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

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

How to write in terms of gradients?

We can write the chain rule as:

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

\[
\begin{bmatrix}
m \times 1 \\
\hline
n \times m
\end{bmatrix}
\]

\[
\begin{bmatrix}
n \times 1
\end{bmatrix}
\]
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
  \[ \hat{y} = v^T \text{relu}(W^T x) \]
  \[ z = W^T x; r = \text{relu}(z) \]
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
  - $\hat{y} = v^T \text{relu}(W^T x)$
  - $z = W^T x$; $r = \text{relu}(z)$
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
  \[ \hat{y} = \mathbf{v}^T \text{relu}(\mathbf{W}^T \mathbf{x}) \]
  \[ z = \mathbf{W}^T \mathbf{x}; \ r = \text{relu}(z) \]
Viewing the Network as a Graph

- Nodes are function outputs (can be scalar or vector valued)
- Arrows are functions
- Example:
  - $\hat{y} = v^T \text{relu}(W^T x)$
  - $z = W^T x; r = \text{relu}(z)$
Viewing the Network as a Graph

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

Green: Known or computed node

\[ W^T x \rightarrow z \xrightarrow{\max(0, z)} r \rightarrow \hat{y} \rightarrow J \rightarrow y \]

\[ W^T x \]

\[ r \]

\[ z \]

\[ \max(0, z) \]

\[ \hat{y} \]

\[ J \]

\[ y \]

\[ v^T r \]
Forward pass

Green: Known or computed node

\[ W^T x \rightarrow z \rightarrow r = \max(0, z) \rightarrow \hat{y} \rightarrow y \]

\[ v^T r \rightarrow J \]

Benjamin Roth, Nina Poerner (CIS LMU München)

Neural Networks: Backpropagation & Regularization
Forward pass

Green: Known or computed node

\[ W^T x \rightarrow z \xrightarrow{\max(0, z)} r \rightarrow \hat{y} \rightarrow J \rightarrow y \]
Forward pass

\[ W^T x \rightarrow z \rightarrow r \rightarrow \hat{y} \rightarrow J \rightarrow y \]

**Green:** Known or computed node

\[ W \]

\[ W^T \]

\[ \max(0, z) \]

\[ v^T r \]

\[ v \]

\[ J \]
Forward pass

**Green**: Known or computed node

\[ W^T x \rightarrow z \rightarrow max(0, z) \rightarrow r \rightarrow \hat{y} \rightarrow J \rightarrow y \]

Here, \( W \) is the weight matrix, \( x \) is the input, \( z \) is the computed node, \( r \) is the result of applying a non-linearity, \( \hat{y} \) is the predicted output, and \( y \) is the actual output.

\[ W^T x \]

\[ max(0, z) \]

\[ v^T r \]

\[ J \]

\[ y \]
Backward pass

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

$$
\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial y} \left( \frac{\partial y}{\partial \hat{y}} \right)
$$

Diagram:

- $W$ to $z$ (max(0, z) to $r$
- $x$ to $z$
- $r$ to $\hat{y}$
- $\hat{y}$ to $J$
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

$$W \begin{bmatrix} \frac{\partial r}{\partial z} \end{bmatrix}^T \frac{\partial \hat{y}}{\partial r} \frac{d\hat{y}}{dJ} + \frac{\partial \hat{y}}{\partial r} \frac{dJ}{d\hat{y}}$$

$$z \rightarrow r \rightarrow \hat{y} \rightarrow J \rightarrow y$$

$$x \rightarrow z \rightarrow r \rightarrow \hat{y} \rightarrow v \rightarrow \hat{y}$$

$$\frac{\partial \hat{y}}{\partial v} \frac{dJ}{d\hat{y}}$$
Backward pass

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

\[(\frac{\partial z}{\partial W})^T (\frac{\partial r}{\partial z})^T \frac{\partial \hat{y}}{\partial r} \frac{dJ}{d\hat{y}}\]
Backward pass

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

$\left(\frac{\partial z}{\partial W}\right)^T \left(\frac{\partial r}{\partial z}\right)^T \frac{\partial \hat{y}}{\partial r} \frac{dJ}{d\hat{y}}$

Diagram:

- $W$
- $z$
- $r$
- $\hat{y}$
- $J$
- $y$
- $x$
- $v$

$\frac{\partial \hat{y}}{\partial v} \frac{dJ}{d\hat{y}}$

$\max(0, z)$
Outline

1 Backpropagation

2 Regularization
Regularization

- Overfitting vs. underfitting
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 $\theta$
- The amount of penalty is controlled by a parameter $\lambda$

$$J'(\theta) = J(\theta, x, y) + \frac{\lambda}{2} \theta^T \theta$$
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: \texttt{relu}, \texttt{tanh}, ...

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
Summary

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

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