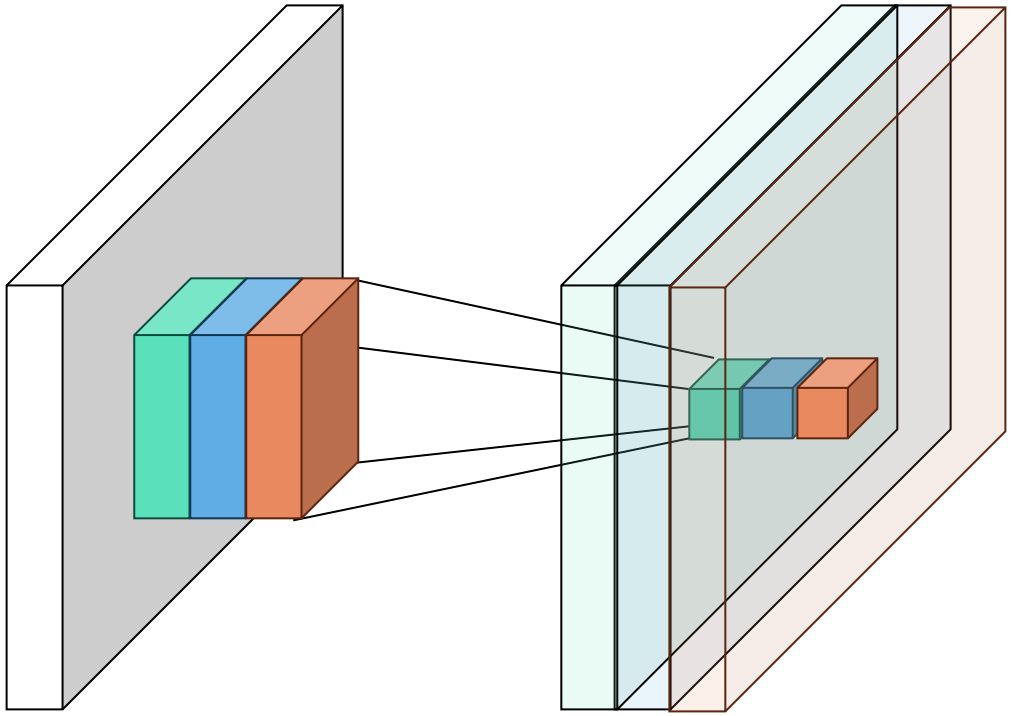
Different Elements in Training Convolutional Neural Networks

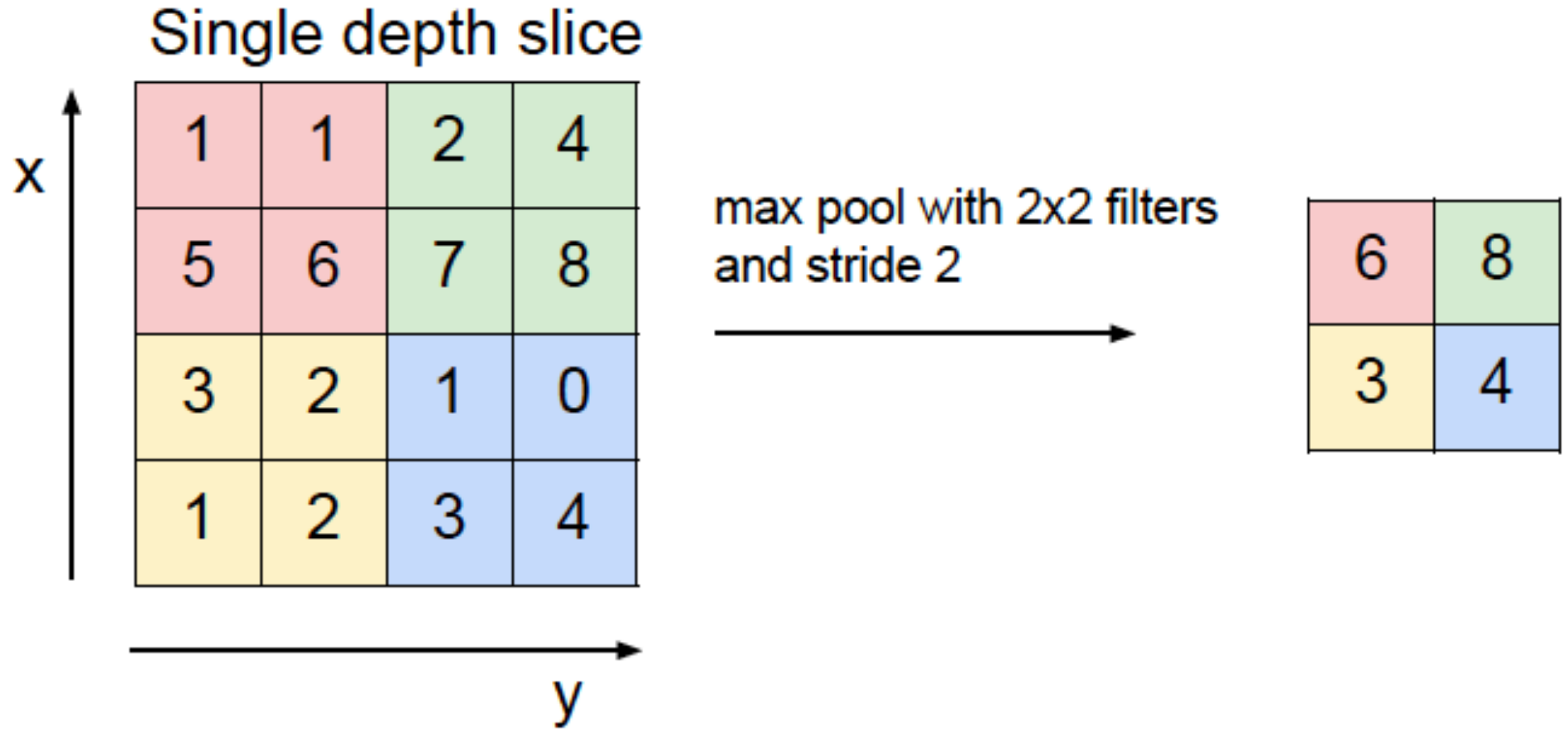
Xiaolong Wang

Last Class -- Convolution



image

Last Class – Max Pooling



This Class

- Optimization methods
- Learning rate
- Recall activation function
- Gradients from SoftMax Loss

Images partially from: http://cs231n.stanford.edu/, https://slazebni.cs.illinois.edu/fall20/

Optimization methods

Mini-batch SGD

Loop:

- Sample a batch of data with size m: $(x_1, y_1), \ldots, (x_m, y_m)$, and forward the data to compute the loss $l(W, x_i, y_i)$.
- Use back-prop to compute the gradients for each layer:

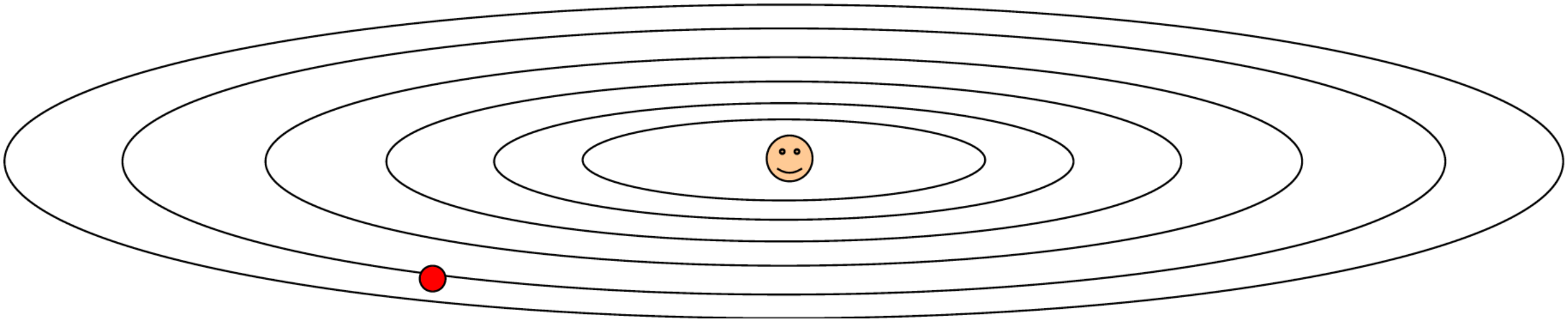
$$\nabla \hat{L} = \frac{1}{m} \sum_{i=1}^{m} \nabla l(W, x_i, y_i)$$

- Update the model parameters with learning rate α:

$$W \leftarrow W - \alpha \nabla \hat{L}$$

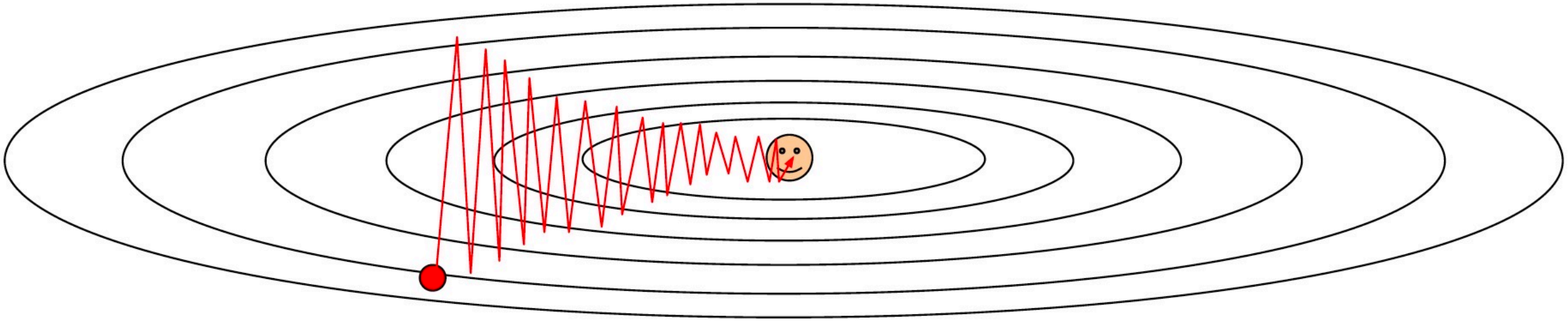
Problems with SGD

- Gradients are unstable



Problems with SGD

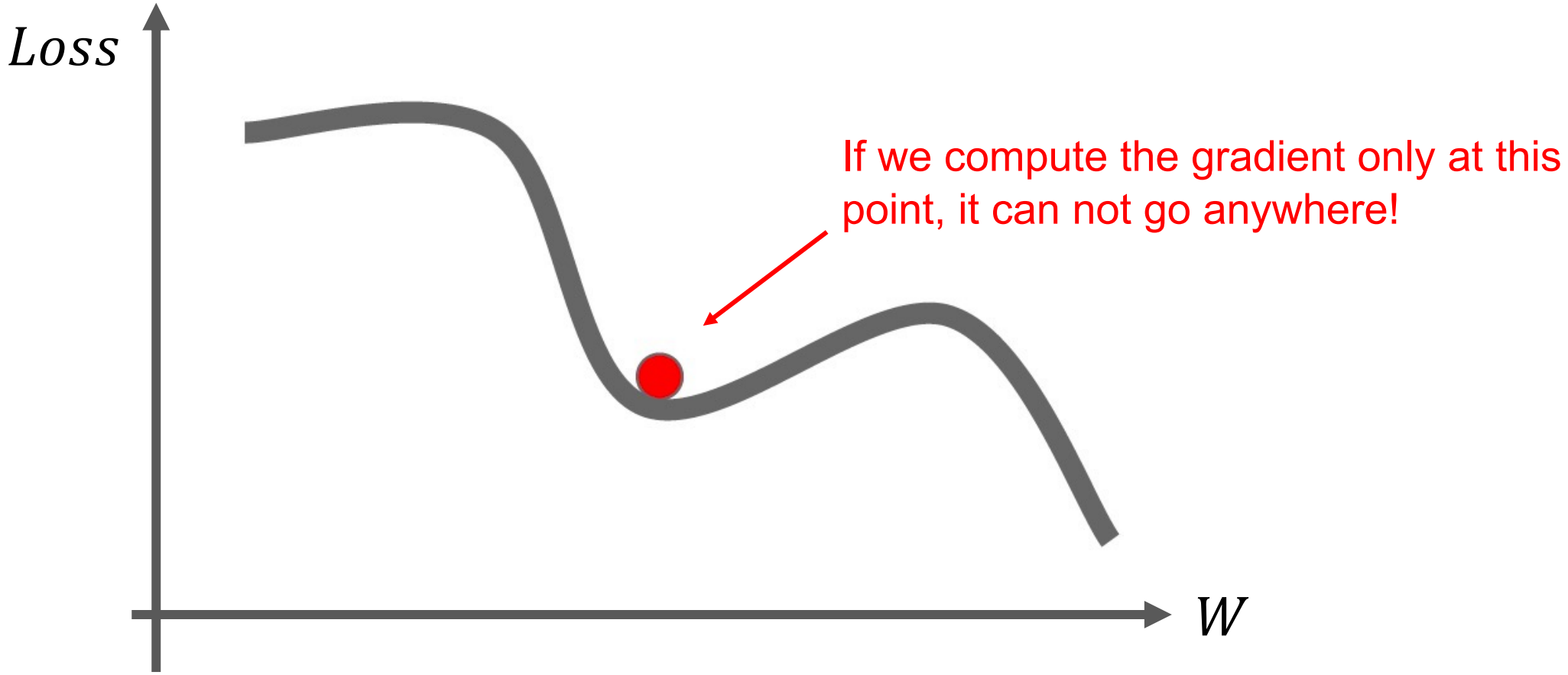
- Gradients are unstable



- Each time the gradient is estimated on a small batch, it will jitter a lot.

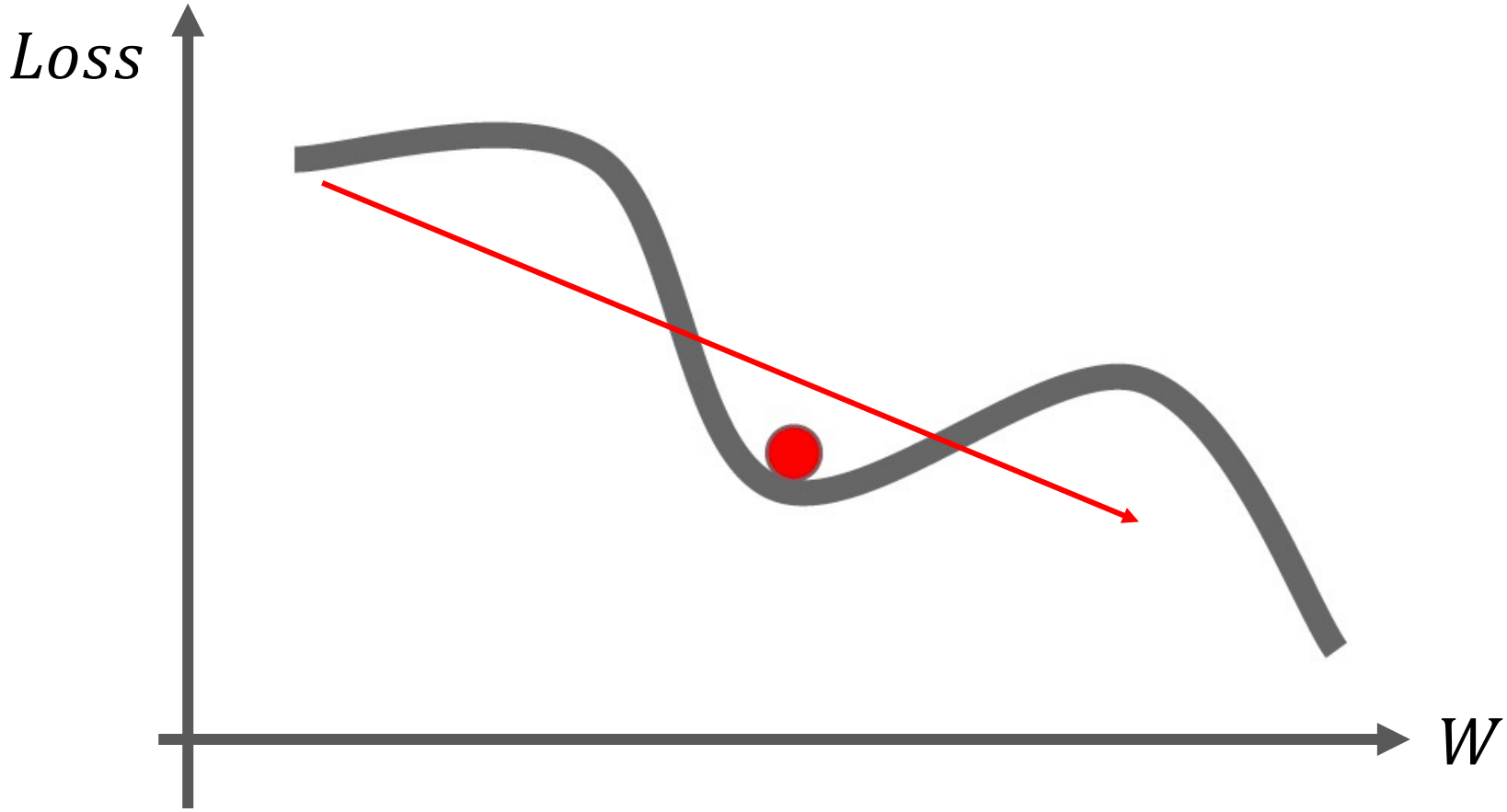
Problems with SGD

- It is easy to get stuck in a local minima:



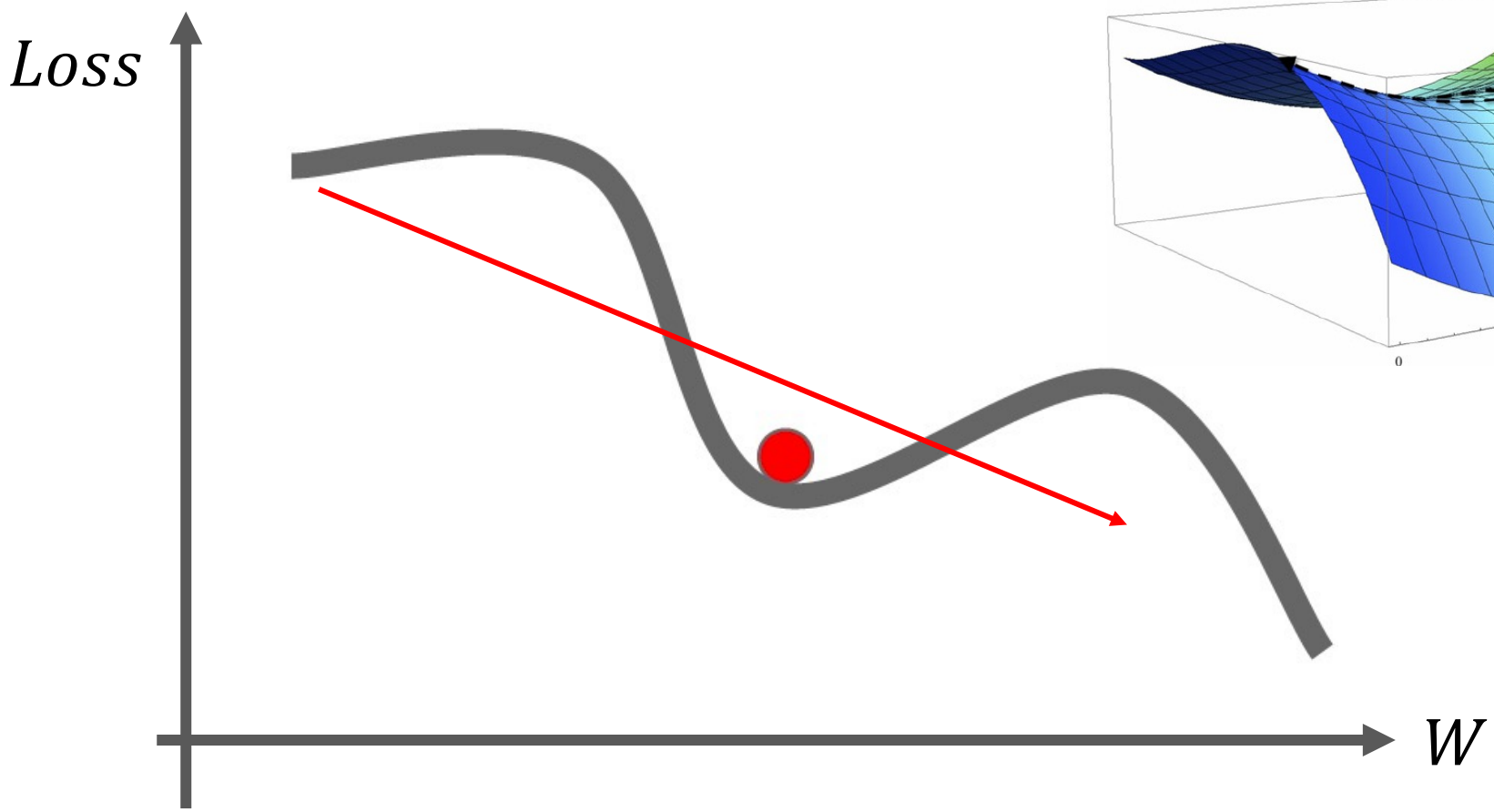
Problems with SGD

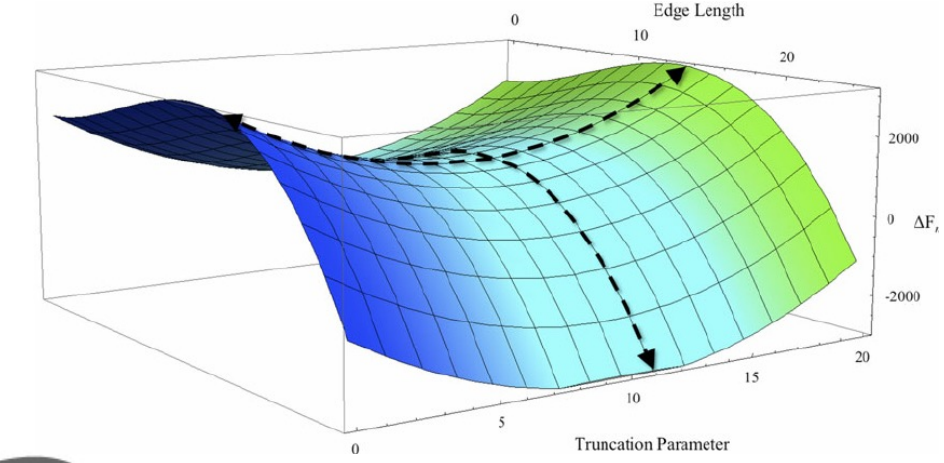
- Can we follow the trend instead just a local gradient?



Problems with SGD

- Can we follow the trend instead just a local gradient?





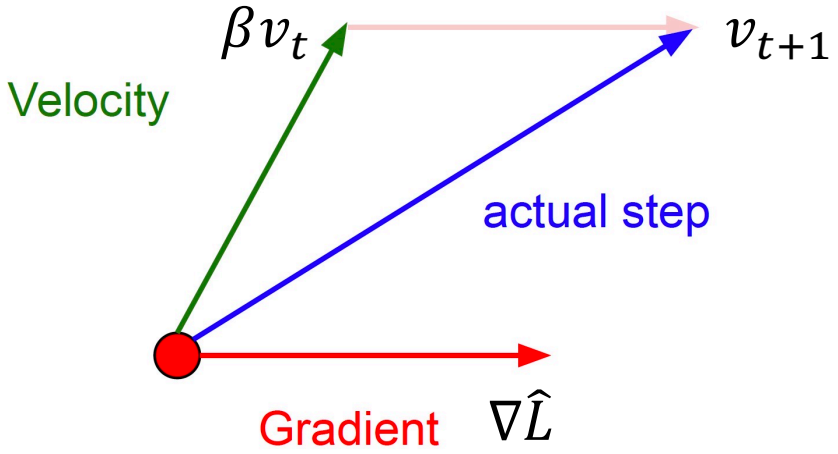
Saddle Points

SGD with momentum

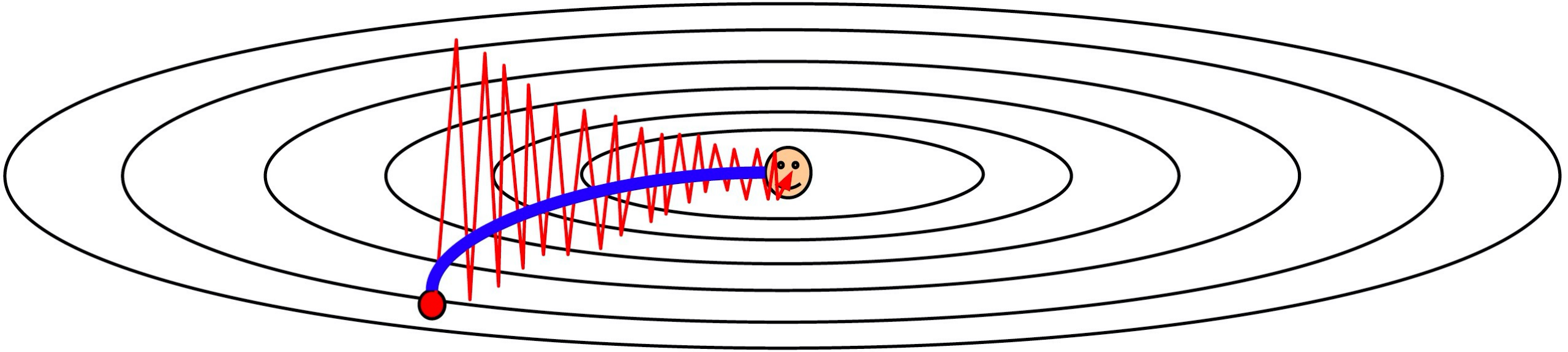
- Use a momentum variable as a weighted average of previous gradients:

$$v_{t+1} = \beta v_t + \nabla \hat{L}$$
$$W_{t+1} = W_t - \alpha v_{t+1}$$

We use t to index different training iterations, $\beta = 0.9$ for training image classifiers.



SGD with momentum



Adaptive Gradient Descent

- Different parameters / weights can converge in different speed
- Especially different weights in different layers have different scales of gradients
- Can we adjust the learning rate automatically?

AdaGrad

- Track the previous gradients, accumulate the magnitudes, and adjust the learning rate.

- We want to give large learning rate for gradients with low magnitudes and give small learning rate for gradients with high magnitudes:

$$m_{t+1} \leftarrow m_t + \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} \frac{\partial L}{\partial W_t}$$

RMSProp

- The history of gradients are accumulated without forgetting in AdaGrad

- Introducing a decay factor β (≥ 0.9) to reduce the effect of the previous gradients

$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} \frac{\partial L}{\partial W_t}$$

J. Duchi, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011

Adam

- Combining RMSProp and momentum:

$$v_{t+1} = \beta_1 v_t + \frac{\partial L}{\partial W_t}$$

$$m_{t+1} \leftarrow \beta_2 m_t + (1 - \beta_2) \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} v_{t+1}$$

Default parameters from paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 1e-3$, $\epsilon = 1e-8$

D. Kingma and J. Ba, Adam: A method for stochastic optimization, ICLR 2015

Which Optimizer to use? (Common Practice)

- SGD is widely used in (with larger lr, e.g., lr = 1e-2):
  - Image classification
  - Object detection
  - Other recognition tasks

- Adam is widely used in (with lower lr, e.g., lr = 2e-4):
  - Image synthesis
  - Image reconstruction
  - Other reconstruction tasks

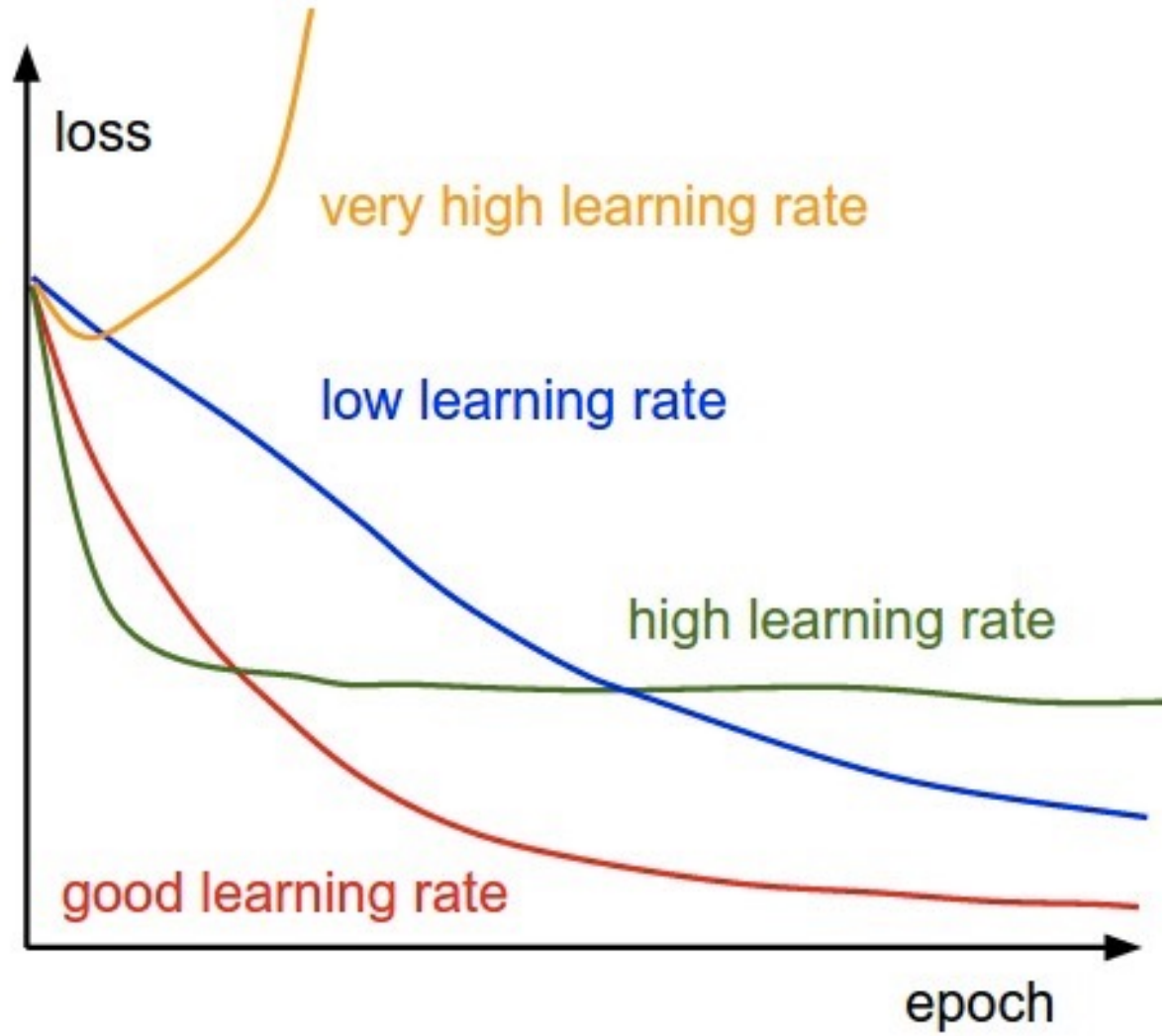
Which Optimizer to use? (Common Practice)

- SGD gives better performance than Adam in recognition tasks, but Adam is usually more stable and converge faster

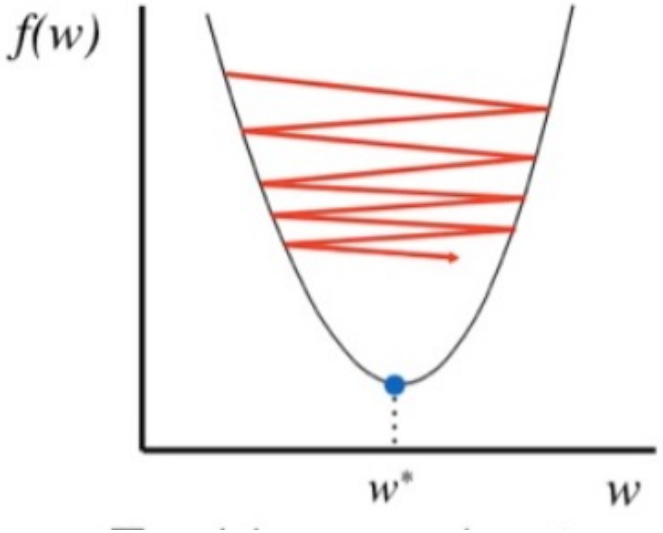
- For challenging tasks, if SGD does not work, we can try to use Adam

Learning Rate α

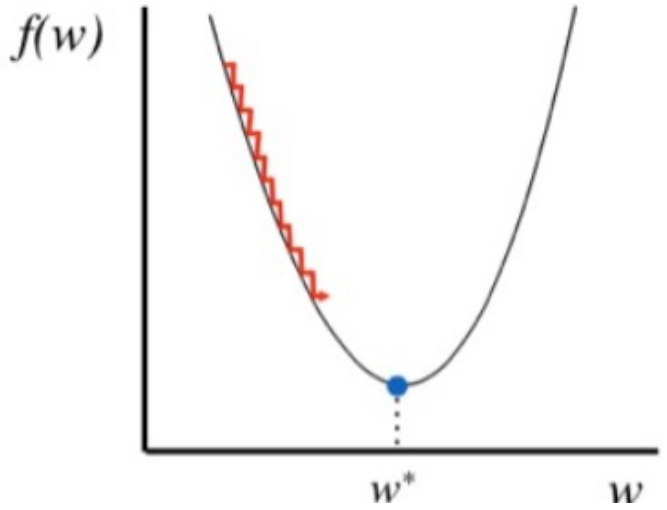
Learning rate



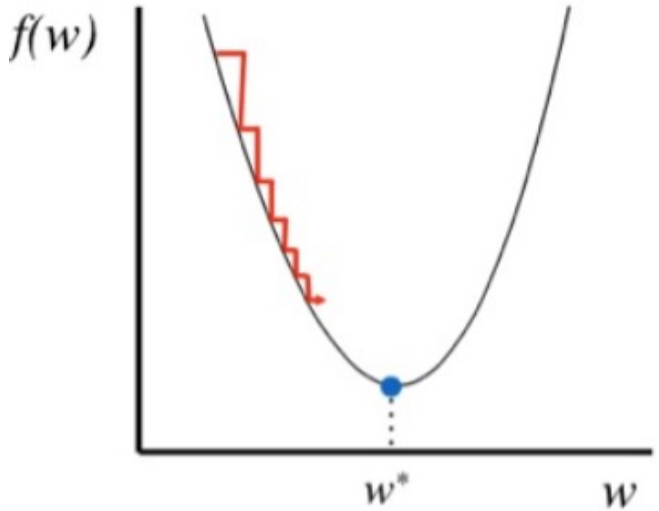
Learning rate



Learning rate too large:
Hard to converge
and will even diverge



Learning rate too small:
Converge very slowly
and can easily fall in local minima

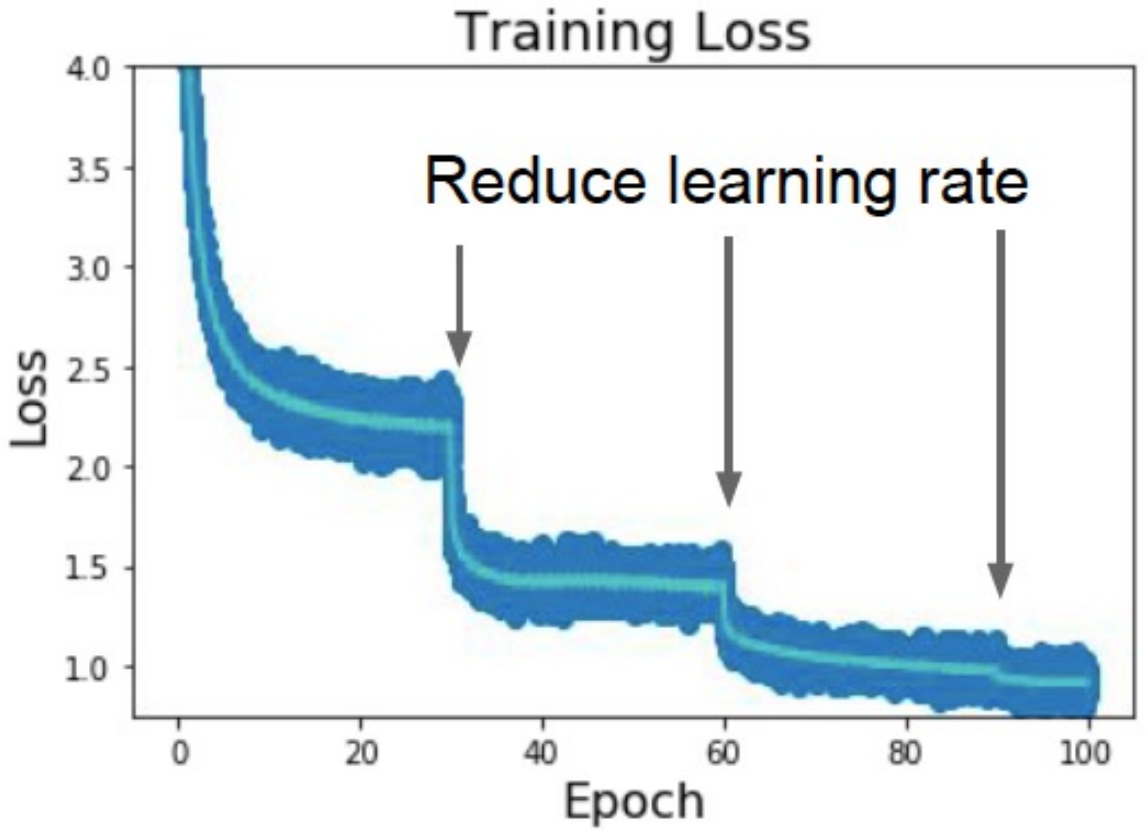


Start with large learning rate
and reduce over time

https://www.slideshare.net/ssuserf88631/scalable-machine-learning-73621818

Learning Rate Decay

- The most common practice: multiply the LR with a constant every K epochs



Training image classifier with ImageNet:

Start LR=0.1

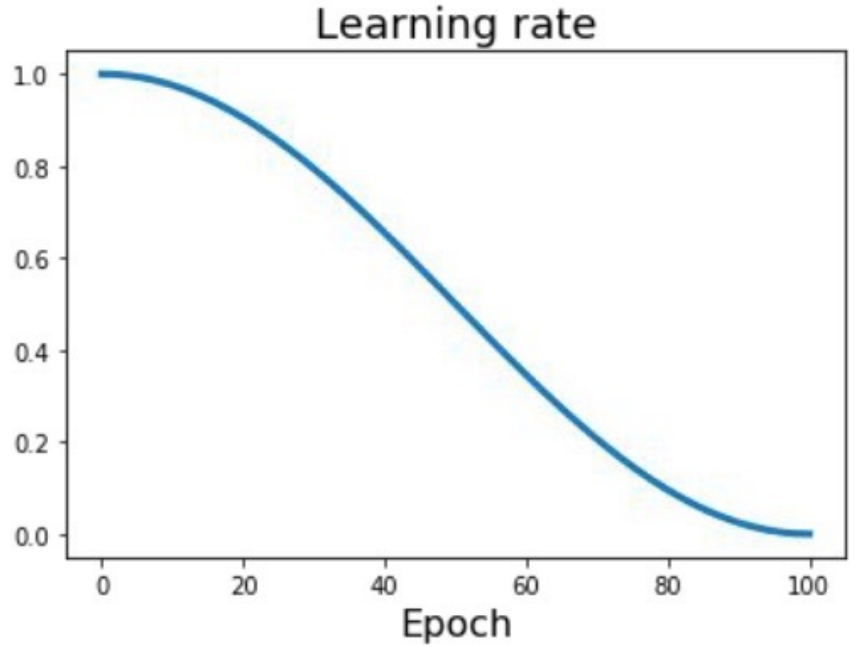
Decay LR by multiplying 0.1 every 30 epochs:
0~30 epochs: 0.1
30~60 epochs: 0.01
60~90 epochs: 0.001

Learning Rate Decay

- Cosine decay, continuously decay over time:

$$a_t = \frac{1}{2} a_0 (1 + \cos(t\pi/T))$$

with T as the total number of epochs, t as the current epoch #

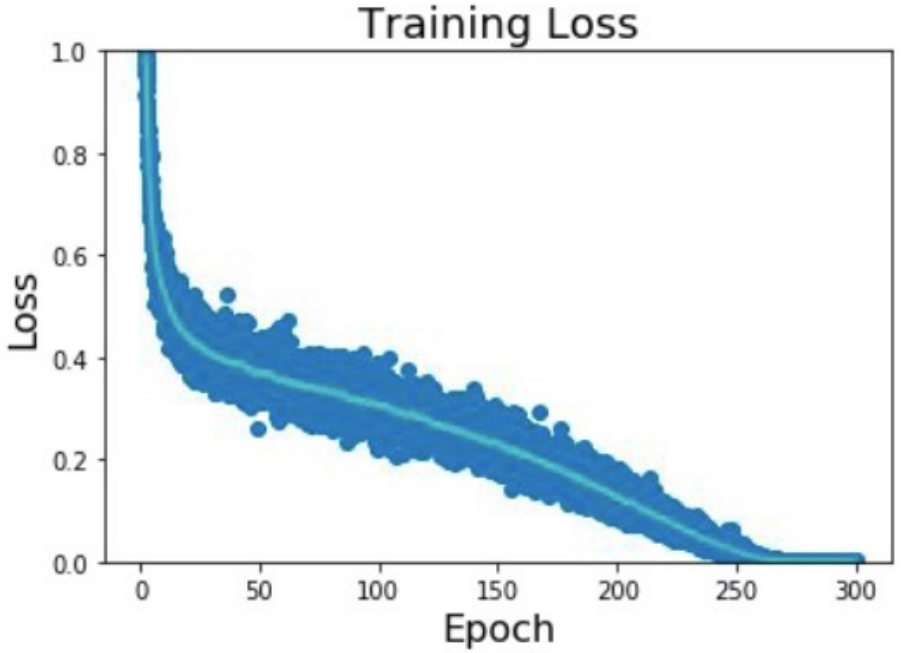


Learning Rate Decay

- Cosine decay, continuously decay over time:

$$a_t = \frac{1}{2} a_0 (1 + \cos(t\pi/T))$$

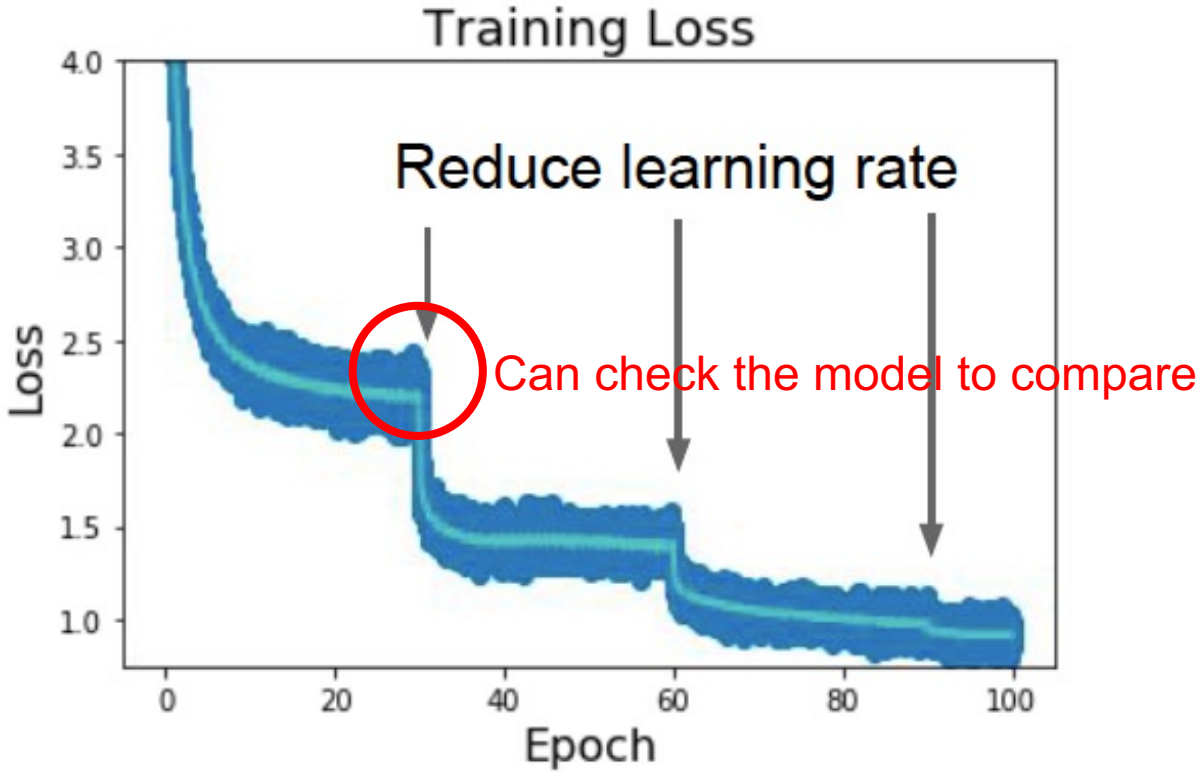
with T as the total number of epochs, t as the current epoch #



Some experience on setting learning rate

- Try to make the initial LR as large as possible
  - Good for exploration
- The initial LR is the most important one
  - Most related to the performance
  - Less easy to overfit
- Check the model performance after first LR for debug

Some experience on setting learning rate

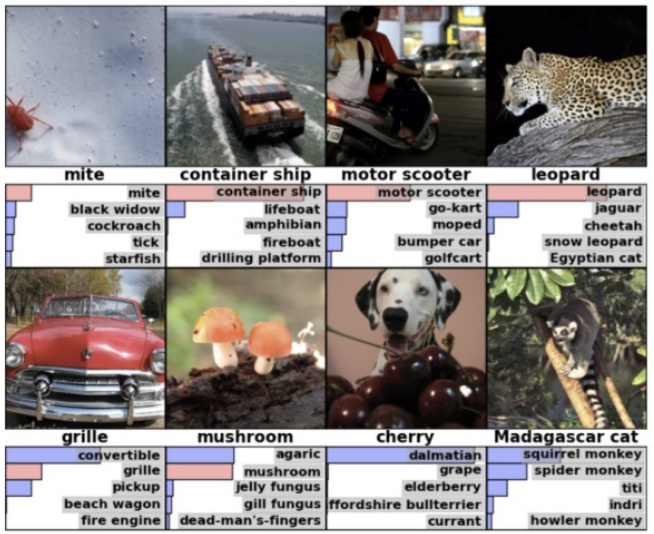


- Instead of decaying LR at 30,60,90 epochs, if you want to save time, you can train with the first LR longer (50 epochs) and train with the following LR with shorter time (10 epochs each)

Warm Up for LR

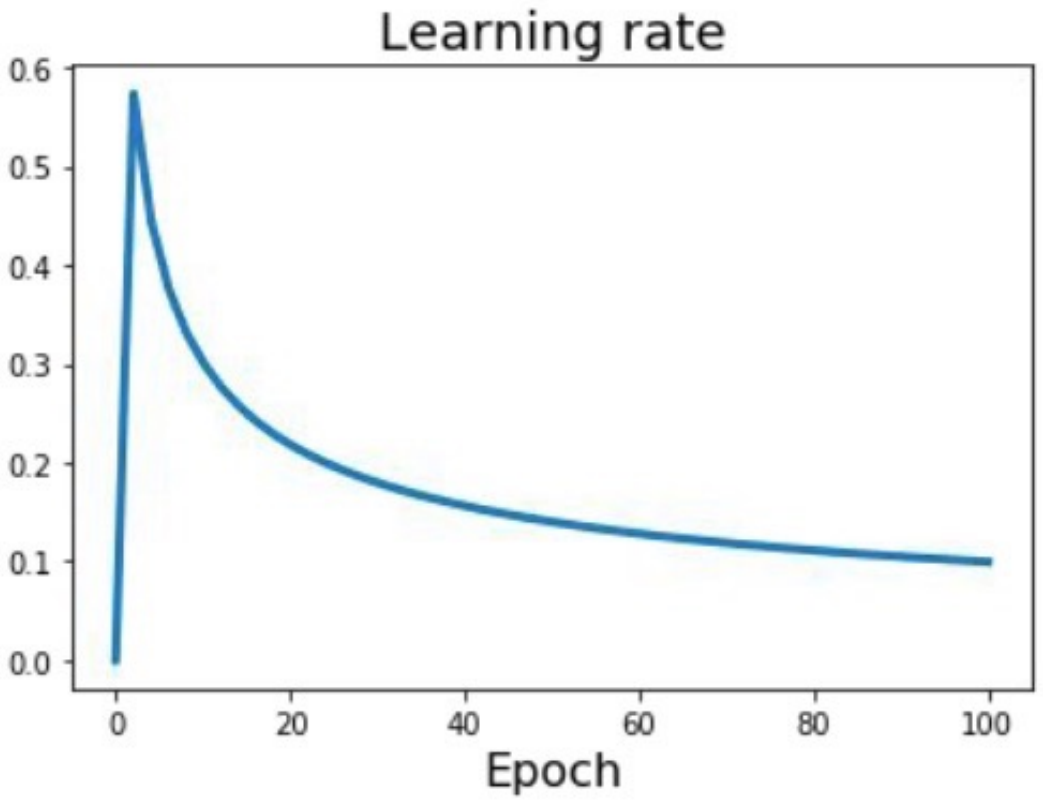
- Sometimes the pre-trained wights or initialized weights are hard for training using large LR.

- For example, using ImageNet classification pre-trained network and further finetune for detection.



Warm Up for LR

- We can first use a small LR and then increase it to a large LR in a few hundred iterations



Increase the LR from 0, in the first 5000 iterations

Empirical rule

- If you times the batch size by k, you should increase the learning rate by k.

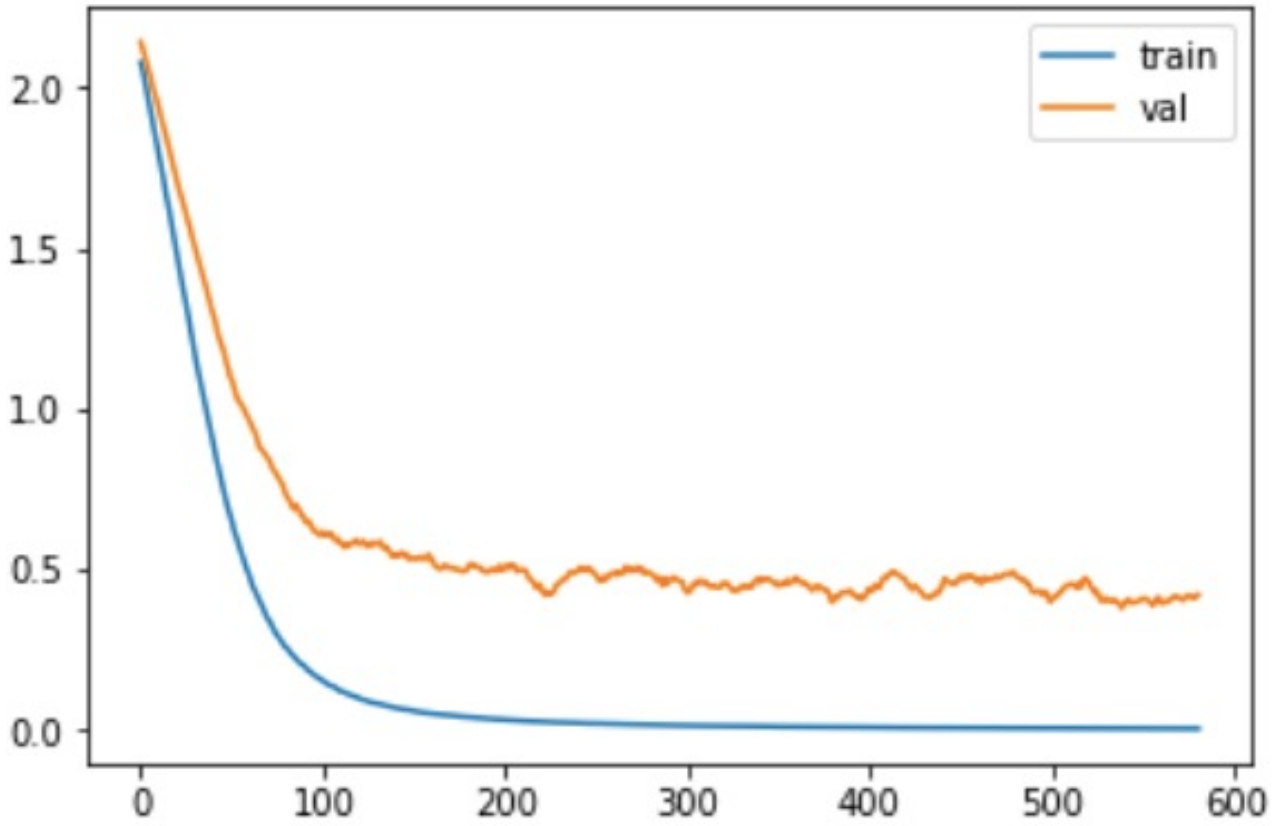
- For example, if you train a classification network using batch size 128, and learning rate 0.1

-> batch size 256, learning rate 0.2

-> batch size 512, learning rate 0.4

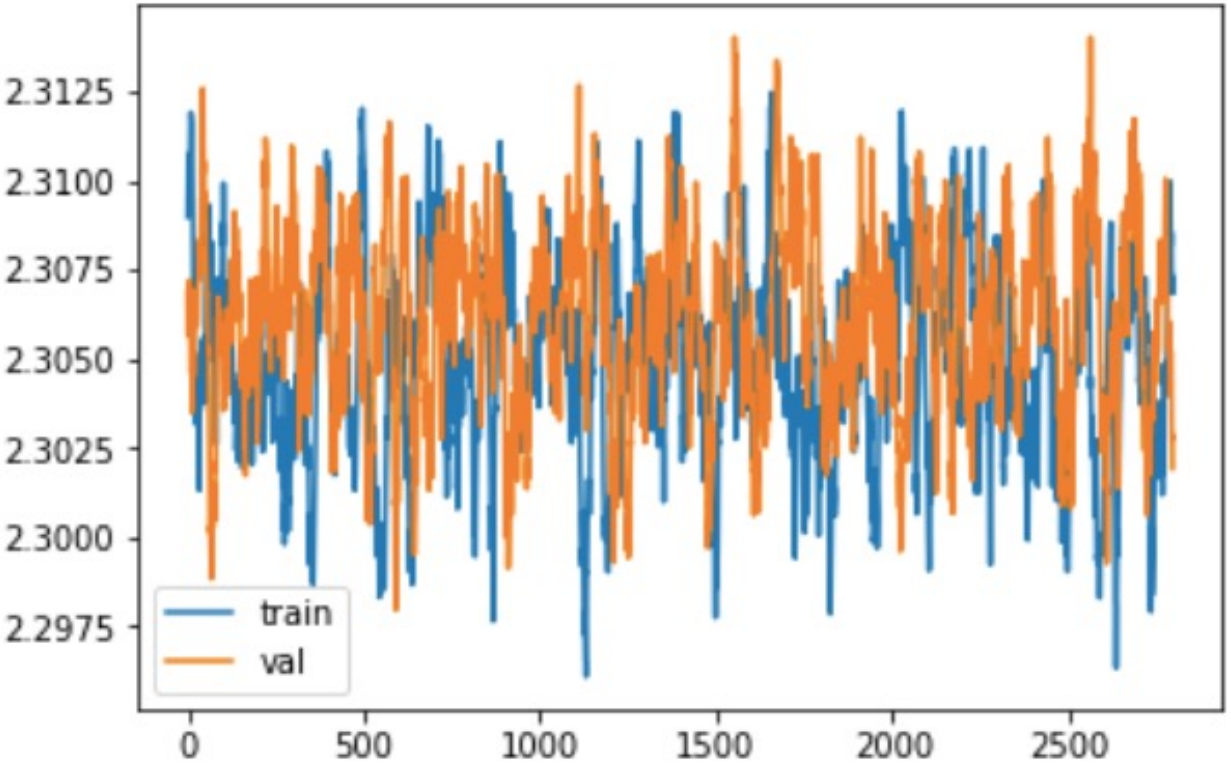
Debug with Training Curve

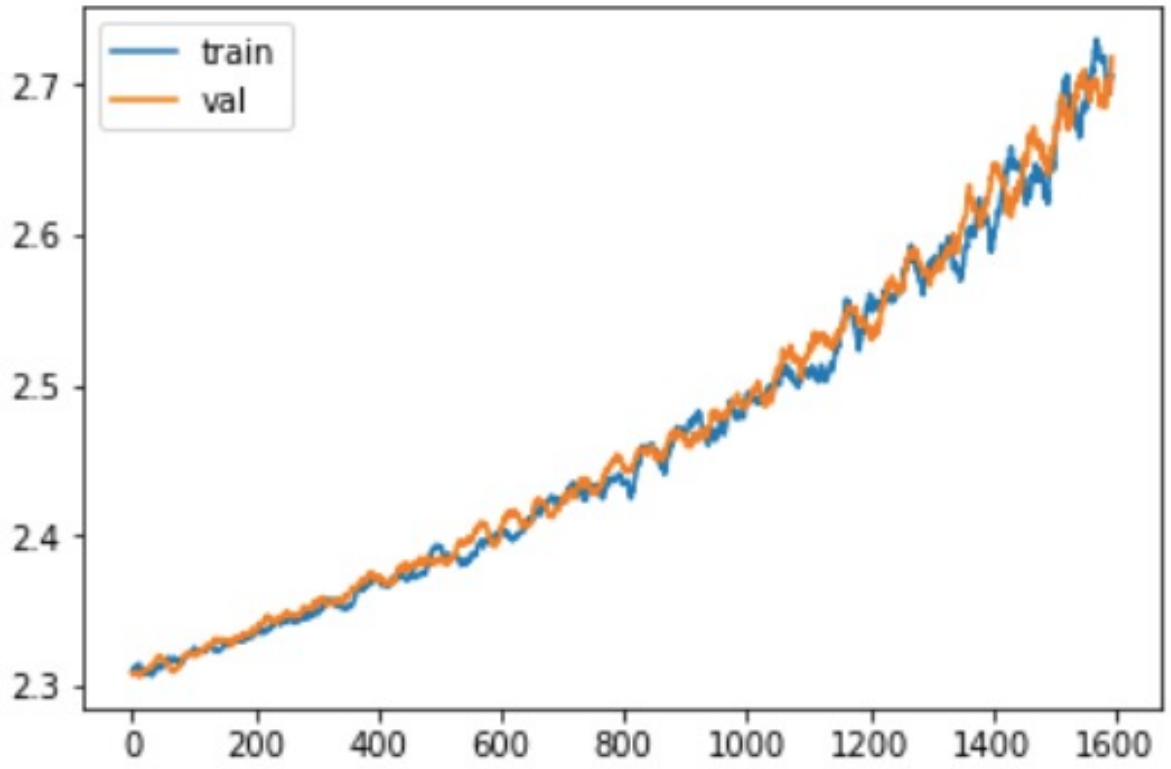
- When it works well:



Debug with Training Curve

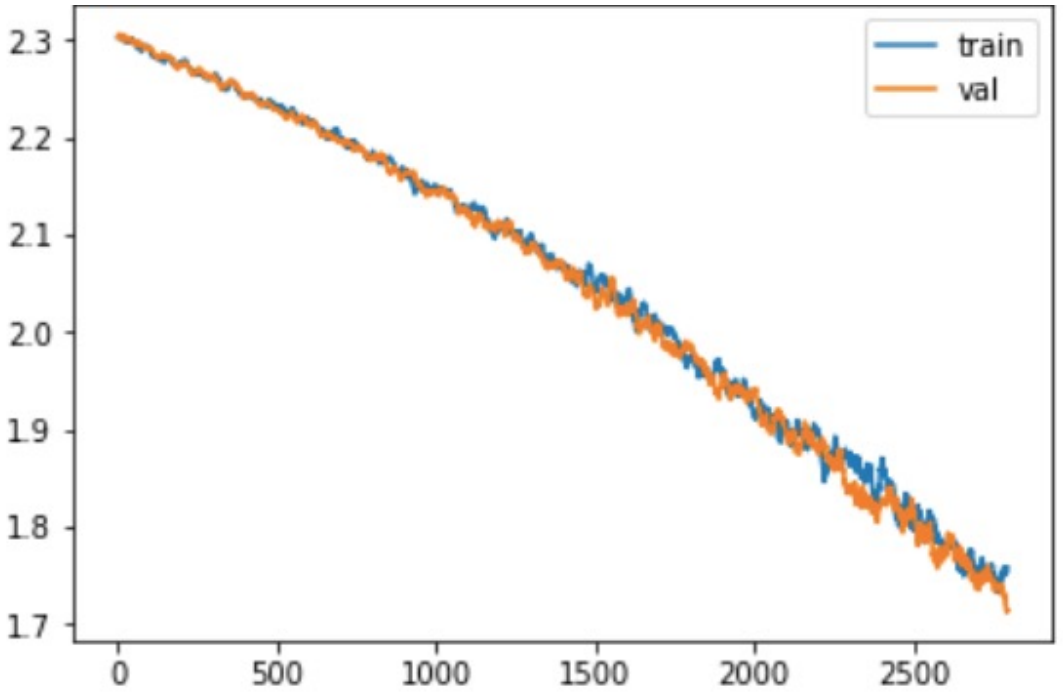
- When there is a bug and it does not get training



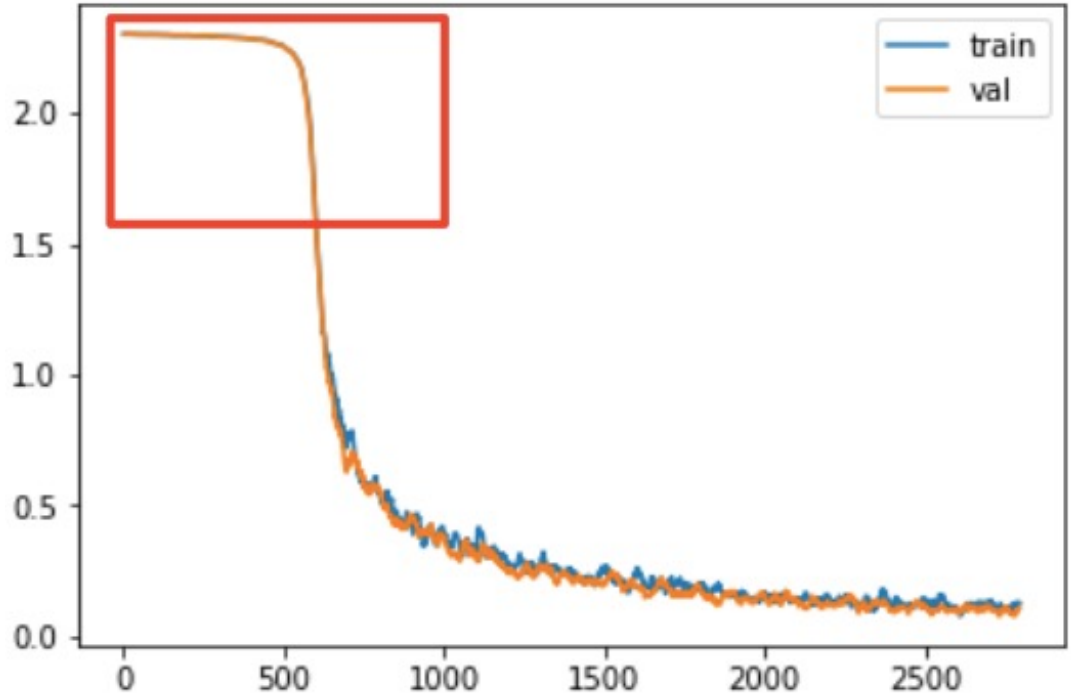


Debug with Training Curve

- In the process of training



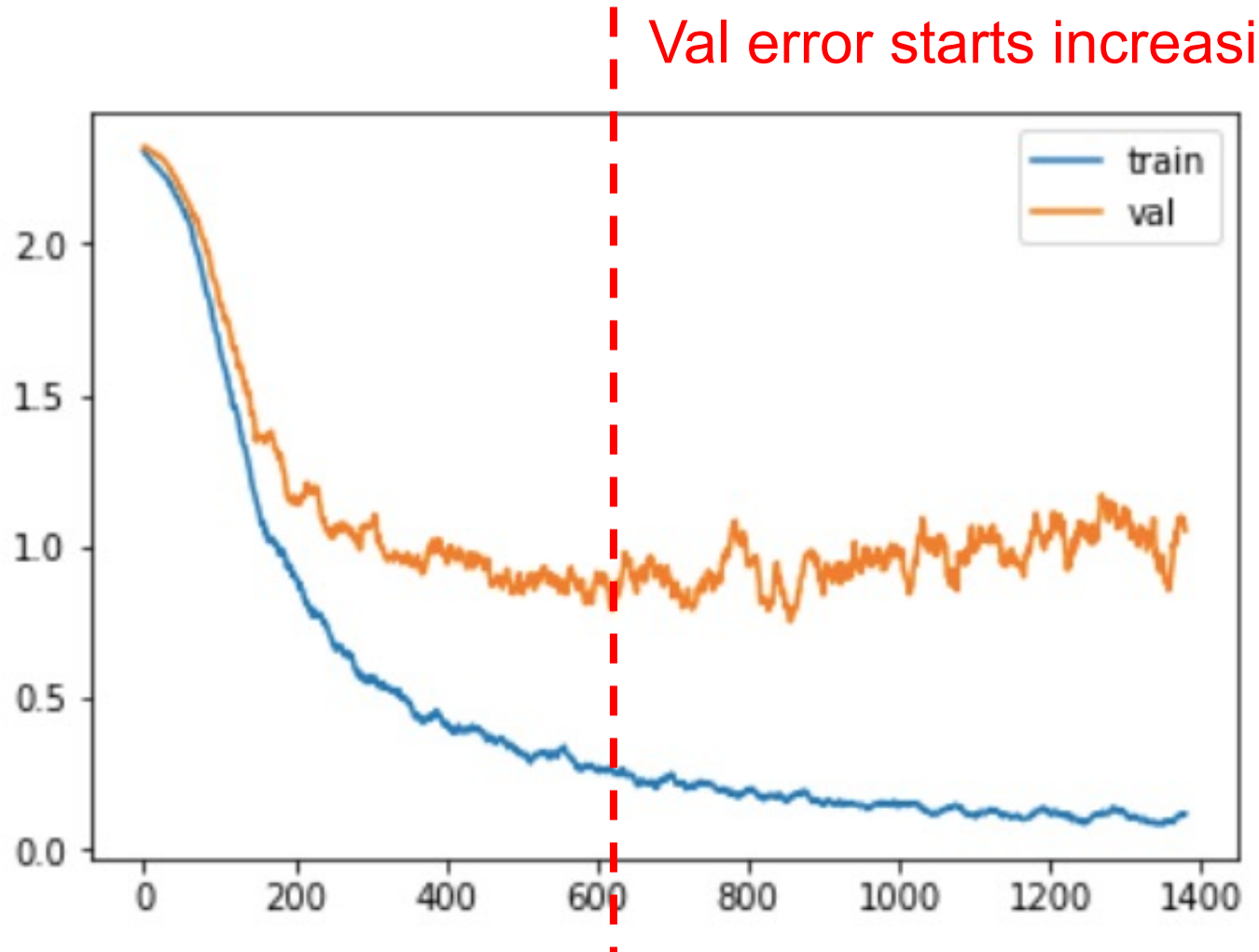
Not converged yet
possibly increase learning rate



Slow start
Bad initialization?

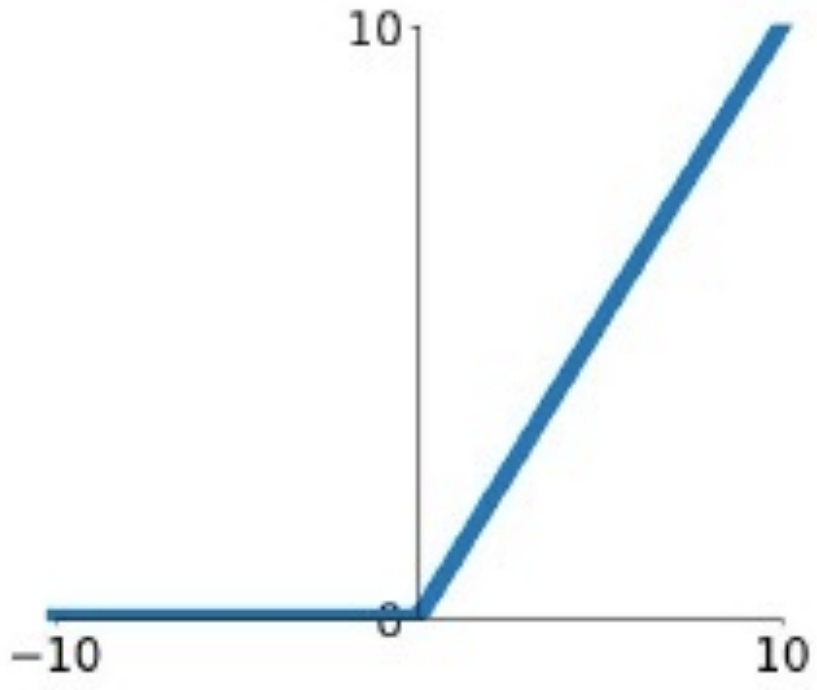
Debug with Training Curve

- Overfitting



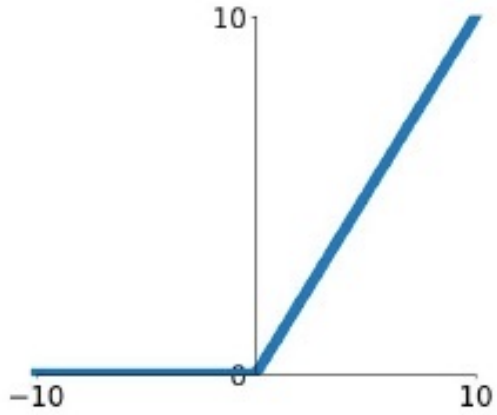
Recall ReLU Activation Function

ReLU function



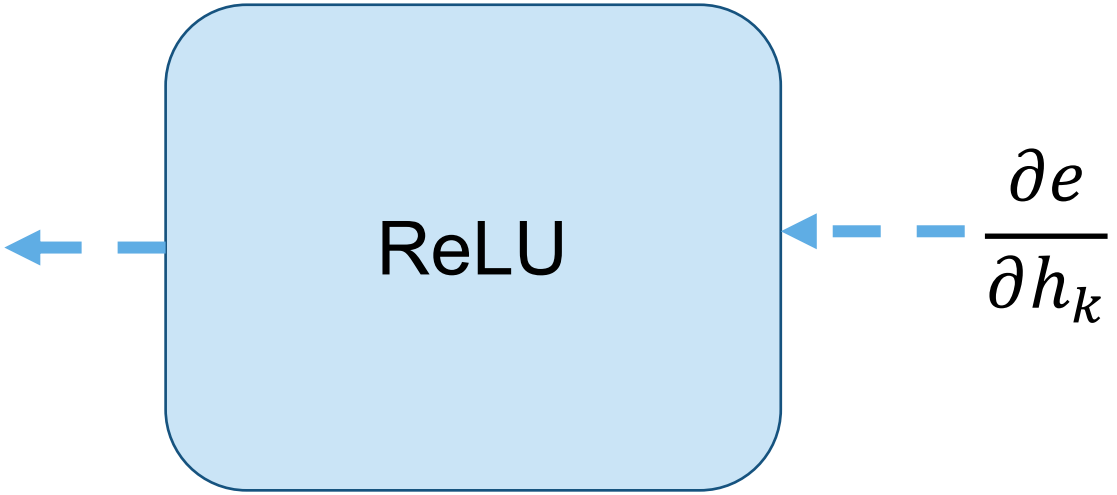
$$f(x) = \max(0, x)$$

Gradients of ReLU function

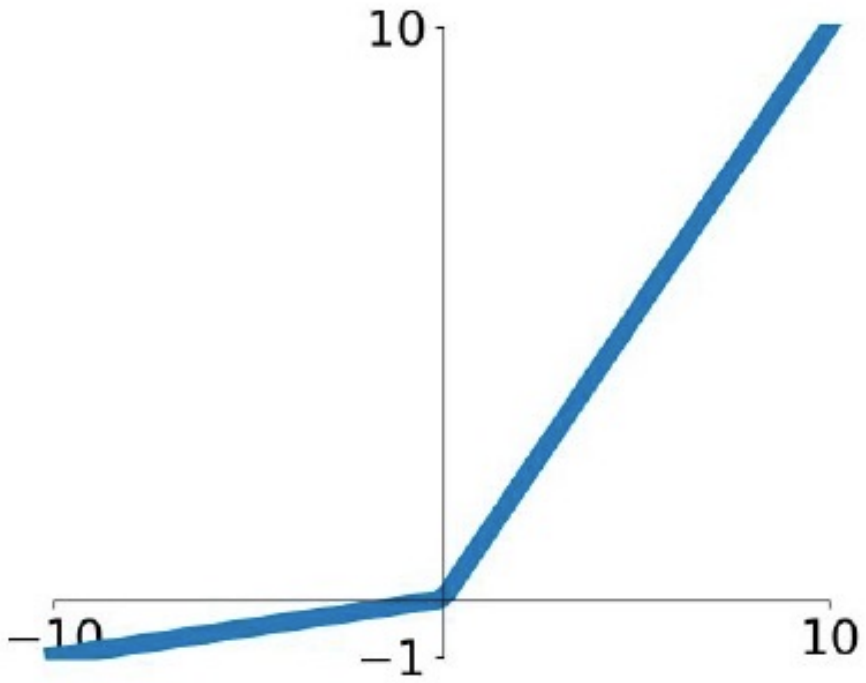


$$\frac{\partial e}{\partial h_{k-1}(i,j)} = \frac{\partial e}{\partial h_k(i,j)} \quad \text{, if } h_{k-1}(i,j) > 0$$

$$\frac{\partial e}{\partial h_{k-1}(i,j)} = 0, \qquad \text{if } h_{k-1}(i,j) \leq 0$$



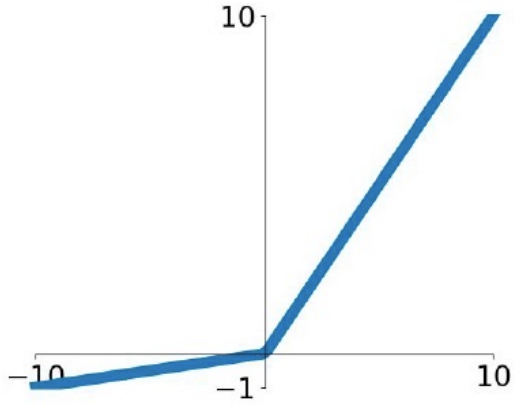
Leaky ReLU function



$$f(x) = \max(0.1x, x)$$

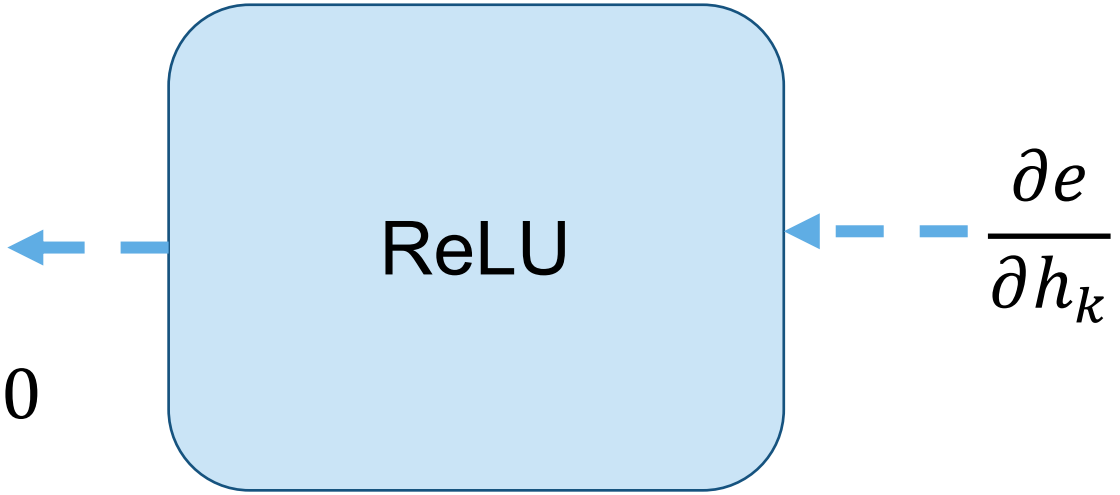
Gradients of Leaky ReLU function

$$f(x) = \max(0.1x, x)$$



$$\frac{\partial e}{\partial h_{k-1}(i,j)} = \frac{\partial e}{\partial h_k(i,j)} \quad , \text{if } h_{k-1}(i,j) > 0$$

$$\frac{\partial e}{\partial h_{k-1}(i,j)} = 0.1\frac{\partial e}{\partial h_k(i,j)} , \text{if } h_{k-1}(i,j) \leq 0$$



Gradients from SoftMax Loss

Gradients from SoftMax Loss

$$L = -\log P_W(y_i|x_i) = -\sum_j y_j \log p_j$$

y is a one-hot vector ([0,0,1,….,0])

$$p_i = \frac{\exp x_i}{\sum \exp x_j}$$

$$\frac{\partial L}{\partial x_i} = -y_i(1 - p_i) + \sum_{j \neq i} y_j p_i = \sum_j y_j p_i - y_i = p_i - y_i$$

This Class

- Optimization methods
- Learning rate
- Recall activation function
- Gradients from SoftMax Loss

Next Class

More elements in training Convolutional Neural Networks

Appendix: Gradients from SoftMax Loss

$$L = -\log P_W(y_i|x_i) = -\sum_j y_j \log p_j$$

y is a one-hot vector ([0,0,1,….,0])

$$p_i = \frac{\exp x_i}{\sum \exp x_j}$$

$$\frac{\partial L}{\partial x_i} = -\sum_j y_j \frac{\partial \log p_j}{\partial x_i} = -\sum_j y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i}$$

$$= -y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} - \sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i}$$

$$\frac{\partial L}{\partial x_i} = -y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} - \sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i}$$

$$p_i = \frac{\exp x_i}{\sum \exp x_j}$$

$$h(x) = \frac{f(x)}{g(x)}. \quad \frac{\partial h(x)}{\partial x} = \frac{f'(x) \times g(x) - g'(x) \times f(x)}{(g(x))^2}$$

$$\frac{\partial p_i}{\partial x_i} = \frac{\exp x_i \sum \exp x_j - \exp x_i \exp x_i}{\left(\sum \exp x_j\right)^2}$$

$$= \frac{\exp x_i \left(\sum \exp x_j - \exp x_i\right)}{\left(\sum \exp x_j\right)^2} = \frac{\exp x_i}{\sum \exp x_j}\frac{\sum \exp x_j - \exp x_i}{\sum \exp x_j}$$

$$= \frac{\exp x_i}{\sum \exp x_j}\left(\frac{\sum \exp x_j}{\sum \exp x_j} - \frac{\exp x_i}{\sum \exp x_j}\right) = p_i(1 - p_i)$$

$$-y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} = -y_i \frac{1}{p_i} p_i(1 - p_i) = -y_i(1 - p_i)$$

$$\frac{\partial L}{\partial x_i} = -y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} - \sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i}$$

$$p_j = \frac{\exp x_j}{\sum \exp x_j}$$

$$\left(\frac{1}{x}\right)' = \frac{1}{x^2}$$

$$\frac{\partial p_j}{\partial x_i} = -\frac{\exp x_i}{\left(\sum \exp x_j\right)^2} \exp x_j = -\frac{\exp x_i}{\sum \exp x_j} \frac{\exp x_j}{\sum \exp x_j}$$

$$= -p_i p_j$$

$$\sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i} = \sum_{j \neq i} y_j \frac{1}{p_j}(-p_i p_j) = -\sum_{j \neq i} y_j p_i$$

$$\frac{\partial L}{\partial x_i} = -y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} - \sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i}$$

$$-y_i \frac{1}{p_i}\frac{\partial p_i}{\partial x_i} = -y_i \frac{1}{p_i} p_i(1 - p_i) = -y_i(1 - p_i)$$

$$\sum_{j \neq i} y_j \frac{1}{p_j}\frac{\partial p_j}{\partial x_i} = \sum_{j \neq i} y_j \frac{1}{p_j}(-p_i p_j) = -\sum_{j \neq i} y_j p_i$$

$$\frac{\partial L}{\partial x_i} = -y_i(1 - p_i) + \sum_{j \neq i} y_j p_i = \sum_{j} y_j p_i - y_i = p_i - y_i$$