Deep Learning on FPGAs
Introduction to Deep Learning

Sebastian Buschjäger

Technische Universität Dortmund - Fakultät Informatik - Lehrstuhl 8

October 21, 2017
Recap Computer Science Approach

- Technical Problem
- Mathematical Method
- Algorithm
- Implementation

Classification ✓
Perceptron ✓
Simple learning rule ✓
System and language ✓
Recap Data Mining

Important concepts:

- **Classification** is one data mining task
- **Training data** is used to define and solve the task
- **A Method** is a general approach / idea to solve a task
- **A algorithm** is a way to realise a method
- **A model** forms the extracted knowledge from data
- **Accuracy** measures the model quality given the data
Recap Perceptron classifier

A perceptron is a linear classifier $f : \mathbb{R}^d \rightarrow \{0, 1\}$ with

$$
\hat{f}(\vec{x}) = \begin{cases} 
+1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i \geq b \\
0 & \text{else}
\end{cases}
$$

For learning

1: $\vec{w} = rand(1, \ldots, d + 1)$
2: while ERROR do
3: for $(\vec{x}_i, y_i) \in D$ do
4: $\vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \hat{f}(\vec{x}_i))$
5: end for
6: end while
Homework

So Who did the homework?
Homework

So Who did the homework?
And How good was your prediction?

Some of my results:
0 vs 1: 99.9% accuracy
1 vs 2: 98.6% accuracy
3 vs 6: 98.8% accuracy
5 vs 6: 94.6% accuracy
8 vs 9: 97.4% accuracy

Runtime \( \sim 3 \) s per model with 100 runs over data

Machine: Laptop with Intel i7-4600U @ 2.10GHz, 8GB RAM

Tip: Compile with -O3 -march -mnative

DeepLearning on FPGAs
Homework

So Who did the homework?
And How good was your prediction?

Some of my results

- 0 vs 1: 99.9% accuracy
- 1 vs 2: 98.6% accuracy
- 3 vs 6: 98.8% accuracy
- 5 vs 6: 94.6% accuracy
- 8 vs 9: 97.4% accuracy

Runtime $\sim 3s$ per model with 100 runs over data

Machine Laptop with Intel i7-4600U @ 2.10GHz, 8GB RAM

Tip Compile with -O3 -march -mnative
Data Mining Features are important

Fact 1 State space grows exponentially with increasing dimension.
Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

For $\mathcal{X}^1$, there are 10 different observations
For $\mathcal{X}^2$, there are $10^2 = 100$ different observations
For $\mathcal{X}^3$, there are $10^3 = 1000$ different observations ...
Data Mining Features are important

Fact 1 State space grows exponentially with increasing dimension.
Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

For $\mathcal{X}^1$, there are 10 different observations
For $\mathcal{X}^2$, there are $10^2 = 100$ different observations
For $\mathcal{X}^3$, there are $10^3 = 1000$ different observations . . .

Fact 2 Training data is generated by a noisy real-world process
We usually have no influence on the type of training data
We usually cannot interfere with the real-world process
Data Mining Features are important

Fact 1 State space grows exponentially with increasing dimension.
Example $\mathcal{X} = \{1, 2, \ldots, 10\}$

For $\mathcal{X}^1$, there are $10$ different observations
For $\mathcal{X}^2$, there are $10^2 = 100$ different observations
For $\mathcal{X}^3$, there are $10^3 = 1000$ different observations . . .

Fact 2 Training data is generated by a noisy real-world process
We usually have no influence on the type of training data
We usually cannot interfere with the real-world process

Thus Training data should be considered incomplete and noisy
Data Mining Features are important (2)

Wolpert 1996 There is no free lunch
Every method has its advantages and disadvantages
Most methods are able to perfectly learn a given toy data set
Problem occurs with noise, outlier and generalisation
Data Mining Features are important (2)

Wolpert 1996 There is no free lunch
Every method has is advantages and disadvantages
Most methods are able to perfectly learn a given toy data set
Problem occurs with noise, outlier and generalisation

Conclusion All methods are equally good or bad
But Some methods prefer certain representations
Data Mining Features are important (2)

Wolpert 1996 There is no free lunch
Every method has is advantages and disadvantages
Most methods are able to perfectly learn a given toy data set
Problem occurs with noise, outlier and generalisation

Conclusion All methods are equally good or bad
But Some methods prefer certain representations

Feature Engineering Finding the right representation for data
Reduce dimension? Increase dimension?
Add additional information? Regularities?
Transform data completely?
Data Mining Features are important (3)

Raw data without transformation. Linear model is a bad choice. Parabola model would be better.

Data transformed with $\phi(x_1, x_2) = (x_1, x_2 - 0.3 \cdot x_1^2)$. Now linear model fits the problem.
Data Mining Features are important (4)

Conclusion: Good features are crucial for good results!
Question: How to get good features?
Data Mining  Features are important (4)

Conclusion: Good features are crucial for good results!
Question: How to get good features?

1 **By hand:** Domain experts and data miner examine the data and try different features based on common knowledge.
2 **Semi supervised:** Data miner examines the data and tries different similarity functions and classes of methods
3 **Unsupervised:** Data miner only encodes some assumptions about regularities into the method.
Data Mining  Features are important (4)

**Conclusion:** Good features are crucial for good results!

**Question:** How to get good features?

1. **By hand:** Domain experts and data miner examine the data and try different features based on common knowledge.
2. **Semi supervised:** Data miner examines the data and tries different similarity functions and classes of methods.
3. **Unsupervised:** Data miner only encodes some assumptions about regularities into the method.

**Note 1:** Hand-crafted features give us insight about the process
**Note 2:** Semi/unsupervised features give us insight about the data

**Our focus:** Unsupervised feature extraction.
Our Goal  End-to-End learning

Our focus  Unsupervised feature extraction
→ “End-To-End learning”
Our Goal End-to-End learning

Our focus Unsupervised feature extraction
→ “End-To-End learning”

So far
Deep Learning seems to be the best method

So…
What is Deep Learning?
Deep Learning Basics

Well... it's currently one of the big things in AI!

- **Since 2010**: DeepMind learns and plays old Atari games
- **Since 2012**: Google is able to find cats in youtube videos
- **December 2014**: Near real-time translation in Skype
- **October 2015**: AlphaGo beats the European Go champion
- **October 2015**: Tesla deploys Autopilot in their cars
- **March 2016**: AlphaGo beats the Go Worldchampion
- **June 2016**: Facebook introduces DeepText
- **August 2017**: Facebook uses neural-based translation
- ...
Deep Learning Example
Deep Learning Basics

Deep Learning is a branch of Machine Learning dealing with

- (Deep) Artificial Neural Networks (ANN)
- High Level Feature Processing
- Fast Implementations
Deep Learning Basics

Deep Learning is a branch of Machine Learning dealing with

- (Deep) Artificial Neural Networks (ANN)
- High Level Feature Processing
- Fast Implementations

ANNs are well known! So what’s new about it?

- We have more data and more computation power
- We have a better understanding of optimization
- We use a more engineering-style approach

Our focus now Artificial Neural Networks
Data Mining Model optimization

Important We need some basics about optimization

Recap

\[ \vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \hat{f}(\vec{x}_i)) \]
Data Mining Model optimization

Important We need some basics about optimization

Recap

\[ \vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \hat{f}(\vec{x}_i)) \]

So far We formulated an optimization algorithm to find perceptron weights that minimize classification error
Data Mining Model optimization

Important We need some basics about optimization

Recap

\[ \vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \hat{f}(\vec{x}_i)) \]

So far We formulated an optimization algorithm to find perceptron weights that minimize classification error

This is a common approach in Data Mining:

- Specify model family
- Specify optimization procedure
- Specify a cost / loss function
Data Mining Model optimization

Important We need some basics about optimization

Recap

\[ \vec{w} = \vec{w} + \alpha \cdot \vec{x}_i \cdot (y_i - \hat{f}(\vec{x}_i)) \]

So far We formulated an optimization algorithm to find perceptron weights that minimize classification error

This is a common approach in Data Mining:
- Specify model family
- Specify optimization procedure
- Specify a cost / loss function

Note: Loss function \( \neq \) Accuracy
- The loss function is minimized during learning
- Accuracy is used to measure the model’s quality after learning
Data Mining Stochastic gradient descent (SGD)

Given

A loss function $E$, the model parameter $\theta$, learning rate $\alpha_t$
Data Mining Stochastic gradient descent (SGD)

Given

A loss function $E$, the model parameter $\vec{\theta}$, learning rate $\alpha_t$

Framework

1: $\vec{\theta} = random()$
2: while ERROR do
3: choose random $(x, y) \in \mathcal{D}$
4: $\vec{\theta} = \vec{\theta} - \alpha_t \cdot \frac{\partial E(x,y)}{\partial \vec{\theta}}$
5: end while
Data Mining Stochastic gradient descent (SGD)

Given

A loss function $E$, the model parameter $\vec{\theta}$, learning rate $\alpha_t$

Framework

1: $\vec{\theta} = \text{random}()$
2: while ERROR do
3: choose random $(\vec{x}, y) \in D$
4: $\vec{\theta} = \vec{\theta} - \alpha_t \cdot \frac{\partial E(x,y)}{\partial \vec{\theta}}$
5: end while

e.g. 100 iterations
e.g. minimum change in $\theta$
Data Mining Stochastic gradient descent (SGD)

Given

A loss function $E$, the model parameter $\vec{\theta}$, learning rate $\alpha_t$

Framework

1: $\vec{\theta} = \text{random}()$
2: while ERROR do
3: choose random $(\vec{x}, y) \in \mathcal{D}$
4: $\vec{\theta} = \vec{\theta} - \alpha_t \cdot \frac{\partial E(x,y)}{\partial \vec{\theta}}$
5: end while

(estimated) gradient of loss depends on $\theta$ and $(x, y)$

e.g. 100 iterations

e.g. minimum change in $\theta$
**Data Mining** Perceptron Learning

**Observation** We implicitly did this for the perceptron

1: \( \vec{w} = \text{rand}(1, \ldots, d + 1) \)
2: **while** ERROR **do**
3: **for** \((\vec{x}, y) \in \mathcal{D}\) **do**
4: \( \vec{w} = \vec{w} + \alpha \cdot \vec{x} \cdot (y - \hat{f}(\vec{x})) \)
5: **end for**
6: **end while**

**So** The perceptron works well and follows a general framework
Data Mining The XOR Problem

Question What happens if data is not linear separable?
Data Mining  The XOR Problem

Question  What happens if data is not linear separable?

Data linear separable, but noisy  Data not linear separable

Algorithm will never converge, thus
Use fixed number of iterations
Introduce some acceptable error margin
**Data Mining** The XOR Problem

**Question** What happens if data is not linear separable?

![Diagram showing linear separable data with noise and non-linear separable data](image)

- **Data linear separable, but noisy**
- **Data not linear separable**

**Answer** Algorithm will never converge, thus:

- Use fixed number of iterations
- Introduce some acceptable error margin
Data Mining Idea - use more perceptrons

Recap (Hand crafted) Feature transformation always possible
But What about an automatic way?
Rosenblatt 1961
Use multiple perceptrons → Multi-Layer Perceptron (MLP)
Data Mining  MLP learning

**Goal** We need to learn weights $w$ / bias $b$ for each perceptron

**So far** We intuitively derived a learning algorithm

Loss function (MSE)

$$\ell(D, \hat{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{f}(\vec{x}_i))^2$$

Observation

We need to take the derivative of the loss function

But Loss functions looks complicated

Observation 1

Square-Root is monotone

Observation 2

Constant factor does not change optimization
Data Mining MLP learning

**Goal** We need to learn weights $w$ / bias $b$ for each perceptron

**So far** We intuitively derived a learning algorithm

**Now** Follow stochastic gradient descent algorithm

**Loss function (MSE)**

$$
\ell(D, \hat{w}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{f}(\vec{x}_i))^2}
$$

**Observation** We need to take the derivative of the loss function
Data Mining MLP learning

Goal We need to learn weights $w$ / bias $b$ for each perceptron
So far We intuitively derived a learning algorithm
Now Follow stochastic gradient descent algorithm
Loss function (MSE)

$$\ell(D, \hat{w}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left( y_i - \hat{f}(\vec{x}_i) \right)^2}$$

Observation We need to take the derivative of the loss function
But Loss functions looks complicated
Observation 1 Square-Root is monotone
Observation 2 Constant factor does not change optimization
Data Mining MLP learning (2)

New loss function

\[ \ell(D, \hat{w}) = \frac{1}{2} \left( y_i - \hat{f}(\vec{x}_i) \right)^2 \]

\[ \nabla_{\hat{w}} \ell(D, \hat{w}) = \frac{1}{2} 2(y_i - \hat{f}(\vec{x}_i)) \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}} \]
Data Mining MLP learning (2)

New loss function

\[ \ell(D, \hat{w}) = \frac{1}{2} \left( y_i - \hat{f}(\vec{x}_i) \right)^2 \]

\[ \nabla_{\hat{w}} \ell(D, \hat{w}) = \frac{1}{2} 2(y_i - \hat{f}(\vec{x}_i)) \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}} \]

Observation We need to compute derivative \( \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}} \)
Data Mining  MLP learning (2)

New loss function

\[
\ell(D, \hat{w}) = \frac{1}{2} \left( y_i - \hat{f}(\vec{x}_i) \right)^2
\]

\[
\nabla_{\hat{w}} \ell(D, \hat{w}) = \frac{1}{2} 2(y_i - \hat{f}(\vec{x}_i)) \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}}
\]

Observation  We need to compute derivative \( \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}} \)

\[
\hat{f}(\vec{x}) = \begin{cases} 
+1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i + b \geq 0 \\
0 & \text{else}
\end{cases}
\]
Data Mining MLP learning (2)

New loss function

\[
\ell(D, \hat{w}) = \frac{1}{2} \left( y_i - \hat{f}(\vec{x}_i) \right)^2
\]

\[
\nabla_{\hat{w}} \ell(D, \hat{w}) = \frac{1}{2} 2(y_i - \hat{f}(\vec{x}_i)) \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}}
\]

Observation We need to compute derivative \( \frac{\partial \hat{f}(\vec{x}_i)}{\partial \hat{w}} \)

\[
\hat{f}(\vec{x}) = \begin{cases} 
+1 & \text{if } \sum_{i=1}^{d} w_i \cdot x_i + b \geq 0 \\
0 & \text{else}
\end{cases}
\]

Observation \( f \) is not continuous in 0 (it makes a step)

Thus Impossible to derive \( \nabla_{\hat{w}} \ell(D, w) \) in 0, because \( f \) is not differentiable in 0!
Data Mining MLP activation function

Another problem Combinations of linear functions are still linear

\[ f(x) = 5x + 3 \]
\[ g(x) = 10x_1 - 5x_2 \]
\[ f(g(x)) = 5(10x_1 - 5x_2) + 3 = 50x_1 - 25x_2 + 3 \]

Solution
We need to make \( f \) continuous
We need to introduce some non-linearity
Data Mining MLP activation function

Another problem Combinations of linear functions are still linear

\[ f(x) = 5x + 3 \]
\[ g(x) = 10x_1 - 5x_2 \]
\[ f(g(x)) = 5(10x_1 - 5x_2) + 3 = 50x_1 - 25x_2 + 3 \]

Solution
We need to make \( f \) continuous
We need to introduce some non-linearity

Observation
The input of a perceptron depends on the output of previous one

Thus
Apply non-linear activation function to perceptron output
Data Mining MLP activation function (2)

**Bonus** This seems to be a little closer to real neurons

**Constraint** Activation should be easy to compute

**Idea** Use sigmoid function

\[
\sigma(z) = \frac{1}{1 + e^{-\beta z}}, \beta \in \mathbb{R}_{>0}
\]

**Note** $\beta$ controls slope around 0
Data Mining Sigmoid derivative

Given \( \sigma(z) = \frac{1}{1+e^{-\beta \cdot z}}, \beta \in \mathbb{R}_{>0} \)
Data Mining Sigmoid derivative

Given $\sigma(z) = \frac{1}{1+e^{-\beta z}}$, $\beta \in \mathbb{R}_{>0}$

Derivative

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \left( 1 + e^{-\beta z} \right)^{-1} = (-1) \left( 1 + e^{-\beta z} \right)^{-2} (-\beta) e^{-\beta z}$$
Data Mining  Sigmoid derivative

Given $\sigma(z) = \frac{1}{1 + e^{-\beta z}}$, $\beta \in \mathbb{R}_{>0}$

Derivative

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \left(1 + e^{-\beta z}\right)^{-1} = (-1) \left(1 + e^{-\beta z}\right)^{-2} (-\beta) e^{-\beta z}$$

$$= \frac{\beta e^{-\beta z}}{(1 + e^{-\beta z})^2} = \beta \frac{e^{-\beta z}}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}$$
Data Mining Sigmoid derivative

Given \( \sigma(z) = \frac{1}{1+e^{-\beta z}}, \beta \in \mathbb{R}_{>0} \)

Derivative

\[
\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \left(1 + e^{-\beta z}\right)^{-1} = (-1) \left(1 + e^{-\beta z}\right)^{-2} (-\beta) e^{-\beta z}
\]

\[
= \frac{\beta e^{-\beta z}}{(1 + e^{-\beta z})^2} = \beta \frac{e^{-\beta z}}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}
\]

\[
= \beta \frac{e^{-\beta z} + 1 - 1}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}
\]
Data Mining Sigmoid derivative

Given $\sigma(z) = \frac{1}{1+e^{-\beta z}}$, $\beta \in \mathbb{R}_{>0}$

Derivative

\[
\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \left( 1 + e^{-\beta z} \right)^{-1} = (-1) \left( 1 + e^{-\beta z} \right)^{-2} (-\beta) e^{-\beta z}
\]

\[
= \frac{\beta e^{-\beta z}}{(1 + e^{-\beta z})^2} = \beta \frac{e^{-\beta z}}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}
\]

\[
= \beta \left( \frac{1 + e^{-\beta z}}{1 + e^{-\beta z}} - \frac{1}{1 + e^{-\beta z}} \right) \frac{1}{1 + e^{-\beta z}}
\]
Data Mining Sigmoid derivative

Given \( \sigma(z) = \frac{1}{1+e^{-\beta z}}, \beta \in \mathbb{R}_{>0} \)

Derivative

\[
\frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \left(1 + e^{-\beta z}\right)^{-1} = (-1) \left(1 + e^{-\beta z}\right)^{-2} (-\beta) e^{-\beta z}
\]

\[
= \frac{\beta e^{-\beta z}}{(1 + e^{-\beta z})^2} = \beta \frac{e^{-\beta z}}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}
\]

\[
= \beta \frac{e^{-\beta z} + 1 - 1}{1 + e^{-\beta z}} \frac{1}{1 + e^{-\beta z}}
\]

\[
= \beta \left( \frac{1 + e^{-\beta z}}{1 + e^{-\beta z}} - \frac{1}{1 + e^{-\beta z}} \right) \frac{1}{1 + e^{-\beta z}}
\]

\[
= \beta(1 - \sigma(z)) \sigma(z)
\]
Data Mining MLP activation function

For inference We compute $\sigma(z)$
For training  We compute $\beta \sigma(z)(1 - \sigma(z))$
Thus Store activation $\sigma(z)$ for fast computation
Data Mining MLP activation function

**For inference** We compute $\sigma(z)$

**For training** We compute $\beta \sigma(z)(1 - \sigma(z))$

Thus Store activation $\sigma(z)$ for fast computation

**Note** Binary classification assumes $\mathcal{Y} = \{0, +1\}$

Thus Output perceptron also needs sigmoid activation

**But** For different labels (e.g. $\{-1, +1\}$) use another activation
Data Mining  MLP activation function

For inference  We compute $\sigma(z)$
For training  We compute $\beta \sigma(z)(1 - \sigma(z))$
Thus Store activation $\sigma(z)$ for fast computation

Note  Binary classification assumes $Y = \{0, +1\}$
Thus  Output perceptron also needs sigmoid activation
But  For different labels (e.g. $\{-1, +1\}$) use another activation

Still
We need to compute $\frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial E(x,y)}{\partial b_j^{(l)}}$
Thus  We need a more notation
MLPs A more detailed view

\[ w_{i,j}^{(l+1)} \cong \text{Weight from neuron } i \text{ in layer } l \text{ to neuron } j \text{ in layer } l + 1 \]

\[ f_j^{(l+1)} = h(\sum_{i=0}^{M(l)} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}) \]
Towards learning MLPs

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial E(x,y)}{\partial b_{j}^{(l)}}$.
Towards learning MLPs

Goal
We need to compute \( \frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}} \) and \( \frac{\partial E(x,y)}{\partial b_j^{(l)}} \)

Recap Chain-Rule

\[ \frac{\partial}{\partial x} (3x + 5)^2 = \]
Towards learning MLPs

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial E(x,y)}{\partial b_{j}^{(l)}}$

Recap Chain-Rule

$$\frac{\partial}{\partial x} (3x + 5)^2 = 2 \cdot (3x + 5) \cdot 3$$
Towards learning MLPs

Goal
We need to compute \( \frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}} \) and \( \frac{\partial E(x,y)}{\partial b_j^{(l)}} \)

Recap Chain-Rule

\[
\frac{\partial}{\partial x} (3x + 5)^2 = 2 \cdot (3x + 5) \cdot 3 = 6(3x + 5)
\]
Towards learning MLPs

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w^{(l)}_{i,j}}$ and $\frac{\partial E(x,y)}{\partial b^{(l)}_j}$

Recap Chain-Rule

$$\frac{\partial}{\partial x} (3x + 5)^2 = 2 \cdot (3x + 5) \cdot 3 = 6(3x + 5)$$

More formally
Given two functions $f : \mathbb{R}^m \rightarrow \mathbb{R}$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}^m$. Let $\vec{u} = g(\vec{x})$ and $\vec{x} \in \mathbb{R}^k$:

$$\frac{\partial f(g(\vec{x}))}{\partial x_i} = \frac{\partial f(\vec{u})}{\partial x_i} = \sum_{l=1}^{m} \frac{\partial f(\vec{u})}{\partial u_l} \cdot \frac{\partial u_l}{\partial x_i}$$
Towards backpropagation (1)

**Goal**

We need to compute $\frac{\partial E(x,y)}{\partial w^{(l)}_{i,j}}$ and $\frac{\partial E(x,y)}{\partial b^{(l)}_{j}}$
Towards backpropagation (1)

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial E(x,y)}{\partial b_{j}^{(l)}}$

Recall
$$y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_{i}^{(l)} + b_{j}^{(l+1)} \text{ and } f_{j}^{(l+1)} = h\left(y_{j}^{(l+1)}\right)$$
Towards backpropagation (1)

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w^{(l)}_{i,j}}$ and $\frac{\partial E(x,y)}{\partial b^{(l)}_j}$

Recall
$y^{(l+1)}_j = \sum_{i=0}^{M^{(l)}} w^{(l+1)}_{i,j} f^{(l)}_i + b^{(l+1)}_j$ and $f^{(l+1)}_j = h(y^{(l+1)}_j)$

Observation
$E$ depends on all $f^L_j$, which depends on $f^{L-1}_j$ . . .

$$\frac{\partial E}{\partial w^l_{i,j}} = \frac{\partial E}{\partial f^l_j} \cdot \frac{\partial f^l_j}{\partial y^l_j} \cdot \frac{\partial y^l_j}{\partial w^l_{i,j}}$$
Towards backpropagation (1)

Goal
We need to compute $\frac{\partial E(x,y)}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial E(x,y)}{\partial b_{j}^{(l)}}$

Recall
$y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_{i}^{(l)} + b_{j}^{(l+1)}$ and $f_{j}^{(l+1)} = h(y_{j}^{(l+1)})$

Observation
$E$ depends on all $f_{j}^{L}$, which depends on $f_{j}^{L-1}$ . . .

$$\frac{\partial E}{\partial w_{i,j}^{l}} = \frac{\partial E}{\partial f_{j}^{l}} \cdot \frac{\partial f_{j}^{l}}{\partial y_{j}^{l}} \cdot \frac{\partial y_{j}^{l}}{\partial w_{i,j}^{l}}$$

Contains all derivatives from $L$ to $l$
Backpropagation for $w_{i,j}^l$

Recall $y_j^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h \left( y_j^{(l+1)} \right)$
Backpropagation for $w_{i,j}^l$

Recall $y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_{i}^{(l)} + b_{j}^{(l+1)}$ and $f_{j}^{(l+1)} = h \left( y_{j}^{(l+1)} \right)$

$$\frac{\partial E}{\partial w_{i,j}^l} =$$
Backpropagation for $w_{i,j}^l$

Recall $y_j^{(l+1)} = \sum_{i=0}^{M(l)} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h\left(y_j^{(l+1)}\right)$

$$\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l}$$
Backpropagation for $w_{i,j}^l$

Recall $y_j^{(l+1)} = \sum_{i=0}^{M(l)} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h\left(y_j^{(l+1)}\right)$

$$\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l} = \left(\sum_{i=0}^{M(l+1)} \frac{\partial E}{\partial f_i^{(l+1)}} \cdot \frac{\partial f_i^{(l+1)}}{\partial y_i^{(l+1)}} \cdot \frac{\partial y_i^{(l+1)}}{\partial f_j^l}\right) \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l}$$
Backpropagation for $w_{i,j}^l$

Recall $y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_{i}^{(l)} + b_{j}^{(l+1)}$ and $f_{j}^{(l+1)} = h \left(y_{j}^{(l+1)}\right)$

$$\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_{j}^l} \cdot \frac{\partial f_{j}^l}{\partial y_{j}^l} \cdot \frac{\partial y_{j}^l}{\partial w_{i,j}^l} = \left(\sum_{i=0}^{M^{(l+1)}} \frac{\partial E}{\partial f_{i}^{(l+1)}} \cdot \frac{\partial f_{i}^{(l+1)}}{\partial y_{i}^{(l+1)}} \cdot \frac{\partial y_{i}^{(l+1)}}{\partial f_{j}^{(l+1)}} \cdot \frac{\partial f_{j}^{(l)}}{\partial y_{j}^{(l)}} \cdot \frac{\partial y_{j}^{(l)}}{\partial w_{i,j}^l}\right)$$

Recursion with $\delta_{i}^{(l+1)} = \frac{\partial E}{\partial f_{i}^{(l+1)}} \frac{\partial f_{i}^{(l+1)}}{\partial y_{i}^{(l+1)}}$
Backpropagation for $w_{i,j}^l$

Recall $y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_{i}^{(l)} + b_{j}^{(l+1)}$ and $f_{j}^{(l+1)} = h \left( y_{j}^{(l+1)} \right)$

$$\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_{j}^l} \cdot \frac{\partial f_{j}^l}{\partial y_{j}^l} \cdot \frac{\partial y_{j}^l}{\partial w_{i,j}^l} = \left( \sum_{i=0}^{M^{(l)}} \frac{\partial E}{\partial f_{i}^{(l+1)}} \cdot \frac{\partial f_{i}^{(l+1)}}{\partial y_{i}^{(l+1)}} \cdot \frac{\partial y_{i}^{(l+1)}}{\partial f_{j}^l} \right) \frac{\partial f_{j}^l}{\partial y_{j}^l} \cdot \frac{\partial y_{j}^l}{\partial w_{i,j}^l}$$

recursion with $\delta_{i}^{(l+1)} = \frac{\partial E}{\partial f_{i}^{(l+1)}} \frac{\partial f_{i}^{(l+1)}}{\partial y_{i}^{(l+1)}}$

$$= \left( \sum_{i=0}^{M^{(l)}} \delta_{i}^{(l+1)} \cdot \frac{\partial y_{i}^{(l+1)}}{\partial f_{j}^l} \right) \frac{\partial f_{j}^l}{\partial y_{j}^l} \cdot \frac{\partial y_{j}^l}{\partial w_{i,j}^l}$$
Backpropagation for $w_{i,j}^l$

Recall $y_{j}^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h\left(y_j^{(l+1)}\right)$

\[
\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l} = \left(\sum_{i=0}^{M^{(l+1)}} \frac{\partial E}{\partial f_i^{(l+1)}} \cdot \frac{\partial f_i^{(l+1)}}{\partial y_i^{(l+1)}} \cdot \frac{\partial y_i^{(l+1)}}{\partial f_j^l} \right) \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l}
\]

recursion with $\delta_i^{(l+1)} = \frac{\partial E}{\partial f_i^{(l+1)}} \frac{\partial f_i^{(l+1)}}{\partial y_i^{(l+1)}}$
Backpropagation for $w_{i,j}^l$

Recall $y_j^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h(y_j^{(l+1)})$

$$
\frac{\partial E}{\partial w_{i,j}^l} = \frac{\partial E}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l} = \left( \sum_{i=0}^{M^{(l+1)}} \frac{\partial E}{\partial f_i^{(l+1)}} \cdot \frac{\partial f_i^{(l+1)}}{\partial y_i^{(l+1)}} \cdot \frac{\partial y_i^{(l+1)}}{\partial f_j^l} \right) \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l}
$$

Recursion with $\delta_i^{(l+1)} = \frac{\partial E}{\partial f_i^{(l+1)}} \frac{\partial f_i^{(l+1)}}{\partial y_i^{(l+1)}}$

$$
= \left( \sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{i,j}^{(l+1)} \right) \frac{\partial f_j^l}{\partial y_j^l} \cdot f_i^{(l-1)}
$$
Backpropagation for $w_{i,j}^l$

Recall $y_j^{(l+1)} = \sum_{i=0}^{M^{(l)}} w_{i,j}^{(l+1)} f_i^{(l)} + b_j^{(l+1)}$ and $f_j^{(l+1)} = h \left( y_j^{(l+1)} \right)$

$$\frac{\partial E}{\partial w_{i,j}^l} \equiv \frac{\partial E}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l} = \left( \sum_{i=0}^{M^{(l+1)}} \frac{\partial E}{\partial f_i^{(l+1)}} \cdot \frac{\partial f_i^{(l+1)}}{\partial y_i} \cdot \frac{\partial y_i^{(l+1)}}{\partial f_j^l} \right) \frac{\partial f_j^l}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{i,j}^l}$$

Recursion with $\delta_i^{(l+1)} = \frac{\partial E}{\partial f_i^{(l+1)}} \frac{\partial f_i^{(l+1)}}{\partial y_i}$

$$= \left( \sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{i,j}^{(l+1)} \right) \frac{\partial f_j^l}{\partial y_j^l} \cdot f_i^{(l-1)} = \delta_j^{(l)} \cdot f_i^{(l-1)}$$

with $\delta_j^{(l)} = \left( \sum_{i=0}^{M^{(l+1)}} \delta_i^{(l+1)} \cdot w_{i,j}^{(l+1)} \right) \frac{\partial f_j^l}{\partial y_j^l}$
Backpropagation for $b^l_j$

Recall $y^{(l+1)}_j = \sum_{i=0}^{M^{(l)}} w^{(l+1)}_{i,j} f^{(l)}_i + b^{(l+1)}_j$ and $f^{(l+1)}_j = h\left(y^{(l+1)}_j\right)$

$$\frac{\partial E}{\partial b^l_j} = \frac{\partial E}{\partial f^l_j} \cdot \frac{\partial f^l_j}{\partial y^l_j} \cdot \frac{\partial y^l_j}{\partial b^l_j} = \left(\sum_{i=0}^{M^{(l+1)}} \frac{\partial E}{\partial f^{(l+1)}_i} \cdot \frac{\partial f^{(l+1)}_i}{\partial y^{(l+1)}_i} \cdot \frac{\partial y^{(l+1)}_i}{\partial f^l_j} \right) \frac{\partial f^l_j}{\partial y^l_j} \cdot \frac{\partial y^l_j}{\partial b^l_j}$$

Recursion with $\delta^{(l+1)}_i = \frac{\partial E}{\partial f^{(l+1)}_i} \cdot \frac{\partial f^{(l+1)}_i}{\partial y^{(l+1)}_i}$

$$= \left(\sum_{i=0}^{M^{(l+1)}} \delta^{(l+1)}_i \cdot w^{(l+1)}_{i,j}\right) \frac{\partial f^l_j}{\partial y^l_j} \cdot 1 = \delta^l_j \cdot 1$$

with $\delta^l_j = \left(\sum_{i=0}^{M^{(l+1)}} \delta^{(l+1)}_i \cdot w^{(l+1)}_{i,j}\right) \cdot \frac{\partial f^l_j}{\partial y^l_j}$
Backpropagation for activation $h$ / loss $E$

Gradient step

$$w^{(l)}_{i,j} = w^{(l)}_{i,j} - \alpha \cdot \delta^{(l)}_j f^{(l-1)}_i$$

$$b^{(l)}_j = b^{(l)}_j - \alpha \cdot \delta^{(l)}_j$$

Recursion

$$\delta^{(l-1)}_j = \frac{\partial h(y^{(l-1)}_i)}{\partial y^{(l-1)}_i} \sum_{k=1}^{M^{(l)}} \delta^{(l)}_k w^{(l)}_{j,k}$$

$$\delta^{(L)}_j = \frac{\partial E(f^{(L)}_j)}{\partial f^{(L)}_j} \cdot \frac{\partial h(y^{(L)}_j)}{\partial y^{(L)}_j}$$

Note Assume $L$ layers in total
Backpropagation Different notation

**Notation** We used scalar notation so far

**Fact** Same results can be derived using matrix-vector notation

→ Notation depends on your preferences and background
**Backpropagation** Different notation

**Notation** We used scalar notation so far

**Fact** Same results can be derived using matrix-vector notation

→ Notation depends on your preferences and background

**For us** We want to implement backprop. from scratch, thus scalar notation is closer to our implementation

**But** Literature usually use matrix-vector notation for compactness
Backpropagation Different notation

**Notation** We used scalar notation so far

**Fact** Same results can be derived using matrix-vector notation

→ Notation depends on your preferences and background

**For us** We want to implement backprop. from scratch, thus scalar notation is closer to our implementation

**But** Literature usually use matrix-vector notation for compactness

\[
\begin{align*}
\delta^{(l-1)} &= \left( W^{(l)} \right)^T \delta^{(l)} \odot \frac{\partial h(y^{(l-1)})}{\partial y^{(l-1)}} \\
\delta^{(L)} &= \nabla_{y^{(L)}} \ell(y^{(L)}) \odot \frac{\partial h(y^{(L)})}{\partial y^{(L)}}
\end{align*}
\]
**Backpropagation** Different notation

**Notation** We used scalar notation so far

**Fact** Same results can be derived using matrix-vector notation

→ Notation depends on your preferences and background

**For us** We want to implement backprop. from scratch, thus scalar notation is closer to our implementation

**But** Literature usually use matrix-vector notation for compactness

\[
\delta^{(l-1)} = \left( W^{(l)} \right)^T \delta^{(l)} \odot \frac{\partial h(y^{(l-1)})}{\partial y^{(l-1)}}
\]

\[
\delta^{(L)} = \nabla_{y^{(L)}} \ell(y^{(L)}) \odot \frac{\partial h(y^{(L)})}{\partial y^{(L)}}
\]

Hadamard-product / Schur-product: piecewise multiplication

Vectorial derivative!
Backpropagation Some remarks

Observation Backpropagation is a recursive algorithm
Use Dynamic programming for implementation
→ Start with output layer and the go back
Backpropagation Some remarks

Observation Backpropagation is a recursive algorithm
Use Dynamic programming for implementation
→ Start with output layer and the go back

Remark 1 We use SGD to optimize a loss function
→ This requires gradient information

Remark 2 We use backpropagation to compute this gradient
Backpropagation Some remarks

Observation Backpropagation is a recursive algorithm
Use Dynamic programming for implementation
→ Start with output layer and the go back

Remark 1 We use SGD to optimize a loss function
→ This requires gradient information

Remark 2 We use backpropagation to compute this gradient

Important note
SGD is a general optimization approach
Backpropagation is a general way to compute gradients in directed acyclic graphs

Remark 3 With Neural Networks we combine both
Backpropagation Some implementation ideas

Observation: Backprop. is independent from activation $h$ and loss $\ell$
Backpropagation Some implementation ideas

Observation: Backprop. is independent from activation $h$ and loss $\ell$

Thus Implement neural networks layer-wise

- Each layer has activation function
- Each layer has derivative of activation function
- Each layer has weight matrix (either for input or output)
- Each layer implements delta computation
- Output-layer implements delta computation with loss function
- Layers are either connected to each other and recursively call backprop. or some “control” function performs backprop.
Backpropagation Some implementation ideas

Observation: Backprop. is independent from activation \( h \) and loss \( \ell \)

Thus Implement neural networks layer-wise

- Each layer has activation function
- Each layer has derivative of activation function
- Each layer has weight matrix (either for input or output)
- Each layer implements delta computation
- Output-layer implements delta computation with loss function
- Layers are either connected to each other and recursively call backprop. or some “control” function performs backprop.

Thus Arbitrary network architectures can be realised without changing learning algorithm
MLP Some ideas about architectures

Question So what is a good architecture?
MLP Some ideas about architectures

**Question** So what is a good architecture?

**Answer** Depends on the problem. Usually, architectures for new problems are published in scientific papers or even as PHD thesis.

Non-linear activation
A network should contain at least one layer with non-linear activation function for better learning

Sparse activation
To prevent over-fitting, only a few neurons of the network should be active at the same time

Fast convergence
The loss function / activation function should allow a fast convergence in the first few epochs

Feature extraction
Combining multiple layers in deeper networks usually allows (higher) level feature extraction
MLP Some ideas about architectures

Question So what is a good architecture?
Answer Depends on the problem. Usually, architectures for new problems are published in scientific papers or even as PHD thesis.

Some general ideas

- **Non-linear activation** A network should contain at least one layer with non-linear activation function for better learning

- **Sparse activation** To prevent over-fitting, only a few neurons of the network should be active at the same time

- **Fast convergence** The loss function / activation function should allow a fast convergence in the first few epochs

- **Feature extraction** Combining multiple layers in deeper networks usually allows (higher) level feature extraction
Data mining From MLP to Deep Learning

Observation

- **1 perceptron** Separates space into two sets
- **Many perceptrons in 1 layer** Identifies convex sets
- **Many perceptrons in 2 layer** Identifies arbitrary sets
Data mining From MLP to Deep Learning

Observation

- **1 perceptron** Separates space into two sets
- **Many perceptrons in 1 layer** Identifies convex sets
- **Many perceptrons in 2 layer** Identifies arbitrary sets

**Hornik et. al 1989** MLP is a universal approximator

→ Given enough hidden units, a MLP is able to represent any “well-conditioned” function **perfectly**
Data mining From MLP to Deep Learning

Observation

- **1 perceptron** Separates space into two sets
- **Many perceptrons in 1 layer** Identifies convex sets
- **Many perceptrons in 2 layer** Identifies arbitrary sets

Hornik et. al 1989 MLP is a universal approximator

→ Given enough hidden units, a MLP is able to represent any “well-conditioned” function perfectly

Barron 1993 Worst case needs exponential number of hidden units
Data mining From MLP to Deep Learning

Observation

- 1 perceptron Separates space into two sets
- Many perceptrons in 1 layer Identifies convex sets
- Many perceptrons in 2 layer Identifies arbitrary sets

Hornik et. al 1989 MLP is a universal approximator
→ Given enough hidden units, a MLP is able to represent any
  “well-conditioned” function perfectly

Barron 1993 Worst case needs exponential number of hidden units

But That does not necessarily mean, that we will find it!

- Usually we cannot afford exponentially large networks
- Learning of $\vec{w}$ might fail due to data or numerical reasons
Deep Learning From MLP to Deep Learning

So... How did Deep Learning become so popular?
Deep Learning From MLP to Deep Learning

So... How did Deep Learning become so popular?

Krizhevsky et. al 2012
Trade width for depth
→ Extract features and combine them in later layers
Deep Learning From MLP to Deep Learning

So... How did Deep Learning become so popular?

Krizhevsky et. al 2012
Trade width for depth
→ Extract features and combine them in later layers

Zhang et. al 2017
$O(N + d)$ weights are enough for sample of size $N$ in $d$ dimensions
→ “One” neuron per sample

But This introduces new challenges
Deep Learning Vanishing gradients

Observation 1 \[ \sigma(z) = \frac{1}{1+e^{-\beta \cdot z}} \in [0, 1] \]

Observation 2 \[ \frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z)) \in [0, 0.25\beta] \]

Observation 3 Errors are multiplied from the next layer
Deep Learning Vanishing gradients

Observation 1 \[ \sigma(z) = \frac{1}{1 + e^{-\beta \cdot z}} \in [0, 1] \]

Observation 2 \[ \frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z)) \in [0, 0.25\beta] \]

Observation 3 Errors are multiplied from the next layer

Thus The error tends to become very small after a few layers

Hochreiter et. al 2001 Vanishing gradients
Deep Learning Vanishing gradients

Observation 1 \( \sigma(z) = \frac{1}{1+e^{-\beta \cdot z}} \in [0, 1] \)

Observation 2 \( \frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z)) \in [0, 0.25\beta] \)

Observation 3 Errors are multiplied from the next layer

Thus The error tends to become very small after a few layers

Hochreiter et. al 2001 Vanishing gradients

So far No fundamental solution found, but a few suggestions

- Change activation function
- Exploit different optimization methods
- Use more data / carefully adjust stepsizes
- Reduce number of parameters / depth of network
Deep Learning ReLu activation

Rectified Linear (ReLu)

\[
h(z) = \begin{cases} 
  z & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases}
\]

\[
\frac{\partial h(z)}{\partial z} = \begin{cases} 
  1 & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases}
\]

Note ReLu is not differentiable in \(z = 0\)!

But Usually that is not a problem

Practical \(z = 0\) is pretty rare, just use 0 there. It works well

Mathematical There exists a subgradient of \(h(z)\) at 0
**Deep Learning** ReLu activation

Rectified Linear (ReLu)

\[ h(z) = \begin{cases} 
  z & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases} = \max(0, z) \]

\[ \frac{\partial h(z)}{\partial z} = \begin{cases} 
  1 & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases} \]
Deep Learning ReLu activation

Rectified Linear (ReLu)

\[ h(z) = \begin{cases} 
  z & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases} = \max(0, z) \]

\[ \frac{\partial h(z)}{\partial z} = \begin{cases} 
  1 & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases} \]

Note ReLu is not differentiable in \( z = 0 \)!
Deep Learning ReLu activation

Rectified Linear (ReLu)

\[
h(z) = \begin{cases} 
  z & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases} = \max(0, z)
\]

\[
\frac{\partial h(z)}{\partial z} = \begin{cases} 
  1 & \text{if } z \geq 0 \\
  0 & \text{else}
\end{cases}
\]

**Note** ReLu is not differentiable in \( z = 0 \)!

**But** Usually that is not a problem

- **Practical** \( z = 0 \) is pretty rare, just use 0 there. It works well
- **Mathematical** There exists a subgradient of \( h(z) \) at 0
Deep Learning ReLu activation (2)

Subgradients A gradient shows the direct of the steepest descent
⇒ If a function is not differentiable, it has no steepest descent
⇒ There might be multiple (equally) “steepest descents”

For ReLu
\[
\frac{\partial h}{\partial z} \bigg|_{z=0} \text{from} [0, 1]
\]

Big Note
Using a subgradient does not guarantee that our loss function decreases! We might change weights to the worse!

Nice properties of ReLu
Super-easy forward, backward and derivative computation
Either activates or deactivates a neuron (sparsity)
No vanishing gradients, since error is multiplied by 0 or 1
Still gives network non-linear activation
Deep Learning ReLu activation (2)

Subgradients A gradient shows the direct of the steepest descent
⇒ If a function is not differentiable, it has no steepest descent
⇒ There might be multiple (equally) “steepest descents”

For ReLu We can choose $\frac{\partial h(z)}{\partial z}\bigg|_{z=0}$ from $[0, 1]$

Big Note Using a subgradient does not guarantee that our loss function decreases! We might change weights to the worse!
Deep Learning ReLu activation (2)

**Subgradients** A gradient shows the direct of the steepest descent
⇒ If a function is not differentiable, it has no steepest descent
⇒ There might be multiple (equally) “steepest descents”

**For ReLu** We can choose \( \frac{\partial h(z)}{\partial z} \bigg|_{z=0} \) from [0, 1]

**Big Note** Using a subgradient does not guarantee that our loss function decreases! We might change weights to the worse!

**Nice properties of ReLu**
- Super-easy forward, backward and derivative computation
- Either activates or deactivates a neuron (sparsity)
- No vanishing gradients, since error is multiplied by 0 or 1
- Still gives network non-linear activation
Deep Learning Loss function

Usually Squared error

\[ E = \frac{1}{2} (y - f^{(L)})^2 \Rightarrow \frac{\partial E}{\partial f^{(L)}} = - (y - f^{(L)}) \]
Deep Learning Loss function

Usually Squared error

\[ E = \frac{1}{2} \left( y - f^{(L)} \right)^2 \Rightarrow \frac{\partial E}{\partial f^{(L)}} = - \left( y - f^{(L)} \right) \]

Recall

\[ \frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \]

Thus \( \delta^{(L)}_j = - \left( y - f^{(L)} \right) \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \to \text{small for sigmoid!} \)
Deep Learning Loss function

Usually Squared error

\[ E = \frac{1}{2} \left( y - f^{(L)} \right)^2 \Rightarrow \frac{\partial E}{\partial f^{(L)}} = - \left( y - f^{(L)} \right) \]

Recall

\[ \frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \]

Thus

\[ \delta_j^{(L)} = - \left( y - f^{(L)} \right) \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \rightarrow \text{small for sigmoid!} \]
Deep Learning Loss function (2)

Recall
\[
\frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \quad \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}}
\]
Deep Learning Loss function (2)

Recall
\[
\frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \quad \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}}
\]

Mohamed et. al 2009
Cross-entropy
\[
E = - (y \ln (f^{(L)}) + (1 - y) \ln (1 - f^{(L)})) \quad \Rightarrow \quad \frac{\partial E}{\partial f^{(L)}} = \frac{f^{(L)} - y}{(1 - f^{(L)}) f^{(L)}}
\]
Deep Learning Loss function (2)

Recall

\[ \frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \]

Mohamed et. al 2009
Cross-entropy

\[ E = - (y \ln (f^{(L)}) + (1 - y) \ln (1 - f^{(L)})) \Rightarrow \frac{\partial E}{\partial f^{(L)}} = \frac{f^{(L)} - y}{(1 - f^{(L)}) f^{(L)}} \]

Idea View \( y \) and \( \hat{y} \) as categorical distribution
Then Minimize distance between both distributions

Nice bonus

\[ \delta_j^{(L)} = \frac{f^{(L)} - y}{(1 - f^{(L)}) f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} = f^{(L)} - y \] cancels small sigmoids
Deep Learning Loss function (2)

Recall
\[ \frac{\partial h(z)}{\partial z} = h(z) \cdot (1 - h(z)), \quad \delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} \]

Mohamed et. al 2009

Cross-entropy
\[ E = - \left( y \ln \left( f^{(L)} \right) + (1 - y) \ln \left( 1 - f^{(L)} \right) \right) \Rightarrow \frac{\partial E}{\partial f^{(L)}} = \frac{f^{(L)} - y}{(1 - f^{(L)}) f^{(L)}} \]

Idea View \( y \) and \( \hat{y} \) as categorical distribution

Then Minimize distance between both distributions

Nice bonus
\[ \delta^{(L)}_j = \frac{f^{(L)} - y}{(1 - f^{(L)}) f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} = f^{(L)} - y \text{ cancels small sigmoids} \]

Important
Make sure that \( \sum f^L = 1 \) → This is called softmax layer
Data Mining  Convergence of SGD

Recall  We use the SGD framework

1: $\vec{\theta} = random()$
2:  while ERROR do
3:  choose random $(\vec{x}, y) \in D$
4:  $\vec{\theta} = \vec{\theta} - \alpha_t \cdot \frac{\partial E(x,y)}{\partial \vec{\theta}}$
5:  end while

Bottou et al. 2017

SGD converges if

1) $\frac{\partial E(x,y)}{\partial \vec{\theta}} = \nabla_{\vec{\theta}} E[\nabla_{\vec{\theta}} E(D)]$ is unbiased estimator of true gradient

2) $\alpha_t \to 0$, if $E$ is not convex

Note  If $E$ is non-convex we may find a local minima
Data Mining  Convergence of SGD

Recall  We use the SGD framework

1: \( \vec{\theta} = \text{random}() \)
2: \( \textbf{while} \ \text{ERROR} \ \textbf{do} \)
3: \( \text{choose random } (\vec{x}, y) \in \mathcal{D} \)
4: \( \vec{\theta} = \vec{\theta} - \alpha_t \cdot \frac{\partial E(x,y)}{\partial \vec{\theta}} \)
5: \( \textbf{end while} \)

Bottou et al. 2017  SGD converges if
\( 1) \ \frac{\partial E(x,y)}{\partial \vec{\theta}} = \nabla_{\theta} E[\nabla_{\theta} E(\mathcal{D})] \) is unbiased estimator of true gradient
\( 2) \ \alpha_t \to 0, \text{ if } E \text{ is not convex} \)

Note  If \( E \) is non-convex we may find a local minima
SGD Step size

What about the step size?

- If its too small, you will learn slow (→ more data required)
- If its too big, you might miss the optimum (→ bad results)

Therefore usually

Small $\alpha = 0.001 \ldots 0.1$ with a lot of data

Note We can always reuse our data (multiple passes over dataset)

Step size is problem specific as always!

Practical suggestion

Simple heuristic
Try out different stepsizes on small subsample of data
Pick that one that most reduces the loss
Use it for on the full dataset

Sidenote
Changing the stepsize while training also possible
SGD Stepsize

What about the stepsize?

- If it's too small, you will learn slow (→ more data required)
- If it's too big, you might miss the optimum (→ bad results)

**Thus usually** Small $\alpha = 0.001 \ldots 0.1$ with a lot of data

**Note** We can always reuse our data (multiple passes over dataset)

**But** Stepsize is problem specific as always!
SGD Stepsize

What about the stepsize?

- If its too small, you will learn slow (→ more data required)
- If its too big, you might miss the optimum (→ bad results)

Thus usually Small $\alpha = 0.001 - 0.1$ with a lot of data

Note We can always reuse our data (multiple passes over dataset)

But Stepsize is problem specific as always!

Practical suggestion Simple heuristic

- Try out different stepsizes on small subsample of data
- Pick that one that most reduces the loss
- Use it for on the full dataset

Sidenote Changing the stepsize while training also possible
\[ \Delta \hat{\theta}^{old} = \alpha_1 \cdot \nabla_{\theta} E(D, \hat{\theta}^{old}) + \alpha_2 \Delta \hat{\theta}^{old} \]

\[ \hat{\theta}^{new} = \hat{\theta}^{old} - \Delta \hat{\theta}^{old} \]
\[ \Delta \hat{\theta}^{\text{old}} = \alpha_1 \cdot \nabla \theta E(D, \hat{\theta}^{\text{old}}) + \alpha_2 \Delta \hat{\theta}^{\text{old}} \]

\[ \hat{\theta}^{\text{new}} = \hat{\theta}^{\text{old}} - \Delta \hat{\theta}^{\text{old}} \]

**Theoretically more sound**

- **Nesterov 1983** / **Sutskever et. al 2013** Nesterov momentum
- **Tielman et al. 2012** / **Graves 2013** RMSProp
- **Kingma and Lei Ba 2015** Momentum tuned for SGD: ADAM
  ...and many more AdaGrad, AdaMax, AdaDelta, ...

**Bonus** Methods often give heuristic for step-size
SGD Utilize parallelism

(Mini-)Batch
Compute derivatives on batch and average direction
→ parallel computation + only 1 parameter update

\[ \hat{\theta}^{\text{new}} = \hat{\theta}^{\text{old}} - \alpha \cdot \frac{1}{K} \sum_{i=0}^{K} \nabla_{\theta} E(\vec{x}_i, \hat{\theta}^{\text{old}}) \]

Note That works particularly well on GPUs or FPGAs . . .
SGD Initial solution

For SGD
Need initial solution $\theta$

Common in practice
Bias $b = 0$, weights $w_{ij}^l \sim \mathcal{N}(0, 0.05)$
Bias $b = 0$, weights $w_{ij}^l \sim \mathcal{U}(-0.05, 0.05)$
**SGD Initial solution**

**For SGD**
Need initial solution $\theta$

**Common in practice**
Bias $b = 0$, weights $w_{ij}^l \sim \mathcal{N}(0, 0.05)$
Bias $b = 0$, weights $w_{ij}^l \sim \mathcal{U}(-0.05, 0.05)$

**Why care?**

$$
\delta^{(L)} = \frac{\partial E(f^{(L)})}{\partial f^{(L)}} \cdot \frac{\partial h(y^{(L)})}{\partial y^{(L)}} = -(y_i - f_j^L) f_j^L (1 - f_j^L)
$$

$$
\delta^{(L)} = 0 \text{ if } f_j^L = 0 \text{ or } f_j^L = 1
$$

**Thus** We stuck in local minima if we have a bad initialization
Deep Learning Slow learning rate

Recall
Input of neuron depends on output of previous neurons
Deep Learning Slow learning rate (2)

**Observation** During training, activations change over time

**Thus** Input distribution for neurons also change over time
Deep Learning Slow learning rate (2)

Observation During training, activations change over time
Thus Input distribution for neurons also change over time

Note This is what we want!
But This prevents us from using larger step-sizes
Deep Learning  Slow learning rate (2)

Observation  During training, activations change over time  
Thus  Input distribution for neurons also change over time  

Note  This is what we want!  
But  This prevents us from using larger step-sizes  

Ioffe and Szegedy 2015  
Internal covariate shift of activations  

Idea  
Normalize neuron inputs to be zero mean / unit variance
Deep Learning Slow learning rate (3)

During training
Given mini batch $\mathcal{B} = \{(y^l_j)_i\}_{i=1,...,K}$, compute

$$
\bar{y}_j^l = \frac{1}{K} \sum_{i=0}^{K} (y^l_j)_i
$$

$$(y^l_j)_i = \frac{(y^l_j)_i - \bar{y}_j^l}{\sqrt{\sigma_{\mathcal{B}}} + \varepsilon}
$$
Deep Learning Slow learning rate (3)

During training
Given mini batch $\mathcal{B} = \{(y_j^l)_i\}_{i=1,...,K}$, compute

$$y_j^l = \frac{1}{K} \sum_{i=0}^{K} (y_j^l)_i$$

$$\frac{(y_j^l)_i}{\sqrt{\sigma_B} + \varepsilon}$$

Note
During inference there is usually no mini batch
Thus
Estimate $y_j^l$ over all training data while training
Data Mining Large models tend to overfit

Common intuition 1
Large models tend to memorize data → no generalization
Data Mining Large models tend to overfit

Common intuition 1
Large models tend to memorize data $\rightarrow$ no generalization

Han et. al 2016 $\sim 1.2 - 140$ Million parameters in $\geq 8$ layers
**Data Mining** Large models tend to overfit

**Common intuition 1**
Large models tend to memorize data → no generalization

**Han et. al 2016** \(\sim\) 1.2 – 140 Million parameters in \(\geq 8\) layers

**Common intuition 2**
Training error always decreases, but test error may increase again
**Data Mining** Large models tend to overfit

**Common intuition 1**
Large models tend to memorize data $\rightarrow$ no generalization

**Han et. al 2016** $\sim 1.2 - 140$ Million parameters in $\geq 8$ layers

**Common intuition 2**
Training error always decreases, but test error may increase again

**Bishop ’95 / Sjörborg & Lijung ’95**
Limit SGD to volume around initial solution
Data Mining Large models tend to overfit

Common intuition 1
Large models tend to memorize data \( \rightarrow \) no generalization

Han et. al 2016 \( \sim 1.2 - 140 \) Million parameters in \( \geq 8 \) layers

Common intuition 2
Training error always decreases, but test error may increase again

Bishop ’95 / Sjörborg & Lijung ’95
Limit SGD to volume around initial solution

Common practice Early stopping
\( \rightarrow \) Use fixed number of iterations or timesteps
**Deep Learning** Force redundancy

**Hinton et al. 2013 / Srivastava et al. 2014** DropOut
Ignore neuron with probability $p$ during forward-pass in training
→ sometimes $f_{j}^{l} = 0$ during training
Deep Learning Force redundancy

Hinton et al. 2013 / Srivastava et al. 2014 DropOut
Ignore neuron with probability $p$ during forward-pass in training
→ sometimes $f^l_j = 0$ during training

Wan et al. 2014: DropConnect
Ignore weight with probability $p$ during forward-pass in training
→ sometimes $w^l_{i,j} = 0$ during training
Summary

Important concepts

- **For parameter optimization** we define a loss function
- **For parameter optimization** we use gradient descent
- **Neurons** have activation functions to ensure non-linearity and differentiability
- **Backpropagation** is an algorithm to compute the gradient
- **Deep Learning** requires new activation functions
- **Deep Learning** requires new loss functions
- **Deep Learning** sometimes require a lot fine-tuning
Homework

Homework until next meeting

- Implement the following network to solve the XOR problem

\[ x_1 \rightarrow \text{node} \rightarrow \text{node} \rightarrow \text{output} \]

- Implement backpropagation for this network
  - Try a simple solution first: Hardcode one activation / one loss function with fixed access to data structures
  - If you feel comfortable, add new activation / loss functions

Tip 1: Verify that the proposed network uses 9 parameters

Tip 2: Start with \( \alpha = 1.0 \) and 10000 training examples

Note: We will later use C, so please use C or a C-like language

Question: How many iterations do you need until convergence?