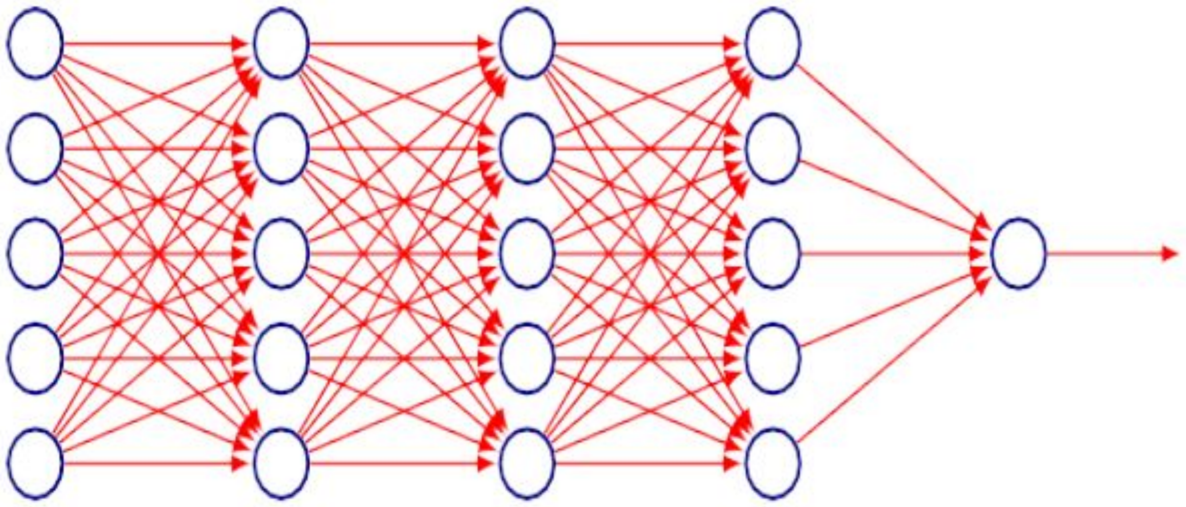
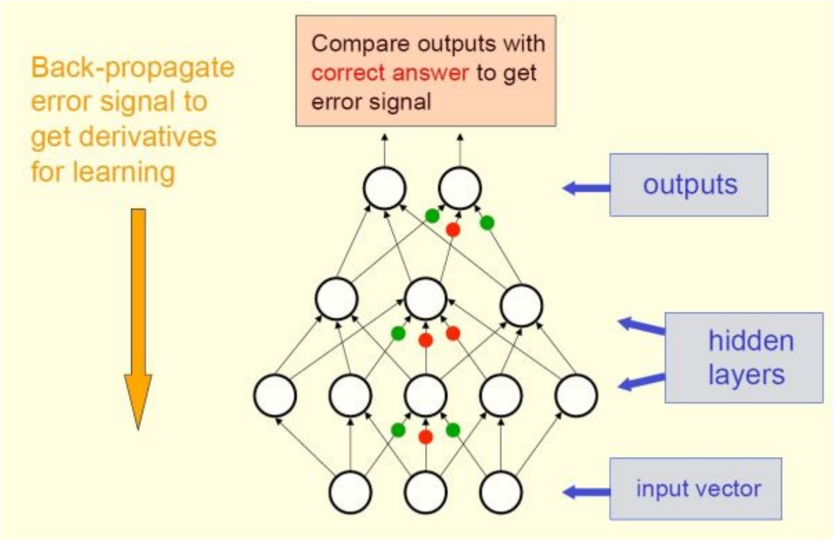
Deep Learning Basics

Computing Lab-II, Spring 2021

Training a multi-layer perceptron (MLP)



Backpropagation



- **Feedforward Propagation**: Accept input x, pass through intermediate stages and obtain output $\hat{y}$
- **During Training**: Use $\hat{y}$ to compute a scalar cost $J(\theta)$
- Backpropagation allows information to flow backwards from cost to compute the gradient

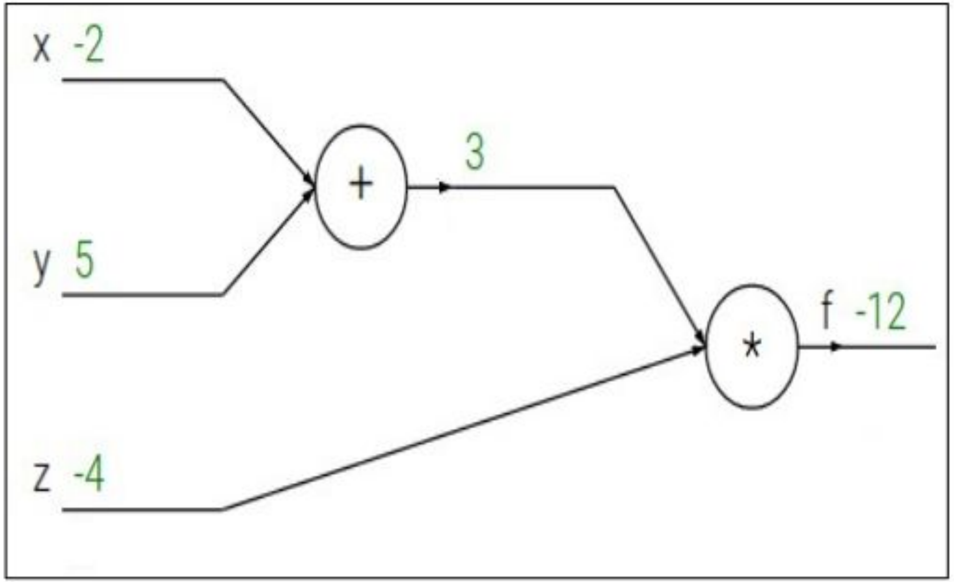
Backpropagation

- From the training data we don’t know what the hidden units should do
- But, we can compute how fast the error changes as we change a hidden activity
- Use error derivatives w.r.t hidden activities
- Each hidden unit can affect many output units and have separate effects on error – combine these effects
- Can compute error derivatives for hidden units efficiently (and once we have error derivatives for hidden activities, easy to get error derivatives for weights going in)

Differentiating a Computation Graph

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Chain Rule

We can write

$$\mathrm{f}(x, y, z) = g(h(x, y), z)$$

Where $h(x, y) = x + y$, and $g(a, b) = a * b$

By the chain rule, $\frac{df}{dx} = \frac{dg}{dh}\frac{dh}{dx}$ and $\frac{df}{dy} = \frac{dg}{dh}\frac{dh}{dy}$

Differentiating a Computation Graph

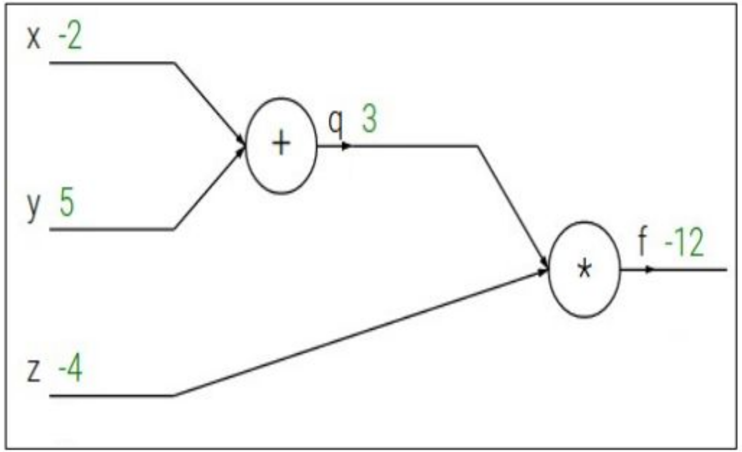
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



Differentiating a Computation Graph

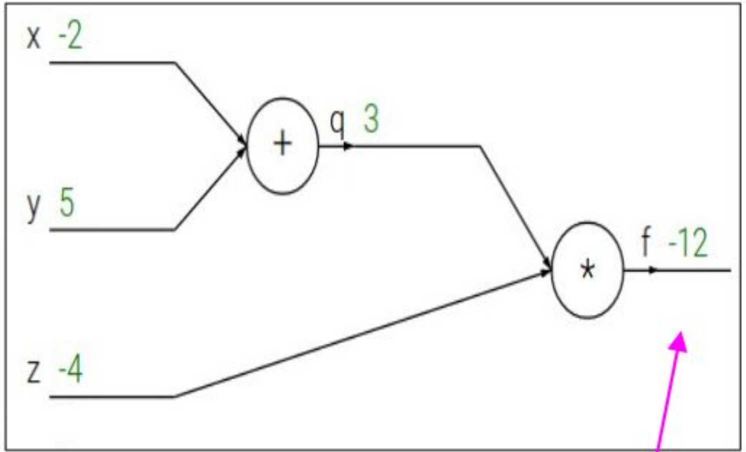
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

Differentiating a Computation Graph

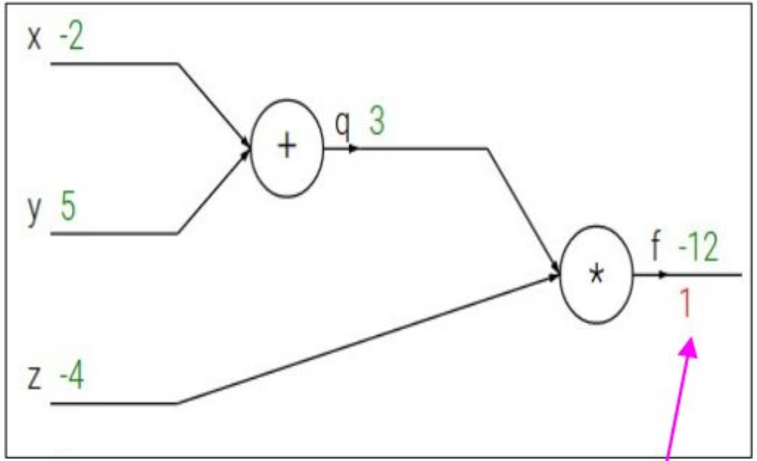
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial f}}$$

Differentiating a Computation Graph

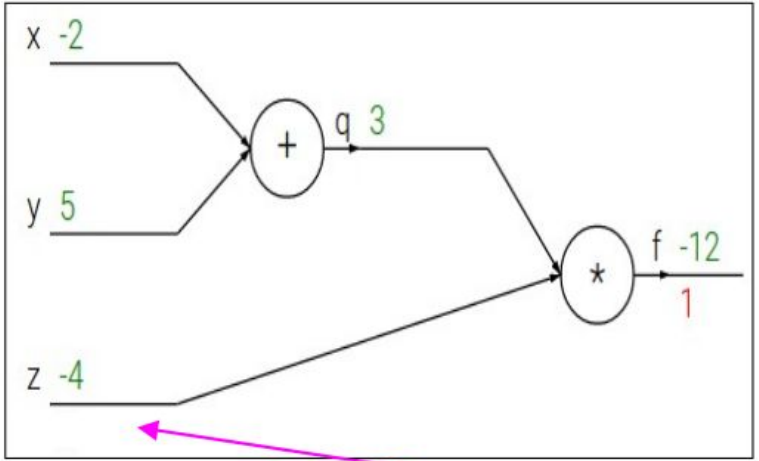
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

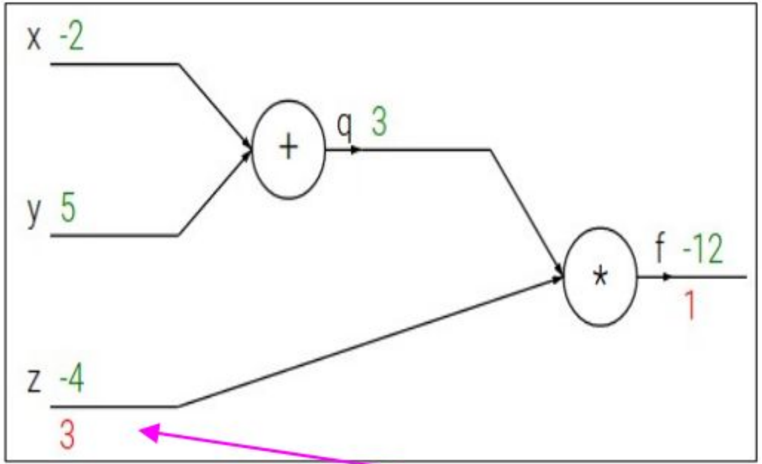
Differentiating a Computation Graph

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial z}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Differentiating a Computation Graph

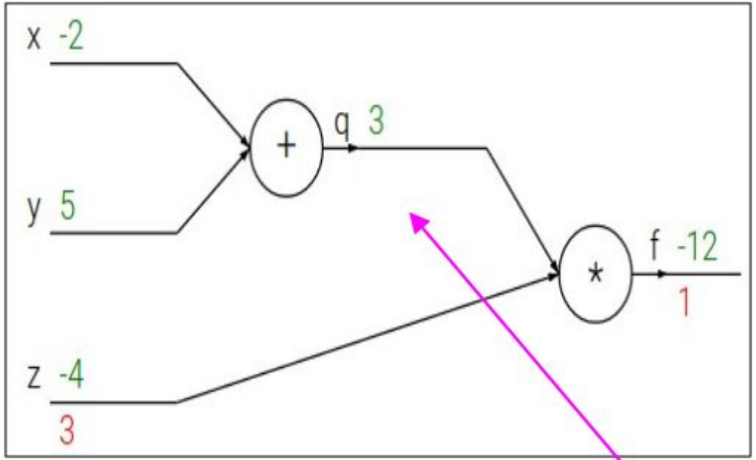
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

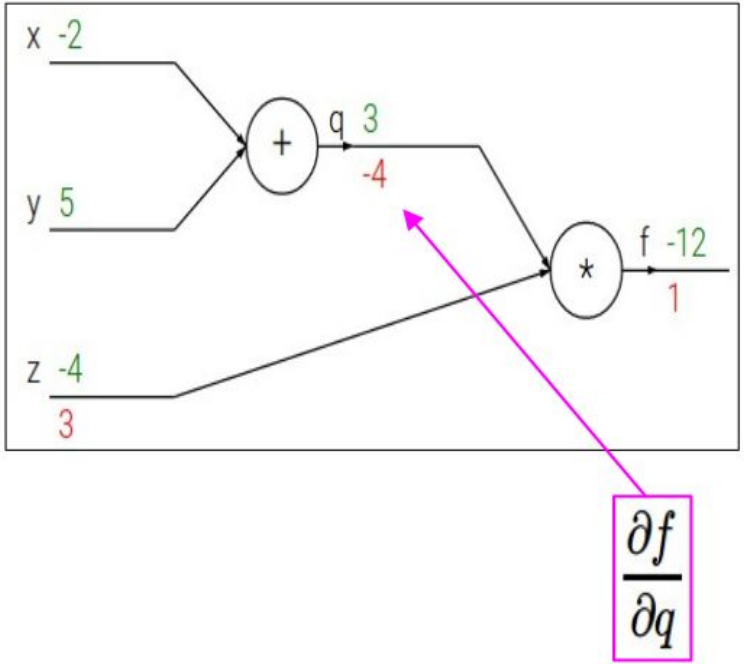
Differentiating a Computation Graph

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

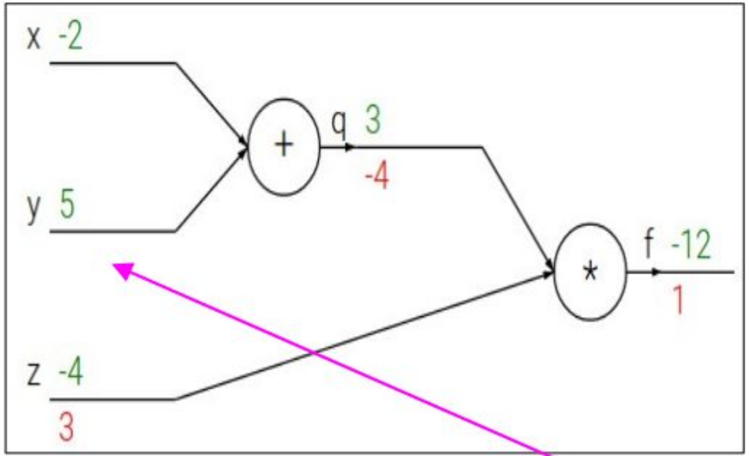


Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Differentiating a Computation Graph

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial y}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Differentiating a Computation Graph

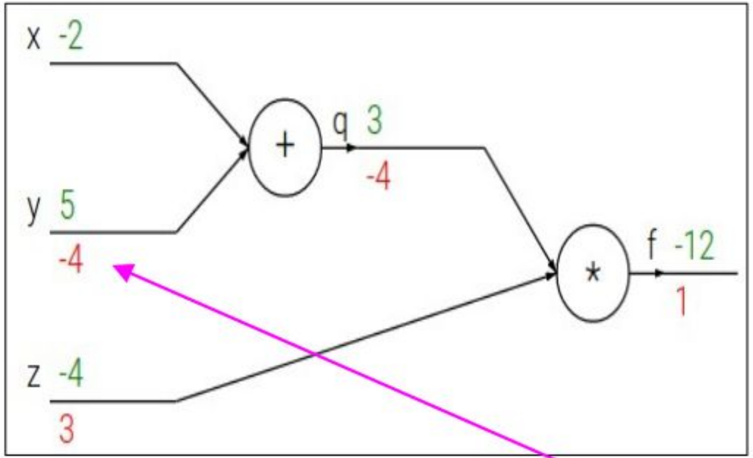
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

$$\frac{\partial f}{\partial y}$$

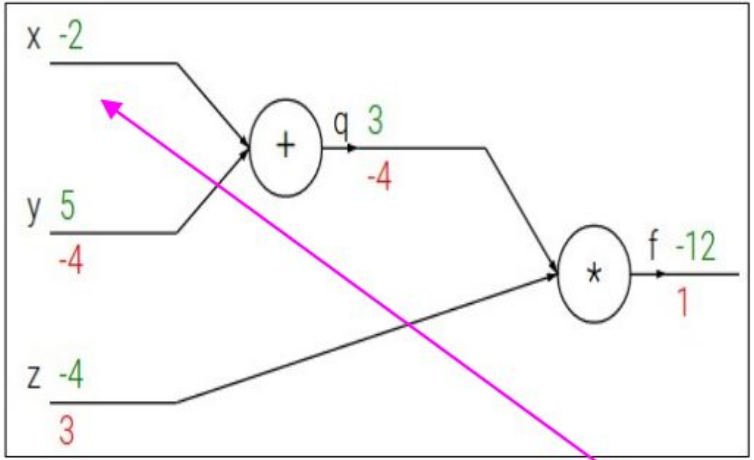
Differentiating a Computation Graph

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial x}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Differentiating a Computation Graph

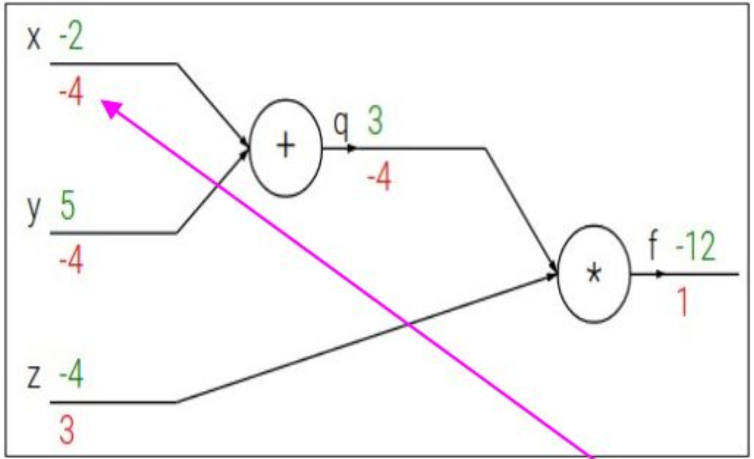
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Batch Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

while stopping criteria not met **do**

Compute gradient estimate over N examples:

$$\hat{g} \leftarrow + \frac{1}{N} \nabla_\theta \sum_i L\left(f(x^{(i)}; \theta), y^{(i)}\right)$$

Apply Update: $\theta = \theta - \epsilon \hat{g}$

end while

Positive: Gradient estimates are stable

Negative: Need to compute gradients over the entire training for one update

Batch Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter $\boxed{\theta}$ $\rightarrow (x, y, z)$ in example computation graph

1: **while** stopping criteria not met **do**

2: Compute gradient estimate over N examples:

$$\hat{g} \leftarrow + \frac{1}{N} \boxed{\nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})} \rightarrow \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

in example computation graph

4: Apply Update: $\theta = \theta - \epsilon \hat{g}$

5: **end while**

Positive: Gradient estimates are stable

Negative: Need to compute gradients over the entire training for one update

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1. while stopping criteria not met do
2. Sample example $(x^{(i)}, y^{(i)})$ from training set
3. Compute gradient estimate:
4. $\hat{g} \leftarrow +\nabla_\theta \quad L\left(f(x^{(i)};\theta), y^{(i)}\right)$
5. Apply Update: $\theta = \theta - \epsilon\hat{g}$
6. end while

ϵ_k is learning rate at step k

Sufficient condition to guarantee convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and} \sum_{k=1}^{\infty} {\epsilon_k}^2 = \infty$$

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter $\boxed{\theta}$ → (x, y, z) in example computation graph

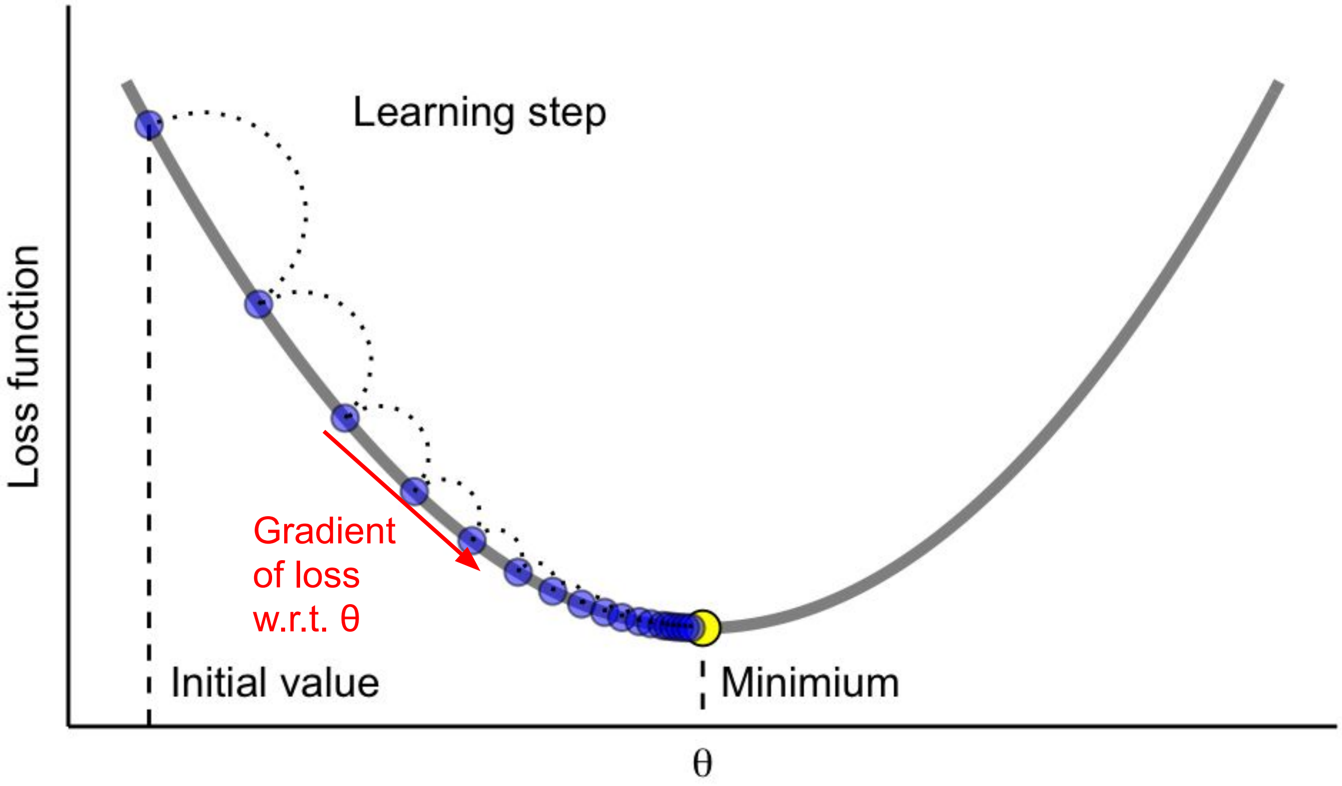
1. while stopping criteria not met do
2. Sample example $(x^{(i)}, y^{(i)})$ from training set
3. Compute gradient estimate:
4. $\hat{g} \leftarrow + \boxed{\nabla_\theta \quad L(f(x^{(i)};\theta), y^{(i)})}$ → $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$ in example computation graph
5. Apply Update: $\theta = \theta - \epsilon \hat{g}$
6. end while

ϵ_k is learning rate at step k

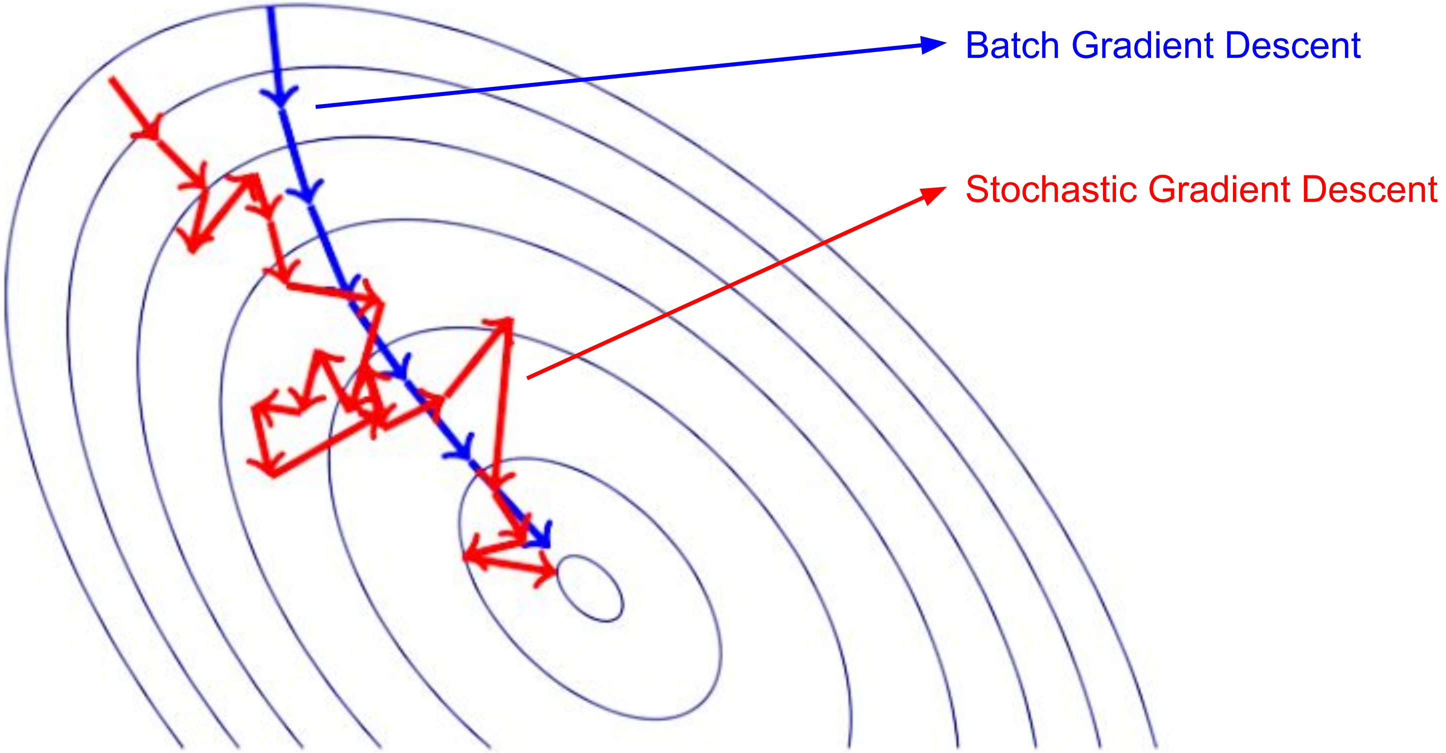
Sufficient condition to guarantee convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and} \sum_{k=1}^{\infty} {\epsilon_k}^2 = \infty$$

Loss function decreases after each iteration



Stochastic Gradient Descent: Visualization



Learning Rate Schedule

- In practice the learning rate is decayed linearly till iteration τ

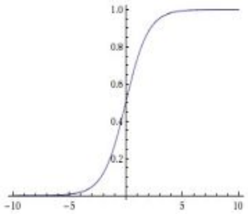
$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \frac{k}{\tau}$$

- τ is usually set to the number of iterations needed for a large number of passes through the data
- ϵ_τ should roughly be set to 1% of ϵ_0
- How to set ϵ_0 ?

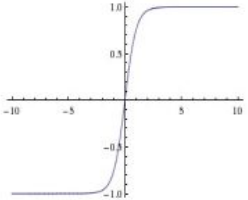
Activation Functions

Sigmoid

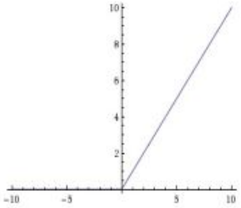
$$\sigma(x) = 1/(1+e^{-x})$$



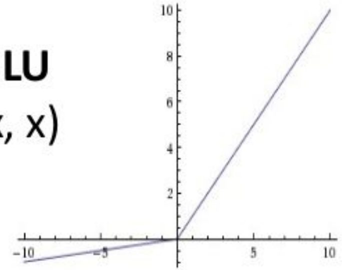
tanh tanh(x)



ReLU max(0,x)

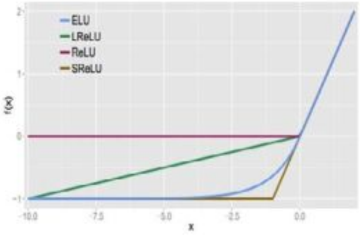


Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Regularization

- To prevent overfitting or help the optimization
- “Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

L2 parameter regularization

- also known as ridge regression or Tikhonov regularization
- For every w add $\frac{1}{2}\lambda w^2$ to the objective function.
- Gradient of this term: λw
- Weight decay: Encourages small weights

Weight Decay as Constrained Optimization

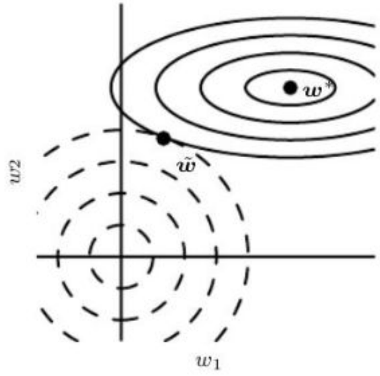


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to zero. In the second dimension, the objective function is very sensitive to movements away from $\boldsymbol{w}^*$. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

L1 regularization

- For every w add $\lambda|w|$ to the objective function.
- it leads the weight vectors to become sparse during optimization

Hyperparameters to play with:

- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

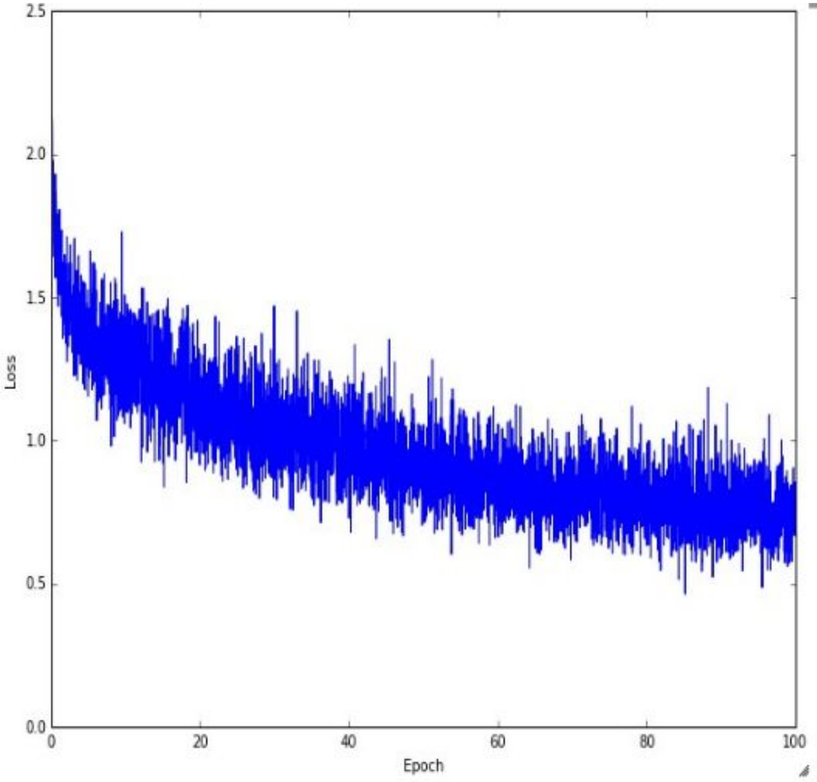
neural networks practitioner
music = loss function

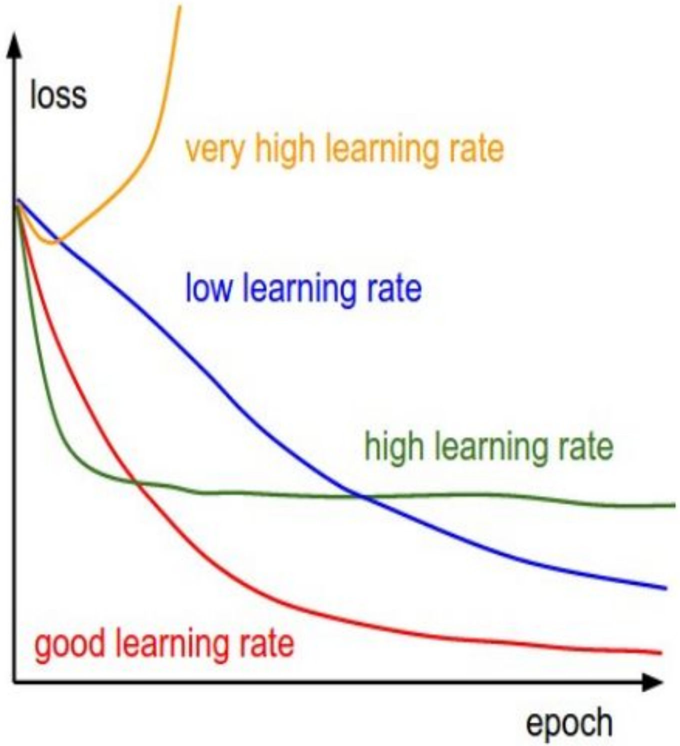
Training

1. Check that loss is reasonable.

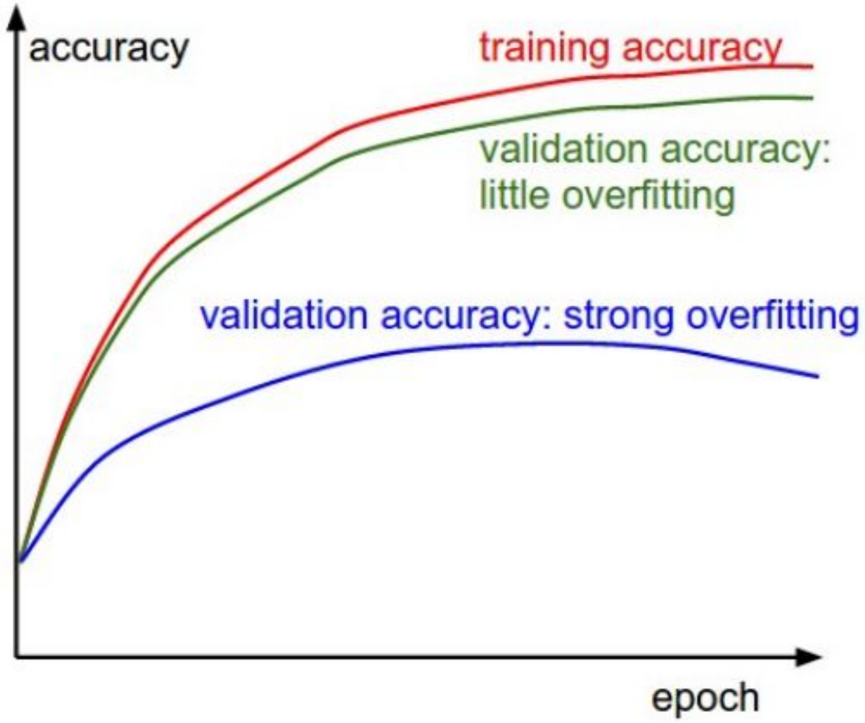
- Loss goes up as you increase regularization.
- Make sure that you can overfit very small portion of the training data
- Start with small regularization and find learning rate that makes the loss go down.
  - loss not going down: means learning rate too low
  - loss exploding: learning rate too high

Monitor and visualize the loss curve

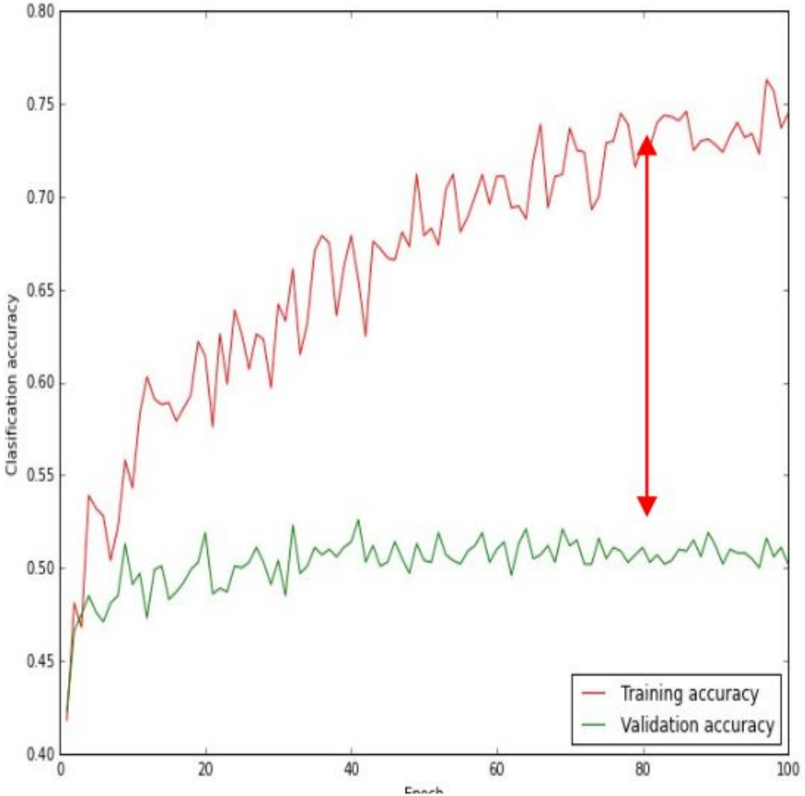




Train/Val accuracy



Monitor and visualize the accuracy:



big gap = overfitting

=> increase regularization strength?

no gap

=> increase model capacity?