

$$z = e^{-(x+1)^2 - (y-1)^2} + 0.6e^{-(x-1)^2 - 0.5(y+1)^2 + 0.1(x-3)(y-3)}$$

Gradient descent, quadratic minima, differing length scales

ML = Optimisation

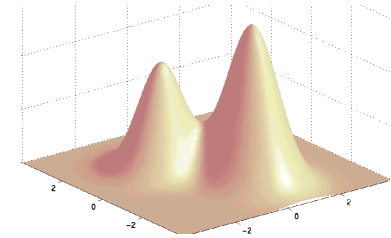
- Many learning machines can be thought of as functions of the form

$$\hat{y} = f(\mathbf{x}|\mathbf{w})$$

(or more generally $\hat{y} = \mathbf{f}(\mathbf{x}|\mathbf{w})$)

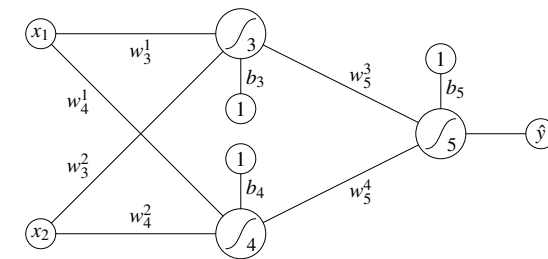
- Given an input pattern (set of features) \mathbf{x} the learning machine makes a prediction \hat{y}
- We try to choose the parameters \mathbf{w} so that the predictions are good
- In practice training a learning machine comes down to optimising some loss function

- Motivation**
- Gradient Descent
- Why Gradient Descent is Difficult



MLP

- We can depict a neural network such as an MLP by a diagram



- Stands for the function $(\hat{y} = f(\mathbf{x}|\mathbf{w}))$

$$\hat{y} = g(w_5^3 g(w_3^1 x_1 + w_3^2 x_2 + b_3) + w_5^4 g(w_4^1 x_1 + w_4^2 x_2 + b_4) + b_5)$$

where, for example, $g(V) = \frac{1}{1+e^{-V}}$

Training

- Given a (labelled) training dataset

$$\mathcal{D} = \{(\mathbf{x}_k, y_k) \mid k = 1, \dots, m\}$$

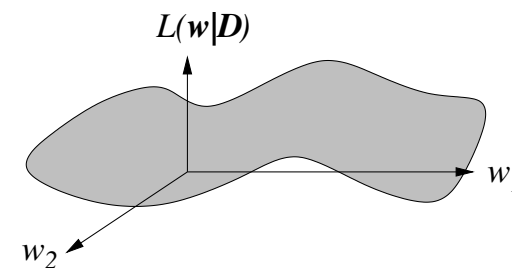
- We define an error or loss function that we want to minimise

$$L(\mathbf{w}|\mathcal{D}) = \frac{1}{m} \sum_{k=1}^m (f(\mathbf{x}_k|\mathbf{w}) - y_k)^2$$

- We then use the machine with the weights \mathbf{w}^* which minimise $L(\mathbf{w}|\mathcal{D})$

Computing Gradients

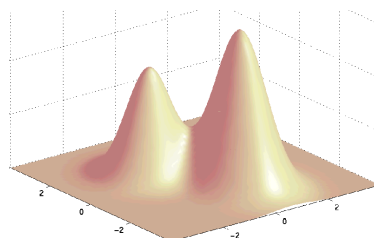
- $L(\mathbf{w}|\mathcal{D})$ is a complex function of the weights \mathbf{w}



- To minimise we $L(\mathbf{w}|\mathcal{D})$ we compute the gradient $\nabla L(\mathbf{w}|\mathcal{D})$
- For neural networks an efficient algorithm for computing the gradient is known as back-propagation of errors (*back-prop*)

Outline

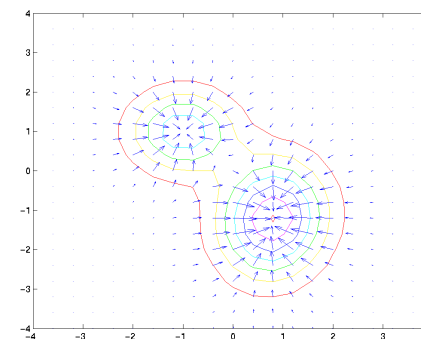
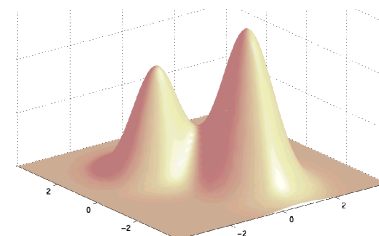
- Motivation
- Gradient Descent**
- Why Gradient Descent is Difficult



Gradient Optimisation

- A maximum or minimum occurs when $\nabla L(\mathbf{w}|\mathcal{D}) = \mathbf{0}$
- E.g.

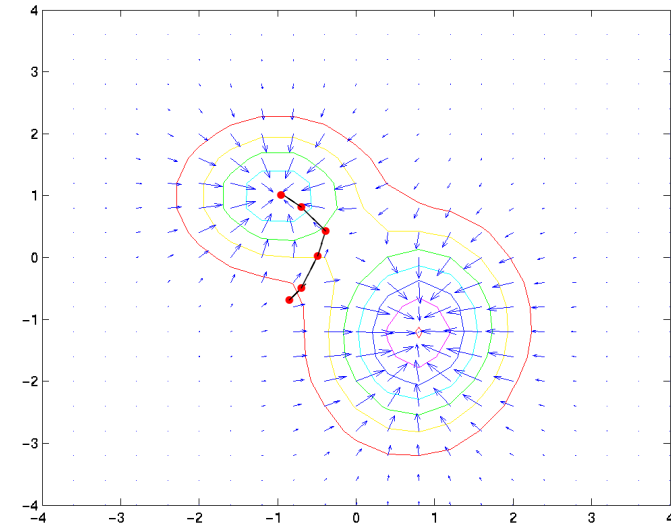
$$L = e^{-(x+1)^2 - (y-1)^2} + 0.6e^{-(x-1)^2 - 0.5(y+1)^2 + 0.1(x-3)(y-3)}$$



Gradient Descent

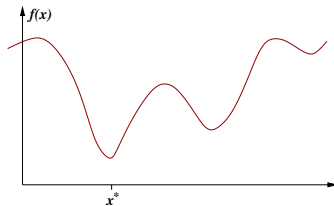
- For a simple function $L(w|\mathcal{D})$ we can solve $\nabla L(w|\mathcal{D}) = 0$ explicitly. E.g. the linear perceptron
- For a non-linear functions we usually can't solve this set of simultaneous equations
- We can find a maximum or minimum **iteratively**
- If we know the gradient then we can follow the gradient
 - ★ Maximisation: $w \rightarrow w' = w + r \nabla L(w|\mathcal{D})$
 - ★ Minimisation: $w \rightarrow w' = w - r \nabla L(w|\mathcal{D})$

Hill-Climbing



What Goes Right

- Almost all minima are quadratic (Morse's theorem)



- Taylor expanding around a minimum x^*

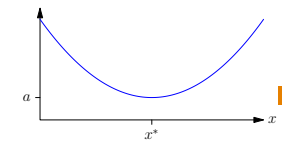
$$\begin{aligned} f(x) &= f(x^*) + (x - x^*)f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \dots \\ &= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \dots \end{aligned}$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x - x^*)^2$$



- then

$$f'(x) = b(x - x^*), \quad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newton's method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4 , etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions
- The Taylor expansion of a function $f(x)$ about x_0

$$f(x) = f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2} (x - x_0)^T \mathbf{H} (x - x_0) + \dots$$

where \mathbf{H} is the **Hessian** matrix with elements

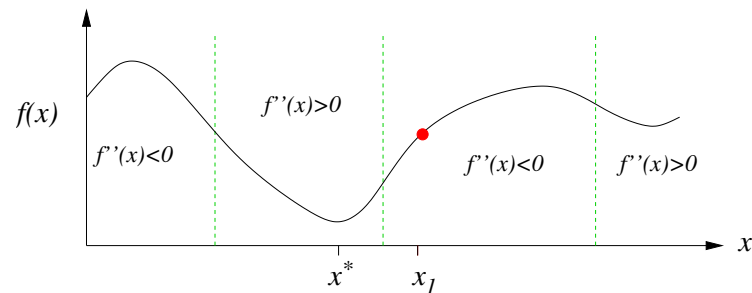
$$H_{ij} = \frac{\partial^2 f(x_0)}{\partial x_i \partial x_j}$$

- Newton's method in high dimension is

$$x^* = x - \mathbf{H}^{-1} \nabla f(x)$$

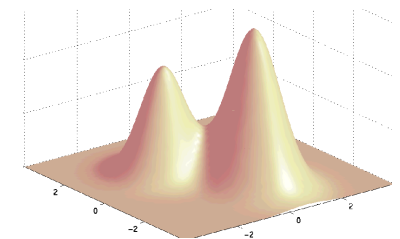
Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding) — for deep learning it is impossible even to store the Hessian
- Away from minima they can be misleading



Outline

1. Motivation
2. Gradient Descent
3. **Why Gradient Descent is Difficult**



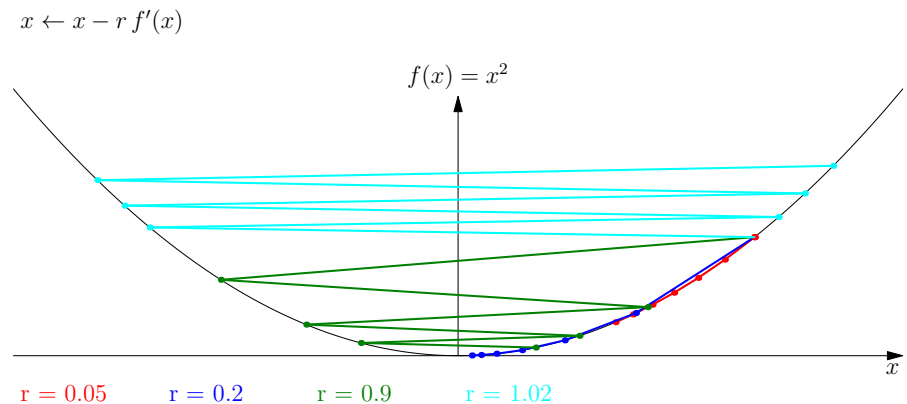
Step Size

- Gradient descent

$$x' = x - r \nabla f(x)$$

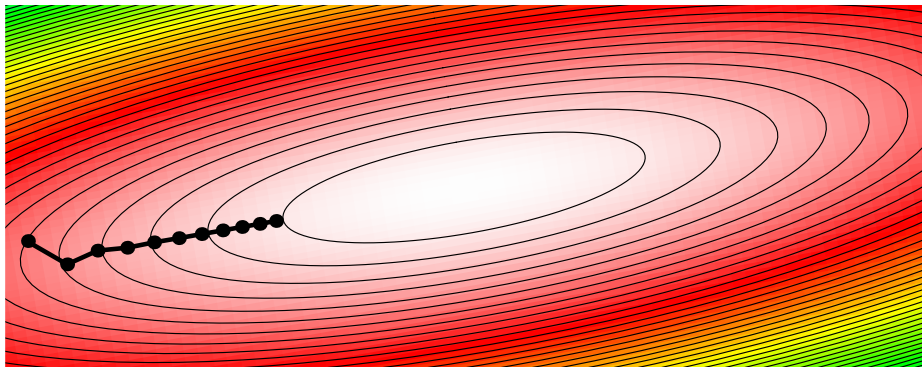
- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

Step Size



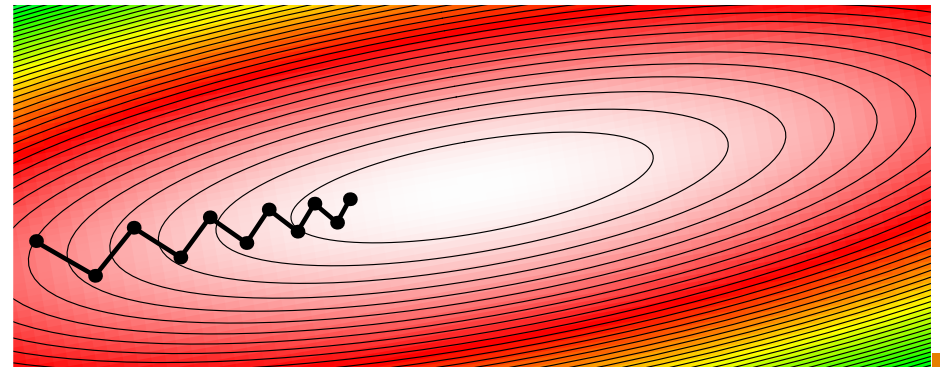
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



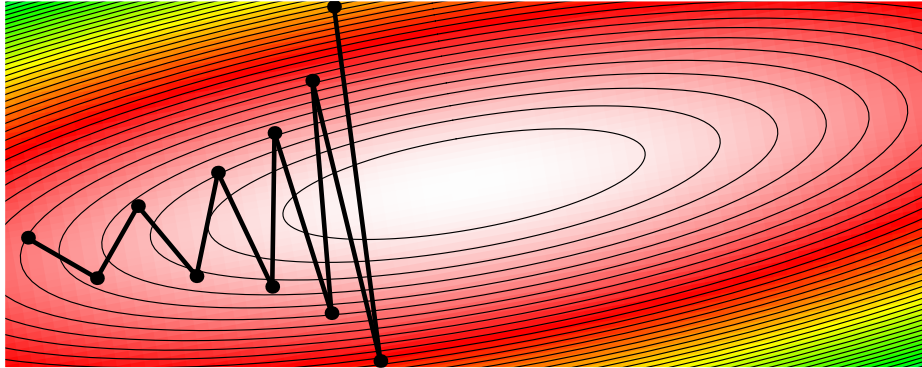
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



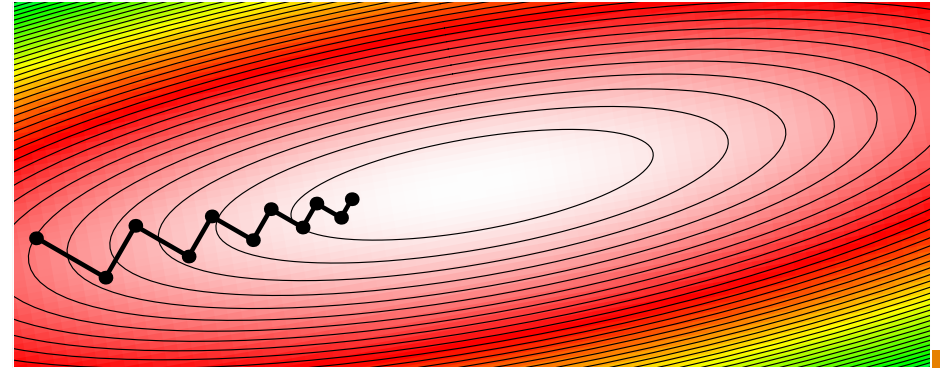
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



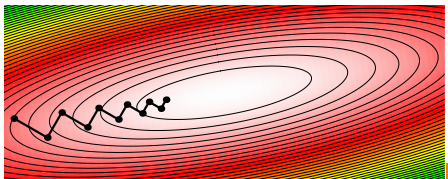
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag



- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential
- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
 - Performs Line Minimisation
 - Uses gradient, but does not go along it
 - For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
 - Used on least squares problem only
 - Uses linear approximation of function to approximate Hessian
 - Adapts from hill-climbing to Newton method
 - Avoids line-minimisation

Levenberg-Marquardt

- Want to minimise $\|\epsilon(\mathbf{w})\|^2$ where $\epsilon_i(\mathbf{w}) = f(\mathbf{x}_i|\mathbf{w}) - y_i$
- Use linear approximation

$$\epsilon_i(\mathbf{w}) \approx \epsilon_i(\mathbf{w}^{(k)}) + (\mathbf{w} - \mathbf{w}^{(k)})^\top \nabla \epsilon_i(\mathbf{w}^{(k)})$$

$$\text{with } \nabla \epsilon_i(\mathbf{w}^{(k)}) = \nabla f(\mathbf{x}_i|\mathbf{w}^{(k)})$$

- Solve quadratic minimisation of approximate error
 $\text{argmin}_{\mathbf{w}} L_{\text{approx}}(\mathbf{w})$ with $\mathbf{J} = \nabla \epsilon(\mathbf{w}^{(k)})$

$$\begin{aligned} L_{\text{approx}}(\mathbf{w}) &= \|\epsilon(\mathbf{w}^{(k)}) + \mathbf{J}(\mathbf{w} - \mathbf{w}^{(k)})\|^2 \\ &= \epsilon(\mathbf{w}^{(k)})^\top \epsilon(\mathbf{w}^{(k)}) + 2(\mathbf{w} - \mathbf{w}^{(k)})^\top \mathbf{J}^\top \epsilon(\mathbf{w}^{(k)}) \\ &\quad + (\mathbf{w} - \mathbf{w}^{(k)})^\top \mathbf{J}^\top \mathbf{J} (\mathbf{w} - \mathbf{w}^{(k)}) \end{aligned}$$

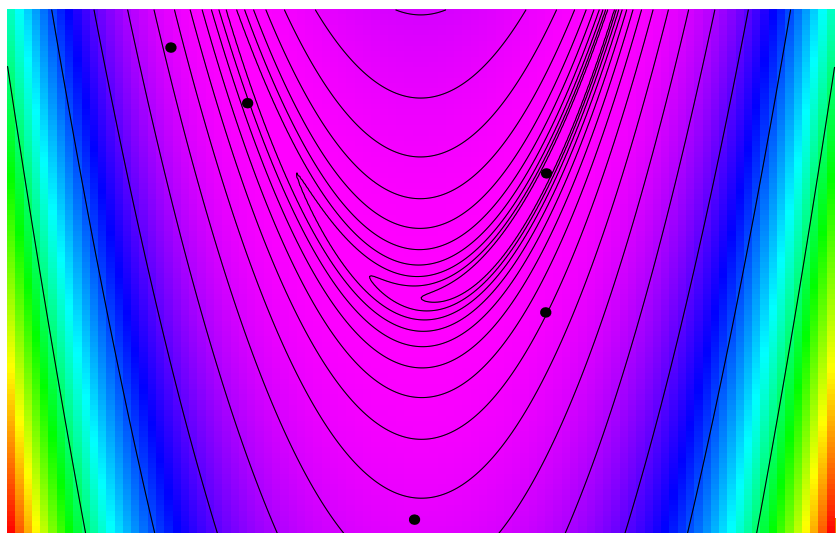
Trust Region

- Solution given by $\nabla_{\mathbf{w}} L_{\text{approx}}(\mathbf{w}) = 0$ gives

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \epsilon(\mathbf{w}^{(k)})$$

- Can lead us in the wrong direction
- Instead use $\mathbf{w}^{(k+1)} = \text{argmin}_{\mathbf{w}} L_{\text{approx}}(\mathbf{w}) + \nu \|\mathbf{w} - \mathbf{w}^{(k)}\|^2$
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - (\mathbf{J}^\top \mathbf{J} + \nu \mathbf{I})^{-1} \mathbf{J}^\top \epsilon(\mathbf{w})$$
- ν limits the step size
- If predicted reduction in error is accurate then reduce ν , else if predicted reduction in error is very poor increase ν

$$\epsilon_1 = 10(x_2 - x_1^2) \text{ and } \epsilon_2 = 1 - x_1$$



Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian