- HW 2 update
- project presentation

Techniques for Improving Generalization

Weight Decay

$\min f(\theta) + \frac{\lambda}{2}\|\theta\|_2^2$

L2 regularization: $\frac{\lambda}{2}\|\theta\|_2^2$

decay

Implementation: $\theta \leftarrow (1 - \eta\lambda)\theta - \eta \nabla f(\theta)$

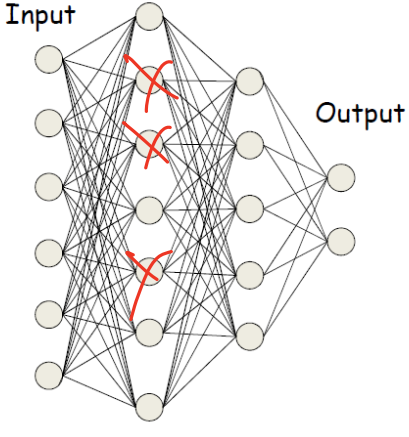
η: step size

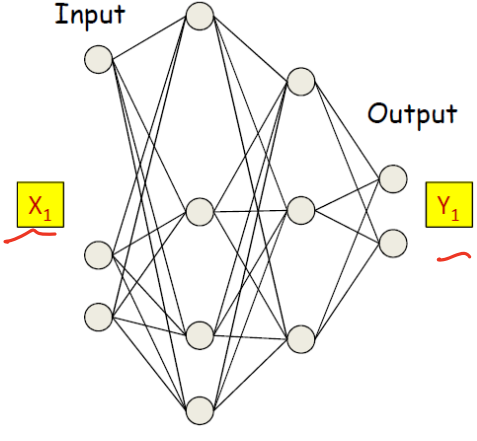
Dropout

Intuition: randomly cut off some connections and neurons.

Training: for each input, at each iteration, randomly "turn off" each neuron with a probability $1 - \alpha$

- Change a neuron to 0 by sampling a Bernoulli variable.
- Gradient only propogatd from non-zero neurons.



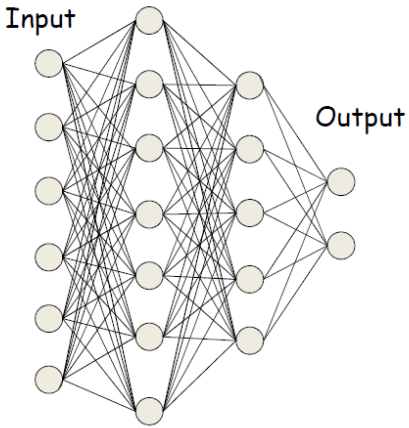


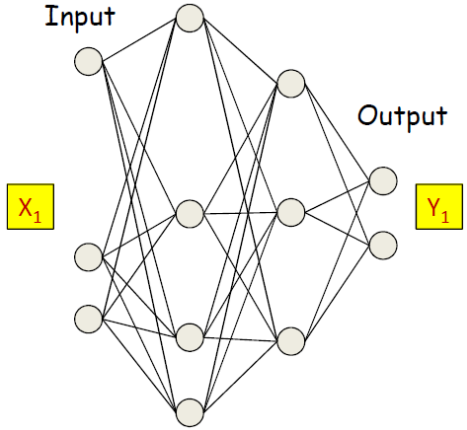
Dropout

Dropout changes the scale of the output neuron:

- $y = \text{Dropout}(\sigma(WX))$
- $\mathbb{E}[y] = \alpha\mathbb{E}[\sigma(Wx)]$

Test time: $y = \alpha\sigma(Wx)$ to match the scale

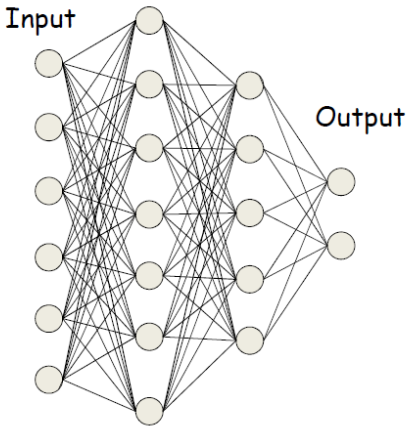


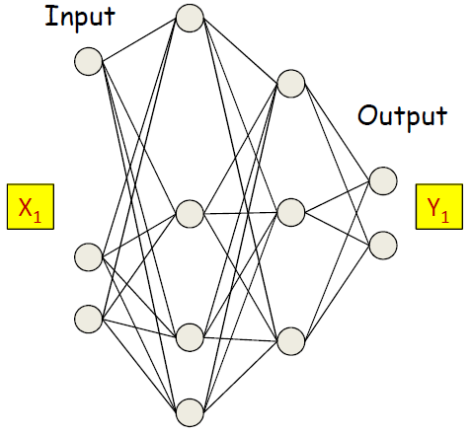


Understanding Dropout

- Dropout forces the neural network to learn redundant patterns.
- Dropout can be viewed as an implicit L2 regularizer (Wager, Wang, Liang '13).

Linear, in expectation

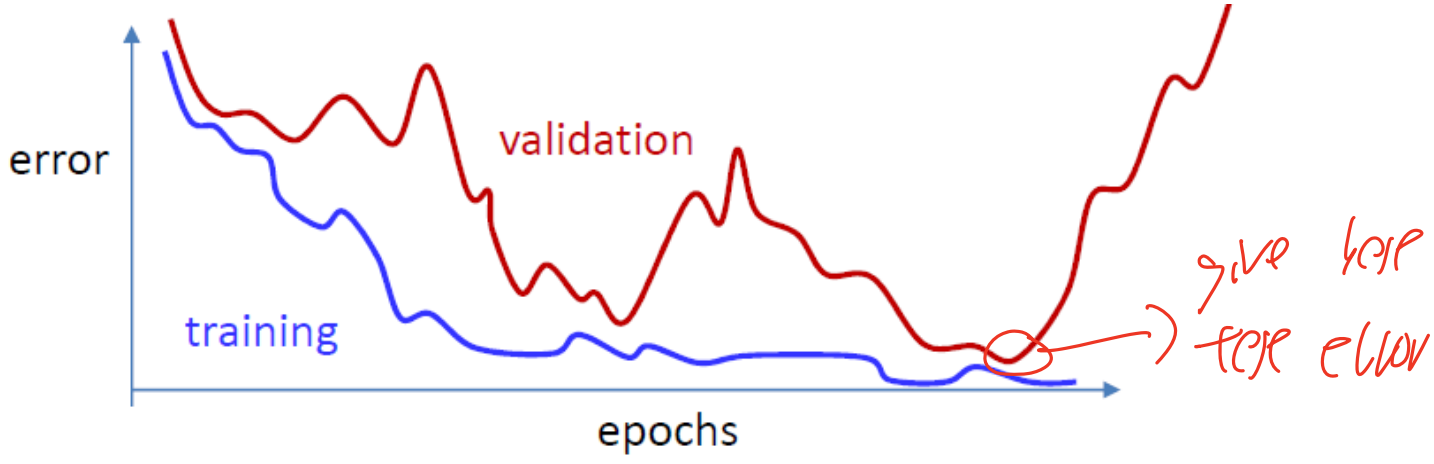




Early Stopping

view training time as a hyper-parameter

- Continue training may lead to overfitting.
- Track performance on a held-out validation set.
- Theory: for linear models, equivalent to L2 regularization.



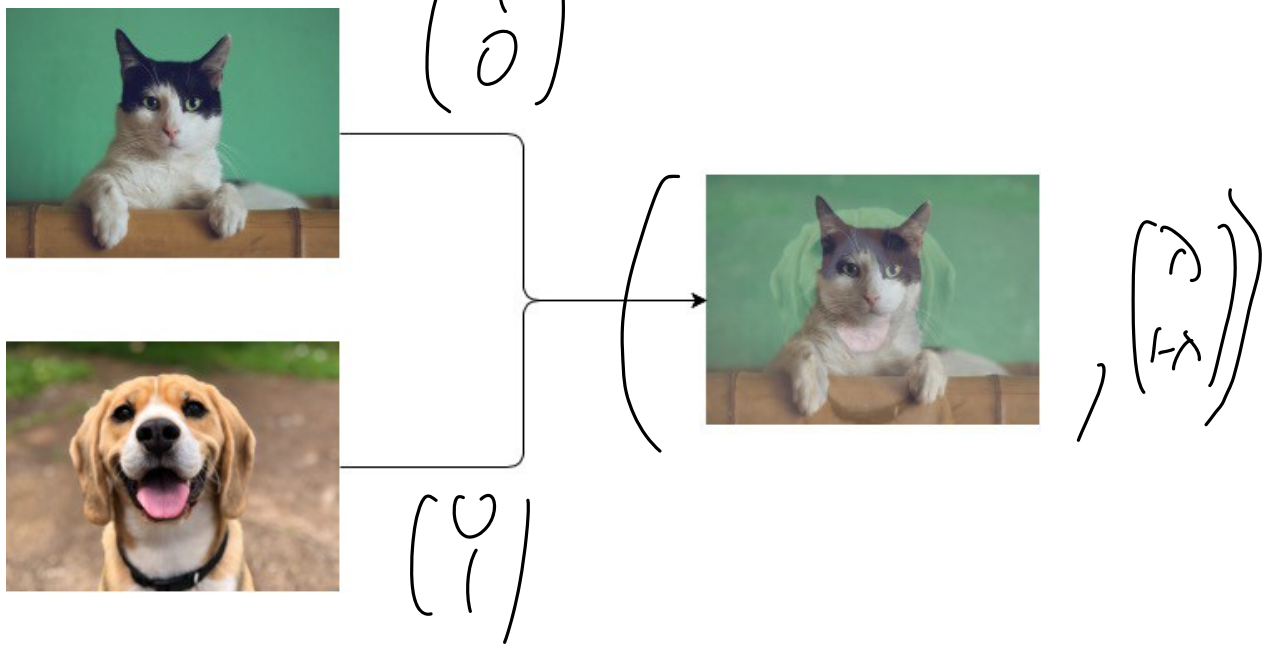
Data Augmentation

Depend on data types.

Computer vision: rotation, stretching, flipping, etc

Mixup data augmentation

- $\hat{x} = \lambda x_i + (1 - \lambda) x_j$
- $\hat{y} = \lambda y_i + (1 - \lambda) y_j$
- $\lambda \sim \textbf{Beta}(0.2)$



Data Augmentation

sentiment analysis
sentence → (+, −)

Depend on data types.

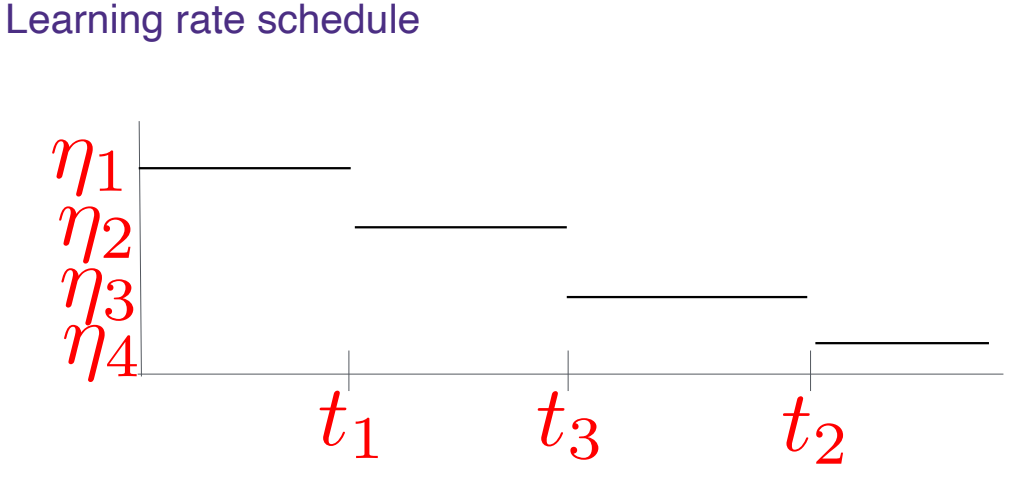
Natural language processing:

- Synonym replacement
  - *This article will focus on summarizing data augmentation in NLP.*
  - *This write-up will focus on summarizing data augmentation in NLP.*

- Back translation: translate the text data to some language and then translate back
  - *I have no time.* -> 我没有时间. -> *I do not have time.*

Learning rate scheduling

Start with large learning rate. After some epochs, use small learning rate.



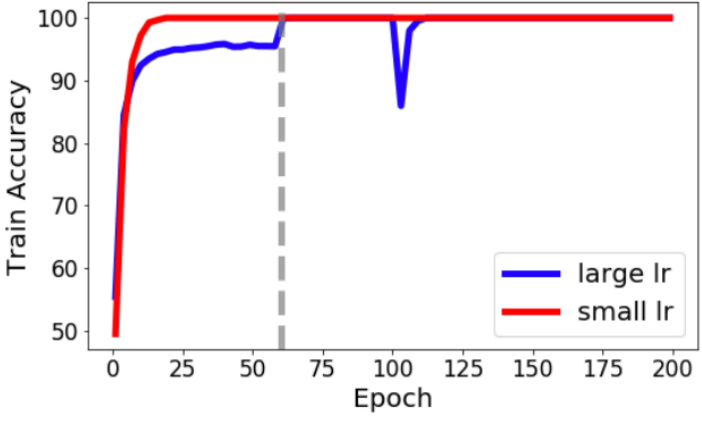
Learning rate scheduling

Start with large learning rate. After some epochs, use small learning rate.

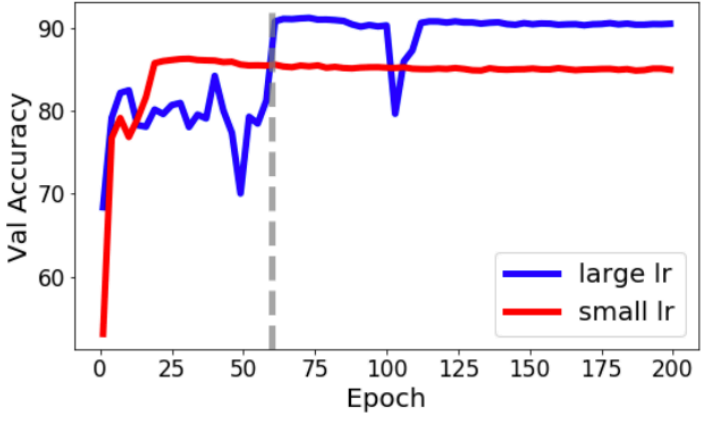
Theory:

- Linear model / Kernel: large learning rate first learns eigenvectors with large eigenvalues (Nakkiran, '20).
- Representation learning (Li et al., '19)

learn coarse feature → better generalization



Train

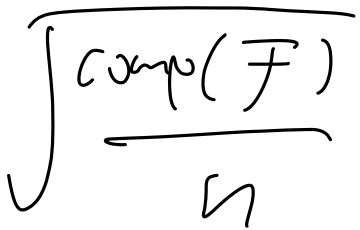


Validation

Normalizations

- Batch normalization (Ioffe & Szegedy, ’15)
- Layer normalization (Ba, Kiros, Hinton, ’16)
- Weight normalization (Salimans, Kingma, ’16)
- Instant normalization (Ulyanov, Vedaldi, Lempitsky, ’16)
- Group normalization (Wu & He, ’18)
- …

Generalization Theory for Deep Learning



Basic version: finite hypothesis class

Finite hypothesis class: with probability $1-\delta$ over the choice of a training set of size n, for a bounded loss ℓ, we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D}\left[\ell(f(x), y)\right] \right| = O\left(\sqrt{\frac{\log|\mathcal{F}| + \log 1/\delta}{n}}\right)$$

for all

- For a fixed $f \in \mathcal{F}$, by Hoeffding ineq, w.p. $1-\delta$

$$\frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D}\left(\ell(f(x), y)\right) = O\left(\sqrt{\frac{1+\log(1/\delta)}{n}}\right)$$

- union bound: event_1, ... event_m

$$P\left(\bigcup_{j=1}^{m} \text{event-}j\right) \le \sum_{j=1}^{m} P(\text{event-}j)$$

Let event_j be the event the j^{th} function in $\mathcal{F}$ s.t.

$$\text{gen\_error\_}j > \sqrt{\frac{\log|\mathcal{F}|}{n}}, \quad P(\text{event-}j) \le \delta$$

$$P(\exists f \in \mathcal{F},\ \text{gen-error}_f > \sqrt{\frac{\log(1/\delta)}{n}})$$

$$= P\left(\bigcup_j \text{event-}j\right)$$

$$\leq |\mathcal{F}| \cdot \delta$$

let $\delta' = |\mathcal{F}| \cdot \delta$

$$\Rightarrow \sqrt{\frac{\log(|\mathcal{F}|) + \log(1/\delta)}{n}}$$

$Z_1, \ldots Z_M$

$$\left| \hat{Z}_n - E(Z) \right| \approx O\left(\frac{1}{\sqrt{n}}\right)$$

VC-Dimension

Motivation: Do we need to consider **every** classifier in $\mathcal{F}$?
Intuitively, **pattern of classifications** on the training set should suffice. (Two predictors that predict identically on the training set should generalize similarly).

Let $\mathcal{F} = \{f : \mathbb{R}^d \to \{+1, -1\}\}$ be a class of binary classifiers.

The **growth function** $\Pi_{\mathcal{F}} : \mathbb{N} \to \mathbb{F}$ is defined as:

$$\Pi_{\mathcal{F}}(m) = \max_{(x_1, x_2, \ldots, x_m)} \left| \left\{ (f(x_1), f(x_2), \ldots, f(x_m)) \mid f \in \mathcal{F} \right\} \right|.$$

The VC dimension of $\mathcal{F}$ is defined as:

$$\text{VCdim}(\mathcal{F}) = \max\{m : \Pi_{\mathcal{F}}(m) = 2^m\}.$$

VC-dimension Generalization bound

Theorem (Vapnik-Chervonenkis): with probability $1-\delta$ over the choice of a training set, for a bounded loss ℓ, we have

$$\sup_{f\in\mathcal{F}}\left|\frac{1}{n}\sum_{i=1}^{n}\ell(f(x_i),y_i)-\mathbb{E}_{(x,y)\sim D}\left[\ell(f(x),y)\right]\right| = O\left(\sqrt{\frac{\mathrm{VCdim}(\mathcal{F})\log n+\log 1/\delta}{n}}\right)$$

Examples:

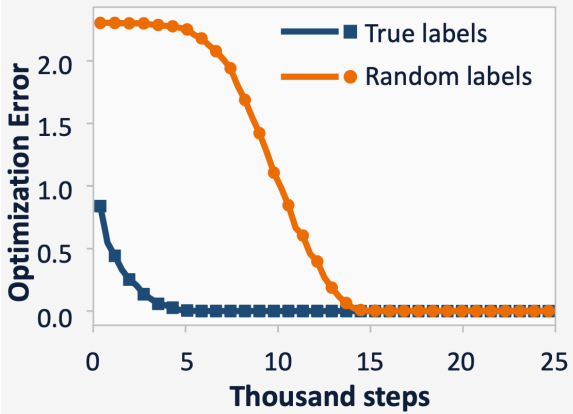
- Linear functions: VC-dim = O(dimension)
- Neural network: VC-dimension of fully-connected net with width W and H layers is $\widetilde{\Theta}(WH)$ (Bartlett et al., '17).

Problems with VC-dimension bound

1. In over-parameterized regime, bound >> 1.
2. Cannot explain the random noise phenomenon:
   - Neural networks that fit random labels and that fit true labels have the same VC-dimension.

Practice: gradient descent

$$\theta(t+1) \leftarrow \theta(t) - \eta \frac{\partial L(\theta(t))}{\partial \theta(t)}$$



Optimization error -> **0** for both ***true labels*** and ***random labels*** !

Zhang Bengio Hardt Recht Vinyals 2017
Understanding DL Requires Rethinking Generalization

PAC Bayesian Generalization Bounds

Setup: Let P be a prior over function in class $\mathcal{F}$, let Q be the posterior (after algorithm's training).

Theorem: with probability $1-\delta$ over the choice of a training set, for a bounded loss ℓ, we have

$$\sup_{f\in\mathcal{F}} \left| \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D}\left[\ell(f(x), y)\right] \right| = O\left(\sqrt{\frac{KL(Q \,||\, P) + \log 1/\delta}{n}}\right)$$

data-dependent

cannot choose Q after training

Rademacher Complexity

Intuition: how well can a classifier class **fit random noise?**

(Empirical) **Rademacher complexity:** For a training set $S = \{x_1, x_2, \ldots, x_n\}$, and a class $\mathcal{F}$, denote:

$$\hat{R}_n(S) = \mathbb{E}_\sigma \sup_{f \in \mathcal{F}} \sum_{i=1}^{n} \sigma_i f(x_i) .$$

where $\sigma_i \sim \text{Unif}\{+1, -1\}$ (Rademacher R.V.).

(Population) **Rademacher complexity:**

$$R_n = \mathbb{E}_S \left[\hat{R}_n(s) \right].$$

Rademacher Complexity Generalization Bound

Theorem: with probability $1 - \delta$ over the choice of a training set, for a bounded loss ℓ, we have

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D} \left[\ell(f(x), y) \right] \right| = O\left(\frac{\hat{R}_n}{n} + \sqrt{\frac{\log 1/\delta}{n}} \right)$$

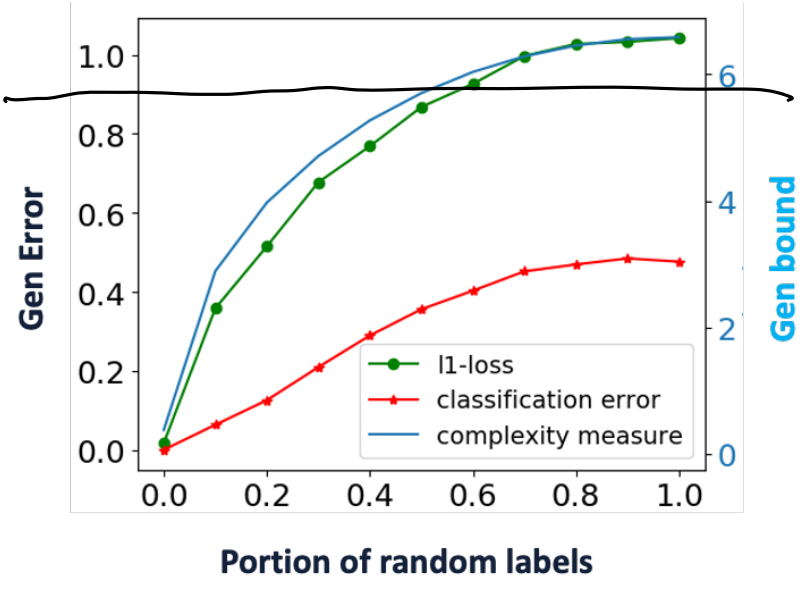
$\hat{\mathcal{R}}_n = O\left(\sqrt{n}\cdot\right)$

and

$$\sup_{f \in \mathcal{F}} \left| \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim D} \left[\ell(f(x), y) \right] \right| = O\left(\frac{R_n}{n} + \sqrt{\frac{\log 1/\delta}{n}} \right)$$

Kernel generalization bound

Use Rademacher complexity theory, we can obtain a generalization bound $O(\sqrt{y^\top (H^*)^{-1} y/n})$ where $y \in \mathbb{R}^n$ are n labels, and $H^* \in \mathbb{R}^{n \times n}$ is the kernel (e.g., NTK) matrix.



Norm-based Rademacher complexity bound

ReLU: $\rho = 1$

Theorem: If the activation function is σ is ρ-Lipschitz. Let

$$\mathcal{F} = \{x \mapsto W_{H+1}\sigma(W_h\sigma(\cdots\sigma(W_1x)\cdots), \|W_h^T\|_{1,\infty} \leq B \forall h \in [H]\}$$

then $R_n(\mathcal{S}) \leq \|X^\top\|_{2,\infty}(2\rho B)^{H+1}\sqrt{2\ln d}$ where $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$ is the input data matrix.

$x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{m\times d}$, $W_2, \ldots W_H \in \mathbb{R}^{m\times m}$, $W_{H+1} \in \mathbb{R}^m$

$$\|W_1^T\|_{1,\infty} = \max_{i=1,\ldots,m} \|W_1^T(:,i)\|_1 = \max_{i=1,\ldots,m} \sum_{k=1}^{d} |W_1^T(k,i)|$$

W_1^T d m

$$\|W_h^T\|_{1,\infty} = \max_{i=1,\ldots,m} \|W_h^T(:,i)\|_1$$

$$\|X^T\|_{2,\infty} = \max_{i=1,\ldots,d} \|X^T(:,i)\|_2$$