Deep Learning (CS324)

3. Deep networks & backpropagation* Continued

Prof. Jianguo Zhang
SUSTech

*Based on http://www.deeplearningbook.org chapters 5 and 6

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

Gradient of loss wrt parameters

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it?

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \frac{\partial L}{\partial a^L} \frac{\partial a^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdots \frac{\partial a^l}{\partial w^l}$$

where l < l + 1 < l +2 < … < L -2 < L - 1 < L

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

Gradient of output layer *l* wrt parameters

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

Can be computed locally: we only need the Jacobian of the *l*-th layer output wrt to its parameters

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

Gradient of loss wrt to output of layer *l*

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

This instead depends on gradient at layer *l+1*

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

This instead depends on gradient at layer *l+1*

$$\frac{\partial L}{\partial a^l} = \left(\frac{\partial a^{l+1}}{\partial a^l}\right)^T \frac{\partial L}{\partial a^{l+1}}$$

Backpropagation

- Goal: computing the gradient of the loss with respect to the parameters so we can descend the gradient according to

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla_{w^{(t)}} L$$

- How to compute it? For the *l*-th layer,

$$\frac{\partial L}{\partial w^l} = \left(\frac{\partial a^l}{\partial w^l}\right)^T \frac{\partial L}{\partial a^l}$$

This instead depends on gradient at layer *l+1*

$$\frac{\partial L}{\partial a^l} = \left(\frac{\partial a^{l+1}}{\partial x^{l+1}}\right)^T \frac{\partial L}{\partial a^{l+1}}$$

Forward & Back algorithm

- **Step 1.** Compute forward propagations for all layers recursively

$$\boxed{a^l = h^l(x^l) \text{ and } x^{l+1} = a^l}$$

- **Step 2.** Once done with forward propagation, follow the reverse path.
  - Start from the last layer and for each new layer compute the gradients
  - Cache computations when possible to avoid redundant operations

$$\boxed{\frac{\partial \mathcal{L}}{\partial a^l} = \left(\frac{\partial a^{l+1}}{\partial x^{l+1}}\right)^T \cdot \frac{\partial \mathcal{L}}{\partial a^{l+1}}} \qquad \boxed{\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial a^l}{\partial w^l} \cdot \left(\frac{\partial \mathcal{L}}{\partial a^l}\right)^T}$$

- **Step 3.** Use the gradients $\frac{\partial \mathcal{L}}{\partial w^l}$ with Stochastic Gradient Descend to train

Forward propagation in MLPs

Algorithm 6.3 Forward propagation through a typical deep neural network and the computation of the cost function. The loss $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ depends on the output $\hat{\boldsymbol{y}}$ and on the target $\boldsymbol{y}$ (see section 6.2.1.1 for examples of loss functions). To obtain the total cost J, the loss may be added to a regularizer $\Omega(\theta)$, where θ contains all the parameters (weights and biases). Algorithm 6.4 shows how to compute gradients of J with respect to parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. For simplicity, this demonstration uses only a single input example $\boldsymbol{x}$. Practical applications should use a minibatch. See section 6.5.7 for a more realistic demonstration.

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

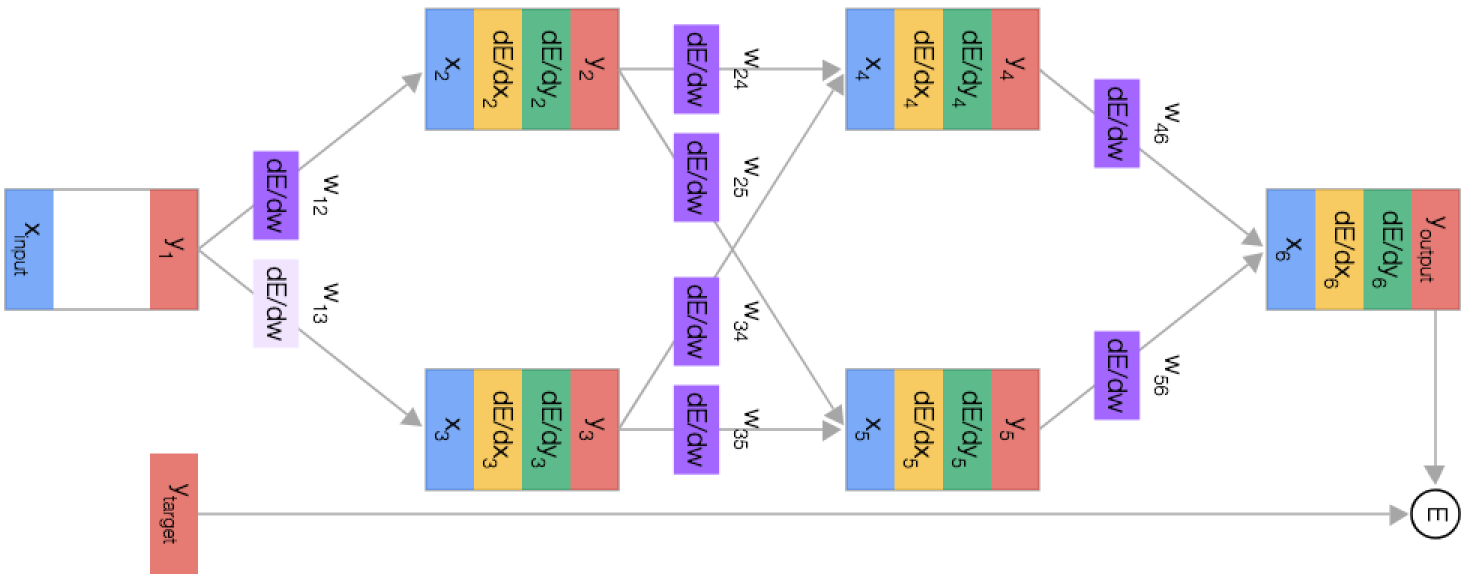
Backpropagation in MLPs

Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses, in addition to the input $\boldsymbol{x}$, a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

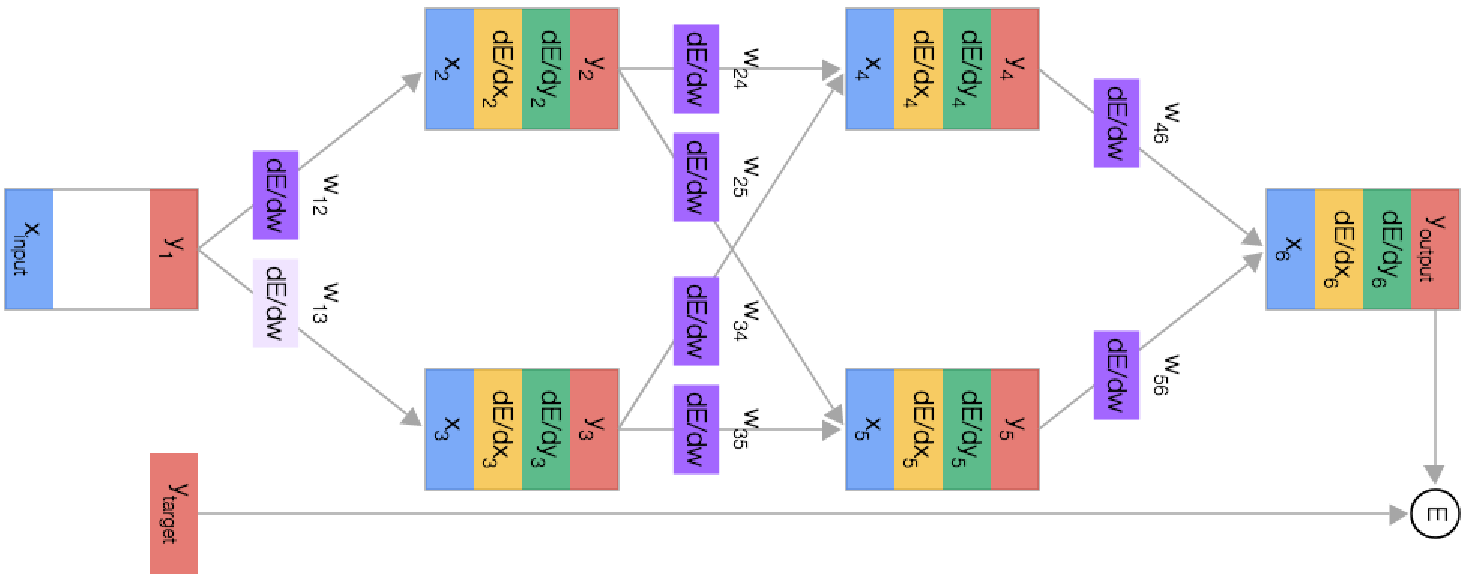
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
Compute gradients on weights and biases (including the regularization term, where needed):
$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$
end for

Backpropagation demo



Nice visualisation & explanation of forward prop and back-prop in neural nets
https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

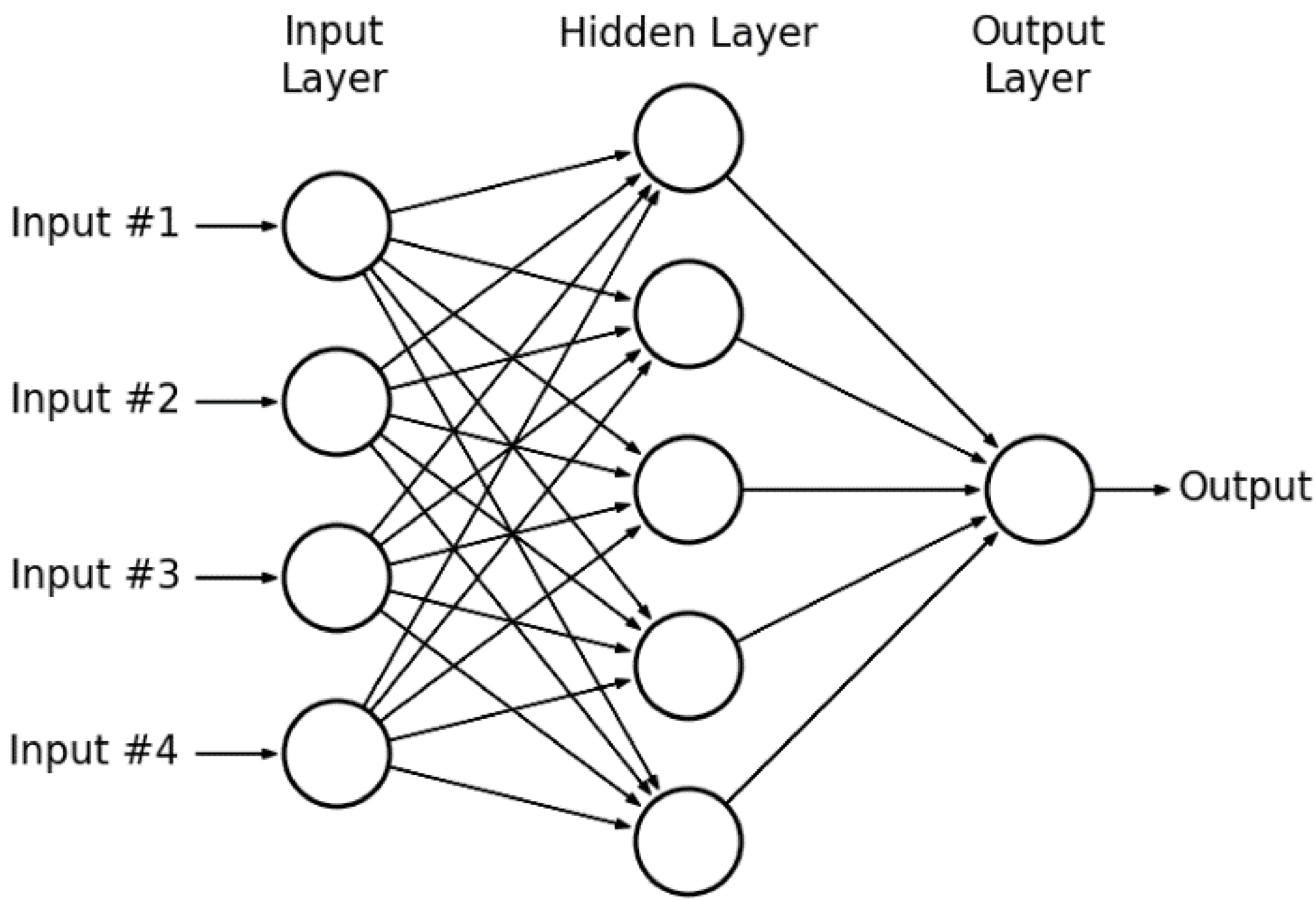
Backpropagation demo



Nice visualisation & explanation of forward prop and back-prop in neural nets
https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

…and another step-by-step example
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Units and layers



Backpropagation in PyTorch

- See https://discuss.pytorch.org/t/what-does-the-backward-function-do/9944 and https://medium.com/@zhang_yang/how-pytorch-tensors-backward-accumulates-gradient-8d1bf675579b to understand what backward() does
- And check the blitz tutorial to see how it's used https://github.com/uvadlc/uvadlc_practicals_2018/blob/master/pytorch/blitz/neural_networks_tutorial.ipynb