Machine Learning

A. G. Schwing & M. Telgarsky

University of Illinois at Urbana-Champaign, 2018

L9: Deep Neural Networks

Goals of this lecture

Goals of this lecture

- Understanding forward and backward pass

Goals of this lecture

- Understanding forward and backward pass
- Learning about backpropagation

Goals of this lecture

- Understanding forward and backward pass
- Learning about backpropagation

Reading material

Goals of this lecture

- Understanding forward and backward pass
- Learning about backpropagation

Reading material

- I. Goodfellow et al.; Deep Learning; Chapters 6-9

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \boldsymbol{w}^T \psi(x^{(i)}, \hat{y}))}{\epsilon} - \boldsymbol{w}^T \psi(x^{(i)}, y^{(i)}) \right)$$

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \boldsymbol{w}^T \psi(x^{(i)}, \hat{y}))}{\epsilon} - \boldsymbol{w}^T \psi(x^{(i)}, y^{(i)}) \right)$$

What is a possible issue/limitation?

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \boldsymbol{w}^T \psi(x^{(i)}, \hat{y}))}{\epsilon} - \boldsymbol{w}^T \psi(x^{(i)}, y^{(i)}) \right)$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x, y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \boldsymbol{w}^T\psi(x^{(i)}, \hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)}, y^{(i)}) \right)$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x, y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}}\left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)})\right)$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x,y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Replace $\boldsymbol{w}^T\psi(x,y)$ with a general function $F(\boldsymbol{w},x,y) \in \mathbb{R}$

Recap: Our earlier framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + \boldsymbol{w}^T\psi(x^{(i)},\hat{y}))}{\epsilon} - \boldsymbol{w}^T\psi(x^{(i)},y^{(i)}) \right)$$

What is a possible issue/limitation?

Linearity in the feature space $\psi(x, y)$. Fix: use kernels. But still learning a model **linear** in the parameters $\boldsymbol{w}$

How to fix this?

Replace $\boldsymbol{w}^T\psi(x, y)$ with a general function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(\boldsymbol{w}, x^{(i)},\hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)},y^{(i)}) \right)$$

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to get to

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)})\right)$$

How to get to

- **Logistic regression**

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to get to

- **Logistic regression**
- **Binary SVM**

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to get to

- **Logistic regression**
- **Binary SVM**
- **Multiclass regression**

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2+\sum_{i\in\mathcal{D}}\left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y})+F(\boldsymbol{w},x^{(i)},\hat{y})}{\epsilon} - F(\boldsymbol{w},x^{(i)},y^{(i)})\right)$$

How to get to

- **Logistic regression**
- **Binary SVM**
- **Multiclass regression**
- **Multiclass SVM**

General framework:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\epsilon} - F(\boldsymbol{w}, x^{(i)}, y^{(i)})\right)$$

How to get to

- **Logistic regression**
- **Binary SVM**
- **Multiclass regression**
- **Multiclass SVM**
- **Deep Learning**

Deep Learning:

What function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$ to choose? ($y \in \{1, \ldots, K\}$)

Deep Learning:

What function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$ to choose? ($y \in \{1, \ldots, K\}$)

- Choose any differentiable composite function

$$F(\boldsymbol{w}, x, y) = f_1(\boldsymbol{w}_1, y, f_2(\boldsymbol{w}_2, f_3(\ldots f_n(\boldsymbol{w}_n, x)\ldots))) \in \mathbb{R}$$

Deep Learning:

What function $F(\boldsymbol{w}, x, y) \in \mathbb{R}$ to choose? ($y \in \{1, \ldots, K\}$)

- Choose any differentiable composite function

$$F(\boldsymbol{w}, x, y) = f_1(\boldsymbol{w}_1, y, f_2(\boldsymbol{w}_2, f_3(\ldots f_n(\boldsymbol{w}_n, x)\ldots))) \in \mathbb{R}$$

- More generally: functions can be represented by an acyclic graph (computation graph)

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

Nodes are

Example:

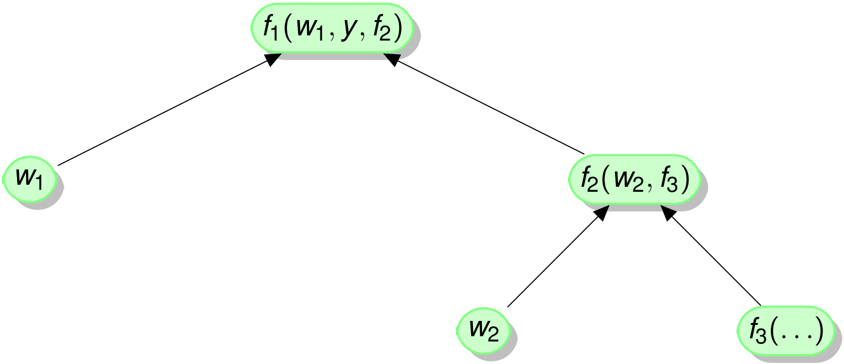
$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

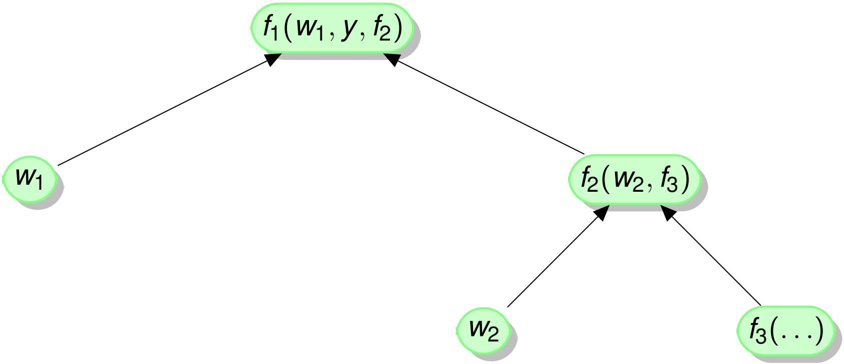
Nodes are weights, data, and functions:



Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:



Internal representation used by deep net packages.

What are the individual functions/layers f_1, f_2 etc.?

What are the individual functions/layers f_1, f_2 etc.?

- Fully connected layers

What are the individual functions/layers f_1, f_2 etc.?

- Fully connected layers
- Convolutions

What are the individual functions/layers f_1, f_2 etc.?

- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$

What are the individual functions/layers f_1, f_2 etc.?

- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling

What are the individual functions/layers f_1, f_2 etc.?

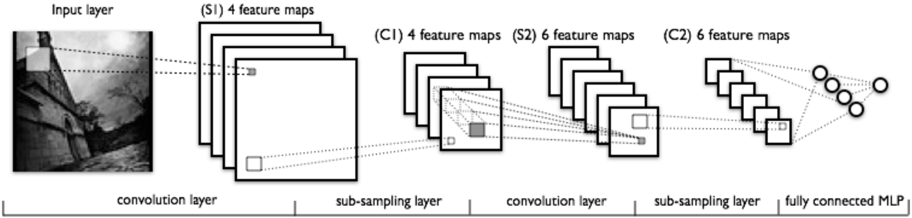
- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer

What are the individual functions/layers f_1, f_2 etc.?

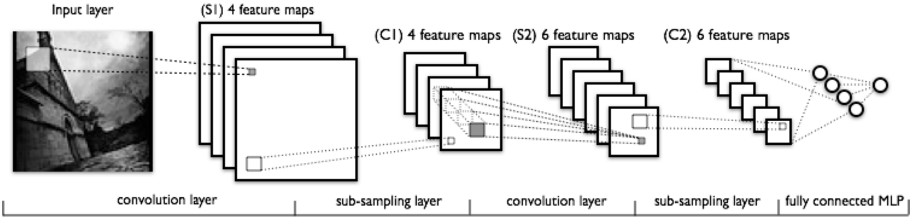
- Fully connected layers
- Convolutions
- Rectified linear units (ReLU): $\max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer
- Dropout

Example function architecture: LeNet

Example function architecture: LeNet



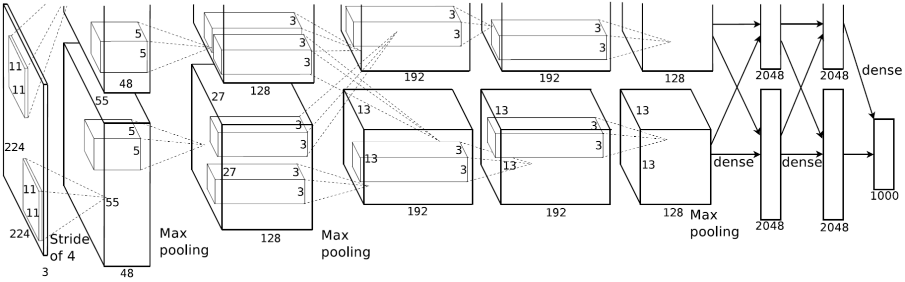
Example function architecture: LeNet



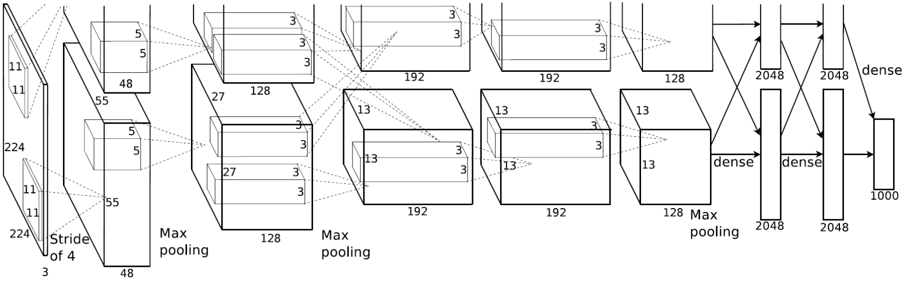
Decreasing spatial resolution and the increasing number of channels

Example function architecture: AlexNet

Example function architecture: AlexNet

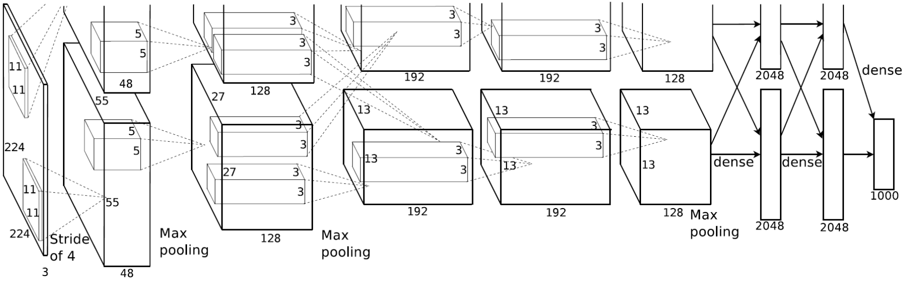


Example function architecture: AlexNet



Decreasing spatial resolution and the increasing number of channels

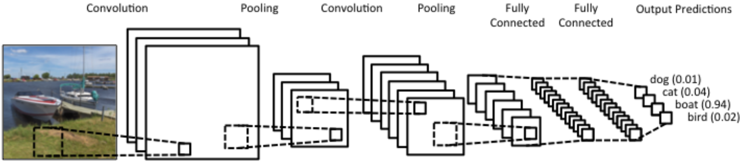
Example function architecture: AlexNet



Decreasing spatial resolution and the increasing number of channels

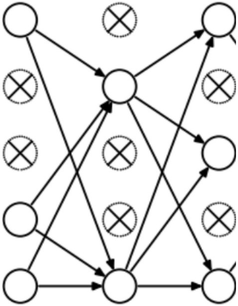
Why is the output 1000-dimensional?

Another deep net:



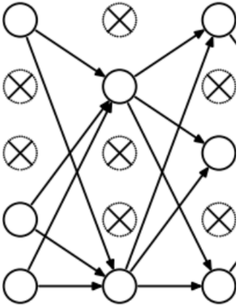
Those nets are structurally simple in that a layer's output is used as input for the next layer. This is not required.

Dropout layer:



Dropout layer:

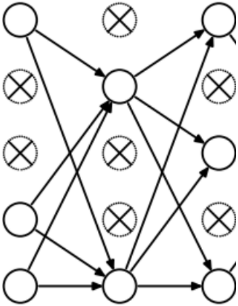
Randomly set activations to zero



Dropout layer:

Randomly set activations to zero

Trainable parameters $\boldsymbol{w}$:

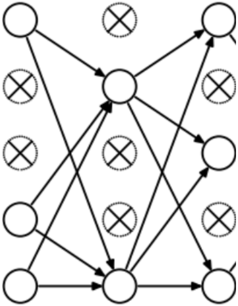


Dropout layer:

Randomly set activations to zero

Trainable parameters $\boldsymbol{w}$:

- None



Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as maximizing the regularized cross entropy:

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\mathrm{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with}$$

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \left\{ \begin{array}{rcl} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{array} \right.$$

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is C?

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}}\left(\ln\sum_{\hat{y}}\exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)})\right)$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}}\sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is C? Weight decay (aka regularization constant)

Deep net training:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

Often also referred to as maximizing the regularized cross entropy:

$$\max_{\boldsymbol{w}} -\frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) \quad \text{with} \quad \begin{cases} p_{\text{GT}}^{(i)}(\hat{y}) & = & \delta(\hat{y} = y^{(i)}) \\ p(\hat{y}|x) & \propto & \exp F(\boldsymbol{w}, x, \hat{y}) \end{cases}$$

What is C? Weight decay (aka regularization constant)

$$\min_{\boldsymbol{w}} \underbrace{\frac{C}{2}\|\boldsymbol{w}\|_2^2}_{\text{weight decay}} \underbrace{- \sum_{i\in\mathcal{D}} \sum_{\hat{y}} p_{\text{GT}}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)})}_{\text{torch.nn.CrossEntropyLoss(gt, } F\text{)}}$$

Program:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to optimize this?

Program:

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

How to optimize this?

Stochastic gradient descent with momentum: What was this again?

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)})\right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)})\right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes logits F as input

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes logits F as input
- $\frac{\partial F(\boldsymbol{w},x,\hat{y})}{\partial \boldsymbol{w}}$

Gradient of

$$\min_{\boldsymbol{w}} \frac{C}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i\in\mathcal{D}} \left(\ln \sum_{\hat{y}} \exp F(\boldsymbol{w}, x^{(i)}, \hat{y}) - F(\boldsymbol{w}, x^{(i)}, y^{(i)}) \right)$$

is?

$$C\boldsymbol{w} + \sum_{i\in\mathcal{D}} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\boldsymbol{w}, x^{(i)}, \hat{y})}{\partial \boldsymbol{w}}$$

How to compute this numerically:

- $p(\hat{y}|x) = \frac{\exp F(\boldsymbol{w},x,\hat{y})}{\sum_{\tilde{y}} \exp F(\boldsymbol{w},x,\tilde{y})}$ via soft-max which takes logits F as input
- $\frac{\partial F(\boldsymbol{w},x,\hat{y})}{\partial \boldsymbol{w}}$ via backpropagation

Backpropagation example:

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 & = & f_3(w_3, x) \\ x_1 & = & f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} =$$

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 = f_3(w_3, x) \\ x_1 = f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} =$$

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

Backpropagation example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 &= f_3(w_3, x) \\ x_1 &= f_2(w_2, x_2) \end{cases}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_3}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

What is $\frac{\partial F(\boldsymbol{w},x,y)}{\partial w_2}$?

$$\frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \underbrace{\frac{\partial f_1}{\partial f_2}} \cdot \frac{\partial f_2}{\partial w_2}$$

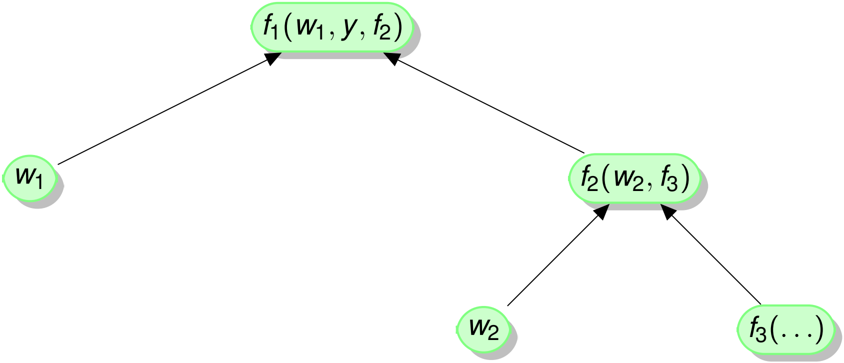
Generally: To avoid repeated computation, backpropagation on an acyclic graph. Nodes in this graph are weights, data, and functions.

Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$

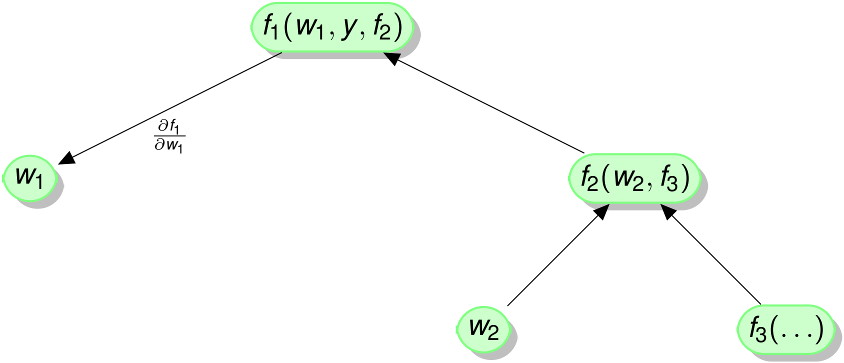
Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



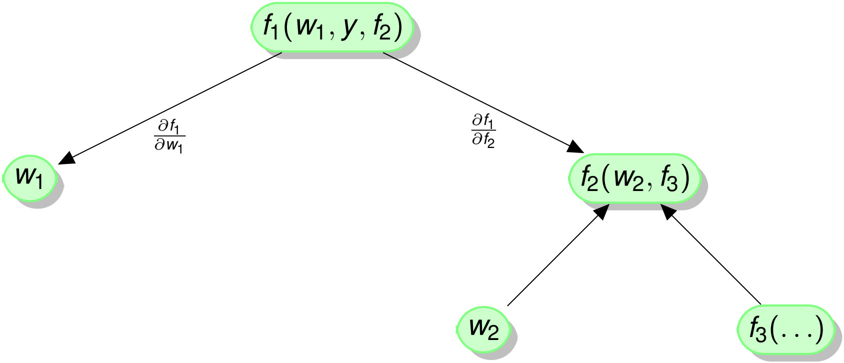
Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



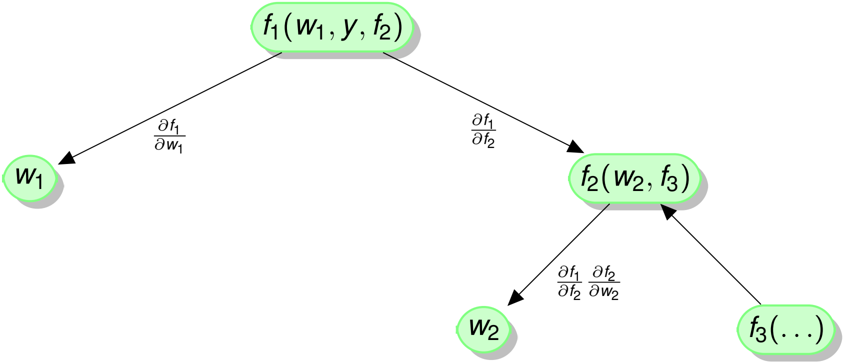
Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



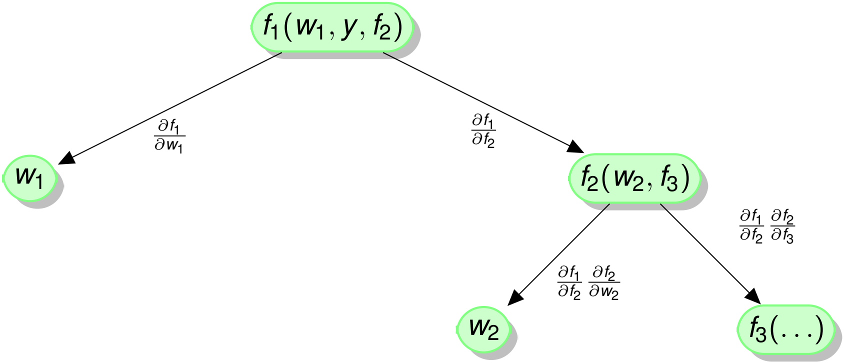
Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



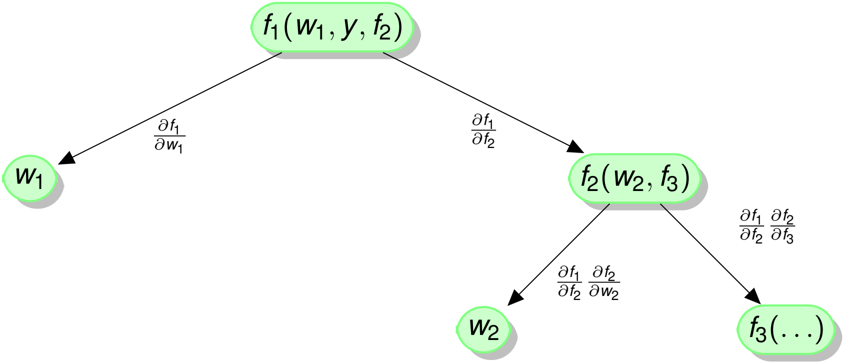
Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



Composite function represented as acyclic graph

$$F(\boldsymbol{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(\ldots)))$$



Repeated use of chain rule for efficient computation of all gradients

What information needs to be stored at a function node:

What information needs to be stored at a function node:

- Inference:

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:
  - Store intermediate results for fully connected layer, convolution

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:
  - Store intermediate results for fully connected layer, convolution
  - Some functions can be combined to reduce intermediate storage, e.g., X + ReLU, X + Sigmoid, X + tanh

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:
  - Store intermediate results for fully connected layer, convolution
  - Some functions can be combined to reduce intermediate storage, e.g., X + ReLU, X + Sigmoid, X + tanh

Difference between activation functions and layers

What information needs to be stored at a function node:

- Inference: we can forget the intermediate result
- Learning:
  - Store intermediate results for fully connected layer, convolution
  - Some functions can be combined to reduce intermediate storage, e.g., X + ReLU, X + Sigmoid, X + tanh

Difference between activation functions and layers

Recommendation: implement a simple deep net framework yourself

Remark:

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

- We are no longer guaranteed to find the global optimum

Remark:

Since $F(\boldsymbol{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex.

Implications:

- We are no longer guaranteed to find the global optimum
- Initialization of $\boldsymbol{w}$ matters

Initialization:

Initialization:

- Not well understood in general

Initialization:

- Not well understood in general
- Needs to break symmetry

Initialization:

- Not well understood in general
- Needs to break symmetry
- Random uniform

$$\text{Uniform}\left(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right)$$

Initialization:

- Not well understood in general
- Needs to break symmetry
- Random uniform

$$\text{Uniform}\left(-\frac{1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}}\right)$$

- Glorot and Bengio (2010)

$$\text{Uniform}\left(-\sqrt{\frac{6}{\text{fan in + fan out}}}, \sqrt{\frac{6}{\text{fan in + fan out}}}\right)$$

Remark:

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

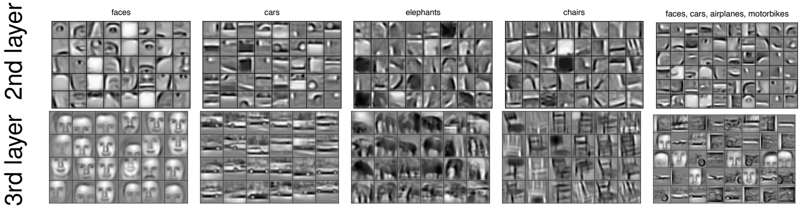
Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output

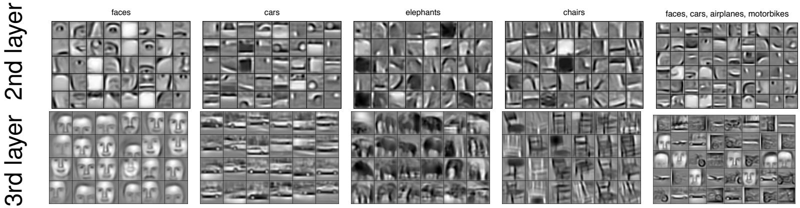


Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



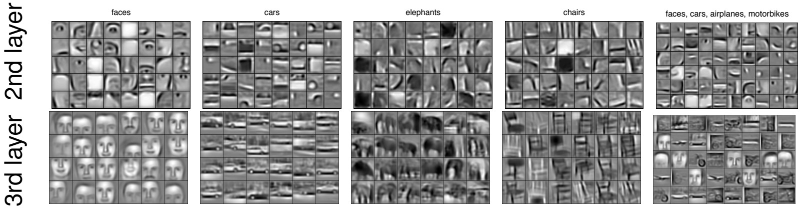
Disadvantage of deep nets compared to usage of features:

Remark:

A deep net with a single fully connected layer is equivalent to logistic regression

Advantages of deep nets compared to usage of hand-crafted features:

Deep nets automatically learn feature space transformations (hierarchical abstractions of data) such that data is easily separable at the output



Disadvantage of deep nets compared to usage of features:

Deep nets are computationally demanding (GPUs) and require significant amounts of training data

Why this recent popularity:

Why this recent popularity:

- Sufficient computational resources

Why this recent popularity:

- Sufficient computational resources
- Sufficient data

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances
- Sufficient evidence that it works

Why this recent popularity:

- Sufficient computational resources
- Sufficient data
- Sufficient algorithmic advances
- Sufficient evidence that it works

This combination lead to significant performance improvements on many datasets

Algorithmic advances:

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Normalizes data when training really deep nets

Algorithmic advances:

- Rectified linear unit ($\max\{0, x\}$) activation as opposed to sigmoid
  - Fixed the vanishing gradient problem for lower layers close to the input
- Dropout
  - Decorrelates different units, i.e., they learn different features
- Good initialization heuristics
  - Less prone to getting stuck in bad local optima
- Batch-Normalization during training
  - Normalizes data when training really deep nets
  - Normalize by subtracting mean and dividing by standard deviation

Choices in deep learning packages:

Choices in deep learning packages:

- Use an appropriate loss function

Choices in deep learning packages:

- Use an appropriate loss function
- Design a composite function $F(\boldsymbol{w}, x, y)$

Choices in deep learning packages:

- Use an appropriate loss function
- Design a composite function $F(\boldsymbol{w}, x, y)$

Know what you are doing, i.e., know all the dimensions.

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

- MSELoss

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

- MSELoss

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

- BCELoss

```
loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

- MSELoss

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

- BCELoss

```
loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

- BCEWithLogitsLoss

```
loss(o,t)=-1/n\sum_i(t[i]*log(sigmoid(o[i]))
                +(1-t[i])*log(1-sigmoid(o[i])))
```

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

- MSELoss

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

- BCELoss

```
loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

- BCEWithLogitsLoss

```
loss(o,t)=-1/n\sum_i(t[i]*log(sigmoid(o[i]))
                +(1-t[i])*log(1-sigmoid(o[i])))
```

- L1Loss

Loss functions:

- CrossEntropyLoss

```
loss(x, class) = -log(exp(x[class]) / (\sum_j exp(x[j])))
               = -x[class] + log(\sum_j exp(x[j]))
```

- NLLLoss

```
loss(x, class) = -x[class]
```

- MSELoss

```
loss(x, y) = 1/n \sum_i |x_i - y_i|^2
```

- BCELoss

```
loss(o,t)=-1/n \sum_i i(t[i]*log(o[i])+(1-t[i])*log(1-o[i]))
```

- BCEWithLogitsLoss

```
loss(o,t)=-1/n\sum_i(t[i]*log(sigmoid(o[i]))
                +(1-t[i])*log(1-sigmoid(o[i])))
```

- L1Loss
- KLDivLoss

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with ‘LogSoftmax’:

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with 'LogSoftmax':

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why?

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with ‘LogSoftmax’:

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why? Numerical robustness (‘log-sum-exp trick’)

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with 'LogSoftmax':

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why? Numerical robustness ('log-sum-exp trick')

$$\log \sum_j \exp x_j = c + \log \sum_j \exp\left(x_j - c\right)$$

Why this form for the NLLLoss?

```
loss(x, class) = -x[class]
```

Intended to be used in combination with 'LogSoftmax':

$$f_i(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

Why? Numerical robustness ('log-sum-exp trick')

$$\log \sum_j \exp x_j = c + \log \sum_j \exp\left(x_j - c\right)$$

Don't try without, it **will** fail!

Example (PyTorchCS446.py):

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Example (PyTorchCS446.py):

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

What are the input dimensions?

Popular architectures:

Popular architectures:

- LeNet

Popular architectures:

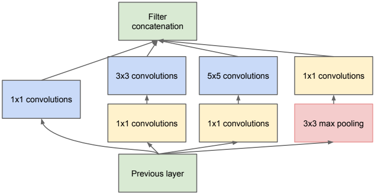
- LeNet
- AlexNet

Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)

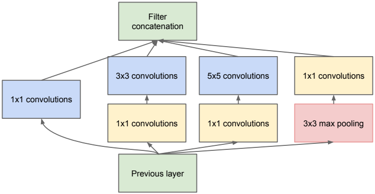
Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)

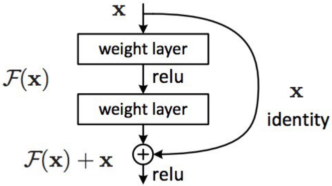


Popular architectures:

- LeNet
- AlexNet
- VGG (16/19 layers, mostly 3x3 convolutions)
- GoogLeNet (inception module)



- ResNet (residual connections)



Imagenet Challenge:

Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories

Imagenet Challenge:

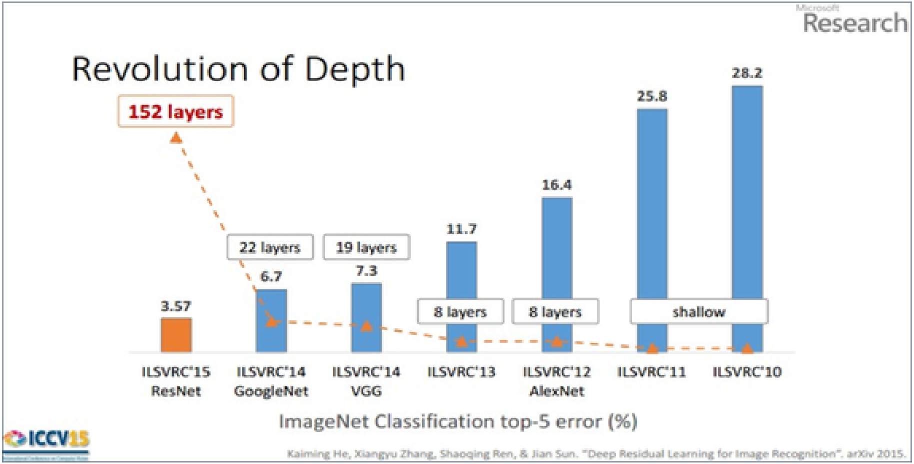
- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources

Imagenet Challenge:

- A large dataset: 1.2M images, 1000 categories
- AlexNet was run on the GPU, i.e., sufficient computational resources
- Rectified linear units rather than sigmoid units simplify optimization

Results:

Results:


Microsoft
Research
Revolution of Depth
152 layers
22 layers
19 layers
3.57
6.7
7.3
11.7
16.4
25.8
28.2
8 layers
8 layers
shallow
ILSVRC'15
ResNet
ILSVRC'14
GoogleNet
ILSVRC'14
VGG
ILSVRC'13
ILSVRC'12
AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Quiz:

Quiz:

- What are deep nets?

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression
- What is back-propagation in deep nets?

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression
- What is back-propagation in deep nets?
- What components of deep nets do you know?

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression
- What is back-propagation in deep nets?
- What components of deep nets do you know?
- What algorithms are used to train deep nets?

Important topics of this lecture

- Deep nets
- Backpropagation

Up next:

- Ensemble methods