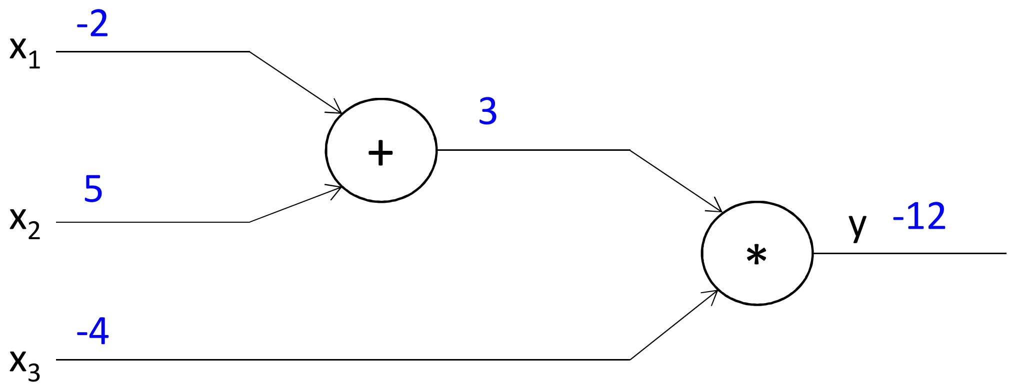
Backpropagation

Example

$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3$$

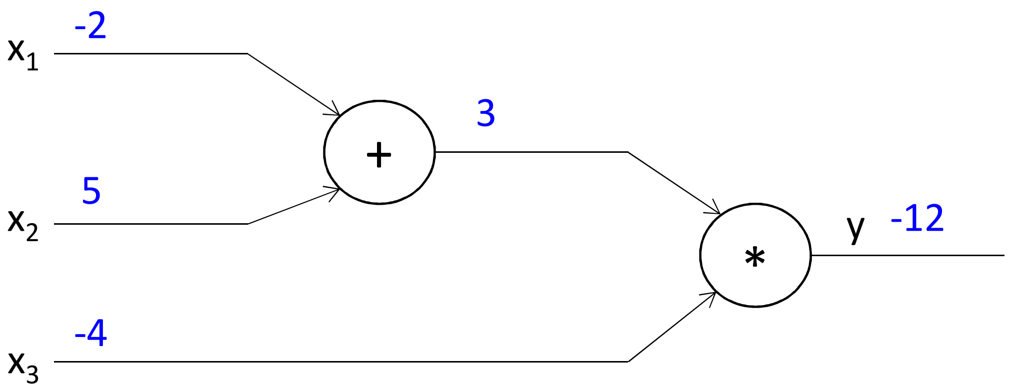
$$e.g.,\ x_1 = -2, x_2 = 5, x_3 = -4$$



Example

$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3$$

$$e.g.,\ x_1 = -2, x_2 = 5, x_3 = -4$$



$$z = x_1 + x_2 \qquad \frac{\partial z}{\partial x_1} = 1, \frac{\partial z}{\partial x_2} = 1$$

$$f = z * x_3 \qquad \frac{\partial f}{\partial z} = x_3, \frac{\partial f}{\partial x_3} = z$$

$$want: \qquad \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}$$

Example

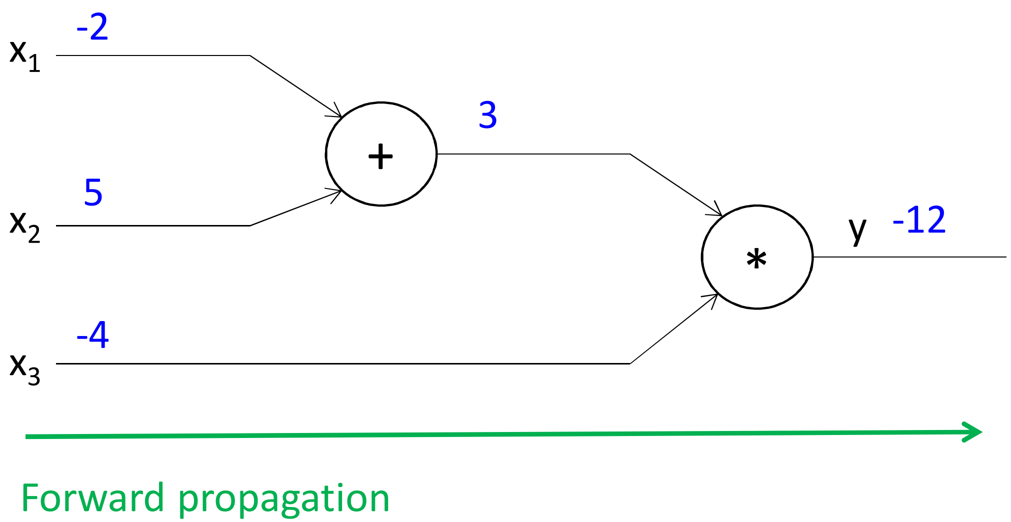
$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3$$

$$e.g.,\ x_1 = -2, x_2 = 5, x_3 = -4$$

$$z = x_1 + x_2 \qquad \frac{\partial z}{\partial x_1} = 1, \frac{\partial z}{\partial x_2} = 1$$

$$f = z * x_3 \qquad \frac{\partial f}{\partial z} = x_3, \frac{\partial f}{\partial x_3} = z$$

$$want: \quad \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}$$



Example

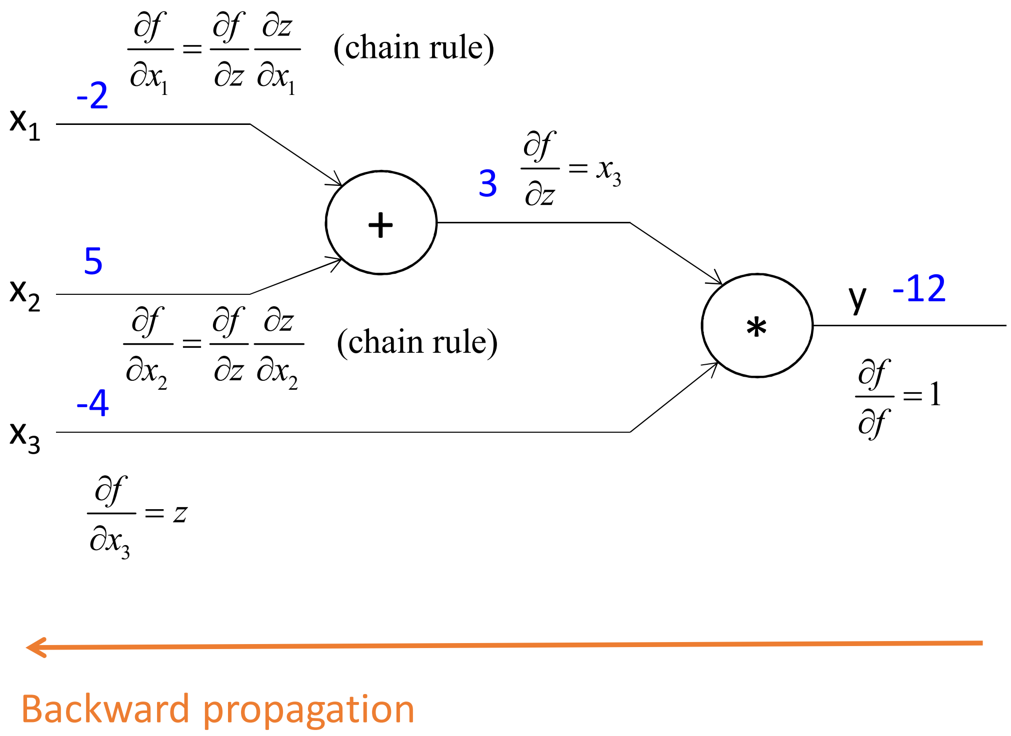
$$f(x_1, x_2, x_3) = (x_1 + x_2) * x_3$$

$$e.g.,\ x_1 = -2, x_2 = 5, x_3 = -4$$

$$z = x_1 + x_2 \qquad \frac{\partial z}{\partial x_1} = 1, \frac{\partial z}{\partial x_2} = 1$$

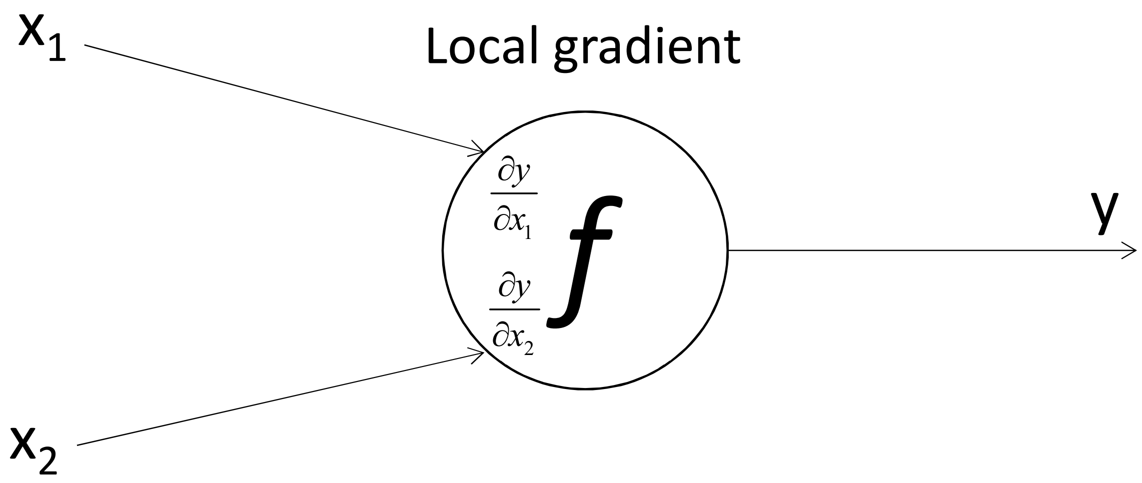
$$f = z * x_3 \qquad \frac{\partial f}{\partial z} = x_3, \frac{\partial f}{\partial x_3} = z$$

$$want: \quad \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}$$

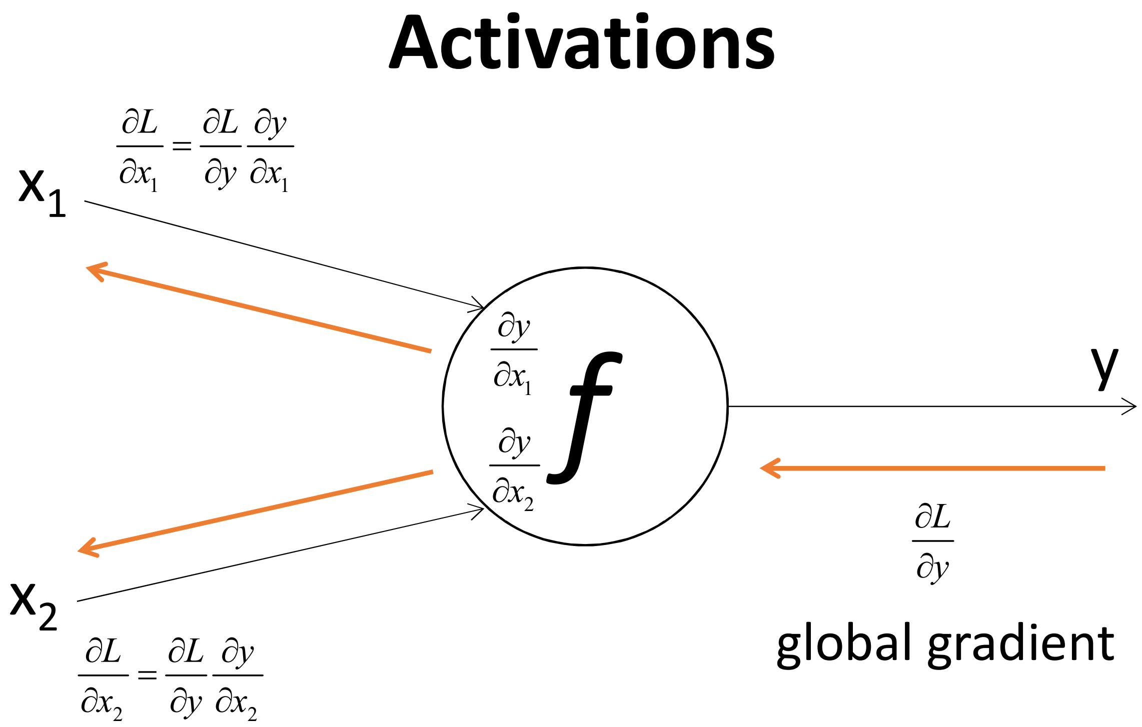


Local Gradient

Activations



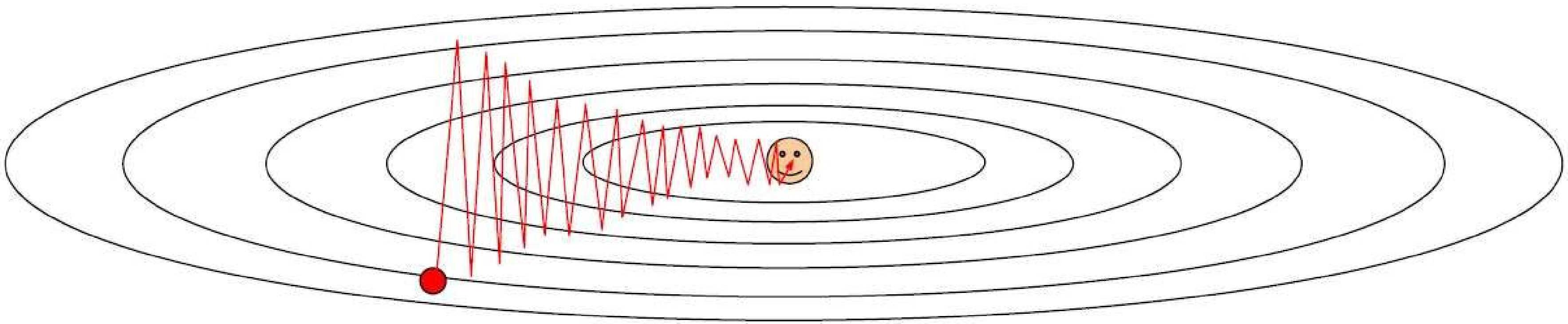
Global Gradient



Gradient

Stochastic Gradient Descent (SGC)

- Suppose loss function is steep vertically but shallow horizontally:



✓ Very slow progress along flat direction, jitter along steep one

Optimize Loss function

- Optimization through gradient descent

$$W \leftarrow W - \underline{\eta} \frac{\partial J(W)}{\partial W} \quad \Rightarrow \quad \theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta)$$

Learning rate

✓Small learning rates converge slowly and gets stuck in false local minima
✓Large learning rates overshoot, become unstable and diverge
✓Stable learning rates converge smoothly and avoid local minima

Optimizer

- Momentum

$$v \leftarrow \alpha v - \eta \nabla_{\theta} J(\theta)$$
$$\theta \leftarrow \theta + v$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L\left(f(x^{(i)}; \theta), y^{(i)} \right)$$

✓ v plays the role of velocity
  - ➢ It is the direction and speed at which the parameters move through parameter space
  - ➢ The velocity is set to an exponentially decaying average of the negative gradient

Optimizer

- Stochastic gradient descent with momentum

Require: Learning rate (η), momentum parameter (α)
Initial parameter (ϑ), initial velocity (v)

While stopping criterion not met **do**

Sample a minibatch of ***m*** examples from the training set with ***y***

Compute gradient estimate: $g \leftarrow \frac{1}{m}\nabla_{\theta}\sum_i L\left(f(x^{(i)};\theta), y^{(i)}\right)$

Compute velocity update: $v \leftarrow \alpha v - \eta g$

Apply update: $\theta \leftarrow \theta + v$

End while

Optimizer

- Nesterov accelerated gradient
  - ✓ The gradient is evaluated after the current velocity is applied
  - ✓ Thus, add a *correction factor* to the standard momentum

- Stochastic gradient descent with Nesterov momentum

Require: Learning rate (η), momentum parameter (α)
Initial parameter (ϑ), initial velocity (v)

While stopping criterion not met **do**

Sample a minibatch of ***m*** examples from the training set with ***y***

Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

Compute gradient estimate: $g \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L\left(f(x^{(i)}; \tilde{\theta}), y^{(i)} \right)$

Compute velocity update: $v \leftarrow \alpha v - \eta g$

Apply update: $\theta \leftarrow \theta + v$

End while

Optimizer - AdaGrad

- AdaGrad (Adaptive Gradient)
  - ✓ Individually adapts η of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient
  - ✓ The parameters with the largest partial derivatives have a correspondingly rapid decrease in their learning rate
  - ✓ The parameters with small partial derivatives have a relatively small decrease in their learning rate
  - ✓ AdaGrad performs well for some but not all deep learning models

Optimizer - AdaGrad

- AdaGrad (Adaptive Gradient)

Require: Learning rate (η), initial parameter (ϑ), small constant (δ)
Initialize gradient accumulation variable $\boldsymbol{r}$ = 0
While stopping criterion not met **do**

Sample a minibatch of $\boldsymbol{m}$ examples from the training set with $\boldsymbol{y}$

Compute gradient estimate: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f(x^{(i)}; \theta), y^{(i)} \right)$

Accumulate squared gradient: $r \leftarrow r + g \otimes g$

Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{\delta + r}} \otimes g$ (Division and square root applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta$

End while

Optimizer - RMSProp

- RMSProp (Root Mean Square Propagation)
  - ✓ Modifies AdaGrad to perform better in the nonconvex setting by changing the gradient accumulation into an exponentially weighted moving average
  - ✓ Converge rapidly when applied to a convex function
  - ✓ Uses an exponentially decaying average todiscard history from the extreme past

Require: Learning rate (η), decay rate (ρ) initial parameter (ϑ), small constant (δ)

Initialize gradient accumulation variable $\boldsymbol{r}$ = 0

While stopping criterion not met **do**

Sample a minibatch of $\boldsymbol{m}$ examples from the training set with $\boldsymbol{y}$

Compute gradient estimate: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(f(x^{(i)};\theta), y^{(i)} \right)$

Accumulate squared gradient: $r \leftarrow \rho r + (1-\rho) g \otimes g$

Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{\delta + r}} \otimes g$ (applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta$

End while

Optimizer – Adam

- Adam (Adaptive Moments)
  - ✓ Combination of RMSProp and momentum with a few important distinctions

  - ✓ Momentum is incorporated directly as an estimate of the first-order moment
    - ➢ Add momentum to RMSProp: apply momentum to the rescaled gradients
  - ✓ Includes bias corrections to the estimates of both the first-order and the second-order moments

Optimizer – Adam

- Adam (Adaptive Moments)

Require: Step size (ε), decay rates for moment (ρ_1,ρ_2),
small constant (δ), Initial parameters (ϑ)

Initialize 1st and 2nd moment variables ***s*** = 0, ***r*** = 0

Initialize time step *t* = 0

While stopping criterion not met **do**

Sample a minibatch of ***m*** examples from the training set with ***y***

Compute gradient estimate: $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f(x^{(i)};\theta), y^{(i)}\right)$

$t \leftarrow t+1$

Update biased first moment estimate: $s \leftarrow \rho_1 s + (1-\rho_1)g$

Update biased second moment estimate: $r \leftarrow \rho_2 r + (1-\rho_2)g \otimes g$

Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$

Correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

Compute update: $\Delta\theta \leftarrow -\varepsilon \frac{\hat{s}}{\sqrt{\hat{r}+\delta}}$

Apply update: $\theta \leftarrow \theta + \Delta\theta$

End while

END