Machine Learning Basics III

Benjamin Roth, Nina Poerner

CIS LMU München

Outline

1. Deep Feedforward Networks
   - Motivation
   - Non-Linearities
   - Training
2. Regularization

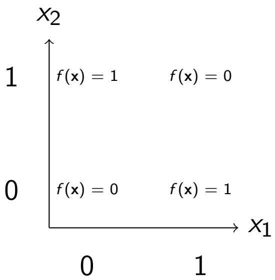
Outline

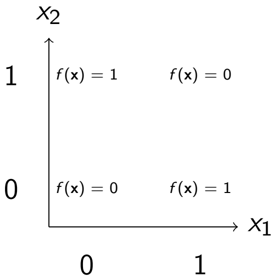
Why Regression is not Enough

- Let $x_1, x_2 \in \{0, 1\}$
- We want XOR function, s.t.

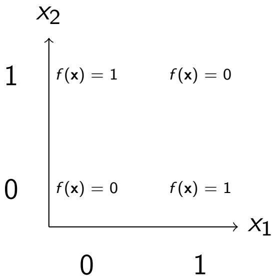
$$f(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 \neq x_2 \\ 0 & \text{otherwise} \end{cases}$$

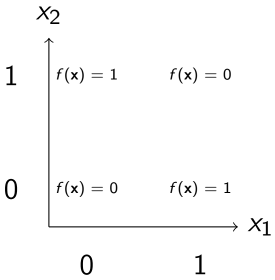
- Can we learn this function using only logistic regression?



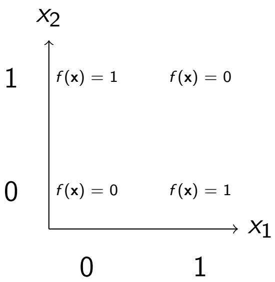

x_2
1
$f(\mathbf{x}) = 1$
$f(\mathbf{x}) = 0$
0
$f(\mathbf{x}) = 0$
$f(\mathbf{x}) = 1$
x_1
0
1

- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$



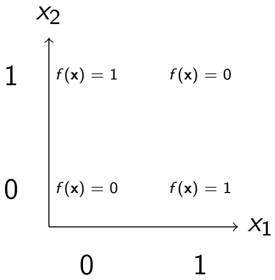


- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

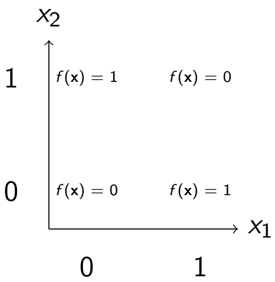
$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

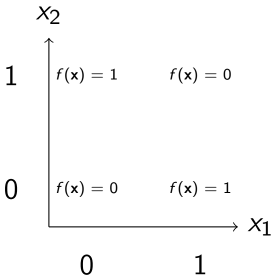
- $\theta_0 \leq 0$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

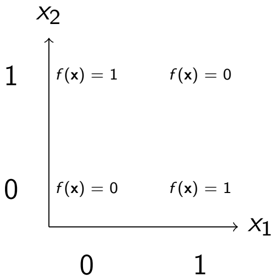
- $\theta_0 \leq 0$
- $\theta_0 + \theta_1 > 0$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

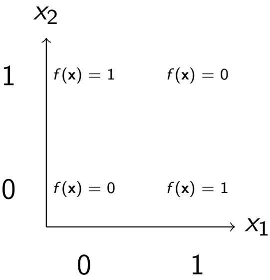
- $\theta_0 \leq 0$
- $\theta_0 + \theta_1 > 0$
- $\theta_0 + \theta_2 > 0 \implies \theta_2 > 0$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

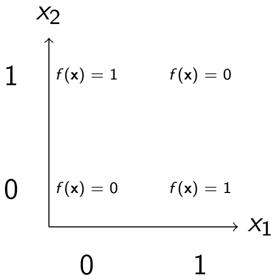
- $\theta_0 \leq 0$
- $\theta_0 + \theta_1 > 0$
- $\theta_0 + \theta_2 > 0 \implies \theta_2 > 0$
- $\theta_0 + \theta_1 + \theta_2 \leq 0$



- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\theta_0 \leq 0$
- $\theta_0 + \theta_1 > 0$
- $\theta_0 + \theta_2 > 0 \implies \theta_2 > 0$
- $\theta_0 + \theta_1 + \theta_2 \leq 0$ XXX

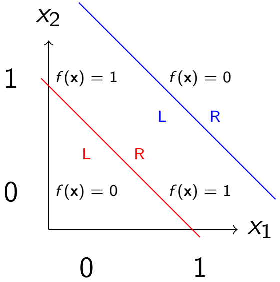


- $g(x_1, x_2) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
- $f(x_1, x_2) = \begin{cases} 1 & \text{if } g(x_1, x_2) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

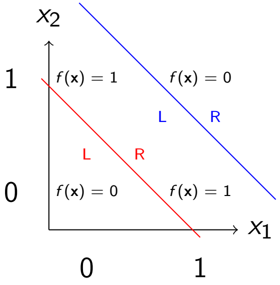
$$\implies f(x_1, x_2) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$

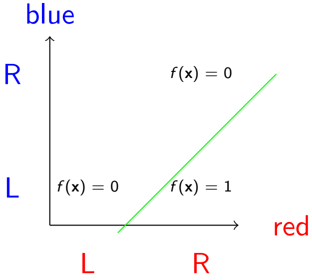
- $\theta_0 \leq 0$
- $\theta_0 + \theta_1 > 0$
- $\theta_0 + \theta_2 > 0 \implies \theta_2 > 0$
- $\theta_0 + \theta_1 + \theta_2 \leq 0$ XXX
- The classes are not linearly separable!

Why Regression is not Enough

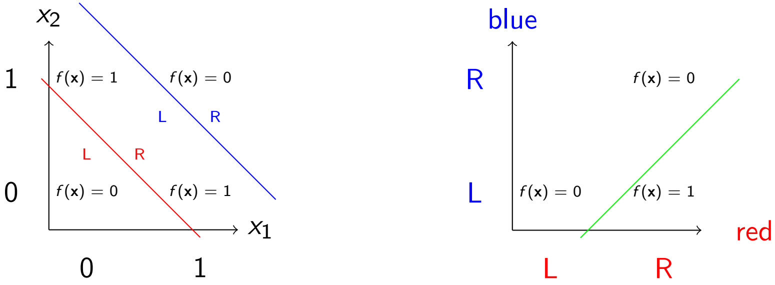


Why Regression is not Enough



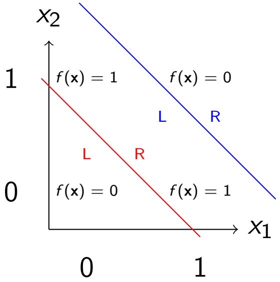


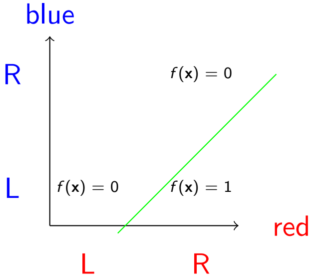
Why Regression is not Enough



- $b(x_1, x_2) = \sigma(\theta_{b0} + \theta_{b1} \cdot x_1 + \theta_{b2} \cdot x_2)$
- $r(x_1, x_2) = \sigma(\theta_{r0} + \theta_{r1} \cdot x_1 + \theta_{r2} \cdot x_2)$

Why Regression is not Enough





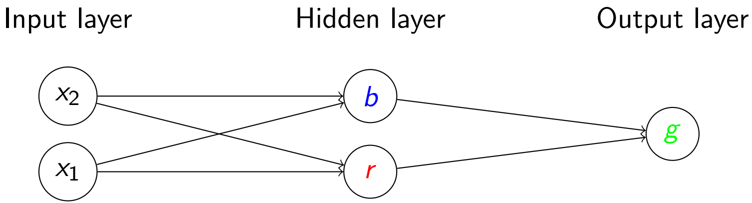
- $b(x_1, x_2) = \sigma(\theta_{b0} + \theta_{b1} \cdot x_1 + \theta_{b2} \cdot x_2)$
- $r(x_1, x_2) = \sigma(\theta_{r0} + \theta_{r1} \cdot x_1 + \theta_{r2} \cdot x_2)$
- $g(x_1, x_2) = \sigma(\theta_{g0} + \theta_{g1} \cdot b(x_1, x_2) + \theta_{g2} \cdot r(x_1, x_2))$
- $f(x_1, x_2) = \mathbb{I}[g(x_1, x_2) > 0.5]$

Deep Feedforward Networks

- *Network*: $f(\mathbf{x};\boldsymbol{\theta})$ is a *composition* of two or more functions $f^{(n)}$
- e.g., $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$
- Each $f^{(n)}$ represents one *layer* in the network.
- Input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

Deep Feedforward Networks

- *Network*: $f(\mathbf{x};\boldsymbol{\theta})$ is a *composition* of two or more functions $f^{(n)}$
- e.g., $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$
- Each $f^{(n)}$ represents one *layer* in the network.
- Input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

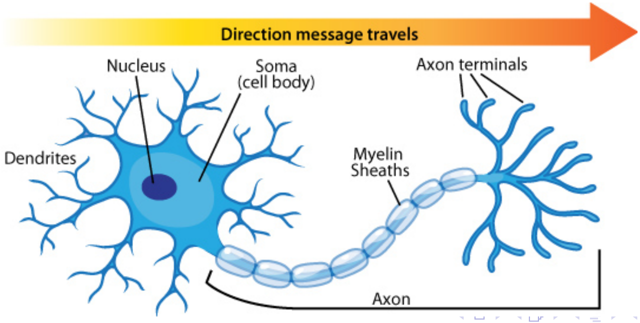


Outline

1. Deep Feedforward Networks
   - Motivation
   - Non-Linearities
   - Training
2. Regularization

"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)
- **all-or-none** $\approx$ nonlinear

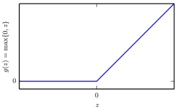


Why we need Non-Linearities

- Fully linear multi-layer neural networks are not very expressive:
- $f(x_1, x_2) = \theta_{g1}(\theta_{r1}x_1 + \theta_{r2}x_2) + \theta_{g2}(\theta_{b1}x_1 + \theta_{b2}x_2)$
  $\iff f(x_1, x_2) = (\theta_{g1}\theta_{r1} + \theta_{g2}\theta_{b1})x_1 + (\theta_{g1}\theta_{r2} + \theta_{g2}\theta_{b2})x_2$
- Apply non-linear *activation functions* to neurons!

Non-Linearities for Hidden Layers

- Rectified Linear Unit (relu)
  - $\mathrm{relu}(z) = \max(0, z)$
  - relu has consistent gradient of 1 when a neuron is *active*, but zero gradient otherwise



- Two-layer FFN with relu can solve XOR:

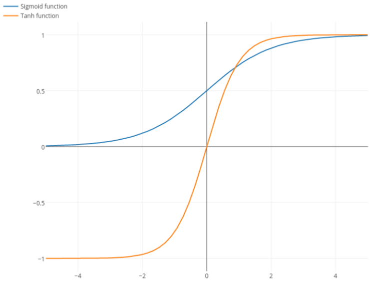
$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}, \mathbf{v}) = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x} + \mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Question: Would this FFN still solve XOR if we remove relu? Why not?

Non-Linearities for Hidden Layers (contd.)

- $\sigma(z)$
- $\tanh(z) = 2\sigma(2z) - 1$
- Sigmoidal functions have only a small "linear" region before they saturate ("flatten out") in both directions.
- This means that gradients become very small for big inputs
- Practice shows that this is okay in conjunction with log-loss



Non-Linearities for Output Units

- Depends on what you are trying to predict!
- If you are predicting a real number (e.g., house price), a linear activation might work...
- For classification:
  - To predict every class individually:
    - Elementwise σ
    - $\rightarrow$ no constraints on how many classes can be true
    - n independent Bernouilli distributions
  - To select one out of n classes:
    - $\mathrm{softmax}(\mathbf{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$
    - $\rightarrow$ all probabilities sum to 1.
    - Multinoulli (categorical) distribution.

Outline

1. Deep Feedforward Networks
   - Motivation
   - Non-Linearities
   - Training
2. Regularization

Deep Feedforward Networks: Training

- Loss function defined on output layer, e.g. $||\mathbf{y} - f(\mathbf{x}; \boldsymbol{\theta})||_2$
- No loss function defined directly on hidden layers
- Instead, training algorithm must decide how to use hidden layers most effectively to minimize the loss on output layer

Deep Feedforward Networks: Training

- Loss function defined on output layer, e.g. $||\mathbf{y} - f(\mathbf{x}; \boldsymbol{\theta})||_2$
- No loss function defined directly on hidden layers
- Instead, training algorithm must decide how to use hidden layers most effectively to minimize the loss on output layer
- Hidden layers can be viewed as providing a complex, more useful feature function $\phi(\mathbf{x})$ of the input (e.g., blue and red separators)
- Conceptually similar to hand-engineered input features to linear models, but fully data-driven

Backpropagation

- Forward propagation: Input information x propagates through network to produce output $\hat{y}$.
- Calculate cost $J(\theta)$, as you would with regression.
- Compute gradients w.r.t. all model parameters θ...
- ... how?
  - We know how to compute gradients w.r.t. parameters of the output layer (just like regression).
  - How to calculate them w.r.t. parameters of the hidden layers?

Chain Rule of Calculus

- Let $x, y, z \in \mathbb{R}$.
- Let functions $f, g : \mathbb{R} \to \mathbb{R}$.
- $y = g(x)$
- $z = f(g(x))$
- Then

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

Chain Rule of Calculus: Vector-valued Functions

- Let $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n, z \in \mathbb{R}$
- Let functions $f : \mathbb{R}^n \rightarrow \mathbb{R}, g : \mathbb{R}^m \rightarrow \mathbb{R}^n$
- $\mathbf{y} = g(\mathbf{x})$
- $z = f(g(\mathbf{x})) = f(\mathbf{y})$
- 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 is the matrix of all first-order partial derivatives of a vector-valued function.

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & & \frac{\partial y_2}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

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

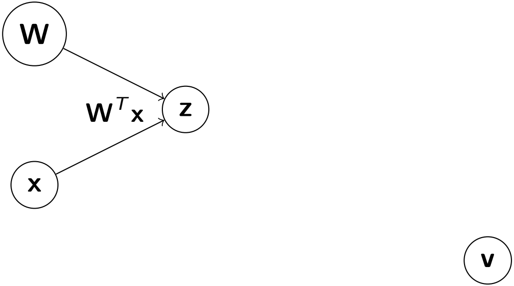
$$\underset{m\times 1}{\nabla_{\mathbf{x}} z} = \underset{n\times m}{\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)}^T \underset{n\times 1}{\nabla_{\mathbf{y}} z}$$

Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
- $\hat{y} = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x})$
- $\mathbf{z} = \mathbf{W}^T \mathbf{x}; \mathbf{r} = \mathrm{relu}(\mathbf{z})$

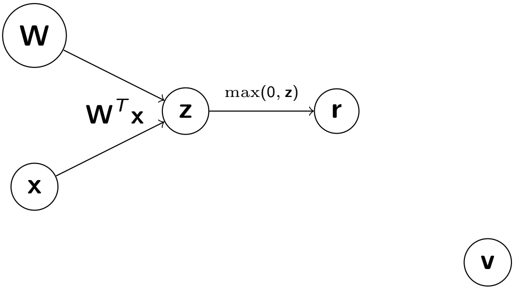
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
- $\hat{y} = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x})$
- $\mathbf{z} = \mathbf{W}^T \mathbf{x}; \mathbf{r} = \mathrm{relu}(\mathbf{z})$



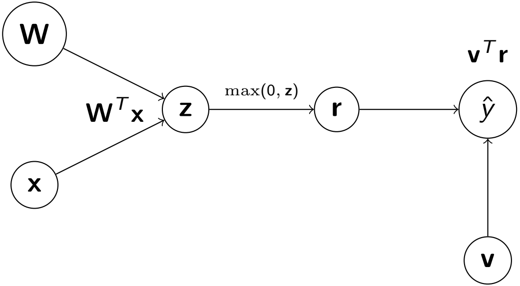
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
- $\hat{y} = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x})$
- $\mathbf{z} = \mathbf{W}^T \mathbf{x}; \mathbf{r} = \mathrm{relu}(\mathbf{z})$



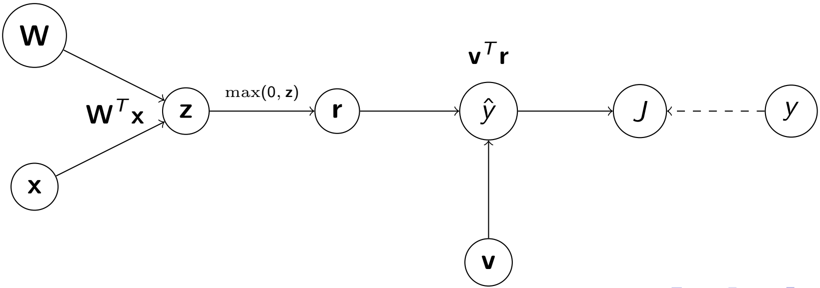
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
- $\hat{y} = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x})$
- $\mathbf{z} = \mathbf{W}^T \mathbf{x}; \mathbf{r} = \mathrm{relu}(\mathbf{z})$



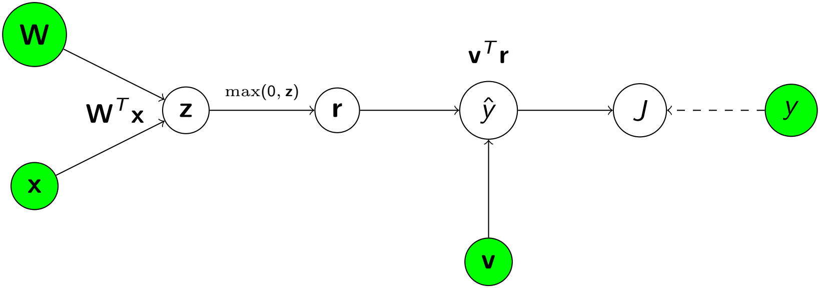
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
- $\hat{y} = \mathbf{v}^T \mathrm{relu}(\mathbf{W}^T \mathbf{x})$
- $\mathbf{z} = \mathbf{W}^T \mathbf{x}; \mathbf{r} = \mathrm{relu}(\mathbf{z})$



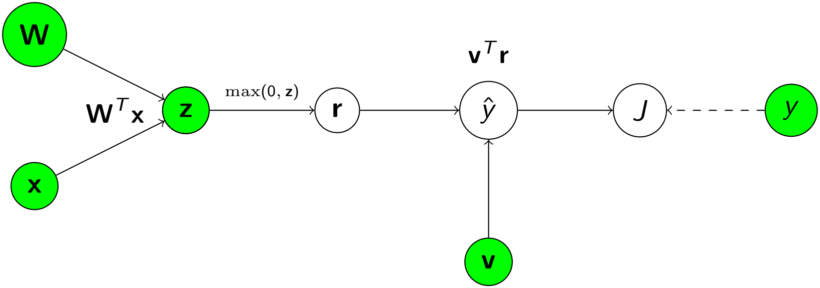
Forward pass

Green: Known or computed node



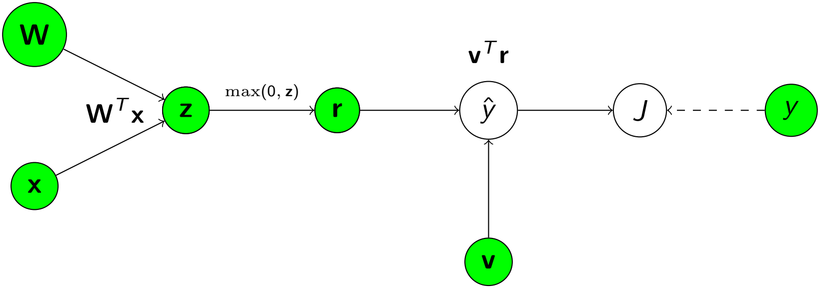
Forward pass

Green: Known or computed node



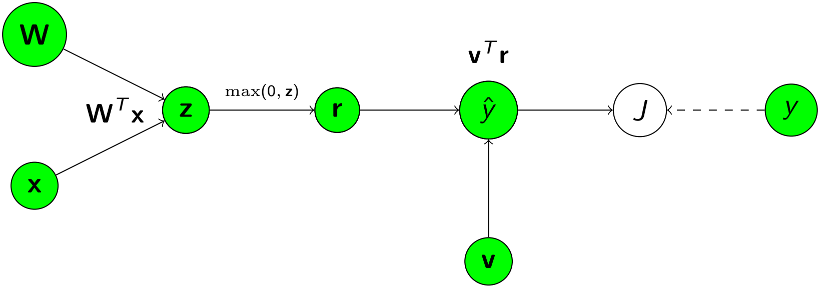
Forward pass

Green: Known or computed node



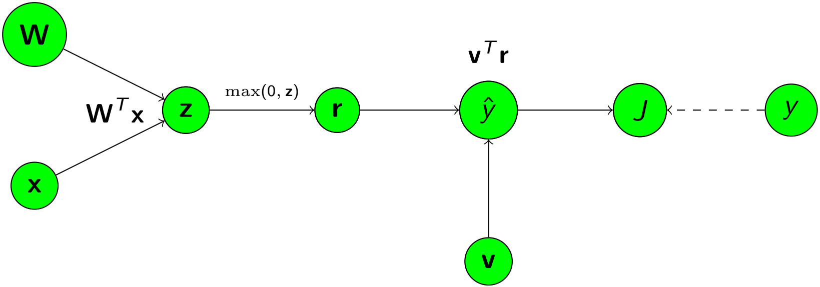
Forward pass

Green: Known or computed node



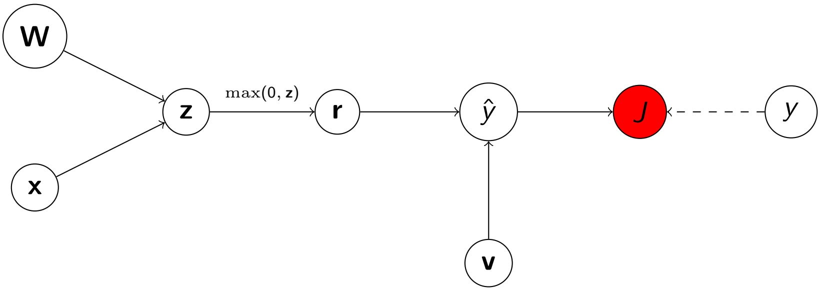
Forward pass

Green: Known or computed node



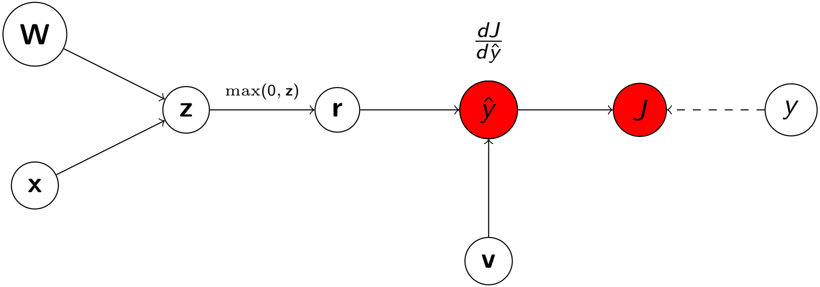
Backward pass

Red: Gradient of J w.r.t. node known or computed



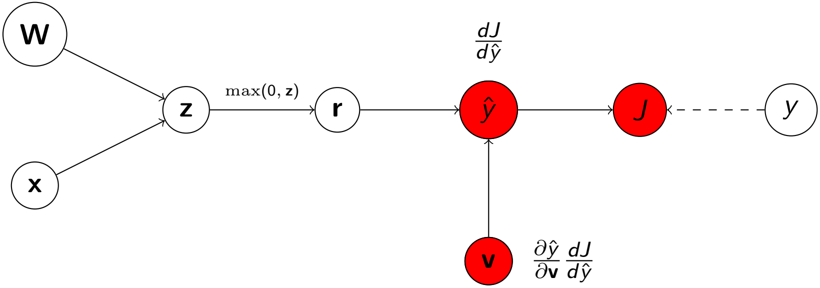
Backward pass

Red: Gradient of J w.r.t. node known or computed



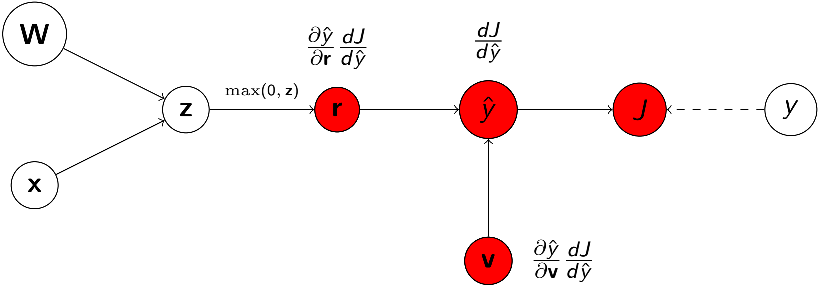
Backward pass

Red: Gradient of J w.r.t. node known or computed



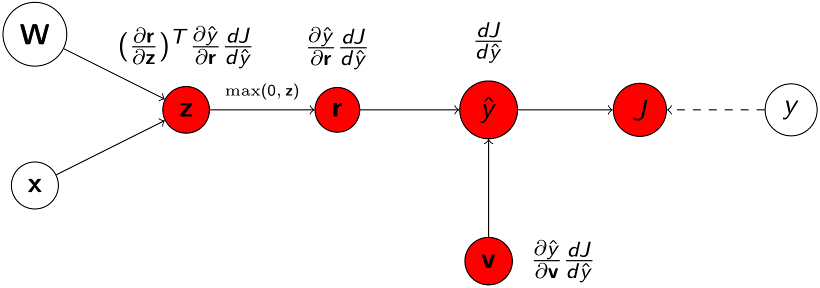
Backward pass

Red: Gradient of J w.r.t. node known or computed



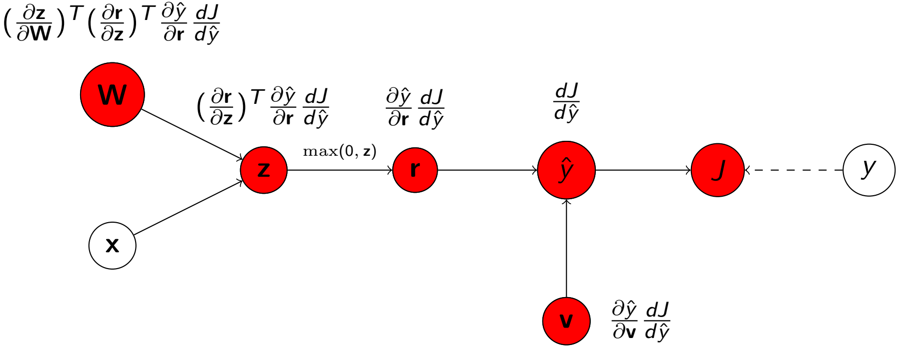
Backward pass

Red: Gradient of J w.r.t. node known or computed



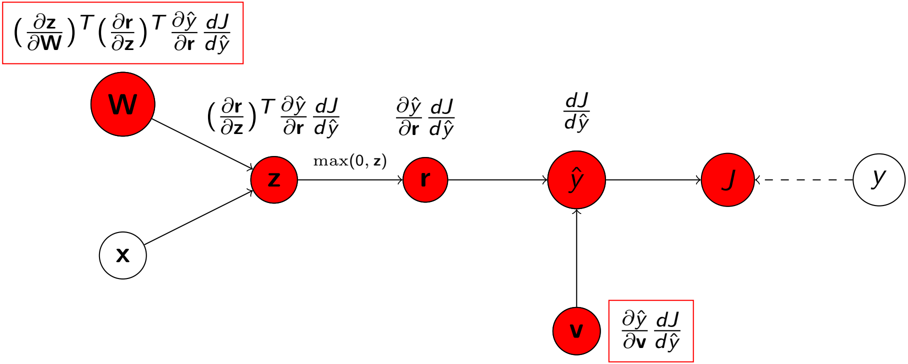
Backward pass

Red: Gradient of J w.r.t. node known or computed



Backward pass

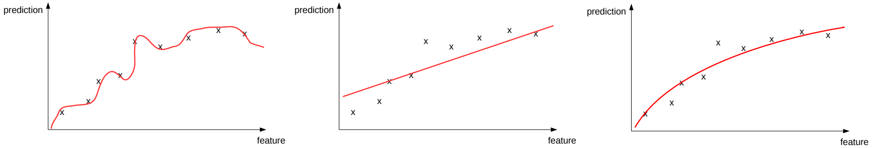
Red: Gradient of J w.r.t. node known or computed



Outline

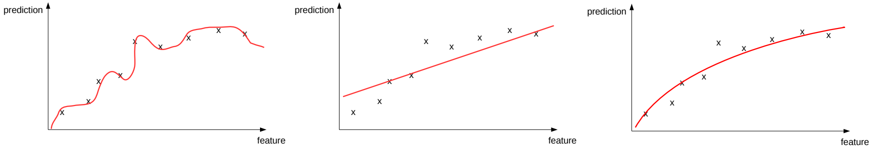
1. Deep Feedforward Networks
   - Motivation
   - Non-Linearities
   - Training
2. Regularization

Regularization



- Overfitting vs. underfitting

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 model for some solutions in hypothesis space
- Unpreferred solutions are penalized: only chosen if they fit training data much better than preferred solutions

Regularization

- Large parameters $\rightarrow$ overfitting
- Prefer models with smaller weights
- Popular regularizers:
  - Penalize large L2 norm (= Euclidian norm) of weight vectors
  - Penalize large L1 norm (= Manhattan norm) of weight vectors

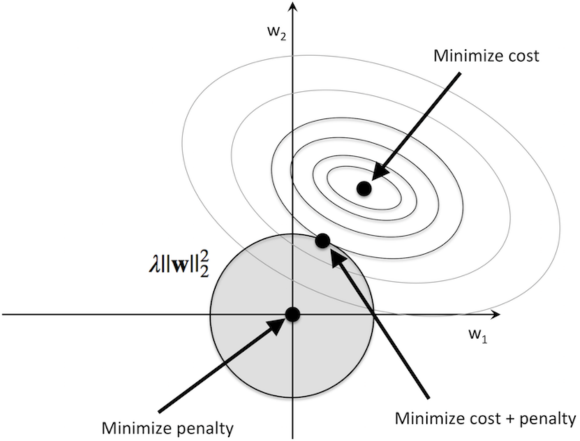
L2-Regularization

- Add term that penalizes large L2 norm of weight vector $\boldsymbol{\theta}$
- The amount of penalty is controlled by a parameter λ

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}, \mathbf{x}, \mathbf{y}) + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

L2-Regularization

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



Summary

- Feedforward networks: layers of (non-linear) function compositions

Summary

- Feedforward networks: layers of (non-linear) function compositions
- Non-Linearities for hidden layers: $\text{relu}, \tanh$, ...

Summary

- Feedforward networks: layers of (non-linear) function compositions
- Non-Linearities for hidden layers: $\mathrm{relu}, \tanh$, ...
- Non-Linearities for output units (classification): $\sigma, \mathrm{softmax}$

Summary

- Feedforward networks: layers of (non-linear) function compositions
- Non-Linearities for hidden layers: $\mathrm{relu}, \tanh$, ...
- Non-Linearities for output units (classification): $\sigma, \mathrm{softmax}$
- Training via backpropagation: compute gradient of cost w.r.t. parameters using chain rule

Summary

- Feedforward networks: layers of (non-linear) function compositions
- Non-Linearities for hidden layers: $\mathrm{relu}, \tanh$, ...
- Non-Linearities for output units (classification): $\sigma, \mathrm{softmax}$
- Training via backpropagation: compute gradient of cost w.r.t. parameters using chain rule
- Regularization: penalize large parameter values, e.g. by adding L2-norm of parameter vector to loss

Outlook

- "Manually" defining forward- and backward passes in numpy is time-consuming
- Deep Learning frameworks let you define forward pass as a "computation graph" made up of simple, differentiable operations (e.g., dot products).
- They do the backward pass for you
- tensorflow + keras, pytorch, theano, MXNet, CNTK, caffe, ...