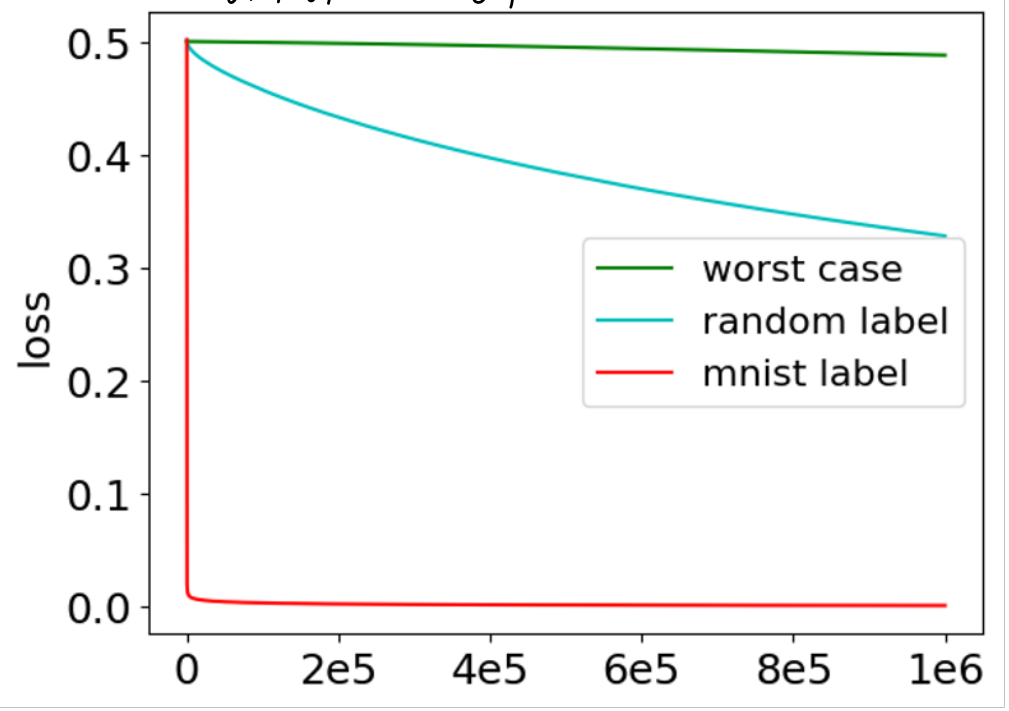
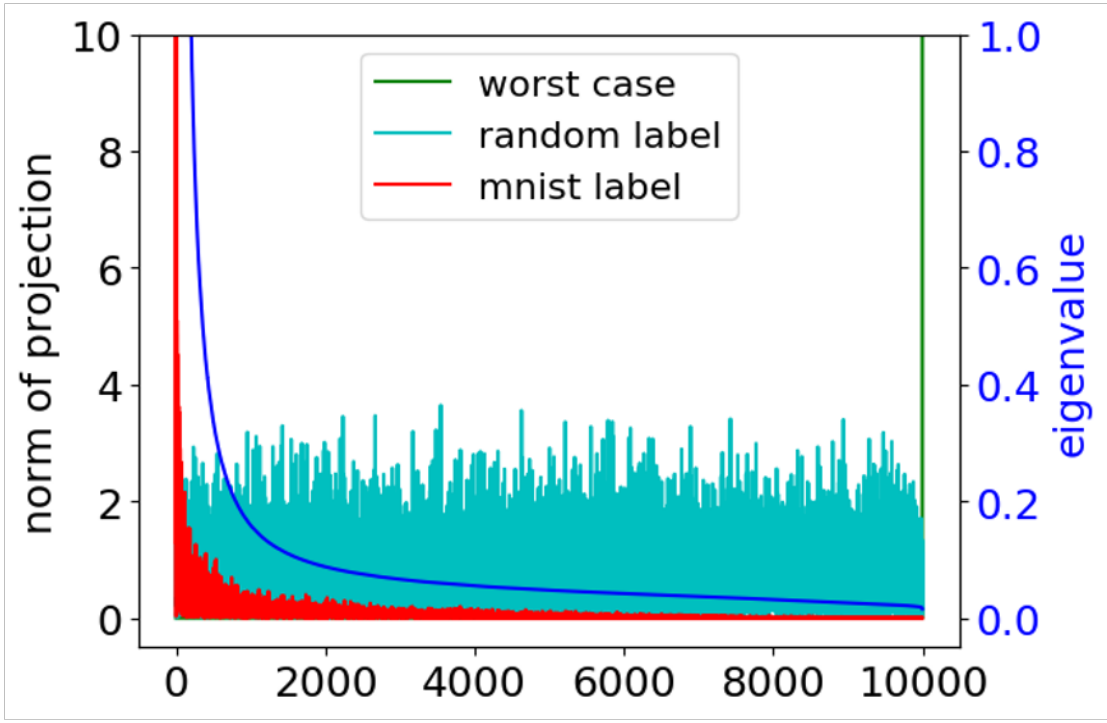
Neural Tangent Kernel

What determines the convergence rate?

$\frac{du(t)}{dt} = -H^*(u(t)-y)$, assume $u(t)=0$

$\Sigma = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}$ $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n$





Convergence Rate

$H^* = U \Sigma U^T$, U: eigenvectors, Σ: eigenvalues

$\frac{du^*(t)^T u_i}{dt} = -\sigma_i (u(t)-y)^T u_i$

Projections

worst case: $y = c \cdot u_n$

best: $y = c \cdot u_1$

Neural Tangent Kernel

Recipe for designing new kernels

$$f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right) \Rightarrow k\left(x, x'\right) = \mathbb{E}_{\theta_{\mathrm{NN}} \sim \mathcal{W}}\left[\left\langle \frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right)}{\partial \theta_{\mathrm{NN}}}, \frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x'\right)}{\partial \theta_{\mathrm{NN}}} \right\rangle\right]$$

Transform a neural network of any architecture to a kernel!

Fully-connected NN → Fully-connected NTK

Convolutional NN → Convolutional NTK

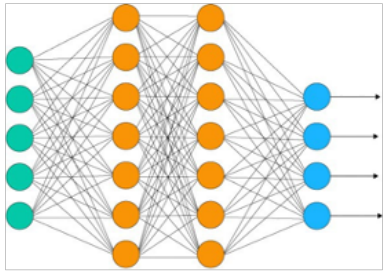
Graph NN → Graph NTK

.....

Fully-Connect NTK

$$\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}$$

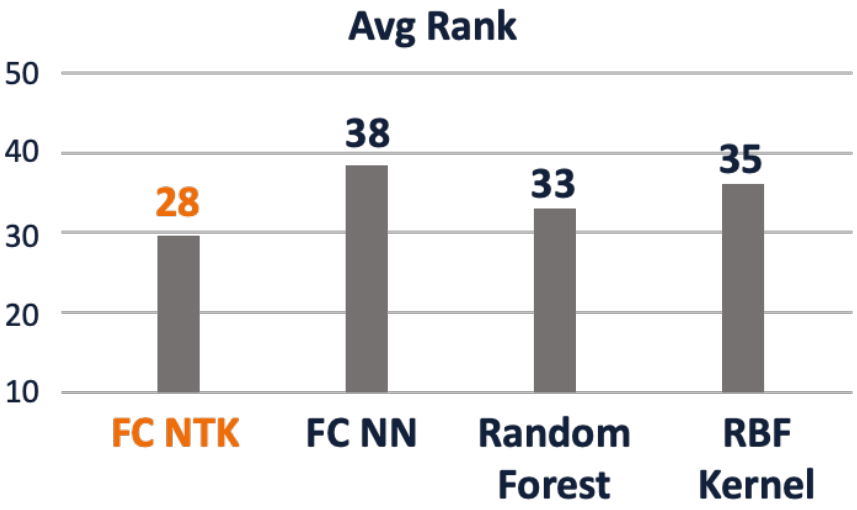
Features



FC NN

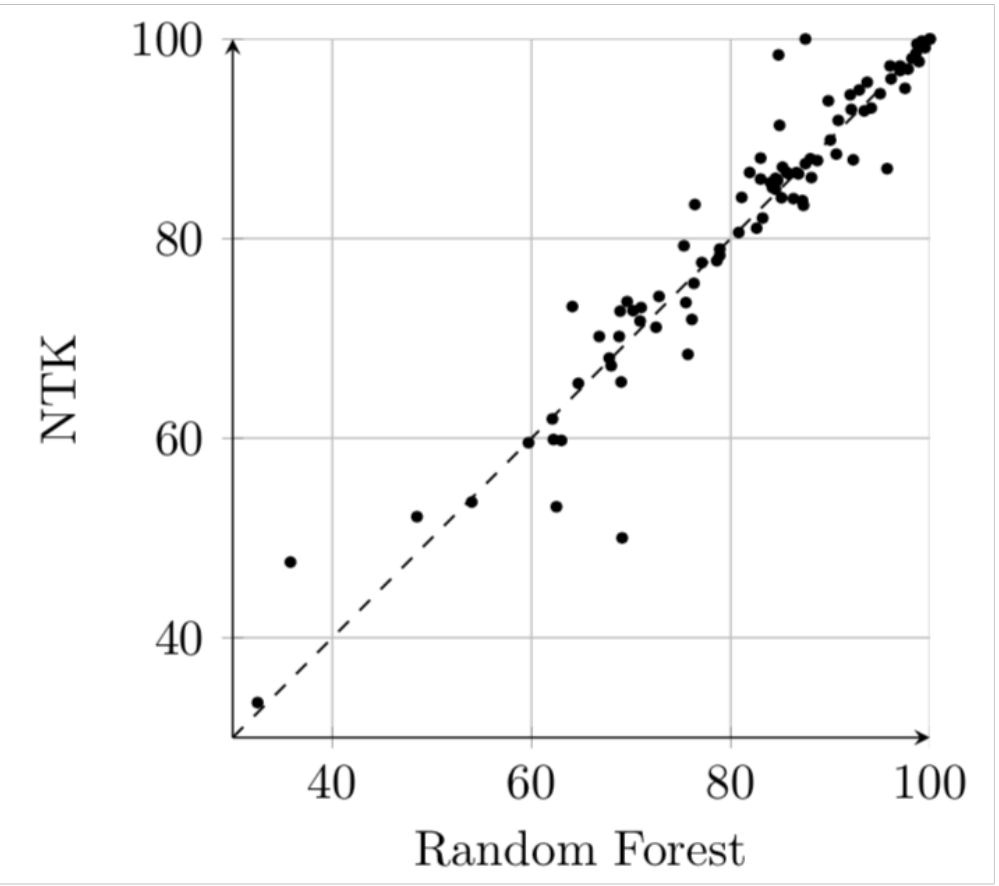
$$k\left(\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}, \begin{pmatrix} -0.3 \\ 0.5 \\ \cdots \\ -0.8 \end{pmatrix}\right)$$

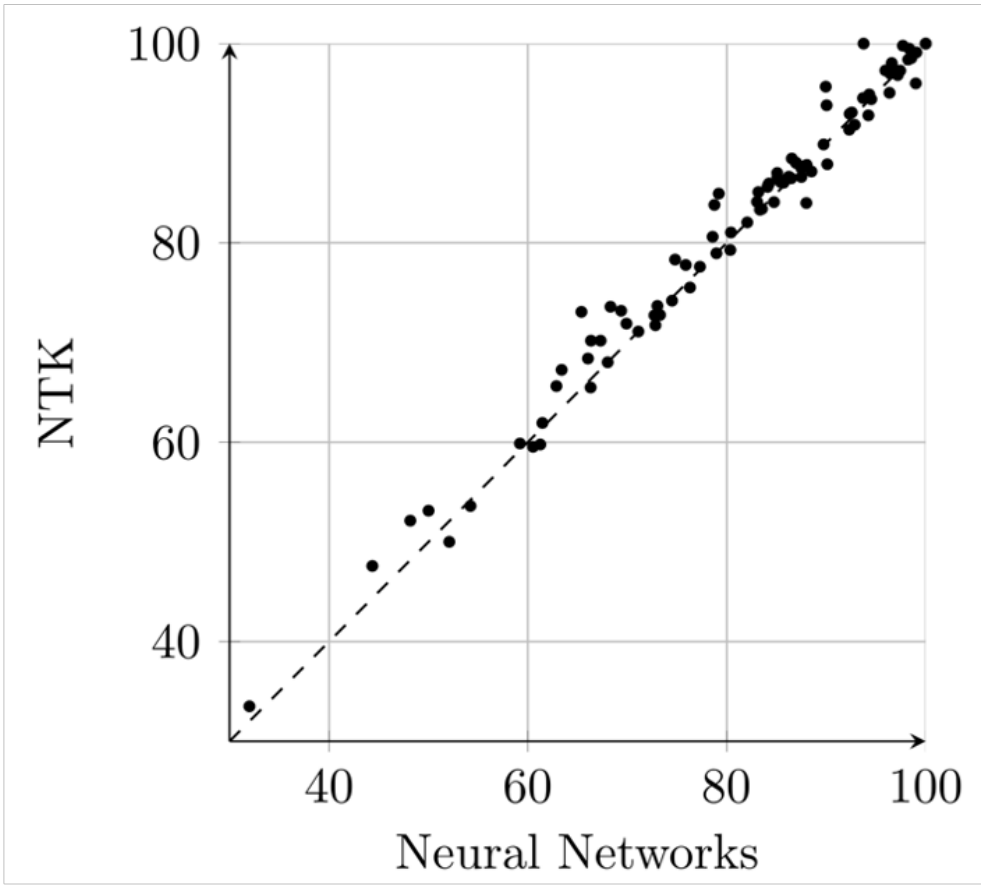
FC NTK



Classifier	Avg Acc	P95	PMA
FC NTK	82%	72%	96%
FC NN	81%	60%	95%
Random Forest	82%	68%	95%
RBF Kernel	81%	72%	94%

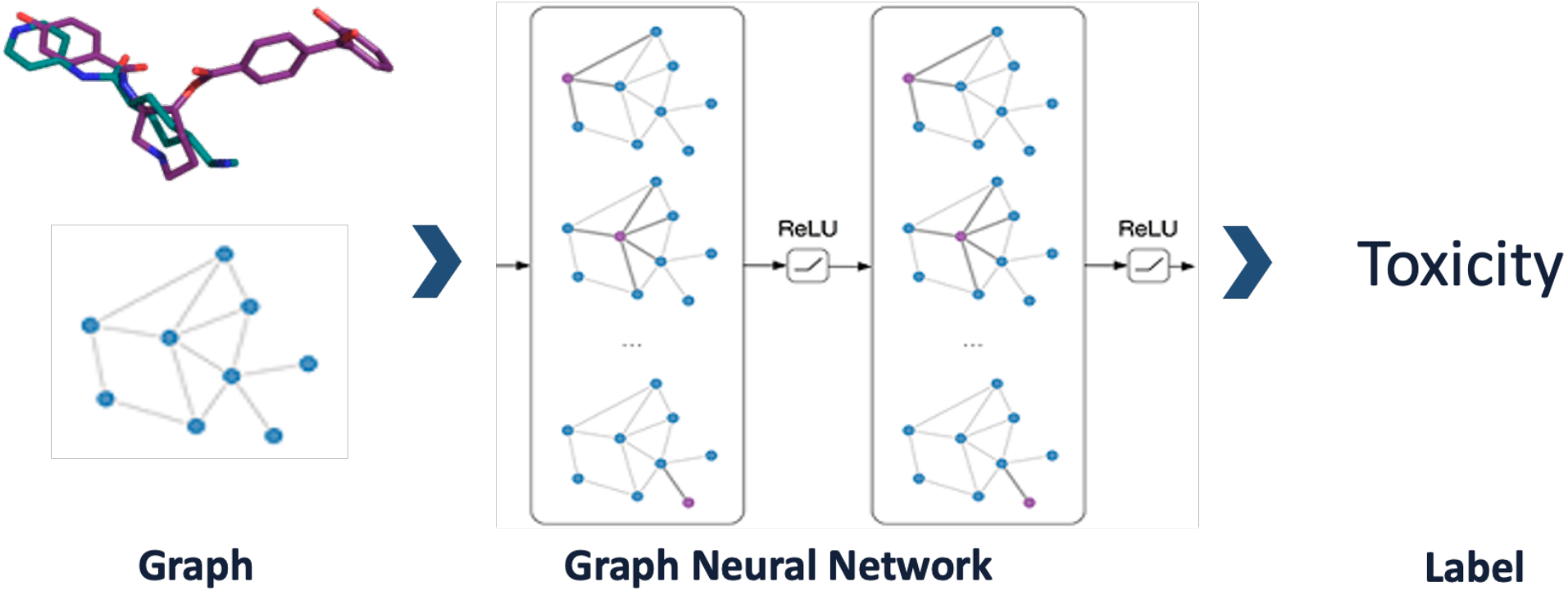
Pairwise Comparisons





Classification Accuracy

Graph Neural Network

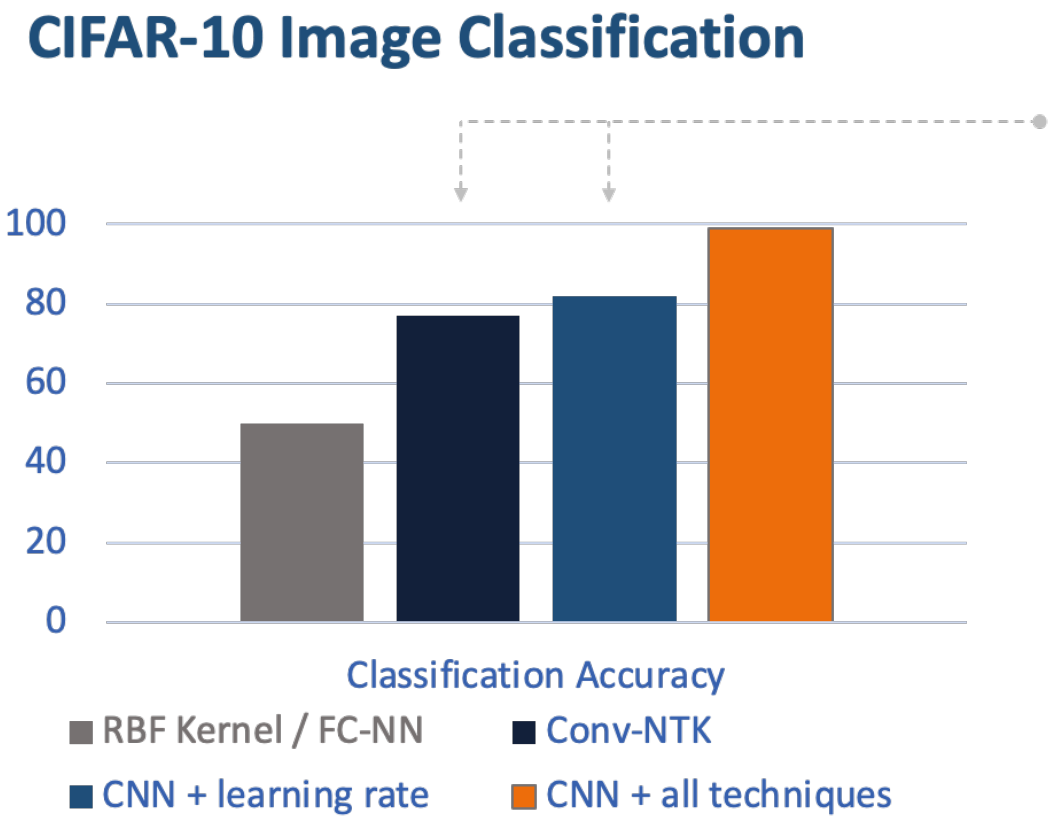


Graph Neural Tangent Kernel

Graph **Graph NN** **Graph NTK**

	Method	COLLAB	IMDB-B	IMDB-M	PTC
GNN	GCN	79%	74%	51%	64%
	GIN	80%	75%	52%	65%
GK	WL	79%	74%	51%	60%
	GNTK	84%	77%	53%	68%

What are left open?



Open Problems:

Why there is a gap:

finite-width?

learning rate?

Understanding techniques:

batch-norm

dropout

data-augmentation

...

Deep Learning Generalization

Measure of Generalization

Generalization: difference in performance on train vs. test.

$$\frac{1}{n}\sum_{i=1}^{n}\ell(f(x_i), y_i) - \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(f(x), y)]$$

Assumption $(x_i, y_i)\ i.i.d. \sim \mathcal{D}$

Problems with the theoretical idealization

Data is not identically distributed:

- Images (Imagenet) are scraped in slightly different ways
- Data has systematic bias (e.g., patients are tested based on symptoms they exhibit)
- Data is result of interaction (reinforcement learning)
- Domain / distribution shift

Meta Theorem of Generalization

Meta theorem of generalization: with probability $1 - \delta$ over the choice of a training set of size n, we have

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

Some measures of complexity:

$\log(|\mathcal{F}|)$

- (Log) number of elements
- VC (Vapnik-Chervonenkis) dimension
- Rademacher complexity
- PAC-Bayes
- …

Classical view of generalization

Decoupled view of generalization and optimization:

- Optimization: find a global minimum: $\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{m} \ell(f(x_i), y_i)$
- Generalization: how well does the global optimizer generalize

Practical implications: to have a good generalization, make sure $\mathcal{F}$ is not too "complex".

Strategies:

- **Direct capacity control:** bound the size of the network / amount of connections, clip the weights, etc.
- **Regularization:** add a penalty term for "complex" predictors: weight decay (ℓ_2 norm), dropout, etc.

Techniques for Improving Generalization

Weight Decay

$\text{Loss} + \frac{\lambda}{2}\|\theta\|_2^2$

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

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

Adam → $\theta_i^{(t+1)} \leftarrow (1 - \eta_t \theta \lambda)\theta_i(t) - \eta_t \sqrt{\epsilon}$

AdamW

want uniform parameters shrinkage for all

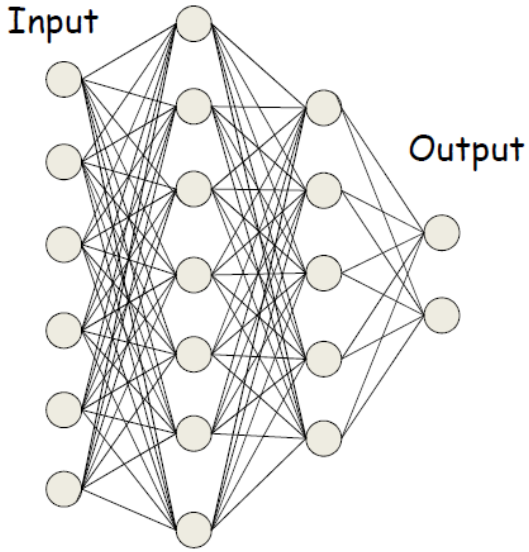
$\theta_i^{(t+1)} \leftarrow (1 - \eta_t(t)\lambda)\theta_i(t) - \eta_t \sqrt{\epsilon}$

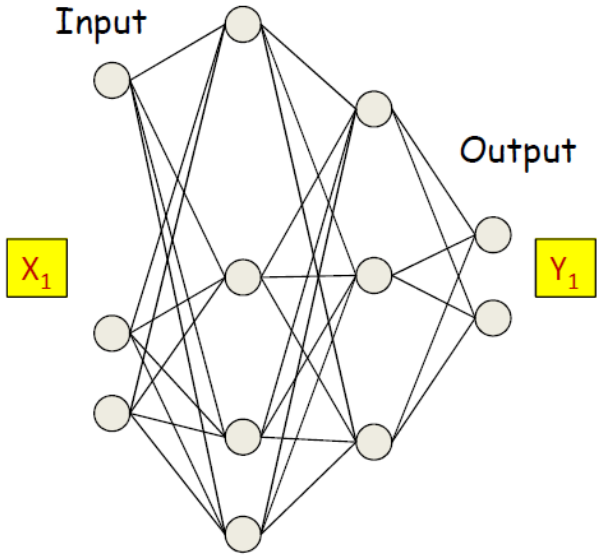
Dropout

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.



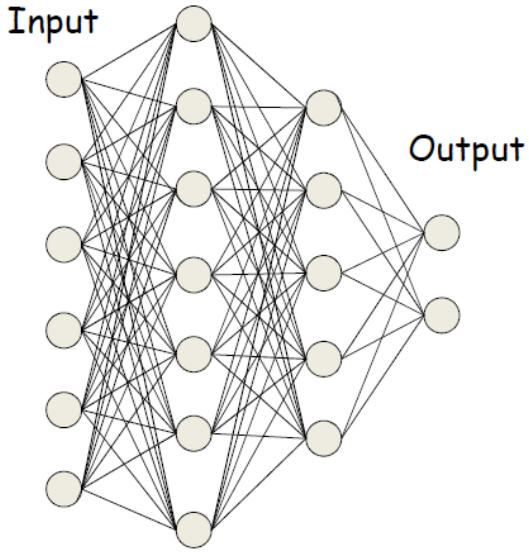


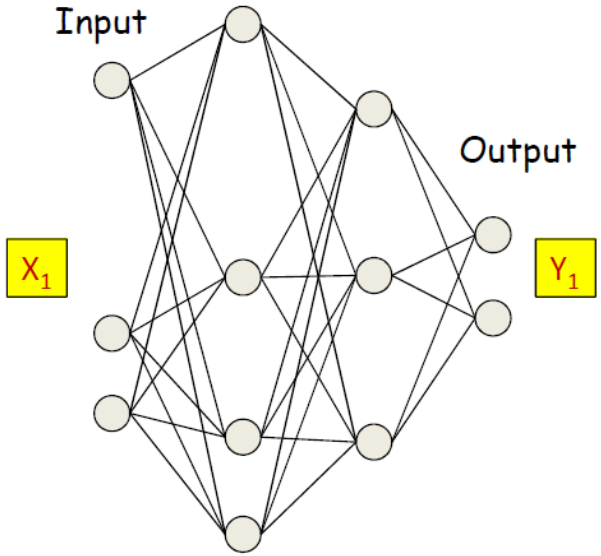
Dropout

Dropout changes the scale of the output neuron:

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

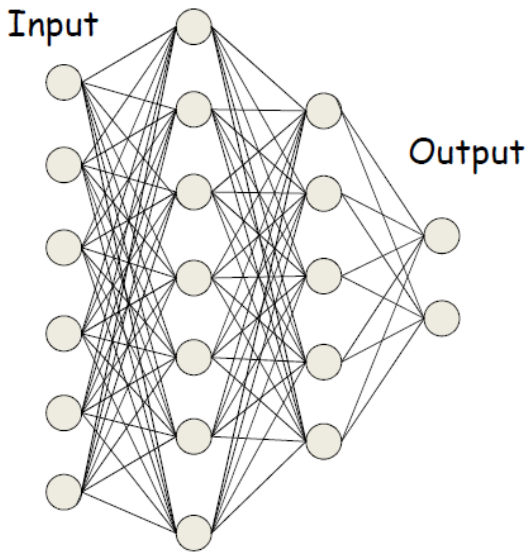
Test time: $y = \alpha\sigma(Wx)$ to match the scale

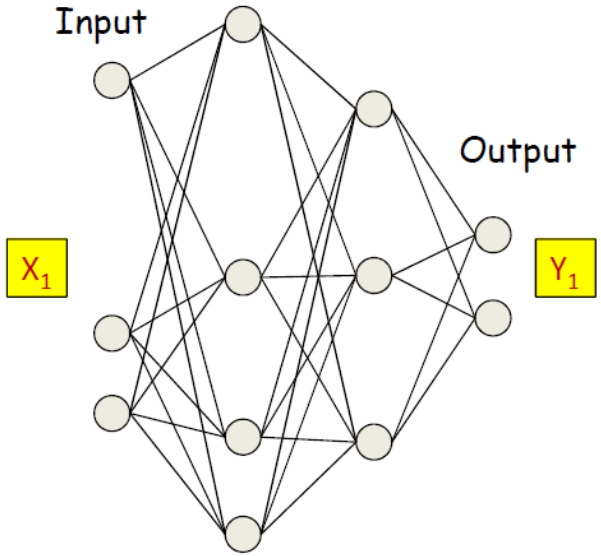




Understanding Dropout

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



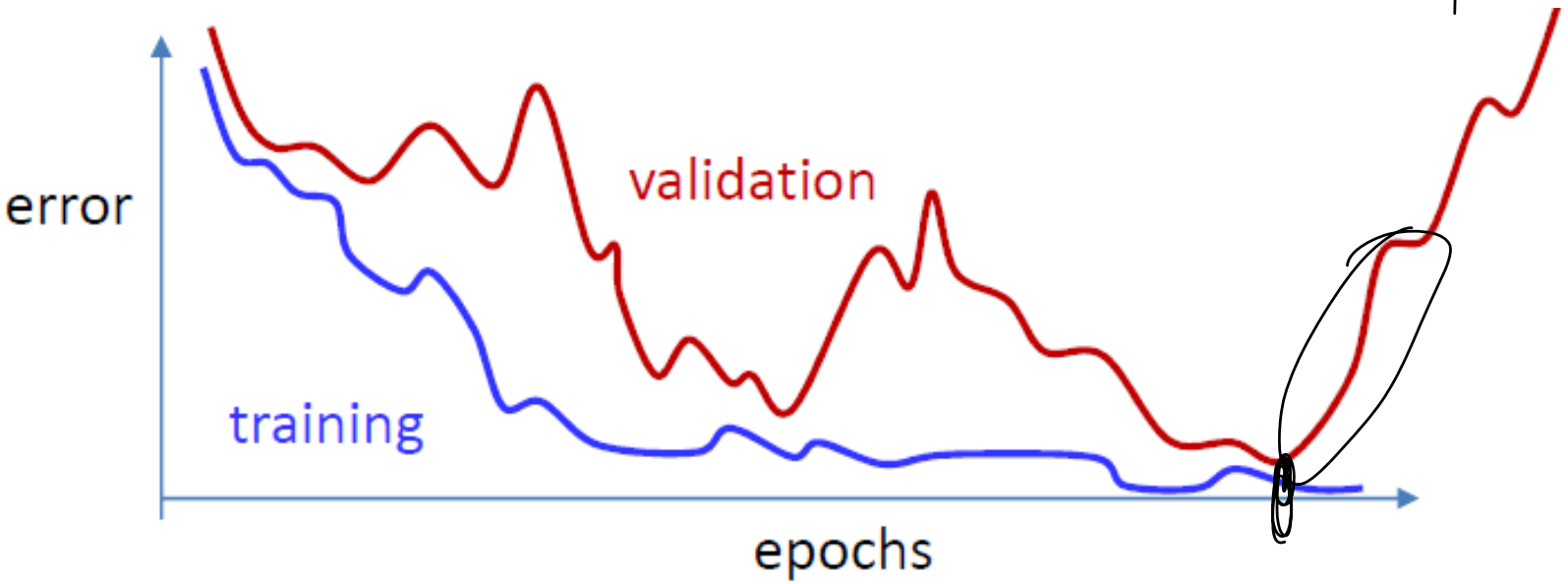


Early Stopping

view # of iterations as a hyperparameter

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

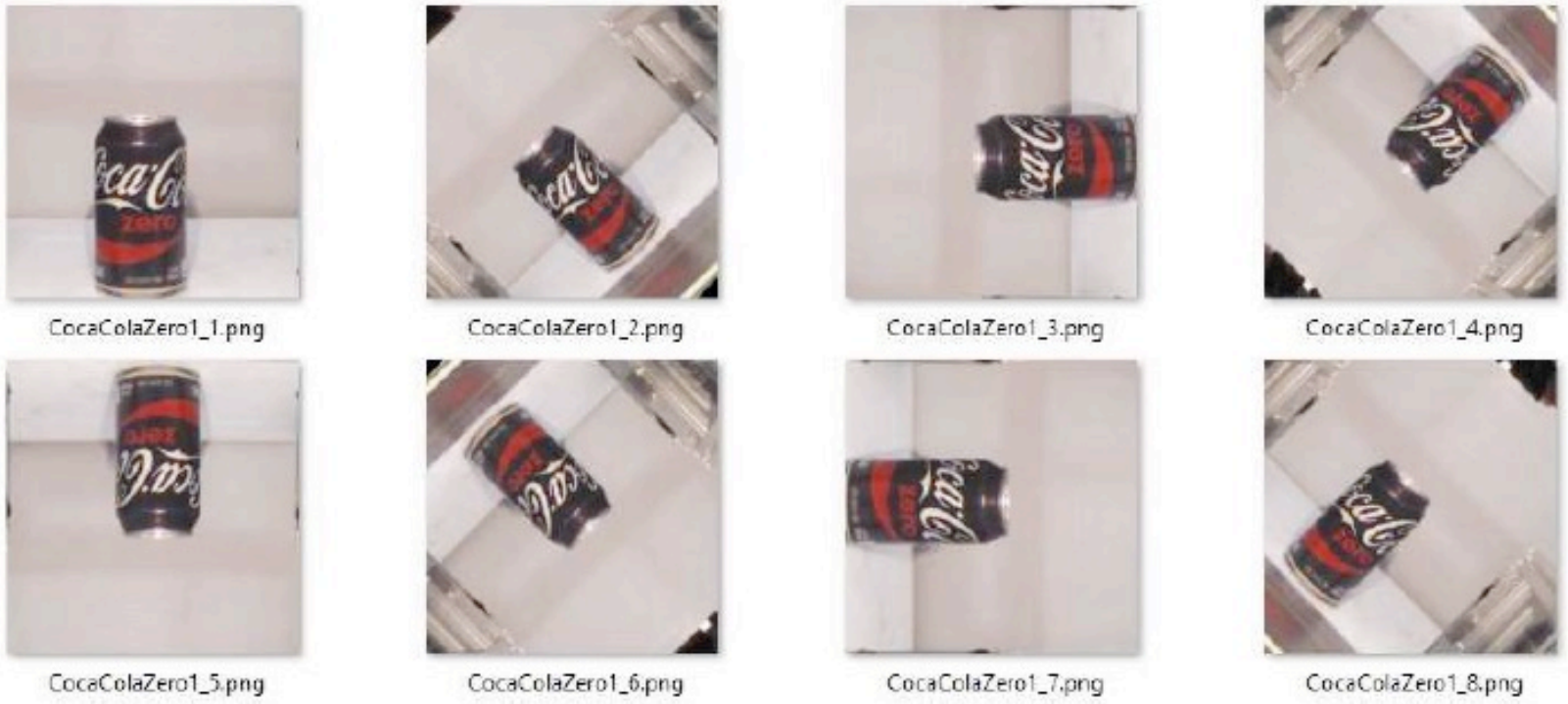
if val(t+t') > val(t) stop



Data Augmentation

Depend on data types.

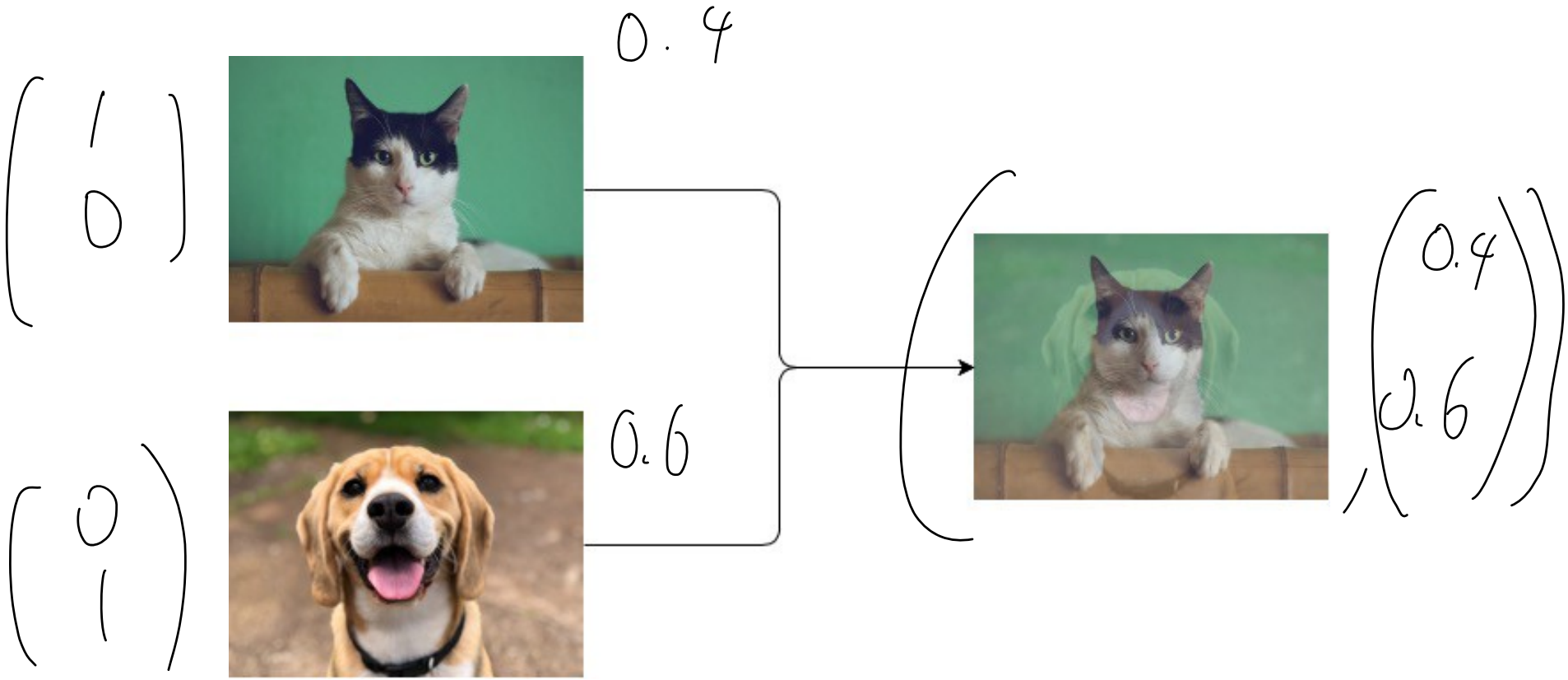
Computer vision: rotation, stretching, flipping, etc



Mixup data augmentation

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

(x_i, y_i),
(x_j, y_j),
classification with one-hot coding



Data Augmentation

Depend on data types.

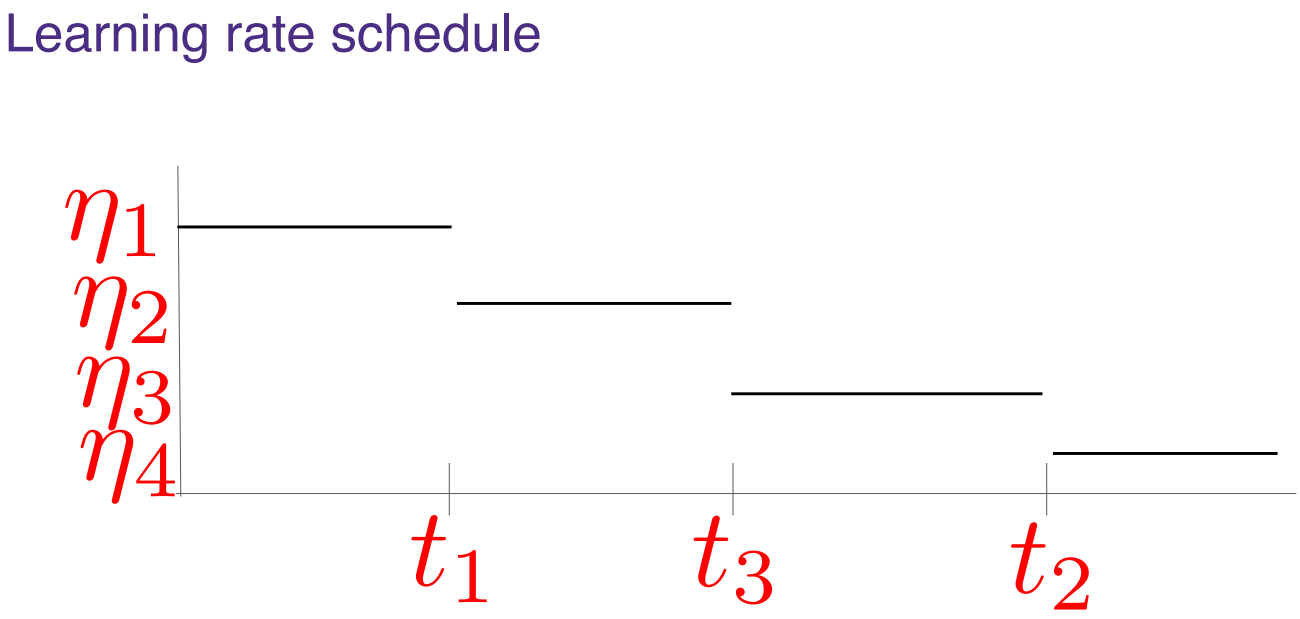
Natural language processing:

- 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.



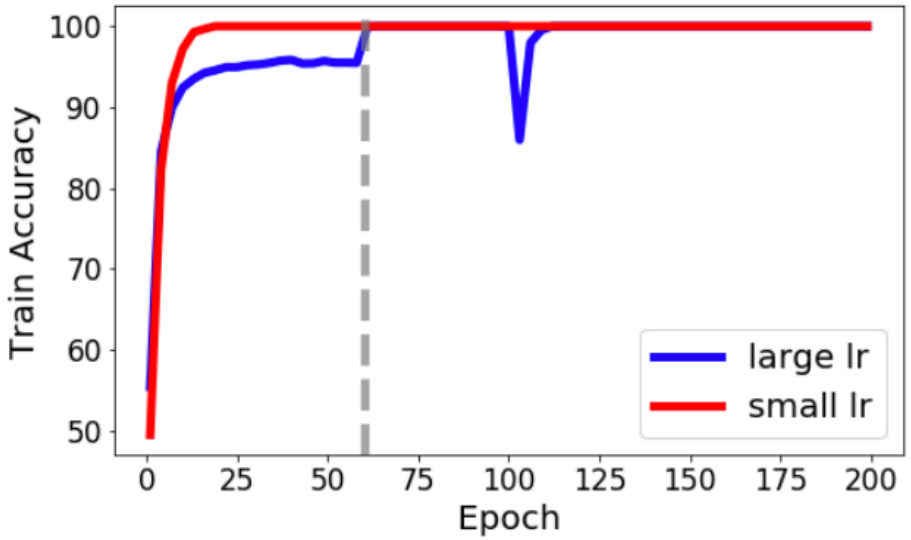
warm, cos, small

Learning rate scheduling

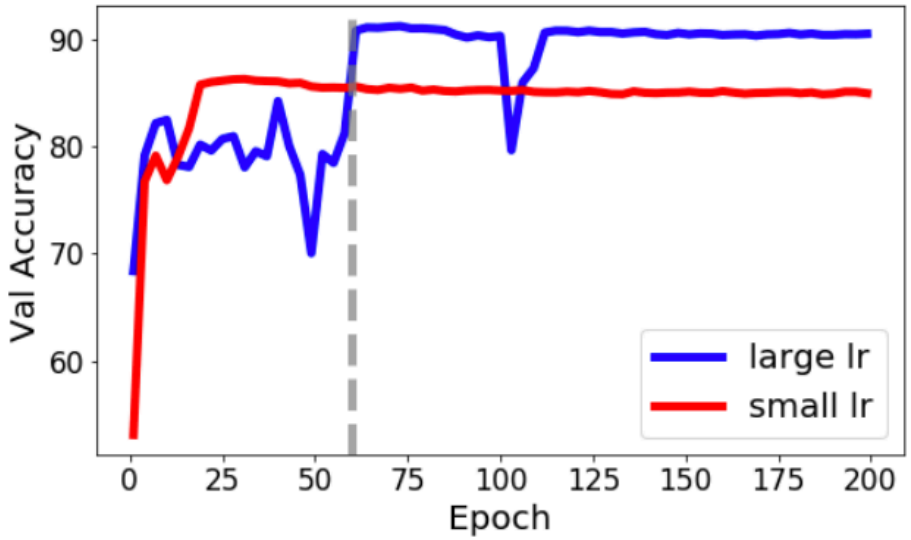
Start with large learning rate. After some epochs, use small learning rate.

Theory:

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



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)
- …