Continuing our introduction to optimization, let's see an important application:

Least Squares Let $f(x) = \frac{\|Ax-b\|_2^2}{2}$

i.e we want to minimize

Then we can compute

$$\nabla f(x) = A^T(Ax-b)$$

$$\nabla^2 f(x) = A^TA$$

From this we see that $f(x)$ is strongly convex iff A^TA has full rank

What does gradient descent look like, specialized to this problem?

$$x_{t+1} = x_t - \eta A^T r_t$$

where $r_t \overset{\Delta}{=} Ax_t - b$

is the <u>residual</u>

Variants

There are some important variations on vanilla GD

Stochastic Gradient Descent

What do you do if the function you want to minimize is too big to fit in memory?

e.g. $R(\underset{\uparrow}{f}(x; w)) = \sum_{i=1}^{N} \ell(f(x_i; w), y_i)$

one term for each example you want to label

```
For t=1 to T
    Choose i at random
    Set $w_{t+1} = w_t - \eta \nabla_w \ell(f(w_i; w), y_i)$
```

Can also choose a set of k random exs., called a minibatch, to estimate overall gradient

Nesterov Acceleration

Can also incorporate information about past gradients to converge even faster

```
For $t = 1$ to $T$
    set $y_{t+1} = x_t - \eta \nabla f(x_t)$
    set $x_{t+1} = y_{t+1} + \mu \underbrace{(y_{t+1} - y_t)}_{\text{momentum}}$
```

This achieves a squareroot improvement:

GD $\sim \frac{\beta}{\alpha}$ iterations to reduce distance to optimum by a constant factor

AGD $\sim \sqrt{\frac{\beta}{\alpha}}$ iterations

Gradient Flow

For theoretical convenience sometimes it will be helpful to work with the continuous time limit — i.e. the ODE

$$\dot{x}(t) = -\nabla f(x(t))$$

This is the natural limit of taking the stepsize to zero

$$x_{k+1} = x_k - \eta \nabla f(x_k)$$

$$\xrightarrow{\text{rearranging}} \quad \frac{x_{k+1} - x_k}{\eta} = -\nabla f(x_k)$$

Taking a step back, let's say outloud what is so great about <u>convex</u> optimization:

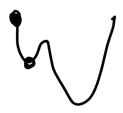
<u>many different algorithms</u>

$$\left.\begin{matrix} \text{GD} \\ \text{SGD} \\ \text{AGD} \\ \vdots \end{matrix}\right. \rightsquigarrow \text{same } x^*, \text{ provably optimal}$$

But in non-convex optimization, like when we are training a deep net, all that goes out the window

We could try to apply GD/SGD/AGD... but what goes wrong?

<u>Issue #1</u>: Can't find the global minimum

E.g. 

Usually we find a local minimum, and in general finding a global minimum is NP-hard

<u>Issue #2</u>: which point we reach depends on our initialization, the randomness of our algorithm

Usually there are tons of points that achieve <u>zero</u> risk

And which one we reach is important for generalization

(ref: "sharp" vs. "flat" minima)

So, the journey is more important than the destination

Neural Tangent Kernels

Next, we will prove a fundamental result about fitting an <u>overparameterized</u> deep net to data

First, fix any training set of size N. Then:

> **"For a wide enough deep net, gradient descent finds a 0-error solution"**

There are some caveats:

1. The network needs to have depth ≥ 2, right nonlinearities

(2) How wide is wide enough?
Arora et al. showed that poly(N) suffices

(3) This all depends on having the right scaling parameters for the random initialization

<u>Historical Notes</u>: This line of work was kicked off by an important work of Jacot, Gabriel and Hongler

In hindsight, concurrent works of Du et al. and Allen-Zhu et al. exploited the same principles

We will follow the "lazy training" framework of Chizat and Bach, which is an abstraction of what happens as you increase the width

The main ideas are:

1. If we take the linear approximation to a deep network, if width > # examples, we can fit perfectly

2. As we increase the width, we need to move less in the parameter space

$\Longrightarrow$ Thus the Jacobian doesn't change too much

The Gradient Flow

For convenience, let's suppress the datapoints:

$$f(w) \triangleq \begin{bmatrix} f(x_1; w) \\ \vdots \\ f(x_N; w) \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Throughout, we want to minimize the squared loss:

$$R(\alpha f(w)) \triangleq \frac{1}{2} \|\alpha f(w) - y\|^2$$

Here α is a scaling parameter

$$\text{increasing the width} \longleftrightarrow \text{increasing } \alpha$$

More on this later.

We will be interested in the behavior of the gradient flow:

$$\begin{aligned} \dot{w}(t) &= -\nabla_w R(\alpha f(w(t))) \\ &= -\alpha J_t^T \nabla R(\alpha f(w(t))) \\ &= -\alpha J_t^T (\alpha f(w(t)) - y) \end{aligned}$$

where J_t is the Jacobian at time t

i.e. $$J_t \triangleq \begin{bmatrix} \nabla f(x_1; w(t))^T \\ \vdots \\ \nabla f(x_N; w(t))^T \end{bmatrix}$$

Assuming w is a vector of p parameters, this is an $N \times p$ matrix

<u>Main Theorem</u> (informal) If α is big enough, then

$$\textcircled{1} \quad R(\alpha f(w(t))) \leq R(\alpha f(w(0)))\, e^{-ct\alpha^2}$$

$$\textcircled{2} \quad \| w(t) - w(0) \| \leq \frac{\sqrt{R(\alpha f(w(0)))}}{\alpha}$$

<u>Proof Strategy</u>: We will define an auxilliary flow, $u(t)$, <u>using a linear approximation to</u> f and show $u(t)$ and $w(t)$ remain close

In particular, define

$$f_0(u) \triangleq f(w(0)) + J_0(u - w(0))$$

which gives us

$$\dot{u}(t) = -\nabla_u R(\alpha f_0(u(t)))$$
$$= -\alpha J_0^T \nabla R(\alpha f_0(u(t)))$$
$$= -\alpha J_0^T(\alpha f_0(u(t)) - y) \quad (*)$$

<u>Lemma 1</u>: If $J_0 J_0^T$ is full rank, then $R(\alpha f_0(u(t)))$ goes to zero

<u>Proof</u>: Let's compute the change in predictions:

$$\frac{d}{dt} \alpha f_0(u(t)) = \frac{d}{dt}\left[\alpha\left(f(w(0)) + J_0(u(t) - w(0))\right)\right]$$
$$= \alpha J_0 \dot{u}(t)$$

Substituting $(*)$ we have:

$$= -\alpha^2 J_0 J_0^T (\alpha f_0(u(t)) - y)$$

Thus we conclude

$$\frac{d}{dt}(\alpha f_0(u(t)) - y) = -\alpha^2 J_0 J_0^T (\alpha f_0(u(t)) - y)$$

Let's call $r(t) \triangleq \alpha f(u(t)) - y$ the <u>residual prediction</u>. Then

$$\dot{r}(t) = -\alpha^2 J_0 J_0^T r(t)$$

$$\Rightarrow r(t) = e^{-\alpha^2 J_0 J_0^T t} r(0)$$

And this converges to zero by the full rank assumption ☒

<u>Key definition</u>: The matrix $K \triangleq J_0 J_0^T$ is called the <u>neural tangent kernel</u>

In particular recall $J_0 J_0^T =$

$$\begin{bmatrix} \nabla f(x_1; w(0))^T \\ \vdots \\ \nabla f(x_N; w(0))^T \end{bmatrix} \begin{bmatrix} \nabla f(x_1; w(0))^T \ldots \nabla f(x_N; w(0))^T \end{bmatrix}$$

Thus $K(x_i, x_j) = \nabla f(x_i; w(0))^T \nabla f(x_j; w(0))$

It is known that under the right scaling, $f(\cdot; w(0))$ converges to a <u>Gaussian</u> <u>Process</u>

An interpretation

Actually the gradient flow for u is something we have already seen, but in disguise

Recall, for least squares:

$$\min_x \frac{\|Ax - b\|^2}{2}$$

gradient descent turned into

$$x_{t+1} = x_t - \eta A^T r_t$$

where $r_t = A x_t - b$

Now we can rewrite this as a recurrence for r_t instead

$$\Rightarrow A x_{t+1} = A x_t - \eta A A^T r_t$$

$$\Rightarrow A x_{t+1} - b = A x_t - b - \eta A A^T r_t$$

$$\Rightarrow r_{t+1} = r_t - \eta A A^T r_t$$

Thus the limit as we take the stepsize to zero is

$$\dot{r}(t) = - A A^T r(t)$$

This looks like what we had with

$$A \longleftrightarrow J_0$$

Explicitly, consider

$$\min_u \frac{\|f_0(u) - y\|^2}{2} =$$

$$\min_u \frac{\|J_0 u - (\overbrace{y + J_0 w(0) - f(w(0))}^{b})\|^2}{2}$$

Aha! So that's what u is doing:

"u is solving a least squares problem using the linear approx. to the deep net at initialization"

An Interlude

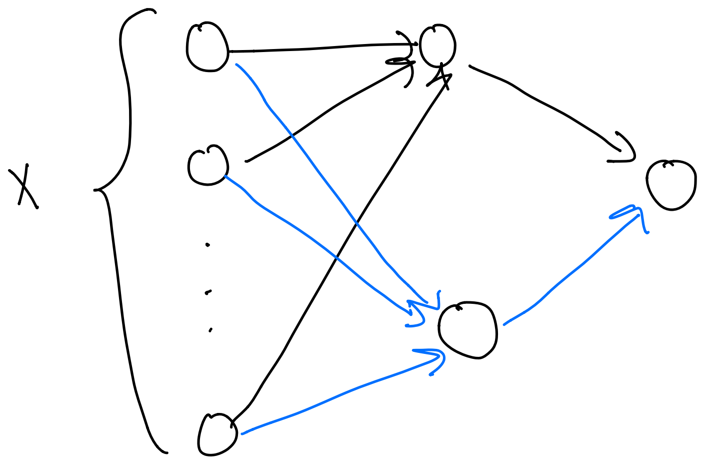
Why is α an accurate abstraction of what happens when we increase the width?

Intuitively as $\alpha \to \infty$ we are zooming in at $w(0)$ so the linear approx. is more accurate

And as we increase the width, we need to move less to affect the output

Thought Experiment

what if we increase the width by literally duplicating the network?



Call the new function

$$\hat{f}(x; \underset{\uparrow}{w_1}, \underset{\uparrow}{w_2}) = 2f(x;w)$$

duplicates of w

Moreover $\nabla_{w_1} \hat{f}(x; w_1, w_2) = \nabla_{w_2} \hat{f}(x; w_1; w_2)$
$= \nabla_w f(x; w)$

Thus our new gradient flow would correspond to zooming in by doubling α

Analyzing the Gradient Flow

Now let's return to $w(t)$, and, as we did before for $u(t)$, study the evolution in prediction space:

$$\frac{d}{dt} \alpha f(w(t)) = \alpha J_t \dot{w}(t)$$

$$= -\alpha^2 \underset{\uparrow}{J_t J_t^\top} \nabla R(\alpha f(w(t)))$$

The main difference is this term now depends on t, so it is not exactly the gradient flow on the least squares obj.

Still it does not change much $\Rightarrow$ we can upper bound how fast risk decays

<u>Notation</u>

$$z(t) \triangleq \alpha f(w(t))$$

$$\Phi(t) \triangleq \alpha J_t J_t^\top$$

<u>Lemma 2</u>: Suppose

$$\dot{z}(t) = -\Phi(t) \nabla R(z(t))$$

and let $\lambda = \inf_{t \in [0,\tau]} \lambda_{\min}(\Phi) > 0$. Then

$$R(z(t)) \leq R(z(0))\, e^{-2t\lambda}$$

for any $t \in [0, \tau]$

<u>Detour</u>: Our proof strategy will be to upper bound

$$\frac{d}{dt} R(z(t))$$

and show that it is at most what it would be if $\Phi(t) = \lambda I$, which again corresponds to least squares

But how does this translate into a bound on $R(z(t))$?

This is answered by Gronwall's Inequality

<u>Fact:</u> Consider an interval $[a,b]$ and suppose

$$\dot{u}(t) \leq \beta(t)\, u(t) \quad \forall t \in [0,\tau]$$

Then u is upperbounded by the solution to

$$\dot{v}(t) = \beta(t)\, v(t)$$

assuming same boundary condition.
In particular

$$u(t) \leq \underbrace{u(0)\, e^{\int_a^t \beta(s)\, ds}}_{v(t)}$$

Now, returning to Lemma 2:

<u>Proof:</u> Using the definition of the risk we have

$$\frac{d}{dt}\frac{1}{2}\|z(t)-y\|^2 = \langle \dot{z}(t), z(t)-y\rangle$$

$$= \langle -\Phi(t)\nabla R(z(t)), z(t)-y\rangle$$

$$= \langle -\Phi(t)(z(t)-y), z(t)-y\rangle$$

$$\leq -2\lambda \frac{\|z(t)-y\|^2}{2}$$

Thus we have

$$\frac{d}{dt}R(z(t)) \leq -2\lambda\, R(z(t))$$

Now invoking Gronwall's Inequality, we have:

$$R(z(t)) \leq R(z(0))\, e^{-\int_0^t 2\lambda\, ds}$$

$$= R(z(0))\, e^{-2t\lambda}$$

$\blacksquare$

We also have the following useful corollary:

<u>Corollary:</u> $\|z(t) - y\| \leq \|z(0) - y\| e^{-t\lambda}$

This follows because

$$\|z(t) - y\| = \sqrt{2R(z(t))}$$

by definition, and similarly for $\|z(0) - y\|$

Ultimately, we want to show we never venture too far from the initialization

This'll be the reason we can relate $w(t)$ and $u(t)$

<u>Notation</u>

$$v(t) \triangleq w(t)$$
$$g(v(t)) \triangleq \alpha f(w(t))$$
$$s(t) \triangleq \alpha J t$$

Lemma 3: Suppose

$$\dot{v}(t) = -S(t)^T \nabla R(g(v(t)))$$

and let $Q(t) = S(t)S(t)^T$. Further suppose for all $t \in [0, \tau]$ that

$$\lambda I \preceq Q(t) \preceq \lambda_{max} I$$

Then we have

$$\|v(t) - v(0)\| \leq \frac{\sqrt{\lambda_{max}}}{\lambda} \|g(v(0)) - y\|$$

$$\leq \frac{\sqrt{2\lambda_{max} R(g(v(0)))}}{\lambda}$$

Proof: This follows from a direct computation + Corollary:

$$\|v(t) - v(0)\| = \left\| \int_0^t \dot{v}(t)\, ds \right\|$$

$$\leq \int_0^t \|\dot{v}(t)\|\, ds$$

$$= \int_0^t \|S(t)^T \nabla R(g(v(t)))\|\, ds$$

$$\leq \sqrt{\lambda_{max}} \int_0^t \| g(v(t_s)) - y \| \, ds$$

Now we can invoke the Corollary to get

$$\| v(t) - v(0) \| \leq \sqrt{\lambda_{max}} \int_0^t \| g(v(0)) - y \| e^{-s\lambda} \, ds$$

$$\leq \frac{\sqrt{\lambda_{max}}}{\lambda} \| g(v(0)) - y \|$$

$$= \frac{\sqrt{2 \lambda_{max} R(g(v(0)))}}{\lambda} \qquad \boxtimes$$

Putting it all together, recall that

$$\Phi(t) = \alpha^2 J_t J_t^T$$

So if we have bounds

$$(\Delta) \qquad \alpha^2 \sigma_{min} I \preceq \Phi(t) \preceq \alpha^2 \sigma_{max} I$$

we'd get

$$① \quad R(\alpha f(w(t))) \leq R(\alpha f(w(0)))\, e^{-2t\alpha^2 \sigma_{min}}$$

$$② \quad \| w(t) - w(0) \| \leq \frac{\sqrt{2\sigma_{max} R(\alpha f(w(0)))}}{\alpha \sigma_{min}}$$

But where do we get bounds like (Δ)?

If we make α large enough $\Rightarrow$

we don't move far $\Rightarrow$

the eigenvalues of $\Phi(t)$ never change much from $\Phi(0)$ $\Rightarrow$

we can use a slightly degraded band (i.e. $\sigma_{min} \to \frac{\sigma_{min}}{2}$, $\sigma_{max} \to 2\sigma_{max}$)

See full details in Telgarsky's notes

Epilogue

what random initialization puts you in the NTK regime?

Set each layer

$$x \mapsto A_i x + b_i$$

to have $\overset{n_i \times n_{i-1}}{A_i} = \frac{1}{\sqrt{n_i}} W_i$ ← standard Gaussian entries

Later in the course we will cover the mean field view, which describes the dynamics under a different scaling as a PDE

So should you choose your architecture to be very (infinitely) wide?

No, but it does give rise to better kernel-based algorithms than we had before e.g. ~65% accuracy on CIFAR-10

Actually you can analytically compute the kernel matrix without ever setting up the deep net

Explicit formulas in, e.g., Yang