Neural Networks: Backpropagation & Regularization

Benjamin Roth, Nina Poerner

CIS LMU München

Outline

1. Backpropagation
2. Regularization

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 $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n, z \in \mathbb{R}$
- Let functions $f : \mathbb{R}^n \rightarrow \mathbb{R}, g : \mathbb{R}^m \rightarrow \mathbb{R}^n$
- $\boldsymbol{y} = g(\boldsymbol{x})$
- $z = f(g(\boldsymbol{x})) = f(\boldsymbol{y})$
- 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.

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

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

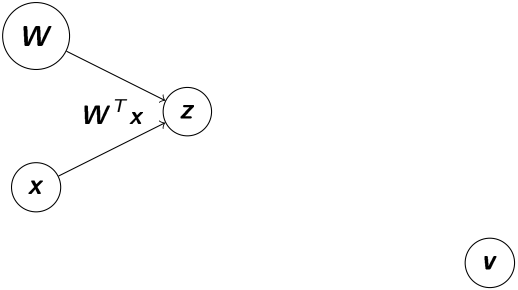
$$\underset{m\times 1}{\nabla_{\boldsymbol{x}} z} = \underset{n\times m}{\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)}^T \underset{n\times 1}{\nabla_{\boldsymbol{y}} z}$$

Viewing the Network as a Graph

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

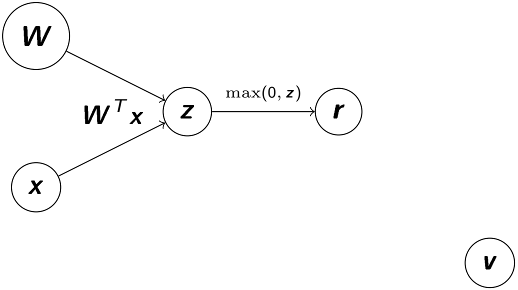
Viewing the Network as a Graph

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



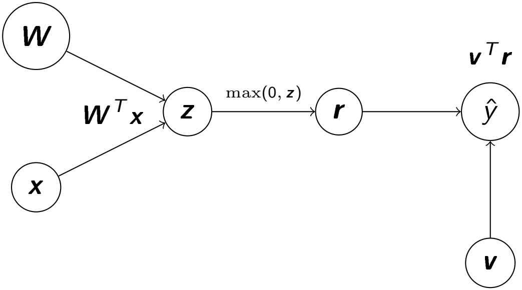
Viewing the Network as a Graph

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



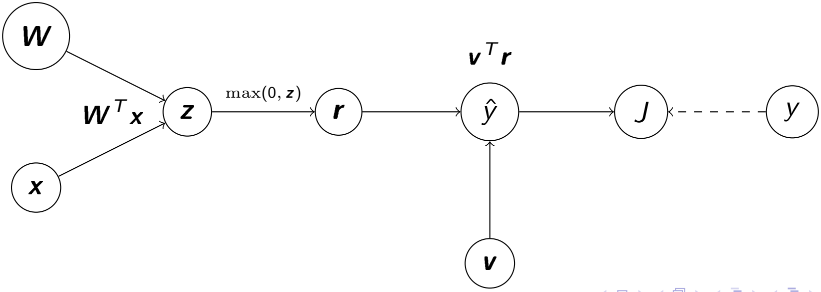
Viewing the Network as a Graph

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



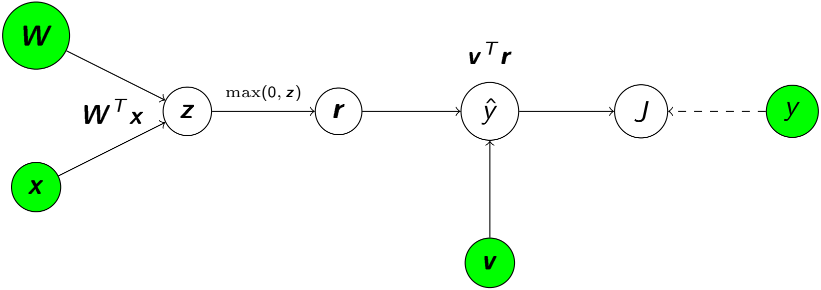
Viewing the Network as a Graph

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



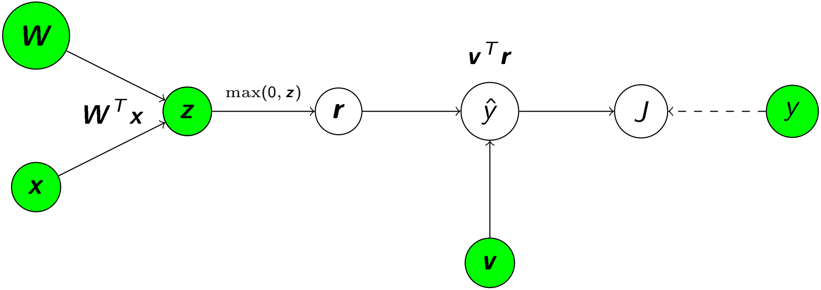
Forward pass

Green: Known or computed node



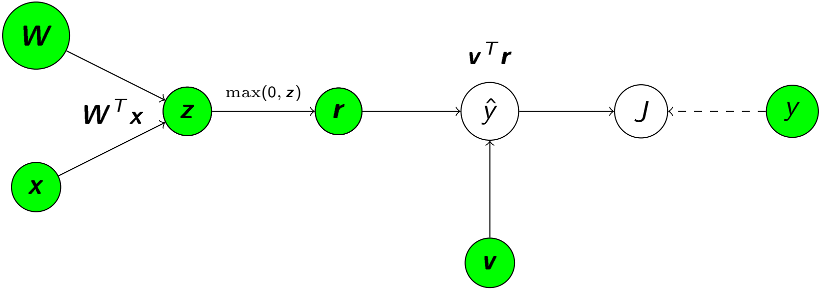
Forward pass

Green: Known or computed node



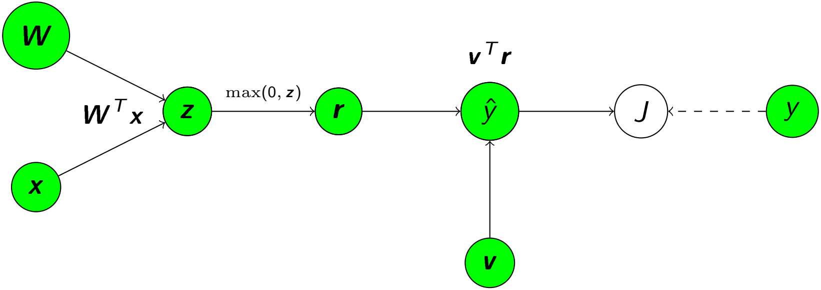
Forward pass

Green: Known or computed node



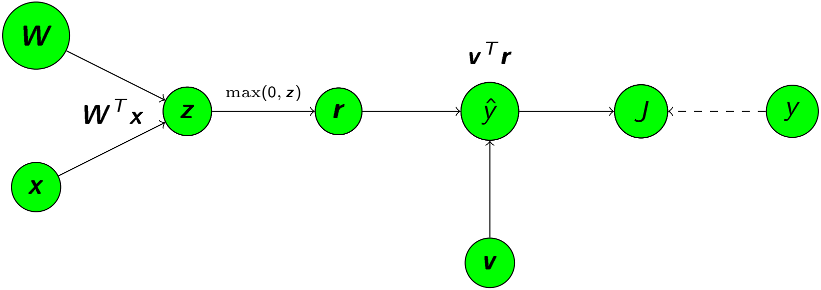
Forward pass

Green: Known or computed node



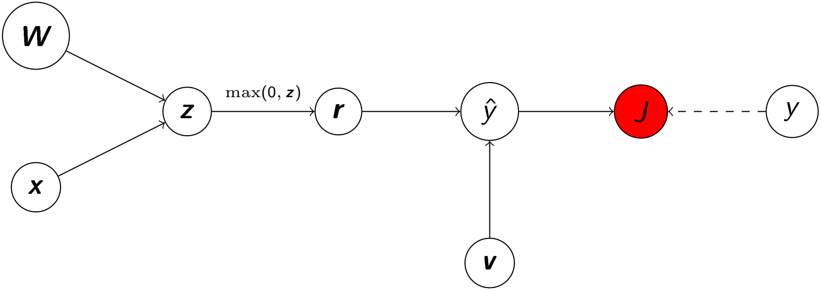
Forward pass

Green: Known or computed node



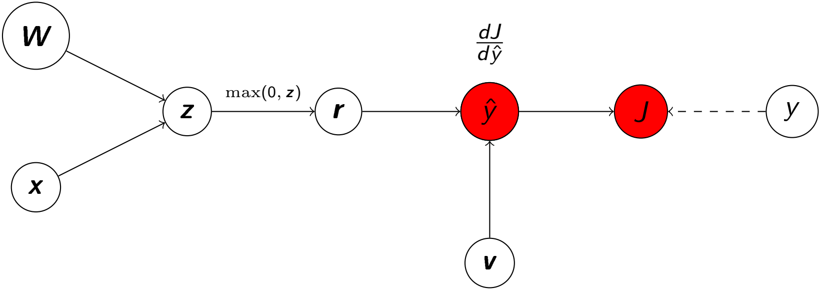
Backward pass

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



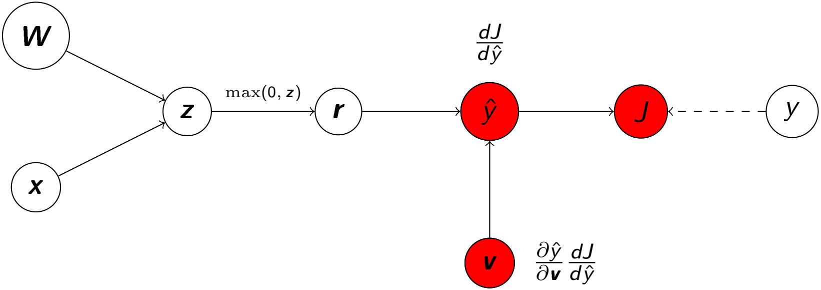
Backward pass

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



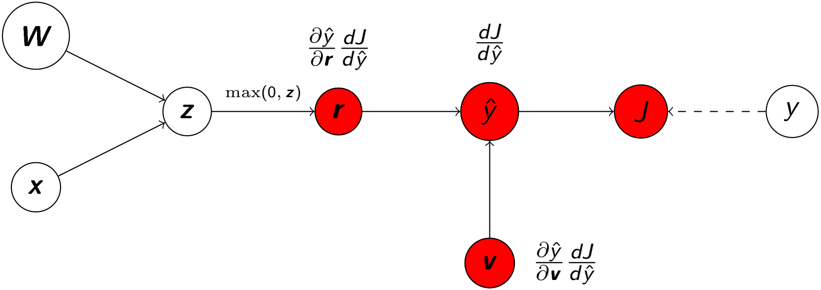
Backward pass

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



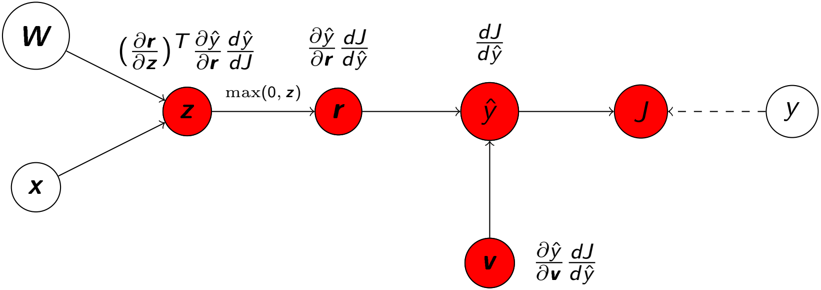
Backward pass

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



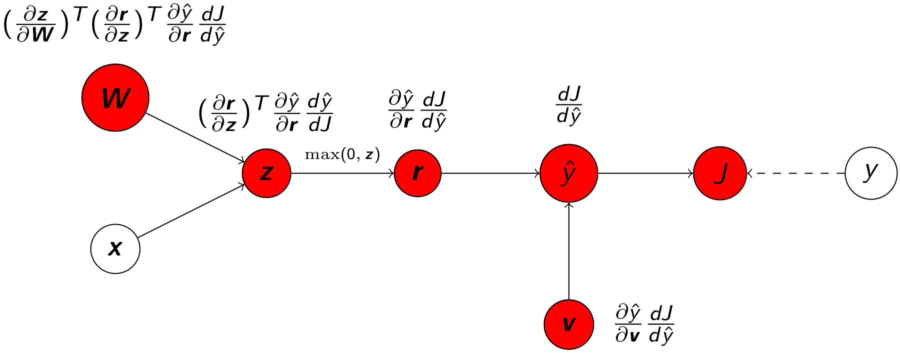
Backward pass

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



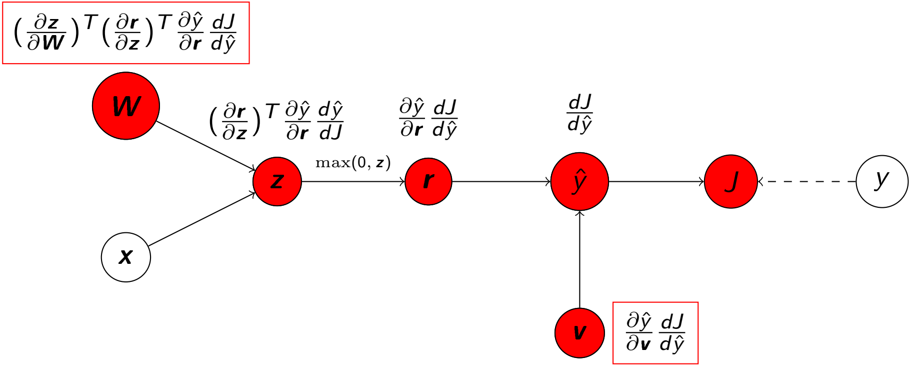
Backward pass

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



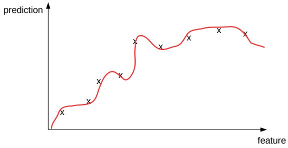
Backward pass

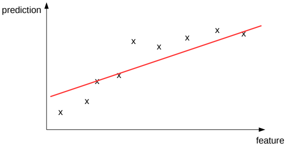
Red: Gradient of J w.r.t. node known or computed

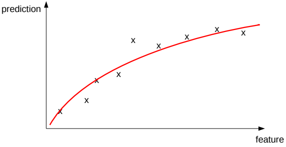


Outline

Regularization

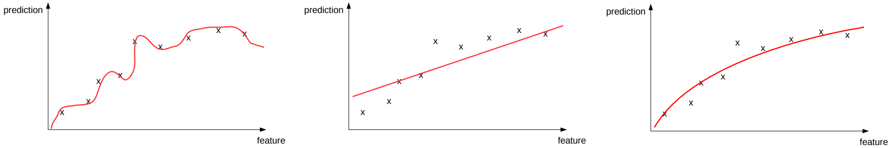






- Overfitting vs. underfitting

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

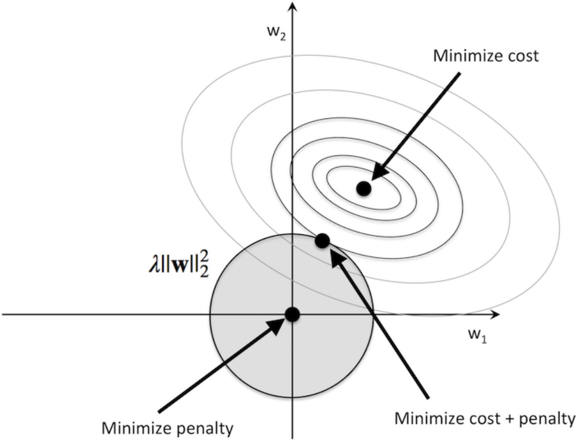
L2-Regularization

- 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: $\mathrm{relu}, \tanh$, ...

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, ...