einsum optimizer

is all you need

Einsum

```
einsum('ij,jk->ik', f1, f2)
```

$$\sum_j f1(i,j) f2(j,k) \rightarrow g(i,k)$$

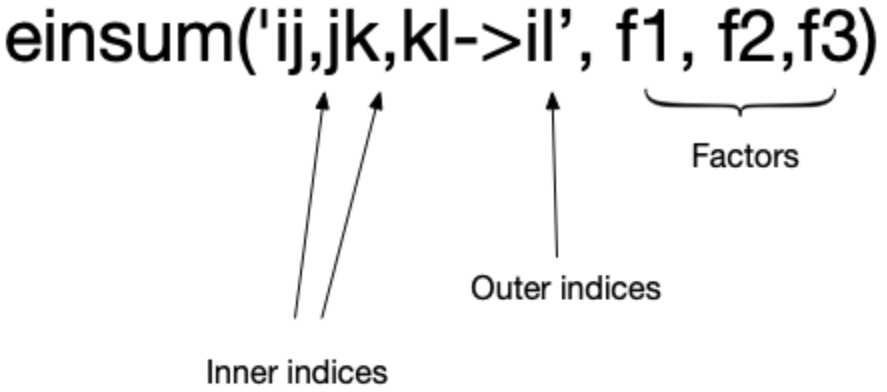
$f1, f2$ — Factors

i, j, k — Indices

Inner: j

Outer: i, k

Beyond original einstein summation

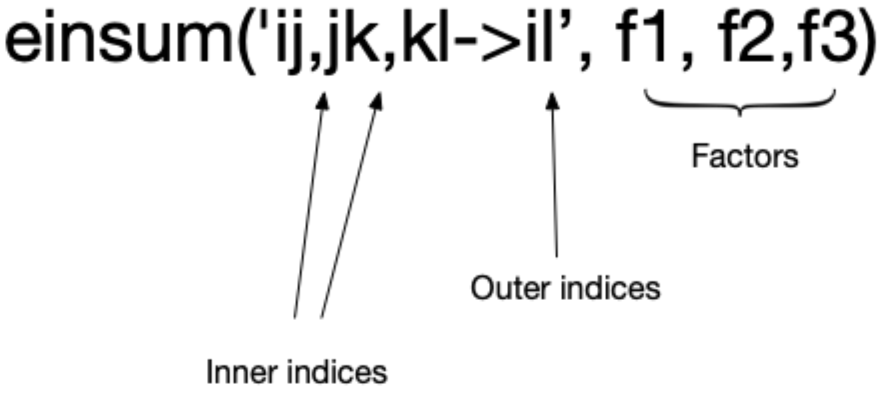


- outer indices can be empty

$$\sum_i f(i)$$

einsum(‘i->’, f)

Beyond original einstein summation

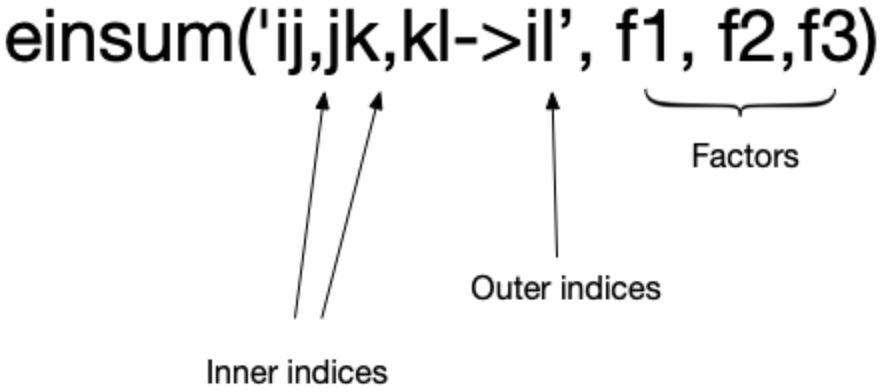


- inner indices can be empty

```
einsum('ij->ij', f)
```

Beyond original einstein summation

```
einsum('ij->ji', f)
```

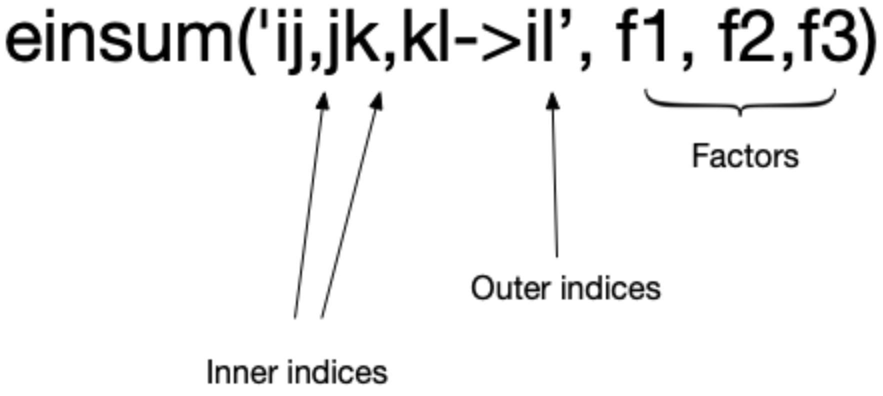


- order matters

einsum('ij->ji', f)

Beyond original einstein summation

```
einsum('ij->ji', f)
```



- outer indices can repeat on left side: einsum('ii->i', f)

Matmul as einsum

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

einsum('ij,j->i', A, x)

Convolution as einsum

a	b

1	2	3	4

$$\begin{pmatrix} a & b & 0 & 0 \\ 0 & a & b & 0 \\ 0 & 0 & a & b \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

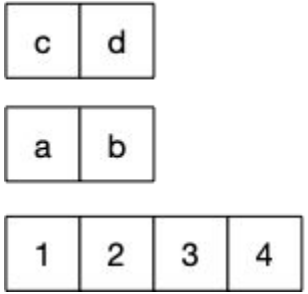
better memory

$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

better time

- need a linearly indexed view
- ie, "fold" or "im2col" operation

Factored convolutions



$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}$$

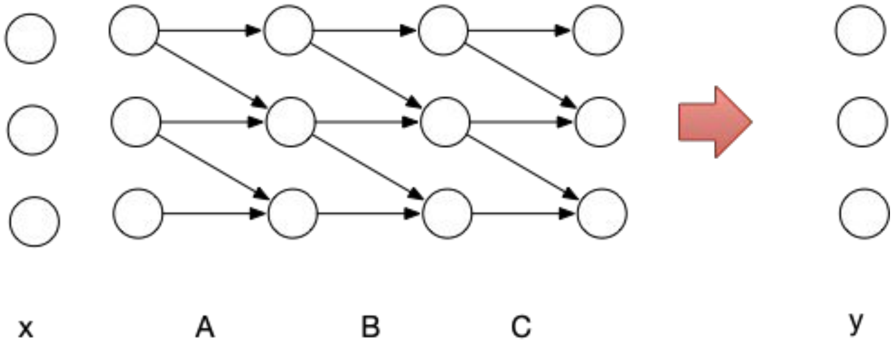
Factored convolutions

EINCONV: EXPLORING UNEXPLORED TENSOR NETWORK DECOMPOSITIONS FOR CONVOLUTIONAL NEURAL NETWORKS

- factored convolution
- depthwise convolution
- spatially separable convolution
- bottleneck layer
- 1x1 convolution

- 492 3D convolutions

Linear Neural Network

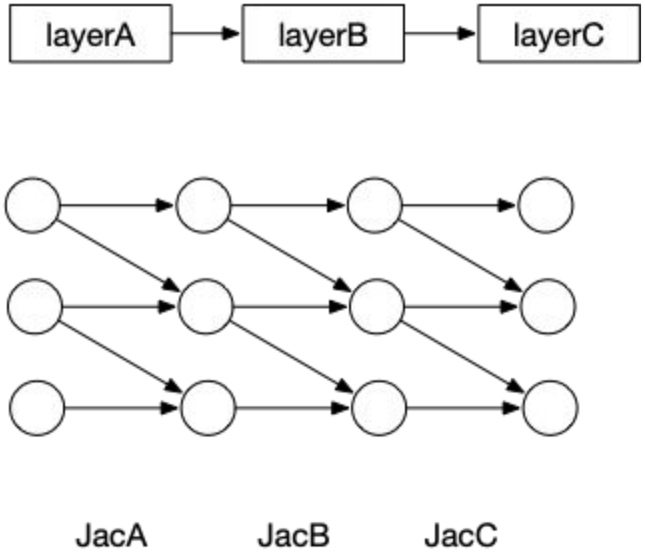


=sum over weighted walks

$y' = x'ABC$

einsum('i,ij,jk,kl->l', x, A, B, C)

Derivative



einsum(‘ij,jk,kl->i’, jacA, jacB, jacC)

Hessian

$$\sum_{a,b,c,d} (f1(a,b)f2(b,c)f3(c,d))'$$

$$\sum_{a,b,c,d} f1'(a,b)f2(b,c)f3(c,d)$$

$$\sum_{a,b,c,d} f1(a,b)f2'(b,c)f3(c,d)$$

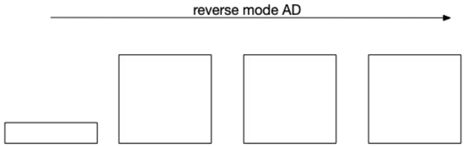
$$\sum_{a,b,c,d} f1(a,b)f2(b,c)f3'(c,d)$$

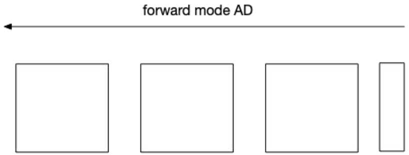
h1(a,b,i)=f1'(a,b) if i==1 else f1(a,b)

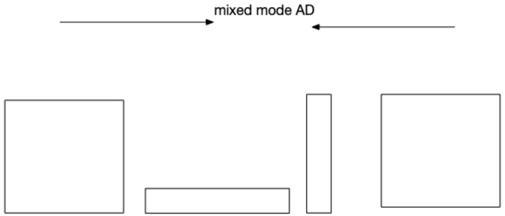
$$\sum_{a,b,c,d,i} h1(a,b,i)h2(b,c,i)h3(c,d,i)$$

Efficient computation: factoring

$$\sum_{a,b,c,d} f1(a,b)f2(b,c)f3(c,d) = \sum_a \sum_b f(a,b) \left(\sum_c f(b,c) \left(\sum_d f(c,d) \right) \right)$$





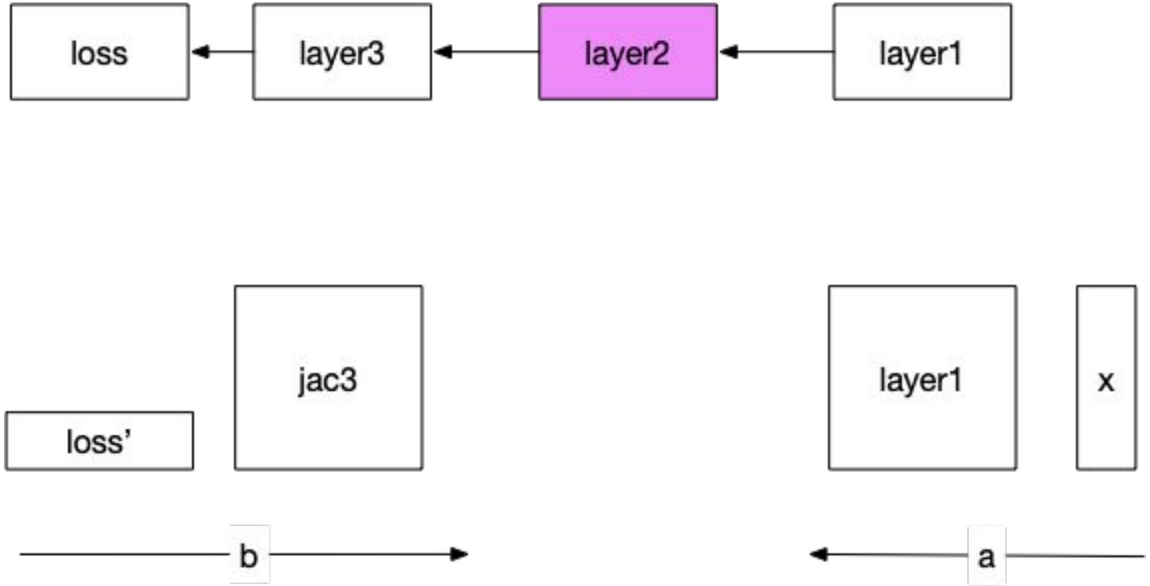


Efficient computation: factoring

$$\sum_{ijkl} f1(i) f2(j) f3(k) f4(l) = \sum_{i} f1(i) \sum_{j} f2(j) \sum_{k} f3(k) \sum_{l} f4(l)$$

https://colab.research.google.com/drive/1ItfFMp6WGdZLSrFtI-ppnVcD2ahLxIHg#scrollTo=ICV7-1SrEDAI

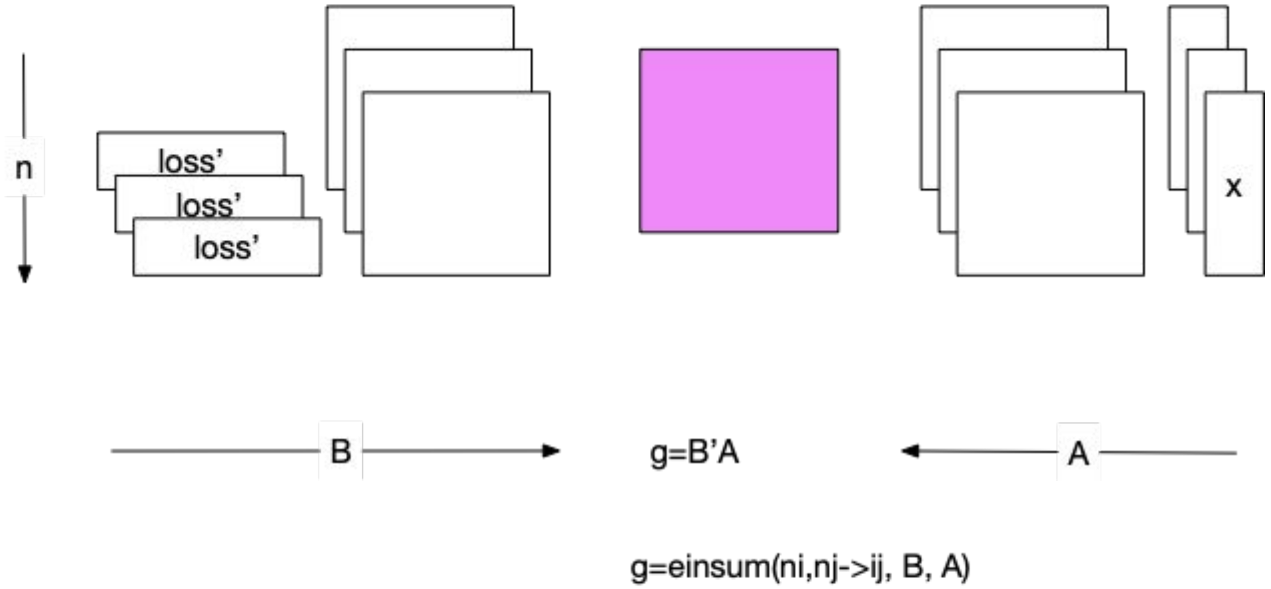
More gradients



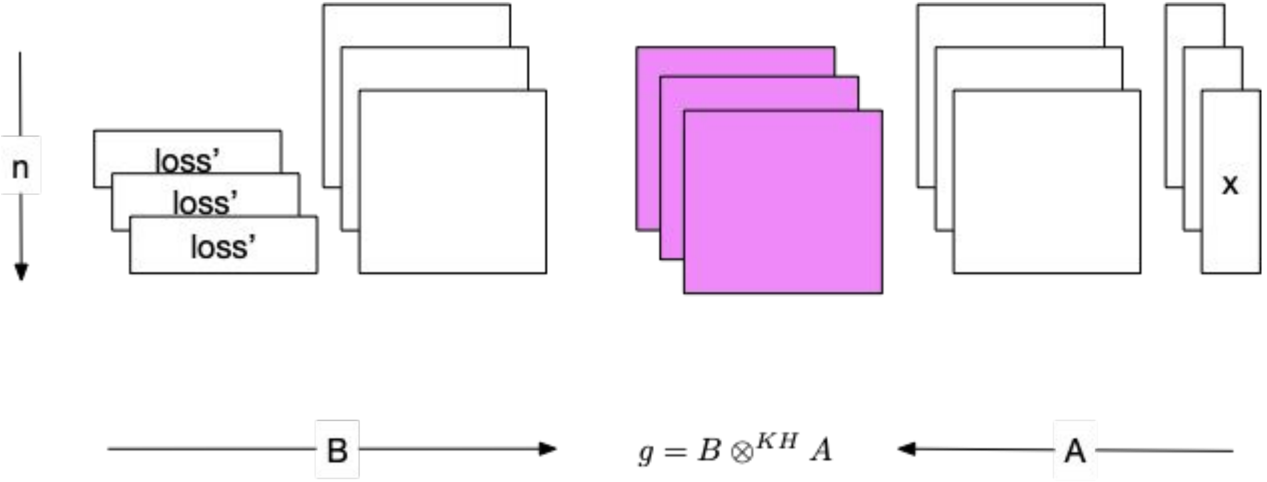
g=b'a

g=einsum(i,j->ij, b, a)

More gradients



Per example gradients



$$g = B \otimes^{KH} A$$

g=einsum(ni,nj->nij, B, A)

Gradient norms squared

per-example gradient norms squared

```
g=einsum(i,i,j,j->, b, b, a, a)
g=einsum(ni,ni,nj,nj->n, B, B, A, A)
```

stick into einsum optimizer => discover the trick from Goodfellow "**Efficient Per-Example Gradient Computations**"

```
(B*B).sum(dim=1) * (A*A).sum(dim=1)
```

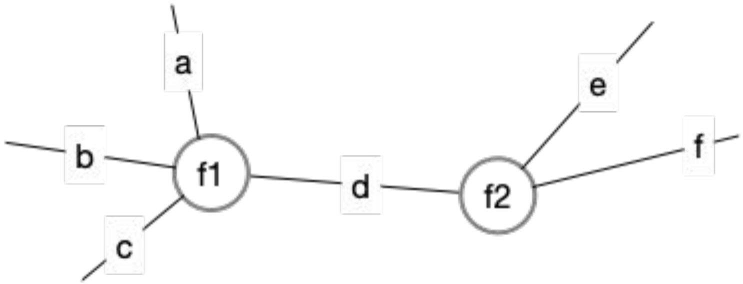
Gradient norms squared: conv layers?

- hessian trace
- gradient norms
- Hessian-vector products
- batch of per-example Hessian traces

https://colab.research.google.com/drive/16nKr_LmiiH8pgGkF1gNNahK83WVCpqk4#scrollTo=_E8yzItiWaUS

Graphical view

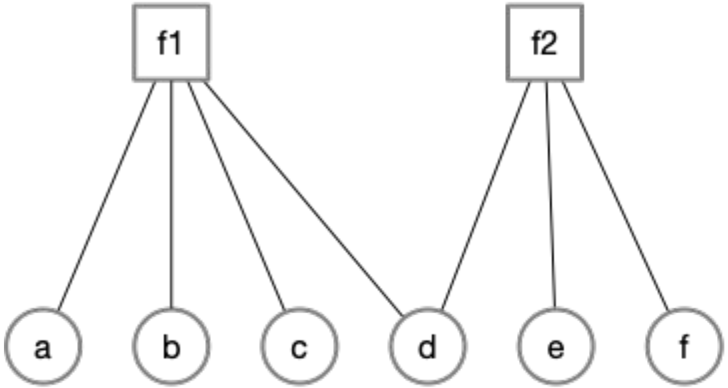
$$\text{einsum}(\text{"}\underbrace{abcd}_{\text{f1}},\underbrace{def}_{\text{f2}} -> \text{"})$$



factor vertices

needs hyper-edges

Graphical view

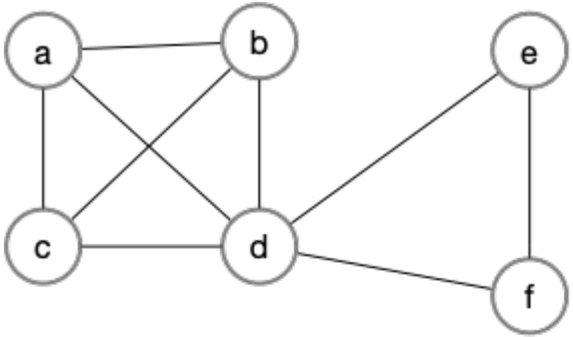


factor graph

terms are factor nodes
indices are vertex nodes

$$\text{einsum}(''\underbrace{abcd}_{\text{f1}},\underbrace{def}_{\text{f2}} -> '')$$

Graphical view



vertices connected if they share a factor

$$\text{einsum}(\text{"}\underbrace{abcd}_{\text{f1}},\underbrace{def}_{\text{f2}}->\text{"})$$

index vertices