Machine Learning - Exercise 4

Companion Slides

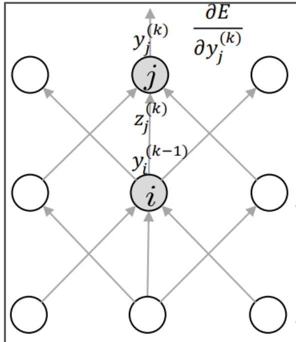
Ali Athar   Sabarinath Mahadevan

December 6, 2018

Visual Computing Institute
RWTH AACHEN UNIVERSITY

Exercise Goal

Lecture Recap

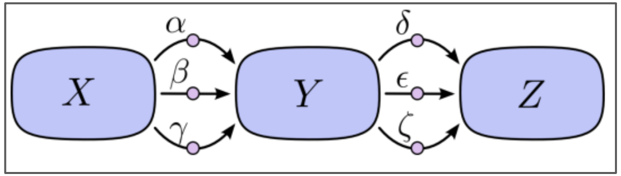


$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g\left(z_j^{(k)}\right)}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

Backpropagation for fixed network

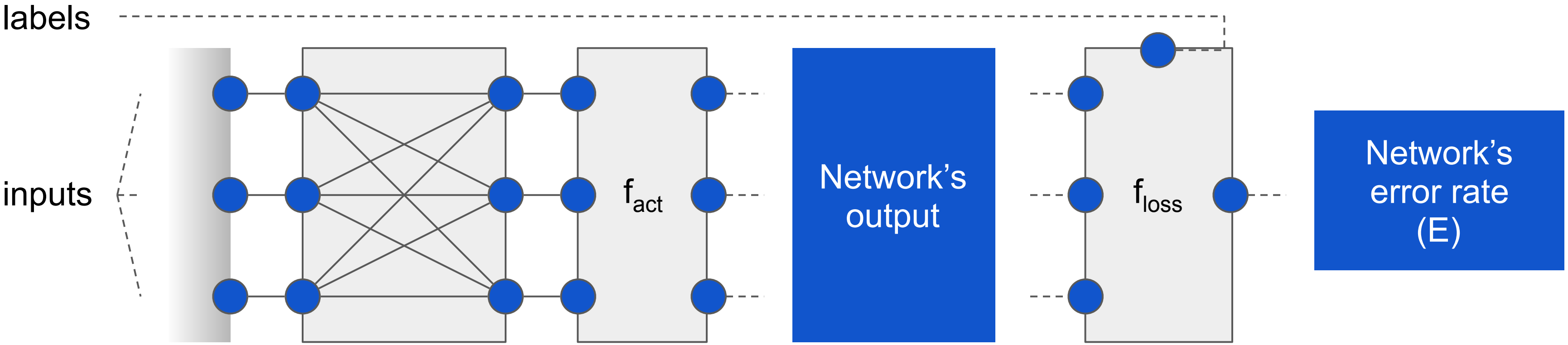


General backpropagation with computational graphs

This exercise is about

- Understanding backpropagation, deriving formulas, optimizing them
- Implement simple neural network framework yourself
- Digit recognition

Recap: Neural Networks



Linear module, Θ_i = (W, b)

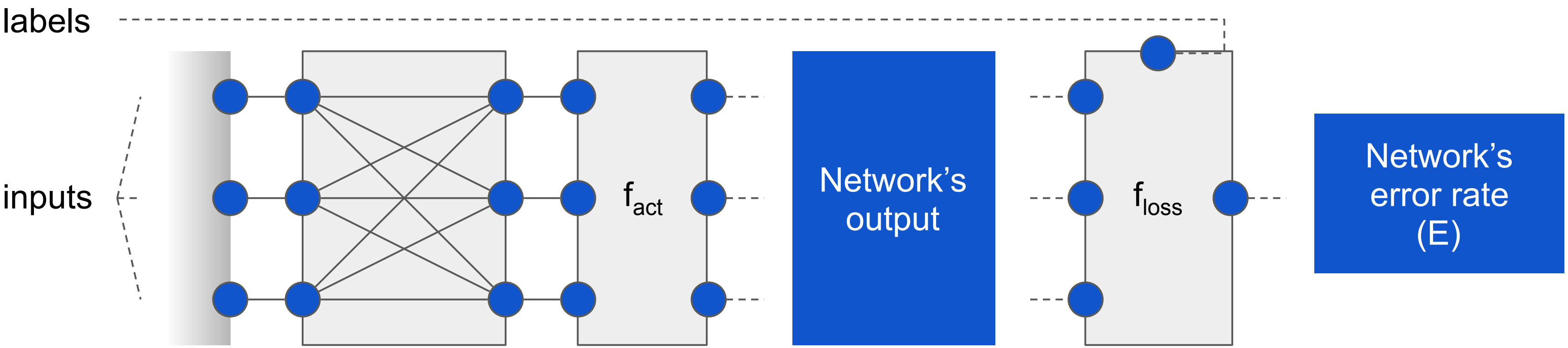
Activation function (tanh, σ, ReLu)

Parameters

- Training data (inputs) $X = \{x_i\}_{i=1..N}$ with $x_i \in \mathbb{I}$, N the batch size
- Training labels $T = \{t_i\}_{i=1..N}$ with $x_i \in \mathbb{O}$
- Network is a parametrized, (sub-)differentiable function $F(X,\Theta) : \mathbb{I} \times \mathbb{P} \rightarrow \mathbb{O}$
  - e.g., $\mathbb{O} = \mathbb{R}^{Dim}$ (regression), $\mathbb{O} = [0,1]^{Dim}$ (prob. classification)
- Loss (criterion) $\mathbf{L}(T,F(X,\Theta)) : \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{R}$, put on top of output to measure performance
  - find optimal parameters: $\Theta^* = \mathrm{argmin}_{\Theta}\ \mathbf{L}(T,F(X,\Theta))$

Visual Computing Institute

Recap: Backpropagation

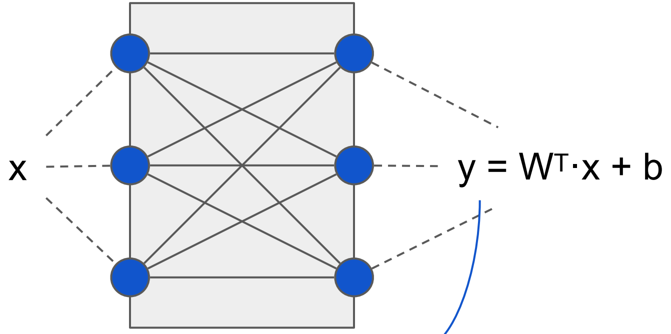


- Optimize towards lower error rate, i.e., lower E
  - Take derivative of E with respect to each modules parameters, follow gradient
  - Example: Gradient Descent: $\Theta = \Theta - \lambda * D_{\Theta}(E(x))$
  - $D_{\Theta}(E(x)) = D_{\Theta}(E)$ for brevity
- How to calculate $D_{\Theta}(E)$
  - Reverse order of modules
  - Module gets $D_{out}(E)$, calculates $D_{\Theta}(E)$, passes $D_{in}(E)$ to next module

Derivative w.r.t. modules parameters Θ at point x

Learning rate

Example: Linear/Fully Connected Module



$\Theta = (W, b)$

Element-wise:

$$y = \left[\sum_{i=1}^{N_{in}} (w_{ij} \cdot x_i) + b_j\right]_j$$

!

Without batching

Given: Derivative with respect to output $\frac{dE(x)}{dy}$

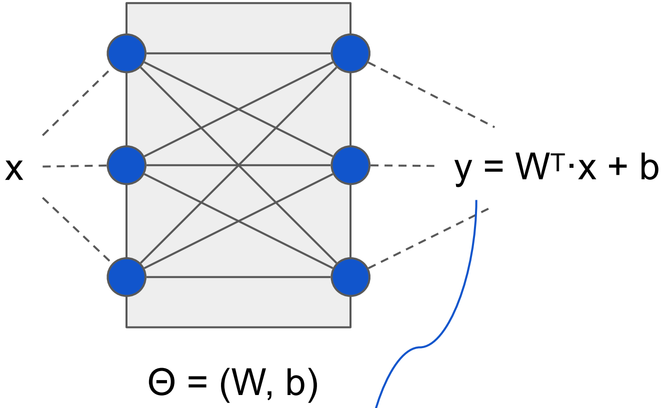
Calculate:

- Derivatives with respect to parameters Θ

$$\frac{dE(x)}{dw_{ij}} = \frac{dE(x)}{dy} \cdot \frac{dy}{dw_{ij}} = \frac{dE(x)}{dy_j} \cdot x_i = \left[x \cdot \frac{dE}{dy}\right]_{ij}$$

$$\frac{dE(x)}{db} = \frac{dE(x)}{dy} \cdot \frac{dy}{db} = \frac{dE(x)}{dy} \cdot I = \frac{dE(x)}{dy}$$

Example: Linear/Fully Connected Module



Element-wise:

$$y = \left[\sum_{i=1}^{N_{in}} (w_{ij} \cdot x_i) + b_j\right]_j$$

Without batching

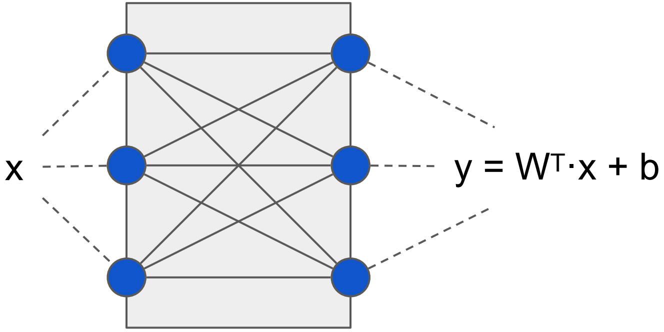
Given: Derivative with respect to output $\frac{dE(x)}{dy}$

Calculate:

- Derivative with respect to input

$$\frac{dE(x)}{dx} = \frac{dE(x)}{dy} \cdot \frac{dy}{dx} = \frac{dE(x)}{dy} \cdot W^T$$

Example: Linear/Fully Connected Module



$\Theta = (W, b)$

Putting it together:

```
fprop(x):
    cache.x = x
    return WT*x + b
```

run training data through (forwards)

```
bprop(dE):
        dW = cache.x * dE
        db = dE
        return dE * W
```

run gradients through (backwards)

```
update(rate):
        W = W - rate*dW
        b = b - rate*dbT
```

update the parameters (grad. descent)

Mini-Batching

- Batch learning
  - All training samples processed at once, parameters updated once at the end
  - Stable, well understood, many acceleration techniques, but slow
- Stochastic learning
  - Each training sample separately, parameters updated at each step
  - Noisy (though may lead to better results), fast
- Mini-batching
  - Middle ground, batches of data processed, bundled updates
  - Combine advantages, reduce drawbacks
- Example
  - Linear Module *f* with input dimension N_{in} and output dimension N_{out}, batch size *n*

$$f(x) = W^T \cdot \begin{bmatrix} | & | & & | \\ x^1 & x^2 & \dots & x^n \\ | & | & & | \end{bmatrix} + \begin{bmatrix} | & | & & | \\ b & b & \dots & b \\ | & | & & | \end{bmatrix}$$

broadcast (i.e., repeat) *b*

mini-batch matrix

Batching Update Rule

- (Mini-)Batch learning
  - Multiple samples processed at once
  - Calculate gradient for each sample, but don't update the parameters
  - After processing the batch, update using a sum of all gradients
  - Learning rate has to be adapted, e.g., divide E by batch size
  - Example: Gradient Descent

$$\Theta = \Theta - \lambda \cdot \sum_{k=1}^{N} D_\Theta(E(x^k))$$

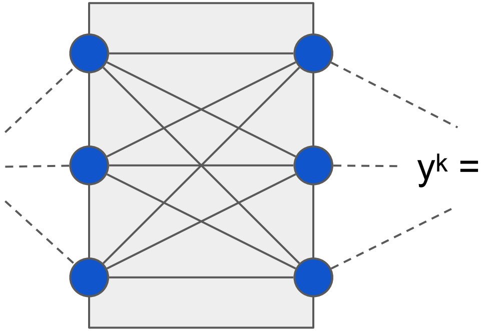
Derivative of E w.r.t parameters Θ at point x^k

  - To make things easier, we write

$$\sum_{k=1}^{N} D_\Theta(E(x^k)) = D_\Theta(E(x)) = D_\Theta(E)$$

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Example: Linear/Fully Connected Module - Batching



x^k $y^k = W^T \cdot x^k + b$

$\Theta = (W, b)$

k^{th} element in the batch

Deriv. w.r.t outputs assumed to be given row-wise:

$$\frac{dE(x)}{dy} = \begin{bmatrix} - & \frac{dE(x^1)}{dy^1} & - \\ & \cdots & \\ - & \frac{dE(x^n)}{dy^n} & - \end{bmatrix}$$

Given: Derivatives with respect to outputs $\frac{dE(x^k)}{dy^k}$

Plural!

Calculate:

- Derivatives with respect to parameters Θ

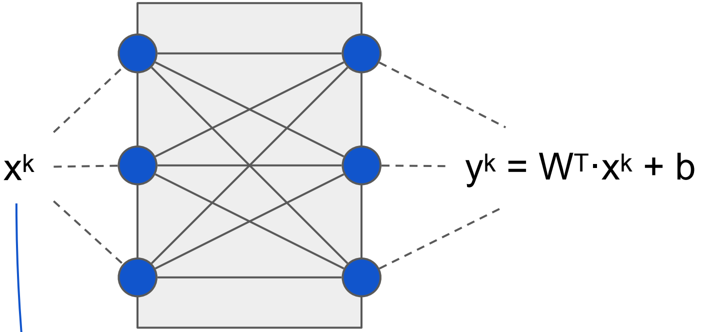
$$\frac{dE(x)}{dw_{ij}} = \sum_{k=1}^{N} \left(\frac{dE(x^k)}{dy^k} \cdot \frac{dy^k}{dw_{ij}} \right) = \left[\sum_{k=1}^{N} \left(x^k \cdot \frac{dE(x^k)}{dy^k} \right) \right]_{ij}$$

$$= \left[x \cdot \frac{dE(x)}{dy} \right]_{ij}$$

$$\frac{dE(x)}{db} = \sum_{k=1}^{N} \left(\frac{dE(x^k)}{dy^k} \cdot \frac{dy^k}{db} \right) = \sum_{k=1}^{N} \left(\frac{dE(x^k)}{dy^k} \right)$$

$$= sum_{col-wise} \left(\frac{dE(x)}{dy} \right)$$

Example: Linear/Fully Connected Module - Batching



$\Theta = (W, b)$

k^{th} element in the batch

Given: Derivatives with respect to outputs $\dfrac{dE(x^k)}{dy^k}$

Plural!

Calculate:

- Derivatives with respect to inputs

$$\frac{dE(x)}{dx} = \left[\frac{dE(x^k)}{dx^k}\right]_k = \left[\frac{dE(x^k)}{dy^k} \cdot \frac{dy^k}{dx^k}\right]_k$$

$$= \left[\frac{dE(x^k)}{dy^k} \cdot W^T\right]_k = \frac{dE(x)}{dy} \cdot W^T$$

Deriv. w.r.t outputs assumed to be given row-wise:

$$\frac{dE(x)}{dy} = \begin{bmatrix} - & \frac{dE(x^1)}{dy^1} & - \\ & \cdots & \\ - & \frac{dE(x^n)}{dy^n} & - \end{bmatrix}$$

Example: Training a Network

```
network = [module₁,module₂, …, moduleₙ], loss = f_loss

for X, T in batched(inputs, labels) do
        z = X
        for module in net do
                z = module.fprop(z)
        end for
        E = loss.fprop(z,T)
        dz = loss.bprop(1/batchSize)        // Normalization for batch size
        for module in reversed(net) do
                dz = module.brop(dz)
        end for
        for module in net do
                module.update(rate)
        end for
end for
```

Debugging Tip: Gradient Checking

Check the Jacobian *J* from *bprop* with numerical differentiation

- Numerical approach: Column-wise (here for the first column)

$$x_+ = (x_1 + h_1, x_2, ..., x_n) \qquad J_{\_,1} = \frac{fprop(x_+) - fprop(x_-)}{2h_1}$$
$$x_- = (x_1 - h_1, x_2, ..., x_n)$$

- Backprop: Row-wise (here for the first row)

$$fprop(x)$$
$$J_{1,\_} = bprop(1, 0, ..., 0)$$

- Advice
  - Use (small) random *x*

$$h_i = \sqrt{\epsilon} * max(x_i, 1)$$

Expected Results/Tips for MNIST

- [Linear(28x28, 10), Softmax]
  - should give ± 750 errors
- [Linear(28x28, 200), tanh, Linear(200,10), Softmax]
  - should give ± 250 errors
- Typical learning rates
  - $\lambda \in [0.1, 0.01]$
- Typical batch sizes
  - $N_B \in [100, 1000]$
- Weight initialization
  - $W \in \mathbb{R}^{MxN}$
  - $W \sim N\left(0, \sqrt{\frac{2}{M+N}}\right)$, i.e., sampled from normal distribution around 0 with deviation $\sqrt{\frac{2}{M+N}}$
  - $b = 0$
- Pre-process the data
  - Dividide values by 255 (= max pixel value)