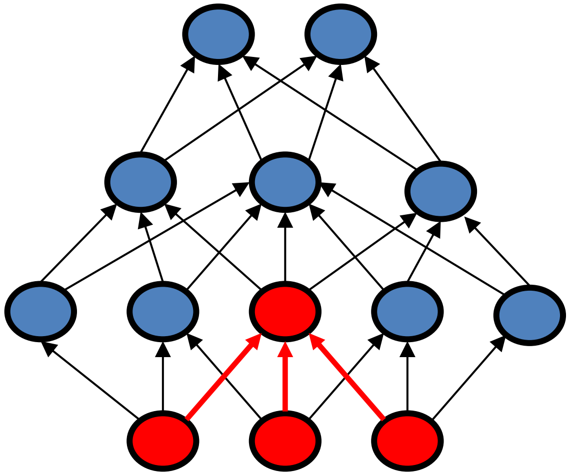
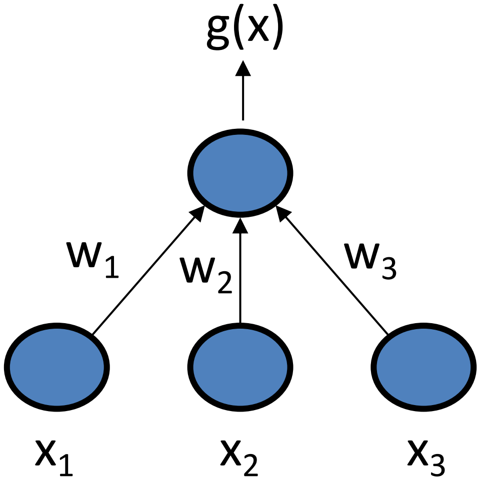
Deep neural networks and back-propagation

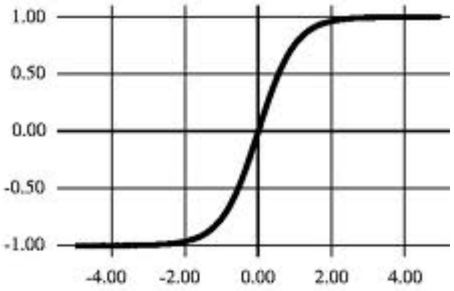
王兴刚

https://xinggangw.info

Neural network







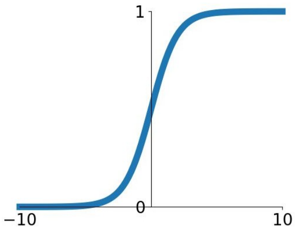
$f(net)$

$$g(\mathbf{x}) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(\mathbf{w}^t \mathbf{x})$$

Activation functions

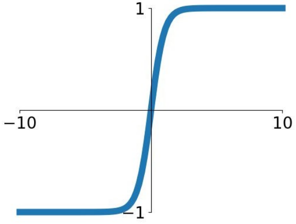
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



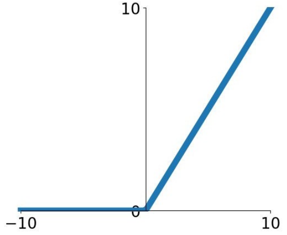
tanh

$$\tanh(x)$$



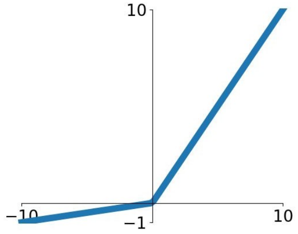
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

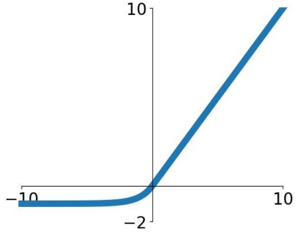


Maxout

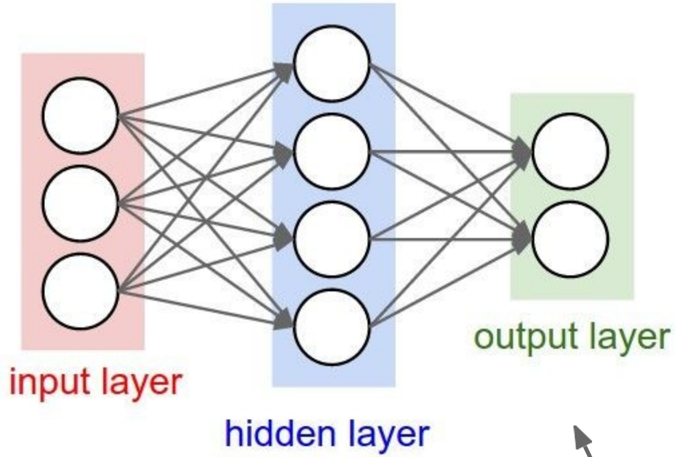
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

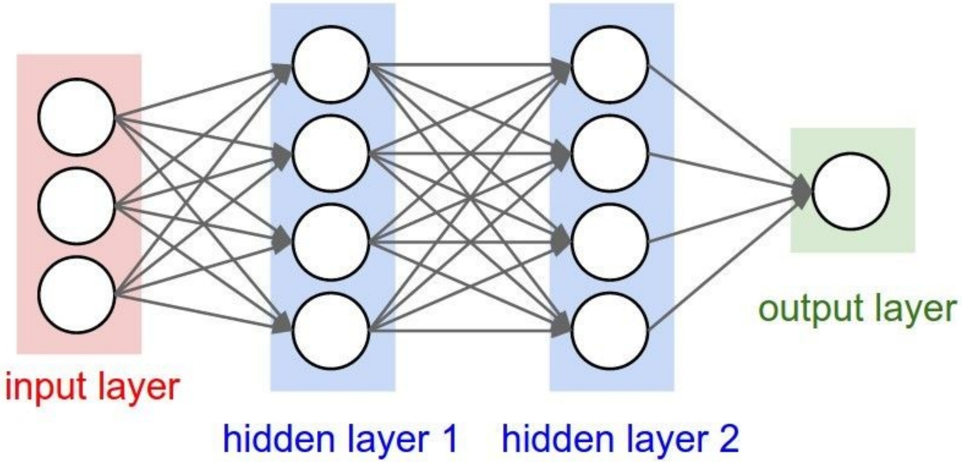
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural networks: Architectures





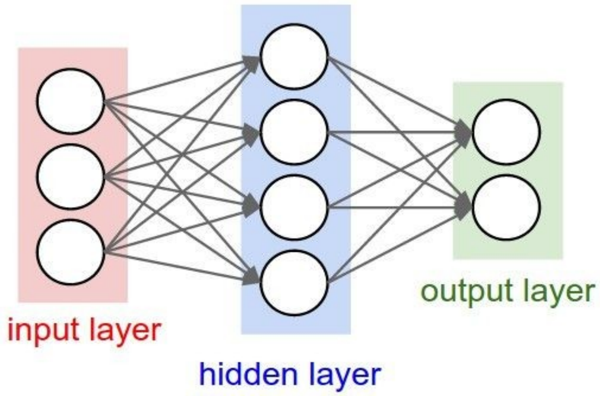
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Forward

$$z_1 = x^T W_1 + b_1$$
$$a_1 = \tanh(z_1)$$
$$z_2 = {a_1}^T W_2 + b_2$$
$$\hat{y} = a_2 = \text{softmax}(z_2)$$



$$z_2 = [s_1, s_2, \ldots, s_C]$$

$$\text{softmax}(s_k | z_2) = \frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}$$

Loss function

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{n\in N}\sum_{i\in C} y_{n,i}\log \hat{y}_{n,i}$$

Back propagation

Let o_k denote the k-th node of the input layer of the following softmax layer. The calculation of softmax function is given as follows.

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}} \tag{1}$$

The standard cross entropy loss function L is given as follows.

$$L = -\sum_j y_j \log p_j, \tag{2}$$

Then the derivation of the softmax cross entropy loss is given as follows.

$$\begin{aligned}
\frac{\partial L}{\partial o_i} &= -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} && (3)\\
&= -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial o_i} && (4)\\
&= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i) && (5)\\
&= -y_i(1-p_i) + \sum_{k \neq i} y_k (p_i) && (6)\\
&= -y_i + y_i p_i + \sum_{k \neq i} y_k (p_i) && (7)\\
&= p_i \left(\sum_k y_k \right) - y_i && (8)\\
&= p_i - y_i && (9)
\end{aligned}$$

Back propagation

Forward:

$$z_1 = x^T W_1 + b_1$$
$$a_1 = \tanh(z_1)$$
$$z_2 = {a_1}^T W_2 + b_2$$
$$\hat{y} = a_2 = \text{softmax}(z_2)$$

Backward:

$$\delta_3 = \hat{y} - y$$
$$\delta_2 = (1 - \tanh^2 z_1) \circ \delta_3 W_2^T$$
$$\frac{\partial L}{\partial W_2} = a_1^T \delta_3$$
$$\frac{\partial L}{\partial b_2} = \delta_3$$
$$\frac{\partial L}{\partial W_1} = x^T \delta 2$$
$$\frac{\partial L}{\partial b_1} = \delta 2$$

Back propagation

Backward:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial W_2} = (\hat{y} - y){a_1}^T$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial b_2} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \underbrace{\frac{\partial L}{\partial z_2}}_{(\hat{y} - y)}\underbrace{\frac{\partial z_2}{\partial a_1}}_{W_2}\underbrace{\frac{\partial a_1}{\partial z_1}}_{(1 - tanh^2 z_1)}\underbrace{\frac{\partial z_1}{\partial W_1}}_{x^T}$$

Forward:

$$z_1 = x^T W_1 + b_1$$
$$a_1 = \tanh(z_1)$$
$$z_2 = {a_1}^T W_2 + b_2$$
$$\hat{y} = a_2 = \text{softmax}(z_2)$$

Back propagation

Forward:

$$z_1 = x^T W_1 + b_1$$
$$a_1 = \tanh(z_1)$$
$$z_2 = {a_1}^T W_2 + b_2$$
$$\hat{y} = a_2 = \text{softmax}(z_2)$$

Backward:

$$\delta_3 = \hat{y} - y$$
$$\delta_2 = (1 - \tanh^2 z_1) \circ \delta_3 W_2^T$$
$$\frac{\partial L}{\partial W_2} = a_1^T \delta_3$$
$$\frac{\partial L}{\partial b_2} = \delta_3$$
$$\frac{\partial L}{\partial W_1} = x^T \delta 2$$
$$\frac{\partial L}{\partial b_1} = \delta 2$$

Implementation

Forward:

```python
# Helper function to predict an output (0 or 1)
def predict(model, x):
    W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
    # Forward propagation
    z1 = x.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    return np.argmax(probs, axis=1)
```

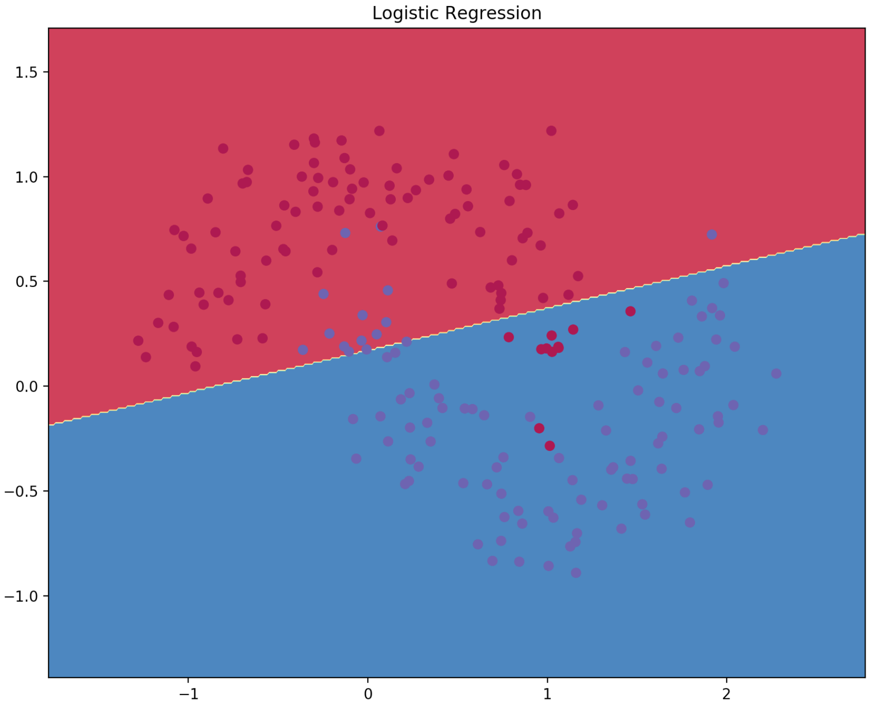
Implementation

Backward:

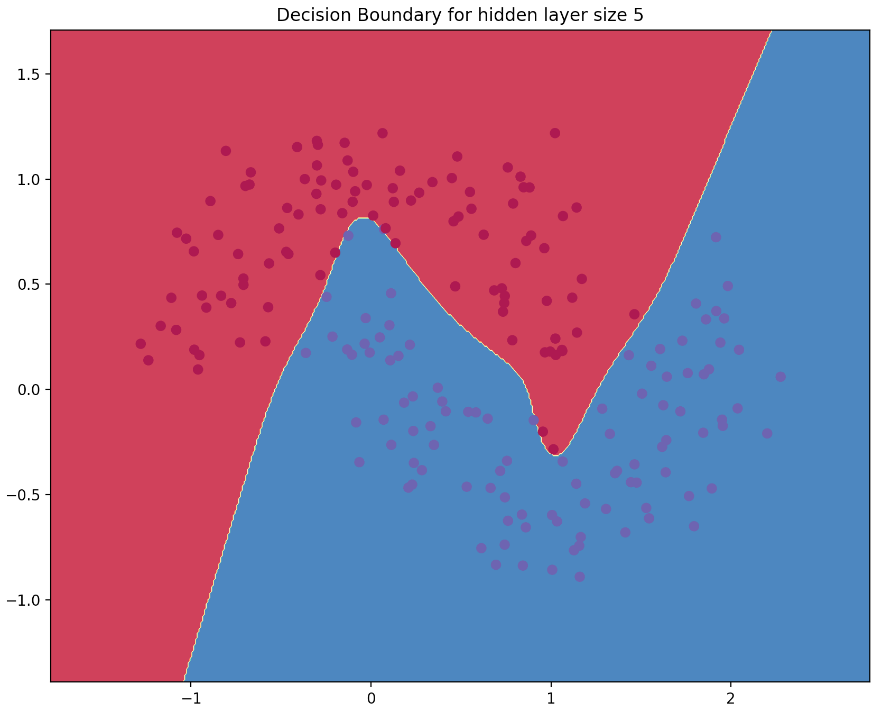
```
while True:
  data_batch = dataset.sample_data_batch()
  loss = network.forward(data_batch)
  dx = network.backward()
  x += - learning_rate * dx
```

```
    # Gradient descent. For each batch...
    for i in xrange(0, num_passes):

        # Forward propagation
        z1 = X.dot(W1) + b1
        a1 = np.tanh(z1)
        z2 = a1.dot(W2) + b2
        exp_scores = np.exp(z2)
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

        # Backpropagation
        delta3 = probs
        delta3[range(num_examples), y] -= 1
        dW2 = (a1.T).dot(delta3)
        db2 = np.sum(delta3, axis=0, keepdims=True)
        delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
        dW1 = np.dot(X.T, delta2)
        db1 = np.sum(delta2, axis=0)

        # Add regularization terms (b1 and b2 don't have regularization terms)
        dW2 += reg_lambda * W2
        dW1 += reg_lambda * W1

        # Gradient descent parameter update
        W1 += -epsilon * dW1
        b1 += -epsilon * db1
        W2 += -epsilon * dW2
        b2 += -epsilon * db2
```

Results



Logistic Regression
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2



Decision Boundary for hidden layer size 5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2

L0 regularization

Cost = $\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}\boldsymbol{w}_j)^2 + \lambda||\boldsymbol{w}||_0$

$||\boldsymbol{w}||_0$ means non-zero elements in w

$||\boldsymbol{w}||_0$ is non-convex and not differentiable.

L1 regularization

$$\mathrm{L}'(\mathrm{w}) = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}\boldsymbol{w}_j)^2 + \lambda|\boldsymbol{w}|_1$$

L1 regularization: $|\boldsymbol{w}|_1 = |w_1| + |w_2| + \cdots$

$$L'(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda\,|\boldsymbol{w}|_1 \qquad \frac{\partial L'}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial \boldsymbol{w}} + \lambda\,\mathrm{sign}(\boldsymbol{w})$$

Parameter update:

$$w^{t+1} = w^t - \eta\frac{\partial L'}{\partial \boldsymbol{w}} = w^t - \eta\left(\frac{\partial L}{\partial \boldsymbol{w}} + \lambda\,\mathrm{sign}(\boldsymbol{w})\right)$$

$$= w^t - \eta\frac{\partial L}{\partial \boldsymbol{w}} - \eta\lambda\,\mathrm{sign}(\boldsymbol{w})$$

$\eta\lambda\,\mathrm{sign}(\boldsymbol{w})$ always makes weight smaller (closing to zero)

L2 regularization

$$\mathrm{L}'(\mathrm{w}) = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}\boldsymbol{w}_j)^2 + \lambda\frac{1}{2}|\boldsymbol{w}|_2$$

L1 regularization: $|\boldsymbol{w}|_2 = |w_1|^2 + |w_2|^2 + \cdots$

$$L'(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda\,|\boldsymbol{w}|_2 \qquad \frac{\partial L'}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial \boldsymbol{w}} + \lambda\boldsymbol{w}$$

Parameter update:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta\frac{\partial L'}{\partial \boldsymbol{w}} = \boldsymbol{w}^t - \eta\left(\frac{\partial L}{\partial \boldsymbol{w}} + \lambda\boldsymbol{w}^t\right)$$

$$= (1 - \eta\lambda)\boldsymbol{w}^t - \eta\frac{\partial L}{\partial \boldsymbol{w}}$$

$(1 - \eta\lambda)\boldsymbol{w}^t$ always makes weight smaller (closing to zero)

Regularization

Regularization就是向你的模型加入某些规则，加入先验，缩小解空间，减小求出错误解的可能性。

[1,0,0,0] l1 = 1, l2 = 1
[0.25,0.25,0.25,0.25] l1 = 1, l2 = 0.25

L1鼓励系数稀疏

More tricks for training deep networks

- Optimizers: Momentum, AdaGrad, RMSProp, AdaDelta etc
- Learning rates
- Weight initialization
- Regularization: dropout, early stopping
- Batch normalization
- …

实践内容

- **采用BP神经网络在half moon和cifar10数据集上进行分类**

1. 依照课件上的内容，实现BP神经网络，在half moon上可视化非线性分类。
2. 在cifar10上，采用自己实现的BP神经网络来训练和测试并计算正确率。
3. 通过调整网络每层的节点数目、learning rate、正则化参数、网络层数、激活函数等，来争取获得最优的分类正确率。