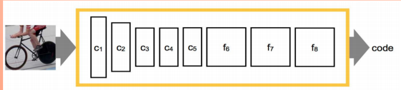
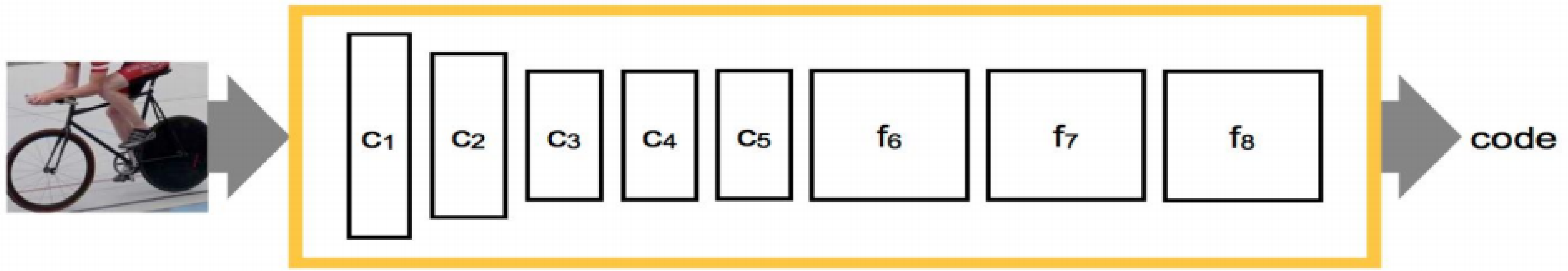
Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- ✓ Appreciating CNNs
- ✓ Visualization CNN
- • Representation and Practice



Representation and Practice

Learned Representations



Blank Slide

Learned Representations

CNN Features can be used for wider applications:

- Train the CNN (deep network) on a very large database such as imageNet.

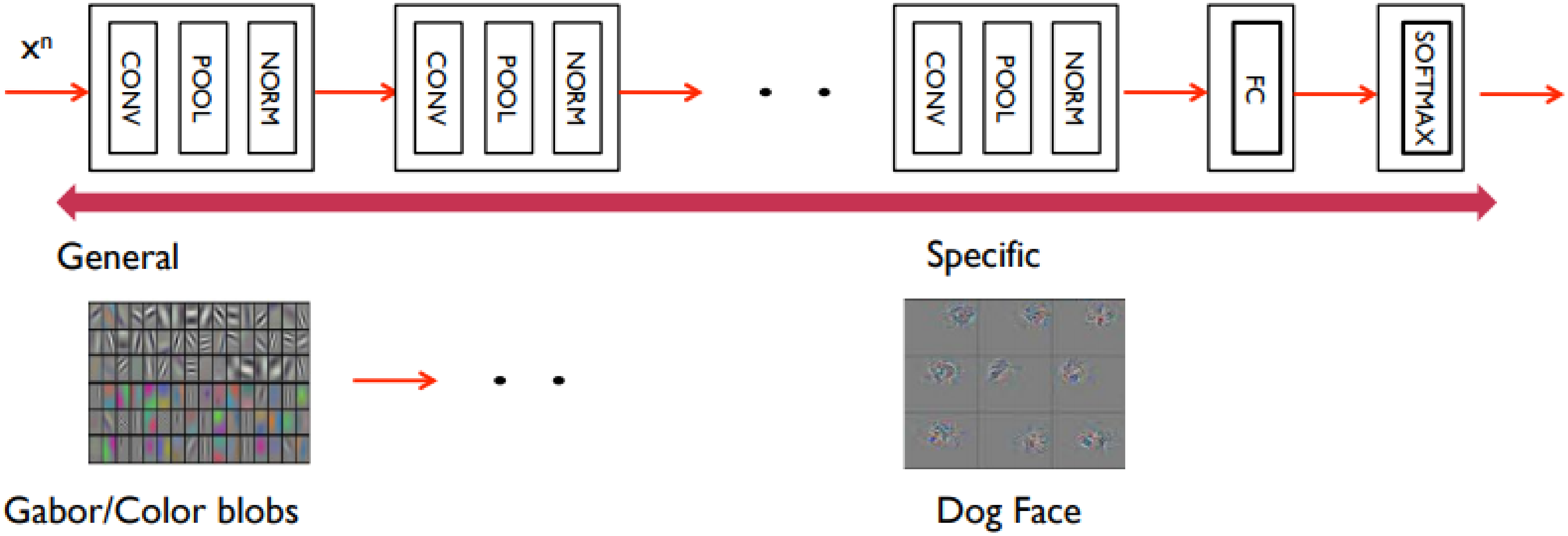
- Re-use CNN to solve smaller problems
  - Remove the last layer (classification layer)
  - Output is the code/feature representation

New Settings

- Extend to more classes
  – Extend from 1000 classes (say people) to another new 100
- Extend to new tasks
  – Extend from object classification to scene classification
- Extend to new data sets
  – Extend from imageNet to PASCAL (SLR to webcams)
- When we have a lesser amount of data.

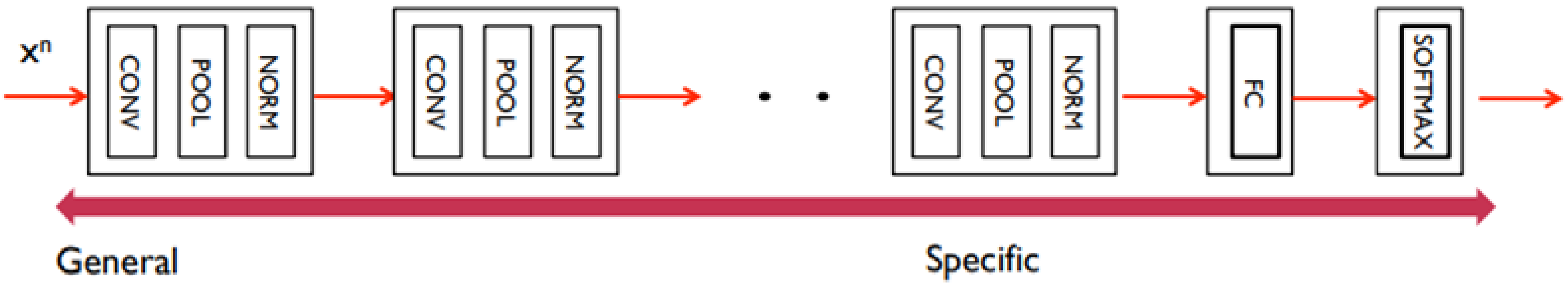
Transfer Learning

- **A key observation that we noticed in visualization:-**



Transfer Learning

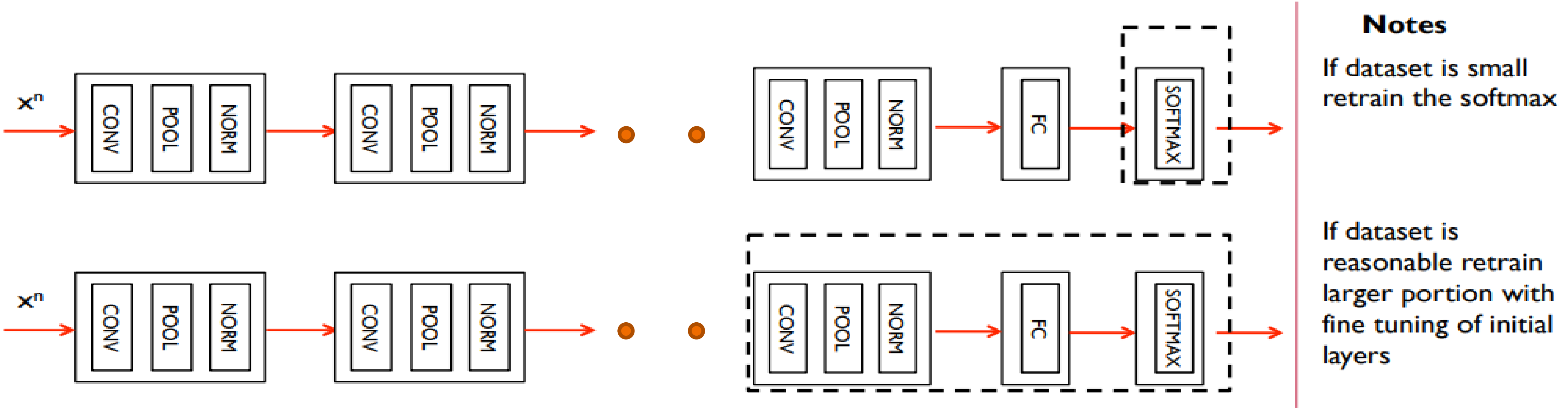
- **A key observation that we noticed in visualization:-**



- Further questions?
- Can we quantify the layer generality/specificity ?
- Where does the transition occur?
- Is the transition sudden or spread over layers?

Transfer Learning

- Take away message



- Initializing a network with transferred features almost always gives better generalization

Summary And Insights

- Pre-Training is important
  - With a related data set, synthetic data set.

- Find proxy/related tasks and learn features.
  - Unsupervised (eg. Autoencoder)
  - Word2Vec
  - Siamese

- Finetune/adapt/transfer learn
  - Work with smaller number of examples

Blank Slide

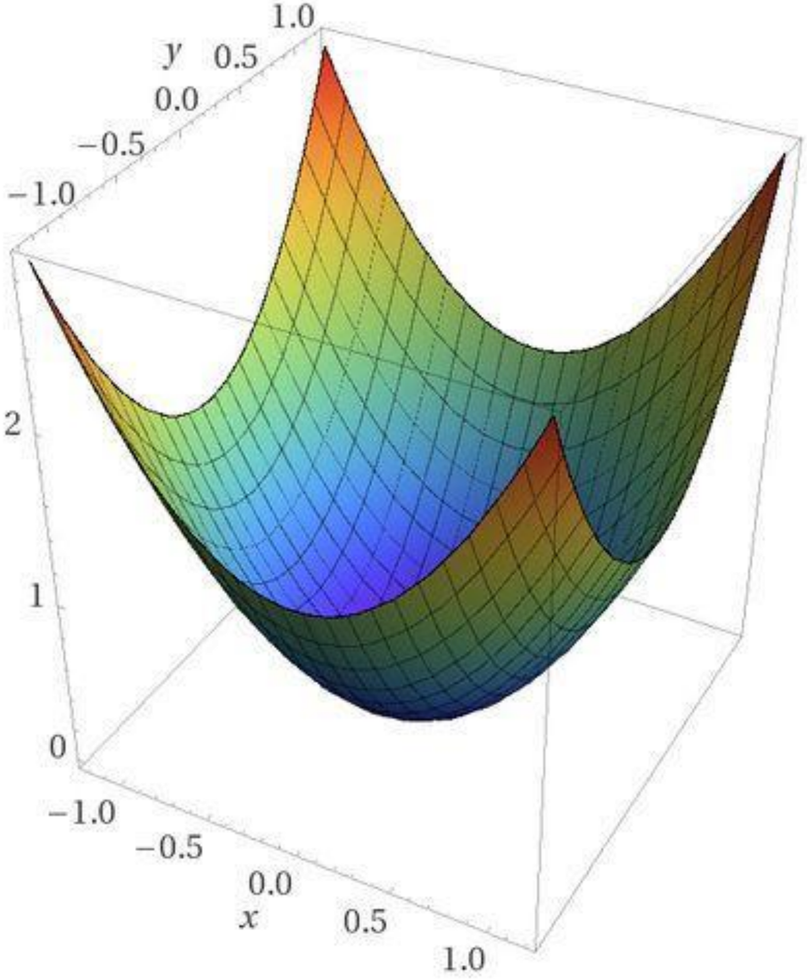
Blank Slide

Questions?

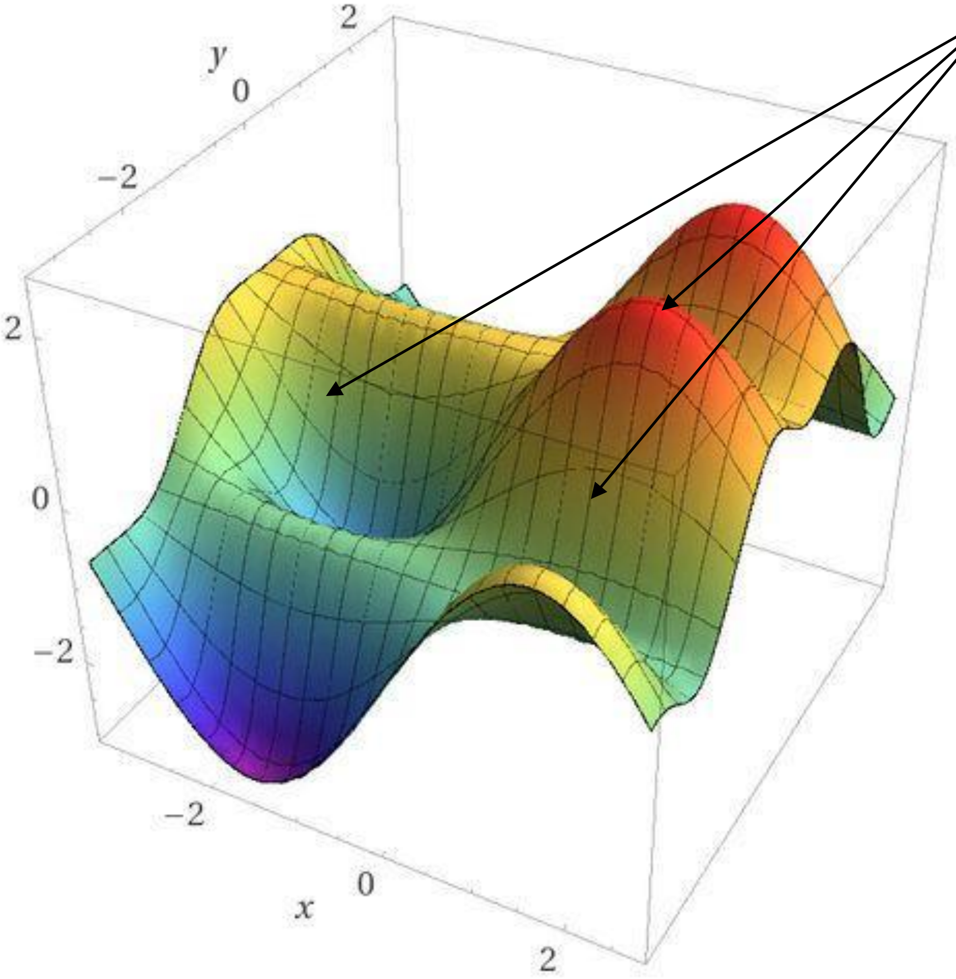
Beyond Backpropagation

Tips and tricks for training deep neural networks

Finding Global Minima: Why is it hard?



Loss space in our expectation



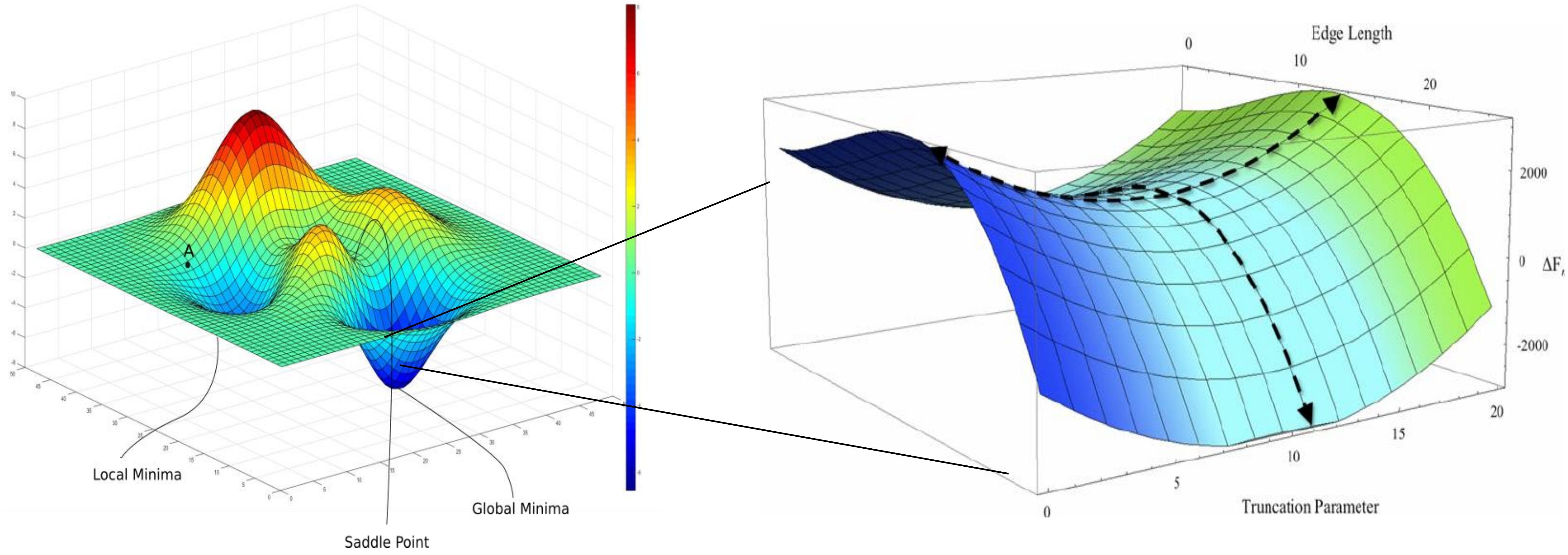
local minima

$$\nabla J = 0$$

Loss space in reality

Image Credit: https://blog.paperspace.com

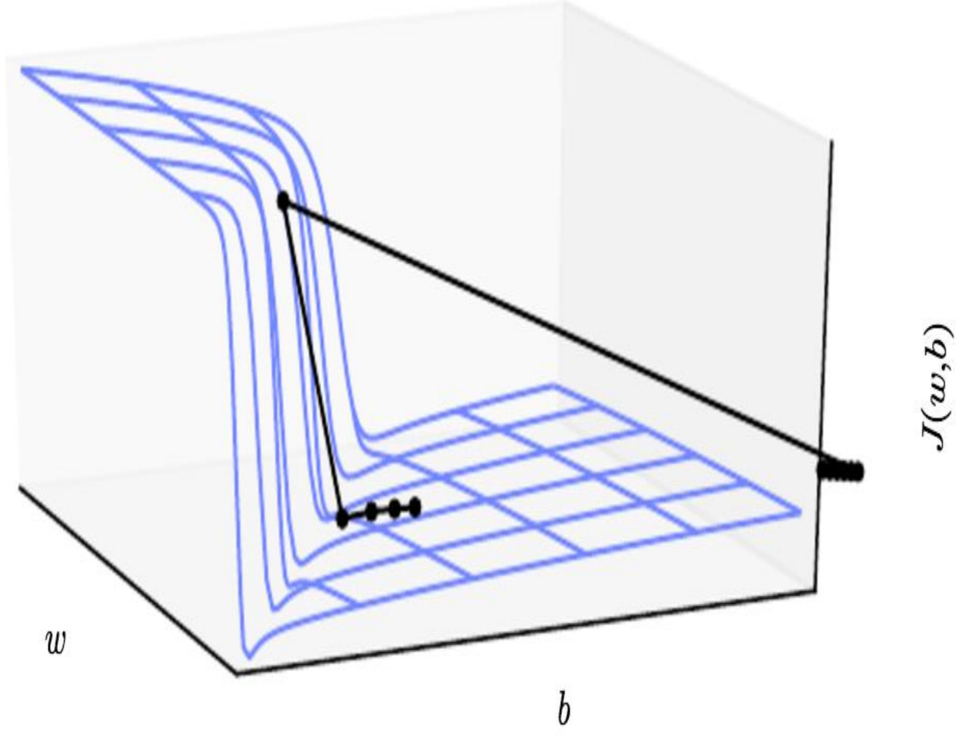
Finding Global Minima: Why is it hard?

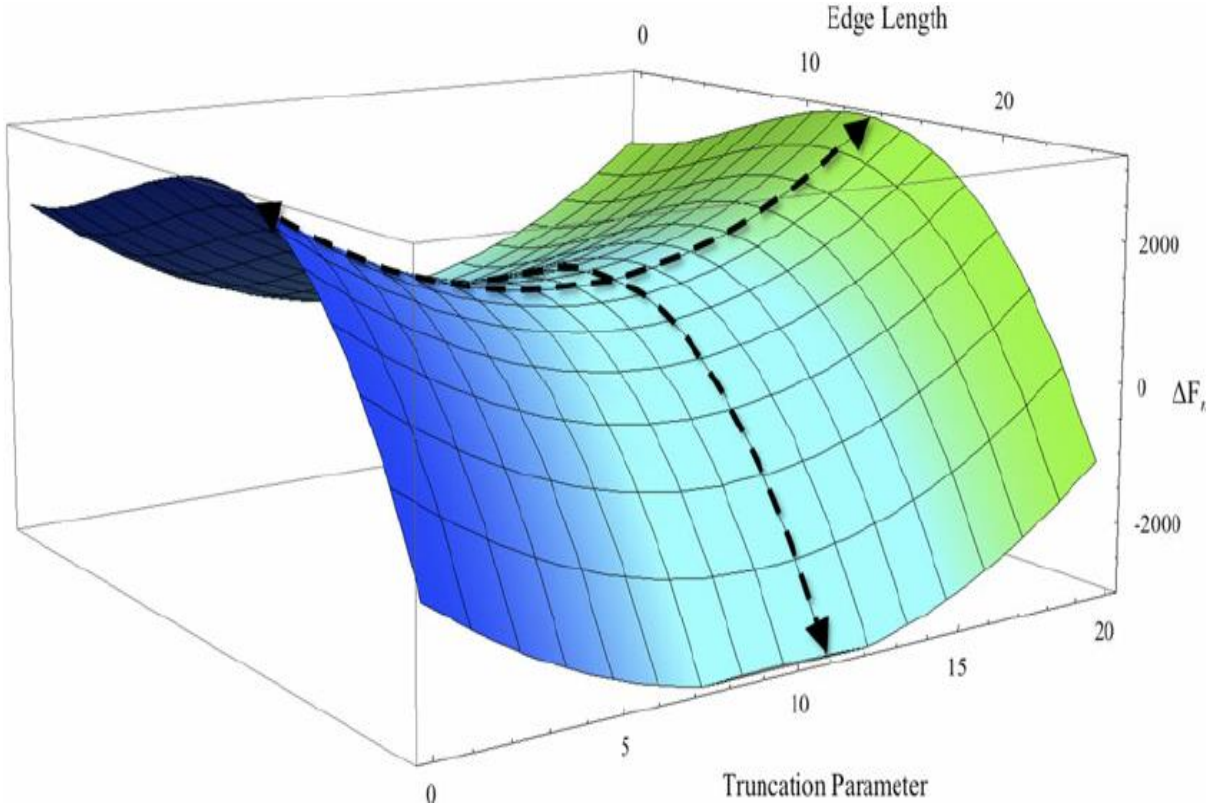


Saddle Point: Local minima in one direction, local maxima in another direction

Image Credit: https://blog.paperspace.com

Finding Global Minima: Why is it hard?





Cliffs : gradient is too high

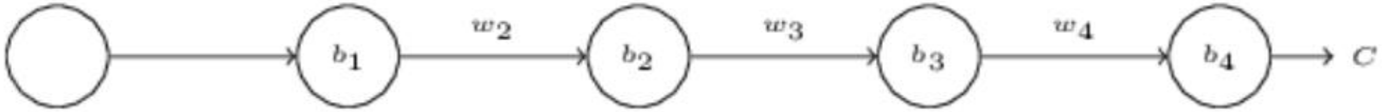
Plateau : gradient is almost zero

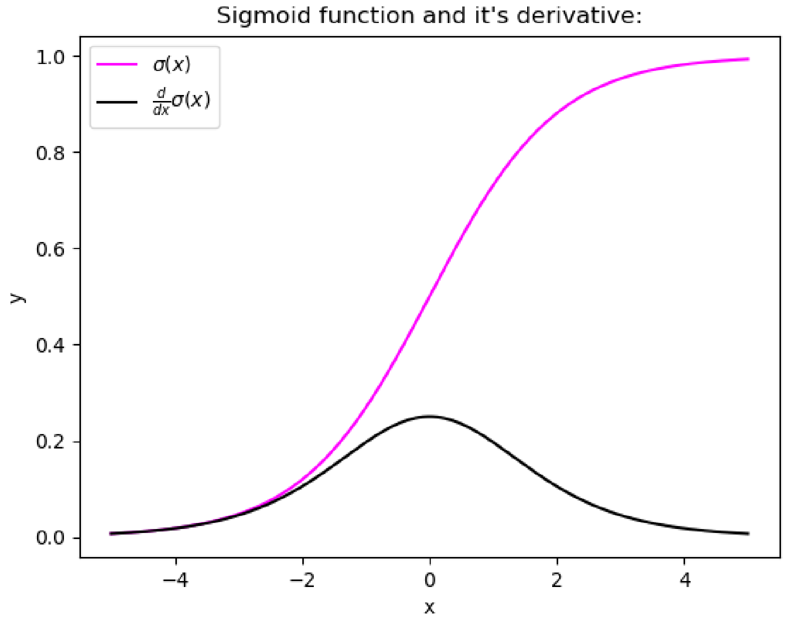
Image Credit: towrdsdatascience.com

Finding Global Minima: Why is it hard?

The Vanishing Gradient Problem:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

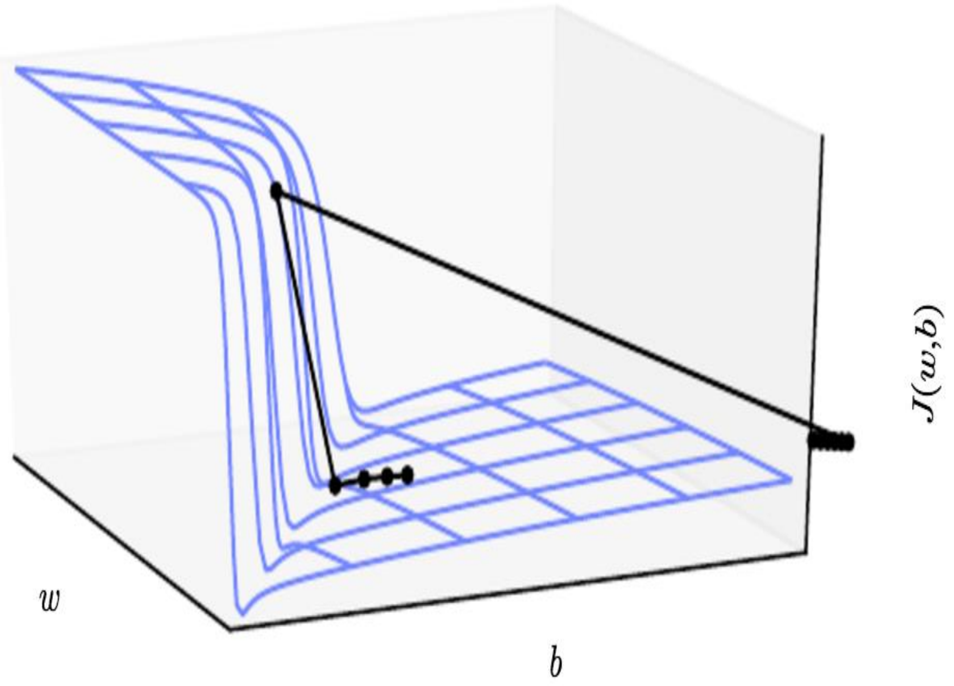




$$\left|\left|\frac{\delta h_i}{\delta h_{i-1}}\right|\right|_2 < 1$$

Image Credit:machinelearningjourney.com

Finding Global Minima: Why is it hard?



$$\|\frac{\delta h_i}{\delta h_{i-1}}\|_2 > 1$$

Solution: Gradient clipping

When gradient is too high, the repetitive multiplication results in exploding gradient

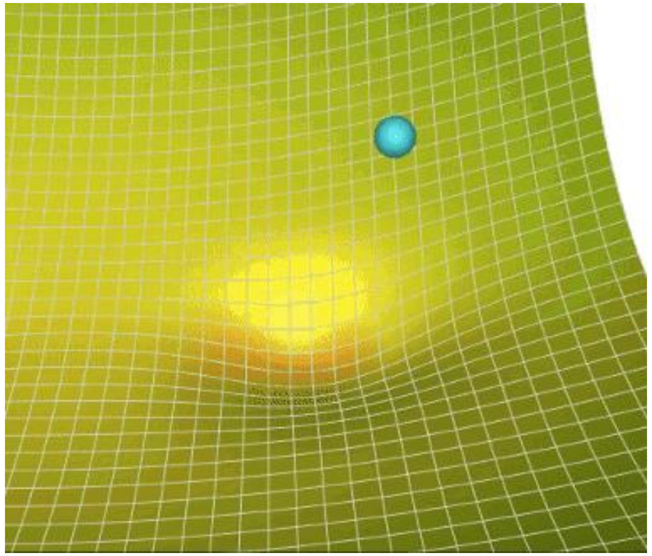
Image Credit: towrdsdatascience.com

Solution – I: Better Optimizers

Ideal optimizer:

- Finds minimum fast and reliably well
- Doesn't get stuck in local minima, saddle points, or plateau region

$$w_{t+1} = w_t - \alpha * \frac{\delta L}{\delta w_t}$$



Vanilla Gradient Descent: One step for the entire dataset

Stochastic Gradient Descent: One step for each stochastically chosen sample

Mini-batch Gradient Descent: One step for each mini-batch of samples chosen stochastically

Best of both worlds!

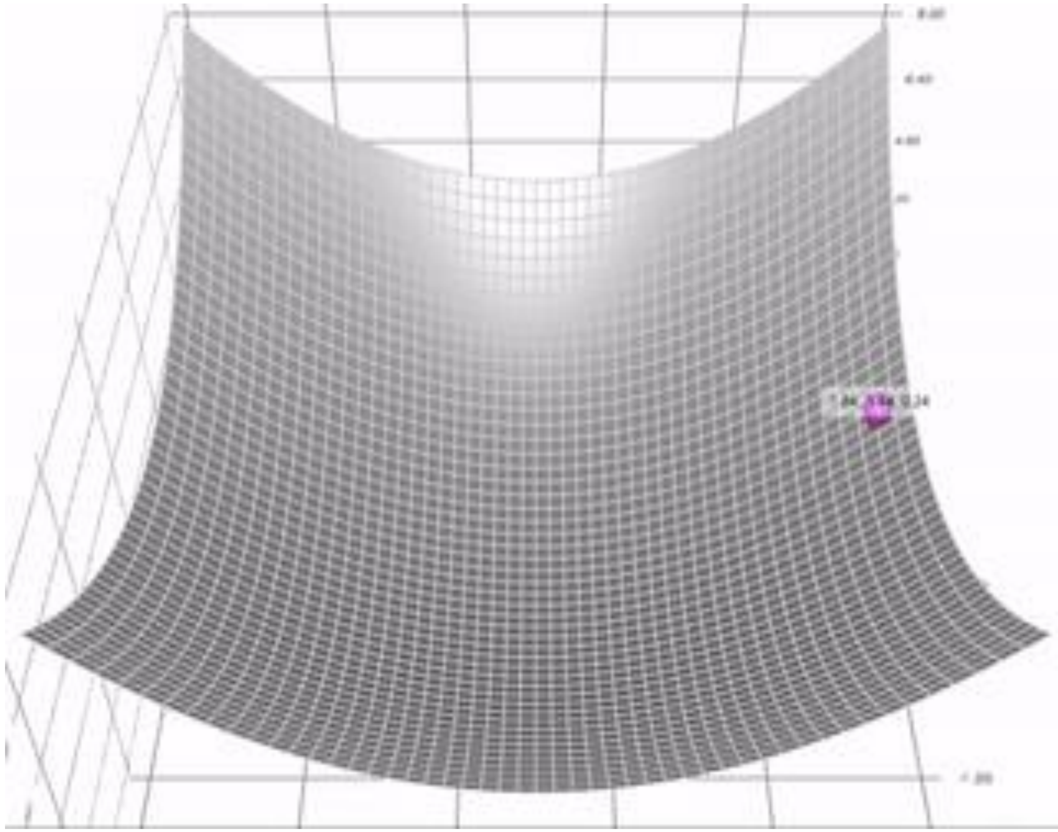
```
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0)
```

Image Credit: towrdsdatascience.com

Solution – I: Better Optimizers

Gradient Descent with Momentum

- Imagine a rolling down a ball inside a frictionless bowl
- The ball doesn’t stop at the bottom of the surface
- Uses the accumulated momentum to go forward



$$v_t = \underbrace{\boxed{\gamma\, v_{t-1}}}_{\text{Momentum}} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

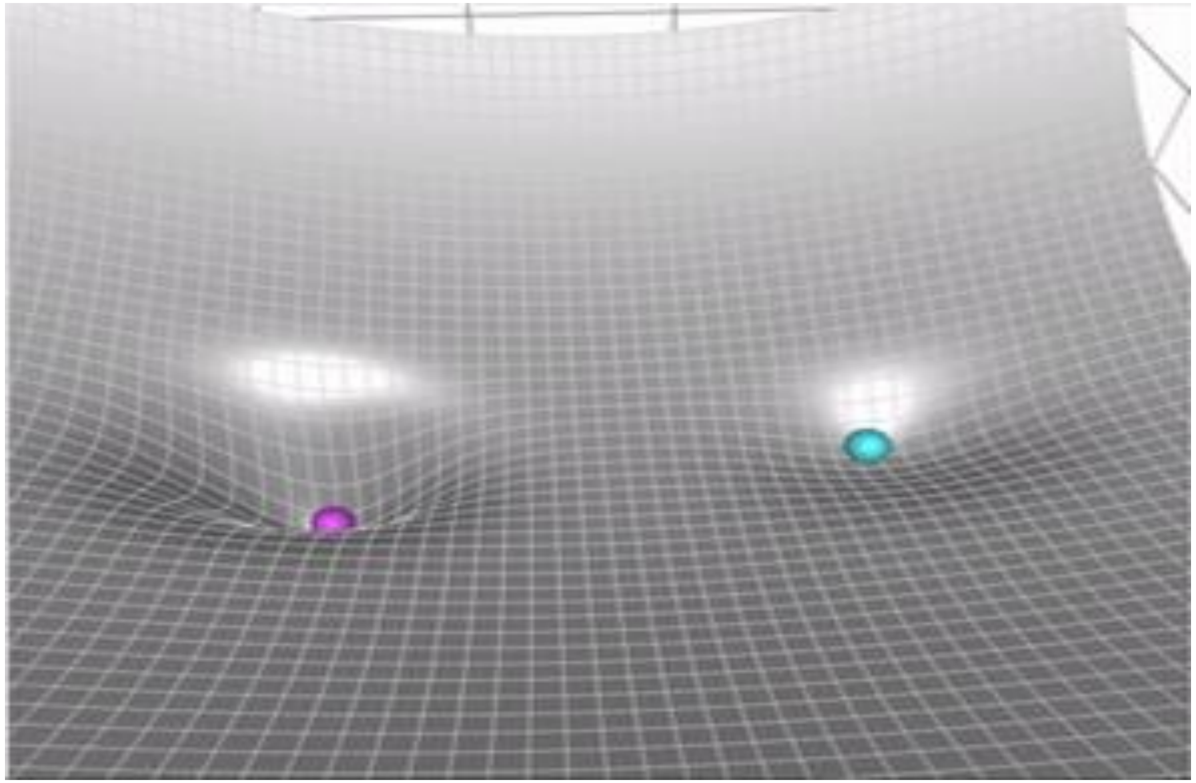
$$\theta = \theta - v_t$$

Image Credit: towrdsdatascience.com

Solution – I: Better Optimizers

Gradient Descent with Momentum

Intuitively, this helps us to come out of local minima



$$v_t = \boxed{\gamma\, v_{t-1}} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

Momentum

```
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

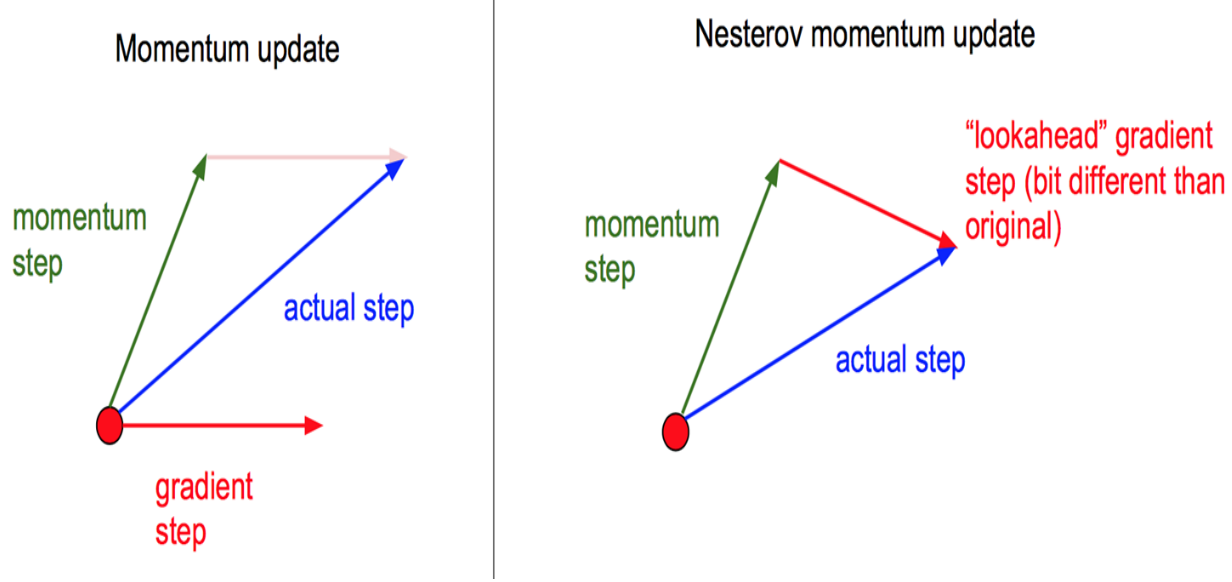
Image Credit: towrdsdatascience.com

Solution – I: Better Optimizers

Gradient Descent with Nesterov Momentum

- Following the slope blindly is not desirable and optimal
- The ball should predict and slow itself down before going up again

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta \boxed{- \gamma v_{t-1}})$$
$$\theta = \theta - v_t$$




```python
import torch.optim as optim

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, nesterov=True)
```

Image Credit: towrdsdatascience.com

Solution – I: Better Optimizers

Other Variants:

- RMSprop
- Adagrad
- **Adam**
- Adadelta
- etc.

```python
import torch.optim as optim

optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99)
```

```python
import torch.optim as optim

optimizer = optim.Adagrad(model.parameters(), lr=0.01)
```

```python
import torch.optim as optim

optimizer = optim.Adam(model.parameters(), lr=0.01, betas=(0.9, 0.999))
```

```python
import torch.optim as optim

optimizer = optim.Adadelta(model.parameters(), lr=0.01, rho=0.9)
```

Simplified View: Variable Learning Rates for Features/Dimensions

Solution – II: Learning Rate Scheduling

- Adjust the learning rate during training by reducing the learning rate at per predefined scheduled

- Common schedulers –
  - step decay
  - exponential decay
  - cosine decay
  - reduce on plateau
  - etc.

```python
import torch.optim as optim


scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[30,80], gamma=0.1)
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.1)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode = 'min')
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer)
```

Solution – III: Weight Initialization

- All zero initialization
  - Initializing all the weights with zeros leads the neurons to learn the same features during training.

- Random initialization
  - Gaussian
  - Xavier
    - uniform
    - normal
  - Kaiming
    - uniform
    - normal

```python
import torch

w = torch.empty(3, 5)
torch.nn.init.kaiming_uniform_(w, mode='fan_in', nonlinearity='relu')
```

Questions?