Forward propagation with implicit layers

Fixed points as points of no change

- A fixed point for a function $f : \mathbb{R}^n \to \mathbb{R}^n$ is a point $z \in \mathbb{R}^n$ for which the function has no influence on the point
- In other words, the input and output of f is the same: $z = f(z)$
- Optimisation, e.g., gradient descent, is a fixed point problem

$$w_{t+1} = w_t - \eta \frac{\partial \ell}{\partial w} \Rightarrow$$

$$w = f(w) \text{ where } f(w) = w - \eta \frac{\partial \ell}{\partial w}$$

- If f is differentiable, then we can deploy our favourite auto-diff library

Fixed points with parameters

- More generally, in our neural networks we also have parameters
- We can then have a parameterised fixed-point problem with function $f : \mathbb{R}^p \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, which is parameterised by $\alpha \in \mathbb{R}^p$

$$z = f(a, z)$$

- For this α-parameterised system of equations we can ask things like how will the fixed point change with α?

Fixed-point iteration

- We have a function with a “self-dependency”: $z = f(a, z)$
- Hopefully, by repeatedly applying the function, the output should converge to a “fixed” value z^*
- For instance, for a neural network layer that feeds itself: $z = \tanh(Wz + x)$

$$z^{(t=2)} = \tanh(Wz^{(t=1)} + x)$$
$$z^{(t=3)} = \tanh(Wz^{(t=2)} + x)\ldots$$

such that $z^{(t)} \approx z^{(t-1)} = z^*$

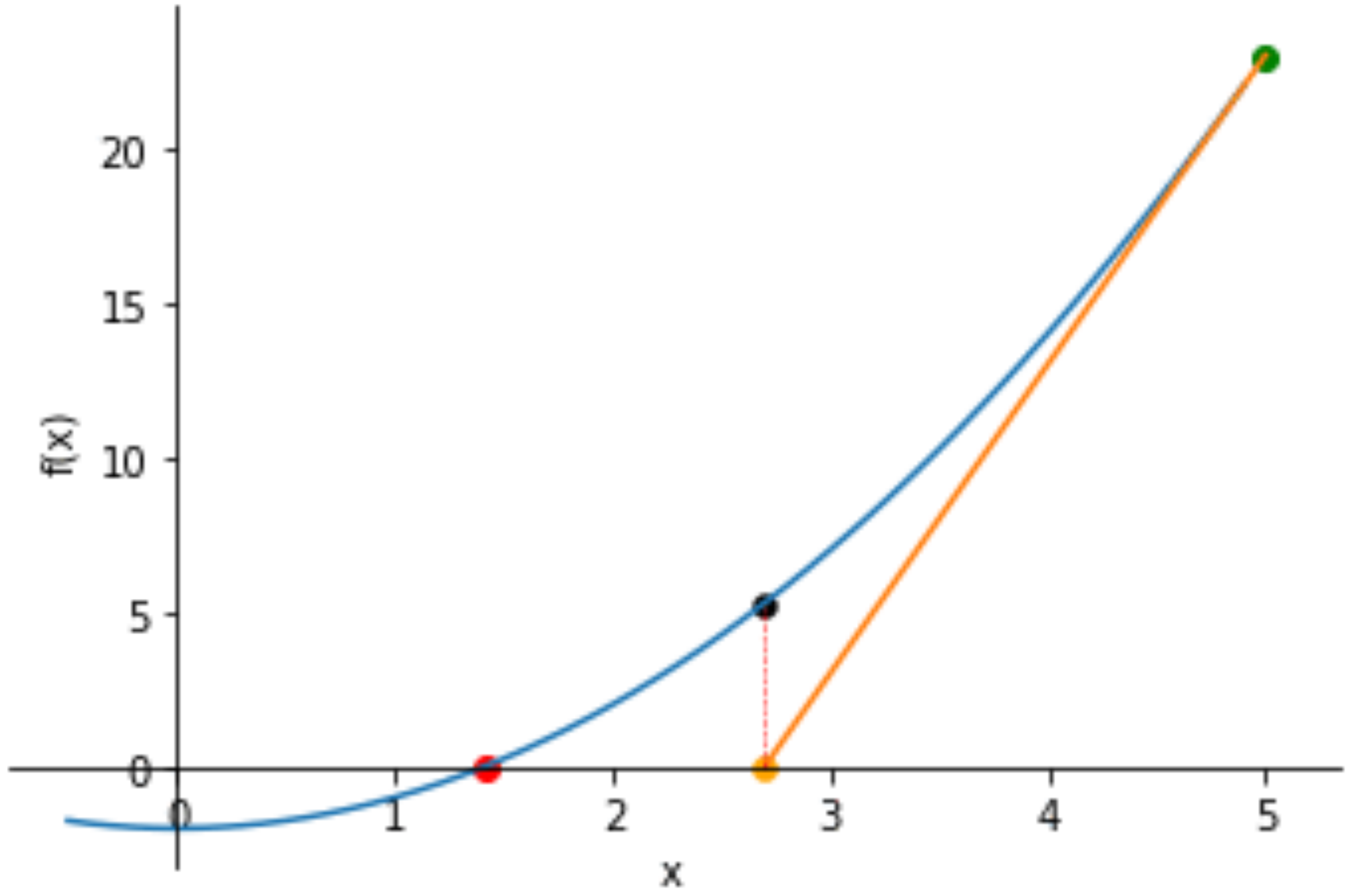
- Relates to recurrent backdrop

Fixed-points & root-finding

- Fixed-point problems can be rewritten as system of (non-linear) equations
- Then, finding the fixed point is equivalent to root-finding

$$z = f(x, z) \Rightarrow g(x, z) = z - f(x, z) = 0$$

- We want to solve the equation $g(x, z) = 0$
  - For instance, solve $g(x) = x^2 - 2 = 0$
  - We iteratively improve on a an initial solution
  - Until we converge to the "root" of $x^2 - 2 = 0$



Convergence

- $z = f(x, z)$ not need always to converge to a fixed value $z^\star$
- It can also diverge or oscillate
- If the function $f(x, z)$ is on the real line with real values and Lipschitz continuous with a Lipschitz constant $L < 1$ (does not change too fast), then it converges (wiki)

$$|z^{(t)} - z^{(t-1)}| \leq L^{t-1}|z^{(1)} - z^{(0)}|$$

- For the usual NNs and nonlinearities we can assume convergence

Root-finding: Naive forward iteration

- Simplest way to solve the root: naive iterations

$$z^{(t=1)} = \text{random init}$$
$$z^{(t=2)} = \tanh(Wz^{(t=1)} + x)$$
$$z^{(t=3)} = \tanh(Wz^{(t=2)} + x)\ldots$$

- Still, may take large number of iterations
- Also, might not even reach the min tolerance or diverge for certain parameter values

```
# iterate until convergence
while self.iterations < self.max_iter:
    z_next = torch.tanh(self.linear(z) + x)
    self.err = torch.norm(z - z_next)
    z = z_next
    self.iterations += 1
    if self.err < self.tol:
        break
```

Root-finding: Newton's method

- As we define the implicit layer abstractly $g(x, z) = 0$, any root-finding algorithm works
- A good alternative is Newton's method that uses the Jacobian (or quasi-Newton)

$$\boxed{\text{For } g : \mathbb{R}^n \to \mathbb{R}^n, \;\; z := z - \left(\frac{\partial g}{\partial z}\right)^{-1} g(z)}$$

- For the Jacobian $\dfrac{\partial g}{\partial z}$ we can use our auto-diff libraries (PyTorch, Jax, ...)

Root-finding: Newton's method

$$\boxed{\text{For } g : \mathbb{R}^n \to \mathbb{R}^n, \;\; z := z - \left(\frac{\partial g}{\partial z}\right)^{-1} g(z)}$$

- The Jacobian and its inverse can be quite expensive, especially for many iterations
- We must store intermediate states in memory
- Also, backpropagating with long chains and repeating inverses can be computationally unstable when inverse close to singular (determinant $\to 0$)
- Forward prop might converge $\leftarrow$ still backprop gradients might be with errors