CSCE 633: Machine Learning

Lecture 28: Neural Networks: Backpropagation and Some Applications

Texas A&M University

Backpropagation

Multilayer Perceptron: Representation

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output:
  $y \in \{0, 1\}$ or $y \in \{1, \ldots, K\}$ (classification)
  $y \in \mathbb{R}$ or $y \in \mathbb{R}^K$ (regression)
- Training data: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Model: $h_{\mathbf{W},\mathbf{b}}(\mathbf{x})$
  represented through forward propagation (see previous slides)
- Model parameters: weights $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}$ and biases $\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}$

Multilayer Perceptron: Evaluation criterion

$$J(\mathbf{W}, \mathbf{b}, \mathcal{D}^{train}) = \frac{1}{2}\|h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) - y\|_2^2 \text{ (regression)}$$

$$J(\mathbf{W}, \mathbf{b}, \mathcal{D}^{train}) = y \log h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) + (1 - y) \log(1 - h_{\mathbf{W},\mathbf{b}}(\mathbf{x})) \text{ (classification)}$$

Backpropagation

Multilayer Perceptron: Evaluation criterion

Regression

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{M}\frac{1}{2}\|h_{\mathbf{W},\mathbf{b}}(\mathbf{x_n}) - y_n\|_2^2 + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

Classification

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{M}(y_n \log h_{\mathbf{W},\mathbf{b}}(\mathbf{x_n}) + (1-y_n)\log(1-h_{\mathbf{W},\mathbf{b}}(\mathbf{x_n}))) + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

We will perform gradient descent

Backpropagation

Gradient descent for regression

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^{M} \frac{1}{2} \| h_{\mathbf{W},\mathbf{b}}(\mathbf{x_n}) - y_n \|_2^2 + \frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{(l)})^2$$

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}}$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}}$$

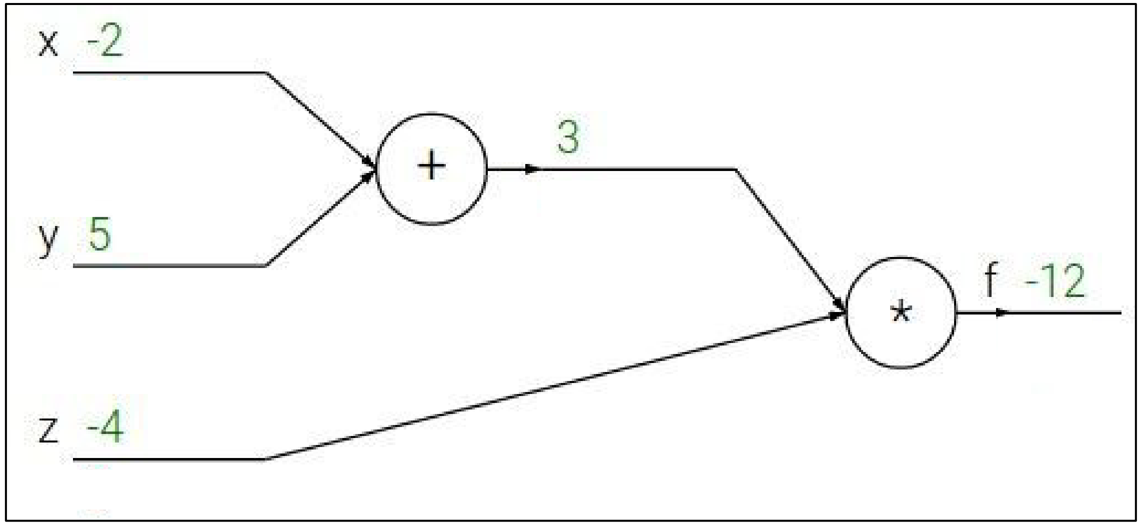
Note: Initialize the parameters randomly $\rightarrow$ symmetry breaking

Use backpropagation to compute partial derivatives $\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}}$ and $\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}}$

Backpropagation Example in Computational Graph[1]

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

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



[1] http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

Backpropagation Example in Computational Graph

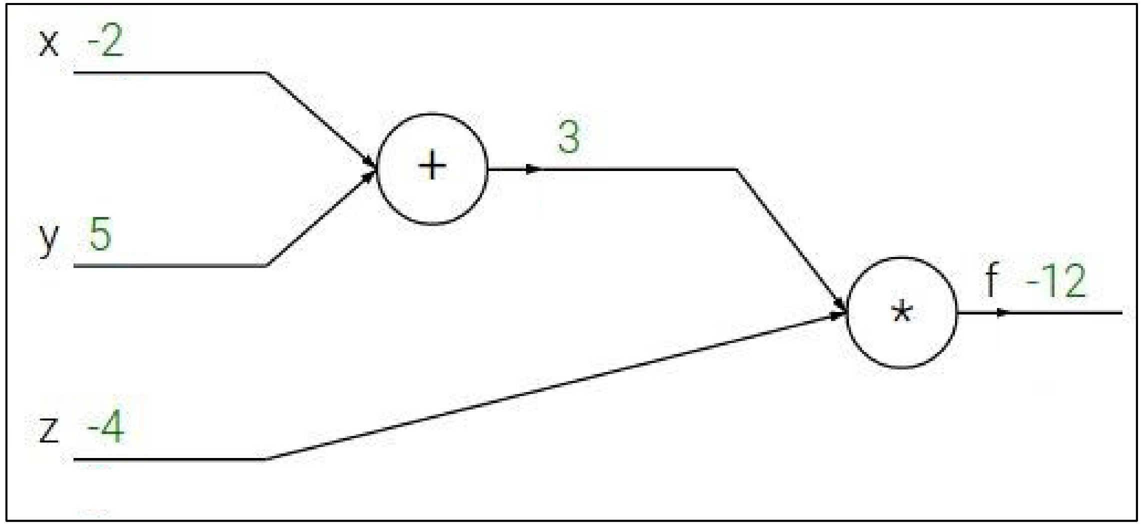
$$f(x, y, z) = (x + y)z$$

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



Backpropagation Example in Computational Graph

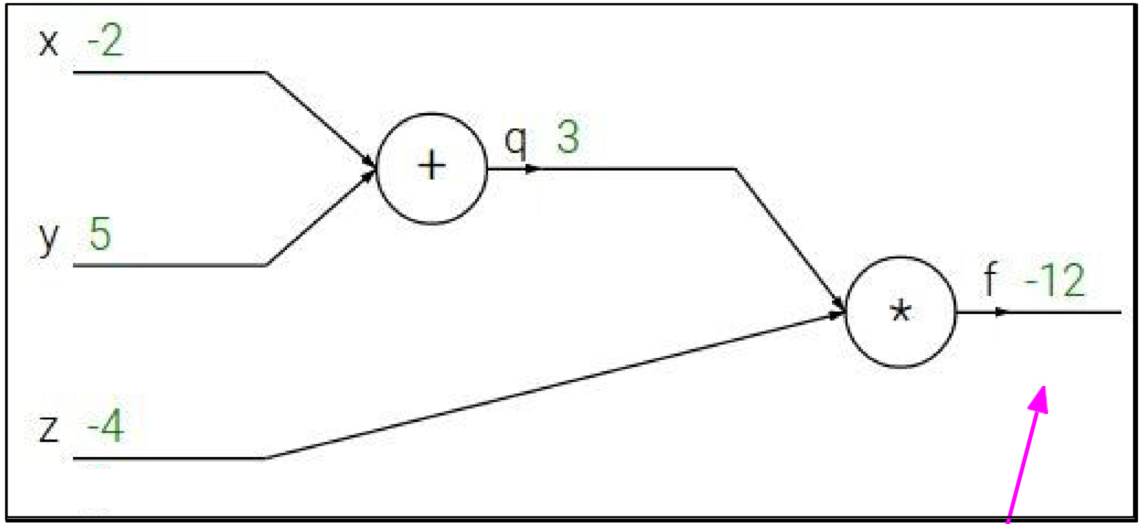
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation Example in Computational Graph

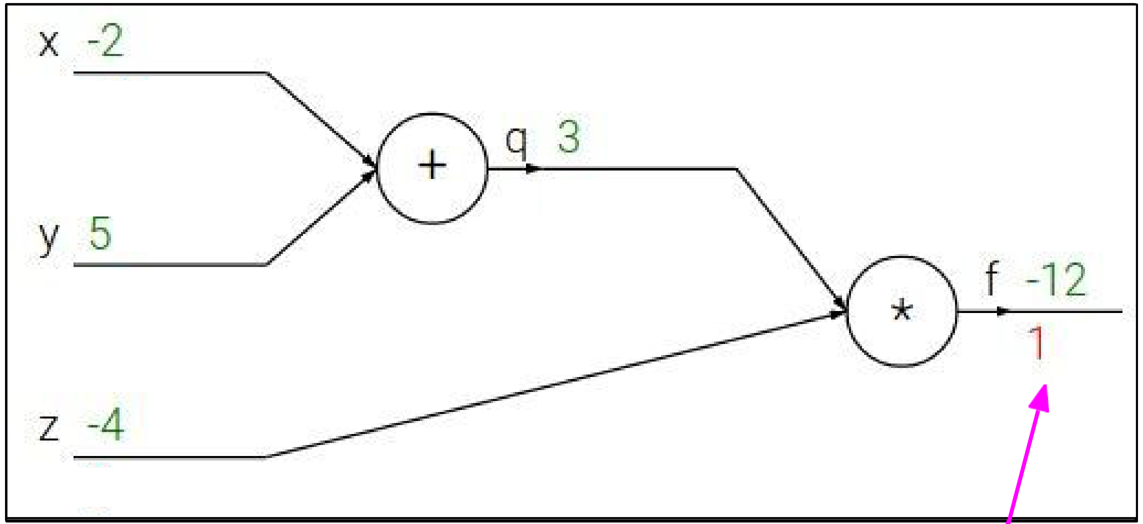
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation Example in Computational Graph

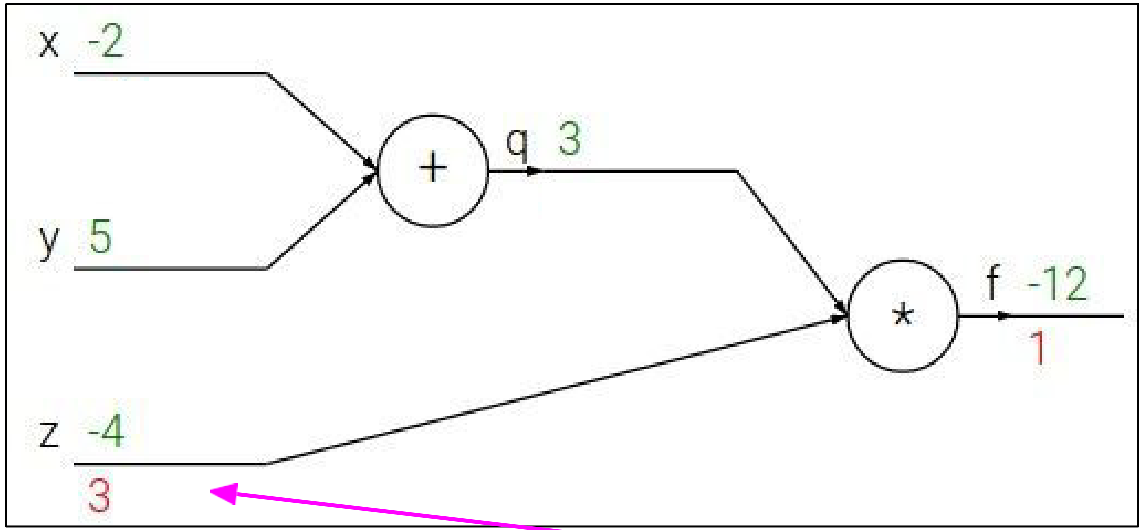
$$f(x,y,z) = (x+y)z$$

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



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

Backpropagation Example in Computational Graph

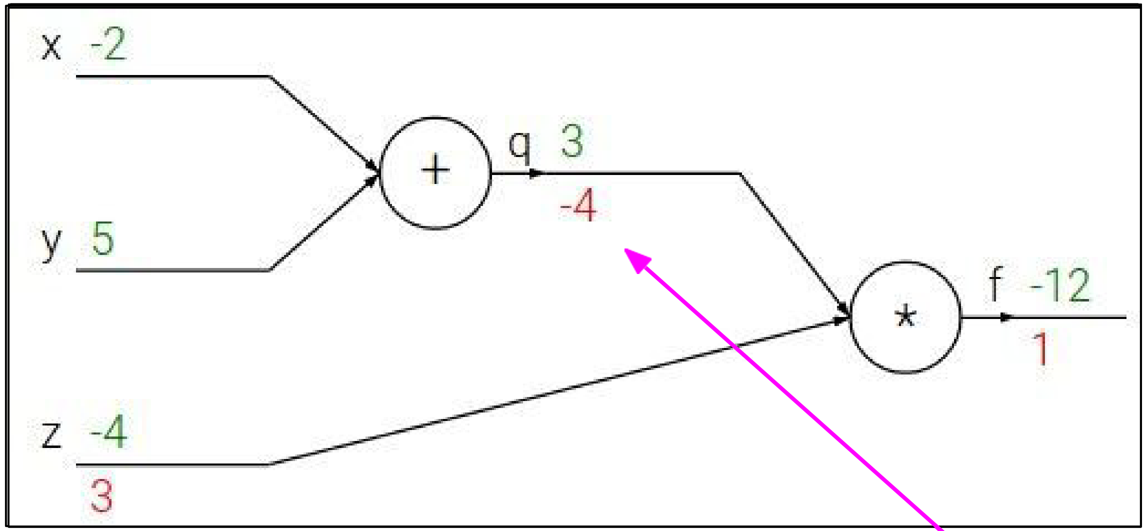
$$f(x, y, z) = (x + y)z$$

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

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

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

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



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

Backpropagation Example in Computational Graph

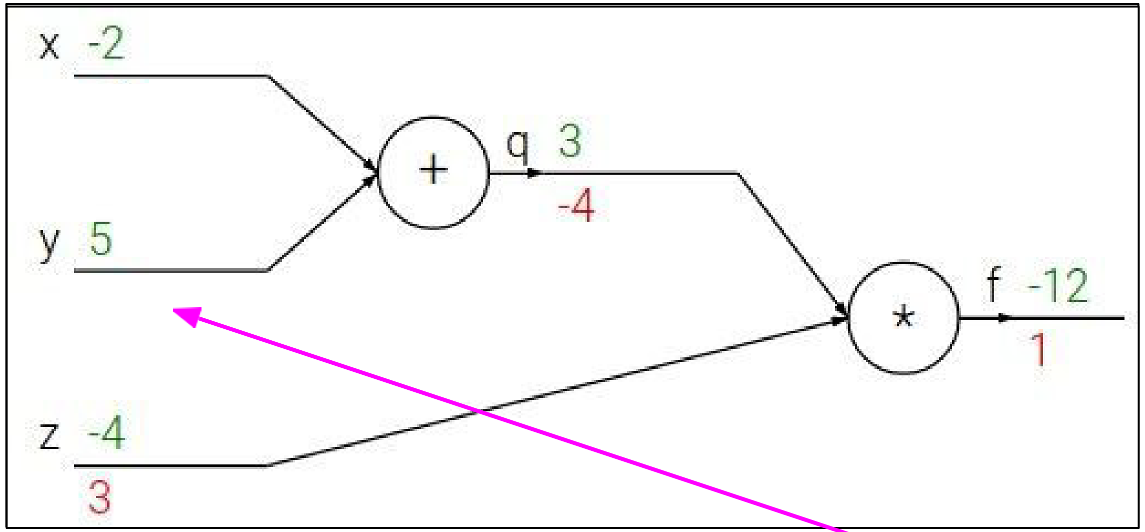
$$f(x,y,z) = (x+y)z$$

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



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

Backpropagation Example in Computational Graph

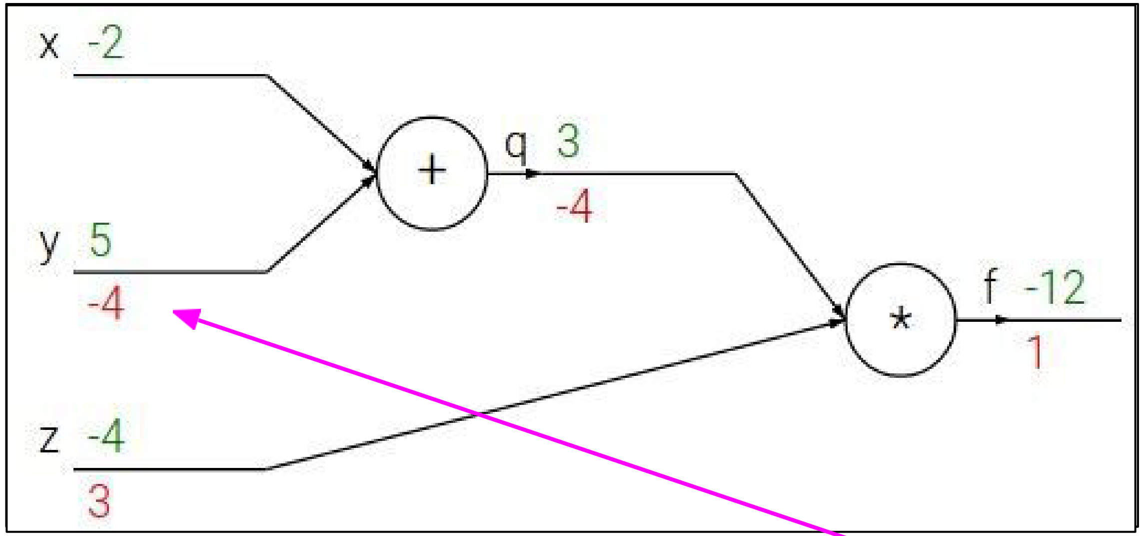
$$f(x, y, z) = (x + y)z$$

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



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

Chain rule:

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

Backpropagation Example in Computational Graph

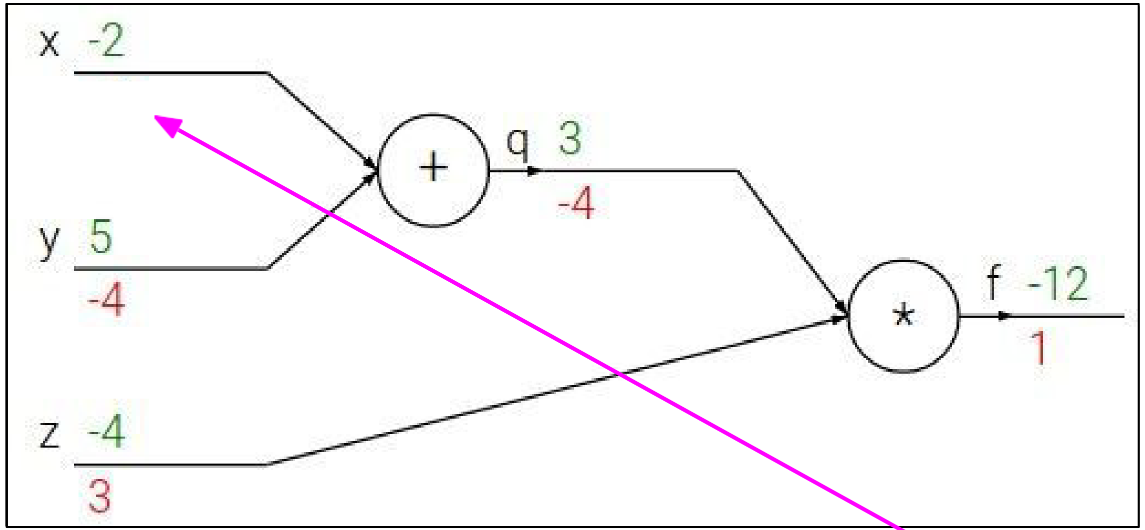
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation Example in Computational Graph

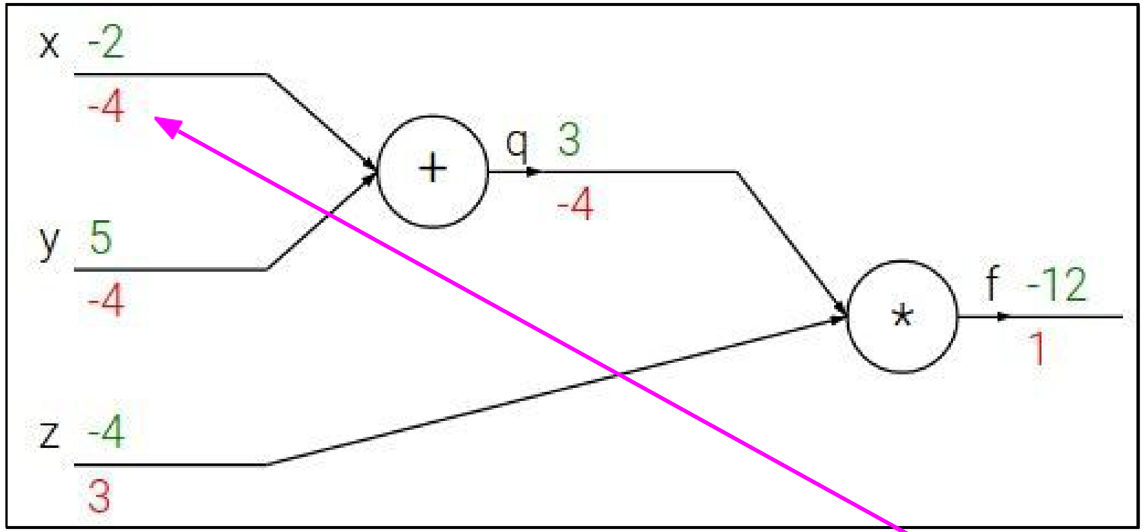
$$f(x, y, z) = (x + y)z$$

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

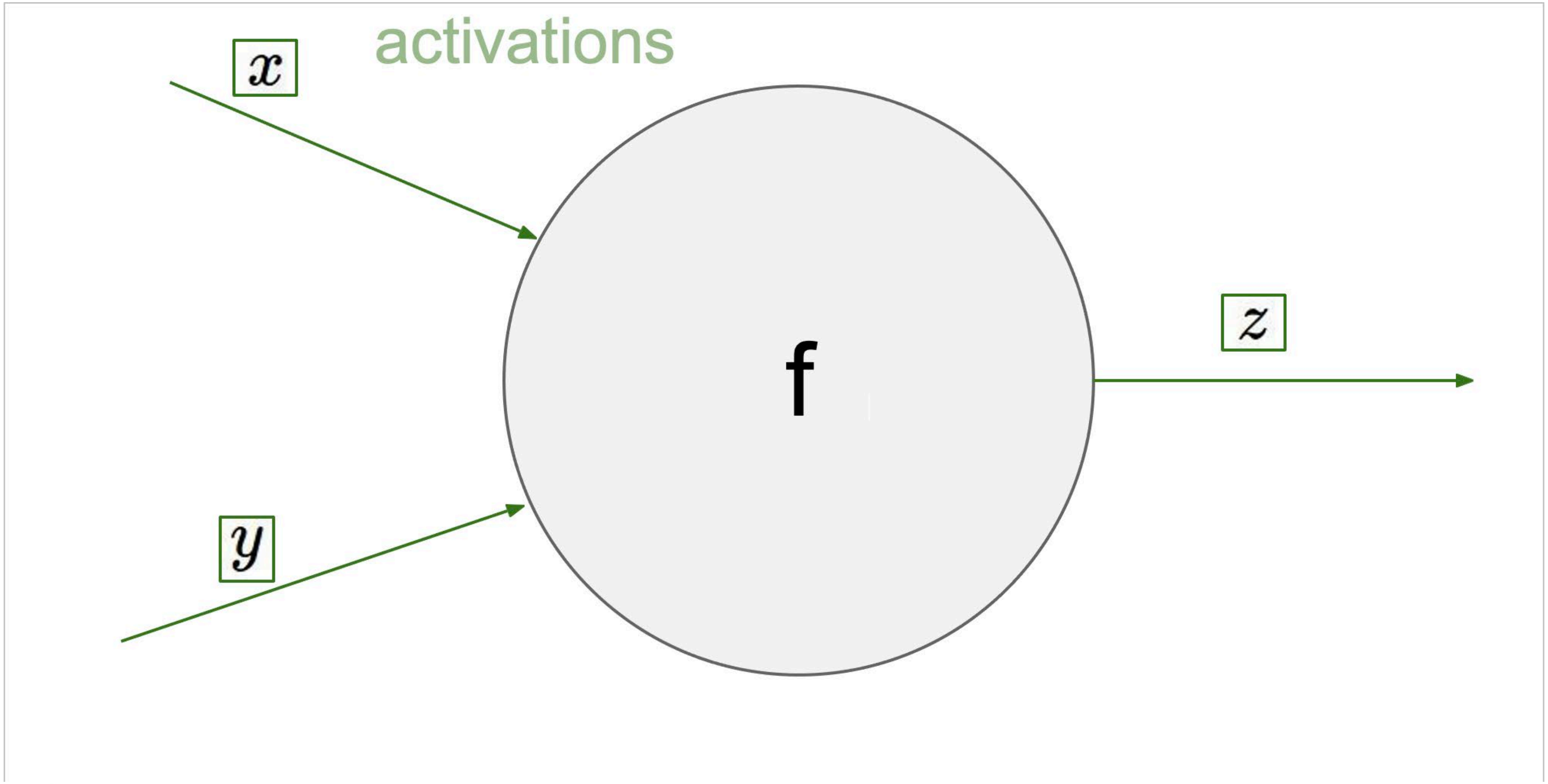


Chain rule:

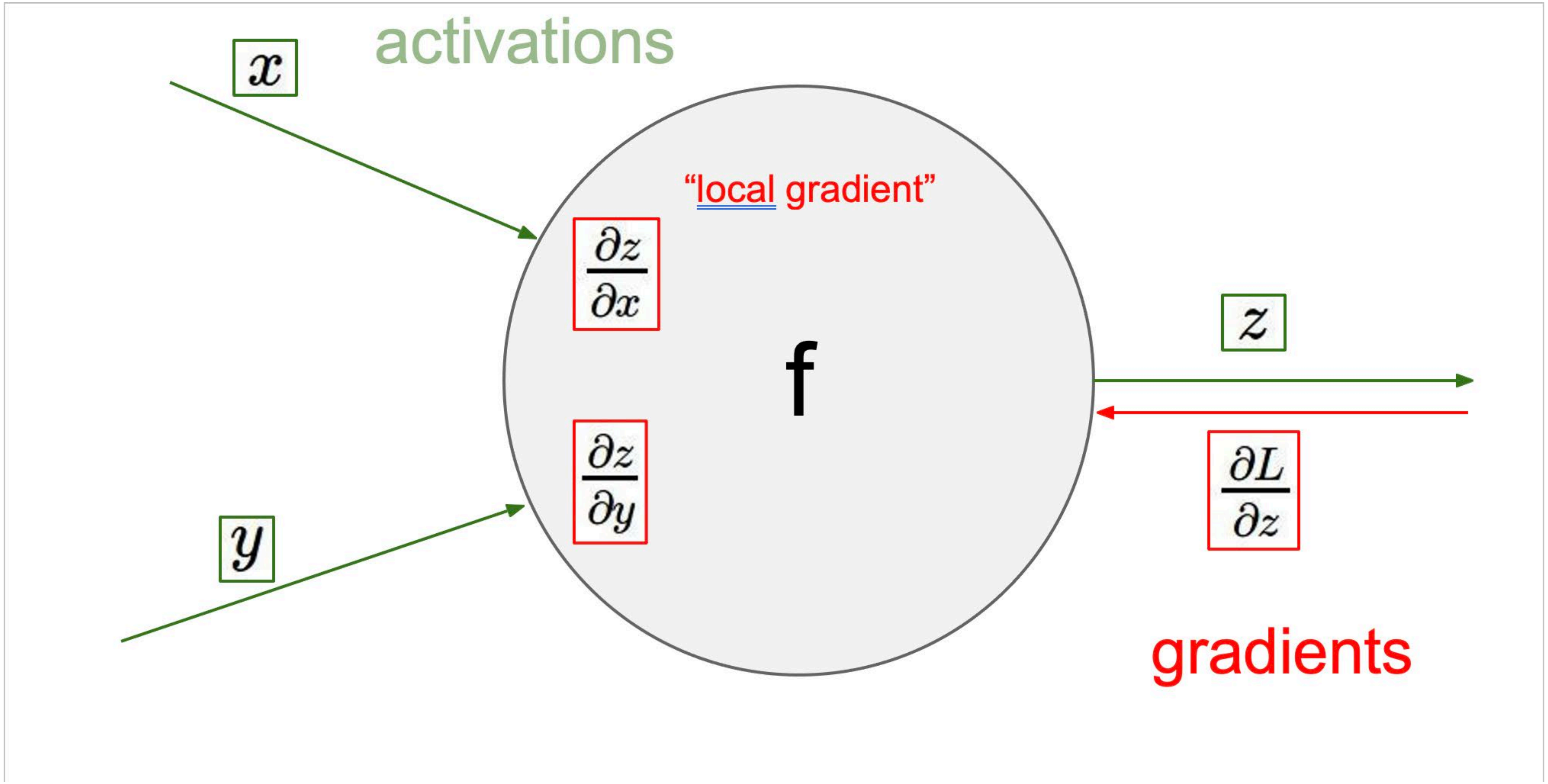
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

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

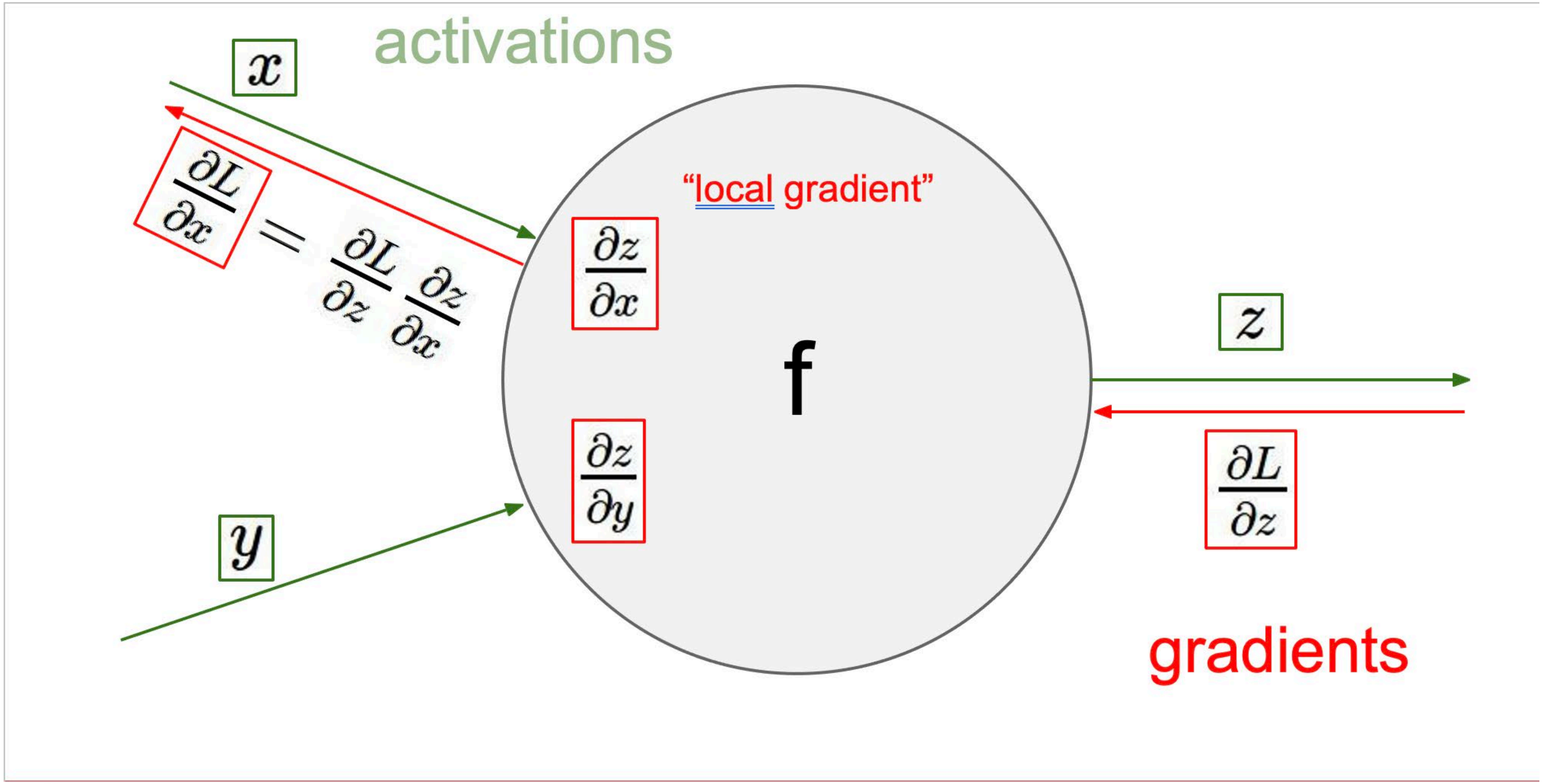
Modular Backpropagation



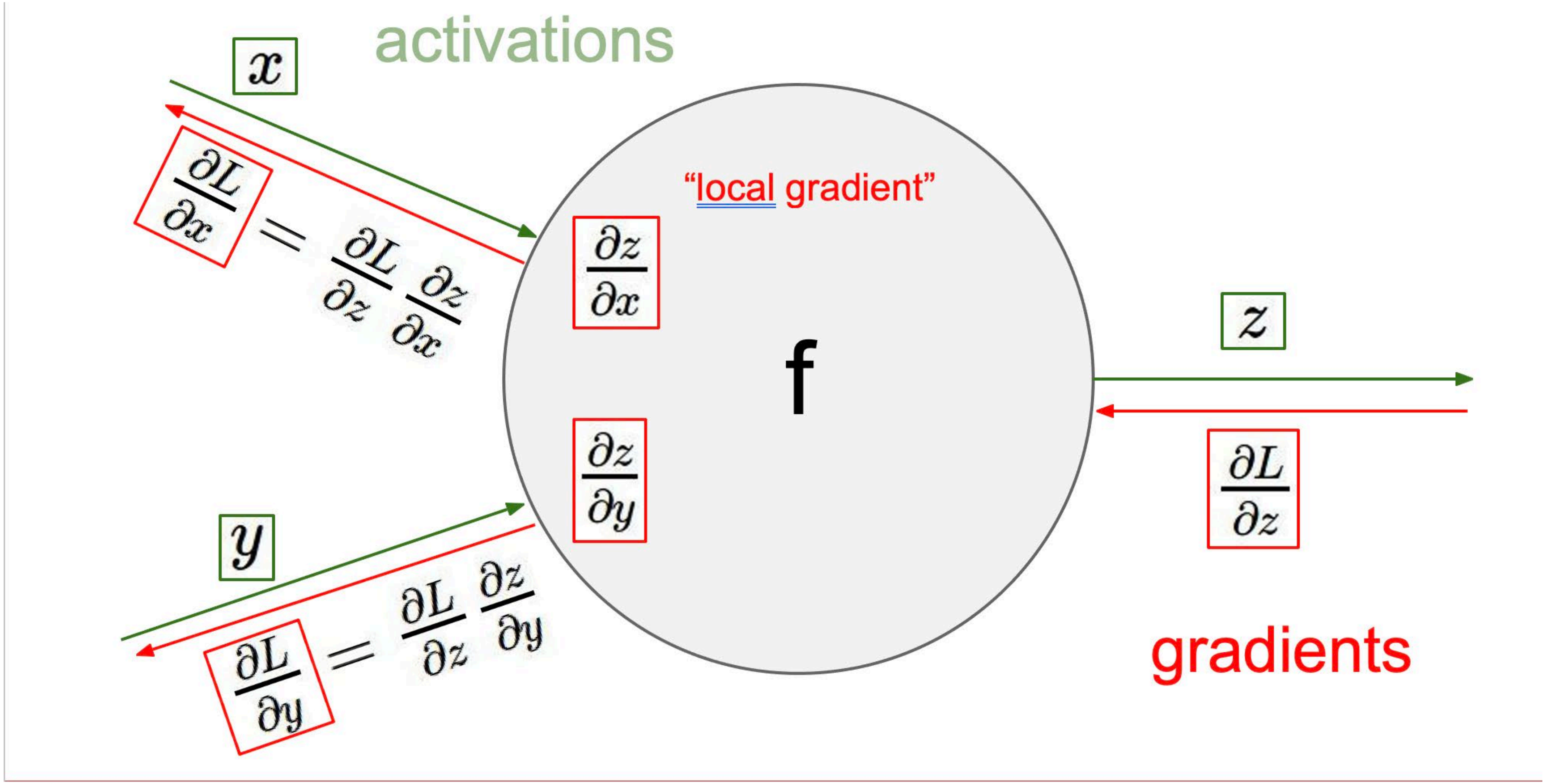
Modular Backpropagation



Modular Backpropagation



Modular Backpropagation



Backpropagation

Implementation

- For each node i in output layer L
  - $\delta_i^{(L)} = (\alpha_i^{(L)} - y_n)f'(z_i^{(L)})$
- For each node i in layer $l = L-1, L-2, \ldots, 2$
  - Hidden nodes: $\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$
- Compute the desired partial derivatives as:

$$\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}} = \alpha_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}} = \delta_i^{(l+1)}$$

- Update the weights as:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}}$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}}$$

Optimization

- Gradient Descent (GD)

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over N examples:
3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: **end while**

- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

Optimization

- Stochastic Gradient Descent (SGD)

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 $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate:
4: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
5: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
6: **end while**

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

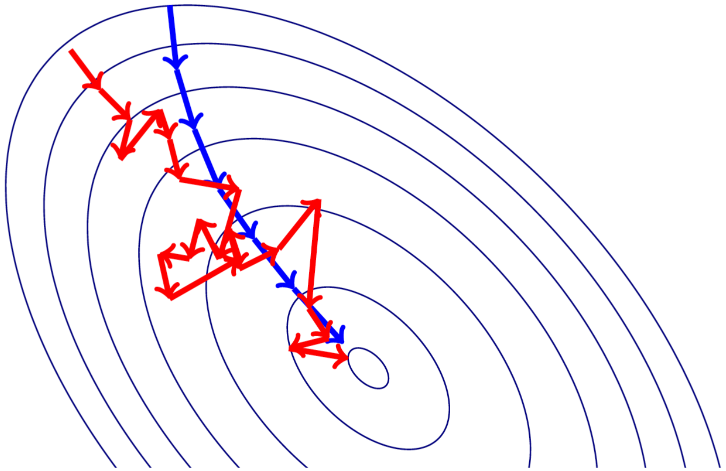
$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Optimization

- GD versus SGD

Batch Gradient Descent:

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

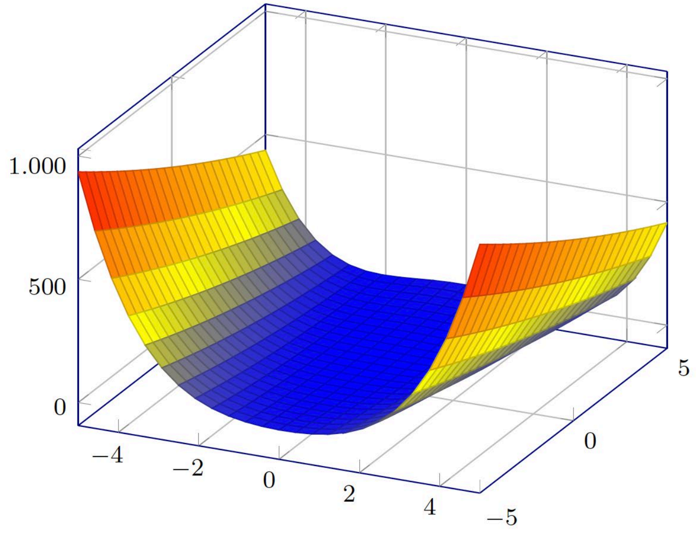


SGD:

$$\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$$
$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$

Optimization

- Momentum
  - The Momentum method is a method to accelerate learning using SGD.
  - In particular SGD suffers in the following scenarios:
    - Error surface has high curvature
    - Small but consistent gradients
    - Noisy gradients



- Gradient Descent would move quickly down the walls, but very slowly through the valley floor

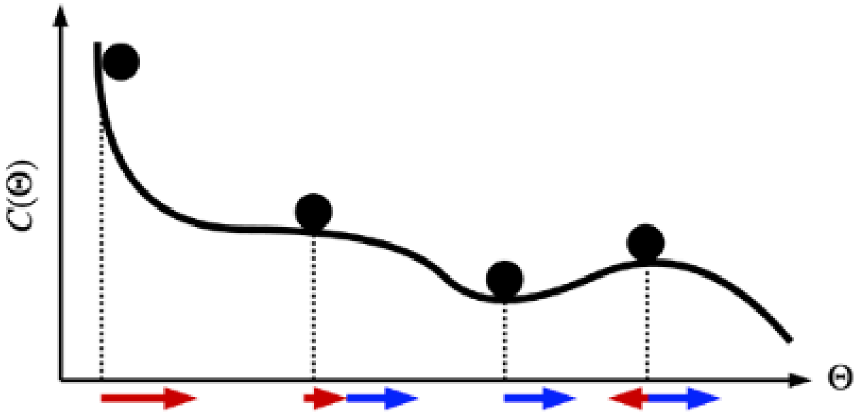
Optimization

Update rule in SGD:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta \boldsymbol{g}^{(t)}$$

where $\boldsymbol{g}^{(t)} = \nabla_\Theta C(\Theta^{(t)})$

- Gets stuck in local minima or saddle points

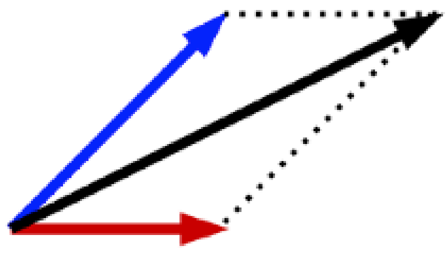


Momentum: make the same movement $\boldsymbol{v}^{(t)}$ in the last iteration, corrected by negative gradient:

$$\boldsymbol{v}^{(t+1)} \leftarrow \lambda \boldsymbol{v}^{(t)} - (1-\lambda)\boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \eta \boldsymbol{v}^{(t+1)}$$

$\boldsymbol{v}^{(t)}$ is a moving average of $-\boldsymbol{g}^{(t)}$



Optimization

- Popular Solver Examples: AdGrad, RMSProp, Adam

$$\text{SGD: } \theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$$

$$\text{Momentum: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

$$\text{Nesterov: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right) \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

$$\text{AdaGrad: } \mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \text{ then } \Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{RMSProp: } \mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}} \text{ then } \Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{Adam: } \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t} \text{ then } \Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

Optimization

- AdaGrad
  - **Idea**: Downscale a model parameter by square-root of sum of squares of all its historical values
  - Parameters that have large partial derivative of the loss -> learning rates for them are rapidly declined
  - Some interesting theoretical properties

Algorithm 4 AdaGrad

Require: Global Learning rate ϵ, Initial Parameter θ, δ

Initialize $\mathbf{r} = 0$

```
while stopping criteria not met do
    Sample example (x^(i), y^(i)) from training set
    Compute gradient estimate: ĝ ← +∇_θ L(f(x^(i); θ), y^(i))
    Accumulate: r ← r + ĝ ⊙ ĝ
    Compute update: Δθ ← −(ε / (δ + √r)) ⊙ ĝ
    Apply Update: θ ← θ + Δθ
end while
```

$$\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$$

$$\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$$

$$\theta \leftarrow \theta + \Delta\theta$$

Optimization

- RMSProp
  - AdaGrad might shrink the learning rate too aggressively, we can adapt it to perform better by accumulating an exponentially decaying average of the gradient

Algorithm 5 RMSProp

Require: Global Learning rate ϵ, decay parameter ρ, δ

Initialize $\mathbf{r} = 0$

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
4: Accumulate: $\mathbf{r} \leftarrow \rho\mathbf{r} + (1 - \rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
5: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
6: Apply Update: $\theta \leftarrow \theta + \Delta\theta$
7: **end while**

Optimization

- Adam
  - Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

Algorithm 7 RMSProp with Nesterov

Require: ϵ (set to 0.0001), decay rates ρ_1 (set to 0.9), ρ_2 (set to 0.9), θ, δ

Initialize moments variables $\mathbf{s} = 0$ and $\mathbf{r} = 0$, time step t = 0

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
4: $t \leftarrow t + 1$
5: Update: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$
6: Update: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
7: Correct Biases: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}$, $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$
8: Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta}$
9: Apply Update: $\theta \leftarrow \theta + \Delta\theta$
10: **end while**

Outline

- Perceptron
- Approximating linear functions
- Activation Function
- Backpropagation
- Optimization
- Neural Network Training and Design

Neural Network Training

Minibatch

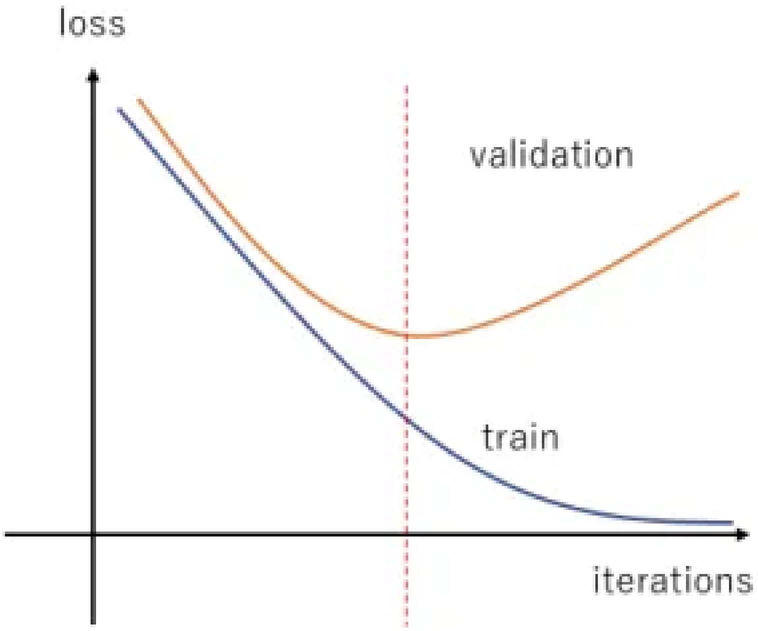
- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches (In theory, growingly larger)

- Advantage: Computation time per update does not depend on number of training examples.
- This allows convergence on extremely large datasets

- “Large Scale Learning with Stochastic Gradient Descent”, Leon Bottou.

Neural Network Training

Challenge: Overfitting

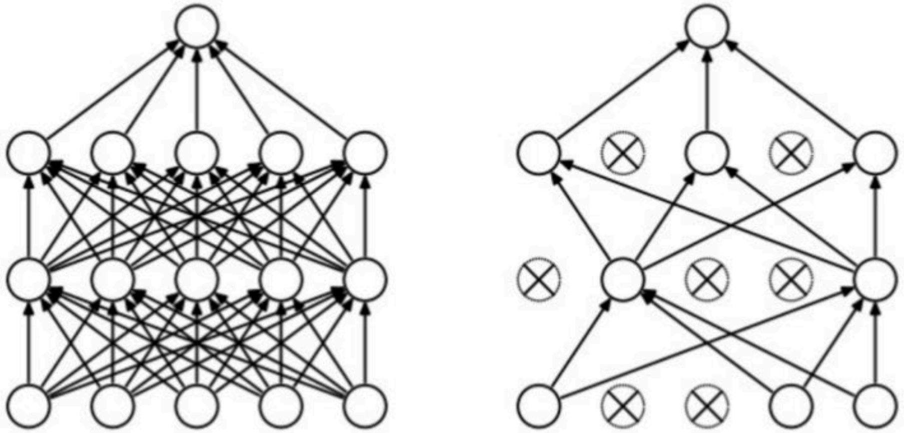


Neural Network Training

Dropout

How to avoid overfitting

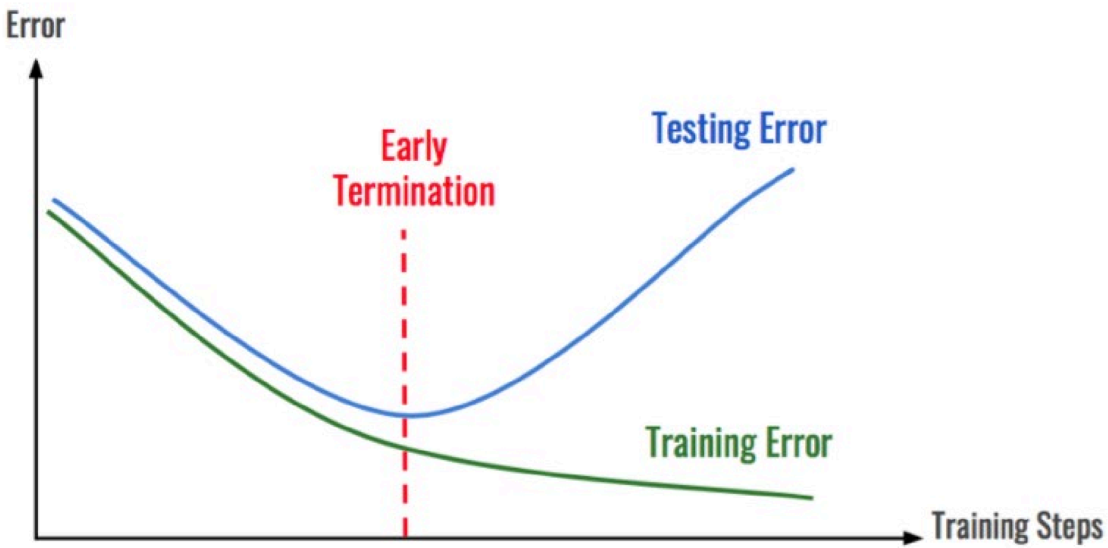
- An alternative method that complements the above is dropout
- While training, dropout keeps a neuron active with some probability p (a hyperparameter), or sets it to zero otherwise



(a) Standard Neural Net (b) After applying dropout.

Neural Network Training

- Early stop



Neural Network Training

- Batch Normalization
  - In ML, we assume future data will be drawn from same probability distribution as training data
  - For a hidden layer, after training, the earlier layers have new weights and hence may generate a new distribution for the next hidden layer
  - We want to reduce this internal covariate shift for the benefit of later layers

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Neural Network Training

- Batch Normalization
  - First three steps are just like standardization of input data, but with respect to only the data in mini-batch.
  - We can take derivative and incorporate the learning of last step parameters into backpropagation.
  - Note last step can completely un-do previous 3 steps
  - But even if so, this un-doing is driven by the later layers, not the earlier layers; later layers get to “choose” whether they want standard normal inputs or not

Neural Network Design

How to chose the number of layers and nodes

- No general rule of thumb, this depends on:
  - Amount of training data available
  - Complexity of the function that is trying to be learned
  - Number of input and output nodes
- If data is linearly separable, you don't need any hidden layers at all
- Start with one layer and hidden nodes proportional to input size
- Gradually increase

Hyperparameter Tuning

- Learning rate: how much to update the weight during optimization
- Number of epochs: number of times the entire training set pass through the neural network
- Batch size: the number of times the entire training set pass through the neural network
- Activation function: the function that introduces non-linearity to the model (e.g. sigmoid, tanh, ReLU, etc.)
- Number of hidden layers and units
- Weight initialization: Uniform distribution usually works well
- Dropout for regularization: probability of dropping a unit

We can perform grid or randomized search over all parameters

Challenge

High memory requirements

- Memory is used to store input data, weight parameters and activations as an input propagates through the network
- Activations from a forward pass must be retained until they can be used to calculate the error gradients in the backwards pass
- Example: 50-layer neural network
  - 26 million weight parameters, 16 million activations in the forward pass
  - 168MB memory (assuming 32-bit float)

Parallelize computations with GPU (graphics processing units)

Challenge

Backpropagation does not work well

- Deep networks trained with backpropagation (without unsupervised pretraining) perform worse than shallow networks
- Gradient is progressively getting more dilute
  - Weight correction is minimal after moving back a couple of layers
- High risk of getting "stuck" to local minima
- In practice, a small portion of data is labelled

Perform pretraining to mitigate this issue

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.2%	1.2%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.7%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

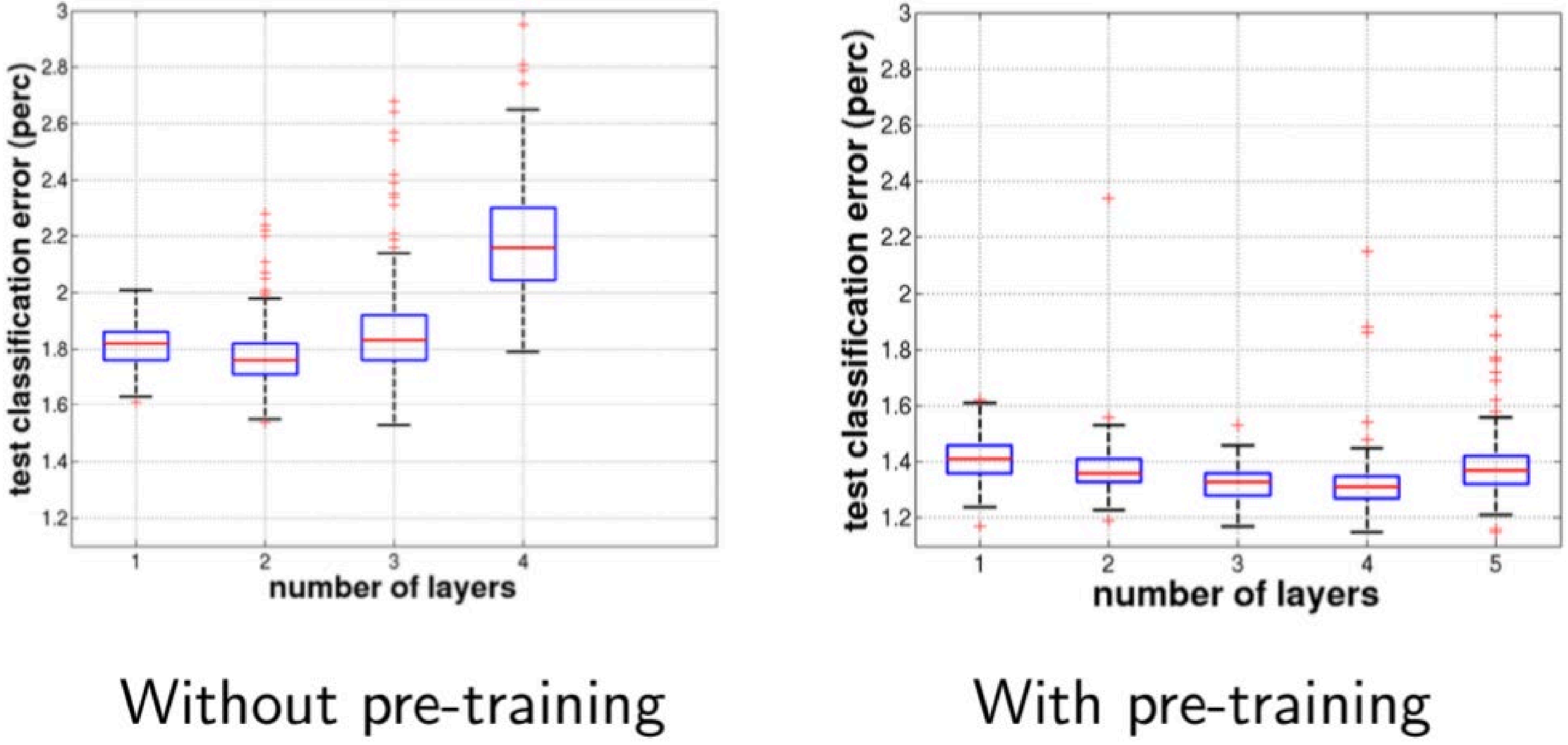
(Bengio et al., NIPS 2007)

Unsupervised Pretraining

- This idea came into play when research studies found that a DNN trained on a particular task (e.g. object recognition) can be applied on another domain (e.g. object subcategorization) giving state-of-the-art results
- **1st part: Greedy layer-wise unsupervised pre-training**
  - Each layer is pre-trained with an unsupervised learning algorithm
  - Learning a nonlinear transformation that captures the main variations in its input (the output of the previous layer)
- **2nd part: Supervised fine-tuning**
  - The deep architecture is fine-tuned with respect to a supervised training criterion with gradient-based optimization
- We will examine the deep belief networks and stacked autoencoders

Unusual form of regularization: minimizing variance and introducing bias towards configurations of the parameter space that are useful for unsupervised learning

Unsupervised Pretraining



[Source: Erhan et al., 2010]