

Machine Learning Basics III

Benjamin Roth

CIS LMU München

Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

Outline

- 1 Classification
 - Logistic Regression
- 2 Gradient Based Optimization
 - Gradient Descent for Logistic Regression
 - Stochastic Gradient Descent
- 3 Deep Feedforward Networks
 - Design Choices for Output Units
 - Design Choices for Hidden Units
 - Backpropagation
- 4 Regularization
 - L2-Regularization

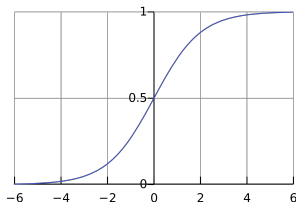
From Regression to Classification

- So far, linear regression:
 - ▶ A simple linear model.
 - ▶ Probabilistic interpretation.
 - ▶ Find optimal parameters using Maximum Likelihood Estimation.
- Can we do something similar for classification?
- \Rightarrow Logistic Regression (*... it's not actually used for regression ...*)

Logistic Regression

- Binary logistic model:
Estimate the probability of a binary response $y \in \{0, 1\}$ based on features \mathbf{w} .
- Logistic Regression is a *Generalized Linear Model*:
Linear model is related to the response variable via a **link function**.

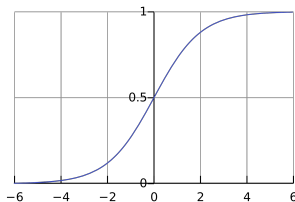
$$p(Y = 1|\mathbf{x}; \boldsymbol{\theta}) = f(\boldsymbol{\theta}^T \mathbf{x})$$



(Note: Y denotes a random variable, whereas y , $y^{(i)}$, $0, 1$ denote values that the random variable can take on. If the random variable is obvious from the context, it may be omitted.)

Logistic Regression

- Recall linear regression: $p(y|\mathbf{x}; \boldsymbol{\theta}) = N(y; \boldsymbol{\theta}^T \mathbf{x}, \mathbf{I})$
 - ▶ Predicts $y \in \mathbb{R}$
- Classification: Outcome (per example) 0 or 1
 - ▶ Logistic sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



- ▶ Logistic Regression: Linear function + logistic sigmoid

$$p(Y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T \mathbf{x})$$

Binary Logistic Regression

Probability of different outcomes (for one example):

- Probability of positive outcome:

$$p(Y = 1|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

- Probability of negative outcome:

$$p(Y = 0|\mathbf{x}; \boldsymbol{\theta}) = 1 - p(Y = 1|\mathbf{x}; \boldsymbol{\theta})$$

Probability of a Training Example

- Probability for actual label $y^{(i)}$ given features $\mathbf{x}^{(i)}$
- Can be written for both labels (0 and 1) without case distinction
- Label exponentiation trick: use $x^0 = 1$

$$\begin{aligned} & p(Y = y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) \\ &= \begin{cases} p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 1 \\ p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 0 \end{cases} \\ &= \begin{cases} p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^1 p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^0 & \text{if } y^{(i)} = 1 \\ p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^0 p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^1 & \text{if } y^{(i)} = 0 \end{cases} \\ &= p(Y = 1 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^{y^{(i)}} p(Y = 0 | \mathbf{x}^{(i)}; \boldsymbol{\theta})^{1-y^{(i)}} \end{aligned}$$

Binary Logistic Regression

- Conditional Negative Log-Likelihood (NLL):

$$\begin{aligned} NLL(\boldsymbol{\theta}) &= -\log p(\mathbf{y}|\mathbf{X}; \boldsymbol{\theta}) \\ &= -\log \prod_{i=1}^m p(Y = y^{(i)}|\mathbf{x}^{(i)}; \boldsymbol{\theta}) \\ &= -\log \prod_{i=1}^m p(Y = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta})^{y^{(i)}} (1 - p(Y = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta}))^{1-y^{(i)}} \\ &= -\sum_{i=1}^m y^{(i)} \log \sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}^{(i)})) \end{aligned}$$

- No closed form solution for minimum
- Use numerical / iterative methods.
- LBFGS, Gradient descent ...

Logistic Regression

- Logistic regression: Logistic sigmoid function applied to a a weighted linear combination of feature values.
- To be interpreted as the probability that the label for a specific example equals 1.
- Applying the model on test data: Predict $y^{(i)} = 1$ if

$$p(Y = 1|\mathbf{x}^{(i)}; \boldsymbol{\theta}) > 0.5$$

- No closed form solution for maximizing NLL, iterative methods necessary.
- Next up: Gradient descent.

Outline

- 1 Classification
 - Logistic Regression
- 2 Gradient Based Optimization
 - Gradient Descent for Logistic Regression
 - Stochastic Gradient Descent
- 3 Deep Feedforward Networks
 - Design Choices for Output Units
 - Design Choices for Hidden Units
 - Backpropagation
- 4 Regularization
 - L2-Regularization

Optimization

- Optimization: Minimize some function $J(\theta)$ by altering θ .
- Maximize $f(\theta)$ by minimizing $J(\theta) = -f(\theta)$
- $J(\theta)$:
 - ▶ “*criterion*”, “*objective function*”, “*cost function*”, “*loss function*”, “*error function*”
 - ▶ In a probabilistic machine learning setting often (conditional) negative log-likelihood:

$$-\log p(\mathbf{X}; \theta)$$

or

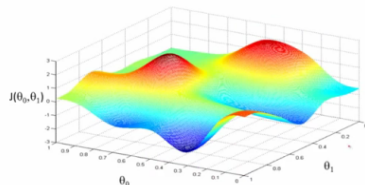
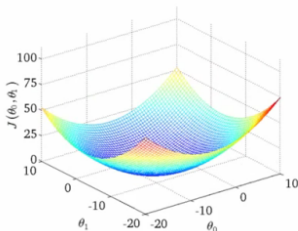
$$-\log p(\mathbf{y}|\mathbf{X}; \theta)$$

as a function of θ

- ▶ $\theta^* = \arg \min_{\theta} J(\theta)$

Optimization

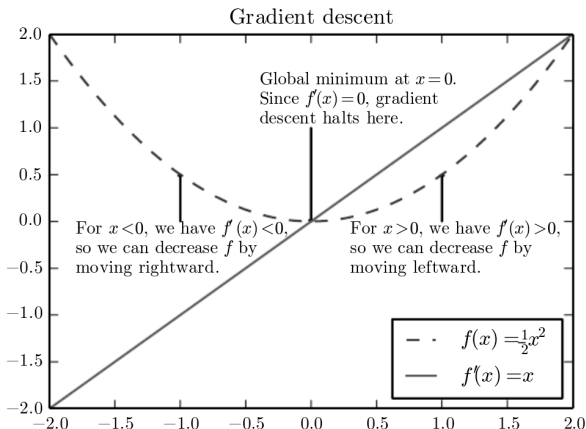
- If $J(\theta)$ is convex, it is minimized where $\nabla_{\theta} J(\theta) = \mathbf{0}$
- If $J(\theta)$ is not convex, the gradient can help us to improve our objective nevertheless (and find a local optimum).
- Many optimization techniques were originally developed for convex objective functions, but are found to be working well for non-convex functions too.
- Use the fact that gradient indicates the slope of the function in the direction of steepest increase.



Gradient-Based Optimization

- Derivative: Given a small change in input, what is the corresponding change in output?

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x)$$



- $f(x - \epsilon \text{sign } f'(x)) < f(x)$ for small enough ϵ

Gradient Descent

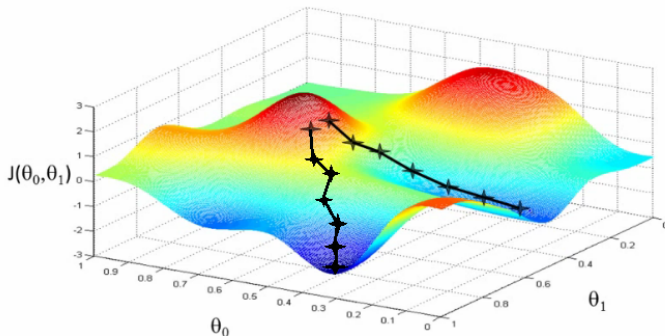
- For $J(\boldsymbol{\theta}) : \mathbb{R}^n \rightarrow \mathbb{R}$
- If partial derivative $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} > 0$, $J(\boldsymbol{\theta})$ will increase for small increases of θ_j
 \Rightarrow go in opposite direction of gradient (since we want to minimize)
- Steepest descent: iterate

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- η is the learning rate (set to small positive constant).
- Converges if $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is (close to) $\mathbf{0}$

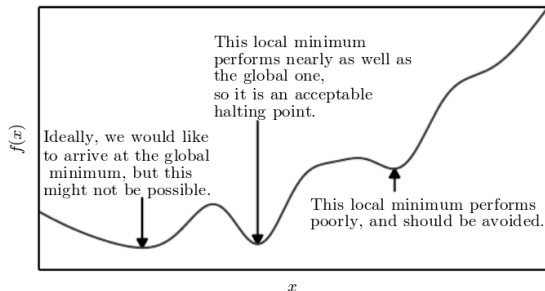
Local Minima

- If function is non-convex, different results can be obtained at convergence, depending on initialization of θ .

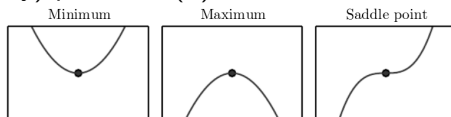


Local Minima

- Minima can be global or local:



- Critical (stationary) points: $f'(x) = 0$



- For neural networks, only good (not perfect) parameter values can be found.

Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

Gradient Descent for Logistic Regression

$$\begin{aligned}\nabla_{\theta} NLL(\theta) &= -\nabla_{\theta} \sum_{i=1}^m y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\theta^T \mathbf{x}^{(i)})) \\ &= -\sum_{i=1}^m (y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)})) \mathbf{x}^{(i)}\end{aligned}$$

- The gradient descent update becomes:

$$\theta_{t+1} := \theta_t + \eta \sum_{i=1}^m (y^{(i)} - \sigma(\theta_t^T \mathbf{x}^{(i)})) \mathbf{x}^{(i)}$$

- Note: Which feature weights are increased, which are decreased?

Derivation of Gradient for Logistic Regression

This is a great exercise! Use the following facts:

Gradient	$(\nabla_{\theta} f(\boldsymbol{\theta}))_j = \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_j}$
----------	--

Derivative of a sum	$\frac{d}{dz} \sum_i f_i(z) = \sum_i \frac{df_i(z)}{dz}$
---------------------	--

Chain rule	$F(z) = f(g(z)) \Rightarrow F'(z) = f'(g(z))g'(z)$
------------	--

Derivative of logarithm	$\frac{d \log z}{dz} = 1/z$
-------------------------	-----------------------------

D. of logistic sigmoid	$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$
------------------------	--

Partial d. of dot-product	$\frac{\partial \boldsymbol{\theta}^T \mathbf{x}}{\partial \theta_j} = \mathbf{x}_j$
---------------------------	--

Gradient Descent: Summary

- Iterative method for function minimization.
- Gradient indicates rate of change in objective function, given a local change to feature weights.
- Subtract the gradient:
 - ▶ **decrease** parameters that (locally) have **positive** correlation with objective
 - ▶ **increase** parameters that (locally) have **negative** correlation with objective
- Gradient updates only have the desired properties in a small region around previous parameters θ_t . Control locality by step-size η .
- Gradient descent is slow: For relatively small step in the right direction, all of training data has to be processed.
- This version of gradient descent is often also called *batch gradient descent*.

Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

Stochastic Gradient Descent (SGD)

- *Batch gradient descent* is slow: For relatively small step in the right direction, all of training data has to be processed.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \sum_{i=1}^m \log p(y_i | \mathbf{x}_i; \boldsymbol{\theta})$$

- *Stochastic gradient descent* in a nutshell:
 - ▶ For each update, only use random sample \mathbb{B}_t of training data (mini-batch).

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \sum_{i \in \mathbb{B}_t} \log p(y_i | \mathbf{x}_i; \boldsymbol{\theta})$$

- ▶ Mini-batch size can also just be 1.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_{\boldsymbol{\theta}} \log p(y_t | \mathbf{x}_t; \boldsymbol{\theta})$$

- \Rightarrow More frequent updates.

Stochastic Gradient Descent (SGD)

- The actual gradient is *approximated* using only a sub-sample of the data.
- For objective functions that are highly non-convex, the random deviations of these approximations may even help to escape local minima.
- Treat batch size and learning rate as hyper-parameter.

Outline

- 1 Classification
 - Logistic Regression
- 2 Gradient Based Optimization
 - Gradient Descent for Logistic Regression
 - Stochastic Gradient Descent
- 3 Deep Feedforward Networks
 - Design Choices for Output Units
 - Design Choices for Hidden Units
 - Backpropagation
- 4 Regularization
 - L2-Regularization

Deep Feedforward Networks

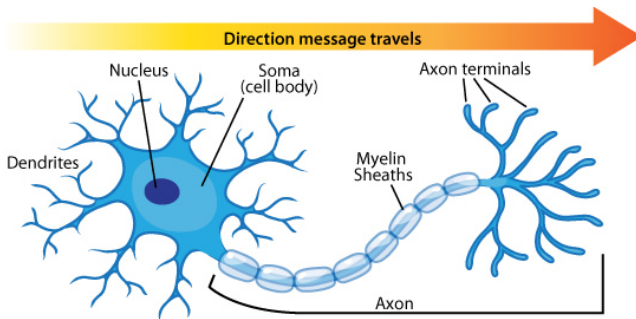
- Function approximation: find good mapping $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$
- *Network*: Composition of functions $f^{(1)}, f^{(2)}, f^{(3)}$ with multi-dimensional input and output
- Each $f^{(i)}$ represents one *layer* $f(\mathbf{x}) = f^{(1)}(f^{(2)}(f^{(3)}(\mathbf{x})))$
- *Feedforward*:
 - ▶ Input \rightarrow intermediate representation \rightarrow output
 - ▶ No feedback connections
 - ▶ Cf. *recurrent* networks

Deep Feedforward Networks: Training

- Loss function defined on output layer, e.g. $\|\hat{y} - f(\mathbf{x}; \boldsymbol{\theta})\|_2^1$
- Quality criterion on other layers not directly defined.
- Training algorithm must decide how to use those layers most effectively (w.r.t. loss on output layer)
- Non-output layers can be viewed as providing a feature function $\phi(\mathbf{x})$ of the input, that is to be learned.

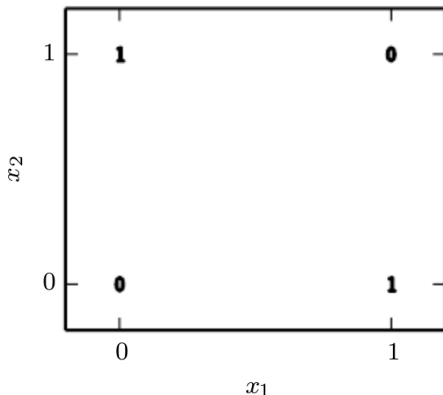
"Neural" Networks

- Inspired by biological neurons (nerve cells)
- Neurons are connected to each other, and receive and send electrical pulses.
- *"If the [input] voltage changes by a large enough amount, an all-or-none electrochemical pulse called an action potential is generated, which travels rapidly along the cell's axon, and activates synaptic connections with other cells when it arrives."* (Wikipedia)



Activation Functions with Non-Linearities

- Linear Functions are limited in what they can express.
- Famous example: XOR
- Simple layered non-linear functions can represent XOR.



Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

Design Choices for Output Units

- Typically can be interpreted as probabilities.
 - ▶ Logistic sigmoid
 - ▶ Softmax
 - ▶ mean and variance of a Gaussian, ...
- Trained with negative log-likelihood.

Softmax

- Logistic sigmoid

- ▶ Vector \mathbf{y} of binary outcomes, with no constraints on how many can be 1.
- ▶ Bernoulli distribution.

- Softmax

- ▶ Exactly one element of \mathbf{y} is 1.
- ▶ Multinoulli (categorical) distribution.

$$p(Y = i | \phi(\mathbf{x}))$$

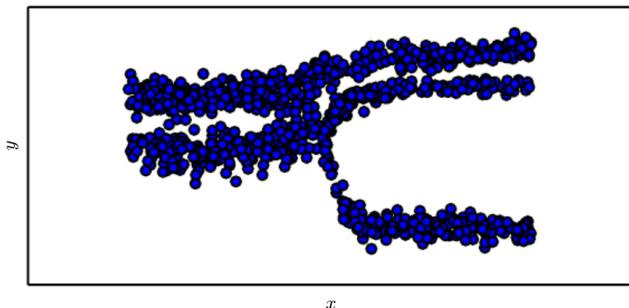
$$\sum_i p(Y = i | \phi(\mathbf{x})) = 1$$

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Parametrizing a Gaussian Distribution

- Use final layer to predict parameters of Gaussian mixture model.
- Weight of mixture component: softmax.
- Means: no non-linearity.
- Precisions ($\frac{1}{\sigma^2}$) need to be positive: softplus

$$\text{softplus}(z) = \ln(1 + \exp(z))$$



Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

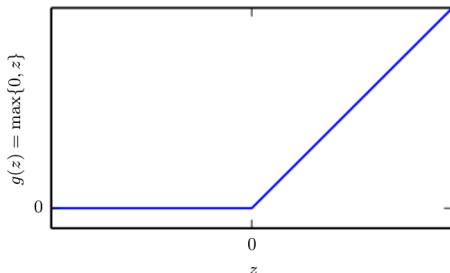
- L2-Regularization

Rectified Linear Units

- Rectified Linear Unit:

$$\text{relu}(z) = \max(0, z)$$

$$z = \mathbf{x}^T \mathbf{w} + b$$



- Consistent gradient of 1 when unit is *active* (i.e. if there is an error to propagate).
- Default choice for hidden units.

A Simple ReLU Network to Solve XOR

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}) = \mathbf{w}^T \max(0, \mathbf{W}^T \mathbf{x} + \mathbf{c})$$

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Other Choices for Hidden Units

- A good activation function aids learning, and provides large gradients.
- Sigmoidal functions (logistic sigmoid)
 - ▶ have only a small region before they flatten out in either direction.
 - ▶ Practice shows that this seems to be ok in conjunction with Log-loss objective.
 - ▶ But they don't work as well as hidden units.
 - ▶ ReLU are better alternative since gradient stays constant.
- Other hidden unit functions:
 - ▶ maxout: take maximum of several values in previous layer.
 - ▶ purely linear: can serve as low-rank approximation.

Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

- Forward propagation: Input information x propagates through network to produce output \hat{y} (and cost $J(\theta)$ in training)
- Back-propagation:
 - ▶ compute gradient w.r.t. model parameters
 - ▶ Cost gradient propagates backwards through the network
- Back-propagation is part of learning procedure (e.g. stochastic gradient descent), not learning procedure in itself.

Chain Rule of Calculus: Real Functions

- Let

$$x, y, z \in \mathbb{R}$$

$$f, g : \mathbb{R} \rightarrow \mathbb{R}$$

$$y = g(x)$$

$$z = f(g(x)) = f(y)$$

- Then

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Chain Rule of Calculus: Multivariate Functions

- Let

$$\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n, z \in \mathbb{R}$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$g : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

$$\mathbf{y} = g(\mathbf{x})$$

$$z = f(g(\mathbf{x})) = f(\mathbf{y})$$

- Then

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

- In order to write this in vector notation, we need to define the Jacobian matrix.

Jacobian

- The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

$$\mathbf{J} = \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial g(\mathbf{x})_1}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_1}{\partial x_m} \\ \frac{\partial g(\mathbf{x})_2}{\partial x_1} & & \frac{\partial g(\mathbf{x})_2}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial g(\mathbf{x})_n}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_n}{\partial x_m} \end{bmatrix}$$

- How to write in terms of gradients?
- We can write the chain rule as:

$$\nabla_{\mathbf{x}} z =$$

Jacobian

- The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

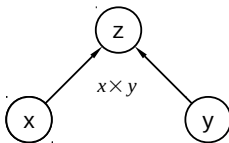
$$\mathbf{J} = \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial g(\mathbf{x})_1}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_1}{\partial x_m} \\ \frac{\partial g(\mathbf{x})_2}{\partial x_1} & & \frac{\partial g(\mathbf{x})_2}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial g(\mathbf{x})_n}{\partial x_1} & \dots & \frac{\partial g(\mathbf{x})_n}{\partial x_m} \end{bmatrix}$$

- How to write in terms of gradients?
- We can write the chain rule as:

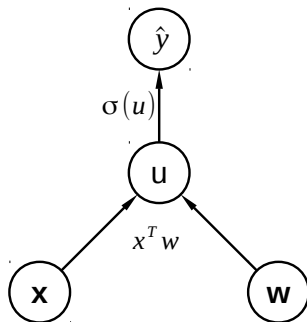
$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z$$

Viewing the Network as a Graph

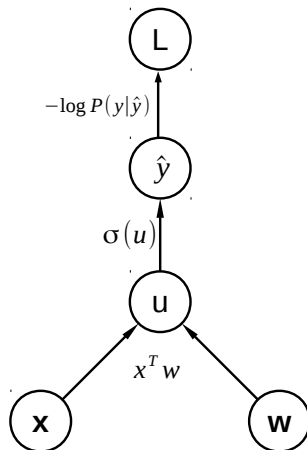
- Nodes are function outputs (can be scalar or vector valued)
- Arrows are inputs
- Example: Scalar multiplication $z = xy$.



Which Function?

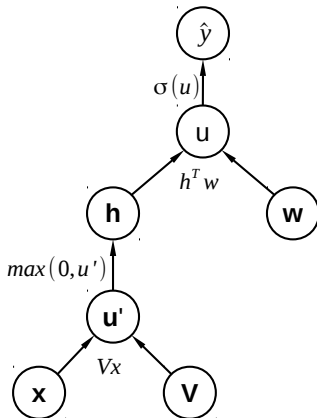


Graph with Cost



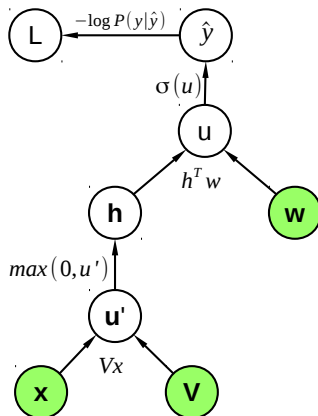
Which Function?

- Parameter vectors can be converted to matrix as needed.



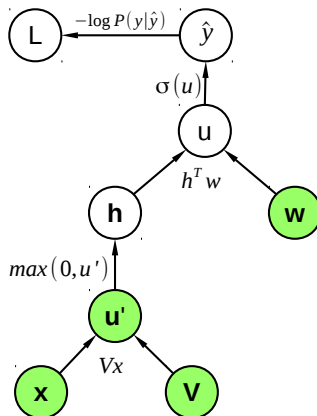
Forward Pass

- Green: known or computed.



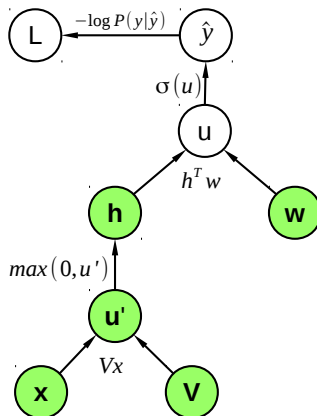
Forward Pass

- Green: known or computed.



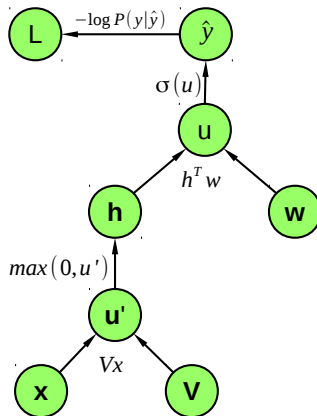
Forward Pass

- Green: known or computed.



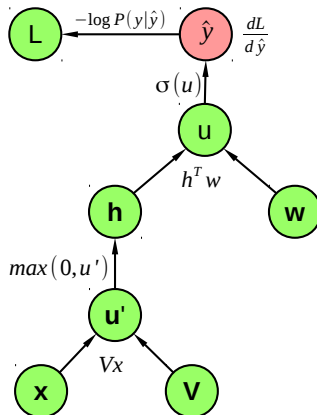
Forward Pass

- End of forward pass (some steps skipped).



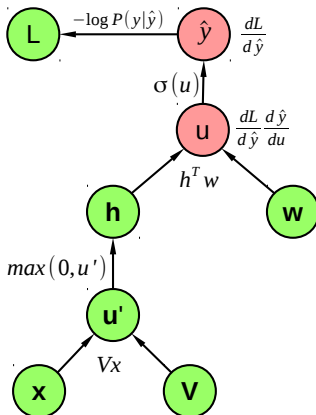
Backward Pass

- Red: gradient of cost computed for node.



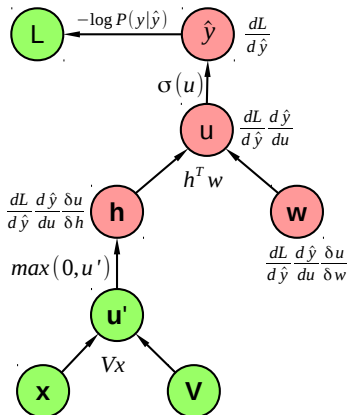
Backward Pass

- Red: gradient of cost computed for node.



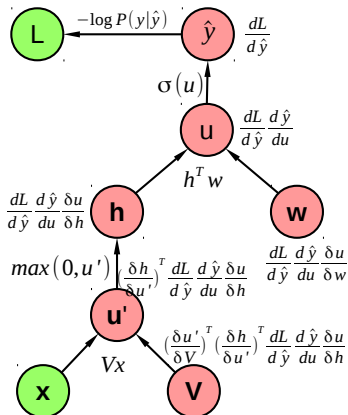
Backward Pass

- Red: gradient of cost computed for node.



End of Backward Pass

- We have the gradients for all parameters, let's use them for SGD.



Summary

- Gradient descent: Minimize loss by iteratively subtracting gradient from parameter vector.
- Stochastic gradient descent: Approximate gradient by considering small subsets of examples.
- Regularization: penalize large parameter values, e.g. by adding l2-norm of parameter vector.
- Feedforward networks: layers of (non-linear) function compositions.
- Output non-linearities: interpreted as probability densities (logistic sigmoid, softmax, Gaussian)
- Hidden layers: Rectified linear units ($\max(0, z)$)
- Backpropagation: Compute gradient of cost w.r.t. parameters using chain rule.

Outline

- 1 Classification
 - Logistic Regression
- 2 Gradient Based Optimization
 - Gradient Descent for Logistic Regression
 - Stochastic Gradient Descent
- 3 Deep Feedforward Networks
 - Design Choices for Output Units
 - Design Choices for Hidden Units
 - Backpropagation
- 4 Regularization
 - L2-Regularization

Outline

1 Classification

- Logistic Regression

2 Gradient Based Optimization

- Gradient Descent for Logistic Regression
- Stochastic Gradient Descent

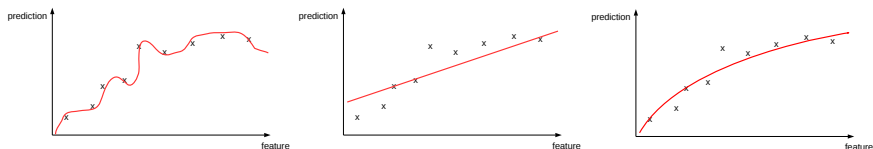
3 Deep Feedforward Networks

- Design Choices for Output Units
- Design Choices for Hidden Units
- Backpropagation

4 Regularization

- L2-Regularization

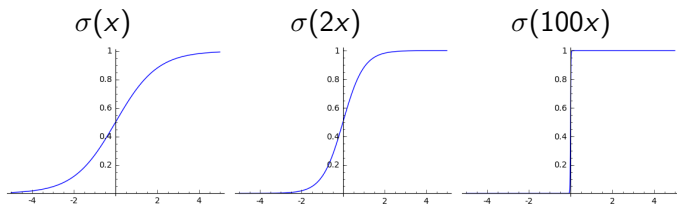
Regularization



- Overfitting vs. underfitting
- Regularization: Any modification to a learning algorithm for reducing its generalization error but not its training error
- Build a preference into ML algorithm for one solution in hypothesis space over another
- Solution space is still the same
- Unpreferred solution is penalized: only chosen if there it fits training data much better

L2-Regularization

- Large parameters \rightarrow overfitting



- Prefer models with smaller feature weights.
- Popular regularizers:
 - ▶ Penalize large L2 norm.
 - ▶ Penalize large L1 norm (aka LASSO, induces sparsity)

Regularization

- Add term that penalizes large l2 norm.
- The amount of penalty is controlled by a parameter λ
 - ▶ Linear regression:

$$J(\theta) = \text{MSE}(\theta) + \frac{\lambda}{2} \theta^T \theta$$

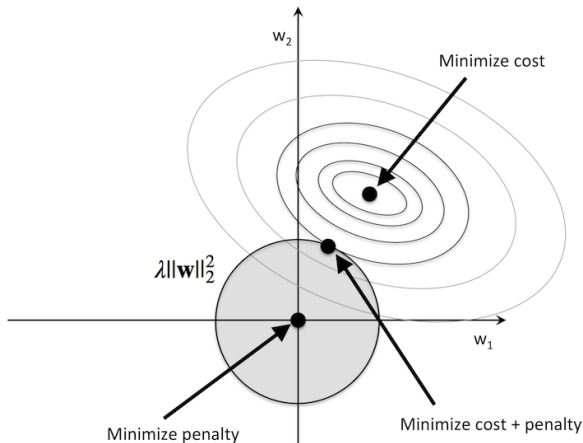
- ▶ Logistic regression:

$$J(\theta) = \text{NLL}(\theta) + \frac{\lambda}{2} \theta^T \theta$$

- From a Bayesian perspective, l2-regularization corresponds to a Gaussian prior on the parameters.

L2-Regularization

- The surface of the objective function is now a combination of the original cost, and the regularization penalty.



L2-Regularization

- Gradient of regularization term:

$$\nabla_{\theta} \frac{\lambda}{2} \boldsymbol{\theta}^T \boldsymbol{\theta} = \lambda \boldsymbol{\theta}$$

- Gradient descent for regularized cost function:

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \eta \nabla_{\theta} (NLL(\boldsymbol{\theta}_t) + \lambda \boldsymbol{\theta}_t^T \boldsymbol{\theta}_t)$$

$$\Leftrightarrow$$

$$\boldsymbol{\theta}_{t+1} := (1 - \eta\lambda) \boldsymbol{\theta}_t - \eta \nabla_{\theta} NLL(\boldsymbol{\theta}_t)$$