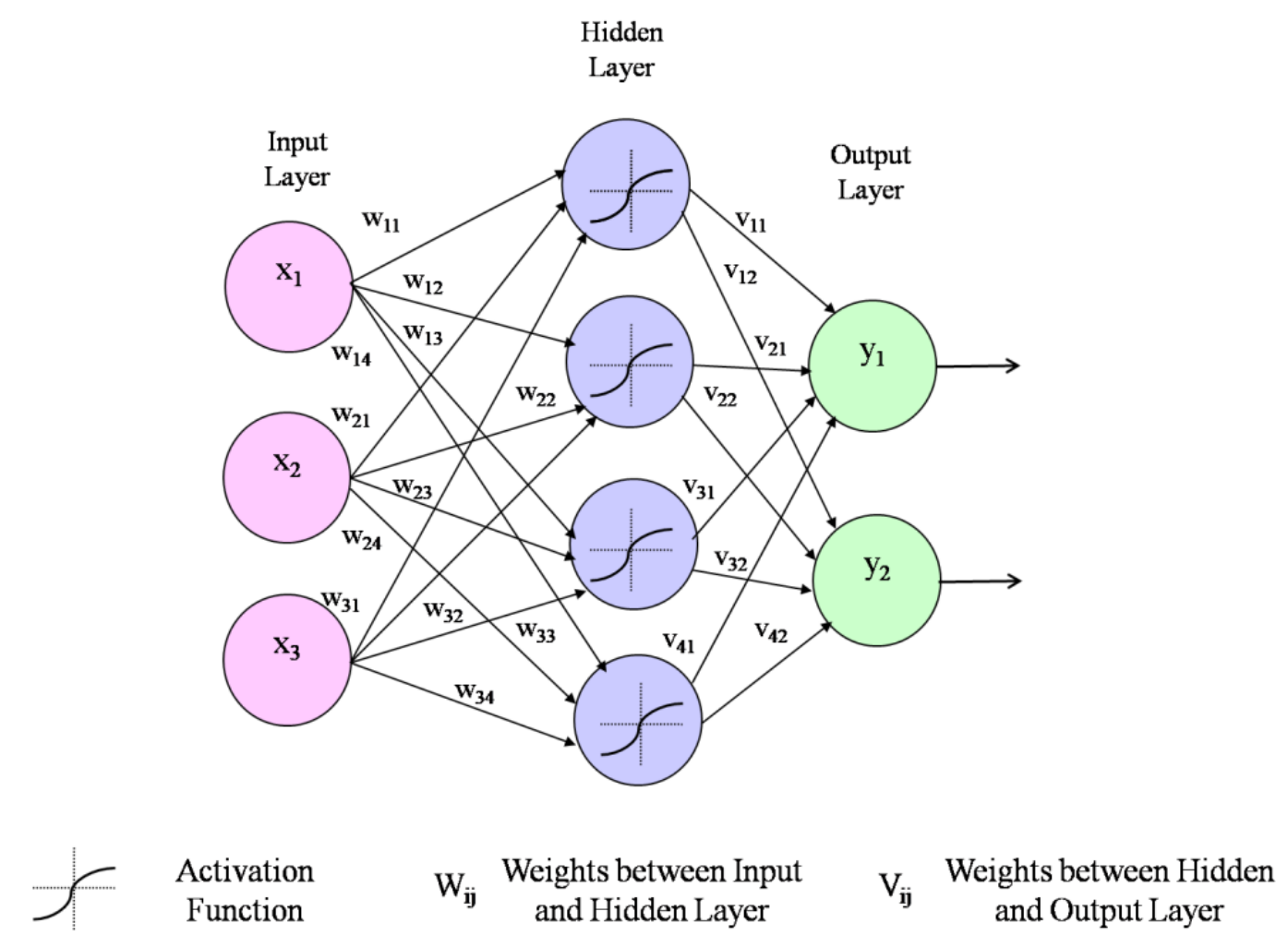
The Structure of a Neural Network

There are many variants of neural networks, such as convolutional neural networks (CNN), recurrent neural networks (RNN), transformers, and countless others. In recent years there’s been a boom in research of these variants. But the first step to understanding any of them is to build up the simplest, plain vanilla form with no added frills.

A **multilayer perceptron** (**MLP**) is a fully connected class of [feedforward](https://en.wikipedia.org/wiki/Feedforward_neural_network) [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANN). The term MLP is used ambiguously, sometimes loosely to mean *any* feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of [perceptrons](https://en.wikipedia.org/wiki/Perceptron) (with threshold activation)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]; see [§ Terminology](https://en.wikipedia.org/wiki/Multilayer_perceptron#Terminology). Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.[[1]](https://en.wikipedia.org/wiki/Multilayer_perceptron#cite_note-1)

An MLP consists of at least three [layers](https://en.wikipedia.org/wiki/Layer_(deep_learning)) of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear [activation function](https://en.wikipedia.org/wiki/Activation_function).





Feed Forward Neural Network? - How this works

In multilayer FFNN, a weighted summation process is used to specify the flow wherein every layer is fully connected to the next layer. So that, each neuron can be capable enough to send its current activation to any connected unit. The transmitted activation is multiplied by the weight of the connection and at the receiving neuron moved through some squashing function (like sigmoid, ReLU, tan h) in order to introduce nonlinearities [15, 51]. The learning is performed by updating the weights to minimize an error function which defined as the difference between desired and the actual output activation vector [22]. Usually, the backpropagation algorithm is used to accomplish the task of learning by taking a partial derivative of the error with respect to the weights of the last layer and then used to modify the weights. Similarly, the partial derivative is computed for the errors with respect to the weights of the second to last layer, and this process is repeated until all the weights connected to the input layer get updated. In spite of being a universal function approximator, FFNNs are poor in dealing with many forms of practical problems such as object and face recognition [5, 34-35, 49, 57]. The reason is full connectivity of the network due to which the number of weights grows rapidly with the input dimension. In addition, the disconcerting fact about the FFNN is their spatial ignorance [9], because separate weights are involved in learning the same object at different location instead of weight transferring. All this happens because every pair of neurons between two layers has their weights. In order to eradicate the problems related with the implemenation of FFNN, the convolutional neural networks (CNNs) are then come into existence. CNNs can exploit the two dimensional spatial constraints imposed by the input modality and at the same time reduce the number of parameters involved in the training process. —>

CNNs are a special class of deep network which has been applied successfully to the data with grid-like topology (image data and time-series data) [26-27, 42, 44]. CNN mainly composed of convolutional, pooling, activation and fully connected layer. The convolutional layer is the integral part of a CNN, where the convolution operation is applied to the input. Convolution operation leverages the three important ideas named as equivalent representations, sparse connectivity and tied weights which play an essential role in the improvement of machine learning systems specifically to solve the tasks related to computer vision [43]. Since the architecture of CNN consists of multiple convolutional layers so as for images. In this context, a bank of filters is applied to an image (at every convolutional layer), and the output is obtained in a piled manner. The piling of outputs increases the abstract feature and makes the pixel-wise analysis more complicated. In order to alleviate this complexity, pooling layers are inserted after the convolutional layers. A schematic representation of the most commonly used CNN architecture with a pooling operation is depicted in Figure 2 with an explanation.

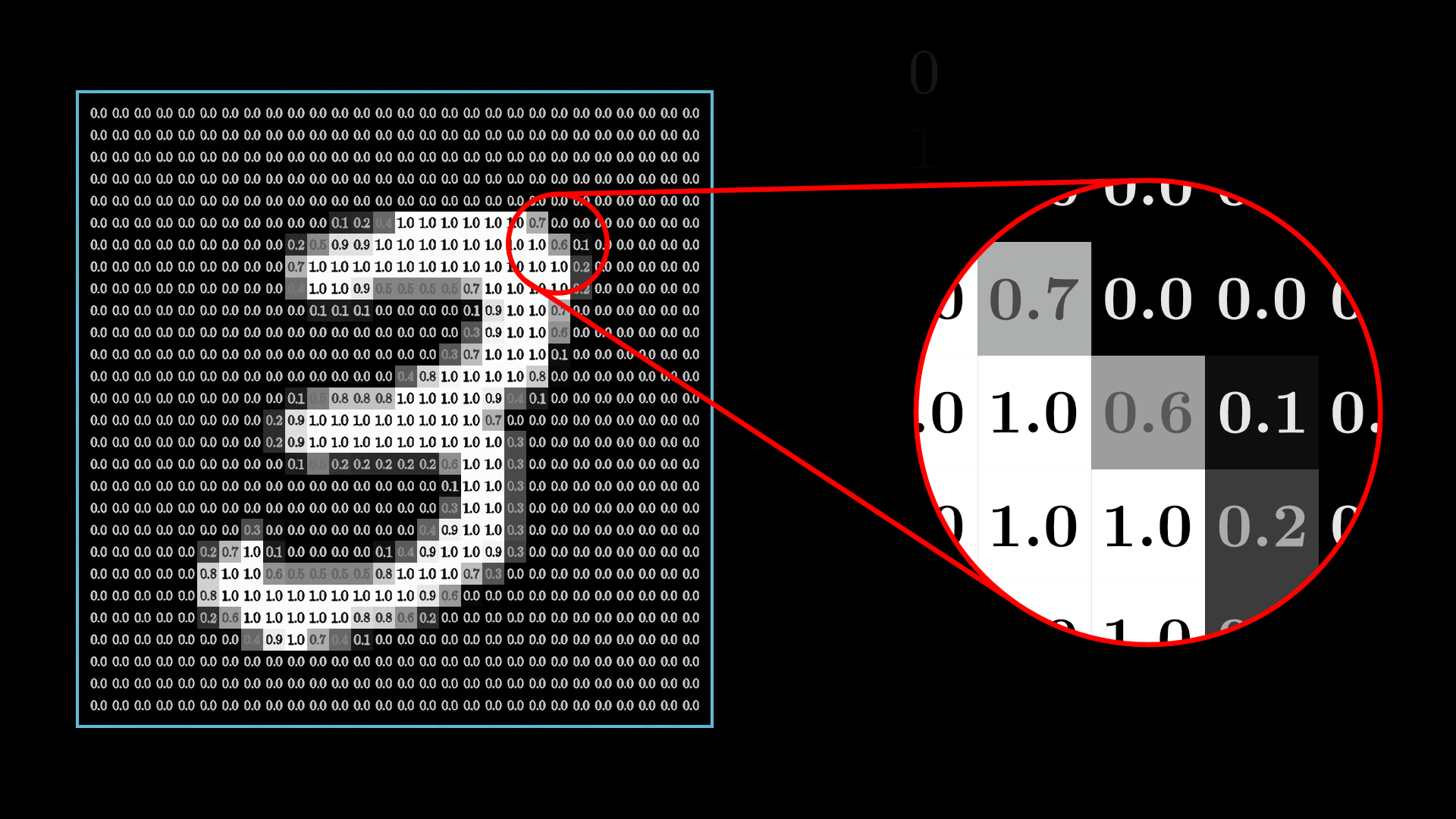
Convolutional neural network is better than a feed-forward network since CNN has features parameter sharing and dimensionality reduction. Because of parameter sharing in CNN, the number of parameters is reduced thus the computations also decreased. STILL RELATIVELY OLD ARCHITECTURE -> BETTER VERSIONS DISCUSSED LATER.

back to the story of basics:

Neural networks are really just a bunch of neurons connected together.

This number inside the neuron is called the “activation” of that neuron, and the image you might have in your mind is that each neuron is lit up when its activation is a high number.

All the information passing through our neural network is stored in these neurons. So we need to represent the inputs and outputs of our network (the images and digit predictions) in terms of these neuron values between 0.0 and 1.0.



The input layer and output layer are rather straightforward: the input neurons will consist of the total input states of our image/data whereas the output represents the total possible outputs.

The Hidden Layers

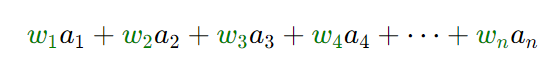
how does the hidden layers work?

Each link has a different relative strength that dictates the network's behavior. where each hidden layer effectively breaks the input problem into solvable pieces -> detect edge, pattern, etc. -> layers of abstraction. The complications start when we assign a weight/Bias to each of the connections.

Each weight is an indication of how its neuron in the first layer is correlated with this new neuron in the second layer.

If the neuron in the first layer is on, then a positive weight suggests that the neuron in the second layer should also be on, and a negative weight suggests that the neuron in the second layer should be off.

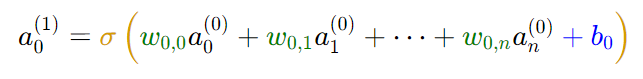
So to actually compute the value of this second-layer neuron, you take all the activations from the neurons in the first layer, and compute their weighted sum.



So it’s common to pump this weighted sum into some function that squishes the real number line into the range between 0 and 1.One common function that does this is called the “sigmoid” function, also known as a logistic curve, which we represent using the symbol

σ. Very negative inputs end up close to 0, very positive inputs end up close to 1, and it steadily increases around 0. So the activation of the neuron here will basically be a measure of how positive the weighted sum is. You can then add a bias to the weighted sum before this sigmoid function to make the output have meaning when active.

—> HOW DOES THIS NETWORK “LEARN”? - most of my research

ReLU sigmoid function

superscript: layers

subscript: weight

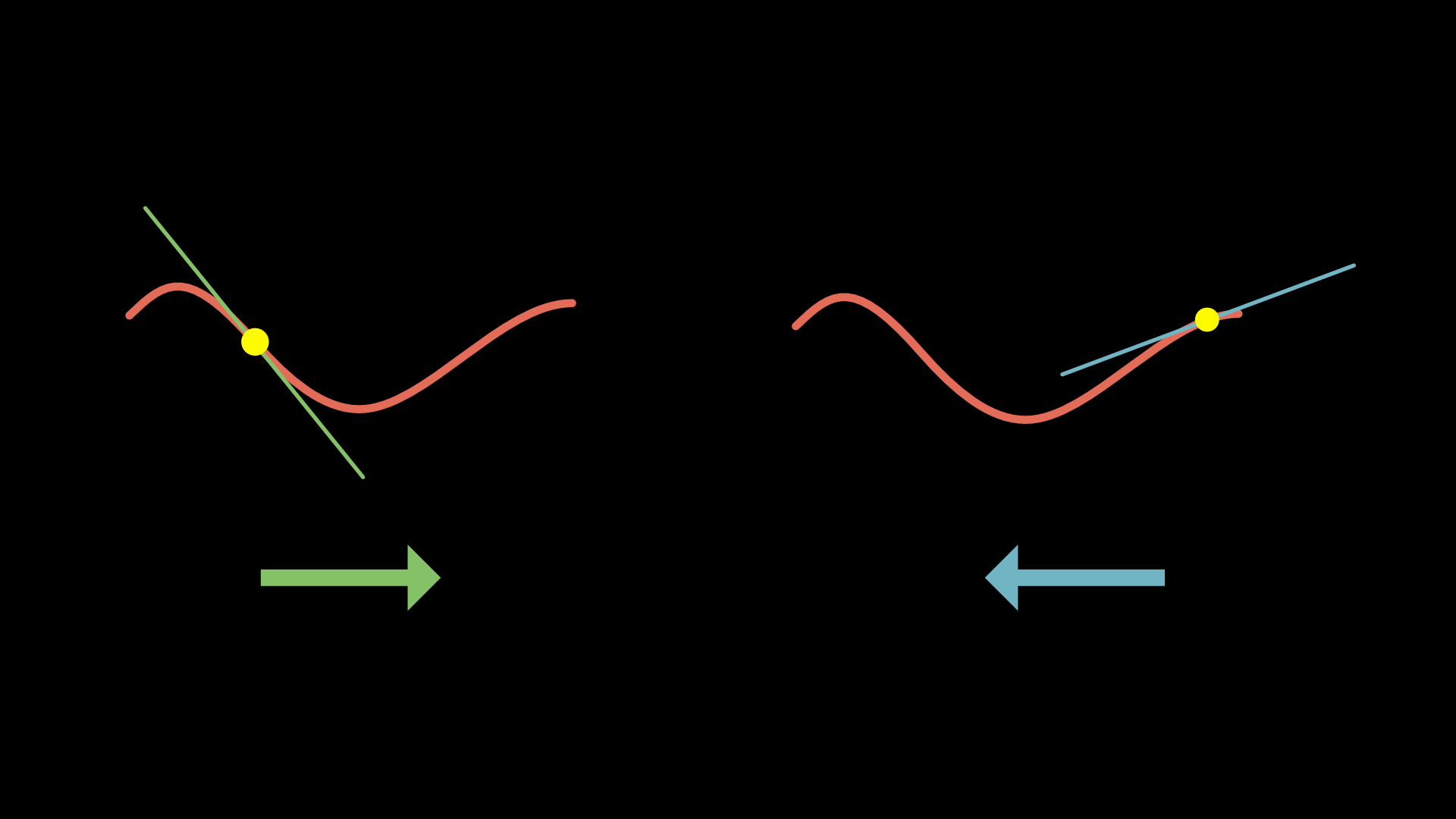
In this way, values propagate from one layer to the next, entirely determined by the weights and biases. The brightest of the ten neurons in the output layer is the network’s choice for which digit the image represents.

TESTED ON MNIST DATA AND MRI DATA : PICTURES HERE

fundamentally, the learning just comes down to finding the minimum of a specific function.

What ends up happening is that you determine the “cost” of the training where you compare the difference in observed output vs desired output. You adjust this cost function with the sum of squares of the differences between each output and adjust.

figure out that minimum explicitly by solving for when the slope is zero. However, that’s not always feasible for really complicated functions. Specifically, find the slope of the function where you are. If the slope is negative, shift to the right. If the slope is positive, shift to the left.



Notice, even for this simplified single-input cost function, there are many possible valleys you might land in. It depends on which random input you start at, and there’s no guarantee that the local minimum you land in will be the smallest possible value for the cost function.

Also, notice how if you make your step sizes proportional to the slope itself when the slope flattens out towards a minimum, your steps will get smaller and smaller, and that keeps you from overshooting.

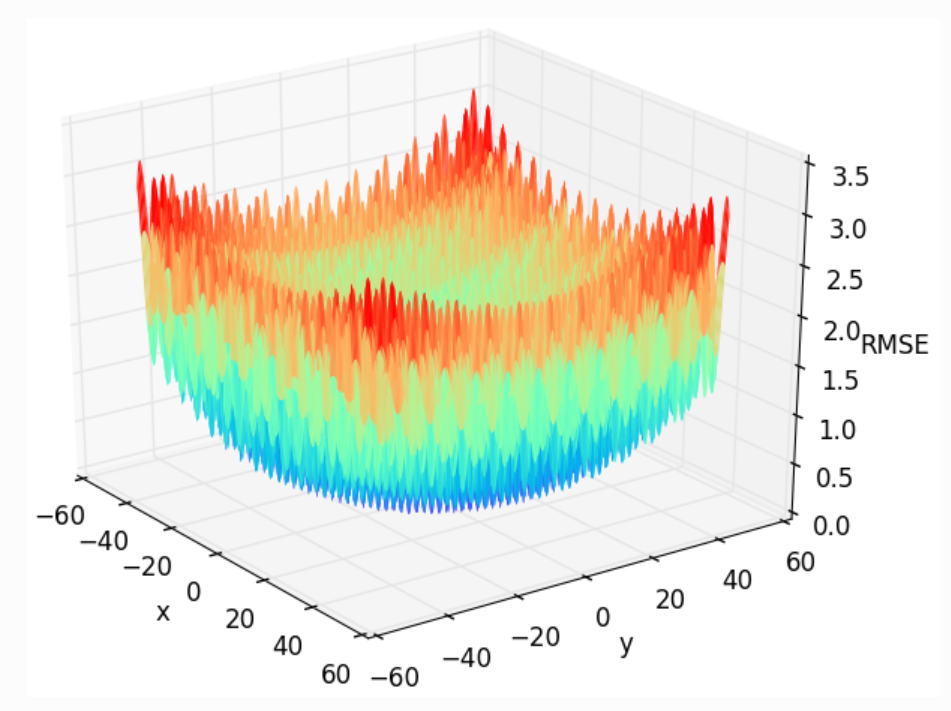
—------------> MORE ON THIS LATER IN FINDING OPTIMIZATION ALGORITHMS

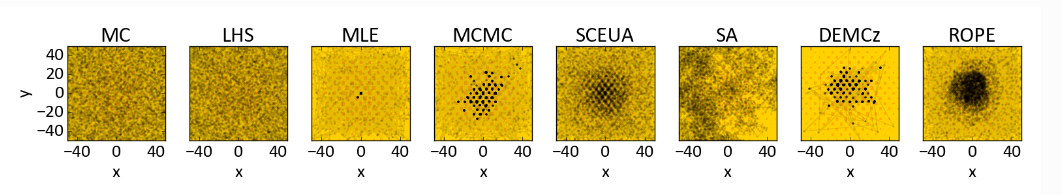
### Beyond Slope: Using The Gradient

In this higher-dimensional space, it doesn’t really make sense to talk about the “slope” as a single number. Instead, we need to use a vector to represent the direction of steepest ascent.

Your training propagation simply adjusts the weights and biases until the gradient find the global minimum.

HOW TO ANALYZE: G FUNCTION





describe how backpropogation works and leads into more advanced systems than this

* what is AI
* CNN vs other types -> Spacial, Neural, others - connection to AI - associative memory and chipset. introduce future/more advanced applications here. What data can you analyze/abstract/augment/and create?
* gradient descent -> math slide -> IN DEPTH
* relation and applications
* other things that affect the efficiency and usage (power spectral Density)
* My code environment setup (Docker, npm, nvm, STACK’s, deployment)

describe gradient descent optimization and the mathematical representation, with graphs, that illustrate the important components alongside training, backpropagation, and future applications.

**pooling types** (PICTURES)

Pooling Pooling is a non-linear transformation that permits to summarize the output of a net at a certain location with a single value. This single value is obtained from the statistic of the neighboring outputs which makes the feature descriptions more robust and invariant to small translations of the input data [27, 53]. The pooling layer progressively cuts down the dimensionality of the input, consequently reduces the demand of memory for storing the parameters, and improves the statistical efficiency [3, 16-18, 25, 28-30, 41, 53-54]. In deep networks, over-fitting is also controlled with the application of the pooling layers [16, 54]. Numerous application fields of CNN such as visual recognition, tracking, object detection, and face recognition have been used pooling operation to create invariance to small shifts and distortions of the input data. Average, max, fractional-max out, stochastic pooling and mixed pooling are some popular pooling regimes which have been used in numerous variant of CNN to solve the problems related to computer vision. Not only CNN, the Scale Invariant Feature Transform (SIFT) and Histogram of Oriented Gradient (HOG) methods of feature extraction also utilize pooling regimes to design a robust object detection system that can firmly recognize the objects in clutter and occlusion [6, 32]. In general, choice of pooling regime matters a lot in solving the computer vision problem as its foremost objective is to convert the joint feature representation (mapped by convolution) into useable one. Thus, the pooling operation plays a significant role in various computer vision architectures. Pooling operation also reduces computational cost as it cut down the resolution of feature map while preserving the useful features required for the task to perform. In short, the results desired from pooling operation are compact representation, robustness to noise and clutter, and invariance to shift, skew or lightening condition. Many research communities are working in the direction of development of advanced pooling mechanisms to make efficient use of these pivotal features of pooling regimes. The motivation behind this study is to appraise the readers from the regular advancements in the development of pooling regimes. Additionally, the survey provides a common platform to discuss all the popular pooling schemes with their pros and cons.

create a table and diagram showing the pros and cons of the pooling techniques? how they work? Future applications and augmentations of them to make it better?

* types , how, why they work, which is better, future applications
* results and conclusions for various data sets
* training vs non-trained networks

Convolutional neural networks (CNN) is a contemporary technique for computer vision applications, where pooling implies as an integral part of the deep CNN. Besides, pooling provides the ability to learn invariant features and also acts as a regularizer to further reduce the problem of overfitting. Additionally, the pooling techniques significantly reduce the computational cost and training time of networks which are equally important to consider.

**Noise Injection** (PICTURES)

* types and outcomes for relative data sets
* untrained occlusion noise on MRI scan - from laptop
* effects on training - occlusion and other types
* training algorithms? - effects of epoch adjustment?
* kernel size?
* pooling size?

## **Gradient noise**

* They show that adding this noise makes networks more robust to poor initialization and helps training particularly deep and complex networks. They suspect that the added noise gives the model more chances to escape and find new local minima, which are more frequent for deeper models.

**feature classifications**  (PICTURES)

* <https://biomedical-engineering-online.biomedcentral.com/articles/10.1186/s12938-022-01022-6>
* what, how , why , applications

# **Gradient Centralization for Better Training Performance**

* <https://keras.io/examples/vision/gradient_centralization/>

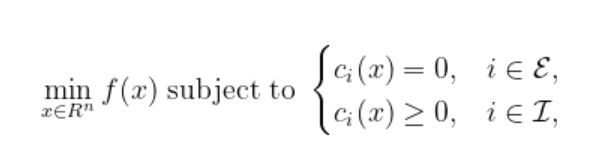
—-------------------------I ALSO ANALYZED OPTIMIZATION ALGORITHMS-------------------------

* gradient algorithm control by hand -> testing pictures from script and then actuals - > with MY algorithm noise injected function data (GRADIENT ALGORITHM FOR MRI IMAGE CLASSIFICATION SPICIFICALLY - ADDITIONALLY GOAL IS TO ANALYZE THE DIFFERENT APPROACHES ONE MAY TAKE IN THE CONTRUCTION OF SUCH A SYSTEM )
  + <https://www.ruder.io/optimizing-gradient-descent/>
  + <https://towardsdatascience.com/complete-step-by-step-gradient-descent-algorithm-from-scratch-acba013e8420>
  + <https://www.3blue1brown.com/lessons/gradient-descent>

What is Optimization?

If you’ve been studying machine learning long enough, you’ve probably heard terms such as SGD or Adam. They are two of many optimization algorithms. Optimization algorithms are the heart of machine learning which are responsible for the intricate work of machine learning models to learn from data. It turns out that optimization has been around for a long time, even outside of the machine learning realm.

Mathematically speaking, optimization is the process of maximizing or minimizing an objective function f(x) by searching for the appropriate variables x subject to some constraints cᵢ, which could be written compactly as follows.



where ℰ and ℐ are sets of indices for equality and inequality constraints, respectively.

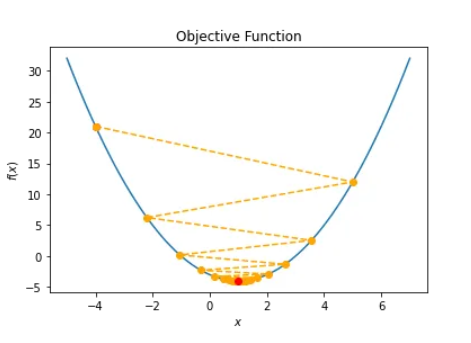
Now, how do we solve the **min** function? Thanks to calculus, we have a tool called *gradient*.

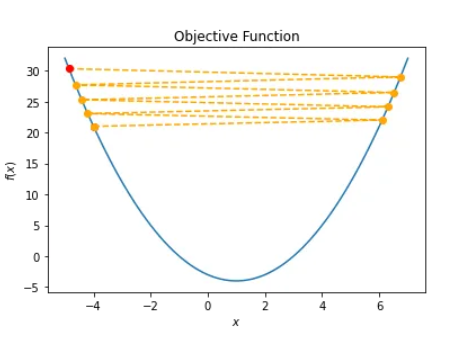
We could quantize the smooth path of the ball into tiny steps. At *k*-th step, we will have two quantities: the step length *αₖ* and the direction *pₖ*.

define *learning path* during the optimization process. What we mean by learning path is just points *x* after each descent step.

Define a simple gradient descent algorithm as follows. For every point *xₖ* at the beginning of step *k*, we maintain the step length *αₖ* constant and set the direction *pₖ* to be the negative of gradient value (steepest descent at *xₖ*). We take steps using the formula







The first scenario converges like a charm. Even though the step length is constant, the direction is decreasing towards zero and hence results in a convergence.

The second scenario also converges even though the learning path is oscillating around the solution due to the big step length.

The third scenario moves towards the solution. However, the step length is so small so that the number of iterations is maxed out. Increasing max\_iter will solve the issue even though it will take much longer to arrive at the solution.

The fourth scenario diverges due to the big step length. Here, we set max\_iter = 8 to make the visualization more pleasing.

A better strategy? Line search

line search strategy, the algorithm chooses a direction *pₖ* (could be as simple as the steepest descent -∇*f(x)*) and searches along this direction from the current iterate *xₖ*

**

*As mentioned before, by solving this exactly, we would derive the maximum benefit from the direction pₖ, but an exact minimization may be expensive and is usually unnecessary.*

*HENCE ONE OF THE ISSUES WITH OPTIMIZATION!!!*

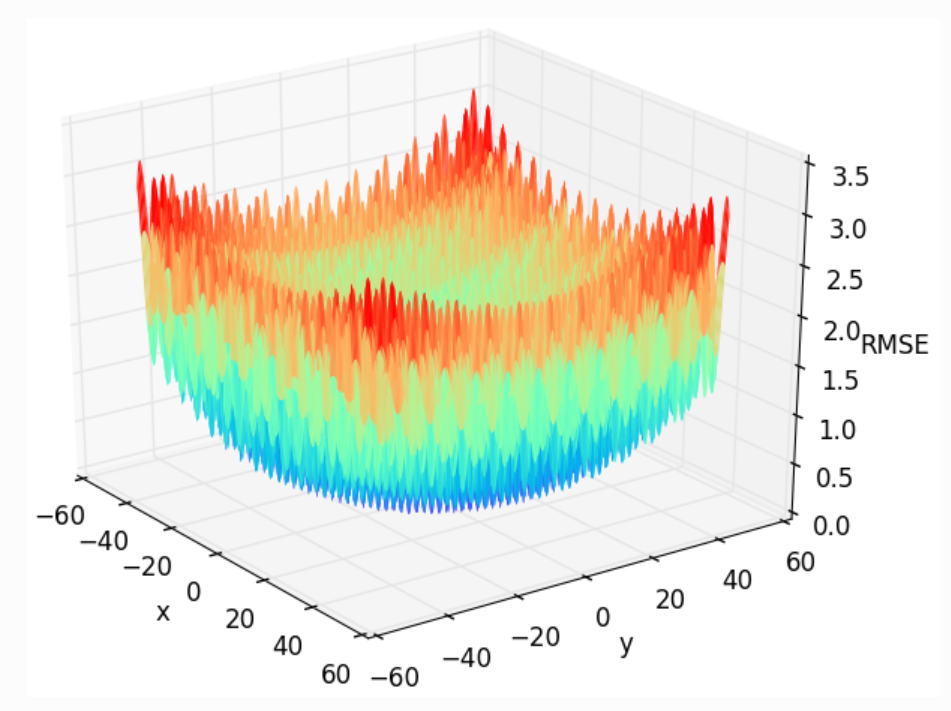
An approximation technique:  
*Armijo Condition*:



—> THE HARDER WAY →

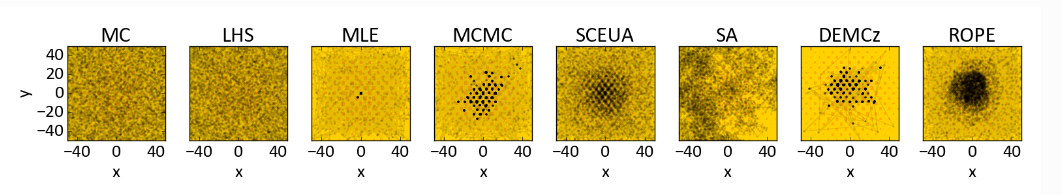
For testing optimization: Griewank function. Why use this function?



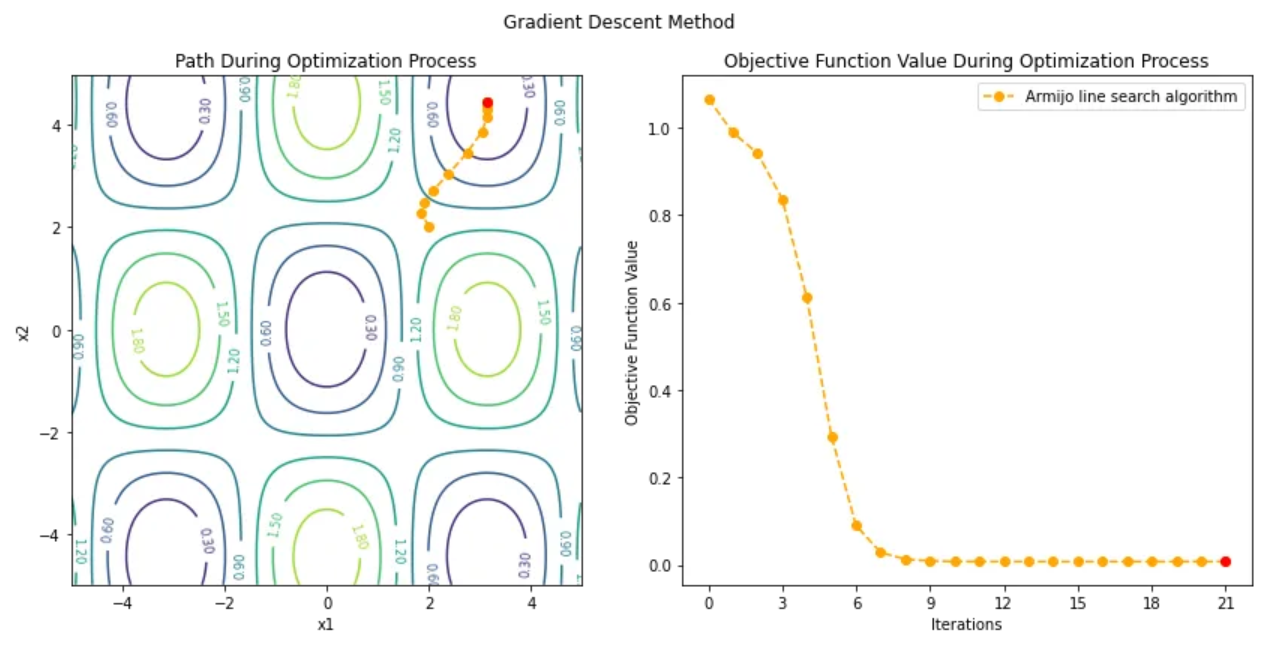


HAS MANY LOCAL MAX AND MIX: GOOD FOR TESTING

Diferent algorithms for G function: MLE seems most optimal: utilize this?



path analysis:



x0=[2,1] UNSUCCESSFULL PATH

Initial condition: y = 1.0669, x = [2 2]

Iteration: 1 y = 0.9886, x = [1.8572 2.2897], gradient = 0.2035

Iteration: 2 y = 0.9414, x = [1.9025 2.488 ], gradient = 0.2858

Iteration: 3 y = 0.8372, x = [2.0788 2.713 ], gradient = 0.4378

Iteration: 4 y = 0.6117, x = [2.3753 3.035 ], gradient = 0.5682

Iteration: 5 y = 0.2941, x = [2.7514 3.461 ], gradient = 0.5082

Iteration: 6 y = 0.0894, x = [3.0423 3.8777], gradient = 0.2863

Iteration: 7 y = 0.0282, x = [3.1321 4.1495], gradient = 0.1438

Iteration: 8 y = 0.0127, x = [3.1398 4.2931], gradient = 0.0726

Iteration: 9 y = 0.0087, x = [3.1400 4.3657], gradient = 0.0364

Iteration: 10 y = 0.0077, x = [3.1400 4.4021], gradient = 0.0182

Iteration: 11 y = 0.0075, x = [3.1400 4.4203], gradient = 0.0091

Iteration: 12 y = 0.0074, x = [3.1400 4.4294], gradient = 0.0045

Iteration: 13 y = 0.0074, x = [3.1400 4.4339], gradient = 0.0023

Iteration: 14 y = 0.0074, x = [3.1400 4.4362], gradient = 0.0011

Iteration: 15 y = 0.0074, x = [3.1400 4.4373], gradient = 0.0006

Iteration: 16 y = 0.0074, x = [3.1400 4.4379], gradient = 0.0003

Iteration: 17 y = 0.0074, x = [3.1400 4.4382], gradient = 0.0001

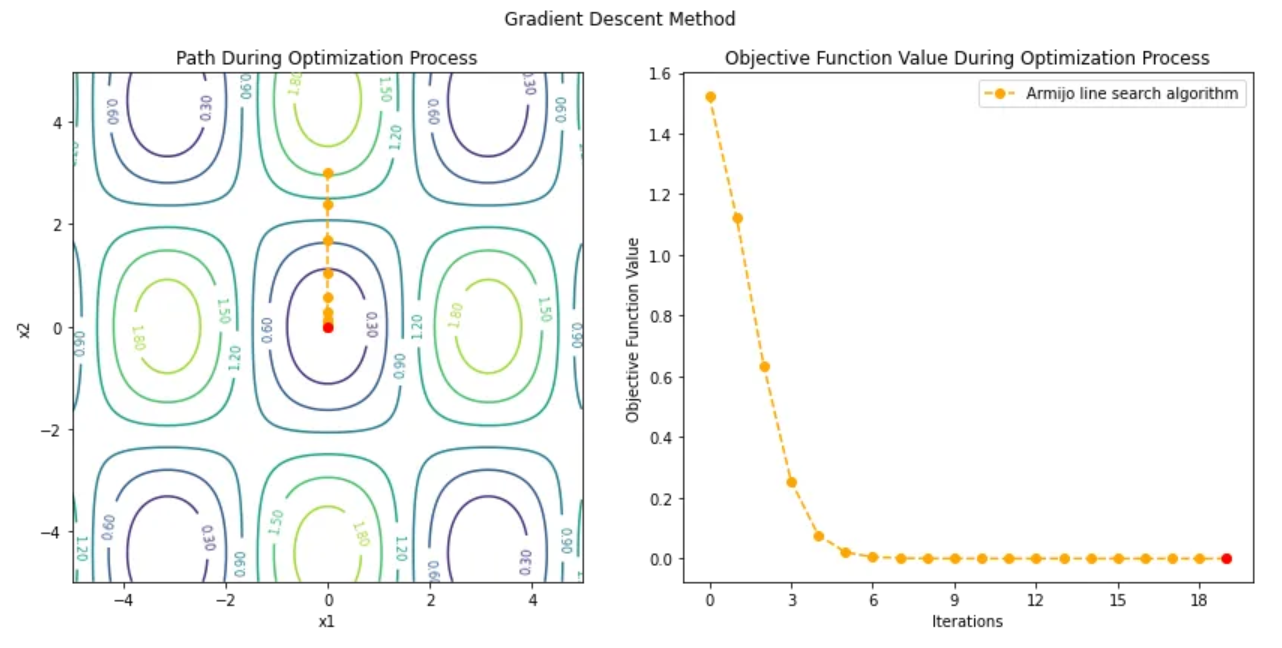
Iteration: 18 y = 0.0074, x = [3.1400 4.4383], gradient = 0.0001

Iteration: 19 y = 0.0074, x = [3.1400 4.4384], gradient = 0.0000

Iteration: 20 y = 0.0074, x = [3.1400 4.4384], gradient = 0.0000

Iteration: 21 y = 0.0074, x = [3.1400 4.4384], gradient = 0.0000

Solution: y = 0.0074, x = [3.1400 4.4384]



x0 = [0,3] SUCCESSFUL PATH

Initial condition: y = 1.5254, x = [0 3]

Iteration: 1 y = 1.1245, x = [0.0000 2.3959], gradient = 0.7029

Iteration: 2 y = 0.6356, x = [0.0000 1.6929], gradient = 0.6591

Iteration: 3 y = 0.2558, x = [0.0000 1.0338], gradient = 0.4726

Iteration: 4 y = 0.0778, x = [0.0000 0.5612], gradient = 0.2736

Iteration: 5 y = 0.0206, x = [0.0000 0.2876], gradient = 0.1430

Iteration: 6 y = 0.0052, x = [0.0000 0.1447], gradient = 0.0723

Iteration: 7 y = 0.0013, x = [0.0000 0.0724], gradient = 0.0362

Iteration: 8 y = 0.0003, x = [0.0000 0.0362], gradient = 0.0181

Iteration: 9 y = 0.0001, x = [0.0000 0.0181], gradient = 0.0090

Iteration: 10 y = 0.0000, x = [0.0000 0.0090], gradient = 0.0045

Iteration: 11 y = 0.0000, x = [0.0000 0.0045], gradient = 0.0023

Iteration: 12 y = 0.0000, x = [0.0000 0.0023], gradient = 0.0011

Iteration: 13 y = 0.0000, x = [0.0000 0.0011], gradient = 0.0006

Iteration: 14 y = 0.0000, x = [0.0000 0.0006], gradient = 0.0003

Iteration: 15 y = 0.0000, x = [0.0000 0.0003], gradient = 0.0001

Iteration: 16 y = 0.0000, x = [0.0000 0.0001], gradient = 0.0001

Iteration: 17 y = 0.0000, x = [0.0000 0.0001], gradient = 0.0000

Iteration: 18 y = 0.0000, x = [0.0000 0.0000], gradient = 0.0000

Iteration: 19 y = 0.0000, x = [0.0000 0.0000], gradient = 0.0000

Solution: y = 0.0000, x = [0.0000 0.0000]

The easy implementation assumes a constant learning rate, whereas a harder one searches for the learning rate using Armijo line search. The algorithm itself is the most basic and has many variations and implementations depending on the nature of the objective function. Like many other optimization algorithms, the Gradient Descent algorithm may be trapped in a local minimum.

—------------------------------------------------------------------------------------------------------------------------

Related experiments

* how and why it was conducted -> results and conclusions

Relative originality (PICTURES)

* large data sets - my data access
* my visualization automation mechanisms
* optimization processing and AI decision-making

Future applications

* spinglass applications and similarities to NNs
* connecting to very large training sets - MRI mapping
* utilization in various fields - chemistry compound mapping, etc.
* associative memory
* AI chipsets

**bonus material**

* fragmented research components
* code documentation and publication
* From my paper: Introduction to weight quantization: chatgpt-2 how LLTM works in chatgpt → future architectures with diagrams?

**Japan Pictures Slide**

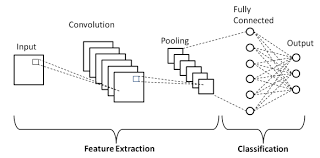
—-------------------------------------------SLIDES—----------------------------------------------------

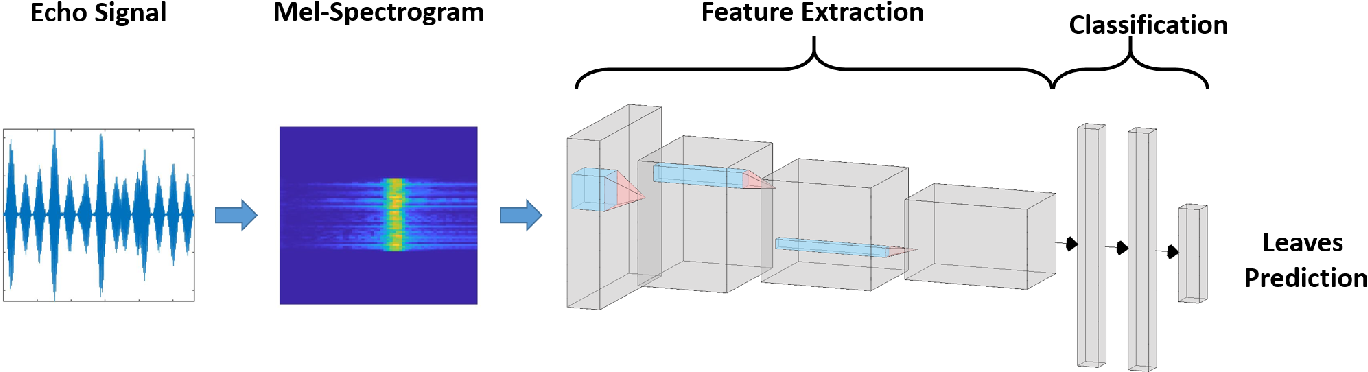
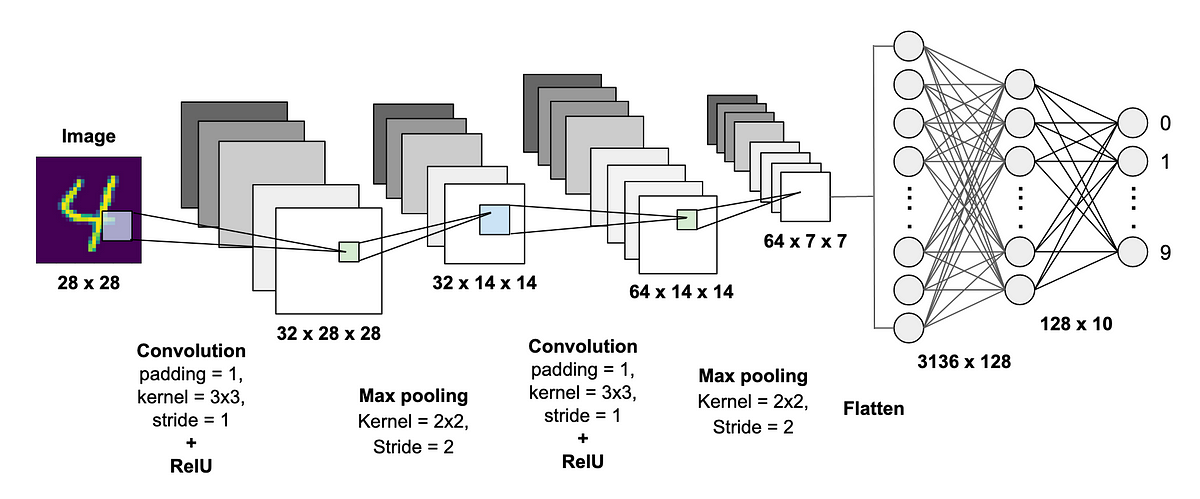
Use Huang template

ask chat-gpt to build a slide with note annotation?

INTRO CONTENT WILL BE ABOUT MY LAB AND THEN EXPLAIN AI => CNN -> 3blu1brown content -> lead into each layer analysis

NN/s can be used with any form of input: sound, image, data to make predictions and interpolate/produce reasonable output.





—---------------------------------------------------------------------------------------------

Report:

Timeline -> construct working NN -> access to DB’s -> visualizer -> data is really big …

Appendix:

derphi0 = np.dot(gfk, pk)

phi\_a0 = f(xk + alpha0\*pk)

while not phi\_a0 <= phi0 + c1\*alpha0\*derphi0:

alpha0 = alpha0 \* rho

phi\_a0 = f(xk + alpha0\*pk)

return alpha0, phi\_a0

def Griewank(xs):

"""attempt to calculate the length required"""

d = len(xs)

sqrts = np.array([np.sqrt(i + 1) for i in range(d)])

cos\_terms = np.cos(xs / sqrts)

"""If it does not fit address length it will crash here with EXEC error"""

sigma = np.dot(xs, xs) / 4000

pi = np.prod(cos\_terms)

return 1 + sigma - pi

References

[1] Reddit, Machine Learning. Available:

https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama\_geoffrey\_hint

on/clyj4jv/

[2] Achille A., Soatto S., Information dropout: Learning optimal representations

through noisy computation, IEEE Transactions on Pattern Analysis and Machine

Intelligence, 2018, 2897-2905.

[3] Boureau Y.-L., Le Roux N., Bach F., Ponce J., LeCun Y., Ask the locals: multiway local pooling for image recognition, in Computer Vision (ICCV), 2011 IEEE

International Conference on, 2011, 2651-2658.

[4] Cai M., Shi Y., Liu J., Stochastic pooling maxout networks for low-resource

speech recognition, in Acoustics, Speech and Signal Processing (ICASSP), 2014

IEEE International Conference on, 2014, 3266-3270.

[5] Cheng Y., Zhao X., Cai R., Li Z., Huang K., Rui Y., Semi-Supervised Multimodal

Deep Learning for RGB-D Object Recognition, in IJCAI, 2016, 3345-3351.

[6] Dalal N., Triggs B., Histograms of oriented gradients for human detection, in

Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer

Society Conference on, 2005, 886-893.

[7] DeVries T., Taylor G. W., Improved regularization of convolutional neural

networks with cutout, arXiv preprint arXiv:1708.04552, 2017, 1-8.

[8] Donahue J., Jia Y., Vinyals O., Hoffman J., Zhang N., Tzeng E., Darrell T., Decaf:

A deep convolutional activation feature for generic visual recognition, in

International conference on machine learning, 2014, 647-655.

[9] Dumpala S. H., Chakraborty R., Kopparapu S. K., k-FFNN: A priori knowledge

infused Feed-forward Neural Networks, arXiv preprint arXiv:1704.07055, 2017,

1-9.

