CPSC 340/540 Tutorial 5

Winter 2024 Term 1

T1A: Tuesday 16:00-17:00;
T1C: Thursday 10:00-11:00;
Office Hour: Wednesday 15:00-16:00

Slides can be found at Piazza and my personal page after T1C.


piazza CPSC 340 2024W1 Q & A Resources

Tutorials
Manually sort using

Tutorials	Date
Tutorial 1 (T1D, T1F, T1G)	click to edit date
Slides for T1A and T1C	click to edit date

Yi (Joshua) Ren

https://joshua-ren.github.io/
renyi.joshua@gmail.com
PhD with Danica

Machine Learning:
Learning dynamics, LLM, Compositional Generalization


Publications

Notes and TA

- Here are links for TA sessions of CPSC 340 (Machine Learning and Data Mining - Fall 2024):

Week 1: basic knowledge review

More helpful on theory

Less helpful on coding

Slides Credit: To various pervious TA's of this course

- Gradient Descent
- Robust Regression
- Some mid-term questions

Regression: (different regularizers)

Unit ball, i.e., $\|x\|_p = 1$

- Recap of different norms

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

  - L0-norm: non-zero elements in a vector
  - L1-norm: usually use to introduce sparsity (vertex at axis)
  - L2-norm: Gaussian, Euclidian distance, most common
  - L∞-norm: select the maximum value

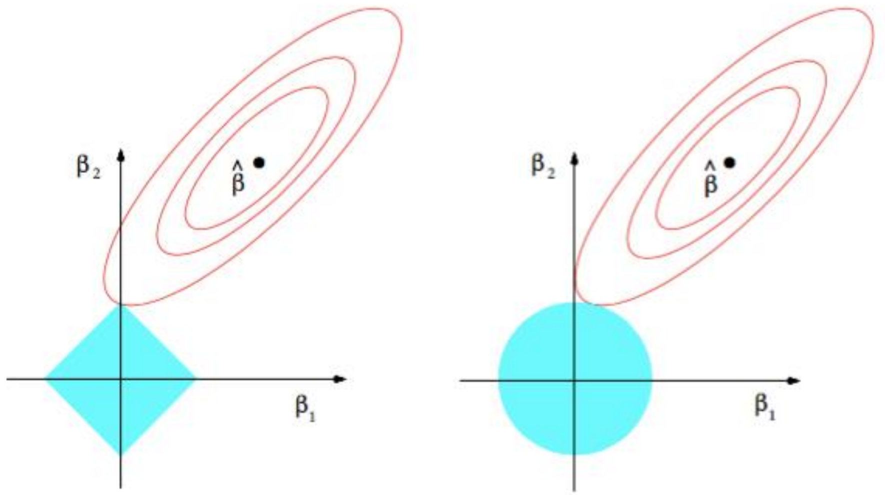
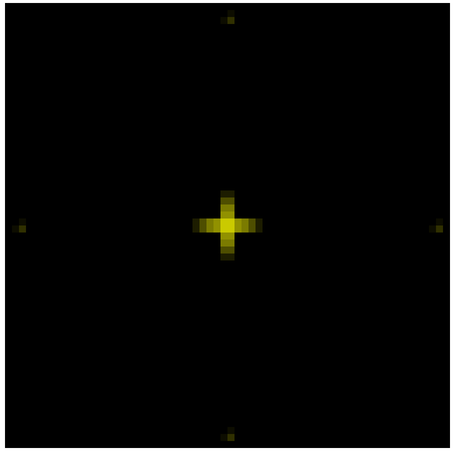
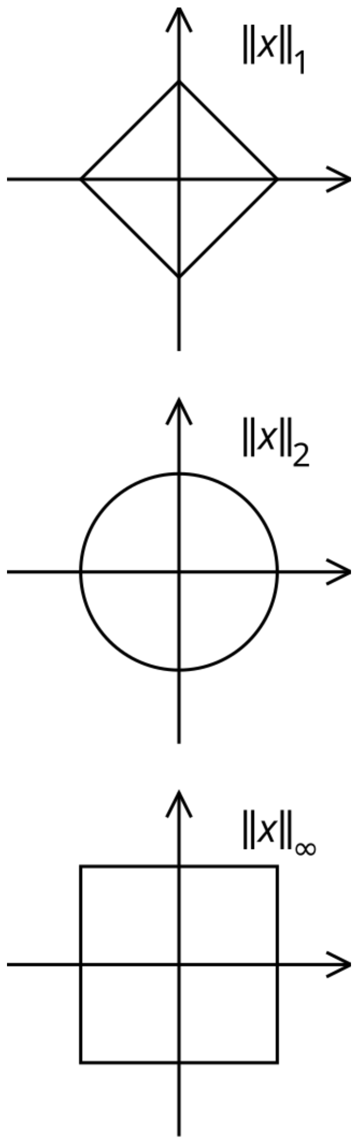


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Unit ball, p=0 to 2





Assignment 3 – 1.1

$$\left(\sum_{i=1}^{n} |w^T x_i - y_i|\right)^2. \qquad \sum_{i=1}^{n} v_i (w^T x_i - y_i)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2.$$

General tricks:

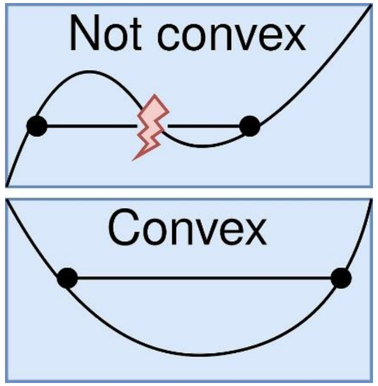
1. Recall the definition of norms, inner product, etc.
2. Consider diagnal matrix
3. Use $\boldsymbol{r} = [r_1, \dots, r_n]^T$; $r_i = w^T x_i - y_i$

$$\boldsymbol{r} = \boldsymbol{X}\boldsymbol{w} - \boldsymbol{Y}$$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}. \qquad \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} x_i x_j = \mathbf{x}^{\mathrm{T}} A \mathbf{x}, \qquad \sum_{i=1}^{n} \lambda_i x_i x_i = \mathbf{x}^{\mathrm{T}} \boldsymbol{\Lambda} \mathbf{x}$$

Convexity: (basic facts)

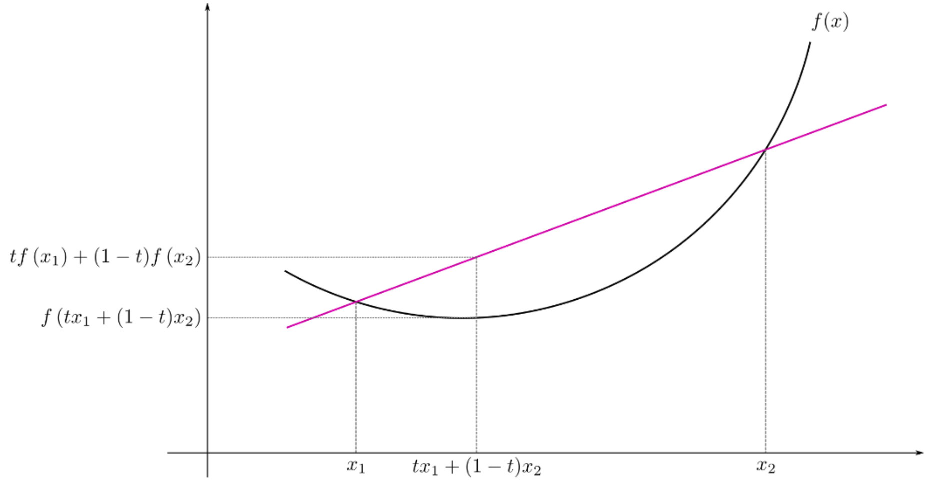
- Intuition:



- Definition:

For all $0 \leq t \leq 1$ and all $x_1, x_2 \in X$:

$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$



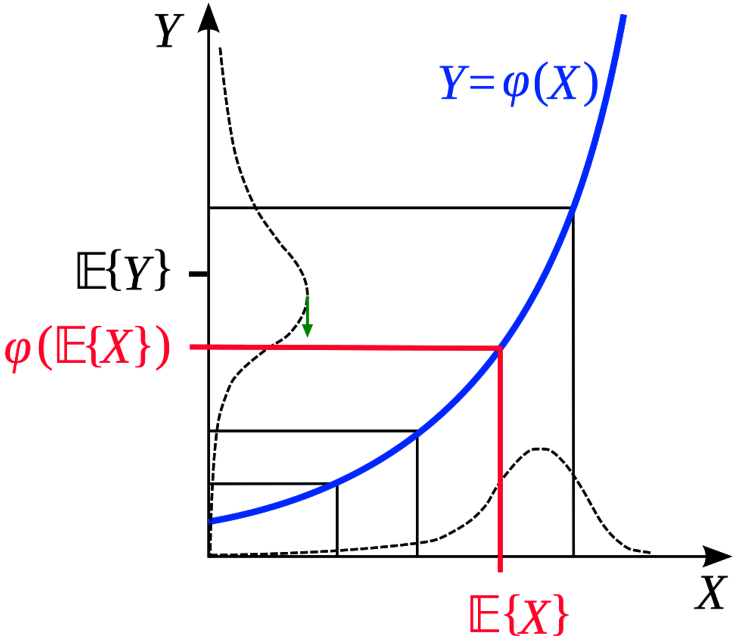
- Checking measurement:

$$f'' > 0$$

Hessian matrix is PSD

- Good to know – Jensen's Inequality

$$\varphi(\mathrm{E}[X]) \leq \mathrm{E}[\varphi(X)] = \mathrm{E}[Y],$$



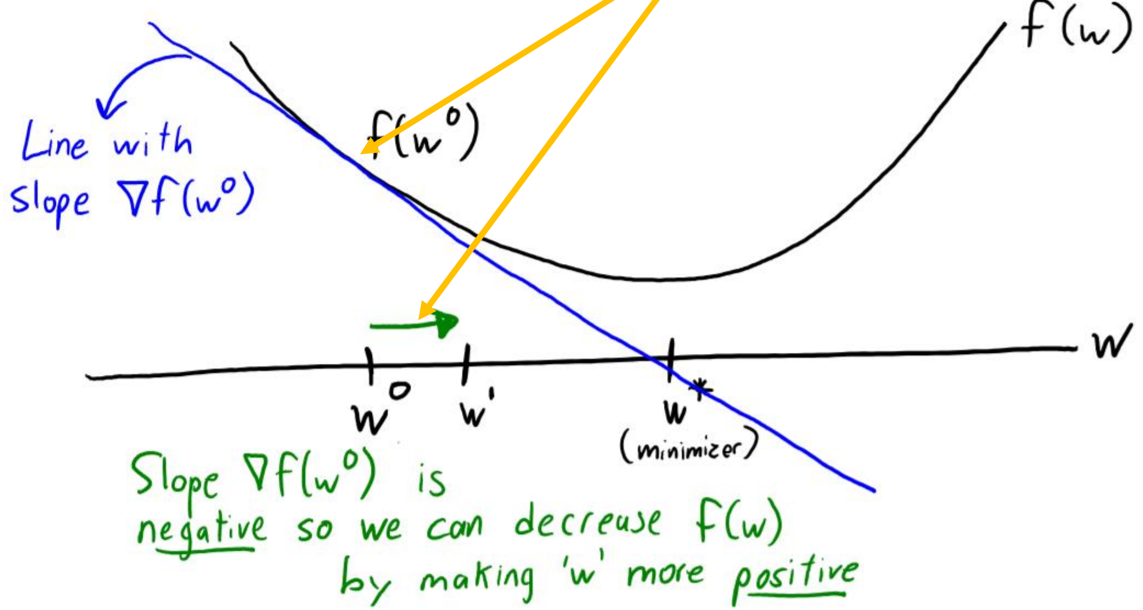
- Compose functions:

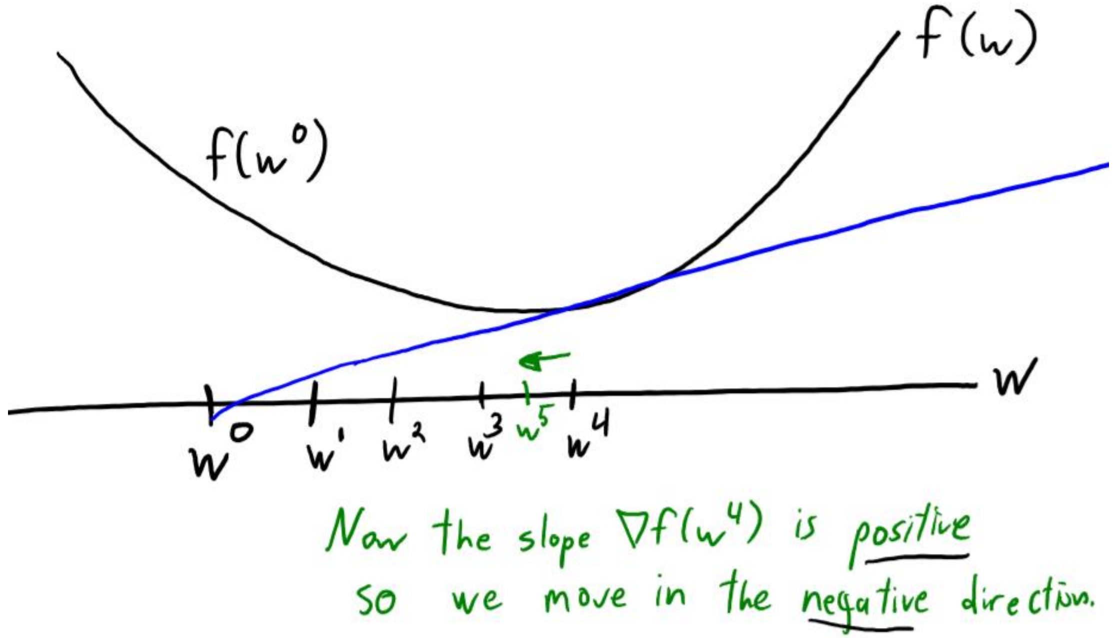
√ Norm, linear, sum, max, ...

? f(g()), multiplication, ...

Gradient Descent: (what is GD)

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \alpha^{t}\nabla_{\boldsymbol{w}} f(\boldsymbol{x})$$



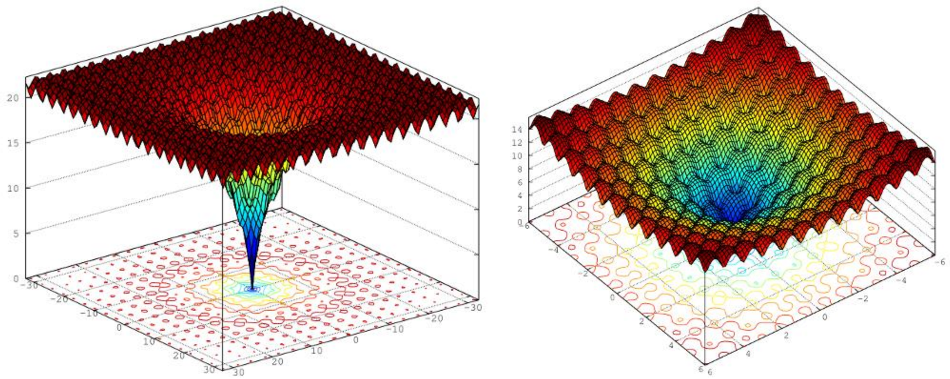


Gradient Descent: (why we need GD?)

$$w^* = (X^T X)^{-1} X^T y \qquad\qquad w^{t+1} = w^t - \alpha^t X^T (X w^t - y)$$

- More efficient when data is high-dim (inverse of a d*d matrix)
  - Normal equations cost $O(nd^2 + d^3)$.
  - Gradient descent costs O(ndt) to run for 't' iterations.
    - Each of the 't' iterations costs O(nd).

- Require too many assumptions
  - $X^T X$ might be non-invertable (e.g., $n < d$ → rank of it would be at most n)
  - The problem must be convex to ensure w^* is a good solution.
    For GD, in deep learning, loss landscape like this ... Any reasonable good local optimum is good enough



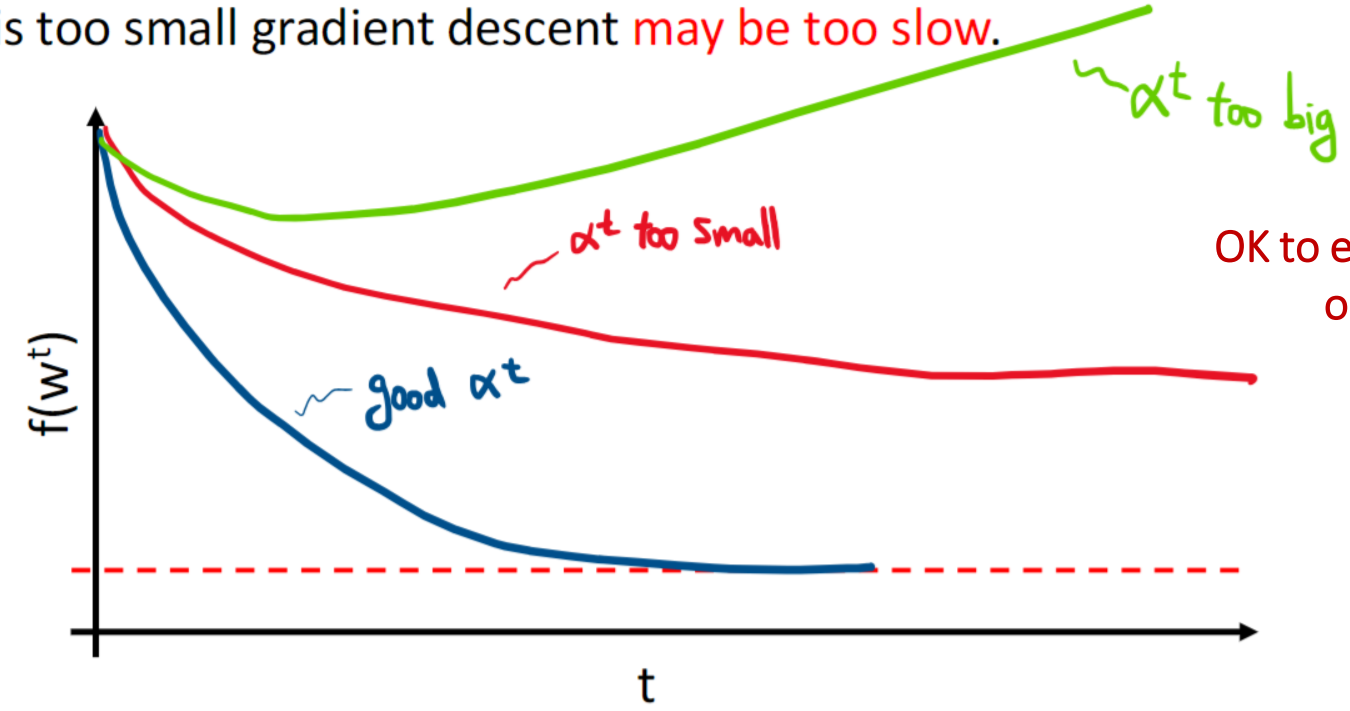
Gradient Descent: (building blocks)

$$w^{t+1} = w^t - \alpha^t \nabla_w f(x)$$

- Influence of learning rate – General facts:

If α^t is too large gradient descent may not converge.

If α^t is too small gradient descent may be too slow.

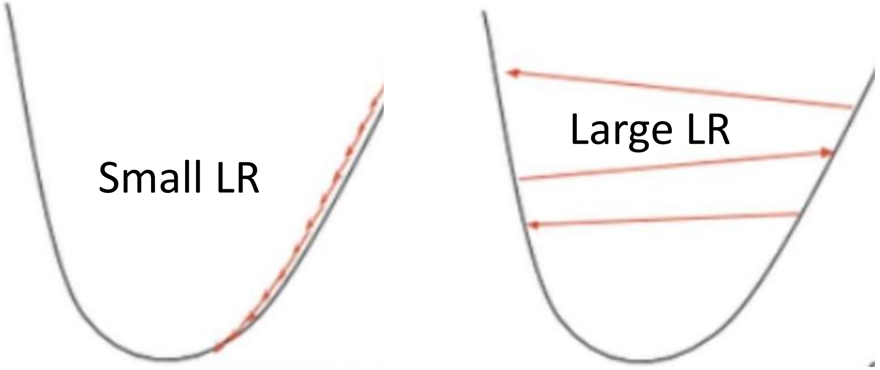


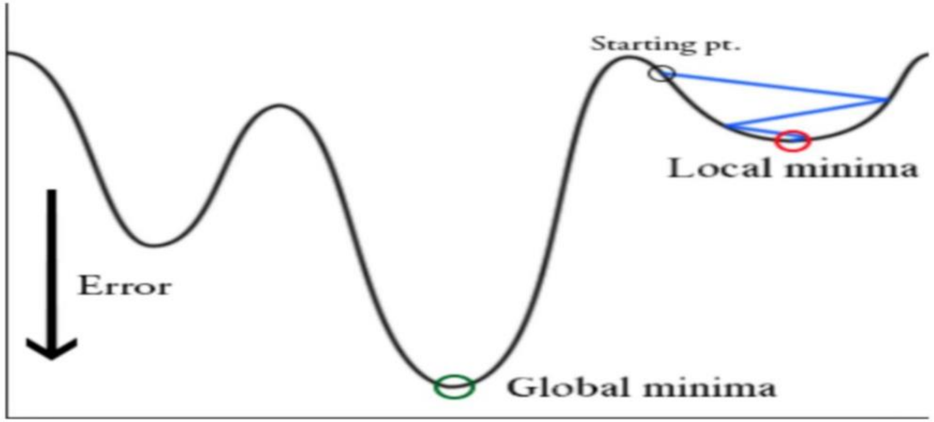
OK to explain linear regression or simple problems

Gradient Descent: (building blocks)

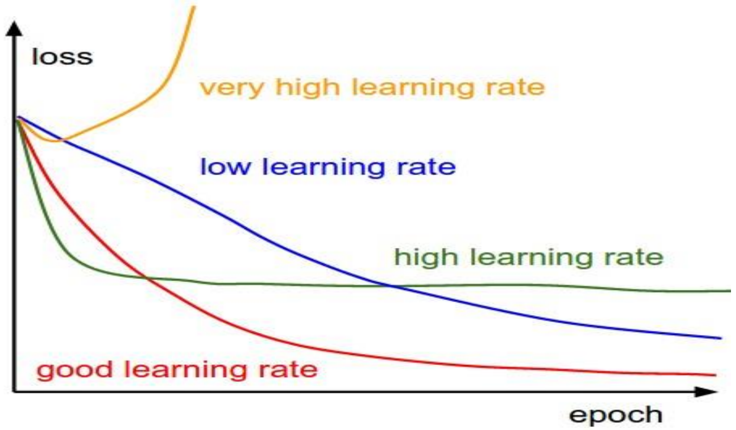
$$w^{t+1} = w^t - \alpha^t \nabla_w f(x)$$

- More subtle difference for deep learning:
  - Think about, which case is better?

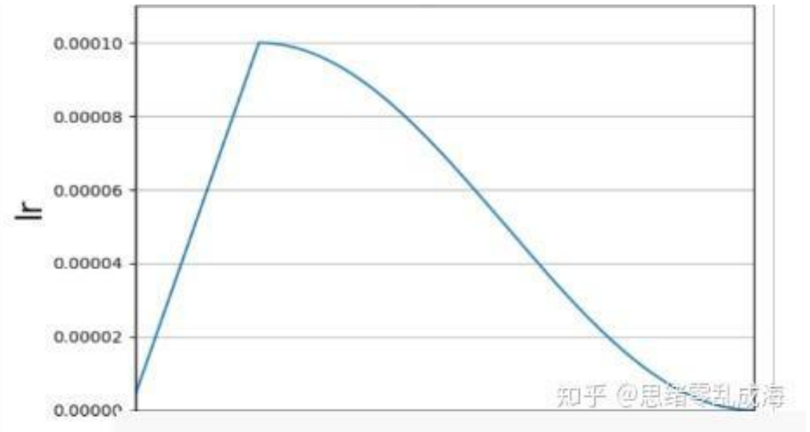




  - More subtle influence of LR in deep learning



  - Usually in practice: different LR schedulers



Gradient Descent: (building blocks)

$$w^{t+1} = w^t - \alpha^t \nabla_w f(x)$$

- Various a lot for different types of targets and networks. Chain rule is widely used. (One example in my paper)

$$\begin{aligned}
[\mathcal{G}^t_{\mathrm{DPO}}]_l &= \frac{\partial \mathcal{L}_{\mathrm{DPO}}}{\partial a}\frac{\partial a}{\partial b}\nabla_\pi b|_{\pi_{\theta^t}} \nabla_{\mathbf{z}_l}\pi^t|_{\mathbf{z}^t_l} \\
&= -\frac{1}{a}a(1-a)\langle \nabla_\pi b|_{\pi_{\theta^t}}, [\mathcal{A}^t(\chi_u)]_l\rangle \\
&= -(1-a)\langle \beta\left(\mathcal{L}_{\mathrm{SFT}}([\mathbf{x}_u, \mathbf{y}^-_u]_l) - \mathcal{L}_{\mathrm{SFT}}([\mathbf{x}_u, \mathbf{y}^+_u]_l)\right), [\mathcal{A}^t(\chi_u)]_l\rangle \\
&= -\beta(1-a)\left(\langle \mathcal{L}_{\mathrm{SFT}}([\mathbf{x}_u, \mathbf{y}^-_u]_l), [\mathcal{A}^t(\chi_u)]_l\rangle - \langle \mathcal{L}_{\mathrm{SFT}}([\mathbf{x}_u, \mathbf{y}^+_u]_l), [\mathcal{A}^t(\chi_u)]_l\rangle\right) \\
&= -\beta(1-a)\left((\pi_{\theta^t}(\mathbf{y}^-_u) - \mathbf{e}_{\mathbf{y}^-_u}) - (\pi_{\theta^t}(\mathbf{y}^+_u) - \mathbf{e}_{\mathbf{y}^+_u})\right)_l \\
&\approx \beta(1-a)\left(\mathbf{e}_{\mathbf{y}^-_u} - \mathbf{e}_{\mathbf{y}^+_u}\right)_l.
\end{aligned} \tag{16}$$

- But implementation is suprisingly simple, thanks to Autogradient mechanism (e.g., in Pytorch)

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

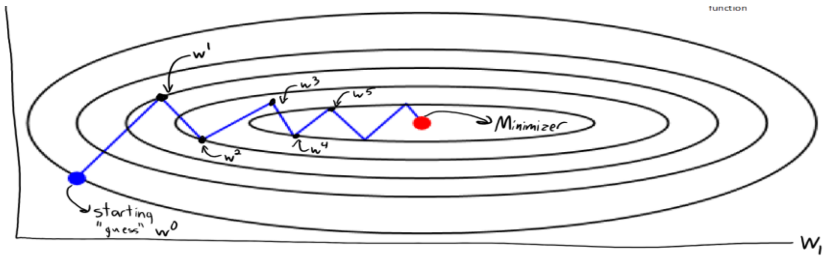
```
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

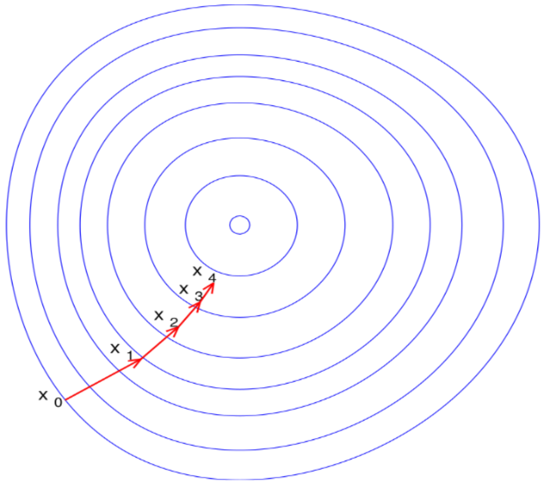
```
inputs, labels = data

# zero the parameter gradients
optimizer.zero_grad()

# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

Gradient Descent: (building blocks)

$$w^{t+1} = \boldsymbol{w^t} - \alpha^t \nabla_w f(x)$$

- Influence of the relative size of different dimension in $\boldsymbol{w}$ (LR is too small for one dim, but too large for another dim)



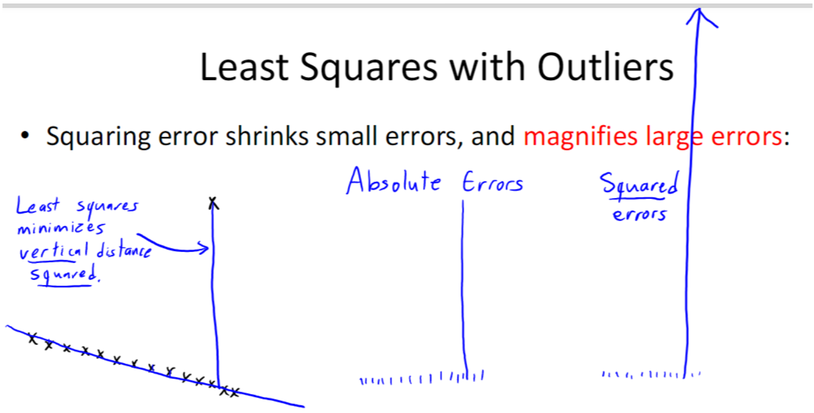


- Solution to it: **layer-normalization** (very common in deep learning) or **adaptive LR method** (e.g., Adam)

- Gradient Descent
- Robust Regression
- Some mid-term questions

Robust regression: standard solutions

- Facts:

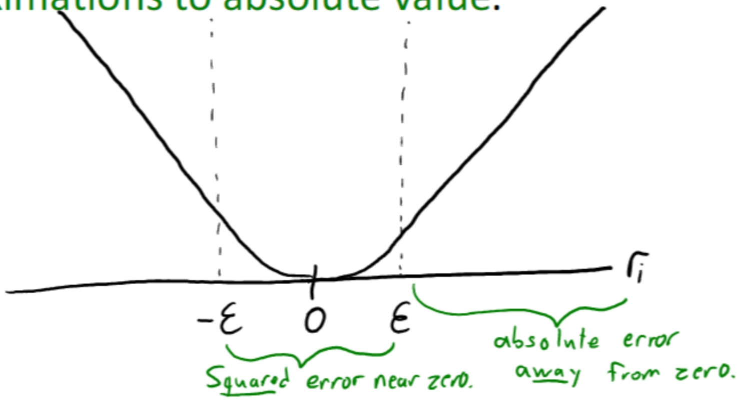


- How to solve it?

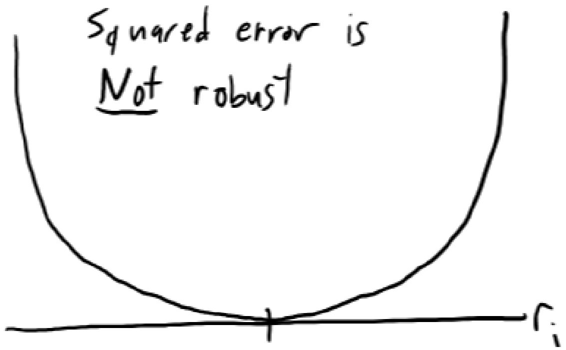
- There are differentiable approximations to absolute value.
  - Common example is Huber loss:

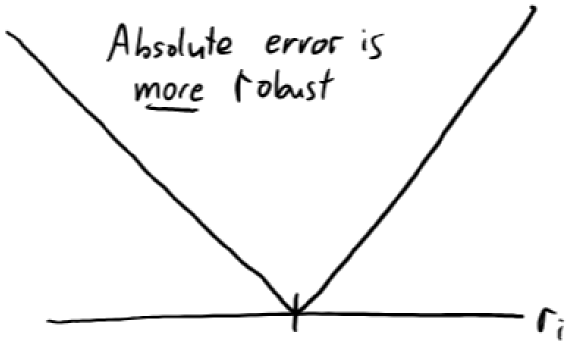
$$f(w) = \sum_{i=1}^{n} h(w^T x_i - y_i)$$

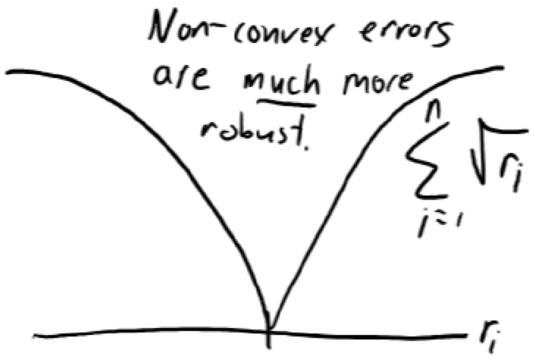
$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{for } |r_i| \leq \varepsilon \\ \varepsilon(|r_i| - \frac{1}{2}\varepsilon) & \text{otherwise} \end{cases}$$

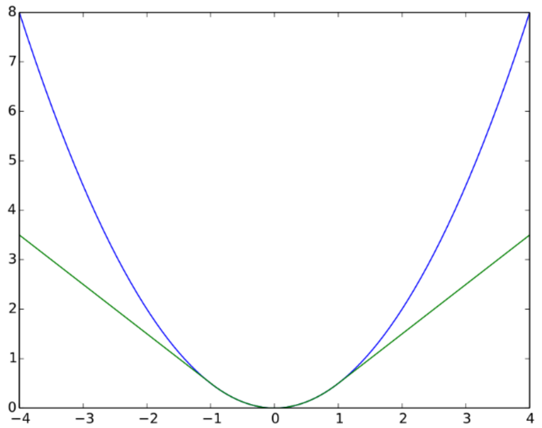


- Why?



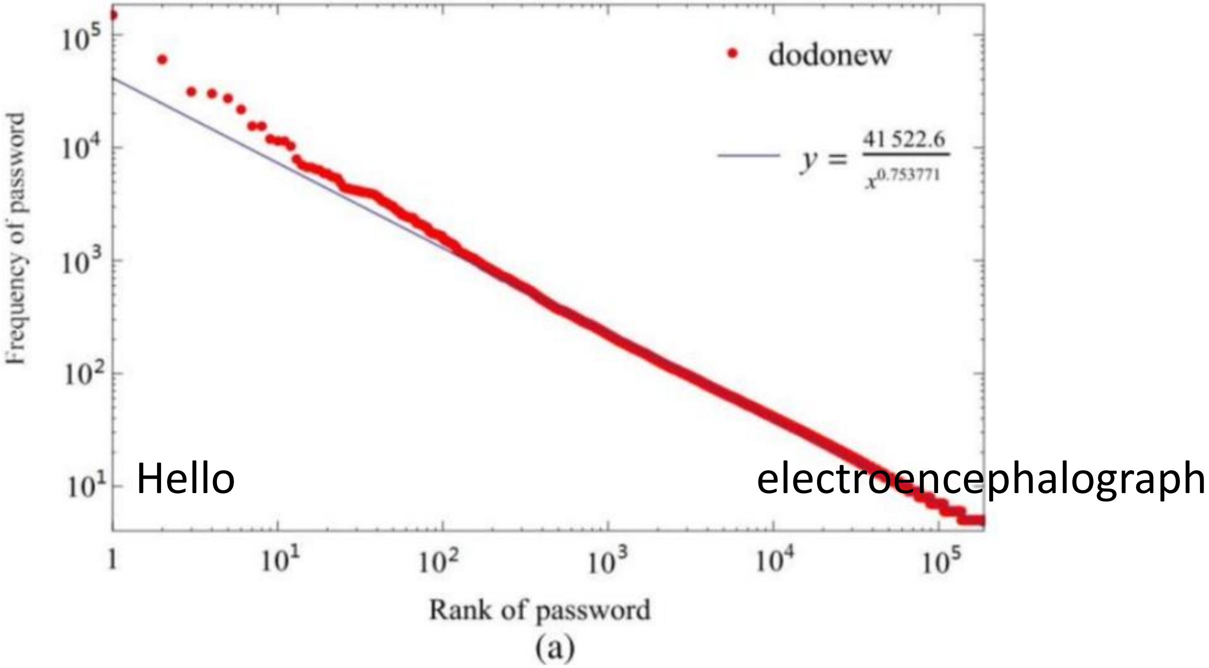






Robust regression: all about outliers, but is it indeed bad?

- In practice, long tail and Zipf's law:



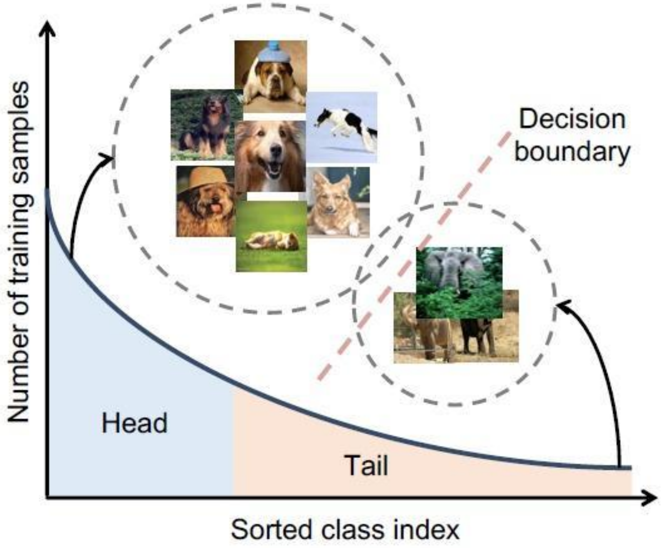


Fig. 1. The label distribution of a long-tailed dataset (*e.g.*, the iNaturalist species dataset [23] with more than 8,000 classes). The head-class feature space learned on these sampled is often larger than tail classes, while the decision boundary is usually biased towards dominant classes.

They are rare, but not outliers, and should be very important!

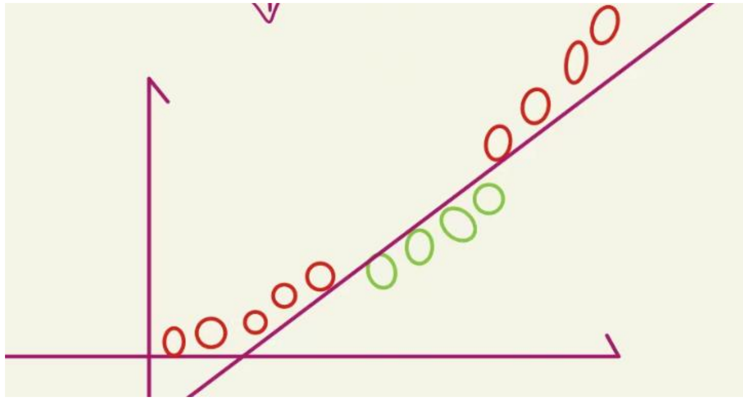
- Gradient Descent
- Robust Regression
- Some mid-term questions

2017-Q1

(d) What is the effect of the number of features d that our model uses on the two parts of the fundamental trade-off?

Larger d → higher-dim input → Curse of dim, need more data → current data is not enough → imagine only 1 data → **seriously overfitting** → ...

- But slighly increase d might be helpful sometimes (also 2D-3D).



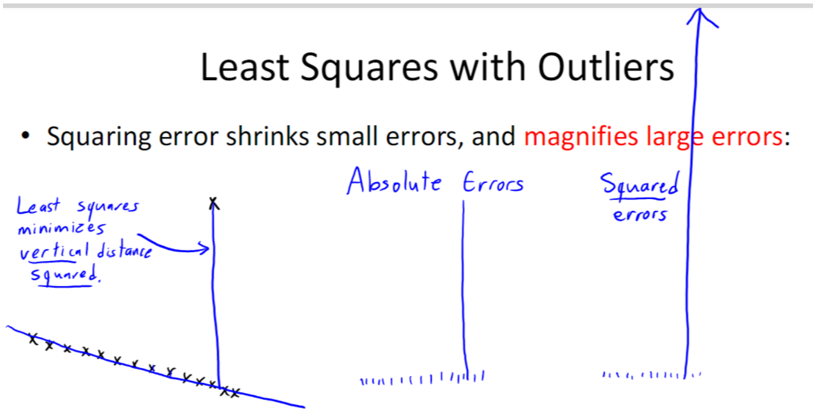
(g) Besides finding a clustering of the data, what is another use of the k-means algorithm?

Assign **pseudo labels** for downstream tasks; use the means as representation for the group (coreset selection); different clusters for different experts; etc...

2017-Q1

(j) In regression, what is a situation where we would want to minimize the L1-norm error ($\|Xw - y\|_1$) instead of the least squares error ($\|Xw - y\|^2$)?

- **When you have outliers.**



(k) Why would we want to approximate the L∞-norm error with the log-sum-exp function?

Argmax v.s. Softmax
Stepwise v.s. Sigmoid

Many good properties, easy-to-handle derivative, smooth, etc.

Thanks for your time!

Questions?