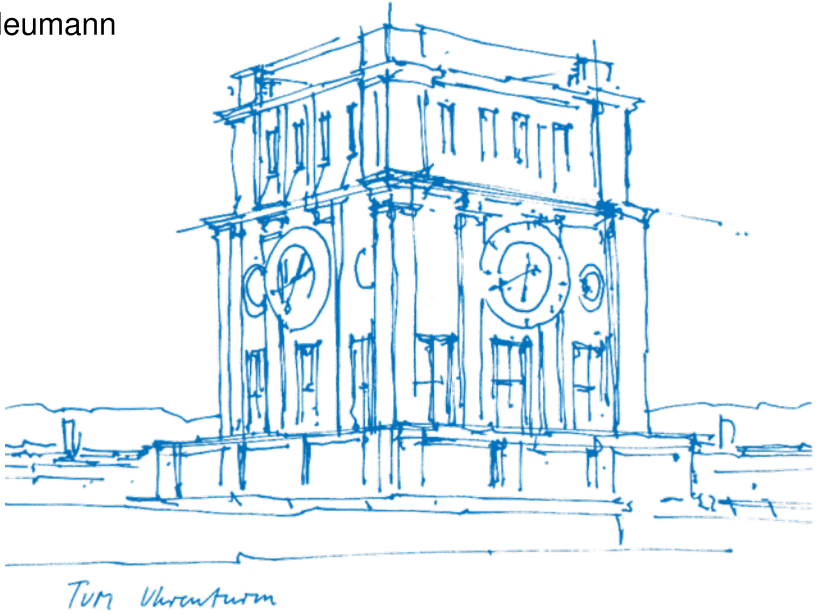
Chair III: Database Systems
Chair XXV: Data Science and Engineering
Department of Informatics
Technical University of Munich

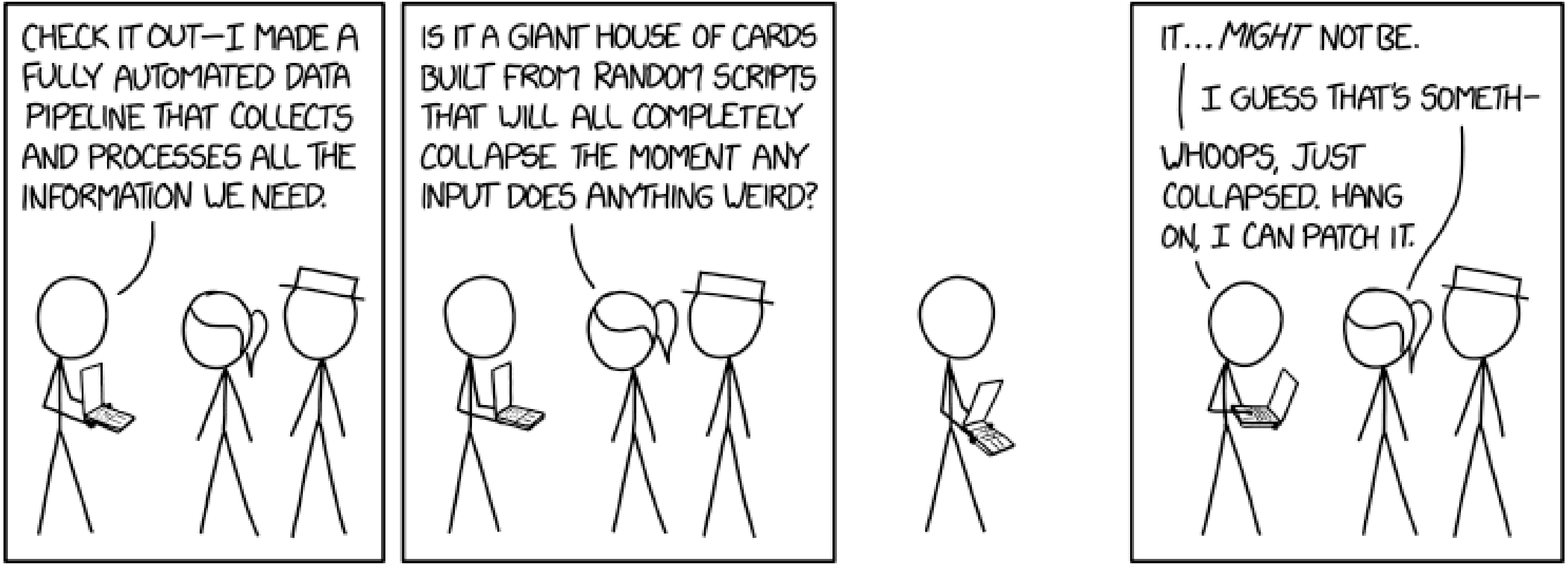
LLVM Code Optimisation for Automatic Differentiation

Maximilian E. Schüle, Maximilian Springer, Alfons Kemper, Thomas Neumann

Philadelphia, PA, USA, July 12, 2022

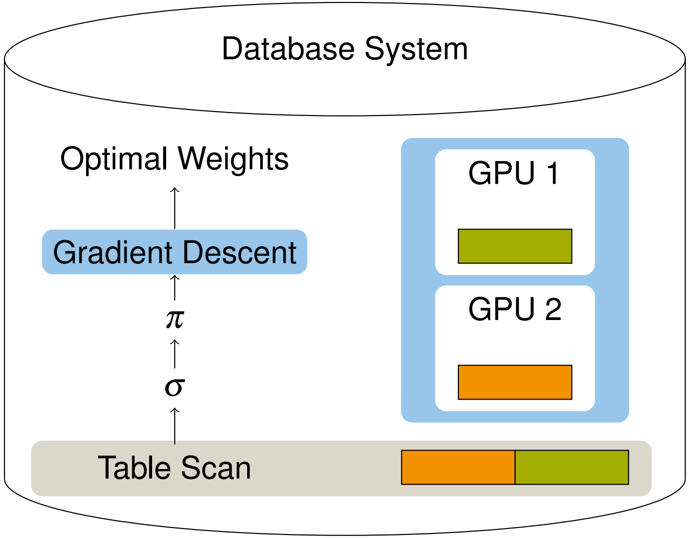


In-Database Machine Learning: Problem



In-Database Machine Learning: Solution

In-Database Machine Learning



- SQL sufficient for machine learning (ML)
  - Turing-complete with recursive tables
  - Sample operator for stochastic gradient descent
- Idea
  - Data preprocessing using SQL
  - No need for data extraction out of a database system
  - Label data within the database system using SQL

Structure

UMBRA

ML in SQL-92
Gradient descent with recursive tables

ML Operators
Automatic Differentiation

GPU support
Code-Generation for GPU

ML in SQL-92

ML in SQL-92: Gradient Descent with Recursive SQL

loss function $l_{x,y}(\vec{a,b})$, approximated values $m_{a,b}(x)$, given labels y, mean squared error:

$$m_{a,b}(x,y) = a \cdot x + b \approx y \tag{1}$$

$$l_{x,y}(a,b) = (a \cdot x + b - y)^2 \tag{2}$$

$$\nabla l_{x,y}(a,b) = \begin{pmatrix} \partial l / \partial a \\ \partial l / \partial b \end{pmatrix} = \begin{pmatrix} 2(ax+b-y) \cdot x \\ 2(ax+b-y) \end{pmatrix}. \tag{3}$$

Minimise $l_{x,y}(a,b)$: gradient descent (learning rate γ):

$$\begin{pmatrix} a_{t+1} \\ b_{t+1} \end{pmatrix} = \begin{pmatrix} a_t \\ b_t \end{pmatrix} - \gamma \nabla l_{x,y}(a_t, b_t), \tag{4}$$

$$\begin{pmatrix} a_\infty \\ b_\infty \end{pmatrix} \approx \lim_{t \to \infty} \left(\begin{pmatrix} a_t \\ b_t \end{pmatrix} \right). \tag{5}$$

```sql
create table data (x float, y float);
insert into data ...

with recursive gd (id, a, b) as (
 select 0,1::float,1::float
UNION ALL
 select id+1,
   a-0.05*avg(2*x*(a*x+b-y)),
   b-0.05*avg(2*(a*x+b-y))
 from gd, data
 where id<5 group by id,a,b)
select * from gd order by id;
```

Listing 1: Gradient descent.

Five iterations, loss function with two weights and $\gamma = 0.05$.

ML in SQL-92: Gradient Descent with Recursive SQL

loss function $l_{x,y}(\vec{a,b})$, approximated values $m_{a,b}(x)$, given labels y, mean squared error:

$$l_{x,y}(a,b) = (a \cdot x + b - y)^2 \tag{6}$$

$$\nabla l_{x,y}(a,b) = \begin{pmatrix} \partial l / \partial a \\ \partial l / \partial b \end{pmatrix} = \begin{pmatrix} 2(ax+b-y)\cdot x \\ 2(ax+b-y) \end{pmatrix}. \tag{7}$$

Minimise $l_{x,y}(a,b)$: gradient descent (learning rate γ):

$$\begin{pmatrix} a_{t+1} \\ b_{t+1} \end{pmatrix} = \begin{pmatrix} a_t \\ b_t \end{pmatrix} - \gamma \nabla l_{x,y}(a_t,b_t), \tag{8}$$

$$\begin{pmatrix} a_\infty \\ b_\infty \end{pmatrix} \approx \lim_{t\to\infty}\left(\begin{pmatrix} a_t \\ b_t \end{pmatrix}\right). \tag{9}$$

```sql
create table data (x float, y float);
insert into data ...

with recursive gd (id, a, b) as (
 select 0,1::float,1::float
UNION ALL
  select id+1, a-0.05*avg(d_a), b-0.05*avg(d_b)
  from umbra.derivation(TABLE (
    select id,a,b,x,y from gd,data where id<5),
    lambda (x) ((x.a * x.x + x.b - x.y)^2))
  group by id,a,b)
select * from gd order by id;
```

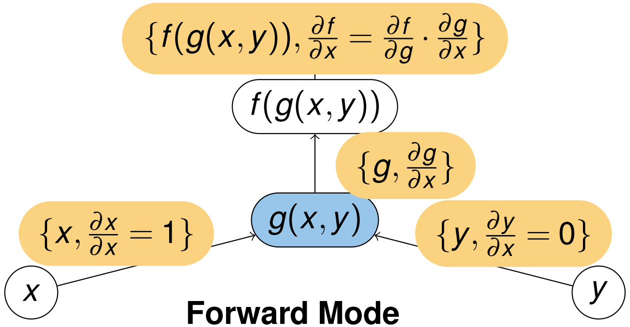
Listing 2: Gradient descent.

Five iterations, loss function with two weights and $\gamma = 0.05$.

ML Operators

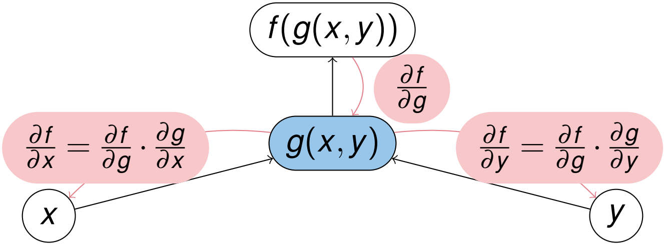
Forward or Reverse Mode Automatic Differentiation?

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$



Forward Mode

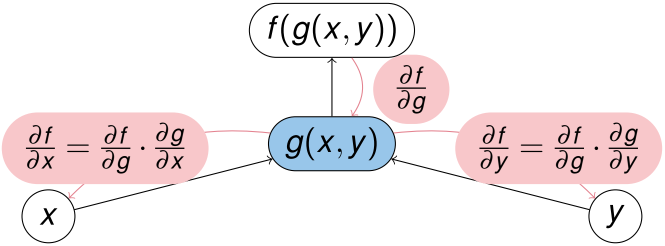
- requires one pass only (operator-overloading)
- leads to expression swell (?)



Reverse Mode

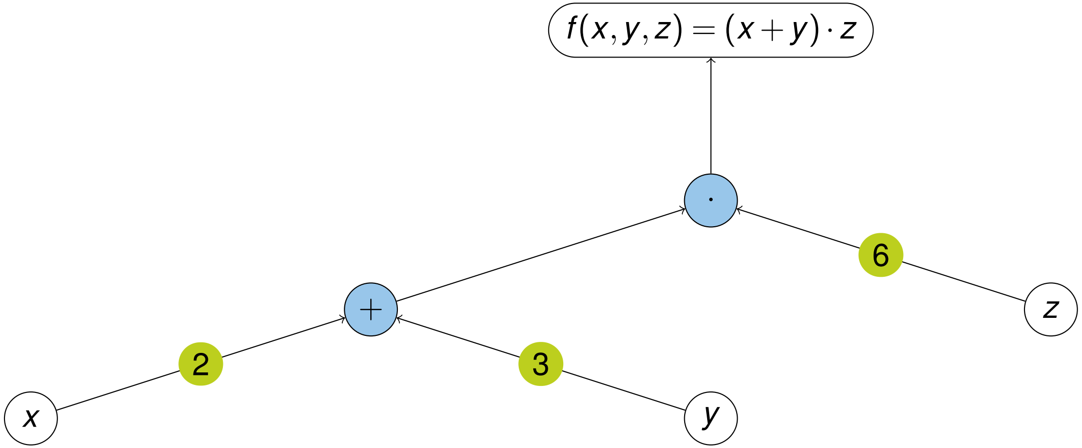
- computes all derivatives in one pass
- separate pass required

Reverse Mode Automatic Differentiation



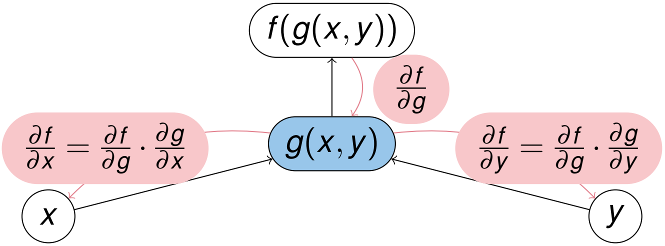
Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse



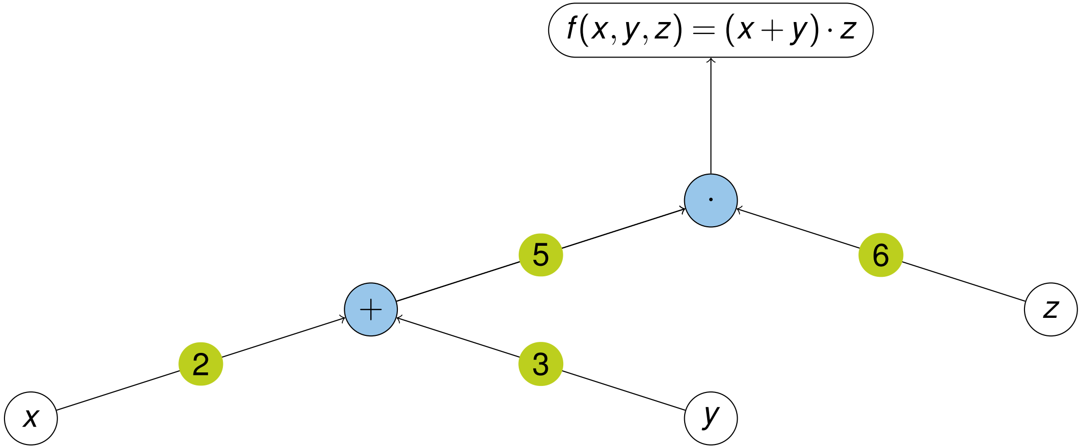

```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation



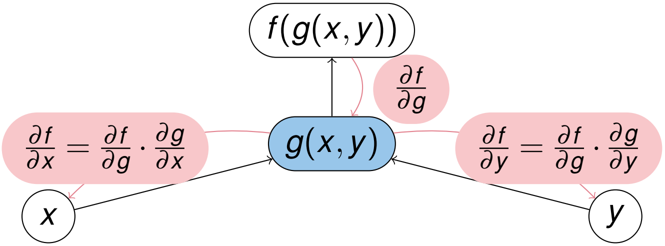
Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse



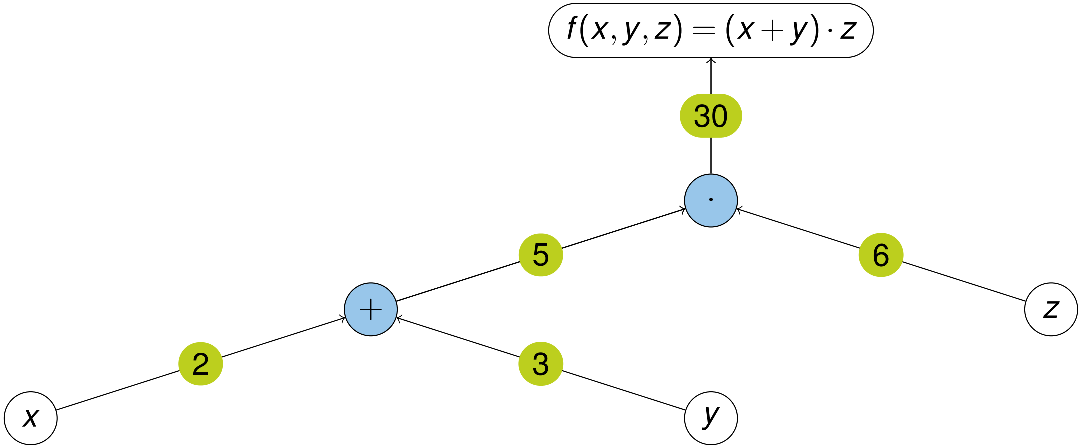

```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation



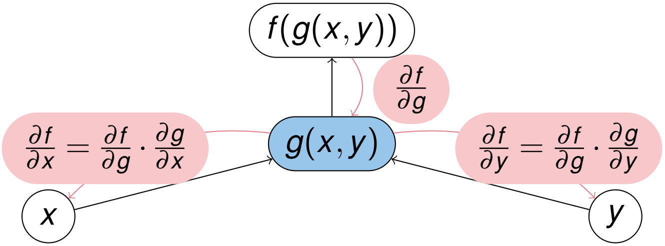
Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse



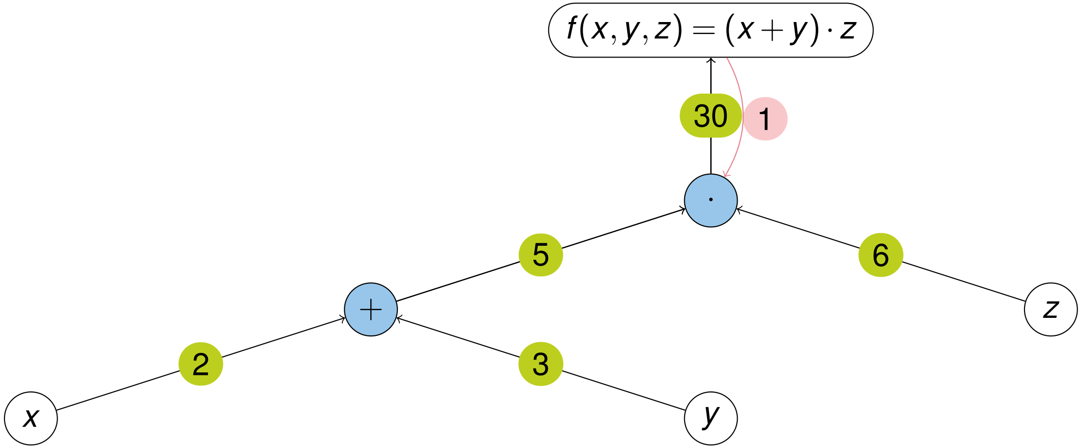

```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation



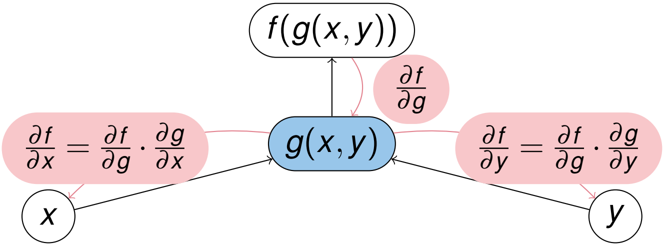
Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse



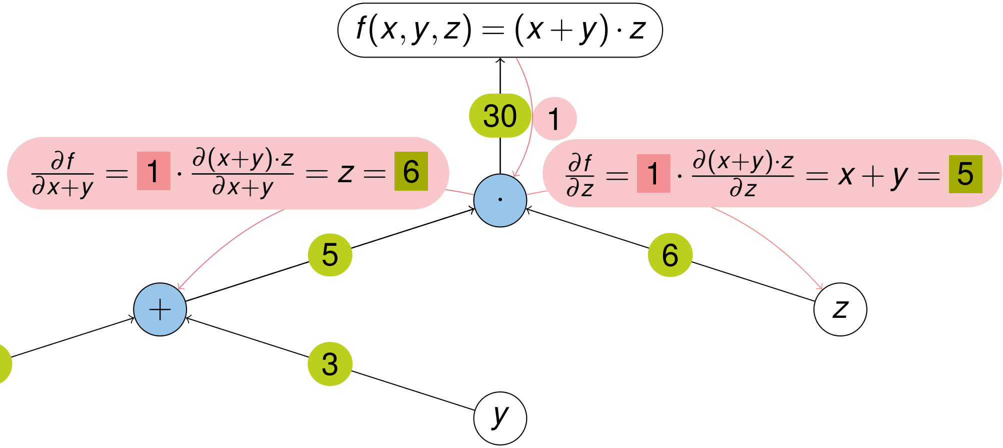

```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation



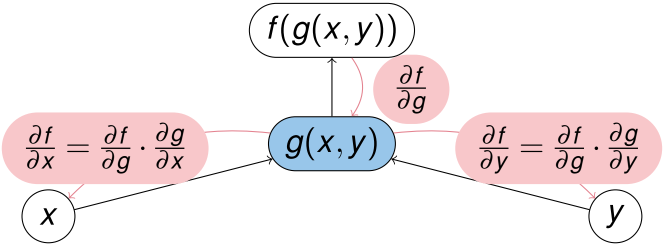
Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse



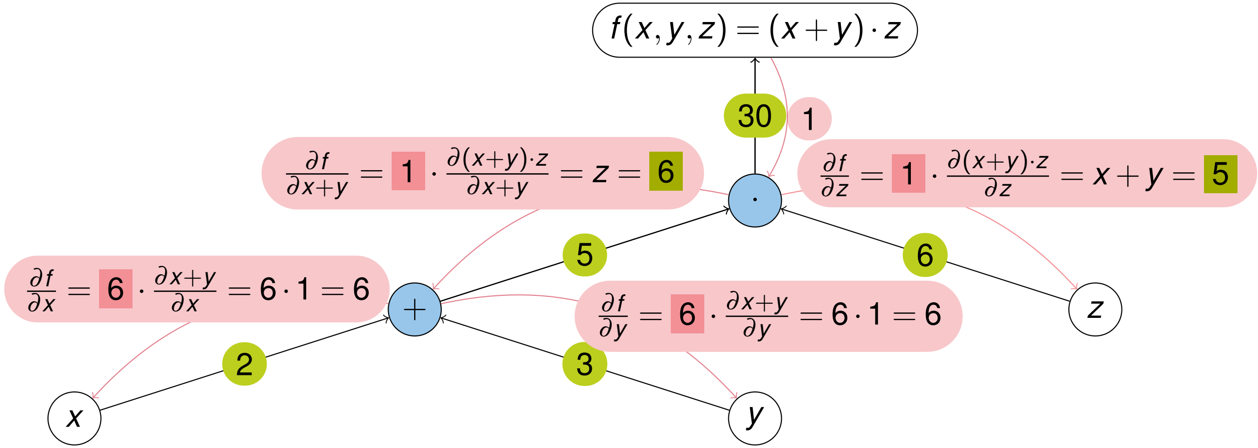

```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation



Automatic differentiation using reverse mode

- applying the chain rule to backpropagate the derivatives
- subexpressions are cached in LLVM registers for reuse




```
select * from umbra.derivation(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Reverse Mode Automatic Differentiation

function DERIVE(Z, *seed*)
  if Z matches $X + Y$ **then** DERIVE(X,*seed*); DERIVE(Y,*seed*)
  else if Z matches $X - Y$ **then** DERIVE(X,*seed*); DERIVE(Y,$-$*seed*)
  else if Z matches $X \cdot Y$ **then** DERIVE(X,$seed \cdot y$); DERIVE(Y,$seed \cdot x$)
  else if *isVariable*(Z) **then** $\frac{\partial}{\partial Z} \leftarrow \frac{\partial}{\partial Z} + seed$

Input: $(x+y)\cdot z$, $x := 2$, $y := 3$, $z := 6$
derive($(x+y)\cdot z$,1)

```
void deriveDerivation(/*...*/ Value seed, std::unordered_map<const IU*, Value>& derivatives) const {
 switch (z.getFunction()) {
  case KnownFunction::Add:
   context.deriveDerivatives(z.getLeft(), seed, derivatives);
   context.deriveDerivatives(z.getRight(), seed, derivatives);
   break;
  case KnownFunction::Mul:
   context.deriveDerivatives(z.getLeft(), seed.evaluateBinary(KnownFunction::Mul, right), derivatives);
   context.deriveDerivatives(z.getRight(), seed.evaluateBinary(KnownFunction::Mul, left), derivatives);
   break;
/*...*/
```

Reverse Mode Automatic Differentiation

function DERIVE(Z, *seed*)
  if Z matches $X + Y$ **then** DERIVE(X,*seed*); DERIVE(Y,*seed*)
  else if Z matches $X - Y$ **then** DERIVE(X,*seed*); DERIVE(Y,$-$*seed*)
  else if Z matches $X \cdot Y$ **then** DERIVE(X,$seed \cdot y$); DERIVE(Y,$seed \cdot x$)
  else if *isVariable*(Z) **then** $\frac{\partial}{\partial Z} \leftarrow \frac{\partial}{\partial Z} + seed$

Input: $(x+y)\cdot z$, $x := 2$, $y := 3$, $z := 6$
derive($(x+y)\cdot z$,1)

- derive($x+y$,$1 \cdot z = 1 \cdot 6$)
- derive(z,$1 \cdot (x+y) = 1 \cdot 5$)

```
void deriveDerivation(/*...*/ Value seed, std::unordered_map<const IU*, Value>& derivatives) const {
 switch (z.getFunction()) {
  case KnownFunction::Add:
   context.deriveDerivatives(z.getLeft(), seed, derivatives);
   context.deriveDerivatives(z.getRight(), seed, derivatives);
   break;
  case KnownFunction::Mul:
   context.deriveDerivatives(z.getLeft(), seed.evaluateBinary(KnownFunction::Mul, right), derivatives);
   context.deriveDerivatives(z.getRight(), seed.evaluateBinary(KnownFunction::Mul, left), derivatives);
   break;
/*...*/
```

Reverse Mode Automatic Differentiation

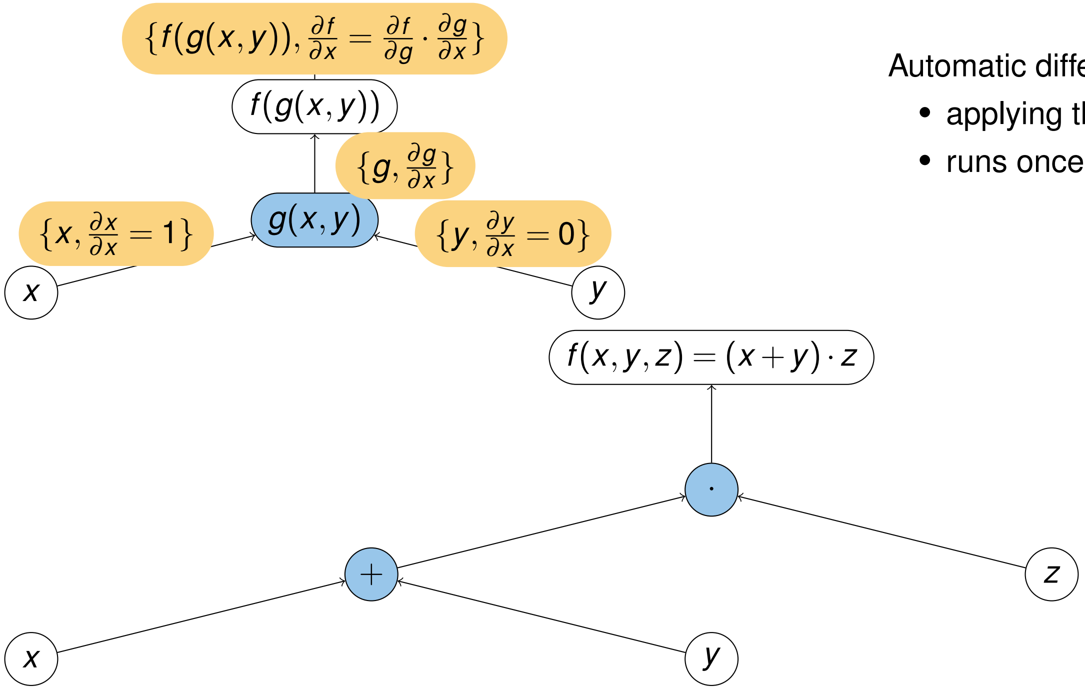
function DERIVE(Z, *seed*)
  if Z matches $X + Y$ **then** DERIVE(X,*seed*); DERIVE(Y,*seed*)
  else if Z matches $X - Y$ **then** DERIVE(X,*seed*); DERIVE(Y,$-$*seed*)
  else if Z matches $X \cdot Y$ **then** DERIVE(X,$seed \cdot y$); DERIVE(Y,$seed \cdot x$)
  else if *isVariable*(Z) **then** $\frac{\partial}{\partial Z} \leftarrow \frac{\partial}{\partial Z} + seed$

Input: $(x+y)\cdot z$, $x := 2$, $y := 3$, $z := 6$
derive($(x+y)\cdot z$,1)

- derive($x+y$,$1\cdot z = 1\cdot 6$)
  - derive(x,$6\cdot 1 = 6$)
  - derive(y,$6\cdot 1 = 6$)
- derive(z,$1\cdot(x+y) = 1\cdot 5$)

```
void deriveDerivation(/*...*/ Value seed, std::unordered_map<const IU*, Value>& derivatives) const {
 switch (z.getFunction()) {
  case KnownFunction::Add:
   context.deriveDerivatives(z.getLeft(), seed, derivatives);
   context.deriveDerivatives(z.getRight(), seed, derivatives);
   break;
  case KnownFunction::Mul:
   context.deriveDerivatives(z.getLeft(), seed.evaluateBinary(KnownFunction::Mul, right), derivatives);
   context.deriveDerivatives(z.getRight(), seed.evaluateBinary(KnownFunction::Mul, left), derivatives);
   break;
/*...*/
```

Forward Mode Automatic Differentiation

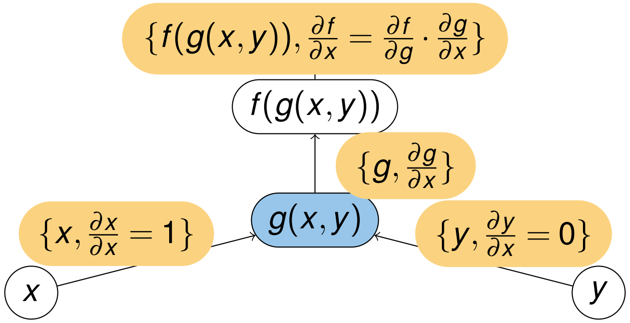


Automatic differentiation using forward mode

- applying the chain rule in the forward pass
- runs once per derivative

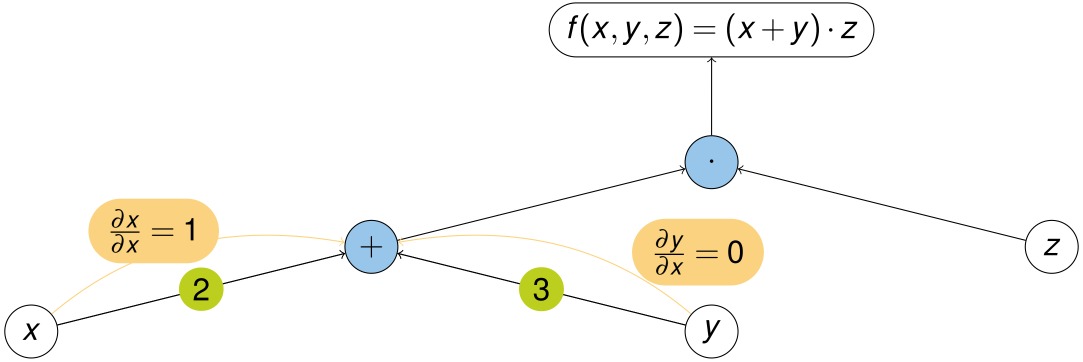
```
select * from umbra.derivation2(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Forward Mode Automatic Differentiation



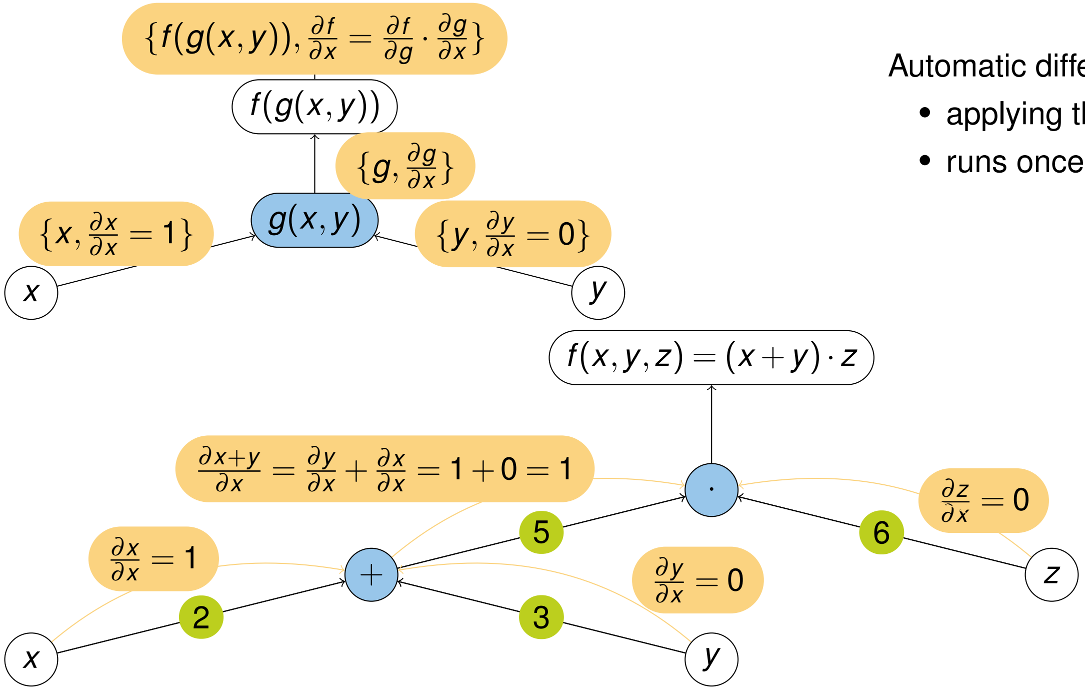
Automatic differentiation using forward mode

- applying the chain rule in the forward pass
- runs once per derivative




```
select * from umbra.derivation2(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Forward Mode Automatic Differentiation

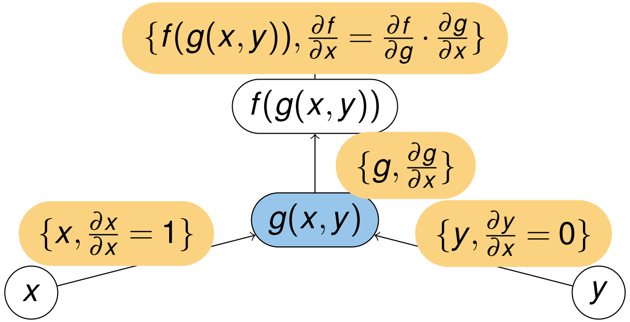


Automatic differentiation using forward mode

- applying the chain rule in the forward pass
- runs once per derivative

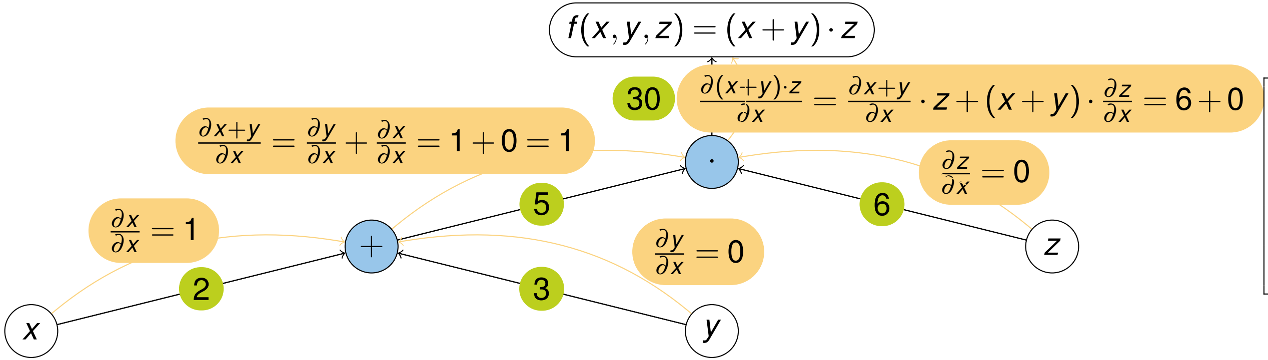
```
select * from umbra.derivation2(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Forward Mode Automatic Differentiation



Automatic differentiation using forward mode

- applying the chain rule in the forward pass
- runs once per derivative




```
select * from umbra.derivation2(
  TABLE(select 2 x,3 y,6 z),
  lambda(x)((x.x+x.y)*x.z));
-- x y z d_x d_y d_z
-- 2 3 6 6 6 5
```

Forward Mode Automatic Differentiation

function EVAL(Z, V)
  if *isVariable*(Z) **then**
    if Z matches V **then return** $\{z, 1\}$
    elsereturn $\{z, 0\}$
  else$\{x, x'\} \leftarrow$ EVAL(X, V); $\{y, y'\} \leftarrow$ EVAL(Y, V)
    if Z matches $X + Y$ **then return** $\{x + y, x' + y'\}$
    else if Z matches $X - Y$ **then return** $\{x - y, x' - y'\}$
    else if Z matches $X \cdot Y$ **then return** $\{x \cdot y, x' \cdot y + x \cdot y'\}$

Input: $(x + y) \cdot z$, $x := 2$, $y := 3$, $z := 6$
eval($(x + y) \cdot z$,x):

```
shared_ptr<dual> forwardDeriveBinaryExpression(BinaryExpression* z, IU* v) {
  switch (z->getFunction()) {
    case KnownFunction::Add: {
      auto left = forwardDeriveExpression(z->getInput(0), v);
      auto right = forwardDeriveExpression(z->getInput(1), v);
      return make_shared<Autodiff::dual>(/*...*/
```

Forward Mode Automatic Differentiation

function EVAL(Z, V)
  if *isVariable*(Z) **then**
    if Z matches V **then return** $\{z, 1\}$
    elsereturn $\{z, 0\}$
  else$\{x, x'\} \leftarrow$ EVAL(X, V); $\{y, y'\} \leftarrow$ EVAL(Y, V)
    if Z matches $X + Y$ **then return** $\{x + y, x' + y'\}$
    else if Z matches $X - Y$ **then return** $\{x - y, x' - y'\}$
    else if Z matches $X \cdot Y$ **then return** $\{x \cdot y, x' \cdot y + x \cdot y'\}$

Input: $(x + y) \cdot z$, $x := 2$, $y := 3$, $z := 6$
eval($(x + y) \cdot z$,x):

- eval($(x + y)$,x):
- eval(z,x): returns {6,0}

```
shared_ptr<dual> forwardDeriveBinaryExpression(BinaryExpression* z, IU* v) {
   switch (z->getFunction()) {
      case KnownFunction::Add: {
         auto left = forwardDeriveExpression(z->getInput(0), v);
         auto right = forwardDeriveExpression(z->getInput(1), v);
         return make_shared<Autodiff::dual>(/*...*/
```

Forward Mode Automatic Differentiation

function EVAL(Z, V)
    if *isVariable*(Z) **then**
        if Z matches V **then return** $\{z, 1\}$
        elsereturn $\{z, 0\}$
    else$\{x, x'\} \leftarrow$ EVAL(X, V); $\{y, y'\} \leftarrow$ EVAL(Y, V)
        if Z matches $X + Y$ **then return** $\{x + y, x' + y'\}$
        else if Z matches $X - Y$ **then return** $\{x - y, x' - y'\}$
        else if Z matches $X \cdot Y$ **then return** $\{x \cdot y, x' \cdot y + x \cdot y'\}$

Input: $(x + y) \cdot z$, $x := 2$, $y := 3$, $z := 6$
eval($(x + y) \cdot z$,x):

- eval($(x + y)$,x):
  - eval(x,x): returns {2,1}
  - eval(y,x): returns {3,0}
- eval(z,x): returns {6,0}

```
shared_ptr<dual> forwardDeriveBinaryExpression(BinaryExpression* z, IU* v) {
  switch (z->getFunction()) {
    case KnownFunction::Add: {
      auto left = forwardDeriveExpression(z->getInput(0), v);
      auto right = forwardDeriveExpression(z->getInput(1), v);
      return make_shared<Autodiff::dual>(/*...*/
```

Forward Mode Automatic Differentiation

function EVAL(Z, V)
  if *isVariable*(Z) **then**
    if Z matches V **then return** $\{z, 1\}$
    elsereturn $\{z, 0\}$
  else$\{x, x'\} \leftarrow$ EVAL(X, V); $\{y, y'\} \leftarrow$ EVAL(Y, V)
    if Z matches $X + Y$ **then return** $\{x + y, x' + y'\}$
    else if Z matches $X - Y$ **then return** $\{x - y, x' - y'\}$
    else if Z matches $X \cdot Y$ **then return** $\{x \cdot y, x' \cdot y + x \cdot y'\}$

Input: $(x + y) \cdot z$, $x := 2$, $y := 3$, $z := 6$
eval($(x + y) \cdot z$,x):

- eval($(x + y)$,x): returns $\{5, 1 + 0 = 1\}$
  - eval(x,x): returns {2,1}
  - eval(y,x): returns {3,0}
- eval(z,x): returns {6,0}

```
shared_ptr<dual> forwardDeriveBinaryExpression(BinaryExpression* z, IU* v) {
  switch (z->getFunction()) {
    case KnownFunction::Add: {
      auto left = forwardDeriveExpression(z->getInput(0), v);
      auto right = forwardDeriveExpression(z->getInput(1), v);
      return make_shared<Autodiff::dual>(/*...*/
```

Forward Mode Automatic Differentiation

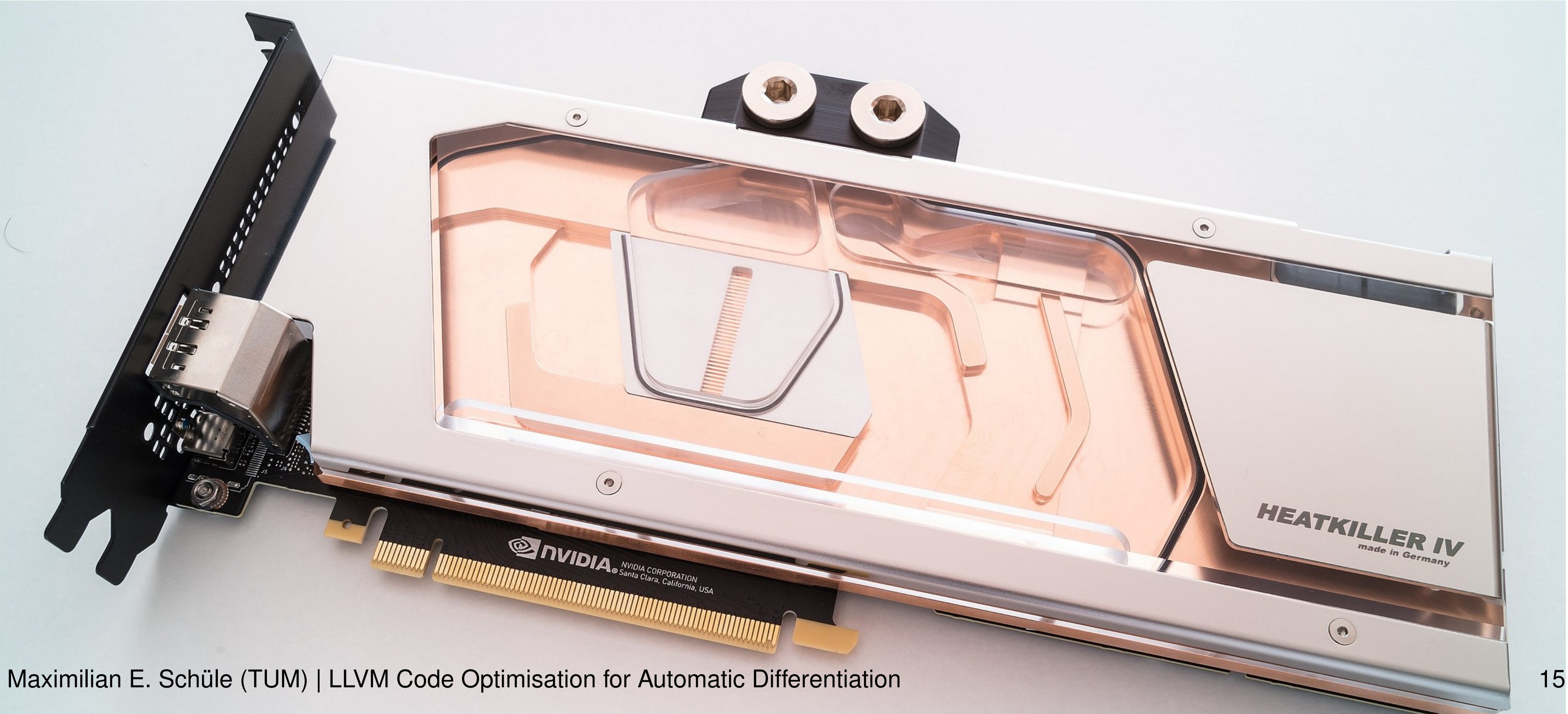
function EVAL(Z, V)
  if *isVariable*(Z) **then**
    if Z matches V **then return** $\{z, 1\}$
    elsereturn $\{z, 0\}$
  else $\{x, x'\} \leftarrow$ EVAL(X, V); $\{y, y'\} \leftarrow$ EVAL(Y, V)
    if Z matches $X + Y$ **then return** $\{x + y, x' + y'\}$
    else if Z matches $X - Y$ **then return** $\{x - y, x' - y'\}$
    else if Z matches $X \cdot Y$ **then return** $\{x \cdot y, x' \cdot y + x \cdot y'\}$

Input: $(x+y) \cdot z$, $x := 2$, $y := 3$, $z := 6$
eval($(x+y) \cdot z$,x): returns {5,$1 \cdot 6 + 5 \cdot 0 = 6$}

- eval($(x+y)$,x): returns {5,$1 + 0 = 1$}
  - eval(x,x): returns {2,1}
  - eval(y,x): returns {3,0}
- eval(z,x): returns {6,0}

```
shared_ptr<dual> forwardDeriveBinaryExpression(BinaryExpression* z, IU* v) {
   switch (z->getFunction()) {
      case KnownFunction::Add: {
         auto left = forwardDeriveExpression(z->getInput(0), v);
         auto right = forwardDeriveExpression(z->getInput(1), v);
         return make_shared<Autodiff::dual>(/*...*/
```

Code-Generation for GPU



Code-Generation for GPU

LLVM IR: forward mode (before compiler optimisation)

- Example: linear regression, mean squared error:

$$L = (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)^2,$$
$$\frac{\partial L}{\partial w_0} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_0,$$
$$\frac{\partial L}{\partial w_1} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_1.$$

- $(w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)$ computed twice
- red: GPU-specific (determine memory positions)

```
%4 = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x(), !range !8
%5 = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x(), !range !9
%6 = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x(), !range !10
%CUDABuiltin_cpp_97_ = mul i32 %6, %5
%CUDABuiltin_cpp_97_0 = add i32 %CUDABuiltin_cpp_97_, %4
%CodeGen_cpp_1539_ = zext i32 %CUDABuiltin_cpp_97_0 to i64
%Autodiff_cpp_700_ = getelementptr double, double* %arg4, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_2 = getelementptr double, double* %arg5, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_4 = getelementptr double, double* %3, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_6 = getelementptr double, double* %2, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_8 = getelementptr double, double* %0, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_10 = getelementptr double, double* %1, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_519_ = load double, double* %Autodiff_cpp_700_, align 8
%Autodiff_cpp_519_11 = load double, double* %Autodiff_cpp_700_4, align 8
%Autodiff_cpp_519_12 = load double, double* %Autodiff_cpp_700_6, align 8
%Autodiff_cpp_501_ = fmul double %Autodiff_cpp_519_11, %Autodiff_cpp_519_12
%Autodiff_cpp_519_13 = load double, double* %Autodiff_cpp_700_10, align 8
%Autodiff_cpp_519_14 = load double, double* %Autodiff_cpp_700_8, align 8
%Autodiff_cpp_501_15 = fmul double %Autodiff_cpp_519_13, %Autodiff_cpp_519_14
%Autodiff_cpp_493_ = fadd double %Autodiff_cpp_501_, %Autodiff_cpp_501_15
%Autodiff_cpp_493_16 = fadd double %Autodiff_cpp_519_, %Autodiff_cpp_493_
%Autodiff_cpp_519_17 = load double, double* %Autodiff_cpp_700_2, align 8
%Autodiff_cpp_497_ = fsub double %Autodiff_cpp_493_16, %Autodiff_cpp_519_17
%Autodiff_cpp_501_18 = fmul double %Autodiff_cpp_497_, 2.000000e+00
%Autodiff_cpp_501_20 = fmul double %Autodiff_cpp_519_13, %Autodiff_cpp_501_18
%7 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1539_
store double %Autodiff_cpp_501_20, double* %7, align 8
%Autodiff_cpp_519_21 = load double, double* %Autodiff_cpp_700_, align 8
%Autodiff_cpp_519_22 = load double, double* %Autodiff_cpp_700_4, align 8
%Autodiff_cpp_519_23 = load double, double* %Autodiff_cpp_700_6, align 8
%Autodiff_cpp_501_24 = fmul double %Autodiff_cpp_519_22, %Autodiff_cpp_519_23
%Autodiff_cpp_519_25 = load double, double* %Autodiff_cpp_700_10, align 8
%Autodiff_cpp_519_26 = load double, double* %Autodiff_cpp_700_8, align 8
%Autodiff_cpp_501_27 = fmul double %Autodiff_cpp_519_25, %Autodiff_cpp_519_26
%Autodiff_cpp_493_28 = fadd double %Autodiff_cpp_501_24, %Autodiff_cpp_501_27
%Autodiff_cpp_493_29 = fadd double %Autodiff_cpp_519_21, %Autodiff_cpp_493_28
%Autodiff_cpp_519_30 = load double, double* %Autodiff_cpp_700_2, align 8
%Autodiff_cpp_497_31 = fsub double %Autodiff_cpp_493_29, %Autodiff_cpp_519_30
%Autodiff_cpp_501_32 = fmul double %Autodiff_cpp_497_31, 2.000000e+00
%Autodiff_cpp_501_34 = fmul double %Autodiff_cpp_519_22, %Autodiff_cpp_501_32
%Autodiff_cpp_704_ = add i32 %CUDABuiltin_cpp_97_0, 32768
%CodeGen_cpp_1599_35 = zext i32 %Autodiff_cpp_704_ to i64
%8 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1599_35
store double %Autodiff_cpp_501_34, double* %8, align 8
ret void
```

Code-Generation for GPU

LLVM IR: forward mode (after compiler optimisation)

- Example: linear regression, mean squared error:

$$L = (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)^2,$$
$$\frac{\partial L}{\partial w_0} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_0,$$
$$\frac{\partial L}{\partial w_1} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_1.$$

- $(w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)$ computed **once**
- red: GPU-specific (determine memory positions)

```
%3 = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x(), !range !8
%4 = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x(), !range !9
%5 = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x(), !range !10
%CUDABuiltin_cpp_97_ = mul i32 %5, %4
%CUDABuiltin_cpp_97_0 = add i32 %CUDABuiltin_cpp_97_, %3
%CodeGen_cpp_1539_ = zext i32 %CUDABuiltin_cpp_97_0 to i64
%Autodiff_cpp_700_ = getelementptr double, double* %arg4, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_2 = getelementptr double, double* %arg5, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_4 = getelementptr double, double* %2, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_6 = getelementptr double, double* %1, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_8 = getelementptr double, double* %0, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_700_10 = getelementptr double, double* %"01", i64 %CodeGen_cpp_1539_
%Autodiff_cpp_519_ = load double, double* %Autodiff_cpp_700_, align 8
%Autodiff_cpp_519_11 = load double, double* %Autodiff_cpp_700_4, align 8
%Autodiff_cpp_519_12 = load double, double* %Autodiff_cpp_700_6, align 8
%Autodiff_cpp_501_ = fmul double %Autodiff_cpp_519_11, %Autodiff_cpp_519_12
%Autodiff_cpp_519_13 = load double, double* %Autodiff_cpp_700_10, align 8
%Autodiff_cpp_519_14 = load double, double* %Autodiff_cpp_700_8, align 8
%Autodiff_cpp_501_15 = fmul double %Autodiff_cpp_519_13, %Autodiff_cpp_519_14
%Autodiff_cpp_493_ = fadd double %Autodiff_cpp_501_, %Autodiff_cpp_501_15
%Autodiff_cpp_493_16 = fadd double %Autodiff_cpp_519_, %Autodiff_cpp_493_
%Autodiff_cpp_519_17 = load double, double* %Autodiff_cpp_700_2, align 8
%Autodiff_cpp_497_ = fsub double %Autodiff_cpp_493_16, %Autodiff_cpp_519_17
%Autodiff_cpp_501_18 = fmul double %Autodiff_cpp_497_, 2.000000e+00
%Autodiff_cpp_501_20 = fmul double %Autodiff_cpp_519_13, %Autodiff_cpp_501_18
%6 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1539_
store double %Autodiff_cpp_501_20, double* %6, align 8
%Autodiff_cpp_501_34 = fmul double %Autodiff_cpp_519_11, %Autodiff_cpp_501_18
%Autodiff_cpp_704_ = add i32 %CUDABuiltin_cpp_97_0, 32768
%CodeGen_cpp_1599_35 = zext i32 %Autodiff_cpp_704_ to i64
%7 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1599_35
store double %Autodiff_cpp_501_34, double* %7, align 8
ret void
```

Code-Generation for GPU

LLVM IR: reverse mode

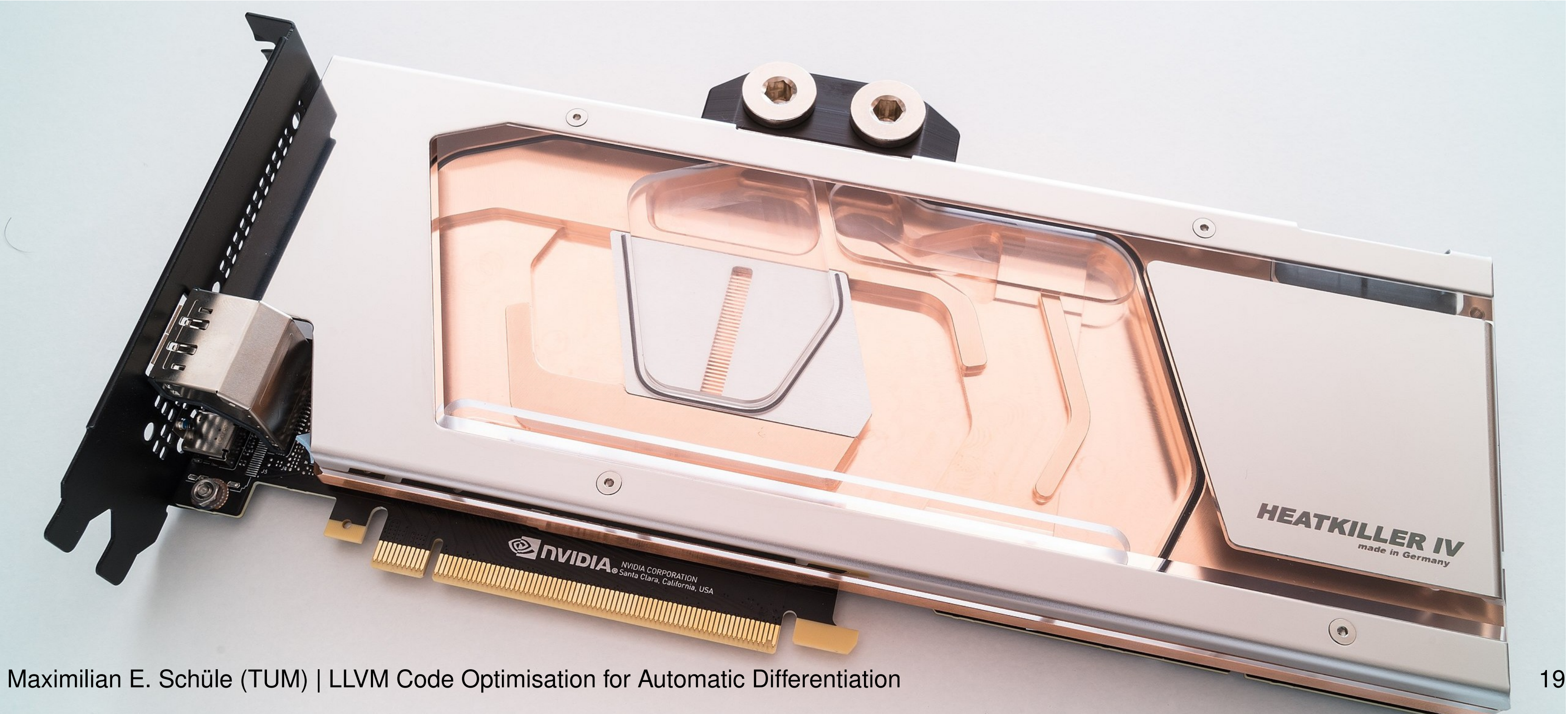
- Example: linear regression, mean squared error:

$$L = (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)^2,$$
$$\frac{\partial L}{\partial w_0} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_0,$$
$$\frac{\partial L}{\partial w_1} = 2 \cdot (w_0 \cdot x_0 + x_1 \cdot w_1 + b - y) \cdot x_1.$$

- $(w_0 \cdot x_0 + x_1 \cdot w_1 + b - y)$ computed **once**
- red: GPU-specific (determine memory positions)

```
%4 = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x(), !range !8
%5 = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x(), !range !9
%6 = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x(), !range !10
%CUDABuiltin_cpp_97_ = mul i32 %6, %5
%CUDABuiltin_cpp_97_1 = add i32 %CUDABuiltin_cpp_97_, %4
%CodeGen_cpp_1539_ = zext i32 %CUDABuiltin_cpp_97_1 to i64
%Autodiff_cpp_616_ = getelementptr double, double* %arg0, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_616_3 = getelementptr double, double* %arg00, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_616_5 = getelementptr double, double* %3, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_616_7 = getelementptr double, double* %2, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_616_9 = getelementptr double, double* %0, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_616_11 = getelementptr double, double* %1, i64 %CodeGen_cpp_1539_
%Autodiff_cpp_519_ = load double, double* %Autodiff_cpp_616_, align 8
%Autodiff_cpp_519_12 = load double, double* %Autodiff_cpp_616_5, align 8
%Autodiff_cpp_519_13 = load double, double* %Autodiff_cpp_616_7, align 8
%Autodiff_cpp_501_ = fmul double %Autodiff_cpp_519_12, %Autodiff_cpp_519_13
%Autodiff_cpp_519_14 = load double, double* %Autodiff_cpp_616_11, align 8
%Autodiff_cpp_519_15 = load double, double* %Autodiff_cpp_616_9, align 8
%Autodiff_cpp_501_16 = fmul double %Autodiff_cpp_519_14, %Autodiff_cpp_519_15
%Autodiff_cpp_493_ = fadd double %Autodiff_cpp_501_, %Autodiff_cpp_501_16
%Autodiff_cpp_493_17 = fadd double %Autodiff_cpp_519_, %Autodiff_cpp_493_
%Autodiff_cpp_519_18 = load double, double* %Autodiff_cpp_616_3, align 8
%Autodiff_cpp_497_ = fsub double %Autodiff_cpp_493_17, %Autodiff_cpp_519_18
%Autodiff_cpp_317_19 = fmul double %Autodiff_cpp_497_, 2.000000e+00
%Autodiff_cpp_302_ = fmul double %Autodiff_cpp_519_14, %Autodiff_cpp_317_19
%Autodiff_cpp_302_23 = fmul double %Autodiff_cpp_519_12, %Autodiff_cpp_317_19
%7 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1539_
store double %Autodiff_cpp_302_, double* %7, align 8
%Autodiff_cpp_623_ = add i32 %CUDABuiltin_cpp_97_1, 32768
%CodeGen_cpp_1599_26 = zext i32 %Autodiff_cpp_623_ to i64
%8 = getelementptr inbounds double, double* %result, i64 %CodeGen_cpp_1599_26
store double %Autodiff_cpp_302_23, double* %8, align 8
ret void
```

Evaluation


NVIDIA
NVIDIA CORPORATION
Santa Clara, California, USA
HEATKILLER IV
made in Germany

Evaluation: Comparison Forward/Reverse Mode

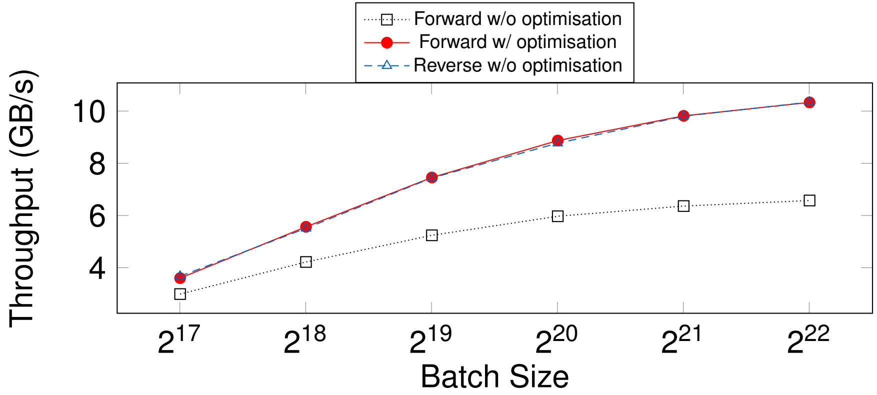


Figure: Execution time.

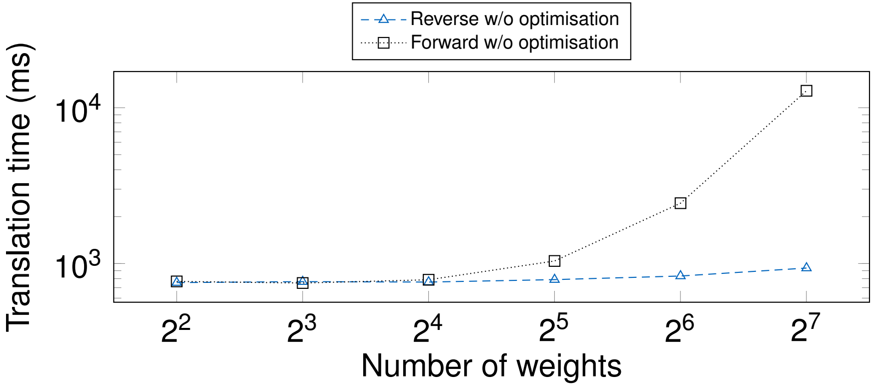
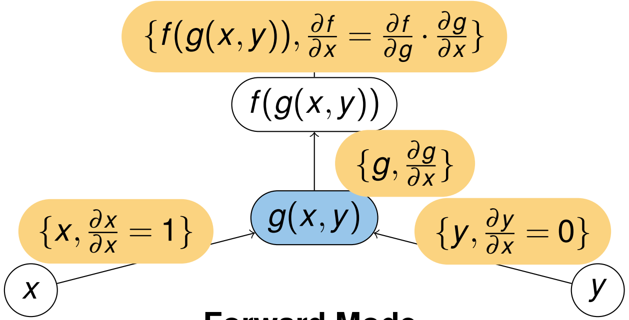


Figure: Compilation time (1 thread).

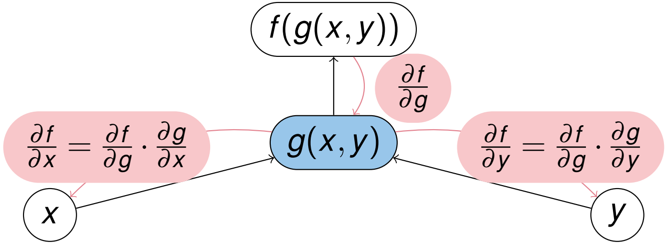
- *System*: NVIDIA GeForce RTX 3050 Ti Laptop, Intel Core i7-11800H.
- *Runtime*: noalias optimises the generated code, forward and reverse mode indeed lead to the same performance
- *Compilation time*: depends on number of variables

Conclusion



Forward Mode

- compilation time dependent on number of derivatives
- optimised execution time similar to reverse mode



Reverse Mode

- runtime and compilation time do not depend on number of derivatives
- well suited without optimisation

Future Work

- Automatic differentiation to SQL: use reverse mode
- Performance comparison also when training neural networks

Thank you for your attention!