NEURAL NETWORKS

JAMIE CLINE

- What is a neural network?
- How do they learn?
- The mathematics involved

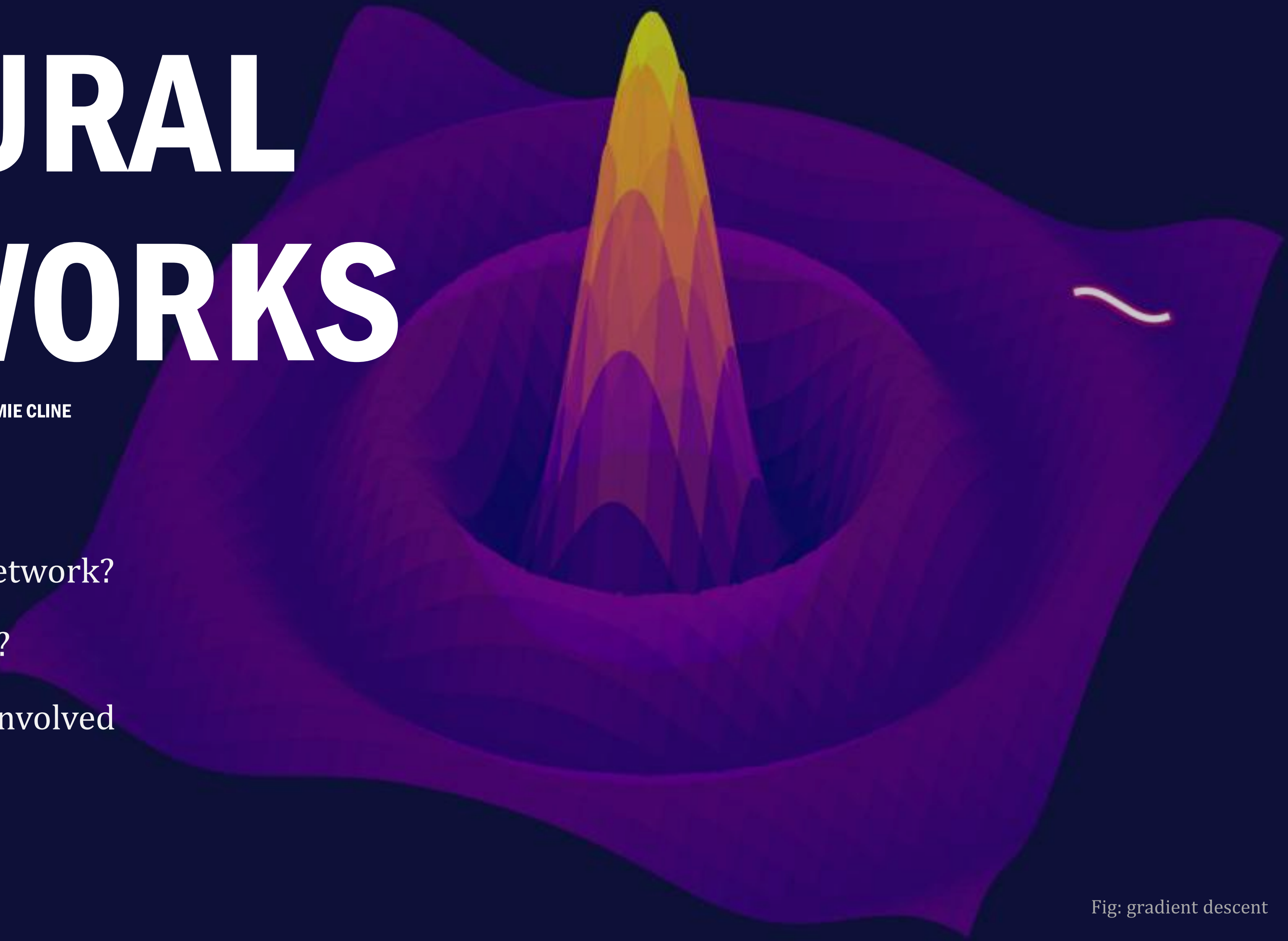


Fig: gradient descent

CONTENTS

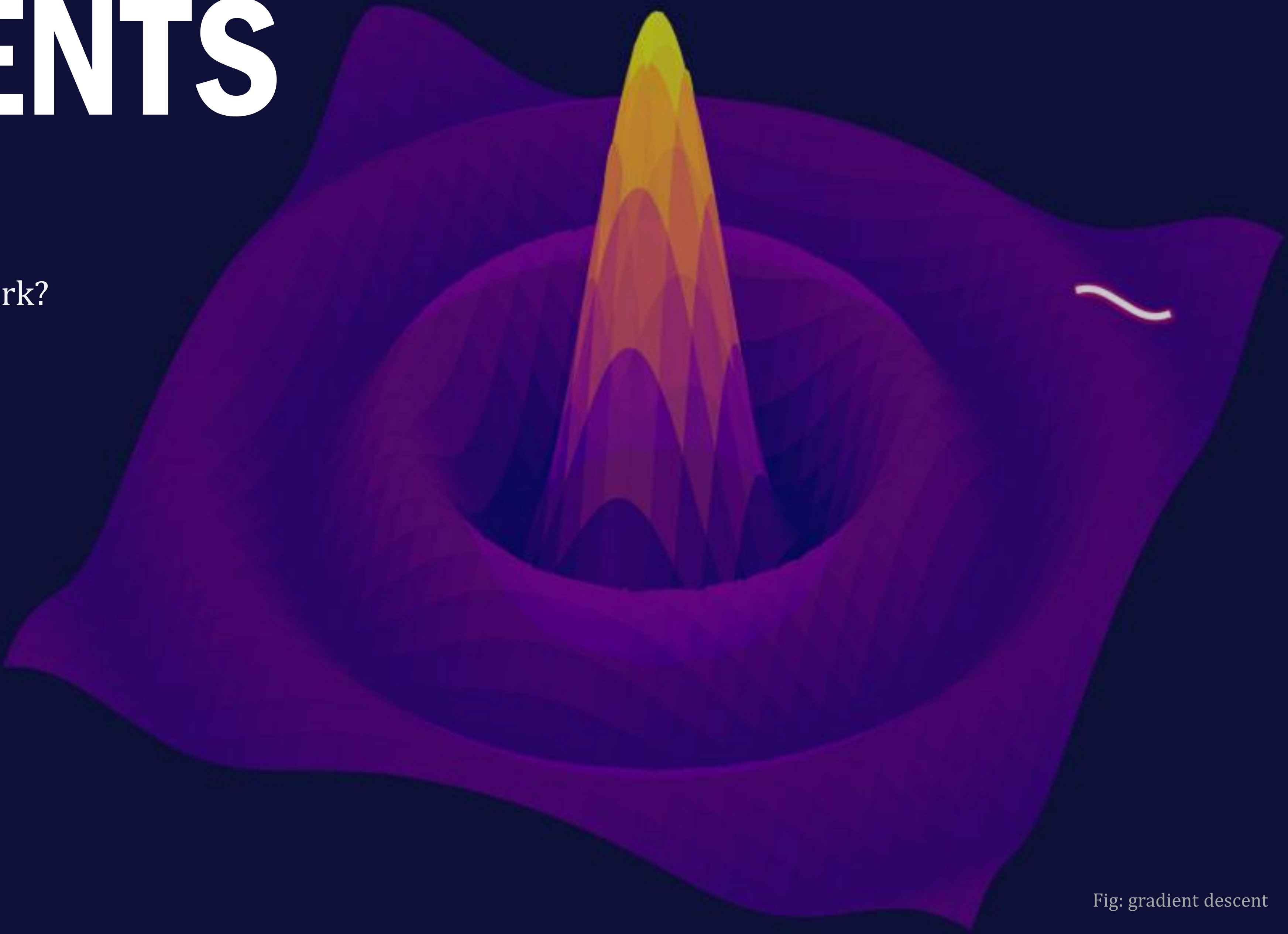


Fig: gradient descent

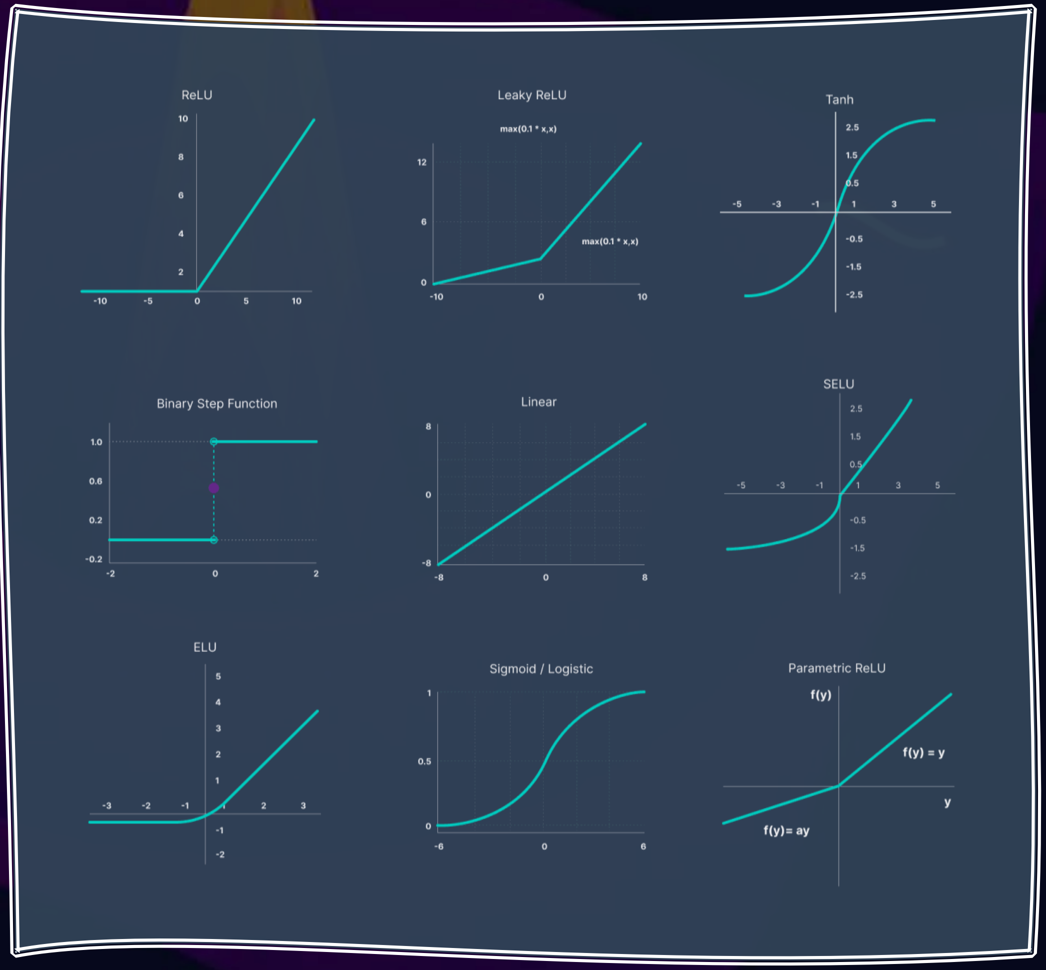
Intro to Neurons & Layers

- Neural networks are functions made of neurons and layers.
- Each neuron stores its activation
- A non-linear activation function is applied

$$z^{(\ell+1)} = \boldsymbol{W}^{(\ell+1)} a^{(\ell)} + b^{(\ell+1)}$$

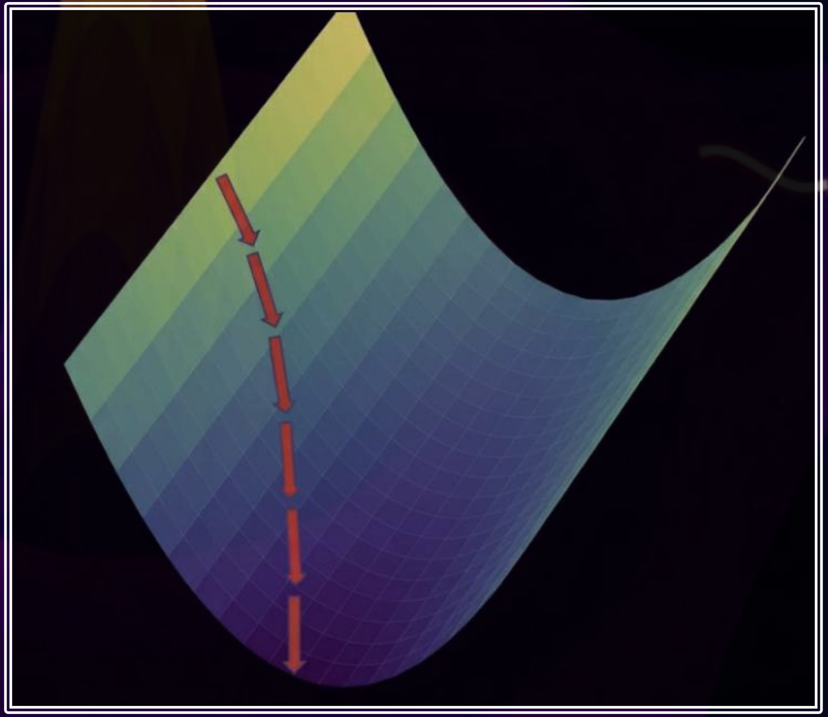
$$a^{(\ell+1)} = \sigma\left(z^{(\ell+1)}\right)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



How do they learn?

- The weights, biases, and learning rate tunes the model.
- The cost function tells the network how good the parameters are.
- Gradient descent is used to minimize C



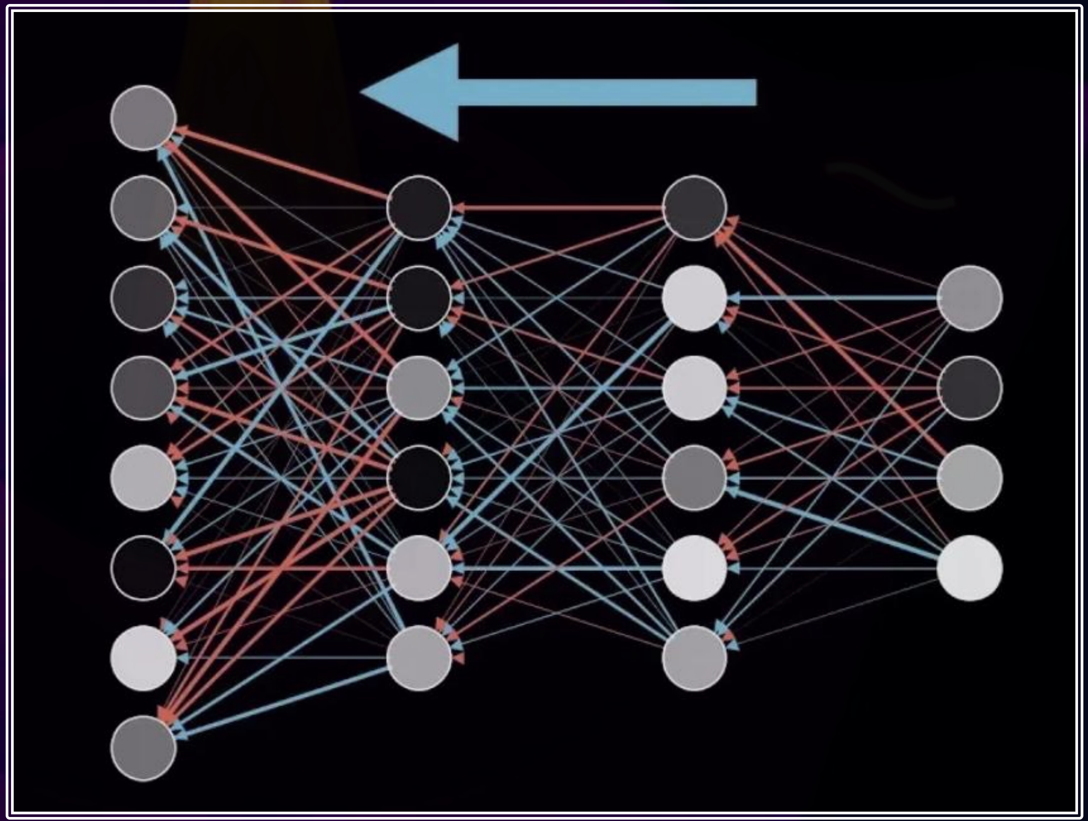
$$C_n = \frac{1}{n}\sum_{j}(\mathrm{a}_{\mathrm{j}}^{(\ell)} - \mathrm{y}_{\mathrm{j}})^2$$

$$params \coloneqq params - \eta\nabla C$$

Backpropagation

- How do we know how much each weight effects the final error?
- Output layer requests change in previous layer
- Requests propagate backwards

$$\partial C/\partial a^{(\ell)} = 2\left(a^{(\ell)} - y\right)$$
$$\partial \mathrm{a}^{(\ell)}/\partial z^{(\ell)} = \sigma'(z^{(\ell)})$$
$$\partial z^{(\ell)}/\partial w^{(\ell)} = a^{(\ell-1)}$$
$$\frac{\partial C}{\partial w^{(\ell)}} = 2\left(a^{(\ell)} - y\right)\sigma'\left(z^{(\ell)}\right)a^{(\ell-1)}$$



Weight Decay

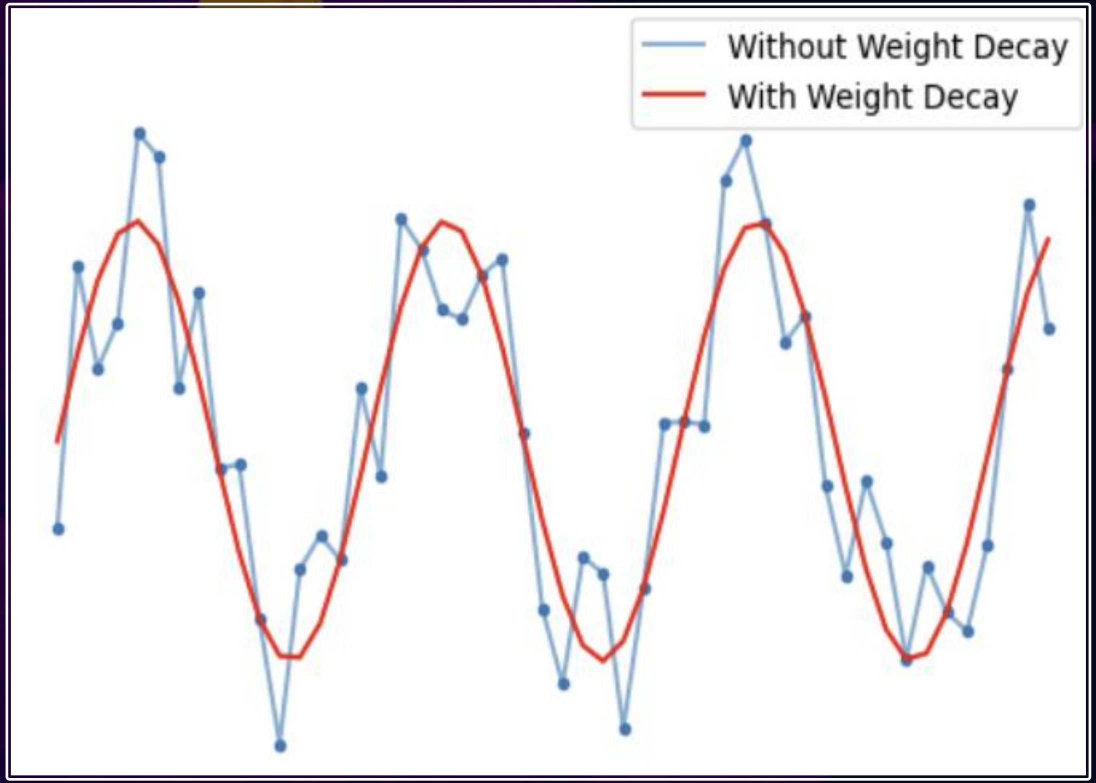
- Adjusting parameters like this can cause overfitting
- Adding regularization to the cost can reduce this.

$$C_{total} = C_{data} + \lambda R(W)$$

$$\left\| W^{l} \right\|_F^2 = \sum_{i,j} \left(w_{ij}^{(l)} \right)^2$$

$$R(W) = \sum_l \left\| W^{(l)} \right\|_F^2$$

$$W^{(l)} \coloneqq W^{(l)} - \eta\left(\frac{\partial C_{data}}{\partial W^{(l)}} + 2\lambda W^{(l)}\right)$$



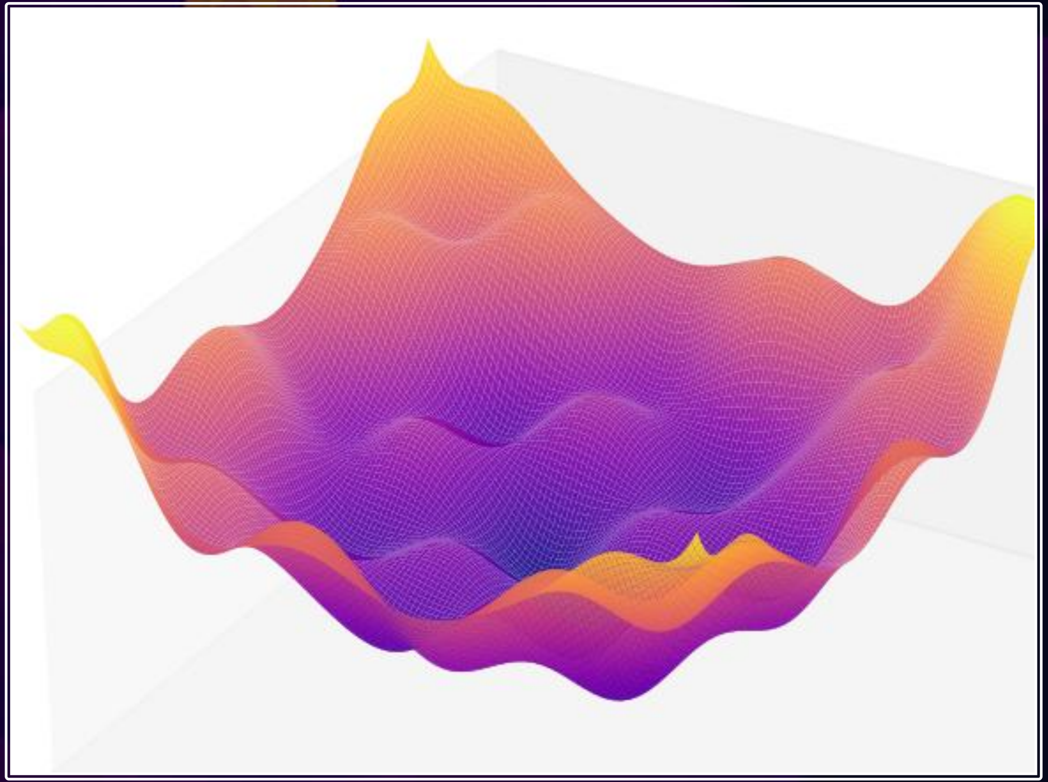
The Hessian

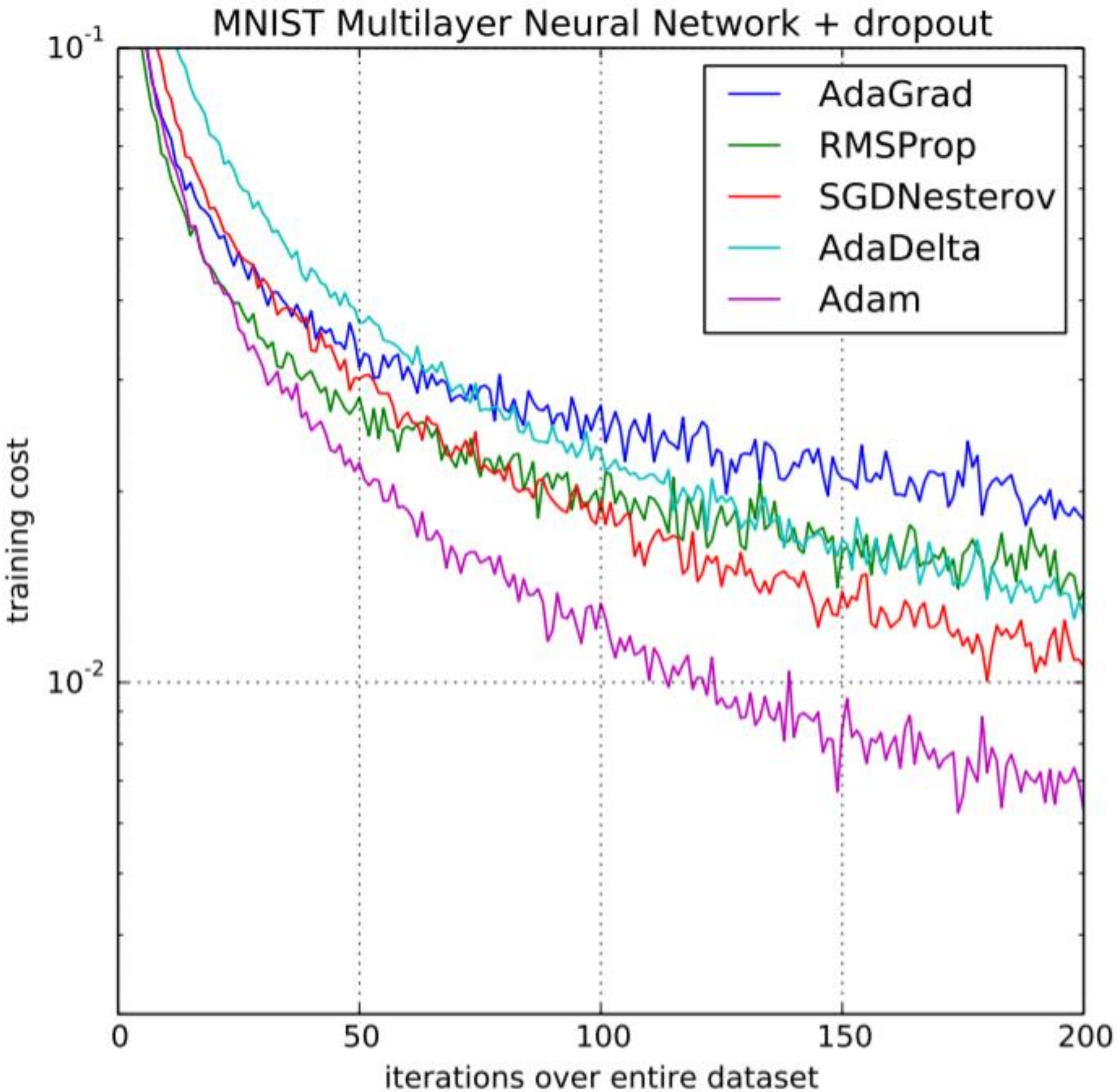
- The hessian is a matrix of second derivatives.
- We can approximate cost with Taylor expansion up to second order close to θ.

$$g(\theta) = \nabla_\theta C(\theta)$$

$$H(\theta) = \nabla_\theta^2 C(\theta)$$

$$C(\theta + \Delta) \approx C(\theta) + g(\theta)^T \Delta + \frac{1}{2} \Delta^T H(\theta) \Delta$$





Adam Optimisation

Adam (AME) Optimisation

- Incorporates momentum into gradient descent
- Keeps track of two running averages
- Direction moment :

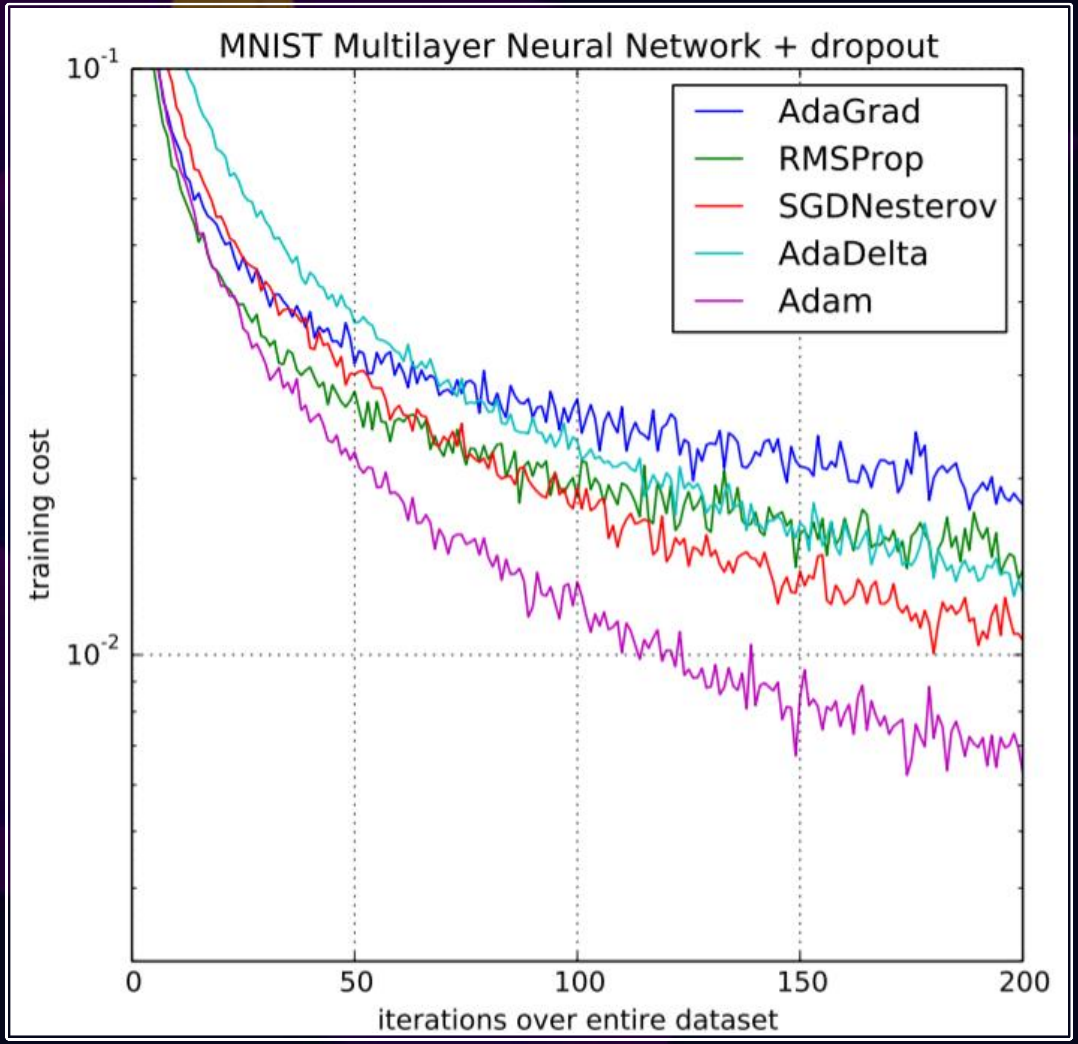
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- Scale of curvature moment:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- And then updates each parameter:

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$



CONCLUSION

- Neural networks can approximate any function
- Layers and activations give universal expressivity
- Training adjusts weights to minimize cost
- Regularization and optimization shape performance
- Foundation of modern AI and scientific computing

main.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a grid
x = np.linspace(-3, 3, 400)
y = np.linspace(-3, 3, 400)
X, Y = np.meshgrid(x, y)
R = X**2 + Y**2

# Cost function
Z = np.sin(R)+(R) + 2*np.sin(3*X)*np.sin(3*Y)

# Plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='plasma', rstride=3, cstride=3, linewidth=0, alpha=0.9)

# Styling
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.set_box_aspect((1,1,0.5))
plt.tight_layout()
plt.show()
```