Day - 25, oct 27, 2024.

→ **Non**-linearity in Neural Networks. Purposes of Nonlinearities

→ Activation functions → ReLu, ELu, leaky ReLu, Sigmoid, tanh

→ Optimization part : SGD, Adam, AdaGard, AdaDelta

→ Backpropagation via Flow Graphs and Chain Rule Rule

→ Flow Graphs, Recursive Application

→ Regularized Linear Regression

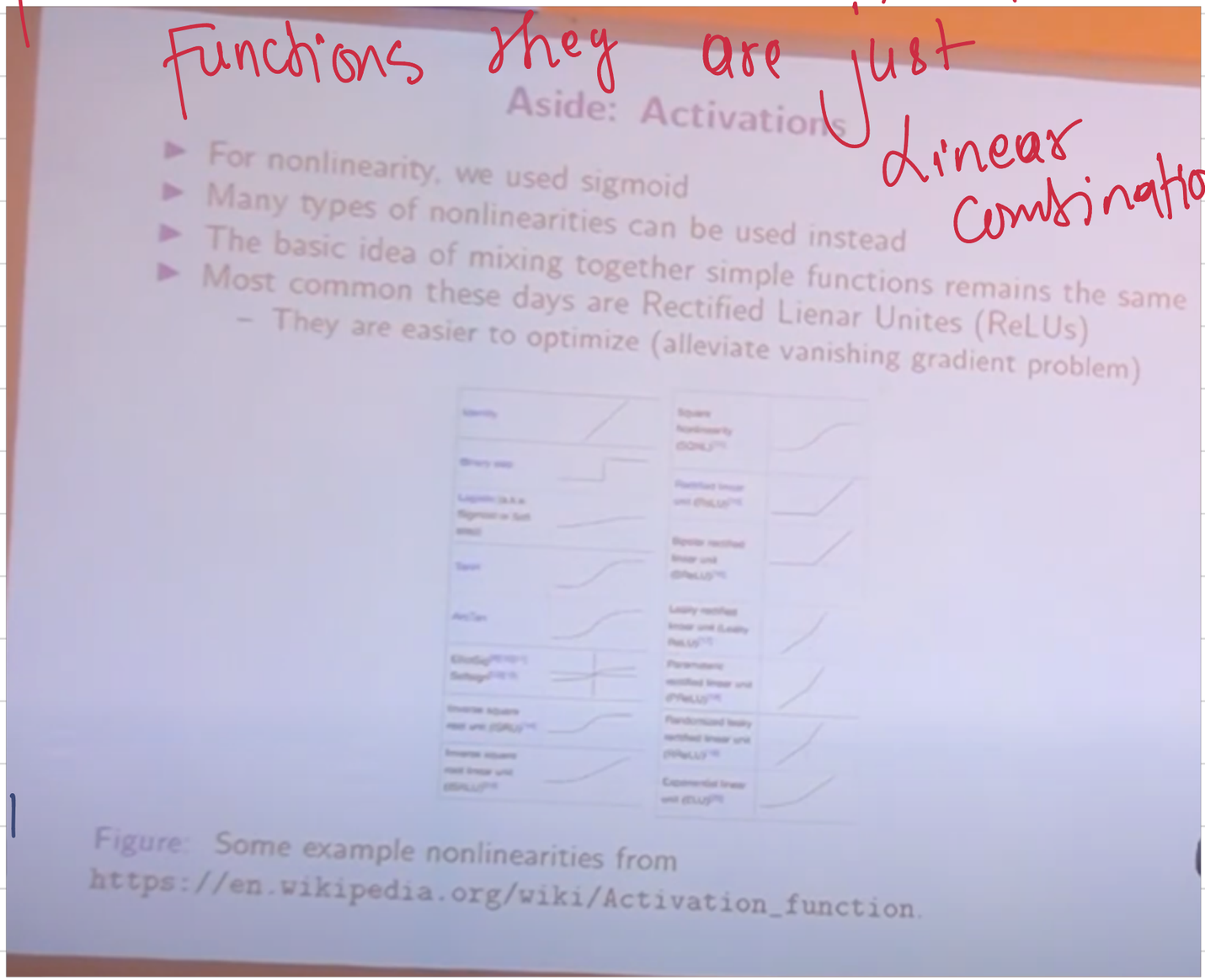
No Matter How Deep Neural Networks without Activation Functions they are just Linear Combination

① Sigmoid: Sequeezes the input & weights range [0, 1]

② tanh: [-1, 1]

③ ReLu: [0, ∞) → Simplicity, effectiveness, mitigate gradient vanishing problem

④ Leaky ReLu: Allow Smaller negative values [-ve, +ve].



⑤ ELU (Exponential Linear Unit):

ReLU + Leaky ReLU → Smooth transitions for negative Inputs.

⑥ SoftMax Function: used in the output Layer for Multi-class problem

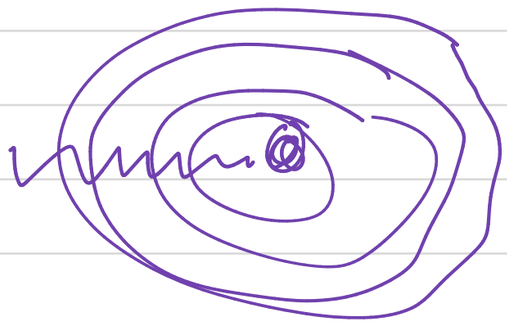
Why Non-linearity ?

→ Because it introduces Complexity into network, allowing it to capture intricate features, make more accurate predictions, enable the network to learn Hierarchical representations of data.

Optimizers: Reduce the loss of the loss function, find the minima values, optimal parameters to minimize loss functions

SGD → traditional iterative, takes step in the direction of steepest descent to reach the minimum of the loss function

Adam → adapts learning rate for each parameter, faster convergence & better performance, widely used



AdaGard → adjusts the learning rate for each parameter based on the history of gradients, allowing for more aggressive updates for less frequent parameters.

Adadelta → A variant of AdaGord better version overcome shortcomings such as diminishing learning rate.

Speed Measurement in Common Scenario.

Adam > RMSprop > Adadelta > AdaGord > SGD

adjusts learning rate parameter

Backpropagation via flow charts

→ Backpropagation calculates the gradient of the loss function w.r.t. model's parameters and then updating the parameters to minimize the loss. It will be nightmare if we compute each one of them using only derivatives. So, we can use composition function → get solved by chain Rule.

Chain Rule allows us to differentiate Composite functions.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial w}$$

Chain Rule Revisited

- Chain rule for vector-valued transformations $x \xrightarrow{g} h \xrightarrow{f} y$.

$$D(f \circ g)(x) = Df(h)\, Dg(x)$$

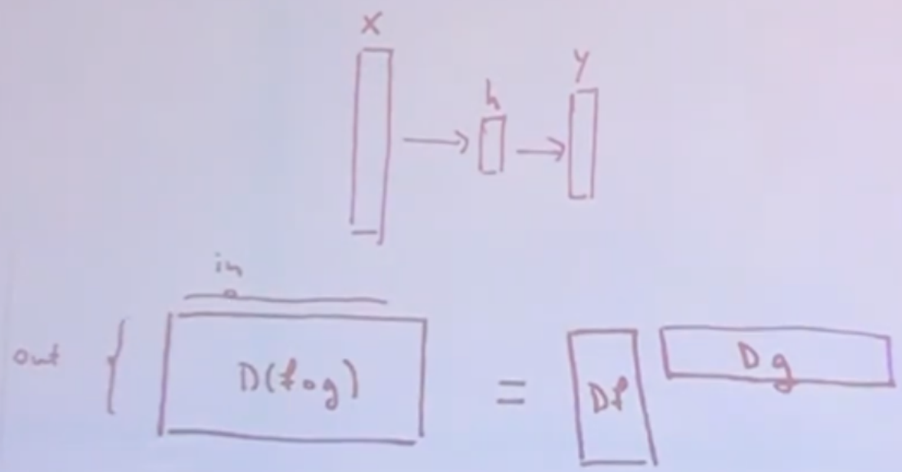


Figure: The first row shows the sequence of vector-valued transformations. The ij^{th} cell of the derivative matrices tells you how the i^{th} output changes when you perturb the j^{th} input slightly.

Optimization: Stochastic Gradient Descent

- Optimize by moving in a direction expected to decrease loss
- SGD: Estimated direction of gradient, after looking at small batch of data
- These days, other methods used too – Adam, AdaGrad, AdaDelta, ...

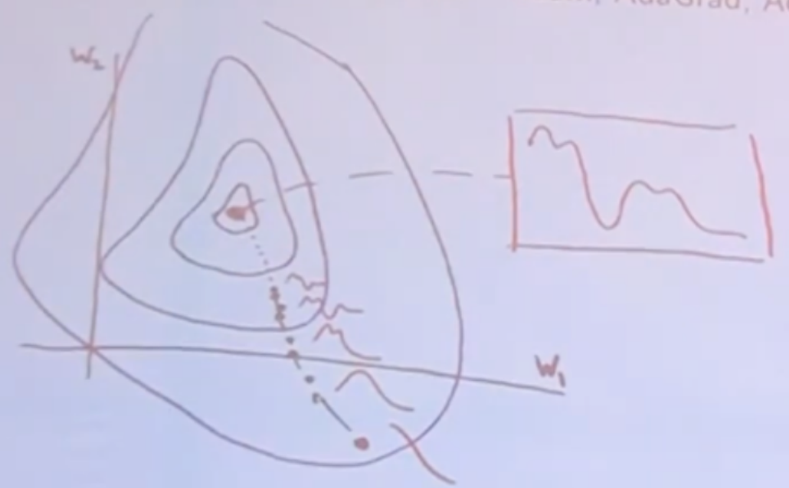


Figure: Each optimization step modifies model parameters to make the fitted function closer ot the true (local) minimizer of the loss.

See how chain Rule helps to solve gradient of loss functions (Composite functions) into simpler form. SGD iteratively steps into steepest direction to minimize the loss function.

Backpropagation via Flow Graphs

- For each iteration of SGD, we need to estimate the gradient of the loss with respect to the parameters
- When you perturb W_k a little, it changes all the downstream $h_k, h_{k+1}, \ldots, h_K$ up to the prediction, which uses that final representation
- There is a lot of composition going on here... courage alone won't help you take these derivatives
- Chain rule saves the day again!

Flow Graphs

- A flow graph shows how different vectors are computed from one another
  - Nodes are vectors
  - Edges are specific operations on input, which when combined with other incoming edges gives the output
- Crucial: Operations along each edge should be easily differentiable with respect to input

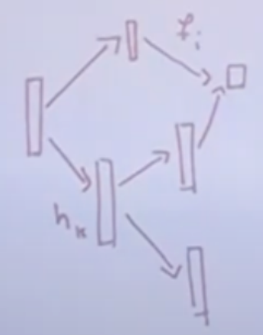


Figure: The main idea of a flow graph.

$$DF_i(h_k) = \sum \left(DF_i(h_j) \cdot D h_j(h_k) \right) \text{ for } j \in C(i)$$

$C(i) \rightarrow$ represents the set of immediate childre node i.

$h_k \rightarrow$ Sum of products, F_i - State that the derivative of f_i.

So, the solution is to break child Nodes first.

Chain Rule on Graphs

- When differentiating function f_i with respect to upstream h_k, the chain rule implies

$$Df_i(h_k) = \sum_{j \in C(i)} Df_i(h_j)\, Dh_j(h_i)$$

Break Complex into smaller problems!

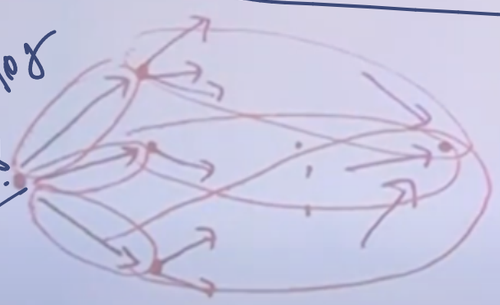
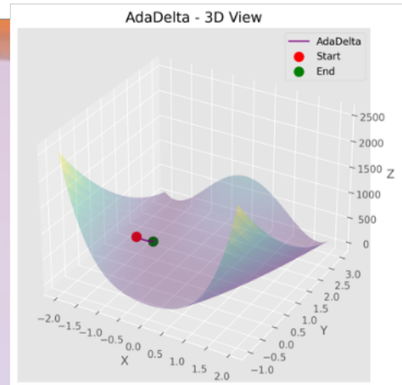
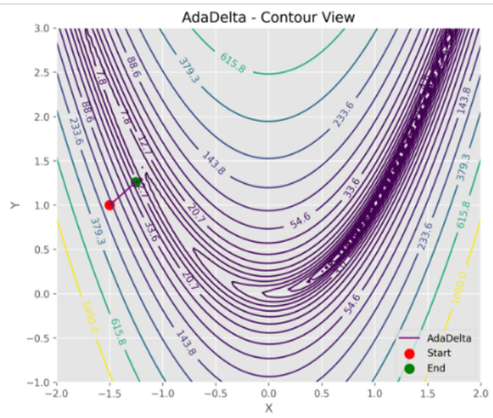
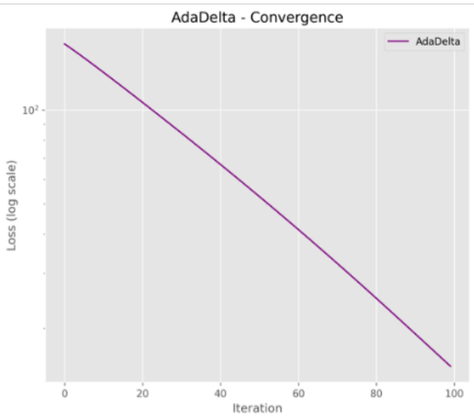
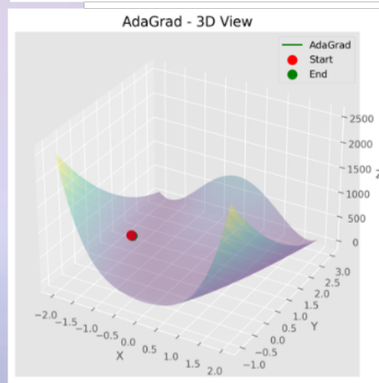


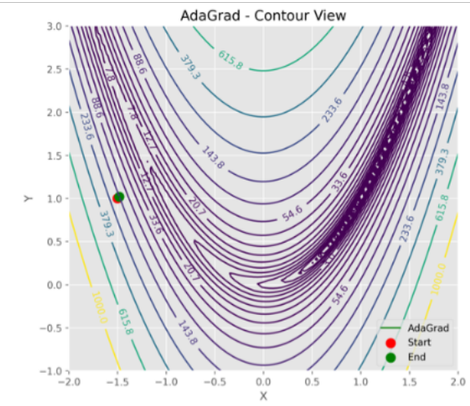
Figure: The graph version of the chain rule sums over the immediate children, referring recursively to similar gradients.

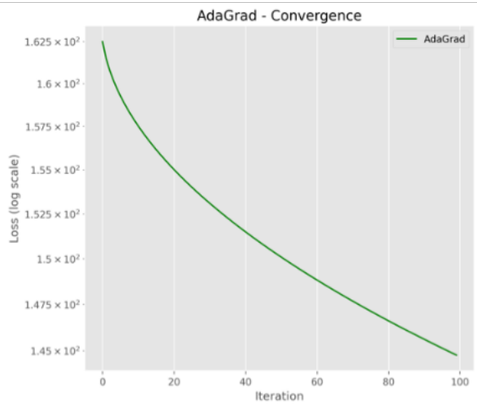


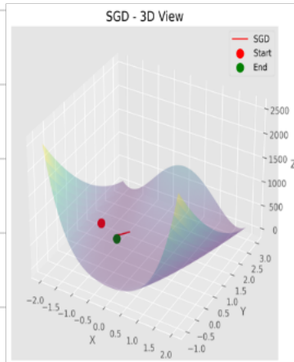


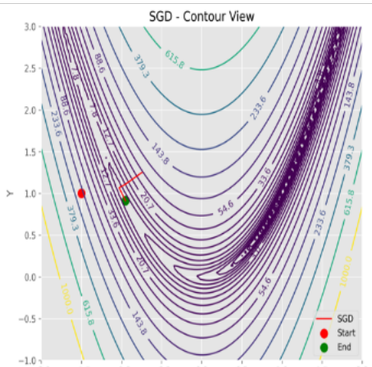


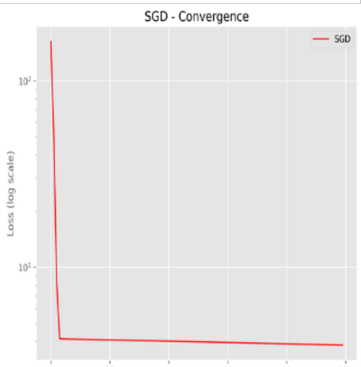


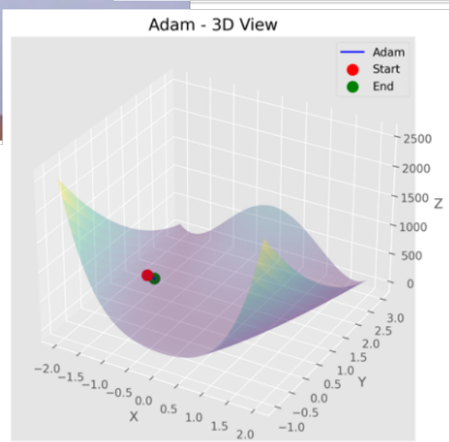


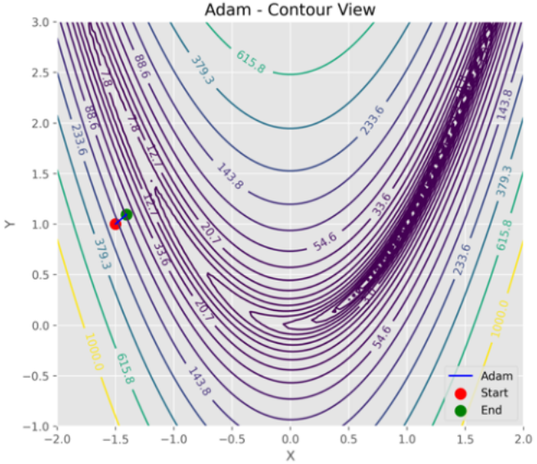


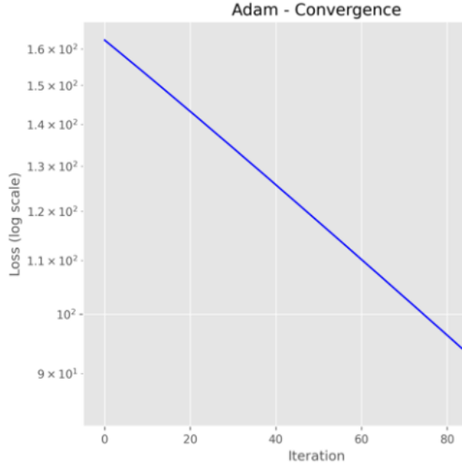












Chain Rule.

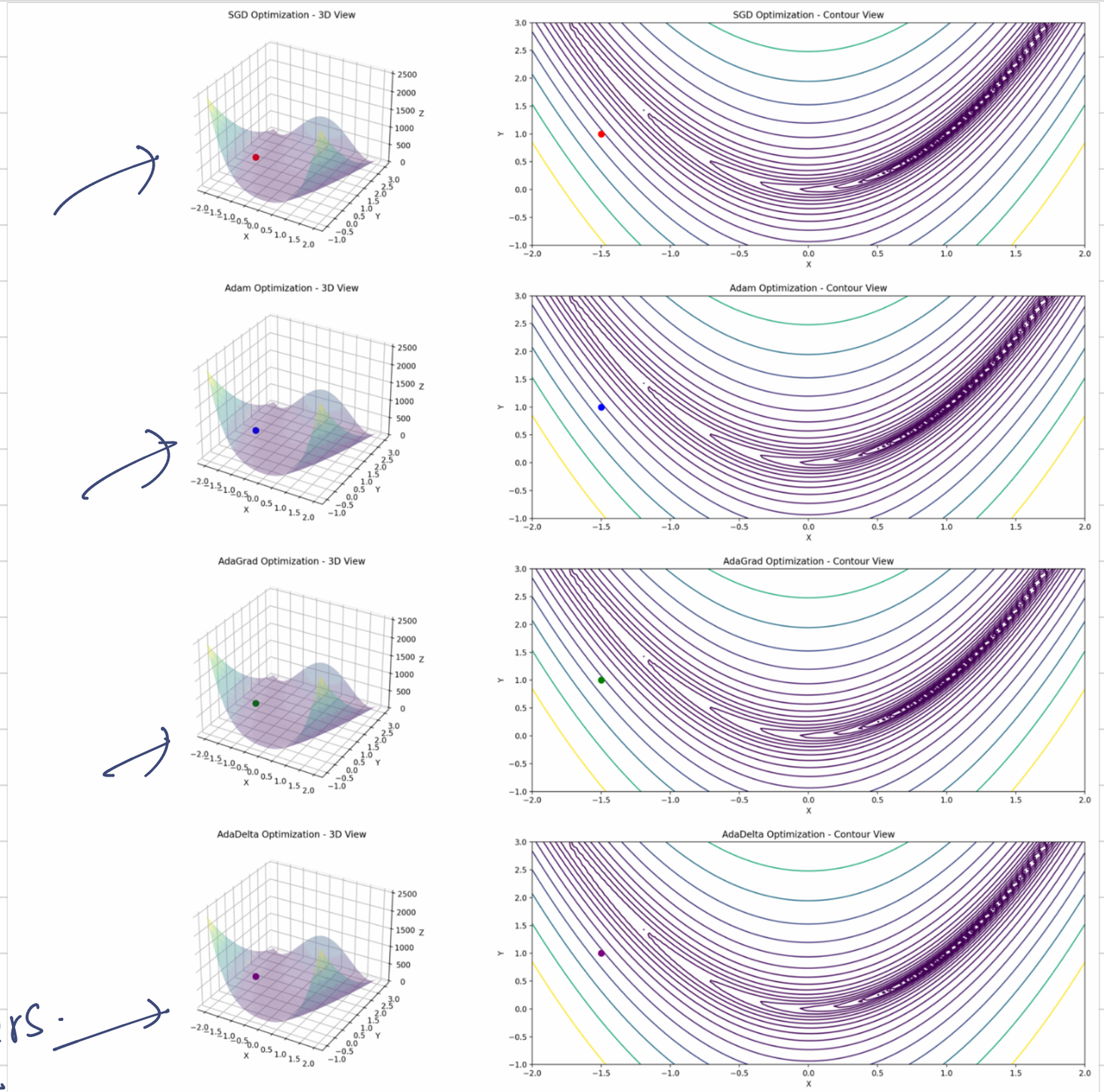
$$\frac{dL}{dw} = \frac{dL}{da} \cdot \frac{da}{dw}$$

→ SGD → AdaGard
Adam → AdaDelta

→ Back propagation via
→ Flow Graph

→ ReLu, Sigmoid, Softmax, tanh.

Figure showing optimizers. →



From Textbook - (Knowledge Representation) PL: Inference using Resolution

Resolution → powerful inference used in PL to prove or determine the validity of logical Arguments.

Resolution Rule

Given two clauses:

C1: $A \vee B$

C2: $\neg B \vee C$

C3: $A \vee C$ (infer a new clause)

So, if we have two clauses, one containing a literal and the other containing negation we can resolve the complementary literals and Combining the remaining literals.

KB: (Knowledge Base).

- $P \rightarrow Q$
- $Q \rightarrow R$
- $\neg R$

① Convert into CNF.

- $\neg P \vee Q$
- $\neg Q \vee R$
- $\neg R$

(2) Apply Resolution:

- Resolve $(\neg P \vee Q)$ and $(\neg Q \vee R)$: $\neg P \vee R$
  - Resolve $(\neg P \vee R)$ and $\neg R$: $\neg P$

Example:

$A \rightarrow B$
$B \rightarrow C$
A is true.

(1) Convert into CNF:

$A \rightarrow B$ becomes $\neg A \vee B$
$B \rightarrow C$ becomes $\neg B \vee C$

A remains A

Assume $\neg C$

- $\neg A \vee B$, $\neg B \vee C$, A and $\neg C$

Apply Resolution.

Resolve A with $\neg A \vee B$ to get B

" B with $\neg B \vee C$ to get C.

" C with $\neg C$ to get Contradiction

(1) Convert KB and Negation of theorem in CNF

- CNF is Conjunction of clauses ($\wedge$)
  - Each clause is a disjunction of literals ($\vee$)

KB: $p \rightarrow q$, $q \rightarrow r$, $\neg r$. $\rightarrow$ To prove $\neg p$:

① Convert Implications to Disjunctions:

- $P \rightarrow Q$ is equivalent to $\neg P \vee Q$

$P \rightarrow$ Cat is animal.

$Q \rightarrow$ Dog is animal

$\neg$(Cat or Dog is bird) $\rightarrow$ True.

Neither Cat nor Dog is bird.

So, $Q \rightarrow R$ is $\neg Q \vee R$

So, Negate Theorem $\neg p$ is p $\neg(\neg p) \rightarrow p$.

C1: $\neg p \vee Q$

C2: $\neg Q \vee R$.

C3: p

Resolution steps:

$P \to Q \quad Q \to R \;,\; \neg R.$

(1) Resolve $P \to \neg P \vee Q$

$P(\neg P \vee Q) \Rightarrow Q.$ (5)

(2) Resolve $Q \to \neg Q \vee R$

$Q(\neg Q \vee R) \Rightarrow R$ (6)

(3) Resolve R with $\neg R$ $\longrightarrow$ empty clause ()

$R(\neg R)$

(4) $Q \to \neg P \vee Q$

$\to P$ (7)

(5) $R \to \neg Q \vee R.$

$\to Q$ (8)