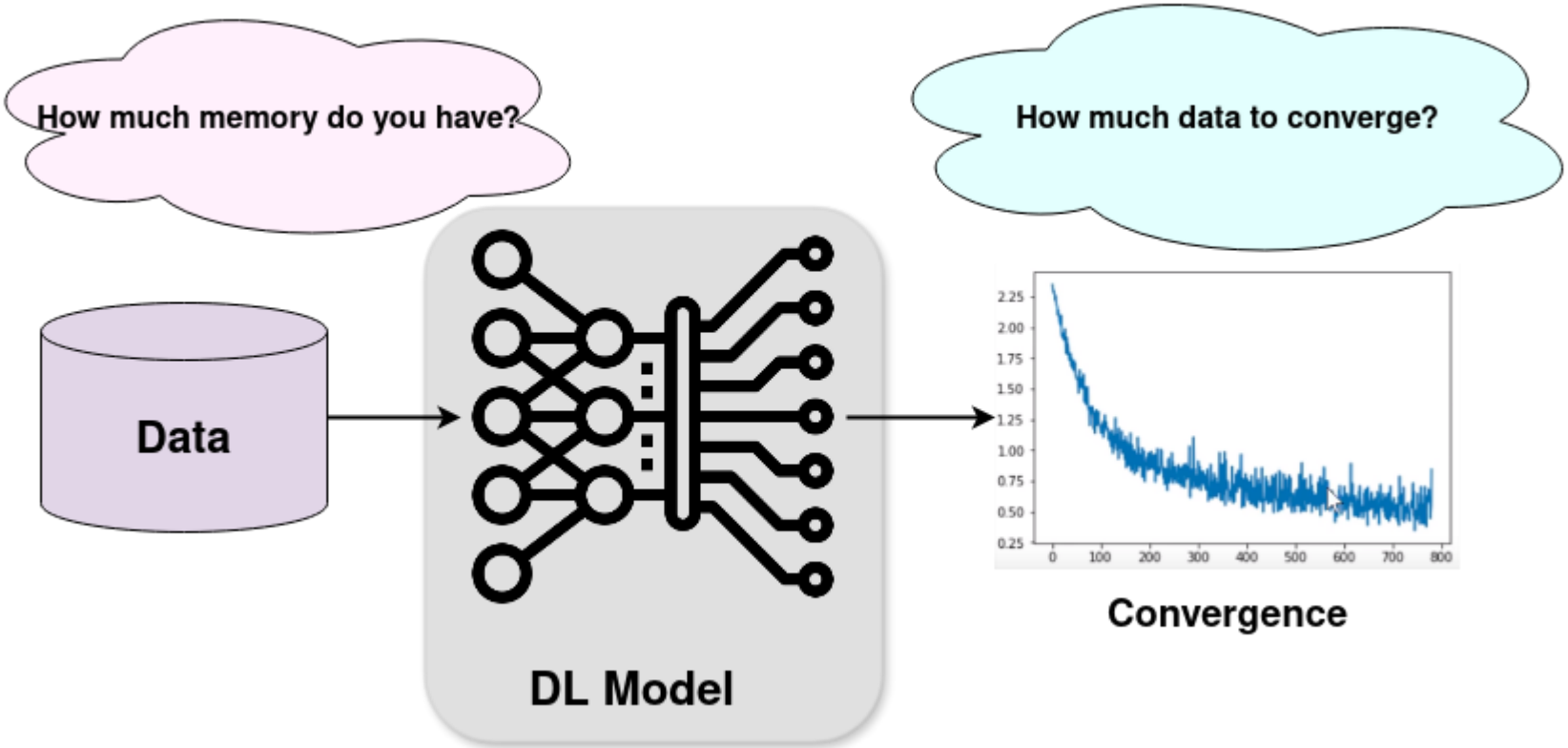
Deep Learning

Optimization

Training the model



Important Terms

Epoch – The number of times the algorithm runs on the whole training dataset.

Sample – A single row of a dataset

Batch – It denotes the number of samples to be taken to for updating the model parameters in one iteration

Iterations - The number of batches required to complete one epoch

Learning rate – It is a parameter that provides the model a scale of how much model weights should be updated

Cost Function/Loss Function – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value

Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

Learning Strategies

1. Stochastic or Sequential Learning:

   Each data row is considered as batch, hyperparameters(weights and bias) updated after each row of dataset

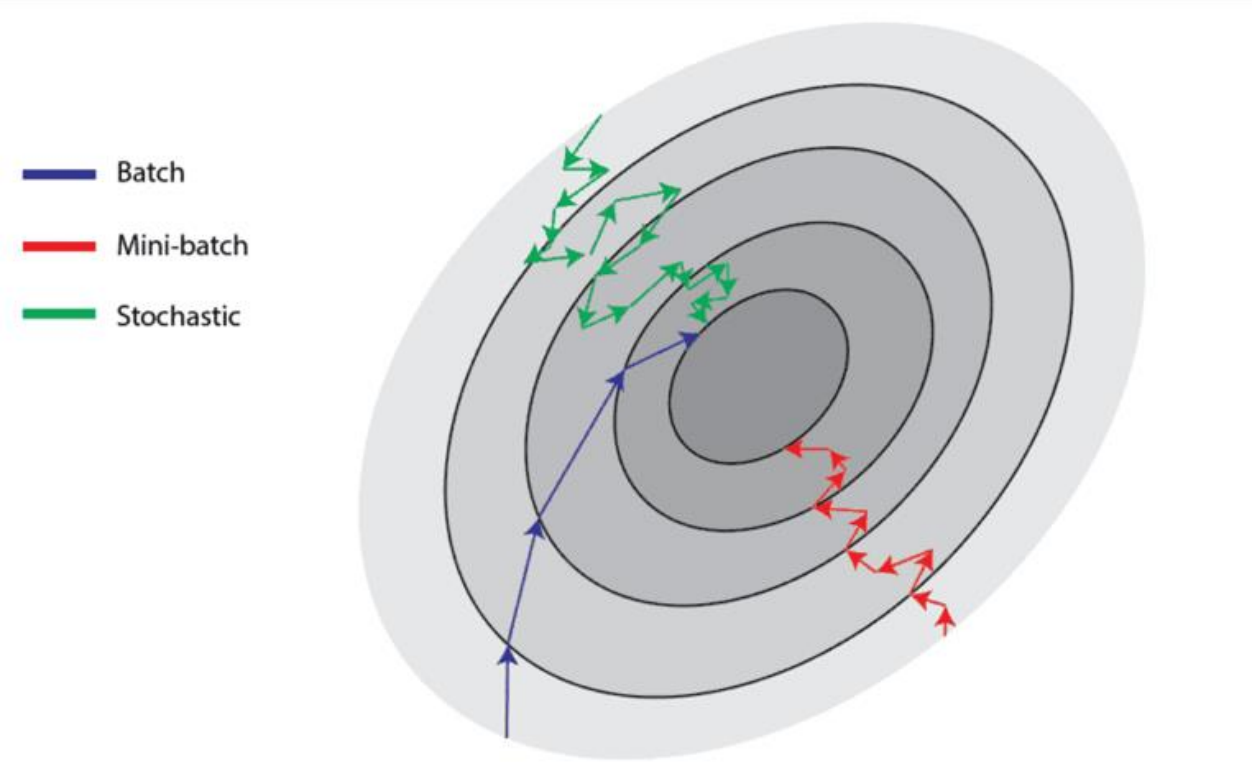
1. Batch Learning

   Entire data is considered as batch

1. Mini Batch Learning

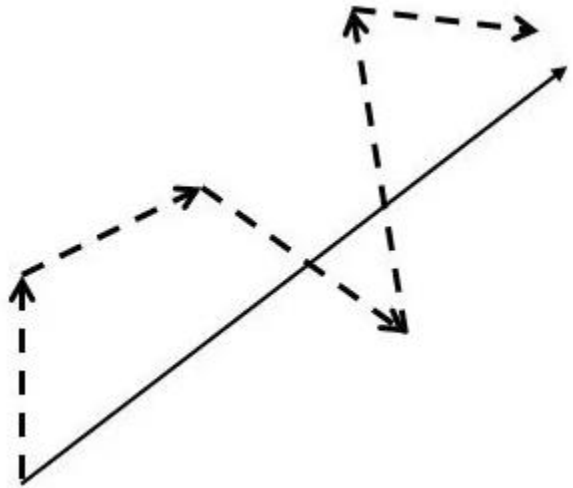
   Dividing the dataset into chunks and learning on chunks

Learning Strategies



Batch Learning & Sequential Learning

- Learning on each sample may lead to slower convergence
- The straight path can be achieved by Batch and dotted path by Sequential learning



Sequential Learning

Pros:

- Adaptable to changing patterns in data
- Well-suited for incremental learning
- Allows for continuous learning

Cons:

- Requires frequent updation hence computationally intensive
- Difficult to implement especially for online learning

Batch Learning

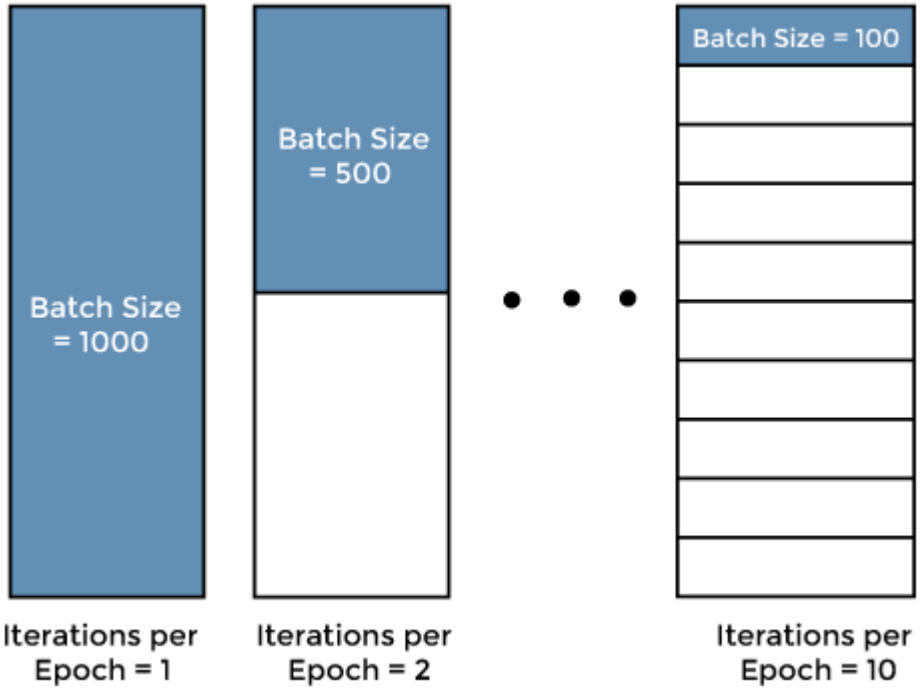
Pros:

- Computationally efficient for large datasets
- Easier to implement on offline learning
- Parallel processing can be done

Cons:

- Require retraining on entire dataset to adapt changes
- Might not be suitable for real-time or online learning

Epoch vs iteration per Batch



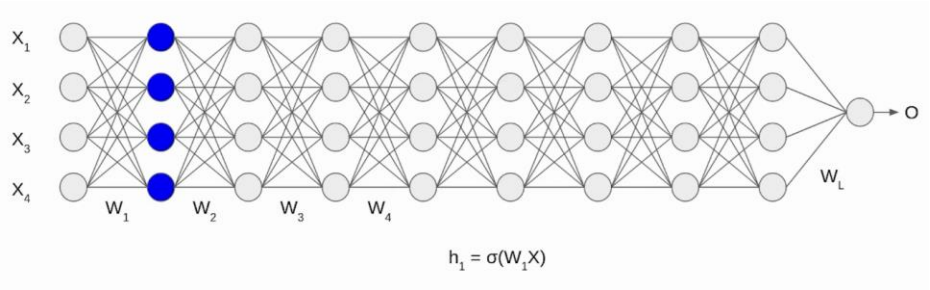
Batch Normalization

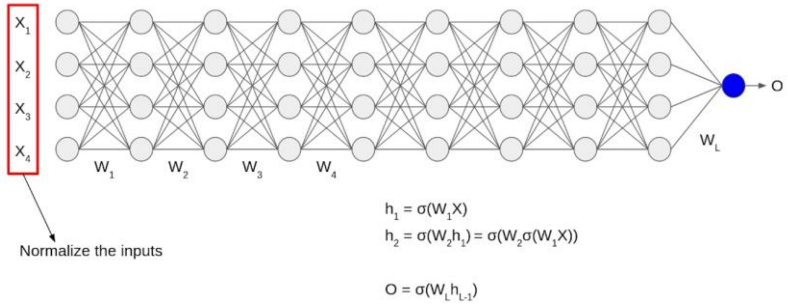
- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.
- The reason we normalize is partly to ensure that our model can generalize appropriately.
- A typical neural network is trained using a collected set of input data called **batch**.
- Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

Challenges..

- Although, our input X was normalized with time the output will no longer be on the same scale. As the data go through multiple layers of the neural network and L activation functions are applied, it leads to an internal co-variate shift in the data.

X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage.





Batch Normalization

- It is a two-step process.
  - First, the input is normalized
  - Rescaling and offsetting

Normalization of the Input

- Normalization is the process of transforming the data to have a mean zero and standard deviation one.

- In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

  Here, m is the number of neurons at layer h.
- Next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

Normalization of the Input…..

- We will normalize the hidden activations using these values. For this, we will subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).
- The smoothing term(ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

Rescaling of Offsetting

- Here two components of the BN algorithm come into the picture, γ(gamma) and β (beta).
- These parameters are used for re-scaling (γ) and shifting(β) of the vector containing values from the previous operations.

$$h_i = \gamma h_{i(norm)} + \beta$$

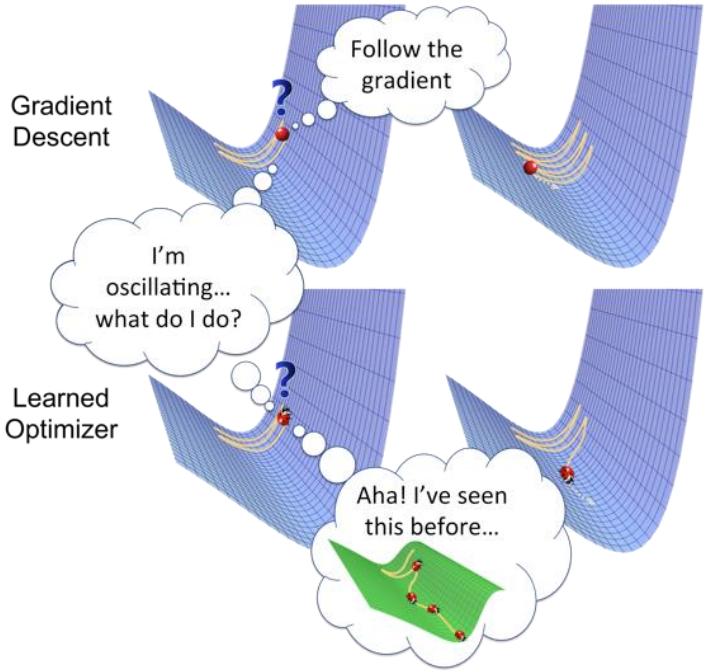
- These two are learnable parameters, during the training neural network ensures the optimal values of γ and β are used.

Benefits of Batch Normalization

- **Speeds up learning**: By reducing internal covariate shift, it helps the model train faster.
- **Regularizes the model**: It adds a little noise to your model, and in some cases, you might not even need to use dropout or other regularization techniques.
- **Allows higher learning rates**: Gradient descent usually requires small learning rates for the network to converge. Batch normalization helps us use much larger learning rates, speeding up the training process.
- **smoothens the loss function:** Batch normalization smoothens the loss function that in turn by optimizing the model parameters improves the training speed of the model.

Optimizer

Facilitate the learning process by iteratively refining the weights and biases based on feedback received



Need of Optimizer

- To train a neural network, we need to define a loss function to measure the difference between the network's predictions and the ground truth label.
- During training, we look for a **specific set of weight parameters** that the neural network can use to make an accurate prediction.
- This simultaneously leads to a lower value of the loss function (Gradient descent).

$$\theta_j \leftarrow \theta_j - \epsilon \nabla_{\theta_j} \mathcal{L}(\theta)$$

- In gradient descent we need to calculate the gradient of the loss function with respect to the weight parameters that we want to improve. Then, the weight parameters are **updated in the direction of the negative gradient** of the loss function .

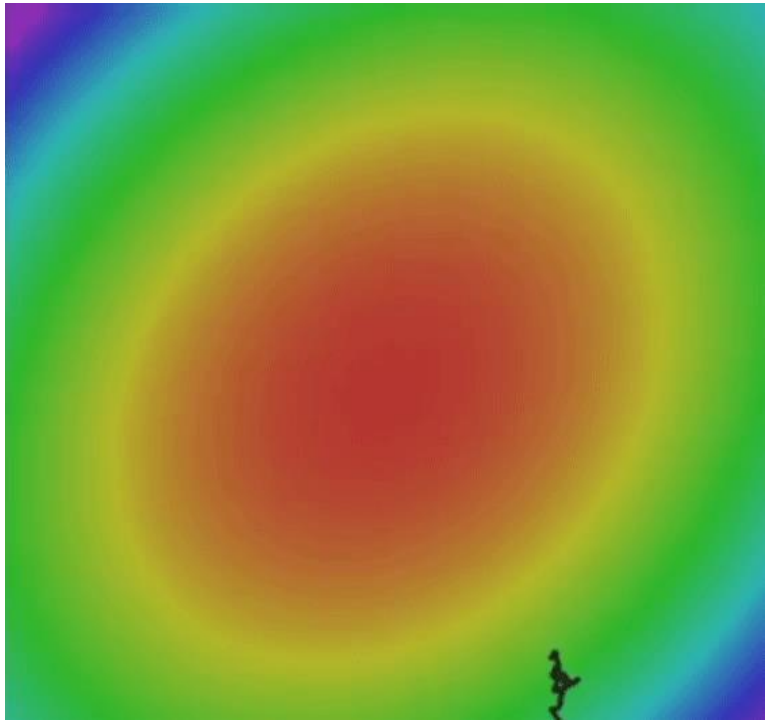
Need of optimizer

At the saddle points and the local minima the loss function becomes flat and the gradient at this point goes towards zero

Fig: Saddle point (right), local minima(left)

A gradient close to zero in a saddle point or in a local minimum does not improve the weight parameters and prevents the whole learning process.

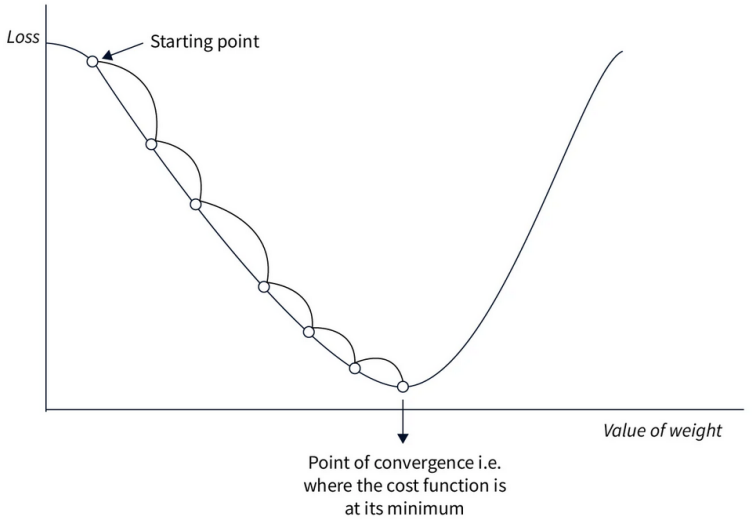
- The difference in values and directions of each gradient results in a zig-zag motion towards the optimal weights and can slow down learning.
- The values of gradients calculated for different data samples from the training set may vary in value and direction.
- The gradients are “noisy” or have a large variance.



Gradient Descent

$$\theta = \theta - \alpha \cdot \nabla_\theta L(\theta)$$

where θ is the model parameter, $L(\theta)$ is the loss function, and α is the learning rate.



Pros and Cons

Pros

- Simple to implement
- Can work well with a well tuned learning rate

Cons

- Converges very slowly
- Sensitive to choice of learning rate

Stochastic Gradient Descent

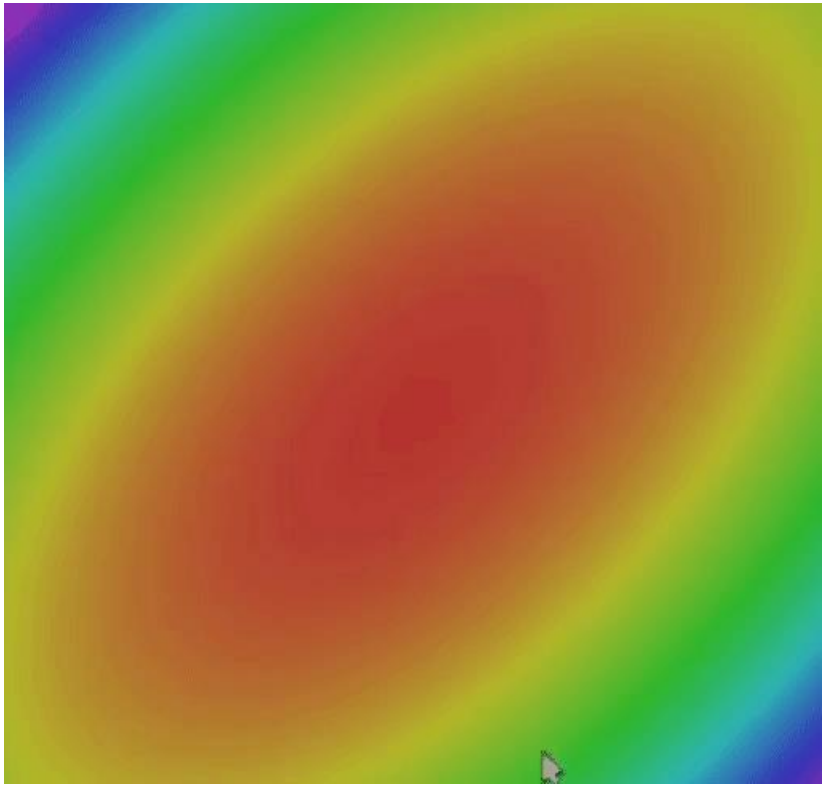
SGD

$$\theta_j \leftarrow \theta_j - \epsilon \nabla_{\theta_j} \mathcal{L}(\theta)$$

SGD mit Impuls

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$



- Comparison between regular stochastic gradient descent (black) and stochastic gradient descent with momentum (blue).
- Both algorithms try to reach the global minimum of the loss function.
- The momentum term results in the individual gradients having less variance and thus less zig-zagging.

SGD Cont...

- The equation on the right represents the rule for the updates of the weights according to the SGD with momentum.
- **Momentum appears here as an additional term**, which is added to the regular update rule.
- By adding this impulse term, we let our **gradient build up some sort of velocity** during training.
- The velocity is the running sum of the gradients weighted by rho.
- In general, velocity can be seen to increase with time.
- By using the momentum term, **saddle points and local minima become less dangerous** for the gradient. This is because the step size toward the global minimum now depends not only on the slope of the loss function at the current point, but **also on the velocity** that has built up over time.

Pros and Cons

Pros

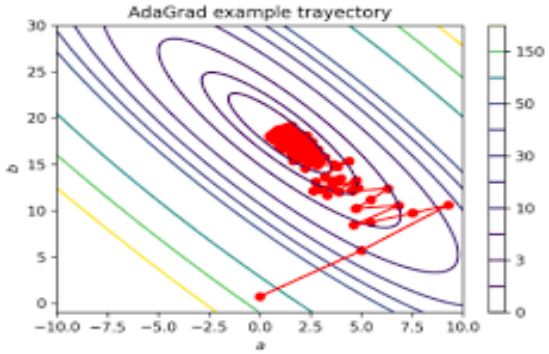
- It can help the optimizer to move more efficiently through "flat" regions of the loss function.
- It can help to reduce oscillations and improve convergence.

Cons

- Can overshoot good solutions and settle for suboptimal ones if the momentum is too high.
- Requires tuning of the momentum hyperparameter.

AdaGrad (Adaptive Gradient Descent)

- The idea is to use different learning rates for each parameter based on iteration
- The reason behind the need for different learning rates is that the learning rate for sparse features parameters needs to be higher compare to the dense features parameter because the frequency of occurrence of sparse features is lower



AdaGrad (Adaptive Gradient Descent)

Learning rate will automatically decrease because the summation of the previous gradient square will always keep on increasing after every time step.

$$\text{SGD} \Rightarrow \ w_t = w_{t-1} - \eta \ \frac{\partial L}{\partial w_{t-1}}$$

$$\text{Adagrad} \ \Rightarrow w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w_{t-1}}$$

$$\text{where } \eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

ε is a small $+$ve number to avoid divisibilty by 0

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2 \quad \textit{summation of gradient square}$$

Pros and Cons

Pros

- It can work well with sparse data.
- Automatically adjusts learning rates based on parameter updates.

Cons

- Can converge too slowly for some problems.
- Can stop learning altogether if the learning rates become too small.

RMSProp (Root Mean Square Propagation)

AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_\theta \mathcal{L}(\theta)^2$$

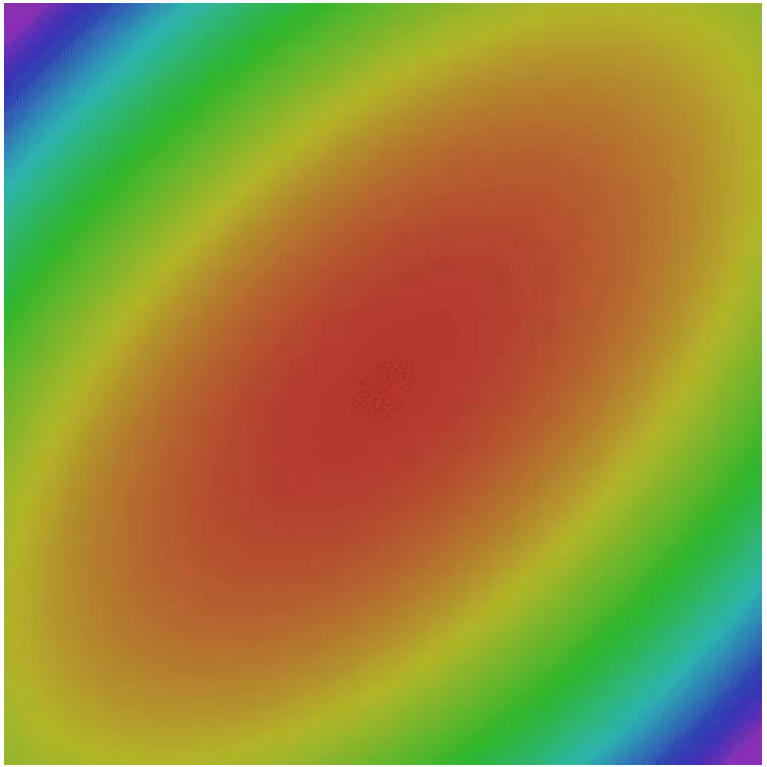
$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

RMS Prop

$$g_0 = 0, \alpha \simeq 0.9$$

$$g_{t+1} \leftarrow \alpha \cdot g_t + (1-\alpha)\nabla_\theta \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_\theta \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$



SGD: Black
SGD Mom: Blue
RMS Prop: Red

RMS Prop…

- Over time, sum of the squared gradient would **grow larger.**
- The current gradient is divided by this large number, the update step for the weights becomes very small. It is as if we were using **a very low learning rate**, which becomes even lower the longer the training takes.
- In the worst case, we would get stuck at AdaGrad and the training would go on forever.
- In RMSProp, the running sum of squared gradients g_{t+1} is maintained. However, instead of allowing this sum to increase continuously over the training period, we allow the sum to decrease.

RMSProp Cont..

- For RMSProp, the sum of squared gradients is multiplied by a decay rate α and the current gradient – weighted by (1- α) – is added. The update step in the case of RMSProp looks the same as in AdaGrad.
- Here we divide the current gradient by the sum of the squared gradients to get the nice property of speeding up the updating of the weights along one dimension and slowing down the motion along the other.

Pros and Cons

Pros

- It can work well with sparse data.
- Automatically adjusts learning rates based on parameter updates.
- Can converge faster than Adagrad.

Cons

- It can still converge too slowly for some problems.
- Requires tuning of the decay rate hyperparameter.

AdaDelta

- Extension of Adagrad that attempts to solve its radically diminishing learning rates.
- Instead of summing up all the past squared gradients from 1 to “t” time steps, restriction on the window size

$$\text{Adagrad} \Rightarrow \quad \eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}} \quad \text{where} \quad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2$$

$$\text{so, as "t"} \uparrow\uparrow \text{ , } \alpha_t \uparrow\uparrow \text{, and } \eta_t^{'} \downarrow\downarrow$$

AdaDelta

The previous equation shows that as the time steps “t” increase the summation of squared gradients “α” increases which led to a decrease in learning rate “η”. In order to resolve the exponential increase in the summation of squared gradients “α” is replaced with exponentially weighted averages of squared gradients

$$\eta'_t = \frac{\eta}{\sqrt{S_{dw_t} + \varepsilon}} \quad \text{where} \quad S_{dw_t} = \beta S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_{t-1}}\right)^2$$

ε is a small $+$ve number to avoid divisibilty by 0

Pros and Cons

Pros

- Can work well with sparse data.
- Automatically adjusts learning rates based on parameter updates.

Cons

- Can converge too slowly for some problems.
- Can stop learning altogether if the learning rates become too small.

ADAM (Adaptive Moment Estimation)

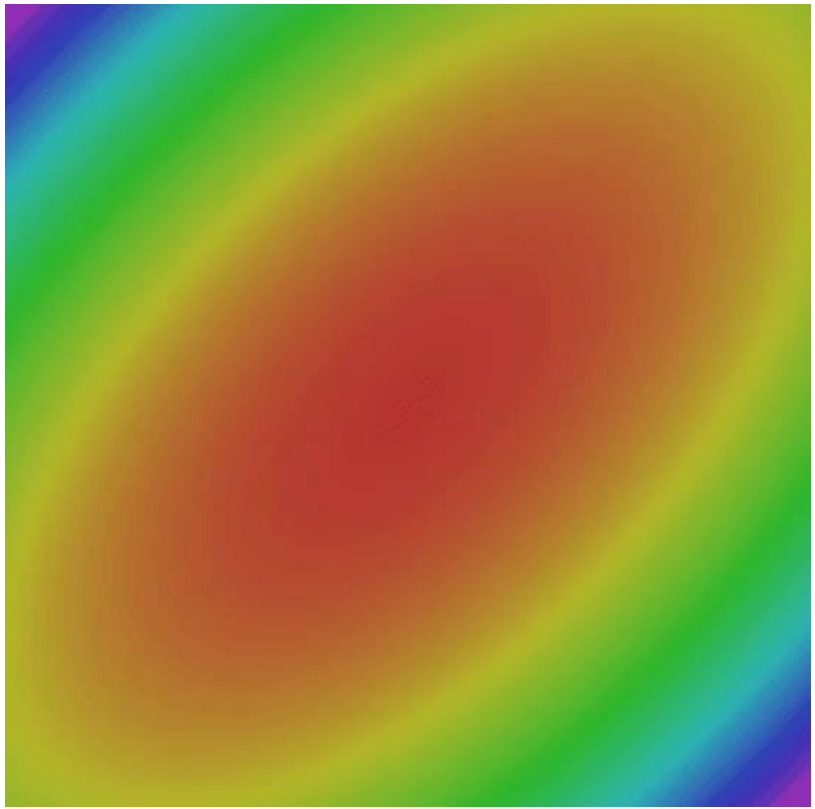
$$m_0 = 0, v_0 = 0$$

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta)$$ Impuls

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$ RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$ RMS Prop + Impuls

SGD-Black
Stochastic Grad- Blue
RMS Prop-Red
ADAM: Rubby Red



ADAM Cont..

1. The idea behind Adam optimizer is to utilize the momentum concept from “SGD with momentum” and adaptive learning rate from “Ada delta”.
2. It uses the squared gradients to scale the learning rate like RMSprop, and it takes advantage of momentum by using the moving average of the gradient instead of the gradient itself, like SGD with momentum. This combines Dynamic Learning Rate and Smoothening to reach the global minima.
3. Adam tweaks the gradient descent method by considering the moving average of the first and second-order moments of the gradient. This allows it to adapt the learning rates for each parameter intelligently.
4. This algorithm calculates the exponential moving average of gradients and square gradients.
5. The parameters of β1 and β2 are used to control the decay rates of these moving averages.

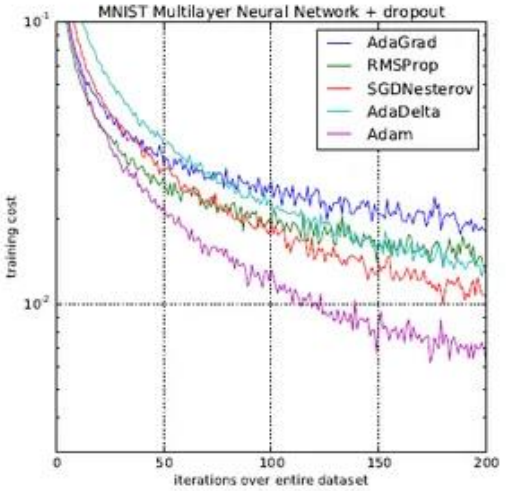
Pros and Cons

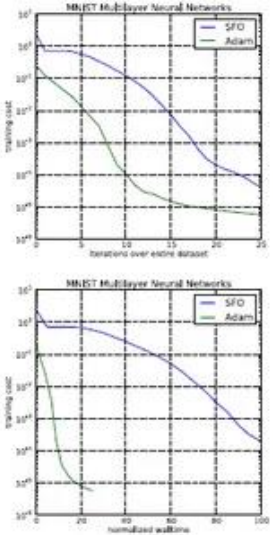
Pros

- Can converge faster than other optimization algorithms.
- Can work well with noisy data.

Cons

- It may require more tuning of hyperparameters than other algorithms.
- May perform better on some types of problems.





Conclusion

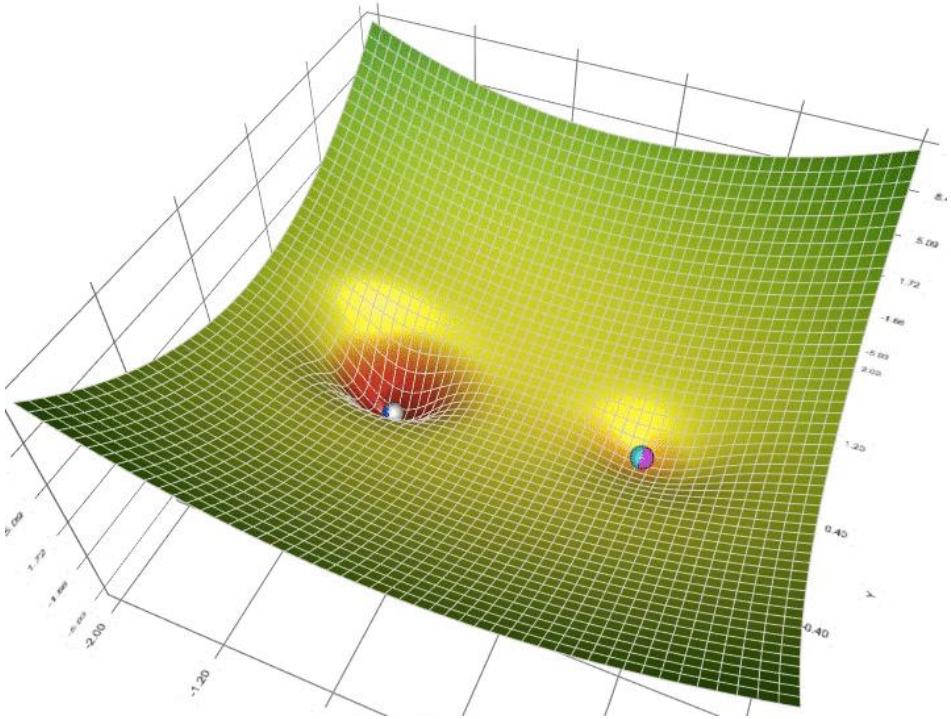
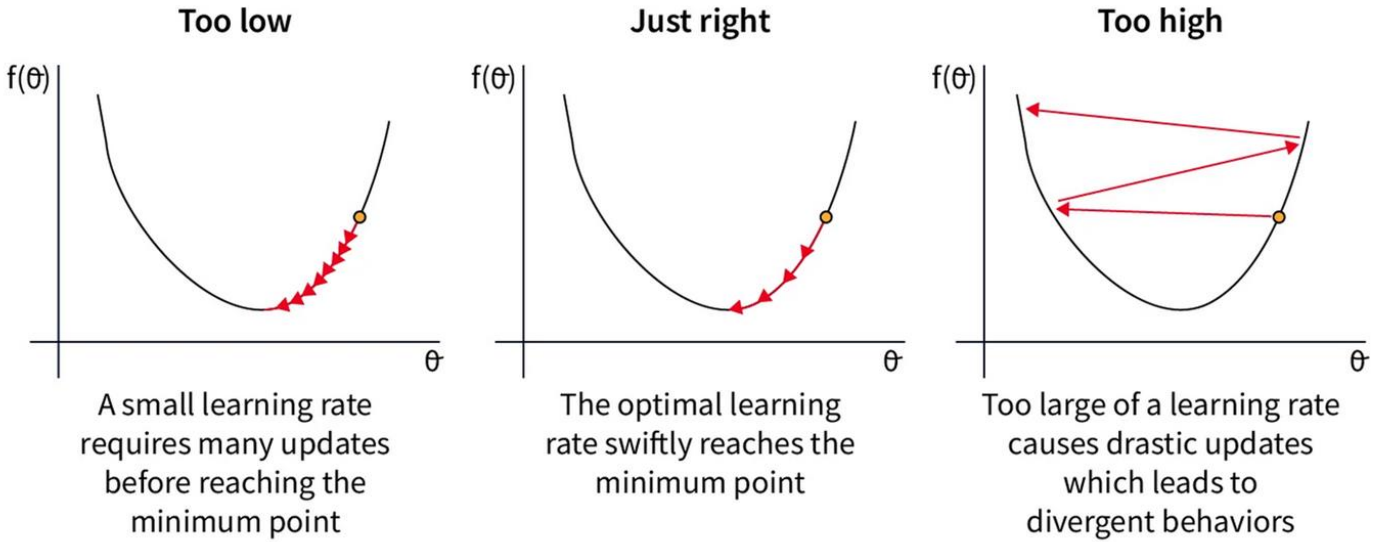


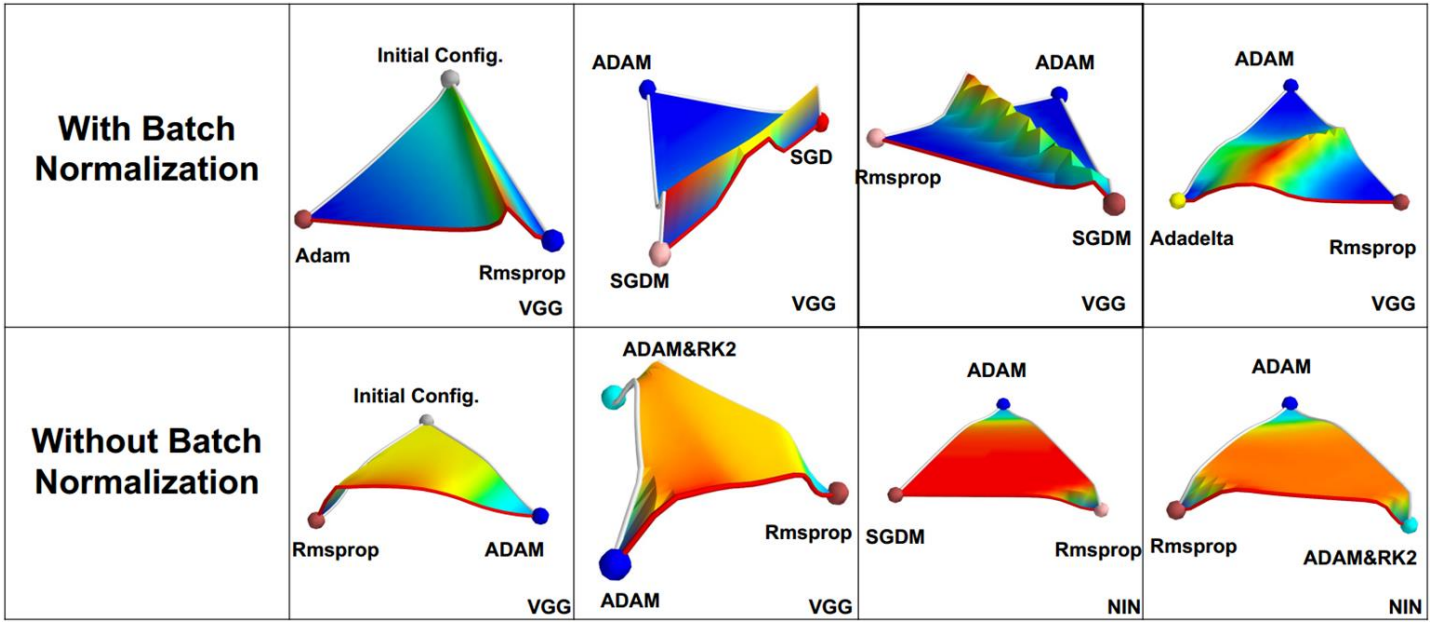
Fig: Visual representation on surface: GD(cyan), SGD+momentum(magenta), AdaGrad(white), RMSProp(green), Adam(blue)

Learning rate

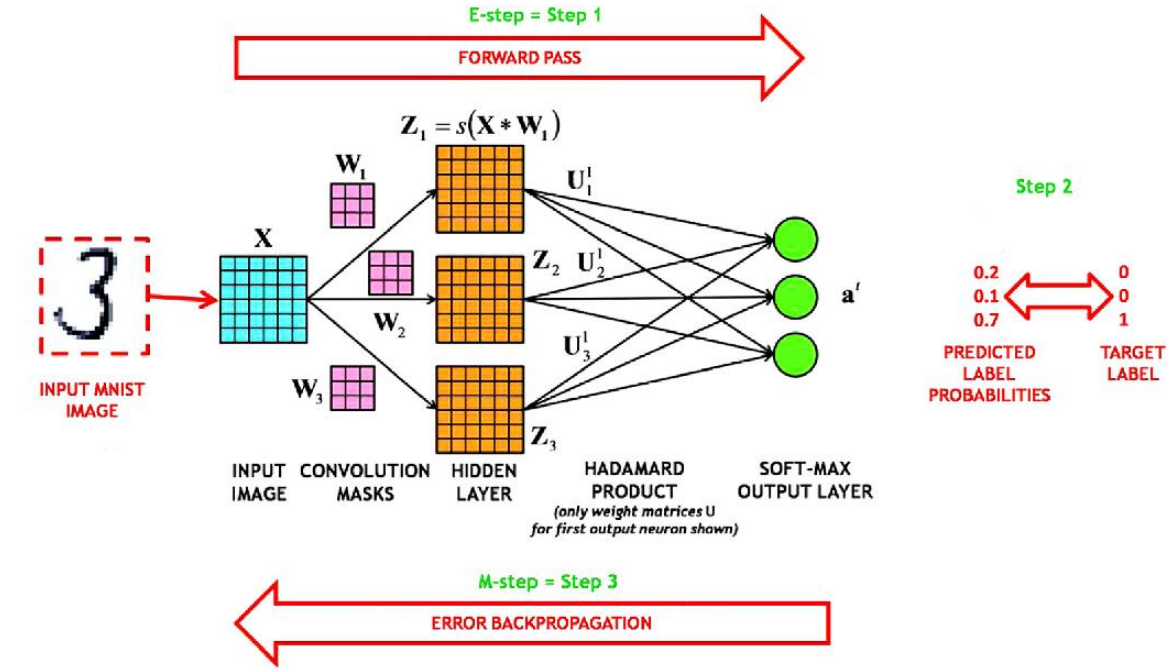


Batch Normalization

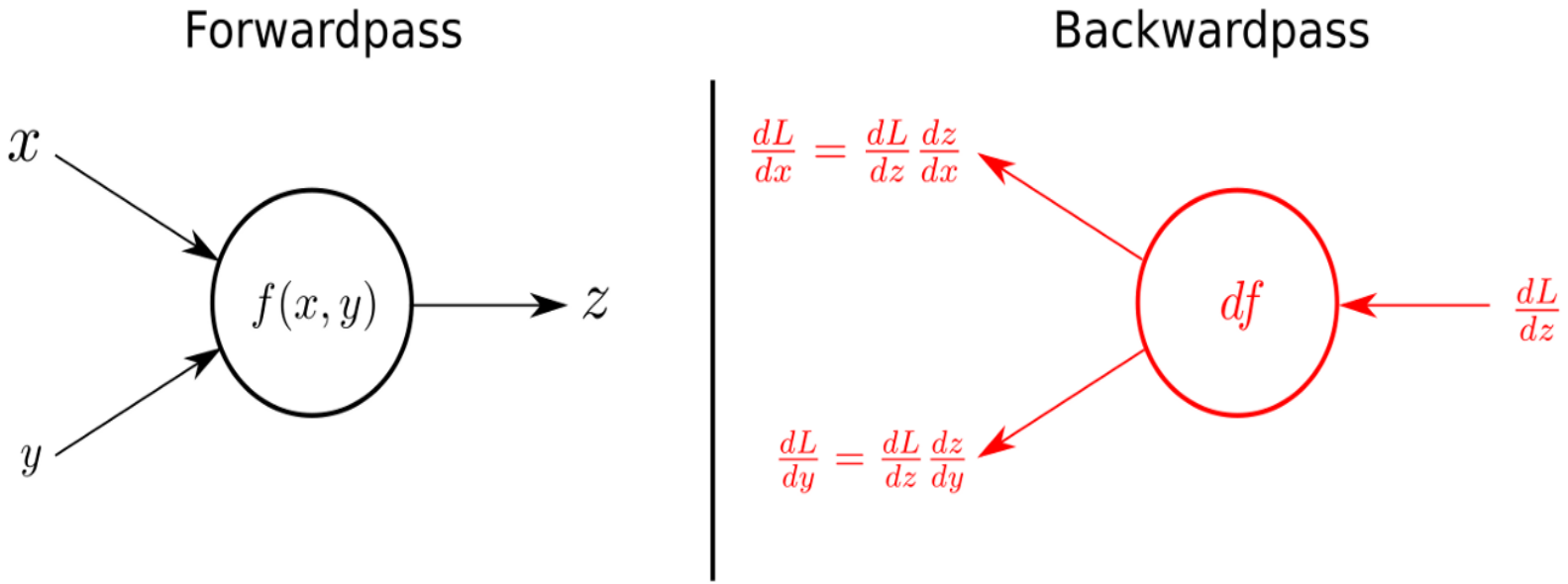
BN greatly reduces the plateau of loss function and makes updating easier for optimizer.



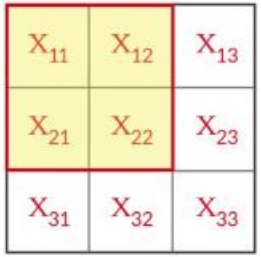
CNN

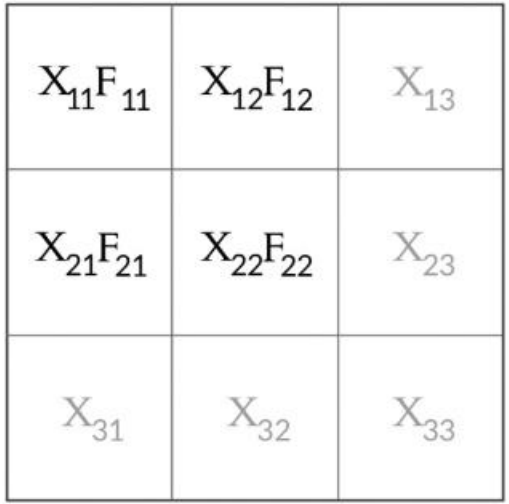

E-step = Step 1
FORWARD PASS
$Z_1 = s(X * W_1)$
W_1
U_1^1
X
Z_2 U_2^1
W_2
U_3^1
W_3
Z_3
a^1
INPUT MNIST IMAGE
INPUT IMAGE
CONVOLUTION MASKS
HIDDEN LAYER
HADAMARD PRODUCT
(only weight matrices U for first output neuron shown)
SOFT-MAX OUTPUT LAYER
Step 2
0.2
0.1
0.7
0
0
1
PREDICTED LABEL PROBABILITIES
TARGET LABEL
M-step = Step 3
ERROR BACKPROPAGATION

CNN Learning: Intuition



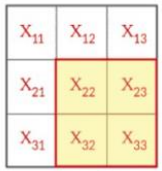
Forward Pass





$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Forward Pass



X_{11}	X_{12}	X_{13}
X_{21}	$X_{22}F_{11}$	$X_{23}F_{12}$
X_{31}	$X_{32}F_{21}$	$X_{33}F_{22}$

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

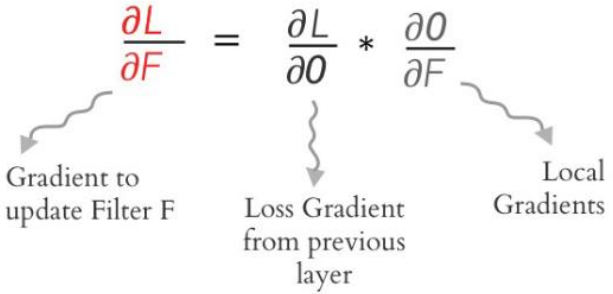
$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22}$$

$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22}$$

$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22}$$

Backward Pass

Loss gradient by chain rule



For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

Loss gradient w.r.t filter

Expanding the chain rule

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$

Loss gradient w.r.t filter

Replacing local gradients of filter

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X}$$

$$\begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O}$$ Loss gradient from previous layer

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

Loss gradient w.r.t filter

Resembles convolution between input X and loss gradient

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \quad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O} \text{ Loss gradient from previous layer}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

Lost gradient w.r.t input

For every element of X_i

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial X_i}$$

Lost gradient w.r.t input

Expand chain rule

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

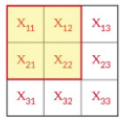
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

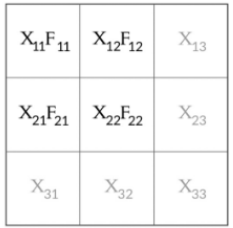
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

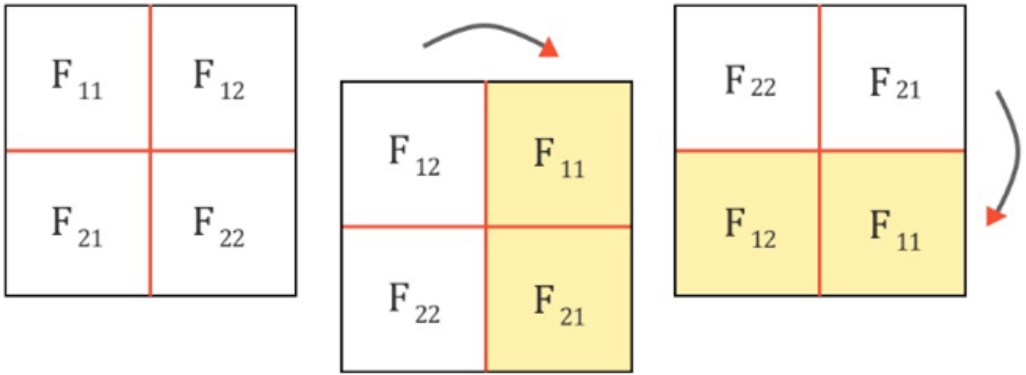




$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

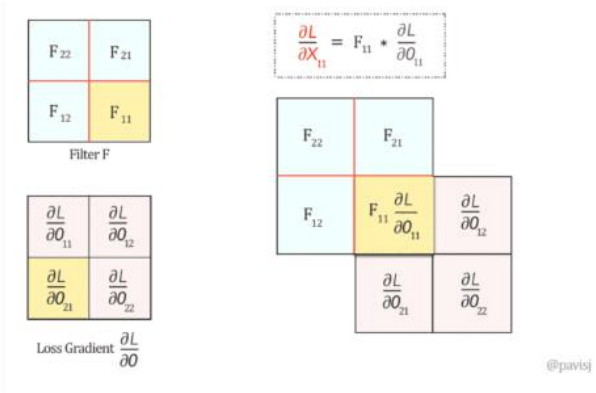
Lost gradient w.r.t input

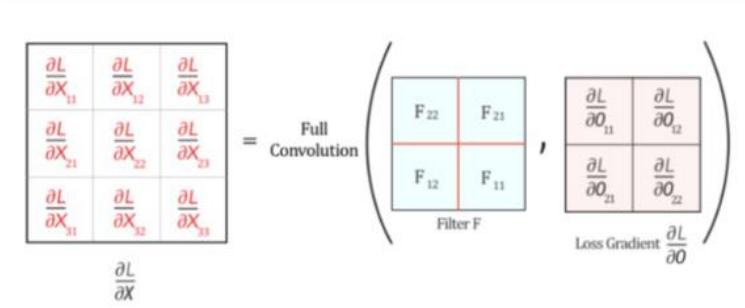
Rotating filter by 180 degrees



Lost gradient w.r.t input

Full convolution between filter and gradient





Summarising Back Propagation

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X\text{, Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(\text{180° rotated Filter } F\text{, Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Thank you